Lecture 13-2: GUI/Event Programming
JComponent
This class diagram shows only a **small** subset of the methods that you will gain when you extend JFrame.
Displaying Your JFrame

✔ Give the window a size
✔ Give it a screen location
✔ Give it a title
✔ Clean up JFrame annoyances
  ✔ Disable layout managers
  ✔ Change default close operation
  ✔ Disallow resizable frames
✔ Show the frame!

```java
import javax.swing.*;

public class HelloSwingWorld extends JFrame {
    public HelloSwingWorld () {
        // initialize JFrame here
        setSize (400, 300); // 400 by 300 pixels
        setPosition (100, 100); // 100, 100 from the top left
        setTitle ("Hello Swing World");
        setLayout (null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
    }

    public static void main (String[] args) {
        JFrame obj = new HelloSwingWorld();
        obj.setVisible(true);
    }
}
```
In Swing, a JFrame is similar to a window in your operating system

- All components will appear inside the JFrame window
  - Buttons, text labels, text fields, etc.
It is exciting to display a window, but in order to interact with the user, we need some components in the frame.

- Components are things like buttons, text fields, labels, scroll bars, radio buttons, check boxes, drop-down lists, etc.
- There are many available components, each is its own class.
- However, they are all inherited from the JComponent parent class.

Each component would normally need its own import:

- Buttons: import javax.swing.JButton
- Text fields: import javax.swing.JTextField
- Etc.
- But we already imported all with import javax.swing.*
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

Linux
JComponent

• JButton
• JLabel
• JTextField
• JCheckBox
• JRadioButton (x2)
• JComboBox
• JTextArea
• JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
There are many features common to all JComponents

For example, JComponents will all need to have their size and location defined

The common methods can appear in the parent JComponent class

Each JComponent will also have some features that are unique

These methods will appear in the child classes, JButton, JTextField, etc.
Absolute Positioning

- The JFrame position is relative to the screen origin
The position of any JComponent is relative to the frame origin.
JComponent Size and Position

- Most JComponent sizes are in units of pixels
  - JTextArea is the exception, it’s size is in letters
- Locations will be in pixels

- For any components you want on the frame:
  - Instantiate the component
  - Set it’s size
  - Set it’s location
  - Use the frame add( ) method to place the component within the frame
JComponent Size and Position

- Most JComponent sizes are in units of pixels
  - JTextArea is the exception, it’s size is in letters
- Locations will be in pixels

For any components you want on the frame:

- Instantiate the component
- Set it’s size
- Set it’s location
- Use the frame `add()` method to place the component within the frame

<table>
<thead>
<tr>
<th>JFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>constructor</code></td>
</tr>
<tr>
<td><code>JFrame(String)</code></td>
</tr>
<tr>
<td><code>update</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><code>query</code></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Absolute Positioning
Absolute Positioning

Each JButton size is 300 x 150
Absolute Positioning

Each JButton size is 300 x 150
Each JButton size is 300 x 150
Absolute Positioning

Each JButton size is 300 x 150
JButton
• These are the methods you are most likely to use with JButtons

• The full list of JButton methods is huge!

```java
JButton

«constructor»
+ JButton()
+ JButton(String)

«update»
+ void setLocation(int, int)
+ void setSize(int, int)
+ void setText(String)
+ void setToolTipText(String)
+ void setVisible(boolean)

«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isVisible()
```
JButton Constructors

- **Two constructors**
  - The String parameter can be used to set the text that appears on the button
- **It is common to make any JComponent a private class attribute**

```java
import javax.swing.*;
public class ButtonDemo extends JFrame {
    private JButton myButton;
    public ButtonDemo () {
        // initialize JFrame here
        myButton = new JButton("Click Me");
    }

    public static void main (String[] args) {
        ButtonDemo obj = new ButtonDemo();
        obj.setVisible(true);
    }
}
```
JButton Size and Location

- Use setLocation and setSize
  - setLocation arguments should be X, then Y
  - setSize arguments should be width, then height

```java
import javax.swing.*;
public class ButtonDemo extends JFrame {
    private JButton myButton;
    public ButtonDemo () {
        // initialize JFrame here
        myButton = new JButton("Click Me");
        myButton.setLocation(10, 30);
        myButton.setSize(100, 60);
    }

    public static void main (String[] args) {
        ButtonDemo obj = new ButtonDemo();
        obj.setVisible(true);
    }
}
```
Add the JButton to the Frame

- You can now add the JButton to the frame
- JButton is a child of JComponent, and the JFrame add method can place any JComponent within the frame

```java
import javax.swing.*;
public class ButtonDemo extends JFrame {
    private JButton myButton;
    public ButtonDemo () {
        // initialize JFrame here
        myButton = new JButton("Click Me");
        myButton.setLocation(10, 30);
        myButton.setSize(100, 60);
        this.add(myButton);
    }

    public static void main (String[] args) {
        ButtonDemo obj = new ButtonDemo();
        obj.setVisible(true);
    }
}
```
Other JButton Methods

- Tool Tips are the little boxes of text that appear when you hover your mouse pointer over a component.

```java
JButton
«constructor»
+ JButton()
+ JButton(String)
«update»
+ void setLocation(int, int)
+ void setSize(int, int)
+ void setText(String)
+ void setToolTipText(String)
+ void setVisible(boolean)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isVisible()
```
import javax.swing.*;
public class ButtonDemo extends JFrame {

    private JButton myButton;

    public ButtonDemo () {
        // initialize JFrame
        this.setSize (300, 150);
        this.setPosition (100, 100);
        this.setTitle ("Button Demo");
        this.setLayout (null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);

        // initialize JButton
        myButton = new JButton();
        myButton.setText("Click Me");
        myButton.setToolTipText("Stop hovering, start clicking");
        myButton.setLocation(100, 40);
        myButton.setSize(100, 40);
        this.add(myButton);
    }

    public static void main (String[] args) {
        ButtonDemo obj = new ButtonDemo();
        obj.setVisible(true);
    }
}
JLabel

Look at me!
**JLabel**

- Sometimes you need to add some non-editable text to the frame
  - For example, some JComponents don’t already have text included (like JTextField)
  - Or you may want to label a group of components

