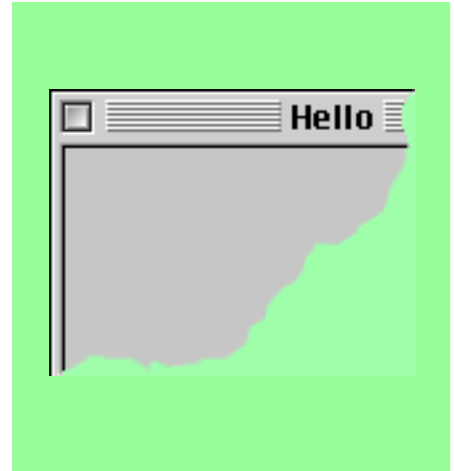


---

# Application programming & UI layout

---



*Applications aren't just for text anymore*

# Minimalist Java

The smallest legal Java application:

```
public class X {  
    public static void main(String a[ ]) {  
    }  
}
```

The method `main` is required and invoked by JRE

When Java is started...

1. The JVM is started
2. A runtime object is created
3. Your program's class is loaded...
4. ...and linked into the runtime object
5. Several threads are started, e.g.,
  - a. garbage collection thread
  - b. finalization thread
  - c. the main thread, which...
6. ...invokes `yourClass.main`

# Skeletal Java

```
public class X {
```

```
    more stuff here – class-wide declarations
```

```
    public static void main(String a[]) {
```

```
        more stuff here – main method code
```

```
    }
```

```
    more stuff here – other methods of the class
```

```
}
```

```
more stuff here (and in other files) – more classes
```

# More minimalist Java

A very simple Java graphic application:

```
import javax.swing.*;  
  
public class SimpleGraphicApp {  
    public static void main(String a[]) {  
        JFrame f = new JFrame("Hello");  
        f.setSize(200, 100);  
        f.setVisible(true);  
    }  
}
```



# Framed windows

A `JFrame` is an object that is an instance of the `javax.swing.JFrame` class

It's a window with all the usual trimmings (title bar, resize box, etc.)

It's a place where program stuff can appear

# Some **JFrame** methods

`setVisible`      *makes the frame appear (or disappear)*

`setSize`          *makes the frame a certain size  
(by default, it's 0 by 0)*

`setLocation`      *sets the position on the screen*

`setBounds`        *sets the position and size*

*...and lots more*

# Frame contents

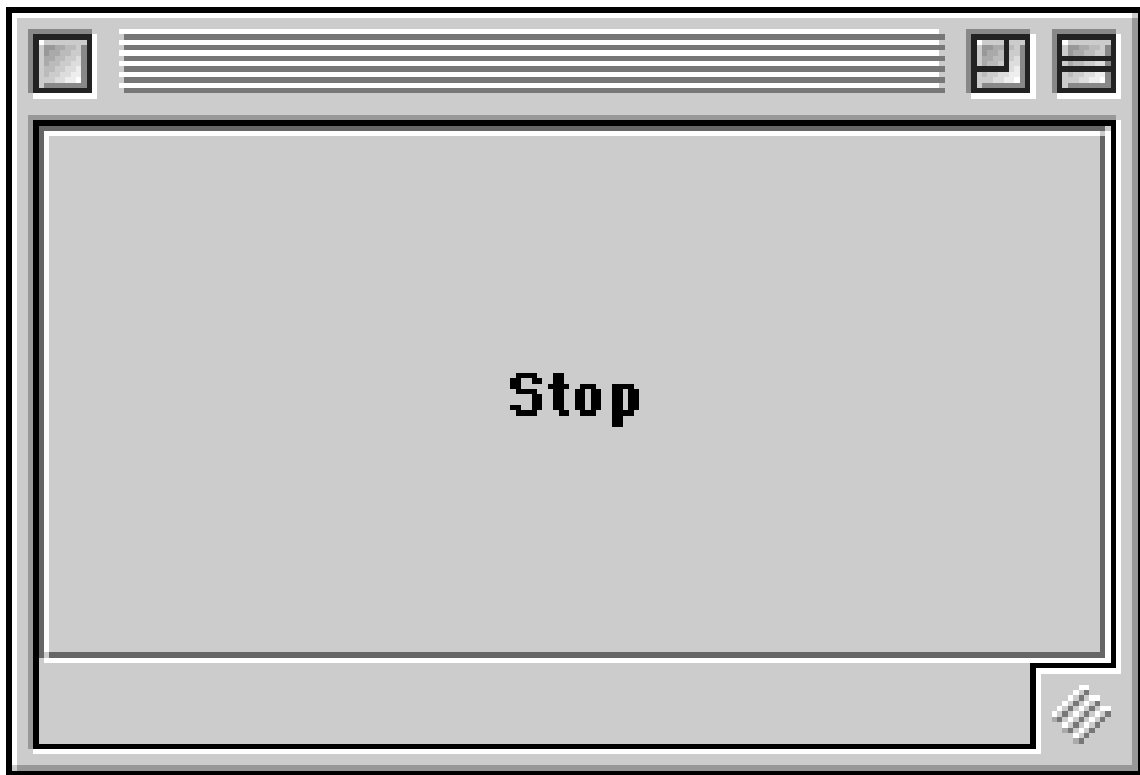
A `JFrame` can contain UI things  
like buttons, checkboxes, text fields, etc.

