CS 134 Midterm
Fall 2012

This is a *closed book* exam. You have 90 minutes to complete the exam. All intended answers will fit in the spaces provided. You may use the back of the preceding page for additional space if necessary, but be sure to mark you answers clearly.

Be sure to give yourself enough time to answer each question— the point values should help you manage your time.

Problem	Points	Description	Score
1	16	GUI Programming	
2	15	Loops	
3	18	Classes	
4	12	Conditions	
5	15	Recursion	
6	14	Declarations	
Total	90		

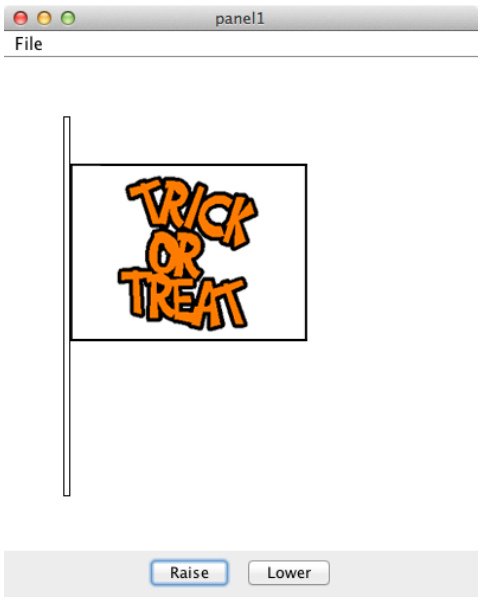
I have neither given nor received aid on this examination.

Signature: _____

Name: _____

1. (*16 points*) ..... GUI Components

We want you to write a class that raises a flag. Here is a picture of what the window should look like.



When you press the *Raise* button, the flag climbs the pole. When you press the *Lower* button, the flag descends the pole. You do not need to worry about the flag extending beyond the pole or the flag falling below the pole. In other words, continually hitting the *Raise* button will continue to raise the flag regardless of its height. Continually hitting the *Lower* button will continue to lower the flag regardless of its height.

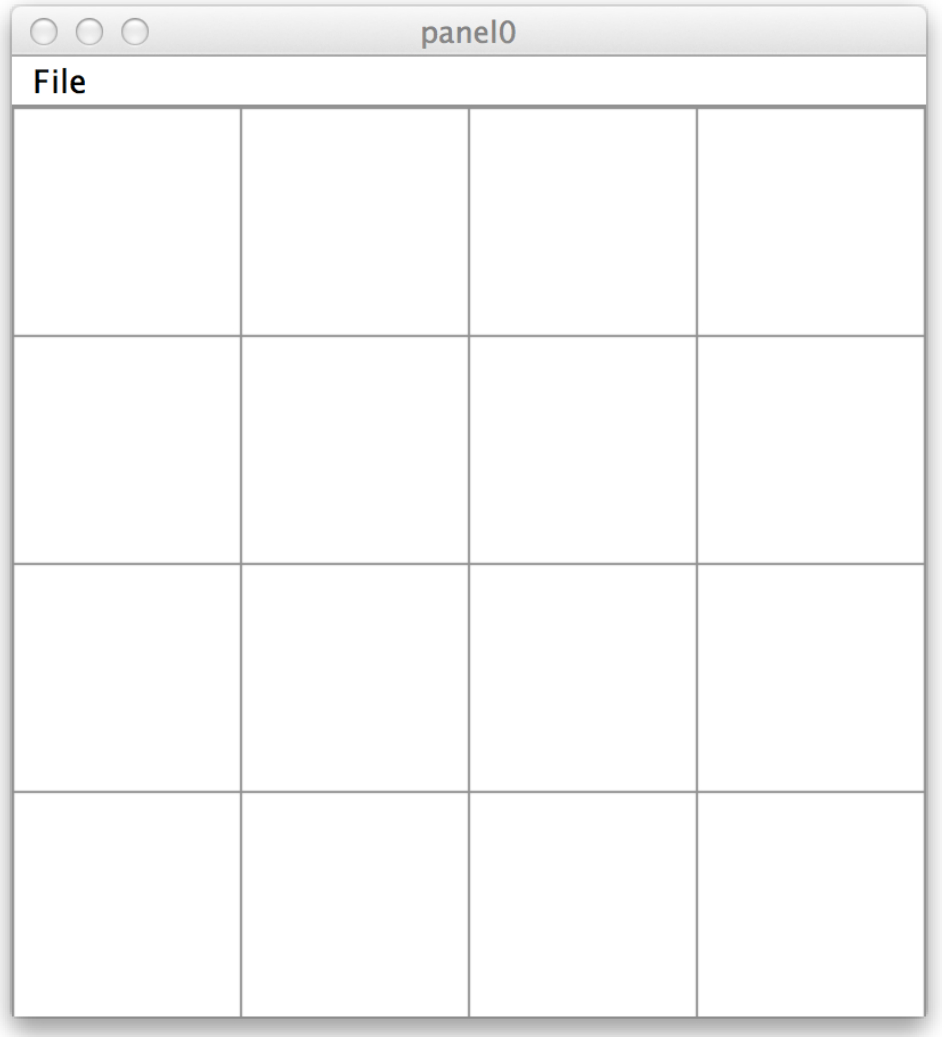
We have provided code to display the flag and code to add buttons to the display, but you must write the rest of the `FlagRaising` class so that the program has the correct behavior when someone clicks on the buttons. If you add any additional methods to the class, write them in the space provided.

```
public class FlagRaising extends WindowController
{
    // Pole Location and Dimensions
    private final static int POLE_X = 50;
    private final static int POLE_Y = 50;
    private final static int POLE_WIDTH = 5;
    private final static int POLE_HEIGHT = 320;

    // Flag Location and Dimensions
    private final static int FLAG_HEIGHT = 150;
    private final static int FLAG_X = POLE_X+POLE_WIDTH+1;
    private final static int FLAG_Y = POLE_Y + POLE_HEIGHT/2 - FLAG_HEIGHT/2;

    // Amount by which flag should move
    private final static int STEP = 5;

    private VisibleImage flag;
    private JButton raise, lower;

    public void begin() {
        new FramedRect(POLE_X, POLE_Y, POLE_WIDTH, POLE_HEIGHT, canvas);
        flag = new VisibleImage(getImage("flag.gif"), FLAG_X, FLAG_Y, canvas);

        raise = new JButton("Raise");
        lower = new JButton("Lower");




        JPanel p = new JPanel();
        p.add(raise);
        p.add(lower);
        getContentPane().add(p, BorderLayout.SOUTH);

        validate();
    }









}
```

