Creating Graphical Applications with the Swing Toolkit

Most graphical applications are built upon an application toolkit that provides many classes to make the job easy. We will use Swing but there are at least 3 other Java toolkits; AWT, SWT and JavaFX. Each of these toolkits is quite complex. We will only scratch the surface of Swing, learning just enough about the toolkit to create a simple graphical application. If you are interested, there is a lot more to learn on your own or with the help of Mr. Malloy.

Creating graphical applications is fun but another important aspect of this lesson is to demonstrate how abstract classes and interfaces are used, extensively, in real world programs. To create our simple application you need to know about 3 classes provided by Swing. You will subclass one of these Swing classes. In addition, you will implement several interfaces that Swing expects in order to implement things like processing input from the keyboard and mouse.

The 3 classes you will learn to use are named JFrame, JComponent and Graphics.

Before we get to that, though, you need to create the runner class for your Swing application. Call it MySwingApplication. Like all of our runner classes, this class will have a main method. This main method is different from those you created in previous labs or projects. This main method has only 2 statements. The first creates an instance of itself; i.e. a MySwingApplication object. The second calls a Swing utility method, invokeLater. Here is the complete main method:

```
public static void main() {
    MySwingApplication mySwingApplicationSwing = new MySwingApplication();
    javax.swing.SwingUtilities.invokeLater(mySwingApplicationSwing);
}
```

So, what is going on here? The Swing toolkit wants to take control of your program. Once it takes control, it will call methods that you supply through interfaces it defines when something happens in your application to which you need to respond. Again, the best examples are keyboard or mouse input but there are many more. This short but obscure piece of code hands control of your program to Swing.

Lets look at the invokeLater method in more detail. It is a static method defined in the SwingUtilities class.

Class SwingUtilities

```
public class SwingUtilities extends Object implements SwingConstants
```

A collection of utility methods for Swing.

Method

```
static voidinvokeLater(Runnable runnable)
invokeLater takes a single input parameter, an object that
implements the Runnable interface, as defined below:

     interface Runnable { public void run(); }
```

Any class that implements the Runnable interface must define a run method that takes no arguments and returns no value. In the case of the runnable parameter to the invokeLater method, the run method of that parameter will be called by Swing when it is good and ready for your program to continue executing. You will put the initialization code that normally goes into your main method into this run method. The implementation of main I provided above passes the mySwingApplication object to invokeLater, so the MySwingApplication class must implement the Runnable interface. Got it?

Lets put all of this together in a running class.

Activity 1 – Creating MySwingApplication

1. Create a new project. Call it MySwingApplication.
2. Create a new class. Call it MySwingApplication, too.
3. The MySwingApplication class file should import all of the Swing packages;
   import javax.swing.*
4. The MySwingApplication class must implement the Runnable interface.
   a. Add "implements Runnable" to the end of the class definition statement for MySwingApplication.
   b. Define a public method, run, that takes no parameters and returns no value in your MySwingApplication class.
5. Define the main method for MySwingApplication, as described above.
6. Make sure your application compiles. It won't do anything useful, so no need to execute it, yet.

Next you will create a window on the screen within which your application will display its output. To do this, you need to learn about the JFrame class:

Class JFrame

```
public class JFrame extends Frame implements
WindowConstants, Accessible, RootPaneContainer

A JFrame is a top-level window with a title and a border.
```

Some Constructors

JFrame()
Constructs a new top-level window that is initially invisible.

JFrame(String title)
Constructs a new top-level window that is initially invisible with the specified title.

Some Methods

public void setSize(int width, int height)
sets the width and height of the JFrame. The JFrame constructor sets the width and height to 0, so a call to setSize is required to have a useful top-level window.

public void setVisible(boolean b)
Shows or hides this top-level window depending on the value of parameter b. The JFrame constructor initializes the window to invisible, so a call to setVisible is required to have a useful top-level window.

public void setDefaultCloseOperation(int operation)
Sets the operation that will happen by default when the user initiates a "close" on this frame. You must specify one of the following choices:

- DO_NOTHING_ON_CLOSE (defined in WindowConstants): Don't do anything; require the program to handle the operation in the windowClosing method of a registeredWindowListener object.
- HIDE_ON_CLOSE (defined in WindowConstants): Automatically hide the frame after invoking any registered WindowListener objects.
- DISPOSE_ON_CLOSE (defined in WindowConstants): Automatically hide and dispose the frame after invoking any registered WindowListener objects.
- EXIT_ON_CLOSE (defined in JFrame): Exit the application using the System exit method. Use this only in applications.

