AP CS Unit 6: Inheritance Programs

Program 1. Complete the Rectangle class.

The Rectangle class represents a rectangle in a standard coordinate plane with the sides of the rectangle being parallel to the x and y axes.

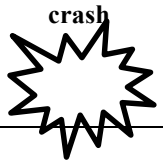
(x1, y1) represent the lower left hand corner of the rectangle. (x2, y2) represent the upper right hand corner of the rectangle.

```
public class Rectangle{
    private int x1, y1, x2, y2;

    public Rectangle( int x1, int y1, int x2, int y2){
        if ( x1 >= x2 || y1 >= y2 )
            throw new IllegalArgumentException("x1 >= x2 or  y1 >= y2");
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }

    public int getPerimeter(){
        returns the perimeter of the rectangle.
            ---->

    }

    public int getArea(){
        returns the area of the rectangle
    }
```

Use the distance method that the Point class inherits from Point2D class.

Test your code with the code to the right.

```
public class RunRectangle{
    public static void main( String [] args ){
        Rectangle r1 = new Rectangle( -1, 4, 2, 8 );
        System.out.println( r1.getPerimeter() );        // 14
        System.out.println( r1.getArea() );             // 12

        Rectangle r2 = new Rectangle( 2, 5, 2, 6  );
//java.lang.IllegalArgumentException: x1 >= x2 or  y1 >= y2
    }
}
```



Now write the Square class that extends the Rectangle class. The Square class has no instance variables. Its constructor has three parameters: the x and y coordinates of the lower left hand corner and the length of a side.

When finished the Square class should be no more than 5 line long.

Use the code to the right to test it.

```
public class RunSquare{
    public static void main( String [] args ){
        Square s = new Square( 5, 6, 9 );
        System.out.println( s.getArea() );    // 81
        System.out.println( s.getPerimeter() );  // 36
    }
}
```

Program 2. Complete the Course and Weighted classes.

```
public class Course{
    private String name;
    private double grade;

    public Course( String s ){
        name = s;
        grade = 0;
    }

    public void setGrade( double g ){
        grade = g;
    }

    public double getGrade(){
        return grade;
    }

    @Override          // see note below
    public String toString(){
        return name + ": " + grade;
    }
}
```

The Weighted class is a subclass of Course and it represents a course with a weighted grade (e.g. an AP or honors course)

The Weighted class should have the following:
- one instance variable, a double, which is the weight for a particular course. (The course name and unweighted grade are part of the super class.)

The constructor header is this:
public Weighted(double w, String s)
where w is the weight and s is the name

A method that returns the weighted grade:
public double getWeightedGrade()

And override the toString method.

See the sample code below to determine what these should do.

Do NOT add any extra instance variables or methods.

Here's some a quote from the Oracle documentations,

"When overriding a method, you might want to use the `@Override` annotation that instructs the compiler that you intend to override a method in the superclass. If, for some reason, the compiler detects that the method does not exist in one of the superclasses, then it will generate an error." This is a good idea because sometimes people accidently misspell the method name and end up writing a new method instead of overriding an existing method. It is optional.
- https://docs.oracle.com/javase/tutorial/java/IandI/override.html

Use this class to test your Weighted class.

Notice that the subclass cannot directly access the private instance variables of Course but can call its public methods.

```
public class RunCourse1{
        public static void main( String [] args ){
                Course c = new Course( "Intro to Java" );
                c.setGrade( 92 );
                System.out.println( c.getGrade() );     // 92.0
                System.out.println( c );                // Intro to Java: 92.0

                Weighted w = new Weighted( 1.2, "APCS" );
                w.setGrade( 88 );
                System.out.println( w.getGrade() );                // 88.0
                System.out.println( w.getWeightedGrade() );        // 105.6
                System.out.println( w );        // APCS: 88.0, weighted: 105.6

        }
}
```

Program 3. Using the Course and Weighted classes from problem 2, complete the following program.

```
public class RunManyCourses{
    public static void main( String [] args ){
        String [] names1 = {"Honors English", "APCS", "Chemistry", "History" };
        double [] grades1 = { 71, 85, 94, 87 };
        Double [] weights1 = { 1.1, 1.2, null, null };

        Course [] c = setup( names1, grades1, weights1 );
        for ( Course cor : c )
            System.out.println( cor );

        double avg = getUnweightedAverage( c );
        System.out.println ( "Unweighted average: " + avg );
        double wavg = getWeightedAverage( c );
        System.out.println ( "Weighted average: " + wavg );
        System.out.println ( "\n**********************\n" );

        String [] names2 = {"Reading", "Math", "AP US History" };
        double [] grades2 = { 90, 80, 85 };
        Double [] weights2 = { null, null, 1.2 };
        c = setup( names2, grades2, weights2 );
        for ( Course cor : c )
            System.out.println( cor );

        avg = getUnweightedAverage( c );
        System.out.println ( "Unweighted average: " + avg );
        wavg = getWeightedAverage( c );
        System.out.println ( "Weighted average: " + wavg );
    }

    public static Course [] setup( String [] s, double [] g, Double [] wts ){
        Returns an array filled with Course and Weighted objects.  Use the String elements for
the names.  If a particular weight is null then create a Course object; otherwise create a
Weighted object.  Use the g array to set the grades.
    }

    public static double getUnweightedAverage( Course[] c ){
        Returns the unweighted average of all the courses.
    }

    public static double getWeightedAverage( Course[] c ){
        Returns the weighted average of all the courses.  If a course is not weighted then use its
regular, unweighted grade.
    }
}
```

Your output should look like this:

```
Honors English: 71.0, weighted:
78.10000000000001
APCS: 85.0, weighted: 102.0
Chemistry: 94.0
History: 87.0
Unweighted average: 84.25
Weighted average: 90.275

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

```
Reading: 90.0
Math: 80.0
AP US History: 85.0, weighted: 102.0
Unweighted average: 85.0
Weighted average: 90.66666666666667
```

Program 4. Copy the Ring class and write the MagicRing class.

