Test 1 Review: CompSci 6

Name (print): _____

Honor Acknowledgment (signature): _____

DO NOT SPEND MORE THAN 15 MINUTES ON ANY OF THE QUESTIONS! If you do not see the solution to a problem right away, move on to another problem and come back to it later. The final page is a list of common methods of classes we have studied in class so that you do not need to memorize such details.

Before starting, make sure your test contains 12 pages.

If you think there is a syntax error, then ask.

	value	grade
Problem 1	10 pts.	
Problem 2	15 pts.	
Problem 3	9 pts.	
Problem 4	21 pts.	
TOTAL:	55 pts.	

PROBLEM 1 : (*Why the cover up?:* (10 pts))

Part A (4 points)

Complete the method, `isInside`, that returns true if the given `Point` object is inside the given `Mover` object and false otherwise. In this case, inside is defined as within or on the boundary of the rectangle that surrounds the `Mover` object (i.e., as defined by its center and size).

For example, if `m` represents a `Mover` with top-left coordinates $(100, 100)$ and dimensions $200x100$ pixels, then the table below shows the results of several calls to `isInside`:

Function call	**Return Value**
`isInside(new Point(150, 150), m)`	true
`isInside(new Point(100, 100), m)`	true
`isInside(new Point(300, 200), m)`	true
`isInside(new Point( 50, 150), m)`	false
`isInside(new Point( 99,  99), m)`	false

Complete the method `isInside` below:

```
/**
 * @return true if pt is within the rectangle that bounds m,
 *         or false otherwise
 */
public boolean isInside (Point pt, Mover m)
{









}
```

Part B (6 points)

Complete the method, `averageArea`, that returns the average area of a collection of `Mover` objects. Since a `Mover` can paint itself as anything though, you should use the rectangle that bounds the `Mover` object, i.e., its size, to calculate its area. Recall that the area of a rectangle is its $width * height$. If the given collection is empty, i.e., its `size` is 0, then the method should return 0.

Complete the method `averageArea` below:

```
/**
 * @return the average of the areas of all the rectangles that bound the
 *         shapes drawn in the list movers, or 0 if the list is empty
 */
public double averageArea (ArrayList<Mover> movers)
{









}
```

PROBLEM 2 : (*Extreme Weather (15 points)*)

Part A (6 points)

The hailstone sequence, sometimes called the $3n + 1$ sequence, is defined by a function $f(n)$:

```
f(n) =  n / 2       if n is even
     = 3n + 1       if n is odd
```

We can use the value computed by $f(n)$ as successive arguments of $f(n)$ as shown below, the successive values of n form the hailstone sequence (it is called a hailstone sequence because the numbers go up and down mimicking the process that forms hail).

```
while (n != 1)
{
    n = f(n);
}
```

Although it is conjectured that this loop always terminates, no one has been able to prove it. However, it has been verified by computer for an enormous range of numbers. Several sequences are shown below with the initial value of n on the left.

```
 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
 8  4  2  1
 9 28 14  7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
11 34 17 52 26 13 40 20 10  5 16  8  4  2 1
14  7 22 11 34 17 52 26 13 40 20 10  5 16 8 4 2 1
22 11 34 17 52 26 13 40 20 10  5 16  8  4 2 1
```

Complete the method, `hailstone`, that returns the **number of steps** in the hailstone sequence that starts with a given number (i.e., for the first three examples shown above, the function should return 17, 4, and 20, respectively).

```
public int hailstone (int n)
{









}
```

Part B (9 points)

Complete the function, `longestHailstone`, that, given a list of numbers, returns the one that yields the longest hailstone sequence. You can assume that the list contains only positive integer values.

In order to get full credit, you must use the method you wrote in the previous part to determine how long each sequence is.

Complete `longestHailstone` below:

```
public int longestHailstone (ArrayList<Integer> numbers)
{









}
```

PROBLEM 3 : (*Working Together (21 pts)*)

Assume that every player on a team is modelled using the class `Player` declared as shown below.

```
public class Player
{
    private String myName;
    private ArrayList<Integer> myMinutes; // minutes played in each game

    public Player (String name, ArrayList<Integer> minutes)
    {
        myName = name;
        myMinutes = minutes;
    }

    public String getName ()
    {
        return myName;
    }

    public int getMinutesForGame (int whichGame)
    {
        // assume 0 <= whichGame < myMinutes.length
        return myMinutes.get(whichGame).intValue();
    }

    public int getNumberGamesPlayed ()
    {
        return myMinutes.size();
    }
}
```

Part A (10 points)

Write the function `readTeam` below that reads data from a file and stores it in an initially empty list of `Player`'s. Assume that the text file with team information being read is in the format shown below. There are two lines for each player in the team. The first line contains first and last name, and the second line contains a list of the number of minutes played in each game during the season. Note that the number of games played varies from player to player.

A sample data file is shown below with the first player with 4 games and the second play with 7 games:

```
Fred Smith
1 13 5 1
Chris Jones
8 2 5 6 4 7 16
```

Complete `readTeam` below.

```
/**
 * Reads all player information from the file represented by
 * the parameter Scanner and returns it in a list
 */
public ArrayList<Player> readTeam (Scanner input)
{
    input.useDelimiter("\n");









}
```

Part B (4 points)

Write the function `totalNumberOfMinutes` below that, given a `Player`, returns the total number of minutes that player played during the season.

