

1 For each of the following pairs, which represents a class and which represents an object of that class?

| | Class | Object |
|---------------------|-----------|-----------|
| Superhero, Superman | Superhero | Superman |
| Justin, Person | Person | Justin |
| Rover, Pet | Pet | Rover |
| Magazine, Time | Magazine | Time |
| Christmas, Holiday | Holiday | Christmas |
| | | |

2 Write a method called *multiConcat* that takes a String and an integer as parameters. Return a String that consists of the string parameter concatenated with itself count times, where count is the integer parameter. For example, if the parameter values are "hi" and 4, the return value is "hihihihi". Return the original string if the integer parameter is less than 2.

```

/*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 01/30/2015
//
//Write a method called multiConcat that takes a String and an
//integer as parameters. Return a String that consists of the
//string parameter concatenated with itself count times, where
//count is the integer parameter. For example, if the parameter
//values are "hi" and 4, the return value is "hihihihi". Return
//the original string if the integer parameter is
//less than 2.
//
//
//*****/

public class TestString //The .java to submit to javac
{

    public static void main(String[] args) //The main method
    {
        String text = multiConcat("hi",4); //Declare a variable
                                           //and assign it to the call

        System.out.printf("%s\n",text); //Print out the result
    } //End of main entrance
}
```

```

public static String multiConcat(String s, int width) //multiConcat
//method
{
    String NewString=""; //Use NewString because s is immutable

    if(width < 2) //if cont is less than 2
    {
        return(s); //return the original string
    }//End if
    else //Otherwise
    {
        for(int i = 0; i < width; i++) //Set up the loop from 0 to
        //count - 1
        {
            NewString = NewString + s; //add the same string (i-
            //1) times
        }

        return(NewString); //give it back to caller
    }//End else
} //End multiConcat

} //End .java class to submit to javac

```

3 Overload the *multiConcat* method from Exercise 5.25 such that if the integer parameter is not provided, the method returns the string concatenated with itself. For example, if the parameter is "test", the return value is "testtest".

```

//*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 01/30/2015
//
//Overload the multiConcat method from Exercise 5.25 such that
//if the integer parameter is not provided, the method returns
//the string concatenated with itself. For example, if the
//parameter is "test", the return value is "testtest".
//
//
//*****

public class TestString //The .java to submit to javac
{
    public static void main(String[] args) //The main method
    {
        String text = multiConcat("hi",4); //Declare a variable
    }
}

```

```

//and assign it to the call
String text2 = multiConcat("test"); //Declare a variable
//and assign it to the call
//test2 tests out the overloading feature of java compiler

System.out.printf("%s\n",text); //Print out the result
System.out.printf("%s\n",text2); //Print out the result
} //End of main entrance

public static String multiConcat(String s, int width) //multiConcat
//method
{
    String NewString=""; //Use NewString because s is immutable

    if(width < 2) //if cont is less than 2
    {
        return(s); //return the original string
    } //End if
    else //Otherwise
    {
        for(int i = 0; i < width; i++) //Set up the loop from 0 to
        //count - 1
        {
            NewString = NewString + s; //add the same string (i-
            //1) times
        }

        return(NewString); //give it back to caller
    } //End else
} //End multiConcat

public static String multiConcat(String s) //multiConcat overloaded
{
    String NewString=""; //Use NewString because s is immutable
    for(int i = 0; i < 2; i++) //Set up the loop from 0 to count - 1
    {
        NewString = NewString + s; //add the same string (i-
        //1) times
    }

    return(NewString); //give it back to caller

} //End multiConcat

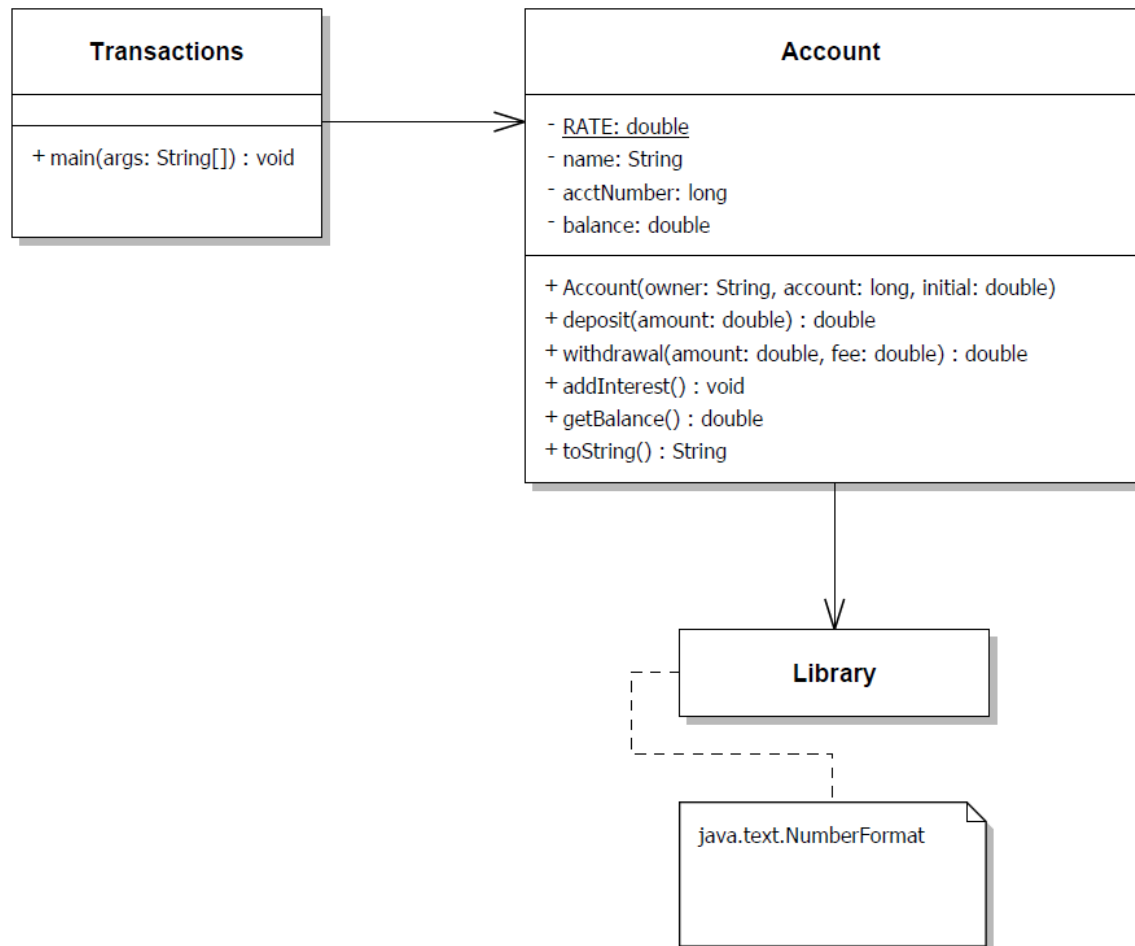
} //End .java class to submit to javac

```

4 ***Explain why a static method cannot refer to an instance variable***

Static methods cannot reference instance variables because instance variables don't exist until an object exists.

