CS 102: Section 02

Data Structures and Algorithms

Spring 2017 Midterm Solutions

1. (10 points) Short answer questions:

   (a) (1 point) What is the last index number in an instantiated array of n elements?
   $(n-1)$

   (b) (2 points) In Java, classes have 2 fundamentally different purposes. Name or describe each of these purposes.

      1. Library of static methods.
      2. Template for creation of objects.

   (c) (2 points) Name two different **classes** we have used for writing output. **System, JOptionPane, or Scanner**

   (d) (4 points) Give one example of the possible outputs for each of the following 2 code segments (assume they are each part of the main method in 2 different classes) and explain the purpose of each segment at areas indicated below:

```
1.  int x = 0, y = 0;
2.  do {
3.         x = (int)(10*Math.random() + 1);
4.         y = (int)(10*Math.random() + 1);
5.     } while (x != y);
6.     System.out.println(x + "  " + y);

Example output:  6  6

Purpose: Generate random numbers between 1 and 20 until the 2
         numbers generated are equal.
```

```
1.   int x,y;
2.   x = y = 0;
3.   while (x == y) {
4.         x = (int)(10*Math.random() + 1);
5.         y = (int)(10*Math.random() + 1);
6.    }
7     System.out.println(x + "  " + y);

Example output:   8  5

Purpose: Generate random numbers between 1 and 20 until the 2
         numbers generated are not equal.
```

2. (5 points) The following question concerns the class definition shown below:

```
 1. public class RandomArray {
 2.   public static void main(String[] args) {
 3.        int size = 3;
 4.        int[] inches = new int[size];
 5.        try{
 6.           for (int i = 0; i <= size; i++){
 7.                inches[i] = (int)(20 * Math.random() + 1);
 8.                System.out.println(""+inches[i]+" inches of rain.");
 9.            }
10.         }catch(ArrayIndexOutOfBoundsException aio) {
11.             System.out.println("Inside catch block.");
12.        }
13.        System.out.println("All's well.");
14.     }
15.  }
```

Circle the leftmost and rightmost output examples below. The second output from the left has the generation of number 24 when the expression in line 7 and generate only numbers between 1...20. The third output from the left ignores the fact that trying to access inches[3] would throw an ArrayIndexOutOfBoundsException.

From the 4 sequences of output below, circle the possible outputs of this code and explain why the non-circled outputs are not possible output:

```
19 inches of rain.      14 inches of rain.      18 inches of rain.      12 inches of rain.
8 inches of rain.       24 inches of rain.      5 inches of rain.       10 inches of rain.
6 inches of rain.       2  inches of rain.      10 inches of rain.      11 inches of rain.
Inside catch block.     Inside catch block.     All's well.             Inside catch block.
All's well.             All's well.                                     All's well.
```

3. (10 points) Consider the code for class `DigitSum` given below:

```
 1. import java.util.*;
 2. public class DigitSum{
 3.    public static void main(String[] args) {
 4.       Scanner scan = new Scanner(System.in);
 5.       System.out.println("This program sums the digits in an integer.");
 6.       System.out.println("Enter a positive integer: ");
 7.       int n = scan.nextInt();
 8.       int nSave = n;
 9.       int digitSum = 0;
10.       while (n > 0) {
11.           digitSum += n % 10;
12.           n /= 10;
13.       }  // end while
14.       System.out.println("The sum of the digits in " +
                             nSave + " is " + digitSum);
15.     } // end main  } // end class

/* Sample output */
This program sums the digits in an integer.
Enter a positive integer: 161
The sum of the digits in 161 is 8.
```

Using `DigitSum` as a model, fill in the blank part of the class named `DigitCount` that counts the number of digits in the number the user enters. You need only fill in the body of the `main`

method shown below. Sample output is given below the class definition. If any of the lines in DigitSum can be used verbatim in DigitCount, write "Same" to the right of each line that is the same as it is in DigitSum. For example, "Same" is written to the right of line number 4 below. Lines 1 and 3 should also be marked "Same", but they are given for clarity. You may also choose to write every line.

```
 1. import java.util.*;
 2. public class DigitCount{
 3.    public static void main(String[] args) {
 4.       Scanner scan = new Scanner(System.in);
 5.       System.out.println("This program counts the number of digits in an integer.");
 6.       System.out.println("Enter a positive integer: ");
 7.       int n = scan.nextInt();
 8.       int nSave = n;
 9.       int digitCount = 0;
10.       while (n > 0) {
11.           digitCount++;
12.           n /= 10;
13.       }  // end while
14.       System.out.println("The number of digits in " +
                             nSave + " is " + digitCount);
15.     } // end main  } // end class
 }}
/* Sample output */
This program counts the number of digits in an integer.

Enter a positive integer: 1861
The number of digits in 1861 is 4
```

Alternate solution (not using DigitSum) by Frank Swiatowicz:

```
 1. import javax.swing.*;
 2. public class DigitCount{
 3.    public static void main(String[] args) {
 4.       JOptionPane.showMessageDialog(null,"This program counts the number of digits in an integer.");
 5.       String nstr = JOptionPane.showInputDialog("Enter a positive integer: ");
 6.       JOptionPane.showMessageDialog(null, "The number of digits in " +
                                              nstr + " is " + nstr.length());
 7.     } // end main
 8. } // end class

/* Sample output */
This program counts the number of digits in an integer.

Enter a positive integer: 1861
The number of digits in 1861 is 4
```