2. (*15 points*) ..... Loops

Below is a simple but complete Java program that draws a picture. In the window below, draw a sketch of the picture it produces. You may assume that the canvas in the window is 400 pixels wide and 400 pixels tall.

```
public class Mystery extends WindowController {

    public void begin() {
        double w = 100;
        while (w > 0) {
            int h = 3;
            while (h > 0) {
                new Line(100+w+2*h, 200, 100+w+2*h, 200-w, canvas);
                h = h - 1;
            }
            w = w - 25;
        }
    }
}
```



3. (*18 points*) ..... Classes

We want you to write a class `Door` that models the door to a haunted house.

Your door won't look exactly like the one above, but it will be similar. Figure 1 on the next page gives a picture of the door when it's closed followed by a picture of the door when it's open.

We have provided starter code on pages 6–8. The code has open spaces for instance variables and method bodies. You will fill in these spaces with appropriate code. We have also provided a `HauntedHouse` class on page 9 that extends `WindowController`. This class shows how we might use the `Door` class. You will not add any code to the `HauntedHouse` class.

You need to declare instance variables for the door and handle. Then add code to the constructor to place the door and handle on the screen. We have already written code in the constructor to create the gray passage and eyes hidden by the door, and a black frame around it. The constructor takes the `x` and `y` coordinates for the top-left corner of the door and creates it accordingly.

Like most doors, your door can be opened by clicking on the handle, and the door can be closed by clicking anywhere on the door, including the handle. Nothing should happen when trying to open an already open door. Similarly, trying to close an already closed door by clicking on the door should have no effect. To support this functionality, the `Door` class must implement an `openDoor` method and a `closeDoor` method as well as `isDoorOpen`, `handleContains`, and `doorContains` methods. Fill in those method bodies below making sure to add any instance variables you need to complete the method.

