COSC 123
Computer Creativity

Graphics and Events

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Draw shapes, text in various fonts, and colors.

2) Build window applications using `JFrame`/`JPanel` and Swing components.

3) Understand events, event listeners, and event adapters.

4) Write code for handling mouse, keyboard, and window events.

Java Programs Overview

To this point, all our Java programs have received input and displayed output in the console (text window).

Types of Java programs:

- 1) **Console applications** - text-based applications which perform input and output using the console

- 2) **Graphical applications** - stand-alone Java applications which have a graphic user interface with components such as windows, control buttons, menus, and check boxes.

Graphical Applications Overview

A ***graphical application*** is a Java program with a graphical user interface.

A ***frame window*** is a window on the screen that has a border and a title bar.

A frame window is defined in Java using the `JFrame` class that is present in the `javax.swing` package.

- The `javax.swing` package is also called the ***Swing toolkit***.

Creating a Frame Windows

To create a frame window:

- `import javax.swing.JFrame`
- create our own class (like `MyFrame`) which `extends JFrame`
- provide a constructor for our `MyFrame` class
- set the size of our frame using the `setSize` method
  - usually performed in `MyFrame` constructor

To use the `MyFrame` window:

- define a mainline which instantiates a `MyFrame` instance
- use the `setTitle` method to set the frame title (optional)
- use the `setVisible` method to display the frame on the screen

Graphical Applications
Creating a Frame Window

```
import javax.swing.JFrame;
public class MyFrame extends JFrame
{  public static void main(String[] args)
   {  MyFrame frame = new MyFrame();
      frame.setTitle("Frame Title");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}

   public MyFrame()
   {  final int DEFAULT_FRAME_WIDTH = 300;
      final int DEFAULT_FRAME_HEIGHT = 300;
      setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
   }
}
```

A "Hello World!" Window

The window displays `Hello World!`

```
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class DrawHello extends JPanel
{  public static void main(String[] args)
   {  JFrame frame = new JFrame();
      frame.setTitle("Hello World!");
      frame.setSize(300, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
      frame.getContentPane().add(new DrawHello());
   }

    public void paint(Graphics g) {
        g.drawString("Hello World!", 50, 100);
    }
}
```

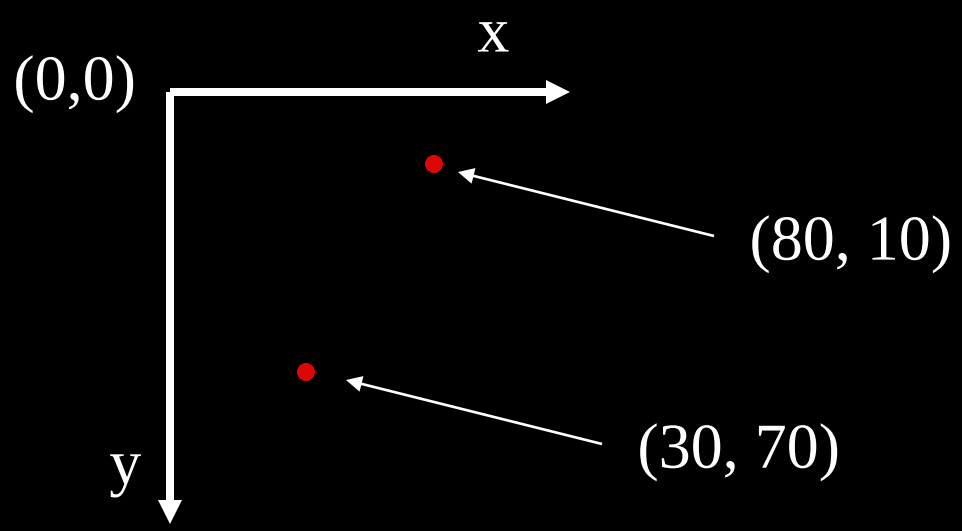
Question: What does `extends` mean?

The Coordinate System

Drawing on the screen is done by specifying coordinates which refer to a location on the screen.

- The ***origin*** is the upper-left hand corner of the screen.
- The x coordinate gets bigger as we move to the right.
- The y coordinate gets bigger as we move down.

Diagram:



drawString Method

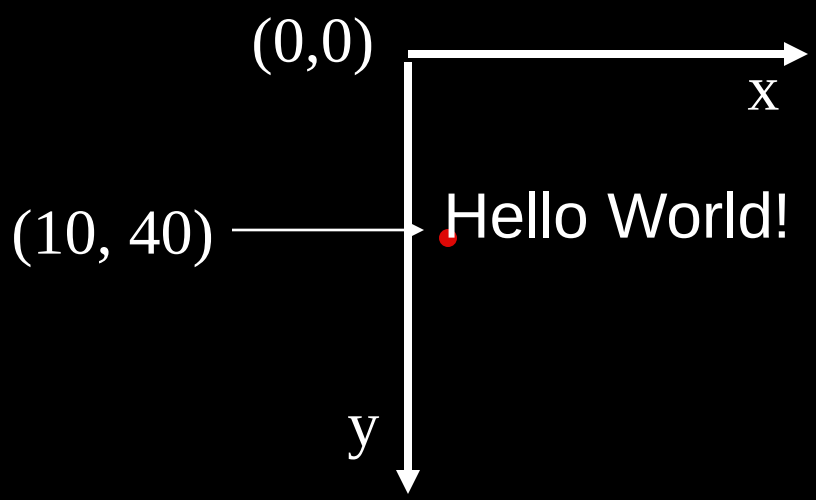
The `drawString` method draws a text string on the screen.

Usage:

- `g.drawString(`*`message, x, y`*`)`
  - ⇨ x, y co-ordinates are the base point of the message

Example:

- `g.drawString("Hello World!", 10, 40);`



Drawing

To draw shapes on the screen, we use the `draw` method.

The `draw` method takes a shape that we create and draws it on the screen.

Example:

- `Rectangle box = new Rectangle(10, 10, 20, 30);`
- `g2.draw(box);`

Drawing Shapes

There are several methods to draw shapes:

- 1) Ellipse:
  - Ellipse2D.Double egg = new Ellipse2D.Double(topx, topy, width, height);
  - Ellipse2D.Double egg = new Ellipse2D.Double(5, 10, 15, 20);
- 2) Rectangle:
  - Rectangle box = new Rectangle(topx, topy, width, height);
  - Rectangle box = new Rectangle(10, 10, 20, 30);
- 3) Line:
  - Line2D.Double = new Line2D.Double(x1, y1, x2, y2);
- 4) Point:
  - Point2D.Double = new Point2D.Double(x,y);

You can also fill a shape with a color using the `fill` method:

- g2.fill(box);
- g2.fill(egg);

Changing Colors

There are 3 basic display colors which are combined to form all colors displayed on a computer.

