



AP[®] Computer Science

Teacher's Guide

Deborah Power Carter
Lancaster Country Day School
Lancaster, Pennsylvania

connect to college success[™]
www.collegeboard.com

AP[®] Computer Science Teacher's Guide

Deborah Power Carter
Lancaster Country Day School
Lancaster, Pennsylvania

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,000 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT[®], the PSAT/NMSQT[®], and the Advanced Placement Program[®] (AP[®]). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information, visit www.collegeboard.com.

© 2007 The College Board. All rights reserved. College Board, Advanced Placement Program, AP, AP Central, AP Vertical Teams, Pre-AP, SAT, and the acorn logo are registered trademarks of the College Board. AP Potential and connect to college success are trademarks owned by the College Board. PSAT/NMSQT is a registered trademark of the College Board and National Merit Scholarship Corporation. All other products and services may be trademarks of their respective owners. Visit the College Board on the Web: www.collegeboard.com.

Contents

Welcome Letter from the College Board	v
Equity and Access	vii
Participating in the AP Course Audit	xi
Preface	xii
Chapter 1. About AP Computer Science	1
Overview: Past, Present, Future	1
Course Description Essentials	2
Key Concepts and Skills.....	3
Chapter 2. Advice for AP Computer Science Teachers	9
Getting Started: Help from the College Board.....	9
Real Teachers, Real Advice	13
Chapter 3. Course Organization	22
Syllabus Development.....	22
Six Sample Syllabi	23
Sample Syllabus 1 (AP Computer Science A)	25
Sample Syllabus 2 (AP Computer Science A)	32
Sample Syllabus 3 (AP Computer Science A)	39
Sample Syllabus 4 (AP Computer Science AB).....	49
Sample Syllabus 5 (AP Computer Science AB).....	56
Sample Syllabus 6 (AP Computer Science AB).....	62
Chapter 4. The AP Exams in Computer Science	73
All About the Exams	73
Exam Preparation	75
After the Exam.....	80

Contents

Chapter 5. Resources for Teachers	82
How to Address Limited Resources	82
Resources	86
Professional Development	97
Appendix A. GridWorld Case Study	101
Incorporating the Case Study Throughout Your Course.....	101
Appendix B. Supplemental Documents.....	108
Contents (by Syllabus Contributor)	108

Welcome Letter from the College Board

Dear AP® Teacher:

Whether you are a new AP teacher, using this AP Teacher’s Guide to assist in developing a syllabus for the first AP course you will ever teach, or an experienced AP teacher simply wanting to compare the teaching strategies you use with those employed by other expert AP teachers, we are confident you will find this resource valuable. We urge you to make good use of the ideas, advice, classroom strategies, and sample syllabi contained in this Teacher’s Guide.

You deserve tremendous credit for all that you do to fortify students for college success. The nurturing environment in which you help your students master a college-level curriculum—a much better atmosphere for one’s first exposure to college-level expectations than the often large classes in which many first-year college courses are taught—seems to translate directly into lasting benefits as students head off to college. An array of research studies, from the classic 1999 U.S. Department of Education study *Answers in the Tool Box* to new research from the University of Texas and the University of California, demonstrate that when students enter high school with equivalent academic abilities and socioeconomic status, those who develop the content knowledge to demonstrate college-level mastery of an AP Exam (a grade of 3 or higher) have much higher rates of college completion and have higher grades in college. The 2005 National Center for Educational Accountability (NCEA) study shows that students who take AP have much higher college graduation rates than students with the same academic abilities who do not have that valuable AP experience in high school. Furthermore, a Trends in International Mathematics and Science Study (TIMSS, formerly known as the Third International Mathematics and Science Study) found that even AP Calculus students who score a 1 on the AP Exam are significantly outperforming other advanced mathematics students in the United States, and they compare favorably to students from the top-performing nations in an international assessment of mathematics achievement. (Visit AP Central® at apcentral.collegeboard.com for details about these and other AP-related studies.)

For these reasons, the AP teacher plays a significant role in a student’s academic journey. Your AP classroom may be the only taste of college rigor your students will have before they enter higher education. It is important to note that such benefits cannot be demonstrated among AP courses that are AP courses in name only, rather than in quality of content. For AP courses to meaningfully prepare students for college success, courses must meet standards that enable students to replicate the content of the comparable college class. Using this AP Teacher’s Guide is one of the keys to ensuring that your AP course is as good as (or even better than) the course the student would otherwise be taking in college. While the AP Program does not mandate the use of any one syllabus or textbook and emphasizes that AP teachers should be granted the creativity and flexibility to develop their own curriculum, it is beneficial for AP teachers to compare their syllabi not just to the course outline in the official AP Course Description and in chapter 3 of this guide, but also to the syllabi presented on AP Central, to ensure that each course labeled AP meets the standards of a college-level course. Visit AP Central® at apcentral.collegeboard.com for details about the AP Course Audit, course-specific Curricular Requirements, and how to submit your syllabus for AP Course Audit authorization.

Welcome Letter

As the Advanced Placement Program® continues to experience tremendous growth in the twenty-first century, it is heartening to see that in every U.S. state and the District of Columbia, a growing proportion of high school graduates have earned at least one grade of 3 or higher on an AP Exam. In some states, more than 20 percent of graduating seniors have accomplished this goal. The incredible efforts of AP teachers are paying off, producing ever greater numbers of college-bound seniors who are prepared to succeed in college. Please accept my admiration and congratulations for all that you are doing and achieving.

Sincerely,

A handwritten signature in black ink that reads "Marcia L. Wilbur". The signature is written in a cursive style with a large initial "M" and a stylized "L".

Marcia Wilbur
Director, Curriculum and Content Development
Advanced Placement Program

Equity and Access

In the following section, the College Board describes its commitment to achieving equity in the AP Program.

Why are equitable preparation and inclusion important?

Currently, 40 percent of students entering four-year colleges and universities and 63 percent of students at two-year institutions require some remedial education. This is a significant concern because a student is less likely to obtain a bachelor's degree if he or she has taken one or more remedial courses.

Nationwide, secondary school educators are increasingly committed not just to helping students complete high school but also to helping them develop the habits of mind necessary for managing the rigors of college. As *Educational Leadership* reported in 2004:

The dramatic changes taking place in the U.S. economy jeopardize the economic future of students who leave high school without the problem-solving and communication skills essential to success in postsecondary education and in the growing number of high-paying jobs in the economy. To back away from education reforms that help all students master these skills is to give up on the commitment to equal opportunity for all.

Numerous research studies have shown that engaging a student in a rigorous high school curriculum such as is found in AP courses is one of the best ways that educators can help that student persist and complete a bachelor's degree. However, while 57 percent of the class of 2004 in U.S. public high schools enrolled in higher education in fall 2004, only 13 percent had been boosted with a successful AP experience in high school. Although AP courses are not the only examples of rigorous curricula, there is still a significant gap between students with college aspirations and students with adequate high school preparation to fulfill those aspirations.

Strong correlations exist between AP success and college success. Educators attest that this is partly because AP enables students to receive a taste of college while still in an environment that provides more support and resources for students than do typical college courses. Effective AP teachers work closely with their students, giving them the opportunity to reason, analyze, and understand for themselves. As a result, AP students frequently find themselves developing new confidence in their academic abilities and discovering their previously unknown capacities for college studies and academic success.

1. Andrea Venezia, Michael W. Kirst, and Anthony L. Antonio, *Betraying the College Dream: How Disconnected K–12 and Postsecondary Education Systems Undermine Student Aspirations* (Palo Alto, Calif.: The Bridge Project, 2003), 8.

2. Frank Levy and Richard J. Murnane, "Education and the Changing Job Market." *Educational Leadership* 62 (2) (October 2004): 83.

3. In addition to studies from University of California–Berkeley and the National Center for Educational Accountability (2005), see the classic study on the subject of rigor and college persistence: Clifford Adelman, *Answers in the Tool Box: Academic Intensity, Attendance Patterns, and Bachelor's Degree Attainment* (Washington, D.C.: U.S. Department of Education, 1999).

4. *Advanced Placement Report to the Nation* (New York: College Board, 2005).

5. Wayne Camara, "College Persistence, Graduation, and Remediation," *College Board Research Notes (RN-19)* (New York: College Board,

Which students should be encouraged to register for AP courses?

Any student willing and ready to do the work should be considered for an AP course. The College Board actively endorses the principles set forth in the following Equity Policy Statement and encourages schools to support this policy.

The College Board and the Advanced Placement Program encourage teachers, AP Coordinators, and school administrators to make equitable access a guiding principle for their AP programs. The College Board is committed to the principle that all students deserve an opportunity to participate in rigorous and academically challenging courses and programs. All students who are willing to accept the challenge of a rigorous academic curriculum should be considered for admission to AP courses. The Board encourages the elimination of barriers that restrict access to AP courses for students from ethnic, racial, and socioeconomic groups that have been traditionally underrepresented in the AP Program. Schools should make every effort to ensure that their AP classes reflect the diversity of their student population.

The fundamental objective that schools should strive to accomplish is to create a stimulating AP program that academically challenges students and has the same ethnic, gender, and socioeconomic demographics as the overall student population in the school. African American and Native American students are severely underrepresented in AP classrooms nationwide; Latino student participation has increased tremendously, but in many AP courses Latino students remain underrepresented. To prevent a willing, motivated student from having the opportunity to engage in AP courses is to deny that student the possibility of a better future.

Knowing what we know about the impact a rigorous curriculum can have on a student's future, it is not enough for us simply to leave it to motivated students to seek out these courses. Instead, we must reach out to students and encourage them to take on this challenge. With this in mind, there are two factors to consider when counseling a student regarding an AP opportunity:

1. Student motivation

Many potentially successful AP students would never enroll if the decision were left to their own initiative. They may not have peers who value rigorous academics, or they may have had prior academic experiences that damaged their confidence or belief in their college potential. They may simply lack an understanding of the benefits that such courses can offer them. Accordingly, it is essential that we not gauge a student's motivation to take AP until that student has had the opportunity to understand the advantages—not just the challenges—of such course work.

Educators committed to equity provide all students in a school with an understanding of the benefits of rigorous curricula. Such educators conduct student assemblies and/or presentations to parents that clearly describe the advantages of taking an AP course and outline the work expected of students. Perhaps most important, they have one-on-one conversations with the students in which advantages and expectations are placed side by side. These educators realize that many students, lacking confidence in their abilities, will be listening for any indication that they should not take an AP course. Accordingly, such educators, while frankly describing the amount of homework to be anticipated, also offer words of encouragement and support, assuring the students that if they are willing to do the work, they are wanted in the course.

The College Board has created a free online tool, AP Potential™, to help educators reach out to students who previously might not have been considered for participation in an AP course. Drawing upon data based on correlations between student performance on specific sections of the PSAT/NMSQT®

and performance on specific AP Exams, AP Potential generates rosters of students at your school who have a strong likelihood of success in a particular AP course. Schools nationwide have successfully enrolled many more students in AP than ever before by using these rosters to help students (and their parents) see themselves as having potential to succeed in college-level studies. For more information, visit <http://appotential.collegeboard.com>.

Actively recruiting students for AP and sustaining enrollment can also be enhanced by offering incentives for both students and teachers. While the College Board does not formally endorse any one incentive for boosting AP participation, we encourage school administrators to develop policies that will best serve an overarching goal to expand participation and improve performance in AP courses. When such incentives are implemented, educators should ensure that quality verification measures such as the AP Exam are embedded in the program so that courses are rigorous enough to merit the added benefits.

Many schools offer the following incentives for students who enroll in AP:

- Extra weighting of AP course grades when determining class rank
- Full or partial payment of AP Exam fees
- On-site exam administration

Additionally, some schools offer the following incentives for teachers to reward them for their efforts to include and support traditionally underserved students:

- Extra preparation periods
- Reduced class size
- Reduced duty periods
- Additional classroom funds
- Extra salary

2. Student preparation

Because AP courses should be the equivalent of courses taught in colleges and universities, it is important that a student be prepared for such rigor. The types of preparation a student should have before entering an AP course vary from course to course and are described in the official AP Course Description book for each subject (available as a free download at apcentral.collegeboard.com).

Unfortunately, many schools have developed a set of gatekeeping or screening requirements that go far beyond what is appropriate to ensure that an individual student has had sufficient preparation to succeed in an AP course. Schools should make every effort to eliminate the gatekeeping process for AP enrollment. Because research has not been able to establish meaningful correlations between gatekeeping devices and actual success on an AP Exam, the College Board strongly discourages the use of the following factors as thresholds or requirements for admission to an AP course:

- Grade point average
- Grade in a required prerequisite course
- Recommendation from a teacher

Equity and Access

- AP teacher’s discretion
- Standardized test scores
- Course-specific entrance exam or essay

Additionally, schools should be wary of the following concerns regarding the misuse of AP:

- Creating “Pre-AP courses” to establish a limited, exclusive track for access to AP
- Rushing to install AP courses without simultaneously implementing a plan to prepare students and teachers in lower grades for the rigor of the program

How can I ensure that I am not watering down the quality of my course as I admit more students?

Students in AP courses should take the AP Exam, which provides an external verification of the extent to which college-level mastery of an AP course is taking place. While it is likely that the percentage of students who receive a grade of 3 or higher may dip as more students take the exam, that is not an indication that the quality of a course is being watered down. Instead of looking at percentages, educators should be looking at raw numbers, since each number represents an individual student. If the raw number of students receiving a grade of 3 or higher on the AP Exam is not decreasing as more students take the exam, there is no indication that the quality of learning in your course has decreased as more students have enrolled.

What are schools doing to expand access and improve AP performance?

Districts and schools that successfully improve both participation and performance in AP have implemented a multipronged approach to expanding an AP program. These schools offer AP as capstone courses, providing professional development for AP teachers and additional incentives and support for the teachers and students participating at this top level of the curriculum. The high standards of the AP courses are used as anchors that influence the 6–12 curriculum from the “top down.” Simultaneously, these educators are investing in the training of teachers in the pre-AP years and are building a vertically articulated, sequential curriculum from middle school to high school that culminates in AP courses—a broad pipeline that prepares students step-by-step for the rigors of AP so that they will have a fair shot at success in an AP course once they reach that stage. An effective and demanding AP program necessitates cooperation and communication between high schools and middle schools. Effective teaming among members of all educational levels ensures rigorous standards for students across years and provides them with the skills needed to succeed in AP. For more information about Pre-AP® professional development, including workshops designed to facilitate the creation of AP Vertical Teams® of middle school and high school teachers, visit AP Central.

Participating in the AP Course Audit

Overview

The AP Course Audit is a collaborative effort among secondary schools, colleges, and universities, and the College Board. For their part, schools deliver college-level instruction to students and complete and return AP Course Audit materials. Colleges and universities work with the College Board to define elements common to college courses in each AP subject, help develop materials to support AP teaching, and receive a roster of schools and their authorized AP courses. The College Board fosters dialogue about the AP Course Audit requirements and recommendations and reviews syllabi.

Starting in the 2007-08 academic year, all schools wishing to label a course “AP” on student transcripts, course listings, or any school publications must complete and return the subject-specific AP Course Audit form, along with the course syllabus, for all sections of their AP courses. Approximately two months after submitting AP Course Audit materials, schools will receive a legal agreement authorizing the use of the “AP” trademark on qualifying courses. Colleges and universities will receive a roster of schools listing the courses authorized to use the “AP” trademark at each school.

Purpose

College Board member schools at both the secondary and college levels requested an annual AP Course Audit in order to provide teachers and administrators with clear guidelines on curricular and resource requirements that must be in place for AP courses and to help colleges and universities better interpret secondary school courses marked “AP” on students’ transcripts.

The AP Course Audit form identifies common, essential elements of effective college courses, including subject matter and classroom resources such as college-level textbooks and laboratory equipment. Schools and individual teachers will continue to develop their own curricula for AP courses they offer—the AP Course Audit will simply ask them to indicate inclusion of these elements in their AP syllabi or describe how their courses nonetheless deliver college-level course content.

AP Exam performance is not factored into the AP Course Audit. A program that audited only those schools with seemingly unsatisfactory exam performance might cause some schools to limit access to AP courses and exams. In addition, because AP Exams are taken and exam grades reported after college admissions decisions are already made, AP course participation has become a relevant factor in the college admissions process. On the AP Course Audit form, teachers and administrators attest that their course includes elements commonly taught in effective college courses. Colleges and universities reviewing students’ transcripts can thus be reasonably assured that courses labeled “AP” provide an appropriate level and range of college-level course content, along with the classroom resources to best deliver that content.

For more information

You should discuss the AP Course Audit with your department head and principal. For more information, including a timeline, frequently asked questions, and downloadable AP Course Audit forms, visit apcentral.collegeboard.com/courseaudit.

Preface

Congratulations! You're embarking on an exciting adventure, teaching AP Computer Science (AP CS), and this Teacher's Guide was designed to help you prepare for that challenge.

AP teachers have different backgrounds. Perhaps,

- you're a first-time teacher or a seasoned veteran;
- you've been teaching computer science for a while, or this is your first computer science course;
- your background is in computer science, mathematics, business, or an entirely different field;
- you have a degree in computer science, earned a living as a programmer, or studied computer science on your own; or
- you lobbied your school administration to offer AP Computer Science or have been drafted to teach it.

Regardless of how you got to this point in your journey, you're sure to find an abundance of helpful advice from teachers who have followed a similar path.

Chapter 1, *About AP Computer Science*, begins with an overview by Judith Hromcik from Arlington High School in Arlington, Texas, a former member of the AP Computer Science Development Committee. Then there is a review of the *AP Computer Science Course Description*, a separate publication that includes curriculum outlines for both AP Computer Science A and AP Computer Science AB, along with a detailed Commentary. The chapter continues with information on prerequisites for computer science students, followed by a description of the major concepts of AP CS.

Chapter 2, *Advice for AP Computer Science Teachers*, provides instructions for preparing to teach AP Computer Science. You'll find information about helpful College Board resources, as well as advice from seasoned teachers on a wide range of subjects, from philosophical to practical. Even experienced teachers are sure to find some helpful suggestions here.

Chapter 3, *Course Organization*, offers guidance for planning your course syllabus. This chapter includes six complete syllabi from experienced computer science teachers: four high school AP Computer Science teachers and two university instructors of introductory computer science courses. There are three syllabi each for AP CS A and AP CS AB. In addition to the course planner, each syllabus contributor has included strategies for teaching and evaluating, as well as a detailed list of resources.

Chapter 4, *The AP Exams in Computer Science*, has information on the format and content of the exams, as well as scoring information, advice for preparing your students, and a list of sources for practice questions. The chapter concludes with some challenging and fun activities to help keep students engaged in the days or weeks *after* they have taken the exams.

Chapter 5, *Resources for Teachers*, begins with suggestions for making the most of limited resources, including proven advice from AP Computer Science teachers. This chapter has several lists of teaching resources, including textbooks, software, instructor materials, and videos/DVDs, plus a section on professional development opportunities offered by the College Board.

Appendix A, GridWorld Case Study, provides a lot of tips for teaching your course with the case study, including one teacher’s detailed suggestions for incorporating it throughout the course. A list of outside resources for supplementing the case study and making it fun is included.

Appendix B, Supplemental Documents, includes copies of student activities from chapter 3’s syllabus contributors. You’ll find practice worksheets, review notes, and lab assignments, along with any required source code.

Both new and experienced teachers are sure to find ideas and inspiration to help keep their courses fresh and interesting. The journey ahead won’t be easy, and you should immediately remove “complacent” from your vocabulary, but I promise you an exciting and challenging adventure!

Debbie Carter



Dedication: To Bill, whose constant, loving support made this book possible.

Debbie Carter teaches AP Computer Science at Lancaster Country Day School in Lancaster, Pennsylvania, where she also facilitates the use of technology by teachers and students. She has served the AP Computer Science community as a workshop consultant, AP Exam Reader, and Question Leader, as well as through various professional development committees. She also serves on the board of directors for the Computer Science Teachers Association.

Chapter 1

About AP Computer Science

Overview: Past, Present, Future

When I began thinking about what I would write for this section of the Teacher’s Guide, I decided to find a definition of “computer science.” So, I went to Google and typed in: “What is computer science?” This is the definition that I found:

Computer Science: The systematic study of computing systems and computation. The body of knowledge resulting from this discipline contains theories for understanding computing systems and methods; design methodology, algorithms, and tools; methods for the testing of concepts; methods of analysis and verification; and knowledge representation and implementation.⁶

The College Board added computer science to the Advanced Placement Program in the 1983-84 academic year; as the discipline has evolved, the AP Computer Science course and exams have also changed. One example is language. Pascal, which was created to introduce students to programming, was the first language used in AP Computer Science. At the time, it was the best vehicle to teach the computer science concepts of problem solving, algorithms, logic, and design. Pascal was a structured, modular language that allowed recursion, pointers, and user-defined data types. It served AP CS well for 15 exam cycles.

In the early 1990s it became clear that colleges were moving away from Pascal toward languages that allowed the creation of abstract data types that could be written in separate modules and incorporated into any program. Object-oriented programming was being taught in first- and second-year college-level computer science courses. So, the AP Computer Science Development Committee chose C++ as the language that would best keep the AP courses comparable to their college counterparts, and teachers learned a new syntax as well as how to design and implement classes. The AP subset for C++ did not include inheritance, however, so they were really teaching object-based programming instead of object-oriented programming.

Between the time that the decision was made to switch to C++ and the first AP Exam in C++ in 1999, Java came onto the scene. Java is a safer language than C++ and has a clean way of implementing inheritance. At the 1999 AP Reading, we were already hearing speculation that Java was on the way—and indeed, Java became the new language for AP Computer Science in 2004. It was chosen because it was judged to be one of the best available languages to teach the fundamental concepts that colleges require beginning computer science students to know, and because a significant percentage of colleges are currently using it as their introductory language, increasing the likelihood that AP students will receive college credit for their work.

6. “Glossary,” *High Performance Computing and Communications: Towards a National Information Infrastructure* (National Coordination Office for Information Technology and Development). www.hpcc.gov/pubs/blue94/section.6.html.

Chapter 1

Look back at the definition of computer science. With all three languages, Pascal, C++, and Java, AP CS instructors have taught and are teaching the concepts listed in this definition. The design methodology has changed, and so has the knowledge representation and implementation, but we are still teaching these essential principles.

What will happen in the future? Is there a new language on the horizon? I don't know. I do know that computer science is a dynamic field. As it changes and grows, so will AP Computer Science.

*Judith Hromcik
AP Computer Science Teacher
Arlington High School
Arlington, Texas*

For more details about the transitions in the AP Computer Science program, see Susan Horwitz's article, "And On to Java," available on the Course Home Pages at apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab [AP Central > Computer Science A (or AB) Course Home Page]⁷

Course Description Essentials

The *AP Computer Science Course Description* serves as an AP teacher's primary resource for information on the courses and exams. The Course Description outlines course content, explains the kinds of skills that students are expected to demonstrate, and gives valuable information about the exams. Throughout this Teacher's Guide, you will be referred back to the Course Description; it will be invaluable as you plan and teach AP Computer Science, especially during your first year, but regularly thereafter.

The Course Description includes:

- General AP Program information
- Goals for the Computer Science courses
- The all-important Topic Outline, which details concepts that will be covered on the AP CS Exams
- A Commentary on the Topic Outline, with examples and some language-specific guidelines
- Information about case studies
- Sample multiple-choice questions, with answers
- Sample free-response questions, with suggested solutions
- The AP CS Java Subset (language features and standard library methods)
- Implementation classes for linked list and tree nodes (for use when students implement their own linked lists and binary trees)

The AP Computer Science Development Committee, a group of six teachers from secondary schools and colleges in different parts of North America, is responsible for setting the direction of the program and maintaining the Course Description. In consultation with the computer science Assessment Specialists at ETS, the committee is also responsible for creating the exams. In addition, the Chief Reader, a college

7. Throughout this *Teacher's Guide*, you'll find references to content on AP Central. For easy recognition, navigation shortcuts will be shown in brackets.

faculty member who coordinates the scoring of the free-response questions at the annual AP Reading, aids in the development process. The Chief Reader attends the meetings of the Development Committee to ensure that the exams' free-response questions can be scored reliably. (See chapter 4 for information on how AP Computer Science Exams are scored.)

There are no computing prerequisites for either AP Computer Science course; each is designed to serve as a first course in computer science for students with no prior computing experience. However, it is essential that students have knowledge of basic algebra and experience in problem solving. In addition, since documentation plays a central role in programming methodology, students should have acquired facility in written communication before beginning a computer science course.

To be successful on the AP CS Exams, students need to be familiar with all of the key concepts and skills defined in this chapter. As you review the various components of the Course Description, you'll want to identify any areas where you need to increase your understanding. Keep those areas in mind as you read through this Teacher's Guide, so that you can flag resources that will help you in this endeavor.

The Course Description can be downloaded for free from the Course Home Pages on AP Central: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab. [AP Central > Course Home Pages > AP Computer Science A or AB Course Description]

Key Concepts and Skills

This section describes some key concepts and skills with which AP Computer Science students need to become familiar. The Commentary on the Topic Outline, contained in the *AP Computer Science Course Description* and from which much of the information in this section is drawn, provides more detailed descriptions for each of the following major concepts, including examples within other College Board publications and expectations for students of AP Computer Science A and Computer Science AB courses. (See the previous section of this chapter for a link to the online Course Description.) The following pages provide an introductory overview of the major areas of emphasis.

I. Object-Oriented Program Design

The overall goal for designing a piece of software (a computer program) is to solve a given problem. The design process should be based on a thorough understanding of the problem, and it should include the goal of creating an understandable, adaptable solution.

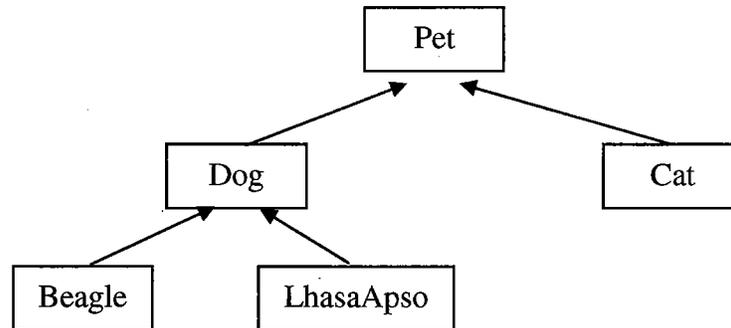
Novice programmers should begin learning about the design process by studying well-designed programs and modifying them. Later in the course, students should be able to work from a specification to develop a design for a program or part of a program.

The fundamental component of an object-oriented program is an *object*, an entity that has *state* (storage of some data) and *behavior* (operations that access or change its state and that may interact with other objects). Objects are defined by *classes*; a class specifies the components and operations of an object, and each object is an *instance* of a class. If you learned to program using FORTRAN and punched cards, understanding object-oriented design may be more difficult for you than for your students!

A student should be able to understand the relationships among the different classes that comprise a program. Two common relationships between classes are often referred to as “is-a” (inheritance) and “has-a” (composition) relationships.

Chapter 1

In an *inheritance* hierarchy, a subclass inherits characteristics from its superclass. For example, if we have a class Dog, with a superclass Pet and subclasses Beagle and LhasaApso, we could say that a Beagle “is-a” Dog, a Dog “is-a” Pet, etc.



The other common relationship among classes is composition. An object of one class has one or more instances of another class or classes as its attributes. For example, a Kennel would have Pets, and a Dog would have a Tail and two Ears. Ear, Tail, and Kennel would all be separate classes. We could say that a Dog “has-a” Tail, and a Kennel “has-a (some)” Pets.

Attempting to apply “is-a” and “has-a” to other relationships will often help identify cases where inheritance and composition are not appropriate relationships. For example, we would not say that a Tail “is-a” Dog, so it would not be appropriate to define their relationship with inheritance (though it might be technically possible). Likewise, we wouldn’t say that a Beagle “has-a” Dog, so we would not define the Beagle class as having a Dog component.

When designing programs to solve problems, students should also recognize the appropriate use of components of existing libraries. The AP Java Subset, as defined in the appendixes of the Course Description, specifies the Java classes and methods with which students must be familiar when taking the AP CS Exam.

Note: In their free-response solutions, students will not be penalized for using methods from standard Java classes that are not listed in the AP Java Subset. Chris Nevison, a former Chief Reader for AP Computer Science, wrote, “The AP CS Development Committee tries to structure problems so that students will not gain an advantage by knowing a ‘short-cut’ method that is out of the subset. So in most cases, knowing additional methods outside the subset will not make things any easier, although it might provide a different way of getting a solution.”

Within a program solution, individual classes must be designed. Again, students should first learn about class design by exploring well-designed classes, but they will progress to designing their own classes, based on a description of the type of entity that each class represents. For classes that they design, students should be able to define the instance variables (choosing appropriate data types) as well as method declarations (the method identifiers, return types, and parameters).

Students should understand that an *interface* is a specification for a set of methods that a class must implement. A class that *implements* the interface must implement each of the methods that the interface specifies. An understanding of *abstract classes* and their appropriate use is also assumed within the AP CS course.

II. Program Implementation

To implement a program, a student must understand the fundamental programming constructs of the language, as well as the design of the program.

Object-oriented development involves implementing the individual methods of each class in the design. In this phase, we might also discover that additional classes are needed, from either the Java libraries or another class that we develop. For example, we might decide that the `Kennel` class needs a `Cage` class (with a `size`), rather than just a list of `Pets`, because different `Pets` need different `Cages` of different sizes.

Good program implementation adheres to the fundamental principles of encapsulation and information hiding. *Encapsulation* is the process of organizing some information and the operations on that information into one unit, a class. *Information hiding* is the technique of keeping the data representation hidden from the client (external classes) by declaring it `private`.

Programming constructs are tools a programmer uses to implement a program. These are tied closely to the language, though most object-oriented programming languages have similar constructs, so the skills acquired by a Java programmer would carry over quite easily to another object-oriented programming language.

An understanding of the following constructs is fundamental to programming in Java. A *declaration* assigns an identifier to a construct and defines it as appropriate. A *variable* declaration specifies the type: primitive (`int`, `double`, or `boolean`) or a class reference. A *constant* is declared in the same way, but it may not change value; the keyword `final` indicates a constant. A *method* declaration consists of the access modifier, the return type (`void`, if no value is returned), the method identifier, and the parameter list, if any. *Parameters* are declared as part of a method declaration; these declarations are similar to variable declarations. For example, a `Radio` class might have the following method:

```
public void changeStation ( double frequency )
```

where `frequency` is a `double` value that represents the station to which the `Radio` object should be reset.

Declarations of *classes* and *interfaces* are usually accomplished during the design phase, as described in section I above.

Program statements are executed sequentially, unless a *control construct* alters that sequence. Students are expected to be familiar with the following types of control constructs:

A *method call* is among the most common control constructs. A call to a method transfers control to the method, but control returns to the same location after the method has completed its execution. An object method is part of a class and operates on an instance of that class. A method may simply return information about the object (an *accessor* method) or may change the state of the object (a *mutator* or *modifier* method). It is essential that students learn early to manipulate objects of both library classes and user-defined classes.

Chapter 1

A *conditional* control structure causes different results, based on the evaluation of its condition. For example, the following conditional

```
if (age < 6)
{
    ageMsg = "Too young.";
}
else
{
    ageMsg = "Old enough.";
}
```

sets a text message based on the value of `age`.

Another fundamental control construct is *iteration*, repeated execution of one or more statements, which is accomplished by loop constructs such as the `for` loop and the `while` loop.

Recursion is another common means of producing repetitive behavior. A recursive method calls itself (creating a new instance of the method with each call) with a different parameter value, eventually arriving at a “base case,” which stops the recursion. Valid recursion must have logic that makes successive recursive calls progress toward the base case.

Modern programming languages have extensive libraries that supply many common classes and methods. AP Computer Science students should learn about available Java libraries and their appropriate use, consulting the library documentation (the Java API) for assistance.

III. Program Analysis

Persons working in the field of computer science analyze programs for correctness and to understand their efficiency for solving problems.

Testing is the most obvious way to analyze correctness; although it does not prove correctness, it can certainly disprove it! Data for testing must be selected to reflect a range of typical cases, including boundary cases and erroneous cases, which should be handled with error messages or exceptions.

Debugging refers to the discovery and correction of errors in a program. These errors can be found through testing or careful analysis. Students should be able to handle all three categories of errors: *runtime*, *compiler*, and *logic* errors. Debugging techniques include hand-tracing of code, adding extra output statements to trace the execution of a program, and using a debugger to trace execution and display selected values as it runs and when it crashes. Students should be encouraged to experiment with available debugging facilities, but they should also be able to rely on more primitive debugging methods, which will serve them in whatever environment they encounter in the future.

In addition to writing their own programs, students should be able to read and modify code that was written by another programmer. They should also be able to extend a class by declaring a new subclass that inherits from it.

Exceptions provide valuable information about runtime errors, and students will need to recognize common Java exceptions. AP Computer Science AB students will also need to handle errors by throwing exceptions.

Formal analysis allows us to verify correctness by proof and is preferable to tracing and testing, especially for life-critical software. Formal analysis uses assertions, including preconditions and postconditions as well as loop invariants.

The analysis of an algorithm's efficiency is an important part of analyzing programs. Students need to be able to make informal comparisons of running times and space requirements of different pieces of code. AP Computer Science AB students also need to apply formal methods of analysis, such as asymptotic (using Big-Oh notation), worst-case, and average-case analysis.

Programs are limited by the finite representations of numbers in any given system. Students should have some understanding of binary representation of numbers as well as the practical consequences of their limits.

IV. Standard Data Structures

A number of standard data structures are used in programming. Primitive data types (`int`, `double`, and `boolean`) are used to hold one piece of information. Strings and arrays are common to most high-level languages.

Classes enable us to define new types that encapsulate both data and operations (methods). A class may be simply a container for related data or a more complex object with complex operations and interactions with other objects.

The AP Computer Science AB course focuses on both *abstract data types* (ADTs) and data structures. An ADT is described in terms of its operations, rather than any specific implementation. For example, a List may be described as an ordered collection of items.

A number of data structures can be used to implement common ADTs. Some of the ADTs are implemented by one or more Java library classes; students should be familiar with those classes, but they should also be able to implement their own versions with appropriate data structures.

The Commentary on the Topic Outline in the Course Description provides an excellent, well-organized presentation of the ADTs, data structures, and the associated Java library implementations with which AP Computer Science AB students are expected to be familiar.

V. Standard Algorithms

Standard algorithms serve as good solutions to standard problems. These algorithms, many of which are intertwined with data structures, provide excellent examples for the analysis of program efficiency. Programs that implement standard algorithms also serve as good models for program design.

Many standard algorithms involve array operations: traversing an array, inserting and deleting an element, and searching (both sequential and binary searches). Standard sorts also manipulate the elements of arrays. (See chapter 2 for suggestions that involve visual representations of these algorithms.) Students also need to analyze these standard algorithms, either informally (in the AP Computer Science A course) or formally (in AP Computer Science AB).

As can be expected, the AP Computer Science AB course also deals with standard algorithms that are associated with the additional data structures in its curriculum.

VI. Computing in Context

A working knowledge of the major hardware and software components of computer systems is necessary for the study of computer science. These topics need not be covered in detail, but they should be addressed as appropriate throughout the course. Students will usually become familiar with some of these elements as they use computers for programming and other applications.

Given the tremendous impact that computers and computing have on today's society, teachers must take care to encourage the development of intelligent and responsible attitudes and behavior regarding the use of computers. References to responsible use of computer systems should be integrated into an AP CS course wherever appropriate, rather than taught as a separate unit. Typical issues include privacy rights; reliability, especially of life-critical applications; viruses and other malicious attacks on computer systems; and intellectual property rights of artists, writers, musicians, and programmers.

Students should learn to take responsibility for the programs they write and for the consequences of the use of their programs. Attitudes are acquired, not taught, and students will note their teacher's attitudes and behaviors, even if they profess to disagree with some of them.

Chapter 2

Advice for AP Computer Science Teachers

We have called in the experts to help you prepare for teaching AP Computer Science. The College Board has developed numerous resources, and we'll introduce you to them here. In addition, other AP CS teachers will share some of their favorite teaching strategies, both philosophical and practical.

Getting Started: Help from the College Board

To best prepare to teach an AP Computer Science course, you should first:

1. Register at AP Central.
2. Join the electronic discussion group (EDG) for AP Computer Science.
3. Review the *AP Computer Science Course Description*.
4. Sign up for a professional development workshop.
5. Begin reviewing other materials that are available on AP Central.

AP Central

Your first stop should be AP Central (apcentral.collegeboard.com), the College Board Web site for AP teachers. Registration is free, and the site contains everything you need to get started: the *AP Computer Science Course Description*, case-study documents and code, teaching resource materials and lesson plans, feature articles, and the Teachers' Resources section. Teachers' Resources is a database of over 4,500 original reviews of textbooks, software, videos, Web sites, and other teaching material, assessing each for its suitability for the AP classroom.

When you register on AP Central, go to your Personal Profile to select the courses in which you are interested. By making those selections, you will get personalized news and promotions on the AP Central Home Page and also have the option of subscribing to e-mail newsletters that alert you at least twice a year to additions to AP Central that relate to your selected courses.

Navigation Tips

Once you have selected your AP courses (for your Personal Profile), clicking on the **My AP Central** button will streamline your navigation by presenting the most common choices (Course Description, Course Home Page, Exam Questions, etc.) for those courses.

Chapter 2

The **Course Home Page** gives a complete listing of resources (and links) for each course: course guidelines, exam information, case-study materials, teaching strategies, feature articles, instructions for subscribing to the electronic discussion group and e-newsletter, etc. The Course Home Page for AP Computer Science A can be found at apcentral.collegeboard.com/compscia; the Course Home Page for AP Computer Science AB can be found at apcentral.collegeboard.com/compsciab.

See chapter 5 for more detailed information about AP Central, including a two-page AP Central Quickstart Guide and full Web addresses for resources.

Electronic Discussion Groups (EDGs)

The AP electronic discussion groups (EDGs) are moderated, Web-based groups that allow users to post messages online to be viewed by the entire group. Messages can also be sent and received via e-mail. The AP Computer Science EDG is especially active. Both new and experienced teachers post requests for help and advice on specific issues, and other AP CS teachers, as well as some college instructors of introductory computer science courses, usually respond quickly with suggestions. You may also search the EDG archives for previously posted advice on a particular subject.

Visit either of the Course Home Pages to sign up for the EDG: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab. [AP Central > Course Home Pages > Registration for Electronic Discussion Groups]

Professional Development Workshops

It is strongly recommended that you attend an AP Summer Institute prior to teaching the course for the first time and participate in one-day workshops and conferences available in your region throughout the academic year.

If you are a new teacher of AP Computer Science, the primary advantages of a summer institute are the hands-on lab experience and the chance to spend extended time with computer science colleagues. During the academic year, workshops provide an opportunity to discuss issues (technical and/or pedagogical) that have arisen during your teaching. Often new teachers don't even know the questions to ask until they have tried things in their classrooms.

See the Professional Development section in chapter 5 for more information on summer institutes and workshops.

The Java Engagement for Teacher Training Program (JETT) is a partnership between the Association for Computing Machinery's (ACM) K-12 Task Force and the College Board. JETT provides quality pedagogically oriented workshops and resources in Java for secondary school computer science teachers. Hosted by universities, JETT workshops run for one or more days and typically include an equity component, some hands-on sessions, and interaction with university faculty and student assistants. JETT workshops are listed at AP Central, as well as at the JETT Web site: <http://jett.acm.org>.