The value is set to HIDE_ON_CLOSE by default, so a call to this method with parameter JFrame.EXIT_ON_CLOSE is advisable.

public Component add(Component comp)

```
Adds the specified component to the contents of this JFrame.
This method is actually inherited by JFrame from one of its
parent classes, which we are not discussing in this lesson.

If you are curious to learn more about JFrame, more complete
documentation is found at
```
https://docs.oracle.com/javase/6/docs/api/javax/swing/JFrame.html

Activity 1a – Creating a top-level window

7. Declare an instance variable of type JFrame in the MySwingApplication class. Call it jFrame.
8. In the run method of the MySwingApplication class:
    a. Initialize jFrame with a title of your choice.
    b. Set the size of jFrame to something interesting; for example, 400 by 400.
    c. Set the default close operation of jFrame to the constant value, `JFrame.EXIT_ON_CLOSE`.
    d. Using the setVisible method, make jFrame visible on the screen.
9. Compile and run your application. Your window should appear on the screen!

Next you will draw something in the window you just created. To do this, you need to learn about the Graphics and JComponent classes:

Class Graphics

```
public abstract class Graphics extends Object

A Graphics object encapsulates state information needed for
the basic rendering operations that Java supports. This
state information includes the following properties:

The Component object on which to draw.
The current color.
The current font.
```

Some Methods

```
public abstract void fillRect(int x,
            int y,
            int width,
            int height)
```
Fills the specified rectangle with the current color.

```
public abstract void fillOval(int x,
            int y,
            int width,
            int height)
```
Fills the specified oval with the current color. Note that a circle is just an oval with width == height.

```
public abstract void setColor(Color c)
```
Sets this graphics context's current color to the specified color. All subsequent graphics operations using this graphics context use this specified color. We haven't talked about the Color class but note that it has static properties for common colors; e.g. Color.black, Color.red, Color.green, Color.blue, …

If you are curious to learn more about Graphics, more complete documentation is found at https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

Class JComponent

```
public abstract class JComponent extends Container
implements Serializable
```

JComponent is the base class for all Swing components. Notice it is an abstract class! This means you will have to define your own component that extends JComponent. There are only two methods defined by JComponent that you need to know about now:

```
protected void paintComponent(Graphics g)
```
Swing calls the paintComponent method when it wants your application to display the contents of the component. You actually draw by calling methods in the supplied Graphics object.

```
public void setSize(int width, int height)
```
sets the width and height of the JComponent. This is exactly the same method as we learned, above, for JFrame. In fact, both JFrame and JComponent are subclasses of another class where setSize is defined but I am glossing over those details, here.

Activity 2 – Displaying a simple shape

1. Create a new class. Call it CanvasComponent. Modify the class declaration that BlueJ creates for you so that CanvasComponent extends JComponent. Import all the awt and swing packages so that you can access the necessary class definitions:
   a. import java.awt.*
   b. import javax.swing.*
2. Declare properties that describe the simple shape you want to display. For instance, to display a rectangle you will need properties to record the x position, y position, width and height. You need similar properties for a circle or oval.
3. Declare a constructor for CanvasComponent that takes two input parameters, a width and height both of type int.
   a. Use the setSize method to set the size of the CanvasComponent to the width and height parameters provided.
   b. Initialize the properties you declared in 2 to appropriate initial values so that the entire shape fits within the width and height of the CanvasComponent.
4. Declare a paintComponent method with input parameter and return value as described above.
   a. Set the color of the graphics parameter to a color of your choice; e.g. Color.red.
   b. Display your simple shape. For instance, if you chose a rectangle to display, call the fillRect method of the graphics parameter object.
5. In your SwingApplication class, add a canvas component object to your frame:
   a. Declare a property of type CanvasComponent, named canvasComponent.
   b. In your run method, after you set the size and before you call setVisible, initialize canvasComponent by calling the class constructor.
   c. Add canvasComponent to the jFrame object you created on activity 1a, using the add method jFrame.
6. Compile and run your application. Your simple shape should appear in your window!