- Red, green, and blue are used in the RGB color model.
- Any color can be defined by specifying what percentage of red, blue, and green is in the color.

The class for colors in Java is called `Color`.

- `import java.awt.Color;`
- `Color orange = new Color(1.0F, 0.8F, 0.0F);`

Changing what color your text or shapes is drawn in:

- `g.setColor(orange);`

There are also static colors predefined in Java:

- `Color.black`, `Color.red`, `Color.white`, `Color.orange`, etc.

Changing Fonts

The `drawString` method uses a default font if none is given.

A font consists of:

- a font face name (Serif, SansSerif, Monospaced, Dialog, etc.)
- a style (`Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`, etc.)
- a font size (specified in points: 1 inch = 72 points)

The font class in Java is called `Font`:

- `import java.awt.Font;`
- `Font bigFont = new Font("Serif", Font.BOLD, 36);`

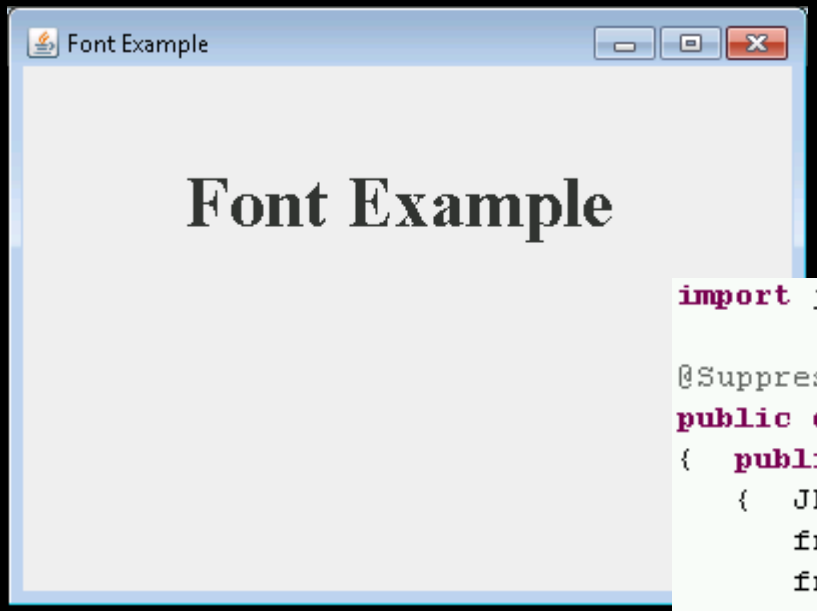
Set the current font:

- `g.setFont(bigFont);`

Then use drawString:

- `g.drawString("Hello World!", 50, 100);`

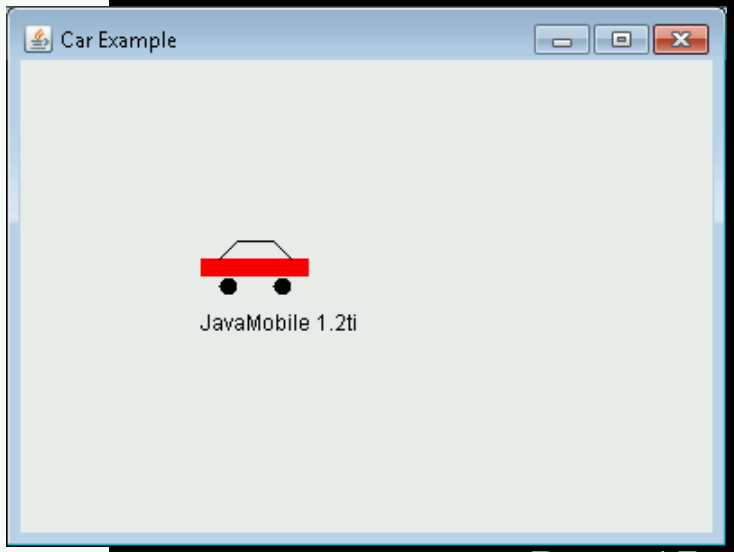
Drawing Fonts




```
import java.awt.Font;

@SuppressWarnings("serial")
public class DrawFont extends JPanel
{  public static void main(String[] args)
   {  JFrame frame = new JFrame();
      frame.setTitle("Font Example");
      frame.setSize(400, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
      frame.getContentPane().add(new DrawFont());
   }

    public void paint(Graphics g) {
        Font f = new Font("Serif", Font.BOLD, 36);
        g.setFont(f);
        g.drawString("Font Example", 80, 80);
    }
}
```

Car Drawing Example

```
public class DrawCar extends JPanel
{  public static void main(String[] args)
   {  JFrame frame = new JFrame();
      frame.setTitle("Car Example");
      frame.setSize(400, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
      frame.getContentPane().add(new DrawCar());
   }

    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        Rectangle body = new Rectangle(100, 110, 60, 10);
        Ellipse2D.Double frontTire = new Ellipse2D.Double(110, 120, 10, 10);
        Ellipse2D.Double rearTire = new Ellipse2D.Double(140, 120, 10, 10);
        Point2D.Double r1 = new Point2D.Double(110, 110);
        Point2D.Double r2 = new Point2D.Double(120, 100);
        Point2D.Double r3 = new Point2D.Double(140, 100);
        Point2D.Double r4 = new Point2D.Double(150, 110);
        Line2D.Double fWind = new Line2D.Double(r1, r2);
        Line2D.Double roofTop = new Line2D.Double(r2, r3);
        Line2D.Double rWind = new Line2D.Double(r3, r4);
        g2.setColor(Color.red);
        g2.fill(body);
        g2.setColor(Color.black);
        g2.fill(frontTire);
        g2.fill(rearTire);
        g2.draw(fWind);
        g2.draw(roofTop);
        g2.draw(rWind);
        g2.drawString("JavaMobile 1.2ti", 100, 150);
    }
}
```



Exercises

1) Draw three circles of different colors.

2) Draw a "better" looking car.

Adding Components to a Frame

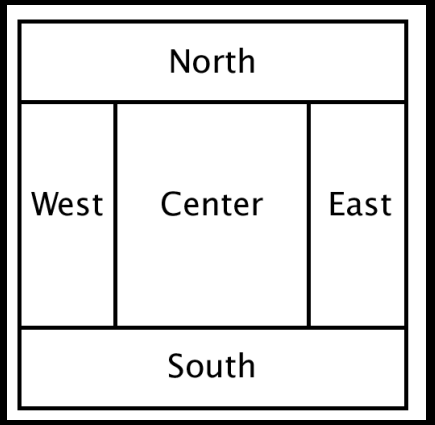
Content is added to the frame on the content pane.