Funding Assistance

The **College Board Fellows Program** is a competitive grant program that provides stipends for secondary school teachers planning to teach AP courses in schools that serve minority and/or low-income students who have been traditionally underrepresented in AP courses. The stipends help defray the cost of attending an endorsed AP Summer Institute. To qualify, a school must have 50 percent or more underrepresented

minority students or be located in an area where the average income level is equivalent to, or below, the national annual average for a low-income family of four (approximately \$36,000). Approximately 250 awards are distributed each year.

The **College Board Pre-AP Fellows Program** is a competitive grant program that provides funding to AP Vertical Teams from schools in minority-dominant or economically disadvantaged areas with few or no AP courses to receive training in the following areas: English, mathematics, music theory, social studies, and studio art. Grants go toward funding the team's attendance at an endorsed Pre-AP Summer Institute. Grants of \$10,000 each are awarded to AP Vertical Teams that best satisfy the eligibility requirements. Ten awards are available each grant year.

Although a computer science teacher is not a required member of a Vertical Team in mathematics, Vertical Team training can better equip you to teach introductory programming courses or to help integrate problem-solving and programming into mathematics courses students take before their AP courses.

Application forms for both programs become available each September on AP Central. Visit apcentral.collegeboard.com/apgrants or write to apecuity@collegeboard.org for more information.

If you are not eligible for the College Board Fellows Program: Many school districts or individual schools fund the professional development of their teachers. Check with your principal or district supervisor about available funding.

Publications

In addition to this Teacher's Guide, here are some other print or electronic publications that will be useful to you. Unless otherwise noted, AP Central resources are available on the AP Computer Science Home Page.

The *AP Computer Science Course Description* outlines course content, explains the kinds of skills students are expected to demonstrate, and gives valuable information about the exams. Sample multiple-choice questions with an answer key are included, as are sample free-response questions and suggested solutions. **Advice:** Download a new copy of the Course Description whenever it is updated and check for changes from the previous edition.

Case Study: AP Computer Science Exams include questions based on a case study. Text and code for the case study can be downloaded from AP Central, along with related teachers' resources.

Released Exams in each AP subject are published every four or five years, on a staggered schedule. The *AP Computer Science A and Computer Science AB Released Exams* book contains complete copies of both the A and AB exams, including multiple-choice questions and answers. Released Exams also include free-response questions, along with a description of the scoring process, examples of students' actual responses, scoring standards, and commentary that explains why the responses received the scores they did. You can purchase Released Exams at the College Board Store: <http://store.collegeboard.com>.

Note: The 2004 *AP Computer Science A and Computer Science AB Released Exams* book contains the first Java versions of the exams.

Chapter 2

Sample Syllabi for AP Computer Science are available on AP Central. Teachers in secondary schools have written most of the syllabi, but since AP courses cover college-level material, syllabi from college professors are also included. An additional collection of six syllabi is available for purchase at the College Board Store.

AP Coordinator

Each participating school designates an AP Coordinator who takes primary responsibility for organizing and administering that school's AP program. The AP Coordinator may be a full- or part-time administrator, counselor, or faculty member. AP Coordinators manage the receipt, distribution, administration, and return of AP Exam materials.

AP teachers and the AP Coordinator work closely together throughout the academic year. Early in the spring, AP teachers consult with the Coordinator to help determine the correct number and type of exams that need to be ordered. During the exam administration weeks, Coordinators may designate AP teachers to serve as proctors for exams in a subject area other than the one they teach.

Coordinators are the bridge between AP teachers, students, and administrators and the AP Program. Questions about exam fees, dates and deadlines, and exam-specific policies (such as the procedures for accommodations and for handling conflicts and make-up exams) should be directed to the AP Coordinator.

Regional Offices

The College Board maintains six regional and three State Services offices to serve students and educators. Your questions and comments regarding College Board programs and services should be directed to these offices. Each office provides information and features specific to its region of the country. Go to the inside back cover of this Teacher's Guide for contact information for your regional office.

Real Teachers, Real Advice

Before the First Day of Class

Making the First Decisions

New teachers should start with the following activities. If you have inherited an AP Computer Science program from another teacher, some or all of these steps may have been done for you.

- **Choose your textbook:** There are many good computer science textbooks. Some were specifically written for AP Computer Science courses; others may or may not be appropriate for your AP classes. It is important to choose a textbook that presents and uses objects (not just the built-in Java classes) in one of the first chapters. (In order to think in an object-oriented way, students need lots of practice using and modifying classes, and the sooner they begin, the better.) Any textbook can be supplemented for use with AP Computer Science, but the order of presentation is more critical than an omitted topic or two. Chapter 5 lists several textbooks that are appropriate for an AP CS course; most of those textbooks are reviewed on AP Central.

[AP Central > Teachers' Resources tab > Choose Course and enter Title and/or Author's Name to find the review.]

- **Choose an IDE:** An Interactive Development Environment (IDE) is software that facilitates program development. Although it is not absolutely necessary for the AP course, you will probably want to use an IDE with your students. This software allows users to type source code (often providing some assistance like code completion), compile each class (with help locating errors), group related classes into a project, and run the project, all from the same window. Some IDEs have special features like on-screen diagrams that show the relationships of classes within a project or access to the application program interface (API) from within the IDE.

You can provide more than one IDE; some are simpler than others, and you may wish to progress from one to another during your course. You don't have to decide everything at the beginning of the year, but it will save lab setup time if you can do it all at once. (See the section on Setting Up Your Lab, below, for more timesaving suggestions.)

Chapter 5 has a list of several IDEs that are commonly used in AP CS classes. Each of the sample syllabi in chapter 3 specifies which IDEs are used by the instructors of those courses.

- **Decide how to handle user input:** While user input is not part of the AP Java subset (see the AP Computer Science Java Subset appendix in the Course Description), and it will not be tested on the AP Exam, your assignments will probably require the processing of input from users.

Because Java's console input mechanism is fairly complex, many textbook authors provide a simplified console input class. (A few of these are listed in the Software section of chapter 5.) Or you might prefer to use a pop-up (graphical) input box.

Chapter 2

I start off by using `JOptionPane.showInputDialog()` in `javax.swing`, which is as simple an I/O (input/output) as I need. It returns a `String`, and parsing the `String` appropriately is a generally useful exercise for beginners. You do not have the overhead of the `BufferedReader` and `Exceptions`.

In the past, I used different simple I/O classes that came with the textbooks that were required for the course, but I found they really did not simplify things for me because some explanation still was needed about what was going on.

Using `JOptionPane.showInputDialog()` allows me to concentrate on the big ideas about input values—what they are and how to work with them—without worrying about topics such as `Streams` and `Exceptions` until later in the course. I do come back and talk about console I/O, generally at the same time as when I discuss file I/O. (I also use the File Chooser Dialog box GUI, which removes the problems of where to put files to be used by the application.)

You can mix the GUI dialog box with a console app with only the overhead of having to wait for the dialog box to appear. Students are very familiar with using dialog boxes (more than they are with command line input, in my experience).

—Don Slater, Carnegie Mellon University, Pittsburgh, Pennsylvania,
writing to the AP Computer Science EDG

Setting Up Your Lab

- Download and install the Java Development Kit (JDK) from Sun. (See Tech Support in chapter 5.) Most IDEs require that Java be installed first.
- Download the Java API, as well as Owen Astrachan’s AP version, which includes only the classes and methods that will be tested on the AP Exam, along with some AP-specific commentary (www.cs.duke.edu/csed/ap/subset/doc). The Java API can be accessed online, but many instructors prefer to store a local copy on their lab computers, to reduce network traffic and minimize reliance on Internet access.
- Download and install the IDE.
- Download and install the MBS case study. Some IDEs have specific requirements for setting up the case-study project. Appendix A lists sources of instructions for setting up the case study in several different environments.

Tip: Set up one lab computer with the software that you expect to use, and keep track of what you’ve done. Test, test, test!

- Using your IDE, create a simple project and run it.
- Set up and run the case study.
- Add shortcuts to the desktop or menu for the IDE, API(s), and any other resources that your students will need regularly. My lab computers have shortcuts for two IDEs, the complete Java API, the AP API, and the MBS API.
- Once you are satisfied, repeat the steps for the remaining computers. If possible, use cloning software to copy an image of the computer that you have set up, and send it to the other machines in your lab, making them identical. Talk with your network administrator about this option.

Planning Your Course

Chapter 3 offers suggestions for organizing your AP Computer Science course, along with several syllabi from teachers of AP CS or similar college courses. You will definitely want to have a “big picture” plan

before the first day of class, but reality dictates that you maintain the flexibility to adjust as you go, depending on how much time your students need to master each topic and complete each assignment.

You have completed the first steps; now it's time to think about some of the finer points of teaching.

Teaching Strategies

Help Students Develop a Good Programming Style

In addition to teaching our students how to design and program solutions that work, we can also help them establish positive habits regarding the style of their code. Good programming style will make students' work easier to debug and modify, not to mention easier to grade.

As a former math teacher who spent several years as a professional programmer and then returned to the classroom to teach computer science, I was initially unsure of the extent to which I should impose my own compulsive ideas about programming style on my students. Just as some students don't like to "show their work" in mathematics classes, some computer science students resist indenting code, choosing meaningful variable names, and other techniques that require extra typing but make their code easier to decipher. I was reassured when I attended my first computer science educators' conference and heard seasoned teachers make assertions like, "I refuse to grade un-indented code."

Rules of style generally cover aspects of programming such as the following:

- Indentation
- Position of curly braces
- Meaningful identifier names for classes, methods, and variables
- Variable initialization and use of named constants
- Comments (method preconditions and postconditions, general class comments)
- Use of "white space" (spaces and blank lines). I often tell my students, "White space is free—use it liberally!" (I use "paperless" grading—see chapter 5—so it truly *is* free.)

You'll probably want to develop your own rules for your students—even among professional programmers, there are style issues that cause heated debates—but first you might want to read some general thoughts on style, as well as specific guidelines from these authors and educators:

Fran Trees, whose syllabus appears in chapter 3, establishes "Company Rules" for her students, in much the same way that many employers do. (www.users.drew.edu/ftrees/Csci6/General_Information/CompanyRules.htm)

In "17 Bits of Style," Maria Litvin and Gary Litvin state, "Style is a crucial component of professionalism in software development. Clean code that follows stylistic conventions is easier to read, maintain, and share with colleagues. Programmers who code in good style are less likely to have silly bugs and will actually spend less time developing and debugging their code. Finally, good style in programs is a concern because for us humans, style and aesthetics are a concern in everything we do."⁸

8. Maria Litvin and Gary Litvin, *Java Methods: An Introduction to Object-Oriented Programming*, Appendix B (Andover, Mass.: Skylight Publishing, 2001). ("17 Bits of Style" is also separately available online at www.skylit.com/javamethods/appxb.pdf.)

Chapter 2

Thornton Rose, a contract software developer, has written two articles about “Good Java Style” (parts 1 and 2): www.developer.com/java/other/article.php/600581 and www.developer.com/java/other/article.php/600651.

Use a Variety of Teaching Techniques to Engage Students

Students would rather write programs than listen to lectures, but we have to teach them before they can write programs. Very few teachers can lecture for an entire class period without losing their students’ attention, so how can we keep students engaged in their learning?

Play with toys: Some of the most difficult computer science concepts can be illustrated using simple toys. Stacking cups or nesting dolls can demonstrate recursion; those same stacking cups can be lined up and sorted by size using various algorithms. Pop-it beads or carabiners can illustrate linked lists.

Be dramatic: Role-playing can help familiarize students with “object” concepts and method-calling syntax, as well as complex interactions between objects of various classes.

Order your students around: Outline an array with masking tape on the floor and have students (literally) step through the process of being sorted. Outline an extra “temp” box (for swapping) to the side. (**Suggestion:** Write names or numbers on large nametags, rather than sorting by height. I once had three students of nearly identical height in a demonstration, resulting in a lot of quibbling and confusion. Also, some students are sensitive about their height.)

Be animated: Algorithm animation Web sites are terrific for classroom demonstrations; most allow the user to control the process one step at a time. (Chapter 5 lists several algorithm animation Web sites.)

The following Web sites offer general suggestions as well as specific techniques for engaging students in computer science classes:

Becker, Katrin. *Making CS More Fun*. University of Calgary, 2001.
<http://pages.cpsc.ucalgary.ca/~becker/Main/MakeCSfun.html>

Bergin, Joseph, and Myles McNally. *Non-Programming Resources for an Introduction to CS*. ITiCSE 2000.
<http://csis.pace.edu/~bergin/iticse2000/>

Levine, David, and Steven Andrianoff. *Role Playing in an Object-Oriented World*. St. Bonaventure University, Department of Computer Science, 2003.
<http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html>

McConnell, Jeffrey. *Active and Cooperative Learning*. Canisius College Computer Science Department, 2001.
www-cs.canisius.edu/~mccannel/active_learning.html

Encourage Participation in Computer Science

If your roster is similar to those of most computer science teachers, there aren’t a lot of girls in your AP Computer Science classes. You may also have found that other groups are underrepresented. The College Board encourages open enrollment for all AP courses, as explained in the Equity and Access section in chapter 1.

Gender equity is a topic of much discussion among computer science educators; the challenge of attracting females to the field is nearly universal. The percentage of girls in AP Computer Science is much lower than in any other AP mathematics or science course. In fact, in 2004, among all students who took AP Computer Science Exams, the percentage of females was lower than for any other AP subject (16 percent for AP CS A and 11 percent for AP CS AB).⁹ Percentages were similar for prior years.

Considerable research has been conducted in an attempt to explain why so few girls take computer science courses. It is certainly not due to differences in ability; research into the cognitive differences in males and females suggests that boys and girls are equally capable of performing well in mathematics and computation despite a difference in style of learning. Yet females often perceive themselves to be less naturally inclined than males toward mathematics and computer science.¹⁰

How can we explain this difference in motivation between males and females? Is it because of lack of information or misconceptions about computer science?

Perceptions about the field, both as a discipline of study and in the world of work, aren't always accurate. Below, Kelly Keenan analyzes some of the ways in which girls may differ from boys in their interests, needs, and learning styles. Kelly is the director of Academic Computing and the Women in Science and Engineering Program at Westover School in Middlebury, Connecticut. She has some terrific advice for attracting and retaining girls in computer science classes—advice that could be used to attract students from any underrepresented group.

I teach at an all-girls' school, so I have a lot of experience with girls. Most of my suggestions are based on broad generalities about differences between girls and boys, in an effort to analyze why some traditional methods of teaching computer science don't work for most girls. However, we need to look at children and teens individually, rather than as a group. We can make generalizations, but not all girls are alike, any more than all boys are alike. Once we learn to teach to each individual's needs, we will reach the entire class.

Background. Many girls don't have basic knowledge of how computers work. They use a computer to write their history paper, or do their physics homework, or communicate with friends, but they don't "play" with it.

I offered a half-year course where students learned how to install Ethernet cards and memory, talk "computerese," shop for a computer, identify and discuss different operating systems, solve discrete and boolean mathematical equations, and build their own Web pages. Girls came back from break and told me things like, "I talked with the guy next to me on the plane about computer operating systems," "I helped my grandparents buy a new computer," and "I taught my friend about boolean equations."

Offer an introductory programming class that uses a simple language—my students liked *Karel J Robot*—because girls often think that doing math problems is boring if the problems don't directly relate to something in the real world. If you can't have a full-term class, ask a math teacher to give you a week for programming.

9. College Board, AP Program Summary Report 2004. Available at apcentral.collegeboard.com/document_library, on the "Exam Data" page.

10. Phoenix Moorman and Elizabeth Johnson, "Still A Stranger Here: Attitudes Among Secondary School Students Towards Computer Science," presented at *ITICSE'03*, June 30-July 2, 2003, Thessaloniki, Greece (ACM, 2003): 193-97. <http://doi.acm.org/10.1145/961511.961564>.

Keeping them engaged. If you can attract girls and keep them in your class, they will help recruit more girls, by talking about computer science outside of class. How do you keep them interested?

1. **“Cool” assignments.**

- Compile address books: You can do nifty sorting and searching routines with them.
- Design quiz games: We used sample questions from AP Computer Science, history courses, calculus—whatever other AP classes students were taking that year. The girls used this tool to study for their upcoming exams, and we put the final product on the server so that other students could use it, too.
- Write an application that relates to another subject: One year everyone was in chemistry and learning the periodic table, so we wrote an application that would help students find the information they needed. If they see some real-world value in the program that they are writing, both girls and boys will be more interested in it. Start with a very simplified version, and increase its functionality as you go along. (Interdisciplinary projects earn big points in many schools.)

Consider offering a choice of assignments that use the same concepts but in different contexts: a mathematical function, a physics problem, or a social problem.

2. **Group projects.** Working in pairs helps build confidence. Let girls and boys partner sometimes, but also have girls pair up with each other.

One of my favorite things to do is a whole-class project. We determine all the data types in class and write the class declarations together. Then students pick which classes or methods they want to implement. As the teacher, I am the project/engineering manager and make sure that all the parts talk to each other. In the end, students learn how software is written in the real world.

3. **Sharing outside of class.** The best way for girls to find out about the “cool” stuff you are doing in class is to find a way to share it with the entire school. This works best for a whole-class project if it is something of which the entire class can be proud. Girls from class can show other girls what parts of the program they did.

4. **Reduced risks.** I always ask my girls, “How many of you have ever taken apart a computer?” No one raises her hand. Then I ask, “How many of you have watched your brother or another guy take apart a computer?” Almost every girl raises her hand. I think that girls often don’t want to take things apart because they are afraid of getting into trouble or breaking something. Many boys don’t think about the consequences until their parents are standing over them, or they think about them but are willing to take the risk anyway.

When my students were installing cards, I taught them how to avoid static charges and what would happen if they didn’t. At the same time, I was also telling them that if any of the machines didn’t work afterwards, it was my responsibility and I would fix them. Sometimes students write code that crashes the computer; they need to understand that they aren’t going to get in trouble for this. I’ve found that girls are apt to blame themselves, while boys tend to blame external factors.

5. **Vocabulary.** Girls may not have a lot of computer vocabulary when they get to your class. Make the assumption that *no one* in the class understands the words you are saying. With boys in the classroom, girls may not ask if they don’t understand something. They will listen for the context and hope that they can figure out what you are talking about.

Provide vocabulary lists with definitions of technical terms that you use in class. You may find that students share them with students who are not in your class.

6. **Role models.** Invite successful young women in the field to give short talks to your students.

I think many classes are missing out not only on more girls, but on some of the boys who just need a teacher or parent to say, “Why don’t you try this and see if you like it? I know you can do it.” Who couldn’t use a little encouragement from time to time? I didn’t join the school band until the music teacher came to me and said, “I hear you have study hall during band and that you can read music. I need someone to play the chimes and xylophone. I expect to see you in the next class.” I went because of the encouragement; I stayed because I was learning and it was fun.

Kelly Keenan’s informal observations and advice align well with findings and suggestions made by Jane Margolis, Allan Fisher, and Faye Miller, computer science faculty members at Carnegie Mellon University, who interviewed both male and female students in an effort to determine whether women approach the study of computer science differently from men. They found that while most of the male students describe an early and persistent magnetic attraction between themselves and computers, women much more frequently link their computer science interest to a larger societal framework. The authors discuss the importance of “re-visioning” computer science, so that the answer to the question “What is computer science?” incorporates and values women’s perspectives as well as men’s. They offer concrete suggestions for developing a more contextual approach to computer science, including the creation of an “immigration course” for new students, to expose them to a wide variety of computer science issues and applications; interdisciplinary work with multifaceted problems; and a “CS in the Community” course that engages students with nonprofit groups in the local community.¹¹

How can you motivate students to find out more about computer science?

- Prepare a flyer or presentation that describes computer science and your school’s courses. You may want to emphasize abilities and interests that suggest enjoyment and success in computer science (logic puzzles, organizational skills, math ability) as well as career advantages, such as flexible working conditions. You might also explain what skills aren’t prerequisites (hardware knowledge, game expertise, etc.); many students assume that they would be at a disadvantage in a computer science course because they don’t have certain skills or interests. Share your enthusiasm with students in math classes and with parents and guardians at AP registration meetings or school open houses.
- Ask math teachers to identify students who have strong math ability, especially those from underrepresented groups. Contact those students individually, in passing, or through a letter or e-mail, encouraging them to consider taking a computer science course.
- If students are currently having success in an introductory computer science course, ask them to continue with the AP course; they may need extra encouragement.
- Offer a “Girls’ Technology Day” or a summer workshop for girls, familiarizing them with tools that might be especially motivating for girls (Web page development, cooperative activities, graphical programming, and projects that involve communication).

11. Jane Margolis, Allan Fisher, and Faye Miller, “Caring About Connections: Gender and Computing” (Carnegie Mellon University, 1999). www-2.cs.cmu.edu/~gendergap/papers/IEEE99.html

Grading Students' Projects

Your students will spend considerable time completing their programming assignments, and you will want to develop effective methods for evaluating their work and providing feedback. Learning to design and write programs is a gradual process, and your comments will help students refine that process as they proceed through your course. Effective project evaluations address more than just correctness; design, style and documentation, and efficiency are also important. It's important to praise the positive things when offering feedback on students' work; this will help instill confidence and encourage an upbeat attitude in the class. Other suggestions for motivating and encouraging students can be found throughout this *Teacher's Guide*.

Correctness	Program should conform to specifications stated in the problem statement. Program should demonstrate correct handling of special cases and error conditions. Test data should include typical values, out-of-range values, boundary values, and special case values.
Design	Each class should consist of small, coherent, independent methods. Classes should be loosely coupled.
Style and Documentation	Program should be easy to read and understand. (See section on programming style, above, for details.)
Efficiency	Algorithms should be chosen with regard to efficiency of both time and space. Algorithms should be “elegantly” written rather than written with “brute force.”

Many teachers develop scoring guidelines to address each of these areas, dividing the points among the categories. Instructor materials found in some textbooks offer scoring guidelines for the assignments in each chapter.

Clarifying your expectations can eliminate a lot of confusion about your scoring standards. As I explain to my students, “But it works!” justifies only 25 percent of the total possible points.

After you have taught AP Computer Science for a few years, you might want to consider applying to be an AP Exam Reader. (See chapter 5 for more information.) Participating in this process can give tremendous insight into developing your own scoring guidelines, and the opportunity to share ideas with other AP colleagues is invaluable.

Connecting with Other Teachers and Parents

You are very likely the only AP Computer Science teacher at your school (and possibly the only computer science teacher), so you will need to make an effort to connect with other teachers in your field. After you have joined the EDG on AP Central, it's a good idea to read the messages on a regular basis. You'll find answers to questions that you have been meaning to ask, as well as some that you didn't know you should be asking. When you encounter a problem or need some suggestions for teaching a specific topic, first check the archives—if you don't find your answer there, don't hesitate to ask for help. There is a whole community just waiting to respond.

Attend a conference: Information about some national organizations that offer conferences is in chapter 5. Conferences enable you to make connections with other teachers, hear inspiring speakers, and pick up great ideas all at the same time.

Advice for AP Computer Science Teachers

At a school open house or in a mailing, tell your students' parents or guardians about the AP Computer Science course; many of them don't really understand what computer science is. Mention that you'd like to let your students know about career opportunities in technology. Perhaps one of the adults works for a company that would send a guest speaker or arrange for a field trip to its site.

- Make the most of every single precious minute of class time. If your students have good attendance, you may not need to give a lot of homework. Most of my students have a big course load in which homework is a necessity—they also may have jobs and participate in sports or other activities. Assuming good attendance, I can schedule time for labs, assignments, and tests, and they can spend a couple of hours a week at home going through a test-prep book or studying for my next test.
- Some students prefer to go at their own pace. Some need more handholding. On lab days, I generally will walk a group of students through a lab at a slower pace and let the others fly. Along the way, I try to help everyone become more autonomous.
- Change gears if things aren't working in your class. Abandon your plans, and meet the students' needs. Be flexible. Hold them accountable, but be realistic in your expectations. Smile. Have fun. You shouldn't be doing this unless you love it (most days, at least!).

—Jill Kaminski, Chaparral High School,
Parker, Colorado

Chapter 3

Course Organization

Syllabus Development

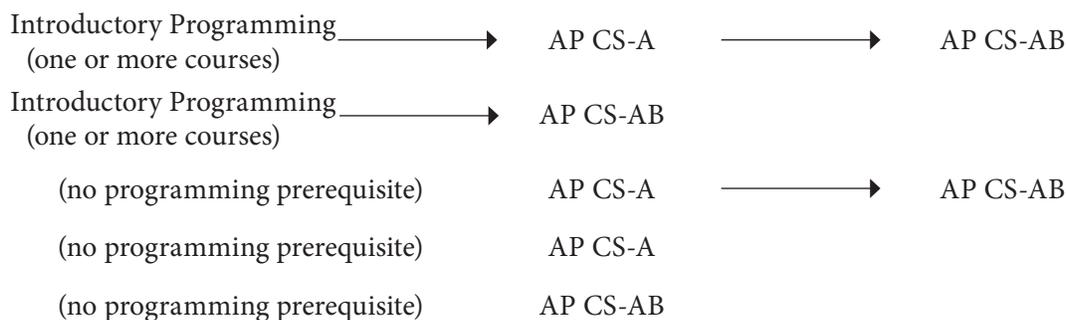
You have read the advice—now it’s time to begin planning your own AP Computer Science course. While you do not have to decide every last detail before the first day of school, you will certainly want to have an outline and timeline for the topics to be covered in the course.

Here is some important advice from the “Teaching Tips and Strategies” article on the AP Computer Science Home Page on AP Central:

“Use your textbook as a guide to the study of computer science, but **keep the AP Computer Science Course Description**, particularly the Topic Outline and the Java Subsets, **in sight at all times**. Regardless of the textbook chosen, it will be necessary to modify the content and the order of presentation to meet the complete AP CS course requirements. . . . Refer to the Topic Outline and Java Subsets regularly to ensure that you are covering the testable material.”

Textbook authors sometimes include a syllabus, pacing and lesson plan guide, scope and sequence, and/or options for reordering or skipping chapters. A few are mapped to the AP curriculum. Explore the teachers’ resources that accompanied your textbook, including any Web site that the author or publisher provides for teachers who adopt the textbook.

Your course outline will depend in part on your students’ previous programming experience. Your school’s programming/CS sequence probably looks like one of these:



Depending on which AP CS topics (if any) your students have covered in previous courses, you may be able to pass quickly through some of the earlier topics with a short review and quick assessment of their mastery.

Here is more guidance from “Teaching Tips and Strategies”: “**Emphasize Object-Oriented Programming** (OOP) throughout the year in both [Computer Science] A and AB classes. Begin the course by presenting students with mostly complete programs and allowing students to complete small portions of

them. As the student’s knowledge grows, increase the amount of work the student is required to complete in the project. Within a relatively short time students will construct complete solutions on their own. By modeling good OOP design in the early projects, you teach the students to create good solution structures and prepare them for success in the field of computer science.”

One approach some teachers have found successful is incorporating the case study throughout the course.

I used to plan my course around a list of the AP CS Subset and topics: if statements, loops, methods, classes, searching, sorting, case study, etc., etc. I found that when I used this approach, my students’ brains were like shift registers. They could retain the content during the unit of study, but it seemed necessary to “shift” it out in order to store new information. For the average student, overall retention of course concepts in May wasn’t great. And the AP CS Exam isn’t the kind of test you can cram for.

I then followed the sage advice of former Chief Reader Chris Nevison: “Don’t **teach** the case study! **Use** the case study to **teach computer science**.” This advice also appears in the case-study teacher’s manual: “The case study and the accompanying teacher’s manual were designed in such a way that you can use these materials throughout the course. You may, in fact, wish to teach many computer science concepts from the AP CS curriculum through the case study itself.”

So that’s what I do, beginning in early November. And it’s fun! I introduce the case study in a positive way, and we continue to find creative things to do to bugs and other creatures. My students appreciate that this is more of a “real” program and less of the kind of program usually written in beginning computer science classes (e.g., a student class with a driver). It’s well-designed, there’s enough there to hold interest, and it’s particularly effective for teaching inheritance and data structures.

—Jill Kaminski, Chaparral High School,
Parker, Colorado.

For complete details of how Jill uses the case study in her course, refer to appendix A.

Six Sample Syllabi

The syllabi in this section were contributed by teachers from a variety of educational settings: high school and college, public and private, with varying populations and budgets. You may find one or more that fit your style and your students’ degree of preparation, or you may decide to mix elements of several syllabi. You may also need to adjust the timing of the course to fit your school’s calendar.

Be sure to read through all of the sample syllabi, even those that don’t match your needs for timing or content, because each teacher shares his or her general teaching philosophy and strategies, as well as specific projects and resources.¹² This chapter includes a rich collection of gems; mine it thoroughly!

12. Newer editions of many of the texts mentioned in the following syllabi are now available: see chapter 5, Resources for Teachers.

Important Note: The AP Course Audit

The syllabi included in this Teachers Guide were developed prior to the initiation of the AP Course Audit and the identification of the current AP Computer Science Curricular Requirements. These syllabi contain rich resources and will be useful in generating ideas for your AP course. In addition to providing detailed course planners, the syllabi contain descriptions of classroom activities and assignments, along with helpful teaching strategies. However, they should not necessarily be used in their entirety as models that would be authorized under the guidelines of the AP Course Audit. In particular, these syllabi were developed while the Marine Biology Simulation was the current case study for AP Computer Science, so they do not incorporate the most recent case study (GridWorld), which will come into use for the 2007-08 academic year.

To view the current AP Curricular Requirements and examples of syllabi that have been developed since the launch of the AP Course Audit and therefore meet all of the AP Computer Science Curricular Requirements, please see AP Central.

<http://apcentral.collegeboard.com/courseaudit/resources>

Sample Syllabus 1 (AP Computer Science A)

Bekki George

James E. Taylor High School

Katy, Texas

rebeccageorge@katyisd.org

<http://schools.katyisd.org/campus/ths/index.htm>

School Profile

Location and Environment: James E. Taylor High School is located 25 miles west of downtown Houston in a suburban area near Katy, Texas, a town with a population of 12,000. The Katy Independent School District itself encompasses a large area consisting of approximately 65,000 households and 41,500 students. The socioeconomic level is predominantly middle to upper-middle class.

Grades:	9–12	
Type:	Public school	
Total Enrollment:	2,770	
Ethnic Diversity:	Asian American	10.0 percent
	Hispanic/Latino	7.2 percent
	African American	3.4 percent
	Native American	0.1 percent
College Record:	71 percent attend four-year colleges	
	17 percent attend community colleges	

Personal Philosophy

I want my students to enjoy computer science as much as I do. I get very excited when they understand what I'm teaching. Every day, I hear "All right!" or a squeal of delight when one of my kids gets something difficult to work. Their enthusiasm is contagious. My reward comes when students can apply the programming tools they have learned to real-life examples on their own. Computer science is more than just programming. Students should leave my class with a clear understanding of Java and the ability to adapt to any new programming language that they are taught in college. I want them to have the confidence to tackle any problem-solving obstacles they encounter.

Class Profile

Five sections (27 students per section)

Classes meet for 50 minutes each school day.

Course Overview

AP Computer Science 1 (the official name for my course) is both a college-prep course for potential computer science majors and a foundation course for students planning to study in other technical fields such as engineering, physics, chemistry, and geology. The course emphasizes programming methodology, procedural abstraction, and in-depth study of algorithms, data structures, and data abstractions, as well as a detailed examination of a large case study program. Instruction includes preparation for the AP Computer Science A Exam.

Chapter 3

Texts

Bergin, Joseph et al. *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Redwood City, Calif.: Dreamsongs Press, 2005. <http://csis.pace.edu/~bergin/KarelJava2ed/Karel%2B%2BJavaEdition.html>. Introduces objects and inheritance.

College Board. *AP Marine Biology Simulation Case Study*. New York: College Entrance Examination Board, 2002. Download from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

Horstmann, Cay. *Big Java*. Hoboken, N.J.: Wiley, 2002.

Lambert, Ken, and Martin Osborne. *Fundamentals of Java, Comprehensive Course*. 2nd ed. Boston: Course Technology, 2002.

Course Planner

The Resources listings include the following text references: *Karel J Robot* (KJR), *Marine Biology Simulation Case Study* (MBS), *Big Java* (BJ), and *Fundamentals of Java* (FJ).

Unit (Weeks)	Title, Topics, and Student Objectives	Resources, Assessments, and Strategies
1 (0-3)	Karel J Robot Topics: <ul style="list-style-type: none">• Objects• Classes• Looping• Conditionals Objectives: <ul style="list-style-type: none">• Write and use simple classes with Karel J Robot• Learn the basics of conditionals and looping	Resource: <ul style="list-style-type: none">• KJR Assessments: <ul style="list-style-type: none">• Program-specific tasks for Karel• Create a SmartRobot Class to teach Karel more commands: turnRight(), turnAround(), climbStair().• Clear a field of beepers (using loops).• Redistribute a field of beepers (using loops and conditionals).• Run a hurdle race:<ul style="list-style-type: none">○ same height and equally spaced;○ same height and unequally spaced;○ different heights and unequally spaced.
2 (4)	Java Basics Topics: <ul style="list-style-type: none">• Java basics• Using the compiler• Input and output Objectives: <ul style="list-style-type: none">• Understand terminology: compiler, IDE, JVM• Edit, compile, and run a simple program in Java• Understand the different compile time errors, runtime errors, and logic errors• Use BufferedReader for input• Use output with System.out and format output to look nice	Resource: <ul style="list-style-type: none">• FJ: lesson 3, Critical thinking Assessments: <ul style="list-style-type: none">• Labs: Triangle, Rectangle, Square: Area, and perimeter program• Get input for the registrar's office program. Strategies: <p>Assign a lot of small programs that illustrate different types of input and output—make sure students have used every type of input and displayed it in different ways.</p>

Unit (Weeks)	Title, Topics, and Student Objectives	Resources, Assessments, and Strategies
3 (5)	<p>Defining Variables, Arithmetic Expressions</p> <p>Topics:</p> <ul style="list-style-type: none"> Using and understanding variables Comments Arithmetic expressions in Java programs <p>Objectives:</p> <ul style="list-style-type: none"> Understand terminology: comments, variables, constants, reserved words, literals Declare and initialize variables and constants in Java Understand mathematical expressions in Java and their precedence Use casting to make their data more accurate Use the assignment operator correctly 	<p>Resource:</p> <ul style="list-style-type: none"> FJ: lesson 3, Projects <p>Assessments:</p> <p>Labs:</p> <ul style="list-style-type: none"> Paycheck program; have employee information entered and calculate pay. Modify the paycheck program to also include any overtime hours in the calculations. <p>Strategies:</p> <ul style="list-style-type: none"> Students need practice with how the different types, double and int, relate when they are used in mathematical operations. Worksheets are helpful here. Present a lot of small program examples in which they have to find the errors.
4 (6–7)	<p>Introduction to Classes and OOP</p> <p>Topics: Creating and using classes</p> <p>Objectives:</p> <ul style="list-style-type: none"> Understand terminology: constructor, accessor, mutator, instance variable, and more Understand the difference between public and private access in a class Use and comprehend the DecimalFormat class and the Random class Write classes from scratch 	<p>Resource:</p> <ul style="list-style-type: none"> BJ: chapter 3 <p>Assessments:</p> <p>Labs: Purse class and StampMachine class</p> <p>Strategies:</p> <ul style="list-style-type: none"> Go slowly and show as many examples as possible. Start with very simple examples. Give students classes to complete.
5 (8–12)	<p>Conditionals and Looping</p> <p>Topics: if, if-else, while, for</p> <p>Objectives:</p> <ul style="list-style-type: none"> Understand terminology: control statements, counter, infinite loop, iteration, nested loops, logical operators, truth tables Construct syntactically correct loops and conditional statements Understand the different errors that may occur with loops Use logical operators to make programs more robust Construct truth tables 	<p>Resources:</p> <ul style="list-style-type: none"> FJ: lessons 4 and 6, Projects <p>Assessments:</p> <p>Labs:</p> <ul style="list-style-type: none"> Approximate PI using Leibniz’s method Base Conversion; convert from base 10 to base 2 Guess My Number game Euclidean algorithm program Perimeter and area of rectangles using all combinations of certain range <p>Strategies:</p> <ul style="list-style-type: none"> This unit needs a lot of programs. Students need practice writing different types of loops and conditionals. Worksheets that focus on the output of certain statements are very helpful here.

Chapter 3

Unit (Weeks)	Title, Topics, and Student Objectives	Resources, Assessments, and Strategies
6 (13–14)	<p>The String Class</p> <p>Topic:</p> <ul style="list-style-type: none"> String class <p>Objectives:</p> <ul style="list-style-type: none"> Instantiate String objects Understand that Strings are immutable Use appropriate String methods to solve problems 	<p>Resource:</p> <ul style="list-style-type: none"> FJ: lesson 10.1 <p>Assessments:</p> <ul style="list-style-type: none"> FJ: exercise 10.1 Lab: LineEditor Class (<i>AP CS Course Description</i>) <p>Strategies:</p> <p>Work several examples using the substring method.</p>
7 (15–17)	<p>ArrayList</p> <p>Topic:</p> <ul style="list-style-type: none"> Using ArrayList class <p>Objective:</p> <ul style="list-style-type: none"> Use the ArrayList methods 	<p>Resources:</p> <ul style="list-style-type: none"> FJ: lesson 10.7 BJ: 13.1 and 13.2 <p>Assessments:</p> <ul style="list-style-type: none"> BJ: exercise p.13.1 WordList (2004 AP CS A Exam, Free-Response Question 1, AP Central) <p>Strategies:</p> <ul style="list-style-type: none"> Stress the difference between add and set. Draw pictures of the ArrayList after add, set, and remove have been performed.
8 (18)	<p>Arrays</p> <p>Topics:</p> <ul style="list-style-type: none"> Declaring and initializing arrays Manipulating arrays with loops Creating parallel arrays <p>Objectives:</p> <ul style="list-style-type: none"> Understand terminology: array, element, index, logical size, physical size, parallel arrays Declare one-dimensional arrays in Java Use initializer lists when declaring arrays Manipulate arrays using loops and array indices Use the physical and logical size of an array together to guarantee they do not go beyond the bounds of their array Understand how parallel arrays can be useful when processing certain types of data Work with arrays of primitive data types as well as arrays of objects 	<p>Resource:</p> <ul style="list-style-type: none"> FJ: lesson 8, Projects <p>Assessments:</p> <p>Lab: For one-dimensional arrays, read in numbers and place each one in an even, odd, and/or negative list.</p> <p>Strategies:</p> <ul style="list-style-type: none"> Students need practice manipulating loops that work with arrays. Worksheets where they show the output from sample programs are always helpful here. Students also need to be reminded about the indexing of arrays beginning at zero.