- JLabels serve a different purpose than JButtons, but many methods overlap

<table>
<thead>
<tr>
<th>JLabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>«constructor»</td>
</tr>
<tr>
<td>+ JLabel()</td>
</tr>
<tr>
<td>+ JLabel(String)</td>
</tr>
<tr>
<td>«update»</td>
</tr>
<tr>
<td>+ void setLocation(int, int)</td>
</tr>
<tr>
<td>+ void setSize(int, int)</td>
</tr>
<tr>
<td>+ void setText(String)</td>
</tr>
<tr>
<td>+ void setVisible(boolean)</td>
</tr>
<tr>
<td>+ void setBorder(Border)</td>
</tr>
<tr>
<td>+ void setVerticalAlignment(int)</td>
</tr>
<tr>
<td>+ void setHorizontalAlignment(int)</td>
</tr>
<tr>
<td>«query»</td>
</tr>
<tr>
<td>+ int getWidth()</td>
</tr>
<tr>
<td>+ int getHeight()</td>
</tr>
<tr>
<td>+ int getX()</td>
</tr>
<tr>
<td>+ int getY()</td>
</tr>
<tr>
<td>+ boolean isVisible()</td>
</tr>
</tbody>
</table>
JLabel Size and Location

- The size and location are measured in pixels, just like buttons
  - Changing the size does not change the font size
- The dimensions refer to an imaginary bounding box around the parameter of the label
The size and location are measured in pixels, just like buttons.
- Changing the size does **not** change the font size.
- The dimensions refer to an imaginary bounding box around the parameter of the label.

Size too small for the amount of text.
JLabel Size and Location

- The size and location are measured in pixels, just like buttons.
- Changing the size does **not** change the font size.
- The dimensions refer to an imaginary bounding box around the parameter of the label.

Size too small for the amount of text.

GOTCHA
It is possible to see the bounding box by setting a visible border.

This can help you visualize position and size.

To do so requires using a few new classes.

Remove the borders once you have the correct size and location!

```java
public class JLabel {

    public JLabel();
    public JLabel(String text);

    public void setLocation(int x, int y);
    public void setSize(int width, int height);
    public void setText(String text);
    public void setVisible(boolean isVisible);
    public void setBorder(Border border);

    public int getWidth();
    public int getHeight();
    public int getX();
    public int getY();
    public boolean isVisible();
}
```
Border Interface

- **Notice that** `setBorder` **has a** `Border` **as the parameter**
- **Border** is an interface
  - Requires that you create a class that implements several methods
  - Another example of type conformance
- **Fortunately, someone has already done the heavy lifting...**

```java
interface Border

- update
  + void paintBorder(Component, Graphics, int, int, int, int)

- query
  + Insets getBorderInsets(Component)
  + boolean isBorderOpaque()
```
BorderFactory Class

• The BorderFactory class has several static methods that return various styles of border

• Since the methods are static, you don’t need to create an instance of BorderFactory

```java
JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

```java
JLabel myLabel;
myLabel = new JLabel("Woo bevel!");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```
BorderFactory Class

- The BorderFactory class has several static methods that return various styles of border
- Since the methods are static, you don’t need to create an instance of BorderFactory

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

JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```
JLabel Alignment

- You can also optionally change the horizontal and vertical alignment
  - Use one of several constants that are part of the SwingConstants class

```
JLabel
«constructor»
+ JLabel()
+ JLabel(String)
«update»
+ void setLocation(int, int)
+ void setSize(int, int)
+ void setText(String)
+ void setVisible(boolean)
+ void setBorder(Border)
+ void setVerticalAlignment(int)
+ void setHorizontalAlignment(int)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isVisible()
```
### JLabel Alignment

myLabel.setHorizontalAlignment(SwingConstants.LEFT);
myLabel.setVerticalAlignment(SwingConstants.CENTER);

```java
// horizontal alignment
SwingConstants.LEFT
SwingConstants.RIGHT
SwingConstants.CENTER

// vertical alignment
SwingConstants.TOP
SwingConstants.BOTTOM
SwingConstants.CENTER
```

myLabel.setHorizontalAlignment(SwingConstants.RIGHT);
myLabel.setVerticalAlignment(SwingConstants.TOP);
import javax.swing.*;
public class LabelDemo extends JFrame {
    private JLabel myLabel;

    public LabelDemo () {
        // initialize JFrame
        this.setSize (300, 150);
        this.setPosition (100, 100);
        this.setTitle ("Label Demo");
        this.setLayout (null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);

        // initialize JLabel
        myLabel = new JLabel("Hello World!");
        myLabel.setLocation(100, 20);
        myLabel.setSize(100, 20);
        myLabel.setHorizontalAlignment(SwingConstants.LEFT);
        myLabel.setVerticalAlignment(SwingConstants.CENTER);
        this.add(myLabel);
    }

    public static void main (String[] args) {
        LabelDemo obj = new LabelDemo();
        obj.setVisible(true);
    }
}
JComponent

- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane
Add the JButton to the Frame

- You can now add the JButton to the frame
- JButton is a child of JComponent, and the JFrame add method can place any JComponent within the frame

```java
import javax.swing.*;
public class ButtonDemo extends JFrame {
    private JButton myButton;
    public ButtonDemo () {
        // initialize JFrame here
        myButton = new JButton("Click Me");
        myButton.setLocation(10, 30);
        myButton.setSize(100, 60);
        this.add(myButton);
    }

    public static void main (String[] args) {
        ButtonDemo obj = new ButtonDemo();
        obj.setVisible(true);
    }
}
```
JLabel

Look at me!
Sometimes you need to add some non-editable text to the frame

For example, some JComponents don’t already have text included (like JTextField)

Or you may want to label a group of components

JLabels serve a different purpose than JButtons, but many methods overlap
JLabel Size and Location

• The size and location are measured in pixels, just like buttons
  • Changing the size does not change the font size
• The dimensions refer to an imaginary bounding box around the parameter of the label
JLabel Size and Location

- The size and location are measured in pixels, just like buttons
  - Changing the size does not change the font size
- The dimensions refer to an imaginary bounding box around the parameter of the label

Size too small for the amount of text.
JLabel Size and Location

- The size and location are measured in pixels, just like buttons.
- Changing the size does not change the font size.
- The dimensions refer to an imaginary bounding box around the parameter of the label.