One common component to add is a `JPanel` that allows you to draw graphics.

- A ***container*** is a component that can hold other components.

There are five regions of a `JFrame` where you can place components:

- North, West, Center, East, South



Example:

```
Container contentPane = getContentPane();
MyPanel panel = new MyPanel();
contentPane.add(panel, "Center");
```

Java Swing Components

The Java Swing package contains the user interface components that we will use in our graphical applications.

Component	Import Package
JButton	javax.swing.JButton
ButtonGroup	javax.swing.ButtonGroup
Check box	javax.swing.JCheckBox
Combo box	javax.swing.JComboBox
JFrame	javax.swing.JFrame
JLabel	javax.swing.JLabel
JPanel	javax.swing.JPanel
Radio button	javax.swing.JRadioButton
Text field	javax.swing.JTextField

GUI Example


```
// Setup layout of main panel
mainPanel = new JPanel();
mainPanel.setLayout(new GridLayout(7,1));

// First and last name text fields
JPanel tmpPanel = new JPanel();
JLabel tmpLabel = new JLabel("First name: ");
tmpPanel.add(tmpLabel);
txtFname = new JTextField(20);
tmpPanel.add(txtFname);
mainPanel.add(tmpPanel);
tmpPanel = new JPanel();
tmpLabel = new JLabel("Last name: ");
tmpPanel.add(tmpLabel);
txtLname = new JTextField(20);
tmpPanel.add(txtLname);
mainPanel.add(tmpPanel);

// Male and female radio buttons
tmpPanel = new JPanel();
rbMale = new JRadioButton("Male");
rbFemale = new JRadioButton("Female");
rbGroupSex = new ButtonGroup();
rbGroupSex.add(rbMale);
rbGroupSex.add(rbFemale);
tmpPanel.add(rbMale);
tmpPanel.add(rbFemale);
mainPanel.add(tmpPanel);
```

GUI Example (2)


```
// Checkboxes
tmpPanel = new JPanel();
cbxBike = new JCheckBox("I have a bike");
cbxCar = new JCheckBox("I have a car");
tmpPanel.add(cbxBike);
tmpPanel.add(cbxCar);
mainPanel.add(tmpPanel);

// Color list box
tmpPanel = new JPanel();
tmpLabel = new JLabel("My favorite color is: ");
tmpPanel.add(tmpLabel);
lbxColor = new JComboBox();
lbxColor.addItem("red");
lbxColor.addItem("blue");
lbxColor.addItem("yellow");
lbxColor.addItem("green");
tmpPanel.add(lbxColor);
mainPanel.add(tmpPanel);

// Hobbies text area
tmpPanel = new JPanel();
tmpLabel = new JLabel("My hobbies are: ");
taHobbies = new JTextArea(3,20);
tmpPanel.add(tmpLabel);
tmpPanel.add(taHobbies);
mainPanel.add(tmpPanel);
```

GUI Example (3)


```
// Action buttons
tmpPanel = new JPanel();
btnHere = new JButton("CLICK HERE!");
btnReset = new JButton("Reset");
btnSubmit = new JButton("Submit");
tmpPanel.add(btnHere);
tmpPanel.add(btnReset);
tmpPanel.add(btnSubmit);
mainPanel.add(tmpPanel);

contentPane.add(mainPanel, "West");
```

GUI Components
JLabel

The JLabel class is used to display a label (or text) on the screen that cannot be edited by the user.

```
JLabel myLabel = new JLabel("My Label",
                            SwingConstants.RIGHT);
```

A label can be aligned by using:

- Center - SwingConstants.CENTER
- Right - SwingConstants.RIGHT
- Left - SwingConstants.LEFT

GUI Components
JTextField and JTextArea

JTextField allows us to read in a single line of text.
JTextArea allows us to handle multiple lines of text.

With a JTextField, you may give the # of characters:

```
JTextField txtField = new JTextField(5); // 5 chars.
```

With a JTextArea, you can give the # of rows/cols:

```
JTextArea txtArea = new JTextArea(5,40);//5 rows, 40 cols
```

GUI Components
JTextField and JTextArea Methods

Some useful methods for text fields:

```
JTextField txtField = new JTextField();

txtField.setText("Hello World!"); // Set the field text
txtField.setEditable(false);  // Do not allow field edits
txtField.setFont(hugeFont);   // Change the field font
txtField.getText();           // Get current field text
```

GUI Components
JRadioButton Overview

The JRadioButton class allows the user to select from disjoint inputs (i.e. the user can select only one out of a list).

```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");

ButtonGroup sizeGroup = new ButtonGroup();
sizeGroup.add(smallButton);
sizeGroup.add(mediumButton);
sizeGroup.add(largeButton);
```

The ButtonGroup class allows the programmer to specify which buttons are grouped with each other.

You can select buttons or determine if buttons are selected by:

```
smallButton.setSelected(true);
if (smallButton.isSelected()) return "Small";
```

GUI Components
JCheckBox Overview

The JCheckBox class allows the user to select yes/no valued inputs (i.e. true or false).

```
JCheckBox boldCheckBox = new JCheckBox("Bold");
```

- Note: Do not place check boxes inside a button group because they are not mutually exclusive.

GUI Components
JComboBox Overview

The JComboBox class allows the user to select from a large list of disjoint inputs where radio buttons are too awkward.

- A JComboBox allows you to select an item from the list.
- If the list is editable, you can type in your own selection that may not already be in the list.

```
JComboBox itemCombo = new JComboBox();
itemCombo.addItem("Item 1");
itemCombo.addItem("Item 2");
```

You can get the selected item in the list by:

```
String st = (String) itemCombo.getSelectedItem();
```

- Note that JComboBox, JCheckBox, and JRadioButton all generate action events that should be detected using an action listener.

GUI Components
JButton Overview

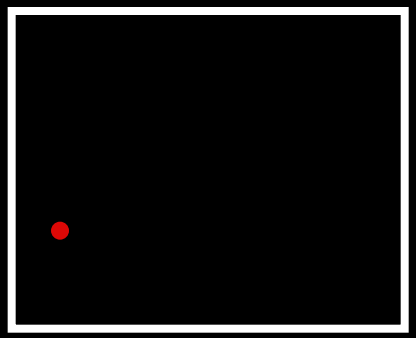
The JButton class allows you to put a button on your frame.

