Some Simple Java Programming pt. 2

Contents

1 Skills/Concepts

This module is a continuation of the Simple Java Programming module.

This module will continue to build on the previous concepts of:

- Packages
- Classes
- Java Drawing APIs

In addition, this module will discuss public/private scope and revisit inheritance.

2 Preparation

You should be fine booting up any Linux distribution as long as it has a Java SDK.

We will be using an existing git repository that contains a fair amount of code instead of producing a new program from scratch. This is to avoid copy and paste errors and allow us to cover material faster.

To get the code, execute the following commands:

```
git clone https://www.github.com/cyberpost500/java-project/
```

The code will be in the `java-project` directory.

3 Finally, some graphics…

3.1 Background

Creating graphics in Java is quite similar to Python, except that Java has built-in methods for all of the work, whereas in Python an external library does the heavy lifting.

3.2 Review/Run/Modify the Code

0. Check out the appropriate branch

```
git checkout 05_circles
```

1. Compile the code

```
ant build
```

2. Execute the class

```
ant run
```

3. Move the position where the circle is drawn:

```
                //drawArc(x, y, width, height, startAngle, arcAngle◀)
                g.drawArc(100, 100, 50, 50, 0, 360);◀
```

4. Add another shape. See the javadocs for `java.awt.Graphics`.

The javadocs for the default JDK should be installed to `/usr/share/doc/default-jdk-doc/api/`

(This is from the `default-jdk-doc` package, you may need to `sudo apt-get install default-jdk-doc`).

Open the javadocs and navigate to the `java.awt.Graphics` package.

```
x-www-browser /usr/share/doc/default-jdk-doc/api/
```

5. Save, build, and run

You don't need to do it in separate steps like we've been doing, you can let the dependencies for the run target take care of it for you.

```
ant run
```

6. Save your changes to a new branch (so we can re-use your changes later)

```
git checkout -b mine
git commit -a
```

Be sure to save your commit message, otherwise it will not commit the changes.

7. Switch back to master before continuing

```
git checkout master
```

8. Confirm your changes are still there

```
git diff master..mine
```

9. View the changes via a difftool

```
git difftool master..mine
```

3.3 Conclusion

Now the program displays a circle and optionally any other shapes you may have put into it. You may notice the method and argument names are a little different than they were in Python. If the documentation isn't clear enough or is too confusing, you can always google for a stack overflow or stack exchange article with an example of the API usage.

4 Shape Classes

4.1 Background

We'll be using a common base class for our shapes again. This will provide a convenient way to re-use code we've already written for accomplishing very similar tasks. But one new facet we'll be exploring is something called an interface. An interface is a sort of contractual obligation that any class "implementing" that interface must adhere. Basically, every method called out in an interface has to be written into any class that implements that interface.

We'll be using an interface as the most abstract representation of our shape, and then we'll create classes that implement this interface. Later on we'll extend those child classes with more children.

4.2 Review/Run/Modify the Code

0. Check out the appropriate branch

```
git checkout 06_shapes
```

1. Review the code

Spend a few minutes reviewing each of the new files and then view the updated version of the `MainWindow` class.

- `./src/io/github/cyberpost500/hungryshapes/shapes/Shape.java`
- `./src/io/github/cyberpost500/hungryshapes/shapes/Circle.java`
- `./src/io/github/cyberpost500/hungryshapes/MainWindow.java`

2. Add another circle to the window
3. Save, build, and run

```
ant run
```

4. Revert the changes back

4.3 Conclusion

You should have a passing familiarity with the object hierarchy. We'll spend a bit of time discussing this during the meeting and covering the relationship between the interface, the base class, and the main window.

5 Animated Shapes

5.1 Background

The animated shapes will still use graphical primitives, but now we'll need some additional helping classes to provide asynchronous updates to the user interface. We'll use a custom Runnable class that will be used to create threads that perform the task of updating the user interface and updating any animated shape positions.

5.2 Review/Run/Modify the Code

0. Switch to the appropriate branch

```
git checkout 07_animated_circles
```

1. Review the code

Spend a few minutes reviewing each of the new files and then view the updated version of the `MainWindow` class.

- `src/io/github/cyberpost500/hungryshapes/shapes/MovingShape.java`
- `src/io/github/cyberpost500/hungryshapes/MainWindow.java`

2. Switch to the `08_bouncing_shape` branch
3. Create a new shape using the custom code you created earlier as the `draw` method in a new shape class that extends the `BouncingShape`

- create a new file for the shape
- create a `public class` that `extends BouncingShape`
- define a constructor
- define a move method
- add the new shape to the `MainWindow` (both in the updater thread and the painting thread)

4. Discussion to follow.

5.3 Conclusion

These modifications may seem daunting, but they'll be easier in the future once you become more familiar with Java. Also, you should begin to appreciate the reusability of classes and the utility of extending classes to accomplish new behavior with similar objects.