CS 120: Software Design I

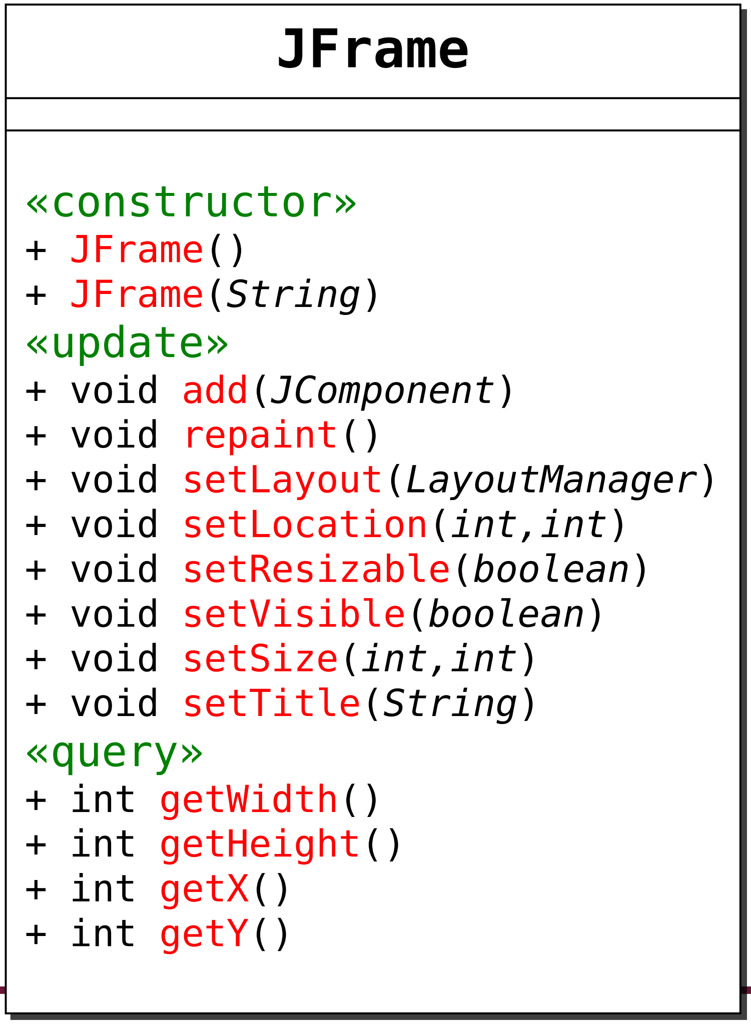
Lecture 13-2:
GUI/Event Programming
JComponent

UNIVERSITY of WISCONSIN LA CROSSE™

Professor Elliott Forbes
eforbes@uwlax.edu

JFrame Class Diagram

- **This class diagram shows only a small subset of the methods that you will gain when you extend JFrame**



```
JFrame
------------------------------
------------------------------
«constructor»
+ JFrame()
+ JFrame(String)
«update»
+ void add(JComponent)
+ void repaint()
+ void setLayout(LayoutManager)
+ void setLocation(int,int)
+ void setResizable(boolean)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setTitle(String)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
```

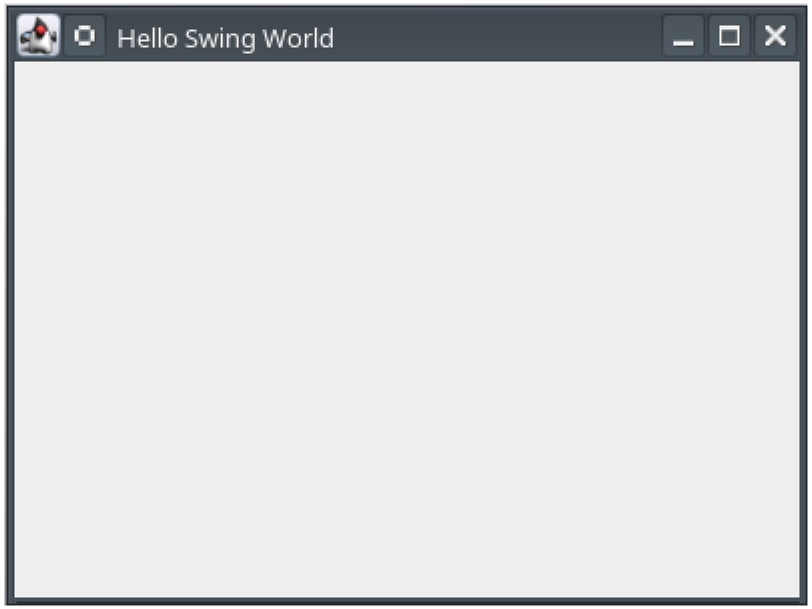
Displaying Your JFrame

- ✔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!

```
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);
  }

}
```

JFrame

- **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.



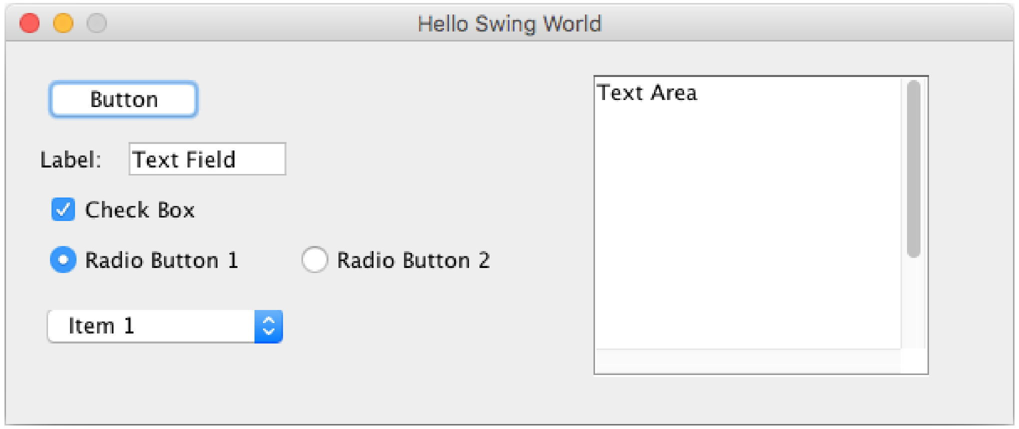
JComponent

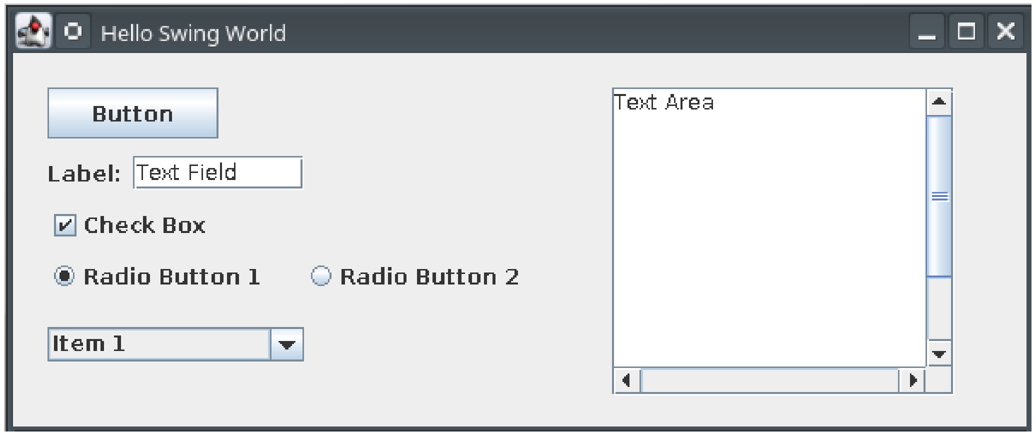
- **It is exciting to display a window, but in order to interact with the user, we need some components in the frame**
  - Components are things like buttons, text fields, labels, scroll bars, radio buttons, check boxes, drop-down lists, etc.
  - There are many available components, each is its own class
  - 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

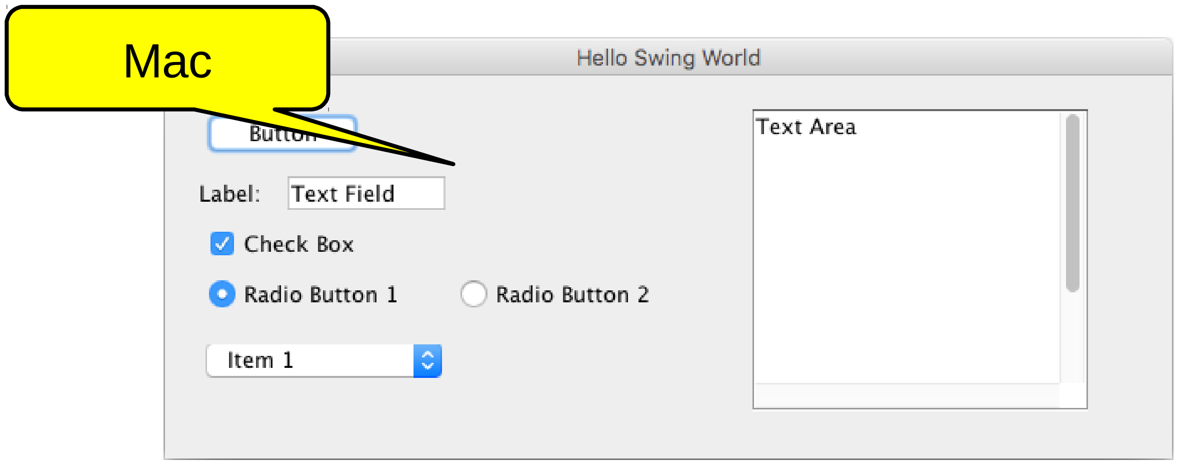
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

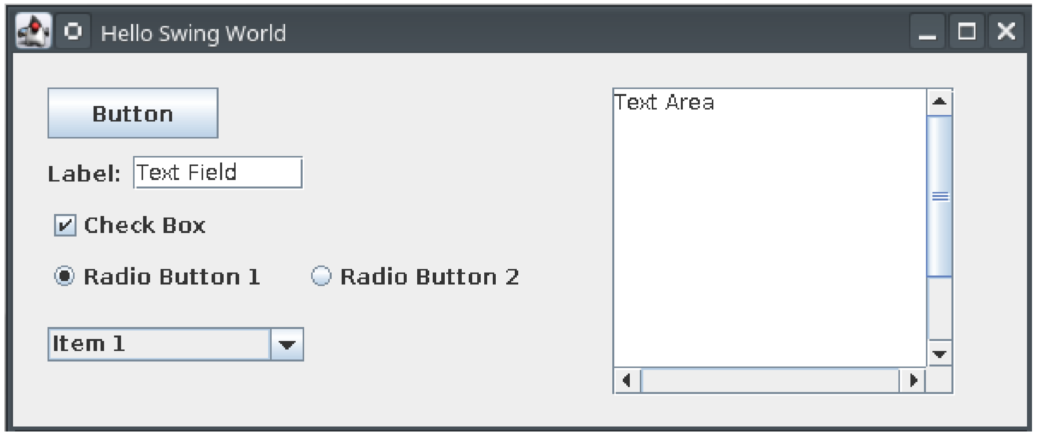




JComponent

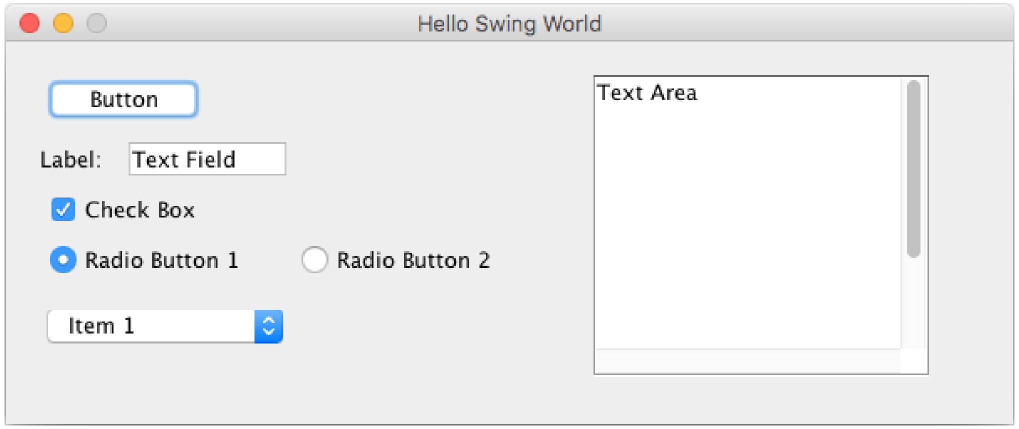
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

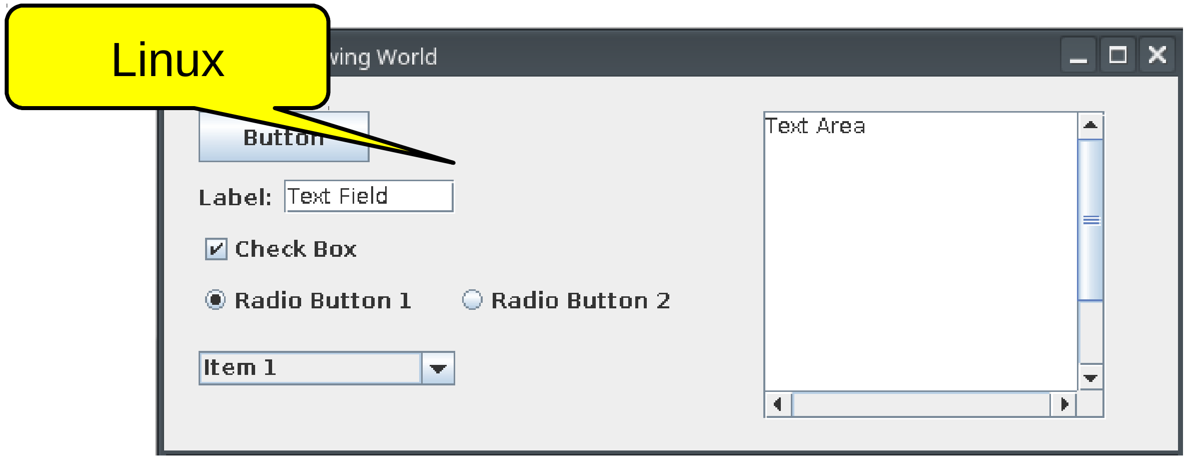




JComponent

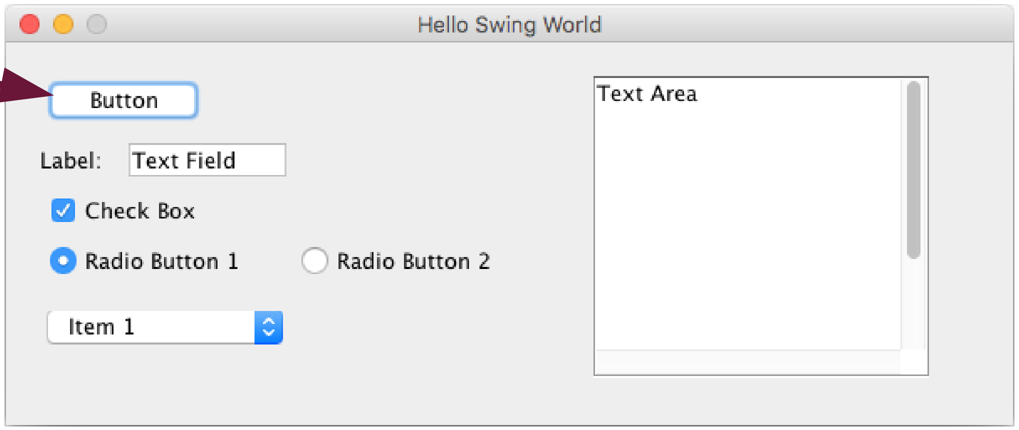
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

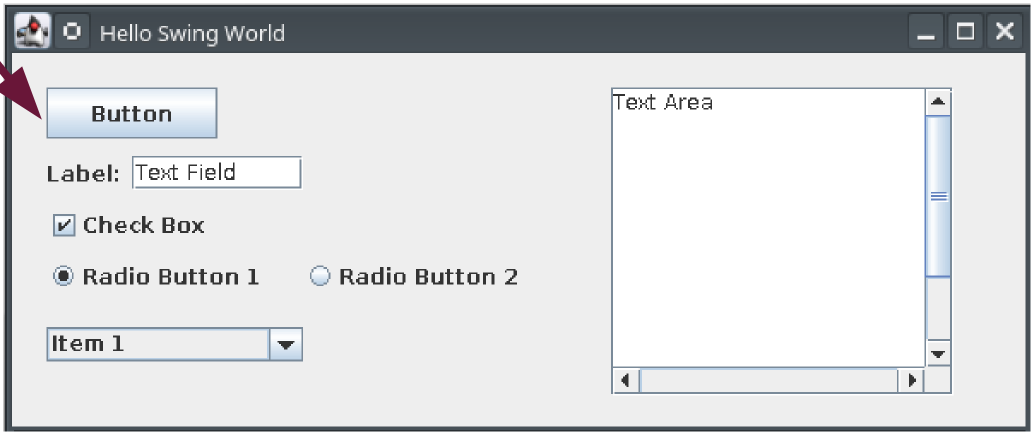




JComponent

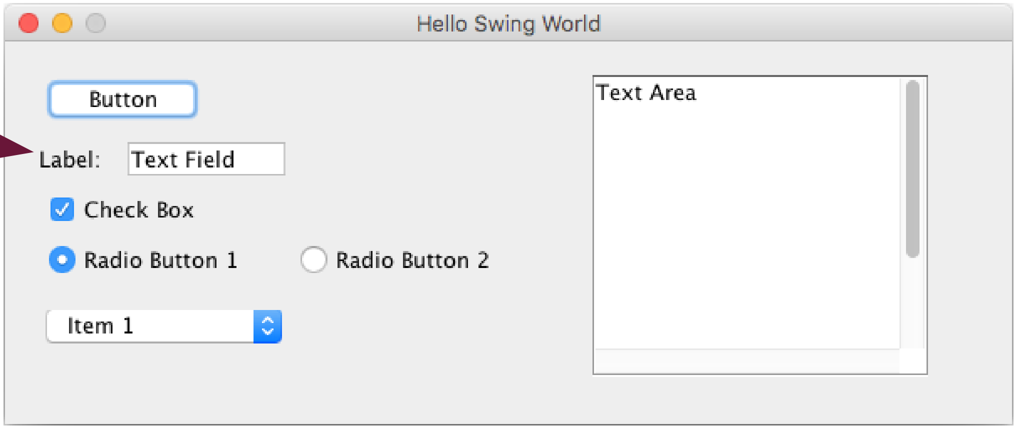
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

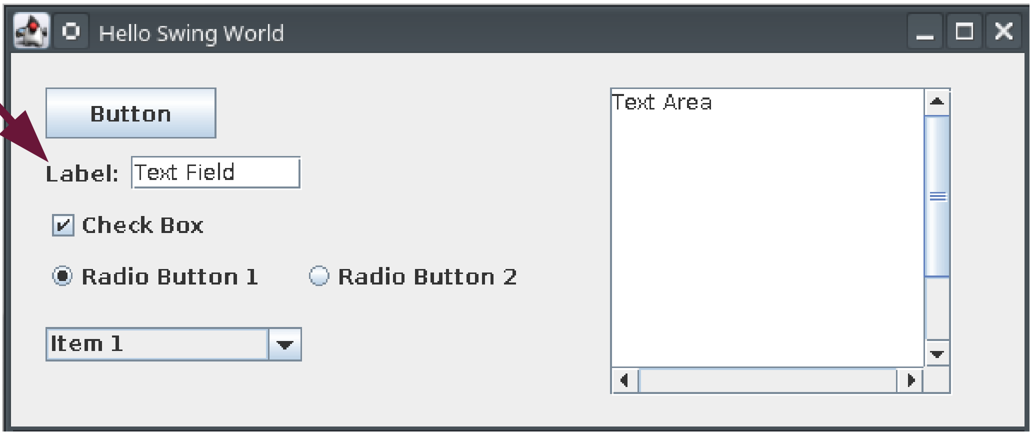




JComponent

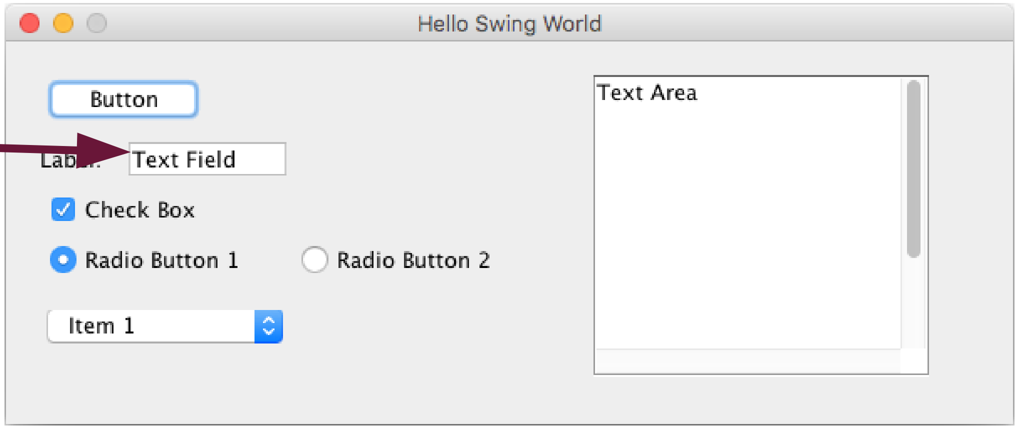
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

