School of Computing and Mathematics
CSC-20037 Programming II

Practical session 1, Week 7:

Simple Animation

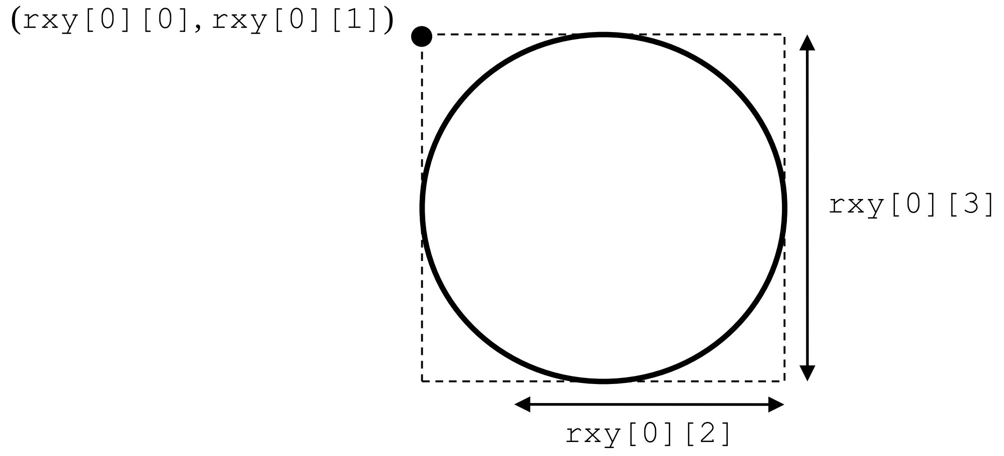
In this practical will complete the implementation of the Animate button (`animateButton` object) in your drawing application.

Task 1

Simply put, animation is achieved by redrawing graphic objects at different positions in succession. If done fast enough this gives the impression that graphic objects move.

You already know that every time you call the `paintComponent` method of the `canvas` object the canvas is **cleared and redrawn** in accordance with the code in the `paintComponent` method. In the case of your drawing application `paintComponent` actually calls our `draw` helper-method that does the drawing on the `Graphics` context.

Let's take for example the **first** oval object that the user of your application draws. This graphic object will have its coordinates stored in the two dimensional array called `rxy` (your choice of name for this array may be different). The x and y coordinates top/left corner of the oval are stored in `rxy[0][0]`, `rxy[0][1]` respectively, the width is stored in `rxy[0][2]` and the height in `rxy[0][3]`:



There are several things we can do to this oval in order to animate it. For example, if we successively add a (positive) value to, say, its x-coordinate (`rxy[0][0]`) while redrawing the oval after each increment we will observe the oval moving to the right on the canvas. Likewise, we can keep incrementing the oval's y-coordinate (`rxy[0][1]`) in which case the oval will appear to be moving downwards on the canvas. By adding or subtracting to the coordinates of the oval's position we can therefore cause the oval to move in any direction we want on the canvas. We can also effect animation by changing the width and/or height of the oval incrementally or even do all of the above at the same time!

But, who or what will be incrementally changing the oval's properties in order to achieve animation?

For this task, you can (laboriously) use the event issued every time you press the "Animate" button to **incrementally** change one (or more) of the **first** oval's properties with every click of the button. (By now you should have a clear idea of how to construct a suitable framework of Java code that can: process an event; that will in turn cause some code to be executed; in order to change the oval's properties; and then render the updated oval on the canvas.)

Remember to check whether a first oval is drawn before you attempt to make changes to its properties.

You will be successful in this task if every time you press the "Animate" button you see a change happen to the oval in its position and/or shape.

Task 2

For animation you really need the changes to the animated objects to happen at regular intervals and automatically without the need to press a button every time. An instance of the `Timer` class (the one defined in the `javax.swing` package and not the one in the `java.util` package) facilitates this.

A `Timer` object is an automatic way to issue an `ActionEvent` at a regular interval (specified in milliseconds – thousands of a second). Like all event causing mechanisms the `Timer` object needs to be associated with an instance of a class that implements the `ActionListener` interface. For example to create a `Timer` object that causes an `ActionEvent` with a frequency of 5 times a second (i.e. every 200 milliseconds):

```
...
Timer animationTimer = new Timer(200, animator);
...
```

where `animator` is an instance of a class that implements the `ActionListener` interface:

```
...
MyAnimatorClass animator = new MyAnimatorClass();
...
class MyAnimatorClass implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        ...  // Changes to cause animation go here.
    }
}
...
```

But a `Timer` object will not start issuing `ActionEvents` right away i.e. from the moment it is created. You can start a timer using its `start` method:

```
...
animationTimer.start();
...
```

Now spend a bit of time thinking where you should be placing the code examples above in order to animate your oval in the previous task using a timer. Note that you will need to move the code that you have written in task 1 elsewhere and write something different in its place.

Task 3

A very important (and fatal to your program) consequence of the code that you have written so far is that two segments of code could be attempting to **modify** the same data at the same time.

It could happen for example that as you are drawing ovals on the canvas, thus modifying the `rxy` array, your animation code attempts to modify the same array. Or you may press the "Clear Canvas" button just as the animation code changes the coordinates of an object on canvas. This will give rise to an exception and it is one of the most difficult types of programming errors to detect and fix.

In order to prevent such occurrence you need to make sure that only one segment of code can access the same data at any given time. There are many ways that can safeguard against this but discussing them here would take us beyond the scope of this practical. Instead, you can make sure that whenever you start drawing on the canvas, any animation is halted by stopping the timer from issuing `ActionEvents`:

```
...
animationTimer.stop();
...
```

Where (and it will be in more than one place) should you put this command?

Task 4

Experiment with animating different objects in different ways.

For example, can you make a ball (an oval) bounce around the canvas but not escaping off the canvas' edges?

With this practical you will complete the drawing application in the way it was intended for this module. However, nothing stops you from extending the application further in your own time. At this point you should be able to implement almost anything you set your mind to with a little investigation and effort.

One question should remain in your mind about this application: what if you want to be able to draw unlimited lines, rectangles, ovals and freehand pixels on your canvas instead of being limited to a fixed number?

A big part of CSC-20037 is concerned with the answer to the above question. By the end of the semester you should be able to change your program (if you want to) in order to overcome the above limitation.

Please note: This is not the last practical of CSC-20037. Practical sessions will continue as normal until the end of the semester.