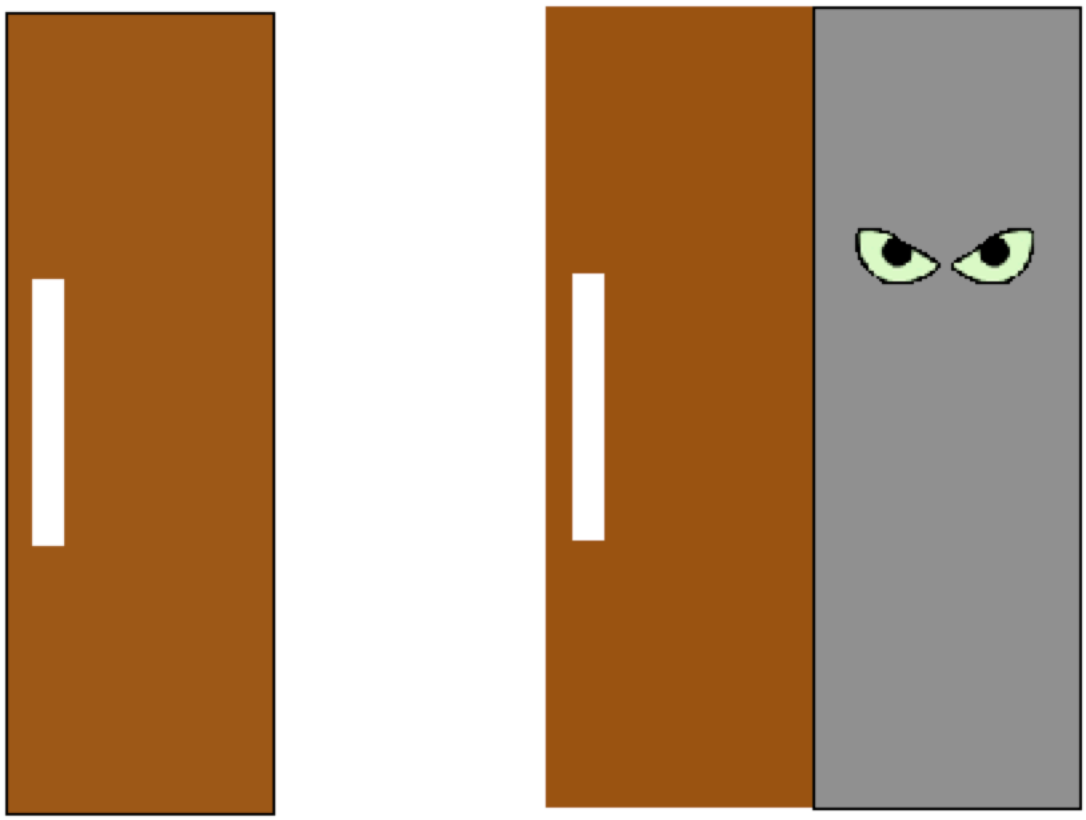


Figure 1: *Left:* A closed door. *Right:* An open door.

```
public class Door {

    // dimensions and color of the door
    private static final int DOOR_WIDTH = 100;
    private static final int DOOR_HEIGHT = 300;
    private static final Color DOOR_COLOR = new Color(139, 69, 19);

    // Position of eyes relative to top left corner of door
    private static int EYE_X_OFFSET = 15;
    private static int EYE_Y_OFFSET = 25;

    // dimsions of the handle
    private static final int HANDLE_WIDTH = DOOR_WIDTH/8;
    private static final int HANDLE_HEIGHT = DOOR_HEIGHT/3;

    // Position of handle relative to top left corner of door
    private static int HANDLE_X_OFFSET = DOOR_WIDTH/10;
    private static int HANDLE_Y_OFFSET = DOOR_HEIGHT/3;
```

```
// Instance Variables:









public Door(double x, double y, Image eyesImage, DrawingCanvas canvas) {

    // make door passage way and eyes first so door covers them when "closed"
    new FilledRect(x, y, DOOR_WIDTH, DOOR_HEIGHT, canvas).setColor(Color.gray);
    new VisibleImage(eyesImage, x + EYE_X_OFFSET, y + EYE_Y_OFFSET, canvas);









    // make frame last so it is always showing
    new FramedRect(x, y, DOOR_WIDTH, DOOR_HEIGHT, canvas);
}
```

```
    public void openDoor() {









    }

    public void closeDoor() {









    }

    public boolean isDoorOpen() {


    }

    public boolean handleContains(Location pt) {


    }

    public boolean doorContains(Location pt) {


    }

}
```

```
public class HauntedHouse extends WindowController {

    final private static int DOOR_X = 150;
    final private static int DOOR_Y = 50;

    final private static int TEXT_X = DOOR_X + 10;
    final private static int TEXT_Y = DOOR_Y - 25;

    private Text message;
    private Door door;

    public void begin() {
        door = new Door(DOOR_X, DOOR_Y, getImage("eyes.gif"), canvas);
        message = new Text("", TEXT_X, TEXT_Y, canvas);
    }

    public void onMouseClick(Location pt) {
        if (door.handleContains(pt)) {
            door.openDoor();
        } else if (door.doorContains(pt)) {
            door.closeDoor();
        }

        if (door.isDoorOpen()) {
            message.setText("BOO!");
        } else {
            message.setText("");
        }
    }
}
```