4.3 Draw a UML class diagram for the Transactions program.



5 Which of the following are valid declarations? Which instantiate an array object? Explain your answers.

```
int primes = {2, 3, 4, 5, 7, 11};
```

Not valid. Square brackets are needed. Additionally, a list has multiple elements that must be declared as an array. Therefore, the declaration above is invalid.

```
float elapsedTimes[] = {11.47, 12.04, 11.72, 13.88};
```

This declaration is valid. One may also declare it as `float[] elapsedTimes = {11.47, 12.04, 11.72, 13.88};`. An array object of type float is created as a result.

```
int[] scores = int[30];
```

*This declaration is invalid and therefore no object will be created. To make it valid, add the keyword **new** before `int[30]` or an initializer list but not both. For example, `int scores[] = new int[30];` or `int[] scores = {2, 3, 4, 5, 6,...,17};`*

```
int[] primes = new {2, 3, 5, 7, 11};
```

*This declaration is invalid because it has the new keyword **and** the initializer list. To make it valid, remove the keyword **new**, such as follows: `int[] primes = {2, 3, 5, 7, 11};`*

```
int[] scores = new int[30];
```

This declaration is valid and it will create an array object of type `int` as a result.

```
char grades[] = {'a', 'b', 'c', 'd', 'f'};
```

This declaration is valid and will create an array object of type `char`. The declaration is also valid as follows: `char[] grades = {'a', 'b', 'c', 'd', 'f'};`

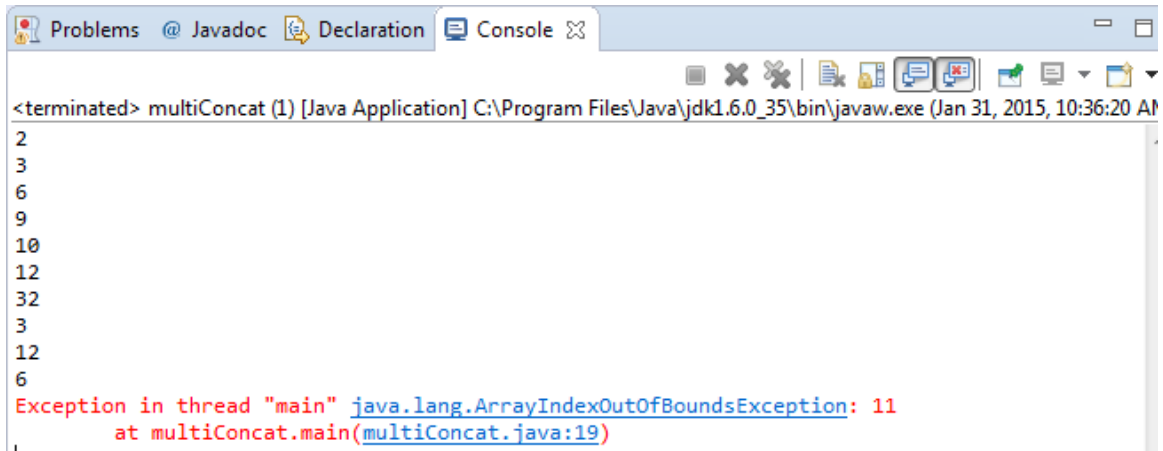
```
char[] grades = new char[];
```

This declaration is invalid because the size of the array is missing. To make it work, try as follows: `char[] grades = new char[5];`

6 Describe what problem occurs in the following code. What modifications should be made to it to eliminate the problem?

```
int[] numbers = {3, 2, 3, 6, 9, 10, 12, 32, 3, 12, 6};  
for (int count = 1; count <= numbers.length; count++)  
    System.out.println (numbers[count]);
```

The out threw an array out of bound exception error at run time. As shown below:



```
<terminated> multiConcat (1) [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 10:36:20 AM)
2
3
6
9
10
12
32
3
12
6
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 11
    at multiConcat.main(multiConcat.java:19)
```

The compiler managed to compile the code without any issue, but the problem was that the loop went to index 12, adding one more element to the array, which was outside of the 0-11 range. For loop is 0th base indexing. To fix this problem, use the **for each** loop, as shown below:

```
int[] numbers = {3, 2, 3, 6, 9, 10, 12, 32, 3, 12, 6};
for (int count : numbers)
    System.out.println (count);
```

Which yielded a correct result, as shown below:



```
<terminated> multiConcat (1) [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 10:42:50 AM)
3
2
3
6
9
10
12
32
3
12
6
HelloHelloHelloHelloHelloHello
JavaJava
```

7 Write an array declaration and any necessary supporting classes to represent the following statements:

- students' names for a class of 25 students

```
String[] students = new String[25];
```

- b. students' test grades for a class of 40 students

```
char[] grades = new char[40];
```

- c. credit-card transactions that contain a transaction number, a merchant name, and a charge

```
final int MAXIMUM = 10;
```

```
transactions[] charges = new transactions(172015, "Hieu Pham",  
1.85); //Creates and initialies
```

```
public class transactions  
{  
    private int transactionNumber;  
    private String merchantName;  
    private double charge;  
    //Constructor  
    public transaction(int tn, String mn, double chrg)  
    {  
        transactionNumber = tn;  
        merchantName = mn;  
        charge = chrg;  
    }  
  
    //may be a mutator here...  
    //may be an accessor here...  
}
```

- d. students' names for a class and homework grades for each student

```
final int MAXIMUM = 100;
```

```
student[] CST200 = new student[MAXIMUM]; //Create 100 students
```

```
public class student  
{  
    private String name;  
    private String postID;  
    private char[] scores;  
    //Constructor
```

```

    public student(String Name, String pID, char[] Scores)
    {
        name = Name;
        postID = pID;
        scores = Scores;
    }

    //may be a mutator here...
    //may be an accessor here...
}

```

- e. for each employee of the L&L International Corporation: the employee number, hire date, and the amount of the last five raises

```

double raises[] = {0.75, 1.15, 0.85, 1.12, 2.05};

employees MyEmployees = new employees(000,"02/27/2012", raises);

public static class employees
{
    private int EmployeeID;
    private String StartDate;
    private double PerfRaise[] = new double[5];

    public employees(int EID, String DOH, double Raise[])
    {
        int iterator = 0;

        EmployeeID = EID;
        StartDate = DOH;

        for(double aNumber : Raise)
        {
            PerfRaise[iterator] = aNumber;
            iterator++;
        }

        //Close constructor

        public String toString()
        {
            return(EmployeeID + "\n"
                + StartDate + "\n"
                + PerfRaise[0] + "\n"
                + PerfRaise[1] + "\n"
                + PerfRaise[2] + "\n"
                + PerfRaise[3] + "\n"
                + PerfRaise[4] + "\n");

            //Close the string
        }
    }
}
//Close class employees

```


8 **Revise the *Coin* class such that its state is represented internally using a boolean variable. Test the new versions of the class as part of the *CountFlips* and *FlipRace* programs.**

The revised Coin class

```

/*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 01/31/2015
//
//Revise the Coin class such that its state is represented
//internally using a boolean variable. Test the new versions
//of the class as part of the CountFlips and FlipRace
//programs.
//
*****/

public class Coin
{
    //private final int HEADS = 0; // tails is 1
    // HEADS is not used at after the mod,
    //so I commented it out.

    private boolean face; //Revised right here, HQP 01/31/2015
    //Changed from int to boolean type.

    //-----
    // Sets up this coin by flipping it initially.
    //-----
    public Coin()
    {
        flip();
    }

    //-----
    // Flips this coin by randomly choosing a face value.
    //-----
    public void flip ()
    {
        face = ((int)(Math.random() * 2) != 0);
        //Forced/type-casting into a boolean evaluation here.
        //HQP, 01/31/2015
    }

    //-----
    // Returns true if the current face of this coin is heads.
    //-----
    public boolean isHeads()
    {
        return(face); //Changed this to return the state
        //the boolean value of face, HQP 01/31/2015
    }
}
```

```

    }

    //-----
    // Returns the current face of this coin as a string.
    //-----
    public String toString()
    {
        return (face != true) ? "Heads" : "Tails";
        //Changed to judge face based on its boolean state
        //HEADS = 0, and 0 is not true
        //HQP, 01/21/2015
    }
}