You now have a very simple, static application; a window that contains a single, simple shape. In order to make your application more dynamic, you need to respond to user input. In Activity 3, you will use the mouse to drag your simple shape around on the screen.

To accomplish this your JComponent must implement not one but two interfaces, the MouseListener and MouseMotionListener interfaces.

Interface MouseListener

The MouseListener interface defines 5 methods:
void mouseClicked(MouseEvent e)
This method is called by Swing when a mouse button is pressed and released.

void mousePressed(MouseEvent e)
This method is called by Swing when a mouse button is pressed.

void mouseReleased(MouseEvent e)
This method is called by Swing when a mouse button is released.

void mouseEntered(MouseEvent e)
This method is called by Swing when the cursor moves inside the bounding box of the component that defines the MouseListener interface.

void mouseExited(MouseEvent e)
This method is called by Swing when the cursor moves outside the bounding box of the component that defines the MouseListener interface.

Interface MouseMotionListener

The MouseMotionListener interface defines 2 additional methods:
void mouseDragged(MouseEvent e)
This method is called by Swing when a mouse button is pressed and then the mouse is moved with the button depressed.

void mouseMoved(MouseEvent e)
This method is called by Swing when the mouse is moved without any button depressed.

To actually receive mouse events from the Swing toolkit, you also need to tell Swing that your JComponent wants to receive these events. The JComponent class defines two additional methods to accomplish this:

Additional JComponent Methods

public void addMouseListener(MouseListener l)
Adds the specified mouse listener to receive mouse events from this component.

public void addMouseMotionListener(MouseMotionListener l)
Adds the specified mouse motion listener to receive mouse events from this component.

public void repaint()

Notifies Swing that the appearance of this component changed needs to be redrawn. As a result Swing will call to this component's paintComponent method as soon as possible.

If you are curious to learn more about JComponent, more complete documentation is found at
https://docs.oracle.com/javase/7/docs/api/javax/swing/JComponent.html.

Activity 3 – Moving your simple shape

To complete this activity, you only need mousePressed, mouseReleased and mouseDragged methods but if you implement an interface you must provide all methods, even if they don't do anything. So, we will declare the other methods but they will have empty method bodies.

1. Modify the class declaration for CanvasComponent to indicate it implements MouseListener and MouseMotionListener. These interfaces are defined in the event sub-package of the awt package so you will need to import some more definitions. Add the following line to the CanvasComponent class, at the top of the file adjacent to the other import statements.
    a. import java.awt.event.*;
2. Declare the 5 MouseListener methods and the 2 MouseMotionListerner methods in the methods section of CanvasComponent. For now, make all the method bodies empty; i.e. just open curly bracket, closed curly bracket.
3. Implement the mousePressed method of CanvasComponent, according to the following instructions:
    a. Retrieve the position of the cursor when the mouse button was pressed from the MouseEvent parameter, e, to the mousePressed method. The MouseEvent class defines two methods to do this; e.getX () and e.getY(). Both getX and getY take no parameters and return a result of type int. Assign these return values to two new CanvasComponent properties, mouseFromX and mouseFromY. You are storing these results in properties so that you can access them later in the mouseDragged method.
    b. If the mouseFromX and mouseFromY are inside the rectangle defined by rectX, rectY, rectWidth and rectHeight then your shape will move when the mouse moves. Check for this condition and record the result in a new CanvasComponent property, named shapeSelected of type boolean. You will examine this property in the mouseDragged method, too.
4. Implement the mouseDragged method of CanvasComponent, according to the following instructions. If your shape is selected, as indicated by the shapeSelected property:

    a. Retrieve the position of the cursor when the mouse button was pressed as you did in 3a, above. Assign the x and y position to local variables, mouseToX and mouseToY.
    b. Increment rectX and rectY by the difference between the current position minus the position when the button was pressed. After you do this, update mouseX and mouseY to the new cursor position, from a.
    c. Ask Swing to redraw your CanvasComponent by calling CanvasComponent's repaint method.

