CS2013: Programming with Data Structures

Javadoc Comments, User Manual, & Project Deliverables

Keenan Knaur
Adjunct Lecturer

California State University, Los Angeles
Computer Science Department

User Manual

User Manual - Overview

- The User Manual is what explains to the user (i.e. ME) how to use your software.

- The user manual shall be submitted as a .pdf.
  - Other file formats will not be accepted or read.

- The user manual shall be as detailed as possible and will be as long as possible.
  - There is no minimum / maximum number of pages, but if your user manual is too sparse, and does help me to run your software, I will deduct from your grade.

User Manual - Content

- The user manual should include very detailed explanations of the following:
  - How to install and run your software.
  - How to use the software once it is running.
    - What are the exact steps to use all features of your software?
    - If I have to spend more than a minute figuring out what to do, then your instructions are not good enough and points will be deducted.

- Document any bugs that still exist.
  - If there are still issues with your software, they need to be documented.
  - If you fail to mention any bugs and I find them, it will have a more significant impact on your grade than if you document them yourself.
  - Of course you should not have any bugs!.

User Manual - Other Comments

- Never make assumptions about the level of knowledge of your user.
  - Unless your software is for a very specific subset of the population, never assume any user will know how to use your software.
  - This includes myself. When I grade your programs, I will be playing the role of your average user. I will click buttons and run commands in any order unless specified otherwise in your User Manual.
  - Again, the Manual must be easy to understand and the instructions very clear. I should not have to guess as to how I should operate your software.

Javadoc Comments

What are Javadoc Comments?

- Java has a third type of comment called **javadoc** comments.
- These comments start and end with /** */ and can appear on one line or across multiple lines.
  - NOTE: If you do not open with the double asterisk **, your comment will not be considered a Javadoc type comment.
- Javadoc comments use basic html and special annotations to document the source code.
- Once documented, your javadoc comments can be exported to a set of HTML pages that look just like the Java API.

What do you document?

- A javadoc comment is written in HTML and must come before a class, interface, datafield, constructor, or method declaration.

- All javadoc comments have two parts:
  - A description of the item you are documenting.
  - One or more block tags (annotations) describing specific features of the item your are documenting.

- Javadoc comments can contain any valid HTML.
  - i.e. large paragraphs should be denoted using the <p> tag.

- The following slide gives and example of how a method can be documented.

Example Javadoc Comment

```
/**
 * Returns an Image object that can then be painted on the screen. The url
 * argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 *
 * This method always returns immediately, whether or not the image exists. When
 * this applet attempts to draw the image on the screen, the data will be
 * loaded. The graphics primitives that draw the image
 * will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
 public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
 }
```

Javadoc Comments - Item Description

- You should use a concise, clearly defined description for each item you are documenting.
- Use simple, clear English with correct spelling, grammar, and punctuation.
- The first sentence is the most important, and should succinctly summarize the item you are documenting.
- Use the <code> html tag for all Java keywords, names, and code samples.
- Omit parenthesis when referring to a method that has no parameters or a method this is overloaded.
  - Example: **`The <code>add</code> method inserts items into the vector.`**

Javadoc Comments - Descriptions

- Method descriptions should begin with a verb since methods define a certain behavior or operation.
  - Example:
    - **`Determine whether this container is empty or not.`**

      is better than
    - **`This method is used to determine whether this container is empty or not.`**

- Avoid abbreviations if you can (this even includes common abbreviations such as a.k.a., etc.)

Javadoc Comments - Tags

- Javadoc tags identify important meta information about the code.
  - Example the @author tag easily identifies the author of the particular code.

- Each tag has a specific format.

Javadoc Comments - `@author` Tag

- **Form:** `@author name`

- **Used Where:** Interface and Class comments.

- **Used For:**
  - lists the names of all authors of the code
  - use the full name of the author or "unascribed" if the author is unknown
  - list authors in chronological order one tag per author.
  - creator of the class is listed first
  - any other people who worked on the class are listed next in the order in which they started to work on it.