Size too small for the amount of text.
It is possible to see the bounding box by setting a visible **border**

- This can help you visualize position and size
- To do so requires using a few new classes

**Remove the borders once you have the correct size and location!**

```java
JLabel
«constructor»
+ JLabel()
+ JLabel(String)
«update»
+ void setLocation(int, int)
+ void setSize(int, int)
+ void setText(String)
+ void setVisible(boolean)
+ void setBorder(Border)
+ void setVerticalAlignment(int)
+ void setHorizontalAlignment(int)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isVisible()
```
Border Interface

- **Notice that** `setBorder` **has a Border as the parameter**
- **Border is an interface**
  - Requires that you create a class that implements several methods
  - Another example of type conformance
- **Fortunately, someone has already done the heavy lifting...**

```java
interface Border
{
    void paintBorder(Component, Graphics, int, int, int, int);
    Insets getBorderInsets(Component);
    boolean isBorderOpaque();
}
```
BorderFactory Class

- The BorderFactory class has several **static** methods that return various styles of border
- Since the methods are **static**, you don’t need to create an instance of BorderFactory

```java
JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

```java
JLabel myLabel;
myLabel = new JLabel("Woo bevel!");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```
• The BorderFactory class has several static methods that return various styles of border
• Since the methods are static, you don’t need to create an instance of BorderFactory

import java.awt.Color;
import javax.swing.border.*;

JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

myLabel = new JLabel("Woo bevel!");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);
myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
You can also optionally change the horizontal and vertical alignment

- Use one of several constants that are part of the SwingConstants class

```java
JLabel
«constructor»
+ JLabel()
+ JLabel(String)
«update»
+ void setLocation(int, int)
+ void setSize(int, int)
+ void setText(String)
+ void setVisible(boolean)
+ void setBorder(Border)
+ void setVerticalAlignment(int)
+ void setHorizontalAlignment(int)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isVisible()
```
JLabel Alignment

myLabel.setHorizontalAlignment(SwingConstants.LEFT);
myLabel.setVerticalAlignment(SwingConstants.CENTER);

myLabel.setHorizontalAlignment(SwingConstants.RIGHT);
myLabel.setVerticalAlignment(SwingConstants.TOP);

// horizontal alignment
SwingConstants.LEFT
SwingConstants.RIGHT
SwingConstants.CENTER

// vertical alignment
SwingConstants.TOP
SwingConstants.BOTTOM
SwingConstants.CENTER

myLabel.setHorizontalAlignment(SwingConstants.RIGHT);
myLabel.setVerticalAlignment(SwingConstants.TOP);
import javax.swing.*;
public class LabelDemo extends JFrame {
    private JLabel myLabel;

    public LabelDemo () {
        // initialize JFrame
        this.setSize (300, 150);
        this.setPosition (100, 100);
        this.setTitle ("Label Demo");
        this.setLayout (null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);

        // initialize JLabel
        myLabel = new JLabel("Hello World!");
        myLabel.setLocation(100, 20);
        myLabel.setSize(100, 20);
        myLabel.setHorizontalAlignment(SwingConstants.LEFT);
        myLabel.setVerticalAlignment(SwingConstants.CENTER);
        this.add(myLabel);
    }

    public static void main (String[] args) {
        LabelDemo obj = new LabelDemo();
        obj.setVisible(true);
    }
}
JTextField
JTextField

- JTextFields are useful for cases when you want the user to enter small amounts of text
  - Includes methods used to get input from the user
  - Next week we will write code to handle actions taken by the user
- These differ from JLabels since the user can modify the text

```
JTextField

«constructor»
+ JTextField()
+ JTextField(String)

«update»
+ void setLocation(int,int)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setText(String)
+ void requestFocus()
+ void selectAll()

«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ String getText()
+ String getSelectedText()
```
Many of the new methods only work when used with event handlers that we will write next week.

```java
import javax.swing.*;
public class TextDemo extends JFrame {
    private JTextField myTextField;
    public TextDemo () {
        // initialize JFrame here
        myTextField = new JTextField("Initial text");
        myTextField.setLocation(10, 30);
        myTextField.setSize(100, 60);
        this.add(myTextField);
    }

    public static void main (String[] args) {
        JFrame obj = new TextDemo();
        obj.setVisible(true);
    }
}
```
JCheckBox
JCheckBox can be used to allow the user to select/unselect some, none, or all of a set of options.

```
«constructor»
+ JCheckBox()
+ JCheckBox(String, boolean)

«update»
+ void setLocation(int, int)
+ void setVisible(boolean)
+ void setSize(int, int)
+ void setText(String)

«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isSelected()
```
JCheckBox Constructor

- JCheckBox components **do not** need a separate JLabel for each check box
- The JCheckBox constructor can optionally take the label text and initial value as arguments
  - Checked has the value **true**
  - Unchecked has the value **false**

<table>
<thead>
<tr>
<th>JCheckBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>«constructor»</td>
</tr>
<tr>
<td>+ JCheckBox()</td>
</tr>
<tr>
<td>+ JCheckBox(String,boolean)</td>
</tr>
<tr>
<td>«update»</td>
</tr>
<tr>
<td>+ void setLocation(int,int)</td>
</tr>
<tr>
<td>+ void setVisible(boolean)</td>
</tr>
<tr>
<td>+ void setSize(int,int)</td>
</tr>
<tr>
<td>+ void setText(String)</td>
</tr>
<tr>
<td>«query»</td>
</tr>
<tr>
<td>+ int getWidth()</td>
</tr>
<tr>
<td>+ int getHeight()</td>
</tr>
<tr>
<td>+ int getX()</td>
</tr>
<tr>
<td>+ int getY()</td>
</tr>
<tr>
<td>+ boolean isSelected()</td>
</tr>
</tbody>
</table>
Next week we will read values from checkboxes using isSelected

```java
import javax.swing.*;
public class CheckDemo extends JFrame {
    private JCheckBox myCheckBox;
    public CheckDemo () {
        // initialize JFrame here
        myCheckBox = new JCheckBox("Initial text",true);
        myCheckBox.setLocation(10, 30);
        myCheckBox.setSize(100, 60);
        this.add(myCheckBox);
    }

    public static void main (String[] args) {
        JFrame obj = new CheckDemo();
        obj.setVisible(true);
    }
}
```
JRadioButton
JUnit

• Use JRadioButton when the user should select **exactly one** item out of several possibilities
• JRadioButton components also **do not** need a separate JLabel for each radio button
• You need a JRadioButton for each radio button that you would like
  • However, we need to group them together in order to implement the single selection
  • This requires using a second class

JRadioButton

<table>
<thead>
<tr>
<th>JRadioButton</th>
</tr>
</thead>
<tbody>
<tr>
<td>«constructor»</td>
</tr>
<tr>
<td>+ JRadioButton()</td>
</tr>
<tr>
<td>+ JRadioButton(String,boolean)</td>
</tr>
<tr>
<td>«update»</td>
</tr>
<tr>
<td>+ void setLocation(int,int)</td>
</tr>
<tr>
<td>+ void setVisible(boolean)</td>
</tr>
<tr>
<td>+ void setSize(int,int)</td>
</tr>
<tr>
<td>+ void setText(String)</td>
</tr>
<tr>
<td>«query»</td>
</tr>
<tr>
<td>+ int getWidth()</td>
</tr>
<tr>
<td>+ int getHeight()</td>
</tr>
<tr>
<td>+ int getX()</td>
</tr>
<tr>
<td>+ int getY()</td>
</tr>
<tr>
<td>+ boolean isSelected()</td>
</tr>
</tbody>
</table>
JRadioButton Groups