When creating a button, it can have just text, just a picture, or a picture and text.

```
leftButton = new JButton("left");
leftButton = new JButton(new ImageIcon("left.gif"));
leftButton = new JButton("left",newImageIcon("left.gif"));
```

Coordinates

Question: Select from the coordinates below the pair that best describes this point's location. Assume box is 100 by 100.



A) (10,80)

B) (80,10)

C) (10,20)

D) (20,10)

Components

Question: What is the best component to use if the user can select yes/no to multiple items independently?

A) `JRadioButton`

B) `JComboBox`

C) `JCheckBox`

D) `JButton`

Components

Question: What is the best component to use if the user must pick only one item from 50 possible choices?

A) `JRadioButton`

B) `JComboBox`

C) `JCheckBox`

D) `JButton`

Events and Event Handling
GUI Programming Philosophy

In ***graphical applications***, the programmer must ***react*** instead of ***dictate*** the events that occur in a program.

As a programmer, you design a graphical user interface with windows, buttons, and components that the user can interact with. You do not know the order or the sequence of events the user will generate, but you must be able to react to them.

Events and Event Handling Overview

An ***event*** is a notification to your program that something has occurred.

- For graphical events (mouse click, data entry), the Java window manager notifies your program that an event occurred.
  - There are different ***kinds*** of events such as keyboard events, mouse click events, mouse movement events, etc.

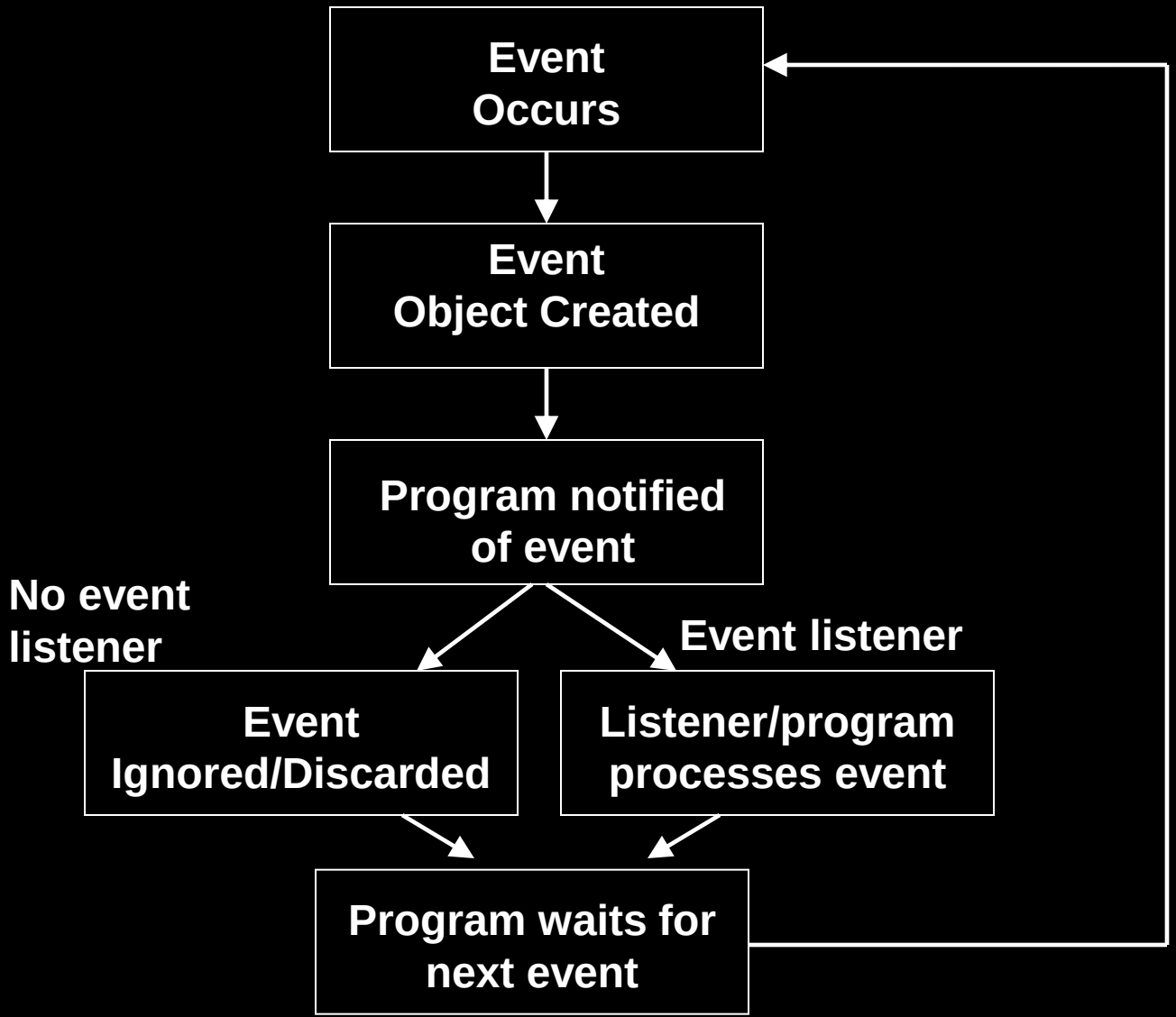
An ***event handler*** or ***listener*** is part of your program that is responsible for "listening" for event notifications and handling them properly.

- An event listener often only listens for certain types of events.

An ***event source*** is the user interface component that generated the event.

- A button, a window, and scrollbars are all event sources.

Event Handling Overview



Mouse Event Example

Handling mouse click events requires three classes:

- 1) The **event class** - that stores information about the event.
  - For mouse clicks, this class is `MouseEvent`.
  - The `MouseEvent` class has methods `getX()` and `getY()` that indicate the position of the mouse at the time the event was generated.
  - Each event class has the method `Object getSource()` that returns the source of the event.
- 2) The **listener class** - allows your program to detect events. Building your own listener class requires implementing a pre-defined interface.
  - For mouse clicks, the interface is `MouseListener`. `MouseAdapter` is a class that implements the `MouseListener` interface.
- 3) The **event source** - is the component in your GUI that generated the event.