4. (5 points) Consider the code given below.

```
public interface ILoS {
    public int length();
}

public class MTLoS implements ILoS {
    public int length() {
       return 0;
    }
}
```

```
public class ConsLoS implements ILoS {
    private String first;
    private ILoS rest;

    public ConsLoS(String f, ILoS r) {
          first = f;
          rest = r;
     }

     public int length() {
         return 1 + rest.length();
     }
}
```

```
public class TestILoS{
    public static void main(String[] args) {
        ILoS myLoS =
               new ConsLoS ("This ",
                   new ConsLoS ("is ",
                       new ConsLoS("a ",
                           new ConsLoS("list ",
                               new ConsLoS("of ",
                                   new ConsLoS("words.", new MTLoS()))))));
        System.out.println(myLoS.countChars());
    }
}
```

Suppose the ILoS class contained the following method signature:

```
public int countChars();
```

Write the methods that must be included in the the MTLoS class and the ConsLoS class for the countChars() method. This method is called on an ILoS and it produces the int that is the tally of the characters in all Strings in the list. For example, the output from running TestILoS is the integer 24.

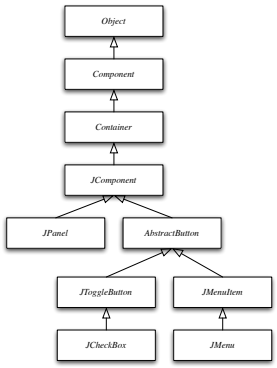
MTLoS:

```
public int countChars() {
    return 0;
}
```

ConsLoS:

```
public int countChars() {
    return first.length() + rest.countChars();
}
```

5. (10 points) Refer to the Java Inheritance hierarchy shown below to answer the following questions.



(a) (3 points) List or mark all classes in the hierarchy in which a method to be called on a *JCheckBox* object could be defined. *Explain your answer.*
JCheckBox, JToggleButton, AbstractButton, JComponent, Collection, Component, Object. A method called in JCheckBox could be defined in any ancestor class of JCheckbox, or in JCheckBox itself.

(b) (3 points) List or mark all the classes that inherit code from class *JComponent*. **JCheckBox, JToggleButton, JMenu, JMenuItem, AbstractButton, or JPanel. Any descendant class inherits code from any ancestor.**

(c) (4 points) Suppose you were asked to implement the *JCheckBox* and *JMenu* classes as Java programs. Write class header lines (signatures) for each of the *JCheckBox* and *JMenu* classes. Be sure that each class header reflects the inheritance relationship shown in the figure above.

```
public class JCheckBox extends JToggleButton {}
public class JMenu extends JMenuItem {}
```

6. (25 points) Short answer and True/False. Explaining your answers may factor into the correctness.

(a) (2 points) What are *instance variables* and *instance methods*? What distinguishes instance members from non-instance members inside a class definition?

Instance variables are global variables declared with a class that are non-static. Instance methods are non-static methods defined within a class.

(b) (2 points) How are *instance methods* called from within the main method of the same class?

First, an object of the class type must be declared and instantiated and then an instance method can be called on the object just created.

(c) (2 points) What is the purpose of a constructor? Write a constructor signature for a class called `PersonData` that has 2 instance variables: a String **name** and an int **age**.

The purpose of a constructor is to instantiate or initialize all instance variables in the class.

```
public PersonData(String name, int age) {
    this.name = name;
    this.age = age;
}
```

(d) (1 point) What is the purpose of the keyword *new*?

The keyword *new* is used to create new objects from a class template.

(e) (2 points) List two keywords that could be used to create inheritance ("is-a") relationships between files x.java and y.java.

extends and implements

(f) (2 points) What is an *interface*?

An interface is a collection of method stubs and final constant data fields. Interfaces dictate all methods that must be fully coded in subtypes (classes that implement the interface.

(g) (2 points) Suppose you came across a class with the header

```
public class Car implements Vehicle
```

Circle the statements in the group of 4 below that would create a new object of type Vehicle.

```
Vehicle myCar = new Vehicle();

Car myExtraCar = new Car();

Car myOtherCar = new Vehicle();

Vehicle anotherCar = new Car();
```

Circle choice 2 and 4. Choices 1 and 3 are not valid because an interface type like Vehicle cannot be instantiated. We know from the line "public class Car implements Vehicle" that Vehicle is an interface.

(h) (1 point) True or False? When passing parameters, the name of the argument must match the name of the parameter.
False.

(i) (1 point) True or False? The package java.lang must be imported in every java class you write.
False. java.lang is automatically imported into every java program.

(j) (1 point) True or False? Every package p besides java.lang must be imported in order to use the classes defined in package p.
False. You can also refer to any methods from a package using the "fully qualified name", e.g., for Scanner, you could write java.util.Scanner everywhere a Scanner method is used.

(k) (1 point) True or False? When passing parameters, the type of the argument must match the type of the parameter.
True

(l) (1 point) True or False? Syntax errors always occur during runtime.
False. Syntax errors occur during compile-time.

(m) (2 points) Name 2 keywords that may be used to avoid program failure when exceptions occur during runtime.
try, catch, throws, throw

(n) (5 points) Consider the following boolean expression, where variable `y` is a positive `int`:

```
(y % 2 ==  0) && (y % 7 == 0)
```

For what value(s) of `y` does this expression evaluate to `false`?
Odd numbers and even numbers that are not evenly divisible by 7.

For what value(s) of `y` does this expression evaluate to `true`?
Even numbers that are evenly divisible by 7.

For what value(s) of `y` does this expression short-circuit?
Any odd number.