Given the example players in the previous part, your function should return the value `20` minutes when called with the `Player` object representing Fred Smith and `48` minutes when called with the `Player` object representing Chris Jones.

Complete `totalNumberOfMinutes` below. Note, this method is not part of the `Player` class and so does not have access to its private instance variables.

```
/**
 * Return the sum of all the minutes player played.
 */
public int totalNumberOfMinutes (Player player)
{









}
```

Part C (7 points)

Write the function `mostMinutes` below that, given a list of players representing a team, returns the `Player` who has played the most total minutes during the season.

Given the example team from the previous parts, your function should return the `Player` object representing Chris Jones because he played for 48 total minutes, while his only other team member played for only 20 minutes.

You will not receive full credit for this part unless you call the function `totalNumberOfMinutes` that you wrote in Part B at least once and use its result in determining the result of this function. Assume `totalNumberOfMinutes` works as specified regardless of what you wrote in Part B.

Complete `mostMinutes` below.

```
/**
 * Return the player that has played the most minutes regardless
 * of the number of games played.
 */
public Player mostMinutes (ArrayList<Player> team)
{









}
```

PROBLEM 4 : (*It never stops:* (9 points))

Complete the sub-class of `Mover` such that it changes the color used to draw itself each step of the animation. The class will be constructed with a collection of colors that determines the order in which the colors will change over time. In other words, each time the `move` method is called, it should change some state in the object such that the next color in the list will be used to draw the shape when `paint` is called. When the end of the list is reached, it should start over with the first color in the list.

Complete the class `ColorCycler` started below by adding any instance variables you may need, initializing them in the constructor, and filling in the methods `paint` and `move`.

```
public class ColorCycler extends Mover
{
    private ArrayList<Color> myColors;
    // add any additonal instance variables you may need









    public ColorCycler (Point center, Dimension size, Point velocity, ArrayList<Color> colors)
    {
        super(center, size, velocity, colors.get(0));
        myColors = colors;
        // initialize any other instance variables here




    }

    public void paint (Graphics pen)
    {
        pen.setColor(                                              );
        // rest of implementation not shown
    }

    public void move (Dimension bounds)
    {
```

```
    }
}
```

Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class. There should be no methods you have never seen before here.

Point

```
public class Point {
    // coordinates
    public int x;
    public int y;

    // Constructs and initializes a point at the
    // specified (x,y) location.
    public Point (int x, int y)

    // Returns distance from this point to given one
    public int distance (Point other)

    // Adds dx and dy to this point's coordinates
    public void translate (int dx, int dy)
}
```

Dimension

```
public class Dimension {
    // lengths
    public int width;
    public int height;

    // Constructs and initializes a dimension
    // with the specified (w, h) lengths.
    public Dimension (int w, int h)
}
```

Color

```
public class Dimension {
    // Constructs and initializes a color with
    // the specified (r, g, b) components.
    public Color (int red, int green, int blue)

    // Returns the red component
    public int getRed ()

    // Returns the green component
    public int getGreen ()

    // Returns the blue component
    public int getBlue ()

    // Returns a color that is brighter than
    // this one
    public Color brighter ()

    // Returns a color that is darker than
    // this one
    public Color darker ()
}
```

Random

```
public class Random {
    // Creates a new random number generator.
    public Random ()

    // Returns pseudorandom, uniformly distributed,
    // int value between 0 (inclusive) and the
    // specified value (exclusive)
    public int nextInt (int n)
}
```

ArrayList

```
public class ArrayList {
    // Constructs an empty list
    public ArrayList ()

    // Returns the number of elements in this list.
    public int size ()

    // Searches for the first occurence of the given
    // argument, returns  -1 if not found
    public int indexOf (Object itemm)

    // Returns element at index in this list.
    public Object get (int index)

    // Appends specified element to end of this list.
    public boolean add (Object o)
}
```

Scanner

```
public class Scanner
{
    // Create Scanner for that reads data from a file.
    public Scanner (File file)

    // Create Scanner for that reads data from a string.
    public Scanner (String str)

    // Change delimiters used to separate items
    public void useDelimiter (String characters)

    // Check if more items are available
    public boolean hasNext ()

    // Get next delimited item as a string
    public String next ()

    // Get next delimited item as an integer value
    public int nextInt ()
}
```

Integer

```
public class Integer {
    // The smallest value of type int
    public static final int MIN_VALUE

    // The largest value of type int
    public static final int MAX_VALUE

    // Returns the integer represented by the
    // argument as a decimal integer.
    public static int parseInt (String s)

    // Returns a new String object representing
    // the specified integer.
    public static String toString (int i)

    // Returns value of Integer object as an int
    public int intValue ()
}
```

String

```
public class String {
    // Returns length of this string.
    public int length ()

    // Returns index within this string of first
    // occurrence of the specified substring.
    // If str is not found, then returns -1
    public int indexOf (String str)

    // Returns index within this string of last
    // occurrence of the specified substring.
    // If str is not found, then returns -1
    public int lastIndexOf (String str)
}
```