MouseListener Interface

The `MouseListener` interface must be implemented by your class that handles mouse events. It has the methods:

```
public interface MouseListener
{   void mouseClicked(MouseEvent event);
    // Called when the mouse has been clicked on component
    void mouseEntered(MouseEvent event);
    // Called when the mouse enters a component
    void mouseExited(MouseEvent event);
    // Called when the mouse exits a component
    void mousePressed(MouseEvent event);
    // Called when a mouse button pressed on a component
    void mouseReleased(MouseEvent event);
    // Called when mouse button released on a component
}
```

To add a listener, use the method:

- `addMouseListener(`*`listener_name`*`);`

Mouse Event Example Code

```
public class MouseSpy extends JPanel {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Mouse Spy!");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new MouseSpy());
        MouseSpyListener listener = new MouseSpyListener();
        frame.addMouseListener(listener);
    }

    public void paint(Graphics g) {
        g.drawString("Mouse spy!", 50, 100);
    }
}

class MouseSpyListener implements MouseListener
{   public void mouseClicked(MouseEvent event)
    {  System.out.println("Mouse clicked. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseEntered(MouseEvent event)
    {  System.out.println("Mouse entered. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseExited(MouseEvent event)
    {  System.out.println("Mouse exited. x = " + event.getX() + " y = " + event.getY());}
    public void mousePressed(MouseEvent event)
    {  System.out.println("Mouse pressed. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseReleased(MouseEvent event)
    {  System.out.println("Mouse released. x = " + event.getX() + " y = " + event.getY());
    }
}
```

Event Listeners and Inner Classes

Typically, your event listener class will perform some function based on the user input.

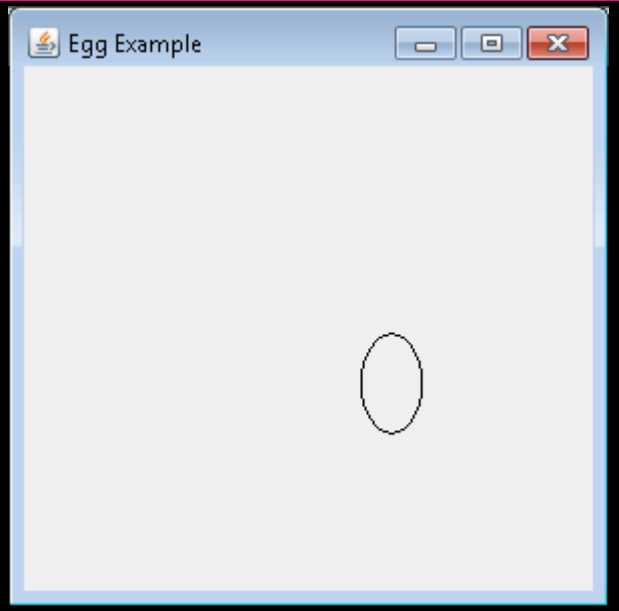
- This function often involves accessing the private variables of the `Frame` class you defined.
- However, if the listener class is implemented outside of the `Frame` class, it has no more access rights to the private instance variables of that class then any other class.
- The solution to this problem is to use inner classes.

An ***inner class*** is a class that is defined inside another class.

- The methods of the inner class have access to the private variables of the outer class.
- The inner class is typically defined as `private`.
- An inner class object remembers the object that created it.

Egg Draw Example Code

```
public class EggExample extends JPanel {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Egg Example");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new EggExample());
    }

    private Ellipse2D.Double egg;
    private static final double EGG_WIDTH = 30;
    private static final double EGG_HEIGHT = 50;

    public EggExample()
    {   egg = new Ellipse2D.Double(0, 0, EGG_WIDTH, EGG_HEIGHT);
        // add mouse click listener
        MouseClickListener listener = new MouseClickListener();
        addMouseListener(listener);
    }

    public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;
        g.clearRect(0, 0, 300, 300);        // Clear the window
        g2.draw(egg);
    }

    // inner class definition
    private class MouseClickListener extends MouseAdapter
    {   public void mouseClicked(MouseEvent event)
        {   int mouseX = event.getX();
            int mouseY = event.getY();
            egg.setFrame(mouseX - EGG_WIDTH / 2,
                    mouseY - EGG_HEIGHT / 2, EGG_WIDTH, EGG_HEIGHT);
            repaint();
        }
    }
}
```



Draws an ellipse where the user clicks.

- The mouse listener is an inner class.
- Every time the user clicks the mouse, the listener repositions the ellipse and calls `repaint` to redraw.

WindowListener Interface

The `WindowListener` interface must be implemented by your frame class to handle its events. It has the methods:

```
public interface WindowListener
{   void windowOpened(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowActivated(WindowEvent e);
    void windowDeactivated(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowDeiconfied(WindowEvent e);
    void windowClosing(WindowEvent e);
}
```

Most programs only care about the window closed event.

- That is where `System.exit(0)` is typically placed.
- There is also a `WindowAdapter` class that can be extended instead of implementing all interface methods.

Frame Window Event Example

```
public class FrameTest extends JFrame
{   public static void main(String[] args)
    {   FrameTest frame = new FrameTest();
        frame.setTitle("Close me!");
        frame.setVisible(true);
    }

    public FrameTest()
    {   final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;
        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);

        WindowCloser listener = new WindowCloser();
        addWindowListener(listener);
    }

    private class WindowCloser extends WindowAdapter
    {
        public void windowClosing(WindowEvent event)
        {
            System.exit(0);
        }
    }
}
```

Note the use of `WindowAdapter` as we only care about the window closing event.

Action Listeners

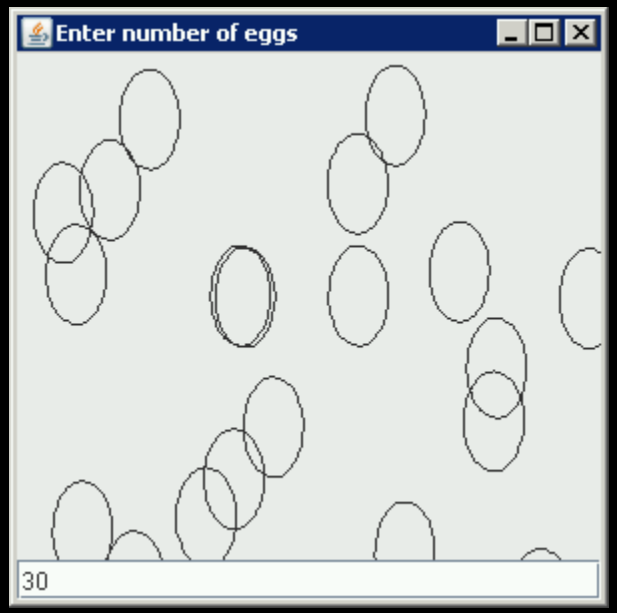
GUI components like buttons, text fields, combo boxes, and check boxes all generate action events.

The **ActionListener** interface has a single method:

```
public interface ActionListener
{  public void actionPerformed(ActionEvent event);
}
```

An action event is generated when you click on the control or press Enter for text fields.

Eggs.Java Example



In this example, we will create a `JFrame` with a `JPanel` and a `JTextField` and ask the user for the number of ellipses ("eggs") to draw on the screen.