```

The revised CountFlips program

```

//-----
// Used Lewis' CountFlips program with minimal modification
//-----
public class CountFlips
{
    public static void main (String[] args)
    {
        final int FLIPS = 1000;
        int heads = 0, tails = 0;

        Coin myCoin = new Coin();

        for (int count=1; count <= FLIPS; count++)
        {
            myCoin.flip();

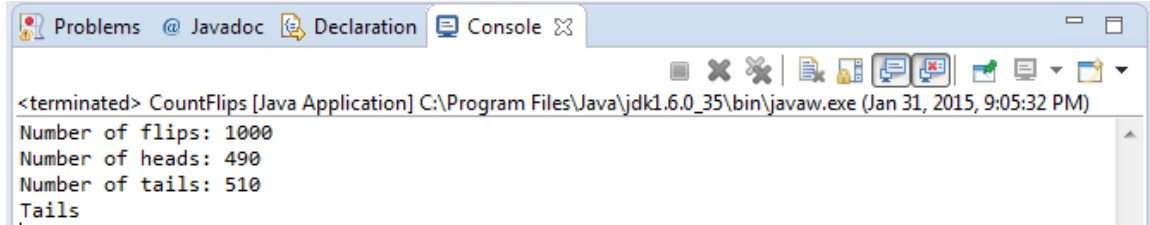
            if (myCoin.isHeads())
                heads++;
            else
                tails++;
        }

        System.out.println ("Number of flips: " + FLIPS);
        System.out.println ("Number of heads: " + heads);
        System.out.println ("Number of tails: " + tails);
        //Added println() below to print out myCoin object
        //and its boolean state, HQP, 01/31/2015

        System.out.println (myCoin);
    }
}

```

Output of CountFlips:



The screenshot shows a Java IDE window with the 'Console' tab selected. The console output is as follows:

```
<terminated> CountFlips [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 9:05:32 PM)
Number of flips: 1000
Number of heads: 490
Number of tails: 510
Tails
```

The FlipRace program

```
//-----
//  Used Lewis' FlipRace program with minimal modification
//-----
public class FlipRace
{
    public static void main (String[] args)
    {
        final int GOAL = 3;
        int count1 = 0, count2 = 0;
        Coin coin1 = new Coin(), coin2 = new Coin();

        while (count1 < GOAL && count2 < GOAL)
        {
            coin1.flip();
            coin2.flip();

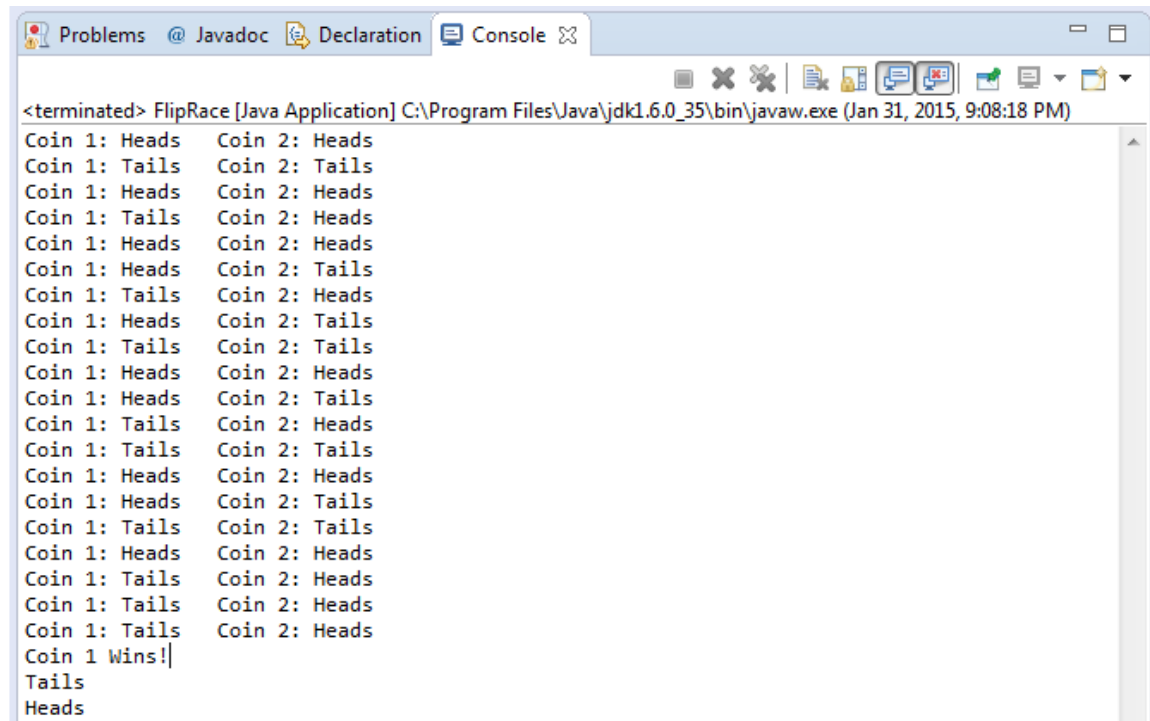
            System.out.println ("Coin 1: " + coin1 + "\tCoin 2: " + coin2);

            //Increment or reset the counters
            count1 = (coin1.isHeads()) ? count1+1 : 0;
            count2 = (coin2.isHeads()) ? count2+1 : 0;
        }

        if (count1 < GOAL)
        {
            System.out.println ("Coin 2 Wins!");
        }
        else
        {
            if (count2 < GOAL)
                System.out.println ("Coin 1 Wins!");
            else
                System.out.println ("It's a TIE!");
        }
        //Revised to group the if/else block logically
        //so I can do the println() lines below

        System.out.println(coin1);
        System.out.println(coin2);
    }
}
```

Output of FlipRace:



```
<terminated> FlipRace [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 9:08:18 PM)
Coin 1: Heads   Coin 2: Heads
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Tails   Coin 2: Heads
Coin 1: Heads   Coin 2: Heads
Coin 1: Heads   Coin 2: Tails
Coin 1: Tails   Coin 2: Heads
Coin 1: Heads   Coin 2: Tails
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Heads   Coin 2: Tails
Coin 1: Tails   Coin 2: Heads
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Heads   Coin 2: Tails
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Heads   Coin 2: Tails
Coin 1: Tails   Coin 2: Heads
Coin 1: Tails   Coin 2: Heads
Coin 1: Tails   Coin 2: Heads
Coin 1 Wins!
Tails
Heads
```

9 Repeat programming project in item 8 above, representing the state of the coin using an *enumerated* type.

The revised Coin class

```
/**
 * *****
 * //Author: Hieu Pham
 * //ID: 0953-827
 * //Section: 28317
 * //Date: 01/31/2015
 * //
 * //Revise the Coin class such that its state is represented
 * //internally using an enum variable. Test the new versions
 * //of the class as part of the CountFlips and FlipRace
 * //programs.
 * //
 * *****
 */

enum coinstate{head, tail} //Must have

public class Coin
{
    //private final int HEADS = 0; // tails is 1
    // HEADS is not used at after the mod,
    //so I commented it out.
```

```

private boolean flips;
private coinstate face;
//Revised right here, HQP 01/31/2015
//Changed from int to boolean type.

//-----
// Sets up this coin by flipping it initially.
//-----
public Coin()
{
    flip();
}

//-----
// Flips this coin by randomly choosing a face value.
//-----
public void flip ()
{

    flips = ((int)(Math.random() * 2) != 0);

    //Forced/type-casting into a boolean evaluation here.
    //HQP, 01/31/2015
}

//-----
// Returns head if the current face of this coin is heads.
//-----
public coinstate isHeads()
{
    if(flips == true)
    {
        face = coinstate.tail;
    }
    else
    {
        face = coinstate.head;
    }
    return(face); //Changed this to return the state
    //the boolean value of face, HQP 01/31/2015
}

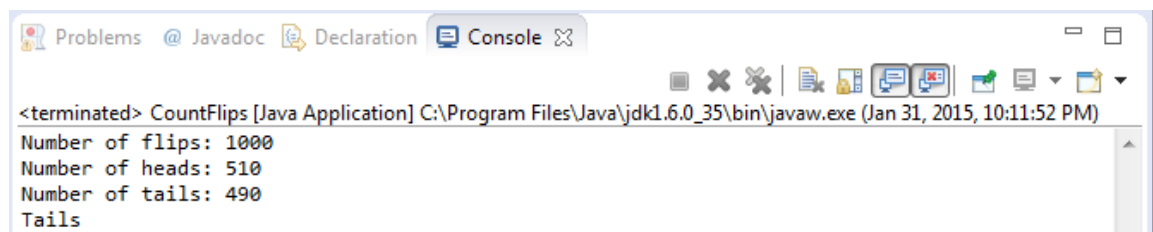
//-----
// Returns the current face of this coin as a string.
//-----
public String toString()
{
    return (face == coinstate.head) ? "Heads" : "Tails";
    //Changed to judge face based on its boolean state
    //HEADS = 0, and 0 is not true
    //HQP, 01/21/2015
}
}

