CSCI 161 Introduction to Computer Science

Review Guide 1 (Selected Solns)

Practice Problems

1. `sentence` is a String storing a reference to the literal, ``i heart CS.'' After calling `sentence.toUppercase()`, `sentence` would now store a reference to ``I HEART CS.''

   TRUE / FALSE
   Explain: False. Strings are *immutable* objects, meaning that its methods cannot change their current state. Instead, the methods return a new String with the desired mutation. Therefore, one must reassign the String, `sentence = sentence.toUppercase()`, for the mutation to last.

2. The output on the BlueJ terminal from calling `System.out.println("5" + 6)` is 56.

   TRUE / FALSE
   Explain: True. When the compiler encounters the plus operator (+), it observes the two operands on either side of it. If one of them is a String, then the + is interpreted as concatenation instead of add.

3. For each of the following Java keywords, describe what it is used for, and write a snippet of code demonstrating its use.

   (a) `else if`
   An else if statement must follow either an if-statement or another else-if statement. It contains a boolean condition and executes the prescribed statements statements depending on the result of the condition. If it triggers, it must mean that all previous if and else-if statements were false. All other else-if and else statements that follow a triggering else-if statement are skipped.

```
1 if (x < 0) {
2     System.out.println("x is negative!");
3 }
4 else if (x > 0) {
5     System.out.println("x is positive!");
6 }
7 else {
8     System.out.println("x is zero!");
9 }
```

(b) `double`

A double is a primitive data type, specifically, a double-precision floating point number. This can be used various contexts, including (1) variable declaration, (2) type-casting, and (3) return-type declaration. I will show an example of (1) and (2).

```
int x = 2, y = 3;
double z; // declare a variable that can store doubles
z = x / (double) y; //cast y to a double
System.out.println(z); // 0.66666666667
```

(c) `public class ...`

This statement starts a class declaration.

```
public class WaterBottle {
...
}
```

(d) `void`

A void is used to state that a method returns nothing.

```
public void doStuff() {     //do stuff, but cannot return }
```

(e) `null`

A null is used to say that an object variable references nothing. Aside: It is the default value of object variables if they were declared and not instantiated. A user can also explicitly set an object variable to null.

```
WaterBottle b;
if (b == null) {
//this will fire
}
```

4. Explain the differences between a constructor and a mutator (setter) method.

While constructors and setters are similar in that they both assign values to an object's fields, there are several important differences. First, constructors can only be called when a new object is being instantiated; methods can be called at anytime (and as often as you like) *after* an object has been instantiated. Second, constructors are used to give initial values to fields, though it should be noted that constructors can call setters to assign these initial values.

5. Write a method `public int getAmountOdd(int n1, int n2, int n3)` that returns the number of its input-parameters which are odd. When it's called, it'd have the following effect: (8pts)

```
getAmountOdd(3,5,6);    // 2
getAmountOdd(0,2,4);    // 0
getAmountOdd(10,2,11); // 1
```

To answer this question, you have to observe two things. First, the three parameters are passed in as integers. Second, you can always tell if an integer is odd or even if its remainder, when divided 2, is 1. In other words, any even integer would be evenly divided by 2, producing no remainder. The modulo operator (%) is used here.

```
/**
 * This method determines the number of its input parameters are odd numbers.
 * @param n1 an integer
 * @param n2 an integer
 * @param n3 an integer
 * @return the number of inputs that are odd
 */
public int getAmountOdd(int n1, int n2, int n3)
{
    int numOdd = 0;  //holds the number of params that are odd
    if (n1 % 2 == 1) {
        numOdd++;
    }
    if (n2 % 2 == 1) {
        numOdd++;
    }
    if (n3 % 2 == 1) {
        numOdd++;
    }
    return numOdd;
}
```

6. Given the following declarations, determine the value *and* the data type of each of the expressions listed below.

```
int x = 50;
double y = 50;
```

(a) `x / 3`
16 (int)

(b) `y / 3`
16.666666... (double)

(c) `x / y`
1 (double)

(d) `x == 5 * 5 + 25`
true (boolean)

(e) `x % 9`
5 (int)

(f) `y *= 2`
100 (double)

7. For each of the following expressions, determine the value stored inside the variable being assigned. Assume that the following variables have been defined. Where applicable, write "Error" and give a reason if the expression will not compile. There is no carry-over from one question to the next.

```
int iResult = 10;
double fResult = 20.2;
boolean bResult = false;
```

(a) `bResult--;`
Error. Booleans can't be decremented.

