Week 9
Graphical User Interfaces
(Chapter 10)

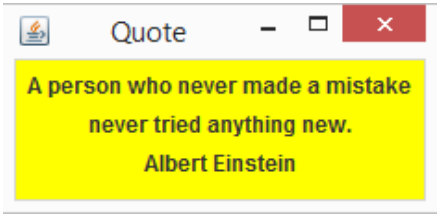
Useful for Lab 8 & 9

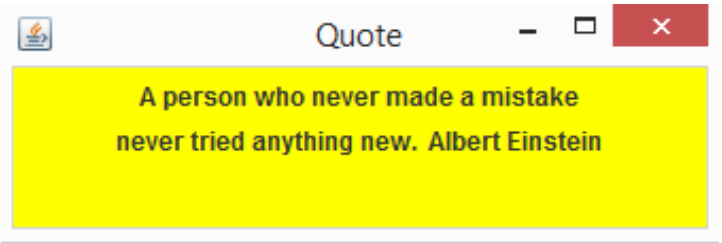
Graphical Applications:

- The example programs we've explored thus far have been text-based.
- They are called *command-line applications*, which interact with the user using simple text prompts.
- Let's examine some Java applications that have graphical components.
- These components will serve as a foundation to programs that have true graphical user interfaces (GUIs).
- A *GUI component* is an object that represents a screen element such as a button or a text field.
- GUI-related classes are defined primarily in the `java.awt` and the `javax.swing` packages.
- The *Abstract Windowing Toolkit* (AWT) was the original Java GUI package.
- The *Swing* package provides additional and more versatile components.
- Both packages are needed to create a Java GUI-based program.

Frame, Panels and Labels:

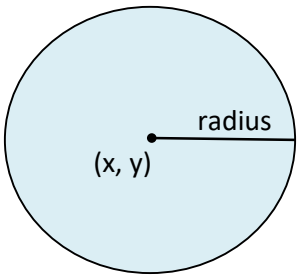
```java
//*****************************************************************
//  Quote.java
//  Demonstrates the use of frames, panels, and labels.
//*****************************************************************
import java.awt.*;
import javax.swing.*;

public class Quote
{
   //-----------------------------------------------------------------
   //  Displays some words of wisdom.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Quote");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel primary = new JPanel();
      primary.setBackground(Color.yellow);
      primary.setPreferredSize(new Dimension(250, 75));

      JLabel label1 = new JLabel("A person who never made a mistake");
      JLabel label2 = new JLabel("never tried anything new.");
      JLabel label3 = new JLabel("Albert Einstein");

      primary.add(label1);
      primary.add(label2);
      primary.add(label3);

      frame.getContentPane().add(primary);
      frame.pack();
      frame.setVisible(true);
   }
}
```



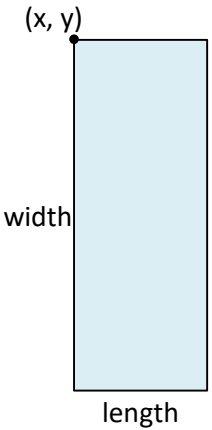


Circle, Rectangle, Square

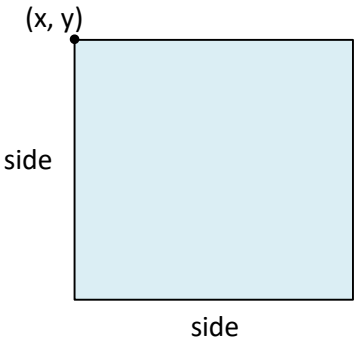
Circle:



Rectangle:



Square:



Shape Class

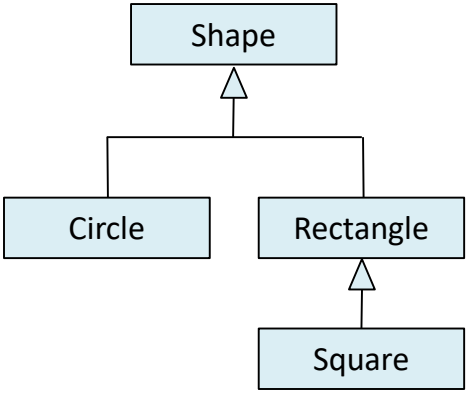
```
import java.awt.Color;     // Java class Color

abstract class Shape {
    private Color fillColor;
    private Color borderColor;
    private Boolean isFilled;           // true if shape is filled with color
    private Point Location;

    // the three constructors initialize the instance fields
    public Shape(Color fillColor, borderColor, int x, int y) { }
     // set borderColor to black since not provided
    public Shape(Color fillColor, int x, int y) { }                // Color.BLACK
     // set fillColor to white and border color to black
    public Shape(int x, int y { }                                  //Color.WHITE

    public void setFillColor(Color c) { }
    public Color getFillColor() { }
    public void setBorderColor(Color c) { }
    public Color getBorderColor() { }
    public Point getLocation() { }

    // Subclasses of Shape do not inherit private members, so we need methods the
    // subclasses can use to get the x, y values from the private Point instance field
    public int getX() { }
    public int getY() { }
    // if fillColor is white returns true, else returns false
    public boolean isFilled() { }
    // moves location by dx and dy
    private void moveLocation(int dx, dy) { }

    abstract double getArea();
    abstract double getPerimeter();
 }
```

Shape Class

- Circle, rectangle and square have similar attributes and behavior, use inheritance to reuse the code. UML diagram:

Circle:

- Add instance variable “radius”
- Implement the same number of constructors (with additional instance variable)
- Implement abstract methods
- Add method “getRadius”



Objects in GUI

- A Graphical User Interface (GUI) in Java is created with three kinds of objects:
  - components,
  - events,
  - listeners
- We've previously discussed *components*, which are objects that represent screen elements:
  - labels,
  - buttons,
  - text fields,
  - menus, etc.
- Some components are *containers* that hold and organize other components:
  - frames,
  - panels,
  - applets,
  - dialog boxes

GUI Development

- To create a Java program that uses a GUI we must:
  - instantiate and set up the necessary components
  - implement listener classes for any events we care about
  - establish the relationship between listeners and the components that generate the corresponding events
- We will explore some new components and see how this all comes together.
- A useful component is a *push button,* defined by the JButton class.
- A push button is a component that allows the user to initiate an action with a press of the mouse. It generates an *action event.*

PushCounter Example

- The `PushCounter` example displays a push button that increments a counter each time it is pushed.
- The components of the GUI are: the button, a label to display the counter, a panel to organize the components, the main frame.
- The `PushCounterPanel` class represents the panel used to display the button and label.
- The `PushCounterPanel` class is derived from `JPanel` using inheritance.
- The constructor of `PushCounterPanel` sets up the elements of the GUI and initializes the counter to zero.

```
//*****************************************************************
//  PushCounterPanel.java
//
//  Demonstrates a graphical user interface and an event listener.
//*****************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;
```

PushCounter Panel Example

```
//-----------------------------------------------------------------
   //  Constructor: Sets up the GUI.
   //-----------------------------------------------------------------
   public PushCounterPanel()
   {
      count = 0;

      push = new JButton("Push Me!");
      push.addActionListener(new ButtonListener());

      label = new JLabel("Pushes: " + count);

      add(push);
      add(label);

      setPreferredSize(new Dimension(300, 40));
      setBackground(Color.cyan);
   }
   //*****************************************************************
   //  Represents a listener for button push (action) events.
   //*****************************************************************
   private class ButtonListener implements ActionListener
   {
      //-----------------------------------------------------------
      //  Updates the counter and label when the button is pushed.
      //-----------------------------------------------------------
      public void actionPerformed(ActionEvent event)
      {
         count++;
         label.setText("Pushes: " + count);
      }
   }
}
```

String shown on the button

Listener

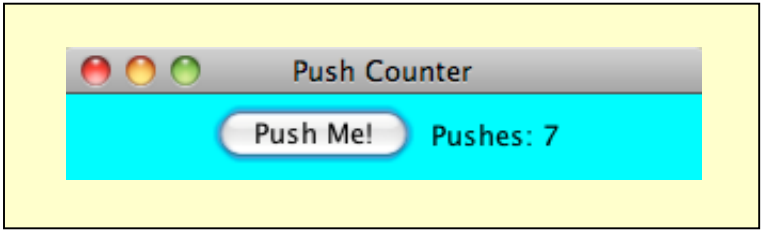
Add button to the panel

Add label to the panel

Where are these methods defined?

