# **Environment Description**

Operating System: Microsoft Windows 10

Integrated Development Environment: NetBeans IDE version 8.2

Application Server: None

Database server: None

Web Browser: None

# **Instructions how to set up and run your project.**

1. Obtain NetBeans and configure it

* Go to <https://netbeans.org/downloads/> and download the Java EE version of NetBeans
* The downloaded file is called: netbeans-8.2-javaee-windows.exe. Run this file.

1. Load the HW2 project code into NetBeans

* Start-up your NetBeans
* Go to File->Open project->Find the HW2 project on the machine->Open Project
* In the list of projects in NetBeans now we have a HW2 project. The code is in the Source Packages->Default package:

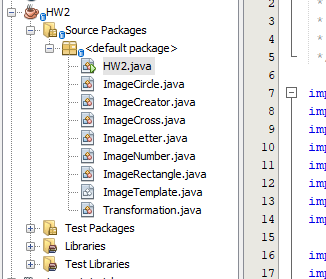


Figure 1 HW2 project

* We are going to use this project to demonstrate 2d graphics in java.

1. Run the application

* Right-click the HW2.java and click Run File

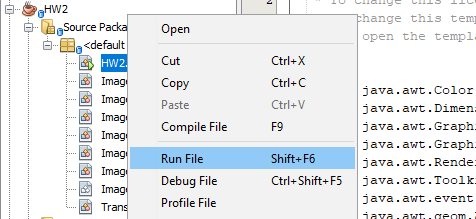


Figure 2 Running the application

# **Description of all the code in the project**

1. The **Transformation** class contains all possible transformations for an image: the translation, the scaling and the rotation.

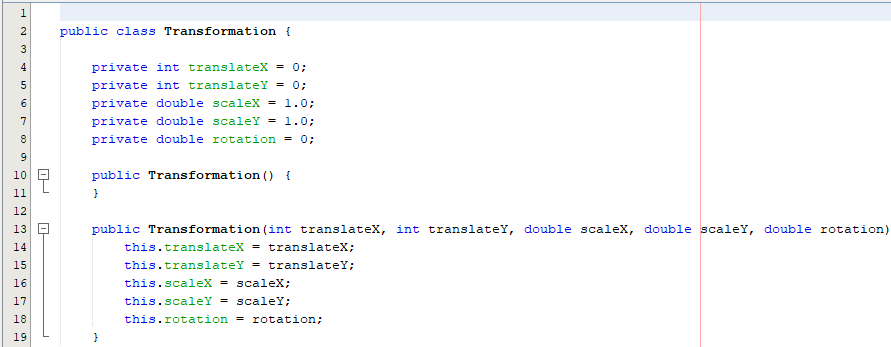


Figure 3 The Transformation class

1. The **ImageTemplate** abstract class contains the base information for an image class in the project. It is extended by 5 concrete classes which extend its functionality: ImageCircle, ImageCross, ImageLetter, ImageNumber, ImageRectangle. The template contains the current image transform, the two colors which are specific for this image and the original offset to the other images. This offset will be kept for the duration of the animation (run of the program).

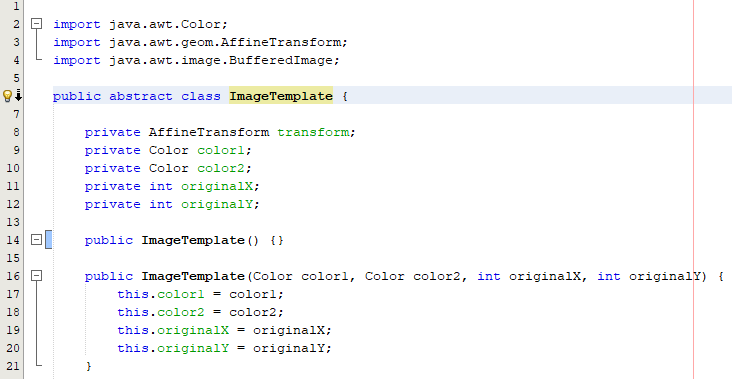


Figure 4 The ImageTemplate abstract class

The ImageTemplate class defines an abstract method getData() which is actually implemented in all the extending classes. This method returns the array of pixels of each image, therefore it is defined in each extending class.