(b) `bResult = bResult && (iResult == fResult);`
false

(c) `iResult = fResult;`
Error. Can't assign double into a int variable without a cast.

(d) `iResult = (int) fResult;`
20

(e) `fResult = iResult;`
10.0

(f) `fResult = (double) (iResult / 3);`
3.0

(g) `iResult = (int) fResult % 3;`
2 (due to 20 % 3)

(h) `bResult = (iResult % 10 == 0) || false && (int) fResult > 0;`
true

8. Consider the following method. Read the javadocs description carefully to get a sense of what the method is *supposed* to do. There are three bugs (one syntax, and two logic) can you locate them?

```
1   /**
2     * Divides one integer by another and returns true if the
3     * result is less than one.
4     * @param a integer to divide
5     * @param b divisor
6     * @return true if a/b is less than one
7     */
8    public boolean divide(int a, b)
9    {
10       boolean result;
11       if (a/b <= 1)
12       {
13           result = true;
14       }
15       else
16       {
17           result = false;
18       }
19       return result;
20   }
```

(a) Syntax bug: parameter `b` needs a data type: `int b`

(b) `a/b` cannot be taken directly, because `b` could be a zero. Fix by throwing an if-statement around it.

(c) `a/b <= 1` should be `a/b < 1`.

Below is a complete Java class definition. Use this definition for the next two questions.

```
1  public class ExampleClass
2  {
3      private int z;
4
5      public ExampleClass(int zValue)
6      {
7          this.z = zValue;
8      }
9
10     /**
11      * Test a value against z
12      * @param n – value to test
13      * @return true if n > z, false otherwise
14      */
15     public boolean testAgainstZ(int n)
16     {
17         if (n > this.z)
18         {
19             return true;
20         }
21         else
22         {
23             return false;
24         }
25     }
26 }
```

(a) In the code on the previous page, label the parts of this class with the appropriate letters. Labels may or may not exist in the given code.

- A. Method signature
- B. Field (instance variable)
- C. Method return type
- D. Method parameter
- E. Assignment statement
- F. Conditional statement
- G. Line comment
- H. Block Comment
- I. Constructor
- J. Default constructor
- K. Boolean statement

(b) What will be the result of this sequence of statements if they are entered in the CodePad?

```
ExampleClass ex1 = new ExampleClass(2);
ex1.testAgainstZ(−3)
```

false. An ExampleClass object is created with z being 2. When you call `ex1.tesetAgainstZ(-3)`, the method will compare -3 with z and return false as a result

(c) Write a separate method called `printMessage` that inputs an integer value `n`, and prints to the terminal `n is greater than z` if that is ther case, and `n is less than equal to z`, otherwise. In both messages, both `n` and `z` must be evaluated. When possible, you should re-use code that's already available to you. Here's an example interaction:

```
ExampleClass ex2 = new ExampleClass(10);
ex2.printMessage(10);
> 10 is less than equal to 10

ex2.printMessage(100);
> 100 is greater than 10
```

The following would do the job:

```
public void printMessage(int x)
{
    if (this.testAgainstZ(x))
    {
        System.out.println(x + " is greater than " + this.z);
    }
    else
    {
        System.out.println(x + " is less than equal to " + this.z);
    }
}
```

9. Below, write a method called `largestOfThree(int a, int b, int c)` that takes three integer arguments. It should print a single line of output that includes all three input values, as well as the largest value of the three. For example, if the user passed in 5, 7, and 1 when calling the method, the output would be "The largest of 5, 7, and 1 is 7." You may write other helper methods, as needed.