PushCounter Panel Example

```java
//*****************************************************************
//  PushCounter.java
//  Demonstrates a GUI and an event listener.
//*****************************************************************
import javax.swing.JFrame;

public class PushCounter
{
   //-----------------------------------------------------------------
   //  Creates the main program frame.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Push Counter");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```



Push Counter Example

- The `ButtonListener` class implements the `ActionListener` interface.
- The only method in the `ActionListener` interface is the `actionPerformed` method.
- The Java API contains interfaces for many types of events.
- The `PushCounterPanel` constructor:
  - instantiates the `ButtonListener` object
  - establishes the relationship between the button and the listener by the call to `addActionListener`
- When the user presses the button, the button component creates an `ActionEvent` object and calls the `actionPerformed` method of the listener.
- The `actionPerformed` method increments the counter and resets the text of the label.

Random Shape Generator

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;

// Main application for random shape generator app
public class ShapeWindow extends JFrame {

    JPanel shapeDriver;

    public ShapeWindow() {
        super();
        // TO-DO: set up the frame
    }

    public static void main(String[] args) {
        // run main application
        JFrame shapeWindow = new ShapeWindow();
    }
}
```

Shape Driver

```
// Driver program for random shape generator
public class ShapeDriver extends JPanel implements KeyListener {
    // Panel constants
    public final int FRAME_WIDTH = 600;
    public final int FRAME_HEIGHT = 600;
    private Random random;

    public ShapeDriver() {
        super();
        /* T0-DO:
        - set up JPanel
        - initialize any other field you want to declare and use
        - add the KeyListiner */
    }

    // Override
    public void paintComponent(Graphics g) {
        // call super class paintComponent method to color background
        super.paintComponent(g);

        // TO-DO use the different Shapes draw methods here
        // The draw methods in the different shapes should take
        // The Graphics object should be passed to the Shapes Draw method
    }

    // Override
    public void keyPressed(KeyEvent e) {
        /* To-DO:
          - if c, r, or s is pressed draw a circle, rectangle or square
          - repaint the JPanel */
    }
```

Hints

- Use an `ArrayList` to keep pointers to all shape objects.
- Whenever a key is pressed:
  - Generate a new shape of the type circle, rectangle or square, with random color, size and location
  - Put it in the `ArrayList`
  - Call method "`repaint`"
- When you make a change to the data, the component is not automatically painted with the new data.
- You must call the "`repaint`" method of the component, which will invoke "`paintComponent`" method with an appropriate Graphics object.
- You should not call the "`paintComponent`" method directly.

JComponent

- `JComponent` class from the `Swing` toolkit represents a blank component.
- Since we don't want to add a blank component, we have to modify the `JComponent` class and specify how the component should be painted.
- The solution is to declare a new class that extends the `JComponent` class.
- Method `paintComponent` is called automatically:
  - when the component is shown for the first time
  - when the window is resized
  - or when it is shown again after it was hidden
- Example of using `JComponent`.

JComponent

```
import java.awt.*;
import javax.swing.*;

class One extends JComponent {
  int count;
  public void paintComponent(Graphics g) {
    System.out.println("Called " + ++count + " times.");
    g.drawOval(100, 100, 60, 60);
  }
}
```

When the components is shown for the first time, the paintComponent method is called automatically.

