LAB 1

Learning Outcomes

- Create a custom class with relevant instance variables
- Add a constructor and getter and setter methods to a class
- Instantiate objects of a custom class
- Identify when and how to add toString() and equals() methods to a class
- Document code thoroughly to generate clear Javadocs

Pre-Lab

Read the Javadoc explanation.

Question: What information is needed in a method's javadoc? What tags will you need to use?

Exercise 1 – Develop a custom class

1. Create a new Project (select File > New > Java Project) and name it: Lab1
2. Within this project, add a new class and name it: Player
3. This class will represent hockey players, so it should have three private instance variables: name (of type String), position (String), and jerseyNum (int)
4. Add a constructor that takes in three parameters to match the instance variables (i.e. same names and types).
5. In this constructor, assign each of the instance variables with the corresponding input parameter. You must use "this." before assigning them since the names are the same.

Question: Why do we need to use the "this" keyword? What happens if we don't use it?

6. Below the constructor, add a getter method called getName that returns the player's name. Note that the return type "String" must be included in the method signature.
7. Add two more getter methods, getPosition and getJerseyNum, that return the corresponding variable values.
8. After the three getter methods, add a setter method called setName that takes in a String variable called newName and assigns that input parameter's value into the name. Note that the return type should be void since nothing has to be returned.

9. Follow this setter method with two more setters: setPosition and setJerseyNum
10. At this point, your Player.java class should have three private instance variables at the top, a constructor, three getters, and three setters.

Exercise 2 – Test the class methods

1. Inside the Lab1 project, add another class with the name TestLab and add a main method (either by checking the "method stub" checkbox in the New Class window or by typing the method signature manually).
2. In the main method, create a new Player variable called p1 and instantiate it as a new Player object. Use your name for the player's name, and use any value for position (i.e. "defence") and jerseyNum.
3. Add three print lines: one to print `p1.getName()`, one for `p1.getPosition()`, and one for `p1.getJerseyNum()`.
4. Save and run the file. The values you used for the player will be printed to the console.
5. Use each of the setters to change your name, position, and jersey number.
6. Copy the print lines with the getters from above and paste them after using the setters.
7. Save and run the file. You should now see the updated values, which were just changed from the setters, printed to the console.
8. Add a print line with just p1 in the parentheses (i.e. printing the object itself, not a specific variable retrieved from a getter).
9. Save and run the file. The bottom print line looks like gobbledygook! This is the default format of printing out an object; it starts with the class name followed by a sequence of characters. In the next exercise, you will add a toString() method which overrides this default format for printing.
10. Create another Player object called p2 and instantiate it with the values that you used in the setters (i.e. the updated p1 values and p2's default values should be the same).
11. Add the following lines of code to compare the two objects:

```
if (p1.equals(p2)) {
    System.out.println("Same player");
} else {
    System.out.println("Different player");
}
```

Question: Which of these lines do you expect to print out?

12. Save and run the file. Was your prediction correct?
13. Much like the toString() method, there is a default equals() method that you can override to control the way comparisons are made between two objects of your class.

Exercise 3 – Adding toString() and equals() methods

1. Go back into the Player file.
2. Near the bottom (after your getters and setters), add a toString method that is public and has a String return type. Within this toString() method, type:
   `return this.name + ":  #" + this.jerseyNum;`
3. Go back to the TestLab file and run it. The print line of the object p1 should now display relevant information from the object in the format specified in toString() rather than the sequence of characters it showed before.
4. Go back again to the Player class.
5. After the toString() method, add a method called equals that is public and has a boolean return type. This method must take in a single parameter of type Player and we'll call it "other" for simplicity.
6. In the equals method, add a conditional that checks
   `if this.name.equals(other.name)`
   and if it does, return true. If not, return false.

Question: Which class contains the equals() method being called here? Is it calling itself?

7. The problem with this equals method currently is that two players who happen to have the same name would be considered the same person which is not correct. Adapt the conditional to return true if the names are equal AND the jersey numbers are equal (HINT: use && for an AND operator, and use == when comparing numbers). This still might not be a perfect way of comparing players (i.e. consider two different people named John who each wear #25 on their respective teams) but it's sufficient for this exercise.
8. Save the file. Go back to TestLab and run it.
9. Experiment by changing one of the names and/or positions to test the equals() method with various cases.

Exercise 4 – Writing comments and generating Javadocs

1. Go into the Player class.
2. In the method getName(), add a single line comment (//) just above the return statement. The lines in this method should now look like this.
   ```
   // Get the player's name.
   return name;
   ```
3. In the constructor, add a few empty lines at the top before initializing the variables.

4. Add a multi-line comment in that space by starting with `/*` and ending on another line with `*/`
5. Add some descriptive text within this comment section. For example,
```
/*
This is the constructor so we will be
initializing the member variables here
*/
```
6. Refer back to the pre-lab reading, which you should have read before starting, to review how Javadoc comments are formatted.
7. Add Javadoc comments to class Player to explain purpose and author of the class, purpose, parameter, and return value of the methods (you can just do it for the constructor, one of the getters, and one of the setters to see how it works).
8. Click on Project > Generate Javadoc...
9. For the Javadoc command, click on the Configure button, locate the "javadoc" program and click "Open". This program will be in the "bin" directory of the Java Development Kit installation on the machine. This could for instance be C:\Program Files\Java\jdk1.8.0_03\bin\javadoc.exe
10. Select the project "Lab1" for which you want to generate the documentation.
11. Ensure that Public and Use Standard Doclet are selected and click Finish.
12. Javadoc generates a set of html files which comprise the documentation for the program. You should see these in the Package Explorer in Eclipse inside "Lab1" under the "doc" directory. You can view any of the html files by selecting "doc" and then double clicking on an html file (start with index.html).

Submission

When you have completed the lab, navigate to the weekly module page on OWL and click the Lab link (where you found this document). Make sure you are in the page for the correct lab. Upload the files listed below and remember to hit Save and Submit. Check that your submission went through and look for an automatic OWL email to verify that it was submitted successfully.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the lab.
- Do not ZIP or use any other form of compressed file for your files. Attach them individually.
- Submit the lab on time. Late submissions will receive a penalty.
- Forgetting to hit "Submit" is not a valid excuse for submitting late.
- Submitting the files in an incorrect submission page will receive a penalty.

- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular lab deadline will receive a penalty.

Files to submit

- Player.java
- TestLab.java