Such things are added to a `JFrame`  
using the `add` method on its `ContentPane`  
(which is an object of class `java.awt.Container`)

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

public class SimpleGraphicApp {

    public static void main(String a[]) {
        JFrame f = new JFrame();
        JButton b = new JButton("Stop");
        Container cp = f.getContentPane();
        cp.add(b);
        f.setSize(200, 100);
        f.setVisible(true);
    }
}
```



# Coding variation

Instead of

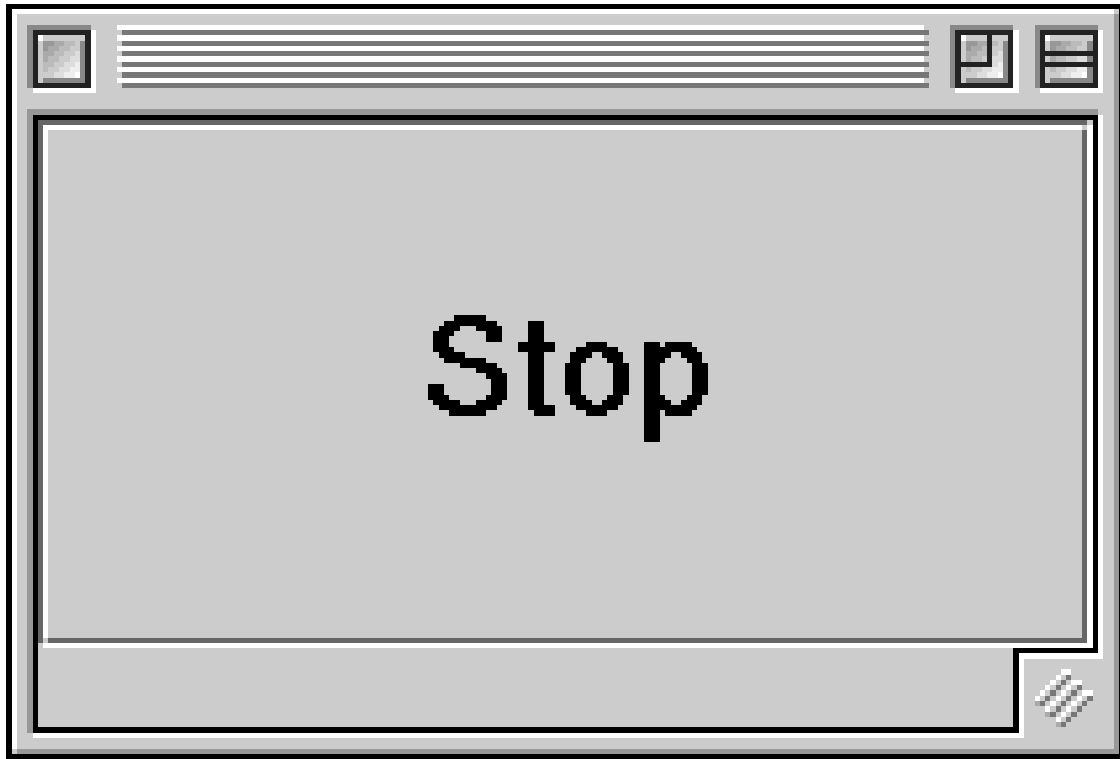
```
Container cp = f.getContentPane();  
cp.add(b);
```

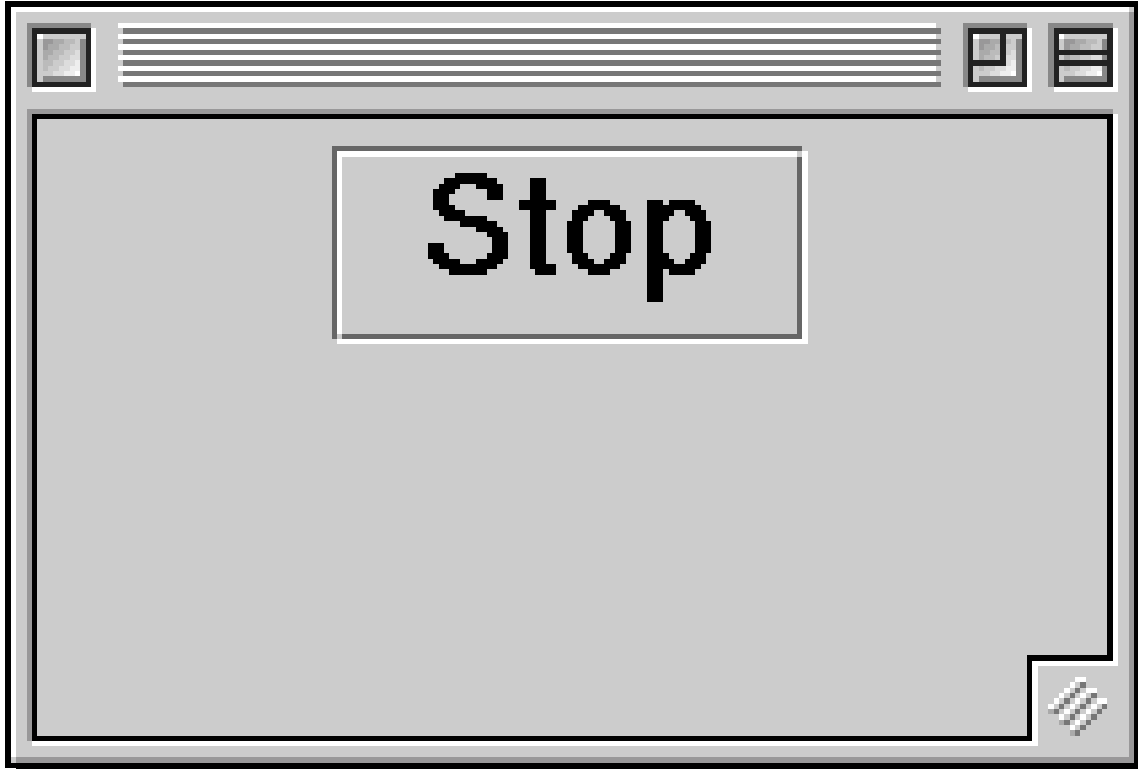
could just do

