



### 1. Vocabulary Terms – Matching [ 5 pts ]

Select the **best** definition for each of the words below by writing the appropriate letter in the blank beside the word.

1. \_\_\_\_ Variable Shadowing
2. \_\_\_\_ .toString()
3. \_\_\_\_ Inheritance
4. \_\_\_\_ Dynamic Binding
5. \_\_\_\_ Method Overriding

- A. The ability for an instance of a subclass to be treated as an instance of its parent class.
- B. The process by which constructor calls are resolved when **new** is used.
- C. When a subclass redefines a method of the exact same signature as one in the parent class.
- D. When there are multiple methods of the same name, in the same class with different parameter lists.
- E. The process by which method calls are resolved at runtime based upon the type of the actual object present, not the declared type of the reference.
- F. A method which Objects should define to specify how to test for equivalence.
- G. A variable whose value cannot be changed once it is initially set.
- H. The region of the program in which a variable can be “seen” by the compiler.
- I. When two variables are in scope at the same time and the more local one hides the more global one.
- J. The process by which a constructor inherits initialization lists from the constructor in a parent class.
- K. Used when one type is a subtype of another. Functionality of the supertype is in the subtype, unless changed by the subtype.
- L. A method which Objects should define to specify how they should be represented as String.
- M. When one constructor calls another using **this()**.

**2. Iteration [ 10 pts ]**

Write the method **public void printStars(int num)**. Which prints a right triangle of Stars of base and height **num**. For example, if this method were called with a parameter of 5, the output should be

```
*  
**  
***  
****  
*****
```

If this method were called with 0 as a parameter, no output would be printed. You must use iteration **ONLY** for this method. If you do not use iteration, or you use any recursion, you will receive no credit for this method. You may assume that **num** will be greater than or equal to 0.

### 3. Multidimensional Array – Coding [ 15 pts ]

- 9 (a) Write the method `public int product(int[][][] data)`. This method should return the product of all items in the array `data`. You may not assume that the array is cubical or rectangular, it may have uneven lengths in any given dimension. You may assume that the array is not *null* and that none of the array references in it are *null*.

```
public int product(int[][][] data){
```

```
}
```

- 3 (b) Declare a variable `myArray` to be a 3 dimensional array of doubles, and then initialize it to be a 5 by 7 by 9 array of doubles.

- 3 (c) What is the data type of `myArray[1]`?

4. **.equals()** and **toString()** [ 20 pts ]

Write a class **Box** which has the following:

- a public int **length**
- a public int **width**
- a public String **color**
- a **.equals** method.
- a **toString()** method.

The **.equals** method should behave as follows:

- if the item passed in is not a **Box**, return **false**.
- if the **Box** passed in has the same **length**, **width**, and the **colors** match, then return **true**, else return **false**.

The **toString()** method should yield a String of the following format:

*A **color** Box that is **length** by **width**.*

For example

*A Red Box that is 3 by 5.*

You do NOT need to write accessors and modifiers for your variables, or any constructors.



## 6. Inheritance – Coding [ pts ]

Given the following code: Write the class **Circle** as follows:

- **Circle** is a subclass of **Shape**
- **Circle** has a double for **radius**
- **Circle** has an accessor for **radius**
- **Circle** has a constructor that takes a String for **color** and a double for **radius**. It initializes **color** via the parent class constructor. It then initializes **radius**.
- **Circle** has a constructor that takes only a String for **color**. It chains to the two parameter constructor with a size of 1.

Below, write the code for class **Circle** as described above:

## 7. Strings – Coding [ 15 pts ]

Write the method `public String replaceOccurrencesOf(String s, char from, char to)`. This method should return a String that is the same as `s`, except that all occurrences of the character `from`, are replaced by the character `to`. You **MUST** use recursion only for this method. If you do not use recursion, or if you use any iteration, you will receive no credit for this method. You may write any helper methods that you wish to. You may assume that `s` is not *null*. Below is an exclusive list of the methods from the String class that you may use. This means that you may use these methods from String, but no other methods from the String class.

1. `public char charAt(int index)`
2. `public String substring(int beginIndex)`
3. `public boolean equals(Object o)`

Here are some examples of what should be returned:

```
replaceOccurrencesOf("", 'x', 'y') = ""
```

```
replaceOccurrencesOf("abcabcdefa", 'a', 'q') = "qbcqbcdefq"
```

```
replaceOccurrencesOf("abcdefg", 'q', 'a') = "abcdefg"
```

```
public String replaceOccurrencesOf(String s, char from, char to) {
```

```
}
```

**8. Constructor Chaining – Tracing [ 15 pts ]**

Given the following code: Write the output when the **main** method is run.