Notes:

1) The `JPanel` component paints itself in the `paintComponent` method. This method MUST call `super.paintComponent` as the first line.

- Note that this is different than the `paint` method in applets.

2) When the user changes the value in the text field, you must call `repaint` to get the `JPanel` to repaint itself.

Eggs.Java Example Code

```
public class Eggs extends JFrame
{
    private JTextField textField;
    private EggPanel panel;

    public static void main(String[] args)
    {
        Eggs frame = new Eggs();
        frame.setTitle("Enter number of eggs");
        frame.setVisible(true);
    }

    public Eggs()
    {
        final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;

        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
        addWindowListener(new WindowCloser());
        // construct components
        panel = new EggPanel();
        textField = new JTextField();
        textField.addActionListener(new TextFieldListener());
        // add components to content pane
        Container contentPane = getContentPane();
        contentPane.add(panel, "Center");
        contentPane.add(textField, "South");
    }
```

Eggs.Java Example Code (2)

```
private class TextFieldListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {   // get user input
        String input = textField.getText();
        // process user input
        panel.setEggCount(Integer.parseInt(input));
        // clear text field
        textField.setText("");
    }
}

private class WindowCloser extends WindowAdapter
{
    public void windowClosing(WindowEvent event)
    {
        System.exit(0);
    }
}
```

Eggs.Java Example Code (3)

```
    private class EggPanel extends JPanel
    {
        private int eggCount;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HEIGHT = 50;

        public void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;
            // draw eggCount ellipses with random centers
            Random generator = new Random();
            for (int i = 0; i < eggCount; i++)
            {
                double x = getWidth() * generator.nextDouble();
                double y = getHeight() * generator.nextDouble();
                Ellipse2D.Double egg = new Ellipse2D.Double(x, y,
                                                EGG_WIDTH, EGG_HEIGHT);
                g2.draw(egg);
            }
        }

        // Sets the number of eggs to be drawn and repaints
        public void setEggCount(int count)
        {   eggCount = count;
            repaint();
        }
    }
}
```

JButton and ActionListener

When a button is clicked, it sends an action event that must be captured using an action listener.

```
leftButton = new JButton("left");
ActionListener listener = new ButtonListener();
leftButton.addActionListener(listener);
```

You may either create one action listener for all buttons (which uses the `event.getSource` method to determine the button pressed) or create a separate listener for each button.

One Button Action Listener

```
public class MyFrame
{  public MyFrame()
   {  ...
      upButton = new JButton("Up");
      ActionListener listener = new UpListener();
      upButton.addActionListener(listener);
      ...
   }
   ...
   private JButton upButton;
   ...
   private class UpListener implements ActionListener
   {  public void actionPerformed(ActionEvent event)
      {  // performs action when up button is clicked
      }
   }
}
```

Multiple Button Action Listener

```
public class MyFrame
{  public MyFrame()
   {  ...
      upButton = new JButton("Up");
      downButton = new JButton("Down");
      leftButton = new JButton("Left");
      rightButton = new JButton("Right");
      ActionListener listener = new DirectionListener();
      upButton.addActionListener(listener);
      downButton.addActionListener(listener);
      leftButton.addActionListener(listener);
      rightButton.addActionListener(listener);
      // create a Panel containing all buttons and add
      // to content pane
   }
   private JButton upButton, downButton, leftButton;
   private JButton rightButton;
```

Multiple Button Action Listener (2)

```
...
private class DirectionListener implements
                                    ActionListener
{  public void actionPerformed(ActionEvent event)
   {  // performs action when any button is clicked
      Object source = event.getSource();
      if (source == upButton)
      // Perform up action
      else if (source == downButton)
      // Perform down action
      else if (source == leftButton)
      // Perform left action
      else if (source == rightButton)
      // Perform right action
   }
}
```

Listeners and Adapters

Question: Which one is a true statement?

A) To handle mouse events, create a class that `extends MouseListener`.

B) To handle mouse events, create a class that `implements MouseAdapter`.

C) To handle mouse events, create a class that `extends MouseAdapter`.

D) You must implement all event methods when your class extends `MouseAdapter`.

Practice Questions

1) Create a program that displays the string "Hello world" at the location where the user clicks the mouse.

- Notes:
  - When the user clicks a mouse, move the location of "Hello World!".
  - Use `MouseAdapter` and inner classes.

2) Create a program that opens up a window with "1" as the title. Then,

- If the user clicks on the window, a new window is opened with value of "2".
- If the user clicks on either open window, a new window is opened with value of "3". This may repeat for any # of windows.
- When a window is closed, all other windows stay open.
- When the last window is closed, the program quits.

Menus Overview

Menus allow the user to select options without using buttons and fields.

- A menu is located at the top of the frame in a menu bar.

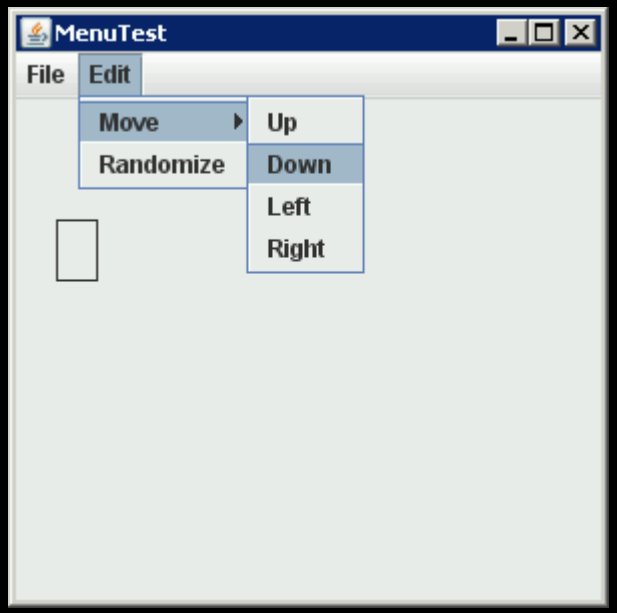
A ***menu*** is a collection of menu items and more menus.

- You add menu items and submenus with the add method.

When a menu item is selected, it generates an action event.

- Thus, each menu item should have a listener defined.

Menus Example



In this example, we will create a menu that allows us to move a rectangle around the window based on user selections.

