MAT 2170: Laboratory 10

Key Concepts

1. Derived classes and inheritance
2. More on methods and animation

Instructions

- Because we will be creating several packages that will be used for more than one assignment, we will be creating a `myLibrary` project with several packages inside it as our own library. This week, you'll be adding `graphics` and `testing` packages, later we'll add a `games` package.

  As usual, the program projects (`PlayPool` and `Fireworks`) should be in a `lab10` directory.
- **Reminder: you are to do your own work** in this course. Sit across the lab table from your friend/s so you aren't tempted to look at their code or ask for help before you've even thought about a problem.
- The final two exercises, the pool table and the fireworks, are not due until week 12 (two weeks), and this lab is worth 50 points. This should be a hint to *start early*, especially since there **will** be a new lab next week.
- Part of your grade will depend on how well you document your work with comments. You will need to expand the comments from those provided, deleting any "*replace this code*" type comments. Be sure to include header comments in *all* files.
- The `Square` classes are to help you get started. Use them as a basis for the `Circle` classes, which are used first in the pool table animation, then expanded for the fireworks. Hence, **follow directions carefully**, and make sure you have the `Square` classes implemented correctly before you copy them to create the `Circle` classes.
- When you finish a task, check it off in the box in the margin. If you need to ask for help with a task, *all* of the boxes before that task should be checked — meaning you have completed them successfully.

Notes

1. The `GPoint` class is described both in your book (pages 301–302), and in the online documentation referenced in this week's slides.
2. About `getWidth()` and `getHeight()`:

   When these methods have been called in the `run()` method, the nearest containing class has been the graphics window. Thus, they returned the width and height of the window (as integers).

   When they are used inside other graphics class methods, however, they refer to the width and height of that object. An example can be seen on page 302 in your text — the `GRect` class contains its own `getWidth()` and `getHeight()` methods. Since they are `public`, they can also be accessed by sending a message to an instance of the class, for example, `R.getWidth()`.

   The graphics window(s) belong to the `GCanvas` class in the `acmLibrary`. If we need to know the window's width or height while we're inside another graphics class, we need to include the window as the receiver of the `getWidth()` or `getHeight()` message. To make this as straight–forward and the least time–consuming as we could, the smart classes will know to what `GCanvas` they belong. This information is stored in the data member `myCanvas`.
3. The `GCanvas` class contains the `remove(GObject obj)` method, which is directed to a graphics window. This method erases the given object `obj` from the window. For example, if `ball` is a `GMovingCircle`, we can use: `remove(ball);`

1 Setting up the directories and projects

1. In your `mat2170` directory, create a new project, `myLibrary`, a Java → Java Class Library. (This should not be in a `labX` directory.) Within this project, create two **packages** by right–clicking on Source Packages and selecting New → Java package, then naming them:
   (a) `graphics`
   (b) `testing`
2. Add the `acmLibrary` jar file to the library of project `myLibrary`.
3. Download `GSquare.java`, `GSmartSquare.java` and `GMovingSquare.java`, placing them in your `myLibrary/src/graphics` directory. You should be able to see the source files you downloaded in the `myLibrary.graphics` package in NetBeans.
4. Download `TestSmartSquare.java`, `TestMovingSquare.java`, `TestCircle.java`, `TestGSpark.java`, and `TestGEmitter.java`, placing them in your `myLibrary/src/testing` directory. You should be able to see the source files you downloaded in the `myLibrary.testing` package in NetBeans.
5. Within a `lab10` directory, create a new project, `PlayPool`, then download `PlayPool.java`, and place it in the default directory. Add the `acmLibrary` jar file to the library of this project.
6. Create another new project, `Fireworks`, still in the `lab10` directory, then download `Fireworks.java` and place it in the default directory of this project. Add the `acmLibrary` jar file to the library of this project.

