Tutorial 2

At the end of this tutorial you should be comfortable with the following:-

- Building a drawing app with Gradle and IntelliJ
- Refactoring strategies for more expressive code.
- More advanced Git - resolving Git source conflicts
- More advanced Git – creating a new branch

Ex1 Creating a Circle Drawing project with IntelliJ using the Gradle Build Engine

Now we can start to build the drawing program.

1. Create a new Gradle Java project called CircleDraw. Right-click on 'src->main->java' in the project tree on the LHS and click New->Java Class. Create a Main class with a main function.
2. Create another with the name Circle when prompted. Add code so that the class looks like the following:-

```
import java.awt.*;
/**
 *  The Circle class encapsulates the information describing a circle and can draw it in
 * an AWT Graphics object using AWT library methods
 * */
public class Circle {
    private int rad;       // Fields
    private Point pos;
    private Color col;

    public Circle(Point initPos, Color col, int radius){ // The constructor
        rad=radius;        // Initialize fields
        pos=initPos;
        this.col=col;
    }

    public void draw(Graphics g) { // A method that draws the object in g
        g.setColor(col);
        g.fillOval(pos.x,pos.y,rad, rad);
    }
}
```

3. Add another class called Drawing, indicating that it inherits from the AWT class Canvas as follows:-

```
public class Drawing extends Canvas {
// A private field called f of class Frame. This is also in the AWT.
  private Frame f;

// The constructor
  public Drawing(){
  f = new Frame("My window");       // Instantiates the Frame
  f.add(this);                      // Adds the Canvas to the Frame
  f.setLayout(null);                // Stops the frame from trying to layout contents
  f.setSize(400, 400);              // Sets the dimensions of the frame
  f.setVisible(true);
  f.addWindowListener(new WindowAdapter() {// Closes the program if close window clicked
      public void windowClosing(WindowEvent e) {
          f.dispose();
      }
   });
  setBackground(Color.WHITE);       // Sets the Canvas background
  setSize(400, 400);                // Sets the Canvas size to be the same as the frame
}
```

4. Add a public method to the class called `paint` that has a `void` return and takes one parameter Graphics `g` The method should otherwise be empty for now.

```
  public void paint(Graphics g){}
```

5. In main, create a new instance of class Drawing

```
Drawing d=new Drawing();
```

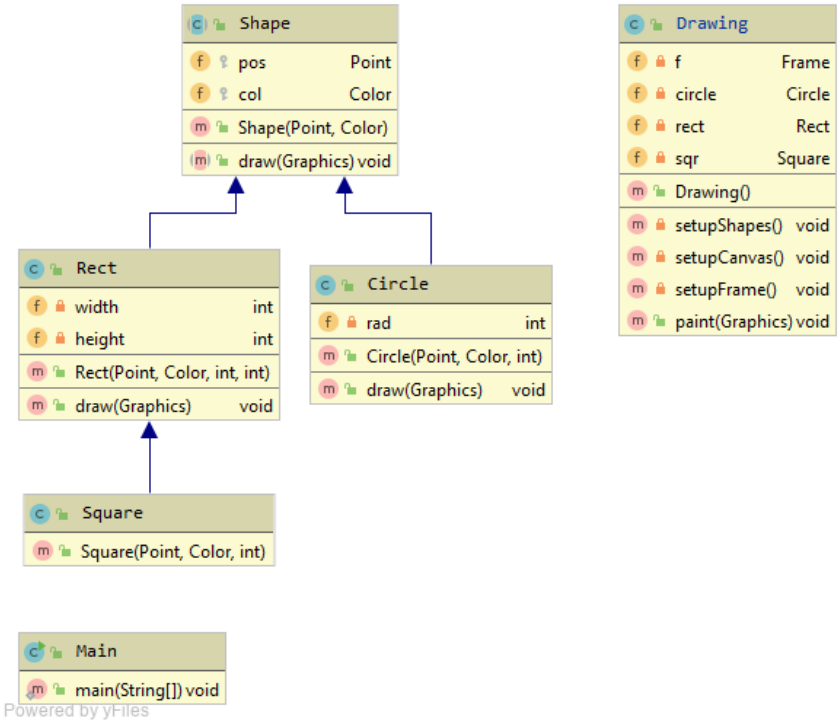
6. Build and run the program – you should see the Frame appear, and it should close when you click the Window's close button.