Unit (Weeks)	Title, Topics, and Student Objectives	Resources, Assessments, and Strategies
9 (19–21)	<p>Searching and Sorting Arrays</p> <p>Topics:</p> <ul style="list-style-type: none"> ● Bubble, Selection, Insertion sorts ● Sequential and Binary searches <p>Objectives:</p> <ul style="list-style-type: none"> ● Write a method for searching an array ● Perform insertions and deletions at given positions in arrays ● Trace through sorting and searching algorithms ● Understand the algorithms behind each of the following searching and sorting techniques: bubble, selection, and insertion sorts; sequential search and binary search ● Understand the efficiency of each sort and search and when it is desirable to use each one 	<p>Resource:</p> <ul style="list-style-type: none"> ● FJ: lesson 10 <p>Assessments:</p> <p>Lab: Students make their own “utility” class that includes all of these sorts and searches.</p> <p>Strategies:</p> <ul style="list-style-type: none"> ● Students need practice tracing through sorts and searches. ● Worksheets: show the sort or search at a certain “pass.” ● Students also do well with a worksheet that talks about the efficiency of each of the strategies they have learned, efficiency for a sorted versus unsorted list, and “best,” “worst,” and “average” efficiency. <p>(See appendix B for lab and worksheets.)</p>
10 (22–24)	<p>MBS (chapters 1–3)</p> <p>Topics:</p> <ul style="list-style-type: none"> ● Experimenting with a large program ● Using classes ● Modifying classes <p>Objectives:</p> <ul style="list-style-type: none"> ● Run the case study and analyze output ● Experiment with the Simulation ● Understand the Fish Class, Simulation Class, and the Environment Interface ● Modify the Fish Class 	<p>Resource:</p> <ul style="list-style-type: none"> ● MBS: chapters 1–3 <p>Assessments:</p> <p>Exercises and analysis from the text</p> <p>Strategies:</p> <ul style="list-style-type: none"> ● Read the manual thoroughly. ● Be familiar with all the classes and interfaces discussed.
11 (25–27)	<p>More on Classes, Inheritance, Interfaces</p> <p>Topics:</p> <ul style="list-style-type: none"> ● Classes ● Inheritance ● Abstract classes ● Interfaces <p>Objectives:</p> <ul style="list-style-type: none"> ● Demonstrate inheritance by extending a class ● Understand polymorphism and know when it is appropriate to override methods in a super class ● Create and extend an abstract class ● Implement an interface 	<p>Resources:</p> <ul style="list-style-type: none"> ● BJ: chapter 11 ● FJ: lessons 9.5 and 9.6 <p>Assessments:</p> <ul style="list-style-type: none"> ● Create an abstract Shape class. ● Pet Parade (2004 AP CS A Exam: Free-Response Question 2, on AP Central) <p>Strategies:</p> <p>Draw pictures of the inheritance hierarchy.</p> <p>Note: This unit could be moved to after unit 12 if you wish to use the MBS to introduce inheritance.</p>

Chapter 3

Unit (Weeks)	Title, Topics, and Student Objectives	Resources, Assessments, and Strategies
12 (28–29)	MBS (chapter 4) Topic: <ul style="list-style-type: none"> Inheritance Objective: <ul style="list-style-type: none"> Use inheritance to extend the Fish Class 	Resource: <ul style="list-style-type: none"> MBS: chapter 4 Assessments: Exercises and analysis from the text Strategies: <ul style="list-style-type: none"> Have fun with this chapter. Allow the students to be creative after working through the exercises and analysis. Create different kinds of fish or other objects.
13 (30–31)	Recursion (and Merge Sort) Topics: <ul style="list-style-type: none"> Recursion Merge Sort Objectives: <ul style="list-style-type: none"> Create a recursive method to solve a problem Understand the difference between recursive and iterative solutions to a problem Understand and use the Merge Sort. 	Resources: <ul style="list-style-type: none"> FJ: lesson 11.1 BJ: section 18.4 Assessments: <ul style="list-style-type: none"> Factorial program Rewrite loop programs with recursion. Strategies: <ul style="list-style-type: none"> Students need lots of practice on recursion. Mathematical worksheets on recursion help introduce this topic. Do a lot of examples and ask, “What is returned by this method?” (See appendix B for worksheets and notes.)
14 (32–36)	Review Topics: <ul style="list-style-type: none"> Review AP Computer Science A topics Objective: <ul style="list-style-type: none"> Prepare for the AP CS A Exam by reviewing material and taking practice exams 	Resources: <ul style="list-style-type: none"> Previous free-response questions from AP Central Barron’s test-prep book for the AP Exams Assessments: Practice exams Strategies: Give as many practice exams as possible.

Teaching Strategies

I try to create a learning environment that is comfortable for all students. Those who have never touched a computer should be as at ease in my class as those who have taught themselves how to program. I aim to foster critical thinking, a lifelong skill, and I accomplish this by giving challenging, yet not impossible, assignments. When new topics are introduced, I like to use a hands-on approach of having students see and run examples. While the novices ask questions, more experienced students can make changes to the examples and experiment with different outcomes.

I measure the effectiveness of my teaching by monitoring student work and giving many short quizzes, in order to gauge students’ knowledge before I test them on a topic. Experienced programmers help the novices in a mentoring program after school. This promotes student leadership and propels in-class learning.

Lab Component

The classroom contains 27 student desks in the middle of the room and 27 computers on tables around the perimeter. For lectures, quizzes, and hands-on tests, students sit at the desks. I give at least two programs per unit, and students work on programs about 70 percent of the time. They can also come in before or after school for extra programming time and help. All computers have *JCreator LE* installed, and students have access to information on how to download it at home.

Student Evaluation

Category	Weight	Comments
Daily/Homework	20%	Most programming assignments fall into this category.
Quizzes	20%	
Tests/Major	60%	

Teacher Resources

College Board. *AP Computer Science Course Description*. New York: College Entrance Examination Board. Download the latest version from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

College Board. *AP Marine Biology Simulation Case Study Teacher's Manual*. New York: College Entrance Examination Board, 2003. Download from the Course Home Pages: see URLs above.

College Board. Computer Science A: Exam Questions. AP Central. apcentral.collegeboard.com/examquestions

Teukolsky, Roselyn. *Barron's How to Prepare for the AP Computer Science Advanced Placement Examination (Java Version)*. 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.

Student Activities

The following activities are reproduced in appendix B:

Searching and Sorting (Unit 9):

- Searching and Sorting Worksheet
- Searching and Sorting Worksheet Key
- Lab Assignment: Searching and Sorting Arrays

Recursion (Unit 13):

- Recursion Worksheet 1
- Recursion Worksheet 2
- Recursion Notes and Worksheet Answers

Sample Syllabus 2 (AP Computer Science A)

Michael Lew

Loyola High School
Los Angeles, California
mlew@loyolahs.edu
www.loyolahs.edu

School Profile

Location and Environment: Founded in 1865, Loyola High School is the oldest high school in Southern California. It is located in the heart of Los Angeles, 6 miles west of the downtown area. Loyola’s central location draws students from a 50-mile radius representing 235 elementary schools. The student body includes a wide range of ethnicities and socioeconomic levels.

Grades: 9–12
Type: Catholic—all boys
Total Enrollment: 1,210
Ethnic Diversity: Hispanic/Latino 20 percent
Asian American 15 percent
African American 8 percent
College Record: 97 percent attend four-year universities
2 percent attend community colleges and vocational colleges

Personal Philosophy

My philosophy in teaching AP Computer Science is to challenge my students to be active learners and critical thinkers. I follow the “guide on the side, not the sage on the stage” approach, which gently introduces the students to each new concept, encouraging student-driven learning via questioning and experimentation.

I believe in hands-on learning. In addition to working on programs at home, students receive instant feedback during in-class lab time once or twice a week. During lab classes, I can easily look at each student’s code, informally tracking their progress and giving help accordingly (paying more attention to the slower learners). I can talk to students about their programs, and they can ask specific questions about problems they are having.

Class Profile

One section of AP Computer Science A is offered each year. Course enrollment is 14 students. I give lectures on the first one or two days of the week, followed by two days of lab time.

Loyola has a rotating schedule with each class meeting for 55 minutes, four times a week. A typical week is shown below:

Monday	Tuesday	Wednesday	Thursday	Friday
1	6	5	4	3
2	1	6	5	4
3	2	1	6	5
4	3	2	1	6
5	4	3	2	

Testing days, in-service days, and religious retreats may decrease the number of class meetings per week to three, and sometimes two, 55-minute periods per week. In addition, seniors have no classes during the month of January, when they participate in a mandatory community service program; since AP Computer Science is a senior-level course, this means there are no computer science classes in January.

During the year I identify students who pick up the material quickly and offer them the opportunity to study the AP Computer Science AB curriculum and take the AB exam. It is, in effect, a directed study where they complete the AB assignments on their own, with minimal intervention from the teacher.

Course Overview

The purpose of this class is to introduce the student to the object-oriented programming paradigm using the Java language. Concepts such as classes, objects, inheritance, polymorphism, and code reusability are covered. Individual hands-on laboratory work helps solidify each concept. Students complete a long-term programming project that they must demonstrate in a formal presentation.

Course Objectives

The student will:

- develop an understanding of the concept of a class and an object
- develop an understanding of how objects model real-world objects
- implement inheritance to construct an object hierarchy
- implement polymorphism to process collections of objects
- study, understand, and extend the Marine Biology Simulation Case Study project
- design and implement a long-term programming project and formally present the project to the class

Texts

College Board. *AP Marine Biology Simulation Case Study*. New York: College Entrance Examination Board, 2002. Download from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

Deitel, H. M., and P. J. Deitel. *Java: How to Program*. 5th ed. Upper Saddle River, N.J.: Prentice Hall, 2003.

Litvin, Maria. *Be Prepared for the AP Computer Science Exam in Java*. Andover, Mass.: Skylight Publishing, 2003.

Teukolsky, Roselyn. *Barron's How to Prepare for the AP Computer Science Advanced Placement Examination (Java Version)*. 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.

Course Planner

Below are my course planners for the fall and spring semesters. I follow a bottom-up teaching model, focusing on fundamentals first, and then progressing to objects, object interaction, inheritance, and polymorphism.

Chapter 3

Beginning in week 10, students are assigned practice free-response questions for homework assignments. These questions come from past AP CS Exams (AP Central) or from the *AP Computer Science Course Description*. Introducing students early to actual AP Exam questions helps them to understand the language and difficulty level that they should expect in my class and on the AP Exam.

Key to notations in Programming Projects column:

Page numbers and problem numbers are references to the Deitel text.	
CSn problems: Examples:	
CS1—WordList 04A1 (ArrayList, Strings)	CS4—Ballot CD 0405 (ArrayList /Iterators)
CS1: Computer Science Project 1	
04A1: 2004 Computer Science A Exam, Question 1 (“WordList” problem)	
CD 0405: <i>AP Computer Science Course Description</i> (May 2004, May 2005)	

Note: Questions on the 2002 and 2003 AP Computer Science Exams were written in C++; I have translated a few of those questions into Java to use with my students.

Fall Semester

Week	Topic	Programming Projects	Tests/Large-Scale Projects
1	Introduction to Computer Science The Java Language and Java Program Structure	Intro to Class Intro to Computer Science	
2	Input/Output Arithmetic Operators If, If-Else Statements	P. 80, 2.27 [Odd/Even] P. 80, 2.28 [Multiple] P. 80, 2.31 [Cast Char] P. 81, 2.35 [Neg / Pos / Zero]	
3	Control Statements While Loops	P. 162, 4.11 [Miles Per Gal] P. 162, 4.12 [Credit Limit] P. 163, 4.13 [Commission] P. 163, 4.14 [Gross Pay]	Test 1 Basic I/O, Arithmetic Operators
4	More Control Statements For Loops	P. 212, 5.6 [Smallest Int] P. 212, 5.7 [Product Odd] P. 212, 5.8 [Factorials] P. 212, 5.9 [Interest]	
5	Methods Arrays	P. 271, 6.8 [Parking] P. 272, 6.14 [Power] P. 273, 6.23 [Celsius / Fahrenheit] P. 274, 6.28 [GCD] P. 275, 6.32 [Cai] P. 276, 6.36 [Recursive Power] P. 278, 6.45 [Recursive Error]	Test 2 Control Statements

Week	Topic	Programming Projects	Tests/Large-Scale Projects
6	. . . And More Control Statements	P. 213, 5.12 [Mail Order] P. 213, 5.15 [Bin, Oct, Hex, Dec] P. 213, 5.16 [Pi] P. 166, 4.31 [Encrypt] P. 167, 4.32 [E] Read chapter 11, 11.1–11.4	
7	Methods	Solve the Quadratic Formula!!	
8	ArrayLists / Iterators Solve These Problems With ArrayList / Iterators Only	P. 323, 7.12 [Array Init] P. 323, 7.15 [Roll Die] P. 324, 7.18 [Airline Res.] P. 330, 7.29 [Tortoise Hare] P. 332, 7.32 [Palindrome]	Test 3 Methods
9	Object Creation	P. 396, 8.2 [Complex] P. 396, 8.3 [Rational] P. 397, 8.4 [Tick] P. 397, 8.5 [Date] P. 397, 8.6 [Date / Time]	
10	Object Interaction	Object Case Studies ----- CS1—WordList 04A1 (ArrayList, Strings) CS2—Company 03A2 (ArrayList)	Test 4 Object Creation
11	Object Interaction	Object Case Studies ----- CS3—Plane 02A4 (ArrayList / Iterators) CS4—Ballot CD 0405 (ArrayList / Iterators) CS5—LineEditor CD 0405 (String)	
12	Object Interaction	Object Case Studies ----- CS6—College 03A1 (ArrayList / Iterators) CS7—GroceryStore 02A2 (ArrayList / Iterators)	
13	Inheritance Polymorphism Interfaces Abstract Classes	Inheritance Case Studies ----- CS8—Bank Account CD 0405 (Inheritance)	Test 5 Object Interaction
14	Inheritance Polymorphism Interfaces Abstract Classes	Inheritance Case Studies ----- CS9—Pet 04A02 (Inheritance / Abstract Class)	
15	Inheritance Polymorphism Interfaces Abstract Classes	Inheritance Case Studies ----- CS10—Company 03A2 With Interface (Inheritance / Interfaces)	
16	-----	Final Exam	Cumulative

Spring Semester

Week	Topic	Programming Projects	Tests / Large-Scale Projects
1–3	Project Presentations	Presentations	
4	Strings Exception Handling	String Project 1: Your Star Wars Name String Project 2: Pig Latin	
5	Searching / Sorting Recursion / Big-Oh	---	
6	Searching / Sorting Recursion / Big-Oh	Searching / Sorting Efficiency Project	
7	Marine Biology Simulation Case Study (MBS)	MBS Role Play Code Walk-Thru	
8	MBS	MBS Project 1—PondStocker 04A3	
9	MBS	MBS Project 2 Design Your Own Fish	
10	MBS	MBS Project 3 Design Your Own Obstacle	
11	MBS	Marine Biology Simulation	
12–13	Review for AP Exam	Review for AP Exam	
14	AP Exam Work on Final Projects	AP Exam Work on Final Projects	
15–18	Spring Final Project Presentations	Spring Final Project Presentations	

Teaching Strategies

Preparing Students for the AP Exam

Weeks 12, 13, and 14 of the second semester are set aside for review for the AP Exam. I use a selection of multiple-choice and free-response questions from the Barron and Litvin review books and AP Central. On alternating days, I assign in-class multiple-choice and free-response questions. On a “multiple-choice question day,” I allot 30 minutes of a 55-minute period for 18 multiple-choice questions, leaving 25 minutes to go over each of the questions in detail. On a “free-response question day,” I use 30 minutes of a 55-minute period for 1 free-response question, leaving 25 minutes to go over the answer to the question in detail. The number of multiple-choice and free-response questions in the 30-minute period is roughly in line with the time students will have on the actual AP Exam.

Student Presentations

As detailed below in the Student Activities section, students are required to do two full-period (55-minute) *PowerPoint* presentations, one in February and one in June. These presentations detail the design and implementation of a large-scale program that students have proposed, designed, and written. After each presentation, time is allowed for questions from me and other students in the class.

As mind-boggling as the 55-minute time frame can seem to students, they ultimately take pride in creating a complete and well-thought-out presentation. Most important, the process allows them to practice conveying technical information in a clear and understandable manner. They learn that there is a fine line between presenting too little and too much technical detail to an audience. I have found that this alternative assessment method really gauges their understanding of the proper object-oriented programming design principles and programming skills.

Lab Component

Students work independently in the lab one to two times a week. Lab work is extremely important for two reasons. First, it allows me to see students coding in action. I casually walk from student to student, looking at their progress on assigned programs and getting a very good idea of how each person is progressing. I can check for understanding or check coding style (indentations, naming conventions for variables, methods, classes, etc.). My instant feedback enables a student to make corrections “on the spot.” Second, students can ask specific questions about concepts with which they are having trouble, and I can give personalized assistance. This is especially helpful during the first month of class when students are new to programming, as well as during the weeks leading up to their first long-term project.

Student Evaluation

Student evaluation is based on homework assignments, tests, and a long-term project.

Category	Weight
Homework	45%
Tests	25%
Long-term project	30%

Homework

On the first day of class, each student receives a Course Planner showing which concepts will be taught each week, along with homework assignments, tests, and the due date for a long-term project. Students are asked to work on homework assignments at home, developing the skeleton of each program. Lab time can be used to complete programs. I check homework assignment problems on the computer; no printouts are required of students. My classroom setup—four rows of four computers each—is conducive to checking four students’ work at one time. (Class size is limited to 14, because invariably, one or two computers are down for one reason or another, and having two spare computers ensures that each student has one to use.) As an example, I direct the four students in row one to open all programs from chapter 2 of the text. I ask them to compile and run the first program; then I check the output to determine if the program is working. If this is successful, I ask students to show their source code, and I check for style, documentation, and overall program organization. After all programs for chapter two are graded (each program is worth 25 points), I progress to the next row.

I emphasize that students must do their own work. Homework assignments are from the Deitel text and from previous AP Exam questions.

Tests

Tests are given after each major topic is covered. Students are given a problem statement, similar to the homework assignments, and are asked to write a working program. They have the entire 55-minute period to design, code, and debug their program. At the end of the period, I collect each student’s disk so that I can grade the work.

Long-Term Projects

See Student Activities section below.

Teacher Resources

College Board. Computer Science A: Exam Questions. AP Central. apcentral.collegeboard.com/examquestions

College Board. *AP Computer Science Course Description*. New York: College Entrance Examination Board. Download the latest version from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

College Board. *AP Marine Biology Simulation Case Study Teacher's Manual*. New York: College Entrance Examination Board, 2003. Download from the Course Home Pages: see URLs above.

Horstmann, Cay. *Object-Oriented Design & Patterns*. Hoboken, N.J.: Wiley, 2003.

Levine, David, and Steven Andrianoff. AP Marine Biology Simulation Role Play (Java Version). St. Bonaventure University, Department of Computer Science, 2003. <http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html>

Student Activities

One of the most successful activities that I assign is a large-scale, multiclass project that students propose, design, implement, and present to the class according to a strict set of guidelines. A project is assigned once in the first semester before the Marine Biology Simulation Case Study is covered, and once again after the MBS in the second semester. The requirements for the second project build on, but are slightly different from, the first.

Students must make design decisions for the first project (which classes to create, which classes must or should interact with other classes, and so forth). After a slightly successful attempt at writing a large-scale project in the first semester, students study the MBS and learn the characteristics of a well-written, large-scale project. They then propose and implement a second project (different from the first—no rewrites are allowed) using the experience they have gained from writing their own first project and studying a well-written project. This usually results in a well-designed and organized project completed in a fraction of the time it took to write the first one.

The proposal for the second project is due just after we study the MBS. Students actually work on it as a review for the AP Exam, with project presentations done after the exam. This has been a great way to keep students (especially seniors) on task after the AP Exam.

Each student has one 55-minute period to present a project, subject to the guidelines listed in the project assignment (see appendix B).

Sample Syllabus 3 (AP Computer Science A)

Frances P. Trees

Drew University

Madison, New Jersey

ftrees@drew.edu

www.drew.edu

University Profile

Location and Environment: Drew University was started as a Methodist seminary in 1867; today it is an independent university with a continued focus on the liberal arts and the use of technology in support of teaching and learning. Drew's population is primarily residential, with most undergraduate students living on campus. The school is located on approximately 200 wooded acres in the town of Madison in northern New Jersey, 30 miles west of Manhattan.

About 70 percent of Drew's undergraduate students have graduated in the top quarter of their secondary school class; about half come from the top 10 percent of their class.

Professors at Drew place teaching and learning first. Full professors, deans, and the president of the university teach courses. Drew prides itself on small classes, with 70 percent of lower-level courses (open to first-year students) numbering 25 students or fewer; over 20 percent have no more than 10. Nearly 80 percent of upper-level courses total 20 or fewer students. The average student-to-faculty ratio is 11 to 1.

Students will have acquired a strong liberal arts background by the time they graduate. To facilitate this, each student must complete a major and a minor. The major is declared at the end of the second year and the minor by the end of the first semester of the junior year. In the 2002-03 academic year, the highest numbers of degrees were awarded in political science, English, and psychology.

Type:	Small, highly-selective four-year liberal arts university.
Total Enrollment:	2,400 students: 1,500 undergraduates; 900 in the graduate and theological schools
Ethnic Diversity:	Out of 1,629 undergraduates enrolled in the fall of 2002, 91 described themselves as Hispanic; 90 as Asian or Pacific Islander; 63 as black, non-Hispanic; and 7 as American Indian or Alaskan Native. Currently, undergraduates come from 42 states, the District of Columbia, and about 12 countries other than the United States.
Gender Ratio:	The male/female ratio is approximately 42:58.

Mathematics and Computer Science Department

Students in Drew's Mathematics and Computer Science Department may major in computer science or in mathematics (general or applied), or they may choose a joint major that includes core courses and electives in both disciplines. Each year, the department graduates approximately 20 majors in computer science, mathematics (general or applied), or a joint computer science–mathematics major.

The first-year computer science sequence is composed of four half-semester courses, each of which can earn students 2 credits:

- CSCI 6 and CSCI 9 provide an introduction to computer science through programming and cover roughly the same material as the AP Computer Science A course. [CSCI is short for Computer Science.] The courses are offered in the first and second halves of the fall semester, respectively.

Chapter 3

- CSCI 12 and CSCI 13 include coverage of material on data structures and algorithms normally covered in the AP Computer Science AB course. This sequence of courses is offered in the spring semester.

CSCI 6 and 9 are required courses for a computer science major, a joint computer science–mathematics major, and an applied mathematics major. CSCI 12 and 13 are required courses for a computer science major and a joint computer science–mathematics major.

Philosophy of the Department

Precise abstraction and quantification play an increasingly important role in many areas of learning, from the natural sciences and the social sciences to finance, humanities, and the arts. The study of mathematics and computer science can provide a foundation that fosters rigor of thought and an analytical approach to solving problems. Graduates with such a background find themselves in demand not only in mathematics and computer science but also in such areas as law and business, where clear thinking and analysis are indispensable.

Personal Philosophy

It is my goal to bring life and fun into the computer science classroom while challenging students to think outside the box. I can achieve this goal only by actively involving each student in each lecture in any way that I can, not always in the traditional question–answer format. I use play-acting, group work, partner projects, role-playing, games, lectures, and challenges. In order for the students to enjoy learning, the teacher has to enjoy teaching!

Introduction to Computer Science and Programming (CS1)

Class Profile

In recent years, the enrollment for CSCI 6 has averaged about 20 students, with a drop of 1 or 2 for each succeeding course in the CSCI 6–9–12 sequence.

This syllabus focuses only on the first semester CSCI 6–CSCI 9 sequence. The class meets for three 65-minute lectures per week plus an additional 65-minute lab period. Since Drew gives each student a personal computer with software and a printer, there is no formal “lab” setting; students simply bring their laptops to class on lab day. A wireless network provides Internet access if needed.

Course Overviews

CSCI 6: Introduction to Computer Science and Programming I

CSCI 6 is an introduction to computer science through programming. The focus is on designing and implementing programs in an object-oriented language. Topics covered include basic data types and variables, control structures, and defining and using classes and methods. This course is taught using the Java programming language along with program development environments.

There are no prerequisites for CSCI 6. Some students come with some programming knowledge (usually not Java), and others have never written any type of program. I assume no prior programming knowledge on the first day of class.

CSCI 9: Introduction to Computer Science and Programming II

CSCI 9 is a continuation of CSCI 6. Additional features and principles of object-oriented programming, including interfaces, inheritance, and use of class libraries, are covered. Topics include introduction to recursion, the Java collections hierarchy, and introduction to analysis of program efficiency in the context of basic searching and sorting algorithms. This course uses the same programming language and development environments as CSCI 6.

The prerequisite for CSCI 9 is completion of CSCI 6 with a minimum grade of C-.

AP Credit Policy: A student receiving a 4 or 5 on the AP Computer Science A or AB Exam is exempt from CSCI 6 and 9. A student cannot receive both AP credit and Drew University credit for CSCI 6 and 9.

Required Texts

Horstmann, Cay. *Big Java*. Hoboken, N.J.: Wiley, 2002.

or

Horstmann, Cay. *Computing Concepts with Java Essentials*. 3rd ed. Hoboken, N.J.: Wiley, 2002.

(Note: *Big Java* is also the required text for CSCI 12 and CSCI 13; *Computing Concepts with Java Essentials* has the same first 19 chapters as *Big Java* and is sufficient for students taking only CSCI 6 and CSCI 9.)

College Board. *AP Marine Biology Simulation Case Study*. New York: College Entrance Examination Board, 2002. Download from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

Reference Text

Trees, Fran, and Cay Horstmann. *Computing Concepts with Java Essentials. Advanced Placement Computer Science Study Guide*. 3rd ed. Hoboken, N.J.: Wiley, 2004.

Software

BlueJ: www.bluej.org

Xinox Software. *JCreator*. 2004. <http://jcreator.com/>

Course Objectives

The goals of the course are to help students:

- understand the basics of computer hardware and software
- understand the principles of object-oriented programming (OOP) and be able to develop classes and projects using these OOP principles in an inheritance hierarchy
- learn Java syntax and good programming style, writing clear and efficient code
- understand a variety of algorithms and their efficiencies
- design and develop object-oriented solutions to problems
- understand the design and implementation of a large case study

Chapter 3

- simulate common sorting and searching algorithms and understand the efficiency of these algorithms
- work independently, in groups, and with a partner to achieve a common goal

Course Planner

The introductory computer science courses are taught using *BlueJ* and *JCreator*. Both of these Integrated Development Environments (IDEs) are free. I find *BlueJ* an integral and motivating aspect of beginning the object-oriented thought process. *JCreator* can be replaced by another IDE such as *Eclipse* or *JBuilder*.

The Topic column in the tables below lists the subjects covered in the daily lectures. These topics are based on the chapters in *Big Java*. The Strategies column lists *PowerPoint* presentations and activities used during the class. *PowerPoint* presentations (in italics) are available at my course Web sites (see Teacher Resources below). Teaching strategies and activities are listed by Strategy #; these are briefly described following the syllabi tables. The Reading/Assignment column has material from *Big Java* (BJ) and corresponding material from the AP Study Guide reference text (APG).

Lab assignments are briefly described after the Teaching Strategies section and can also be accessed at my course Web sites.

CSCI 6			
Class	Topic	Strategies	Reading/Assignment
1	<ul style="list-style-type: none"> • Course Overview, Object-Oriented Programming • Introduction to Objects and Classes • Introduction to <i>BlueJ</i> 	<ul style="list-style-type: none"> • Strategy #1 • <i>CSCI6_Intro_Lecture</i> incorporating Strategy #2 and demonstration of <i>BlueJ</i> using Shapes 	<ul style="list-style-type: none"> • BJ: Read chapters 1 and 2 • BJ: Exercises pages 30–31: #1, 4, 9, 10, 11, 14 • APG: chapters 2 and 3
2	More with Objects and Classes	<ul style="list-style-type: none"> • <i>JavaBasics_Lecture</i> incorporating some method writing for Bank Account with class 	<ul style="list-style-type: none"> • BJ: Exercises pages 72–74: #1–18
3	Fundamental Data Types	<ul style="list-style-type: none"> • <i>Chapter_3_Lecture</i> incorporating some code writing with students • Lab 1 due 	<ul style="list-style-type: none"> • BJ: Read sections 3.1–3.3
4	<ul style="list-style-type: none"> • More with Classes • Primitive Types and Operations • The Math Class 	<ul style="list-style-type: none"> • <i>Chapter_3_continued</i> incorporating some code writing with students • Number base conversion practice 	<ul style="list-style-type: none"> • BJ: Read sections 3.4–3.7, 3.10 • BJ: Exercises pages 123–25: #1(b), 2 (a,b,c), 3, 6, 8, 9, 12, 13, 15, 16, 17, 18 • APG: chapter 4
5	(Skip chapter 4 for now.) <ul style="list-style-type: none"> • Decisions • The if statement • Comparing values and objects 	<ul style="list-style-type: none"> • <i>Chapter_5_Lecture</i> • QUIZ (chapters 1–3) 	<ul style="list-style-type: none"> • BJ: Read sections 5.1–5.3 • BJ: Exercises pages 219–21: #1(a,b,c,i,j), 4, 5, 9, 11, 12, 13, 14, 17
6	<ul style="list-style-type: none"> • More with Decisions • switch; test cases; Boolean expressions • DeMorgan's Law 	<ul style="list-style-type: none"> • <i>Chapter_5_Continued</i> incorporating some code writing with students–partner work 	<ul style="list-style-type: none"> • BJ: Read 5.4 • BJ: Exercises pages 219–20: #1 (the other parts), 2, 3, 6, 8, 10 • APG: chapter 5

CSCI 6			
Class	Topic	Strategies	Reading/Assignment
7	Iteration: while loops; for loops	<ul style="list-style-type: none"> • <i>Chapter_6_Lecture</i> • Strategy #3 • Review of <i>Company Rules</i> • Lab 2 due 	<ul style="list-style-type: none"> • BJ: Read 6.1, 6.2
8	Nested loops	<i>Loop Exercises</i>	<ul style="list-style-type: none"> • BJ: Read 6.3, 6.4.3 • BJ: Exercises pages 272-73: #1, 6, 15, 17
9	More with loop exercises		
10	<ul style="list-style-type: none"> • <i>JCreator</i> Introduction and details • Programming Details 		
11	Random Numbers	<i>Random Numbers</i>	<ul style="list-style-type: none"> • BJ: Read 6.5 • BJ: Pages 272-73: #14
12	EXAM (chapters 1-6)		
13	<ul style="list-style-type: none"> • Input from users: JOptionPane • toString • compareTo • equals 	<ul style="list-style-type: none"> • <i>UserInput</i> • <i>MoreUsefulMethods</i> • Lab 3 due 	<ul style="list-style-type: none"> • BJ: Review chapter 6 • BJ: Read 3.8 (User input), 5.2.3 (equals, compareTo) • APG: chapter 6
14	More with classes	Develop with students: Complex Class	
15	Designing Classes: Intro to CRC cards	<ul style="list-style-type: none"> • <i>Chapter_7_Lecture</i> • Strategy #4 • Lab 4 due 	<ul style="list-style-type: none"> • BJ: Read 7.1-7.2 • Quality tips 7.1, 7.2 • BJ: Exercises pages 321-22: #1-3, 5-7
16	Static methods, fields	<i>Chapter_7_Continued</i>	<ul style="list-style-type: none"> • BJ: Read 7.3-7.8 • Advanced topics 7.1, 7.2, 7.4 • Quality tip 7.3, 7.4 • Common error 7.2 • BJ: Exercises pages 321-25: #15, 16(a), 17(a, c, d), 19, 20, 21, 23, 24, 26 • APG: chapter 7
17	Packages	<i>Chapter_7_Part3</i>	<ul style="list-style-type: none"> • BJ: Read 7.9 • Page 325: #29, 30
18		Online Quiz	
19		EXAM (chapters 1-7)	
20	Testing and debugging	Lab 5 due	<ul style="list-style-type: none"> • BJ: Read sections 8.1, 8.2, • Advanced topics 3.6, 6.4, 6.6, 6.8, 7.2 • Productivity hint 6.1, 8.1 • APG: chapter 8

Chapter 3

CSCI 9			
Class	Topic	Strategies	Reading/Assignment
1	Introduction to Recursion!	Strategy #5	<ul style="list-style-type: none"> • BJ: Read 17.1, 17.5 • BJ: Exercises pages 698: #6, 7, 8 • APG: chapter 14
2	Practice with recursion	Worksheet	
3	Interfaces and Polymorphism	<i>Interfaces and Polymorphism</i>	<ul style="list-style-type: none"> • BJ: Read sections 9.1, 9.2, 9.3 • BJ: Exercises pages 389-90: #1-4, 6, 8
4	<ul style="list-style-type: none"> • Interfaces and Polymorphism • Practice with Recursion 	Practice with Polymorphism and Recursion: Problems with class	<ul style="list-style-type: none"> • BJ: Read Random Fact 9.1 Operating systems • APG: chapter 9
5	Inheritance	<ul style="list-style-type: none"> • <i>Inheritance_Part_1</i> • Short Quiz (Recursion) • Lab 1 due 	<ul style="list-style-type: none"> • BJ: Read sections 11.1, 11.2, 11.3, 11.4 • Inheritance example
6	Inheritance continued	<i>Inheritance_Part_2</i> Animal class	<ul style="list-style-type: none"> • BJ: Read sections 11.5, 11.6 • BJ: Exercises pages 468-70: #1-8, 10, 11 • APG: chapter 10
7	Inheritance continued: Abstract classes	<i>Inheritance_Part_3</i>	<ul style="list-style-type: none"> • BJ: Exercises pages 471: #13-17
8	<ul style="list-style-type: none"> • Array lists • Wrapper Classes 	<i>ArrayLists</i>	<ul style="list-style-type: none"> • BJ: Exercises pages 549-50: #1-4
9	Arrays	<i>Arrays</i>	<ul style="list-style-type: none"> • BJ: Exercises pages 550-51: #5-7, 9,10
10	More with arrays	<i>Arrays</i>	<ul style="list-style-type: none"> • BJ: Exercises pages 550-51: #11, 12, 13, 14 • Worksheet • APG: chapter 11
11	Intro to Sorting (n^2 sorts)	<ul style="list-style-type: none"> • Strategy #7 • Quiz (Inheritance and Polymorphism) • Lab 2 due 	<ul style="list-style-type: none"> • BJ: Read sections 18.1-18.3
12	<ul style="list-style-type: none"> • Sorting (Merge/Quick) • Searching: Sections 18.4-18.8 	<ul style="list-style-type: none"> • Strategy #7 • Sort animations 	<ul style="list-style-type: none"> • BJ: Exercises pages 731-33: #1-11 • APG: chapter 15
13	<ul style="list-style-type: none"> • More Sorting/Searching • Practice Questions distributed (Strategy #6) 	<ul style="list-style-type: none"> • <i>BigOh</i> • <i>More with Invariants</i> 	

CSCI 9			
Class	Topic	Strategies	Reading/Assignment
14		<ul style="list-style-type: none"> • Exam 1 • Lab 3 due 	
15	BRING COMPUTERS TO CLASS <ul style="list-style-type: none"> • Loading MBS • MBS Overview 	Introduction to Java Marine Biology Simulation Case Study	<ul style="list-style-type: none"> • Java MBS Case Study
16	Role Play MBS	Strategy #8	<ul style="list-style-type: none"> • Read chapters 1 and 2 of MBS
17	Fish Problems	Go through overview of chapters 1–4 of the MBS Case Study	<ul style="list-style-type: none"> • Read chapters 3 and 4 of MBS
18		Online Quiz—PsychoFish	
19	<ul style="list-style-type: none"> • Presentations • 2-D arrays • Review Practice Questions 	Strategy #9	
20	Final Exam Review and Cleanup	Lab 4 due	
21		FINAL EXAM (3 hours)	

Time modification note for AP CS teachers:

To modify this syllabus to fit a yearlong AP Computer Science A course that meets for about 45 minutes per day, allow one and a half class periods for each class listed above. You will need about three of your class days for every two of mine. You’ll have to decide where the natural split will be.

Topics covered that are NOT tested on the AP CS A Exam:

char, charAt, some additional String methods (toUpperCase and toLowerCase), switch statement, some Math methods such as max and round, JOptionPane, reading from input, implementing equals in classes, packages, instanceof operator, access modifiers protected and default, clone, Big-Oh notation and analysis, iterators, quick sort, 2-D arrays

Teaching Strategies

Strategy 1: Students are given a paper with one statement on it. This statement asks the student to write directions for (a) making a peanut butter and jelly sandwich or (b) making a paper airplane. I then ask for a volunteer to give me the sandwich directions and another volunteer to make the sandwich. (I bring the ingredients to class.) Needless to say, directions such as “put peanut butter on bread” could result in the student putting the jar of peanut butter on the package of bread! I have the “airplane writers” give their directions to the “peanut butter people”; then we see how many of the latter come up with an airplane that looks similar to the one the “airplane writer” has made. At the end of class, the idea of algorithm development should be clear: The computer will do only what we tell it to do.

Strategy 2: I bring lots of cheap radios to class. One radio is taken apart so it has no case. The parts are then put in a little plastic container that is given to some adventurous student who is asked to make “it” work—with no knowledge of what the end product is. Then other radios of various types are distributed. Thus, the “big picture” of object-oriented paradigm and inheritance is introduced.

Chapter 3

Strategy 3: Practice with hats. A participant in my summer institute shared this with me: One student wears a painter’s cap marked with a big “I” and another wears a cap marked with a “J.” I put numbers 1–4 on the board, and then to the right of that, numbers 1–3. The two students wearing the caps also have signs on their backs labeling them as “I” and “J,” and two other students have the signs “product” and “sum” on their backs. We act out the code on the presentation slide with the loop-control-variables (I and J), pointing to their present value and the variables’ product and sum and keeping track of their values in memory locations on the other side of the board. This technique helps students visualize the nesting of loops.

Strategy 4: Students break into groups to create cyclic redundancy check (CRC) cards for a lottery game and a vending machine. (See Teacher Resources below for CRC card references.)

Strategy 5: Any number of great activities can be used to introduce recursion. Some of my favorites include the reading of *The Cat in the Hat Comes Back* or “Martin and the Dragon,” or using Groucho Marx glasses to get the point of recursion being one step and a smaller journey. Many of these techniques are explained on AP Central; look for the Teaching Strategies link from the Computer Science Home Page.

Strategy 6: There are many successful ways to introduce sorting. I use an overhead projector; an array template; and colored, numbered circles, arrows, partitions, and whatever else the sort requires (sometimes Booleans represented by a blue word “true” and a red word “false”). Most of these visual aids are drawn in *Word*, printed in color on transparencies, and then cut out. There are also many good Web sites offering animations for the various sorts (see Teacher Resources below).

Strategy 7: This is a good place to distribute sample questions from the *AP Computer Science Course Description*. You can also provide the *Java Quick Reference Guide* that is given to students taking the AP Exams (it is available on AP Central).

Strategy 8: Java Marine Biology Simulation Case Study materials are available for downloading from the Computer Science Home Page on AP Central. You can introduce this after you teach arrays and ArrayLists (after my Class #10 in CSCI 9).

Strategy 9: One of our department’s goals is to give the student practice in writing and speaking the subject—in this case, computer science. In the beginning of CSCI 9, I compile a list of topics from computer history, computer hardware, current events, women in computing, and other relevant themes. Each student is required to submit a paper three to five pages long on a chosen topic and make a presentation of five to seven minutes to the class. This assignment earns credit equivalent to a quiz grade. Presentations are made throughout the course. Another great way to bring writing into your curriculum is to visit Microsoft’s Reality Check, a Web site that highlights a new current event topic each week (see Teacher Resources below).

Company Rules

“Company Rules” make everyone’s life easier (including mine when I grade programs). My rules include indentation guidelines, placement of braces, variable naming conventions, commenting code, etc. You can develop your own as you go through the course. I deduct points on assignments if the rules are not followed. My Company Rules are available at my course Web site (see Teacher Resources below).

Lab Component

Programming Assignments: Each programming assignment consists of one or more problems that are completed on the computer. Each problem is saved as a separate project. These assignments are submitted to me electronically by midnight on (or before) the scheduled due date. It is important to give precise

directions for workspace/project creations, as some students have a great deal of difficulty organizing their files.