4. (*12 points*) ..... Conditionals

Below you will find several code fragments involving the use of if statements. Each of these fragments can be improved stylistically in some way. For each fragment, show how to write code that is more clear (and possibly more concise) but that performs exactly the same function. You may assume any variables referenced by this code are declared with appropriate types.

(a)
```
public void onMousePress(Location point) {
        if (ghost1.contains(point)) {
           inAGhost = true;
        } else if (ghost2.contains(point)) {
           inAGhost = true;
        } else {
           inAGhost = false;
        }
    }
```

(b)
```
if (point.getX() < left) {
        pumpkinVelocity = 1;
    } else if (point.getX() >= left && point.getX() < right) {

    } else if (point.getX() >= right) {
        pumpkinVelocity = -1;
    }
```

(c)
```
public void onMouseDrag(Location pt) {
      if (dragging && pt.getX() < LEFT) {
         witch.setColor(Color.red);
      }
      if (dragging && pt.getX() >= LEFT && pt.getY() < BOTTOM) {
         witch.setColor(Color.blue);
      }
   }
```

5. (*15 points*) ..... Recursion

In this question, you will write several methods to model a colony of bats. We provide a class `Bat` with several special methods that you will use below:

```
public class Bat {
    // Return the x and y coordinates of the bat, respectively.
    public double getX()
    public double getY()

    // Change the color of the bat.
    public void setColor(Color c)

    // Return true if the bat overlaps the image on the canvas.
    public boolean overlaps(VisibleImage image)

    // Make the wings move and the bat become a bit "fainter".
    public void flap()
}
```

We define a colony of bats recursively using `ColonyInterface`, `EmptyColony`, and `NonEmptyColony`. We have written the interface and parts of the code below. Your job will be to implement the three methods listed in the interface for the two classes.

```
public interface ColonyInterface {

    // Make every bat in colony flap its wings.  See part (a).
    void fly();

    // Return true if any bat is higher in the sky than point.  See part (b).
    boolean anyAbove(Location point);

    // Have bats overlapping the eggplant eat it, but
    //   only let at most "max" bats eat.   See part (c).
    void eatEggplant(VisibleImage eggplant, int max);
}

public class EmptyColony implements ColonyInterface {

    public EmptyColony() {  }

}

public class NonEmptyColony implements ColonyInterface {

    private Bat first;               // first bat in the swarm
    private ColonyInterface rest;    // the rest of the swarm

    // Create a larger colony from a Bat and an existing colony.
    public NonEmptyColony(Bat bat, ColonyInterface theRest) {
        first = bat;
        rest = theRest;
    }

}
```

Your job is to implement the methods listed in the `ColonyInterface`:

(a) `void fly()`: Makes every bat in the colony fly by flapping its wings.
Complete `fly` for `EmptyColony` and `NonEmptyColony`:
In EmptyColony:

```
public void fly() {

}
```

In NonEmptyColony:

```
public void fly() {

}
```

(b) `boolean anyAbove(Location point)`: Returns true if any bat is higher in the sky than the given location.
Complete `anyAbove` for `EmptyColony` and `NonEmptyColony`:
In EmptyColony:

```
public boolean anyAbove(Location point) {

}
```

In NonEmptyColony:

```
public boolean anyAbove(Location point) {

}
```

(c) `void eatEggplant(VisibleImage eggplant, int max)`: As our bats fly, they grow hungry and gradually fade away. They become solid black again when they eat. This method tries to feed each bat in the colony their favorite food: Eggplant. Specifically, a bat that overlaps the eggplant image parameter is allowed to eat. You may invoke `setColor(Color.black)` on a bat to make it solid black again.

After "max" bats have eaten, there is no eggplant left for other bats. So, even if more than "max" bats overlap the eggplant, only "max" of them should be allowed to eat. Thus, invoking

```
colony.eatEggplant(eggplantImage, 3)
```

will let at most three bats overlapping the image eat. Any remaining bats overlapping the image will go hungry.