Javadoc Comments - `@since` Tag

- **Form:** `@since version`
- **Used Where:** Interface and Class comments.
- **Used For:**
  - Indicates the version of the source code when this class or interface was introduced.
  - usually just a version number, but could also contain a specific date.

Javadoc Comments - `@version` Tag

- **Form:** `@version description`

- **Used Where:** Interface and Class comments.

- **Used For:**
  - indicates the current version number of the source code.
  - usually just a version number which includes the major and minor number
  - does not usually include the build number.
  - could also include a date.

Javadoc Comments - `@deprecated` Tag

- **Form:** `@deprecated`
- **Used Where:** Interface, class and method comments.
- **Used For:**
  - indicates that an item is deprecated.
  - something which is deprecated is no longer maintained or updated and should not be used in newly written code.
  - deprecated items are only included for backwards compatibility with old versions of programs which use your code.

Javadoc Comments - `@param` Tag

- **Form:** `@param name description`
- **Used Where:** Method comments.
-
- **Used For:**
  - Describes a method parameter.
  - name should be the formal parameter name.
  - description should be a brief one line description of the parameter.

Javadoc Comments - `@return` Tag

- **Form:** `@return` description
- **Used Where:** Method comments.
- **Used For:**
  - Describe the return value from a method
  - Does not apply to void methods or constructors.

Javadoc Comments - `@throws` Tag

- **Form:** `@throws exception description`
- **Used Where:** Method comments.
- **Used For:**
  - Indicates any exceptions that the method might throw
  - also gives the possible reasons for the exception occurring.

Javadoc Comments - `@see` Tag

- **Form:** `@see classname`
- **Used Where:** Any item being commented.
- **Used For:**
  - provides a link to another class if that class helps to clarify the item being commented.

Javadoc Comments - `@see class#member` Tag

- **Form:** `@see classname#member`
- **Used Where:** Any item being commented.
- **Used For:**
  - provides a link to another class's member if it provides additional clarity for the item being commented.

General Order of Tags

- If multiple tags are used in the same comment they should be listed in the following order:
  - `@author`
  - `@version`
  - `@param`
  - `@return`
  - `@throws`
  - `@see`
  - `@since`
  - `@deprecated`

Ordering Multiple Tags

- @author, @param, and @throws can be used more than once in the same comment.
  - multiple @author tags should be listed in chronological order (the order in which authors worked on the class).
  - multiple @param tags should be listed in the same order that they appear in the method header.
  - multiple @throws should be listed in alphabetical order according to the type of the exception (remember methods can throw multiple exceptions.)

Exporting the API for your Project

- Eclipse provides a built in tool for exporting your Javadoc comments as an API for your program.
- Project ➜ Generate Javadoc...
- On the next window you can choose which project and any of that projects packages, classes, etc that you want to be exported.
- Click finish when you are done.

Javadoc Examples

- The best examples of how to use Javadoc comments is to just look at the Java source code.
- Choose a class that you are familiar with (I like the String class).
- In any source code file in Eclipse, just look for the String class data type, or just type the word String.
- ctrl-left-click the word String in Eclipse and it should bring up the source code for the String class.
  - NOTE: If this does not work, you most likely do not have your project set to use the JDK instead of the JRE (which is the default option for most projects).
  - Google how to set your Eclipse project to reference the JDK instead of the JRE.

Project Deliverables

Project Deliverables

- The following deliverables are required for **every** project you turn in for this class:
  - A detailed User Manual (.pdf).
  - All of the source code (with the required Javadoc comments).
    - Upload the .java files **individually**, **DO NOT** zip the files.
  - The API files generated from the Javadoc comments.
    - Zip the entire folder that is generated.
  - An executable .jar file so that I can run your project as a standalone application.
    - Google how to do this.

References

- How to write Doc Comments for the Javadoc Tool, Oracle Website

- Javadoc Comments, Sourceforge.net Java Workshop Website

- Java `String` Class Source Code
  - You can view this in Eclipse.