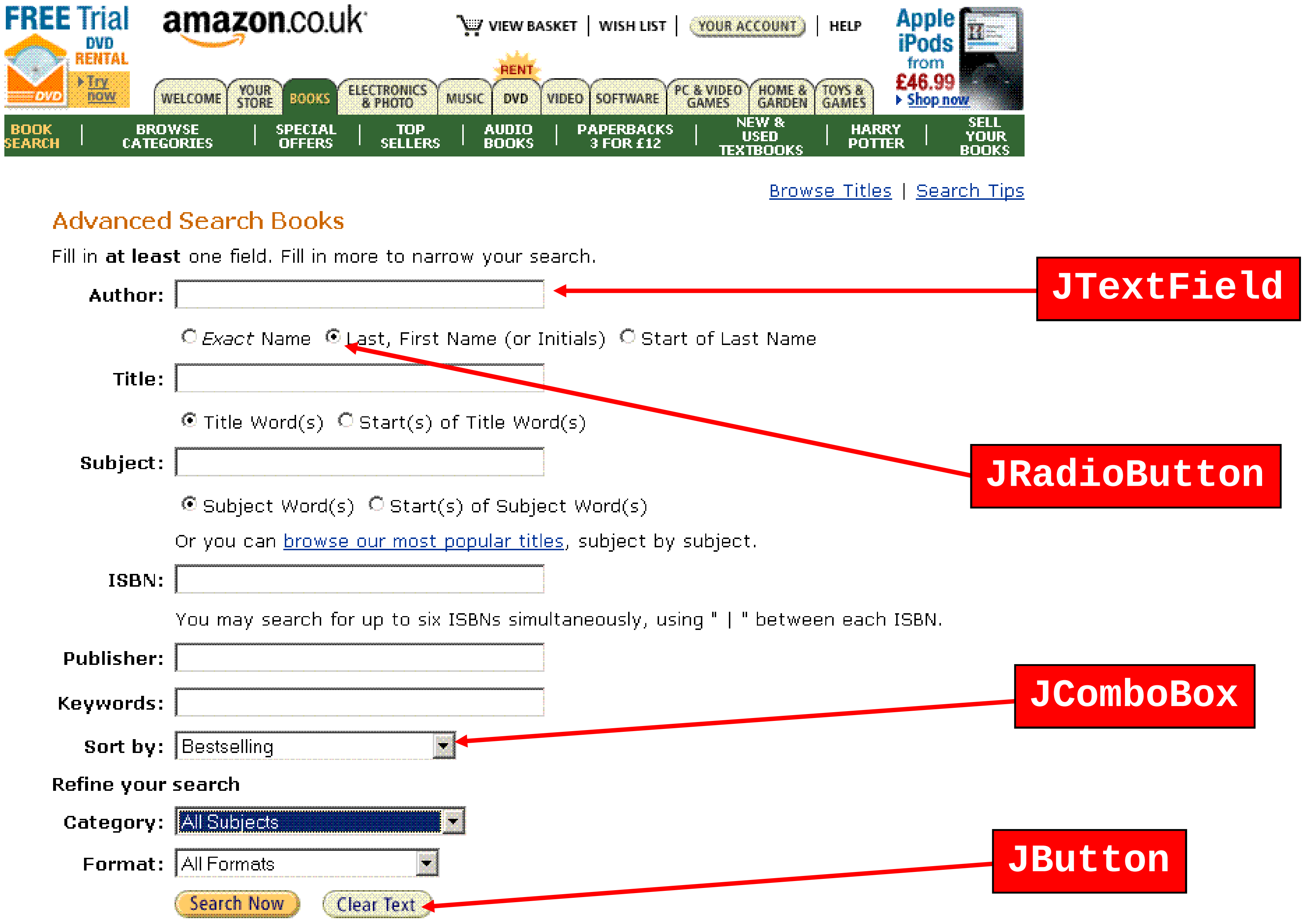
Graphical User Interfaces



Browse Titles | Search Tips

Advanced Search Books

Fill in **at least** one field. Fill in more to narrow your search.

Author:

○ *Exact* Name [● Last, First Name (or Initials)] ○ Start of Last Name

Title:

[● Title Word(s)] ○ Start(s) of Title Word(s)

Subject:

[● Subject Word(s)] ○ Start(s) of Subject Word(s)

Or you can browse our most popular titles, subject by subject.

ISBN:

You may search for up to six ISBNs simultaneously, using " | " between each ISBN.

Publisher:

Keywords:

Sort by: Bestselling

Refine your search

Category: All Subjects

Format: All Formats

GUI Programs Are Different

- They are graphical. The user can interact with several user interface objects (***widgets*** for short).
- The user controls the sequence of activities.
- The program displays the user interface and waits.
- The user has several different choices of what to do next.

- Up to now our programs have been in control.
- The program tells the user what to do next.
- The user can only do what the program says.
- They do not have a choice.

Event Based Programming

- The program sets up its initial screen (window), complete with widgets, and waits.
- The user interacts with the program, generating an ***event***.
  - Presses a key, a key event.
  - Clicks the mouse, a mouse press event.
  - Moves the mouse, a mouse move event.
  - Resizes the window, one of many window events.
  - There are other events as well.
- The widget passes the event to all its ***listeners***.
- The listeners process the event.
- Then the program waits for the next event.

In Java

- The program creates a window object, the basis for interaction.
  - There can be more than one window.
- The program creates several widget objects.
  - These widgets are given names and other properties.
- Each widget is added to the window.
  - It must be positioned in the right place.
- Listener objects are added to each widget.
  - They process each event when it arrives.