The ImageTemplate class also defines the getImage() method which constructs the BufferedImage based on the array of pixels provided by the getData() method.

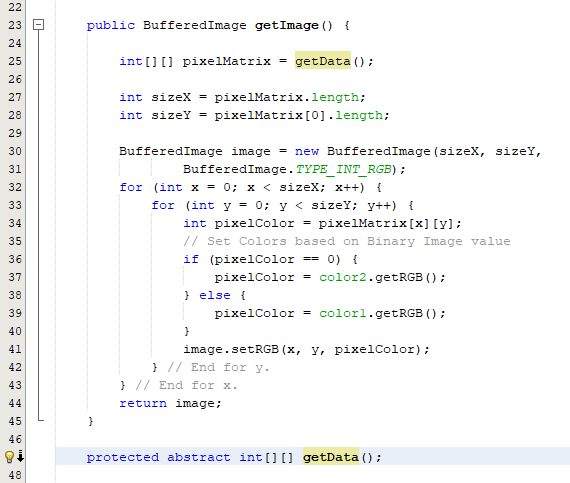


Figure 5 The getData() and getImage() methods

1. The **ImageNumber** class defines the pixel data of a number 7 digit. This is going to be used by our 2D graphics animation in order to display the digit 7 and apply transformations to it.

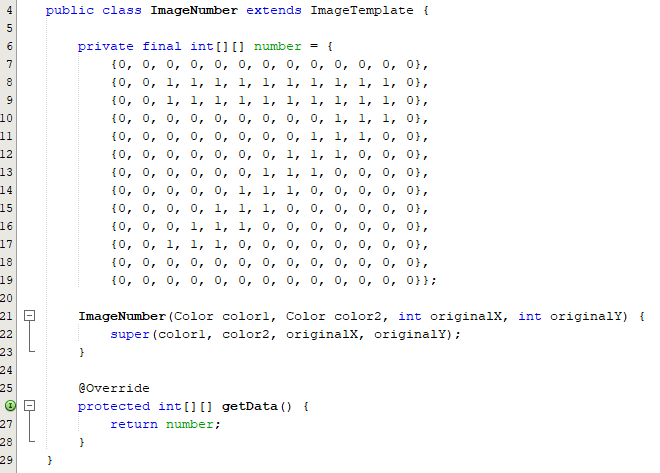


Figure 6 The ImageNumber class

1. The **ImageCreator** class only defines a method which initializes instances of the ImageTemplate class based on a given image type. It initializes the instances with given characteristic values, such as main color, secondary color, original offset on the X axis and original offset on the Y axis.

