CS 2510 Exam 2 – Spring 2011

Name: _____

Student Id (last 4 digits): _____

• Write down the answers in the space provided.
• You may use all parts of the Java language we have learned. If you need a method and you don't know whether it is provided, define it. You do not need to include the curly braces for every `if` or every `else`, as long as the statements you write are correct in standard Java.
• For tests you only need to provide the expression that computes the actual value, connecting it with an arrow to the expected value. For example `s.method() -> true` is sufficient.
• Remember that the phrase "develop a class" or "develop a method" means more than just providing a definition. It means to design them according to the **design recipe**. You are *not* required to provide a method template unless the problem specifically asks for one. However, be prepared to struggle if you choose to skip the template step.
• We will not answer *any* questions during the exam.

Good luck.

Problem	Points	/
1A		/ 4
1B		/ 6
1C		/ 8
1D		/ 8
Total		/26

26 Points

Problem 1

We will be dealing with lists of circles, each circle specified by its location and its radius. To make your work easier, our helpers have defined the class `CartPt` as shown below. As this is not a library class, you are allowed to add methods to this class, if you need to. The definition of the `Circle` class is also given, as are the definitions of the classes/interfaces that represent a list of `Circle`s:

```
// to represent a location on the scene
class CartPt extends Posn{
  CartPt(int x, int y){
    super(x, y);
  }

  // produce a location offset from this one
  // by the given dx, dy
  CartPt offset(int dx, int dy){
    return new CartPt(this.x + dx, this.y + dy);
  }

  // compute the distance from this location
  // to the given one
  double distTo(Posn p){
    return Math.sqrt((this.x - p.x)*(this.x - p.x) +
                     (this.y - p.y)*(this.y - p.y));
  }
}

// to represent a circle in a scene
class Circle{
  CartPt loc;
  int rad;

  Circle(CartPt loc, int rad){
    this.loc = loc;
    this.rad = rad;
  }
}
```

```
// to represent a list of circles
interface ILoCircle{}

// to represent an empty list of circles
class MtLoCircle implements ILoCircle{
  MtLoCircle(){}
}

// to represent a nonempty list of circles
class ConsLoCircle implements ILoCircle{
  Circle first;
  ILoCircle rest;

  ConsLoCircle(Circle first, ILoCircle rest){
    this.first = first;
    this.rest = rest;
  }
}
```

We also provide examples of locations and circles you may use in designing your methods:

```
  CartPt p1 = new CartPt(50, 100);
  CartPt p2 = new CartPt(-10, 100);
  CartPt p3 = new CartPt(50, 300);
  CartPt p4 = new CartPt(50, 450);
  CartPt p5 = new CartPt(450, 100);

  Circle c1 = new Circle(new CartPt(60, 40), 10);
  Circle c2 = new Circle(new CartPt(70, 20), 15);
  Circle c3 = new Circle(new CartPt(50, 90), 10);
  Circle c4 = new Circle(new CartPt(20, 20), 25);
  Circle c5 = new Circle(new CartPt(80, 40), 5);
  Circle c6 = new Circle(new CartPt(100, 20), 15);
  Circle c7 = new Circle(new CartPt(70, 80), 20);
```

A. 4 POINTS

The interface `CircleOrdering` is defined as follows:

```
// to represent an ordering of circles
interface CircleOrdering{

  // is the first circle before the second one?
  int isBefore(Circle c1, Circle c2);
}
```

It returns a negative integer if the first item is smaller, a zero if they are equal, and a positive number of the first item is larger than the second one.

Additionally, our helpers have defined the sort method for the list of circles that consumes an instance of `CircleOrdering` to determine how the list should be sorted. (You are not allowed to use this method in the remainder of the exam, we just mentioned it to explain why we may need the interface `CircleOrdering`.)

Design the class `CirclesBySize` that implements the `CircleOrdering` based on the increasing size of the circles.

Solution [POINTS 4: 2 points for definitions, 2 points for examples/tests]

```
// to represent ordering of circles by their size
class CirclesBySize implements CircleOrdering{

  // circle of smaller size comes before the larger one
  public int isBefore(Circle c1, Circle c2){
    return c1.rad - c2.rad;
  }
}

//to represent ordering of circles by their distance to the top
class CirclesToTop implements CircleOrdering{

  // circle closer to the top comes before the more distant one
  public int isBefore(Circle c1, Circle c2){
    return c1.loc.y - c2.loc.y;
  }
}
```

B. 6 POINTS

Design the method `findFirst` for the classes that represent a list of circles that produces the first circle in this (unordered) list according to the given `CircleOrdering`, for example, the smallest circle or the circle closest to the origin. There is no smallest item in an empty list, so in that case your method should throw an exception.

Your method should work for any ordering, but it is sufficient to provide examples for the ordering you have defined in the previous part.

Solution [POINTS 6: one point for purpose statement, 2 points for the method body, 3 points for examples (one has to test the exception).]

...

C. 8 POINTS

Design the method `overlap` for the classes that represent a list of circles that determines whether any two circles in this list overlap.

Solution [POINTS 8: 2 points for purpose statements, 2 points for the method body, 2 points for the helper with accumulator, 4 points for examples]

...

D. 8 POINTS

Design the method `coalesce` that produces a new list of circles from this one as follows.

The circles represent soap bubbles. When two of them overlap they are replaced by one new bigger circle with the center halfway between their two centers, and the radius that is the sum of the two radii.

Solution [POINTS 8: 2 points for purpose statements, 2 points for the method body, 2 points for the helper with accumulator, 4 points for examples]

...