Java for Non Majors

Final Study Guide

December 7, 2015

I apologize for the delay in posting this. I could make the study guide only after I made the test. Also, my hometown has seen severe flooding over the last week and I spent the last 3 days frantically trying to locate my family and not doing much else. This is not an excuse, just an explanation. I should have gotten this to you earlier. I'm sorry.

The test consists of

1. 15 multiple choice questions
2. 3 code writing questions
3. 1 code debugging question
4. 4 short answer questions

- You will have an opportunity to earn 20 extra credit points.
- Please try and attempt all questions. You get points for trying.
- The test is cumulative, but focuses on topics introduced after the first midterm.
- The multiple choice questions are the hardest questions on the test.
- Anything from the homeworks / quizzes is fair game.
- Code debugging is mostly syntax based (missing brackets, etc.)
- Making me laugh might gain you points (depends on the quality of the joke).

Topics to study

- Basics of Java Programming. You will be expected to be familiar with Java Syntax, the basic components of a Java program, compiling and running code and the tools involved, etc. Questions will draw from (but not be limited to) the following topics:
  - Java reserved words.
  - Data types, variables and type conversions.
  - Operators and operator precedence.
- Console I/O. Basic input output
  - Printing using the System class methods - println, print and printf.

  - Reading in data from te console using the Scanner class.

- Selection and Repetition.
  - Regular sequential execution of code.
  - Logical operators and short circuit evaluations.
  - Selection statement - if, if-else, if-else ladders, switch structure (including case labels, use of break and continue)
  - Loops - while, do-while and for loops, enhanced for loops, choosing loops most suited for a particular task.

- Java Methods.
  - Method syntax - declaring and defining methods.
  - Modifiers, return types and parameter lists.
  - Pass by value and pass by reference.
  - Method overloading.

- Strings
  - Difference between the String class and the StringBuilder class.
  - String comparison.
  - Using methods of the String class and the StringBuilder class.

- Arrays
  - Declaring, initializing and using arrays of primitive types.
  - Passing arrays as parameters to methods and returning arrays from methods.
  - Multi-dimensional arrays.

- Classes and Objects
  - Creating an object of a pre-existing class and using the available methods.
  - Access Modifiers - public, private and protected.
  - Defining a class - data attributes, constructors, accessor and mutator methods, instance methods.
  - Instantiating a class (creating an object), and using instance methods.
  - Difference between static and instance methods.
  - Passing objects as parameters to methods and returning objects from methods.
  - The "this" keyword.
  - Arrays of objects.

- Inheritance, Interfaces and Polymorphism
  - Concept of base (super) and derived (sub) classes.
  - The "super()" keyword.
  - Method overriding.
  - The class "Object"
  - Abstract Classes
  - The concept of delayed or dynamic binding.

  - Casting classes
  - Interfaces
- Java Exceptions
  - Types of exceptions.
  - Claiming, throwing and catching exceptions.
  - The "finally" keyword.
- GUI programming and Graphics
  - Basic Graphics Classes in Java (Swing and AWT).
  - Events and event driven programming
  - The paint() methods, and the Graphics, Color and Polygon classes.

Sample Questions

1. General Instructions

- The multiple choice questions and short answer questions will pretty much be like the questions on the midterm.
- The code debugging question will also follow the same pattern. You only need to point out the syntax and semantic errors. You don't need to fix typos in String literals, or attempt to fix the logic. Basically, you only need to fix the stuff that the compiler will complain about.
- Some samples for the code writing questions are included in the study guide. Please note that the sample questions are of the same difficulty level as the questions on the test. That is the only similarity between the questions here and those on the test. You will be expected to solve a different problem on the test.
- The Sample Runs provided here and on the test are *samples*. They are just a single run of the program. Your solution should work for all legal inputs. Please do not hard code your solution (think that the sample is the only possible input) to the sample run. The sample is just provided to clarify the requirement. For example, if the sample run says N=4, it does not mean N is 4 all the time. It just happens to be 4 this once. It can be any integer.

2. Code Writing: Loops

Write a Java method called `armstrong` to print all the Armstrong Numbers in a given range. This method accepts 2 integer parameters - `start` and `end` and returns nothing.

An Armstrong Number is a 3 digit number where the sum of the cubes of the individual digits of the number equals the number itself. For example, 371 is an Armstrong Number, since $3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$.