2 Making Squares Smart

7. In the `GSmartSquare` class, complete the methods `overflowsWindowVertically()` and `fitsInItsWindow()`. **Be sure** to use the "overflows" methods to complete the "fits" method.
8. Add documentation comments for `overflowsWindowVertically()` and `fitsInItsWindow()`, and update the header comments.
9. Complete the `TestSmartSquare.java` program by adding code which changes the square's color to red if it extends off the right or left edge, blue if it extends off the top or bottom edge, and yellow if it extends past both a top or bottom edge **and** a left or right edge. Make sure your implementations work by running the `TestSmartSquare.java` program.

3 Making Squares Move

10. Complete the `move()` method in the `GMovingSquare` class so the square bounces off the sides of the window. Make use of the "overflows" methods, and the `GMovingSquare` method `setDisplacement()` to accomplish the "bounce" effect. Test your work by running the `TestMovingSquare.java` program.
11. By this point, you have completed, annotated, and tested the `GSmartSquare` and `GMovingSquare` classes. This lays the foundation for nearly identical class based on `GOval`: the `GCircle`, `GSmartCircle` and `GMovingCircle` classes.

4 Making Circles Move

1. Create `GCircle`, `GSmartCircle` and `GMovingCircle` classes with the same functionality as their `Square` counterparts. These classes should be placed in the `myLibrary.graphics` package. You may find that copy–paste paired with EDIT → REPLACE are your friends.

2. In the `myLibrary.testing` package, create a testing program, `TestMovingCircle`, for `GMovingCircle`. It should function in a manner similar to `TestMovingSquare`. Make sure your `GCircle` classes have been created properly before proceeding.

5 Modify the classes for the Pool Game

1. In preparation for making moving circles which "fall into" pool table pockets and become inactive, we need to add a few more members to the `GCircle` class. Be sure to document these methods.

   - Add method `getRadius()` to return the radius of the `GCircle` object that receives the message.
   - Add method `getCenter()` to return the center coordinates (as a `GPoint`) of the `GCircle` object receiving this message. This method should utilize `getRadius()` in its implementation.
   - Add method `captured(GCircle c)` to return `true` when the center point of the parameter `GCircle c` is inside the receiver `GCircle` object and `false` otherwise. Note that you can calculate the distance between circle centers, then compare that distance to receiver's radius to determine whether the receiver has captured `c`. This method should use both `getRadius()` and `getCenter()` in its implementation.

2. Test your new methods by running `TestCircle`. The randomly drawn circle should turn red when its center is inside the black circle at the center of the window, and be green otherwise.

6 Creating the Minimal Game

Pool Table Algorithm

You will be implementing a pool table simulation where pool balls bounce around a table until they fall into a pocket. The general algorithm which you'll be implementing in the steps following is:

```
set activeCount to the slider value
createTable
createPockets
add activeCount poolballs to the window

while there are still active poolballs on the table
  for every object in the window
    if it's a poolball
       if it's fallen into a pocket
          remove it from the window
          decrement activeCount
       otherwise
          move the poolball
    otherwise
       ignore it: it's a pocket, not a poolball
  pause
tell user the game is over
```

Setting up the pool table

- (a) Inspect the program `PlayPool.java`. Build and run it. You should see three balls bouncing around the window. If not, track down the problem.
- (b) In `PlayPool`, implement and then call the method `createTable()`, to cover the table in green felt. (It should use a green `GRect` to make the window appear more like a pool table.)
- (c) Add the method `createPocket(double x, double y)`, which will create a black `GCircle` *centered* at the given coordinates and 15% of the window's width.
- (d) In the `run()` method of `PlayPool` use `createPocket()` to create pockets centered on each of the four corners of the window. The pockets you create should be named, `pocket1`, `pocket2`, `pocket3`, and `pocket4`. Add the pockets to the window.