   There are lots of good solutions for this one. I chose to create a "helper method" called max() that returns the larger of two given integers. Then I simply call max() between a and b, and then call it again on the result of $max(a, b)$ and c. This will determine the largest of the three integers.

```
/**
 * This method prints the max among the three inputs.
 * @param a an integer
 * @param b an integer
 * @param c an integer
 */
public void largestOfThree(int a, int b, int c)
{
    System.out.println("The largest of " + a + ", " + b + ", " + ", and " + c + ↩
        " is " + this.max(this.max(a,b),c));
}

/**
 * This helper method determines and returns the larger
 * of the two given parameters.
 * @param x an integer
 * @param y another integer
 * @return the max between the two given integers
 */
private int max(int x, int y)
{
    if (x > y) {
        return x;
    }
    return y;
}
```

10. This question asks you to write a Java class representing a water bottle.

    - A water bottle must remember the current amount of water it holds. The maximum amount of water a bottle can hold is 20 ounces.
    - A default constructor is to be implemented. It simply creates an empty water bottle.
    - Write the following methods
        - `clean()`, which inputs and returns nothing. When a user cleans the water bottle, the message ``Scrub, scrub, scrub...'' is printed. When the water bottle is cleaned, it is also emptied to 0 ounces.
        - `fillWaterBottle(...)`, which returns nothing, and inputs an integer `amount`. A user fills a water bottle by adding additional ounces of water. No matter how much water the user adds, the amount of water in the bottle should not exceed the maximum of 20 ounces. Assume that a user will never try to fill using with a negative amount.
        - `drink()`, which inputs and returns nothing. Each drink taken should remove 3 ounces of water from the water bottle. No matter how many gulps the user takes, the amount of water in the bottle should not go below 0 ounces.
        - `getCurrentAmount()`, which takes no inputs, and returns the current amount of water in the bottle.
        - `equals(...)`, which returns a boolean and inputs another water bottle object. To test whether two water bottles are equal in content, you just need to ensure that they currently hold the same amount of water.

    Here is code showing how someone might use a water bottle:

```
// Create a new, empty water bottle
WaterBottle nalgene = new WaterBottle();
System.out.println(nalgene.getCurrentAmount());

// Clean the water bottle
nalgene.cleanWaterBottle();

// Add 50 ounces of water
nalgene.fillBottle(50);
System.out.println(nalgene.getCurrentAmount());

// Take a gulp of water
nalgene.drink();
System.out.println(nalgene.getCurrentAmount());

// Take a gulp of water
nalgene.drink();
System.out.println(nalgene.getCurrentAmount());
```

    If we typed this code into the BlueJ Codepad, we would expect the following to be printed:

```
0.0
Scrub, scrub, scrub...
20.0
17.0
14.0
```

```java
public class WaterBottle
{
    private int currentAmount;

    /**
     * Default constructor creates a WaterBottle object that
     * is empty, and therefore holds no amount of water.
     */
    public WaterBottle()
    {
        this.currentAmount = 0;
    }

    /**
     * Cleans the bottle (and resets the amount of water to zero).
     */
    public void cleanWaterBottle()
    {
        System.out.println("Scrub, scrub, scrub...");
        this.currentAmount = 0;
    }

    /**
     * @return the current amount of water in the bottle
     */
    public int getCurrentAmount()
    {
        return this.currentAmount;
    }

    /**
     * Fills the bottle with the given amount of water.
     * The bottle cannot exceed 20 units of water.
     */
    public void fillBottle(int amount)
    {
        this.currentAmount += amount;    //add the given amount

        //now check whether current amount exceeds 20 units.
        //if it does, then set it to 20.
        if (this.currentAmount > 20)
        {
            this.currentAmount = 20;
        }
    }
```

```
    /**
     * Take one gulp of water (displaces 3 units of water).
     * The bottle cannot hold negative units of water.
     */
    public void drink()
    {
        this.currentAmount -= 3; //a gulp takes 3 units
        if (this.currentAmount < 0)
        {
            this.currentAmount = 0;
        }
    }

    /**
     * Determines if this water bottle is content-equivalent
     * to another water bottle. Two bottles are equal if their current
     * amounts are the same.
     *
     * @param other another water bottle to compare with
     * @return true if the two are equivalent, and false otherwise.
     */
    public boolean equals(WaterBottle other)
    {
        return (this.currentAmount == other.currentAmount);
    }
}
```