• Swing does not assume that all radio buttons on a given frame are associated with each other
  • If you don’t group radio buttons, you don’t get the exactly one policy normally assumed with radio buttons

• You might want to have several groups of radio buttons, you need to indicate which radio buttons belong to which groups
ButtonGroup Class

- The ButtonGroup class is used to group radio buttons
- For each group:
  - Instantiate a ButtonGroup object
  - add the JRadioButton objects to the ButtonGroup

```java
// in the frame constructor...
ButtonGroup group = new ButtonGroup();

JRadioButton yes = new JRadioButton("Yes", true);
JRadioButton no = new JRadioButton("No", false);
yes.setLocation(20, 40);
yes.setSize(100, 20);
no.setLocation(20, 65);
no.setSize(100, 20);

// add the buttons to the ButtonGroup
group.add(yes);
group.add(no);

// add the buttons to the frame
this.add(yes);
this.add(no);
```
ButtonGroup Class

• The ButtonGroup class is used to group radio buttons
• For each group:
  • Instantiate a ButtonGroup object
  • add the JRadioButton objects to the ButtonGroup

```java
// in the frame constructor...
ButtonGroup group = new ButtonGroup();

JRadioButton yes = new JRadioButton("Yes",true);
JRadioButton no = new JRadioButton("No",false);
yes.setLocation(20, 40);
yes.setSize(100, 20);
no.setLocation(20, 65);
no.setSize(100, 20);

// add the buttons to the ButtonGroup
group.add(yes);
group.add(no);

// add the buttons to the frame
this.add(yes);
this.add(no);
```

The ButtonGroup does not need to be added to the frame
JComboBox
JComboBox

The JComboBox is another way to allow the user to select exactly one item out of several possibilities

- Do not require groups
- Only need to instantiate one JComboBox

Eclipse will have warnings about JComboBoxes being raw types, you can ignore these warnings for this class

<table>
<thead>
<tr>
<th>JComboBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>«constructor»</td>
</tr>
<tr>
<td>+ JComboBox()</td>
</tr>
<tr>
<td>+ JComboBox(String[])</td>
</tr>
<tr>
<td>«update»</td>
</tr>
<tr>
<td>+ void setLocation(int,int)</td>
</tr>
<tr>
<td>+ void setVisible(boolean)</td>
</tr>
<tr>
<td>+ void setSize(int,int)</td>
</tr>
<tr>
<td>+ void addItem(String)</td>
</tr>
<tr>
<td>+ void setEditable(boolean)</td>
</tr>
<tr>
<td>«query»</td>
</tr>
<tr>
<td>+ int getWidth()</td>
</tr>
<tr>
<td>+ int getHeight()</td>
</tr>
<tr>
<td>+ int getX()</td>
</tr>
<tr>
<td>+ int getY()</td>
</tr>
<tr>
<td>+ String getSelectedItem()</td>
</tr>
<tr>
<td>+ int getSelectedIndex()</td>
</tr>
</tbody>
</table>
• The JComboBox can optionally allow the user to type in an option that is not already on the list
• Does not automatically get added to the list however

```
JComboBox

«constructor»
+ JComboBox()
+ JComboBox(String[])

«update»
+ void setLocation(int, int)
+ void setVisible(boolean)
+ void setSize(int, int)
+ void addItem(String)
+ void setEditable(boolean)

«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ String getSelectedItem()
+ int getSelectedIndex()
```
Notice that the constructor can take an array of Strings as an argument

```java
import javax.swing.*;
public class ComboDemo extends JFrame {
    private JComboBox myComboBox;
    public ComboDemo () {
        // initialize JFrame here
        String options[] = {"apples","bananas","strawberries","pears","watermelons"};
        myComboBox = new JComboBox(options);
        myComboBox.setLocation(10, 30);
        myComboBox.setSize(100, 60);
        this.add(myComboBox);
    }

    public static void main (String[] args) {
        JFrame obj = new ComboDemo();
        obj.setVisible(true);
    }
}
```
JTextArea
JTextArea

- JTextArea is similar to JTextField, but has a bit more support for large amounts of text
- Typically used along with JScrollPane to add scroll bars

```java
JTextArea

«constructor»
+ JTextArea()

«update»
+ void setLocation(int, int)
+ void setVisible(boolean)
+ void setSize(int, int)
+ void setText(String)
+ void append(String)
+ void setLineWrap(boolean)
+ void setWrapStyleWord(boolean)

«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ String getText()
```
• The `setText` method replaces all text in the text area
• The `append` method adds text after the existing text in the text area

```java
import javax.swing.*;
public class TextAreaDemo extends JFrame {
    private JTextArea myTextArea;
    public TextAreaDemo () {
        // initialize JFrame here
        myTextArea = new JTextArea();
        myTextArea.setLocation(10, 10);
        myTextArea.setSize(100, 100);
        myTextArea.setText("Some text");
        myTextArea.append("\nMore text");
        this.add(myTextArea);
    }

    public static void main (String[] args) {
        JFrame obj = new TextAreaDemo();
        obj.setVisible(true);
    }
}
```
The `setText` method replaces all text in the text area.
The `append` method adds text after the existing text in the text area.

```java
import javax.swing.*

public class TextAreaDemo extends JFrame {
    private JTextArea myTextArea;
    public TextAreaDemo() {
        // initialize JFrame here
        myTextArea = new JTextArea();
        myTextArea.setLocation(10, 10);
        myTextArea.setSize(100, 100);
        myTextArea.setText("Some text");
        myTextArea.append("\nMore text");
        this.add(myTextArea);
    }
    public static void main (String[] args) {
        JFrame obj = new TextAreaDemo();
        obj.setVisible(true);
    }
}
```
**JTextArea Word Wrap**

- JTextArea does not automatically add scroll bars, nor does it automatically wrap text

```java
import javax.swing.*;
public class TextAreaDemo extends JFrame {
    private JTextArea myTextArea;
    public TextAreaDemo () {
        // initialize JFrame here
        myTextArea = new JTextArea();
        myTextArea.setLocation(10, 10);
        myTextArea.setSize(100, 100);
        myTextArea.setText("Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod...
        this.add(myTextArea);
    }

    public static void main (String[] args) {
        JFrame obj = new TextAreaDemo();
        obj.setVisible(true);
    }
}
```
JTextArea Word Wrap

- The `setLineWrap` method will automatically wrap text

```java
myTextArea.setLineWrap(true);
myTextArea.setText("Lorem ipsum dolor sit amet, consectetur...");
```

- The `setWrapStyleWord` method will wrap at word boundaries

```java
myTextArea.setLineWrap(true);
myTextArea.setText("Lorem ipsum dolor sit amet, consectetur...");
```

```java
myTextArea.setLineWrap(true);
myTextArea.setWrapStyleWord(true);
myTextArea.setText("Lorem ipsum dolor sit amet, consectetur...");
```
Adding scroll bars unfortunately requires a second class, JScrollPane

The JTextArea is added to the pane, and then the pane is added to the frame

You do not use the JTextArea setSize or setLocation when adding it to a pane, the JTextArea will get the size and location from the pane

GOTCHA
The constructor takes two integers whose values represent the vertical and horizontal scrollbar settings.