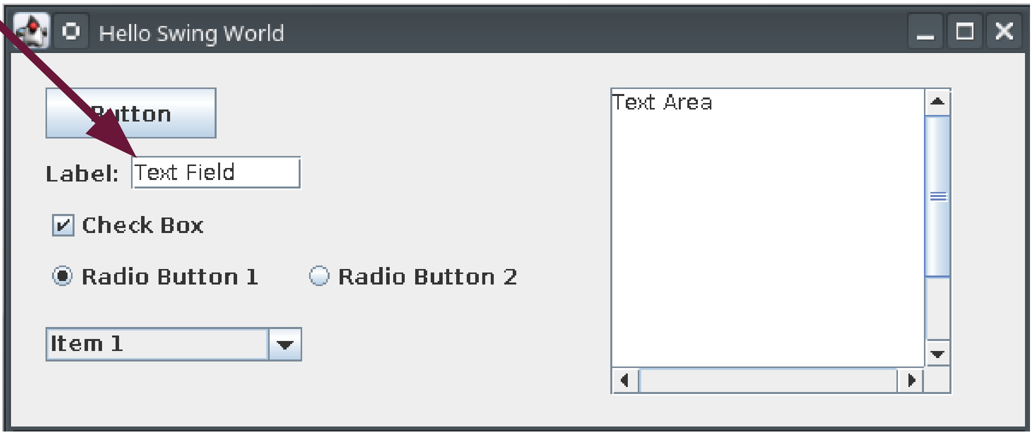




JComponent

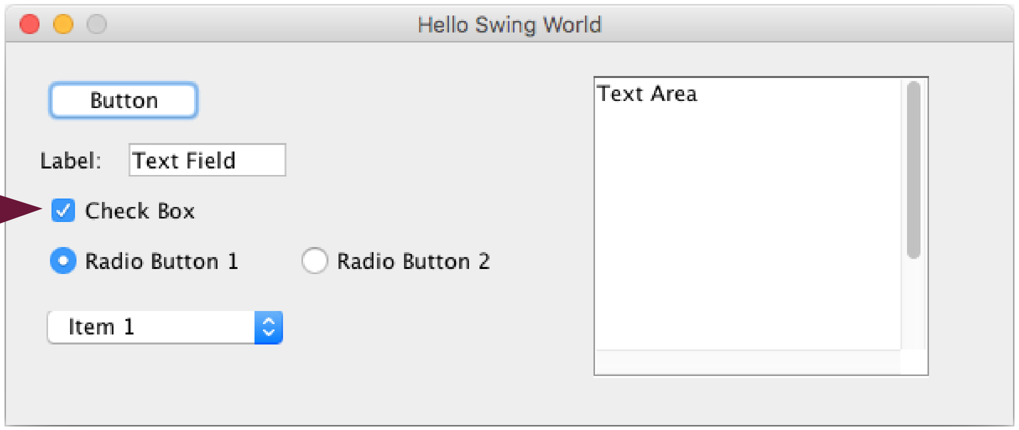
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

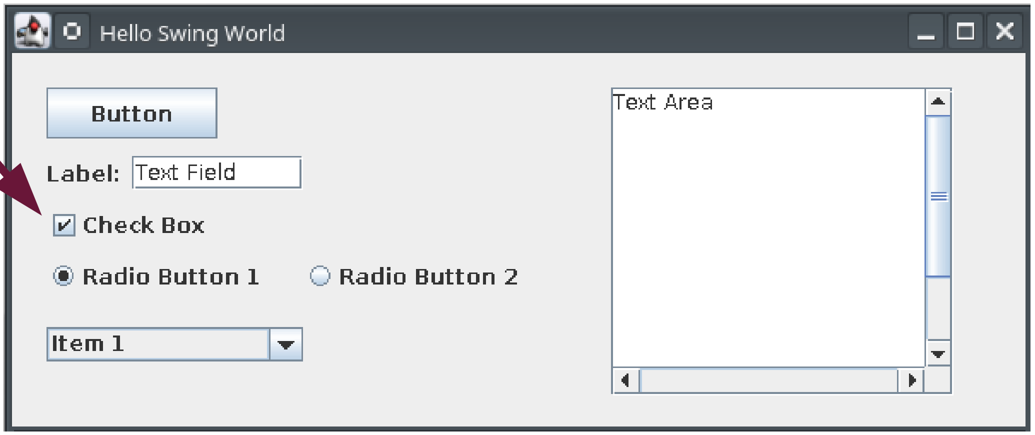




JComponent

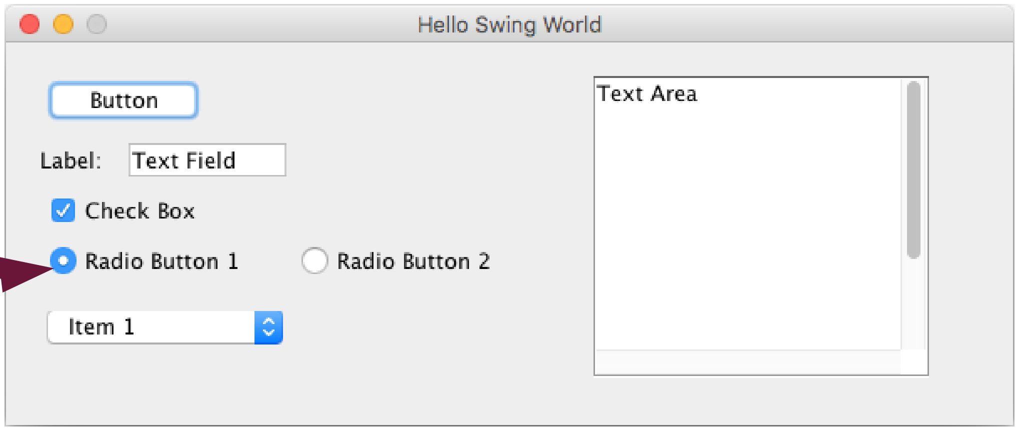
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

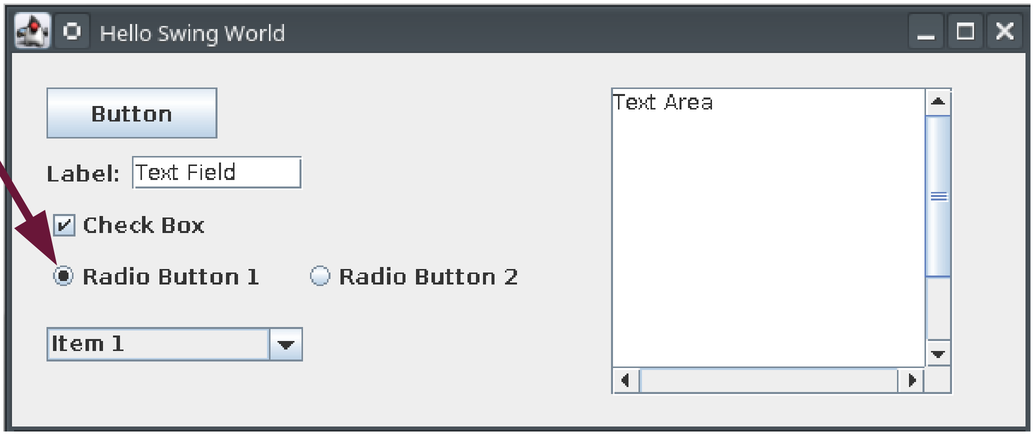




JComponent

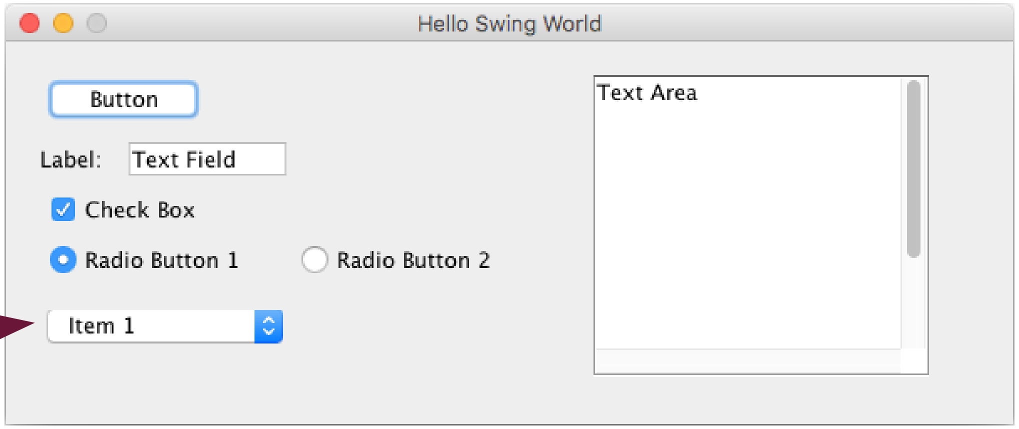
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

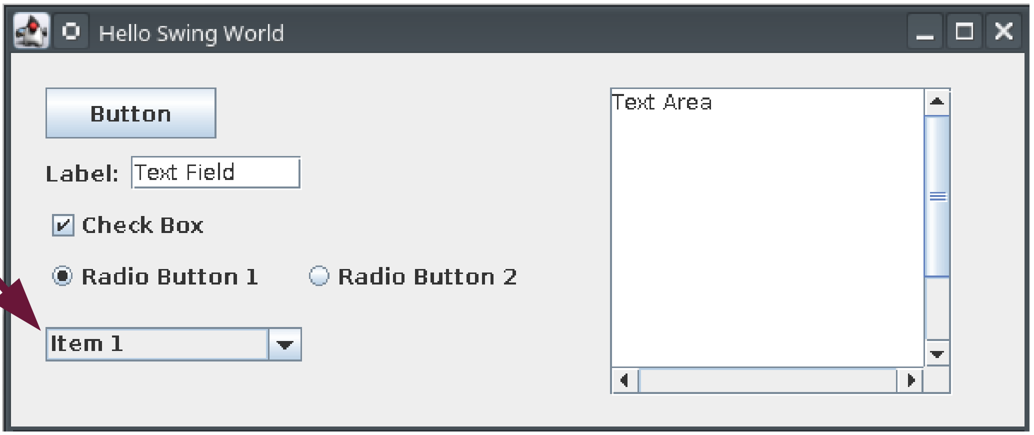




JComponent

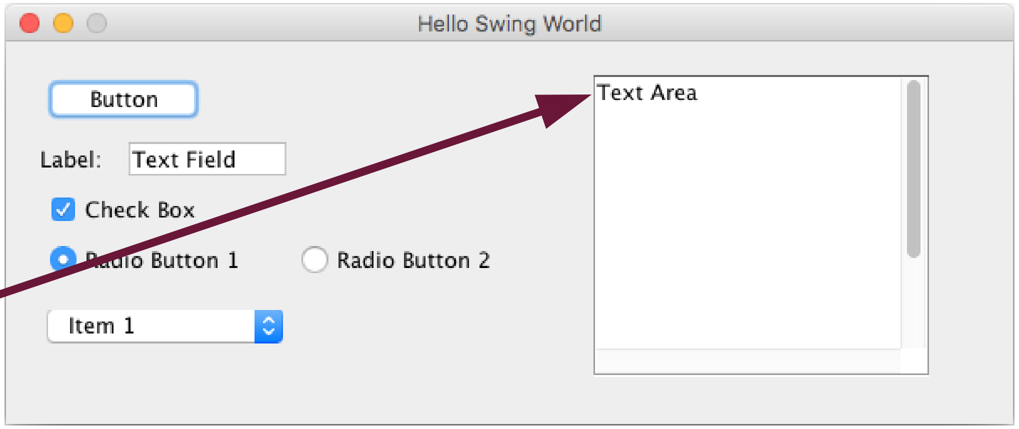
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

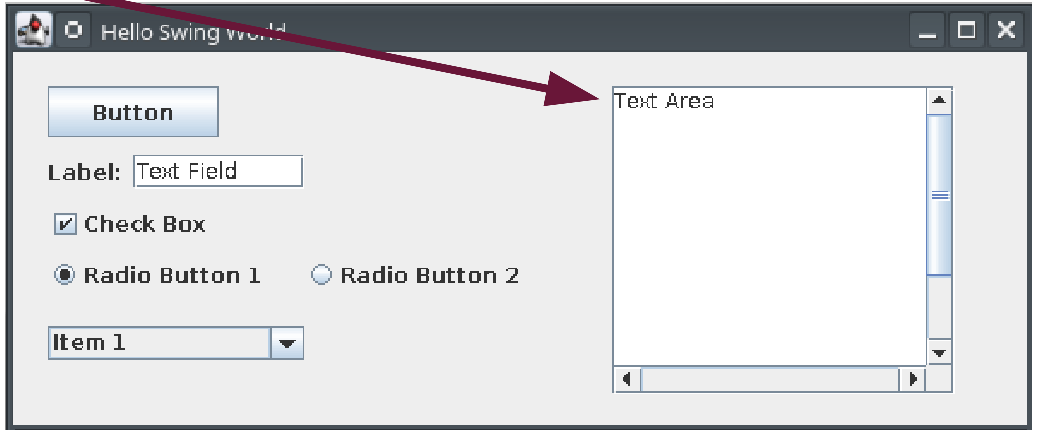




JComponent

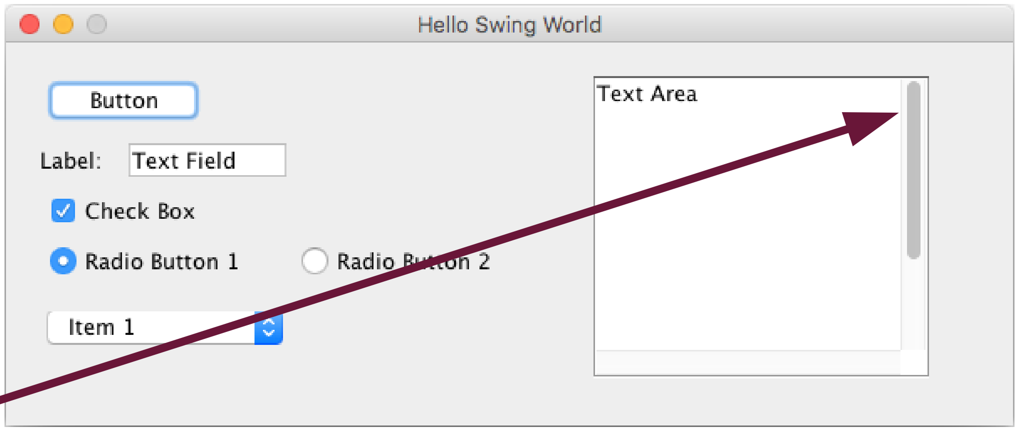
- JButton
- JLabel
- JTextField
- JCheckBox
- JRadioButton (x2)
- JComboBox
- JTextArea
- JScrollPane

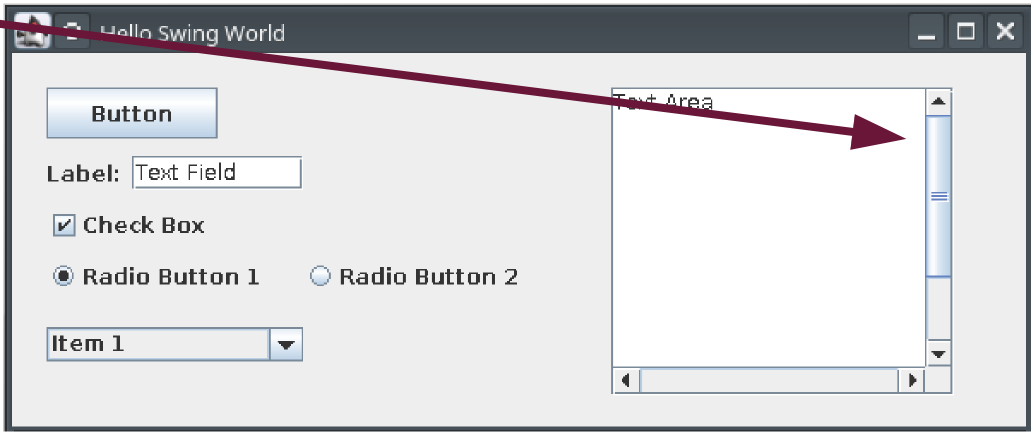




JComponent

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





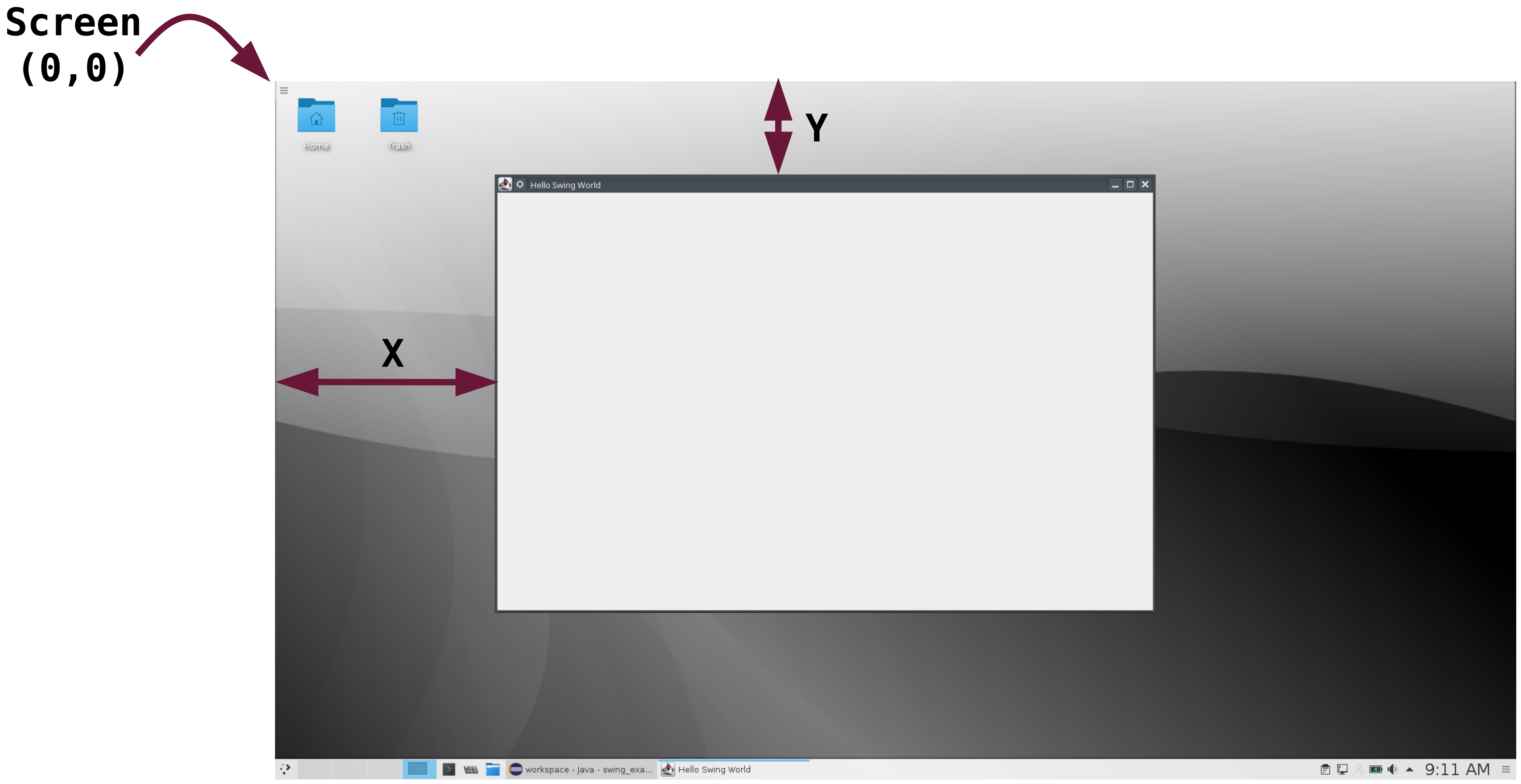
JComponent Methods

- **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.

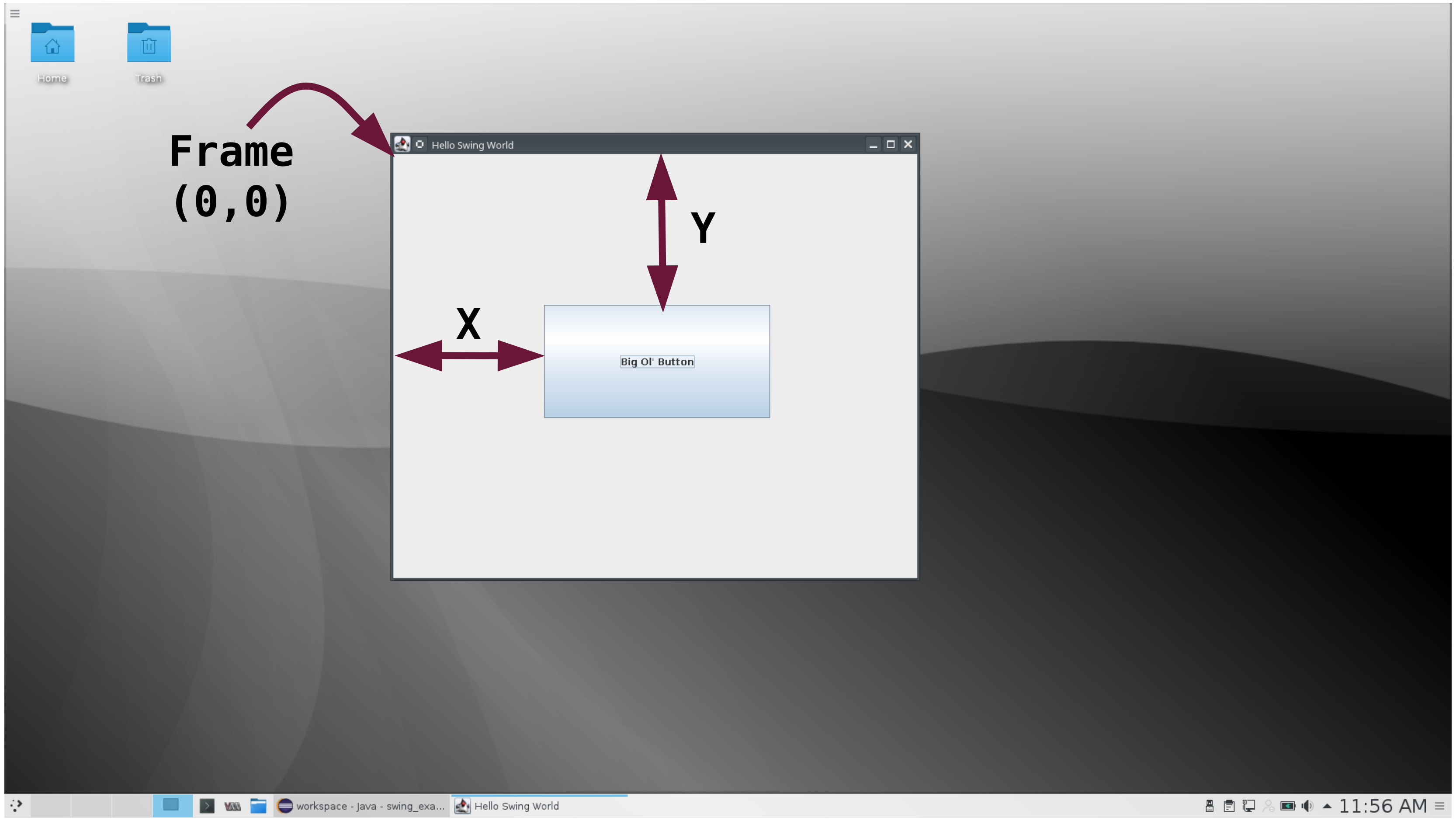
Absolute Positioning

- **The JFrame position is relative to the screen origin**



Absolute Positioning

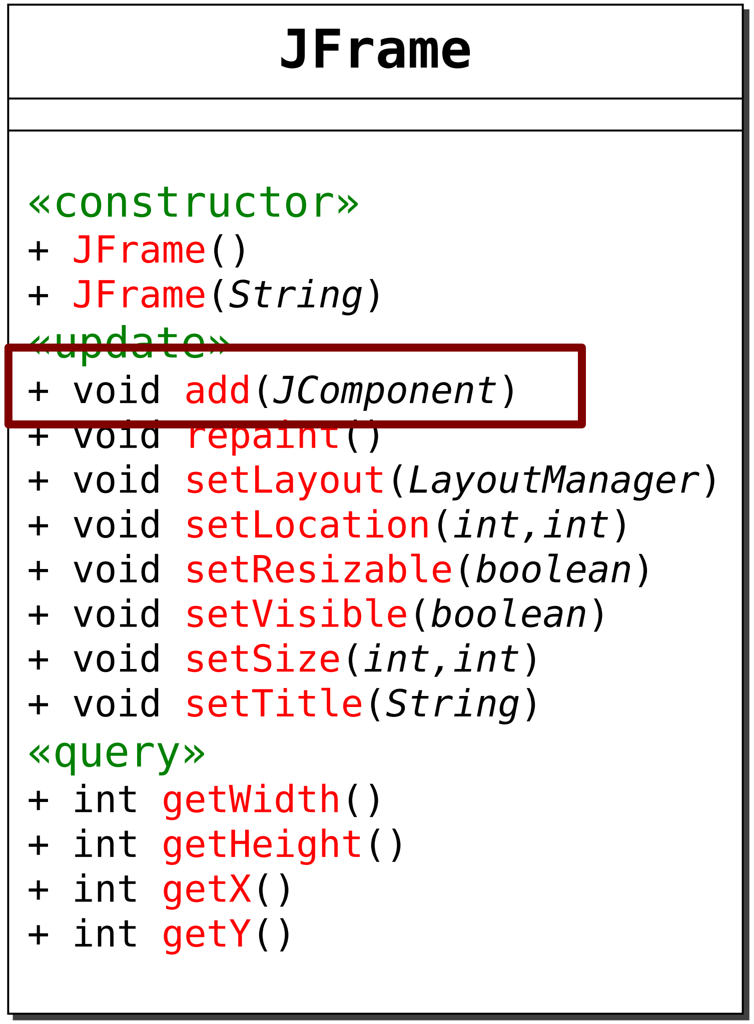
- **The position of any JComponent is relative to the frame origin**



