

GUI Programming with Java



Frame Based GUI's, Choosers and Mouse Events



GUI Programming with JAVA

Frame Based GUI's

- We will look at...
 - JFrame





GUI Programming with JAVA

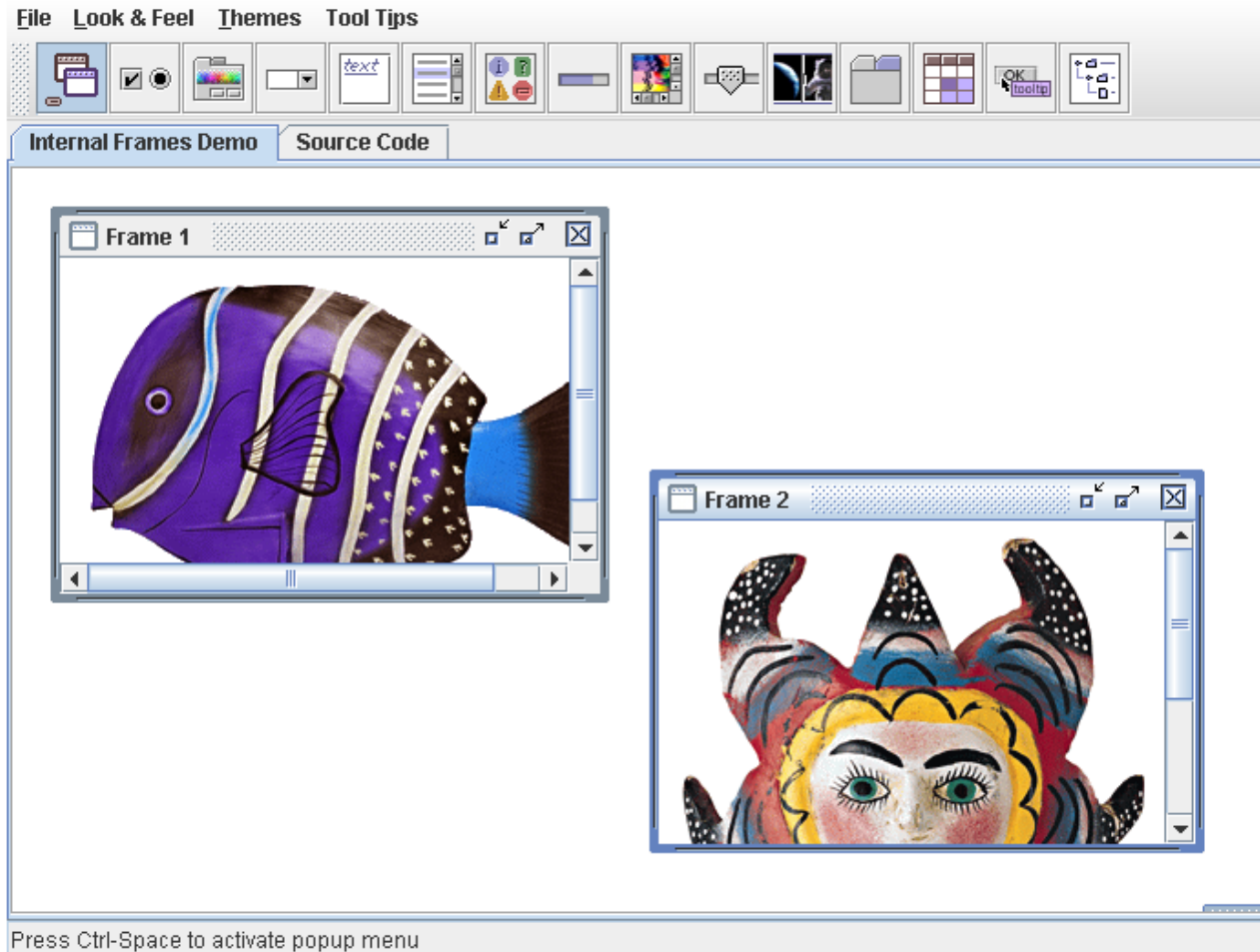
Frame Based GUI's

- Many GUI's applications simulate desktop environments by allowing multiple frames or windows to be displayed within a single root window.
- These frames look and in many cases behave like those on our normal desktop even though the window manager associated with the desktop is not aware of them (they are internal to the application)
- Typical examples include graphics applications, word processors, office applications etc.
- In this session we will examine the methods for creating internal frame based applications.