Use the constants that are part of JScrollPane for these int arguments.

```java
// vertical scrollbars
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
JScrollPane.VERTICAL_SCROLLBAR_NEVER
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED

// horizontal scrollbars
JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED
```
import javax.swing.*;
public class TextAreaDemo extends JFrame {
    private JTextArea myTextArea;
    public TextAreaDemo () {
        // initialize JFrame here
        myTextArea = new JTextArea();
        myTextArea.setLineWrap(true);
        myTextArea.setWrapStyleWord(true);
        myTextArea.setText("Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod..."
                        + "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod...");
        JScrollPane pane = new JScrollPane(myTextArea,
                                             JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                             JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        pane.setLocation(10, 10);
        pane.setSize(100, 100);
        this.add(pane);
    }

    public static void main (String[] args) {
        JFrame obj = new TextAreaDemo();
        obj.setVisible(true);
    }
}
Java Graphical User Interfaces

• At this point, we have a JFrame and one or more JComponents within the JFrame
  • But no way to act on them
• The things you do to interact with windows and screen components are called events
  • Moving the mouse
  • Clicking components
  • Typing keys on the keyboard
  • Clicking and dragging
  • Etc.
• Some events are caused by the mouse interacting with the environment, some with the keyboard
You might think there is only one type of event with a screen component
- However, events may occur more frequently than you assume
- There may also be events that you never think about
Java Graphical User Interfaces

• You might think there is only one type of event with a screen component
  • However, events may occur more frequently than you assume
  • There may also be events that you never think about

What events do you think can occur with a button?
JButton Events (Partial List)

- Some of events possible with JButtons
  - Mouse pressed
  - Mouse released
  - Mouse clicked
  - Mouse entered area
  - Mouse left area
  - Got focus
  - Lost focus
  - Key pressed
  - Key released
  - Key typed

- Each JComponent has a similar list of events that are possible with that component
Some of events possible with JButtons

- Mouse pressed
- Mouse released
- Mouse clicked
- Mouse entered area
- Mouse left area
- Got focus
- Lost focus
- Key pressed
- Key released
- Key typed

Each JComponent has a similar list of events that are possible with that component
JButton Events (Partial List)

- Some of events possible with JButtons
  - Mouse pressed
  - Mouse released
  - Mouse clicked
  - Mouse entered area
  - Mouse left area
  - Got focus
  - Lost focus
  - Key pressed
  - Key released
  - Key typed
- Each JComponent has a similar list of events that are possible with that component
Some of events possible with JButtons

- Mouse pressed
- Mouse released
- Mouse clicked
- Mouse entered area
- Mouse left area
- Got focus
- Lost focus
- Key pressed
- Key released
- Key typed

Each JComponent has a similar list of events that are possible with that component
Event Listeners

- We need to specify what happens when an event occurs.
- Swing uses *listener interfaces* to allow you to write a method that will execute when an event occurs.
  - Remember that an interface forces you to write certain methods.

- There are several types of listeners, grouped by the events that they cause:
  - WindowListener
  - FocusListener
  - MouseListener
  - KeyListener
  - ActionListener

- You only need to implement the listeners that you intend to use!
Event Listeners

• To use a listener
  • Listeners require `import java.awt.event.*`
  • Write a class that `implements` the required listener interface
  • You need to write each method specified by the interface
  • Register the listener with the frame or component

• Swing, Java, and your operating system automatically monitors all events
  • When the event occurs, Java will call your method that you registered
  • You will never need to explicitly call the methods you implemented
WindowListener
The WindowListener interface has methods for frame-related events

- Opened: when the frame is displayed
- Closing: when the frame is being closed
- Closed: when the frame is done closing
- Iconified: when the frame is minimized
- Deiconified: when the frame is brought back from minimized
- Activated: when the window is selected
- Deactivated: when another window is selected
Event Listeners

- To use a listener
  - import
  - implements the required listener interface
  - Write each method specified by the interface
  - Register the listener with the frame or component

```java
import javax.swing.*;

public class HelloSwingWorld extends JFrame {
    public HelloSwingWorld () {
        // initialize JFrame
        this.setSize(400, 300);
        this.setLocation(100, 100);
        this.setTitle("Hello Swing World");
        this.setLayout(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
Event Listeners

• To use a listener
  • import
  • implements
  the
  required
  listener
  interface
  • Write each method
    specified by the
    interface
  • Register the listener
    with the frame or
    component

```java
import javax.swing.*;

public class HelloSwingWorld extends JFrame {
    public HelloSwingWorld () {
        // Initialize JFrame
        this.setSize(400, 300);
        this.setLocation(100, 100);
        this.setTitle("Hello Swing World");
        this.setLayout(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
Listener Interface Options

- These steps require you to write a class

- **Two main approaches:**
  - Use a single class – the one that inherits from JFrame
    - We’ll use this style for now
  - Or use a separate class for each interface
    - We’ll come back to this if we have time
WindowListener

- To use a listener
  - ✓ import
  - implements the required listener interface
  - Write each method specified by the interface
  - Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame {
    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}```
To use a listener

✔ import
✔ implements the required listener interface

• Write each method specified by the interface

• Register the listener with the frame or component

```java
class HelloSwingWorld extends JFrame implements WindowListener{
    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
To use a listener:

✔ import
✔ Implements the required listener interface

• Write each method specified by the interface

• Register the listener with the frame or component

Once we add this, our program will no longer compile, until...

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

public class HelloSwingWorld extends JFrame implements WindowListener{
    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
WindowListener

• To use a listener
  ✔ import
  ✔ implements the required listener interface
  ✔ Write each method specified by the interface

• Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements WindowListener{
  public HelloSwingWorld () {
    // JFrame initialization here (not shown)
  }

  public static void main (String[] args) {
    JFrame frame = new HelloSwingWorld();
    frame.setVisible(true);
  }

  public void windowOpened (WindowEvent event){
    System.out.println("Window has opened!");
  }

  public void windowClosing (WindowEvent event){
    System.out.println("Window is closing!");
  }

  // ran out of room on this slide, but put the other
  // methods here too: windowClosed, windowIconified,
  // windowDeiconified, windowActivated, and
  // windowDeactivated
}
```
Registering a WindowListener