Labs Summaries (CSCI 6)

- **Lab 1:** Work with *BlueJ*; Shapes and Picture
- **Lab 2:** Work with *BlueJ*: `Purse` class and `Point` class
- **Lab 3:** Work with *BlueJ*: `FunNumber` class (methods dealing with interesting number algorithms such as number of digits, sum of digits, reverse digits) and `Word` class with palindrome handling.
- **Lab 4:** Work with *JCreator*: `RationalNumber` class and Roulette game
- **Lab 5:** Work with *JCreator*: utility class

Labs Summaries (CSCI 9)

- **Lab 1:** Reading from text files, Random numbers, DeMorgan’s Laws
- **Lab 2:** Euclid’s Algorithm (recursion), `Measurable` Interface
- **Lab 3:** Inheritance (`Athlete` class) and Lottery
- **Lab 4:** Concentration, Anagrams, MBS problems

Quizzes: The questions on the quizzes come from material covered in lectures, problems done in lab, and research assigned as homework. At least one quiz is during a lab session and requires students to finish solving a problem by completing a partially written program.

Exams: There are two major exams during CSCI 6 and one major exam plus a final exam in CSCI 9. The final exam period is three hours long. All exams are cumulative and contain a combination of short-answer, multiple-choice, and free-response questions.

Student Evaluation

Course grades for each of the two courses are based on the approximate distribution below. Students begin CSCI 9 with a clean slate.

Type	How many?	Point value for each
Programming assignments (labs)	~5	15–60 points
Quizzes	2–4	20–30 points
Exams	1	50–70 points
Final exam	1	100 points

Teacher Resources

Brummond, Nils. Object Oriented Analysis and Design Using CRC Cards. 1996.
www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/

Cockburn, Alistair. Using CRC Cards. 1999.
<http://alistair.cockburn.us/crystal/articles/ucrcc/usingcrccards.html>

Chapter 3

College Board. *AP Computer Science Course Description*. New York: College Entrance Examination Board. Download the latest version from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

College Board. *AP Marine Biology Simulation Case Study Teacher's Manual*. New York: College Entrance Examination Board, 2003. Download from the Course Home Pages: see URLs above.

Horstmann, Cay. Horstmann/Big Java: Instructor's Resource Center. Wiley. www.wiley.com/college/horstmann/0471402486/wave_i.html Includes chapter solutions, source code, test bank, and more.

MazeWorks. Java Games & Puzzles: Tower of Hanoi. www.mazeworks.com/hanoi/
For introducing recursion.

Microsoft Corp. .NET In Action Reality Check. 2005. www.mainfunction.com/DotNetInAction/RealityCheck/default.aspx
This is a great resource for bringing current events into the classroom.

Mitra, Sandeep. Java Sorting Animation Page. Computer Science at SUNY Brockport. 1999. www.acs.brockport.edu/~smitra/cs/javasort.html

Morris, John. Data Structures and Algorithms: Sorting. 1998. www.eng.tau.ac.il/~shtilman/C-programming/Year2/sorting.html
Includes animation of Insertion Sort.

Seuss, Dr. *The Cat in the Hat Comes Back*. New York: Random House, 1958.

Touretzky, David S. "Martin and the Dragon," in *Common Lisp: A Gentle Introduction to Symbolic Computation*, 232-33. Redwood City, Calif.: Benjamin/Cummings Publishing Company, 1990. www-2.cs.cmu.edu/~dst/LispBook/

Trees, Frances P. Computer Science 6. Drew University, Department of Mathematics and Computer Science. www.users.drew.edu/ftrees/Csci6/index.htm
Lectures and materials.

Trees, Frances P. Computer Science 9. Drew University, Department of Mathematics and Computer Science. www.users.drew.edu/ftrees/Csci9/index.html
Lectures and materials.

Sample Syllabus 4 (AP Computer Science AB)

Renee Ciezki

Ironwood High School

Glendale, Arizona

RLCiezki@peoriaud.k12.az.us

<http://ironwoodhigh.peoriaud.k12.az.us/>

School Profile

Location and Environment: Ironwood High School is located in a predominantly middle- to upper-middle-class suburban area in Glendale, Arizona, northwest of Phoenix. It is part of the Peoria Unified School District, the third-largest district in the state. PUSD covers 150 square miles and has 6 high schools and 29 K–8 elementary schools.

Grades:	9–12								
Type:	Public								
Total Enrollment:	2,162								
Ethnic Diversity:	<table> <tr> <td>Hispanic/Latino</td> <td>16.2 percent</td> </tr> <tr> <td>African American</td> <td>5.0 percent</td> </tr> <tr> <td>Asian American</td> <td>4.6 percent</td> </tr> <tr> <td>Native American</td> <td>0.7 percent</td> </tr> </table>	Hispanic/Latino	16.2 percent	African American	5.0 percent	Asian American	4.6 percent	Native American	0.7 percent
Hispanic/Latino	16.2 percent								
African American	5.0 percent								
Asian American	4.6 percent								
Native American	0.7 percent								
College Record:	74 percent go to college after graduation								

Personal Philosophy

Computer science is all about problem solving. In some subjects, students can complete a course without much effort, but this is not so with computer science. I hope students leave my class with improved thinking skills, more confidence in themselves and their ability to communicate and work together, and the tools necessary to continue growing in those areas.

Class Profile

One section of the AP Computer Science AB course (called Advanced Placement Computer Science 2-H at my school) is offered each year. (Students take Advanced Placement Computer Science 1-H, which covers the AP Computer Science A topics, before enrolling in this class.) Ironwood High School is on a 4 × 4 block schedule where classes meet for 90 minutes each day for one semester. This course is usually scheduled in the spring semester, and 15–20 students are enrolled. Unlike most other AP courses offered at Ironwood, only about half the students in AP Computer Science classes are taking, or have taken, other honors classes.

Course Overview

AP Computer Science 2-H is the most advanced course in our programming and computer science course sequence. It continues the study of problem-solving and object-oriented programming methodology begun in earlier courses and moves to the more formal and in-depth study of algorithms, data structures, design, and abstraction. Java is the programming language. The goals of the course are comparable to those in the introductory sequence of courses for computer science majors offered in many college and university computer science departments.

Chapter 3

The course prepares students for the AP Computer Science AB Exam. In addition, students are eligible for three hours of credit through a dual-enrollment arrangement with the local community college.

Texts

The primary text for this course, as officially adopted by the school district, is:

Institute of Computer Technology. *ICT's Advanced Placement Computer Science Curricula for Teachers*. Sunnyvale, Calif.: ICT, 2004. (Ordering info: www.ict.org/apcs.html.)

In addition, we have classroom sets of the following textbooks:

Horwitz, Susan. *Addison-Wesley's Review for the AP Computer Science Exam in Java*. Boston: Addison-Wesley, 2004.

Litvin, Maria. *Be Prepared for the AP Computer Science Exam in Java*. Andover, Mass.: Skylight Publishing, 2003.

Teukolsky, Roselyn. *Barron's How to Prepare for the AP[®] Computer Science Advanced Placement Examination (Java Version)*. 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.

Course Objectives

AP Computer Science 2-H is the final course in the Software Development strand of a program in Career and Technical Education. As such, students are expected to have attained a variety of competencies. See http://staffweb.peoriaud.k12.az.us/Renee_Ciezki/SoftwareDevOptionC.pdf. (For more information on this state-sponsored program, see chapter 5.)

Course Planner

Note: In the chart below, publishers' names refer to the course texts listed above: ICT, AW (Addison-Wesley), Barron's, and Skylight. Author names refer to texts in the Teacher Resources section further along in this syllabus. MBS is the Marine Biology Simulation Case Study.

Week(s)	Unit and Activities
1	Review of OOP and Java General course introduction Diagnostic test: Computer Science A: Sample Multiple-Choice Questions (Course Description), Students review, as necessary (Skylight chapter 2, ICT lessons 1–18) Review of classes (Skylight chapter 3, ICT lesson 29) The Pong Lab (Braught 2003) OOP Quiz Review of Arrays and ArrayList class (ICT lessons 19–20) Cards Lab (Frank 2004)
2	Review of MBS Review of Marine Biology Simulation (MBS chapters 1–4, Skylight chapter 6) AP Marine Biology Simulation Role Play (Levine and Andrianoff 2003) MBS Dynamic Population Quiz Lab: Extend Fish class—choose a unique type of “Fish” Group project: Demo group members' classes in a single project Implement VLBoundedEnv using the abstract class, ListEnv (MBS p. 104)

Week(s)	Unit and Activities
3	<p>Two-Dimensional Arrays (and Practice A Exams) Practice Exam 1 (Skylight p. 179) ICT lesson 21: Two-Dimensional Arrays Lab: MAXIT (See Student Activities section below.) Practice Exam 2 (AW p. 141)—in class, if time permits</p>
4–6	<p>Linked Lists Introduction (Skylight pp. 111-18) Implement SMBoundedEnv class (MBS p. 104) Lab: Royal Navy (Frank 2003)</p> <p style="text-align: right;"><i>Progress Report period ends</i></p> <p>Practice Exam 3 (Barron’s p. 503) ICT lessons 30–31: Linked Lists and Linked-List Lab: Implement SinglyLinkedList class per requirements (Wittry 2004) Implement SLUnboundedEnv class (MBS p. 104) using SinglyLinkedList ICT lesson 32: Doubly-Linked Lists, Barron’s chapter 8 Lab: Extend SinglyLinkedList to create DoublyLinkedList Exercises using circular and doubly linked lists Linked List Test</p>
7–8	<p>Stacks, Queues, and [ArrayList implementation of] Priority Queues General discussion (Barron’s chapter 9) Group project: Quizard of OOP (Litvin 2003) ICT lesson 38: Stacks Lab: APStack (implement using SinglyLinkedList) Practice Exam 4 (Barron’s p. 547)—in class, if time permits ICT lesson 39: Queues Lab: APQueue (implement using SinglyLinkedList) Group project: Mystery methods using APStack and APQueue objects Stack/Queue Quiz</p> <p style="text-align: right;"><i>End of term</i></p>
9	<p>Trees General discussion (Barron’s chapter 10, ICT lesson 34: Binary Trees) Lab: BSTree (ICT lesson 34) ICT lesson 35: Binary Tree Algorithms Lab: BSTree (add methods) ICT lessons 36–37: Deletion from a Binary Tree and Binary Tree Practice</p>
10	<p>More on Trees Complete implementation of BSTree class Implement BSTUnboundedEnv (MBS p. 104) Expression trees, heaps, and PriorityQueue revisited ICT lesson 41: Priority Queues Lab: Heapsort (ICT lesson 41) Tree Quiz</p>
11	<p>Hashing, Sets, and Maps ICT lesson 40: Hash-Coded Data Storage Lab: Hashing (ICT lesson 40) Discussion of Collections in Java (Barron’s chapter 11) Lab: Lab6 (Lewandowski 2004)</p>

Chapter 3

Week(s)	Unit and Activities
12	Hashing, Sets, and Maps (continued) Implement HMUnboundedEnv using HashMap (MBS p. 104) Lab: Concordance—reads any text file and lists words alphabetically (with counts) and by frequency in descending order. Practice Exam 5 (Barron’s p. 591) <p style="text-align: right;"><i>Progress Report period ends</i></p>
13	Algorithms Big-Oh ICT lesson 24: Order of Algorithms Algorithm Analysis Research Project (AARP) Research (See Student Activities section below.) Practice Exam 6 (Barron’s p. 643)
14	More on Algorithms AARP Presentations Lab: SortDemo—menu-driven program demonstrating five sorting algorithms from ICT lessons 23 (Quadratic Sorting Algorithms), 25 (Mergesort), and 26 (Quicksort)
15	More on Design Group project: All SET! (Litvin 2004) Group project: Design your own game and assign roles
16	Practice Exams and Grading <p style="text-align: right;"><i>AP Exam next week</i></p>
17–18	Group Projects Implement design; enjoy the games Final exam <p style="text-align: right;"><i>End of semester</i></p>

Teaching Strategies

On the first day of class, students are given paper copies of the Topic Outline from the *AP Computer Science Course Description* and the *Java Quick Reference Guide* (available on AP Central), shown how to access the Commentary within the Course Description, and instructed to bookmark the link to the Java API Specification. They are expected to have a working understanding of Computer Science A topics by the third week of class. A variety of books and online resources, as well as after-school help, are available. Taking responsibility for learning and contributing to the team are stressed throughout the course. The Topic Outline is consulted at several points throughout the semester to gauge progress.

Blackboard, a Web service for course hosting, is used for organizing assignments, course documents, and external links. The “groups” feature is used during certain projects to facilitate file sharing and collaboration. Most topics are presented with a short explanation in the form of a lecture or demonstration. This is followed by guided practice, reading(s), and/or worksheets. Students may be paired up for a small program (“Pair Programming”) and then do an individual project afterwards. At this level many projects last several days, and new topics are incorporated as they are introduced.

Practice tests from the various study guides are assigned as homework every few weeks. Students time themselves and keep a log of their scores and times. Another strategy is to simulate AP testing and scoring by giving students 25 minutes to answer one free-response question in class and then having them score a set of responses. “Training packs” of about 10 answer sheets are distributed the following day, along with a canonical solution, a set of scoring guidelines, and a score sheet, and we go through the same type of training that takes place at the AP Reading or at an AP workshop.

Otherwise, students have homework only when they find that they need to do additional reading or they need extra time to complete a lab. Extended due dates are assigned to allow a few extra days beyond the class time allotted for work on a lab, giving students time to add enhancements or to complete the lab outside of class.

Lab Component

The classroom contains 28 student computers with flat-screen monitors. The towers are on the floor, leaving desk space for reading, writing, and group work. Since students sit in front of the computers, I require the monitors to be turned off during class discussions, most tests, and some other types of work. However, students use computers approximately 75 percent of class time. I offer help before and after school on most days, and I encourage students to download any necessary software and work at home.

All computers have *BlueJ* and *JCreator Pro* installed. Students are free to use either IDE, and most use both.

Student Evaluation

Course grades are based on the distribution below.

Category	Weight
Regular work	40%
Assessments	40%
Final exam	20%

Regular Work: Labs, practice tests, worksheets, etc.

Assessments: Tests, quizzes, and major projects. Quizzes come in various formats, but the tests contain multiple-choice and free-response questions.

Final Exam: The comprehensive final exam is in the style of the AP Exams, with multiple-choice and free-response questions.

Teacher Resources

College Board Resources

These items can be downloaded from the Course Home Pages: apcentral.collegeboard.com/compscia (Computer Science A) or apcentral.collegeboard.com/compsciab (Computer Science AB).

College Board. Computer Science A or AB: Reference Materials for Exams.

College Board. *AP Computer Science Course Description*. New York: College Entrance Examination Board.

College Board. *AP Marine Biology Simulation Case Study*. New York: College Entrance Examination Board, 2002.

Chapter 3

Software, Web Resources

Balci, Osman et al. Online Interactive Modules for Teaching Computer Science: Animations to Assist Learning Some Key Computer Science Topics. Virginia Tech, Department of Computer Science, n.d. <http://courses.cs.vt.edu/~csonline/>

Blackboard. www.blackboard.com. Course-hosting Web service.

Brought, Grant. The Pong Lab: SIGCSE Nifty Assignments. Dickinson College, Department of Mathematics and Computer Science, 2003. <http://nifty.stanford.edu/2003/pong/>

Frank, Roger. Lab Assignments for the High School Computer Science Classroom. 2005. www.rfrank.net/cslibs/cslibs.htm

Kjell, Bradley. Introduction to Computer Science using Java Technology. Central Connecticut State University, 2003. www.javacommerce.com/displaypage.jsp?name=cs151java.sql&id=18218

Levine, David, and Steven Andrianoff. AP Marine Biology Simulation Role Play (Java Version). St. Bonaventure University, Department of Computer Science, 2003. <http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html>

Lewandowski, Gary. CSCI 180/181 Spring 2004, Lab 6. Xavier University, Mathematics and Computer Science Department, 2004. www.cs.xu.edu/csci180/04s/lab6.html

Litvin, Maria. Quizard of OOP (2003). All SET! (2004). www.skylit.com/oop/index.html

Pair Programming. www.pairprogramming.com

Sun Microsystems, Inc. Java 2 Platform, Standard Edition, v 1.4.2 API specification. 2003. <http://java.sun.com/j2se/1.4.2/docs/api/index.html>

Wittry, Dave. AP Computer Science A & AB. Troy High School, Fullerton, Calif., 2004. www.apcomputerscience.com

Xinox Software. *JCreator*. 2004. <http://jcreator.com/>

Student Activities

Here are two examples of assignments for my course: a game and a research project.

1. *MAXIT*: a two-player game using a two-dimensional array of objects. Instructions for playing the game, as well as a sample screen, are in appendix B. In addition, a sample working applet is available on my Web site: http://staffweb.peoriaud.k12.az.us/Renee_Ciezki/Maxit/Maxit.html.

2. *AARP* (Algorithm Analysis Research Project): A research project covering data structures, searching, and sorting. Full instructions for this project can be found in appendix B.

- Introduction: overview, with links to resources
- Data Structures: a list of required structures and operations
- Topics: searching and sorting algorithms
- Detailed Requirements
- Presentation: rubric for scoring

Instructions are also on my Web site:

http://staffweb.peoriaud.k12.az.us/Renee_Ciezki/aarp/aarp.zip. Download aarp.zip and open aarp.htm (the introductory page).

Sample Syllabus 5 (AP Computer Science AB)

(Robert) Glen Martin

School for the Talented and Gifted (TAG Magnet)

Dallas, Texas

martinrg@sbcglobal.net

www.dallasisd.org/schools/realtor_mag.cfm?View=36&DocID=362

School Profile

Location and Environment: TAG Magnet is an urban magnet school for talented and gifted students. It is located at the Townview Magnet Center, which contains six different magnet high schools. Prospective students apply from anywhere within the Dallas Independent School District, the twelfth-largest school district in the nation, covering 11 municipalities and 351 square miles.

The student population is very diverse, both ethnically and socioeconomically. Thirty-six percent of students receive a free or reduced-price lunch. Class sizes are small (student-to-faculty ratio is 13 to 1).

Students are involved in community service and extracurricular activities, including academic competition. All students also participate in three whole-school interdisciplinary activities during the year. These activities take six full school days and are not course specific.

We have a particularly active Parent Teacher Student Association.

TAG Magnet was designated a 2003 “No Child Left Behind Blue Ribbon School,” was twice named the “Best Public School in Dallas” by *D Magazine*, and has been rated Exemplary every year by the Texas Education Agency.

Grades:	9–12
Type:	Urban magnet high school
Total Enrollment:	180 students
Ethnic Diversity:	Hispanic/Latino 33 percent
	African American 26 percent
	Asian American 4 percent
	Native American 1 percent
College Record:	100 percent attend college

Personal Philosophy

I love teaching AP Computer Science AB (the course is called Advanced Placement Computer Science II at my school). Of my four different computer science courses, this one is easily my favorite. I enjoy seeing my students:

- gain insight into and solve the many little puzzles they encounter in this course, like how to reverse a linked list or find the middle of one with a single loop
- take pride in their ability to solve significant problems using powerful Java data structures such as maps
- learn that recursion is actually useful and often allows short, elegant, easy-to-understand solutions
- acquire the confidence that comes from having a deep understanding of how objects and data structures really work

I encourage very active student participation in the classroom. It's great to see students solve problems, share solutions, and make discoveries. I also enjoy coaching our Computer Science Team. We meet weekly to prepare for the many written and team-programming competitions during the year. I get to see the students having fun and being rewarded for their work while they progress both technically and personally.

Class Profile

Due to the school's small size, we have only one section of AP Computer Science II. Enrollment varies from year to year. In 2004-05, we had 6 students; in 2003-04, 11.

This is a yearlong course that meets on a block schedule (every other day for 90 minutes). Substantial time is allowed in class for lab assignments. However, both computer time and tutoring are available before and after school. Also, Java development software is available on CD-ROMs so students can work on labs at home if they choose to do so.

Students who complete this course usually go on to take my independent study course, which meets in the same classroom and period as AP CS II.

Course Overview

The content and objectives of AP CS II include the course objectives for AP Computer Science A and AB as discussed in the *AP Computer Science Course Description*. I begin with a short review of the Computer Science A material and then move into the Computer Science AB material. The course focuses on the implementation of data structures and the Java built-in classes for data structures. It includes a major focus on the Marine Biology Simulation Case Study.

This course enhances students' problem-solving abilities. It builds analytical skills that are valuable in computer science, in other courses, and in life; and, of course, students also increase their computer science and programming skills—skills that are needed in an ever-increasing array of college courses and workplaces.

Texts

College Board. *AP Marine Biology Simulation Case Study*. New York: College Entrance Examination Board, 2002. Download from the Course Home Pages: apcentral.collegeboard.com/compscia or apcentral.collegeboard.com/compsciab.

Litvin, Maria, and Gary Litvin. *Java Methods AB: Data Structures*. Andover, Mass.: Skylight Publishing, 2003.

Teukolsky, Rosalyn. *Barron's How to Prepare for the AP Computer Science Advanced Placement Examination (Java Version)*. 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.

Course Planner

The following is a breakdown of the week-by-week timeline for this course. It includes labs and tests. Titles and names in parentheses refer to required texts (above) or resources listed later in this syllabus.

Chapter 3

Weeks 1–6

Week 1–2	Review—Writing Classes Lab: Environment Plotter (Brady 2003)
Week 3–4	Review—Arrays / ArrayLists / Iterators Lab: WordList—with and without iterators (AP CS A Exam 2004, free-response question 1) Lab: Iterator Lab (Brady 2003) TAG Magnet Interdisciplinary Activity—entire school
Week 5	Classes and Interfaces (Java Methods AB chapter 1) Lab: none Tests: take-home free response and in-class multiple choice
Week 6	Marine Biology Simulation (MBS) Case Study Review (chapters 1 and 2) Lab: FishStuff12—Robert Glen Martin [See Student Activities below for details.] Test: in-class fill-in-the-blank and multiple choice

Weeks 7–12

Week 7	MBS Case Study Review (chapter 3) Lab: MBS Narrative chapter 3, exercise set 1: exercises 7 and 8 Test: in-class fill-in-the-blank and multiple choice
Week 8	TAG Magnet Interdisciplinary Activity—entire school
Week 9	MBS Case Study Review (chapter 4) Lab: AP CS AB Exam 2004, free-response question 3 (PredatorFish) Test: in-class fill-in-the-blank and multiple choice
Week 10–12	Lists and Iterators (Java Methods AB chapter 2) Lab: Movie Actors Index (Java Methods 2.7) Tests: take-home free response and in-class multiple choice

Weeks 13–18

Week 13–15	Stacks and Queues (Java Methods AB chapter 3) Lab: Browsing (Java Methods 3.3) Lab: Teletext (Java Methods 3.6) Tests: take-home free response and in-class multiple choice
Week 16	Recursion (Java Methods AB chapter 4) (Continued in weeks 19–20 below)
Week 17	First Semester Review
Week 18	Semester Finals

Weeks 19–25

Week 19–20	Recursion (continued) Lab: The Game of Hex (Java Methods 4.6) Tests: take-home free response and in-class multiple choice
Week 21–23	Binary Search Trees (Java Methods AB chapter 5) Lab: Morse Code (Java Methods 5.7) Tests: take-home free response and in-class multiple choice
Week 24–25	Look-up Tables and Hashing (Java Methods AB chapter 6) Lab: Search Engine (Java Methods 6.6) Tests: take-home free response and in-class multiple choice

Weeks 26–31

Week 26–27	Priority Queues (Java Methods AB chapter 7) Lab: Heapsort (Java Methods 7.5) Tests: take-home free response and in-class multiple choice
Week 28	Analysis of Algorithms (Java Methods AB chapter 8) Lab: SortDetective (Levine 2002) Tests: take-home free response and in-class multiple choice TAG Magnet Interdisciplinary Activity—entire school
Week 29–31	Data Structures in the MBS Case Study—chapter 5 Lab: MBS narrative chapter 5, exercise set 1: exercises 3 and 6 Test: in-class fill-in-the-blank and multiple choice

Weeks 32–38

Week 32–34	AP Exam Review
Week 35	AP Exam After the Exam Projects (These projects vary depending on the interest and ability of the students. The content is not critical to the course.)
Week 36–37	Continue Work on Projects
Week 38	Semester Finals

Teaching Strategies

AP CS II is a substantial course that requires a meticulous approach on the part of both teacher and students. I approach each new section of material with the following sequence of activities:

1. Reading and homework exercises (usually spread over several days)
2. Discussion of the material, including detailed interactive review of homework
3. Lab (programming) assignment
4. Take-home free-response test
5. In-class multiple-choice test

I assign most of the exercises in the Litvins’ *Java Methods AB* book for homework. This gives students an opportunity to work on a wide range of free-response questions. Review of a homework problem usually starts with a student giving his or her solution. Then we discuss the correctness, performance, advantages, and disadvantages of the code. This is a very open format with active student participation. Often other students will give alternate solutions.

The limited class time is fully utilized for discussion, labs, and multiple-choice tests, so outside class work (reading and homework) is critical to students’ understanding of the material.

I reserve the last few weeks prior to the AP Exam for review. Students work on multiple-choice and free-response questions at home and in class. I rely heavily on the practice questions in Teukolsky’s AP Exam–preparation book (see Texts, above).

Lab Component

Writing computer programs is critical to understanding the material. I usually assign one lab per chapter. These assignments are typically done on an individual basis; I encourage collaboration but do not allow copying. There is ample class time for most students to complete the lab assignments. I also provide lab time and tutoring before and after school.

My classroom contains both desks and computer tables with desktop machines. These computers have the *Sun Java SDK* and the *JCreator* Interactive Development Environment, tailored for our use. The software contains starter projects and other necessary libraries for simple applications, simple applets, and the Marine Biology Simulation Case Study.

All of the Java-specific software we use in the classroom is available at no cost. I make CDs of this software available for students to install on their home computers—most students take advantage of this.

Student Evaluation

The semester grades are computed by averaging the three six-week term grades and the semester final. They are all equally weighted.

Each six-week grade is computed as follows:

Category	Weight	Comments
Homework	20%	Graded for completion
Daily work	40%	Labs and other significant classroom work
Exams	40%	Take-home free response and in-class multiple choice

Although not strictly required, students are expected to take the AP Computer Science AB Exam and receive a grade of 3 or higher. Students' course grades correlate strongly with their AP grades.

Teacher Resources

Brady, Alyce. *AP Computer Science Teaching Resources*. Kalamazoo College, 2003.
<http://max.cs.kzoo.edu/AP/>

College Board. *AP Computer Science Course Description*. New York: College Entrance Examination Board. Download the latest version from the Course Home Pages: apcentral.collegeboard.com/compscias or apcentral.collegeboard.com/compsciasb.

College Board. *AP Marine Biology Simulation Case Study Teacher's Manual*. New York: College Entrance Examination Board, 2003. Download from the Course Home Pages: see URLs above.

College Board. *Computer Science A and AB Exam Questions*. apcentral.collegeboard.com/examquestions

Levine, David. *The SortDetective*. St. Bonaventure University, Department of Computer Science, 2002.
<http://nifty.stanford.edu/2002/LevineSort/>

Litvin, Maria, and Gary Litvin. *Java Methods AB: Data Structures*. Andover, Mass.: Skylight Publishing, 2003.

Xinox Software. *JCreator 3.10*. 2004. www.jcreator.com/

Student Activities

Here are two student activities that I developed for this course.

1. FishStuff12 Lab

The FishStuff12 Lab helps students develop a detailed understanding of the Marine Biology Simulation Case Study classes and methods discussed in chapters 1 and 2 of the MBS narrative. The assignment and starter code are included in appendix B of this book.

2. ListNode and TreeNode Card Exercises

Recursion is a critical technique for implementing list and tree algorithms. These algorithms involve a recursive call for each `ListNode` or `TreeNode` processed. It is critical that students become comfortable with this category of algorithms, and this activity helps them gain an understanding of how these algorithms work. Details for the `ListNode` activities are in appendix B; they can easily be extended for `TreeNode`. Appendix B also contains copies of cards for both `ListNode` and `TreeNode`.

Sample Syllabus 6 (AP Computer Science AB)

Mike Scott

University of Texas at Austin
scottm@sc.utexas.edu
www.utexas.edu

University Profile

Location and Environment: The University of Texas at Austin, known as UT, is one of the largest universities in the United States, with almost 50,000 students enrolled each fall. The vast majority of undergraduates are Texas residents. Some 3,000 faculty members and 18,000 staff persons work at the university. The campus covers 350 acres, but many students live in off-campus housing.

Type: Four-year research university
Total Enrollment: 50,000 (39,000 undergraduate; 11,000 graduate)
Ethnic Diversity: Asian American: 14.3 percent
Hispanic/Latino: 13.3 percent
African American: 3.5 percent
International students make up 9 percent of the student body.

Computer Science Department

Computer science is one of over 100 degree programs offered for undergraduates at UT. The Computer Science Department is consistently ranked in the top 10 of all such departments in schools throughout the country. In spring 2004 the department granted 350 undergraduate degrees. In fall 2004 approximately 1,200 students were majoring in computer science.

Regardless of what students learn first, they need to know how to program in order to study advanced topics in computer science, and so many college computer science departments teach programming first. The UT Computer Science Department follows this model with a three-semester sequence of programming courses. (It is possible to place out of the first course or the first two courses: see below.) Upon completion of these courses, students should be able to design and implement programs of a significant size to solve complex problems—by “significant” I mean a program requiring over 10 new classes and over 1,000 lines of code for a well-designed solution.

The introductory programming track in the department is made up of:

- CS 305J: Fundamentals of Computing (analogous to AP Computer Science A)
- CS 307: Foundations of Computer Science (analogous to AP Computer Science AB)
- CS 315: Algorithms and Data Structures, or CS 315H (for the Computer Science Department Honors Program). These courses include additional data structures and algorithms in addition to covering some of the topics from CS 307 in greater depth.

Students enter the regular programming track at different points. Students with little or no programming experience, or who have not studied computer science in over a year, are advised to take CS 305J. Students who have taken a year of high school programming or a semester at another college are allowed to begin in CS 307. Students are advised that they need to have a strong background to begin in CS 307; I believe any student who received a grade of 4 or higher on the AP Computer Science A Exam is

ready for CS 307. A student who receives a grade of 4 or 5 on the AP Computer Science AB Exam is allowed to register for CS 315. The official AP credit policy of the department is laid out in the following table:

AP Exam	Score	UT Austin Courses, Credits, and Grades Earned
Computer Science A	5	CS 305J (A)
Computer Science A	4	CS 305J (B)
Computer Science AB	5	CS 305J (A) and CS 307 (CR)
Computer Science AB	4	CS 305J (B) and CS 307 (CR)
Computer Science AB	3	CS 305J (C)

Philosophy of the Department

Learning the fundamentals and theory of computing is of the utmost importance. It is not appropriate simply to teach the latest or most popular programming language or set of tools because the languages and tools computer scientists use are constantly changing. The best approach to learning computer science and the fundamentals of computing is to learn how to program in a high-level language; students will learn the theory of programming and computation at the same time.

CS 307: Foundations of Computer Science (CS2)

Personal Philosophy

I thoroughly enjoy teaching this second course in the programming sequence. Students have already learned the basics of syntax, design, and algorithm development, and they are now prepared to handle more analytical topics. This course is fascinating because students start to examine their own design and code with greater depth than they do in CS1. They are learning not just to write programs, but also to evaluate the programs they write.

Class Profile

Those taking CS 307 are primarily first-year students who wish to major in computer science; a small number of transfer students and students from other departments take the course as well. On average, 150 students enroll in the fall semester and 125 in the spring. The course is normally broken into two lecture sections, so students attend lectures with approximately 75 other students. Lecture sections meet three times a week for 50 minutes and are led by the course instructor. In addition, students are assigned to a small discussion section of approximately 25 students that meets once a week for 50 minutes with a graduate teaching assistant. UT is on a semester schedule, and each course lasts 15 weeks.

Students are expected to complete all homework and programming projects on their own time. The Computer Science Department provides labs for students, but they are open labs (see Lab Component section below).

Chapter 3

Course Prerequisites

The official prerequisites are:

One of the following:

- One year of programming in high school
- CS 305J (or CS 303E, a nonmajors CS1 course) with a grade of at least C
- Consent of the instructor

And one of the following:

- Credit or registration for Math 408C or 408K, differential calculus
- A score of at least 520 on the SAT Subject Test in Mathematics Level 1 or Mathematics Level 2

Course Overview

CS 307, Foundations of Computer Science, emphasizes fundamental computer science concepts: algorithm analysis, recursion, recursive backtracking, advanced sorting algorithms, and the implementation and use of data structures.

Required Text

Riley, David D. *The Object of Data Abstraction and Structures Using Java*. Boston: Addison-Wesley, 2002.

Recommended Text

Winston, Patrick Henry, and Sundar Narasimhan. *On to Java*. 3rd ed. Boston: Addison-Wesley, 2001.

In addition, students are assigned reading from the Web.

Course Objectives

Students in this course should learn fundamental computer science concepts including:

- data types, data structures (linked lists, trees, graphs, stacks, and queues), and algorithms
- recursion
- data abstraction and encapsulation
- correctness: specification, testing, and proving
- object-oriented design and implementation
- reasoning about programs' correctness and efficiency

Course Planner

Week (Mtgs)	Topics	Readings: Riley text, JT (Java Tutorial online), other sources listed in Teacher Resources, below	Programming Project / Homework (Riley: suggested problems from the text)
1 (2)	<ul style="list-style-type: none"> ● Course introduction, administrative matters ● Review of basic programming concepts in Java including primitive data types, control structures, methods and parameters, and Strings ● Using objects and using javadoc-style documentation ● Programming with assertions 	<ul style="list-style-type: none"> ● Riley chapter 01 ● JT: Language Basics ● “Programming With Assertions” (Sun) 	Programming project 1 involves reasoning about existing code with various expressions, completing a string matching algorithm using only the charAt and length methods from the String class, and determining the prime numbers that exist in a given range of values.
2 (2)	<ul style="list-style-type: none"> ● Complete review of Java basics ● Pointers and object variables in Java ● Dereferencing pointers and the dot operator ● Sharing between pointers ● Passing pointers by value to methods 	<ul style="list-style-type: none"> ● “Pointers and Memory” (Parlante) 	(None)
3 (3)	<ul style="list-style-type: none"> ● Built-in arrays in Java ● Arrays of object variables ● Resizing arrays ● Multidimensional arrays ● Manipulating 2-D arrays. Examples: filtering an image, Conway’s Game of Life 	<ul style="list-style-type: none"> ● JT: Object Basics and Simple Data Objects, read the Array topic, including all of its subtopics 	Students must: <ul style="list-style-type: none"> ● Determine the Flesch Readability Index of a passage of text (a String) (Horstmann 2002, p. 276) ● Write methods for an array of ints ● Shuffle the elements ● Determine if a majority element exists ● Find all combinations of 3 elements whose sum equals a given target value

Chapter 3

Week (Mtgs)	Topics	Readings: Riley text, JT (Java Tutorial online), other sources listed in Teacher Resources, below	Programming Project / Homework (Riley: suggested problems from the text)
4 (3)	<ul style="list-style-type: none"> ● Introduction to object-oriented concepts: encapsulation, inheritance, and polymorphism ● Creating classes from scratch; determining behaviors and state data ● Instance variables ● Constructors ● Methods ● The equals method and the toString method ● Tight coupling versus loose coupling in class implementation ● Creation of pre- and postconditions for methods ● Static variables and static methods ● The keyword this and its use: constructor redirection and calling object self reference 	<ul style="list-style-type: none"> ● Riley chapter 02 ● JT: Object-Oriented Programming Concepts. Read the following topics: What Is an Object? What Is a Message? What Is a Class? How Do These Concepts Translate into Code? ● “Classes and Objects” (Stein) ● JT: Object Basics and Simple Data Objects: The Life Cycle of an Object. Also read these topics (via links at the bottom of that page): Creating Objects, Using Objects, Cleaning Up Unused Objects. 	<p>Riley chapter 02: 1, 2, 6, 7, 9, 10, 11</p> <p>Students must implement a class that models mathematical matrices that are used to solve systems of linear equations. They are given a completed design with all methods specified and must complete the class. This is intended as an exercise in working with two-dimensional arrays and in creating a stand-alone class. (See appendix B for the complete lab.)</p>
5 (3)	<ul style="list-style-type: none"> ● Concept of inheritance, is-a vs. has-a relationships ● Syntax for inheritance ● Inheritance requirements in Java ● Relation of the Object class to other classes and class hierarchies ● Inheriting behaviors and data regarding state ● Private, public, and protected access modifiers ● Overriding methods ● Use of super to call parent constructors and to use an overridden method from the parent ● Polymorphism and object variables, what is an appropriate data type ● Dynamic dispatch, compiler checks on methods vs. runtime methods called ● Abstract classes and abstract methods ● Interfaces ● Comparable interface ● Implementing generic algorithms 	<ul style="list-style-type: none"> ● Riley chapter 03 ● “An Explanation of Inheritance” (Reges) 	<p>Riley chapter 03:3–12</p> <p>Students must design and implement a program that requires multiple classes. Sometimes the example programs do not require much use of inheritance or polymorphism. My primary goal is to get them to deal with multiple classes they created. Projects have included card games such as Blackjack, War, and Go Fish.</p>

Week (Mtgs)	Topics	Readings: Riley text, JT (Java Tutorial online), other sources listed in Teacher Resources, below	Programming Project / Homework (Riley: suggested problems from the text)
6 (3)	<ul style="list-style-type: none"> ● Exceptions ● Checked vs. unchecked exceptions ● Try-catch blocks ● Throwing exceptions ● Commonly used exceptions from the Java standard library ● Algorithm analysis ● Formal definition of Big-Oh, Big Omega, and Big Theta ● Calculating $t(N)$ for a method or piece of code ● Application of Big-Oh, $f(N)$ vs. $t(N)$ ● Dominance of terms ● Common Big-Oh functions ● Average case vs. worst case performance ● Predictions of running times based on Big-Oh 	<ul style="list-style-type: none"> ● “What’s an Exception and Why Do I Care?” (Sun) ● Riley chapter 03: 3.1–3.3, 3.8 ● “Big-O Notation” (National Institute of Standards and Technology, Dictionary of Data Structures and Algorithms) 	<p>Riley chapter 03: 1, 2, 4, 5, 7</p> <p>Students complete a more traditional homework assignment. I provide a large number of methods and they must determine the Big-Oh of each. They also run methods with different values of N and use a Timer class, based on the <code>System.currentTimeMillis()</code> method to observe the actual runtime of methods with various Big-Ohs.</p> <p>Note: Students learn the theory behind algorithm analysis and the practical skill of determining Big-Oh. This is applied the rest of the term and almost all algorithms and code discussed after this week are analyzed for Big-Oh.</p>
7 (2)	<ul style="list-style-type: none"> ● Midterm exam (one class period) ● Recursion ● Activation records and the program stack ● Base cases vs. recursive steps ● Tail recursion ● Recursive back tracking ● Examples: directory size, minesweeper, solving mazes, the knapsack problem, and the eight queens problem 	<ul style="list-style-type: none"> ● Riley chapter 06 ● “Recursion Programming” (D’Souza) 	<p>Riley chapter 06: 1, 2, 3, 4, 5, 7</p> <p>Students implement approximately 10 methods using recursion. Half the methods can be solved easily using iteration and half are problems where the recursive solution is easier than an iterative solution.</p>
8 (3)	<ul style="list-style-type: none"> ● Linear and Binary Search ● Review of $O(N^2)$ Sorts: bubble sort, insertion sort, selection sort ● Shell sort ● Quick sort ● Merge sort ● Stability of sorting ● Hybrid sorts ● The Comparable and Comparator interfaces 	<ul style="list-style-type: none"> ● Riley chapter 03: 3.4–3.7, 3.9 ● Riley chapter 12 ● Riley chapter 13: 13.1–13.3 	<p>Riley chapter 03: 8, 9</p> <p>Students trace through various searching and sorting algorithms using small data sets.</p>

Chapter 3

Week (Mtgs)	Topics	Readings: Riley text, JT (Java Tutorial online), other sources listed in Teacher Resources, below	Programming Project / Homework (Riley: suggested problems from the text)
9 (3)	<ul style="list-style-type: none"> ● Concept of Abstract Data Types ● Core operations for all ADTs (add, access, remove) ● Introduction to common ADTs ● The Java Collections Framework and the Collection interface ● A List interface ● Implementing an array-based list ● Amortized Big-Oh analysis 	<ul style="list-style-type: none"> ● Riley chapters 01, 02, and 04 	Riley chapter 01: 3, 4; chapter 02: 1; chapter 04: 1, 4 Students implement a Lexicon class, a collection of phrases and words. This assignment involves sorting and searching and construction of a concrete data structure. (See appendix B for the complete lab.)
10 (3)	<ul style="list-style-type: none"> ● Dynamic data structures ● A linked implementation of a list interface ● Singly linked lists, doubly linked lists, circular linked lists, immutable lists 	<ul style="list-style-type: none"> ● Riley chapters 05, 06 	Riley chapter 05: 1, 2, 4 Students implement a List interface with a doubly linked list.
11 (2)	<ul style="list-style-type: none"> ● Midterm exam (one class period) ● A stack interface ● Implementing the stack interface with a native array of objects, an ArrayList, and a LinkedList ● Using stacks to check balanced symbols in source code ● Infix and postfix notation ● Evaluating postfix expressions using a stack ● Converting infix expressions to postfix expressions 	<ul style="list-style-type: none"> ● Riley chapter 07: 7.1–7.5 	Riley chapter 07: 1, 5, 7, 8 Students complete a review assignment for the midterm.
12 (3)	<ul style="list-style-type: none"> ● A queue interface ● Implementing a queue with an ArrayList and a LinkedList ● Using wraparound and a native array of objects to implement a queue ● Priority queues ● Trees ● Definition and properties of binary trees ● Preorder, inorder, postorder, and level order traversals of binary trees 	<ul style="list-style-type: none"> ● Riley chapter 07: 7.6–7.9 ● Riley chapter 09: 9.1–9.6 	Riley chapter 07: 9; chapter 09: 1–5 Students implement a program that finds word ladders, using stacks and queues to do a breadth-first search.
13 (3)	<ul style="list-style-type: none"> ● Huffman coding using binary trees and priority queues ● Definition of binary search trees ● Implementing binary search trees 	<ul style="list-style-type: none"> ● “Huffman Coding” (Wikipedia) ● Riley chapter 09: 9.8–9.11 ● Riley chapter 08: 8.2 	Riley chapter 09: 8 Students implement a simple binary search tree class.

