LAB 8

Learning Outcomes

- Differentiate between static and non-static variables and methods and their usages
- Compare the effects of changing primitive variables and objects/arrays within instance methods.
- Analyze the approaches to checking for equality between String objects
- Apply String formatting to create a clean alignment in print lines

Pre-Lab

- Create a new Java project called Lab8
- Download the files: VariableTest.java, MemoryDemo.java, StringCompare.java, and Product.java
- Save these downloaded files into the Lab8 src folder

Exercise 1 – Accessing static vs. non-static variables

1. Open VariableTest.java in Eclipse and examine the code. Notice the difference between sNum and iNum.
2. Add a test harness `main (String[] args)` method at the bottom of this class. Within it, add two consecutive print lines: one printing out iNum and one printing out sNum.
3. What does the IDE / compiler say about these lines? Why is one of them valid but the other invalid? Comment out the offending line.
4. Create an object and print out the iNum value of that object.

Exercise 2 – Method parameters

When a primitive variable is declared, it is allocated a memory location. When that primitive variable is initialized (for example, with the value of 5) the value at its memory location takes on the value of 5. In contrast, when a reference variable is declared and initialized, the value at its memory location takes on the value of another memory location. Given this information, can you explain why the following demo works the way that it does? If you are confused, try drawing the memory locations and their values on paper.

1. Open MemoryDemo.java and examine the code but do **not** run it.
2. As you examine the code visually, write down the results you expect it to print out.
3. Once you have written the expected results, run the program to see the actual results. Were you correct? If not, review the paragraph at the start of this exercise to try to understand the reason for these results.

Exercise 3 – String equality

A string created without the use of the new operator is called a string literal (i.e. String s = "hi"). String literals are objects, but the Java compiler will not create two copies of the same string literal. The string literals are saved into a special memory pool and duplicates point to the same literal to reduce the amount of memory that a Java program uses.

1. Open StringCompare.java but do **not** run it.
2. Carefully read through the 6 different test cases written in the test harness. Notice that the first 3 cases use == and the last 3 call the equals() method. Notice also that in each of these sets of 3 cases, the Strings are initialized slightly differently.
3. As you examine these 6 different String comparison cases, write down the results you expect from each of the cases (true or false).
4. Once you have written expected results, run the program to see the actual results. Were you correct? For any cases in which your expected result was incorrect, review the code to understand why this occurred. Review the paragraph on string literals at the start of this exercise if you are confused by these results.

Exercise 4 – Create a receipt program

1. Open Product.java in Eclipse and examine the code.
2. Create a new class called Receipt.
3. The only instance method you need to implement in Receipt.java is a constructor with the following signature: `public Receipt (Product[] cart) { }`
4. In the constructor, loop through each of the products in the cart array and display each one on its own line with clean formatting. Each line must include the product's name, code, cost, and the after-tax cost. Use the following hints and guidelines:
   - Align the print lines in a table-like structure so that each row (line) is a single product and each column is each of the attributes (name, code, etc.)
   - Use `String.format()` or `System.out.printf()` with "%" to help with the alignment. Refer to the zyBook section 9.2 for a refresher.

- If you call prod.getTax() (assuming prod is the Product variable), it will be underlined in yellow to indicate a warning. This means it will technically still compile and run with this line, but it is not recommended. How is this method different from the other getters in Product.java? How should this type of method be called? Call it the proper way so there are no warnings.

5. Add a print line before the loop to display header labels for each of the columns (product name, code, cost, and after-tax cost) with the same table-like alignment.
6. Add a line after the loop that displays the grand total and keep it aligned with the column of the individual product's after-tax costs.
7. Add a test harness `main (String[] args)` method at the bottom of this class. Within it, initialize at least 3 different Product objects and create a Product array containing these items. Then initialize a new Receipt object with the Product array parameter.
8. View an example of the expected output (note that you can use any product names, codes, and costs so your output may not be identical to this one).

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

- VariableTest.java
- Receipt.java