- JFrame and the JComponents all have addXListener methods
  - Where X is the type of listener
  - Not all listeners are supported by all JFrame and JComponents

```java
public class JFrame {
    public JFrame();
    public JFrame(String); // constructor

    // update methods
    public void add(JComponent);
    public void setLocation(int, int);
    public void setSize(int, int);
    public void setTitle(String);
    public void addWindowListener(WindowListener);

    // other methods
}
```
• To use a listener
  ✔ import
  ✔ implements the required listener interface
  ✔ Write each method specified by the interface
  ✔ Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame
    implements WindowListener{
    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        this.addWindowListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void windowOpened (WindowEvent event){
        System.out.println("Window has opened!");
    }

    public void windowClosing (WindowEvent event){
        System.out.println("Window is closing!");
    }

    // ran out of room on this slide, but put the other
    // methods here too: windowClosed, windowIconified,
    // windowDeiconified, windowActivated, and
    // windowDeactivated
```
To use a listener:
- Import
- Implement the required listener interface
- Write each method specified by the interface
- Register the listener with the frame or component

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

class HelloSwingWorld extends JFrame implements WindowListener{
    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        this.addWindowListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void windowOpened (WindowEvent event){
        System.out.println("Window has opened!");
    }

    public void windowClosing (WindowEvent event){
        System.out.println("Window is closing!");
    }

    // ran out of room on this slide, but put the other
    // methods here too: windowClosed, windowIconified,
    // windowDeiconified, windowActivated, and
    // windowDeactivated
}
```
Notice that each WindowListener method requires a WindowEvent parameter
- Used to get information about what caused the event to occur
- Some events have more interesting information than others
- The getSource method returns the memory reference of the object that caused the event to occur
## Listeners and Components

- The following components can register the following listeners

<table>
<thead>
<tr>
<th>Component</th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JButton</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JComboBox</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
FocusListener
Focus

• With GUIs, you often have several components that the user can interact with

• The current component that the user is interacting with is said to have the focus

• For example:
  • Buttons can be selected without being clicked – the user can “click” the button by hitting space
  • One of several text fields can have the cursor – the user can move between text fields with tab
  • A drop-down box can be shown – the user can change the selection with the up/down arrow keys

• The user can change focus from one JComponent to another by pressing the tab key
Focus

Button 1 currently has focus

Text field 2 currently has focus
FocusListener Interface

• The FocusListener interface has methods for focus events
  • Gained: when the component or frame receives focus
  • Lost: when the focus moves to another component or frame

```java
interface FocusListener
{
    void focusGained(FocusEvent)
    void focusLost(FocusEvent)
}
```
The default JTextField behavior does **not** automatically select the text when the field receives focus.

Now we can change this behavior by implementing a FocusListener.

```java
import javax.swing.*;
public class TextDemo extends JFrame {
    private JTextField myTextField;
    public TextDemo () {
        // initialize JFrame here
        myTextField = new JTextField("Initial text");
        myTextField.setLocation(10, 30);
        myTextField.setSize(100, 60);
        this.add(myTextField);
    }

    public static void main (String[] args) {
        JFrame obj = new TextDemo();
        obj.setVisible(true);
    }
}
```
FocusListener

- To use a listener
  - ✔ import
  - ✔ implements the required listener interface
- Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements FocusListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
FocusEvent

- The `getSource` method returns the memory reference of the object that caused the event to occur.
- This has the same use as the WindowEvent.

```java
interface FocusListener {
    void focusGained(FocusEvent)
    void focusLost(FocusEvent)
}
```

```java
class FocusEvent {
    Object getSource()
}
```
FocusListener

- To use a listener
  ✓ import
  ✓ implements the required listener interface
  ✓ Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements FocusListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void focusGained (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.selectAll();
        }
    }

    public void focusLost (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.select(0,0);
        }
    }
}
```
FocusListener

- To use a listener
  ✔ import
  ✔ implements the required listener interface
  ✔ Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements FocusListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);

        public void focusGained (FocusEvent event){
            if (event.getSource() == myTextField){
                myTextField.selectAll();
            }
        }

        public void focusLost (FocusEvent event){
            if (event.getSource() == myTextField){
                myTextField.select(0,0);
            }
        }
    }
}
```

This is how we can determine which component got focus.
FocusListener

- To use a listener
  - ✔ import
  - ✔ implements the required listener interface
  - ✔ Write each method specified by the interface
  - ✔ Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame
    implements FocusListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
        myTextField.addActionListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void focusGained (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.selectAll();
        }
    }

    public void focusLost (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.select(0,0);
        }
    }
}
```
FocusListener

- To use a listener
  ✓ import
  ✓ implements the required listener interface
  ✓ Write each method specified by the interface
  ✓ Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame
{
    private JTextField myTextField;

    public HelloSwingWorld()
    {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
        myTextField.addFocusListener(this);
    }

    public static void main(String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void focusGained(FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.selectAll();
        }
    }

    public void focusLost(FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.select(0,0);
        }
    }
}
The following components can register the following listeners:

<table>
<thead>
<tr>
<th>Component</th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JButton</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JComboBox</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
• Remember that we can possibly implement multiple interfaces

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

public class HelloWorld extends JFrame {
    implements WindowListener, FocusListener {
    private JTextField myTextField;

    public HelloWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
        this.addWindowListener(this);
        myTextField.addFocusListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloWorld();
        frame.setVisible(true);
    }

    public void windowOpened (WindowEvent event){
        myTextField.requestFocus();
    }

    public void focusGained (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.selectAll();
        }
    }

    // don’t forget the other listener methods (not shown)
}
```

Interfaces
Event Listeners

• We need to specify what happens when an event occurs
• Swing uses **listener interfaces** to allow you to write a method that will execute when an event occurs
  • Remember that an interface forces you to write certain methods

• There are several types of listeners, grouped by the events that they cause
  • WindowListener
  • FocusListener
  • MouseListener
  • KeyListener
  • ActionListener

• You only need to implement the listeners that you intend to use!
Event Listeners

• To use a listener
  • Listeners require import java.awt.event.*
  • Write a class that implements the required listener interface
  • You need to write each method specified by the interface
  • Register the listener with the frame or component

• Swing, Java, and your operating system automatically monitors all events
  • When the event occurs, Java will call your method that you registered
  • You will never need to explicitly call the methods you implemented
Interfaces

• Remember that we can possibly implement multiple interfaces

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

public class HelloSwingWorld extends JFrame
    implements WindowListener, FocusListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
        this.setWindowListener(this);
        myTextField.setFocusListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void windowOpened (WindowEvent event){
        myTextField.requestFocus();
    }

    public void focusGained (FocusEvent event){
        if (event.getSource() == myTextField){
            myTextField.selectAll();
        }
    }

