Lab 1: Programming Tools
CS 2334

August 24, 2017

Introduction

The goal of this laboratory is to get everyone up to speed in using Eclipse, the Java compiler, Javadoc and Web-Cat submission. For most, this will require a short period of time to complete. Subsequent labs will occupy the full lab session period.

Objectives

By the end of this laboratory exercise, you should be able to:

1. Install Java version 8 and Eclipse
2. Compile and execute a Java program in Eclipse
3. Apply basic String-manipulation methods to parse and create Strings
4. Create a class and construct an instance of that class
5. Use Javadoc to generate documentation for your code
6. Submit your program and documentation for grading

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Instructions

1. You should have already done the following:
   - Download and install SE Development Kit version 8 from the following address:
     - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.
   - The use of Java 8 is required for this course.

     You can check which version of Java you have installed by opening the Terminal (Mac/Linux) or the Command Prompt (Windows) and running the command *java -version*
   - Download and install the Eclipse IDE for Java Developers from http://www.eclipse.org/downloads/. We strongly recommend that you use Neon (version 4.6).
   - Install the Web-Cat submission plug-in to Eclipse. See: http://cs.ou.edu/∼fagg/classes/cs2334/webcat/webcat.html
2. In Eclipse, create a new Java Project called *lab1*.

   Select *File/New Java Project*

   or

   Select *File/New Project/Java Project*
3. In this project, create a class called `Driver`.
   - Select the lab1 project in the *Package Explorer*
   - Right-click and select *New/Class*
   - Give the class a name (`Driver`), select the `Default` package, and click *Finish*
4. Task 1: In your `Driver` class, create the `main` method. The `main` method should print the text *Hello, world!* to the screen.
5. Compile and execute your program. Left-click on the green *Start* button on the top tool bar. This should result in the string being displayed in your console window. Task 1 is complete.

6. Task 2: Create a class called *Course*. This class must be in the `Default` package and will represent the course enrollment and TA coverage for a single course.

   This class must have the following properties:

   - It must have one instance variable that is a String array of size 4 called *info*
   - It must have one constructor:

     ```
       public Course(String input)
     ```

     where input is guaranteed to be a String of the form a, b, c such that:

     – a is a string consisting of alphanumeric and white space characters, and

     – b, c are strings consisting of only numerical characters (and no spaces).

     For example, "Java I,100,6" is such a string where $a =$ "Java I", $b =$"100", and $c =$ "6". For this program, we're interpreting a to be the course name, b to be the course enrollment, and c the number of teaching assistants assigned to the course.

     This method will store a in all capital letters in `info[0]`, store b in `info[1]`, store c in `info[2]`, and store one of two strings in `info[3]`. If there are at most 30 students per TA, then this string must be set to "Well Covered"; otherwise, this string is set to "Poorly Covered". Note that this last operation requires the use of type conversions.

   - It must have a method with the following header (we also call this a *prototype*):

     ```
       public String toString()
     ```

     which returns a String of the form:

     ```
     Course: info[0], enrollment: info[1], teaching assistants: info[2], ↩
         info[3]
     ```

     where *info[i]* is the value of the i^{th} element in the array.

     Note 1: The arrow in the above string is there to show that the line wraps. It is not printed and your output **should not** contain a newline character here.

Note 2: The spaces and capitalization matter.

Note 3: When an object is passed into a method such as `System.out.print()`, the String returned by the object's `toString()` method is printed in place of the object! All objects have the `toString()` method and you can override it to suit your needs, as you have done for this step.

- All of its methods and data must be documented using Javadoc (see below for details).

7. Replace your Driver class main method with the following:

```
public static void main(String[] args) throws IOException
{
   BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
   String input = br.readLine();

   // Add two more lines of code here.

   br.close();
}
```

After the first two lines of code are executed, the *input* variable will contain the string of characters that have been typed by the user in the console (up to when ENTER is pressed).

8. Use the *input* string to create an instance of the Course class and print it out.

9. Test your code. Here are some examples. The inputs are to be typed into the console. Your program will then respond in the console with the output.

Input:

```
   Java I,100,6
```

Output:

```
   Course: JAVA I, enrollment: 100, teaching assistants: 6, Well Covered
```

Input:

```
Java II,185,4
```

Output:

```
Course: JAVA II, enrollment: 185, teaching assistants: 4, Poorly Covered
```

10. Generate Javadoc using Eclipse.
    - Select your project (e.g., by selecting either the Driver.java or Course.java file)
    - Select *Project/Generate Javadoc...*
    - Select *Private* visibility
    - Use the default destination folder
    - Click *Finish*
11. Open the *lab1/doc/index.html* file using Eclipse or your favorite web browser. Check to make sure that both of your classes are listed and that all of your documented methods have the necessary documentation.
12. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Project (Class) Documentation

Below the import statements at the top of every java source file (and above the class declaration), you must include a documentation block at the top of the file that includes the following:

```
/**
   @author <your name(s)>
   @version <today's date>
   Lab <number>

   <A short, abstract description of the file>
*/
```

Note that the notation:

```
<some text>
```

indicates that you should provide some information and not actually write the less-than and greater than symbols. Also, the double splats (stars) at the beginning of the comment are necessary to be recognized by Javadoc as being important.