5. Tell Swing that your CanvasComponent wants to receive mouse events.
    a. In the constructor for CanvasComponent, call the addMouseListener method.
       FINE POINT: CanvasComponent plays two roles in this step. It is the component from which the mouse events should come; i.e. your program will only receive mouse events that happen within your canvas, not on the frame's menu bar or in some other window. It is also the object that implements the MouseListener interface; i.e. the object you want to receive the mousePressed and other method calls defined by the MouseListener interface. So, your call to addMouseListerner should look something like this:
       this.addMouseListener(this);
       the first "this" in the above statement is acting like a JComponet; you are calling the addMouseListerer method defined by the JComponent class. The second "this" is acting like an object that implements the MouseListener interface. It happens to be the same object but it need not be!
    b. Similarly, call the addMouseMotionListener method so that your CanvasComponent will receive mouse move and drag events.
6. Compile and run your application. Click on your shape and drag. Your shape should move with the mouse!

Adding animation to your application is simple. The Swing toolkit provides a class, called Timer, which will call a method in your application at regular intervals. The method it calls is the single method, called actionPerformed, of a new interface, ActionListener, that you will implement. In Activity 4, you will use a timer object and your implementation of the ActionListener interface to animate your simple shape.

Class Timer

```
public class Timer extends Object implements Serializable
```

```
Timer fires one or more ActionEvents at specified intervals.
An example use is an animation object that uses a Timer as
the trigger for drawing its frames.
```

Constructor

```
Timer(int delay, ActionListener listener)
Creates a Timer and initializes both the initial delay and
between-event delay to delay milliseconds.
```

Method

```
public void start()
Starts the Timer, causing it to start sending action events
to its listeners.
```

If you are curious to learn more about Timer, more complete documentation is found at https://docs.oracle.com/javase/7/docs/api/javax/swing/Timer.html.

Interface ActionListener

The **ActionListener** interface defines 1 method:

`void actionPerformed(ActionEvent e)`

Invoked when an action occurs. actionPerformed is called by a timer object at regular intervals determined by the delay parameter to the constructor.

Additional JComponent Method

public Dimension getSize()
Returns the size of this component in the form of a Dimension object. The Dimension class is very simple. It contains two public properties, width and height. They are public so you may access them directly. The height field of the Dimension object contains this component's height, and the width field of the Dimension object contains this component's width. Alternatively, you may use JComponent accessor methods, getWidth and getHeight.

Activity 4 – Animating your simple shape

1. Modify the class declaration for CanvasComponent to indicate it implements ActionListener.
2. Declare 4 new properties for CanvasComponent. You will use these properties to control the movement of your shape in your actionPerformed method, below.
    a. To keep track of the amount to move your shape each time actionPerformed is called, declare integers animationDeltaX and animationDeltaY. The initial animation motion will be from left to right so initialize animationDeltaX to 1 and animationDeltaY to 0.

   b. To ensure your animation shape does not move too far left, right, up or down declare two integers, gutterX and gutterY. Initialize them both to 10.

3. Declare and implement an actionPerformed method in the methods section of CanvasComponent. It will move your shape if it is not selected. In that case:
   a. Find the width and height of your canvasComponent by calling its getSize component. Store the result in a local variable of type Dimension, called componentSizeDimension.
   b. The next position for your shape, if it keeps moving in the same direction, is the current position, as indicated by rectX and rectY, plus the change in position per delay, indicated by animationDeltaX and animationDeltaY. Update rectX, rectY, animationDeltaX and animationDeltaY according to the following rules:
      i. If the new right side of your shape plus gutterX exceeds the component width then update animationDeltaX and animationDeltaY so that your shape starts moving down. Set rectX to its maximum value; i.e. component width – rectWidth – gutterX. Increment rectY by the new animationDeltaY.
      ii. If the new bottom of your shape plus gutterY exceeds the component height then update animationDeltaX and animationDeltaY so that your shape starts moving left. Set rectY to its maximum value; i.e. component height – rectHeight – gutterY. Increment rectX by the new animationDeltaX.
      iii. If the new left side of your shape is less than gutterX then update animationDeltaX and animationDeltaY so that your shape starts moving up. Set rectX to its minimum value; i.e. gutterX. Increment rectY by the new animationDeltaY.
      iv. If the new bottom of your shape is less than gutterY then update animationDeltaX and animationDeltaY so that your shape starts moving right. Set rectY to its its minimum value; i.e. gutterY. Increment rectX by the new animationDeltaX.
      v. If none of the above conditions is true, simply increment rectX and rectY by animationDeltaX and animationDeltaY.
   c. Ask Swing to redraw your CanvasComponent by calling CanvasComponent's repaint method.
4. Declare an instance variable of type Timer in the CanvasComponent class. Call it animationTimer.
5. At the end of the constructor of the CanvasComponent class:
   a. Initialize animationTimer by calling the Timer constructor. Remember, the constructor takes two parameters a delay (try 20 milliseconds) and an object that implements the ActionListener interface (that would be your canvasComponent object).
   b. Start the animationTimer by calling its start method.
6. Compile and run your application. Your shape should move right to left, then top to bottom, then left to right and then bottom to top. If you click the shape