Menus Example Code

```
public class MenuTest extends JFrame
{
    private JMenuItem exitMenuItem;
    private JMenuItem newMenuItem;
    private JMenuItem upMenuItem;
    private JMenuItem downMenuItem;
    private JMenuItem leftMenuItem;
    private JMenuItem rightMenuItem;
    private JMenuItem randomizeMenuItem;
    private RectanglePanel panel;

    public static void main(String[] args)
    {   MenuTest frame = new MenuTest();
        frame.setTitle("MenuTest");
        frame.setVisible(true);
    }

    public MenuTest()
    {   final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;
        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
        addWindowListener(new WindowCloser());

        // add drawing panel to content pane
        panel = new RectanglePanel();
        Container contentPane = getContentPane();
        contentPane.add(panel, "Center");
```

This is the basic setup for creating a frame.

Note the `JMenuItem` instance variables, one for each menu item.

Menus Example Code (2)

```
// construct menu
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
JMenu fileMenu = new JMenu("File");
menuBar.add(fileMenu);
MenuListener listener = new MenuListener();
newMenuItem = new JMenuItem("New");
fileMenu.add(newMenuItem);
newMenuItem.addActionListener(listener);
exitMenuItem = new JMenuItem("Exit");
fileMenu.add(exitMenuItem);
exitMenuItem.addActionListener(listener);

JMenu editMenu = new JMenu("Edit");
menuBar.add(editMenu);
JMenuItem moveMenu = new JMenu("Move");
editMenu.add(moveMenu);
upMenuItem = new JMenuItem("Up");
moveMenu.add(upMenuItem);
upMenuItem.addActionListener(listener);
downMenuItem = new JMenuItem("Down");
moveMenu.add(downMenuItem);
downMenuItem.addActionListener(listener);
leftMenuItem = new JMenuItem("Left");
moveMenu.add(leftMenuItem);
leftMenuItem.addActionListener(listener);
rightMenuItem = new JMenuItem("Right");
moveMenu.add(rightMenuItem);
rightMenuItem.addActionListener(listener);
```

Still in the constructor, this code begins by creating a `JMenuBar` and setting it as the frame's menu bar.

Then, the file menu is created with two items: new and exit.

Note that an `ActionListener` is added for each menu item, and it is the same listener object.

Later we will see how the listener determines what menu item was selected.

The next code creates the edit menu. Note that the move menu is a submenu of the edit menu.

Menus Example Code (3)

```
        randomizeMenuItem = new JMenuItem("Randomize");
        editMenu.add(randomizeMenuItem);
        randomizeMenuItem.addActionListener(listener);
    }

    private class MenuListener implements ActionListener
    {   public void actionPerformed(ActionEvent event)
        {  // find the menu that was selected
            Object source = event.getSource();
            if (source == exitMenuItem)
                System.exit(0);
            else if (source == newMenuItem)
                panel.reset();
            else if (source == upMenuItem)
                panel.moveRectangle(0, -1);
            else if (source == downMenuItem)
                panel.moveRectangle(0, 1);
            else if (source == leftMenuItem)
                panel.moveRectangle(-1, 0);
            else if (source == rightMenuItem)
                panel.moveRectangle(1, 0);
            else if (source == randomizeMenuItem)
                panel.randomize();
        }
    }
```

The top of the code finishes the edit menu by adding the randomize menu item.

The `MenuListener` is the class that is used to respond to menu action events.

Note that the `getSource` method is used to determine the menu item selected which is then compared with all the menu items created.

Once the appropriate menu item is found, the correct method is called to perform the menu action.

Menus Example Code (4)

```
private class WindowCloser extends WindowAdapter
{   public void windowClosing(WindowEvent event)
    {   System.exit(0);
    }
}

private class RectanglePanel extends JPanel
{
    private Rectangle rect;
    private static final int RECT_WIDTH = 20;
    private static final int RECT_HEIGHT = 30;

    public RectanglePanel()
    {   rect = new Rectangle(0, 0, RECT_WIDTH,RECT_HEIGHT);
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.draw(rect);
    }

    public void reset()
    {   rect.setLocation(0, 0);
        repaint();
    }
```

A standard class extending `WindowAdapter` is used to detect when the window is closed and to terminate the application.

`RectanglePanel` is the panel where the rectangle is drawn.

The rectangle has a fixed size and starts off at (0,0).

The `reset` method is called when the new menu item is selected. It places the rectangle back at (0,0) and calls `repaint` to make sure the panel is redrawn to reflect the changes.

Menus Example Code (5)

```
        public void randomize()
        {   Random generator = new Random();
            rect.setLocation(generator.nextInt(getWidth()),
            generator.nextInt(getHeight()));
            repaint();
        }

        public void moveRectangle(int dx, int dy)
        {   rect.translate(dx * RECT_WIDTH, dy * RECT_HEIGHT);
            repaint();
        }
    }
}
```

The `randomize` method places the rectangle at a random location in the window and redraws the panel.

The `moveRectangle` method moves the rectangle an amount left/right (`dx`) or up/down (`dy`) from its current location.

Exercise

Create an application that has a File menu and an edit menu.

- The file menu should have an exit item that closes the application.
- The edit menu should have two subitems:
  - shape – has submenu of rectangle, square, and circle
  - color – has submenu of red, green, blue, yellow
- When the use selects a shape and color, remember the shape and color. Default is rectangle and red.
- When the user clicks on a place on the screen, draw that shape in that color.

Timer

A ***timer*** can be used to create events at set times. A timer generates `ActionEvents`.

Creating a timer:

```
Timer timer = new Timer(1000, listener);
   // The timer fires every 1000 ms (1 second).
   // The listener class is called every time.
```

Starting and stopping a timer:

```
timer.start();
timer.stop();
```

Timer Example Code

```
public class RandomArtPanel extends JPanel {

   /**
    * A RepaintAction object calls the repaint method of this panel each
    * time its actionPerformed() method is called.  An object of this
    * type is used as an action listener for a Timer that generates an
    * ActionEvent every four seconds.  The result is that the panel is
    * redrawn every four seconds.
    */
   private class RepaintAction implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
         repaint();  // Call the repaint() method in the panel class.
      }
   }

   /**
    * The constructor creates a timer with a delay time of four seconds
    * (4000 milliseconds), and with a RepaintAction object as its
    * ActionListener.  It also starts the timer running.
    */
   public RandomArtPanel() {
      RepaintAction action = new RepaintAction();
      Timer timer = new Timer(4000, action);
      timer.start();
   }
```

This draws a random picture every 4 seconds.

This is the listener for the timer which just calls `repaint`.

Note the creation and starting of the timer.

Keyboard Events

A ***keyboard event*** occurs when a keyboard key is pressed.

Key events allow a program to respond immediately as the user presses keys.

