6.092: Assignment 6: Graphics strikes back!

In the last assignment, you were provided with a BouncingBox class. This class contains logic for two things:

1. drawing boxes on the screen, and
2. moving the boxes around once drawn.

In this assignment, we will explore separating this functionality into two classes. This way, one class can handle moving any drawable object around the screen, and it can move any kind of object that we define in another class, which sill simply draw itself on the screen.

We provide you with Bouncer, a class that only has the code for moving objects by bouncing them off the edges of the window. Bouncer is initialized with an object that implements the Sprite interface to draw an object, and to compute the size of the object so it knows when to bounce. This allows the Bouncer to move any graphical objects, that could be written by multiple people.

Note: Sprite is a computer graphics term for a small animated object.

Setup

This assignment uses the same SimpleDraw class from the previous assignment. However, the other classes are different. You can reuse your previous project, or start a new one. However, you probably want to keep your previous DrawGraphics class, since you may want to borrow some code from it.

1. Create new classes, copying and pasting the source code from below for the following classes: SimpleDraw, DrawGraphics, Rectangle, Bouncer, StraightMover, Sprite. *Alternative*: If you are comfortable with managing files on your computer, you can download the initial project as a .zip file. Extract the files somewhere. Create a new project in Eclipse, selecting "from existing source," and point it at directory you extracted from the .zip.
2. Run this new project (you may need to open SimpleDraw, then choose Run). You should see a moving rectangle that bounces off the edges of the screen.

Implementing Interfaces

Since Bouncer supports drawing objects of different types, let's create one!

Requirement: Create a new class that implements the Sprite interface and draws some shape that is not a rectangle. Change DrawGraphics to contain an ArrayList of Bouncer objects. Add one rectangle and one of your own shapes to the ArrayList, and make them move. When you run your program, you should see one Rectangle and one of your own objects bouncing off the edges of the window.

Suggested Steps

1. Change DrawGraphics to contain an ArrayList of Bouncer objects instead of a single Sprite. This is just like what you did in the previous assignment.
2. Run your program. It should draw the same thing on the screen as before, except now it should be using the ArrayList.
3. Add a second Rectangle to the ArrayList and make it move. When you run your program, you should now have two Rectangles.
4. Create a new class, and give it a name that describes the shape you want to make (eg. Line, Oval, Star, Text, Arrow).
5. Make this class implement the Sprite interface. You will need to add implementations of all the methods in the interface. For now, make these methods do nothing (eg. empty methods, and return 0 if needed).
6. Replace one of the Rectangles in DrawGraphics with your own object. This means the ArrayList will contain a Rectangle and your own object. Run your program. You should now see only one Rectangle, since your object

doesn't have code to draw on the screen yet.
7. Add code to your object's draw() method to draw the shape you want. Run the program until it looks the way you want it to. *Hint:* You may want to use Rectangle as an example.
8. At this point, your shape may not bounce off the edges correctly. It might falls off the right and bottom edges of the window before changing directions. The reason is that you need to return a correct width and height from getWidth() and getHeight(), so the Bouncer knows when to make it change directions. Fix these two methods to return the appropriate values.
9. At the end, when you run your program you should have a rectangle and your own shape moving on the screen, and bouncing off the edges. Feel free to add extra shapes, if you like.

Multiple Movement Patterns

We have split the movement and drawing code into two separate classes. This means we can use different movement classes with both the Rectangle and the new shape you have created.

Requirement: Add an ArrayList of StraightMovers to DrawGraphics. Add a rectangle and one of your shapes to this ArrayList. When we run your program, we should see at least four shapes: two rectangles and two of your own shapes. Two of them should bounce off the edges of the window, and two of them should move off the edge of the screen.

Suggested Steps

1. Add an ArrayList of StraightMovers to DrawGraphics.
2. Call draw() on each StraightMover from the DrawGraphics.draw() method.
3. Put a new rectangle and a new instance of your shape in this ArrayList. *Hint:* You can copy the code for the ArrayList of Bouncers, but you'll have to rename things and give the instances new initial positions.
4. Run your program again. You should see four shapes that move: two that bounce and two that don't.

Creating Interfaces

You'll notice that in your DrawGraphics class, you have code that basically does the same thing with Bouncer and with StraightMover. This is a perfect opportunity to create a new interface to reduce duplication.

Requirement: Create a new interface called Mover. Make Bouncer and StraightMover class implement this new interface. Change DrawGraphics to contain a single ArrayList of Mover. This ArrayList should contain four objects: two Bouncers and two StraightMover. This should eliminate most of the duplication from your DrawGraphics class.

Suggested Steps

1. Create a new interface called Mover. Initially it will be empty.
2. Make both Bouncer and StraightMover implement the Mover interface.
3. Change your DrawGraphics class to contain a single ArrayList of Mover objects. At this point, DrawGraphics will not compile. The problem is that there will be methods that you are calling on Mover instances, but there are no methods in the Mover interface.
4. Add the method(s) you use in DrawGraphics to the Mover interface. *Hint:* This should be the method(s) that have the same *signature* in Bouncer and StraightMover.
5. At this point, your program should run and will look the same as before.

(Optional) Reducing Duplication

The Bouncer and StraightMover classes have a lot of duplicate code. Eliminate this duplication by putting all the common code into one class, and removing it from the Bouncer and StraightMover classes.

Submission Instructions