

Lab 10 – posted 11/9, due 11/21

Objective – create a frame of (at least 10) **moving** objects, i.e. fish in a tank, bugs on the screen, butterflies in a field, little robots, or anything you'd like. Except stars. You CAN'T use stars.

Must be used – an array of polygons for your creatures, and an array of colors to give each creature a unique color. You must also have a button to add creatures, and one to delete creatures.

A note about arrays –

You can create an array like this:

```
int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
```

if you know each element's value ahead of time, or like this:

```
int xPoints[] = new int[10];
```

and assign the values later, if you don't know each element's value ahead of time.

But arrays can be anything, even objects. To follow the example we used in class, we can create an array of 10 star Polygons like this:

```
private int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
private int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
private Polygon star[] = new Polygon[ 10 ];
```

however (and this is important), that last statement **only creates the room in computer memory to hold the array**, but does not create each element. To do that, each separate element must be created with its own **new** command. Therefore, creating an array of objects is always a two-step process:

```
// set up all the stars 0 thru 9
for (int x=0; x<=9; x++)
{
    star[x] = new Polygon( xPoints, yPoints, 10 );
}
```

There. Now, each star can be used separately, like:

```
star[5].translate(0,20); // will move the sixth star 20 pixels right and 10 down
```

Remember arrays start at 0.

Lab instructions:

1. You must use Eclipse to submit a JAR file. See Lab 9 Parts 1 and 3 for instructions. If you don't, you will lose 20 points.
2. You'll need a constructor, paint method, and an actionPerformed method: your class will have to extend JFrame and implement ActionListener.

In your constructor:

3. Create an array of polygons to represent your creatures.
4. Create an array of colors to give each creature a unique color... that is `creatureColor[6]` will go with `creature[6]`.
5. Use a Timer to call your `actionPerformed` method.
6. Since you're using your frame to paint creatures, you CAN'T use it also for your Increase and Decrease buttons. You'll have to create a second frame.

```
private JFrame controlFrame = new JFrame();
private JButton increaseButton = new JButton("new star");
```

and then in your constructor:

```
public Star() // constructor
{
    // main frame
    setSize( width, height );
    setLocation( 10, 10 );
    setDefaultCloseOperation( EXIT_ON_CLOSE );

    // separate control frame
    controlFrame.setLayout( new FlowLayout() );
    controlFrame.setSize( 200, 200 );
    controlFrame.setLocation( 600, 100 );
    controlFrame.add(increaseButton);
    increaseButton.addActionListener( this );
    controlFrame.setVisible( true );
}
```

Note that "controlFrame." appears whenever addressing that separate frame, making it different from the JFrame child that is our actual class.

In your paint method:

7. Your paint method should first paint a big rectangle of a suitable color (foam green for fish, for example), and then paint each creature at its new location:

```
g.setColor( suitableColor );
g.fillRect( 0,0, width, height);

for (x=0; x<=starCount; x++ )
{
    g.setColor( starColor[ x ] );
    g.fillPolygon( star[ x ] );
}
```

Note the use of an upper limit ("starCount") in the `for` loop. Why wouldn't the upper limit just be the number of elements in the array? Because I want to set that limit (how many of my 10 possible

creatures are actually drawn) using two buttons: one to increase the limit, one to decrease. Therefore, starCount will increase (up to 9) when I press my Increase button, and decrease (down to zero) when I press my Decrease button.