it should still move with the mouse until you release the mouse button. At that point it should continue its march.

At this point your application can create a window, draw a simple shape, respond to mouse input and animate your shape. The last mode of input you will implement is keyboard input. You will enhance your application so that if you press the '+' key the speed of your animated shape increases by 1 unit. If you press the '-' key the speed will decrease by 1 unit. To do this you will implement one more interface (naturally!), the KeyListener interface.

Interface KeyListener

The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event implements this interface.
The listener object created from that class is then registered with your frame using the component's addKeyListener method. A keyboard event is generated when a key is pressed, released, or typed. The relevant method in the listener object is then invoked, and the KeyEvent is passed to it.

The KeyListener interface defines 3 methods:
void keyTyped(KeyEvent e)
Invoked when a key has been typed. Implementing this method is usually enough for most applications, and it will be for ours

void keyPressed(KeyEvent e)
Invoked when a key has been pressed. You will leave the implementation of this empty.

void keyReleased(KeyEvent e)
Invoked when a key has been released. You will leave the implementation of this empty.

Additional JFrame Method

public void addKeyListener(KeyListener l)
Adds the specified key listener to receive key events from this window.

Activity 5 – Changing the speed of your moving shape

1. Modify the class declaration for CanvasComponent to indicate it implements KeyListener.
2. Declare a new property for CanvasComponent. Call it motionSpeed. Initialize it to 1. This property will be used in your keyTyped and actionPerformed methods, described below.

3. Declare the 3 KeyListener in the methods section of CanvasComponent. For now, make all the method bodies empty; i.e. just open curly bracket, closed curly bracket.
4. Implement the keyTyped method of CanvasComponent, according to the following instructions:
    a. Retrieve the character typed from the KeyEvent parameter, e, to the keyTyped method. The KeyEvent class defines a method to do this; e.getKeyChar (). getKeyChar takes no parameters and returns a result of type char. Save the returned result in a local variable of the keyTyped method. Call it keyChar.
    b. If keyChar == '+' then increase motionSpeed by 1. If it is '-' and motionSpeed is greater than 0 then decrease it by 1.
5. Modify your actionPerformed method to use motionSpeed. Wherever you increment rectX by anaimationDeltaX you should now increment it by animationDeltaX*motionSpeed. Likewise, for rectY and animationDeltaY.
6. In the run method of MySwingApplication, call the addKeyListener method of your frame. Your canvasComponent implements the KeyListener interface so pass it to addKeyListener as its one parameter.

7. Compile and run your application. You should be able to speed up the movement of your animated shape by pressing '+' and slow it down, even to a stop, by pressing '-'.

Congratulations! You have implemented a graphical application that can display a simple shape, respond to the most common forms of user input and animate your shape. With those building blocks you can create much more complex applications; more complex shapes, multiple shapes, more and different behaviors based on user input. If you want enhance you application design your own additions or try this bonus activity. I'm not going to give you detailed instructions for this activity, just the basic requirements.

Bonus Activity – Chase the cursor

Modify your application so that if you press the mouse button outside of the shape that the animated movement changes direction to move towards the position of the cursor/mouse. See if you can determine a direction so that the center of the shape passes through the location of the mouse press. The shape should move in this direction until it bumps into a window edge (minus the gutter).

If the user drags the mouse rather than just pressing and releasing, your shape should update its direction to continuously chase after the cursor on the screen.

When it is time to change direction you may resume the animated movement according to the existing logic or you may implement a new rule. For instance, you might use a “billiard ball” rule. Technically that is called making the angle of incidence equal to the angle of reflection. Ask for help if you need it to figure out how to do this.