Week (Mtgs)	Topics	Readings: Riley text, JT (Java Tutorial online), other sources listed in Teacher Resources, below	Programming Project / Homework (Riley: suggested problems from the text)
14 (1)	<ul style="list-style-type: none"> • Introduction to balanced binary search trees, red-black trees • Insertion algorithms for red-black trees 		
15 (3)	<ul style="list-style-type: none"> • Introduction to hash tables • Hash functions • Resolving collisions, probing and chaining • Introduction to maps • Introduction to sets • Introduction to graphs 	<ul style="list-style-type: none"> • Riley chapter 08: 8.3–8.6 	

Modification notes for AP CS teachers:

Maps and sets are introduced during the last week of the course—students do not do a programming project involving maps and sets. These are covered in more detail in the subsequent course, CS 315. If I were teaching an AP Computer Science AB course, I would move maps and sets before trees in the course planner. Although I do not use the AP Computer Science case study, I would recommend that CS AB teachers use it throughout the course to highlight various topics, rather than trying to teach it as a separate unit.

Topics covered that are NOT tested on the AP CS AB Exam:

`static non-final` variables; constructor redirection via the `this` keyword; the visibility modifier `protected`; implementing the `equals` method; checked exceptions and `try/catch/finally` statements; the `throws` modifier; formal definition of Big-Oh, Big Omega, and Big Theta; amortized Big-Oh analysis; runtime predictions based on Big-Oh; Shell sort; stability of sorts; red-black Trees.

Teaching Strategies

- My lectures are much more like discussions. I ask questions throughout the class and encourage students to do the same. Most college lectures are not conversational, but I find it beneficial to conduct the class this way. I bring candy to every class to encourage discussion, and anyone who asks or answers a question gets a piece of candy.
- I have slide presentations of almost all topics I teach.
- About 20 percent of lecture time is spent developing and writing code for example problems.
- I have a number of handouts with extra problems for students to work on. These are called “discussion section handouts” and are on the Web. Teaching assistants review the problems in these handouts in the small group discussions.
- I do not reuse exams, so all old exams are posted on the Web; students use these to study for tests.

Lab Component

Students are expected to spend a considerable amount of their own time doing programming projects (7–10 hours per week). There are no closed labs in this course (that is, instruction in the lab, with the teacher

Chapter 3

guiding students through a programming problem). Students are encouraged to work with a partner on programming projects. They are also encouraged to work in the Computer Science Department's microcomputer lab. This lab has approximately 75 computers running various IDEs for students to develop Java programs. Teaching assistants and proctors hold hours in the lab to assist students, and I hold office hours. Many students work from home on their own computers.

Student Evaluation

Category	Weight	Comments
Programming projects and quizzes	15%	Typically there are 12 programming projects and 12 quizzes. Programming projects are graded on a 20-point scale. Quizzes are graded on a 5-point scale. The lowest programming project grade and the lowest quiz grade are each dropped.
Midterm 1	25%	Two hours in length, with 3 coding questions
Midterm 2	25%	Two hours in length, with 3 coding questions
Final exam	30%	Three hours in length, with 4 coding questions

Exams are about 30 percent short-answer questions and 70 percent coding. Coding questions are similar to the free-response questions on the AP Computer Science AB Exam.

Teacher Resources

Books

Arnold, Ken, James Gosling, and David Holmes. *The Java Programming Language*. 3rd ed. Boston: Addison-Wesley, 2000.

Bentley, Jon. *Programming Pearls*. 2nd ed. Boston: Addison-Wesley, 2000.

Gosling, James, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. 2nd ed. Boston: Addison-Wesley, 2000.

Horstmann, Cay. *Big Java*. Hoboken, N.J.: Wiley, 2002.

Horstmann, Cay, and Gary Cornell. *Core Java 2, Volume 1: Fundamentals*. 7th ed. Upper Saddle River, N.J.: Prentice Hall, 2004.

Horstmann, Cay, and Gary Cornell. *Core Java 2, Volume 2: Advanced Features*. 7th ed. Upper Saddle River, N.J.: Prentice Hall, 2004.

Software, Web Resources

BlueJ. IDE.
www.bluej.org

Darby, Gary. DelphiForFun. 2004.
www.delphiforfun.org/index.html

This site is devoted to problems using the Delphi programming environment, which is in turn based on Pascal, but there are many interesting ideas for programming projects.

D'Souza, Erwin Francis. Recursion Programming.
<http://personal.vsnl.com/erwin/recursion.htm>

Helios Software Solutions. *TextPad*. IDE. www.textpad.com
Educational discounts for site licenses.

HowStuffWorks.
www.howstuffworks.com
This is an excellent site for programming project ideas, but there is a lot of advertising.

Huffman Coding. Wikipedia. 2004.
http://en.wikipedia.org/wiki/Huffman_coding

National Institute of Standards and Technology. Dictionary of Algorithms and Data Structures.
www.nist.gov/dads/

Parlante, Nick, ed. Nifty Assignments.
<http://nifty.stanford.edu/>
This is a wonderful collection of really interesting assignments.

Parlante, Nick. Pointers and Memory. Stanford CS Education Library, 2000.
<http://cslibrary.stanford.edu/102/PointersAndMemory.pdf>
This was designed for C++ but is still an excellent explanation of pointers and what can be done with them.

Reges, Stuart. An Explanation of Inheritance. University of Arizona.
www.cs.utexas.edu/users/scottm/cs307/handouts/InheritanceExplanation.htm

Scott, Mike. Computer Science 307: Fundamentals of Computer Science.
www.cs.utexas.edu/users/scottm/cs307
Course Web site.

Stein, Lynn Andrea. Introduction to Interactive Programming in Java: Building New Things: Classes and Objects. 2003.
www.cs101.org/ipij/objects.html

Sun Microsystems, Inc. Trail: Learning the Java Language. 2005.
<http://java.sun.com/docs/books/tutorial/java>

Sun Microsystems, Inc. Programming With Assertions. 2002.
<http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html>

Sun Microsystems, Inc. What Is an Exception? 2005.
<http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html>

Xinox Software. *JCreator*. 2004. IDE. <http://jcreator.com/>

Chapter 3

Student Activities

Two students can work together on many of my assignments. See Pair Programming Standards and Requirements in appendix B.

The following assignments are also detailed in appendix B:

- **Mathematical Matrix**, from week 4 of the syllabus
- **Lexicon**, from week 9 of the syllabus

Chapter 4

The AP Exams in Computer Science

The goal of the AP Computer Science A and AB Exams is to evaluate how well students have mastered the concepts and skills contained in the respective course outlines. This chapter should help you prepare your students to do well on the exams.

All About the Exams

Schedule

AP Exams are administered worldwide in May of each year. Each subject has a designated date and time of administration; the schedule for the upcoming AP Exams can be found online in the Exams section of AP Central.

Eligibility

All students are eligible to take an AP Computer Science (CS) Exam. The College Board strongly discourages practices or policies that restrict students from participating in the AP Program or taking an AP Exam (see Equity and Access section in chapter 1).

Format

The AP Computer Science Exams are three hours long and consist of two sections:

	Percent of Grade	Number and Type of Questions	Minutes Allotted
Section I	50	40 multiple choice	75
Section II	50	4 free response	105

Content

The exams cover the fundamentals of computer science as described in the Topic Outline in the Course Description. Both sections of the exams require students to demonstrate their ability to design, write, analyze, and document programs and subprograms. The exams emphasize programming methodology (including recursion), programming in Java, and procedural abstraction. They also cover algorithms, data structures, and data abstraction. Five to 10 multiple-choice questions and one free-response question are based on the current case study.

Administration

To avoid any conflict of interest, AP Exams are administered by a school's AP Coordinator and designated proctors (see chapter 2). However, in the months leading up to the exams, AP teachers can assist Coordinators with tasks such as the collection of exam fees from students.

Security

The AP Program is committed to reporting grades that accurately reflect a student's performance. For this reason, the AP Program maintains exam administration and security standards designed to ensure that all students are given the same opportunity to demonstrate their abilities, and to prevent any student from gaining an unfair advantage over another because of testing irregularities or improper conduct.

Multiple-choice questions should never be discussed with students. The multiple-choice sections of the AP Exams are kept secure because some questions are reused to ensure the reliability and validity of the exams. Free-response questions are posted on AP Central 48 hours after an exam is administered and may then be discussed by students and teachers. (*Note:* Free-response questions from alternate exams for late testing are not released and can never be discussed.)

For more details, see the exam security article in the AP Coordinator's section of AP Central.

Scoring

AP Exam Readers score the AP Computer Science Exams' free-response questions during the annual AP Reading in June. The Readers are experienced instructors who teach either a high school AP CS course or an equivalent course at a college or university (see chapter 5).

During the days immediately preceding each Reading, the Chief Reader and other Reading leaders, working with Assessment Specialists from ETS, prepare and refine scoring guidelines for each free-response question. These guidelines define the process for computing each student's score for the associated question. (Scores range from 0 to 9 points for each AP CS question, and individual points or half-points are assigned to each portion of the solution.) Training materials, which consist of standard solutions, the scoring guidelines, and several representative solutions, are prepared for each question.

On the first day of the Reading, the Chief Reader gives an overview of the exams and the scoring process. Readers are then divided into groups, one for each question on the A and AB exams, and a thorough training on the question is conducted. Readers score sample student solutions that have been prescored by the leaders, and their scores are compared with the leaders' scores to make sure the scoring guidelines have been applied correctly.

Once the Readers can apply the scoring standards consistently, they begin reading in teams of two, with each student solution read by both persons. Differences are reconciled, with a leader serving as an arbitrator when needed. When the team shows consistent agreement on its scores, its members move to scoring individually, consulting with each other and with the leaders when in doubt about a score. Throughout the Reading, leaders reread a portion of the solutions that have been scored to ensure that the guidelines are applied consistently within the group.

AP Grade Reports

AP grades are reported to students, their schools, and their designated colleges in July. Each school automatically receives an AP Grade Report for each student, a cumulative roster of all students, rosters of all students by exam, an AP Scholar roster for any qualifying students, and a AP Instructional Planning Report. (*Note:* Data for students testing late with an alternate form of the exam are not included in this report.) For a fee, schools may also request their students' free-response booklets.

Using the AP Instructional Planning Report

Schools receive the AP Instructional Planning Report for each of their AP classes in September. The report compares your students' performance on specific topics in the AP Exam to the performance of students worldwide on those same topics, helping you target areas for increased attention and focus in the curriculum. To get the most out of the report, please read the interpretive information on the document. It explains how the data, when used correctly, can provide valuable information for instructional and curricular assessment as well as for planning and development. Contact your school's AP Coordinator for this report.

College AP Credit Policies

Each college sets its own policy for granting credit, advanced placement, or both for AP Exam grades. The policy often takes into account how closely the AP curriculum is aligned with the college's own courses.

For AP Computer Science, the AP policy is often based on the language(s) used in the college's own introductory computer science courses. When the introductory languages don't match, some colleges grant elective credit in the Computer Science department but require computer science majors to start with the college's beginning courses. Other colleges offer a special course for students who have taken AP CS; for example, a condensed course (two semesters in one) or an abbreviated course (fewer hours per week) would be tailored to students who have demonstrated understanding of computer science concepts but don't know the language of that college's introductory courses.

For information about individual colleges' AP policies, visit the Higher Education section of AP Central.

Even if a student doesn't receive college credit or placement for AP Computer Science, he or she will have benefited from the challenge of college-level work and should have increased confidence when taking introductory computer science courses in college.

For more information about the entire exam development and administration cycle, go to the Exams section of AP Central.

Exam Preparation

Suggestions for Teachers

Here are three important guidelines for preparing your students for the exams:

- Organize your course to incorporate the items in the Course Description's Topic Outline and AP CS Java Subset. The Commentary on the Topic Outline, also in the Course Description, provides many details, strategies, and examples for presenting these topics. (See chapter 2 for a link to the online Course Description.)

Chapter 4

- Incorporate the case study as early as possible and frequently. (See appendix A.)
- Give your students plenty of practice with questions like those on the AP Exams.

Include multiple-choice and free-response questions on your own tests, so that students are familiar with the formats. Help students learn to recognize key phrases in the wording of questions so they interpret and answer them correctly. (See “Teaching Tips and Strategies” on the Computer Science Home Pages on AP Central.)

I provide students with a hard copy of the *Quick Reference Guide* (with Java class references) on the first day of class. It’s imperative that they refer to this document often, especially when they are practicing writing answers to free-response questions. Using the guide in this manner reinforces the classes and methods in the AP Java Subset. This makes the task of grading student work somewhat easier, because their code is written within the subset; and there’s a good chance they will continue this practice when they write code on the actual AP Exam.

—Reg Hahne, Atholton High School,
Columbia, Maryland

[Reference Materials for the Computer Science A or AB Exam (in PDF format) are available at apcentral.collegeboard.com/compscica or apcentral.collegeboard.com/compsciab.]

Try to simulate the exam format as much as possible, even if you use smaller chunks of time. Give a set of multiple-choice questions during one class period, making sure that the number of questions is proportionate to the number on the AP Exam (40 in 75 minutes). At other times, give a 30-minute free-response question. Assign a free-response question for homework, specifying a 30-minute limit. (The effectiveness of this strategy will depend on your students’ willingness to adhere to the limit and their work ethic in a nontest situation.) Some high schools administer a whole practice exam during a three-hour period.

Students never use a computer for any assessment in my class. Every test or quiz includes a section where they must handwrite code. I try to grade in an AP style as much as possible. Early in the course that is not practical, as the questions are pretty trivial, but as the course progresses, the code-writing section gets more difficult. By the middle of the year, students are getting a free-response question on every quiz or test. I select certain student responses and distribute them for the class to correct. (I type the responses so that students do not recognize handwriting and know whose answer they are correcting.) This is my sixth year teaching AP Computer Science. My students have told me that since they have seen free-response questions all year, they do not feel overwhelmed when they see them on the AP Exam.

—Jim Westbury, Rockwood Summit High School,
Fenton, Missouri

Sources for Sample Questions

Here are several good sources of questions like those that are found on AP Computer Science Exams:

AP Central

- Free-response questions from prior years' AP Exams. Use the detailed scoring guidelines to score your students' solutions, which will help them understand the benefit of demonstrating their understanding of the algorithms, even if their solutions aren't completely correct.
[AP Central > The Exams > Exam Questions]
- *AP Computer Science Course Description*
Download the latest version from the Course Home Pages on AP Central.

AP Computer Science Review Books

Horwitz, Susan. *Addison-Wesley's Review for the AP Computer Science Exam in Java*. Boston: Addison-Wesley, 2004.

Levine, David, and Kathy Larson. *5 Steps to a 5: AP Computer Science*. New York: McGraw-Hill, 2005.

Litvin, Maria. *Be Prepared for the AP Computer Science Exam in Java*. 2nd ed. Andover, Mass.: Skylight Publishing, 2006.

Litvin, Maria, and Gary Litvin. *175 Multiple-Choice Questions in Java*. Andover, Mass.: Skylight Publishing, 2005.

Teukolsky, Roselyn. *Barron's How to Prepare for the AP Computer Science Advanced Placement Examination (Java Version)*. 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.

Trees, Fran, and Cay Horstmann. *Advanced Placement Study Guide to Accompany Cay Horstmann's Java Concepts Study Guide*. 4th ed. Hoboken, N.J.: Wiley, 2005.

Exam Tips for Students

Discuss test-taking strategies with your students. You may print and distribute the tips I've shared with my class (see below). This document can also be found at:

[AP Central > Computer Science A (or AB) > Exam Tips]

Tips for Students Taking AP Computer Science Exams

Before the Exam

1. Allow more than one night for review. If you have questions, you'll have time to ask someone else for help.
2. Review the information about course content in the *AP Computer Science Course Description*.
 - a) The Topic Outline lists the major topics covered on the AP CS Exams.

Chapter 4

- b) The Commentary on the Topic Outline explains the outline more fully and gives some Java-language examples.
- c) The AP Computer Science Java Subset (appendix A) lists Java language features that will be tested.
- d) The Standard Java Library Methods Required for the A (appendix B) or AB (appendix C) Exam are important: you will be given reference guides to the classes when you take the exam, but being aware of the existence of some of the less common class methods might save you time.
- e) Implementation classes for linked list and tree nodes are covered in CS AB (appendix D).

If you find anything in these sections that is not familiar to you, ask your teacher for clarification.

- 3. Review the case study, both the code and the narrative.
- 4. Review all of your tests from your AP CS course.
- 5. Work sample problems from the Course Description and AP CS Exam review books.
- 6. Don't spend time memorizing specific code, e.g., searches and sorts. Recognizing a binary search will help you analyze it more quickly, but no questions will require regurgitation of specific code.
- 7. Get a good night's sleep before the exam.
- 8. Bring pencils and a watch to the exam.

General Tips

- 1. Problems are usually placed on the exam in order of perceived difficulty. However, some topics that are generally considered difficult might be easier for you than others, so make sure you allow yourself time to try each problem.
- 2. If you get stuck on a problem, don't waste time. Circle the number in the exam booklet and come back to it later.
- 3. Glance at your watch periodically, especially if you find yourself having difficulties or getting distracted. That should help to get you back on track.
- 4. Consult the *Quick Reference* documents during the exam. Be sure to call all methods with the correct parameters.
- 5. Use the case-study code as a syntax and logic guide for all problems, not just case-study problems. You can find examples of almost every kind of syntax you'll need to write or interpret.

Multiple-Choice Questions

- 1. Know when to guess. If you can eliminate one or two of the choices, the odds are in your favor.
- 2. Work backwards, if possible, to eliminate some answers.
- 3. Read all the answers before making your final decision.

Free-Response Questions

- 1. Make sure you have answered the problem.
 - a) As you first read the problem, underline, circle, or star important details.
 - b) Watch out for "You will receive no credit if ..." or "You will not receive full credit if ..."
 - c) Reread the problem definition after you think you've finished writing the code, paying close attention to the details you marked.

- d) Write your code to satisfy the formal definition of the problem, but check your code with all the examples provided. (They are usually chosen by the exam developers to illustrate some of the special cases, and you're wasting a valuable resource if you ignore them.)
 - e) Sometimes the preconditions and postconditions outline the algorithm to solve the problem. Don't ignore them.
2. Never let the beginning of a problem prevent you from getting points at the end.
 - a) Most problems have multiple sections, and sometimes the later sections are easier than the earlier ones.
 - b) You will frequently be told that you may make calls to one or more methods that you were asked to write in previous sections of the problem. The instructions might say, for example, "Assume that [method] works as specified, regardless of what you wrote in part (a). Solutions that reimplement functionality provided by this method, rather than invoking it, will not receive full credit." The best solution will probably include a call to the method written in part (a). You may earn credit for calling the method correctly, even if the code for the method you wrote for part (a) was incorrect or blank. However, if you copy the code from that method into the new section, even if it's completely correct, you will not earn full credit for your code.
 3. Aim for clarity in your programming.
 - a) Organize, indent, assign meaningful variable names, and write neatly. (If you accidentally omit a curly brace, but your indentation clearly conveys your intent, you'll be given the benefit of the doubt.)
 - b) Remember that humans will be grading your work. Be legible.
 - c) Commenting will not help your score, except perhaps if your code is confusing and the comment enlightens the Exam Reader about what you were trying to accomplish. (However, comments are not a substitute for correct code!)
 4. Never give a "recipe" for the answer. If you know how to do even part of the problem, write code for the part you know. Partial credit is often awarded for having the correct loop bounds, initialization in the presence of an attempt to sum values in an array, and so on. Credit is not given for writing a description of what you would do if you had time.
 5. Occasionally, you'll be asked to "justify your answer." You generally won't earn any credit without a justification—a sentence or phrase is often sufficient.
 6. If you write two solutions to a problem, cross out the one you don't want to have scored. Don't waste time erasing it; a simple X is sufficient. (If a Reader finds two solutions to a problem, and neither one is crossed out, the first one will be scored.)
 7. Don't waste time copying the method header or pre- and postconditions; start writing the code just below them.
 8. If you run out of room:
 - a) Find a blank page and continue your code.
 - b) Indicate clearly what you've done, both on the page where you started the response and on the page where you continued it.
 9. Avoid using classes that aren't specifically given to you as part of the exam and aren't part of the Java language or the case study. While you may have developed or used other classes in your AP course, don't use them in your responses on the AP CS Exam.

Good luck, and do your best!

—Debbie Carter, Lancaster Country Day School, Lancaster, Pennsylvania

After the Exam

The AP Computer Science Exams have been administered, but you still have some days or weeks of school before summer break. You want to allow your students to kick back a bit but still keep them motivated to learn and create. Here is your opportunity to explore new areas and have some fun.

Roseann Krane maintains a Web page called “What to do after the College Board Exam from the FAB Team!!” with links to over 30 Web sites that offer ideas for programming, Web, and general computer projects: <http://dimacs.rutgers.edu/~rkrane/afterAP/>.

Other AP Computer Science colleagues have found success with projects like these:

- Team development project.
 - Maria Litvin has developed two excellent projects, complete with instructor notes, GUI classes, and *PowerPoint* presentations: “The Game of SET: A Case Study in OO Design and Team Development” and “Team OOP Projects as a Teaching Tool (The Quizard of OOP).” Both projects are available at the Web site Java/OOP Workshops and Projects: www.skylit.com/oop/index.html.
- Applets
- Graphics
 - Sun Microsystems, Inc. Trail: Creating a GUI with JFC/Swing (also known as “The Swing Tutorial”). 2005. <http://java.sun.com/docs/books/tutorial/uiswing/>
 - Fractal graphics
- Robotics
 - Lego Mindstorms. leJOS: Java for the RCX. 2004. <http://lejos.sourceforge.net>
 - Alphaworks. *Robocode*. 2001. <http://robocode.alphaworks.ibm.com> Introduces objects and inheritance.
- Another language
 - Kopplin, John. Assembly Language: Free Self-Paced Course on the 8051 Microprocessor and Assembly Language Programming. 2005. www.computersciencelab.com/lyris.htm
 - The TeachScheme! Project. www.teach-scheme.org
 - C#, Python, Visual Basic
- Web development (DHTML, XML, WebMatrix, Java Servlets, JavaServer Pages)
- Huffman coding
 - http://en.wikipedia.org/wiki/Huffman_coding
Huffman compression algorithm
- Additional data structures: graphs

- Student-developed programs for clients (teachers, administrators, local businesses)
- Programming for handheld devices
- Database interfaces

Consider choosing a subject in which you're not an expert; by this point in the year, you probably know your students well enough to know how much responsibility they are willing to take for their own learning. You could define a specific assignment or allow them to explore the subject and then formulate a proposal for their own project. Students are often quite motivated to work on a project that they have defined for themselves.

Advice on student-defined projects: Help students set reasonable goals for the time available. Many students don't have a realistic idea of what they can accomplish in a given period of time. A group of my students wanted to produce a chess game in which the user would play the computer; they had two weeks in which to complete the project—and they were seniors! Trying not to dampen their spirits, I encouraged them to start small and build on their accomplishments; so they began with checkers, a (reusable) board class, and two human players choosing the moves. Once they got into the project, their plans became more realistic; and the checkers project was terrific—after a player clicked on a game piece, the computer highlighted the possible moves, and the game class handled multiple jumps, kings, and other tricky aspects of the game. Completing the checkers project proved much more satisfying than partially finishing a chess game that might not have been able to run at all, and students could still work on the chess game, if they wanted to, over the summer.

My students were dreading doing any more programming assignments but ... I've introduced them to *Robocode* from IBM (it's free!) and now, without any prompting, they're spending time at lunch and after school developing battle robots in Java. It reinforces inheritance and polymorphism (among other topics) and comes with its own environment for editing and compiling Robots. (All you need to have installed is a Java runtime environment.)

—Brad Lindemann, Bellarmine College Preparatory,
San Jose, California

Chapter 5

Resources for Teachers

How to Address Limited Resources

What teacher suffers from an overabundance of teaching resources? We need hardware, software, printed materials, and professional development opportunities—and just when we’re tempted to feel comfortable with the status quo, the technology changes, calling for software and/or hardware upgrades, textbook changes, and workshops to update our skills. Even when changes aren’t mandated, exciting new possibilities present themselves, and we want to take advantage of them to enhance our courses. We must be good stewards of our resources.

Below are some suggestions for making the most of limited resources. Even if you have enough to cover the essentials, these suggestions may help you stretch your dollars so that you can purchase some nonessentials to enrich your curriculum.

No money for software?

You can teach your entire course with free software. Sun’s Java Development Kit is free, as are several Integrated Development Environments (IDEs) and supplemental programs. Look for “F” (free) items in the Software chart at the end of this chapter.

In the not-quite-free-but-still-a-bargain category, all listed vendors have some sort of educational pricing, either a flat fee per school, or site-license fees, based on quantity. Look for references to “Academic” or “Educational” pricing on their Web sites.

Not enough computers (or not enough robust computers)?

Try pair programming. In addition to reducing the hardware demand, this approach has added educational benefits. For guidelines, see Mike Scott’s Pair Programming Standards and Requirements in appendix B.

Outdated textbooks?

Many school systems have a textbook approval/purchase cycle of five to seven years. If you’re stuck with C++ textbooks, or you’re not happy with a textbook that was chosen by someone else, take a look at the listings at the end of this chapter for textbooks and teaching tools that can be downloaded from the Web.

Using too much paper?

For years I required students to submit paper copies of their source code so that I could write comments on the listings. I liked being able to annotate in a contrasting color, and I often wrote suggestions on the

side, complete with circles and arrows for clarity. However, my environmental (and frugal) conscience kept nagging at me, so I sought a paperless solution.

I tried pasting students' code into a word processing document and writing changes in a different color, but I found it difficult to write a block comment beside the code (too much tabbing), and it was generally very awkward.

Then I made a discovery: *Microsoft Word* allows me to "Track Changes," showing my modifications in a different color. (*OpenOffice* has a similar feature; I'm guessing that most other word-processing programs do, too.) *Word* also has drawing tools for creating text boxes, ovals, and arrows.

Through trial and error, I have discovered a process that works well for my classes. I now paste student code into a *Word* document, annotate it directly using "Track Changes," and then return a digital copy of that document to the student. Here is an example of a portion of a graded assignment:

```
/** Moves this fish in its environment.
 * will move one cell forward and to the right
 * if this cell is open. Otherwise, it will stay
 * in the same location. Regardless, it changes
 * it's direction to the right every time-step.
 **/
protected void move()
{
    // Find a location to move to.
    Debug.print("CircleFish " + toString() + " attempting to move.");
    Location nextLoc = nextLocation();

    // If the next location is different, move there
    if ( ! nextLoc.equals(location() )
    {
        changeLocation(nextLoc);
        changeDirection(direction().toRight());
        Debug.println(" Moves to " + location());
    }
    else
    {
        // Otherwise, turn right.
        changeDirection(direction().reverse());
        Debug.println(" Now facing " + direction());
    }
}
```

it's == it is

From the instructions:

"If the fish cannot move as described above, it stays in its current location but still turns 90 degrees to the right."

(Your comment tells what *should* happen, but that's not what happens.)

Limited funds for supplemental resources?

As computer science educators, we have the good fortune of having a multitude of colleagues who generously share resources on the Web. Even commercial organizations and individuals who charge fees to earn a living have shared materials free of charge. Look for Web resources in the charts at the end of this chapter.

Grant writing is one way of attempting to obtain funding for various projects. If you need financial assistance in the short term, you'll want to approach granting agencies such as government and business groups that entertain requests for one-shot projects. Your proposal should clearly state a need, show how you plan to meet that need, and explain how you will evaluate the outcome. Answer all of the required questions; if you can obtain the guidelines the funding agency uses to judge requests, make sure your proposal specifically addresses them.

Unfortunately, grant writing can be fraught with difficulties, not the least of which is that demand far exceeds supply. An alternate approach is to build partnerships with local organizations that can yield long-term benefits for your program and school. Such benefits could include monetary support, but this should not be the prime reason for establishing partnerships.

Over many years at our school, we have sought to build long-term partnerships with businesses and universities in our area. With the help of our partners, we've achieved a number of goals—goals generated in collaboration with those partners, so that everyone feels they have contributed to our success. We've been able to provide students and staff with opportunities that we couldn't have provided by ourselves.

We've followed this approach when developing partnerships:

- Invite prospective partners to school to see your program and to talk with students informally.
- Use current partners to help you gain access to new partners.
- Hold regular meetings to keep partners informed about your school and program.
- Do NOT ask for money at any time. By the time a partner saw that we needed financial assistance, we had already built a relationship of trust and understanding. If partners want to spend their money on you, they'll tell you.

I have many examples of these fruitful relationships. For example, when our school needed tutoring help for students, planning for computer labs, and access to certain software and mentoring for students, various partners came forward with staff time and company resources to help us meet these needs. The same was true when we needed help developing employability skills programs, job-shadowing experiences, internships both for students and staff, professional development opportunities, video and production graphics support, and so forth.

Some of our partnerships began when a business contacted us. For example, last year we started a robotics team at the urging of a local engineering firm. Not only did the firm donate a substantial amount of money for the development of this project, but several engineers volunteered to help students plan and construct a large robot for regional and national competition.

Building partnerships is time-consuming. It requires the help of influential people such as your principal and perhaps the superintendent. But the rewards are great. Our school has a high percentage of low-income families, but successful partnership programs have also been developed in moderate-income and affluent areas, and in rural, suburban, and urban communities. Such relationships have yielded immense benefits to students, schools, and their partners.

—Joe Kmoch, Washington High School,
Milwaukee, Wisconsin

Check out my new lab: new computers, flat-screen monitors, a ceiling-mounted projector, and classroom sets of three different test-prep books. Plus, I've been able to attend the past three SIGCSE (Special Interest Group on Computer Science Education) conferences, and I get paid to work on my curriculum.

It wasn't always this way. Five years ago, our donated computers had 486-33MHz chips. I could display one-ninth of a computer screen at a time with a scan converter and a 25" TV set on top of a storage cabinet. We had a few miscellaneous C++ books, and we did our best.

What changed? We're now part of Arizona's Career & Technical Education (CTE) program. The Business Information Technology component of CTE encompasses four strands: Computer Maintenance, Network Technology, Software Development, and Web Page Development.

CTE programs vary by state, and requirements for teachers probably vary as well. I was already CTE-certified since I had worked in the field and had previously taken six hours of required classes in another state. Why not find out what your state offers?

Cautionary note: There is definitely more work involved—meetings, reports, clubs. Much of the extra work, though, does benefit the students. I meet with an Advisory Board several times a year; I'm also required to teach "workplace skills," which include teamwork, oral and written communication, and work ethics. The complete list of Arizona competencies is available on my Web site (see below).

Reports that must be completed include copying IEPs for students with learning accommodations, tracking attainment of competencies, surveying recent graduates by phone or e-mail, and filling out miscellaneous forms.

Many of my students are part of the Academy of Information Technology, a program of the National Academy Foundation (NAF) that provides internships and scholarships. NAF membership also provides staff development opportunities for teachers.

Last year we reactivated our school's chapter of Future Business Leaders of America (FBLA), and my 14 students won 29 trophies at regional and state competitions. At the national leadership conference last summer, the sixth-place prize for Java Programming went to one of my students (who also earned a 5 on the AP Computer Science AB Exam and is currently attending Arizona State University with a scholarship from FBLA).

I've always had to do extra work to retain my AP Computer Science sections. It sure is nice to be recognized and rewarded with new equipment!

—Renee Ciezki, Ironwood High School,
Glendale, Arizona

References

- Academy of Information Technology. National Academy Foundation. www.naf.org/cps/rde/xchg/SID-3F57E0FB-D3A4472F/naaf/hs.xsl/295_349.htm
(Or go to www.naf.org/cps/rde/xchg and click on "About AOIT.")
- Ciezki, Renee. Computer Lab. 2004. http://staffweb.peoriaud.k12.az.us/Renee_Ciezki/default_files/NewLab.htm
- Competency and Indicator List: Business Information Technology Services: Software Development—
Option C. Arizona Department of Education, 2004. http://staffweb.peoriaud.k12.az.us/Renee_Ciezki/SoftwareDevOptionC.pdf
- Future Business Leaders of America-Phi Beta Lambda. www.fbla-pbl.org

No room for a simulated exam in your building?

In southeastern Pennsylvania, the Lancaster-Lebanon Intermediate Unit administers practice AP Calculus and AP Statistics Exams in late April for high school students. For each exam, students come from the two-county area to spend the day at Millersville University. Multiple-choice questions come from a variety of sources, but the free-response questions are taken from the previous year's AP Exams. Teachers are trained by experienced AP Readers to score the solutions, using the scoring guidelines and training materials from that year's AP Reading. Each student response is scored by at least two teachers, and teachers gain valuable experience while enjoying camaraderie with their peers.

Chapter 5

If you don't have the facilities to administer practice exams at your school, you might want to approach a local college's Computer Science Department about providing such a service.

Resources

Many excellent resources are available for teachers of AP Computer Science, but how do you find them? In this section we include several different types of resources: textbooks, College Board publications, exam preparation materials, software (and tech support for installing it), instructor resources, videos, contests, and professional organizations.

Most of the charts have a **Review?** column: a **Y** specifies that the associated item has a review on AP Central. [AP Central > Teachers' Resources] (Some newer versions are now available.)

Textbooks

Title/Reference	A/AB *	Review ?	Print/ Web
Bruce, Kim B., Andrea Pohorecky Danyluk, and Thomas P. Murtagh. <i>Java: An Eventful Approach</i> . Upper Saddle River, N.J.: Prentice Hall, 2005. Sample chapters can be viewed at http://cortland.cs.williams.edu/~cs134/eof .	A	Y	P/W
Deitel, H. M., and P. J. Deitel. <i>Java: How to Program</i> . 6th ed. Upper Saddle River, N.J.: Prentice Hall, 2004.	A/AB	Y	P
Horstmann, Cay. <i>Big Java</i> . 2nd ed. Hoboken, N.J.: Wiley, 2005.	A/AB	Y	P
Horstmann, Cay. <i>Java Concepts</i> . 4th ed. Hoboken, N.J.: Wiley, 2005. (#)When paired with Trees' <i>AP Study Guide</i> (below), all AB topics are covered.	A (#)	Y	P
ICT. <i>ICT's Advanced Placement Computer Science Curricula for Teachers</i> . Sunnyvale, Calif.: Institute of Computer Technology, 2003. (Flat-rate license permits duplication for instructional purposes. Ordering info: www.ict.org/apcs.html)	A/AB		P
Kjell, Bradley. <i>Introduction to Computer Science Using Java Technology</i> . Central Connecticut State University, 2003. www.javacommerce.com/displaypage.jsp?name=cs151java.sql&id=18218	A	Y	W
Lambert, Ken, and Martin Osborne. <i>Fundamentals of Java: AP Computer Science Essentials for the A and AB Exams</i> . 3rd ed. Boston: Course Technology, 2006.	A/AB	Y (2nd ed.)	P
Lambert, Ken, and Martin Osborne. <i>Fundamentals of Java: AP Computer Science Essentials for the A Exam</i> . 3rd ed. Boston: Course Technology, 2006.	A	Y (2nd ed.)	P
Lewis, John, William Loftus, and Cara Cocking. <i>Java Software Solutions for AP[®] Computer Science</i> . Boston: Addison-Wesley, 2003.	A/AB	Y	P
Litvin, Maria, and Gary Litvin. <i>Java Methods A & AB: Object-Oriented Programming and Data Structures</i> . Andover, Mass.: Skylight Publishing, 2006.	A/AB	Y (1st ed.)	P
Mercer, Rick. <i>Computing Fundamentals with Java</i> . Wilsonville, Or.: Franklin, Beedle & Associates, 2002.	A	Y	P
Poplawski, David A. <i>Objects Have Class! An Introduction to Programming with Java</i> . New York: McGraw-Hill, 2002.	A	Y	P

***Coverage of A and/or AB curriculum (as defined in the AP CS Course Description's Topic Outline):**

A/AB: Provides complete (or nearly-complete) coverage of the entire curriculum

A: Primarily covers topics from the first column of the Topic Outline

AB: Primarily covers topics from the second column of the Topic Outline

College Board Resources

Title/Reference	Print/ Web
<i>The next four items can be downloaded from the Course Home Pages: apcentral.collegeboard.com/compscia (Computer Science A) or apcentral.collegeboard.com/compsciab (Computer Science AB).</i>	
College Board. <i>AP Computer Science Course Description</i> . New York: College Entrance Examination Board.	W
College Board. Teaching Tips and Strategies.	W
College Board. <i>GridWorld Case Study</i> . 2007.	W
College Board. <i>2004 AP Computer Science A and Computer Science AB Released Exams</i> . New York: College Entrance Examination Board, 2005.	Print only

For general assistance in navigating AP Central, refer to the AP Central Quickstart Guide at the end of this chapter.

