An Introduction To Graphical User Interfaces

Part 1: You will learn how to write a program that reacts to user interactions with a button or java window. Using GUI containers to hold other GUI components.

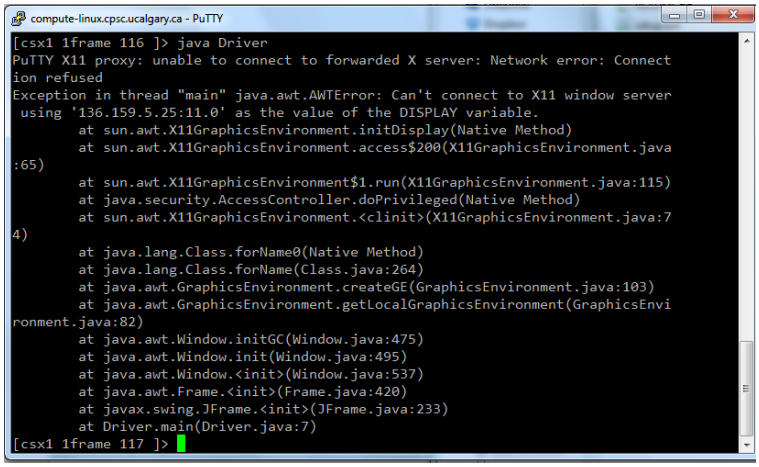
Tip For Success: Reminder

- Look through the examples and notes before class.
- This is especially important for this section because the execution of this programs will not be in sequential order.
- Instead execution will appear to 'jump around' so it will be harder to understand the concepts and follow the examples illustrating those concepts if you don't do a little preparatory work.
- Also the program code is more complex than most other examples.
- For these reasons tracing the code in this section is more challenging

James Tam

Don’t Run The GUI Code Via SSH/Putty!

- The former is graphical
- The latter is text-only



James Tam

Options: Writing GUI Code At Home

1. Install JDK on your home computer: edit, compile and run your programs locally.
2. Use JDK on the CPSC network:
   a) Edit and compile your programs on the CPSC network using a remote login program (such as Putty) and a text-based editor (such as Emacs). The java compiler is called: javac
   b) Transfer your compiled byte code files (`.class`) from your CPSC UNIX account to your home computer using a file transfer program (e.g., Filezilla, secure FTP) although you will have learn its usage on your own
      - There’s no time to take about this in class but you can ask questions about Filezilla after class.
   c) On your home computer open a command line (‘cmd’ in Windows) and run the java interpreter: java (Because program execution is occurring locally the graphics will be drawn by your computer and not via the remote login program).

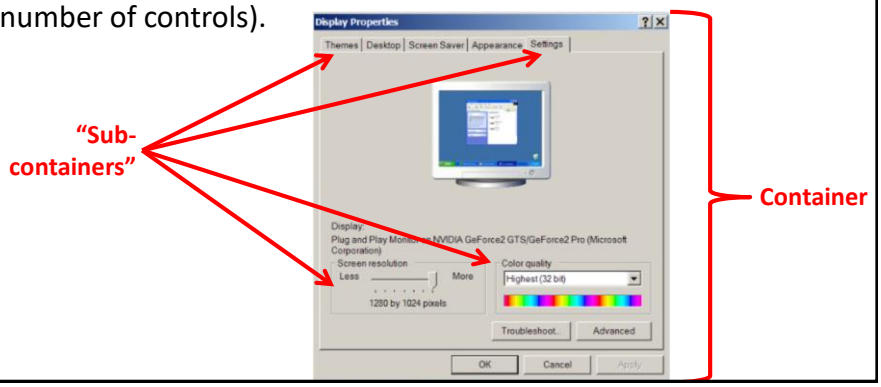
James Tam

Components

- They are many types of graphical controls and displays available:
  - `JButton, JFrame, JLabel, JList, JTextArea, Window`
- A graphical component is also known as a "widget"
- For Sun's online documentation refer to the url:
  - *http://download.oracle.com/javase/7/docs/api/* (especially `java.awt.event`, `javax.swing.event`, and `javax.swing`).

