yoavg / ioop2017

Assignment 2

Yoav Goldberg edited this page on 1 Apr 2017 · 6 revisions

Due Data: 13/4/2017, 23:59.

Objects, Geometry, Abstract Art and Bouncing Balls

In this assignment you will start working with **Objects**. You will define **Classes** and instantiate objects of those classes. The code in this assignment will be used in later assignments.

See here for information about working with multiple classes: MultFiles

Part 1: Geometry

The goal of this part is to develop a program to reason about points, lines, rectangles and their intersection.

You will define two classes: `Point` , `Line` .

We suggest that you work incrementally -- whenever you complete a method, see that everything compiles and that the method works as expected.

1.1 Point

A point has an `x` and a `y` value, and can measure the distance to other points, and if its is equal to another point.

```
public class Point {
   // constructor
   public Point(double x, double y) { }

   // distance -- return the distance of this point to the other point
   public double distance(Point other) { }

   // equals -- return true is the points are equal, false otherwise
   public boolean equals(Point other) { }

   // Return the x and y values of this point
   public double getX() { }
   public double getY() { }
}
```

Remember that the distance between two points `(x1,y1)` and `(x2,y2)` is the square root of: `((x1-x2)*(x1-x2))+((y1-y2)*(y1-y2))` . You can use the `Math.sqrt` method to get the square root of a number in Java:

```
double root_of_13 = Math.sqrt(13);
```

1.2 Line

A line (actually a line-segment) connects two points -- a start point and an end point. Lines have lengths, and may intersect with other lines. It can also tell if it is the same as another line segment.

```
public class Line {
   // constructors
   public Line(Point start, Point end) { }
   public Line(double x1, double y1, double x2, double y2) { }

   // Return the length of the line
   public double length() { }

   // Returns the middle point of the line
```

▸ Pages 21

- piazza
- Assignments

Clone this wiki locally

`https://github.com/yoavg/io`

Clone in Desktop

```
   public Point middle() { }

   // Returns the start point of the line
   public Point start() { }

   // Returns the end point of the line
   public Point end() { }

   // Returns true if the lines intersect, false otherwise
   public boolean isIntersecting(Line other) { }

   // Returns the intersection point if the lines intersect,
   // and null otherwise.
   public Point intersectionWith(Line other) { }

   // equals -- return true is the lines are equal, false otherwise
   public boolean equals(Line other) { }

}
```

The `intersectionWith(Line other)` method is a bit complicated. You can find some hints about how to calculate the intersection between two line segments here.

Testing your code

You can perform some sanity-checks for your code using our provided GeometryTester class.

What to submit?

For this part, you need to submit the code for the `Point` and `Line` classes (and any additional classes you may write). You do not need to specify a `run` target in the makefile for this task.

Part 2: GUI and Abstract Art

In this part, we provide you with some code (the `GUI` class) that handles drawing graphics to a window (The name GUI stands for Graphical User Interface). You do not need to understand how the GUI class works, but you do need to understand how to use it.

Here is a simple program that uses the GUI class:

```
import biuoop.GUI;
import biuoop.DrawSurface;

import java.util.Random;
import java.awt.Color;

public class SimpleGuiExample {

  public void drawRandomCircles() {
    Random rand = new Random(); // create a random-number generator
    // Create a window with the title "Random Circles Example"
    // which is 400 pixels wide and 300 pixels high.
    GUI gui = new GUI("Random Circles Example", 400, 300);
    DrawSurface d = gui.getDrawSurface();
    for (int i = 0; i < 10; ++i) {
       int x = rand.nextInt(400) + 1; // get integer in range 1-400
       int y = rand.nextInt(300) + 1; // get integer in range 1-300
       int r = 5*(rand.nextInt(4) + 1); // get integer in 5,10,15,20
       d.setColor(Color.RED);
       d.fillCircle(x,y,r);
    }
    gui.show(d);
  }

  public static void main(String[] args) {
     SimpleGuiExample example = new SimpleGuiExample();
     example.drawRandomCircles();
  }
}
```

This code is using packages. In particular, it is using the `biuoop` package which we provide, as well as classes from the `java.util` and `java.awt` packages which are part of the java runtime environment (and are always available).

The `GUI` , `DrawSurface` and `Sleeper` classes belong to the `biuoop` package, which are defined in the biuoop-1.4.jar file. In order to use this code, java needs to know where to look for it. After you download the `biuoop-1.4.jar` file and put it in the directory of the `SimpleGuiExample.java` file, Compile and run the code using:

```
> javac -cp biuoop-1.4.jar:. SimpleGuiExample.java
> java -cp biuoop-1.4.jar:. SimpleGuiExample
```

The `-cp` switch is the *class-path*, it tells java to look for classes in the `biuoop-1.4.jar` file and in the current directory (`.`), instead of the default behavior which is to look for classes only in the current directory.

> **Note**: In Windows, the separator in the `-cp` argument is `;` instead of `:` , so your lines should look like: `java -cp biuoop-1.4.jar;. SimpleGuiExample`

Make sure you were able to compile and run this code before you proceed.

