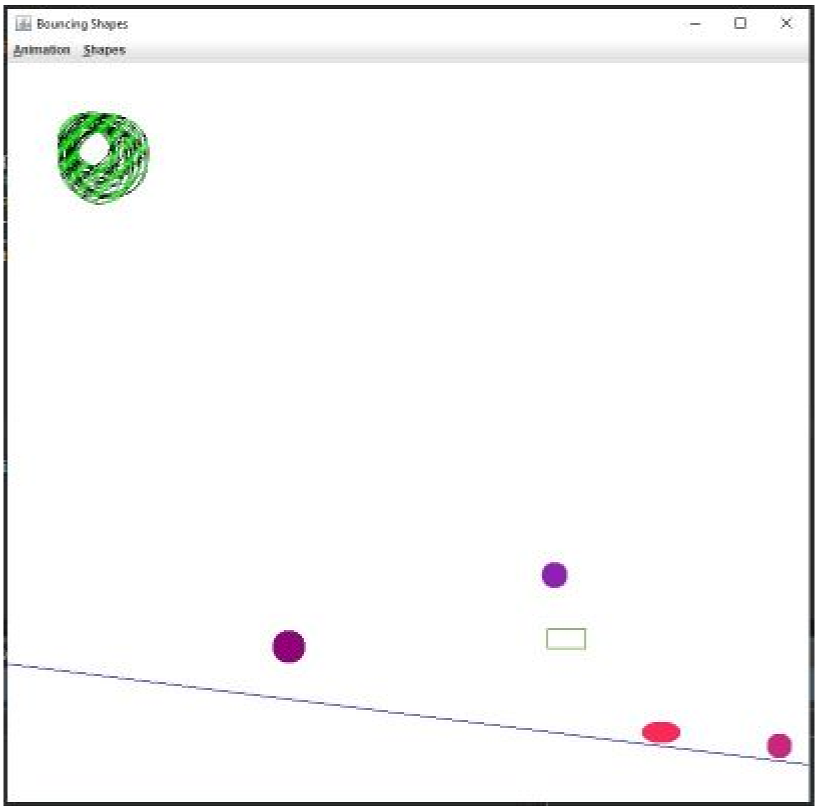
Shapes - Summary

In this assignment, you will create several classes that inherit from Shape. You will also create a MyPanel class that will help assemble and expose the shapes. You will be given code that will animate the shapes on the screen for you to see and play with.

You will be graded on proper object oriented programming which includes:

- extending the Shape class
- implementing or overriding methods
- maintaining proper encapsulation
- calling `super()` constructor as necessary

Here is what your screen might look like:



The shapes will bounce off the slanted floor and walls. Using the menu system, you will be able to add and remove shapes, change the speed of animation, and alter gravity.

Learning Objectives

It is important that you not lose sight of the learning objectives. Students can sometimes get lost and resort to "copying" code that comes from some helpful "resource" and fail to learn the important things. You will be required to write code like this on your exams. If you don't understand something, please do some research and/or ask about it.

> Note: If you look at some of the provided code, there is quite a lot going on there. You do NOT need to understand most of it.

You must know the following:

- `abstract` keyword
- How to call the `super()` constructor, when and why
- How to create instance fields and their getter/setter methods
- Working with object instances, `static` methods, and `static` constants

Shapes

You will need to create **FOUR** or more different shapes that all inherit from `Shape`. Each implementation must implement several methods as shown in the sample code below. Note that the getters/setters are not shown, however, and you need to implement those.

```java
import java.awt.Graphics;
import java.awt.Rectangle;

// Don't name the class just Rectangle, because that will conflict with
java.awt.Rectangle.
// You may have a shape that is simply Circle.
public class RectangleShape extends Shape {

    // TODO: Add only necessary instance fields
    // You need to have getters/setters for each field

    /**
     *  constructor that creates a RectangleShape
     */
    public RectangleShape(int x, int y, int width, int height) {
        // to be implemented
    }

    /**
     * This will create an instance of a RectangleShape object that is placed
     * randomly on the screen in a fully visible location. The width and height
     * will be randomly decided with reasonable values.
     *
     * Notice that this is the only static method in this class.
     * The name of the method MUST have the pattern: "getRandom" + className
     *
     * @return A RectangleShape object.
     */
    public static RectangleShape getRandomRectangleShape() {
        // to be implemented
        return new RectangleShape( /* arguments are necessary here */);
    }

    /**
     * Calculate and return the area of the rectangle
     */
    public double getArea() {
        // to be implemented
    }

    /**
```

```java
     * Draws the shape at its current location and size.
     * @param g The Graphics object to draw on
     */
    public void draw(Graphics g) {
        g.setColor(getColor());
        g.drawRect(/* arguments are necessary here */);
    }

    /**
     * Calculates the bounding rectangle of this shape in its
     * currently location.
     * @return The Rectangle object that tightly bounds the shape.
     */
    public Rectangle getBoundingRect() {
        return new Rectangle(/* arguments are necessary here */);
    }
}
```

You are required to implement at least one shape that inherits from a child of `Shape`. For example, if you implement the `RectangleShape` as shown above, it is very, very easy to implement a `Square` shape by inheriting from `RectangleShape`. The `Square` shape would not have to implement much at all. In fact, it is a requirement to implement as little as possible.

MyPanel

You need to create a class with the name `MyPanel` which must inherit from `AnimatedPanel`. Your class must override two methods:

```java
/**
 * Randomly create one of the Shape objects you've implemented.
 * @return The object that you created.
 */
public Shape getRandomShape() {
    // The student should make use of the required static method
    // that each Shape object must implement (e.g. getRandomSquare)
}

/**
 * Returns a String array that is filled with all the class names of the Shapes
that
 * your program can create.
 * @return String array of all class names
 */
public String[] getShapeClassNames() {
    return new String[] { "Square", "RectangleShape" /* add ALL your shape class
names here */ };
}
```

Super Constructor

Will absolutely, 100% will need to use the `super()` constructor because shape depends on instance fields owned by the parent class, `Shape`. These instance fields (`x` and `y`) are `private`.

> IMPORTANT: You are NOT ALLOWED to modify the `Shape` class in any way! For example, do not change `x` to `public`.

Drawing

There are only a few methods you need to know when it comes to drawing. More importantly, you can use the internet to identify more drawing methods available on the `Graphics` and `Graphics2D` objects. Here are the basic methods to use:

```
// Sample code. You need to provide the correct argument values.
public void draw(Graphics g) {
    g.setColor(Color.BLUE);
    g.drawLine(x0, y0, x1, y1);
    g.fillOval(x, y, width, height);
    g.drawRect(x, y, width, height);
}
```

Code Submission

You will submit:

- `MyPanel.java`
- every <*shape*>.java file

You will NOT submit:

- `Main.java`
- `Shape.java`
- `AnimatedPanel.java`

You must submit your work to https://css.mrstride.com.

Rubric

> IMPORTANT: If you code does not compile, you get 0 points.

```
Points     Description
------     ----------------------------
    10     Pass functional requirements
           - methods correctly named and implemented
           - At least 4 shapes implemented
           - MyPanel methods correctly named and implemented
     5     Conventions, style, and commenting
    15     OOP requirements
```

```
            - inheritance, encapsulation, constructors, instance fields
            - static, private/public, getters/setters
------
    30      TOTAL
```

See UW Quality 142 for Java Coding Conventions.
https://courses.cs.washington.edu/courses/cse142/21au/quality-142.html

About This Document

Original assignment by Rob Nash, Autumn 2014. Major edits and additions by Jeff Stride, September 2024.