GUI Programming with JAVA

Frame Based GUI's



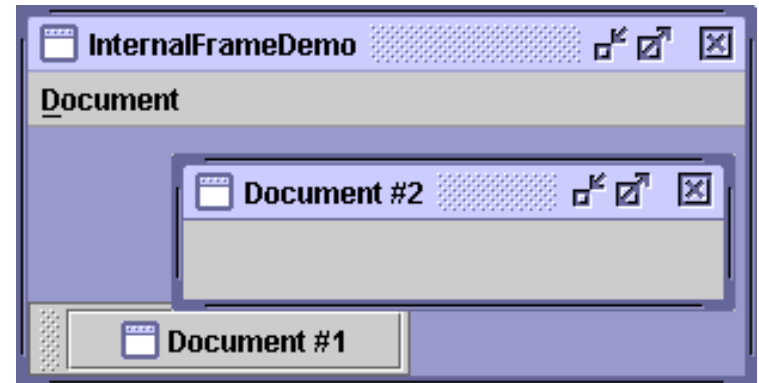
The above screenshot is from the swing demo we looked at. It makes use of internal frames.



GUI Programming with JAVA

Frame Based GUI's

- With the JInternalFrame class you can display a JFrame-like window within another window.
- Usually, you add internal frames to a desktop pane. (using JDesktopPane)
- The desktop pane, in turn, might be used as the content pane of a JFrame.





- Lets now take a look at

InternalFrameDemo1.java

which accompanies this lecture.



GUI Programming with JAVA

Frame Based GUI's

- We base our class on a JFrame. In other words, our application will consist of a JFrame that will in turn consist of a number of inner frames
- We import some new libraries to allow us to make use of the internal frame and the desktop pane

```
import javax.swing.JInternalFrame;  
import javax.swing.JDesktopPane;  
import javax.swing.JFrame;
```



- We begin by creating a JDesktopPane object

```
JDesktopPane desktop = new JDesktopPane();
```

- This is simply a pane on which we can add content, only in this case it will be frames.



GUI Programming with JAVA

Frame Based GUI's

- We then create an internal frame

```
JInternalFrame innerframe = new JInternalFrame("Internal Window" ,  
                                                true, //resizable  
                                                true, //closable  
                                                true, //maximizable  
                                                true); //iconifiable
```

- When creating a frame we specify its title, whether it is resizable, closable, maximizable or iconifiable



- Just like a standard JFrame, we can set the size and visibility and other properties of the JInternalFrame

```
innerframe.setSize(300,300);
```

```
innerframe.setVisible(true); //necessary as of 1.3
```

- We then get the desktop to act as the content pane for the JFrame

```
setContentPane(desktop);
```



- Most frame based applications use menu's so lets expand our program and add a menu to it.
- Lets now take a look at

InternalFrameDemo2.java

which accompanies this lecture.



- You can see that the menu does not do much.
- However the methods for adding menus are exactly the same as standard frames.
- Remember that our program is based on a JFrame



A more complex example

- Take a look at the program
InternalFrameDemo.java
- It makes use of another class called
MyInternalFrame.java
- This example allows us to create a series of new windows using a menu.



Rules of using Internal Frames

- **You must set the size of the internal frame.**
 - If you don't set the size of the internal frame, it will have zero size and thus never be visible. You can set the size using one of the following methods: `setSize`, `pack`, or `setBounds`.
- **As a rule, you should set the location of the internal frame.**
 - If you don't set the location of the internal frame, it will come up at 0,0 (the upper left of its container). You can use the `setLocation` or `setBounds` method to specify the upper left point of the internal frame, relative to its container.



Rules of using Internal Frames

- **To add components to an internal frame, you add them to the internal frame's content pane.**
 - This is exactly like the JFrame situation.
- **Dialogs that are internal frames should be implemented using JOptionPane or JInternalFrame, not JDialog.**
 - To create a simple dialog, you can use the JOptionPane showInternalXxxDialog methods.