Explaining the code example

The `java.awt.Color` class is a part of the Java JDK, and is used to represent colors. In the example, we used a predefined color (`RED`), which is defined in a static field of the `java.awt.Color` class. There are also constructors for creating colors based on proportions of red, green and blue components. See the `java.awt.Color` documentation for further details.

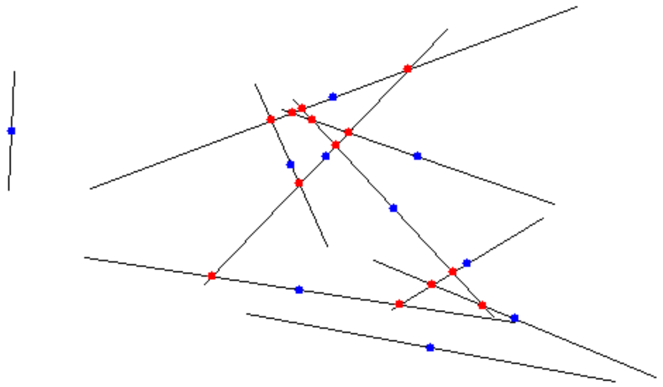
The `biuoop.GUI` and `biuoop.DrawSurface` classes are part of the `biuoop` package that we supply. Creating a new `GUI` object create a screen, with a title and dimensions. In order to draw on the screen, you ask the `GUI` object for a `DrawSurface` . The `DrawSurface` has several methods that you can use to draw lines, circles and so on. Once you done drawing, you call the `show(DrawingSurface d)` method of the `GUI` object with your `DrawingSurface` , and the drawing is displayed on the screen.

`Sleeper` is also a part of the `biuoop` package. This object allows us to halt the program execution for a specified number of milliseconds.

For further details, see the documentation for the biuoop package.

Your Task

Your goal is to use the GUI class to generate random pictures like the following:



In this image, we have 10 lines, drawn in black. The middle point in each line is indicated in blue, while the intersection points between the lines are indicated in red (the points are filled circles with a radius of 3).

Modify the example code above to generate pictures such as this one.

Hints:

- You can draw lines with the `drawSurface.drawLine(x1, y1, x2, y2)` method.
- Use the Point and Line classes from the previous part.
- You probably want to keep an array of the previously drawn lines.
- You may find it helpful to define and use helper methods such as `Line generateRandomLine()` and `void drawLine(Line l, DrawSurface d)`.

If your intersection points are not correct, you may want to debug the Line class some more!

What to submit?

For this task, you should include all the relevant classes. The `run2` target in the makefile should run the drawing code, which should reside in a class called `AbstractArtDrawing`.

(all tasks, including this one, will be submitted in the same zip file)

Part 3: Animation and Bouncing Balls

In this part, you will draw balls (which are circles) and move them across the screen.

What to submit?

For this part, you need to include all the classes defined below.

The target `make run3.2` will run the `BouncingBallAnimation` class, and `make run3.3` will run the `MultipleBouncingBallsAnimation` class.

Task 3.1: Static Balls which you can draw.

Our main object will be a `Ball` (actually, a circle). Balls have size (radius), color, and location (a `Point`). Balls also know how to draw themselves on a DrawSurface.

You need to create a `Ball` class, with at least the following methods:

```
public class Ball {
   // constructor
   public Ball(Point center, int r, java.awt.Color color);

   // accessors
   public int getX();
   public int getY();
   public int getSize();
   public java.awt.Color getColor();

   // draw the ball on the given DrawSurface
   public void drawOn(DrawSurface surface);
}
```

Test your code with the following program, which will create 3 balls and show them on the screen.

```
import biuoop.GUI;
import biuoop.DrawSurface;


public class BallsTest1 {
   public static void main(String[] args) {
      GUI gui = new GUI("Balls Test 1", 400, 400);
      DrawSurface d = gui.getDrawSurface();

      Ball b1 = new Ball(100,100,30,java.awt.Color.RED);
      Ball b2 = new Ball(100,150,10,java.awt.Color.BLUE);
      Ball b3 = new Ball(80,249,50,java.awt.Color.GREEN);

      b1.drawOn(d);
      b2.drawOn(d);
      b3.drawOn(d);

      gui.show(d);
   }
}
```

Task 3.2: A simple animation loop with one ball

Animation is achieved by drawing different pictures on the same area one after the other. Using our graphical package, we can do animation using a loop such as the following:

```
GUI gui = new GUI("title",200,200);
biuoop.Sleeper sleeper = new biuoop.Sleeper();
java.util.Random rand = new java.util.Random();
while (true) {
   DrawSurface d = gui.getDrawSurface();
   Ball ball = new Ball(rand.nextInt(200), rand.nextInt(200), 30, java.awt.Color.BLACK
   ball.drawOn(d);
   gui.show(d);
   sleeper.sleepFor(50);  // wait for 50 milliseconds.
}
```

Put this into the main method of a class, compile, and run it.