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



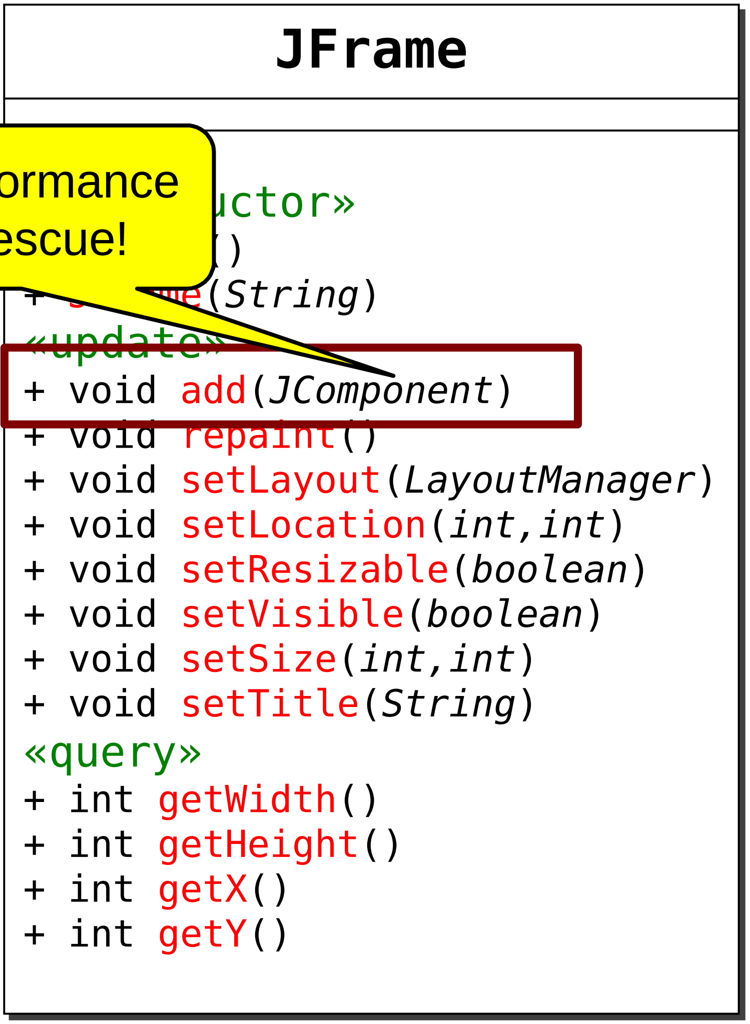
```
JFrame

«constructor»
+ JFrame()
+ JFrame(String)
«update»
+ void add(JComponent)
+ void repaint()
+ void setLayout(LayoutManager)
+ void setLocation(int,int)
+ void setResizable(boolean)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setTitle(String)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
```

JComponent Size and Position

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

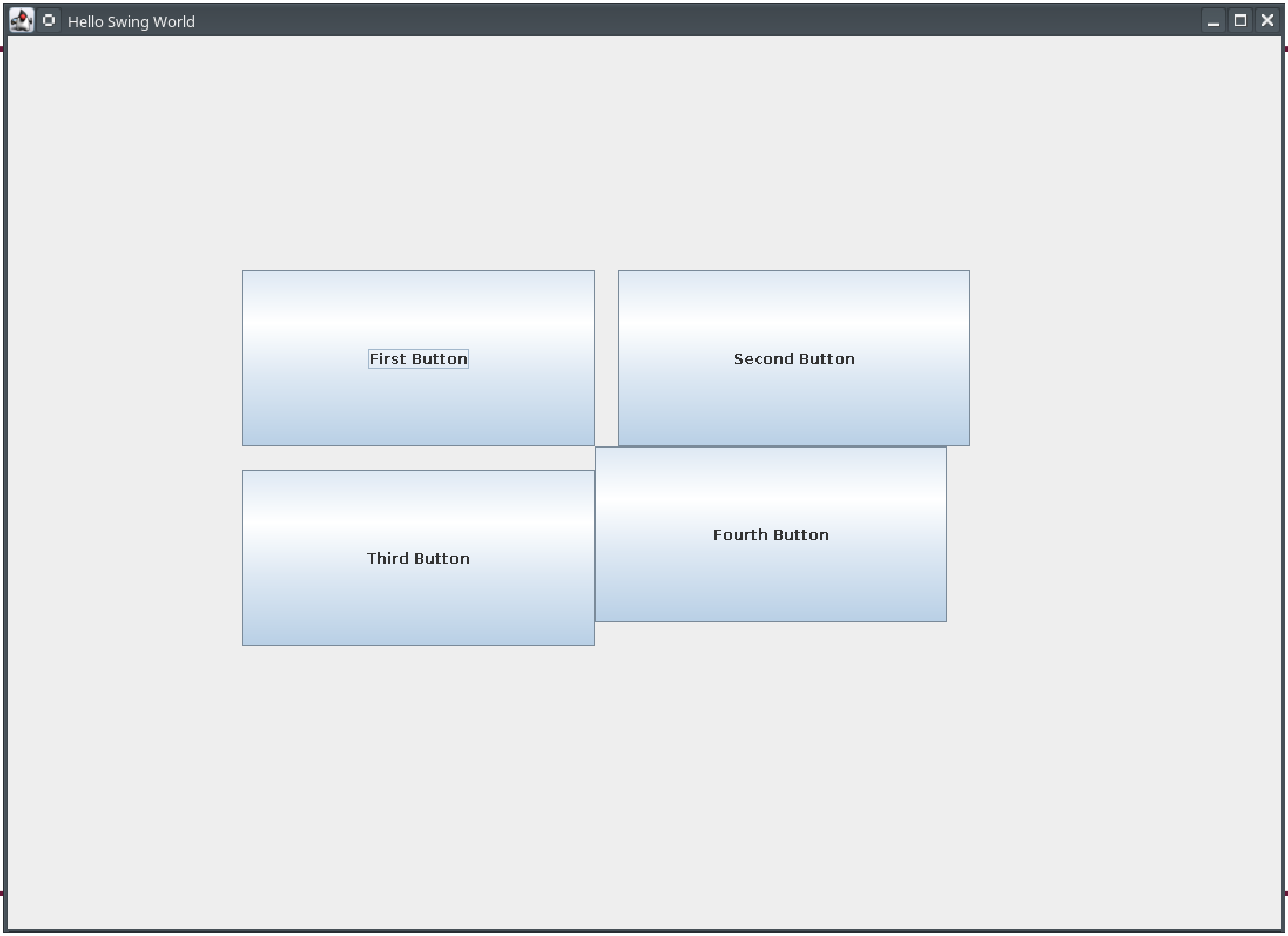
- **For any components you want on the frame:**
  - Instantiate the component
  - Set it’s size
  - Set it’s location
  - Use the frame `add()` method to place the component within the frame

Type conformance to the rescue!

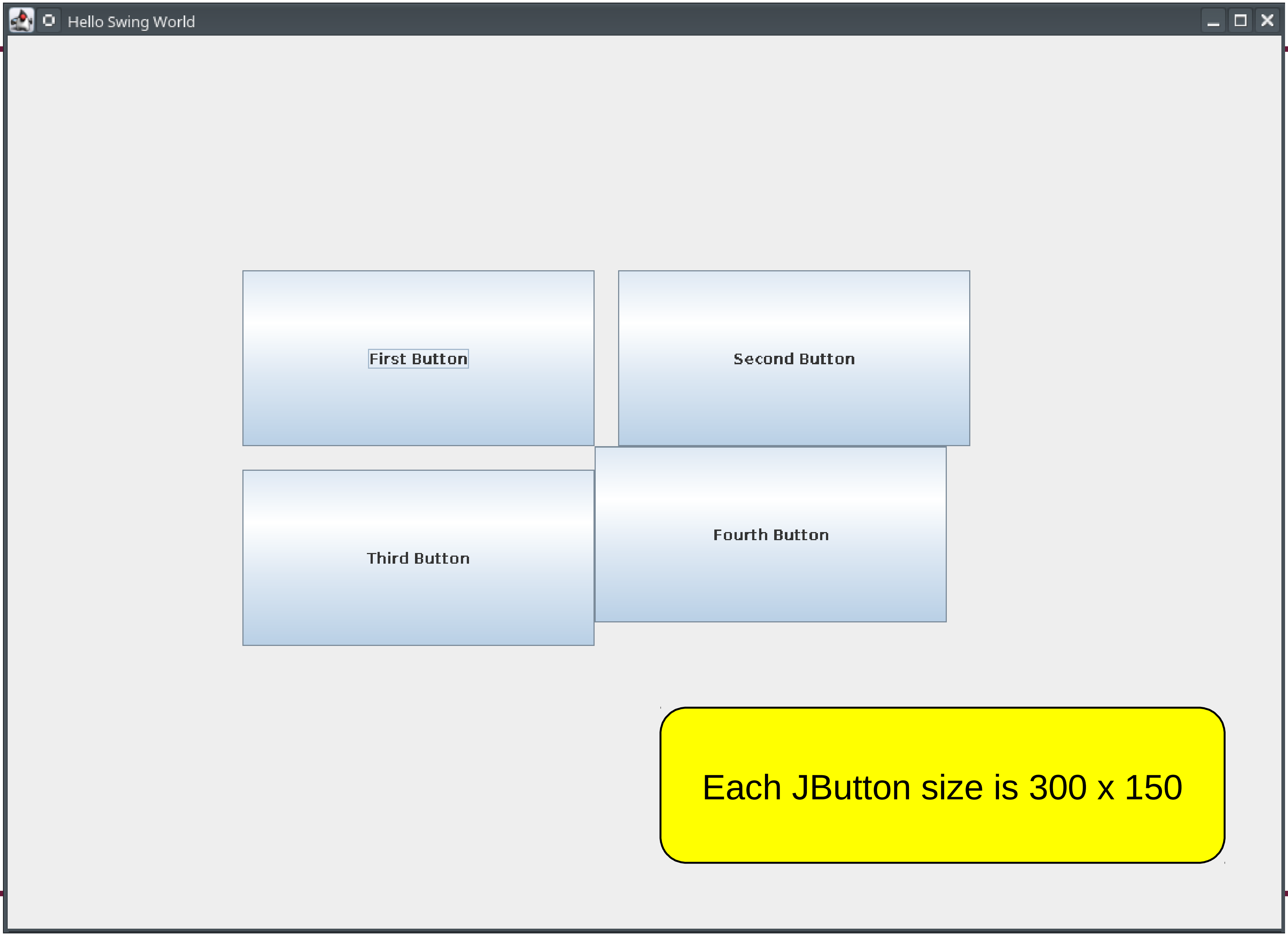


```
JFrame

uctor»
()
+ me(String)
«update»
+ void add(JComponent)
+ void repaint()
+ void setLayout(LayoutManager)
+ void setLocation(int,int)
+ void setResizable(boolean)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setTitle(String)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
```

