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 14 pages.

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

	value	grade
Problem 1	6 pts.	
Problem 2	14 pts.	
Problem 3	20 pts.	
Problem 4	15 pts.	
TOTAL:	55 pts.	

PROBLEM 1 : (*Parse, Parse, Baby:* (8 pts))

Write the method `average` which, given a string of positive whole numbers (each separated by a comma), returns the average of those numbers. The given string is guaranteed to contain at least one value and does not contain any spaces or other punctuation.

For example, the code given below should assign i, j, and k the values 1.0, 2.33, and 277.75, respectively.

```
String s = "1,1,1,1,1";
String t = "4,2,1";
String u = "10,1,100,1000";

double i = average(s);
double j = average(t);
double k = average(u);
```

Complete the method `average` below.

```
public double average (String values)
{









}
```

PROBLEM 2 : (*Where's the love? (14 points)*)

Part A (6 points)

Write a function `countOccurrences` that returns the number of times the given word appears in the given collection of words.

For example, if `words` is an `ArrayList` representing the string values `{"what", "love", "is", "what", "lovers", "love", "love"}` then the table below shows the results of several calls to `countOccurrences`:

Function call	**Return Value**
`countOccurrences("love", words)`	3
`countOccurrences("what", words)`	2
`countOccurrences("lovers", words)`	1
`countOccurrences("compsci", words)`	0

Complete the method `countOccurrences` below:

```
public int countOccurrences (String str, ArrayList<String> words)
{









}
```

Part B (8 points)

Write a function `maxOccurring` that returns the word that appears the most number of times in the given collection.

For example, if, as in the previous part, `words` represents the string values `{"what", "love", "is", "what", "lovers", "love", "love"}` then the call to `maxOccurring` should return the string `"love"`.

In writing `maxOccurring` you must call `countOccurrences` from Part A to receive full credit. Assume `countOccurrences` works as specified, regardless of what you wrote in Part A..

Complete the method `maxOccurring` below:

```
public String maxOccurring (ArrayList<String> words)
{









}
```

PROBLEM 3 : (*Everythings included. (20 points)*)

Consider the class given below that models a rectangle as a `Point` and a `Dimension`. The point represents the top, left coordinate of the rectangle and the dimension represents its width and height. You will complete its methods as part of this problem.

```
public class Rectangle
{
    private Point myTopLeft;
    private Dimension mySize;

    public Rectangle (Point topLeft, Dimension size) {
        myTopLeft = topLeft;
        mySize = size;
    }

    public int getLeft () {
        return myTopLeft.x;
    }

    public int getRight () {
        return myTopLeft.x + mySize.width;
    }

    public int getTop () {
        return myTopLeft.y;
    }

    public int getBottom () {
        return myTopLeft.y + mySize.height;
    }

    public void translate (int xChange, int yChange) {
        // to be implemented in Part A
    }

    public void includePoint (Point pt) {
        // to be implemented in Part B
    }

    public Rectangle union (Rectangle other) {
        // to be implemented in Part C
    }
}
```

Problem continued on next page ...

Part A (3 points)

Write the `Rectangle` method `translate` that takes two integer parameters, representing the number of pixels to move a rectangle horizontally and vertically, and moves the rectangle by changing its left and top coordinate. This method should return nothing, but instead change the state of the rectangle.

Complete the method `translate` below:

```
public void translate (int xChange, int yChange)
{









}
```

Part B (7 points)

Write the `Rectangle` method `includePoint` that takes a `Point` parameter, representing a point on the screen, and expands the rectangle minimally such that it contains the new point. This method should return nothing, but instead change the state of the rectangle.

For example, if the rectangle, `r`, has the top-left coordinate $(100, 100)$ and a size of $50x50$, then the following calls to `includePoint` result in the successive changes to the `r`:

Function call	**New State of Rectangle `r`**
`r.includePoint(new Point(105, 115))`	$(100, 100)$ $50x50$
`r.includePoint(new Point(155, 115))`	$(100, 100)$ $55x50$
`r.includePoint(new Point( 95, 115))`	$(95, 100)$ $60x50$
`r.includePoint(new Point(105, 155))`	$(95, 100)$ $60x55$
`r.includePoint(new Point(105,  95))`	$(95, 95)$ $60x60$

Complete the `Rectangle` method `includePoint` below:

```
public void includePoint (Point pt)
{









}
```

Part C (5 points)

Write the `Rectangle` method `union` that takes a `Rectangle` as a parameter, representing another rectangle, and returns a new rectangle that is minimally positioned and sized such that it covers the space occupied by both rectangles. This method should not modify the state of the current rectangle or the rectangle passed as a parameter.