Increment count each time the paintComponent is called.

```
import javax.swing.*;
public class Two extends JFrame {
  Two() {
    this.setVisible(true);
    this.setSize(400, 400);
    this.add(new One());
  }

  public static void main(String[] args) {
    Two t = new Two();
  }
}
```

Key Events

- A *key event* is generated when a keyboard key is pressed. Key events allow a program to respond immediately to the user while he or she is typing or pressing other keyboard keys such as the arrow keys.
- If key events are being processed, the program can respond as soon as the key is pressed; there is no need to wait for the Enter key to be pressed or for some other component (like a button) to be activated.
- The methods of the `KeyListener` interface:
  - `void keyPressed (KeyEvent event)`
- Called when a key is pressed.
  - `void keyReleased (KeyEvent event)`
- Called when a key is released.
  - `void keyTyped (KeyEvent event)`
- Called when a pressed key or key combination produces a key character.
- The listener class must implement all methods defined in the interface. Provide empty methods for the events that you are not using.

Key Events Example

The `Direction` program responds to key events.

- An image of an arrow is displayed, and the image moves across the screen as the arrow keys are pressed.
- Actually, four different images are used, one each for the arrow pointing in each of the primary directions (up, down, right, and left).

Key Events Example

```
//*******************************************************************
// DirectionPanel.java
//
// Represents the primary display panel for the Direction program.
//*******************************************************************

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DirectionPanel extends JPanel
{
    private final int WIDTH = 300, HEIGHT = 200;
    private final int JUMP = 10; // increment for image movement
    private final int IMAGE_SIZE = 31;
    private ImageIcon up, down, right, left, currentImage;
    private int x, y;
```

Key Events Example

```
//-----------------------------------------------------------------
// Constructor: Sets up this panel and loads the images.
//-----------------------------------------------------------------
public DirectionPanel()
{
    addKeyListener (new DirectionListener());

    x = WIDTH / 2;
    y = HEIGHT / 2;

    up = new ImageIcon ("arrowUp.jpg");
    down = new ImageIcon ("arrowDown.jpg");
    left = new ImageIcon ("arrowLeft.jpg");
    right = new ImageIcon ("arrowRight.jpg");

    currentImage = right;

    setBackground (Color.white);
    setPreferredSize (new Dimension(WIDTH, HEIGHT));
    setFocusable(true);
}
```

Key Events Example

```
//-----------------------------------------------------------------
// Draws the image in the current location.
//-----------------------------------------------------------------

public void paintComponent (Graphics page)
{
    super.paintComponent (page);
    currentImage.paintIcon (this, page, x, y);
}
```

Key Events Example

```
// Listener for keyboard activity. Responds to the user pressing arrow keys by
// adjusting theimage and image location accordingly.
private class DirectionListener implements KeyListener {
    public void keyPressed (KeyEvent event) {
        switch (event.getKeyCode()) {
            case KeyEvent.VK_UP:
                currentImage = up;
                y -= JUMP;
            break;
            case KeyEvent.VK_DOWN:
                currentImage = down; y += JUMP;
            break;
            case KeyEvent.VK_LEFT:
                currentImage = left; x -= JUMP;
            break;
            case KeyEvent.VK_RIGHT:
                currentImage = right; x += JUMP;
                break;
        }
        repaint();
}
        // Provide empty definitions for unused event methods.
        public void keyTyped (KeyEvent event) {}
        public void keyReleased (KeyEvent event) {}
    }
}
```

Virtual key codes are used to report which keyboard key has been pressed, rather than a character generated by the combination of one or more keystrokes