```

The revised CountFlips program

```
//-----  
//  Used Lewis' CountFlips program with minimal modification  
//-----  
  
public class CountFlips  
{  
    public static void main (String[] args)  
    {  
        final int FLIPS = 1000;  
        int heads = 0, tails = 0;  
  
        Coin myCoin = new Coin();  
  
        for (int count=1; count <= FLIPS; count++)  
        {  
            myCoin.flip();  
  
            if (myCoin.isHeads() == coinstate.head)  
                heads++;  
            else  
                tails++;  
        }  
  
        System.out.println ("Number of flips: " + FLIPS);  
        System.out.println ("Number of heads: " + heads);  
        System.out.println ("Number of tails: " + tails);  
        //Added println() below to print out myCoin object  
        //and its boolean state, HQP, 01/31/2015  
  
        System.out.println (myCoin);  
    }  
}
```

Output of CountFlips:



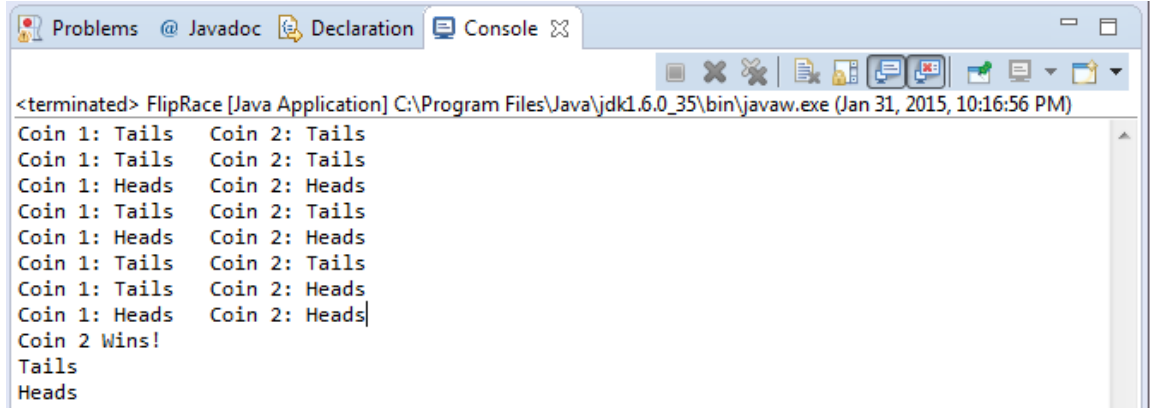
The screenshot shows a Java IDE window with the 'Console' tab selected. The console output displays the results of the CountFlips program execution. The output is as follows:

```
<terminated> CountFlips [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 10:11:52 PM)  
Number of flips: 1000  
Number of heads: 510  
Number of tails: 490  
Tails
```

The revised FlipRace program

```
//-----  
//  Used Lewis' FlipRace program with minimal modification  
//-----  
public class FlipRace  
{  
    public static void main (String[] args)  
    {  
        final int GOAL = 3;  
        int count1 = 0, count2 = 0;  
        Coin coin1 = new Coin(), coin2 = new Coin();  
  
        while (count1 < GOAL && count2 < GOAL)  
        {  
            coin1.flip();  
            coin2.flip();  
  
            System.out.println ("Coin 1: " + coin1 + "\tCoin 2: " + coin2);  
  
            //Increment or reset the counters  
            count1 = (coin1.isHeads() == coinstate.head) ? count1+1 : 0;  
            count2 = (coin2.isHeads() == coinstate.head) ? count2+1 : 0;  
        }  
  
        if (count1 < GOAL)  
        {  
            System.out.println ("Coin 2 Wins!");  
        }  
        else  
        {  
            if (count2 < GOAL)  
                System.out.println ("Coin 1 Wins!");  
            else  
                System.out.println ("It's a TIE!");  
        }  
        //Revised to group the if/else block logically  
        //so I can do the println() lines below  
  
        System.out.println(coin1);  
        System.out.println(coin2);  
    }  
}
```

Output of FlipRace:



```
<terminated> FlipRace [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Jan 31, 2015, 10:16:56 PM)
Coin 1: Tails   Coin 2: Tails
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Tails   Coin 2: Tails
Coin 1: Heads   Coin 2: Heads
Coin 1: Tails   Coin 2: Tails
Coin 1: Tails   Coin 2: Heads
Coin 1: Heads   Coin 2: Heads
Coin 2 Wins!
Tails
Heads
```

10 Design and implement a class called *Sphere* that contains instance data that represents the sphere's diameter. Define the *Sphere* constructor to accept and initialize the diameter, and include getter and setter methods for the diameter. Include methods that calculate and return the volume and surface area of the sphere (see programming project 3.5 for the formulas). Include a *toString* method that returns a oneline description of the sphere. Create a driver class called *MultiSphere*, whose main method instantiates and updates several *Sphere* objects.

The Sphere class

```
/******
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Design and implement a class called Sphere that contains
//instance data that represents the sphere's diameter.
//Define the Sphere constructor to accept and initialize
//the diameter, and include getter and setter methods for
//the diameter. Include methods that calculate and return
//the volume and surface area of the sphere
//(see programming project 3.5 for the formulas).
//Include a toString method that returns a oneline
//description of the sphere.
//
//*****
```



```

import java.text.DecimalFormat;

public class Sphere
{
    private double radius;
    private double area;
    private double volume;

    public Sphere(double rad)
    {
        radius = rad;
    }

    public void setArea(double rad)
    {
        radius = rad; //Always update current values
        area = (4.0 * Math.PI * Math.pow(radius, 2.0));
    }

    public double getArea()
    {
        return(area);
    }

    public void setVolume(double rad)
    {
        radius = rad; //Always update current values
        volume = ((4.0 / 3.0) * Math.PI * Math.pow(radius, 3.0));
    }

    public double getVolume()
    {
        return(volume);
    }

    public void Spherify()
    {
        area = (4.0 * Math.PI * Math.pow(radius, 2.0));
        volume = ((4.0 / 3.0) * Math.PI * Math.pow(radius, 3.0));
    }

    public String toString()
    {
        DecimalFormat fmt = new DecimalFormat("0.###");

        return("Radius = " + fmt.format(radius) + ", "
            + "Area = " + fmt.format(area) + ", "
            + "Volume = " + fmt.format(volume) + "\n");
    }
}

```

The MultiSphere driver class

```

/*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Create a driver class called MultiSphere,
//whose main method instantiates and updates
//several Sphere objects.
//
*****/