Class JFrame

- A `JFrame` object is an independent window or screen.
- Each GUI program must have at least one `JFrame` object.
  - A complicated program will have several screens, each is a different `JFrame` object.
- We must customise the Java `JFrame` class.
  - `public class ButtonFrame extends JFrame`
- The Java keyword `extends` means that our new class will
  - Build on `JFrame`, extending it.
  - Add our own application specific code.
- Extending an existing class means that we can use all of the methods that the existing class has.

Methods of JFrame

- `setTitle(String);`
- `setSize(int dimx, int dimy);`
  - The number of pixels in the x and y directions.
  - We must know the screen resolution before we write the program.
- `setLocation(int x, int y);`
  - The distance of the top left hand corner of the frame from the top left hand corner of the monitor, in pixels.
- `setVisible(boolean);`
  - You won’t see the window unless you `setVisible(true)`.
  - You can make it invisible by `setVisible(false)`.
  - This must be the last thing we do in our constructor.

Creating a JFrame

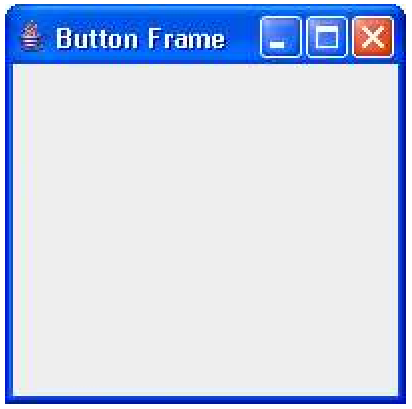
- Let's call our new class `ButtonFrame`.
  - We need a new file called `ButtonFrame.java`, as usual.
- We need to import `javax.swing.*` to get `JFrame`.

```
import javax.swing.*;
public class ButtonFrame extends JFrame
{
   public ButtonFrame()
   {
        setTitle("Button Frame");
        setSize(200, 200);
        setLocation(700, 700);}
        setVisible(true);
   }
}
```

The Main Class

- The main class just creates a `ButtonFrame` object.

```
public class Ex1
{
  public static void main(String[] arg)
  {
      ButtonFrame bf = new ButtonFrame();
  }
}
```



JButton

- A `JButton` object is a button.
- It has a name.
- We can press it.

- Let us declare 2 buttons in our `ButtonFrame`

```
    private JButton button1, button2;
```

- We can create them in the `ButtonFrame` constructor.

```
    button1 = new JButton("Press Me");
    button2 = new JButton("Press Me Too");
```

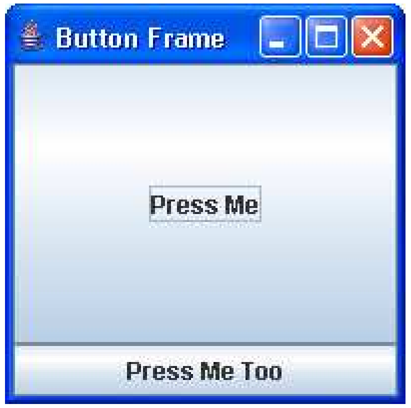
Positioning The Buttons

- Where should the two buttons go?
- There are many different ***layout managers*** in Java.
- The default `JFrame` layout is called ***Border Layout***.
- The screen is divided into a centre and four edges.
  - `“Center”` (note American spelling).
  - `“North”`
  - `“South”`
  - `“East”`
  - `“West”`
- Each can hold one widget.
  - But we can fit more in using panels (later).

Adding Buttons To A Frame

- Let us add the “Press Me” button to the centre and the “Press Me Too” button to the south.

```
add(button1, "Center");
add(button2, "South");
```



Action Listener

- If we press either button, nothing will happen.
  - The user delivers an event to the button,
  - But no one is listening.
- We must add a listener object to the button.
- Each button press generates an `ActionEvent` object.
- So we must arrange for an `ActionListener` object to listen to each button.
- We must define our own class that customises the Java `ActionListener` class.
  - Similar to the way we customised `JFrame`.
- We must `import java.awt.event.*`

ButtonFrame as ActionListener

- The easiest way to get a listener object is to use our `JFrame` object to listen for `ActionEvents`.
- `ButtonFrame` must implement `ActionListener` as well as extend `JFrame`.

```
public class ButtonFrame extends JFrame
  implements ActionListener
```

- We can then get it to listen to each button.

```
  button1.addActionListener(this);
  button2.addActionListener(this);
```

- Remember that `this` is the `ButtonFrame` object.

Action Performed

- Finally, we have to define what is done when an `ActionEvent` is delivered to a listener.
- This is done by writing a method called `actionPerformed`.
- Each object that is listening to a button must have an `actionPerformed` method.
- In our example, the `ButtonFrame` object is listening to both buttons and so must define the method.

```
public void actionPerformed(ActionEvent e)
{
    System.err.println("Ouch");
}
```

Which Button

- In the example, the `ButtonFrame` object is listening to both buttons.
- So the `actionPerformed` method is called no matter which button is pressed.
- How can we tell the difference?

- We can work it out from the `ActionEvent` object.
  - It knows which button was pressed.
- The method `getSource()` return the button object that was pressed.
- We just use an `if` statement to test which one it was.

Responding To The Correct Button

```
public void actionPerformed(ActionEvent e)
{
    if (e.getSource() == button1)
          System.err.println("Ouch");
    else // must be button2
          System.err.println("Stop It");
}
```

Summary

- `JFrame`: Java class that defines a window.
- `JButton`: Java class that defines a button.
- `ActionListener`: Java class that defines a JButton listener.

- `ButtonFrame`: our window that extends `JFrame` and implements `ActionListener`.

- `actionPerformed`: method that is called when a button press event is delivered to a listener.
- `ActionEvent`: Java name for a button press event.