```
public class Ring{
    private int value;

    public Ring( int v ){
        value = v;
    }

    public int getValue(){
        return value;
    }

    @Override
    public String toString(){
        return "ring worth $" + value;
    }
}
```

The MagicRing class is a subclass of the Ring class.

It has one instance variable, a Boolean. If true then this is a lucky magic ring; otherwise it is an unlucky magic ring.

Write a constructor that has one parameter, an int the represents the value of the ring. Randomly assign the instance variable a value so that there's a 50% chance of it being true or false.

Write an accessor method for the instance variable.

Override the toString method so that it is consistent with the outputs shown in the sample code below.

Complete the code below. To the right is some sample output.

```
lucky ring worth $6
ring worth $15
lucky ring worth $15
lucky ring worth $15
ring worth $16
unlucky ring worth $11
ring worth $14
ring worth $15
unlucky ring worth $12
lucky ring worth $7

Total value: 126
There are 4 lucky magic rings
```

```
public class RunRings{
    public static void main( String [] args ){
        Ring [] rings = new Ring[10];
        for ( int n=0; n<10; n++ ){
            rings[n] = get();
            System.out.println( rings[n] );
        }

        int totalValue = getTotalValue( rings );
        System.out.println( "\nTotal value: " + totalValue );
        int count = countLuckyMagicRings( rings );
        System.out.println( "There are " + count + " lucky magic rings"  );
    }

    public static Ring get(){
        50% of the time this returns a ring and 50% of the time it returns a magic ring.  The value
of any ring is a random number between 5 and 20
    }

    public static int getTotalValue( Ring [] r ){
        Returns the sum of all the values of all the rings
    }

    public static int countLuckyMagicRings( Ring [] r ){
        Returns the number of lucky magic rings in the array
    }
}
```

Program 5. This project contains 5 classes though most are short. When you run the finished program, it should display a 4 by 4 grid that looks something like this:

6	9	6	9
4	3	B	2
6	1	Joe	4
4	B!!	1	9

The player is next to 20 bars of gold.
The player is next to 2 bombs.
1 of them is/are deadly.

The grid contains:

- 1 player named Joe (Keep the name short.)
- 2 bombs (B!! if deadly, B if not deadly)
- 13 locations that contain gold bars (the numbers indicate the number of bars at each location.
- The player, bombs, and gold are all assigned random locations in the grid.

```
public abstract class Piece {
    private int row, col;

    public Piece( ) {
        row = -1;
        col = -1;
    }

    public void setLocation( int r, int c ){
        row = r;
        col = c;
    }

    public boolean nextTo( Piece p ){
        returns true if p is adjacent to this piece
(locations at a diagonal count)
    }
}
```

```
public class Gold  extends Piece {
    private int bars;

    public Gold() {
        bars = (int)(9 * Math.random() )+ 1;
    }

    public int get() {
        return bars;
    }

    @Override
    public String toString() {
        return "" + bars;
    }
}
```

```
public class Player extends Piece {
    private String name;

    public Player( String s ) {
        name = s;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

```
public class Bomb extends Piece{
    private boolean deadly;

    public Bomb() {
        50% of the time deadly should be true
    }

    public boolean isDeadly() {
        return deadly;
    }

    @Override
    public String toString() {
        if deadly is true, return "B!!" else return
“B”
    }
}
```

Now here's the class that puts it all together and where you have to do some thinkin'.

```
public class Runner {
	public static void main(String[] args) {
		Piece [][] grid = new Piece[4][4];
		place( grid, new Player( "Joe" ) );
		// use the place method to put two bombs in the grid (two lines of code)
		// put 13 Gold objects in the remaining locations in the grid (one loop)
		// print out grid (nested loops)

		Player p = findPlayer( grid );
		int num = getGold( grid, p );
		System.out.println("The player is next to " + num + " bars of gold." );

		Bomb[] bombs = findBombs( grid, 2 );
		int nextToBombs = 0;
		int nextToDeadly = 0;
		// count the bombs (deadly or not) near the player
		// you'll need a loop and the nextToBombs and nextToDeadly variables
		System.out.println("The player is next to " + nextToBombs + " bombs." );
		if ( nextToBombs>0 )
			System.out.println( nextToDeadly + " of them is/are deadly.\n" );
	}

	public static void place( Piece [][] aa, Piece p ){
		// generate a random row and column
		while (  aa[row][col] != null  ){
			// generate another random row and column
		}
		aa[row][col] = p;
		p.setLocation( row, col );
	}

	public static Player findPlayer( Piece [][] aa ){
		// searches the array for the first piece that is an instance of the Player class
		// return this value
		// return null if a Player is not found (though you will find one)
	}

	public static int getGold( Piece [][] aa, Player p ){
		// Given a player, return the number of adjacent bars of gold.
	}

	public static Bomb[] findBombs( Piece [][] aa, int num ){
		// Returns an array of Bombs that are in aa, num is the number of bombs
	}
}
```