Exam Preparation Materials

Title/Reference	Review ?	Print/ Web
Horwitz, Susan. <i>Addison-Wesley's Review for the AP Computer Science Exam in Java</i> . Boston: Addison-Wesley, 2004.	Y	P
Levine, David, and Kathy Larson. <i>5 Steps to a 5: AP Computer Science</i> . New York: McGraw-Hill, 2005.		P
Litvin, Maria. <i>Be Prepared for the AP Computer Science Exam in Java</i> . 2nd ed. Andover, Mass.: Skylight Publishing, 2006.	Y	P
Litvin, Maria, and Gary Litvin. <i>175 Multiple-Choice Questions in Java</i> . Andover, Mass.: Skylight Publishing, 2005.	Y	P
Teukolsky, Roselyn. <i>Barron's How to Prepare for the AP Computer Science Advanced Placement Examination (Java Version)</i> . 2nd ed. Hauppauge, N.Y.: Barron's Educational Series, 2003.	Y	P
Trees, Fran, and Cay Horstmann. <i>Advanced Placement Study Guide to Accompany Cay Horstmann's Java Concepts</i> . 4th ed. Hoboken, N.J.: Wiley, 2005.	Y (3rd ed.)	P

See chapter 4 in this *Teacher's Guide* for additional resources that are not primarily exam-related but have good sample exam questions.

Software

Title/Reference	Review ?	Free/\$
Java Language		
Java Technology. Sun Microsystems, Inc. http://java.sun.com		F
J# language. (a component of <i>Visual Studio .NET</i>) Microsoft. http://msdn.microsoft.com/academic/program/highschool/default.aspx See license details under Visual Studio, below.	Y	F/\$
Integrated Development Environments (IDEs)		
<i>BlueJ</i> . www.bluej.org	Y	F
<i>CodeWarrior</i> Technology. Metrowerks. www.metrowerks.com Academic pricing.	Y	\$
<i>Eclipse Project</i> . www.eclipse.org/eclipse/index.html		F
<i>JavaStick USB</i> . www.javastick.com An IDE that runs from a memory stick (no installation). Educational and quantity discounts.	Y	\$
<i>JCreator</i> . Xinox Software. http://jcreator.com Academic licenses and quantity discounts.	Y	F/\$
<i>Step into Java</i> . www.publicstaticvoidmain.org An online environment in which code is written and run (no installation).	Y	F
<i>TextPad</i> . Helios Software Solutions. www.textpad.com Educational discounts for site licenses.	Y	\$
<i>Visual Studio .NET</i> . Microsoft. http://msdn.microsoft.com/academic/program/highschool/default.aspx License covers all lab computers and take-home installations for students and teachers. Free in many states, available for a flat fee in others. (Contact msdnaa@microsoft.com for details.)	Y	F/\$
Supplemental Software		
Alphaworks. <i>Robocode</i> . 2001. http://robocode.alphaworks.ibm.com Introduces objects and inheritance.	Y	F
Bergin, Joseph et al. <i>Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java</i> . Redwood City, Calif.: Dreamsongs Press, 2005. http://csis.pace.edu/~bergin/KarelJava2ed/Karel%2B%2B%20JavaEdition.html Introduces objects and inheritance.	Y	F
<i>Jeroo</i> . 2002. www.jeroo.org Introduces objects.	Y	F
Lambert, Kenneth A., and Martin Osborne. <i>BreezySwing</i> and <i>Terminal IO</i> . http://faculty.cs.wvu.edu/martin/Software%20Packages/BreezySwing/Default.htm This is a graphics toolkit developed to accompany Lambert and Osborne's Java textbooks, but it also works well on its own.		F
Litvin, Maria, and Gary Litvin. <i>Java Methods A & AB, Appendix E: EasyReader, EasyWriter, EasySound, EasyDate, and EasyFormat classes</i> . www.skylit.com/javamethods/JM-Appendix-E.html		F
Microsoft Corp. Microsoft® MBS Case Study. 2004. www.mainfunction.com/DotNetInAction/Technologies/display.aspx?id=2673 (J# version, runs under <i>Visual Studio .NET</i> 2003.)	Y	F

Title/Reference	Review ?	Free/\$
The objectdraw Library. Williams College. http://cortland.cs.williams.edu/~cs134/eof/library/ A simplified graphics package for handling mouse events and displaying simple drawings. Designed to accompany Bruce/Danyluk/Murtagh textbook listed above but also works well on its own.	Y	F
Reges, Stuart. Java <i>TextReader</i> Class. University of Arizona, 2001. www.cs.arizona.edu/people/reges/text.html A simplified class for reading input from the keyboard.		F

Tech Support on the Web (installing Java language and IDE software)

Reference	Sun JDK	BlueJ	Code Warrior	Eclipse	JCreator	TextPad	Visual Studio
Litvin: www.skylit.com/javamethods/faqs/index.html	Y				Y		
Horstmann: http://horstmann.com/bigj/help/index.html	Y	Y	Y	Y	Y	Y	
MainFunction (post a question to the forum): www.mainfunction.com							Y
Wittry: www.apcomputerscience.com/#ide		Y			Y		

Instructor Resources (references, projects, lessons, administrative tools)

Title/Reference	Review ?	Print/ Web
Arnold, Ken, James Gosling, and David Holmes. <i>The Java Programming Language</i> . 4th ed. Boston: Addison-Wesley, 2005.	Y (3rd ed.)	P
Balci, Osman et al. Online Interactive Modules for Teaching Computer Science: Animations to Assist Learning Some Key Computer Science Topics. Virginia Tech, Department of Computer Science. http://courses.cs.vt.edu/~csonline/		W
Baldwin, Richard G. C#/Java/JavaScript/XML Programming Tutorials. www.dickbaldwin.com/toc.htm See especially his “Essence of OOP using Java” and “AP Java Study Guide” series.	Y	W
Bentley, Jon. <i>Programming Pearls</i> . 2nd ed. Boston: Addison-Wesley, 2000.		P
Blackboard. www.blackboard.com Course-hosting Web service.		W
Brady, Alyce. AP Computer Science Teaching Resources. Kalamazoo College, 2003. http://max.cs.kzoo.edu/AP/	Y	W
Brill, Gregory, ed. Code Notes for J#. PDF version. New York: Random House, 2003. www.codenotes.com/downloads/downloadsJSbookAction.aspx		W

Chapter 5

Title/Reference	Review ?	Print/ Web
Brummond, Nils. 1996. Object Oriented Analysis and Design using CRC Cards. www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/	Y	W
Carter, Debbie. Java Resources for AP Computer Science Teachers. Lancaster Country Day School, 2005. http://e-lcds.org/fac/carter/JavaResources.htm	Y	W
Cockburn, Alistair. 1999. Using CRC Cards. http://alistair.cockburn.us/crystal/articles/ucrc/usingcrccards.html		W
Computer Ethics Institute. Ten Commandments of Computer Ethics. 1992. www.brookings.edu/dybdocroot/its/cei/overview/Ten_Commandments_of_Computer_Ethics.htm		W
Darby, Gary. DelphiForFun. 2004. www.delphiforfun.org/index.html This site is devoted to problems using the Delphi programming environment, which is in turn based on Pascal, but there are many interesting ideas for programming projects.		W
D'Souza, Erwin Francis. Recursion Programming. http://personal.vsnl.com/erwin/recursion.htm		W
Frank, Roger. Lab Assignments for the High School Computer Science Classroom. 2005. www.rfrank.net/cslabs/cslabs.htm		W
Gosling, James, Bill Joy, Guy Steele, and Gilad Bracha. <i>The Java Language Specification</i> . 3rd ed. Boston: Addison-Wesley, 2005.		P
Green, Roedy. Java Glossary: Error Messages. Canadian Mind Products, 2005. http://mindprod.com/jgloss/errormessages.html This is a fairly long list of Java error messages, with a description of each.		W
Green, Roedy. Java & Internet Glossary. Canadian Mind Products, 2005. http://mindprod.com/jgloss/jgloss.html		W
Hausner, Alejo, Sandeep Mitra, and Peter Brummund. Algorithm Animations: Sorting Algorithm Animations. http://math.ucsd.edu/~fan/math188/bonus/park/sorting.htm	Y	W
Horstmann, Cay. <i>Object-Oriented Design & Patterns</i> . 2nd ed. Hoboken, N.J.: Wiley, 2005.		P
HowStuffWorks. www.howstuffworks.com This is an excellent site for programming project ideas, but there is a lot of advertising.		W
JJ Dream Team and numerous contributors. The AP CS Java Resource CD. http://apcsteacherjavaresourcecd.dev.java.net/ This is not actually a CD but an extensive collection of downloadable projects, instructional materials, exam questions, textbooks, and animations.	Y	W
Levine, David, and Steven Andrianoff. Role Playing in an Object-Oriented World. St. Bonaventure University, Department of Computer Science, 2003. http://web.sbu.edu/cs/dlevine/RolePlay/roleplay.html	Y	W
Litvin, Maria, and Gary Litvin. <i>Java/OOP Papers and Projects</i> . www.skylit.com/oop/index.html		W
Malpohl, Guido. <i>Jplag</i> . 2005. www.jplag.de Online plagiarism-detection system.		W
<i>MazeWorks</i> . Java Games & Puzzles: Tower of Hanoi. www.mazeworks.com/hanoi/ For introducing recursion.	Y	W
Microsoft Corp. .NET In Action—Curriculum. MainFunction, 2005. www.mainfunction.com/DotNetInAction/Curriculum/default.aspx		W

Title/Reference	Review ?	Print/ Web
Microsoft Corp. .NET In Action—Reality Check. MainFunction, 2005. www.mainfunction.com/DotNetInAction/RealityCheck/default.aspx A great resource for bringing current events into the classroom.	Y	W
Microsoft Developers Network Academic Alliance Development Center. www.msdnaa.com	Y	W
Mitra, Sandeep. Java Sorting Animation Page. Computer Science at SUNY Brockport. 1999. www.acs.brockport.edu/~smitra/cs/javasort.html		W
Morris, John. Data Structures and Algorithms: Sorting. 1998. www.eng.tau.ac.il/~shtilman/C-programming/Year2/sorting.html Includes animation of Insertion Sort.		W
MOSS (Measure of Software Similarity). www.cs.berkeley.edu/~aiken/moss.html Online plagiarism-detection system.		W
National Institute of Standards and Technology. Dictionary of Algorithms and Data Structures. www.nist.gov/dads/		W
Pair Programming. www.pairprogramming.com		W
Parlante, Nick, ed. Nifty Assignments. http://nifty.stanford.edu/ A wonderful collection of really interesting assignments.		W
Reges, Stuart. An Explanation of Inheritance. University of Arizona. www.cs.utexas.edu/users/scottm/cs307/handouts/ InheritanceExplanation.htm		W
Seuss, Dr. <i>The Cat in the Hat Comes Back</i> . New York: Random House, 1958. (Recursion)		P
Seuss, Dr. “Too Many Daves,” in <i>The Sneetches and Other Stories</i> . New York: Random House, 1961. (Arrays)		P
Stein, Lynn Andrea. Interactive Programming in Java. 2003. www.cs101.org/ipij/		W
Stock, Adam. The JavaScript Source: Games: Towers of Hanoi. 1999. http://javascript.internet.com/games/hanoi.html Animation.	Y	W
Sun Microsystems, Inc. Java 2 Platform, Standard Edition, v. 1.5.0 API Specification. 2004. http://java.sun.com/j2se/1.5.0/docs/api/index.html	Y	W
Sun Microsystems, Inc. Trail: Learning the Java Language. 2005. http://java.sun.com/docs/books/tutorial/java	Y	W
Touretzky, David S. “Martin and the Dragon,” in <i>Common Lisp: A Gentle Introduction to Symbolic Computation</i> , 232-33. Redwood City, Calif.: Benjamin/Cummings Publishing Company, 1990. www-2.cs.cmu.edu/~dst/LispBook/		W
Wikipedia. www.wikipedia.org A free online encyclopedia.		W
Winston, Patrick Henry, and Sundar Narasimhan. <i>On to Java</i> . 3rd ed. Boston: Addison-Wesley, 2001.	Y	P/W
Wittry, Dave. AP Computer Science A & AB. Taipei American School, Taiwan. www.apcomputerscience.com	Y	W

Videos/DVDs

For a change of pace, or when your substitute can't teach computer science, here are some films that have been recommended by subscribers to the AP Computer Science Electronic Discussion Group.

Title/Reference	Source	Review?
<i>AntiTrust</i> , VHS/DVD, directed by Peter Howitt (2001). Los Angeles: MGM/UA, 2004. A story of some computer guys who go their separate ways after college. One stays with the open source movement, and the other goes corporate. Romantic interest and intrigue.	*	
<i>Bill Gates: Sultan of Software (DVD Library: BIOGRAPHY)</i> , DVD. A&E Home Video, 2000.	A&E	
<i>Breaking the Code</i> , VHS, directed by Herbert Wise. Anchor Bay Entertainment, 1997. Alan Turing broke the German Enigma Code during World War II, but as a homosexual, he also broke a social code of his time, with disturbing consequences.	*	
<i>Code Rush: A Year in the Life of a Silicon Valley Supernova</i> , VHS. PBS Home Video, 2000. Netscape engineers (one is a woman) rush to release <i>Mozilla</i> (whose source code is to be made public).	*	
<i>Connections: Connections 1, 2 & 3</i> , VHS/ DVD. BBC and Time Life, 2001. This series demonstrates connections between many seemingly unrelated events that ultimately led to modern inventions and discoveries. www.ambrosevideo.com	Ambrose Video	
<i>Decoding Nazi Secrets</i> , VHS. WGBH Boston Video, 1999.	*	
<i>Desk Set</i> , VHS/DVD, directed by Walter Lang (1957). 20th Century Fox, 2004. Katherine Hepburn plays the head of a research department at a TV network. Spencer Tracy is an efficiency expert from IBM planning to replace her and her department (all women) with a new computer.	*	
<i>Frontline: Cyber War!</i> , DVD. PBS Video, 2003. Virus attacks.	*	
<i>Investigative Reports: Caught in the Net</i> , VHS, directed by Lisa Enos. A&E Home Video, 1991. Security and privacy scams: some personal accounts.	A&E	
<i>Investigative Reports: Dot.Com Roller Coaster</i> , DVD. A&E Home Video, 2001.	A&E	
<i>Investigative Reports: e-terror</i> , DVD. A&E Home Video, 2000.	*	
<i>Modern Marvels: The Creation of the Computer</i> , VHS/DVD. History Channel, 2000. Computer history.	*	
<i>Modern Marvels: The Internet: Behind the Web</i> , VHS. A&E Home Video, 2000. This history of the Internet, beginning with ARPA net, includes interviews with many of the original players.	*	
<i>Modern Marvels: Video Games: Behind the Fun</i> , VHS/DVD. A&E Home Video, 2001.	*	
<i>Nerds 2.0.1: A Brief History of the Internet</i> , VHS. PBS Home Video, 1998.	*	
<i>Pirates of Silicon Valley</i> , VHS, directed by Martyn Burke (1999). Turner Home Video, 2000. Steve Jobs vs. Bill Gates: the early years.	*	Y
<i>Revolution OS</i> , VHS & DVD. Wonderview Productions, 2001. www.revolution-os.com/ GNU/Linux and the Open Source movement.	*	
<i>Scientific American Frontiers VII: Robots Alive!, Ep 5 of 5</i> , VHS. PBS Home Video.	PBS	
<i>Scientific American Frontiers X: Natural Born Robots, Ep 2 of 5</i> , VHS. PBS Home Video. Includes some follow-up on <i>Robots Alive!</i>	PBS	
<i>Scientific American Frontiers XIII: The Intimate Machine</i> , VHS. PBS Home Video.	PBS	

Title/Reference	Source	Review?
<i>Sorting Out Sorting</i> , VHS. University of Toronto, 1981. www.utoronto.ca/ic/media/vidcol/science.html Animated comparison of nine sorting algorithms.	University of Toronto	Y
<i>Stand and Deliver</i> , VHS/DVD, directed by Ramón Menéndez (1988). Warner Studios, 2004. The story of Jaime Escalante's AP Calculus class.	*	
<i>Star Trek: The Original Series, Episode 53: The Ultimate Computer</i> , VHS, directed by James Goldstone and Murray Golden (1966). Paramount Studio, 1991. A new supercomputer is designed to run a starship almost single-handedly, but it locks the human crew out of ship operations.	*	
<i>Startup.com</i> , DVD, directed by Jehane Noujaim and Chris Hegedus (2001). Artisan Entertainment, 2002. A documentary about two friends and the story of their startup from "dot-com to dot-bomb."	*	
<i>To Dream Tomorrow</i> , VHS, directed and produced by John Füegi and Jo Francis. Flare Productions, 2003. www.mith.umd.edu/flare/lovelace/ The life of Ada Byron Lovelace, one of the few women in the published history of computing, who collaborated with Charles Babbage on his early computational devices.	Flare	Y
<i>Triumph of the Nerds: The Rise of Accidental Empires</i> , VHS. RM Associates, 1996. www.ambrosevideo.com/displayitem.cfm?vid=69 A series of three videos documents the PC revolution from the mid-1970s to the rise of GUIs in the 1990s and gives a peek into the culture of Silicon Valley. Includes interviews with Steve Jobs, Steve Wozniak, Bill Gates, and others, as well as archival footage. PBS has a companion Web site for the series: www.pbs.org/nerds/	Ambrose Video	Y
<i>Truth About Science Fiction</i> , VHS. A&E Home Video, 2000. A look at how some early works of science fiction compare with today's reality.	*	
<i>20th Century with Mike Wallace: Criminals in Cyberspace</i> , VHS/DVD. History Channel, 1999. A penetrating look at "cyber crime."	A&E	

Key to Sources:

*	Available at online book suppliers.
A&E	Arts & Entertainment (A&E) store. http://store.aetv.com/html/home/index.jhtml
PBS	Public Broadcasting System. www.pbs.org
Others	Available from publisher's Web site, as specified

Contests

Programming contests provide great opportunities for our best students to be challenged, work as members of a team, and be recognized for their achievements. Several organizations conduct programming contests for high school students. Formats vary widely; some contests involve traveling to a specific site, while others are conducted online or require that entries be submitted electronically. Some contests include teamwork during the competition; a team is often assigned to a single computer, so students must plan their work accordingly. Other contests require students to enter as individuals, or a team score is based on the sum of individual scores.

Chapter 5

Contest notices are often posted to the AP Computer Science EDG, and AP Central includes information about several contests in the Teachers' Resources section.

Professional Organizations and Conferences

Members of professional computer science educators' organizations have access to publications, discounted registration fees at conferences, e-mail forums, and other curriculum-related resources.

The Special Interest Group on Computer Science Education (SIGCSE) is part of the Association for Computing Machinery (ACM). A statement on its Web site says that SIGCSE “provides a forum for problems common among educators working to develop, implement, and/or evaluate computing programs, curricula, and courses, as well as syllabi, laboratories, and other elements of teaching and pedagogy.” (www.sigcse.org)

Each year, SIGCSE hosts an annual symposium. Keynote speakers, concurrent smaller sessions, vendor exhibits, and poster contests are on the agenda; topics range from philosophical discussions of curriculum to “Nifty Assignments.” However, informal contacts with colleagues are at least as valuable as any scheduled event. While primarily geared towards college-level faculty, SIGCSE has quite a few events for instructors of introductory courses whose content matches that of AP Computer Science courses. In fact, some events deal specifically with AP CS courses, and participants usually include several college instructors.

“Many high school computer science teachers are the only CS teacher in their school and often in their district. To be in a place where over 1,000 CS teachers gathered to share pedagogy, best practices, hints, tips, and questions truly refreshed my spirit and reminded me that I am not alone in the quest to educate the next generation of computer science professionals ... Experienced faculty were more than willing to share insights, and newer teachers were eager to share or question any issues that might have arisen out of regular conversation.”

—Leigh Ann Sudol, AP CS teacher, Fox Lane High School,
Bedford, New York,

“A First-Timer’s Experience at SIGCSE”

[AP Central > Course Home Page > Features and News Stories Articles]

The Computer Science Teachers Association (CSTA), also associated with the ACM, is a relatively new membership organization that describes its focus as supporting and promoting “the teaching of computer science and other computing disciplines at the K–12 level by providing opportunities for teachers and students to better understand the computing disciplines and to more successfully prepare themselves to teach and to learn.” CSTA sponsors one-day symposia that focus on K–12 computer science. (<http://csta.acm.org>)



AP Central®
apcentral.collegeboard.com

THE OFFICIAL ONLINE HOME FOR THE AP® PROGRAM AND PRE-AP®

The College Board created AP Central to provide the most current information about the AP® Program and Pre-AP®, as well as a wide variety of teaching resources and tools. Our goal is to support and connect AP teachers and other professionals involved with or interested in AP or Pre-AP.

REGISTRATION AND LOGIN

To access all the resources, content, and tools available on AP Central, complete the **free** registration. Choose "Save Login Information as a Cookie" to avoid logging in each time you visit.

MEMBERS HOME PAGE

The Members Home Page is a showcase for our feature stories and discipline-related articles. Here we also offer the latest AP Program updates, news stories, and convenient links.

In the "Also" section of the Members Home Page you will find short cuts to useful content areas such as "Using This Site" for details about content and tools and the "AP Document Library" for important AP Program information.

PROGRAM CONTENT: AP COURSES & EXAMS, PRE-AP, AND PROFESSIONAL DEVELOPMENT

Program content is organized into sections in the navigation bar on the left side of the page. When you pass your cursor over the content buttons on the left navigation bar, a window will appear with a listing of the topics available in that section. To access a topic of your choice, simply click its name.

THE PROGRAM

- **Overview of the AP Program** – Mission, history of the Program, and initiatives (Access & Equity, AP International, and AP Scholars)
- **Starting an AP Program** – Information for teachers and Coordinators
- **AP Research & Data** – Participation statistics, performance data, comparability studies, summary reports

THE COURSES

- **Course Home Pages** – Links to official information and teaching resources for every AP course
- **Course Descriptions** – The official *AP Course Descriptions*
- **Sample Syllabi** – Examples of syllabi to help plan your course

THE EXAMS

- **Exam Questions** – Multiple-choice and free-response questions, scoring guidelines, student samples, scoring commentary, and the Chief Reader's Student Performance Q&A
- **Exam Tips** – Teaching suggestions for preparing students for the AP Exams
- **For Coordinators** – Everything Coordinators need to know about exam administration

PROFESSIONAL DEVELOPMENT

- **Institutes & Workshops** – Overview of the College Board's workshops and summer institutes
- **Online Events** – Live professional development events you can participate in via the Internet
- **Professional Opportunities** – Applications to be a consultant and/or an AP Exam Reader

PRE-AP®

- **Workshops** – Detailed descriptions of all Pre-AP workshops
- **Teachers' Corner** – Articles and teaching tips for middle and high school teachers

HIGHER EDUCATION

- **Setting Credit and Placement Policies** – Information on setting AP policies for your college or university
- **Course and Exam Development** – How AP courses and exams are developed

SPECIAL TOOLS: EDGs, RESOURCE REVIEWS, EVENTS SEARCH, AND MORE

Interactive tools are located on the horizontal bar at the top of every page. To use one of the tools, simply click one of the tabs described below.

AP COMMUNITY

Electronic Discussion Groups – Moderated discipline-specific online forums for the exchange of ideas, insights, and practices.

Community Contacts – Contact AP community members directly via email. Search for members by state, professional role, experience level, and course. Member email addresses will not be shared, until and unless the member chooses to respond.

TEACHERS' RESOURCES

Teachers' Resources – Evaluations of reference books, textbooks, documents, Web sites, videos, software, and more. Reviews are currently available for the following AP course areas:

Art History, Biology, Calculus, Chemistry, Chinese, Computer Science, Economics, English, Environmental Science, European History, French, German, Government & Politics, Human Geography, Japanese, Italian, Latin, Music Theory, Physics, Psychology, Spanish, Statistics, Studio Art, U.S. History, and World History.

INSTITUTES & WORKSHOPS

Professional Development Events – Search for workshops, meetings, and other professional development opportunities by discipline, date, and location.

FAQS

Frequently Asked Questions and Answers about the AP Program or a specific course.

CONTACT AP

Questions, Comments, and Submissions – If you would like to ask a question or submit an article or teaching idea, please use the "Contact AP" button.

PERSONALIZATION: MY AP CENTRAL AND EMAIL NEWSLETTERS

MY AP CENTRAL

During registration you can select up to five courses or subject areas that are most important to you. To set or change your personalization selections after registration, choose **Personal Profile** on the AP Central home page. Choose "My AP Central" to go directly to the Course Home Page, Course Description, Exam Questions, Exam Tips, and Sample Syllabi sections for your courses.

The AP Central staff compiles and sends email newsletters for each AP course on a regular basis. When you personalize for a course in your profile you are automatically subscribed to the email newsletter for that course. To unsubscribe, select the "Do not send" option on the Personal Profile page.

PRINTING

Many pages on AP Central are available in a printer-friendly version. To print, choose  on the top right of the page, then choose the "Click to Print Page" button.

USING PDF AND AUDIO FILES

Many College Board documents are available in Adobe® PDF format. Sound files are offered in two formats: RealAudio and MP3. Information about using and troubleshooting PDF and audio files can be found on the pages where these types of files appear.

COMMENTS ABOUT AP CENTRAL

To submit comments and suggestions to AP Central, use the "Contact AP" button located at the top of every page.

Professional Development

In the following section, the College Board outlines its professional development opportunities in support of AP educators.

The teachers, administrators, and AP Coordinators involved in the AP Program compose a dedicated, engaged, vibrant community of educational professionals. Welcome!

We invite you to become an active participant in the community. The College Board offers a variety of professional development opportunities designed to educate, support, and invigorate both new and experienced AP teachers and educational professionals. These year-round offerings range from half-day workshops to intensive weeklong summer institutes, from the AP Annual Conference to AP Central, and from participation in an AP Reading to Development Committee membership.

Workshops and Summer Institutes

At the heart of the College Board's professional development offerings are workshops and summer institutes. Participating in an AP workshop is generally one of the first steps to becoming a successful AP teacher. Workshops range in length from half-day to weeklong events and are focused on all 37 AP courses and a range of supplemental topics. Workshop consultants are innovative, successful, and experienced AP teachers; teachers trained in developmental skills and strategies; college faculty members; and other qualified educational professionals who have been trained and endorsed by the College Board. For new and experienced teachers, these course-specific training opportunities encompass all aspects of AP course content, organization, evaluation, and methodology. For administrators, counselors, and AP Coordinators, workshops address critical issues faced in introducing, developing, supporting, and expanding AP programs in secondary schools. They also serve as a forum for exchanging ideas about AP.

While the AP Program does not have a set of formal requirements that teachers must satisfy prior to teaching an AP course, the College Board suggests that AP teachers have considerable experience and an advanced degree in the discipline before undertaking an AP course.

AP Summer Institutes provide teachers with in-depth training in AP courses and teaching strategies. Participants engage in at least 30 hours of training led by College Board-endorsed consultants and receive printed materials, including excerpts from AP Course Descriptions, AP Exam information, and other course-specific teaching resources. Many locations offer guest speakers, field trips, and other hands-on activities. Each institute is managed individually by staff at the sponsoring institution under the guidelines provided by the College Board.

Participants in College Board professional development workshops and summer institutes are eligible for continuing education units (CEUs). The College Board is authorized by the International Association for Continuing Education and Training (IACET) to offer CEUs. IACET is an internationally recognized organization that provides standards and authorization for continuing education and training.

Workshop and institute offerings for the AP Computer Science teacher (or potential teacher) range from introductory to topic-specific events and include offerings tailored to teachers in the middle and early high school years. To learn more about scheduled workshops and summer institutes near you, visit the Institutes & Workshops area on AP Central: apcentral.collegeboard.com/events.

Online Events

The College Board offers a wide variety of online events, which are presented by College Board-endorsed consultants and recognized subject-matter experts to participants via a Web-based, real-time interface. Online events range from one hour to several days and are interactive, allowing for exchanges between the presenter and participants and between participants. Like face-to-face workshops, online events vary in focus from introductory themes to specific topics, and many offer CEUs for participants. For a complete list of upcoming and archived online events, visit apcentral.collegeboard.com/onlineevents.

Archives of many past online events are also available for free or for a small fee. Archived events can be viewed on your computer at your convenience.

AP Central

AP Central is the College Board's online home for AP professionals. The site offers a wealth of resources, including Course Descriptions, sample syllabi, exam questions, a vast database of teaching resource reviews, lesson plans, course-specific feature articles, and much more. Bookmark the information on AP Central about AP Computer Science A and AB: apcentral.collegeboard.com/compscia and apcentral.collegeboard.com/compsciab

AP Program information is also available on the site, including exam calendars, fee and fee reduction policies, student performance data, participation forms, research reports, college and university AP grade acceptance policies, and more.

AP professionals are encouraged to contribute to the resources on AP Central by submitting articles or lesson plans for publication and by adding comments to Teacher's Resources reviews.

Electronic Discussion Groups

The AP electronic discussion groups (EDGs) were created to provide a moderated forum for the exchange of ideas, insights, and practices among AP teachers, AP Coordinators, consultants, AP Exam Readers, administrators, and college faculty. EDGs are Web-based threaded discussion groups focused on specific AP courses or roles, giving participants the ability to post and respond to questions online to be viewed by other members of the EDG. To join an EDG, visit apcentral.collegeboard.com/community/edg.

AP Annual Conference

The AP Annual Conference (APAC) is a gathering of the AP community, including teachers, secondary school administrators, and college faculty. The APAC is the only national conference that focuses on providing complete strategies for middle and high school teachers and administrators involved in the AP Program. The 2007 conference will be held July 11 to 15 in Las Vegas, Nevada. Conference events include presentations by each course's Development Committee, course- and topic-specific sessions, guest speakers, and pre- and postconference workshops for new and experienced teachers. To learn more about this year's event, please visit www.collegeboard.com/apac.

AP professionals are encouraged to lead workshops and presentations at the conference. Proposals are due in the fall of each year prior to the event (visit AP Central for specific deadlines and requirements).

Professional Opportunities

College Board Consultants and Contributors

Experienced AP teachers and educational professionals share their techniques, best practices, materials, and expertise with other educators by serving as College Board consultants and contributors. They may lead workshops and summer institutes, sharing their proven techniques and best practices with new and experienced AP teachers, AP Coordinators, and administrators. They may also contribute to AP course and exam development (writing exam questions or serving on a Development Committee) or evaluate AP Exams at the annual AP Reading. Consultants and contributors may be teachers, postsecondary faculty, counselors, administrators, and retired educators. They receive an honorarium for their work and are reimbursed for expenses.

To learn more about becoming a workshop consultant, visit apcentral.collegeboard.com/consultant.

AP Exam Readers

High school and college faculty members from around the world gather in the United States each June to evaluate and score the free-response sections of the AP Exams at the annual AP Reading. AP Exam Readers are led by a Chief Reader, a college professor who has the responsibility of ensuring that students receive grades that accurately reflect college-level achievement. Readers describe the experience as providing unparalleled insight into the exam evaluation process and as an opportunity for intensive collegial exchange between high school and college faculty. (More than 8,500 Readers participated in the 2006 Reading.) High school Readers receive certificates awarding professional development hours and CEUs for their participation in the AP Reading. To apply to become an AP Reader, go to apcentral.collegeboard.com/readers.

Development Committee Members

The dedicated members of each course's Development Committee play a critical role in the preparation of the Course Description and exam. They represent a diverse spectrum of knowledge and points of view in their fields and, as a group, are the authority when it comes to making subject-matter decisions in the exam-construction process. The AP Development Committees represent a unique collaboration between high school and college educators.

AP Grants

The College Board offers a suite of competitive grants that provide financial and technical assistance to schools and teachers interested in expanding access to AP. The suite consists of three grant programs: College Board AP Fellows, College Board Pre-AP Fellows, and the AP Start-Up Grant, totaling over \$600,000 in annual support for professional development and classroom resources. The programs provide stipends for teachers and schools that want to start an AP program or expand their current program. Schools and teachers that serve minority and/or low income students who have been traditionally underrepresented in AP courses are given preference. To learn more, visit apcentral.collegeboard.com/apgrants.

Our Commitment to Professional Development

The College Board is committed to supporting and educating AP teachers, AP Coordinators, and

administrators. We encourage you to attend professional development events and workshops to expand your knowledge of and familiarity with the AP course(s) you teach or that your school offers, and then to share that knowledge with other members of the AP community. In addition, we recommend that you join professional associations, attend meetings, and read journals to help support your involvement in the community of educational professionals in your discipline. By working with other educational professionals, you will strengthen that community and increase the variety of teaching resources you use.

Your work in the classroom and your contributions to professional development help the AP Program continue to grow, providing students worldwide with the opportunity to engage in college-level learning while still in high school.

* The AP Computer Science program will introduce a new case study, GridWorld, beginning in the 2007–2008 school year. Look for information on the new case study on the Computer Science Home Pages on AP Central.

Appendix A

GridWorld Case Study

Teaching your AP Computer Science class with the GridWorld Case Study is both a requirement of the course and a rich learning opportunity.

A case study includes the statement of a problem, a set of classes that solve the problem, and a written description of one expert's path from problem statement to solution program(s). Developed by Cay Horstmann of San Jose State University, the GridWorld Case Study provides a graphical environment where visual objects inhabit and interact in a two-dimensional grid. In this case study, "actor" objects are designed and added to a grid, and their behavior is defined according to specifications. The narrative, written by Chris Nevison and Barbara Cloud Wells of Colgate University, guides students through observing and experimenting with the case study, making simple modifications, and defining new kinds of actors, some of which interact with each other. AB-level students will also examine and modify the underlying grid structure.

The case study is a vehicle for presenting many of the topics of the AP Computer Science course. It provides examples of good style, programming language constructs, fundamental data structures, algorithms, and applications. (In fact, one of my favorite exam tips is to use the case study as a syntax and structure reference; students are given a printed copy of the case-study code to use during the AP Exam.)

In this appendix, you'll read how one teacher uses the case study throughout her AP Computer Science course. We have also provided a list of related resources.

Incorporating GridWorld Throughout Your Course

In Chapter 3, Jill Kaminski recommends incorporating the case study throughout the year. Here is how she accomplishes this in her classes:

When I first began teaching AP Computer Science, I planned my course around a list of the AP Java Subset and Topic Outline: if statements, loops, methods, classes, searching, sorting, case study, etc. I found that using this approach, my students' brains resembled shift registers. They could store content during the current unit of study. But they seemed to "shift" it out as the next unit began so that they could store the new information. By May, my average student had, shall we say, underwhelming retention. My less-than-average students fared even worse. And this isn't the kind of test you can cram for.

Then, I began to follow the sage advice of former AP Chief Reader Chris Nevison: "Don't teach the case study! Use the case study to teach computer science." This advice also appears in the prior case study's teacher's manual: "The case study and the accompanying teacher's manual were designed in such a way that you can use these materials throughout the course. You may, in fact, wish to teach many computer science concepts from the AP Computer Science curriculum through the case study itself." What a concept! I revised my course so that the case study was no longer a unit to be taught, but a tool for teaching a variety of topics.

The good news is: it worked! Once my students learned how the case study works, they were able to

Appendix A

learn and apply new concepts within that framework. By May, they remembered the concepts as well as the case study. Scores improved! The people rejoiced!

And the best news of all: I think that GridWorld has even more potential to be used throughout a course than the prior case studies (particularly in the A course). My students appreciate that this is more of a “real” program, and less like the programs we tend to write in beginning computer science classes (e.g., Student or BankAccount classes). It’s well-designed, it’s particularly great for teaching inheritance and data structures, and the possibilities for its enhancement by students are really exciting.

I usually begin my yearlong AP Computer Science-A classes by reviewing the basic Java topics covered in the semester pre-AP course. Student activities involve reading, taking notes, doing free response practice assignments, doing multiple-choice tests, and writing programs. I try to save as much time as possible for the programming, because they enjoy learning by doing, and I find that it’s effective. I usually prefer two or so short multiple-choice tests during a unit to one long test at the end of a unit. My units are normally two to three weeks in length.

I typically begin case study work by November. We go through each part of the narrative, and at the conclusion of each part, we reinforce the concepts with extra labs. I present new concepts within the GridWorld context, and much of the subset is covered this way. And best of all: it’s fun!

Getting Started with GridWorld

You’ll have to roll up your sleeves at some point and do some GridWorld programming. You’re definitely busy, and probably putting it off, but when you do begin, I think you’ll enjoy it.

If possible, grow to love GridWorld before you present it to your students. They won’t love it any more than you do. If this is not possible, then act like you love it. They won’t love it any more than you can act like you do. Do the exercises and labs in the Student Manual, and take notes as you think of ways that you can enhance them.

Part 1:

Introduce the case study in a memorable way. As students walk in your room on Day 1 of GridWorld, let them:

- See GridWorld running on a projector.
- Hear some kind of bug music. Here are a few ideas that are available on iTunes:
 - o “There Ain’t No Bugs On Me”
 - o “Bugs” by Mr. Heath
 - o “Bugs” by Rosenshontz
 - o “Bugs” by Renee Austen
 - o “Bugs!” by Funky Mama
 - o “The Bug” (aka “Sometimes You’re the Windshield”) by Mary Chapin Carpenter, and also by Dire Straits
 - o “The Time of Your Life” by Randy Newman (from *A Bug’s Life*—the whole soundtrack is terrific)
 - o “Flight of the Bumblebees”

- o “Antmusic” or “Ant Invasion” by Adam and the Ants
- See you perhaps dressed for the occasion in a festive bug-related t-shirt or tie. Or perhaps, wear plaid, to represent the grid.

You could integrate bug and/or flower décor in your room. Jazz up the computer lab with a bug-related movie poster: *A Bug's Life*, *The Ant Bully* or *Antz* for fun, or *Them!*, *Eight-Legged Freaks*, *Arachnophobia*, or *The Fly* for horror movie fans. You can find “it” on eBay! Consider an ant farm, or a class pet hermit crab. Perhaps your Science department has some creatures that they can loan you. If these ideas don't match your personality or style, then think about other ways to present GridWorld in a positive light.

Explain to your students what a case study is. They probably don't know. Explain to them why the College Board wants them to analyze and modify a complex program.

Explain to them that in the “real world,” they'll likely need to modify code more often than write it from scratch. Explain to them that they'll be fired for modifying a class that already works and didn't need to be changed!

Click-ability is a welcome addition! Have students experiment with GridWorld using the table at the end of Part 1, so that they get accustomed to the user interface. This will pay great dividends throughout the course. Take advantage of the BlueJ-like drop-down menus to enforce the concepts of state and behavior of objects.

The case study Student Manual is well-written and engaging. Take advantage of this teaching tool! It's a very good textbook for you during these weeks. Throughout your work in the case study, talk through the “Do You Know?” Sets with the whole class, and have students do all of the exercises. These questions and exercises are well designed and valuable.

Part 2:

After students are familiar with the basic behaviors of the actors, incorporate role-play activities in which students become bugs, flowers, and rocks. Nonacting students can give instructions to the actors. You can use a Twister mat as a fun way to represent the grid, or make a grid on the floor with duct tape.

I'm so grateful to Narrative authors Chris Nevison and Barbara Cloud Wells for presenting inheritance early! Inheritance allows infinite creativity in the case study.

Don't move on to Part 3 without letting students spend some time extending the Bug and/or BoxBug classes in various ways. Make sure that students understand the good news about inheritance: you don't have to start from scratch! Most Bug behaviors do not have to be rewritten.

As inheritance is explored and new methods are added to subclasses, use the drop-down menus to illustrate that an object of a subclass has its own unique methods, and also the methods in the parent class. A class's methods are displayed in a distinct section of the menu. This visual cue is a great reminder that an object has methods from its own class, and also methods from its parent classes.