Rules of using Internal Frames

- **You must add an internal frame to a container.**
 - If you don't add the internal frame to a container (usually a JDesktopPane), the internal frame won't appear.
- **You need to call setVisible on internal frames.**
 - Internal frames are invisible by default. You must invoke setVisible(true) to make them visible.



- **Internal frames fire internal frame events, not window events.**
 - Handling internal frame events is almost identical to handling window events.
 - Visit <https://docs.oracle.com/javame/config/cdc/opt-pkgs/api/agui/jsr209/javax/swing/event/InternalFrameEvent.html> for more information on internal frame events.



- For more information on internal frames visit

<http://download.oracle.com/javase/tutorial/uiswing/components/internalframe.html#construct>



GUI Programming with JAVA

Frame Based GUI's

Some useful methods

`setJMenuBar`

you can add menus to internal frames

`setLocation`

position the frame

`setBounds`

Explicitly set the size and location of the internal frame.

`setSize`

Explicitly set the size of the internal frame.

`moveToFront()` or `moveToBack()`

If the internal frame's parent is a layered pane such as a desktop pane, moves the internal frame to the front or back (respectively) of its layer.



Some useful methods

`setClosed(boolean)`

Set or get whether the internal frame is currently closed. The argument to `setClosed` must be true.

`setFrameIcon(Icon)`

Set the icon displayed in the title bar of the internal frame (usually in the top-left corner).

`JInternalFrame[] getAllFrames()`

Returns all `JInternalFrame` objects that the desktop contains.



Event Handling for Internal Frames

- An `InternalFrameListener` is similar to a `WindowListener`.
- Like the window listener, the internal frame listener listens for events that occur when the "window" has been shown for the first time, disposed of, iconified, deiconified, activated, or deactivated.
- Take a look at `InternalFrameEventDemo.java` for a sample of how we can handle internal frame events.



GUI Programming with JAVA

Frame Based GUI's

- We will look at...
 - Using file chooser and color chooser
 - Mouse events



File Choosers

- File choosers provide a GUI for navigating the file system.
- They allow a user to choose a file or directory from a list or entering the name of a file or directory.
- To display a file chooser, you usually use the JFileChooser component to show a modal dialog containing the file chooser.
- Another way to present a file chooser is to add an instance of JFileChooser to a container.