```
f.getContentPane().add(b);
```

```
import javax.swing.*;  
  
public class SimpleGraphicApp {  
  
    public static void main(String a[]) {  
        JFrame f = new JFrame();  
        JButton b = new JButton("Stop");  
        f.getContentPane().add(b);  
        f.setSize(200, 100);  
        f.setVisible(true);  
    }  
}
```

```
import javax.swing.*;  
import java.awt.*;  
  
public class SimpleGraphicApp {  
    public static void main(String a[]) {  
        JFrame f = new JFrame();  
        JButton b = new JButton("Stop");  
        b.setFont(new Font("Helvetica", Font.BOLD, 24));  
        f.getContentPane().add(b);  
        f.setSize(200, 100);  
        f.setVisible(true);  
    }  
}
```





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

public class SimpleGraphicApp {

    public static void main(String a[]) {
        JFrame f = new JFrame();
        Container cp = f.getContentPane();
        cp.setLayout(new FlowLayout());
        JButton b = new JButton("Stop");
        b.setFont(new Font("Helvetica", Font.BOLD, 24));
        cp.add(b);
        f.setSize(200, 100);
        f.setVisible(true);
    }
}
```

# Layout management

It's possible to specify where each GUI component should go, but it's almost never a good idea

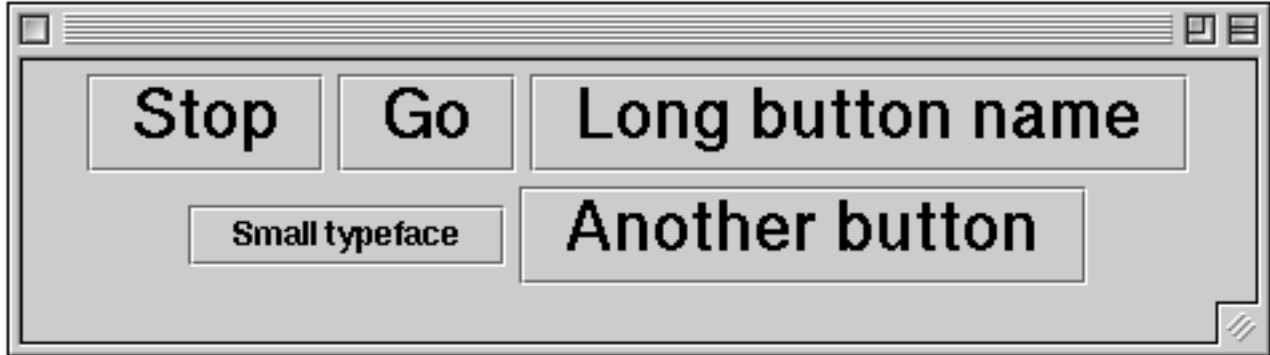
Layout managers allow you to specify a layout **policy**:  
rules for arrangement

# Some layout managers:

FlowLayout	components go with the flow
GridLayout	regimented: rigidly systematic
BoxLayout	flexibly regimented
BorderLayout	adaptively systematic
CardLayout	flip through the deck
GridBagLayout	weirdly systematic

# Flow layout

Items go from left to right and top to bottom  
in the order added,  
with as many items per row as will fit



# Flow layout



# Flow layout

```
cp. setLayout (new FlowLayout (FlowLayout . LEFT));
```



# Grid layout

The items are arranged in a matrix

```
cp.setLayout(new GridLayout(3, 2));
```

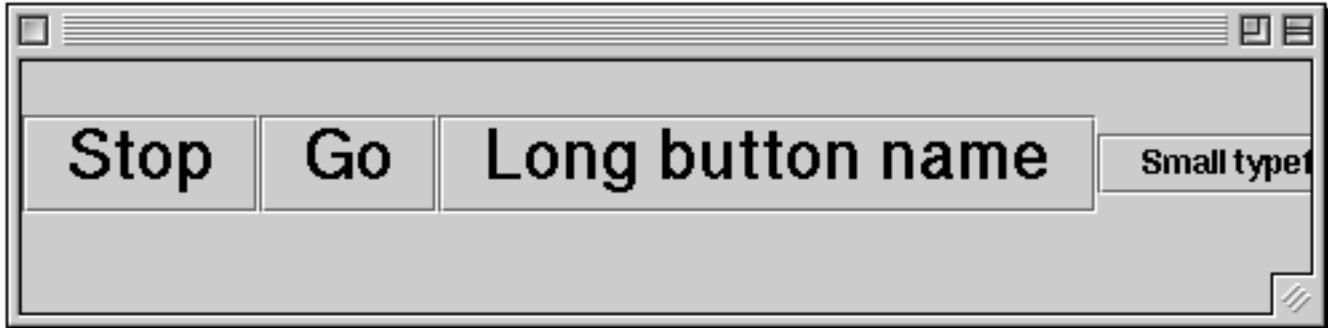


*Note: preferred component size is ignored*

# Box layout

The items are arranged in a single line

```
cp.setLayout(new BorderLayout(cp, BorderLayout.X_AXIS));
```