Adding more circles to the game

In this final section for the Pool Table, you will change the program to a `SliderProgram`, create the number of balls indicated by the slider, and have them bounce around the window until they fall into a pocket. Execution should continue as long as there is still at least one active poolball in the window. In order to accomplish this:

1. Modify `PlayPool` so it extends `SliderProgram`, add an `init()` method which set the window size to `700 by 450`, and sets the slider range to $3 - 15$. Declare a variable named `activeCount` which will store the number of active balls on the table. Initialize `activeCount` to the slider value. Modify the loop which adds the balls to the window so it adds `activeCount`, rather than 3, balls. Test this works.
2. Next, modify `PlayPool` by adding the method `createBall()`, which will create and add a poolball to the window (but not return it since we don't need to name it). All of the attributes (location, direction of movement, etc.) should be determined exactly the way they were in the original `PlayPool`, while the color should be determined randomly. Modify `run()` by replacing the body of the loop that adds the balls with a call to your `createBall()` method instead. Test this works.
3. In preparation for the next step, add a predicate method to `PlayPool` which indicates whether a poolball has been captured by one of the pockets. This method may be simplified by utilizing the `captured()` method you added to the `GCircle` class.
4. Inside the loop that moves the balls check to see if the ball is inside one of the pockets. If it is, remove it from the window and decrement `activeCount`. Otherwise move the ball. In other words implement the algorithm:

```
if ball is inside one of the pockets
  remove ball
  decrement activeCount
else
  move ball
```

5. Currently the program moves the balls `MAX_MOVES` times. Modify the program so it halts when all the moving circles have fallen into a pocket.

Creating Fireworks

1. Derive a new class from the `GMovingCircle` class, `GSpark`, which acts like a spark from a fire, gradually sinking down the window and extinguishing itself. `GSpark.java` should be created in the `myLibrary.graphics` package.

This class should have a new data member, `lifetime`, which indicates the number of moves allotted to the object before it should remove itself from the window. In addition, several constants will be useful.

☐ The class will need a constructor, and will need to override the `move()` method as follows:

- If the spark goes out the top or bottom of its window, it should be removed.
- If the spark has used up all its lifetime moves, it should be removed.
- If the spark is still active
  (a) Slow down movement in the x direction to 85% of the previous Δx.
  (b) If the spark is headed upward, slow it down by setting Δy to 75% of its previous value, otherwise replace Δy with $|\Delta y| \times 1.25$ (to speed up it's movement toward the bottom of the window).
  (c) Make it move and decrement the lifetime

Don't forget to document your code.

☐ 2. Test your `GSpark` class with the `TestGSpark.java` program. Track down any errors and fix them before continuing.

☐ 3. Create a new class, *not* derived from another class, `GEmitter`, in the `myLibrary.graphics` package. This class will have a constructor and a `pulse()` method which pumps out `GSparks`, making it look like a sparkler.

`GEmitter` requires storage for: its position in the window, the number of sparks to emit, the color all the sparks it creates should be, what `GCanvas` it is associated with, and the maximum lifetime for the sparks it will create. A constructor should initialize all these data members. In addition, the object should have its own random generator object.

☐ 4. Add a `pulse()` method to the `GEmitter` class to produce the number of `GSparks` it is suppose to, where:
Sparks will be restricted to having:

- a radius of 1, 2, or 3
- a lifetime of at least 25 moves up to the maximum lifetime this emitter is allowed
- a Δx in the range -20.0 to 20.0
- a Δy in the range -10 to -100.
  (a) If $\Delta y < -75$, decrease Δx to 65% of itself.
  (b) If the y position of the emitter is within 150 pixels from the top of the window, add 50 to Δy
- and the same color all the other sparks from this emitter has

☐ 5. Test your `GEmitter` class using `TestGEmitter.java`. Fix any problems before continuing. Don't forget to document your code.

☐ 6. Complete the `Fireworks.java` program by:

(a) Filling the window with a black background
(b) In five percent of the passes through the loop to move `GSparks`, create a `GEmitter` and make it pulse. The number of sparks it should create is based on SliderA, and the upper bound on the lifetime of one of its sparks is based on 10 * SliderB. The position of the emitter should be no closer than 10 pixels from a left or right edge, and 30 from the top or bottom of the window. The color it is to emit should be randomly chosen from red, yellow, blue green, orange, and magenta.

Be sure you have added header comments and updated block comments in all files to reflect any changes made, and that your name is in the header comments in all files. If you wish to add innovations, we encourage you to do so.