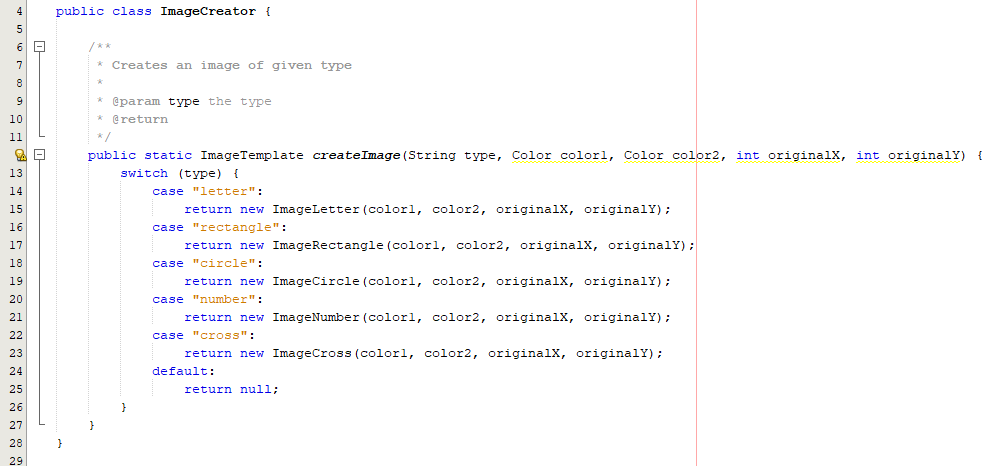


Figure 7 The ImageCreator class

1. The HW2 class extends the JPanel class, defines the flow of the 2D animation and contains the main method for this application. It holds the current frame number, the current Graphics2D transformation and 5 instances of the image template one for each of the following: letter T, rectangle, circle, number 7, cross.



Figure 7 The HW2 class

In the HW2 class, the main method initializes the animation and handles the frame number logic.

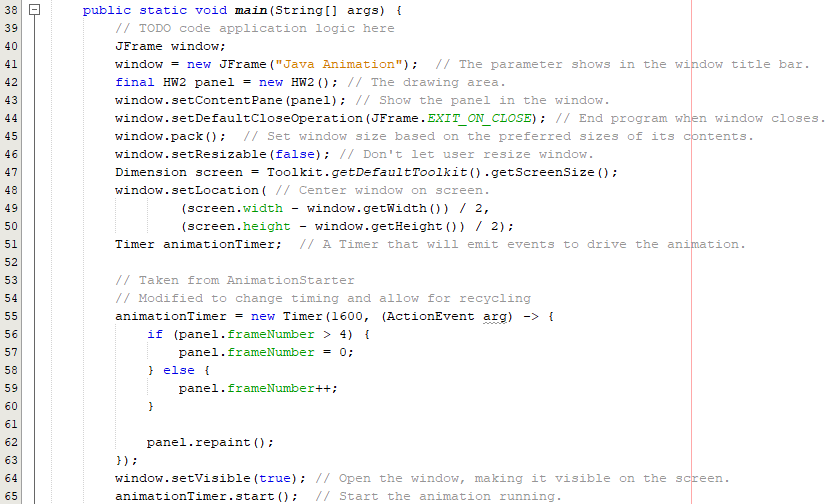


Figure 8 The HW2 main method

The HW2 class, extending the JPanel class, overrides the paintComponent() method. This helps us gain access to the painting mechanism on Panel, based on the frame number. We want to define a different transformation for each frame: 0 -> no transformation, 1 -> translations (-12,12), 2 -> counter clockwise rotation by 55 degrees, 3 -> clockwise rotation by 75 degrees, 4 -> scaling (3, 1.5). Although, not required, a 5th frame was added to scale the images down close to the original size for the whole model to not spiral out too quickly.

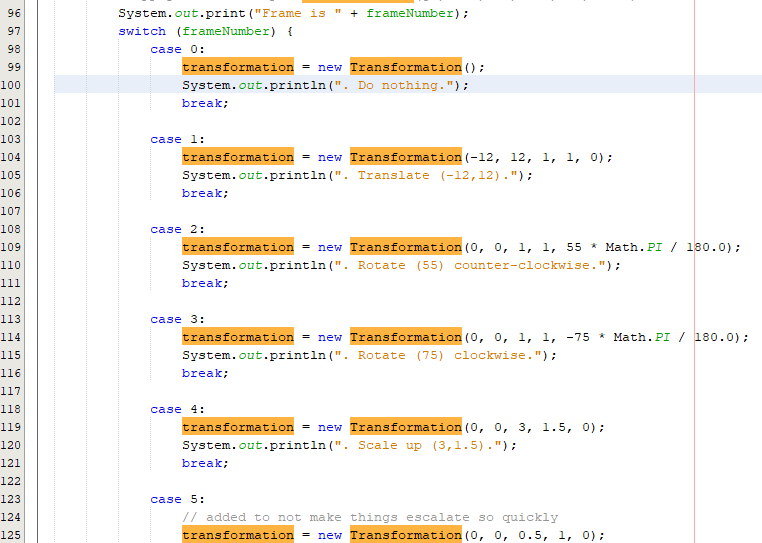


Figure 9 Each frame defines a different transformation

There are 2 situations which must be taken into consideration. The first situation is the original displaying of the images, where we position each image close to the center of the panel, with the original offset measured. The other situation is every frame which follow the initial set-up, where the images are all drawn with the current transformation defined by the current frame number.