Containers

- A special type of `Component` that is used to hold/contain other components (subclass of the basic `Component` class).
- Can be used to group components on the screen (i.e., one container holds another container which in turn groups a number of controls).



Containers (2)

- You must have at least one container object for your GUI:
  - Examples: `JPanel`, `JWindow`, `JDialog`, `JFrame`
  - (The most likely one for the assignment is `JFrame`)
- Components which have been added to a container will appear/disappear and be garbage collected along with the container.

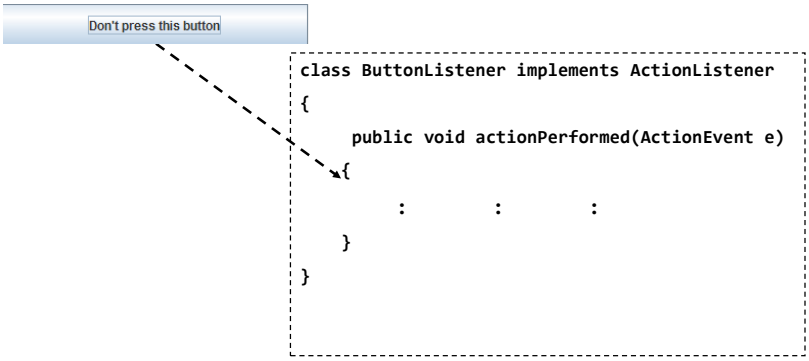
James Tam

Some Relevant Java GUI libraries

1. Java classes for the `Components` and `Containers`
   - e.g., `JButton` class…
   - …located in `javax.swing` (`import javax.swing.*` or `import javax.swing.<class name>`)

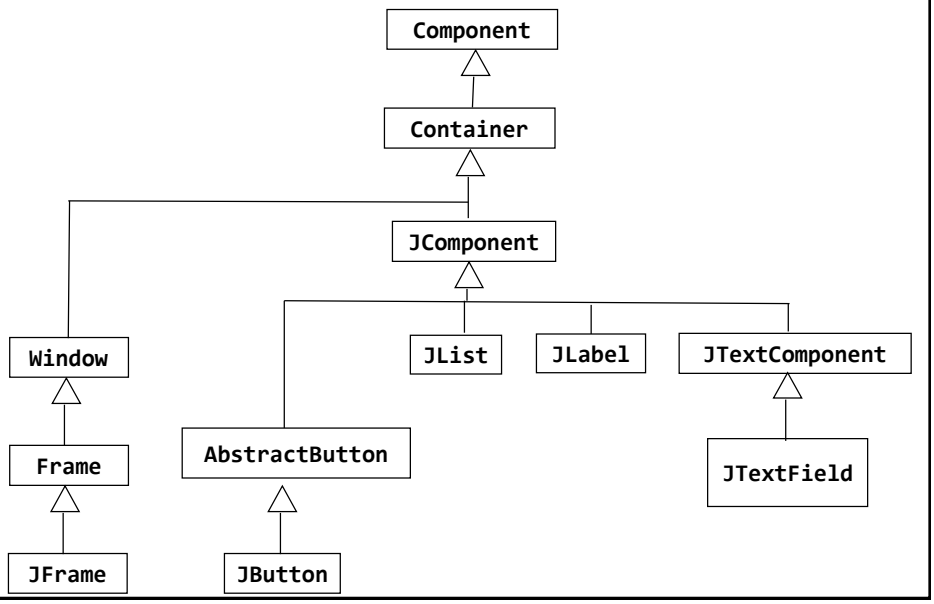
Some Relevant Java GUI libraries (2)

2. Java classes with the code to react to user-initiated events
   - e.g., code that executes when a button is pressed
   - `java.awt.event` (`import java.awt.event.*`, `import javax.swing.event.*`)