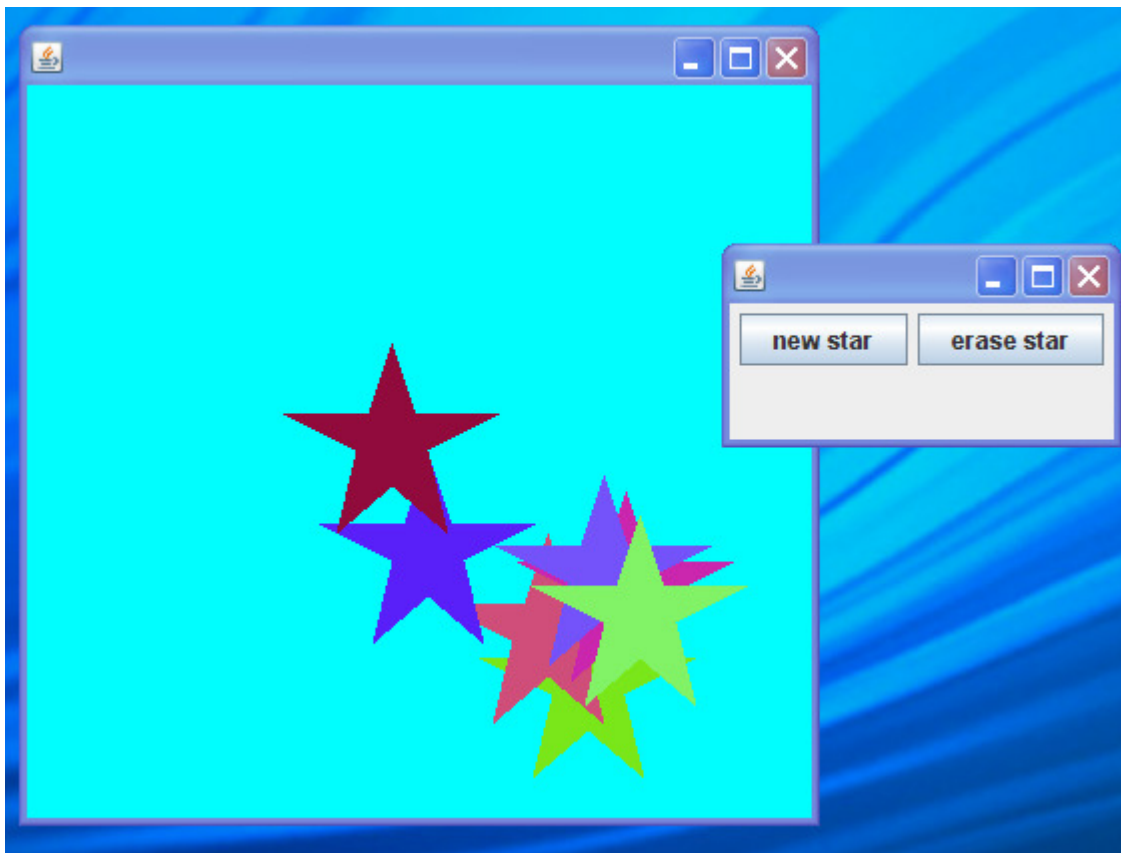
In your actionPerformed method:

8. Your actionPerformed method can use the Polygon method `.translate(x, y);` to move your creature to a new location. Have the creature move in a way that makes sense: randomly for fish, in straight lines for robots, etc.

9. Every time the Timer goes off, make sure you call the repaint() method.

10. Since the Timer and the two buttons share the same actionPerformed method, you'll have to use `if (e.getSource() ==` to tell which event just occurred.

Here's an example of the output, using my stars:



Optional, but useful:

Here's a new idea!

To make your work a little easier, create a Fish class, or a Robot class, or Butterfly class to encapsulate all of the information each creature needs.

A StarClass would look like this:

```
import java.awt.*;
public class StarClass
{
private int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
private int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
public Polygon starImage = new Polygon(xPoints, yPoints, 10);
public Color starColor = new Color(0,0,0);

public StarClass()
    {
    }
} // end class
```

Now, to use the class, any other class could create an array:

```
private StarClass star[] = new StarClass[ 10 ];
```

and then initialize all of the relevant class variables:

```
for (x=0; x<limit; x++)
{
    star[x] = new StarClass();
    star[x].starImage.translate(0,20);
    star[x].starColor = new Color( 255, 0, 0 );
}
```

and you could paint each creature like this:

```
g.fillPolygon( star[ x ].starImage );
```