CS1020: Data Structures and Algorithms I

Tutorial 1 – Java and Simple OOP

(For week 3, starting 27 January 2014)

There are no tutorials in week 3. This tutorial is meant for you to attempt on your own. You may discuss your answers or seek clarification in the IVLE forum.

This tutorial covers topics taught in weeks 1 and 2.

1. [Week 1 topics: Java naming convention, syntax error, logic error]
   The program below has several issues. Study the program and try to compile and run it.

```
import java.util.*;

public class tut1Q1 {

   public static void main(String[] args) {
      Scanner Input = new Scanner(System.in);

      System.out.print("Enter length of box: ");
      int BOX_LENGTH = Input.nextInt();
      System.out.print("Enter width of box : ");
      int BOX_WIDTH = Input.nextInt();
      System.out.print("Enter volume of box: ");
      int BOX_VOL = Input.nextInt();

      double BOX_HEIGHT = ComputeHeight(BOX_LENGTH,
                                        BOX_WIDTH, BOX_VOL);
      System.out.println("Height of box = " + BOX_HEIGHT);
   }

   public double ComputeHeight(int aa, int bb, int cc) {
      return cc / (aa * bb);
   }
}
```

   (a) The names of the class, method and variables in this program do not follow Java Naming Conventions. (There are several sites on Java naming conventions. One is http://www.javacodegeeks.com/2011/08/java-naming-conventions.html)
   Replace all inappropriate names with those that follow the naming conventions.

   (b) The names of the parameters of the **ComputeHeight()** method are poorly chosen. Replace them with more descriptive names.

   (c) There is a compilation error:

```
non-static method ComputeHeight(int,int,int) cannot be
referenced from a static context
         double BOX_HEIGHT = ComputeHeight(…)
```

   Correct the error.

(d) There is a logic error in the computation of the height in the **ComputeHeight()** method. Correct the error.

After all corrections are made, the program should give the correct output. A sample run is shown below:

```
Enter length of box: 8
Enter width of box : 5
Enter volume of box: 60
Height of box = 1.5
```

2. [Weeks 1 and 2: **Math** class, writing method]
The area of a triangle can be computed given the lengths of its sides ***a***, ***b*** and ***c*** with **Heron's Formula**:

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

Where ***s*** is half the triangle's perimeter:

$$s = \frac{a+b+c}{2}$$

Write a program **TriangleArea.java** that reads 3 positive values of type **double** representing the lengths of the sides of a triangle, and computes the area of the triangle. The program should include a method **area()** to compute the triangle's area. You are to determine what parameters the method should have. Print the area in 2 decimal places.

3. [Weeks 1 and 2: **String** class]
The **String** class is a rather special and interesting class. Try to explore on your own and read up more about it as strings are used extensively.

(a) The following code:

```
String str = "one-" + "two" + " buckle my shoe";
System.out.println(str);
```

produces this output:

```
one-two buckle my shoe
```

If the assignment statement above is replaced by the following, what would be the output?

(i) `String str = 1 + 2 + " buckle my shoe";`

(ii) `String str = "" + 1 + 2 + " buckle my shoe"`

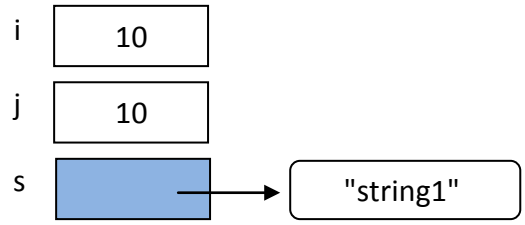
(iii) `String str = '1' + 2 + " buckle my shoe";`

(iv) `String str = 1 + "-2" + " buckle my shoe";`

(b) Given the following code fragment:

```
int i = 10;
int j = i;
String s = new String("string1");
```

the memory can be depicted as below:



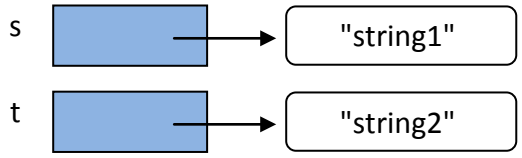
Give a depiction of the memory if the following statements are added to the code above and executed:

```
String t = s;
s = new String("string2");
```

(c) Given the following code fragment:

```
String s = new String("string1");
String t = new String("string2");
```

the memory can be depicted as below:



Give a depiction of the memory if the following statements are added to the code above and executed:

```
String u = s;
s = t;
t = u;
```

(d) For primitive types, the equality operator **==** is used to compare two values to determine if they are equal. For example, given two **int** variables *a* and *b*, the condition (*a* == *b*) checks if *a* and *b* have the same value.

For objects, **==** is a ***reference equality operator*** to compare two references to determine if they refer to the same object. To compare two (distinct) objects to determine if they store the same values, use the **equals()** method instead.

Given the following code fragment:

```
String s1 = new String("programming");
String s2 = new String("programming");
String s3 = s2;
System.out.println("s1 == s2: " + (s1 == s2));
System.out.println("s2 == s3: " + (s2 == s3));
System.out.println("s1 == s3: " + (s1 == s3));
```

```
System.out.println("s1.equals(2): " + (s1.equals(s2)));
System.out.println("s2.equals(3): " + (s2.equals(s3)));
System.out.println("s1.equals(3): " + (s1.equals(s3)));
```

Give a depiction of the memory after the above code is executed, and hence determine the output of the above code.

(e) In parts (b) – (d) above, we create **String** objects using the **String** constructor.

You may also create a **String** object using string literal, for example:

```
String s = "hello";
```

Test out the code below and find out why the output is such.

```
String t1 = "CS1020";
String t2 = "CS1020";
String t3 = "CS" + "1020";
System.out.println("t1 == t2: " + (t1 == t2));
System.out.println("t2 == t3: " + (t2 == t3));
System.out.println("t1 == t3: " + (t1 == t3));
```

(f) You may have heard that strings in Java are **immutable**. What does it mean?

4. [Week 2: Overloading]
In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be *overloaded,* and the process is referred to as *method overloading.* (From http://www.java-samples.com/showtutorial.php?tutorialid=284)

Conditions for overloading:

- The number of parameters are different; or
- The parameter types are different

There are many overloaded methods (even overloaded constructors) in many classes in the API. For instance, the **Math** class has these overloaded methods to return the absolute value of its parameter a:

- `static double abs(double a)`
- `static float abs(float a)`
- `static int abs(int a)`
- `static long abs(long a)`

For example, **Math.abs(-5)** calls the third method; **Math.abs(-3.7)** calls the first method (because **double** is the default type for real numbers); **Math.abs(-3.7f)** calls the second method (to indicate a **float** value, the number is suffixed with f or F).

Given the following imaginary class C:

```
public class C {

   public static int m() {
      return 123;
   }

   public static int m(int a) {
      return 3 * a;
   }

   public static int m(int a, int b) {
      return a + b;
   }

   public static int m(double a, double b) {
      return (int) (a * b);
   }
}
```

What is the output for each of the following statements?

(a) `System.out.println(C.m());`
(b) `System.out.println(C.m(7));`
(c) `System.out.println(C.m(3.5));`
(d) `System.out.println(C.m(10, 20));`
(e) `System.out.println(C.m(10, 20.0));`

- End of Tutorial 1 -