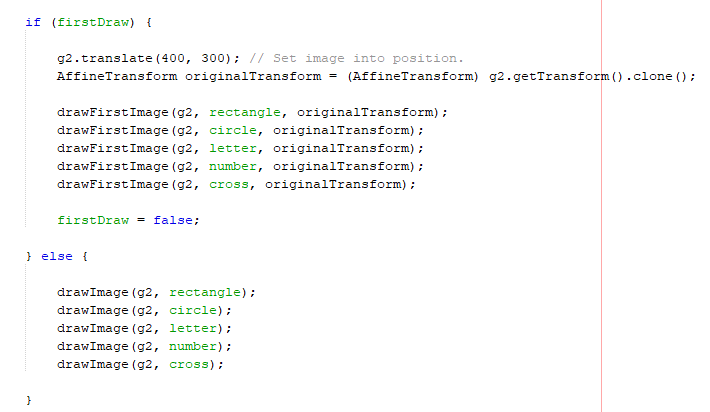


Figure 10 Two different drawing situations

The method **drawFirstImage** defines the initial drawing of the images. It takes the original transform (sets the image in the center of the panel) then translates the image by the original offset. Subsequently it draws the image on the panel. Finally, it saves the current transformation to the current image template, so that it is used as starting point for the next drawing.

The method **drawImage** defines the normal drawing for the images. It takes the current transform saved to the image template in the previous frame and sets it. Subsequently it performs the necessary translation, rotation or scaling, depending on the current transformation and draws the image. Finally, it saves the current transformation to the current image template, so that it is used as starting point for the next frame.

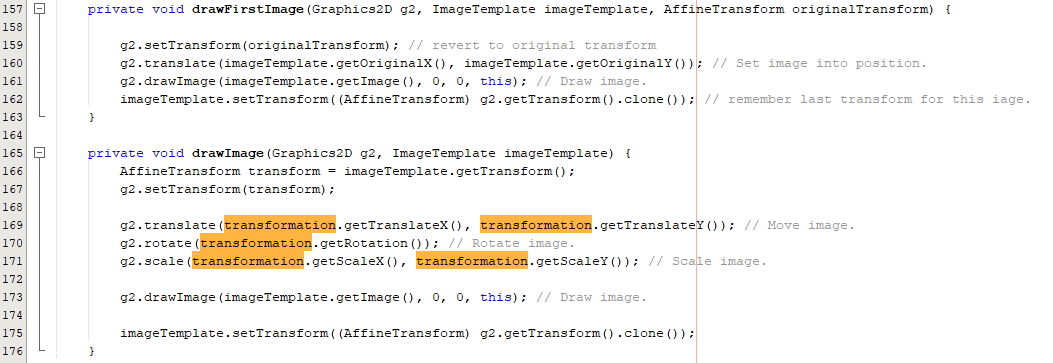


Figure 11 The drawFirstImage and drawImage methods

# **Screen capture demonstrating the successful compilation and running of the project**

1. Run the app. The images appear close to the center of the application window. In each frame they move independently in the application window. We can trace the frames in the console window.
2. Frame 0: do nothing