```

```

import java.text.DecimalFormat;

public class MultiSphere
{
    public static void main(String[] args)
    {
        Sphere sphere1 = new Sphere(2.5), sphere2 = new Sphere(4.7);
        Sphere sphere3 = new Sphere(7.93), sphere4 = new Sphere(11.56);

        DecimalFormat fmt = new DecimalFormat("0.###");

        sphere1.setArea(1.19);
        sphere2.setArea(4.16);
        sphere3.setArea(2.29);
        sphere4.setArea(3.77);

        System.out.println();
        System.out.println("Area of sphere #1 " +
            fmt.format(sphere1.getArea()));
        System.out.println("Area of sphere #2 " +
            fmt.format(sphere2.getArea()));
        System.out.println("Area of sphere #3 " +
            fmt.format(sphere3.getArea()));
        System.out.println("Area of sphere #4 " +
            fmt.format(sphere4.getArea()));
        System.out.println();

        sphere1.setVolume(1.19);
        sphere2.setVolume(4.16);
        sphere3.setVolume(2.29);
        sphere4.setVolume(3.77);

        System.out.println("Volume of sphere #1 " +
            fmt.format(sphere1.getVolume()));
        System.out.println("Volume of sphere #2 " +
            fmt.format(sphere2.getVolume()));
        System.out.println("Volume of sphere #3 " +
            fmt.format(sphere3.getVolume()));
        System.out.println("Volume of sphere #4 " +
            fmt.format(sphere4.getVolume()));
    }
}

```

```

        System.out.println();
        System.out.println();

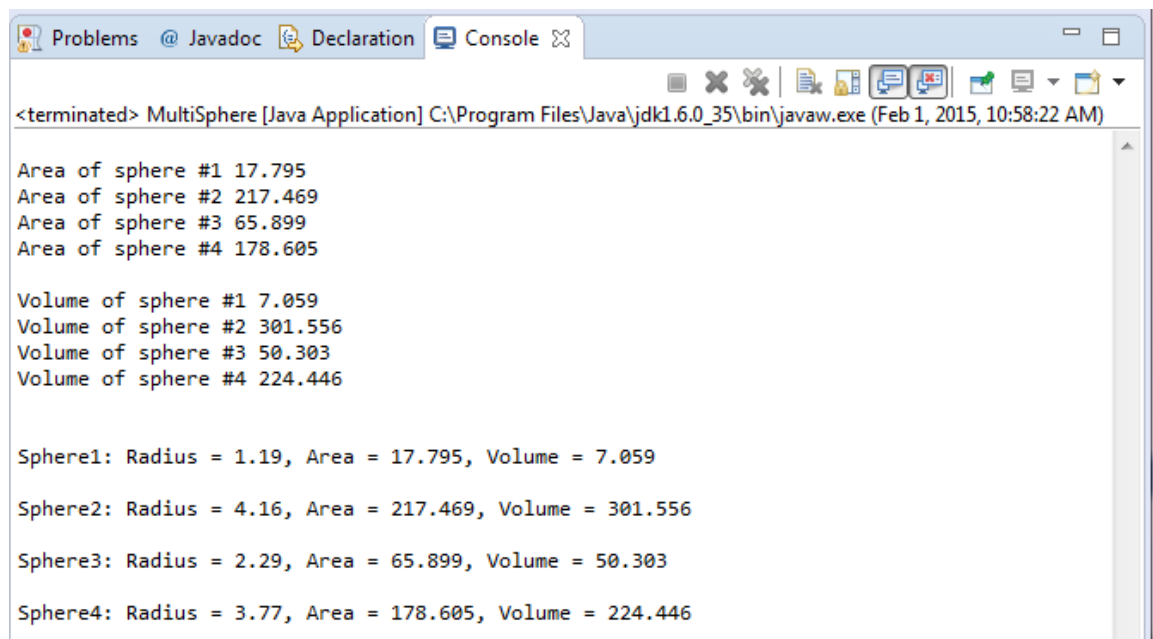
        sphere1.Spherify();
        sphere2.Spherify();
        sphere3.Spherify();
        sphere4.Spherify();

        System.out.println("Sphere1: " + sphere1);
        System.out.println("Sphere2: " + sphere2);
        System.out.println("Sphere3: " + sphere3);
        System.out.println("Sphere4: " + sphere4);

    }
}

```

Output:



```

<terminated> MultiSphere [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Feb 1, 2015, 10:58:22 AM)

Area of sphere #1 17.795
Area of sphere #2 217.469
Area of sphere #3 65.899
Area of sphere #4 178.605

Volume of sphere #1 7.059
Volume of sphere #2 301.556
Volume of sphere #3 50.303
Volume of sphere #4 224.446

Sphere1: Radius = 1.19, Area = 17.795, Volume = 7.059
Sphere2: Radius = 4.16, Area = 217.469, Volume = 301.556
Sphere3: Radius = 2.29, Area = 65.899, Volume = 50.303
Sphere4: Radius = 3.77, Area = 178.605, Volume = 224.446

```

11 Design and implement a class called *Box* that contains instance data that represents the height, width, and depth of the box. Also include a *boolean* variable called *full* as instance data that represents if the box is full or not. Define the *Box* constructor to accept and initialize the height, width, and depth of the box. Each newly created *Box* is empty (the constructor

should initialize full to false). Include getter and setter methods for all instance data. Include a `toString` method that returns a one-line description of the box. Create a driver class called `BoxTest`, whose `main` method instantiates and updates several `Box` objects.

The Box class

```

//*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Design and implement a class called Box that contains
//instance data that represents the height, width, and
//depth of the box. Also include a boolean variable called
//full as instance data that represents if the box is full
//or not. Define the Box constructor to accept and initialize
//the height, width, and depth of the box. Each newly created
//Box is empty (the constructor should initialize full to false).
//Include getter and setter methods for all instance data.
//Include a toString method that returns a one-line
//description of the box.
//
//*****

```

```

import java.text.DecimalFormat;

public class Box
{
    private double height;
    private double width;
    private double depth;
    private double area;
    private double volume;
    private boolean full;

    public Box(double h, double w, double d)
    {
        full = false;
        height = h;
        width = w;
        depth = d;
        this.Boxify();
    }

    public void setArea(double w, double d)
    {
        width = w; //Always update current values
        depth = d; //Always update current values
        area = (width * depth);
    }
}

```

```

    }

    public double getArea()
    {
        return(area);
    }

    public void setVolume(double h, double w, double d)
    {
        height = h; //Always update current values
        width = w;  //Always update current values
        depth = d;  //Always update current values
        volume = (width * depth * height);
    }

    public double getVolume()
    {
        return(volume);
    }

    public void setFilledState(boolean f)
    {
        full = f;
    }

    public String getFilledState()
    {
        return((full & true) ? "Full" : "Empty");
    }

    public void Boxify()
    {
        area = (width * depth);
        volume = (width * depth * height);
        full = (full & true);
    }

    public String toString()
    {
        DecimalFormat fmt = new DecimalFormat("0.###");

        return("Height = " + fmt.format(this.height) + ", "
            + "Width = " + fmt.format(this.width) + ", "
            + "Depth = " + fmt.format(this.depth) + ", "
            + "Area = " + fmt.format(this.area) + ", "
            + "Volume = " + fmt.format(this.volume) + ", "
            + this.getFilledState());
    }
}