Absolute Positioning

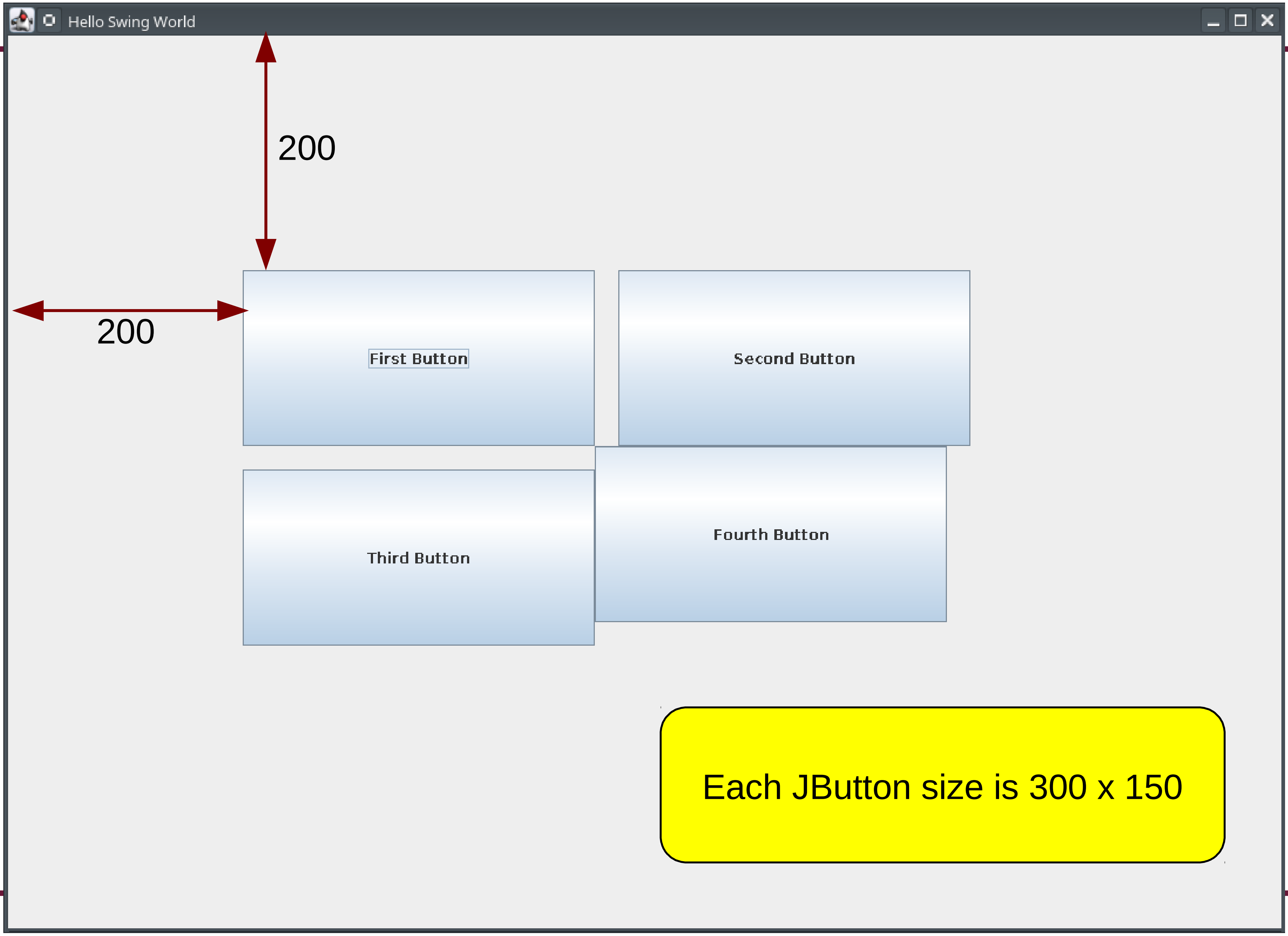


Absolute Positioning



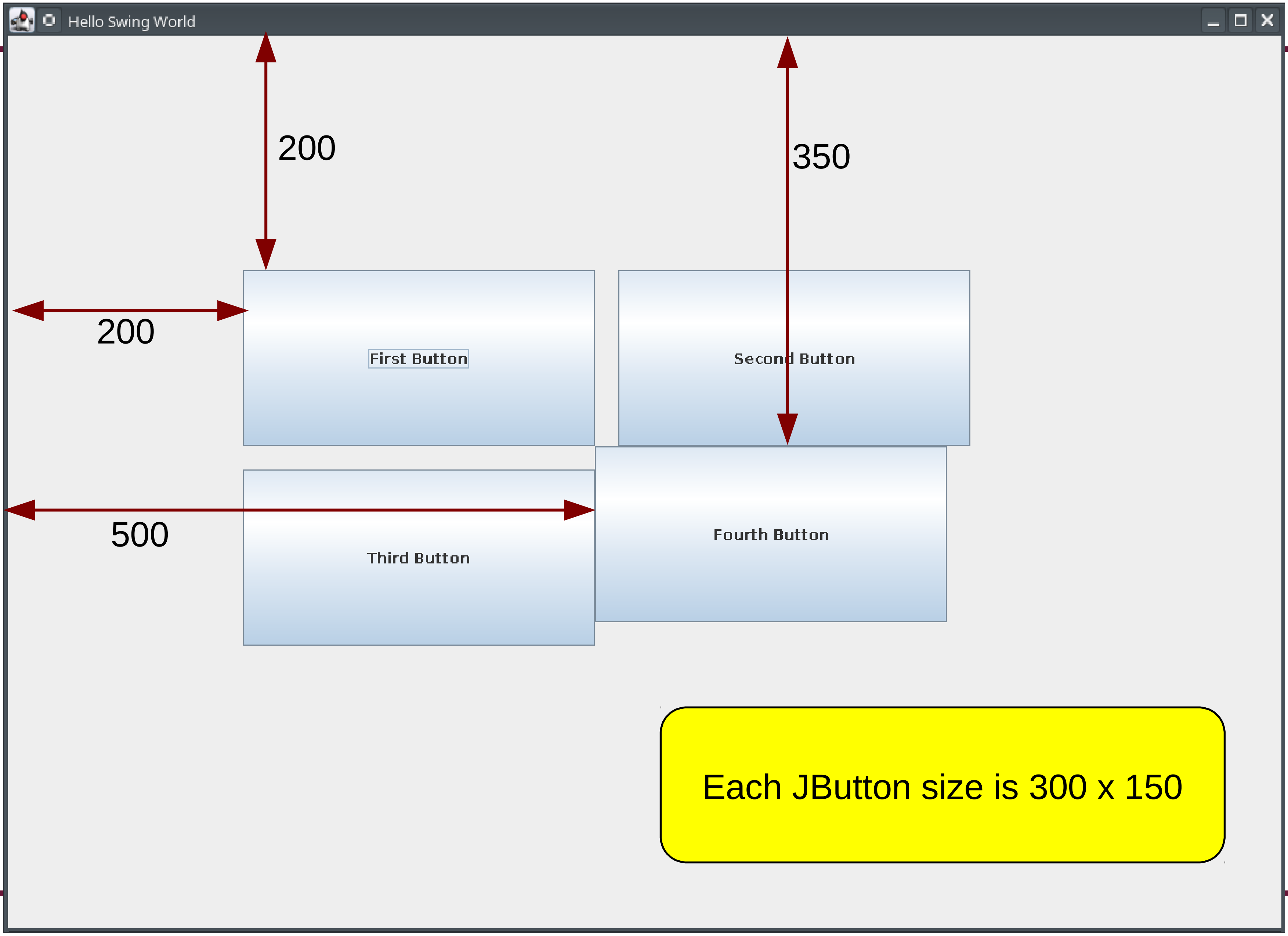
Each JButton size is 300 x 150

Absolute Positioning



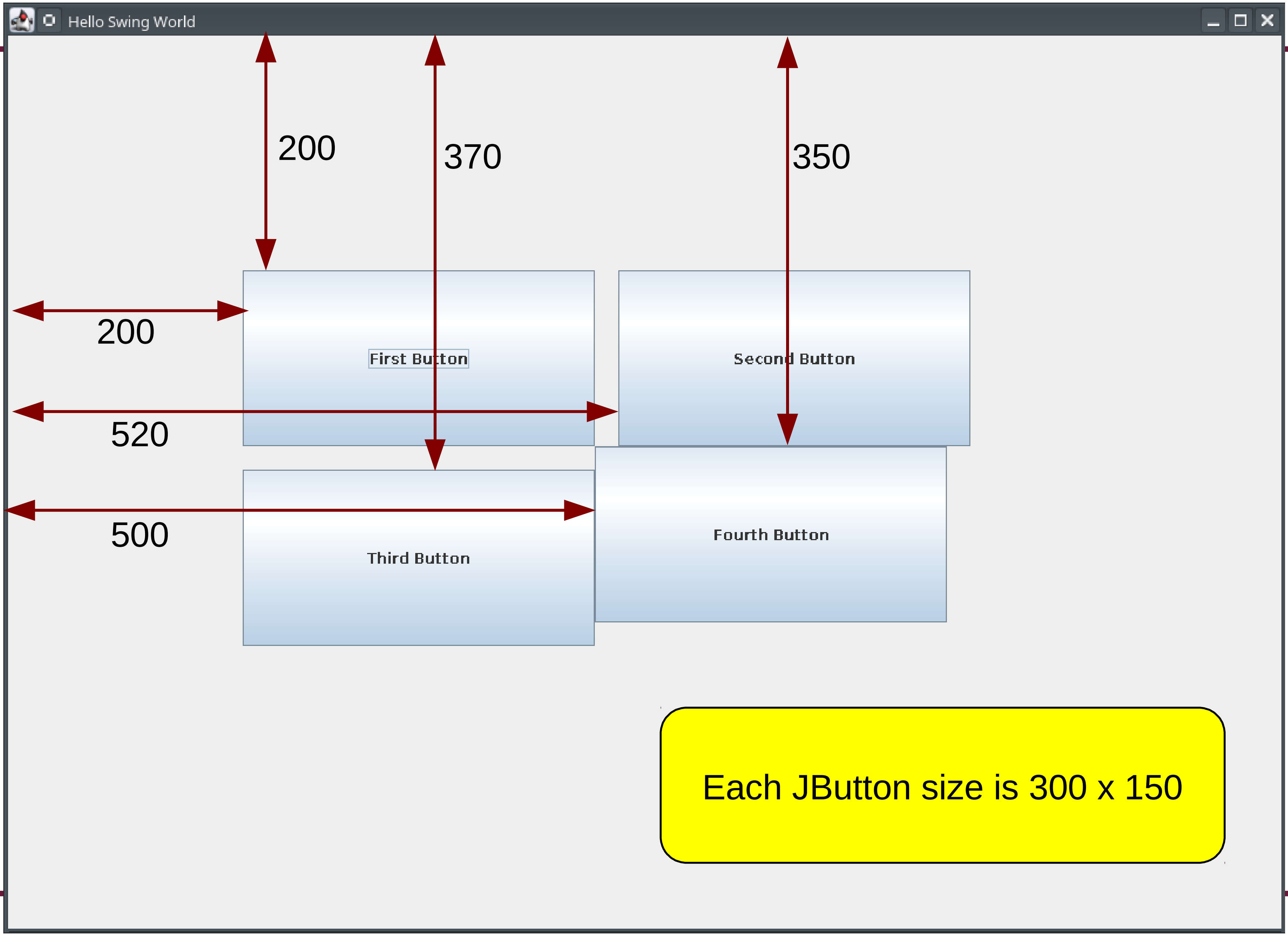
Each JButton size is 300 x 150

Absolute Positioning

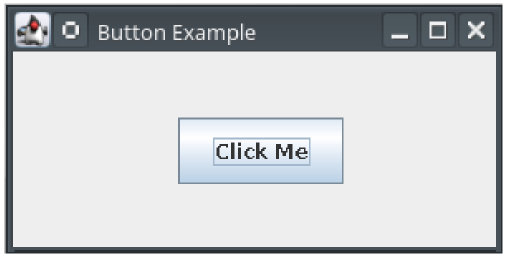


Each JButton size is 300 x 150

Absolute Positioning



Each JButton size is 300 x 150



JButton

JButton

- **These are the methods you are most likely to use with JButtons**
- **The full list of JButton methods is huge!**

```
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
-------------------------------------
-------------------------------------
«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 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);
  }
}
```

```
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()
```

Add the JButton to the Frame

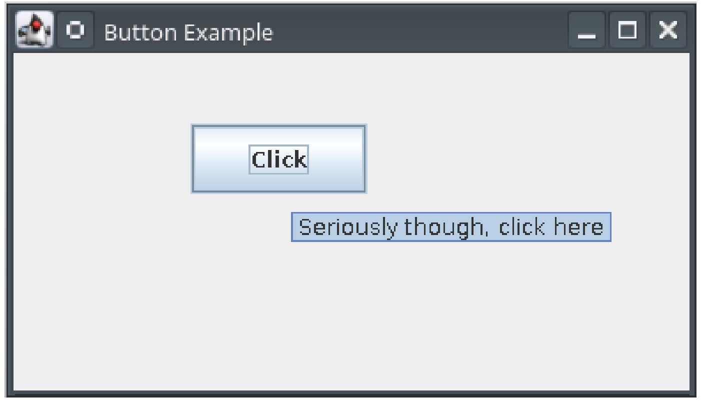
- **You can now add the JButton to the frame**
  - JButton is a child of JComponent, and the JFrame add method can place any JComponent within the frame

```
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);
  }
}
```

```
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()
```

Other JButton Methods

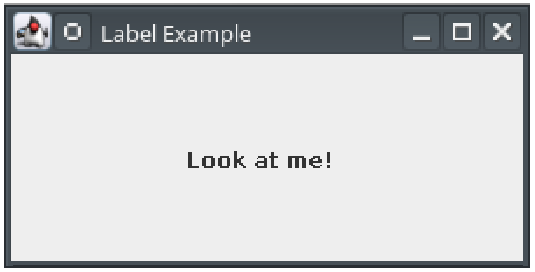
- **Tool Tips are the little boxes of text that appear when you hover your mouse pointer over a component**



```
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()
```

Full Program with JButton

```
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);
  }
}
```



Look at me!

JLabel

JLabel

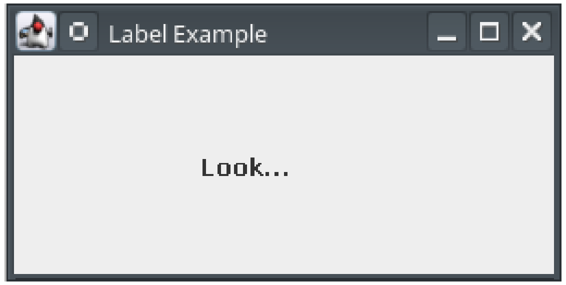
- **Sometimes you need to add some non-editable text to the frame**
  - For example, some JComponents don't already have text included (like JTextField)
  - Or you may want to label a group of components

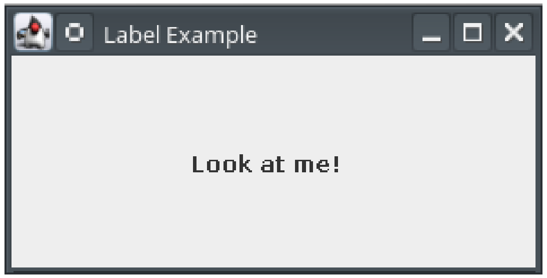
- **JLabels serve a different purpose than JButtons, but many methods overlap**

```
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 Size and Location

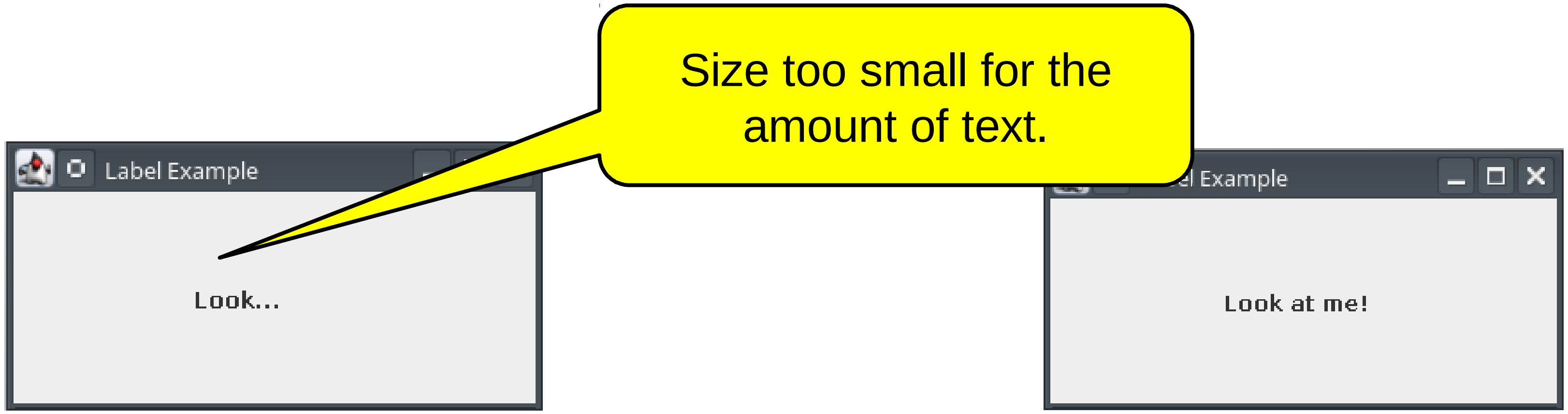
- **The size and location are measured in pixels, just like buttons**
  - Changing the size does not change the font size
- **The dimensions refer to an imaginary bounding box around the parameter of the label**





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**

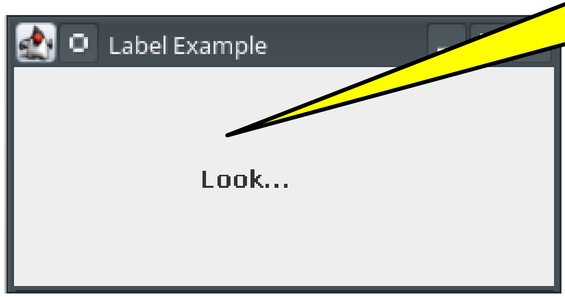


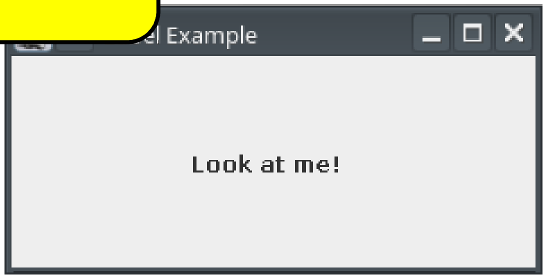
JLabel Size and Location

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

GOTCHA

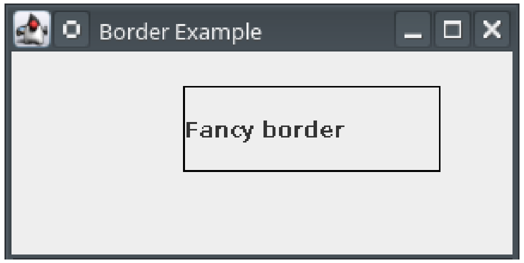
Size too small for the amount of text.

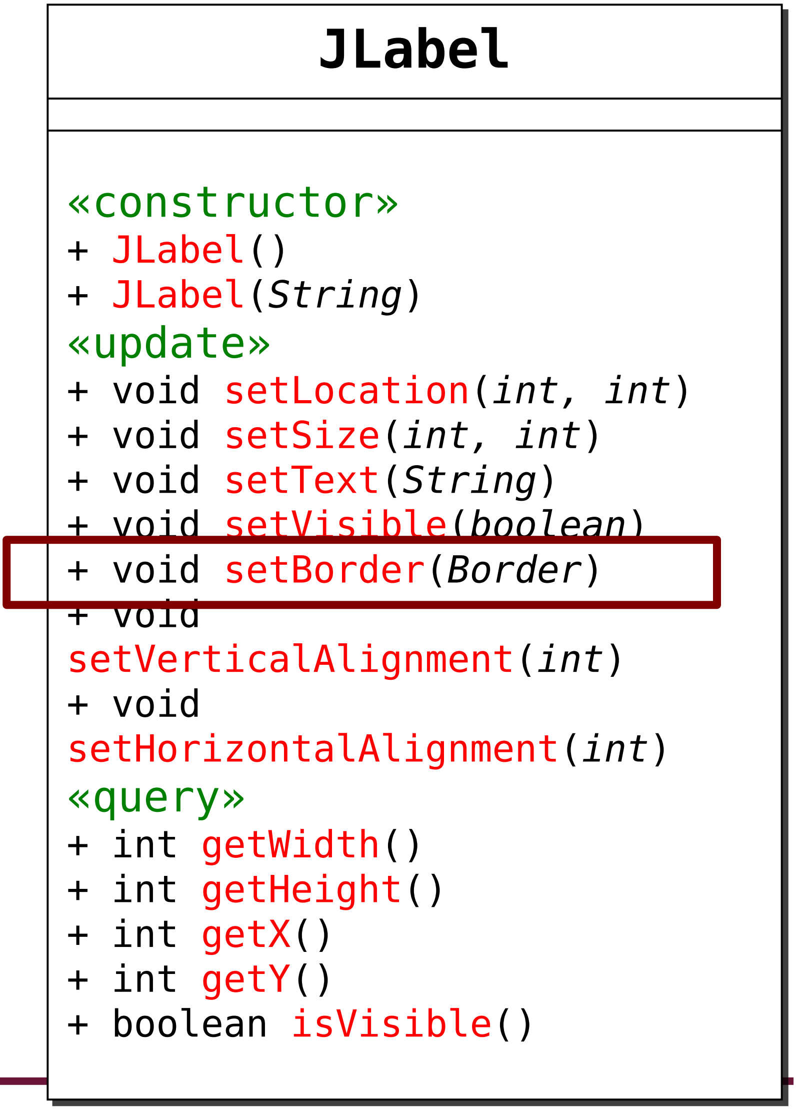




JLabel Borders

- **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!**

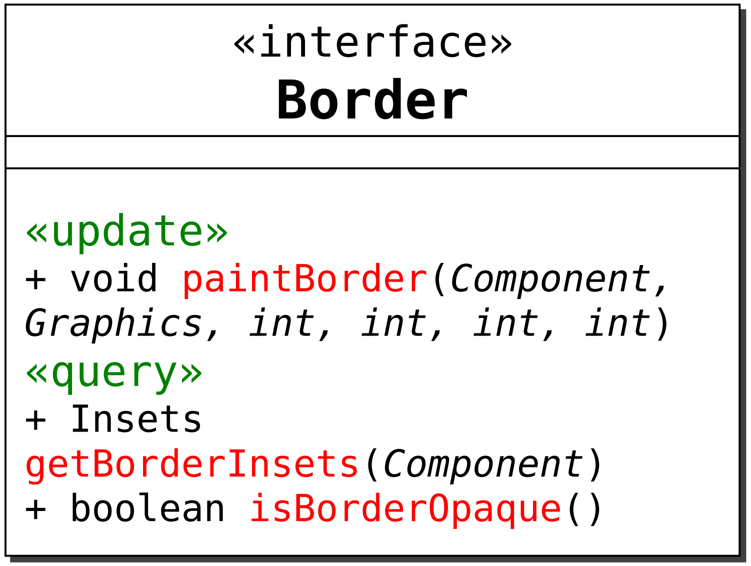




```
JLabel

«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

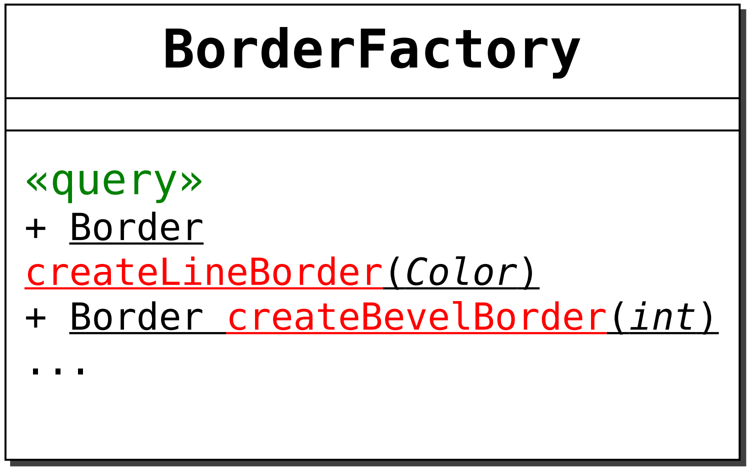
- **Notice that** `setBorder` **has a** `Border` **as the parameter**
- `Border` **is an interface**
  - Requires that you create a class that implements several methods
  - Another example of type conformance
- **Fortunately, someone has already done the heavy lifting...**



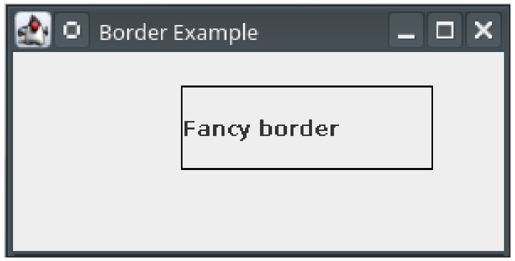
```
«interface»
Border

«update»
+ void paintBorder(Component, Graphics, int, int, int, int)
«query»
+ Insets getBorderInsets(Component)
+ boolean isBorderOpaque()
```

BorderFactory Class

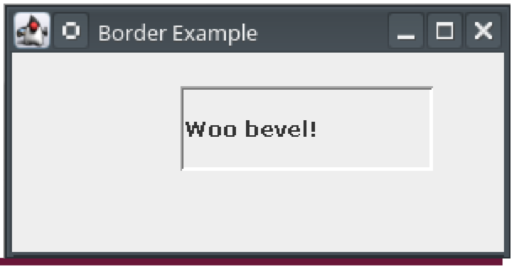
- **The `BorderFactory` class has several `static` methods that return various styles of border**
  - Since the methods are `static`, you don't need to create an instance of `BorderFactory`


```
BorderFactory
-------------

-------------
«query»
+ Border createLineBorder(Color)
+ Border createBevelBorder(int)
...
```

```
JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

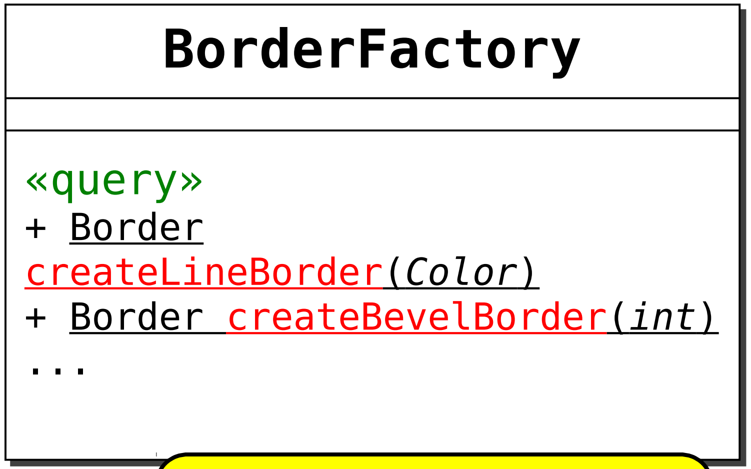


```
JLabel myLabel;
myLabel = new JLabel("Woo bevel!");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```



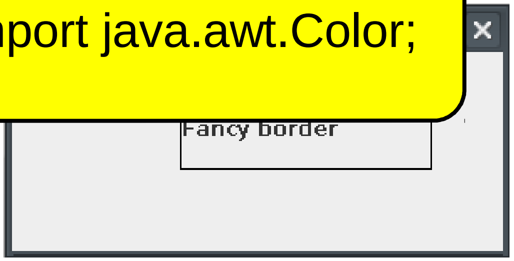
BorderFactory Class

- **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`


BorderFactory
«query»
+ Border createLineBorder(Color)
+ Border createBevelBorder(int)
...