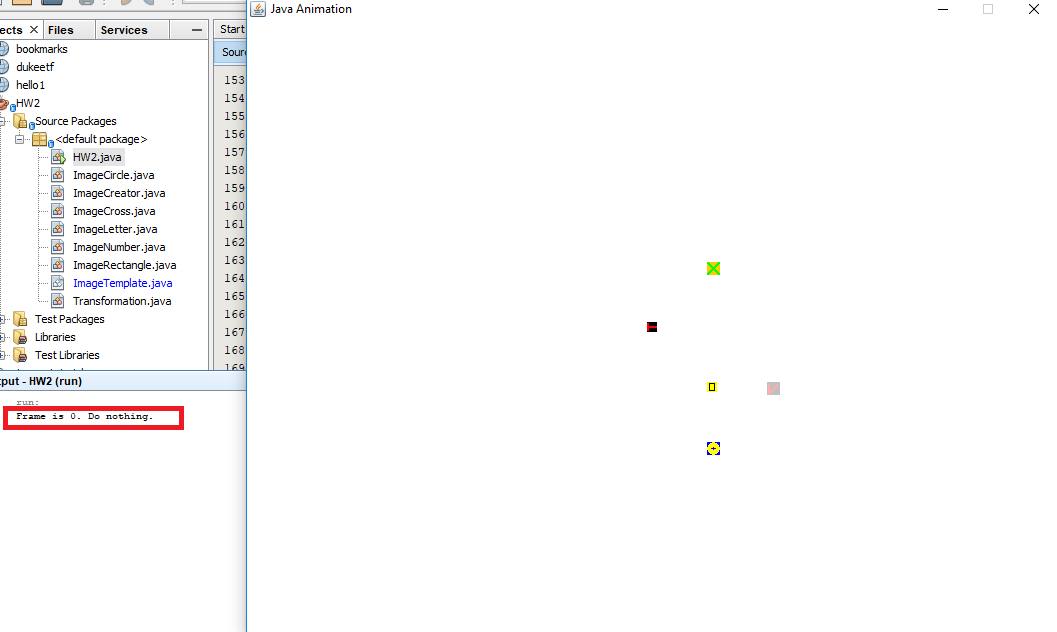


Figure 12 Frame 0

1. Frame 1: Translate -12 on X axis and 12 on Y axis.

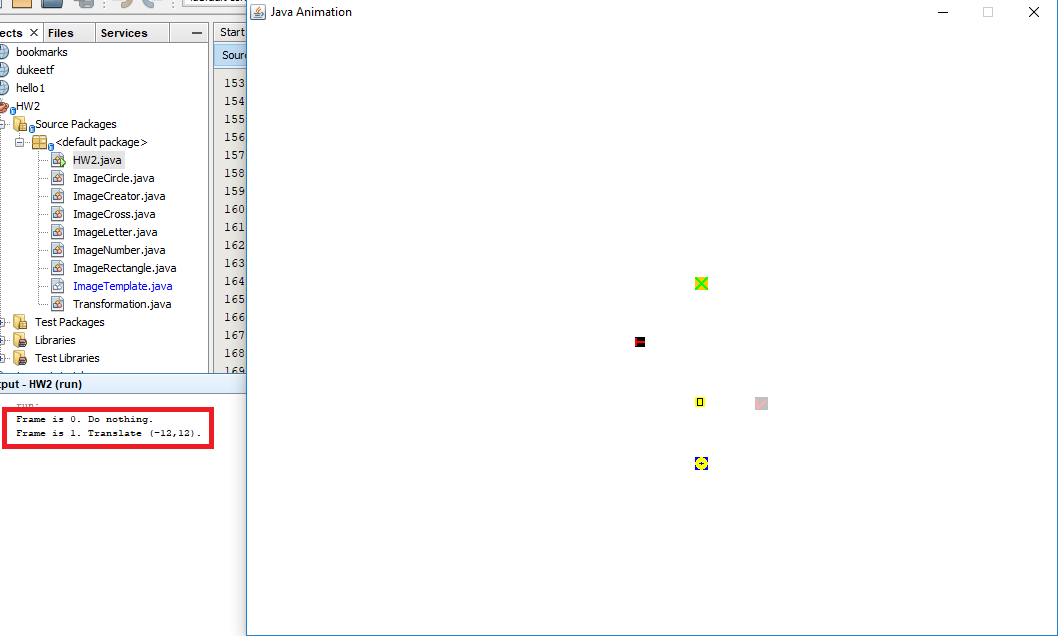


Figure 13 Frame 1

1. Frame 2: Rotate 55 degrees counter clockwise

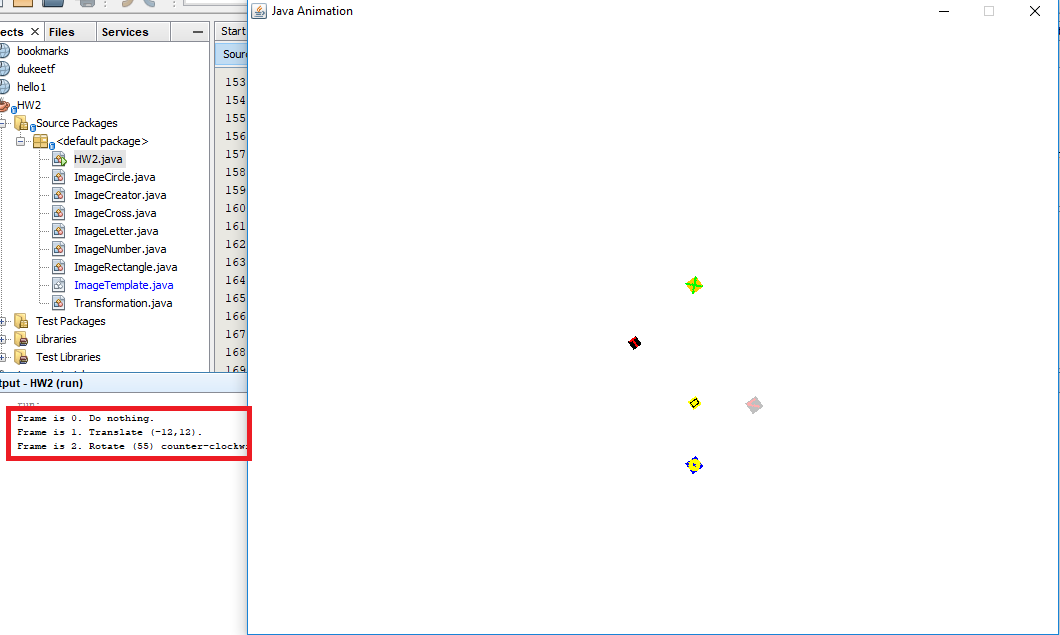


Figure 14 Frame 2