GUI Programming with JAVA

Frame Based GUI's

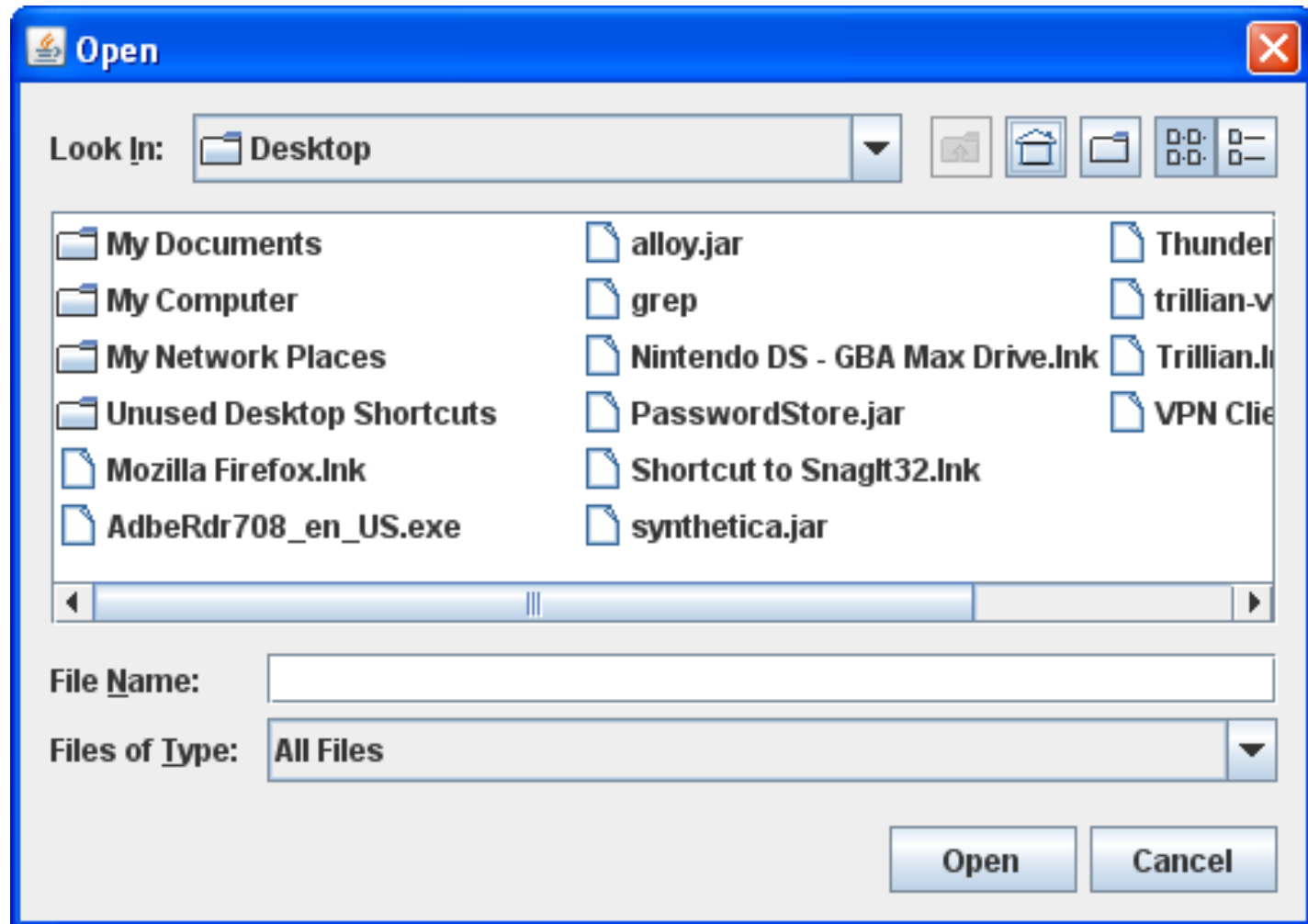
- A JFileChooser object only presents the GUI for choosing files.
- Your program is responsible for doing something with the chosen file, such as opening or saving it.
- The JFileChooser makes it easy to bring up open and save dialogs.
- The look and feel determines what these standard dialogs look like and how they differ.



GUI Programming with JAVA

Frame Based GUI's

- Here is a picture of the Java look and feel's standard open dialog:

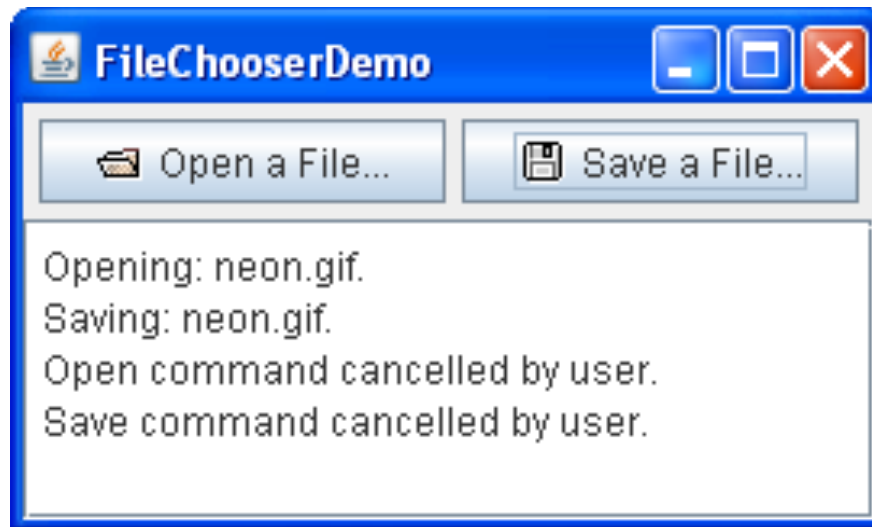




GUI Programming with JAVA

Frame Based GUI's

- Lets now take a look at an example program which brings up an open dialog and a save dialog.