Finishing Up

1. Print the `circle`, `pool`, and `fireworks` files as listed below, staple them together in the order listed, and hand in at the *beginning of Lab 12*:

   (a) `PlayPool`
   (b) `Fireworks`
   (c) `GSpark`
   (d) `GEmitter`
   (e) `GCircle`
   (f) `GSmartCircle`
   (g) `GMovingCircle`

2. Submit electronic copies of both your `myLibrary` project (with the `graphics` and `testing` packages) and `lab10` directory.

3. To publish the `PlayPool` and `Fireworks` programs to your web site, you'll first need to move a copy of the `myLibrary.jar` file to your `www` directory. See the handout from Week 1 when we did this with the `acmLibrary.jar` file. Then publish these projects to your web site. Let me know if you have difficulty.

7 Appendix

7.1 `GSquare` Class Outline

The `GSquare` class has been completed for you, but you will still want to familiarize yourself with its contents and the method implementations.

```
public class GSquare extends GRect
{ // Constructors:
  public GSquare(double x, double y, double size, Color color)...
  public GSquare(double x, double y, double size)...        // default color is black
  public GSquare(double size)...              // default location 0.0, 0.0, and black

  // The following force the GSquare to maintain its square shape when resized.
  //        setSize() and setBounds() override parent methods
  public void setSize(double size)...
  public void setSize(double width, double height)...   // choose smaller for size

  public void setBounds(double x, double y, double width, double height)...
  public void setBounds(double x, double y, double size)...
}
```

7.2 `GSmartSquare` Class Outline

The constructors have been completed, but other member methods will need to be completed or added. Familiarize yourself with the constructors of this class, noting that the data member must be initialized.

```
public class GSmartSquare extends GSquare
{ // Data member
  protected GCanvas myCanvas;      // what window this object belongs to
```

```
  // Constructors
  public GSmartSquare(double x, double y, double size, Color color, GCanvas screen)...
  public GSmartSquare(double x, double y, double size, GCanvas screen)...
  public GSmartSquare(double size, GCanvas screen)...

  // The methods that make this object "smart":
  public boolean overflowsWindowHorizontally()
  { // Make sure that we have a canvas. If not there is nothing for us to overflow,
    // otherwise, check our left and right edges against the window.
    if (myCanvas == null)
      return false;
    return (getX() < 0.0) || (getX() + getWidth() > myCanvas.getWidth());
  }

  public boolean overflowsWindowVertically()
  { /**************************************************************************
     * replace code here
     *************************************************************************/
    return false;  // <==  only here to prevent red complaints from Netbeans
  }

  public boolean fitsInItsWindow()
  { /**************************************************************************
     * replace code here.   **  Make sure to utilize the above two methods. **
     *************************************************************************/
    return false;  // <==  only here to prevent red complaints from Netbeans
  }
}
```

7.3 `GMovingSquare` Class Outline

```
public class GMovingSquare extends GSmartSquare
{ // Data member
  private GPoint displacement;       // speed and direction of movement

  // Constructors
  public GMovingSquare(double x, double y, double size, Color color,
                               double deltax, double deltay, GCanvas screen)...
  public GMovingSquare(double size, Color color,
                               double deltax, double deltay, GCanvas screen)...
                                                  // default position: 0.0, 0.0
  // Inspector for displacement
  public GPoint getDisplacement()...

  // Mutators for displacement
  public void setDisplacement(GPoint displacement)...
  public void setDisplacement(double x, double y)...

  /* Moves the GMovingSquare according to the object's own displacement. */
  public void move()
```

```
  { /*************************************************************************
     * add code here to modify displacement when necessary to "bounce"
     *************************************************************************/
    // move as usual
    move(displacement.getX(), displacement.getY());
  }
}
```