1. Frame 3: Rotate 75 degrees clockwise

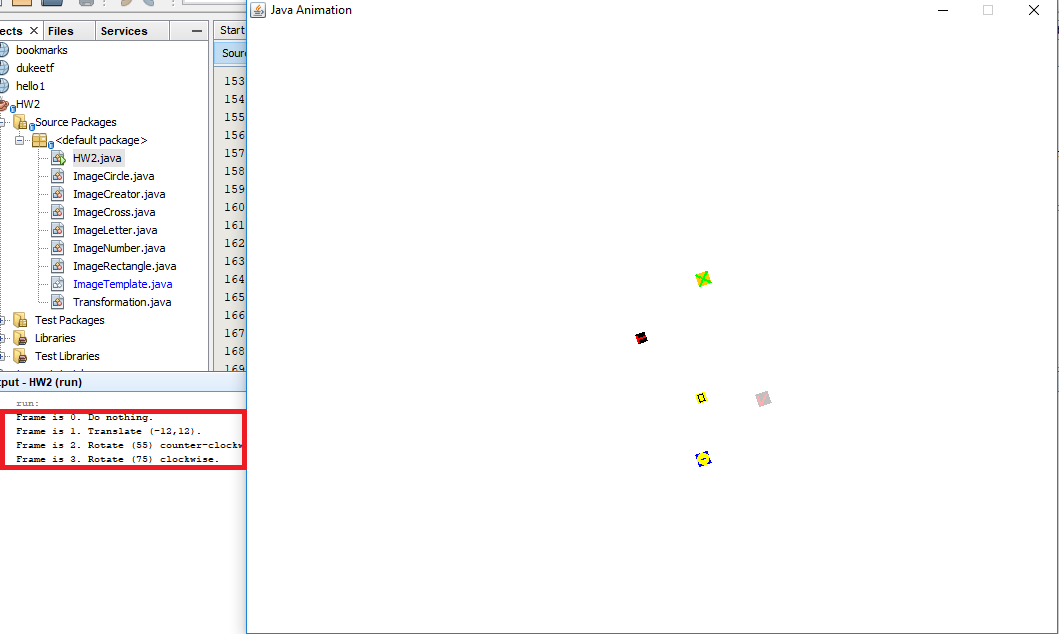


Figure 15 Frame 3

1. Frame 4: Scale up 3 times on the X axis and 1.5 times on the Y axis

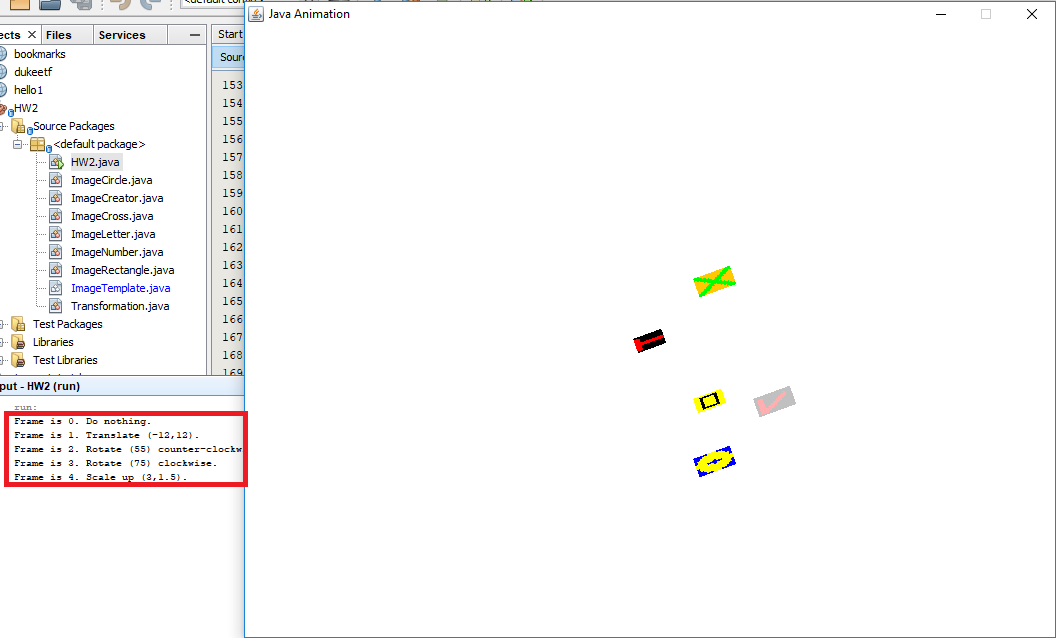


Figure 16 Frame 4

1. Frame 5: Scale down 0.5 on the X axis.

