1.00/1.001

Introduction to Computers and Engineering Problem Solving

Spring 2011 - Quiz 2

Name:	
MIT Email:	
TA:	
Section:	

You have 80 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary packages have already been imported. Do not write outside of the boxes provided for your answers throughout this exam packet or modify provided examination code. Any modifications to provided code will result in zero credit for that problem. Good luck!

Question 1	**/ 45**
Question 2	**/ 40**
Question 3	**/ 15**
Total	**/ 100**

Question 1 – Inheritance (45 Points)

1a. Complete the class below, following the instructions in the comments:

```
public abstract class Aircraft {

/*Aircraft Class Data Members: Declare the three private data
members:  wing span (double), the serial number (int), and the
number of aircraft objects in the program (int). */
```

```

```

```
/*  Aircraft Constructor: Write the constructor. It must
initialize the instance data members, as usual.  The constructor
must increase the number of aircraft objects by 1. */

public Aircraft(int serial, double wingspan) {
```

```

```

```
}

// Aircraft class continued on the next page.
```

/* **Complete the remaining methods for the Aircraft class:** The aircraft class must have get methods for all instance variables, and a boolean method certified(). The certified method will vary based on the subclass of Aircraft. */

// get() methods:

// certified() method:

} //End of Aircraft Class.

1b. You are given the Federal Aviation Administration (FAA) certification interface.

```
public interface FAA {

        double MAXGFORCE = 5.5;

        double getGforce();

}
```

Complete the European Aircraft Safety Agency (EASA) Interface. The EASA interface must include one constant that corresponds to the allowable CO_2 emissions level: 25.9 ppm. Use a double to represent this constant in ppm. The interface must also include a method for setting the vehicle's actual CO_2 emissions level.

```
public interface EASA {
```

```
}
```

1c. Complete the concrete B737 subclass. This subclass represents the Boeing 737 aircraft. It must get both EASA and FAA certification. You must write the appropriate class declaration, data members and methods to ensure compliance with both the EASA and FAA interfaces, and correctly inherit from the aircraft abstract class. The B737 is considered certified if it does not exceed any of the maximum values in the certification interfaces (EASA and FAA). You do not need to consider tolerances of doubles when determining if B737 is certified.

```
public class B737 [                                    ]{
```

```
} // End of B737 Class.
```

Question 2 – Swing (40 points)

In this question, you will complete a raindrop simulation. Your program will allow users to add raindrops (modeled as circles) with specified x, y coordinates to the view. The design uses the model-view-controller pattern from lecture. The controller class is written for you; it compiles and runs correctly:

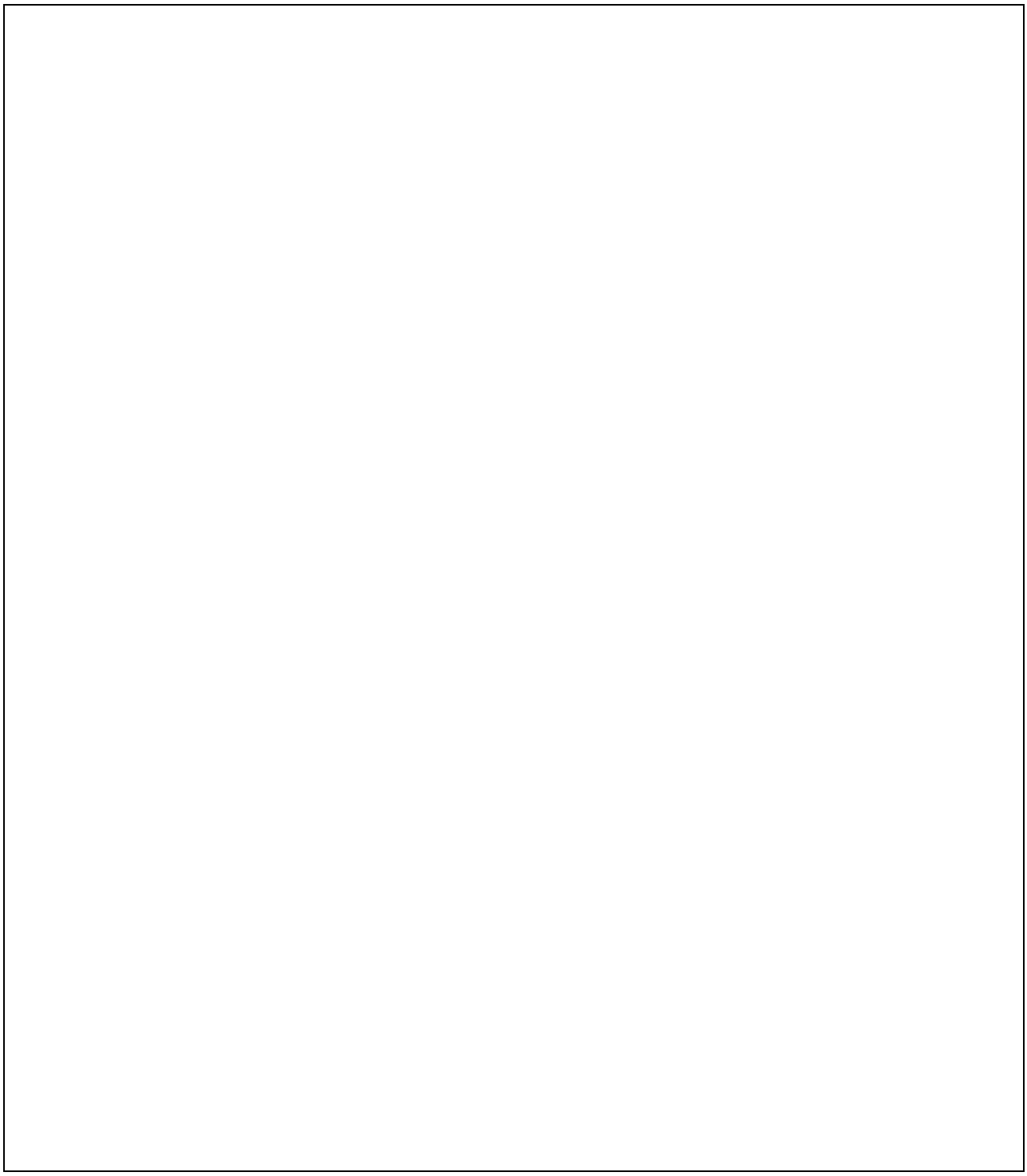
```
public class RainDropController extends JFrame{
     private RainModel model;
     private RainDropView view;
     private JTextField tx1,tx2;
     private JLabel labelA, labelB;
     private JPanel inputs;
     public static final int SIZE = 500;

     public RainDropController(String n){
          super(n);
          model = new RainModel();
          view = new RainDropView(model);
          view.setPreferredSize(new Dimension(SIZE,SIZE));
          inputs = new JPanel();         //Default flow layout
          Container cp =getContentPane();//Default border layout
          cp.add(inputs,BorderLayout.NORTH);
          cp.add(view,BorderLayout.SOUTH);

          tx1 = new JTextField(10);
          tx2 = new JTextField(10);
          labelA = new JLabel("X:");
          labelB = new JLabel("Y:");
          inputs.add(labelA);
          inputs.add(tx1);
          inputs.add(labelB);
          inputs.add(tx2);

          JButton addBTN = new JButton("Add Drop");
          JButton simBTN = new JButton("Simulate Step");
          inputs.add(addBTN);
          inputs.add(simBTN);



//RainDropController Continued on Next Page.
```

```
            addBTN.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e) {
                    int x = Integer.parseInt(tx1.getText());
                    int y = Integer.parseInt(tx2.getText());
                    model.addDrop(x, y);
                    view.repaint();
                }
            });

            simBTN.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e) {
                    model.simulate();
                    view.repaint();
                }
            });
            pack();
        }

        public static void main(String[] args) {
            JFrame fr = new RainDropController("Rain Drop");
            fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            fr.setVisible(true);
        }
}
```

A raindrop is modeled as a shape with x and y coordinates for its center:

```
public class Drop {
    private int x, y;
    public Drop(int xx, int yy){
        x= xx;
        y= yy;
    }
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }
    public void setX(int xx){
        x= xx;
    }
    public void setY(int yy){
        y= yy;
    }
}
```

2.a Please draw the JFrame window as it initially appears. Assume nothing is originally drawn in the `view` Panel. Assume the model and view objects exist, so the program compiles and runs.



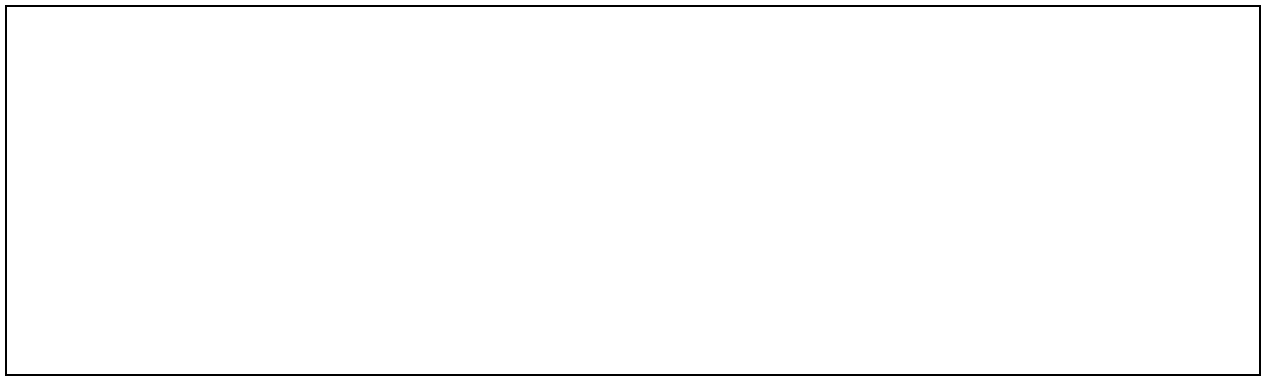
2.b Please complete `addDrop(int x, int y)` in the `RainModel` class. The `addDrop(…)` method takes two integer inputs: the x and y coordinates for the center of the new raindrop that is to be added to the `listOfDrops`.

```
public class RainModel {
     private ArrayList<Drop> listOfDrops;
     public static final int YDIFF = 5;
     public static final int YMAX = 500;
     public RainModel(){
          listOfDrops = new ArrayList<Drop>();
     }
     public ArrayList<Drop> getList(){
          return listOfDrops;
     }

     public void addDrop(int x, int y){
```

```
     }
```