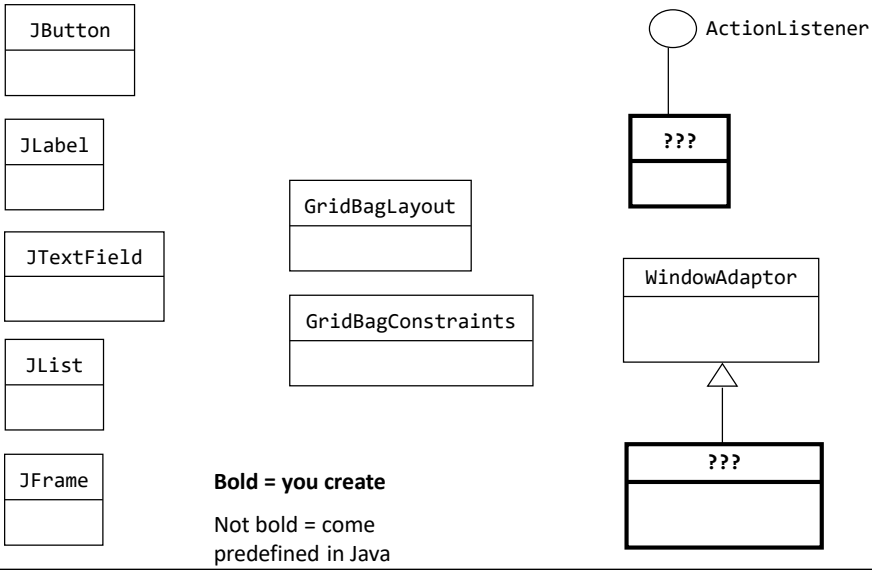


James Tam

Hierarchy: Important Widget Classes



Some Relevant Java GUI Classes For This Section



Traditional Software

- Program control is largely determined by the program through a series of sequential statements.

Example

```
:
if (num >= 0)
{
     // Statements for the body of the if
}
else
{
     // Statements for the body of the else
}
```

When num is non-negative

Num is negative

Traditional Software

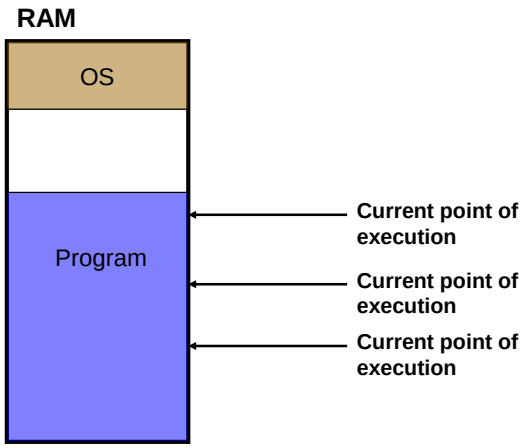
- The user can only interact with the program at places that are specified by the program (e.g., when an input statement is encountered).

Example

```
Scanner aScanner = new Scanner (System.in);
System.out.print("Enter student ID number: ");
id = aScanner.nextInt ();
```

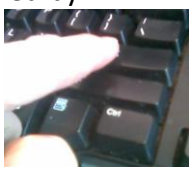
Event-Driven Software

- Program control can also be sequential



Event-Driven Software

- In addition program control *can also* be determined by events



When???