Key Events Example

```
//*******************************************************************
// Direction.java
//
// Demonstrates key events.
//*******************************************************************
import javax.swing.JFrame;

public class Direction
{
    //-----------------------------------------------------------------
    // Creates and displays the application frame.
    //-----------------------------------------------------------------
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Direction");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add (new DirectionPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```

Mouse Events

- Java divides events that are generated when using a mouse into two categories:
  - *mouse events*
  - *mouse motion events*

- When you click the mouse button over a Java GUI component, three events are
- generated:
  - one when the mouse button is pushed down (*mouse pressed*)
  - and two when it is let up (*mouse released* and *mouse clicked*).

- A mouse click is defined as pressing and releasing the mouse button in the same location.

- If you press the mouse button down, move the mouse, and then release the mouse button, a mouse clicked event is not generated.

GUI: Mouse Events

Java divides event that are generated when using a mouse into two categories:

- *mouse events*
- *mouse motion events*

mouse pressed	the mouse button is pressed down
mouse released	the mouse button is released
mouse clicked	the mouse button is pressed down and released without moving the mouse in between
mouse entered	the mouse pointer is moved onto (over) a component
mouse exited	the mouse pointer is moved off of a component

mouse moved	the mouse is moved
mouse dragged	the mouse is moved while the mouse button is pressed down

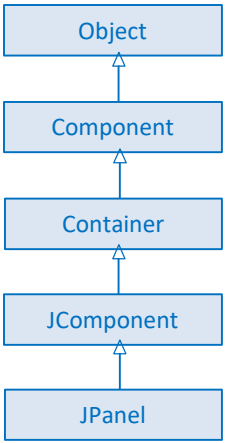
Mouse Events

- Listeners for mouse events are created using the `MouseListener` and `MouseMotionListener` interfaces.
- A `MouseEvent` object is passed to the appropriate method when a mouse event occurs.
- For a given program, we may only care about one or two mouse events.
- To satisfy the implementation of a listener interface, empty methods must be provided for unused events.
- Example:
- The `Dots` program responds to one mouse event.
  - It draws a green dot at the location of the mouse pointer whenever the mouse button is pressed.
  - Unused mouse methods have to be implemented, but the body could be empty.

Mouse Events Example

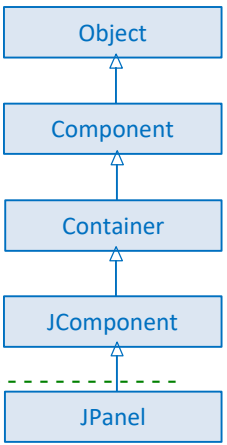
```
//*******************************************************************
//  DotsPanel.java
//
//  Represents the primary panel for the Dots program.
//*******************************************************************

import java.util.ArrayList;
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class DotsPanel extends JPanel
{
   private final int SIZE = 6;  // radius of each dot

   private ArrayList<Point> pointList;
```

Point is a class in package java.awt



Mouse Events Example

```
//-----------------------------------------------------------------
//  Constructor: Sets up this panel to listen for mouse events.
//-----------------------------------------------------------------
public DotsPanel()
{
   pointList = new ArrayList<Point>();

   addMouseListener(new DotsListener());
   setBackground(Color.black);
   setPreferredSize(new Dimension(300, 200));
}

//-----------------------------------------------------------------
//  Draws all of the dots stored in the list.
//-----------------------------------------------------------------
public void paintComponent(Graphics page)
{
   super.paintComponent(page);
   page.setColor(Color.green);

   for (Point spot : pointList)
      page.fillOval(spot.x - SIZE, spot.y - SIZE, SIZE*2, SIZE*2);

   page.drawString("Count: " + pointList.size(), 5, 15);
}
```



addMouseListener

setBackground
setPreferredSize
paintComponent

x coordinate of the upper left corner of the oval

y coordinate of the upper left corner of the oval

Width of the oval

Height of the oval

Mouse Events Example

```
   //*****************************************************************
   //  Listener for mouse events.
   //*****************************************************************
   private class DotsListener implements MouseListener
   {
      //-----------------------------------------------------------------
      //  Adds the current point to the list of points and redraws
      //  the panel whenever the mouse button is pressed.
      //-----------------------------------------------------------------
      public void mousePressed(MouseEvent event)
      {
         pointList.add(event.getPoint());
         repaint();
      }
      //-----------------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //-----------------------------------------------------------------
      public void mouseClicked(MouseEvent event) {}
      public void mouseReleased(MouseEvent event) {}
      public void mouseEntered(MouseEvent event) {}
      public void mouseExited(MouseEvent event) {}
   }
}
```

Method inherited from JComponent.
It calls paintComponent.

Mouse Events Example

```
//*******************************************************************
//  Dots.java
//  Demonstrates mouse events.
//*******************************************************************

import javax.swing.JFrame;

public class Dots
{
   //-----------------------------------------------------------------
   //  Creates and displays the application frame.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Dots");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new DotsPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```