7.4 Contents of `PlayPool.java`

Contents of PlayPool.java

```
1   /*
2    * Header comments go here
3    */
4
5   import acm.graphics.*;
6   import acm.program.*;
7   import acm.util.*;
8   import java.awt.*;
9   import myLibrary.graphics.*;
10
11  public class PlayPool extends GraphicsProgram
12  {
13
14    public void run()
15    {
16      // Add 3 circles
17      for (int i = 0; i < 3; i++)
18      {
19        // Choose the "corner" of the circle
20        double x = rgen.nextDouble(0, getWidth() - SIZE);
21        double y = rgen.nextDouble(0, getHeight() - SIZE);
22
23        // Initial direction of movement
24        double angle = rgen.nextDouble(0.0, 2 * Math.PI);
25
26        // Get the "speed" of movement
27        double speed = rgen.nextDouble(MINIMUM_SPEED, MAXIMUM_SPEED);
28
29        // Set up motion as determined by direction and speed
30        double deltax = speed * Math.cos(angle);
31        double deltay = speed * Math.sin(angle);
32
33        // Draw the new moving circle
34        add(new GMovingCircle(x, y, SIZE, rgen.nextColor(), deltax, deltay));
35      }
36
37      // Make MAX_MOVES number of moves
38      for (int i = 0; i < MAX_MOVES; i++)
39      {
40        // For every object in our window...
41        for (int j = 0; j < getElementCount(); j++)
42        {
43          // the jth object
44          GObject gObject = getElement(j);
45
```

```
46          // Make sure it is a moving circle
47          if (gObject instanceof GMovingCircle)
48          {
49            // Since we have a circle we cast it as such.
50            GMovingCircle ball = (GMovingCircle) gObject;
51
52            /*****************************************************************
53             * Add and modify code here to remove the ball from the window if
54             * it has been captured by a pocket.
55             *****************************************************************/
56            ball.move();
57          }
58        }
59        pause(SLEEPTIME);
60      }
61    }
62    // CONSTANT AND GLOBAL OBJECTS DECLARATION SECTION
63
64    // A delay time for animation
65    private static final double SLEEPTIME = 20;
66    // Number of moves to make
67    private static final int MAX_MOVES = 1000;
68    // attributes of a pool ball:
69    // size of a pool ball
70    private static final double SIZE = 35;
71    // Bounds on pool ball velocity
72    private static final double MINIMUM_SPEED = 5;
73    private static final double MAXIMUM_SPEED = 10;
74    // A random generator object
75    private RandomGenerator rgen = RandomGenerator.getInstance();
76  }
```

7.5 Contents of `Fireworks.java`

Contents of Fireworks.java

```
1   /*
2    *  Header comments go here
3    */
4   import myLibrary.graphics.*;
5   import acm.graphics.*;
6   import acm.program.*;
7   import acm.util.*;
8   import java.awt.*;
9
10  public class Fireworks extends DualSliderProgram
11  {
12      public void init()
13      {
14          setSize(1000, 700);
15          super.init();
16          setRangeA(0, 250);    // rate of spark creation per iteration
17          setRangeB(1, 5);      // upper range of spark lifetime * 10
18      }
19
20      public void run()
21      {
```

```
22          // allows window to be closed
23          if (hasRunBeenPressed())
24          {
25              // Add code here to make black background
26
27              // Produce fireworks until user tires of them
28              while (true)
29              {
30                  // Add code here such that in 5% of the of passes of this loop,
31                  // a GEmitter object is created and its pulse() method is invoked
32
33
34                  // Move all GSparks in the window
35                  // For every object in our window...
36                  for (int j = 0; j < getElementCount(); j++)
37                  {
38                      // Get the jth object
39                      GObject gObject = getElement(j);
40
41                      // Make sure it is a GSpark
42                      if (gObject instanceof GSpark)
43                      {
44                          // Since we have a GSpark, cast it as such and move it.
45                          ((GSpark) gObject).move();
46                      }
47                  }
48                  pause(SLEEPTIME);
49              }
50          }
51      } // end of run()
52
53      // CONSTANT AND GLOBAL OBJECTS DECLARATION SECTION
54      // pause time for animation
55      private static final int SLEEPTIME = 50;
56
57      // Random generator used to vary GEmitter and GSpark attributes
58      RandomGenerator rgen = RandomGenerator.getInstance();
59
60      public static void main(String args[])
61      {
62      (new Fireworks()).start();
63      }
64  } // end of Fireworks program class
```