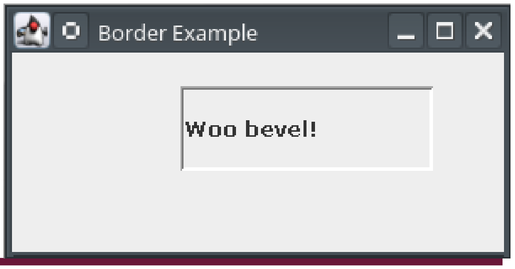
```
JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

import java.awt.Color;



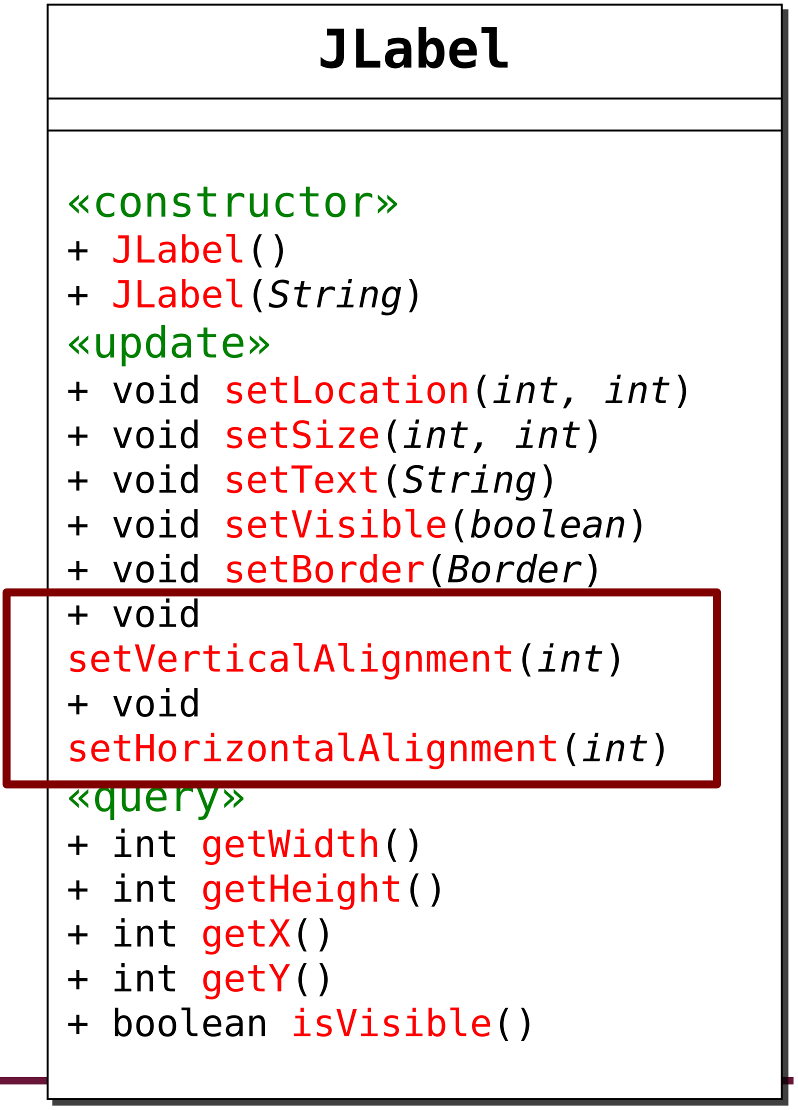
import javax.swing.border.*;

```
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```



JLabel Alignment

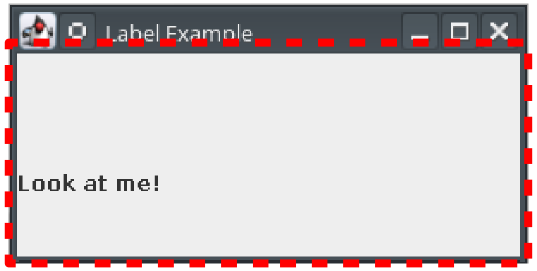
- **You can also optionally change the horizontal and vertical alignment**
  - Use one of several constants that are part of the SwingConstants class



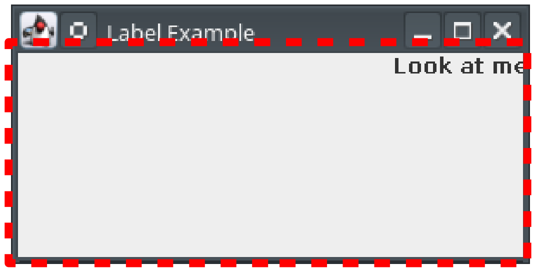
```
JLabel

«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);
```



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

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



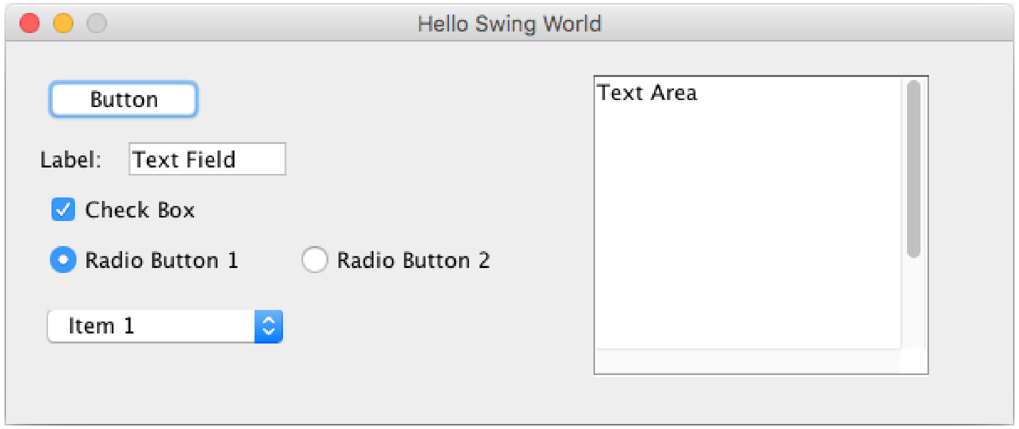
```
myLabel.setHorizontalAlignment(SwingConstants.RIGHT);
myLabel.setVerticalAlignment(SwingConstants.TOP);
```

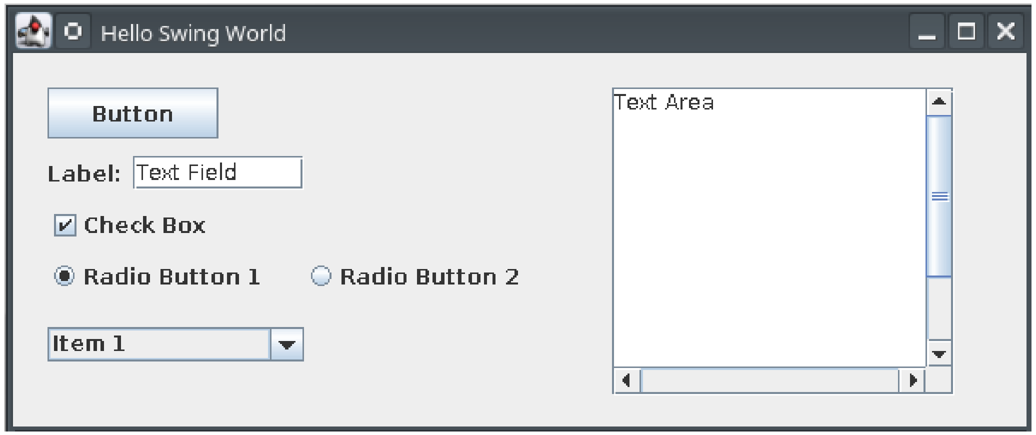
Full Program with JLabel

```java
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



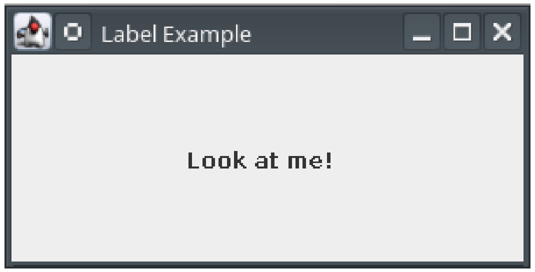


Add the JButton to the Frame

- **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);
  }
}
```

```
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()
```



Look at me!

JLabel

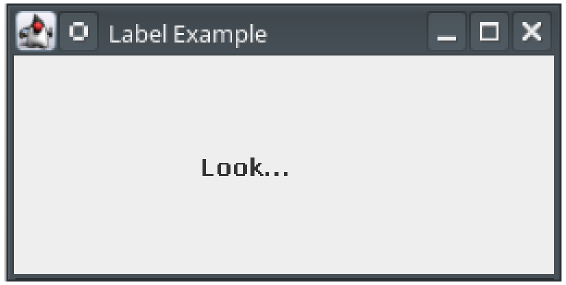
JLabel

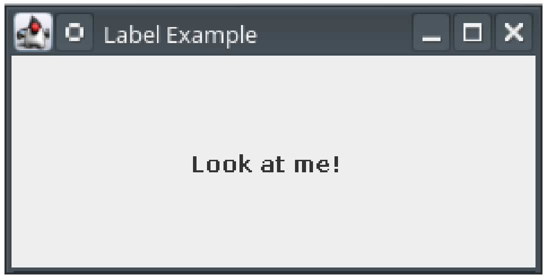
- **Sometimes you need to add some non-editable text to the frame**
  - For example, some JComponents don’t already have text included (like JTextField)
  - Or you may want to label a group of components
- **JLabels serve a different purpose than JButtons, but many methods overlap**

```
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 Size and Location

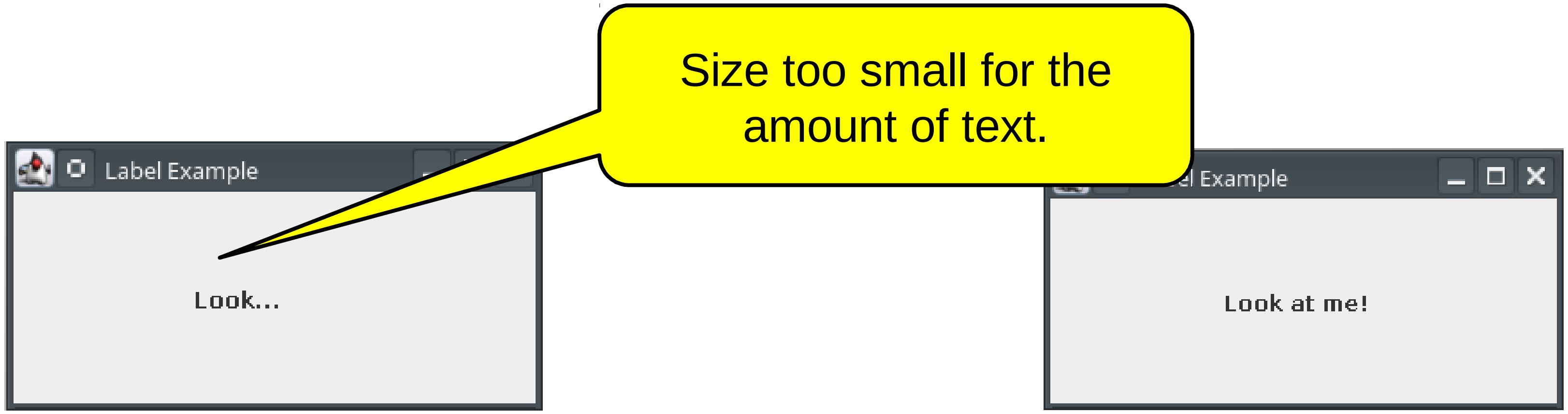
- **The size and location are measured in pixels, just like buttons**
  - Changing the size does not change the font size
- **The dimensions refer to an imaginary bounding box around the parameter of the label**





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**

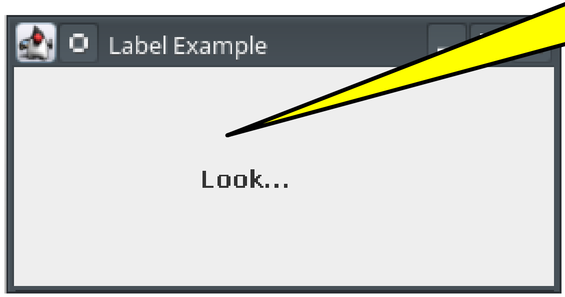


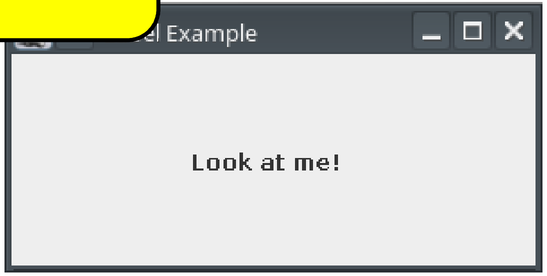
JLabel Size and Location

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

GOTCHA

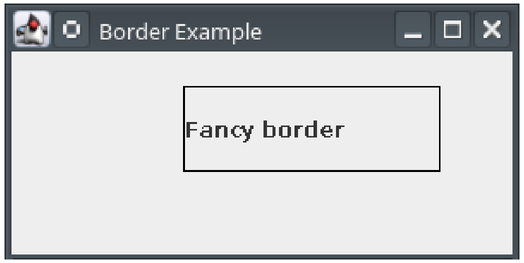
Size too small for the amount of text.





JLabel Borders

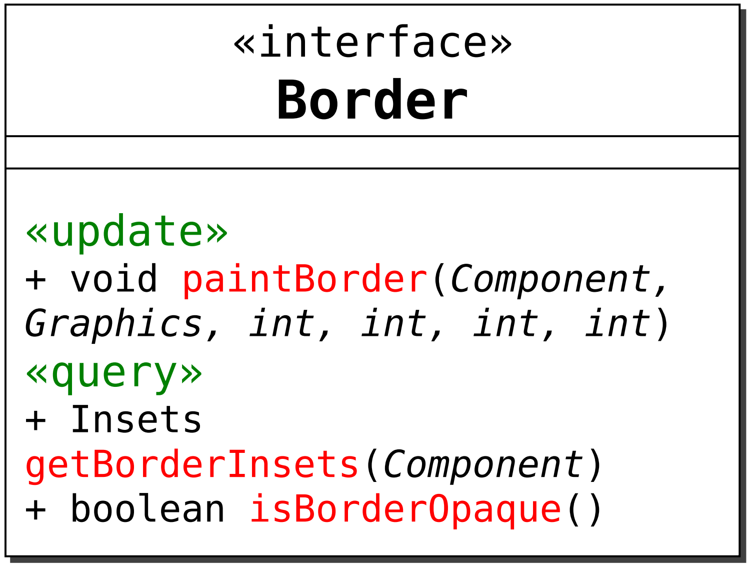
- **It is possible to see the bounding box by setting a visible border**
  - This can help you visualize position and size
  - To do so requires using a few new classes
- **Remove the borders once you have the correct size and location!**



```
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...**



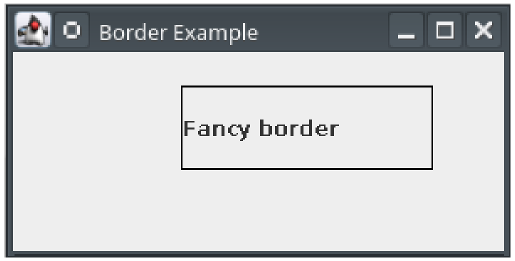
```
«interface»
Border

«update»
+ void paintBorder(Component, Graphics, int, int, int, int)
«query»
+ Insets getBorderInsets(Component)
+ boolean isBorderOpaque()
```

BorderFactory Class

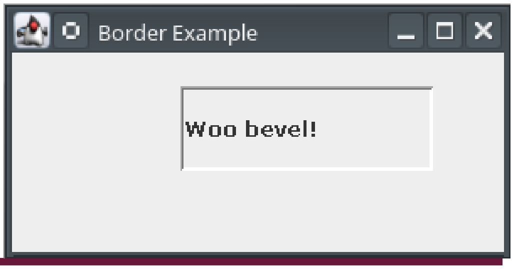
- **The** `BorderFactory` **class has several** `static` **methods that return various styles of border**
  - Since the methods are `static`, you don't need to create an instance of `BorderFactory`

```
BorderFactory
-------------
-------------
«query»
+ Border createLineBorder(Color)
+ Border createBevelBorder(int)
...
```

```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));
```



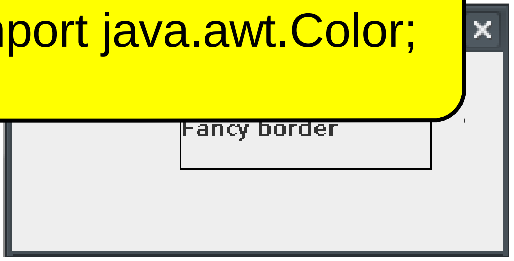
BorderFactory Class

- **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`

```
BorderFactory
-------------

-------------
«query»
+ Border createLineBorder(Color)
+ Border createBevelBorder(int)
...
```

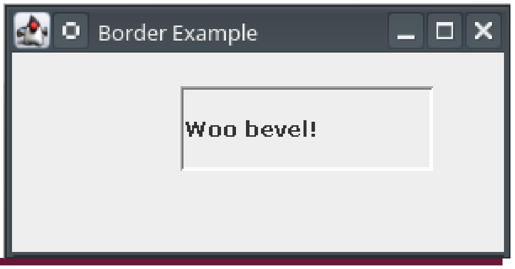
```
JLabel myLabel;
myLabel = new JLabel("Fancy border");
myLabel.setSize(150, 50);
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
```

import java.awt.Color;



import javax.swing.border.*;

```
myLabel.setLocation(100, 20);

myLabel.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```



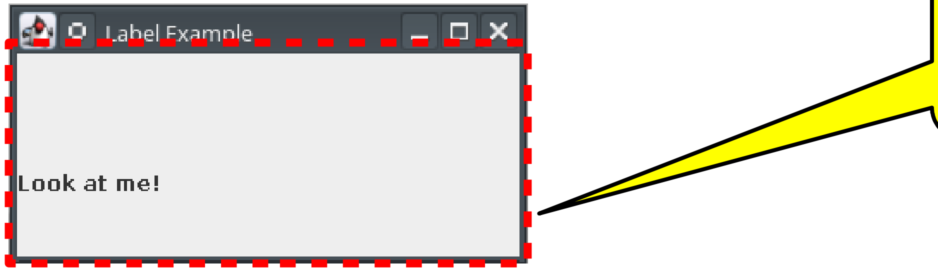
JLabel Alignment

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

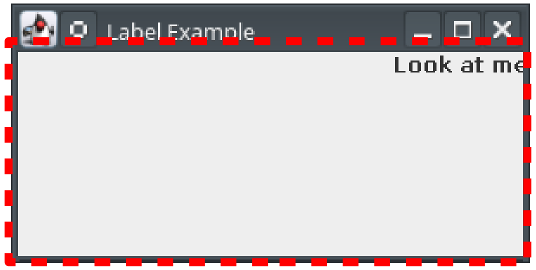
```
JLabel

«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);
```



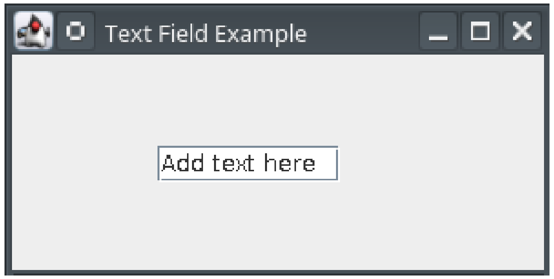
```
// horizontal alignment
SwingConstants.LEFT
SwingConstants.RIGHT
SwingConstants.CENTER

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



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

Full Program with JLabel

```java
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**

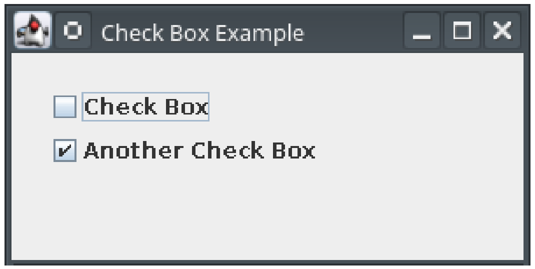
```
JTextField
-----------------------------------
-----------------------------------
«constructor»
+ JTextField()
+ JTextField(String)
«update»
+ void setLocation(int,int)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setText(String)
+ void requestFocus()
+ void selectAll()
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ String getText()
+ String getSelectedText()
```

JTextField

- **Many of the new methods only work when used with event handlers that we will write next week**

```
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);
  }
}
```

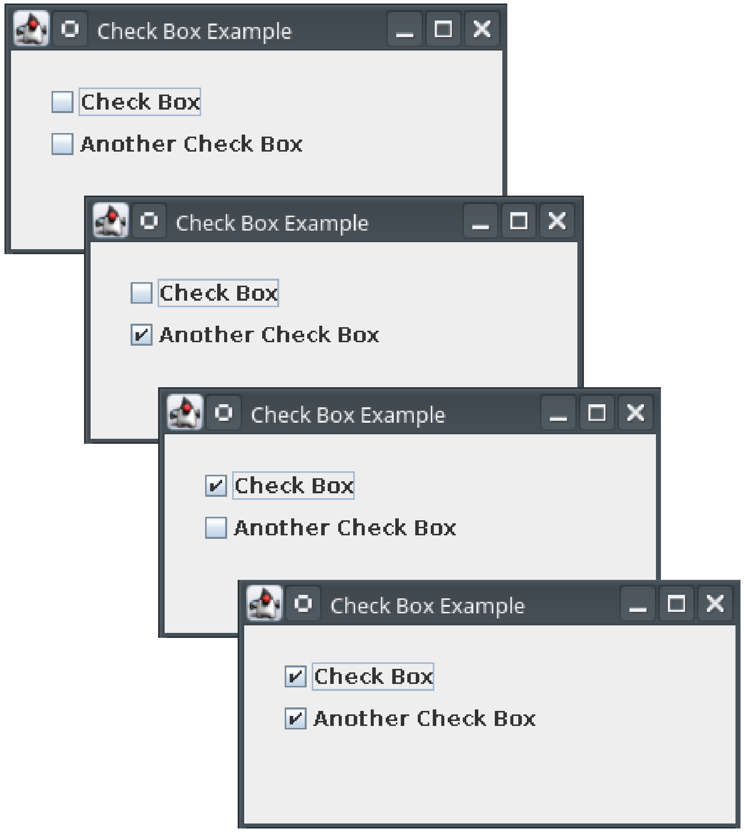
```
JTextField

«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()
```



JCheckBox

JCheckBox

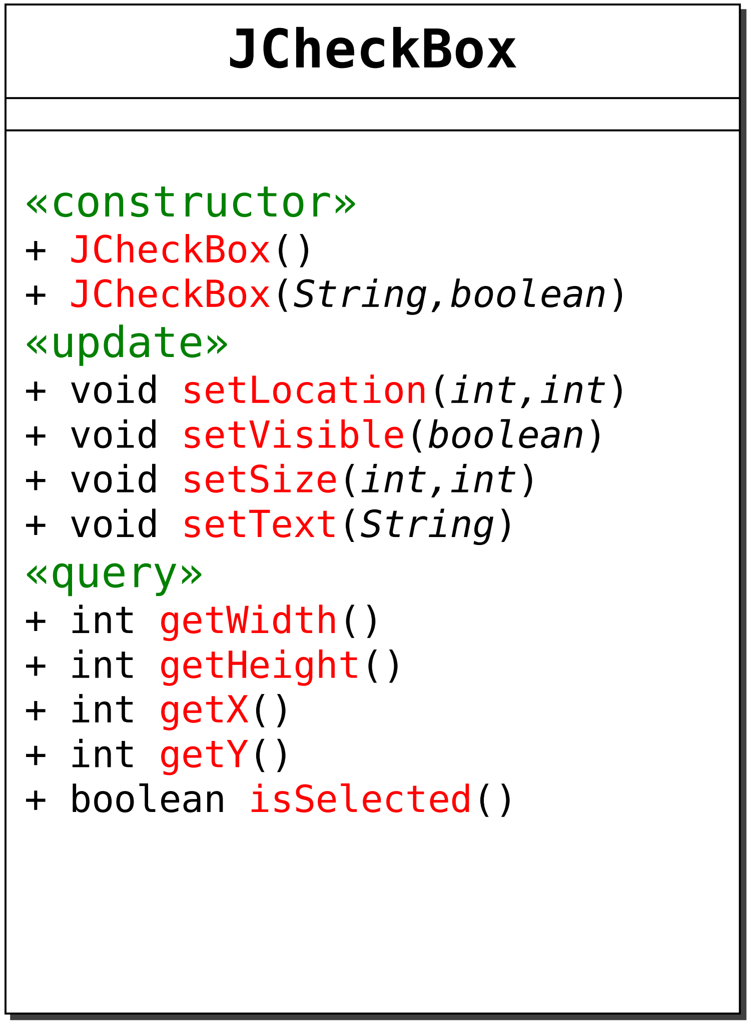
- **JCheckBox can be used to allow the user to select/unselect some, none, or all of a set of options**



```
JCheckBox

«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`



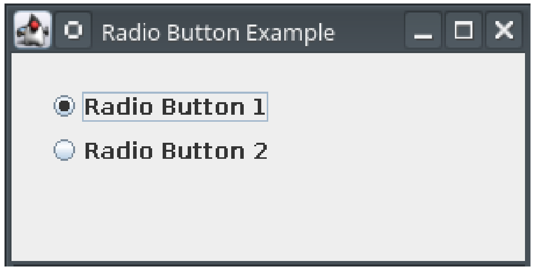
```
JCheckBox

«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

- **Next week we will read values from check boxes using** `isSelected`