For example, if the rectangle, `r`, has the top-left coordinate $(100, 100)$ and a size of $50x50$, and another rectangle, `s`, has the top-left coordinate $(90, 110)$ and a size of $70x30$, then the union of these two rectangles is a rectangle with the top-left coordinate $(90, 100)$ and a size of $70x50$. Additionally, if one of the rectangles completely covers the other, the result is a new rectangle with the same values as the larger rectangle.

Complete the method `union` below:

```
public Rectangle union (Rectangle other)
{









}
```

Part D (5 points)

Write the method `unionAll` that takes a collection of `Rectangle` objects as a parameter, and returns a new `Rectangle` representing the union of all rectangles in the list. You can assume the collection has at least one rectangle.

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

```
public Rectangle unionAll (ArrayList<Rectangle> rects)
{









}
```

PROBLEM 4 : (*Save a lot:* (15 points))

A grocery store runs a variety of deals each week on its products by giving its customers coupons. These coupons vary by the product, its size (the units of which is specific to each product), or possibly the number of items bought. Additionally, each coupon has an expiration date.

Below the code for the base class *Coupon* is given. During the rest of the problem, you will complete the code for several sub-classes of *Coupon* so that they work as described. In writing each subclass, you should declare any instance variables you may need and you should try to write as little new code as possible for each method by taking advantage of calls to the super class version of the method where appropriate. **If the subclass does not need to implement a method, i.e., no new code is needed for the subclass' method to work correctly, then clearly cross the method out rather than implement it — do not simply leave it blank.**

```
public abstract class Coupon
{
    private Date myExpiration;
    private String myProductName;

    public Coupon (Date expiration, String productName)
    {
        myExpiration = expiration;
        myProductName = productName;
    }

    // returns amount of money off of given item
    public double getDiscount (GroceryProduct item, int numBought)
    {
        if (isMatch(item, numBought)) return calculateDiscount(item, numBought);
        else                          return 0.0;
    }

    // returns true only if correct brand and coupon has not expired
    protected boolean isMatch (GroceryProduct item, int numBought)
    {
        Date today = new Date();
        return (today.compareTo(myExpiration) <= 0) &&
               (myProductName.equals(item.getBrand()));
    }

    protected abstract double calculateDiscount (GroceryProduct item, int numBought);
}
```

The classes `GroceryProduct` and `Date` are used within this problem and given at the end of the exam.

Problem continued on next page ...

Part A (8 points)

The class *PercentOff* gives a discount equal to a percentage of the total price of the items bought. For example, given the following code:

```
Coupon c = new PercentOff(new Date(1, 1, 2005), "Swifter", 0.1);
GroceryProduct cleaner = new GroceryProduct("Swifter", 1, 19.99);
System.out.println(c3.amountToSubtract(cleaner, 1));
System.out.println(c3.amountToSubtract(cleaner, 5));
```

should print

```
1.99
9.95
```

because in the first call to *getDiscount* only one item was bought, so ten percent of the item's price, $1.99, was given as the discount. In the last call, five items were bought and the $1.99 discount was applied to each for a total of $9.95.

```
public class PercentOff extends Coupon
{




   public PercentOff (Date expiration, String productName, double percentOff)
   {




   }

   protected boolean isMatch (GroceryProduct item, int numBought)
   {




   }

   protected double calculateDiscount (GroceryProduct item, int numBought)
   {




   }
}
```

Part B (7 points)

The class *GetOneFree* gives a discount equal to the price of one item if a certain number are bought. For example, given the following code:

```
Coupon c = new GetOneFree(new Date(1, 1, 2005), "Pepsi", 2);
GroceryProduct soda = new GroceryProduct("Pepsi", 6, 1.99);
System.out.println(c.getDiscount(soda, 1));
System.out.println(c.getDiscount(soda, 2));
System.out.println(c.getDiscount(soda, 4));
```

should print

```
0.0
1.99
1.99
```

because in the first call to *getDiscount* only one six-pack of sodas was bought; it was not enough to get one free. In the second call two six-packs were bought, just enough to receive a free one — which costs $1.99. In the last call, four six-packs were bought, but only one was considered free. The same discount is applied no matter how many items are purchased over the minimum number.

```
public class GetOneFree extends Coupon
{




   public GetOneFree (Date expiration, String productName, int numNeeded)
   {




   }

   protected boolean isMatch (GroceryProduct item, int numBought)
   {




   }

   protected double calculateDiscount (GroceryProduct item, int numBought)
   {




   }
}
```

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)
}
```