Instance/Class Variable Documentation

As we start to implement classes that contain either *instance* or *class* variables (more on the difference in later labs), you must have a line of documentation for each of these variables. For example, consider a *Person* class. Here are some example documentation lines with variable declarations:

```
/** First name of the Person */
String firstName;

/** Last name of the Person */
String lastName;

/** Unique identifier */
int idNumber;
```

Note that this Javadoc format for documentation (with the double splat) **is not** required for variables defined locally within a method. Nevertheless, you still need to document the meaning of your variables.

Method-Level Documentation

Every method must include documentation above the method prototype using standard Javadoc markup. This documentation should be sufficient for another programmer to understand *what* the method does, but not the details of *how* the method performs its task. For example, consider a method that will test whether a value is within a range and whose prototype is declared as follows:

```
public static boolean isInRange(double min, double max, double value)
```

The documentation for this method will be placed above the prototype in your java file and might look like this:

```
/**
   Indicate whether a value is within a range of values

   @param min Minimum value in the range
   @param max Maximum value in the range
   @param value The value being tested
   @return True if value is between min and max.  False if outside
this range.

*/
```

Note that this example includes all the required documentation elements:

- A short, intuitive description of what the method does (not how it does it),
- A list of the parameter names and a short description of the *meaning* of the parameters, and
- A short description of the return value.

Inline Documentation

Inside of each method, you must also include documentation that describes *how* a method is performing its task. This documentation should be detailed enough for another programmer to understand what you have done and to make modifications to your code. Typically, this documentation is preceded by "//" and occupies a line by itself ahead of the code that is being documented. While each line of code could be documented with its own documentation line, it is typically not necessary or appropriate. Instead, we typically use a single documentation line to capture what a small number of code lines is doing.

In addition, inline documentation should be done at a logical level and should not simply repeat what the line of code says.

Here is an example:

```
public static boolean isInRange(double min, double max, double value)
{
    // Check lower bound
    if (value < min)
    {
        return false;
    }

    // Check upper bound
    if (value > max)
    {
        return false;
    }

    // Within the boundaries
    return true;
}
```

Program Formatting

Our Web-Cat server will enforce a particular program formatting standard. In addition to the project/class- and method-level documentation, it will also check for several other items, including:

- Proper indentation. Indents must be done with spaces only (and not tabs).
- A space between a conditional keyword (such as *if* or *while* and the following open parenthesis.
- Curly brackets on their own lines.
- Code blocks surrounded by curly brackets. For example, an *if (...)* statement must be followed on the next line with an open curly bracket.

- *if* and *while* must be followed by a space, then an open parenthesis.
- Lines with at most 120 characters.
- Use of all declared variables.
- Use of all import statements.

Note that Eclipse can be configured to automatically do much of this formatting for you (see our Web-Cat page).

Submission Instructions

Before submission, finish testing your program by executing your main() method. If your program behaves as you expect, then you are ready to attempt a submission. Here are the details:

- All required components (source code and compiled documentation) are due on Saturday, August 26, 48 hours after the end of your lab session.
- Prepare and submit your program:
  1. Select your project.
  2. From the Project menu, select *Submit Assignment*.
  3. Under *Select the assignment to submit*, select *Lab1: Programming Tools*.
  4. Click *Change Username or Password...*. Enter your Web-Cat username and password. Click *OK*. You should only need to do this step once per session.
  5. Click *Finish*.
  6. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.
- Don't worry about the submission step – you will be able to submit multiple times and view the automatic feedback and grade each time.

- At some time after you submit your code, the server will give you a report with some items automatically graded. How fast the server provides this report depends on many factors, including how many others are submitting at the same time. Until the deadline, you may use the report to improve your code and submit again.
- The Eclipse plug-in will not work well for a small number of you. For a direct method of submission, see:
  http://cs.ou.edu/∼fagg/classes/cs2334/webcat/webcat.html

Hints

- See the **String** class API for how to split and interpret Strings.
- See the **Integer** class API for how to interpret Integers.
- The main class web page has a link to the Java API documentation.

Grading Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described above will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions (where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late. For a submission to be considered *on time*, it must arrive at the server by the designated minute (and zero seconds). For a deadline of 9:00, a submission that arrives at 9:00:01 is considered late (in this context, it is one minute late).

After 15 submissions to Web-Cat, you will be penalized one point for every additional submission.

For labs, the server will continue to accept submissions for one day after the deadline. In these cases, you will still have the benefit of the automatic feedback. However, beyond ten minutes late, you will receive a score of zero.