    // don’t forget the other listener methods (not shown)
}
```
MouseListener
The MouseListener interface has methods for mouse click and movement events

- Clicked: a full press and release of a mouse button
- Pressed: user pressed a button down but has not back up
- Released: the button has been let back up
- Entered: the mouse pointer has entered the area of the screen defined by the bounding box of the component or frame
- Exited: the mouse pointer has left the area defined by the bounding box
MouseListener

- To use a listener
  ✔ import
  ✔ implements the required listener interface
- Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements MouseListener {
    private JButton myButton;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myButton = new JButton("Click me");
        // more JButton initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
MouseListener

- To use a listener
  ✔ import
  ✔ implements the required listener interface
  ✔ Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements MouseListener {
    private JButton myButton;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myButton = new JButton("Click me");
        // more JButton initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void mousePressed (MouseEvent event) {
    }

    public void mouseClicked (MouseEvent event) {
        if (event.getSource() == myButton){
            System.out.println("You clicked the button!"游览);
        }
    }

    // more MouseListener methods here (not shown): mouseReleased, mouseEntered, mouseExited
```
To use a listener

✔ import
✔ implements the required listener interface

Write each method specified by the interface

Register the listener with the frame or component

Remember: you need all listener methods, even if you don't use them

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

public class HelloSwingWorld extends JFrame
        implements MouseListener {
    private JButton myButton;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myButton = new JButton("Click me");
        // more JButton initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void mousePressed (MouseEvent event) {
    }

    public void mouseClicked (MouseEvent event) {
        if (event.getSource() == myButton){
            System.out.println("You clicked the button!");
        }
        else {
            System.out.println("You clicked something else.");
        }
    }

    // moreMouseListener methods here (not shown):
    // mouseReleased, mouseEntered, mouseExited
}
```
MouseListener

- To use a listener
  ✔ import
  ✔ implements the required listener interface
  ✔ Write each method specified by the interface
  ✔ Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements MouseListener {
    private JButton myButton;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myButton = new JButton("Click me");
        // more JButton initialization here (not shown)
        myButton.addMouseListener(this);
        this.addMouseListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void mousePressed (MouseEvent event) {
    }

    public void mouseClicked (MouseEvent event) {
        if (event.getSource() == myButton){
            System.out.println("You clicked the button!");
        } else {
            System.out.println("You clicked something else.");
        }
    }

    // more MouseListener methods here (not shown):
    // mouseReleased, mouseEntered, mouseExited
}
```
MouseEvent

- MouseEvents have a few goodies compared to previous events
  - The mouse button that was clicked
  - The frame position of the mouse
- As before, the `getSource` method returns the memory reference of the object that caused the event to occur

```
«interface»
MouseListener

«update»
+ void mouseClicked(MouseEvent)
+ void mousePressed(MouseEvent)
+ void mouseReleased(MouseEvent)
+ void mouseEntered(MouseEvent)
+ void mouseExited(MouseEvent)
```

```
MouseEvent

«query»
+ Object getSource()
+ int getX()
+ int getY()
+ int getButton()
```
MouseEvent

- getButton returns:
  - 0 = no button
  - 1 = left button
  - 2 = middle button
  - 3 = right button

```java
public void mouseClicked (MouseEvent event) {
    if (event.getSource() == myButton) {
        System.out.println("You clicked the button! ");
    } else {
        System.out.println("You clicked something else.");
    }
    if (event.getButton() == 1) {
        System.out.println("Left");
    } else if (event.getButton() == 2) {
        System.out.println("Middle, how rude!");
    } else if (event.getButton() == 3) {
        System.out.println("Right");
    }
    System.out.println("Mouse is at (" + event.getX() + "," + event.getY() + ")");
}
```

You clicked the button!  
Left  
Mouse is at (173,48)
## Listeners and Components

The following components can register the following listeners:

<table>
<thead>
<tr>
<th>Component</th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JComboBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
KeyListener
• The KeyListener interface is very similar to the MouseListener, but handles keyboard events
  • Typed: when a key on the keyboard was fully pressed and released
  • Pressed: when a key is pressed down, but not yet let up
  • Released: when the key is let up

```java
interface KeyListener
{
    void keyTyped(KeyEvent event);
    void keyPressed(KeyEvent event);
    void keyReleased(KeyEvent event);
}
```
KeyListener

- To use a listener
  ✔ import
  ✔ implements the required listener interface
- Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame
    implements KeyListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
```
**KeyEvent**

- Similar to MouseEvent, the KeyEvent class has some methods for finding out which key was pressed
  - Key codes are integer values given to many of the keys
    - ‘a’ = 65
    - ‘b’ = 66
    - shift = 16
    - etc.
  - Can be compared with constants
    - KeyEvent.VK_SHIFT
    - KeyEvent.VK_LEFT
    - etc.

### KeyListener

<table>
<thead>
<tr>
<th>«interface»</th>
<th>KeyListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>«update»</td>
<td></td>
</tr>
<tr>
<td>+ void keyTyped(KeyEvent)</td>
<td></td>
</tr>
<tr>
<td>+ void keyPressed(KeyEvent)</td>
<td></td>
</tr>
<tr>
<td>+ void keyReleased(KeyEvent)</td>
<td></td>
</tr>
</tbody>
</table>

### KeyEvent

<table>
<thead>
<tr>
<th>«query»</th>
<th>KeyEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Object getSource()</td>
<td></td>
</tr>
<tr>
<td>+ char getKeyChar()</td>
<td></td>
</tr>
<tr>
<td>+ int getKeyCode()</td>
<td></td>
</tr>
</tbody>
</table>
KeyListener

- **To use a listener**
  - ✔ import
  - ✔ implements the required listener interface
  - ✔ Write each method specified by the interface

- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements KeyListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void keyPressed (KeyEvent event) {
        if (event.getKeyCode() == KeyEvent.VK_SHIFT){
            String text = myTextField.getText();
            text = text.toUpperCase();
            myTextField.setText(text);
        }
    }

    // don’t forget the other KeyListener methods (not shown)
    // keyTyped and keyReleased
}
```
To use a listener

- **import**
- **implements** the required listener interface
- **Write** each method specified by the interface
- **Register** the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements KeyListener {
    private JTextField myTextField;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myTextField = new JTextField("Initial text");
        // more JTextField initialization here (not shown)
        myTextField.addKeyListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void keyPressed (KeyEvent event) {
        if (event.getKeyCode() == KeyEvent.VK_SHIFT){
            String text = myTextField.getText();
            text = text.toUpperCase();
            myTextField.setText(text);
        }
    }

    // don’t forget the other KeyListener methods (not shown)
    // keyTyped and keyReleased
}
```
KeyListener Interface

- Not all keys will cause all types of KeyListener events
  - For example, typing the shift key alone does not cause keyTyped to get called
    - Others: Ctrl, Alt, and arrow keys
    - You will need to use keyPressed and keyReleased instead
  - Key combinations do cause keyTyped to get called, however
    - Ctrl-c
    - Shift-x
    - etc.