- Let's take a look at FileChooserDemo.java in the sample 1 - file chooser folder.



GUI Programming with JAVA

Frame Based GUI's

- Bringing up a standard open dialog requires only two lines of code:

//Create a file chooser

```
final JFileChooser fc = new JFileChooser();
```

...

//In response to a button click:

```
int returnVal = fc.showOpenDialog(aComponent);
```



GUI Programming with JAVA

Frame Based GUI's

int returnVal = fc.showOpenDialog(aComponent);

- The argument to the showOpenDialog method specifies the parent component for the dialog.
- The parent component affects the position of the dialog and the frame that the dialog depends on.
- For example, the Java look and feel places the dialog directly over the parent component.
- If the parent component is in a frame, then the dialog is dependent on that frame, disappearing when the frame is iconified and reappearing when the frame is deiconified.



Setting the directory

- By default, a file chooser that hasn't been shown before displays all files in the user's home directory. You can specify the file chooser's initial directory using one of JFileChooser's other constructors, or you can set the directory with the `setCurrentDirectory` method.
- Lets take a look at some of the constructors that may be of use to us.



- Constructor Summary

JFileChooser()

- Constructs a JFileChooser pointing to the user's default directory.

JFileChooser(File currentDirectory)

- Constructs a JFileChooser using the given File as the path.

JFileChooser(String currentDirectoryPath)

- Constructs a JFileChooser using the given path.



GUI Programming with JAVA

Frame Based GUI's

- The call to `showOpenDialog` appears in the `actionPerformed` method of the Open a File... button's action listener:

```
public void actionPerformed(ActionEvent e) {  
    //Handle open button action.  
    if (e.getSource() == openButton) {  
        int returnVal = fc.showOpenDialog(FileChooserDemo.this);  
        if (returnVal == JFileChooser.APPROVE_OPTION) {  
            File file = fc.getSelectedFile();  
            //This is where a real application would open the file.  
            log.append("Opening: " + file.getName() + "." + newline);  
        }  
        else {  
            log.append("Open command cancelled by user." + newline);  
        }  
    }  
    ... }  
}
```



GUI Programming with JAVA

Frame Based GUI's

- Using the file object
- Remember that last semester you studied File IO which would have included a look at the File class.
- You can call other methods on the File object, such as getPath, isDirectory, or exists to get information about the file.
- You can also call other methods such as delete and rename to change the file in some way.
- Of course, you might also want to open or save the file using one of the reader or writer classes provided by the Java platform.



GUI Programming with JAVA

Frame Based GUI's

- The example program uses the same instance of JFileChooser to display a standard save dialog. This time the program calls showSaveDialog:

```
int returnVal = fc.showSaveDialog(FileChooserDemo.this);
```

By using the same file chooser instance to display its open and save dialogs, the program reaps these benefits:

- The chooser remembers the current directory between uses so the open and save versions automatically share the same current directory.
- You have to customize only one file chooser, and the customizations apply to both the open and save versions of it.



GUI Programming with JAVA

Frame Based GUI's

- Finally, the example program has commented-out lines of code that let you change the file selection mode.
- For example, the following line of code makes the file chooser able to select only directories, and not files:

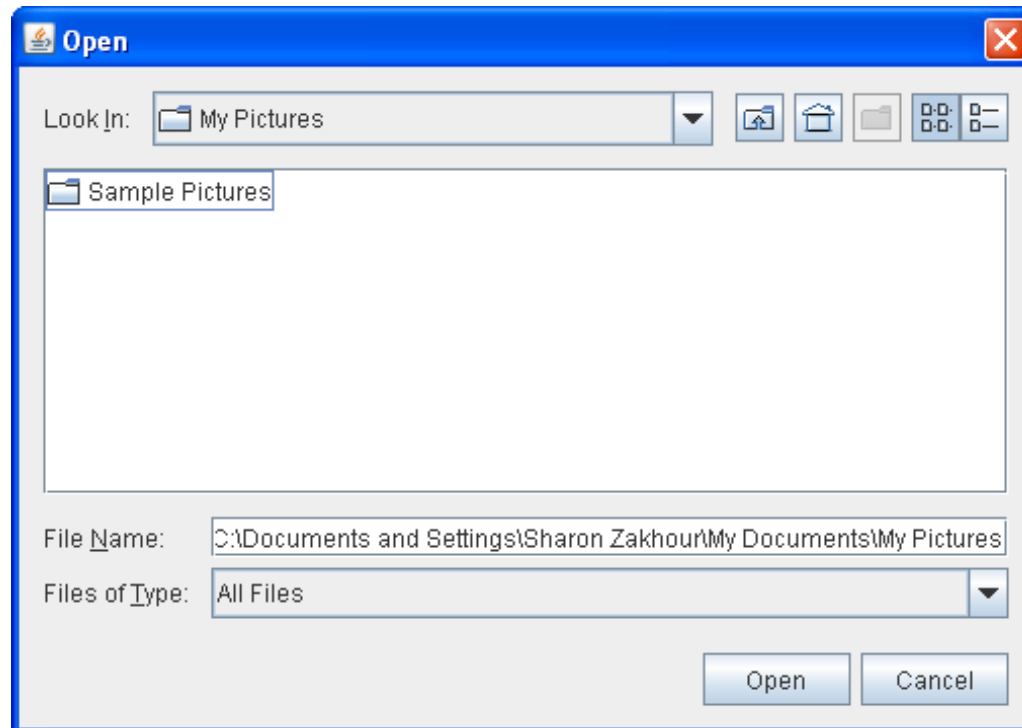
```
fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
```



GUI Programming with JAVA

Frame Based GUI's

- Another possible selection mode is FILES_AND_DIRECTORIES. The default is FILES_ONLY.
- The following picture shows an open dialog with the file selection mode set to DIRECTORIES_ONLY.
- Note that, in the Java look and feel at least, only directories are visible — not files.





GUI Programming with JAVA

Frame Based GUI's

- Colour Choosers
- Colour choosers can be used to pick from available colours
- For example if you were to set the background colour of a panel or allow the user to choose the colours they want in an interface
- JColorChooser can be used for this purpose
- You can then set the return colour to a Color variable and use it in your program: see the example called `ColorChooserExample.java` to see how to use JColorChooser



GUI Programming with JAVA

Frame Based GUI's

- ***Mouse Listeners and Mouse Events***



GUI Programming with JAVA

Frame Based GUI's

- There are numerous Mouse event classes and interfaces which are available in Java
- Each of them attempt to handle different events using the Mouse
- Mouse events are those associated with clicking, dragging, entering, exiting etc.
- Mouse motion events are events that involve the mouse tracking across the screen, i.e., moving and dragging



GUI Programming with JAVA

Frame Based GUI's

- Some of the classes/interfaces to deal with Mouse events are listed below; many of these will be covered in this lecture (most are in the `java.awt.event` package):

MouseListener

MouseMotionListener

MouseEvent

MouseListener

MouseWheelListener

MouseAdapter

MouseInputAdapter



MouseListener

- The MouseListener interface is used to respond to user mouse events
- Specifically it contains handlers for 5 events, namely, mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased



MouseListener

`mouseClicked(MouseEvent)`

- Called when the mouse button has been clicked (pressed and released) on a component.

`mouseEntered(MouseEvent)`

- Called when mouse enters a component

`mouseExited(MouseEvent)`

- Called when mouse exits a component

`mousePressed(MouseEvent)`

- Called when a mouse is pressed in a component

`mouseReleased(MouseEvent)`

- Called when a mouse is released in a component



MouseListener

- In order to listen for events at the class level using `MouseListener` you must add *implements `MouseListener`* to the class (import the `java.awt.event` package)
- Once you implement this you will have to implement all five handler methods
- You need only fully implement the handler method you are interested in, the others can be left blank...the compiler will not allow you to compile if you have not implemented all of the handler methods