```
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);
  }
}
```

```
JCheckBox
---------
---------
«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()
```



JRadioButton

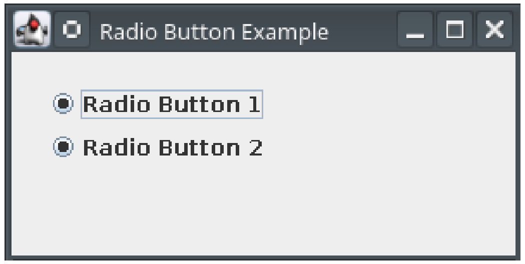
JRadioButton

- **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

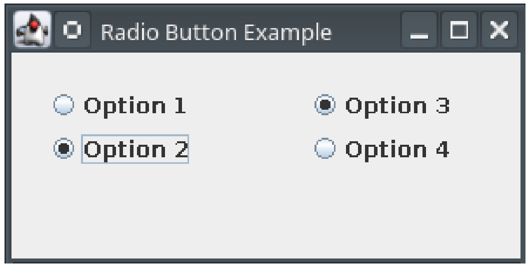
```
JRadioButton
----------------------------------------
----------------------------------------
«constructor»
+ JRadioButton()
+ JRadioButton(String,boolean)
«update»
+ void setLocation(int,int)
+ void setVisible(boolean)
+ void setSize(int,int)
+ void setText(String)
«query»
+ int getWidth()
+ int getHeight()
+ int getX()
+ int getY()
+ boolean isSelected()
```

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**

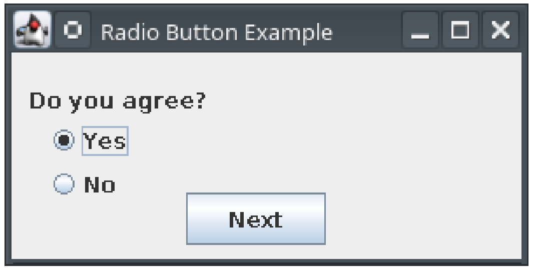


ButtonGroup Class

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

```
// 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
«constructor» + ButtonGroup() «update» + void add(*JComponent*)



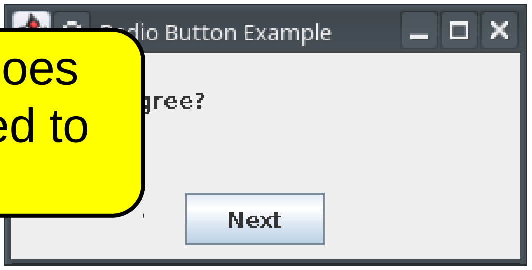
ButtonGroup Class

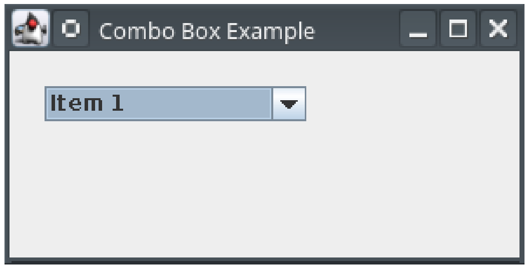
- **The ButtonGroup class is used to group radio buttons**
- **For each group:**
  - Instantiate a ButtonGroup object
  - add the JRadioButton objects to the ButtonGroup

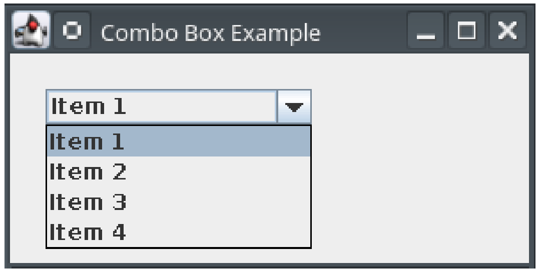
```
ButtonGroup

«constructor»
+ ButtonGroup()
«update»
+ void add(JComponent)
```

```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 ButtonGr
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




Combo Box Example
Item 1


Combo Box Example
Item 1
Item 1
Item 2
Item 3
Item 4

JComboBox

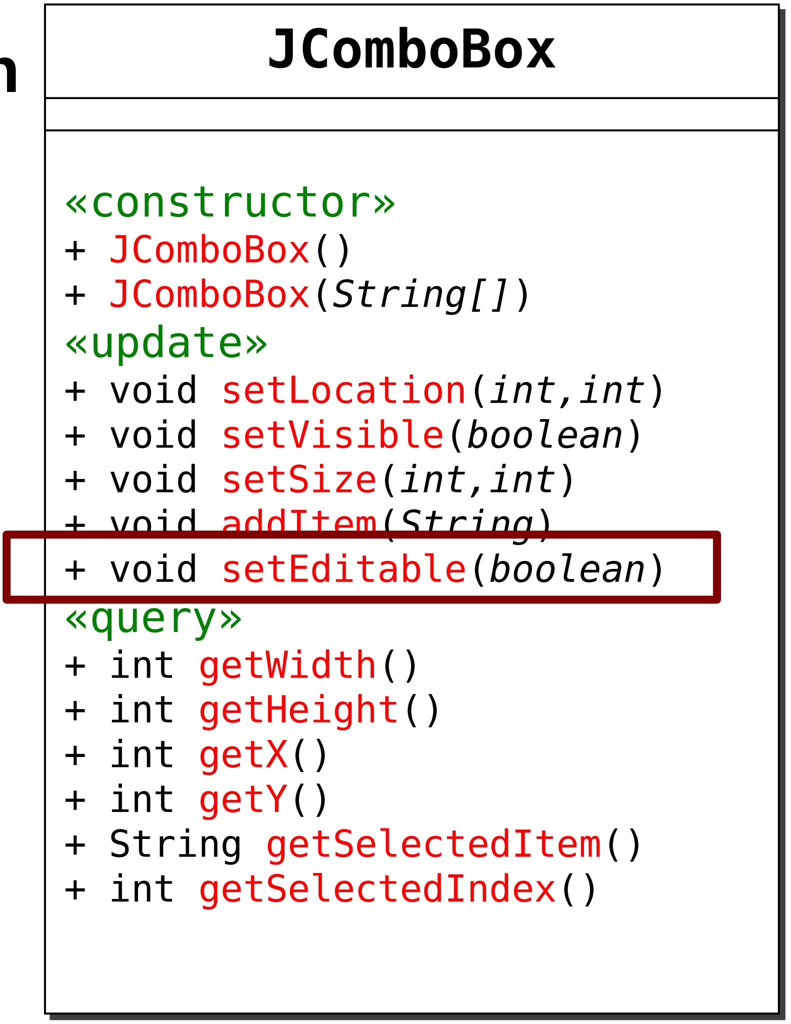
JComboBox

- **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**

```
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()
```

JComboBox

- **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



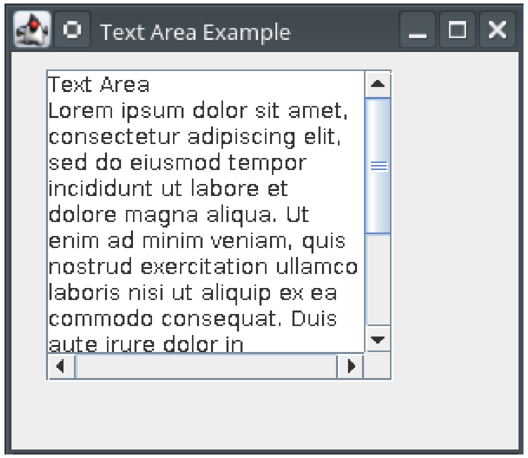
```
JComboBox

«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()
```

JComboBox

- **Notice that the constructor can take an array of Strings as an argument**

```
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);
  }
}
```

```
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()
```

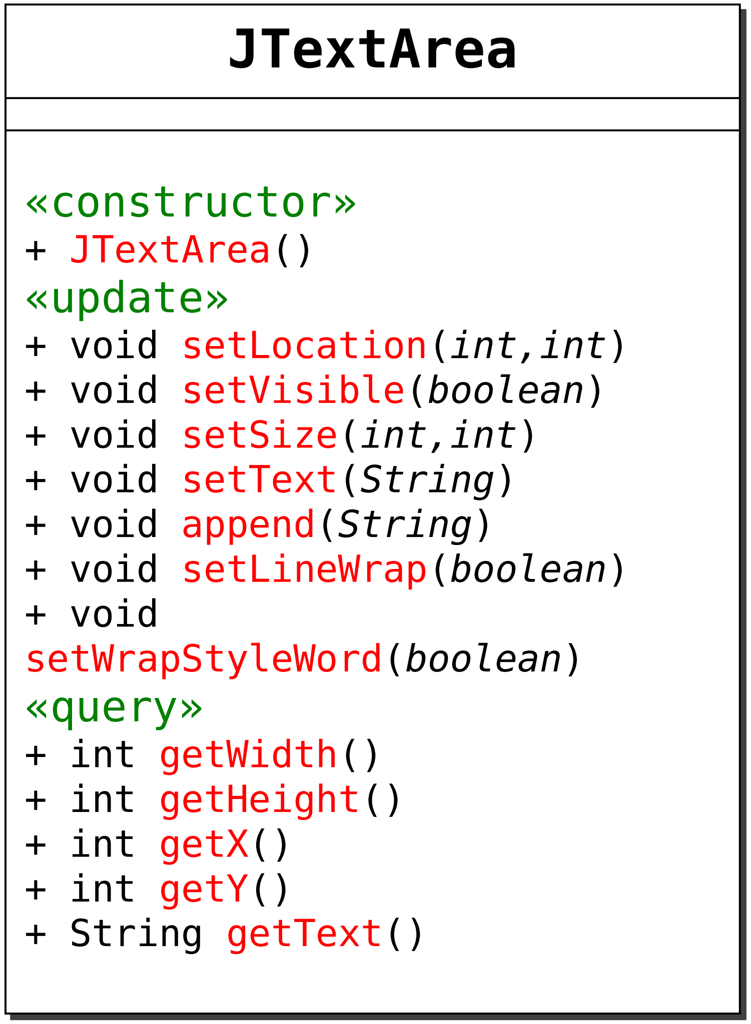


Text Area
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor
incididunt ut labore et
dolore magna aliqua. Ut
enim ad minim veniam, quis
nostrud exercitation ullamco
laboris nisi ut aliquip ex ea
commodo consequat. Duis
aute irure dolor in

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

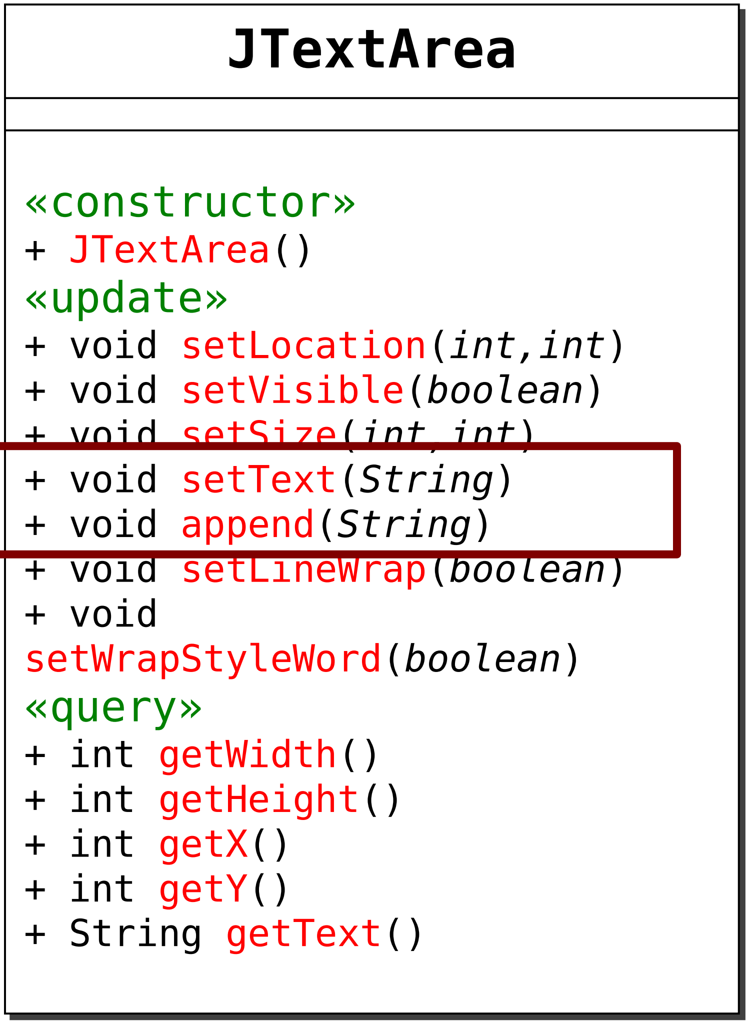


```
JTextArea

«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()
```

JTextArea Append

- **The** `setText` **method replaces all text in the text area**
- **The** `append` **method adds text after the existing text in the text area**

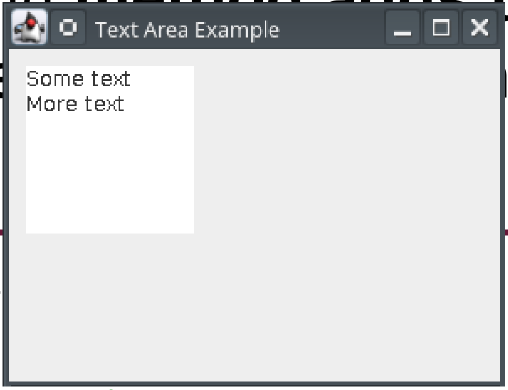
```java
import javax.swing.*;
public class TextAreaDemo extends JFrame {
  private JTextArea myTextArea;
  public TextAreaDemo () {
    // initialize JFrame here
    myTextArea = new JTextArea();
    myTextArea.setLocation(10, 10);
    myTextArea.setSize(100, 100);
    myTextArea.setText("Some text");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

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



```
JTextArea

«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()
```

JTextArea Append

- The `setText` method replaces all text in the text area
- The `append` method adds text after the e e text area



```
Some text
More text
```

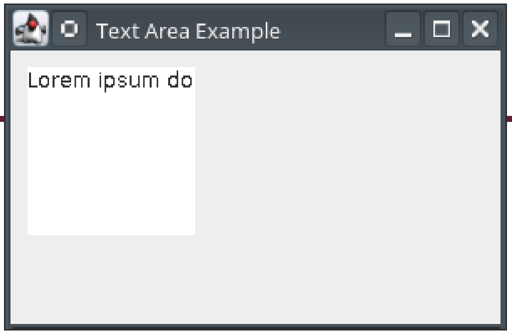
```
import javax.swing.
public class TextAr
  private JTextArea
  public TextAreaDe
    // initialize JFrame here
    myTextArea = new JTextArea();
    myTextArea.setLocation(10, 10);
    myTextArea.setSize(100, 100);
    myTextArea.setText("Some text");
    myTextArea.append("\nMore text");
    this.add(myTextArea);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

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

```
JTextArea
---------

---------
«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()
```

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);
  }
}
```

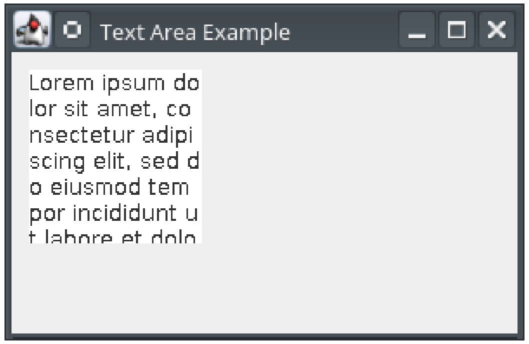


Lorem ipsum do

JTextArea Word Wrap

- **The setLineWrap method will automatically wrap text**

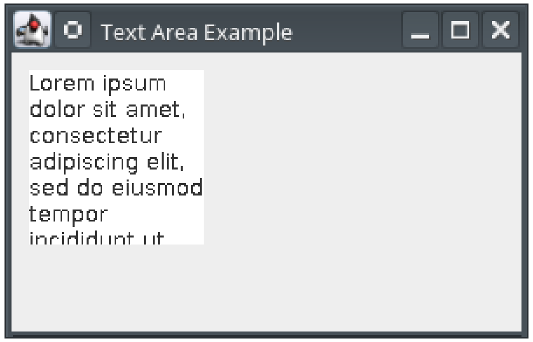
```
myTextArea.setLineWrap(true);
myTextArea.setText("Lorem ipsum dolor sit amet, consectetur...
```



```
Lorem ipsum do
lor sit amet, co
nsectetur adipi
scing elit, sed d
o eiusmod tem
por incididunt u
t labore et dolo
```

- **The setWrapStyleWord method will wrap at word boundaries**

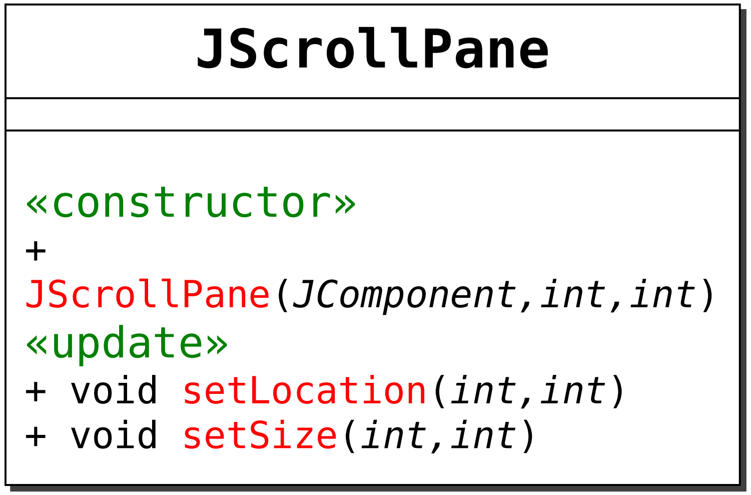
```
myTextArea.setLineWrap(true);
myTextArea.setWrapStyleWord(true);
myTextArea.setText("Lorem ipsum dolor sit amet, consectetur...
```



```
Lorem ipsum
dolor sit amet,
consectetur
adipiscing elit,
sed do eiusmod
tempor
incididunt ut
```

JScrollPane

- **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**



```
JScrollPane

«constructor»
+ JScrollPane(JComponent,int,int)
«update»
+ void setLocation(int,int)
+ void setSize(int,int)
```

JScrollPane Constructor

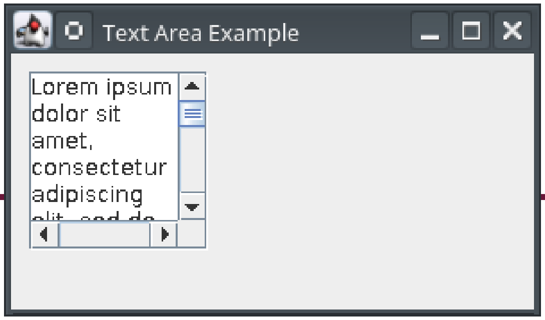
- **The constructor takes two integers whose values represent the vertical and horizontal scrollbar settings**
- **Use the constants that are part of JScrollPane for these int arguments**

```
JScrollPane
-----------

-----------
«constructor»
+ JScrollPane(JComponent,int,int)
«update»
+ void setLocation(int,int)
+ void setSize(int,int)
```

```
// 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
```

JTextArea and JScrollPane

```
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...

    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);
  }
}
```



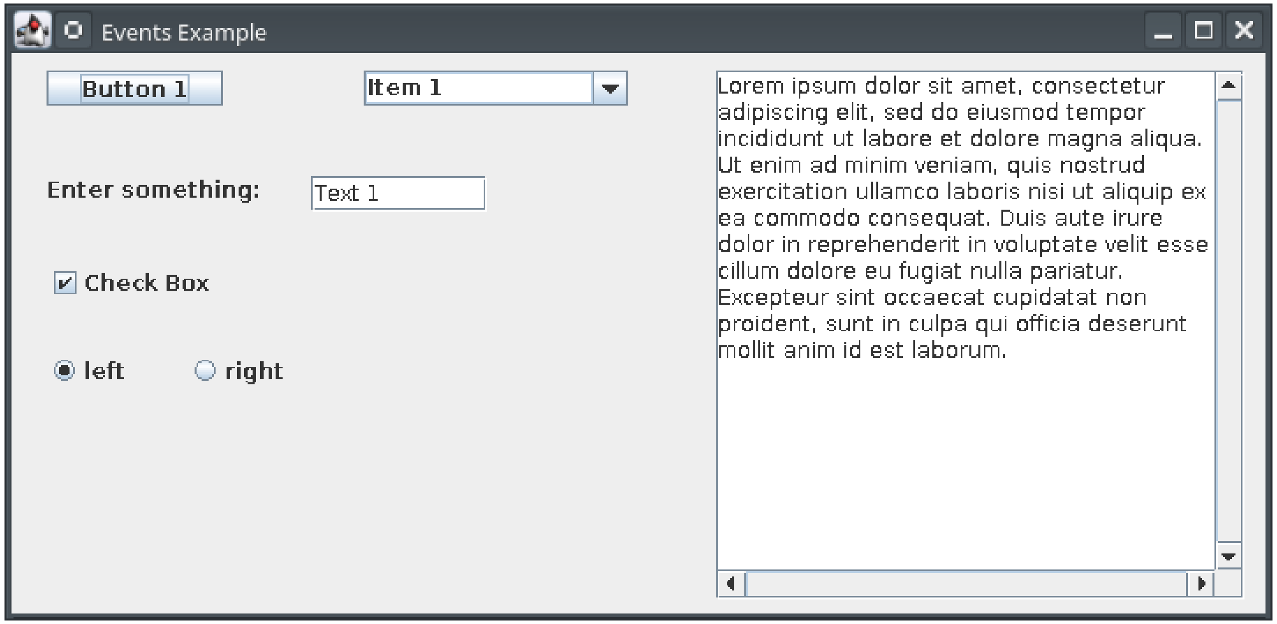
Lorem ipsum
dolor sit
amet,
consectetur
adipiscing

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**

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

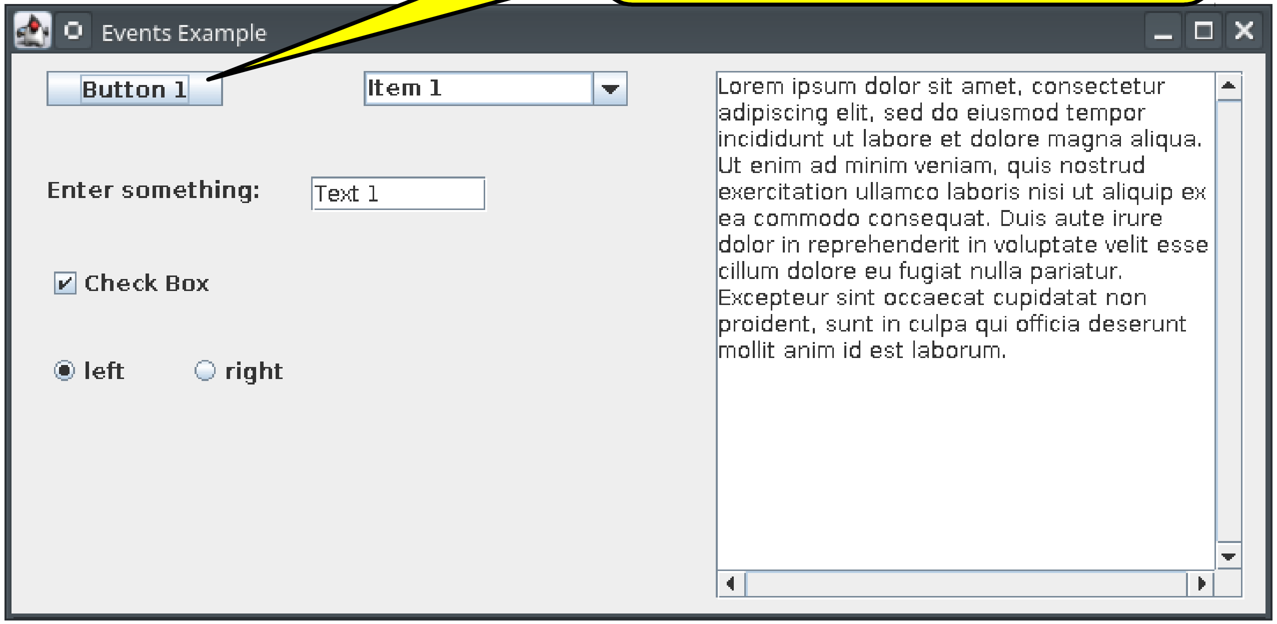


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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 th out

What events do you think can occur with a button?