The loop creates an empty drawing surface, create a ball with a random location, draws it on the surface, shows the surface, waits for 50 milliseconds and so on. Assuming that creating the surface and displaying the ball take roughly zero time, sleeping for 50 milliseconds gets us about 20 frames per second.

The animation is not really good -- things seem to jump all over the place. The problem is that the different frames are not related to each other.

Lets fix that and create an animation with a moving ball. In order to do that, we need to give our ball some speed and direction. The first step would be to define a Velocity class:

```
// Velocity specifies the change in position on the `x` and the `y` axes.
public class Velocity {
   // constructor
   public Velocity(double dx, double dy);

   // Take a point with position (x,y) and return a new point
   // with position (x+dx, y+dy)
   public Point applyToPoint(Point p);
}
```

Next, we add `Velocity`, as well as the following methods, to the `Ball` class:

```
public void setVelocity(Velocity v);
public void setVelocity(double dx, double dy);
public Velocity getVelocity();

public void moveOneStep() {
    this.point = this.getVelocity().applyToPoint(this.point);
}
```

Now change the animation loop from above to:

```
GUI gui = new GUI("title",200,200);
Sleeper sleeper = new Sleeper();
Ball ball = new Ball(0, 0, 30, java.awt.Color.BLACK);
ball.setVelocity(2, 2);
while (true) {
   ball.moveOneStep();
   DrawSurface d = gui.getDrawSurface();
   ball.drawOn(d);
   gui.show(d);
   sleeper.sleepFor(50);  // wait for 50 milliseconds.
}
```

(question: what happens if we do not invoke setVelocity before the loop? make sure the program does not crash!)

This is much better! But the animation is over very quickly, because the ball goes outside of the screen. Change the `moveOneStep()` method so that the ball does not go outside of the screen -- when it hits the border to the left or to the right, it should change its horizontal direction, and when it hits the border on the top or the bottom, it should change its vertical direction. (Changing the horizontal direction can be achieved by setting the velocity's `dx` to `-dx`).

Create a class called `BouncingBallAnimation` with the main method that runs the bouncing ball animation.

Note It is convenient to specify the velocity in terms and `angle` and a `speed` , so instead of specifying dx=2, dy=0, you could specify `(90, 2)` meaning 2 units in the 90 degrees direction (assuming up is angle 0). This can be achieved by creating a new constructor to Velocity, taking an angle an a speed. The problem is that angle and speed are two doubles, and we already have a constructor taking two doubles (dx and dy). In order to solve this, we can create a static method (belonging to the Velocity class) that will create new instances for us, instead of the constructor:

```
public class Velocity {
   ...
   public static Velocity fromAngleAndSpeed(double angle, double speed) {
      double dx = ...;
      double dy = ...;
      return new Velocity(dx, dy);
   }
}
```

We can then use this method to create velocities:

```
Velocity v = Velocity.fromAngleAndSpeed(90, 2);
ball.setVelocity(v);
```

This method is not a suggestion. You need to implement it.

Task 3.3: Multiple Balls

One ball is nice, but a bit boring. We want to see many bouncing balls. In this task, you should create a program called `MultipleBouncingBallsAnimation` . This program is invoked from the commandline, and each argument is a size of a Ball:

```
> java -cp biuoop-1.4.jar:. MultipleBouncingBallsAnimation 12 2 3 4 2 9
```

This will create an animation with 6 balls, of sizes 12, 2, 3, 4, 2 and 9 respectively. Each ball will start in a random location on the screen. Each ball will start with a different speed -- we want larger balls to be slower (but balls above size 50 can all have the same slow speed). Each ball will change direction when hitting the window border.

Notice that the number of balls is not fixed -- use an array to store the balls.

Task 3.4: Multiple Frames

Now we want to create two frames -- one of them is a grey rectangle from (50,50) to (500,500), and the other is a yellow rectangle from (450,450) to (600,600). We want the first half of the balls to bounce inside the first frame, and the second half of the balls to bounce inside the second frame.

Your program should be called `MultipleFramesBouncingBallsAnimation` . As before, this program is invoked from the commandline, and each argument is a size of a Ball:

```
> java -cp biuoop-1.4.jar:. MultipleFramesBouncingBallsAnimation 12 2 3 4 2 9
```

(As before, the number of balls is not fixed, but you can assume an even number of balls. We note that it is a much better practice to somehow handle also the case of a non-even number of balls.)

A note on checkstyle

For this assignment only, because you do not know better yet, it is OK to have the checkstyle error:

> Definition of 'equals()' without corresponding definition of 'hashCode()'.

Summary

Your final submission should include (at least) the following classes, with at least the functionality and methods as described above.

- Point
- Line
- AbstractArtDrawing
- Ball
- Velocity
- BouncingBallAnimation
- MultipleBouncingBallsAnimation
- MultipleFramesBouncingBallsAnimation

Your makefile should include the targets `compile` to compile everything, as well as `run2`, `run3.2`, `run3.3`, `run3.4` as described above.

You do not need to include the file biuoop.jar, and can assume it is available in the same directory as the makefile.