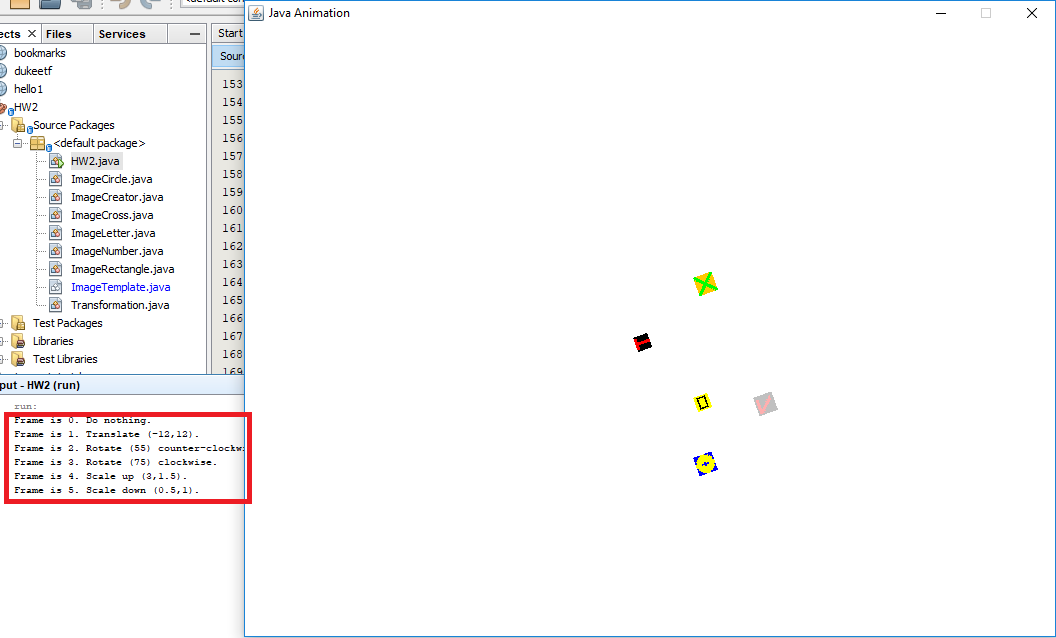


Figure 17 Frame 5

1. Frame 0: The animation restarts from 0 endlessly, however, the starting point is always at the last drawing position. As you can imagine, the images get larger, also more and more distant from the center and rotating endlessly. At some point they might look like this:

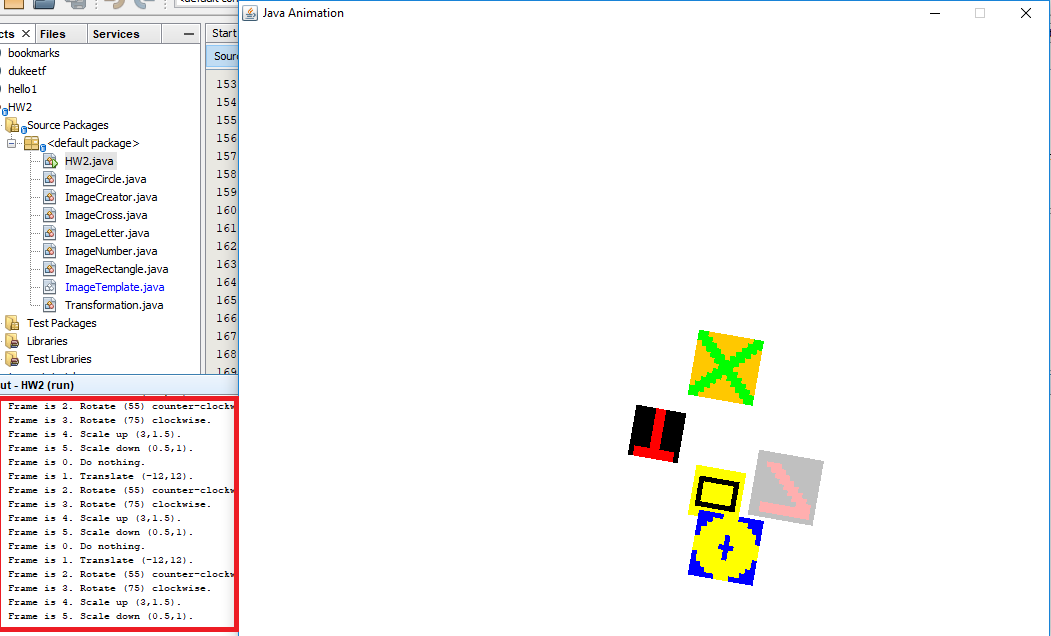


Figure 18 Frame x

# **Lessons Learned**

1. I learned how to work with the Graphics2D class.
2. I learned how to work with the AffineTransform class.
3. I learned how to draw, translate, rotate and scale graphic elements in java.