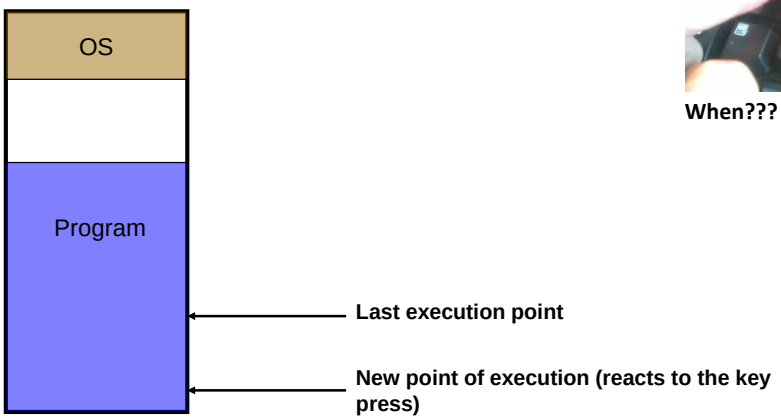


Image: Keyboard and "finger of Tam" by James Tam

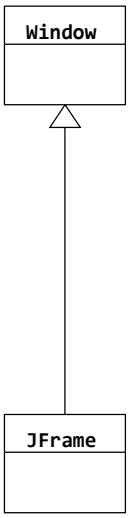
Characteristics Of Event Driven Software

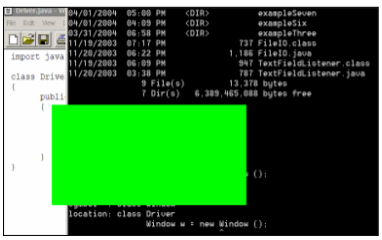
- Program control can be determined by events as well as standard program control statements.
- A typical source of these events is the user.
- These events can occur at any time.

Most Components Can Trigger Events

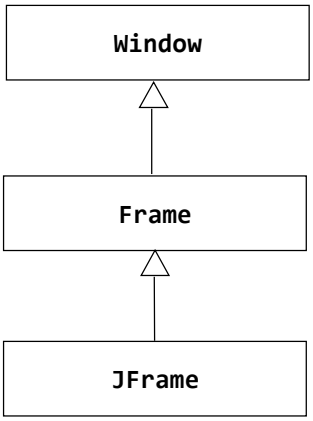
- Graphical objects can be manipulated by the user to trigger events.
- Each graphical object can have 0, 1 or many events that can be triggered.

"Window" Classes





The "Window" Class Hierarchy



Class `JFrame`

- For full details look at the online API:
  - http://download.oracle.com/javase/7/docs/api/javax/swing/JFrame.html
- Some of the more pertinent methods:
  - `JFrame ("<Text on the title bar>")`
  - `setSize (<pixel width>, <pixel height>)`
  - `setVisible (<true/false>)`
  - `setDefaultCloseOperation (<class constants>`[1]`)`

1 DISPOSE_ON_CLOSE, HIDE_ON_CLOSE, DO_NOTHING_ON_CLOSE

Example: Creating A Frame That Can Close (And Cleanup Memory After Itself)

- **Name of the folder containing the full example**: `1frame`

Example: Creating A Frame That Can Close (And Cleanup Memory After Itself)

```
import javax.swing.JFrame;
public class Driver
{
     public static void main (String [] args)
     {
           JFrame mf = new JFrame ("Insert title here");
           mf.setSize (300,200);
           mf.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
           mf.setVisible(true);
     }
}
```

Pitfall 1: Showing Too Early

- When a container holds a number of components the components must be added to the container (later examples).
- To be on the safe side the call to the "`setVisible()`" method should be done after the contents of the container have already been created and added.