Sample Run: (start = 100, end = 999)

```
The Armstrong Numbers between 100 and 999 are:
153
370
371
407
```

Solution: Please note that only the method is required. You need not write the entire program.

```
public void armstrong ( int start, int end)
{
    System.out.println("The Armstrong Numbers between " + start + " and " +
                       end + " are:");

    //Run a loop from start to end. Check each number
    for(int i=start; i<=end; i++)
    {
        int num = i;
        int sum = 0;

        //Summing the cubes of individual digits.
        while(num!=0)
        {
            int digit = num % 10; // get the digit in the units place
            sum = sum + (int) Math.pow(digit,3); // cube and add to sum
            num = num / 10;  // get rid of the current units place
        }

        //Check if the sum equals the original number
        if (sum == i)
            System.out.println(i);
    }
}
```

3. Code Writing: String Manipulation

Write a Java method called `insert3` that reads a series of strings from the user and prints them after inserting the string "blue" at the third index of the given string. The method accepts a single integer parameter `N` that denotes the number of strings. The function returns nothing. The strings entered by the user will always be at least 4 characters long. You need not test for that.

Sample Run: (N = 4)

```
Enter the 4 strings:
Hello World
Helbluelo World
To infinity and beyond
To blueinfinity and beyond
I Wumbo You Wumbo
I Wblueumbo You Wumbo
Java Programming
Javbluea Programming
```

Solution: Once again, we only need to write the method, not the entire class. You can assume that all the necessary libraries have been imported. You can make this assumption of the test as well.

```
public void insert3 (int N)
{
    Scanner in = new Scanner (System.in);
    System.out.println("Enter the " + N + " strings:");

    //run a loop to read and process the N strings.
    for (int i =0; i < N; i++)
```

```
    {
        //StringBuilder has an available "insert" method. String doesn't.
        //Create the buffer directly using the user input.
        StringBuilder sb1 = new StringBuilder ( in.nextLine());
        sb1.insert(3, "blue");
        System.out.println(sb1);

    }
}
```

4. Code Writing: Inheritance + Array of Objects

Write a Java program with the following requirements.

- Write a class called Drink. The class has an integer attribute called size that holds the size of the drink in ounces. The class also has a parametrized constructor and a method called `print` that prints the size.
- Write a class called `Coffee` that inherits from `Drink`. The class has an integer attribute called numShots that denotes number of espresso shots in the drink. The class also has a parametrized constructor and a method called `print` that overrides the base class method to print both the size and the number of shots.
- Write a class called `Beer` that inherits from `Drink`. The class has a double attribute called percentAlcohol that denotes percentage of alcohol content in the drink. The class also has a parametrized constructor and a method called `print` that overrides the base class method to print both the size and the percentage of alcohol.
- Write a class called `ManyDrinks`. This call just contains the `main` method. In the main method, create an array of Drink of size 4. The first 2 elements are of type Coffee, and the last 2 are Beer. Give the drinks values of your choice. Then invoke the `print` method for each of them.

Sample Run:

```
Drink 1
Size : 16 oz
Number of espresso shots : 2
Drink 2
Size : 20 oz
Number of espresso shots : 4
Drink 3
Size : 12 oz
Aclohol content : 4.2%
Drink 4
Size : 25 oz
Aclohol content : 8.6%
```

Solution: Here, you can assume that the classes have their accessor and mutator methods. You need not write them. This code is fairly self explanatory.

```
//This is the base class / superclass
class Drink
{
    int size;
```

```
    public Drink(int s)
    {
        size = s;
    }

    public void print()
    {
        System.out.println("Size : " + size + " oz");
    }
}

// First Subclass
class Coffee extends Drink
{
    int numShots;

    public Coffee (int s, int n)
    {
        super(s);
        numShots = n;
    }

    public void print()
    {
        super.print();
        System.out.println("Number of espresso shots : " + numShots);
    }
}

//Second Subclass
class Beer extends Drink
{
    double percentAlcohol;

    public Beer ( int s, double p)
    {
        super(s);
        percentAlcohol = p;
    }

    public void print()
    {
        super.print();
        System.out.println("Aclohol content : " + percentAlcohol + "%");
    }
}

public class ManyDrinks
{
    public static void main(String [] args)
    {
        //Create an array of the Drink class of size 4
        Drink [] drinkArray = new Drink[4];
```

```
        // Populate the array as per requirements.
        drinkArray[0] = new Coffee (16, 2);
        drinkArray[1] = new Coffee (20,4);
        drinkArray[2] = new Beer (12, 4.2);
        drinkArray[3] = new Beer (25, 8.6);

        //Call the print method after printing the drink number.
        for (int i=0; i< 4; i++)
        {
            System.out.println("Drink " + (i+1));
            drinkArray[i].print();
        }
    }
}
```