```

The BoxTest driver class

```
//*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Create a driver class called BoxTest, whose main method
//instantiates and updates several Box objects.
//
//*****

import java.text.DecimalFormat;

public class BoxTest
{
    public static void main(String[] args)
    {
        Box box1 = new Box(3.35,2.25,1.15), box2 = new
        Box(4.7,3.27,4.17);
        Box box3 = new Box(7.93,5.63,8.23), box4 = new
        Box(11.56,10.15,12.25);

        System.out.println();
        System.out.println("***** Initialized the boxes *****");
        System.out.println("Box1: " + box1);
        System.out.println("Box2: " + box2);
        System.out.println("Box3: " + box3);
        System.out.println("Box4: " + box4);

        DecimalFormat fmt = new DecimalFormat("0.###");

        box1.setArea(1.19, 0.89);
        box2.setArea(2.34, 1.19);
        box3.setArea(6.59, 4.89);
        box4.setArea(11.29, 9.59);

        System.out.println();
        System.out.println("Area of box #1 " +
        fmt.format(box1.getArea()));
        System.out.println("Area of box #2 " +
        fmt.format(box2.getArea()));
        System.out.println("Area of box #3 " +
        fmt.format(box3.getArea()));
        System.out.println("Area of box #4 " +
        fmt.format(box4.getArea()));
        System.out.println();

        box1.setVolume(2.25,1.19, 0.89);
        box2.setVolume(3.15,2.34, 1.19);
        box3.setVolume(4.35,6.59, 4.89);
        box4.setVolume(6.65,11.29, 9.59);
    }
}
```

```

System.out.println("Volume of box #1 " +
    fmt.format(box1.getVolume()));
System.out.println("Volume of box #2 " +
    fmt.format(box2.getVolume()));
System.out.println("Volume of box #3 " +
    fmt.format(box3.getVolume()));
System.out.println("Volume of box #4 " +
    fmt.format(box4.getVolume()));

System.out.println();

box1.setFilledState(((int)(Math.random() * 2) != 0));
box2.setFilledState(((int)(Math.random() * 2) != 0));
box3.setFilledState(((int)(Math.random() * 2) != 0));
box4.setFilledState(((int)(Math.random() * 2) != 0));

System.out.println("Capacity of box #1 " +
    box1.getFilledState());
System.out.println("Capacity of box #2 " +
    box2.getFilledState());
System.out.println("Capacity of box #3 " +
    box3.getFilledState());
System.out.println("Capacity of box #4 " +
    box4.getFilledState());

System.out.println();
System.out.println();

box1.Boxify();
box2.Boxify();
box3.Boxify();
box4.Boxify();

System.out.println("Box1: " + box1);
System.out.println("Box2: " + box2);
System.out.println("Box3: " + box3);
System.out.println("Box4: " + box4);

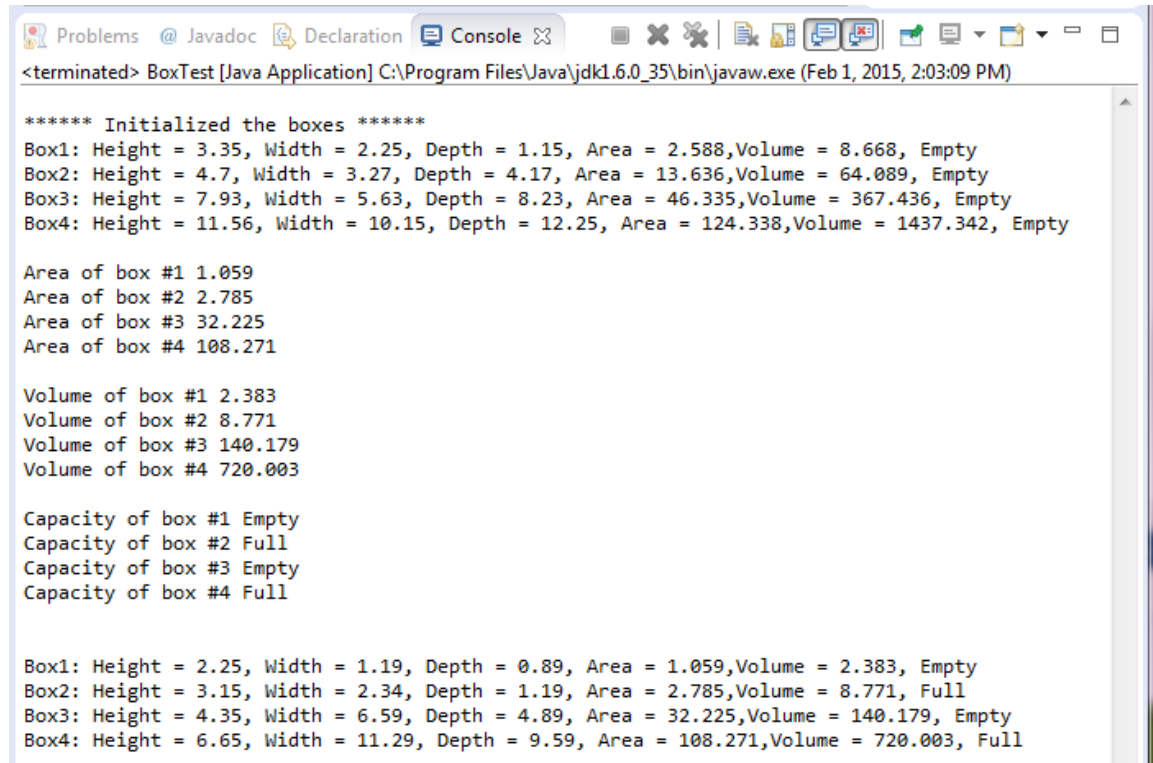
```

```

    }
}

```

Output:



```
<terminated> BoxTest [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Feb 1, 2015, 2:03:09 PM)

***** Initialized the boxes *****
Box1: Height = 3.35, Width = 2.25, Depth = 1.15, Area = 2.588,Volume = 8.668, Empty
Box2: Height = 4.7, Width = 3.27, Depth = 4.17, Area = 13.636,Volume = 64.089, Empty
Box3: Height = 7.93, Width = 5.63, Depth = 8.23, Area = 46.335,Volume = 367.436, Empty
Box4: Height = 11.56, Width = 10.15, Depth = 12.25, Area = 124.338,Volume = 1437.342, Empty

Area of box #1 1.059
Area of box #2 2.785
Area of box #3 32.225
Area of box #4 108.271

Volume of box #1 2.383
Volume of box #2 8.771
Volume of box #3 140.179
Volume of box #4 720.003

Capacity of box #1 Empty
Capacity of box #2 Full
Capacity of box #3 Empty
Capacity of box #4 Full

Box1: Height = 2.25, Width = 1.19, Depth = 0.89, Area = 1.059,Volume = 2.383, Empty
Box2: Height = 3.15, Width = 2.34, Depth = 1.19, Area = 2.785,Volume = 8.771, Full
Box3: Height = 4.35, Width = 6.59, Depth = 4.89, Area = 32.225,Volume = 140.179, Empty
Box4: Height = 6.65, Width = 11.29, Depth = 9.59, Area = 108.271,Volume = 720.003, Full
```

12 Using the *Die* class defined in this chapter, design and implement a class called *PairOfDice*, composed of two *Die* objects. Include methods to set and get the individual die values, a method to roll the dice, and a method that returns the current sum of the two die values. Rewrite the *SnakeEyes* program using a *PairOfDice* object.