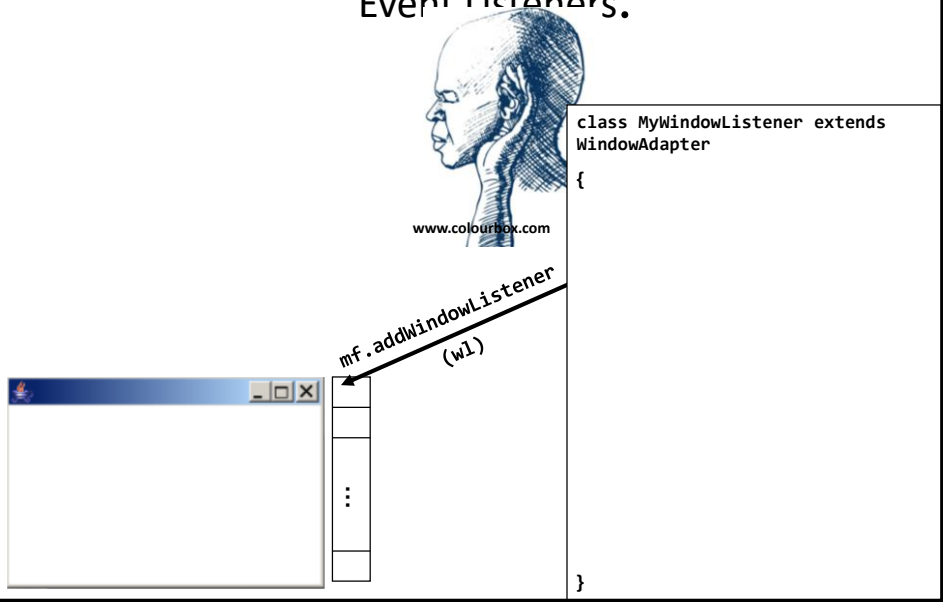
Window Events

- The basic `JFrame` class provides basic capabilities for common windowing operations: minimize, maximize, resize, close.
- However if a program needs to perform other actions (i.e., your own custom code) when these events occur the built in approach won't be sufficient.
  - E.g., the program is to automatically save your work to a file when you close the window.

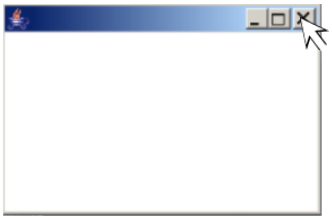
Steps In The Event Model For Handling A Frame Event: Window Closing

1) Define/instantiate the appropriate listener class/object.
2) The frame must register all interested event listeners.
   - Track where notifications should be sent
3) The user triggers the event by closing the window
4) The window sends a message to all listeners of that event.
   - Send the notifications when the even occurs
5) The window event listener runs the code to handle the event (e.g., save information to a file).
   - When the object with an 'interest' in the event has been notified it executes a method appropriate to react to the event.

2. The Frame Must Register All Interested Event Listeners.



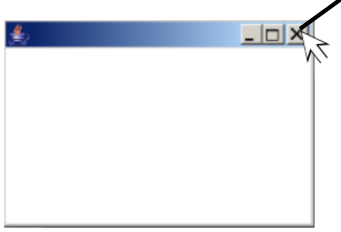
3. The User Triggers The Event By Closing The Window



4. The Window Sends A Message To All Listeners Of That Event.

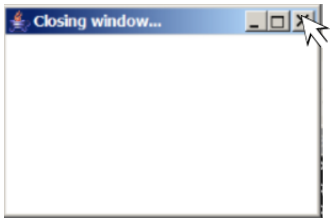
```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
           (WindowEvent e)
        {




         }
}
```

5. The Event Listener Runs The Code To Handle The Event.



```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
          (WindowEvent e)
        {
            /* Code to react to event * /
            JFrame aFrame = (JFrame)
              e.getWindow();
            aFrame.setTitle("Closing
              window...");
            aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

5. The Event Listener Runs The Code To Handle The Event.

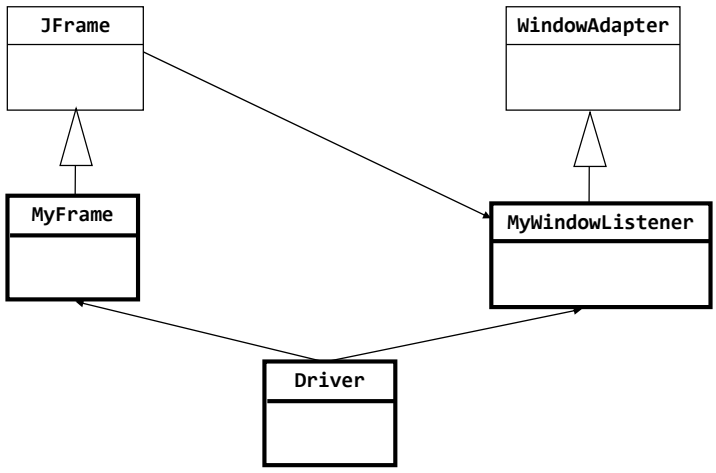