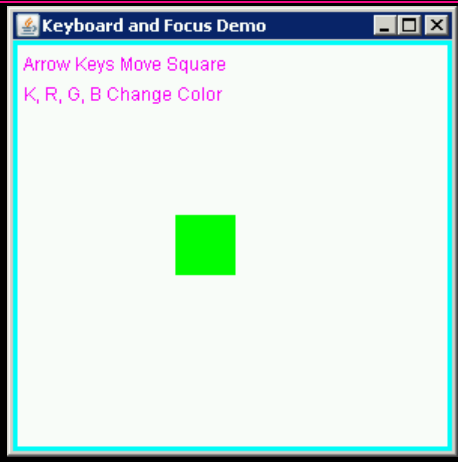
A listener responds when any key is pressed, then decides what to do based on the specific key pressed.

Keyboard events:

```
public void keyPressed(KeyEvent evt);
public void keyReleased(KeyEvent evt);
public void keyTyped(KeyEvent evt);
```

Keyboard Example Code

```
public class KeyboardAndFocusDemo extends JApplet {
   public static void main(String[] args) {
      JFrame window = new JFrame("Keyboard and Focus Demo");
      window.setContentPane( new ContentPanel() );
      window.setSize(300,300);
      window.setLocation(100,100);
      window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      window.setVisible(true);
   }

   /**
    * The init() method of the applet just sets the content pane
    * of the applet to be a panel of type ContentPane, a nested class
    * that is defined in this class and which does all the work.
    */
   public void init() {
      setContentPane( new ContentPanel() );
   }


   public static class ContentPanel extends JPanel
                implements KeyListener, FocusListener, MouseListener {

      private static final int SQUARE_SIZE = 40;
      private Color squareColor;
      private int squareTop, squareLeft;
```



This allows the user to move a square and change its color by pressing keys.

Note that the panel is setup to listen for keyboard, mouse, and focus events.

Keyboard Example Code (2)

```
public ContentPanel() {
   squareTop = 100;  // Initial position of top-left corner of square.
   squareLeft = 100;
   squareColor = Color.RED;  // Initial color of square.

   setBackground(Color.WHITE);
   addKeyListener(this);     // Set up event listening.
   addFocusListener(this);
   addMouseListener(this);
} // end init();

public void paintComponent(Graphics g) {
   super.paintComponent(g);  // Fills the panel with white

   /* Draw a 3-pixel border, colored cyan if the applet has the
      keyboard focus, or in light gray if it does not. */
   if (hasFocus())
      g.setColor(Color.CYAN);
   else
      g.setColor(Color.LIGHT_GRAY);

   int width = getSize().width;  // Width of the applet.
   int height = getSize().height; // Height of the applet.
   g.drawRect(0,0,width-1,height-1);
   g.drawRect(1,1,width-3,height-3);
   g.drawRect(2,2,width-5,height-5);
```

The constructor for the panel adds listeners for the events.

The paintComponent method draws the panel. It also draws the rectangle.

Keyboard Example Code (3)

```
        /* Draw the square. */
        g.setColor(squareColor);
        g.fillRect(squareLeft, squareTop, SQUARE_SIZE, SQUARE_SIZE);

        /* Print a message depending if the panel has the focus. */
        g.setColor(Color.magenta);
        if (hasFocus()) {
           g.drawString("Arrow Keys Move Square",7,20);
           g.drawString("K, R, G, B Change Color",7,40);
        }
        else
           g.drawString("Click to activate",7,20);
    }  // end paintComponent()


    // This will be called when the panel gains the input focus.
    public void focusGained(FocusEvent evt) {
        repaint();  // redraw with cyan border
    }

    // This will be called when the panel loses the input focus.
    public void focusLost(FocusEvent evt) {
        repaint();  // redraw without cyan border
    }
```

If the panel gains or loses focus, repaint is called to update the graphics on the panel.

Keyboard Example Code (4)

```
// This method is called when the user types a key.
public void keyTyped(KeyEvent evt) {
   char ch = evt.getKeyChar();  // The character typed.

   if (ch == 'B' || ch == 'b') {
      squareColor = Color.BLUE;
      repaint();   // Redraw panel with new color.
   }
   else if (ch == 'G' || ch == 'g') {
      squareColor = Color.GREEN;
      repaint();
   }
   else if (ch == 'R' || ch == 'r') {
      squareColor = Color.RED;
      repaint();
   }
   else if (ch == 'K' || ch == 'k') {
      squareColor = Color.BLACK;
      repaint();
   }
}  // end keyTyped()
```

The keyTyped method detects when a user types a key and changes the color of the square accordingly.

Keyboard Example Code (5)

```
public void keyPressed(KeyEvent evt) {
   int key = evt.getKeyCode();  // keyboard code for the pressed key

   if (key == KeyEvent.VK_LEFT) {
      squareLeft -= 8;
      if (squareLeft < 3)
         squareLeft = 3;
      repaint();
   }
   else if (key == KeyEvent.VK_RIGHT) {
      squareLeft += 8;
      if (squareLeft > getWidth() - 3 - SQUARE_SIZE)
         squareLeft = getWidth() - 3 - SQUARE_SIZE;
      repaint();
   }
   else if (key == KeyEvent.VK_UP) {
      squareTop -= 8;
      if (squareTop < 3)
         squareTop = 3;
      repaint();
   }
   else if (key == KeyEvent.VK_DOWN) {
      squareTop += 8;
      if (squareTop > getHeight() - 3 - SQUARE_SIZE)
         squareTop = getHeight() - 3 - SQUARE_SIZE;
      repaint();
   }
}  // end keyPressed()
```

The keyPressed method detects when a key is pressed and moves the square.

Graphical User Interfaces Conclusion

Buttons, text fields, check boxes, combo boxes, and menus are all components in the Java Swing package that can be used to developed a GUI for your application.

Components generate events (usually action events) to indicate when they have been clicked on or accessed by the user.

- We handle the events using listeners and adapters.

The important thing about Swing is not memorizing the components and their methods, but understanding how the components work and generate events.

- Focus on event handling and the concept of using components, not on the definition of the components!

Objectives

Definitions: event, event handler/listener, event source

Java skills:

- Create applets and place on web pages.
- Use the Java coordinate system.
- Draw basic shapes, change colors and fonts.
- Window applications using `JFrame` and `JPanel`.
- Java Swing components: `JButton`, `JCheckBox`, `JComboBox`, `JLabel`, `JPanel`, `JRadioButton`, `JTextField`, `JTextArea`
- Event listeners versus event adapters
- Mouse events: `MouseListener`, `MouseAdapter`
- Window events: `WindowListener`, `WindowAdapter`
- `ActionListener` and use with `JButton`

Objectives (2)

Java skills (cont.):

- Using inner classes.
- Menus: `JMenu`, `JMenuItem`, `JMenuBar`
- `Timer` and timer events
- Keyboard events: `KeyListener`