/***2.c** The `simulate()` method updates the positions of all raindrops within the model class. In this simplified world, each raindrop falls vertically by `YDIFF` pixels every time the "Simulate Step" button is pressed. Complete `simulate()` using this simplified physics law. Also, if the center of a raindrop goes below the bottom edge of the view, reset its y coordinate value to zero. (`YMAX` is specified for you as a boundary condition.)*/

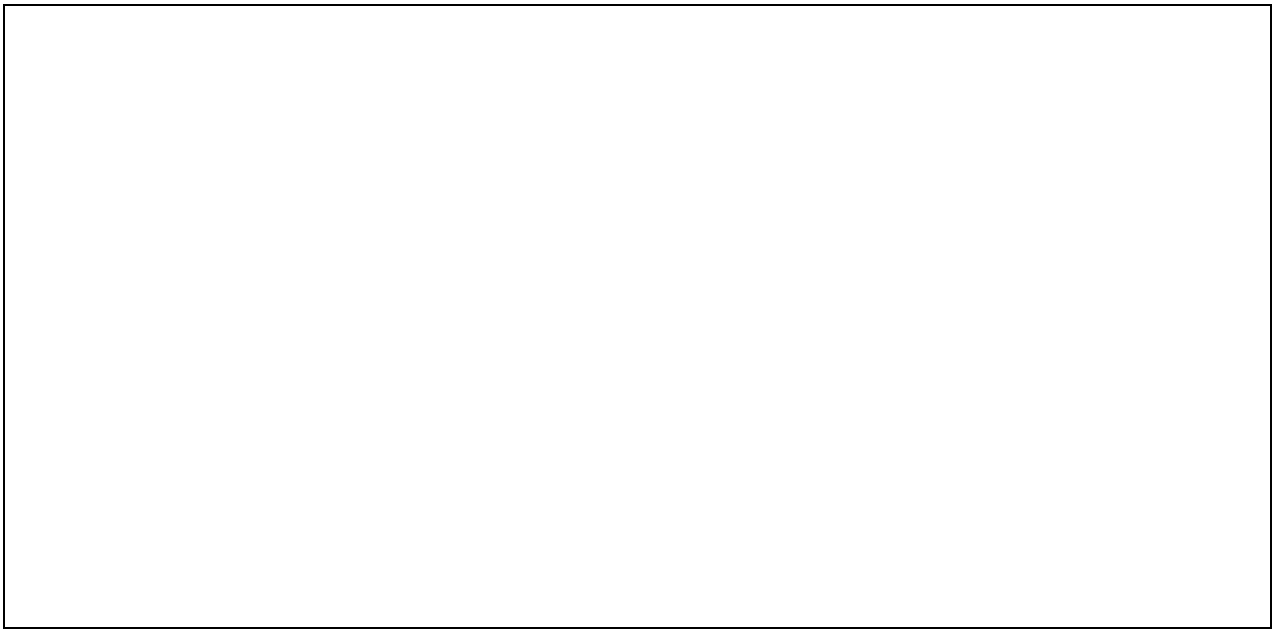
```
     public void simulate(){
```



```
     }
}
```

2.d Please complete the `paintComponent(Graphics g)` method in `RainDropView`. The method draws all raindrops on the view panel. Notice that the (x, y) coordinate defined in the Drop class corresponds to the center of each raindrop.

```
public class RainDropView extends JPanel{

     private RainModel model;
     public static final int R=5; // radius for raindrop

     public RainDropView(RainModel m){
          super();
          model = m;
     }


     public void paintComponent(Graphics g){

          super.paintComponent(g);
          Graphics2D g2 = (Graphics2D)g;
```



```
     }

}
```

Question 3 – Recursion (15 points)

row	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1
	0	**1**	**2**	**3**	**4**	**5**
	column					

The triangular array above is known as *Pascal's Triangle*. The first and last term of each row is 1, and every other term on a row is equal to the sum of the term directly above and the term to the left of the term directly above. For example, the terms on row 3 are computed from the terms on row 2 as follows:

row 2	1	2	1	
row 3	1	1+2=3	2+1=3	1

Write a **recursive** method that returns the term located at row r and column c in Pascal's Triangle. You may assume that r ≥ 0, c ≥ 0 and c ≤ r.

```
public static int pascalTerm(int r, int c) {
```

```

```

```
}
```

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012