MouseListener

```
public class BlankMouseListener implements MouseListener {  
  
    public void mouseClicked(MouseEvent e) { //implement here }  
  
    public void mouseEntered(MouseEvent e) { //implement here }  
  
    public void mouseExited(MouseEvent e) { //implement here }  
  
    public void mousePressed(MouseEvent e) { //implement here }  
  
    public void mouseReleased(MouseEvent e) { //implement here }  
  
}
```



MouseMotionListener

- This listener is used to respond to the movement of the mouse as it tracks across the screen
- These include the dragging and move events
- The two handle methods are:

mouseDragged(MouseEvent e)

mouseMoved(MouseEvent e)



MouseMotionListener

```
public class BlankMouseMotionExample implements MouseMotionListener {  
  
    public void mouseDragged(MouseEvent e) {}  
  
    public void mouseMoved(MouseEvent e) {}  
  
}
```



MouseEvent

- In the two previous interfaces you can observe that a MouseEvent parameter is passed to all of the handler methods shown
- The MouseEvent class is identical in its function to the(ActionEvent parameter that we've already seen numerous times when using buttons
- Just like the(ActionEvent parameter the MouseEvent parameter receives an object of this type which represents the event which has occurred...and importantly contains information relating to the event that took place!



MouseEvent

- The MouseEvent parameter is given a name, this reference name can be used to retrieve information about the event that took place
- Many of the methods are the same as ActionEvent, however, many of them are specific to the mouse
- Some of the mouse specific events are listed in the next slide [you can find them all online at: [MouseEvent at Oracle Site](#)]



MouseEvent

- Assuming that `e` is the name given to the `MouseEvent` parameter below are examples of information that can be retrieved from the `MouseEvent` object:
- `e.getX();` //The x co-ordinate of where the mouse event occurred within the component that has the listener added
- `e.getY();` //The y co-ordinate of where the mouse event occurred within the component that has the listener added
- `e.getXOnScreen();` //The x co-ordinate of where the mouse event occurred within the program screen [absolute x position]
- `e.getYOnScreen();` //The y co-ordinate of where the mouse event occurred within the program screen [absolute y position]



MouseListener

- This interface allows us to implement both the MouseListener and MouseMotionListener at the same time
- Implementing this interface requires the implementation of the five MouseListener handler methods AND the two MouseMotionListener handler methods
- The MouseInputListener is found in the javax.swing.event package
[see more here [MouseListener on Oracle site](#)]



MouseWheelListener

- This interface can be implemented to respond to mouse wheel events
- There is only one handler in this interface, namely,

mouseWheelMoved(MouseWheelEvent e)



MouseAdapter

- Sometimes you may wish to only implement one-or-two of the mouse methods
- An *Adapter* is a design pattern which allows use to adapt classes that already exist in some way
- The *MouseAdapter* class implements the `MouseListener`, `MouseMotionListener` and `MouseWheelListener` interfaces
- You can extend the `MouseAdapter` and you need only implement the handler you wish (example on next slide)



MouseAdapter

```
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;
```

```
public class BlankMouseAdapterExample extends MouseAdapter {
```

```
    public void mouseClicked(MouseEvent e) {} //Compiler happy to allow just one
```

```
}
```



MouseAdapter

- The advantage of the MouseAdapter is that only the method/ methods that are required need to be implemented and so the code is much less congested
- The disadvantage of the MouseAdapter is that it requires the use of the extends keyword which limits the inheritance to ONE class
- Therefore if your class needs to inherit from another class (e.g. JFrame) you cannot inherit from another and the implements may have to be used (or a second external listener)



Other Adapters in GUI Programming

- The following are some of the adapter classes available in Java GUI programming, all with same purpose:

- *ComponentAdapter*
- *ContainerAdapter*
- *FocusAdapter*
- *KeyAdapter*
- *MouseAdapter*
- *MouseMotionAdapter*
- *MouseInputAdapter*
- *WindowAdapter*



Making the Mouse events respond

- The steps are the same as for other listeners
- Implement the listener
- Add the listener to the component
- Implement the handler methods required
- See the example on Moodle called `MouseListenerExample.java`