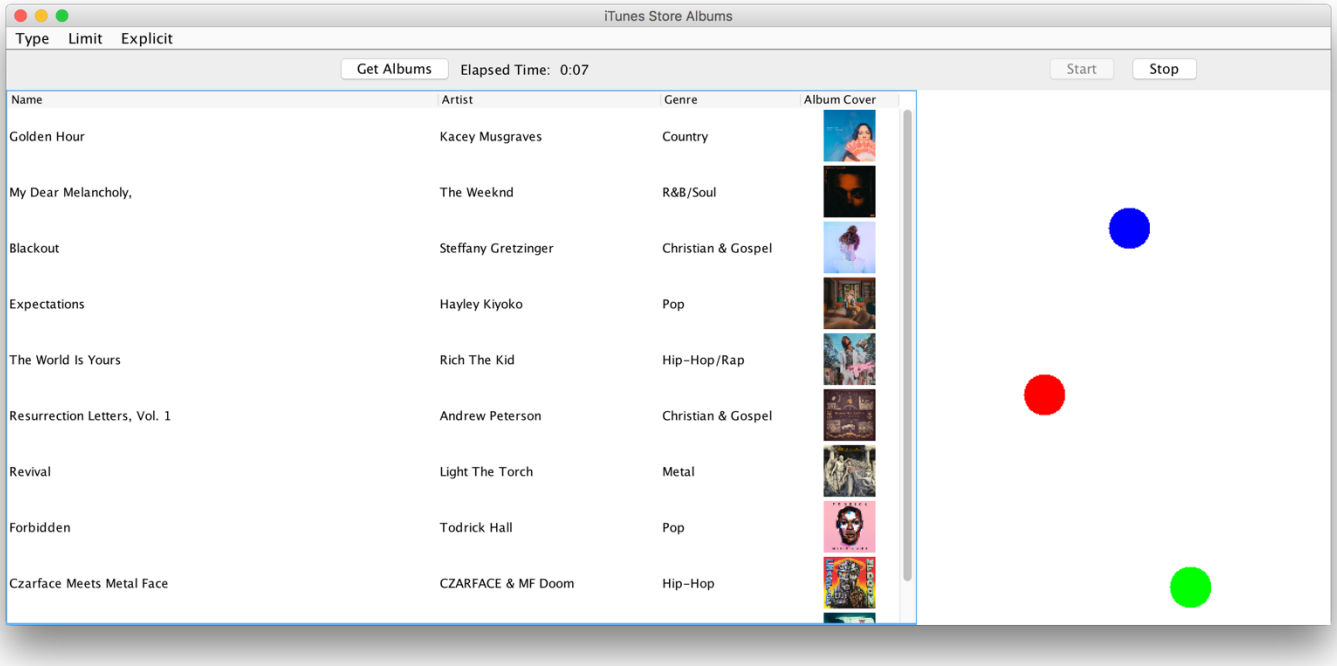
CSCI 470/680E Assignment 4

In this assignment, you will add an animation that runs in a background thread to the GUI application that you wrote for Assignment 3.



Changes to the Assignment 3 Classes

Make the following changes to the classes you wrote for Assignment 3:

- Add a `BouncingBallPanel` object in the `LINE_END` region of the application.

You will probably also need to adjust the overall size of the window to accommodate this new panel.

The BouncingBallPanel Class

This subclass of `JPanel` encapsulates the ball animation and will handle the button events.

Data Members:

- A pair of buttons to start and stop the animation.
- A `BallAnimationPanel` to display the animation.

Methods:

Constructor – Sets up the layout for the panel and adds listeners for the buttons.

`actionPerformed()` – If the source of the `ActionEvent` is the “Start” button”, you should enable the “Stop” button, disable the “Start” button, and call the `start()` method of the `BallAnimationPanel` to start the animation. If the source of the `ActionEvent` is the “Stop”

button, you should enable the “Start” button, disable the “Stop” button, and call the `stop()` method of the `BallAnimationPanel` to stop the animation.

The AnimationPanel Class

This subclass of `JPanel` will be used to display the animation in a separate background thread. Therefore, it should implement the `Runnable` interface.

Data Members:

- An `ArrayList` of `Ball` objects.
- A reference to a `Dimension` object that is initially set to `null`.
- A reference to a `Thread` object that is initially set to `null`.

Methods:

`public void start()` – If the `Thread` reference is `null`, create a new `Thread` object from this class and call the `Thread`’s `start()` method to make it runnable by the thread scheduler.

`public void stop()` – Call `interrupt()` for the `Thread` and then set its reference to `null`. This will cause the loop in the `run()` method to exit. Once the loop is finished, the method is finished as well, so the thread will terminate.