```
public class MyWindowListener extends
WindowAdapter
{
        public void windowClosing
          (WindowEvent e)
        {
            /* Code to react to event * /
            JFrame aFrame = (JFrame)
              e.getWindow();
            aFrame.setTitle("Closing
              window...");
            aFrame.setVisible(false);
            aFrame.dispose();
        }
}
```

An Example Of Handling A Frame Event

- **Name of the folder containing the full online example**:
  `2windowEvents`

An Example Of Handling A Frame Event (2)



The Driver Class

```
import javax.swing.JFrame;

public class Driver
{
     public static final int WIDTH = 300;
     public static final int HEIGHT = 200;
     public static void main (String [] args)
     {
         MyFrame aFrame = new MyFrame ();
         MyWindowListener aListener = new MyWindowListener() ;
         aFrame.addWindowListener(aListener);
         aFrame.setSize (WIDTH,HEIGHT);
         aFrame.setVisible(true);
     }
}
```

Class MyFrame

```
import javax.swing.JFrame;

public class MyFrame extends JFrame
{
     // More code will be added in later examples.
}
```

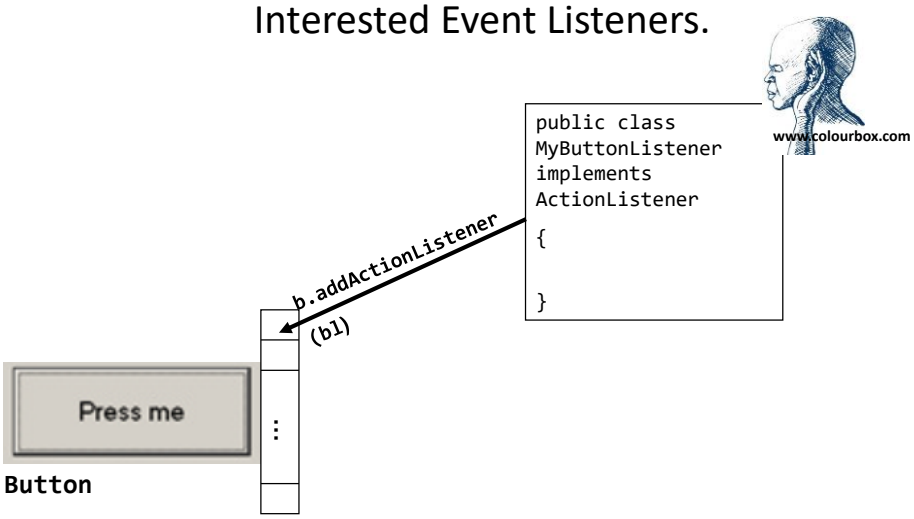
Class MyWindowListener

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class MyWindowListener extends WindowAdapter {
           public void windowClosing (WindowEvent e) {
           JFrame aFrame = (JFrame) e.getWindow();
           aFrame.setTitle("Closing window...");
           // Pause program so user can see the window text
           try
               Thread.sleep(3000);
           catch (InterruptedException ex)
               System.out.println("Pausing of program was
                 interrupted");
           aFrame.setVisible(false);
           aFrame.dispose();
       }
}
```

Steps In The Event Model For Handling A Button Event

1) Define/instantiate the appropriate listener class/object.
2) The button must register all interested event listeners.
3) The user triggers an event by pressing a button.
4) The button sends a message to all listeners of the button press event.
5) The button listener runs the code to handle the button press event.