The PairOfDice Class

```
/******
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Using the Die class defined in this chapter, design and implement a
//class called PairOfDice, composed of two Die objects. Include methods
//to set and get the individual die values, a method to roll the dice,
//and a method that returns the current sum of the two die values.
//
//*****
```



```

public class PairOfDice
{
    private final int MAX = 6; // maximum face value
    private int[] Diefacevalue = new int[2];
    // current value showing on the die
    //-----
    // Constructor: Sets the initial face value of this die.
    //-----
    public PairOfDice()
    {
        Diefacevalue[0] = 6;
        Diefacevalue[1] = 6;
    }
    //-----
    // Computes a new face value for this pair and returns the result.
    //-----
    public int[] roll()
    {
        Diefacevalue[0] = (int)(Math.random() * MAX) + 1;
        Diefacevalue[1] = (int)(Math.random() * MAX) + 1;
        return(Diefacevalue);
    }
    //-----
    // Face value mutator. The face value is not modified if the
    // specified value is not valid.
    //-----
    public void setFaceValues(int value1, int value2)
    {
        if(value1 > 0 && value1 <= MAX)
        {
            Diefacevalue[0] = value1;
        }

        if(value2 > 0 && value2 <= MAX)
        {
            Diefacevalue[1] = value2;
        }
    }
    //-----
    // Face value accessor.
    //-----
    public int[] getFaceValue()
    {
        return Diefacevalue;
    }
    //-----
    //Return the sum of the dice
    //-----
    public int getSumOfDice()
    {
        return(Diefacevalue[0] + Diefacevalue[1]);
    }
    //-----
    // Returns a string representation of this pair die.
    //-----
}

```

```

    public String toString()
    {
        String result = ("Die 1 = " + Integer.toString(Diefacevalue[0])
            + ", " + "Die 2 = " +
            Integer.toString(Diefacevalue[1])
            + ", Their sum = " +
            Integer.toString(Diefacevalue[1] +
            Diefacevalue[0]));
        return(result);
    }
}

```

The modified SnakeEyes driver class

```

/*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Rewrite the SnakeEyes program using a PairOfDice object.
//
*****/
public class SnakeEyes
{
    //-----
    // Creates two Die objects and rolls them several times, counting
    // the number of snake eyes that occur.
    //-----
    public static void main(String[] args)
    {
        final int ROLLS = 500;
        int[] numx = new int[2];
        int[] fv = new int[2];
        int count = 0, sums = 0;

        PairOfDice PoD1 = new PairOfDice(); //Each die will be 6
        System.out.println();
        System.out.println(PoD1);
        System.out.println();

        //Set the face values
        PoD1.setFaceValues(3, 3);
        System.out.println();
        System.out.println(PoD1);
        System.out.println();
        //Get the face values
        fv = PoD1.getFaceValue();
        System.out.println();
        System.out.println("Die 1: " + Integer.toString(fv[0]));
        System.out.println("Die 2: " + Integer.toString(fv[1]));
        System.out.println();
        //Get the sum
        sums = PoD1.getSumOfDice();
        System.out.println();
    }
}

```

```

        System.out.println("Their sum is: " + Integer.toString(sums));
        System.out.println();

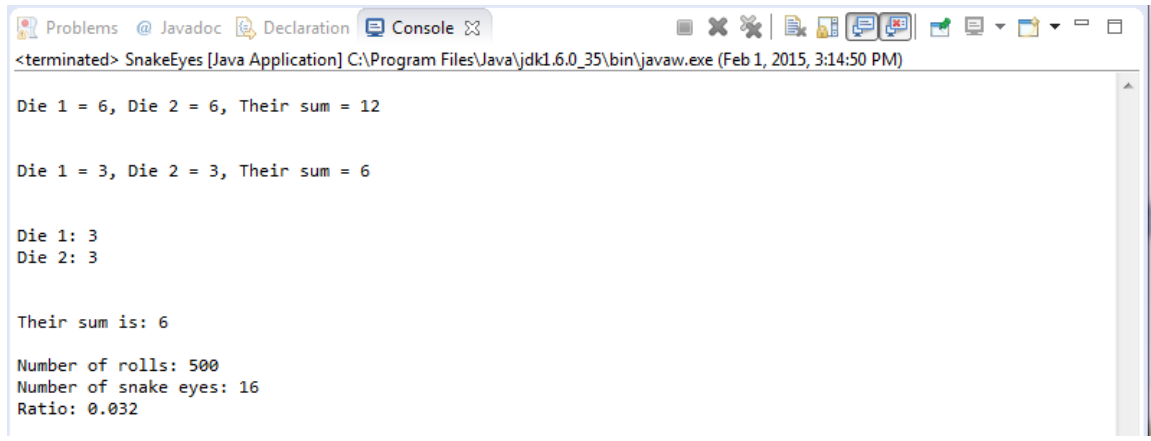
        for (int roll=1; roll <= ROLLS; roll++)
        {
            numx = PoD1.roll();

            if (numx[0] == 1 && numx[1] == 1) // check for snake eyes
                count++;
        }

        System.out.println("Number of rolls: " + ROLLS);
        System.out.println("Number of snake eyes: " + count);
        System.out.println("Ratio: " + (float)count / ROLLS);
    }
}

```

Output:



```

<terminated> SnakeEyes [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Feb 1, 2015, 3:14:50 PM)

Die 1 = 6, Die 2 = 6, Their sum = 12

Die 1 = 3, Die 2 = 3, Their sum = 6

Die 1: 3
Die 2: 3

Their sum is: 6

Number of rolls: 500
Number of snake eyes: 16
Ratio: 0.032

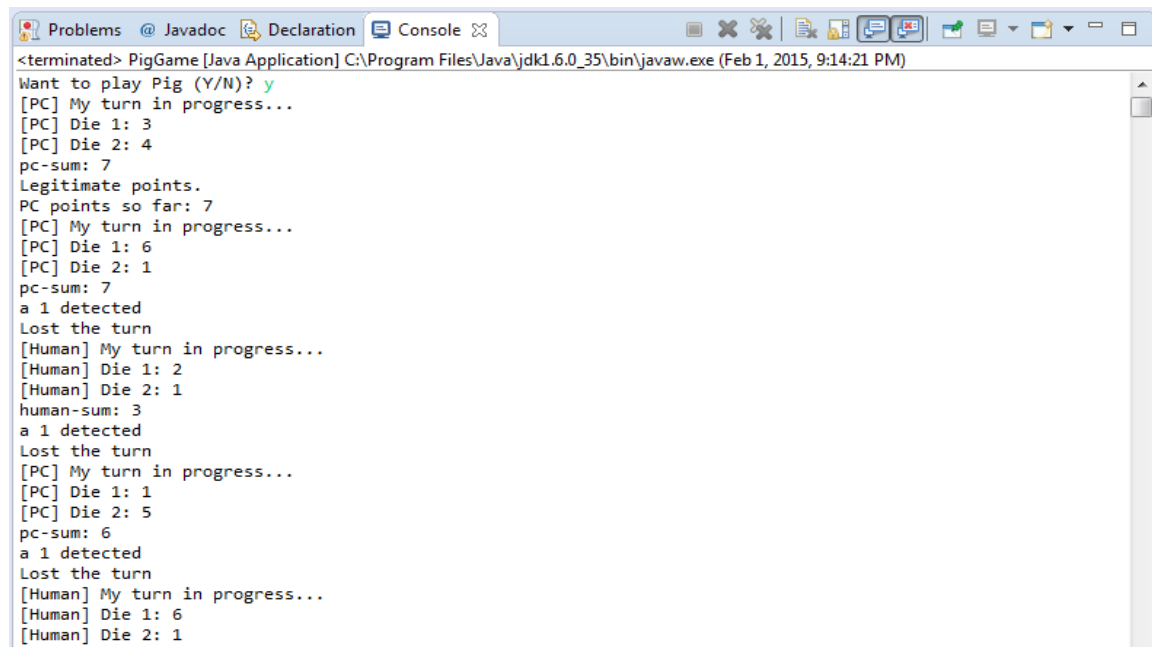
```

13 Using the *PairOfDice* class from programming project 5.10, design and implement a class to play a game called *Pig*. In this game, the user competes against the computer. On each turn, the current player rolls a pair of dice and accumulates points. The goal is to reach 100 points before your opponent does. If, on any turn, the player rolls a 1, all points accumulated for that round are forfeited and control of the dice moves to the other player. If the player rolls two 1's in one turn, the player loses all points accumulated thus far in the game and loses control of the dice. The player may voluntarily turn over the dice after each roll. Therefore the player must decide to either roll again (be a pig) and risk losing points or relinquish control of the dice, possibly allowing the other player to win. Implement

the computer player such that it always relinquishes the dice after accumulating 20 or more points in any given round.