Now we can add some circles

7. Add a private field of class Circle and instantiate it in the constructor before you create the Frame. To do so you will need to instantiate an object of class Point and one of class Color and pass them to the Circle constructor, along with the radius eg:-
```
Point p=new Point(200,200);
Color c=new Color(0x992266);// The RGB number comprises three bytes: red, green and blue
```
8. In Drawing's paint method, call the draw method of the field that is a Circle, passing in the Graphics object that is passed into paint.
9. Build and run using the Gradle tasks – you should see a circle appear in the window!
10. Commit, tag it as v1.0 (VCS->Git->Tag) and push the current changes to Github (VCS->Git->Push (make sure 'Push tags' is checked). You should see on Github that there is now a 'release' of the project available.

Ex3 Drawing Rectangles and Refactoring to get a Shape Superclass

1. Write a class called Rect that does the same as circle, but draws a rectangle of width w and height h. (note the Graphics method to fill a rectangle is `fillRect(x, y,width,height);`)
2. Add a Rect to Drawing so that both the Circle and Rect are drawn. Create a new tag, and commit.
3. There is common code to Circle and Rect that could go in a more abstract superclass called Shape. We could extract this superclass by hand, but IntelliJ has a refactor command to help. Click Refactor->Extract->Superclass, select the fields and methods common to Circle and Rect (ie draw (check the abstract box), col and pos) and call the superclass Shape
4. Change Rect so that it also inherits from Shape, and delete the fields that have now moved to Shape. You will need to add `super(initPos, col);` to the Rect constructor so that the first thing it does is call the Shape constructor.
5. Build and run to make sure that it work correctly.
6. Create a Square class that extends Rect. (hint: it should comprise a constructor containing one line and nothing else!)
7. Now we can refactor to make the constructor of Drawing easier to read. With the mouse, select the following lines :-
```
f = new Frame("My window");       // Instantiates the Frame
f.add(this);                      // Adds the Canvas to the Frame
f.setLayout(null);                // Stops the frame from trying to layout contents
f.setSize(400, 400);              // Sets the dimensions of the frame
f.setVisible(true);
f.addWindowListener(new WindowAdapter() {// Closes the program if close window clicked
public void windowClosing(WindowEvent e) {
        f.dispose();
}
});
```
then click Refactor->Extract->Method and label the new method setupFrame when prompted. Do the same with the two lines that setup the Canvas:-
```
setBackground(Color.WHITE);    // Sets the Canvas background
setSize(400, 400);
```
8. Right click on the src folder in the IntellJ Project view on the LHS and click Diagrams->Show Diagram->Java Class Diagrams. You should get something like this (you might have to click m and f to show fields and methods):-



9. Git Add and Commit your project – tag it v1.1 and push to the remote repo.

Ex4 Create a Package and Re-organize Code

1. Right-click on the src folder in the project view and select New->Package. Call it shapes.
2. Select Shape, Circle, Rect and Square and drag them onto the new package. Agree to the Refactor and click "Do refactor" at the bottom of the screen when prompted.
3. The shapes should all have `package shapes;` at the top, and Drawing should import them.

Ex 5 Resolving Git Conflicts

1. Make a change to one of your .java file in the remote repo, and also change the same .java file (but in a different way!) in your local working folder. There are now two versions of the file in existence, your changed version and a version that someone else – perhaps a collaborator – has changed!
2. Add and commit your local changes to your local repo
3. Do a 'git pull'. It should warn of a conflict, and not perform a merge of local and remote
4. Check the .java file in IntelliJ – it should show both changes with some text to show which change came from where. Change the code to resolve the conflict as you want (you can include one or the other change, or both, or neither or put in a completely different change), making sure that you remove the >>>>>>>, ==========, <<<<<<<<< indicators and any other text added in the comparison.
5. Add and commit your resolved file.
6. Push back to the remote repo. Check that the remote version is the resolved version.

Ex 6 Making a New Git Branch

7. Select the project in the LHS project window. Click VCS->Git->Branches->New Branch
8. Give it a name (without spaces) – eg DevelopmentBranch.
9. You will see the name of the branch in the bottom right hand corner of IntelliJ. You are now working on a new branch
10. Make some changes to your .java source files, add, commit and push – you should now see the new branch on the remote repo
11. Now to switch back to the master branch, click VCS->Git->Branches->master->checkout – this will restore your project to the master branch. You can switch between branches this way.