`protected void paintComponent(Graphics g)` – This method should be overridden to do the do the following:

- Call the superclass version of the method.
- If the `Dimension` object reference is `null`, create a set of `Ball` objects and add them to the `ArrayList`, then get the dimensions of the panel by calling `getSize()`.
- Draw the white background. One easy way to do that is to just a white rectangle the same size as the panel.
- Call the `move()` and `draw()` methods for each `Ball` object in the `ArrayList`. The `Dimension` object needs to be passed to `move()`, while the `Graphics` object needs to be passed to `draw()`.

`public void run()` – This is effectively the “main” method that will run in the separate background thread. It will have a loop that continues while the current thread is equal to the `Thread` reference data member, e.g.:

```
while (Thread.currentThread == animationThread)
```

The loop body should put the thread to sleep for a short amount of time (25 milliseconds is about right), and then call `repaint()` (which will eventually result in `paintComponent()` being executed). If the thread is interrupted, just `return` from the method.

The Ball Class

This class represents a single bouncing ball in the animation.

Data Members:

- The `Color` of the ball.
- The `radius` of the ball (integer).
- The `x` and `y` coordinates of the ball's center point (integers).
- `int dx` – the amount of change in the ball's x coordinate each time the ball moves. A negative value means the ball is currently moving to the left; a positive value means the ball is currently moving to the right.
- `int dy` – the amount of change in the ball's y coordinate each time the ball moves. A negative value means the ball is currently moving up; a positive value means the ball is currently moving down.

Methods:

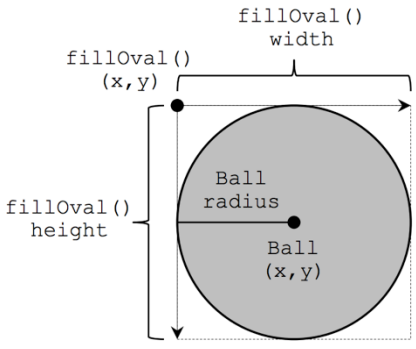
Constructor – Has six parameters corresponding to the data members described above, allowing you to initialize the ball.

`public void move(Dimension d)` – Has one parameter, a `Dimension` object that contains the width and height of the panel in which the ball is moving. Does the following:

- If the x coordinate of the ball is less than or equal its radius OR greater than or equal the width of the panel minus the radius of the ball, reverse the horizontal direction of the ball (i.e., change the sign for `dx`).
- If the y coordinate of the ball is less than or equal its radius OR greater than or equal the height of the panel minus the radius of the ball, reverse the vertical direction of the ball (i.e., change the sign for `dy`).
- Add `dx` to x.
- Add `dy` to y.

`public void draw(Graphics g)` – Has one parameter, the `Graphics` object that will be used to draw the ball. Use the `Graphics` object to set the drawing color to the `Color` of the ball, and then use it to call `fillOval()` to draw the ball.

The `fillOval()` method takes four arguments: the `x` coordinate of the upper left corner of the bounding rectangle of the oval to be filled, the `y` coordinate of the upper left corner of the bounding rectangle of the oval to be filled, the `width` of the oval to be filled, and the `height` of the oval to be filled. The diagram below shows the relationship of these parameters to the `x` and `y` coordinates of the ball's center point and its `radius`.



Programming Notes

- Your `BallAnimationPanel` should create several `Ball` objects with different colors, starting x and y coordinates, and `dx` / `dy` values. The radius for each ball may be the same, or they may be different. For example:

```
ballList.add(new Ball(Color.GREEN, 20, (d.width * 2 / 3),
        (d.height - 28), -2, -4));
```

- Due to the way the collision detection code works, starting a ball too close to the edge of the panel may cause it to "thrash", "dribbling" along the edge of the panel.
- The collision detection algorithm given in the assignment is fairly primitive. You are welcome to experiment and come up with an alternative approach.
- You may want to override the `getPreferredSize()` method for the `BallAnimationPanel`. That will allow you to specify the exact width you want for the panel.

Extra Credit

- Add a "ball bounce" sound effect that plays whenever a ball hits the edge of the panel. (+5 points)
- Add a music clip that plays in a loop while the animation is running. (+10 points)
- Add a control that lets the user set the speed of the animation (changing the sleep time will change the animation speed). (+5 points)
- The collision detection algorithm does not check for collisions between balls, so they will simply "overlap" in space rather than rebounding off each other. You could change that. There will be a small enough number of balls on screen that a brute force approach to detecting ball collisions is likely feasible. Or if you're feeling really ambitious and want to delve into the intricacies of axis-aligned bounding boxes or pairwise pruning, the Wikipedia page on collision detection is a decent starting point. (+10 points)