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

JButton Events (Partial List)

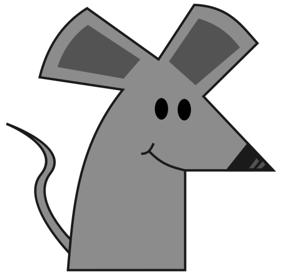
- **Some of events possible with JButtons**
  - Mouse pressed
  - Mouse released
  - Mouse clicked
  - Mouse entered area
  - Mouse left area
  - Got focus
  - Lost focus
  - Key pressed
  - Key released
  - Key typed
- **Each JComponent has a similar list of events that are possible with that component**

JButton Events (Partial List)

- **Some of events possible with JButtons**
  - Mouse pressed
  - Mouse released
  - Mouse clicked
  - Mouse entered area
  - Mouse left area

  Mouse events

  - Got focus
  - Lost focus
  - Key pressed
  - Key released
  - Key typed
- **Each JComponent has a similar list of events that are possible with that component**

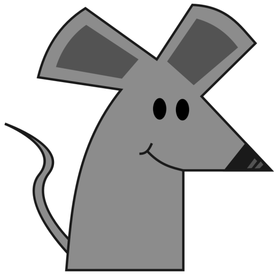


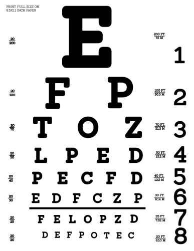
JButton Events (Partial List)

- **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

Mouse events

Focus events



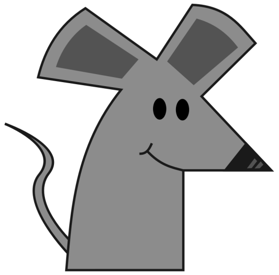


- **Each JComponent has a similar list of events that are possible with that component**

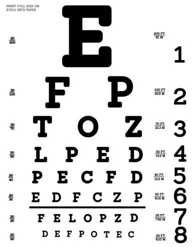
JButton Events (Partial List)

- **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

Mouse events



Focus events



Key(board) events

- **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

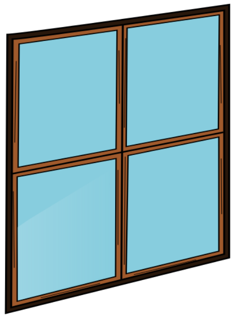
- **There are several types of listeners, grouped by the events that they cause**
  - WindowListener
  - FocusListener
  - MouseListener
  - KeyListener
  - ActionListener

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

Event Listeners

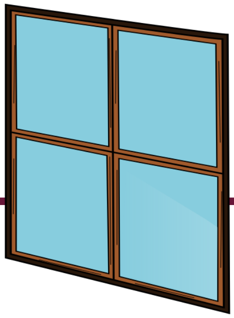
- **To use a listener**
  - Listeners require `import java.awt.event.*`
  - Write a class that `implements` the required listener interface
  - You need to write each method specified by the interface
  - Register the listener with the frame or component

- **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

WindowListener Interface



- **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

```
«interface»
WindowListener

«update»
+ void windowOpened(WindowEvent)
+ void windowClosing(WindowEvent)
+ void windowClosed(WindowEvent)
+ void windowIconified(WindowEvent)
+ void windowDeiconified(WindowEvent)
+ void windowActivated(WindowEvent)
+ void windowDeactivated(WindowEvent)
```

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

```
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

Remember: `this` refers to the frame (our class)

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

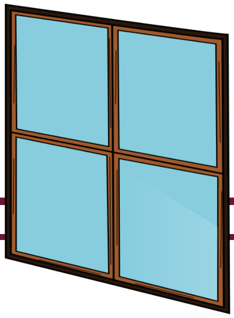
```
import javax.swing.*;

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

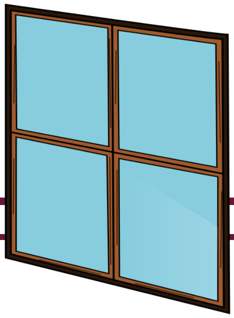
WindowListener



- **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

```
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);
  }
}
```

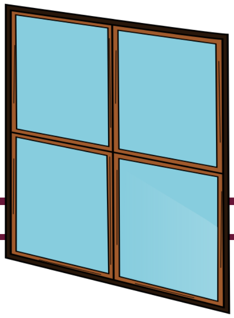
WindowListener



- **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

```
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

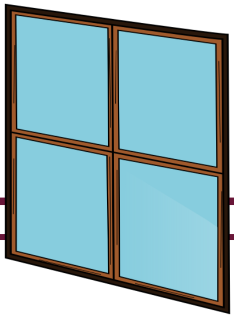


- **To use a liste**
  - ✔`import`
  - ✔`implements` the required listener interface
  - Write each method specified by the interface
  - Register the listener with the frame or component

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

```
                *;
            wt.even

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

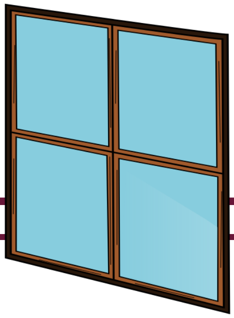
```
import javax.swing.*;
import java.awt.event.*;

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

```
JFrame
------

------
«constructor»
+ JFrame()
+ JFrame(String)
«update»
+ void add(JComponent)
+ void setLocation(int,int)
+ void setSize(int,int)
+ void setTitle(String)
+ void
addWindowListener(WindowListener)
// other methods
```

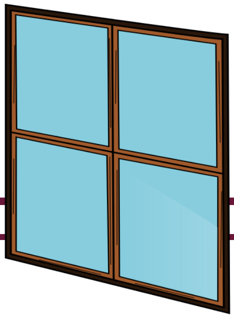
WindowListener



- **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

```
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
}
```

WindowListener



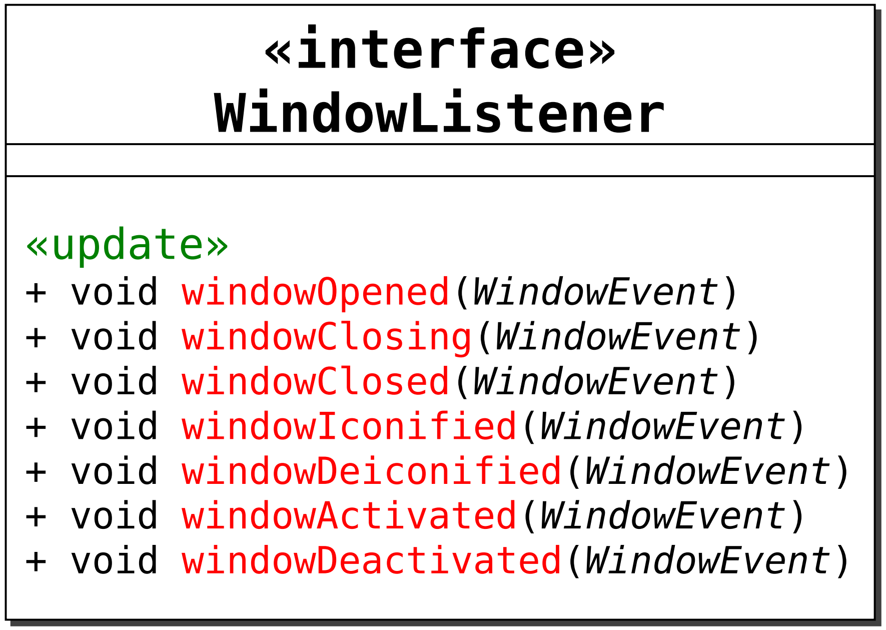
- **To use a listener**
  - ✔ impo
  - ✔ imple
    required listener interface
  - ✔ Write each method specified by the interface
  - ✔ Register the listener with the frame or component

More type conformance goodness here!

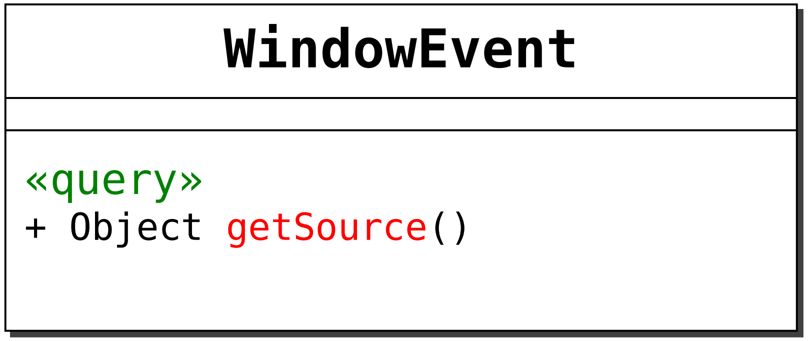
```
       javax.swing.*;
       java.awt.event.*;

       class HelloSwingWorld extends JFrame
                            implements WindowListener{
  public Hell     ngWorld () {
    // JFrame initia    zation 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
}
```

WindowEvent

- **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**



```
«interface»
WindowListener

«update»
+ void windowOpened(WindowEvent)
+ void windowClosing(WindowEvent)
+ void windowClosed(WindowEvent)
+ void windowIconified(WindowEvent)
+ void windowDeiconified(WindowEvent)
+ void windowActivated(WindowEvent)
+ void windowDeactivated(WindowEvent)
```

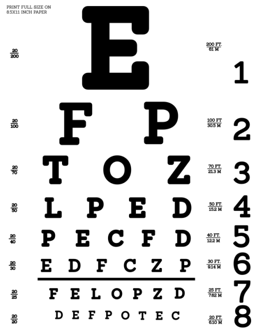


```
WindowEvent

«query»
+ Object getSource()
```

Listeners and Components

- **The following components can register the following listeners**

	WindowListener	FocusListener	MouseListener	KeyListener	ActionListener
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox		X	X	X	X
JTextArea		X	X	X	
Timer					X

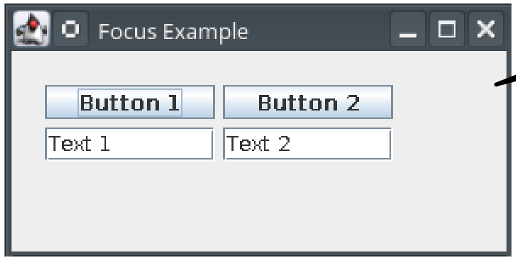


FocusListener

Focus

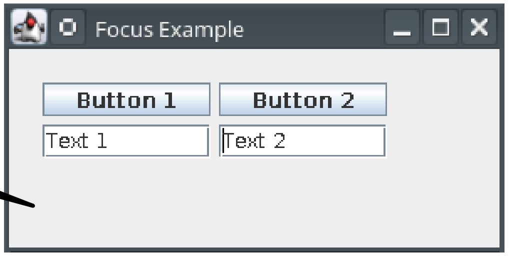
- **With GUIs, you often have several components that the user can interact with**
- **The current component that the user is interacting with is said to have the focus**
- **For example:**
  - Buttons can be selected without being clicked – the user can “click” the button by hitting space
  - One of 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

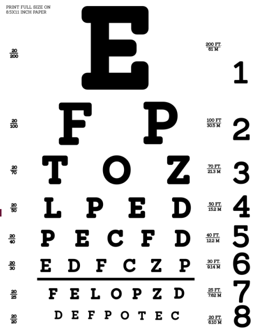


Button 1 currently has focus

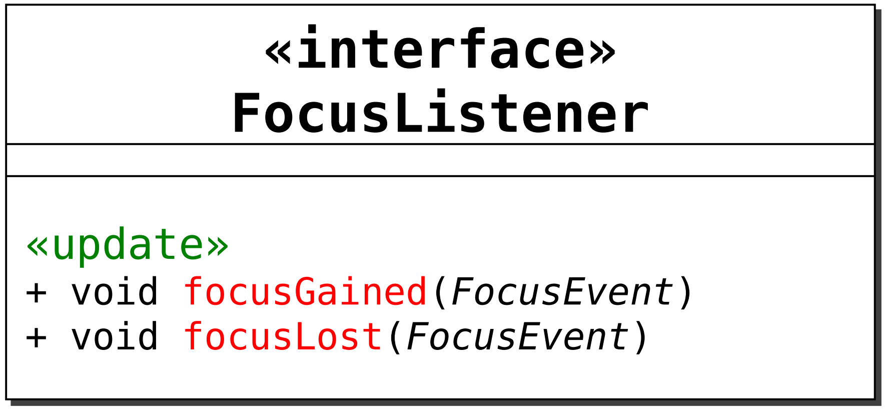
Text field 2 currently has focus



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



```
«interface»
FocusListener

«update»
+ void focusGained(FocusEvent)
+ void focusLost(FocusEvent)
```

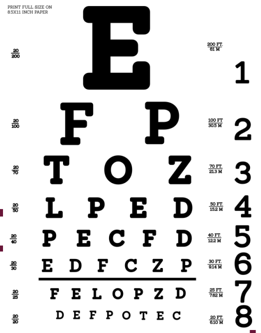
Recall JTextField from Last Week

- **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

```
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()
```

```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



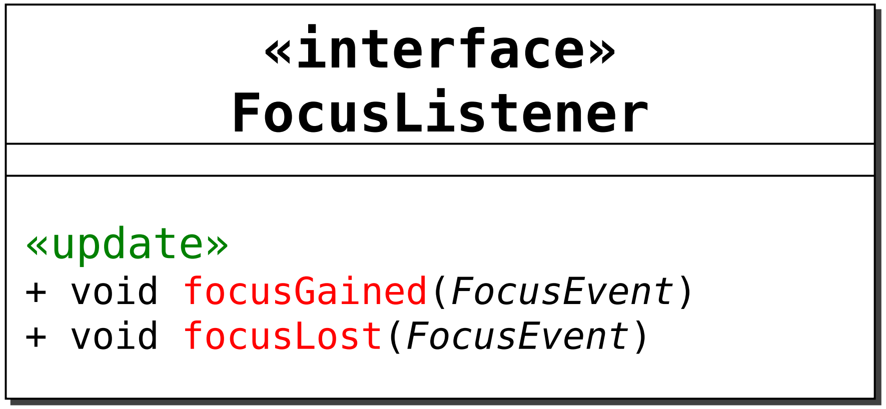
- **To use a listener**
  - ✔`import`
  - ✔`implements` the required listener interface
  - Write each method specified by the interface
  - Register the listener with the frame or component

```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);
  }
}
```

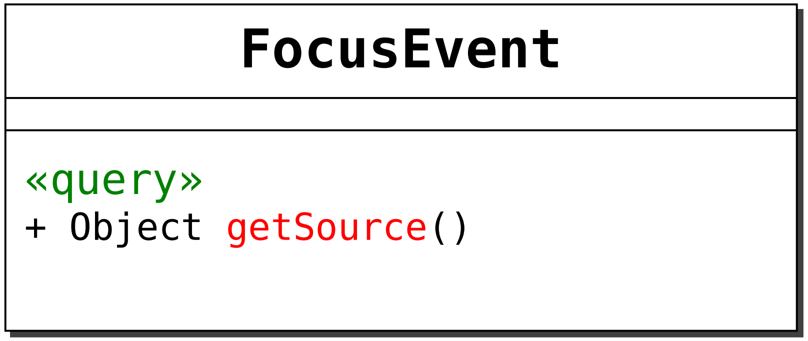
FocusEvent

- **The** `getSource` **method returns the memory reference of the object that caused the event to occur**

- **This has the same use as the WindowEvent**

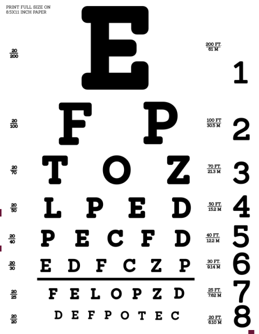


```
«interface»
FocusListener

«update»
+ void focusGained(FocusEvent)
+ void focusLost(FocusEvent)
```



```
FocusEvent

«query»
+ Object getSource()
```

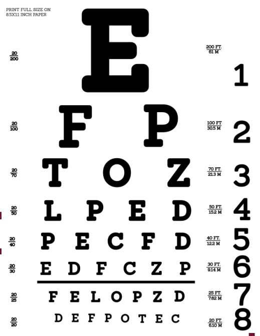
FocusListener



- **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

```
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

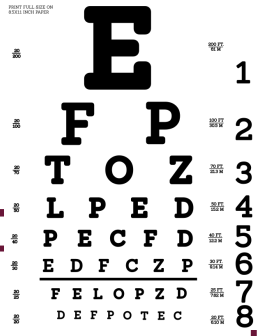


- **To use a listener**
  - ✔`import`
  - ✔`implements` the required listener interface
  - ✔Write each method speci interf
- Register the listener with the frame or component

This is how we can determine which component got focus

```
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)


  ic static void main (String[] args) {
    rame frame = new HelloSwingWorld();
      e.setVisible(true);


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

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

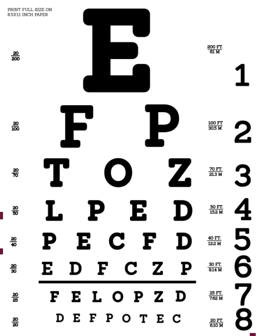
FocusListener



- **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)
    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);
    }
  }
}
```

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
                        impl
  private JTextField myTextFi

  public HelloSwingWorld
    // JFrame initialization
    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);
    }
  }
}
```

We register our class with the text field

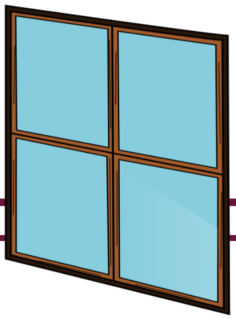
Listeners and Components

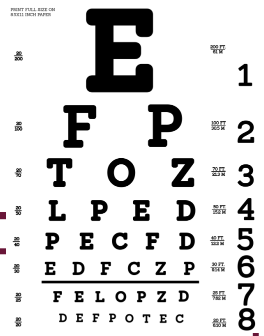
- **The following components can register the following listeners**

	WindowListener	FocusListener	MouseListener	KeyListener	ActionListener
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox		X	X	X	X
JTextArea		X	X	X	
Timer					X

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.addWindowListener(this);
    myTextField.addFocusListener(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)
}
```

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**

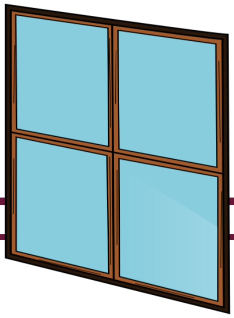
  - WindowListener
  - FocusListener
  - MouseListener
  - KeyListener
  - ActionListener

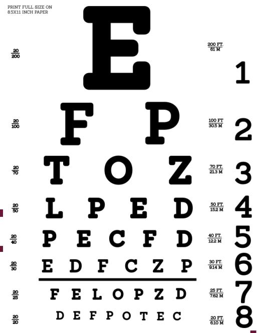
- **You only need to implement the listeners that you intend to use!**

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

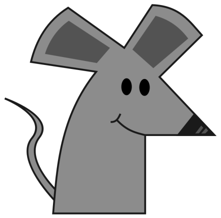
Interfaces





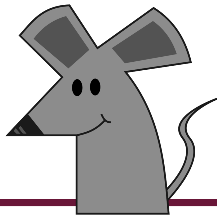
- **Remember that we can possibly implement multiple interfaces**

```
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

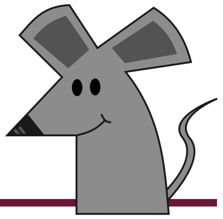
MouseListener Interface



- **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

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

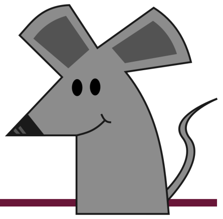
MouseListener



- **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);
  }
}
```

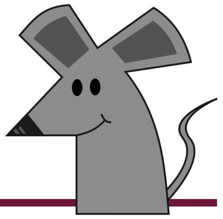
MouseListener



- **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

```
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.");
    }
  }

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

MouseListener

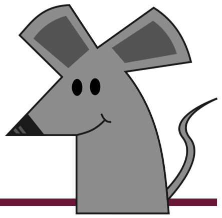


- **To use a listener**
  - ✔`import`
  - ✔`implements` the required listener interface
  - ✔W
    s
    interface
  - Register the listener with the frame or component

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

```
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)

  }
    blic static void main (String[] args) {
          e frame = new HelloSwingWorld();
    frame.se    isible(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
}
```