2. The Graphical Component Must Register All Interested Event Listeners.



3. The User Triggers An Event By Pressing The Button

4. The Component Sends A Message To All Registered Listeners For That Event


```
public class MyButtonListener
implements ActionListener
{
    public void actionPerformed
      (ActionEvent e)
    {


    }
}
```

5. The Component Sends A Message To All Registered Listeners For That Event


```
public class MyButtonListener
implements ActionListener
{
    public void actionPerformed
      (ActionEvent e)
    {
        JButton b = (JButton)
          e.getSource();
        b.setLabel("Stop pressing
         me!");
    }
}
```

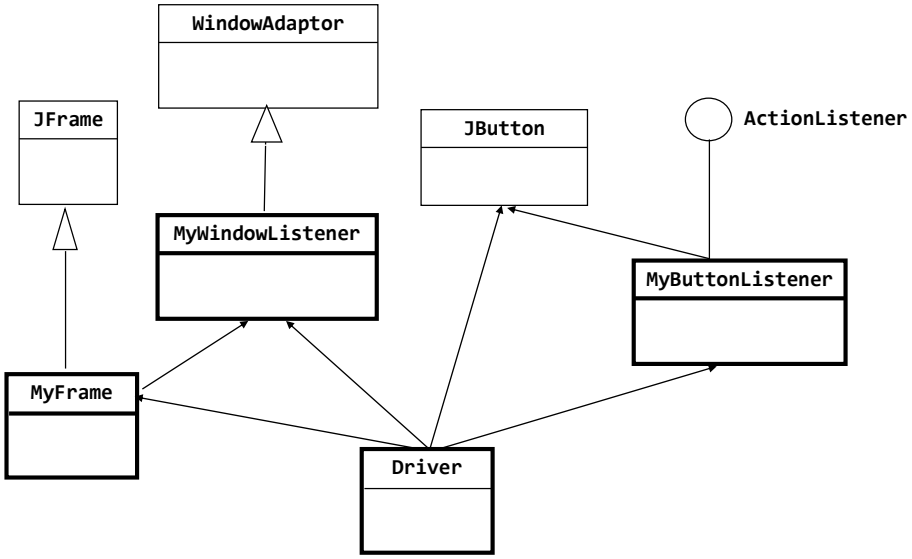
5. The Component Sends A Message To All Registered Listeners For That Event


```
public class MyButtonListener
implements ActionListener
{
    public void actionPerformed
      (ActionEvent e)
    {
        JButton b = (JButton)
          e.getSource();
        b.setLabel("Stop pressing
         me!");
    }
}
```

An Example Of Handling A Button Event

- **Name of the folder containing the full example**: `3ButtonEvents`

An Example Of Handling A Button Event (2)



An Example Of Handling A Button Event: The `Driver` Class

```
import javax.swing.JButton;

public class Driver
{
     public static final int WIDTH = 300;
     public static final int HEIGHT = 200;
     public static void main (String [] args)
     {
           MyFrame aFrame = new MyFrame ();
           MyWindowListener aWindowListener = new
             MyWindowListener();
           aFrame.addWindowListener(aWindowListener);
           aFrame.setSize (WIDTH,HEIGHT);
```

An Example Of Handling A Button Event: The Driver Class (2)

```
        JButton aButton = new JButton("Press me.");
        MyButtonListener aButtonListener =
          new MyButtonListener();
        aButton.addActionListener(aButtonListener);
        aFrame.add(aButton);
        aFrame.setVisible(true);
    }
}
```

An Example Of Handling A Button Event: The ButtonListener Class

```
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MyButtonListener implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        JButton aButton = (JButton) e.getSource();
        aButton.setText("Stop pressing me!");
    }
}
```

You Should Now Know

- The difference between traditional and event driven software
- How event-driven software works (registering and notifying event listeners)
- Getting a program to react to a close window and button press event via event listener classes.
- How to set some common properties of some GUI components: `JFrame`, JButton.