```java
interface KeyListener {
    void keyTyped(KeyEvent);
    void keyPressed(KeyEvent);
    void keyReleased(KeyEvent);
}
```
KeyListener Interface

• Not all keys will cause all types of KeyListener events
  • For example, typing the shift key alone does not cause keyTyped to get called
    • Others: Ctrl, Alt, and arrow keys
    • You will need to use keyPressed and keyReleased instead
  • Key combinations do cause keyTyped to get called, however
    • Ctrl-c
    • Shift-x
    • etc.

```java
interface KeyListener
{
    void keyTyped(KeyEvent e);
    void keyPressed(KeyEvent e);
    void keyReleased(KeyEvent e);
}
```

GOTCHA
Listeners and Components

- The following components can register the following listeners

<table>
<thead>
<tr>
<th></th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JComboBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
ActionListener
ActionListener Interface

- Sometimes it is a bit tedious to define all of the possible behavior for a given component
- The ActionListener interface may have different semantics for each component, but it usually “does what you want” for each component
  - Think of it as a generic listener that is called when you interact with a component in usual ways
  - Consider JButtons:
    - Normally you can click a button -or- give it focus and press the space bar

```java
// ActionListener interface definition

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

- To use a listener
  ✔ import
  ✔ implements the required listener interface
- Write each method specified by the interface
- Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements ActionListener {
    private JCheckBox myCheckBox;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myCheckBox = new JCheckBox("Option");
        // more JCheckBox initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
}
```
To use a listener

✔ import

✔ implements the required listener interface

✔ Write each method specified by the interface

• Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame
    implements ActionListener {
    private JCheckBox myCheckBox;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myCheckBox = new JCheckBox("Option");
        // more JCheckBox initialization here (not shown)
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == myCheckBox){
            if (myCheckBox.isSelected()){
                System.out.println("Option enabled!");
            }
            else {
                System.out.println("Option disabled!");
            }
        }
    }
}
```
ActionListener

- To use a listener
  - ✔ import
  - ✔ implements the required listener interface
  - ✔ Write each method specified by the interface
  - ✔ Register the listener with the frame or component

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

public class HelloSwingWorld extends JFrame implements ActionListener {
    private JCheckBox myCheckBox;

    public HelloSwingWorld () {
        // JFrame initialization here (not shown)
        myCheckBox = new JCheckBox("Option");
        // more JCheckBox initialization here (not shown)
        myCheckBox.addActionListener(this);
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == myCheckBox){
            if (myCheckBox.isSelected()){
                System.out.println("Option enabled!");
            }
            else {
                System.out.println("Option disabled!");
            }
        }
    }
}
```
• No especially interesting KeyEvents, just the usual getSource

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

```java
class ActionEvent
{
    Object getSource();
}
```
**Listeners and Components**

- The following components can register the following listeners:

<table>
<thead>
<tr>
<th>Component</th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JComboBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
## Listeners and Components

- The following components can register the following listeners

<table>
<thead>
<tr>
<th>Component</th>
<th>WindowListener</th>
<th>FocusListener</th>
<th>MouseListener</th>
<th>KeyListener</th>
<th>ActionListener</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFrame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JLabel</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JTextField</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JCheckBox</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JRadioButton</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JComboBox</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>JTextArea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Soooo… what’s this all about?
Swing Timer
Timer

- Swing Timers are used to execute an ActionEvent at periodic time intervals
- It is not a JComponent, so it does not need to be added to the frame to be used
  - However, it is imported with
    ```java
    import javax.swing.*
    ```
- Not the only Timer in Java
  - Another: `import java.util.Timer`
  - Do not use!
• The integer passed to both the constructor and to the setDelay method is the number of milliseconds between its ActionEvents
• Hint, there are 1,000 milliseconds in a second
**Timer**

- When the timer is over, it causes an ActionEvent
- Then starts the timer over automatically

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

public class HelloSwingWorld extends JFrame implements ActionListener {
    private Timer timer;
    private JButton speedUp;
    private int delay;
    public HelloSwingWorld () {
        // JFrame and JButton init here (not shown)
        delay = 1000;
        timer = new Timer(delay, this);
        timer.start();
    }
    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }
    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == timer) {
            System.out.println("Time up!");
        } else if (event.getSource() == speedUp) {
            delay -= 100;
            timer.setDelay(delay);
        }
    }
}
```
## Timer

- When the timer is over, it causes an `ActionEvent`
- Then starts the timer over automatically

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

public class HelloSwingWorld extends JFrame implements ActionListener {
    private Timer timer;
    private JButton speedUp;
    private int delay;
    public HelloSwingWorld () {
        // JFrame and JButton init here (not shown)
        delay = 1000;
        timer = new Timer(delay, this);
        timer.start();
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == timer){
            System.out.println("Time up!");
        }
        else if (event.getSource() == speedUp){
            delay -= 100;
            timer.setDelay(delay);
        }
    }
}
```

Don't forget to register the `ActionListener` with the `JButton`
Timer

- When the timer is over, it causes an ActionEvent
- Then starts the timer over

No need to add the timer to the JFrame

---

<table>
<thead>
<tr>
<th>Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>«constructor»</strong></td>
</tr>
<tr>
<td>+ Timer(int,ActionListener)</td>
</tr>
<tr>
<td><strong>«update»</strong></td>
</tr>
<tr>
<td>+ void start()</td>
</tr>
<tr>
<td>+ void stop()</td>
</tr>
<tr>
<td>+ void setDelay(int)</td>
</tr>
<tr>
<td><strong>«query»</strong></td>
</tr>
<tr>
<td>+ boolean isRunning()</td>
</tr>
</tbody>
</table>

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

class HelloSwingWorld extends JFrame implements ActionListener {
    private Timer timer;
    private JButton speedUp;
    private int delay;

    public HelloSwingWorld () {
        // JFrame and JButton init here (not shown)
        delay = 1000;
        timer = new Timer(delay, this);
        timer.start();
    }

    public static void main (String[] args) {
        JFrame frame = new HelloSwingWorld();
        frame.setVisible(true);
    }

    public void actionPerformed (ActionEvent event) {
        if (event.getSource() == timer){
            System.out.println("Time up!");
        } else if (event.getSource() == speedUp){
            delay -= 100;
            timer.setDelay(delay);
        }
    }
}
```