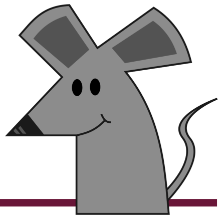
MouseListener



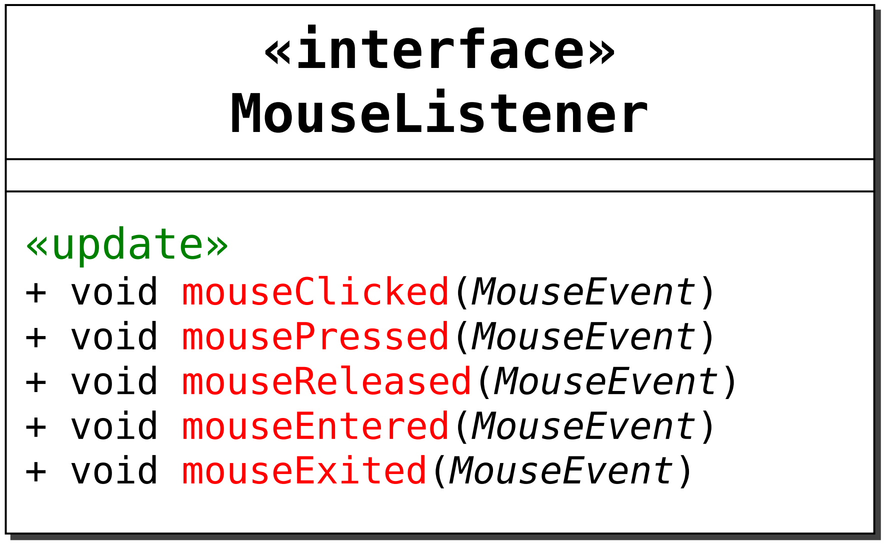
- **To use a listener**
  - ✔ import
  - ✔ implements the required listener interface
  - ✔ Write each method specified by the interface
  - ✔ Register the listener with the frame or component

```
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
}
```

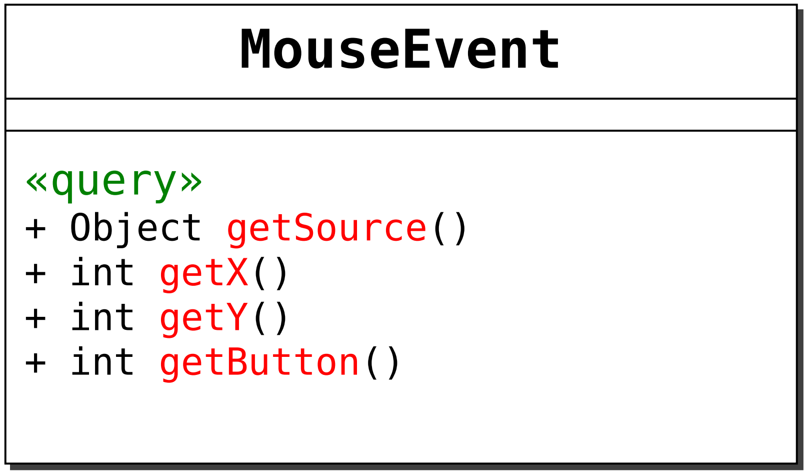
MouseEvent



- **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**

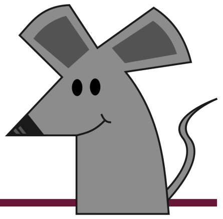


```
«interface»
MouseListener

«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()
```

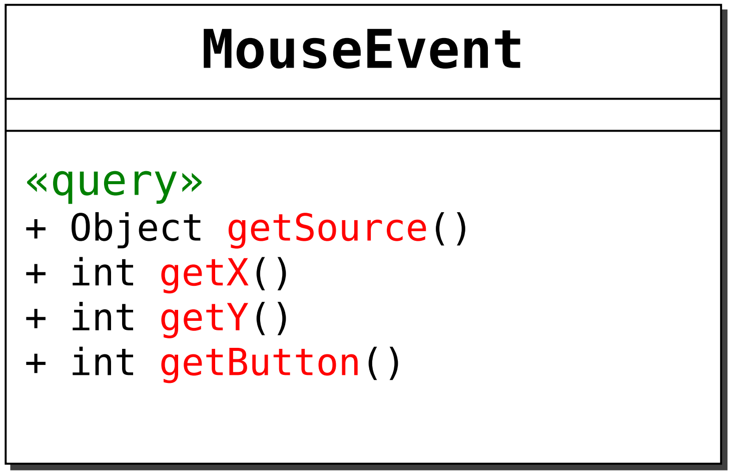
MouseEvent



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

```
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)
```



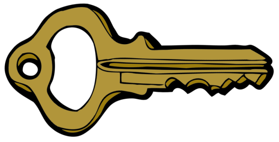
MouseEvent

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

Listeners and Components

- **The following components can register the following listeners**

	WindowListener	FocusListener	MouseListener	KeyListener	ActionListener
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox		X	X	X	X
JTextArea		X	X	X	
Timer					X



KeyListener

KeyListener Interface

- **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

```
        «interface»
        KeyListener
-----------------------------
-----------------------------
«update»
+ void keyTyped(KeyEvent)
+ void keyPressed(KeyEvent)
+ void keyReleased(KeyEvent)
```

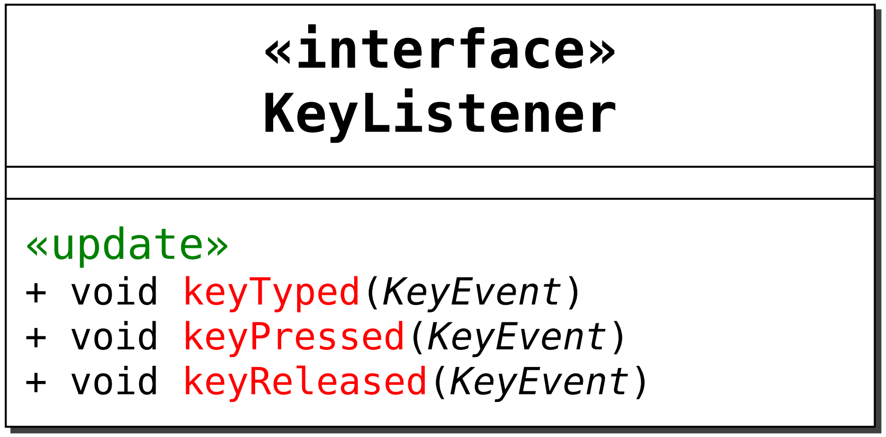
KeyListener

- **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

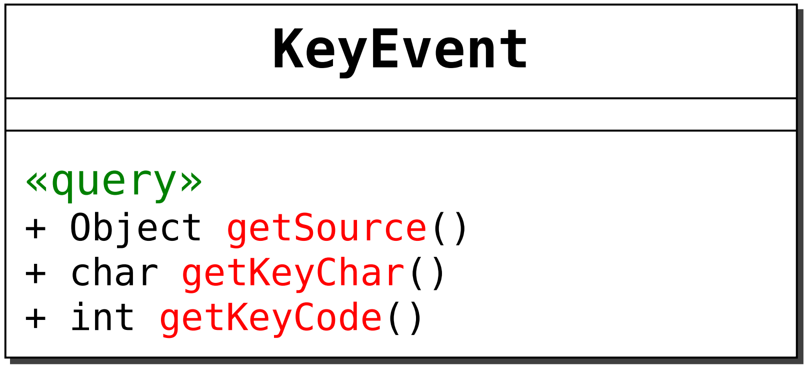
```
import javax.swing.*;
import java.awt.event.*;

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.



```
«interface»
KeyListener

«update»
+ void keyTyped(KeyEvent)
+ void keyPressed(KeyEvent)
+ void keyReleased(KeyEvent)
```



```
KeyEvent

«query»
+ Object getSource()
+ char getKeyChar()
+ int getKeyCode()
```

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

```
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
}
```

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

```
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
}
```

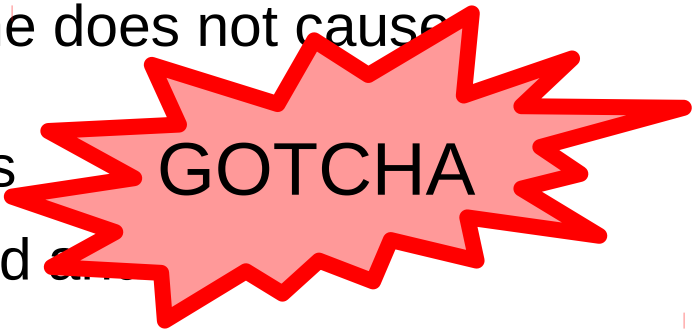
KeyListener Interface

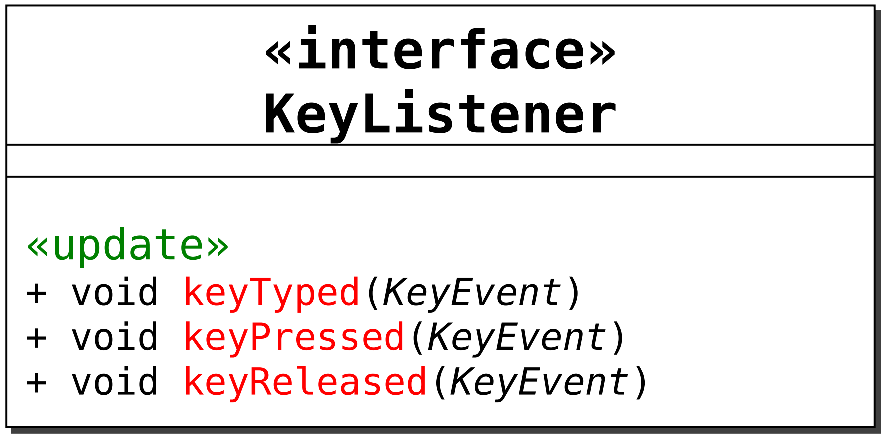
- **Not all keys will cause all types of KeyListener events**
  - For example, typing the shift key alone does not cause keyTyped to get called
    - Others: Ctrl, Alt, and arrow keys
    - You will need to use keyPressed and keyReleased instead
  - Key combinations do cause keyTyped to get called, however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

```
«interface»
KeyListener
---
---
«update»
+ 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 a keyReleased instead
  - Key combinations do cause keyTyped to get called, however
    - Ctrl-c
    - Shift-x
    - etc.





```
«interface»
KeyListener

«update»
+ void keyTyped(KeyEvent)
+ void keyPressed(KeyEvent)
+ void keyReleased(KeyEvent)
```

Listeners and Components

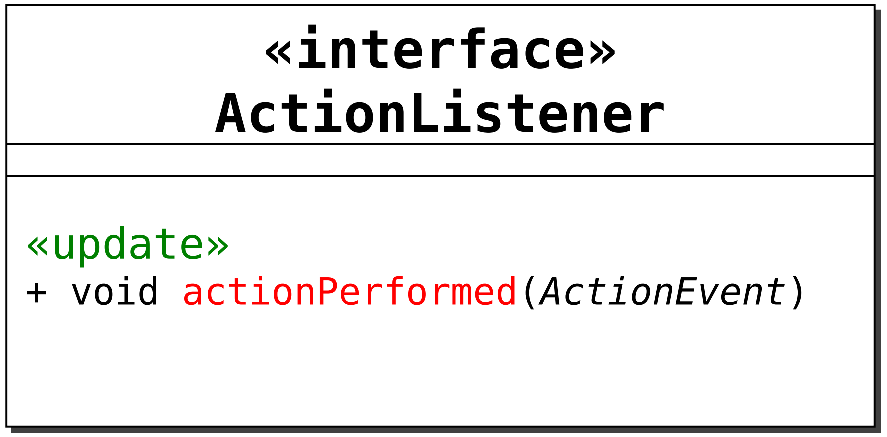
- **The following components can register the following listeners**

	WindowListener	FocusListener	MouseListener	KeyListener	ActionListener
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox		X	X	X	X
JTextArea		X	X	X	
Timer					X

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



```
«interface»
ActionListener

«update»
+ void actionPerformed(ActionEvent)
```

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

```
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);
  }

}
```

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);
  }

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

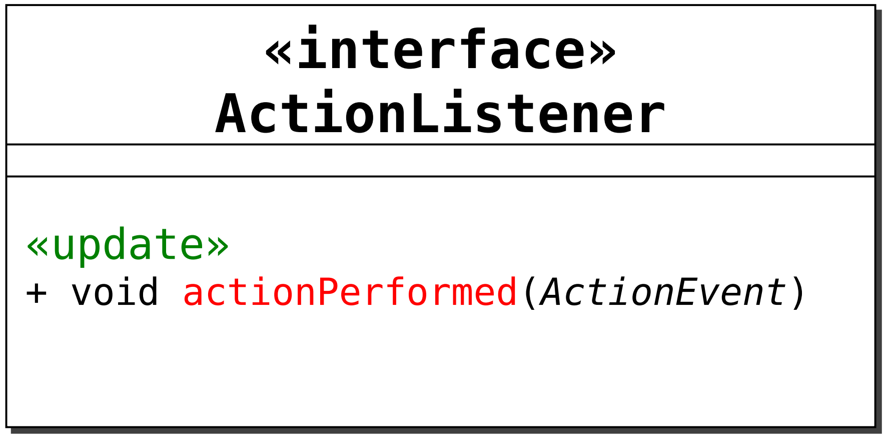
ActionListener

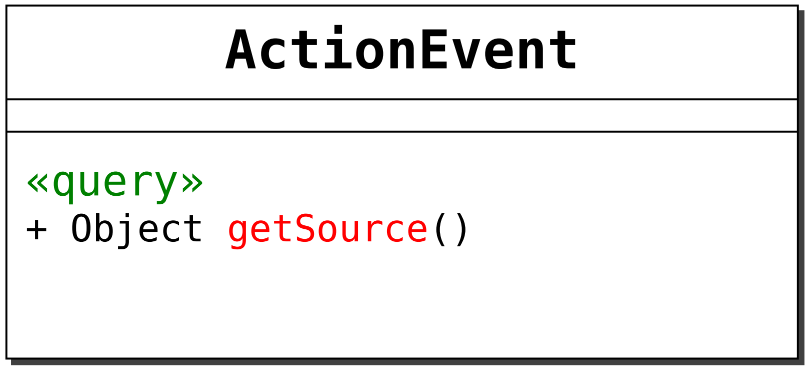
- **To use a listener**
  - ✔`import`
  - ✔`implements` the required listener interface
  - ✔Write each method specified by the interface
  - ✔Register the listener with the frame or component

```
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!");
      }
    }
  }
}
```

ActionEvent

- **No especially interesting KeyEvents, just the usual** `getSource`





Listeners and Components

- **The following components can register the following listeners**

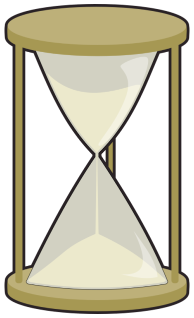
	WindowListener	FocusListener	MouseListener	KeyListener	ActionListener
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox		X	X	X	X
JTextArea		X	X	X	
Timer					X

Listeners and Components

- **The following components can register the following listeners**

	WindowListener	**FocusListener**	**MouseListener**	**KeyListener**	**ActionListener**
JFrame	X	X	X	X	
JButton		X	X	X	X
JLabel		X	X	X	
JTextField		X	X	X	X
JCheckBox		X	X	X	X
JRadioButton		X	X	X	X
JComboBox				X	X
JTextArea				X	
Timer					X

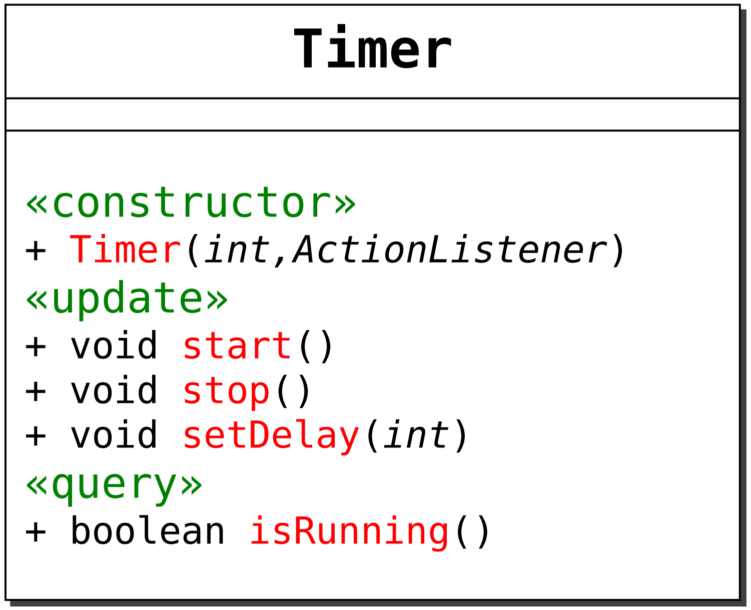
Soooo… what's this all about?



Swing Timer

Timer

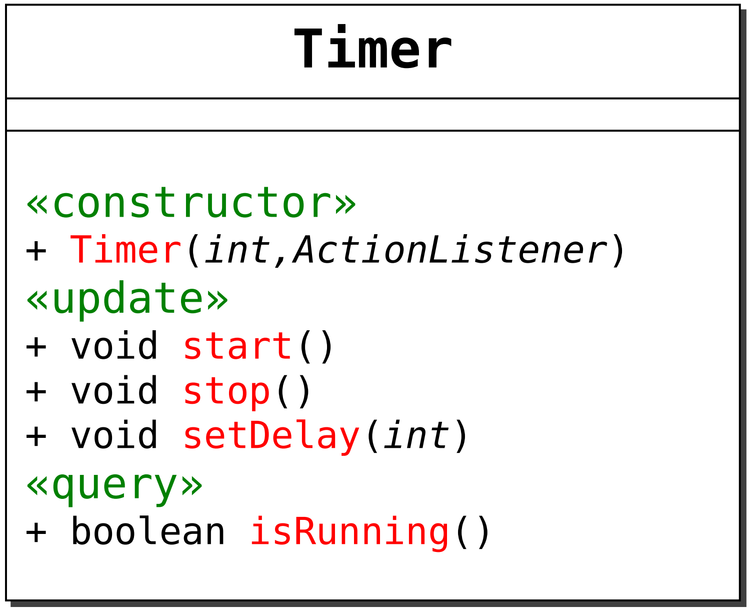
- **Swing Timers are used to execute an ActionEvent at periodic time intervals**
- **It is not a JComponent, so it does not need to be added to the frame to be used**
  - However, it is imported with `import javax.swing.*`
- **Not the only Timer in Java**
  - Another: `import java.util.Timer`
  - Do not use!



```
Timer

«constructor»
+ Timer(int,ActionListener)
«update»
+ void start()
+ void stop()
+ void setDelay(int)
«query»
+ boolean isRunning()
```

Timer Constructor

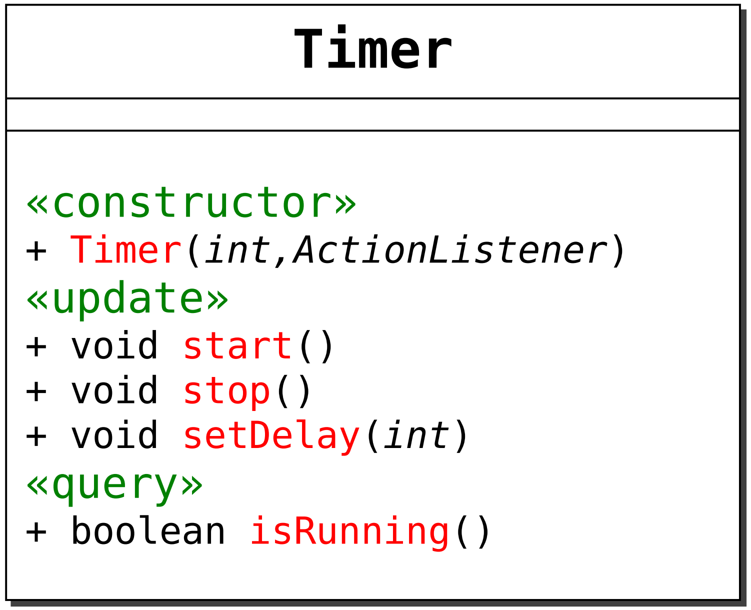
- **The integer passed to both the constructor and to the setDelay method is the number of milliseconds between its ActionEvents**
  - Hint, there are 1,000 milliseconds in a second



```
Timer

«constructor»
+ Timer(int,ActionListener)
«update»
+ void start()
+ void stop()
+ void setDelay(int)
«query»
+ boolean isRunning()
```

Timer

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



```
Timer

«constructor»
+ Timer(int,ActionListener)
«update»
+ void start()
+ void stop()
+ void setDelay(int)
«query»
+ boolean isRunning()
```

```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 cause**
  - The
    auto

Don’t forget to register the ActionListener with the JButton

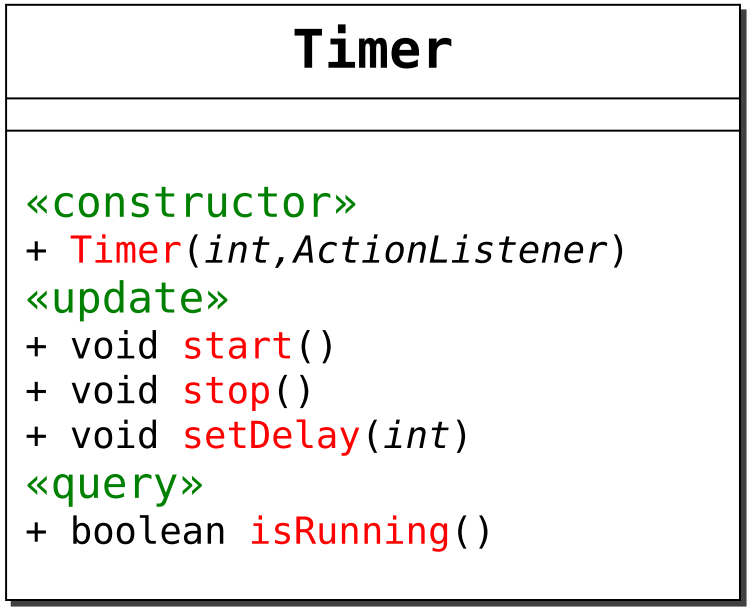
```
Timer

«constructor»
+ Timer(int,ActionListener)
«update»
+ void start()
+ void stop()
+ void setDelay(int)
«query»
+ boolean isRunning()
```

```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

No need to add the timer to the JFrame



```
Timer
-----------------------------
«constructor»
+ Timer(int,ActionListener)
«update»
+ void start()
+ void stop()
+ void setDelay(int)
«query»
+ boolean isRunning()
```

```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);
    }
  }
}
```