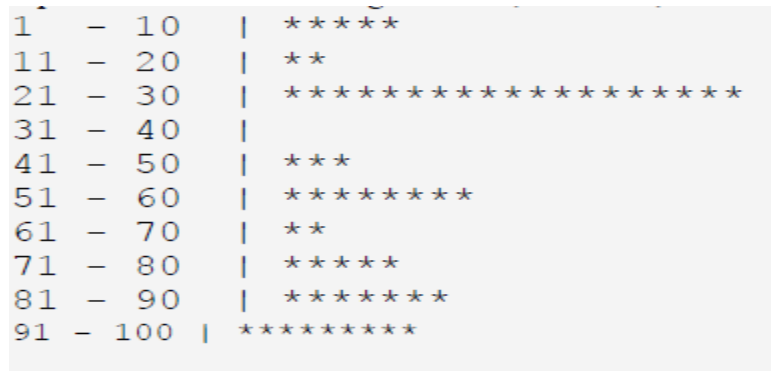
The PigGame driver class will be submitted as PigGame.java

Output:



```
<terminated> PigGame [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Feb 1, 2015, 9:14:21 PM)
Want to play Pig (Y/N)? y
[PC] My turn in progress...
[PC] Die 1: 3
[PC] Die 2: 4
pc-sum: 7
Legitimate points.
PC points so far: 7
[PC] My turn in progress...
[PC] Die 1: 6
[PC] Die 2: 1
pc-sum: 7
a 1 detected
Lost the turn
[Human] My turn in progress...
[Human] Die 1: 2
[Human] Die 2: 1
human-sum: 3
a 1 detected
Lost the turn
[PC] My turn in progress...
[PC] Die 1: 1
[PC] Die 2: 5
pc-sum: 6
a 1 detected
Lost the turn
[Human] My turn in progress...
[Human] Die 1: 6
[Human] Die 2: 1
```

14 Design and implement an application that creates a histogram that allows you to visually inspect the frequency distribution of a set of values. The program should read in an arbitrary number of integers that are in the range 1 to 100 inclusive; then produce a chart similar to the one below that indicates how many input values fell in the range 1 to 10, 11 to 20, and so on. Print one asterisk for each value entered.



The Histogram class

```
//*****
//Author: Hieu Pham
//ID: 0953-827
//Section: 28317
//Date: 02/01/2015
//
//Design and implement an application that creates a histogram that
//allows you to visually inspect the frequency distribution of a set
//of values. The program should read in an arbitrary number of integers
//that are in the range 1 to 100 inclusive; then produce a chart similar
//to the one below that indicates how many input values fell in the range
//1 to 10, 11 to 20, and so on. Print one asterisk for each value entered.
//
//*****

import java.util.Scanner;

public class Histogram
{
    public static void main(String[] args)
    {
        Scanner stdin = new Scanner(System.in);
        //Should have the same effect using the Scanner class

        final int MAXIMUM = 10;
        final int MINIMUM = 1;
        final int GAP      = 10;

        int[] dataset = new int[MAXIMUM]; //Create an array of 10 ints

        for(int b=0; b < dataset.length; b++)
        {
            dataset[b] = 0; // Initialize the array to zero
        }

        //Prompt user for integer data entry
        System.out.println("Please enter a set of integers between 1 and 100.");
        System.out.println("Enter an integer outside this range to stop.");

        System.out.print("Enter Integer: ");

        int value = stdin.nextInt();

        while(value >= MINIMUM && value <= (MAXIMUM*GAP))
        {
            //Divide by gap to increment accordingly
            dataset[(value-1)/GAP] = dataset[(value-1)/GAP] + 1;
        }
    }
}
```

```

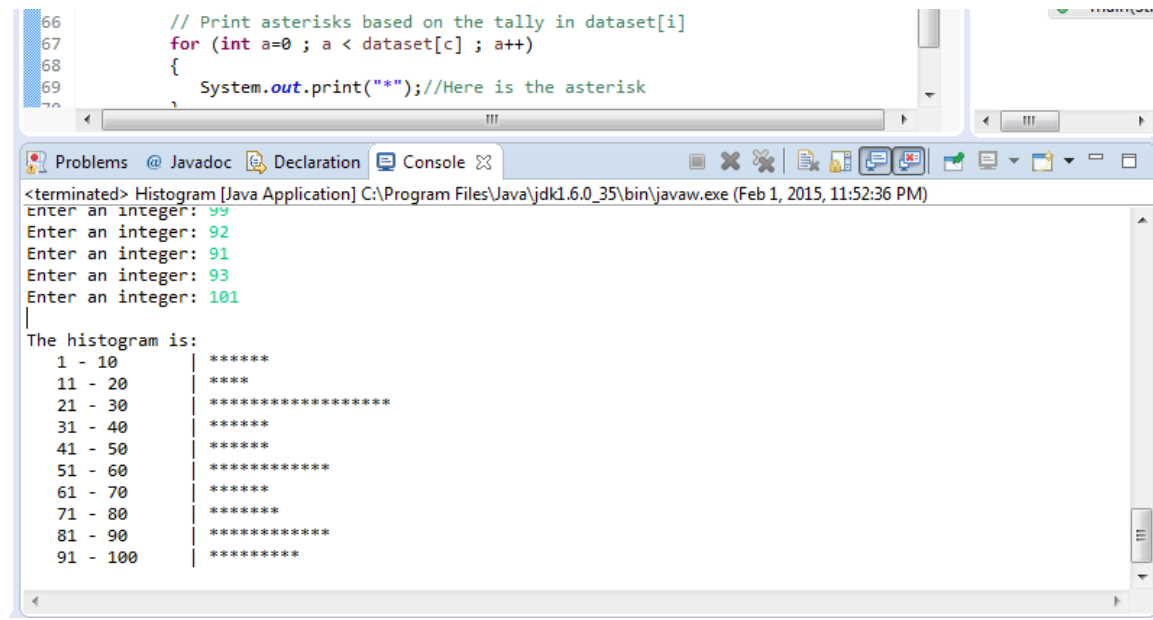
        //Next integer
        System.out.print("Enter an integer: ");
        value = stdin.nextInt();//Keep entering them integers
    }
    //Print out the histogram
    System.out.println("\nThe histogram is:");
    for(int c=0; c < dataset.length; c++)
    {

        //Arranging the fancy output of numbers, dashes, and asterisks
        System.out.print("    " + (c * GAP + 1) + " - " + (c + 1) * GAP
            + "\t| ");

        // Print asterisks based on the tally in dataset[i]
        for(int a=0 ; a < dataset[c] ; a++)
        {
            System.out.print("*");//Here is the asterisk
        }
        System.out.println();    //New line
    }
}
}

```

The Histogram Output:



```

66         // Print asterisks based on the tally in dataset[i]
67         for (int a=0 ; a < dataset[c] ; a++)
68         {
69             System.out.print("*");//Here is the asterisk
70         }

```

<terminated> Histogram [Java Application] C:\Program Files\Java\jdk1.6.0_35\bin\javaw.exe (Feb 1, 2015, 11:52:36 PM)
 Enter an integer: 99
 Enter an integer: 92
 Enter an integer: 91
 Enter an integer: 93
 Enter an integer: 101
 |
 The histogram is:
 1 - 10 | *****
 11 - 20 | *****
 21 - 30 | *****
 31 - 40 | *****
 41 - 50 | *****
 51 - 60 | *****
 61 - 70 | *****
 71 - 80 | *****
 81 - 90 | *****
 91 - 100 | *****

15 Define a class called *Quiz* that manages a set of up to 25 *Question* objects. Define the *add* method of the *Quiz* class to add a question to a quiz. Define the *giveQuiz* method of the *Quiz* class to present each question in turn to the user, accept

an answer for each one, and keep track of the results. Define a class called `QuizTime` with a main method that populates a quiz, presents it, and prints the final results.

I am sorry, but I do not understand this multiple part question. What does the `Question` class look like to store the "Question" objects? I am totally lost in this question.