CSCI 161 Introduction to Computer Science

Review Guide 1

Topics

- Data Types and Object Types
  - Articulate the differences between classes and objects.
  - Interpret and evaluate various expressions, including its resulting data type, and the use of known operators (`+`, `-`, `*`, `/`, `%`, `&&`, `||`, `!`) under various contexts.
  - Understand how to *type cast*, and why it is sometimes necessary.
  - Understand how data types are represented: the difference between storing an atomic (scalar) values for primitives and *references* for objects.
  - Purpose of the `null` value, and how it impacts object comparison.
  - Know how to print, concatenate, and return Strings.
- Conditionals
  - Be able to evaluate and write boolean expressions of varying complexity.
  - Understand the differences between if, if-then-else, and else-if statements, as well as the context in which you would use them.
  - Object reference equality vs. content equality, and how to test for content equality.
  - Making decisions in code using if-statements
  - Nesting if-statements to hone decisions.
- Code Execution
  - Scope and lifetime of local variables, fields, and input parameters, which determines when you'd choose to use them.
  - Understand flowcharts, and how to model algorithms' execution pattern using flowcharts.
  - Understand the purpose and impact on program execution of the `return` statement.
  - External method calls: Know how to instantiate objects (written in a different class) and call their methods using dot-notation.
  - Internal method calls: Know how to call methods from within the class you're writing.
- Reading/Writing Code
  - Determining the need for fields vs. local variables vs. input parameters when asked to write a class that accomplishes a certain tasks.
  - Know the purpose of, and how to write, constructors (also, default constructor vs. others).
  - Know the purpose of visibility modifiers: private vs. public.

- Designing and writing methods from scratch of varying complexity: deciding their return type (if any) and input parameters (if any).
- Be able to apply modularity and abstraction to class design.
- Given only the documentation of a class, be able to use it.
- Given a high-level English description of a class, be able to write it in Java (like you did for recent homework assignments).
- Know how to write Javadocs-style comments.

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:

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

   TRUE / FALSE
   Explain:

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

4. 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
```

5. 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`
   (b) `y / 3`
   (c) `x / y`
   (d) `x == 5 * 5 + 25`
   (e) `x % 9`
   (f) `y *= 2`

6. 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--;`

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

(c) `iResult = fResult;`

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

(e) `fResult = iResult;`

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

(g) `iResult = (int) fResult % 3;`

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

7. Consider the following method. Read the Javadocs comment 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   }
```

8. 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);
```

(c) Write a separate method called `printMessage` that inputs an integer value `n`, and prints to the terminal `n is greater than z` if $n > z$ 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
```

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.

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
```

Write the class on the next page. Don't forget to implement the `equals()` method to test content equivalence.

(You may detach this page)