In our demo, the `WindowController` calls `eatEggplant` on the colony each time the mouse is released after dragging the eggplant image around the canvas.

Complete `eatEggplant` for `EmptyColony` and `NonEmptyColony`.

In EmptyColony:

```
public void eatEggplant(VisibleImage eggplant, int max) {









}
```

In NonEmptyColony:

```
public void eatEggplant(VisibleImage eggplant, int max) {









}
```

6. (*14 points*) ..... Declarations

Directions We have included below the (almost) complete code for a program that shows a ghost floating around the canvas. The program is nearly complete, but if you examine the classes carefully, you will notice that although the following names are used, they are never declared:

GhostController:	**GhostMover:**
`ghost`	`changeDir`
`ghostMover`	`dir`
`speedChoice`	`newSpeed`
`choice`	
`speed`	
`event`	

(To make it easier to find the uses of these names within the program, we have underlined them). We would like you to add the declarations for the names shown above that are needed to make the class complete. In doing this remember:

(a) The only modification you should make to the code is the addition of declarations.

(b) You should declare each name in the most appropriate way (i.e., as an instance variable or as a variable local to a method or as a formal parameter to a method).

(c) You should make each name's declaration as local as possible while making the program correct.

We have attempted to leave enough extra space in each area of the program where declarations could be added to enable you to write the declarations you believe are needed in their appropriate places.

Program Overview The first is a `WindowController` that creates a ghost image, a `GhostMover` object, and a GUI component for controlling the speed of the ghost's hovering. The `GhostMover` class extends `ActiveObject`. Its constructor is passed a ghost image to make hover, and its `run` method makes that image move up and down. The `GhostMover` simply remembers whether the ghost is moving up or down. On each loop iteration, it moves the ghost's image a small amount in the right direction. It uses a random number generator to occasionally switch directions. The choice menu in the `WindowController` adjusts the speed at which the image moves.

```
// Creates a ghost and sets it in motion.  The ghost's
// hover speed is adjusted with a combo box.
public class GhostController extends WindowController implements ItemListener {
    // Initial location of the ghost, and the hovering speeds
    public static final Location GHOST_LOC = new Location(200,200);
    public static final double SLOW = 0.002;
    public static final double MEDIUM = 0.02;
    public static final double FAST = 0.04;









    public void begin(                                          ) {


        // Create the speed combox box
        speedChoice = new JComboBox();
        speedChoice.addItem("Slow");
        speedChoice.addItem("Medium");
        speedChoice.addItem("Fast");
        speedChoice.addItemListener(this);

        Container container = this.getContentPane();
        container.add(speedChoice, BorderLayout.SOUTH);
        container.validate();

        // Make the ghost and start it moving
        ghost = new VisibleImage(getImage("ghost.gif"), GHOST_LOC, canvas);
        ghostMover = new GhostMover(ghost, SLOW);
    }

    // Adjust the speed of the ghost when the combo box changes.
    public void itemStateChanged(                                    ) {


        if (event.getSource() == speedChoice) {


            choice = speedChoice.getSelectedItem().toString();
            if (choice.equals("Slow")) {

                speed = SLOW;
            } else if (choice.equals("Medium")) {
                speed = MEDIUM;
            } else {
                speed = FAST;
            }
            ghostMover.setSpeed(speed);
        }
    }

    public void onMouseMove(Location pt                                  ) {

        ghost.move(pt.getX() - ghost.getX(), 0);
    }
}
```

```
// An active object that makes a ghost float around the screen.
// The floating speed can be changed via a mutator method.
public class GhostMover extends ActiveObject {

    private static final int PAUSE_TIME = 30;
    private static final int VELOCITY_SCALE = 100;

    private double speed;
    private VisibleImage ghost;




    public GhostMover(VisibleImage aGhost, double aSpeed) {

        ghost = aGhost;
        speed = aSpeed;
        this.start();
    }

    public void run(                                        ) {


        changeDir = new RandomIntGenerator(0,50);

        // The ghost bobs up and down.  It uses dir to remember
        // if it is going down or up, and occasionally changes
        // its dir.
        dir = 1;

        while (true) {


            pause(PAUSE_TIME);


            ghost.move(0, speed * dir * VELOCITY_SCALE);

            if (changeDir.nextValue() == 0) {
                dir = dir * -1;
            }

        }
    }

    // Change the hovering speed.
    public void setSpeed(                                           ) {

        speed = newSpeed;
    }

}
```