In Part 2, show students that they can create any kind of actor that they'd like. Show them how easy it is to incorporate their own graphics. If you're a fan of the Marine Biology Simulation (the previous case study), run that code for students, and have them implement basic MBS Fish movement in GridWorld. Your favorite MBS labs do not have to collect dust after the 2007 AP Exam! Or have students write GridWorld programs using other characters they may know and love, like Karel J Robot, Sonic the Hedgehog, or Pac-Man. When students realize that GridWorld can be modified to implement a wide variety of very different

Appendix A

projects, they'll be excited about continuing to work with it, and their understanding of program design will be greatly enhanced.

Part 3:

In this important section of the Student Manual, the inner workings of the case study are unwrapped for students. They have likely thought of many creative ideas for GridWorld projects. Now they will learn the information necessary to implement those using solid design practices.

Incorporate some discussion about the GridWorld design into your classes. Read Cay Horstmann's design rationale, available on the College Board Web site, and share some of this with your students. They will appreciate that the GridWorld designer made the decisions he felt were best, but that he wrestled with some pros and cons along the way. It will really help them solidify their understanding before they begin making major changes to the code.

After completing the Jumper project in the Student Manual, let your students design their own labs! They'll likely need some guidance in the process, but they'll have fun while learning valuable lessons about modifying an existing design. You'll be amazed at what they come up with, and they'll buy in to the assignments.

Part 4:

The introduction of the Critter class allows the case study to remain fresh. You and your students will think of even more ideas to implement. The sky's the limit! At this point, I think that your problem will no longer be "What will I do?" but rather "We don't have enough time to do all the fun stuff I'd like to!"

If you do need ideas, GridWorld enhancements are available on the College Board Web site. These creative contributions provide terrific ideas for programming projects, or may inspire you to create your own ideas.

Part 5:

It is so much fun to teach data structures using a case study like this! The pros and cons of the various structures come to life as students repeatedly extend the AbstractGrid class in various ways.

In Exercise 1, have students use Java's LinkedList class first. Then, make a copy of that project, and have them create their own LinkedList class using the ListNode class. None of the client code from their first LinkedList project needs to change after the import of java.util.LinkedList is removed, and when there are problems (most often NullPointerExceptions!), students have to admit that the problem lies in their LinkedList class. Similarly, students can use the case study as the platform for creating a binary search tree to represent the grid.

Download and use Dave Wittry's Generic Data Structures Viewer tool. This terrific application, written by Dave's students, provides students with a visual representation of the data structures that they create. This makes debugging much easier.

Beyond the Narrative:

Continue to use GridWorld as May approaches to reinforce various topics like inheritance, ArrayLists, recursion, and design. In the process, students will solidify their knowledge of the topics and the case study itself.

GridWorld is a flexible framework for implementing many other grid-based game ideas. Consider Tic-Tac-Toe, Pac-Man, Minesweeper, Hunt the Wumpus, role-playing games, and games commonly downloaded to programmable calculators. And have fun! The best of these ideas will become project staples for years to come.

Different Strokes for Different Folks:

All students are created equal. But not all students are able to perform equally in our classes. I usually have a high percentage of APCS students for whom mine is their only AP course. I encourage their enrollment, and I do my best to help them achieve their best on the exam (even if that means I help them earn a 2). They will still be better off in college for the experience of taking an AP course and exam, according to the 2007 Advanced Placement Report to the Nation.

The key is successful differentiation in our classes. Here are a few ideas pertaining to GridWorld in particular:

- Differentiate by various forms of assessments. Multiple-choice quizzes on GridWorld will help prepare students for the AP Exam, will help the grades of students who do better on multiple-choice than free-response or lab work, and will help improve scores for those who struggle with multiple choice by giving them more practice.
- Differentiate by various forms of presenting assignments. Students should not only read about a programming project, but also see it run (on a projector) prior to attempting it. This will help ESL students and those with poor reading skills.
- Differentiate the amount of work within a programming project. Break down the assignment, and allow credit for the subparts completed. For example, if lab involves creating three subclasses that interact with each other, give credit for each subclass implemented. This way, even if a student doesn't complete the entire lab, he can still gain learning and grades along the way. You can also add additional parts to labs for the advanced students.
- Differentiate the number of programming projects. I usually list multiple assignments within a unit, in order from easiest to hardest. The projects at the bottom are not a punishment for my overachievers, but are designed to provide more fun and challenge to the advanced students. They generally want to get there! This keeps them motivated, and gives me time to work with students who need more of my help. When grading, I assess whether students did their best to complete the highest number of labs possible, with excellent quality, in the given amount of time.

Closing Thoughts and Ideas (on GridWorld and AP Computer Science in general):

I've found that my students will work for food! This is an especially effective strategy when I need to give an extended lecture, since they're happy and they often talk less when their mouths are full. From time to time, try one of the following as a treat:

- Look for fruit snacks in bug shapes.
- Get a large container of Chex Mix. The Chex can represent a grid. Or, add fish objects to the grid, and serve Pepperidge Farm Goldfish crackers. Reuse plastic cups to serve the snacks, and have students write their names on the cups.
- Get a Pez dispenser in the shape of some kind of bug, and reward a kid or two each day with a little Pez candy. They'll be much more likely to answer your questions enthusiastically!

Appendix A

- Read them *On Beyond Bugs! All About Insects* by Tish Rabe.
- Play motivating snippets from movies like *A Bug's Life*, *Cars*, *Finding Nemo*, *Rudy*, and *Invincible*. Look for the inspiring scenes in which our hero is discouraged and a friend picks him up, or when he prevails in the end even though the odds were against him. My favorite is Dory's "Just keep swimming" speech in *Finding Nemo*. I tell my students to remember this line in the middle of any difficult test, when they might be ready to give up.
- Change gears if things aren't working in your class. Abandon your plans, and meet the kids' needs. Be flexible. Hold them accountable, but be realistic in your expectations.
- Have a donut party for them on the morning of the exam. Let them anxiously ask you those last-minute questions. Give them your sage advice, like "never leave a free-response question blank." Tell them that you're proud of them. Read them some inspirational quotes, stories, or poems. Tell them to "Just keep swimming." After the exam, tell them that if they want to, they can still be software engineers when they grow up, no matter what their test score was! The experience will have been worthwhile, and when they take those first college programming courses, they'll be ahead of the class.
- Smile. Have fun. You shouldn't be teaching unless you love it ... at least, most of the time!
- Please consider sharing your GridWorld ideas and successes on the AP Computer Science listserv and the CSTA Web Repository.

Resources

Bergin, Joseph et al. 2005. *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Dream Songs Press.

BlueJ. <http://www.bluej.org>.

College Board. 2007. *Advanced Placement Report to the Nation*.

Computer Science Teachers Association. *CSTA Source: A Web Repository of K-12 Computer Science Teaching and Learning Materials*. <http://csta.acm.org/Resources/sub/WebRepository.html>

Disney/Pixar. 1998. *A Bug's Life*. DVD or VHS.

Disney. 2003. *Finding Nemo*. DVD or VHS. Walt Disney Video.

Disney. 2006. *Invincible*. DVD. Walt Disney Home Video.

"Hunt the Wumpus." *Wikipedia*. http://en.wikipedia.org/wiki/Hunt_the_Wumpus.

iTunes. <http://www.apple.com/itunes/>.

Levine, David, and Steven Andrianoff. 2003. *Role Playing In an Object-Oriented World*. St. Bonaventure, New York: St. Bonaventure University, Department of Computer Science. <http://www.cs.sbu.edu/dlevine/RolePlay/roleplay.html>.

"Minesweeper" (computer game). *Wikipedia*. http://en.wikipedia.org/wiki/Minesweeper_%28game%29.

"Pac-Man." *Wikipedia*. <http://en.wikipedia.org/wiki/Pac-Man>.

Rabe, Tish. 1999. *On Beyond Bugs!: All About Insects*. New York: Random House.

“Sonic the Hedgehog.” *Wikipedia*. http://en.wikipedia.org/wiki/Sonic_the_Hedgehog.

Sony. 1993. *Rudy*. DVD or VHS.

“Tic-Tac-Toe.” *Wikipedia*. <http://en.wikipedia.org/wiki/Tic-tac-toe>.

“Twister.” Board game. Hasbro.

Wittry, Dave. “Generic Data Structures Viewer.” *AP Computer Science A & AB*. Troy High School, Fullerton, California. <http://www.apcomputerscience.com>.

Appendix B

Supplemental Documents

This chapter includes copies of student activities and associated source code provided by contributors of sample syllabi in chapter 3.

If you have a PDF version of this appendix, you may edit the information using any program that allows you to modify a PDF file. If you are using Adobe Reader to read the file, you may copy and paste text from this document into another type of document (such as a word processing file).

Contents (by Syllabus Contributor)

Bekki George

- Searching and Sorting (Unit 9):
Searching and Sorting Worksheet
Searching and Sorting Worksheet Key
Lab Assignment: Searching and Sorting Arrays
- Recursion (Unit 13):
Recursion Worksheet 1
Recursion Worksheet 2
Recursion Notes and Worksheet Answers

Michael Lew

- Project I
- Project II: Model-View-Controller Model with Good OOP Design

Renee Ciezki

- *MAXIT*: A two-player game using a two-dimensional array of objects.
- *AARP* (Algorithm Analysis Research Project):
Introduction: overview, with links to resources
Data Structures: a list of required structures and operations
Topics: searching and sorting algorithms
Detailed Requirements
Presentation: rubric for scoring

(Robert) Glen Martin

- Activity 1: FishStuff12 Lab
- Activity 2: ListNode and TreeNode Card Exercises

Mike Scott

- Pair Programming Standards and Requirements
- Mathematical Matrix Class assignment:
Assignment
Matrix.java (to be completed)
MatrixGenerator.java
ConsoleInput.java
- Lexicon assignment:
Assignment
WordLoader.java
FileHandler.java
ILexicon.java

Appendix B

Searching and Sorting Worksheet

List the number of checks for the Binary Search versus the Linear Search. Also list the values for first, last, mid, A[mid], and found for the Binary Search when array A contains the following data:

2 3 5 7 9 12 13 15 21 24 26 28 35 37

1. NumbSearch = 7

Sequential Searches _____

first	last	mid	A[mid]	found?
-------	------	-----	--------	--------

Binary Searches _____

2 3 5 7 9 12 13 15 21 24 26 28 35 37

2. NumbSearch = 16

Sequential Searches _____

first	last	mid	A[mid]	found?
-------	------	-----	--------	--------

Binary Searches _____

Show all work. Put lists in ascending order. List the number of swaps and the order after each pass. List the total number of passes and total number of swaps.

A. Dumb Bubble Sort

1.

UNSORTED LIST
8 6 1 4 3 5 2

SWAPS

Total Passes _____

Total Swaps _____

2.

UNSORTED LIST
7 6 5 6 3 2 1

SWAPS

Total Passes _____

Total Swaps _____

Appendix B

B. Smart Bubble Sort

1.

UNSORTED LIST

SWAPS

3 6 1 4 0 5 2

Total Passes _____

Total Swaps _____

2.

UNSORTED LIST

SWAPS

6 5 4 5 3 2 1

Total Passes _____

Total Swaps _____

C. Selection Sort

1.

UNSORTED LIST

SWAPS

8 6 3 4 7 5 2

Total Passes _____

Total Swaps _____

2.

UNSORTED LIST

SWAPS

9 8 5 2 3 7 1 4

Total Passes _____

Total Swaps _____

Appendix B

D. Insertion Sort

1. UNSORTED LIST
 8 6 3 4 7 5 2

SWAPS

Total Passes _____

Total Swaps _____

2. UNSORTED LIST
 9 8 5 2 3 7 1 4

SWAPS

Total Passes _____

Total Swaps _____

Determine if the arrays listed below have been sorted by the (A) Bubble, (B) Selection, (C) Insertion, or (D) Can't be determined. (Assume no early exit for the Bubble sort.)

_____ 1. $\frac{2 \ 3 \ 5 \ 4 \ 1}{1 \ 3 \ 5 \ 4 \ 2}$
 1 2 5 4 3
 1 2 3 4 5
 1 2 3 4 5

_____ 2. $\frac{6 \ 2 \ 4 \ 8 \ 3}{2 \ 6 \ 4 \ 8 \ 3}$
 2 4 6 8 3
 2 4 6 8 3
 2 3 4 6 8

_____ 3. $\frac{3 \ 7 \ 5 \ 2 \ 1 \ 6}{3 \ 5 \ 2 \ 1 \ 6 \ 7}$
 3 2 1 5 6 7
 2 1 3 5 6 7
 1 2 3 5 6 7
 1 2 3 5 6 7

_____ 4. $\frac{1 \ 0 \ 2 \ 3 \ 4}{0 \ 1 \ 2 \ 3 \ 4}$
 0 1 2 3 4
 0 1 2 3 4
 0 1 2 3 4

_____ 5. $\frac{1 \ 2 \ 3 \ 5 \ 4}{1 \ 2 \ 3 \ 5 \ 4}$
 1 2 3 5 4
 1 2 3 4 5
 1 2 3 4 5

_____ 6. $\frac{1 \ 3 \ 2 \ 4 \ 5}{1 \ 3 \ 2 \ 4 \ 5}$
 1 2 3 4 5
 1 2 3 4 5
 1 2 3 4 5

Appendix B

Searching and Sorting Worksheet (KEY)

List the number of checks for the Binary Search versus the Linear Search. Also list the values for first, last, mid, A[mid], and found for the Binary Search when array A contains the following data:

2 3 5 7 9 12 13 15 21 24 26 28 35 37

1. NumSearch = 7

Sequential Searches _____4_____

first	last	mid	A[mid]	found?
0	13	6	13	no
0	5	2	5	no
3	5	4	9	no
3	3	3	7	yes

Binary Searches _____4_____

2 3 5 7 9 12 13 15 21 24 26 28 35 37

2. NumSearch = 16

Sequential Searches _____14_____

first	last	mid	A[mid]	found?
0	13	6	13	no
7	13	10	26	no
7	9	8	21	no
7	7	7	15	no

Binary Searches _____4_____

Show all work. Put lists in ascending order. List the number of swaps and the order after each pass. List the total number of passes and total number of swaps.

A. Dumb Bubble Sort

1.

	<u>UNSORTED LIST</u>		<u>SWAPS</u>
	8 6 1 4 3 5 2		
	6 1 4 3 5 2 8		6
	1 4 3 5 2 6 8		5
	1 3 4 2 5 6 8		2
	1 3 2 4 5 6 8		1
	1 2 3 4 5 6 8		1
	1 2 3 4 5 6 8		0

Total Passes 6

Total Swaps 15

2.

	<u>UNSORTED LIST</u>		<u>SWAPS</u>
	7 6 5 6 3 2 1		
	6 5 6 3 2 1 7		6
	5 6 3 2 1 6 7		4
	5 3 2 1 6 6 7		3
	3 2 1 5 6 6 7		3
	2 1 3 5 6 6 7		2
	1 2 3 5 6 6 7		1

Total Passes 6

Total Swaps 19

Appendix B

B. Smart Bubble Sort

1.

UNSORTED LIST

3	6	1	4	0	5	2
3	1	4	0	5	2	6
1	3	0	4	2	5	6
1	0	3	2	4	5	6
0	1	2	3	4	5	6

SWAPS

5
3
2
2

Total Passes _____5_____

Total Swaps _____12_____

2.

UNSORTED LIST

6	5	4	5	3	2	1
5	4	5	3	2	1	6
4	5	3	2	1	5	6
4	3	2	1	5	5	6
3	2	1	4	5	5	6
2	1	3	4	5	5	6
1	2	3	4	5	5	6

SWAPS

6
4
3
3
2
1

Total Passes _____6_____

Total Swaps _____19_____

C. Selection Sort

1.

	<u>UNSORTED LIST</u>							<u>SWAPS</u>
	8	6	3	4	7	5	2	
	2	6	3	4	7	5	8	1
	2	3	6	4	7	5	8	1
	2	3	4	6	7	5	8	1
	2	3	4	5	7	6	8	1
	2	3	4	5	6	7	8	1
	2	3	4	5	6	7	8	0

Total Passes _____6_____

Total Swaps _____5_____

2.

	<u>UNSORTED LIST</u>								<u>SWAPS</u>
	9	8	5	2	3	7	1	4	
	1	8	5	2	3	7	9	4	1
	1	2	5	8	3	7	9	4	1
	1	2	3	8	5	7	9	4	1
	1	2	3	4	5	7	9	8	1
	1	2	3	4	5	7	9	8	0
	1	2	3	4	5	7	9	8	0
	1	2	3	4	5	7	8	9	1

Total Passes _____7_____

Total Swaps _____5_____

Appendix B

D. Insertion Sort

1.

<u>UNSORTED LIST</u>							<u>SWAPS (or shifts)</u>
8	6	3	4	7	5	2	
6	8	3	4	7	5	2	1
3	6	8	4	7	5	2	2
3	4	6	8	7	5	2	2
3	4	6	7	8	5	2	1
3	4	5	6	7	8	2	3
2	3	4	5	6	7	8	6

Total Passes _____6_____

Total Swaps _____15_____

2.

<u>UNSORTED LIST</u>								<u>SWAPS</u>
9	8	5	2	3	7	1	4	
8	9	5	2	3	7	1	4	1
5	8	9	2	3	7	1	4	2
2	5	8	9	3	7	1	4	3
2	3	5	8	9	7	1	4	3
2	3	5	7	8	9	1	4	2
1	2	3	5	7	8	9	4	6
1	2	3	4	5	7	8	9	4

Total Passes _____7_____

Total Swaps _____21_____

Determine if the arrays listed below have been sorted by the (A) Bubble, (B) Selection, (C) Insertion, or (D) Can't be determined. (Assume no early exit for the Bubble sort.)

— B — 1. $\frac{2 \ 3 \ 5 \ 4 \ 1}{1 \ 3 \ 5 \ 4 \ 2}$
 $\frac{1 \ 2 \ 5 \ 4 \ 3}{1 \ 2 \ 3 \ 4 \ 5}$
 $\frac{1 \ 2 \ 3 \ 4 \ 5}{1 \ 2 \ 3 \ 4 \ 5}$

— C — 2. $\frac{6 \ 2 \ 4 \ 8 \ 3}{2 \ 6 \ 4 \ 8 \ 3}$
 $\frac{2 \ 4 \ 6 \ 8 \ 3}{2 \ 4 \ 6 \ 8 \ 3}$
 $\frac{2 \ 3 \ 4 \ 6 \ 8}{2 \ 3 \ 4 \ 6 \ 8}$

— A — 3. $\frac{3 \ 7 \ 5 \ 2 \ 1 \ 6}{3 \ 5 \ 2 \ 1 \ 6 \ 7}$
 $\frac{3 \ 2 \ 1 \ 5 \ 6 \ 7}{2 \ 1 \ 3 \ 5 \ 6 \ 7}$
 $\frac{1 \ 2 \ 3 \ 5 \ 6 \ 7}{1 \ 2 \ 3 \ 5 \ 6 \ 7}$

— D — 4. $\frac{1 \ 0 \ 2 \ 3 \ 4}{0 \ 1 \ 2 \ 3 \ 4}$
 $\frac{0 \ 1 \ 2 \ 3 \ 4}{0 \ 1 \ 2 \ 3 \ 4}$
 $\frac{0 \ 1 \ 2 \ 3 \ 4}{0 \ 1 \ 2 \ 3 \ 4}$

— D — 5. $\frac{1 \ 2 \ 3 \ 5 \ 4}{1 \ 2 \ 3 \ 5 \ 4}$
 $\frac{1 \ 2 \ 3 \ 5 \ 4}{1 \ 2 \ 3 \ 4 \ 5}$
 $\frac{1 \ 2 \ 3 \ 4 \ 5}{1 \ 2 \ 3 \ 4 \ 5}$

— B — 6. $\frac{1 \ 3 \ 2 \ 4 \ 5}{1 \ 3 \ 2 \ 4 \ 5}$
 $\frac{1 \ 2 \ 3 \ 4 \ 5}{1 \ 2 \ 3 \ 4 \ 5}$
 $\frac{1 \ 2 \ 3 \ 4 \ 5}{1 \ 2 \ 3 \ 4 \ 5}$

Appendix B

Lab Assignment: Searching and Sorting Arrays

Part 1:

Create a List Utility class that includes in the class the following sorts:

Bubble, Selection, Insertion

It must also include the following searches:

Sequential, Binary

(You can have students write these sorts and searches so that they work with objects, or you can have them demonstrate the idea with a simple data type like numbers. Sometimes it's nice to have students do the simple data types first, then have them change the code to work for objects later.)

Part 2:

Using their list utility class, students need to put data into a list. (This can be done by file or keyboard input, or they can just generate random data within the program.)

Students will be required to demonstrate that all their sorts and searches work. An example output follows (assuming they used a random number generator to get their data):

Original List contains:

12 6 2 8 34

****Demonstrating Linear Search with an UNSORTED list*****

Search for 6 using a linear search:

Comparing with 12

Comparing with 6

Value found at index 1.

*****Demonstrating Binary Search with an UNSORTED list*****

Search for 6 using a binary search with an UNSORTED list

Comparing with 2, search item is greater

Comparing with 8, should be search item

Search item not found!

*****Sorting List*****

Have students store the original list and make new lists after they sorted the original list.

~~~~~Sorting with a Bubble Sort

*\*\*\* Here you can have them print out every comparison or pass.*

~~~~~Sorting with a Selection Sort

**** Here you can have them print out every comparison or pass.*

~~~~~Sorting with an Insertion Sort

*\*\*\* Here you can have them print out every comparison or pass.*

\*\*\*\*\*Searching after the Sorting\*\*\*\*\*

Sorted List contains:

2 6 8 12 34

\*\*\*\*\*Demonstrating Linear Search on SORTED list\*\*\*\*\*

Search for 6 using a linear search:

Comparing with 12

Comparing with 6

Value found at index 1.

\*\*\*\*\*Demonstrating Binary Search with a SORTED list\*\*\*\*\*

Search for 6 using a binary search with a SORTED list

Comparing with 8, search item is smaller

Comparing with 2, item is greater

Comparing with 6, item is found at index [1]

## Appendix B

### CS AP Recursion Worksheet 1

1. Find  $f(6)$ :

$$f(x) = \begin{cases} f(x-1) + x & x > 1 \\ x-2 & x = 1 \end{cases}$$

2. Find  $f(6)$ :

$$f(x) = \begin{cases} f(f(x-2)) + 1 & x > 1 \\ 2 & x = 1 \\ 1 & x = 0 \end{cases}$$

3. Find  $f(-4)$ :

$$f(x) = \begin{cases} 2(f(x+2)) - f(x+1) + 1 & x < 0 \\ 1 & x = 0 \\ 0 & x > 0 \end{cases}$$

4. Find  $f(12,6)$ :

$$f(x, y) = \begin{cases} f(x-y, y+1) + 2 & x > y \\ x + y & \text{otherwise} \end{cases}$$

5. Find  $f(6,5)$ :

$$f(x, y) = \begin{cases} 2 + f(x-3, y-1) & x > 0, x > y \\ 1 + f(y, x) & x > 0, y \geq x \\ 0 & x \leq 0 \end{cases}$$

6. Find  $f(3,5)$ :

$$f(a, n) = \begin{cases} a & n = 1 \\ a * f(a, n-1) & n > 1 \end{cases}$$

**CS AP      Recursion Worksheet 2**

Give the value for ans using the following method:

```
public static int thing1(int x)
{
    if(x==0)
        return 5;
    else
        return x + thing1(x-1);
}
```

\_\_\_\_\_ 1.    ans = thing1(14);

\_\_\_\_\_ 2.    ans = thing1(0);

\_\_\_\_\_ 3.    ans = thing1(5);

\_\_\_\_\_ 4.    ans = thing1(-2);

Give the value for ans using the following method:

```
public static int thing2(int x)
{
    if(x<=0)
        return x;
    else
        return x * thing2(x-1);
}
```

\_\_\_\_\_ 5.    ans = thing2(4);

\_\_\_\_\_ 6.    ans = thing2(0);

\_\_\_\_\_ 7.    ans = thing2(6);

\_\_\_\_\_ 8.    ans = thing2(-3);

## Appendix B

Give the value for ans using the following method:

```
public static double thing3(double x, int n)
{
    if(n==0)
        return 1.0;
    else
        return x * thing3(x, n-1);
}
```

- \_\_\_\_\_ 9. ans = thing3(3.0, 2);
- \_\_\_\_\_ 10. ans = thing3(2.0, 3);
- \_\_\_\_\_ 11. ans = thing3(4.0, 4);
- \_\_\_\_\_ 12. ans = thing3(1.0, 6);

What is wrong with the following and how could it be fixed?

```
13. public static double recur(double x)
    {
        return recur(x/2);
    }
```

```
14. public static int recur(int x)
    {
        if(x<0)
            return 3;
        else
            return 1 + recur(x);
    }
```

**Recursion Notes**

1. Recursion involves the situation in which a subprogram (method) calls itself.
2. For a recursive process to function correctly, it must have a well-defined terminator or base-case or stopping state.
3. The recursive process must have well-defined steps that lead to the stopping state.
4. Recursive processes can be rewritten using iteration (loop).
5. Recursion generally requires more memory than the equivalent repetition (loop).
6. Two main reasons for recursion:
  - b. It may be the most logical way to solve the problem.
  - c. The code may be much shorter.

**We can write recursive methods that are void or that return a value.**

**Recursive return method example:**

```
public static int tester(int n)
{
    if(n == 1)
        return 2;
    else
        return n + tester(n-1);
}
```

`ans = tester(4); //call from main method`

work:

```
tester(4) = 4 + tester(3)
  tester(3) = 3 + tester(2)
    tester(2) = 2 + tester(1)
      tester(1) = 2
ans = 2+2+3+4 = 11
```

## Appendix B

### Recursive void method example:

```
/* Type and run the following program. What is the output? Why do you
 * think it behaves the way it does? How are the calls to these methods
 * different from the ones we used for the return methods?
 */
import java.io.*;

public class recursionEx
{
    public static BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));

    public static void getLetter1(char c) throws Exception
    {
        c = (char)in.read();
        if (c != '*')
            getLetter1(c);
        System.out.println(c);
    }

    public static void getLetter2() throws Exception
    {
        char c;
        c = (char)in.read();
        if (c != '*')
            getLetter1(c);
        System.out.println(c);
    }

    public static void main(String args[]) throws Exception
    {
        char letter = ' ';

        System.out.println("Type the characters HELLO* at the -> ");
        getLetter1(letter);

        System.out.println("Type the characters WOW* at the -> ");
        getLetter2();
    }
}
```

---

---

**Recursion Worksheet #1 answers:**

- 1) 19
- 2) 7
- 3) 9
- 4) 15
- 5) 8
- 6) 243

**Recursion Worksheet #2 answers:**

- 1) 110
- 2) 5
- 3) 20
- 4) infinite
- 5) 0
- 6) 0
- 7) 0
- 8) -3
- 9) 9
- 10) 8
- 11) 256
- 12) 1
- 13) No base case (or exit condition)
- 14) x never changes

## Appendix B

### AP Computer Science Project I

1. This project is worth 510 points
2. The program must include the following 12 items. Each item is worth 25 points.
  1. Wrapper classes – Integer AND Double (with intValue(), doubleValue() methods)
  2. Relational (==, !=, >, >=, <, <=) and Logical Operators (&&, ||, !)—at least 5 total
  3. if-else statements
  4. while/for loops
  5. At least ONE interface and THREE classes (one of which MUST be abstract), not including class Test in “Test.java”—which brings the total to FIVE
  6. Interaction between all four classes/interfaces (i.e., each class must interact with AT LEAST one other class (not including class Test in “Test.java”)
  7. An inheritance hierarchy must be implemented with the student-designed classes (i.e., not the ActionListener interface)
  8. Polymorphism must be implemented with the student-designed classes
  9. Class ArrayList must be used in at least ONE student-designed class and it MUST be traversed AND accessed via an Iterator
  10. Comments explaining logic and operation of program at “key points”
  11. Animation OR Graphical Interface must be implemented
  12. “javadocs” must be created for your project (n.b., each method should be commented according to javadoc specifications)

Subtotal = 300 points

With pencil, place a box around each section of code that satisfies a requirement. Therefore, you should have 11 sections of boxed code that represent the 11 **required** sections of code. A hard copy of the javadocs generated for your project should be placed after the source code printout.

3. A TWO-page single-spaced TYPED description and computer-generated UML class diagrams describing your project is due **AT THE BEGINNING OF CLASS ON** \_\_\_\_\_. It should outline what your project will do (whether it is a game or some other program). At this time I will let you know whether the project is too simple or too complex for a final project. This does not have to be a final design; you may (and probably will) make design changes. The proposal is worth 50 points.

Subtotal = 350 points

4. A SECOND TWO-page single-spaced TYPED description and computer-generated UML class diagrams describing your project is due **AT THE BEGINNING OF CLASS ON** \_\_\_\_\_. This SECOND proposal should include any changes/additions/deletions that you make to your first proposal. The second proposal is worth 50 points.

Subtotal = 400 points

5. Some ideas for projects are:

|                              |                    |
|------------------------------|--------------------|
| Blackjack (or any card game) | Bowling            |
| Hangman                      | Yahtzee            |
| Monopoly                     | Backgammon         |
| Jeopardy!                    | Othello            |
| Wheel of Fortune             | (Chinese) Checkers |
| Basketball games             | Chess              |
| Football games               | Connect Four       |
| Risk                         | Missile Command    |
|                              | Life               |

6. The Final Project is due on your assigned lottery date. **ONE HUNDRED (100) POINTS WILL BE DEDUCTED FOR LATE PROJECTS.**

7. A *PowerPoint* presentation should be given on your lottery date. The presentation will be worth 110 points. The presentation shall have, at minimum, the following slides:

- a. Title
- b. Description of program operation (or how game is played)
- c. Demonstration of Program (Task switch from *PowerPoint* presentation to your program using Alt-Tab)
- d. UML Diagrams for each class
- e. Use of classes/objects in project—elaborate on how classes represent **physical objects** in your program (**be prepared to justify class names, class data member names, class method names ...**)
- f. Description of class interaction
- g. Description of use of an inheritance hierarchy (**be prepared to justify structure**)
- h. Description of use of polymorphism (**include a code snippet that demonstrates polymorphism**)
- i. Special features implemented in program—elaborate on tricks/special things you did
- j. Known bugs in program
- k. Citation of “second-party” code used in program (**be able to explain code**)
- l. Questions? (This is simply a slide that says “Questions?” that keys the audience for any questions they might have.)
- m. Conclusion—Summary of what you thought of writing the program
  - i. Difficulty level
  - ii. “Fun” level
  - iii. Your evaluation of the final product
  - iv. What you learned (be specific)

GRAND TOTAL = 510 points

## Appendix B

8. Source code and disks should be submitted in a thin THREE-RING BINDER (see example in class). The items that should be turned in are as follows:
  - a. Printouts
    - Source Code with complete comments and javadocs (printed in landscape mode)
    - Computer-generated UML Diagrams
    - *PowerPoint* presentation slides (print 6 slides per page)
  - b. Disks—include **TWO** copies of your disk, both of which should include the following:
    - Source Code with complete comments
    - javadocs
    - Computer-generated UML Diagram
    - *PowerPoint* presentation

*Do not include any miscellaneous files from any other class on your disks. If any other files are located on the disk, it will result in a 50-point deduction.*

Any deviation from these guidelines will result in a 100-point deduction.

### Point Distribution

|                                |                                                             |
|--------------------------------|-------------------------------------------------------------|
| Project Proposal I             | 50 points                                                   |
| Project Proposal II            | 50 points                                                   |
| Requirements 1–12              | 300 points (disk and printout are required for full credit) |
| <i>PowerPoint</i> Presentation | 110 points (disk and printout are required for full credit) |
| <b>Total Points</b>            | <b>510 points</b>                                           |

## AP Computer Science Project II

### Model-View-Controller Model with Good OOP Design

As you learned in your first project and the MBS, good object-oriented design includes the proper use of classes, abstract classes, interfaces, inheritance, and polymorphism. Building on these concepts, we will delve more deeply into OOP, focusing on the “Model-View-Controller” or MVC design pattern. In this design pattern, the “model” represents a well-defined representation of the objects in your program (e.g., the Fish and Environment classes in the MBS). The “model” is “controlled” by an external agent (e.g., the Simulation class in the MBS). Finally, the “view” is a visual representation of our “model” and “controller” (e.g., the MBSDisplay class in the MBS). The MVC design pattern allows us to compartmentalize our project into these three distinct components giving us the flexibility to alter, modify, and expand our programs much more easily. For instance, we may have one model of a game but have two or more controllers (i.e., 1-player or 2-player mode), and even have two or more viewers (i.e., displaying the fish in the MBS from a side view for peer predators (sharks/piranhas), top view for human predators (fisherman), or bottom view for the bottom-dwelling predators!).

In addition, the program should adhere to the fundamental properties of a well-designed object-oriented program, i.e., the 5 “Cs” (from Horstmann, *Object-Oriented Design and Patterns*):

1. Cohesion—A class is cohesive if all of its methods are related to a single data abstraction (a car, a chess piece, a fish, etc.)
2. Completeness—A class is complete if it supports all of the operations that are part of the abstraction that the class represents (e.g., accessors, modifiers)
3. Convenience—A class is convenient if it supplies sufficient tools to achieve any necessary and/or common task (“moving” a chess piece, “dying off” a fish ...)
4. Clarity—A class possesses clarity if it is clear to the programmer using the class; using the class should not generate confusion (method names make sense; operation of class and methods are logical)
5. Consistency—A class is consistent if the operations in the class are consistent with each other with respect to parameters, return values, and behavior (methods perform specified actions; no side effects are present)

In this project, your program will implement all of the aspects of good OOP (i.e., classes, abstract classes, interfaces, inheritance, polymorphism, and the 5 Cs) AND the MVC design pattern. The specifics are listed below.

1. The spring project will be worth 470 points. Each item below is worth 25 points.
  - a. **At least ONE interface, ONE abstract class, and THREE concrete classes, not including class Test in “Test.java”—which brings the total to SIX. Classes, class methods, class data members, and interfaces should be properly named.**
  - b. Interaction between all five classes/interfaces (i.e., each class must interact with AT LEAST one other class (not including class Test in “Test.java”). Use of these classes must follow the MVC design pattern.
  - c. An inheritance hierarchy must be implemented with the student-designed classes (i.e., not the ActionListener interface). **Inheritance hierarchy must be justifiable, with an explanation of why other possibilities would OR would not be conceptually correct.**
  - d. Polymorphism must be implemented with the **student-designed** classes (not ActionListener).
  - e. A justifiable implementation of MVC Model.

## Appendix B

- f. A container class (ArrayList, HashMap, TreeSet . . .) must be used in at least ONE student-designed class.
- g. Animation OR Graphical Interface must be implemented.
- h. “javadocs” must be created for your project. (*Note:* each method should be commented according to javadoc specifications.)

Subtotal = 225 points

**Each of the following cardinal sins will result in a deduction of 20 points.**

- i. Side effects (a method’s changing of an object other than its implicit parameter)
- ii. No separation between M, V, and C

**HELPFUL HINTS:**

- i. Stay away from static methods and classes unless you definitely want to use one.
- ii. Avoid spending too much time on graphics and not enough on design.

- 2. A THREE-page single-spaced TYPED description (1 page narrative, 1 page UML, 1 page MVC) is due **AT THE BEGINNING OF CLASS ON** \_\_\_\_\_. It should outline what your project will do (whether it is a game or some other type of program). At this time I will let you know whether the project is too simple or too complex for a final project. This does not have to be a final design; you may (and probably will) make design changes. The proposal is worth 50 points.

Subtotal = 275 points

- 3. A SECOND THREE-page single-spaced TYPED description is due **AT THE BEGINNING OF CLASS ON** \_\_\_\_\_. This SECOND proposal should include any changes/additions/deletions that you make to your first proposal. The second proposal is worth 50 points.

Subtotal = 325 points

- 4. Some ideas for projects are:

|                              |                    |
|------------------------------|--------------------|
| Blackjack (or any card game) | Bowling            |
| Hangman                      | Yahtzee            |
| Monopoly                     | Backgammon         |
| Jeopardy!                    | Othello            |
| Wheel of Fortune             | (Chinese) Checkers |
| Basketball games             | Chess              |
| Football games               | Connect Four       |
|                              | Missile Command    |

- 5. The Final Project is due on your assigned lottery date. **ONE HUNDRED (100) POINTS WILL BE DEDUCTED FOR LATE PROJECTS.**

6. A *PowerPoint* presentation should be given on your lottery date. The presentation will be worth 145 points. The presentation shall have, at minimum, the following slides:
  - a. Title
  - b. Brief explanation of program
  - c. Demonstration of program (Task switch from *PowerPoint* presentation to your program using Alt-Tab)
  - d. Computer-generated MVC Diagram
  - e. Computer-generated UML Diagrams for each class
  - f. Description of class interaction
  - g. Use of classes/objects in project—elaborate on how classes represent **physical objects** in your program (**be prepared to justify class names, class data member names, class method names ...**)
  - h. OOP Principles—5 Cs: Cohesion, Completeness, Convenience, Clarity, Consistency (**be prepared to defend any or all of your classes**)
  - i. Description of use of an inheritance hierarchy (**be prepared to justify structure**)
  - j. Description of use of polymorphism (**include a code snippet that demonstrates polymorphism**)
  - k. Special features implemented in program—elaborate on tricks/special things you did (include a snippet of code that demonstrates tricks/special things)
  - l. Known bugs in program
  - m. Citation of “second-party” code used in program (**be able to explain code**)
  - n. Questions? (This is simply a slide that says “Questions?” that keys the audience for any questions they might have.)
  - o. Conclusion—Summary of what you thought of writing the program
    - i. Difficulty level
    - ii. “Fun” level
    - iii. Your evaluation of the final product
    - iv. What you learned (be specific)

Subtotal = 470 points

## Appendix B

7. Source code and disks should be submitted in a thin THREE-RING BINDER (see example in class). The items that should be turned in are as follows:
  - a. Printouts
    - Source Code with complete comments and javadocs (printed in landscape mode)
    - Computer-generated MVC diagram
    - Computer-generated UML diagrams
    - *PowerPoint* presentation slides (print 4 slides per page)
  - b. Disks—include **TWO** copies of your disk, both of which should include the following:
    - Source Code with comments
    - javadocs
    - Computer-generated MVC diagram
    - Computer-generated UML Diagram
    - *PowerPoint* presentation

*Do not include any miscellaneous files from any other class on your disks. If any other files are located on the disk, it will result in a 50-point deduction.*

Any deviation from these guidelines will result in a 100-point deduction.

### Point Distribution

|                                |                                                             |
|--------------------------------|-------------------------------------------------------------|
| Project Proposal I             | 50 points                                                   |
| Project Proposal II            | 50 points                                                   |
| Requirements a–o               | 145 points (disk and printout are required for full credit) |
| <i>PowerPoint</i> Presentation | 225 points (disk and printout are required for full credit) |
| Total Points                   | <b>470 points</b>                                           |

| <h1 style="text-align: center;">M A X I T</h1> <p style="text-align: center;">A two-player game</p> <hr/> <p>Objective: Score more points than your opponent.</p> <p>Setup: MAXIT is played on a square grid filled with random numbers. This example uses a 5 x 5 board, and the numbers range from -5 to +5.</p> | Allen: 7 |    |    | Robert: 4 |    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|----|----|-----------|----|
|                                                                                                                                                                                                                                                                                                                    | -1       | -3 | 0  | 1         | 1  |
|                                                                                                                                                                                                                                                                                                                    | 0        | -4 | -5 | 2         | 4  |
|                                                                                                                                                                                                                                                                                                                    | x        | x  | 4  | 1         | 4  |
|                                                                                                                                                                                                                                                                                                                    | x        | x  | -1 | 0         | -2 |
|                                                                                                                                                                                                                                                                                                                    | -5       | -5 | 0  | -2        | 0  |

**Play:** Player 1 can only move horizontally, and Player 2 moves vertically. When the game begins, the board is displayed, and the starting square is marked by an X. Player 1 chooses a square—in this case, by clicking on it—and the value in the square is added to Player 1’s score. Player 2 moves next, choosing any square in the column Player 1 had selected. Player 2’s score is adjusted. Play continues as long as a move is possible. When there is no available square for a player to choose, the game ends, and the player with the higher score is the winner.

**Minimum requirements:** A working game, may be text-based. Three classes, as a minimum, are necessary: Board, Square, Player. The Board contains a two-dimensional array of Square objects.

**Possible enhancements:** Create an applet, allow the user to set the size of the grid, allow the user to specify the range of random values, allow multiple games, etc.

# Algorithm Analysis Research Project

## Introduction

Congratulations! You are about to begin a major APCS project. After reviewing nine data structures and researching 10 common search and sort algorithms, you will organize your findings and present them in the following formats:

- in a “written” report,
- in one or more programs using at least two of the algorithms (one search and one sort), and
- in a presentation that summarizes one concept for your classmates, demonstrated by a program.

Once you have your topics, use books, Web sites, etc., to learn about the algorithms and data structures. Write one or more programs demonstrating their use. Prepare for your “presentation” to the class on your assigned algorithm.

The presentation must include an explanation of the algorithm (what it is, when/how to use it, how it works), an analysis of the algorithm’s efficiency (Big-O), and sample code. Your overall grade will be based on the following:

- Program(s)—10 points (must include both a search algorithm and a sort algorithm)
- Presentation—40 points
- Paper—50 points

*Content: clear explanations and examples, professional quality*

*Organization: logical order, easy to follow, references/links*

*Delivery: enthusiasm, confidence—you know your stuff!*

*Extras: audience “participation” and “demeanor” DO matter*

The following Web sites are great sources of information, ideas, and help:

**Complete Collection of Algorithm Animations (CCAA, with lots of links):** [www.cs.hope.edu/~algnim/ccaa/](http://www.cs.hope.edu/~algnim/ccaa/)

**Or go right to their animations:** [www.cs.hope.edu/~algnim/animator/Animator.html](http://www.cs.hope.edu/~algnim/animator/Animator.html)

### Sorting:

[www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html](http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html)

[www.cs.brockport.edu/cs/javasort.html](http://www.cs.brockport.edu/cs/javasort.html)

<http://cs.smith.edu/~Ethiebaut/java/sort/demo.html>

### Trees:

[www.ibr.cs.tu-bs.de/lehre/ss98/audii/applets/BST/BST-Example.html](http://www.ibr.cs.tu-bs.de/lehre/ss98/audii/applets/BST/BST-Example.html)

[www.student.seas.gwu.edu/~idsv/idsv.html](http://www.student.seas.gwu.edu/~idsv/idsv.html)

### Other helpful sites:

[http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/ds\\_ToC.html](http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/ds_ToC.html)

<http://nz.cosc.canterbury.ac.nz/people/mukundan/dsal/appldsal.html>

# Algorithm Analysis Research Project

## Data Structures

For each data structure, you need (1) an explanation, (2) an analogy, and (3) the expected Big-O for three operations. See Report Guidelines.

| <b>Structures<br/>(All are required)</b> | <b>Operations<br/>(Choose 3 for each data structure)</b> |
|------------------------------------------|----------------------------------------------------------|
| ArrayList<br>LinkedList                  | Insert, may include inserting at the start or end . . .  |
| Stack                                    | Delete                                                   |
| Queue                                    | Search                                                   |
| Priority Queue                           | Get                                                      |
| HashSet                                  | Traverse (to count, print, etc.)                         |
| HashMap                                  |                                                          |
| TreeSet                                  | Other operations—you decide                              |
| TreeMap                                  |                                                          |

## Appendix B

# Algorithm Analysis Research Project

## Topics

| Search Algorithms<br>(Choose 3) | Sort Algorithms<br>(Choose 7) |             |                 |
|---------------------------------|-------------------------------|-------------|-----------------|
| Sequential Search               | Bubble Sort                   | Merge Sort* | Selection Sort* |
| Binary Search                   | Heap Sort*                    | Quick Sort* | Shaker Sort     |
| Hashing                         | Insertion Sort*               | Radix Sort  | Shell Sort      |

\*Required algorithms

Use your review books, other textbooks (language doesn't matter for algorithms!), and your favorite search engine.

Post URLs of helpful sites on the Discussion Board at eCampus.

A good starting point is [www.cs.hope.edu/~algaanim/ccaa/](http://www.cs.hope.edu/~algaanim/ccaa/)

Also see: [http://en.wikipedia.org/wiki/Sorting\\_algorithm](http://en.wikipedia.org/wiki/Sorting_algorithm)

# Algorithm Analysis Research Project

## Detailed Requirements

A one-stop source for learning about advanced data structures, searching, sorting, and Big-O notation!

It is likely that you will find conflicting information. Part of your job is to sort through it all and continue (re)searching and thinking until you have a good understanding and can communicate the information clearly.

**Format:** your choice—online or on paper.

### Minimum requirements:

- Information about the following data structures and the efficiency of various operations on them. As a minimum, you should include (*for each data structure*) an explanation of the structure itself, including an analogy\* and the expected Big-O for three operations that you choose. HINT: A good place to start is <http://java.sun.com/j2se/1.4.2/docs/api/>.

1. **ArrayList**
2. **LinkedList**
3. **Priority Queue**
4. **HashSet**
5. **HashMap**
6. **TreeSet**
7. **TreeMap**

\***Analogy example:** An array is like an apartment building. Everyone who lives there has the same street address, but there is a number to designate which “element” is being referenced at any given time.

- Information about the three search algorithms, including name of the algorithm, how it works, and the efficiency (Big-O), etc., **IN YOUR OWN WORDS . . .**
- Information about seven sort algorithms (including the five required ones): Same requirements as for the search algorithms. *To get credit for this part, you must justify your choice of Big-O for average and worst cases.*
- A variety (more than three) of *cited* references.

## Algorithm Analysis Research Project

### Presentation

| A.A.R.P. Presentation                          |                                                                                            |                                                                                             |                                                                                    |                                                                                               |       |
|------------------------------------------------|--------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|-------|
|                                                | 1                                                                                          | 2                                                                                           | 3                                                                                  | 4                                                                                             | Total |
| <b>Organization</b>                            | <i>Audience cannot understand presentation due to lack of structure.</i>                   | <i>Audience has difficulty following presentation because student jumps around.</i>         | <i>Student presents information in logical sequence.</i>                           | <i>Student presents information in logical, interesting sequence.</i>                         |       |
| <b>Algorithm and Code</b>                      | Student does not explain algorithm and/or does not show sample code.                       | Student's explanation and/or code is confusing or wrong.                                    | Student explains algorithm and shows sample code.                                  | Student makes explanation and code work together to enhance audience understanding.           |       |
| <b>Big-O Analysis</b>                          | Student does not cover this.                                                               | Student only tells the Big-O notation.                                                      | Student explains what the Big-O notation means.                                    | Student addresses average and worst cases.                                                    |       |
| <b>Overall Knowledge of Subject</b>            | Student does not have good grasp of information and cannot answer questions about subject. | Student is uncomfortable with information and is able to answer only rudimentary questions. | Student is at ease with expected answers to all questions, but fails to elaborate. | Student demonstrates full knowledge by answering questions with explanations and elaboration. |       |
| <b>Delivery</b>                                | <i>Student makes no eye contact and is difficult to hear or understand.</i>                | <i>Student reads the information and/or is hard to hear or understand.</i>                  | <i>Student's voice is clear and understandable. Student appears confident.</i>     | <i>Student presents with enthusiasm and shows confidence.</i>                                 |       |
| <b>Quality of Presentation</b>                 | Student seems to have put forth little effort to produce a quality presentation.           | Student's materials and/or examples are below expected levels.                              | Student uses appropriate examples and technology.                                  | Student makes good use of technology and keeps the audience involved.                         |       |
| <b>Audience Participation</b>                  | <i>Student tries to undermine a classmate's presentation.</i>                              | <i>Student talks or works on the computer during a classmate's presentation.</i>            | <i>Student pays attention during classmates' presentations.</i>                    | <i>Student is an active participant during classmates' presentations.</i>                     |       |
| <i>Italicized categories are worth double.</i> |                                                                                            |                                                                                             |                                                                                    | <b>Total Points:</b>                                                                          |       |

**Activity 1: FishStuff12 Lab Assignment**

**Note: One of the purposes of this assignment is to give you practice retrieving references to `Locatables` (fish) from the environment. You may make “temporary” Fish references as you create and use fish. However, do not use these fish references later in the program when you need to use your fish again. Get fish from the environment instead.**

1. Copy the starter project to your network folder. The folder is called **FishStuff12**. You will be editing the `FishStuff12.java` file.
2. Create a bounded environment that has `NUM_ROWS` rows and `NUM_COLS` columns.
3. Create a `SimpleMBSDisplay` with a delay of 500 milliseconds and display your empty environment. The following will work:

```
SimpleMBSDisplay display = new SimpleMBSDisplay(env, 500);
```

4. Add a `randomLocation()` method that returns a random location within the environment. This method needs to use `RandNumGenerator`.
5. Add 10 fish to your environment. These fish need to be
  - a. at random locations (use your `randomLocation` method);
  - b. facing random directions (use `Environment`'s `randomDirection` method);
  - c. red.

Make sure that you handle the case where `randomLocation` returns a location that already contains a fish. Display your environment: `display.showEnv();`

6. Have your fish “breed” into each empty location that is adjacent to a red fish. The new fish need to be green and pointing in the same direction as the “parent” fish. Display the environment after you add each fish. You will need to use `allObjects` and `emptyNeighbors`.
7. Add a `changeColor(Color newColor)` method to the `Fish` class.
8. Move each fish backwards (if the location is empty). If the fish is able to move, turn it pink. Make sure that the fish is facing the correct direction after it moves. Display the environment after every successful move. You will need to use `allObjects`, `reverse`, `getNeighbor`, `isEmpty`, `changeLocation`, `changeDirection`, and `changeColor`.
9. Print out all the fish.
10. Extra Credit—**After printing and turning in your code**, change how you accomplish number 8 above as follows:
  - a. Erase all the code in the `Fish` `nextLocation` method. Replace with code that returns the location behind the fish if it's empty, or the current location otherwise.
  - b. Add code to the `Fish` `move` method to change its color if it changes position.
  - c. Erase all the “number 8” related code in `FishStuff12`. Replace it with code that causes each fish in the environment to act.

## Appendix B

```
/**
 * FishStuff12.java 12/04/03
 *
 * @author - Jane Doe
 * @author - Period n
 * @author - Id 000000
 *
 * I received help from ...
 */

public class FishStuff12
{
    private static final int NUM_ROWS = 15;
    private static final int NUM_COLS = 10;

    public static void main(String[] args)
    {
    }
}
```

## Activity 2: ListNode and TreeNode Cards

- Preparation—The instructor fills out six `ListNode` Cards to represent a linked list. Each card is identified by a letter and contains the contents of a `ListNode` (`value` and `next`). The `value` would be appropriate to the particular algorithm being demonstrated and `next` would be selected from the letters A–E or `null`. Each letter and `null` would be used once except for the head `ListNode`. The cards are stacked in list order with the head `ListNode` on top, the second `ListNode` second, and so on.
- Pre-Activity discussion—The instructor gives an example of a linked list algorithm and execution using connected `ListNode` diagrams on a white board. Discussion will illustrate how a `head` reference is used to start the initial method call, how each `ListNode` is associated with a single recursive call, and how `null` is associated with the base case.
- Activity—The instructor discusses a particular recursive algorithm to solve some specific linked list problem. This discussion includes a description of the base and recursive cases. Depending on the difficulty of the algorithm and the experience of the students, the instructor may also show the Java implementation of the algorithm. Possible problems include counting the number of nodes in a list, adding the values of nodes in a list, finding the maximum value in a list, and printing the list values in reverse order.

Students are then told that they will perform this algorithm on a linked list with each of them processing a single recursive call that corresponds to a single `ListNode`. The execution proceeds as follows:

- The instructor designates one student to process the head `ListNode` and then starts the algorithm execution by “calling” that student and handing him or her the stack of `ListNodes` as a parameter.
- The student processes the top `ListNode` card, making a recursive call to another class member when appropriate. When making the recursive call, the student performs a `getNext` by removing the top card and handing the remaining stack to the next student.
- Each student processes his or her `ListNode`, making a recursive call to the next student as appropriate.
- This continues until `null` is passed to one student. This student processes the base case and returns the result to the student who issued the call.
- As control is passed back, each student performs any remaining processing and passes the result back to the previous caller.
- The final result will be returned to the initial caller, the instructor.

Note: Each card is referenced by its letter. Students might imagine that the `ListNodes` reside in memory in alphabetical order. During the first exercise or two, the `ListNode` cards should be linked together so that they are not in alphabetical order. This helps students understand that consecutive `ListNodes` in a linked list might not reside in consecutive memory locations.

ListNode Cards for Activity 2

| ListNode A |       |
|------------|-------|
| value:     | next: |
|            |       |

| ListNode D |       |
|------------|-------|
| value:     | next: |
|            |       |

| ListNode B |       |
|------------|-------|
| value:     | next: |
|            |       |

| ListNode E |       |
|------------|-------|
| value:     | next: |
|            |       |

| ListNode C |       |
|------------|-------|
| value:     | next: |
|            |       |

|      |
|------|
| null |
|------|

## TreeNode Cards for Activity 2

| TreeNode A |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode E |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode B |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode F |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode C |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode G |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

| TreeNode D |        |        |
|------------|--------|--------|
| left:      | value: | right: |
|            |        |        |

|      |
|------|
| null |
|------|

### Pair Programming Standards and Requirements

In this class you may choose to do your programming assignments with a partner. This is a great tool to help you learn how to program and solve problems. Pair programming does not mean taking an assignment and partitioning it into two pieces and then having each person complete one piece.

Pair programming means working on an assignment together, sitting at the same computer. One person drives (types at the keyboard) while the other person sits next to the driver while observing, commenting, and making suggestions to the driver. This form of programming can speed up the programming process by having a helper immediately available. It also tends to reduce errors in the code because there are two sets of eyes examining the code. It also reduces errors because if the driver tries a questionable solution to a problem, the watcher will hopefully question the faulty logic and a better-thought-out solution will be found.

In order for you to receive the 10 percent extra credit on an assignment for pair programming, the following requirements must be met:

1. You must maintain a log of when you worked on the assignment. Record the day and time either person worked on the assignment. At least 80 percent of your time must be spent working together at one computer. Record the total amount of time spent on the assignment. Example:

```
6/2, 7:30–8:30, Mike, 1 hour
6/2, 7:30–8:30, Kelly, 1 hour
6/3, 7:00–9:00, Mike and Kelly, 4 hours
6/4, 8:00–9:30, Mike and Kelly, 3 hours
6/5, 8:00–10:00, Mike and Kelly, 4 hours
Total time 13 hours, 11 hours of pair programming
```

2. When working as a pair, the driver and watcher should swap roles about every 30 minutes.
3. Within the code, place comments stating who was driving when the code was written. Example:

```
//Mike driving now
public static void main(String[] args)
{   int x = 0;
    int y = 0;
    // more code
}
// end of Mike driving, Kelly driving now
```

4. Record the problems and challenges you encountered on the assignment. What was particularly difficult? What parts of the assignment required you to seek outside help?
5. Record the things you learned about programming and algorithms while doing the assignment.

All of the log requirements should be kept in a simple text file and added as a comment to the beginning of the file you turn in for the assignment.

### The Mathematical Matrix Class CS 307—Fall 2004—Assignment 3

**Purpose:** To practice programming with 2D arrays and create a class to meet a predetermined specification.

**Provided classes:** Matrix.java (to be completed), MatrixGenerator.java, ConsoleInput.java

**What to turn in:** Matrix.java. Remember, if you are working with a partner, you must complete the pair programming requirements to receive the bonus points.

“Linear algebra is a fantastic subject ... it is clean and beautiful. If you have three vectors in 12-dimensional space, you can almost see them.”

—Gilbert Strang, *Linear Algebra and Its Applications*

#### Matrices

**Background:** Mathematical matrices are used to solve systems of linear equations. Matrices are used in applications such as physics, engineering, probability and statistics, economics, biology, and computer science (especially in the area of computer graphics). The most important case and the simplest is when the number of unknowns equals the number of equations. Matrices appear in the following form:

|    |    |    |    |
|----|----|----|----|
| 1  | 5  | 10 | 5  |
| 6  | 4  | 12 | 4  |
| 10 | 5  | 12 | 11 |
| 5  | 11 | 23 | 9  |

The above matrix has 4 rows and 4 columns, but the numbers of rows and columns do not have to be equal. Each entry can be an integer or real number. For this assignment you will only deal with matrices of integers. You will implement a class, Matrix, which models a mathematical matrix and supports various operations (behaviors, methods) on matrices. In my explanation of the behaviors you must implement, I will use the following notation for matrices:

```
a0_0  a0_1  a0_2  a0_3
a1_0  a1_1  a1_2  a1_3
a2_0  a2_1  a2_2  a2_3
```

a1\_0 refers to the element in row 1, column 0. In the actual matrix above, that entry is 6.

Also note we are starting the row and column at 0 instead of 1. Computer programmers are comfortable with starting at 0; this Matrix class is designed for use by programmers, not necessarily mathematicians, who would be more comfortable starting the rows and columns with number 1.

**Requirements:** Create a Matrix class. You must use a “native” two-dimensional array of ints as your underlying storage container in the Matrix class:

```
private int[][] myCells; // or myNums or some other appropriate name
```

Implement the following methods. Use the method signature listed. Remember, you are creating a class that will be used by other programmers. You are creating a tool. There will not be any code to perform

## Appendix B

input or output in the Matrix class itself. (*Note:* If you add input or output code for debugging purposes, it should be disabled or commented out in the finished product.) You will be graded on how your Matrix class meets the requirements below. In grading we will always meet the preconditions of the methods when calling them. See method 3 for a hint on how to work with the private data of a Matrix object.

1. A default constructor

```
// pre: none
// post: up to the student

public Matrix()
```

2. A constructor that accepts a 2D array of integers and constructs a matrix with those values.

```
// pre: mat != null, mat is at least 1 by 1, mat is a
// rectangular matrix
public Matrix(int[][] mat)
```

3. A constructor with parameters for the number of rows, number of columns, and initial value for all elements:

```
// pre: numRows > 0, numCols > 0
public Matrix(int numRows, int numCols, int initialVal)
```

4. A method to change the value of a given element in the Matrix:

```
// pre: 0 <= row < numRows(), 0 <= col < numCols()
public void changeElement(int row, int col, int newValue)
```

5. Methods that return the number of rows or columns in the matrix.

```
public int numRows()
public int numCols()
```

6. Methods to change the number of rows or columns in the matrix. This may be a trimming operation or an expansion. If it is an expansion, set all new values to 0.

```
// pre: newNumRows > 0
public void changeNumRows(int newNumRows)
```

```
// pre: newNumCols > 0
public void changeNumCols(int newNumCols)
```

```
// pre: newNumRows > 0, newNumCols > 0
public void changeSize(int newNumRows, int newNumCols)
```

7. A method to access a particular value

```
// pre: 0 <= row < numRows(), 0 <= col < numCols()
public int getVal(int row, int col)
```

8. A method to add two matrices. Adding matrices is only possible if they have an equal number of rows and columns.

| matrix a |      | matrix b |      | result of a + b |               |
|----------|------|----------|------|-----------------|---------------|
| a0_0     | a0_1 | b0_0     | b0_1 | (a0_0 + b0_0)   | (a0_1 + b0_1) |
| a1_0     | a1_1 | b1_0     | b1_1 | (a1_0 + b1_0)   | (a1_1 + b1_1) |
| a2_0     | a2_1 | b2_0     | b2_1 | (a2_0 + b2_0)   | (a2_1 + b2_1) |

Example with values

| matrix a |   | matrix b |   | result of a + b |    |
|----------|---|----------|---|-----------------|----|
| 1        | 5 | 5        | 6 | 6               | 11 |
| 17       | 3 | 5        | 2 | 22              | 5  |

```
public Matrix add(Matrix rhs)
/* pre: rhs.numRows() = numRows(), rhs.numCols() = numCols()
This method should not alter the calling object or the rhs. It
should create a new, third matrix to store the result and return
that.
*/
```

How would this add method be used? In some other class besides the Matrix we might have the following code:

```
Matrix M1 = new Matrix(2,2,10);
Matrix M2 = new Matrix(2,2,-2);
M2.changeElement(0,1,5);
Matrix M3 = M1.add(M2);
```

What would the values of M3's myCells be after this code has executed?

```
solution:      8      15
              8      8
```

9. A method to subtract two matrices. Works almost the same as addition:

| matrix a |      | matrix b |      | result of a - b |               |
|----------|------|----------|------|-----------------|---------------|
| a0_0     | a0_1 | b0_0     | b0_1 | (a0_0 - b0_0)   | (a0_1 - b0_1) |
| a1_0     | a1_1 | b1_0     | b1_1 | (a1_0 - b1_0)   | (a1_1 - b1_1) |
| a2_0     | a2_1 | b2_0     | b2_1 | (a2_0 - b2_0)   | (a2_1 - b2_1) |

```
public Matrix subtract(Matrix rhs)
/* pre: rhs.numRows() = numRows(), rhs.numCols() = numCols()
This method should not alter the calling object or rhs. It should
create a new, third matrix to store the result and return that.
*/
```

## Appendix B

10. A method to multiply two matrices. This does not work as you would expect with simply multiplying the matching elements. Instead, matrices may be multiplied as long as the number of columns in the first matrix equals the number of rows in the second matrix. The resulting matrix has the same number of rows as the first matrix and same number of columns as the second matrix. Maybe an example would help:

Matrix a is an N(rows) by M(columns) matrix and matrix b is a K by L matrix.  $a * b$  is allowed if  $M = K$ . The result will be an N by L matrix. Here are the mechanics of matrix multiplication.

(Note: underscores have been omitted to conserve space.):

| matrix a (3 × 2) | matrix b (2 × 3) | result of a * b (3 × 3)                                     |
|------------------|------------------|-------------------------------------------------------------|
| a00 a01          | b00 b01 b02      | (a00*b00 + a01*b10) (a00*b01 + a01*b11) (a00*b02 + a01*b12) |
| a10 a11          | b10 b11 b12      | (a10*b00 + a11*b10) (a10*b01 + a11*b11) (a10*b02 + a11*b12) |
| a20 a21          |                  | (a20*b00 + a21*b10) (a20*b01 + a21*b11) (a20*b02 + a21*b12) |

Example with values:

| matrix a | matrix b | result of a * b  |                  |                  |
|----------|----------|------------------|------------------|------------------|
| 2 4      | 3 4 1    | (2*3 + 4*1 = 10) | (2*4 + 4*8 = 40) | (2*1 + 4*4 = 18) |
| 5 2      | 1 8 4    | (5*3 + 2*1 = 17) | (5*4 + 2*8 = 36) | (5*1 + 2*4 = 13) |

```
public Matrix multiply(Matrix rhs)
/* pre: numCols() = rhs.numRows()
This method should not alter the calling object or rhs. It should
create a new, third matrix to store the result and return that.
*/
```

11. A method to multiply every element of the matrix by some value:

```
public void scale(int factor)
```

12. Create a method to transpose a matrix. To transpose a matrix, the columns are taken directly from the rows of the original matrix. The nth row of the original matrix becomes the nth column of the transposed matrix:

| matrix a | transpose of matrix a |
|----------|-----------------------|
| 2 4 0    | 2 5                   |
| 5 2 12   | 4 2                   |
|          | 0 12                  |

Implement a mutator and an accessor for transposing matrices:

```
public Matrix getTranspose() // accessor, this Matrix unchanged
public void transpose() // mutator, transposes this Matrix
```

13. An equals method. Note you should override the equals method from Object, not overload it:

```
public boolean equals(Object rhs)
```

14. A toString method.

```
public String toString()
```

The MatrixGenerator class has two methods, getRandomMatrix and getMatrix. It is provided merely to help you test your Matrix class. It is not needed in the implementation of the Matrix class itself.

```
public int[][] getRandomMatrix(int rows, int cols,
    int lowerLimit, int upperLimit)
/*generates a rows by cols matrix with random values between lowerLimit
and upperLimit inclusive
*/
```

```
public int[][] getMatrix()
/*prompts for number of rows and columns from keyboard and then prompts
for each element
*/
```

**Restrictions: Again, the purpose of this is not to practice with the Java standard library. As such you may not use classes or methods from the Java standard library other than the String class, StringBuffer class, the Math class, and Integer class. You may find these classes useful in generating the String version of your matrix. You can, of course, use the built-in array type, including the length field.**

You must test the class on your own. Your file will be run against a test file to be graded as well as looked at to ensure that you used good style. When you finish, turn in your file, Matrix.java, via the turnin program.

```
/**
 * Matrix.java (to be completed)
 */
public class Matrix
{
    // pre: none
    // post: up to the student
    public Matrix()
    {
    }

    // pre: mat != null, mat is at least 1 by 1, mat is a
    // rectangular matrix
    public Matrix(int[][] mat)
    {
    }

    // pre: numRows > 0, numCols > 0
    public Matrix(int numRows, int numCols, int initialVal)
    {
    }
}
```

## Appendix B

```
// pre: 0 <= row < numRows(), 0 <= col < numCols()
public void changeElement(int row, int col, int newValue)
{
}

public int numRows()
{
}

public int numCols()
{
}

// pre: newNumRows > 0
public void changeNumRows(int newNumRows)
{
}

// pre: newNumCols > 0
public void changeNumCols(int newNumCols)
{
}

// pre: newNumRows > 0, newNumCols > 0
public void changeSize(int newNumRows, int newNumCols)
{
}

// pre: 0 <= row < numRows(), 0 <= col < numCols()
public int getVal(int row, int col)
{
}

public Matrix add(Matrix rhs)
{
    /* pre: rhs.numRows() = numRows(), rhs.numCols() = numCols()
    This method should not alter the calling object or the rhs.
    It should create a new, third matrix to store the result and
    return that.
    */
}

public Matrix subtract(Matrix rhs)
{
    /* pre: rhs.numRows() = numRows(), rhs.numCols() = numCols()
    This method should not alter the calling object or the rhs.
    It should create a new, third matrix to store the result and
    return that.
    */
}
```

```
public Matrix multiply(Matrix rhs)
{
    /* pre: numCols() = rhs.numRows()
       This method should not alter the calling object or the rhs.
       It should create a new, third matrix to store the result and
       return that.
    */
}

public void scale(int factor)
{
}

public Matrix getTranspose()
{
}

public void transpose()
{
}

public boolean equals(Object rhs)
{
}

public String toString()
{
}
}
```

## Appendix B

```
/**
 * MatrixGenerator.java
 * Used to generate matrices of integers.
 *
 * @author Mike Scott
 * @version 10/10/01
 */

import java.util.Random;

public class MatrixGenerator
{
    private static Random ourRandNumGen = new Random();

    /*generates a row by cols matrix with random values
    between lowerLimit and upperLimit inclusive
    pre: row > 0, cols >0, lowerLimit <= upperLimit
    post: returns an integer matrix of size row by cols with all
         elements between lowerLimit and upperLimit inclusive
    */
    public int[][] getRandomMatrix(int rows, int cols, int lowerLimit,
                                   int upperLimit)
    {
        int[][] result = new int[rows][cols];
        for(int r = 0; r < result.length; r++)
            for(int c = 0; c < result[0].length; c++)
                result[r][c] = ourRandNumGen.nextInt(upperLimit
                    - lowerLimit + 1) + lowerLimit;
        return result;
    }
}
```

```
/* prompts for number of rows and columns from keyboard and then
   prompts for each element
*/
public int[][] getMatrix()
{
    int numRows = 0;
    int numCols = 0;
    do
    {
        System.out.print("Enter number of rows for matrix: ");
        numRows = ConsoleInput.readInt();
    }
    while(numRows <= 0);
    do
    {
        System.out.print("Enter number of columns for matrix: ");
        numCols = ConsoleInput.readInt();
    }
    while(numCols <= 0);
    int[][] result = new int[numRows][numCols];
    for(int row = 0; row < numRows; row++)
        for(int col = 0; col < numCols; col++)
            {
                System.out.print("Enter value for row " + row
                    + " column " + col + ": ");
                result[row][col] = ConsoleInput.readInt();
            }
    return result;
}
}
```

### The Lexicon Assignment CS 307—Fall 2004—Assignment 7

**Purpose:** To practice working with sorting and searching algorithms. To practice creating an ADT on your own.

**Provided files:** WordLoader.java, FileHandler.java, ILexicon.java, jargonTest.txt

**Files to turn in:** SortedLexicon.java and UnsortedLexicon.java, Lexicon.java (if appropriate)

**Introduction:** One definition of a lexicon is a collection of words and phrases that have meaning or are understood by a group. The English language can be thought of as a lexicon. The Internet has its own lexicon, as does the world of computer programmers. UT has a lexicon as well. In this assignment you will write a class that models a lexicon. It will simply be a collection of Strings.

#### Requirements and limitations:

1. Consider the ILexicon interface as given below. A lexicon is a collection of words or phrases (Strings). Words and phrases are case-sensitive so, for example, Mike and mike are two different words. The entries into the lexicon can also include spaces and other non-letter characters. For example “Fandango on the Stack.”, “Error 404”, and “and there was much rejoicing”

Here are the methods from the ILexicon interface.

```
// add a phrase to the ILexicon, word != null
public void add(String phrase)
```

```
// add all the entries in lex to this ILexicon
public void add(ILexicon lex)
```

```
// remove all instances of phrase from this ILexicon
public void remove(String phrase)
```

```
// return a new Lexicon that is the
// combination of this ILexicon and lex
public ILexicon merge(ILexicon lex)
```

```
// returns true if phrase is in the ILexicon
public boolean isPresent(String phrase)
```

```
// returns number of phrases in this ILexicon
public int size()
```

```
// returns true if any phrase in this ILexicon starts with pre
public boolean isValidPrefix(String pre)
```

```
// returns true if any phrase in this ILexicon ends with suff
public boolean isValidSuffix(String suff)
```

```
// return number of phrases in this ILexicon that start with pre
public int numPrefixed(String pre)
```

```
// returns number of phrases in this ILexicon that end with suff
public int numSuffixed(String suff)
```

```
// returns an array of the phrases in this ILexicon
public String[] getAll()
```

2. You can use either an `ArrayList` or native array of `Strings` as your internal storage container. You may, if you wish, have more than one `ArrayList` or native array of `Strings` as private data members.
3. Obviously, you may use the `String` class and any of the methods from that class.
4. You must implement two versions of `ILexicon` interface, an `UnsortedLexicon` and a `SortedLexicon`. You may find it useful to have an abstract class `Lexicon`, which implements the common functionality of `SortedLexicon` and `UnsortedLexicon`.
5. Both `SortedLexicon` and `UnsortedLexicon` must override `toString` and `equals`.
6. The `UnsortedLexicon` class must include the following method:

```
// returns a sorted version of this Lexicon
public SortedLexicon getSorted()
```

7. The `SortedLexicon` is to maintain its entries in sorted order. You **may not** use any of the sorting or searching utilities in the Java Standard Library. Feel free to use algorithms from the book or as discussed in class, but you must cite the source of your sorting and/or searching algorithm or code in a comment. You may not use sorting or searching code from another student outside of your pair.
8. The `SortedLexicon` class maintains the set property; that is, there are no duplicate entries allowed in the `SortedLexicon`. If the `add` method is sent a `String` that is already present, then the `SortedLexicon` is unchanged.
9. For every method you write you must state the Big O, with  $N$  = number of words in the lexicon. It is expected that some of the operations in the `SortedLexicon` class will be faster than the same operation on the `UnsortedLexicon` class and vice versa.
10. Files to help test your lexicon:

The `WordLoader` assumes each line of a given text file is a single entry or phrase. The static method `getLex` creates an empty `ILexicon` (currently an instance of `UnsortedLexicon`), reads the phrases from the file with path equal to `name`, and adds them to the `ILexicon` object. This is accomplished via the following method:

```
public static ILexicon getLex(String name)
```

The variable `name` is assumed to be the full path of the file that contains words and phrases to be initially inserted into the `Lexicon`. `jargonTest.txt` contains a variety of words and phrases.

## Appendix B

```
import java.io.*;
/**
 * ConsoleInput.java
 * used for console (terminal) input which
 * is exceedingly ugly unless encapsulated in
 * a class. This is a beginning, allows reading String,
 * int, and double values from "standard input". Currently
 * each function reads an entire line, and parses according
 * to the memberfunction called, e.g., <em>readInt</em>
 *
 * In this, preliminary version bad I/O is caught, but
 * converted to a default value (e.g., 0 for int)
 *
 * This is a singleton class: one, per-program instance
 * is returned by using <em>getInstance()</em>
 *
 * @version 2.0, revised 10/22/2000
 * @author Owen Astrachan
 *
 */
public class ConsoleInput
{
    /**
     * reads a line from standard input, returns string
     * @return the string read from the terminal/keyboard/stdin
     */

    public static String readString()
    {
        String line=null;
        try {
            line = ourReader.readLine();
        }
        catch (IOException ioe) {
            System.out.println("error in readString");
        }

        return line;
    }
}
```

```
/**
 * reads a line from standard input, returns first int on line
 * catches NumberFormatException, returns 0 if caught
 * @return the int read from the terminal/keyboard/stdin
 *
 */

public static int readInt()
{
    String line = null;
    int local = 0;
    try
    {
        line = ourReader.readLine();
    }
    catch (IOException ioe)    { }

    try {
        local = Integer.parseInt(line);
    }
    catch (NumberFormatException nfe) {
        System.out.println("illegal numeric entry "+line);
    }
    return local;
}
```

## Appendix B

```
/**
 * reads a line from standard input, returns first double on line
 * catches NumberFormatException, returns 0 if caught
 * @return the double read from the terminal/keyboard/stdin
 */

public static double readDouble()
{
    String line = null;
    double local = 0.0;
    try {
        line = ourReader.readLine();
    }
    catch (IOException ioe)    { }

    try {
        local = Double.valueOf(line).doubleValue();
    }
    catch (NumberFormatException nfe) {
        System.out.println("illegal numeric entry "+line);
    }
    return local;
}

private static BufferedReader ourReader =
    new BufferedReader(new InputStreamReader(System.in));

}

/**
 * WordLoader.java
 * Mike Scott, 2004
 */
public class WordLoader
{
    public static ILexicon getLex(String name)
    {
        FileHandler myHandler = new FileHandler(name);
        ILexicon result = new UnsortedLexicon();
        String word;
        while(! myHandler.atEndOfFile() )
        {
            word = myHandler.readLine();
            if(word != null && !word.equals("") )
                result.add(word);
        }
        return result;
    }
}
}
```

```

/**
 * FileHandler.java
 * Mike Scott, 2004
 */
import java.io.*;

public class FileHandler
{
    private BufferedReader myReader;
    private boolean bMyAtEndOfFile;
    private String myFileName;

    /**
     * constructor with file name specified.<br>
     * pre: fileName specifies the full path to the file and
     * that file exists.
     * post: connection to file made. ready to call readChar or readLine
     * @param fileName a String with the full path to the file
     */
    public FileHandler(String fileName)
    {
        setNewFile(fileName);
    }

    /**
     * returns the rest of the current line from the file.
     * If 1 or more characters from the line have already been read
     * only the remaining characters are read.<br>
     * pre: atEndOfFile() == false<br>
     * post: return the rest of the current line as a String. if null
     * returned atEndOfFile() == true
     * @return a String with the rest of the current line. The end of line
     * character is not included at the end of the String. If the file
     * has been exhausted or an error occurs returns null
     */
    public String readLine()
    {
        String result = null;
        try
        {
            result = myReader.readLine();
            if(result == null)
            {
                myReader.close();
                bMyAtEndOfFile = true;
            }
        }
        catch(IOException e)
        {
            System.out.println("An error occured while trying to read");
            System.out.println("from the file " + myFileName
                + ". The error was");
            System.out.println(e);
            return null;
        }
        return result;
    }
}

```

## Appendix B

```
/**
 * returns the next character from the file.
 * pre: atEndOfFile() == false<br>
 * post: return the integer Unicode code for the next character in
 * the file or -1 if end of file reached if -1 returned
 * atEndOfFile() == true
 * @return an int, the Unicode code for the next character in the
 * file or -1 if no
 */
public int readChar()
{
    int result = -1;
    try
    {
        result = myReader.read();
        if(result == -1)
        {
            myReader.close();
            bMyAtEndOfFile = true;
        }
    }
    catch(IOException e)
    {
        System.out.println("An error occurred while trying to read");
        System.out.println("from the file " + myFileName
            + ". The error was");
        System.out.println(e);
        return -1;
    }
    return result;
}

/**
 * reset the file to the beginning of the file<br>
 * pre: none<br>
 * post: ready to read first character from the file,
 *       getAtEndOfFile() == false
 */
public void reset()
{
    close();
    setNewFile(myFileName);
}
```

```

/**
 * change the file to be read from to the file name specified.<br>
 * pre: fileName specifies the full path to a file and
 * that file exists.
 * post: connection to file made. ready to call readChar or readLine
 * @param fileName a String with the full path to the file
 */
public void setNewFile(String fileName)
{
    myFileName = fileName;
    bMyAtEndOfFile = false;
    try
    {
        myReader = new BufferedReader(new FileReader(myFileName));
    }
    catch(IOException e)
    {
        System.out.println("An error ocured while trying to
            connect"); System.out.println("to the file " + myFileName
            + ". The error was");
        System.out.println(e);
    }
}

/**
 * has the end of the file been reached?<br>
 * pre: none<br>
 * post: return false if end of file not reached yet.
 */
public boolean atEndOfFile()
{
    return bMyAtEndOfFile;
}

public void close()
{
    try
    {
        myReader.close();
    }
    catch(IOException e)
    {
    }
}

public void finalize()
{
    close();
}
}

```

## Appendix B

```
/**
 * ILexicon.java
 * Mike Scott, 2004
 */
public interface ILexicon
{
    // add a phrase to the Lexicon, word != null
    public abstract void add(String phrase);

    // add all the entries in lex to this Lexicon
    public abstract void add(ILexicon lex);

    // remove a phrase from the Lexicon
    public abstract void remove(String phrase);

    // return a new Lexicon that is the
    // combination of this Lexicon and lex
    public abstract ILexicon merge(ILexicon lex);

    // returns true if phrase is in the Lexicon
    public abstract boolean isPresent(String phrase);

    // returns number of phrases in this Lexicon
    public abstract int size();

    // returns true if any phrase in this Lexicon starts with pre
    public abstract boolean isValidPrefix(String pre);

    // returns true if any phrase in this Lexicon ends with suff
    public abstract boolean isValidSuffix(String suff);

    // returns number of phrases in this Lexicon that start with pre
    public abstract int numPrefixed(String pre);

    // returns number of phrases in this Lexicon that end with suff
    public abstract int numSuffixed(String suff);

    // returns an array of the phrases in this ILexicon
    public String[] getAll()
}
```