*Note: preferred component size is respected*

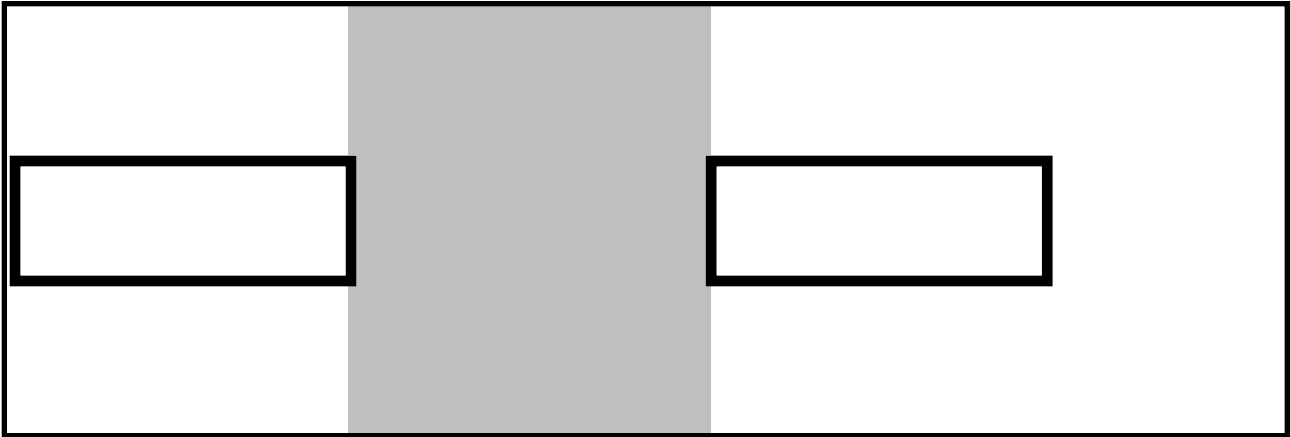
# Box layout

**Flexibility comes from being able to add**

- **rigid areas**
- **struts**
- **glue**

# Box layout

## Rigid areas



Box.createRigidArea(new Dimension(30, 40))

# Box layout

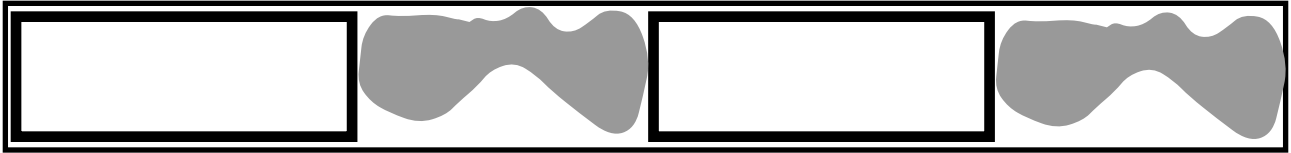
## Struts



`Box.createHorizontalStrut(40)`

# Box layout

## Glue



`Box.createGlue()`

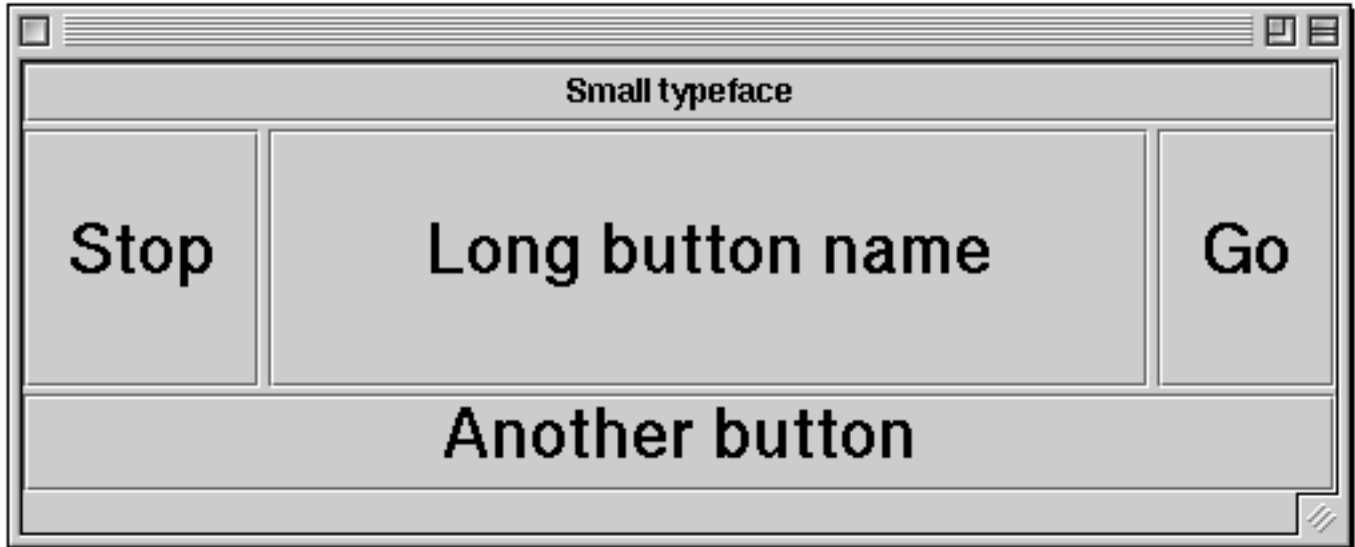
# Border layout

## Layout in directionally-specified areas

North  
West Center East  
South



## *Preferred component size is semi-ignored:*



- Any area with no component shrinks to nothing
- No area may have more than one component (but...)

# Panels

A **JPanel** is a GUI component  
that is also a container

Like a **JFrame**, it can contain other components  
*including* other **JPanel**s  
and it has a Layout Manager  
but no visual presence (except size & location)

# Containment hierarchy

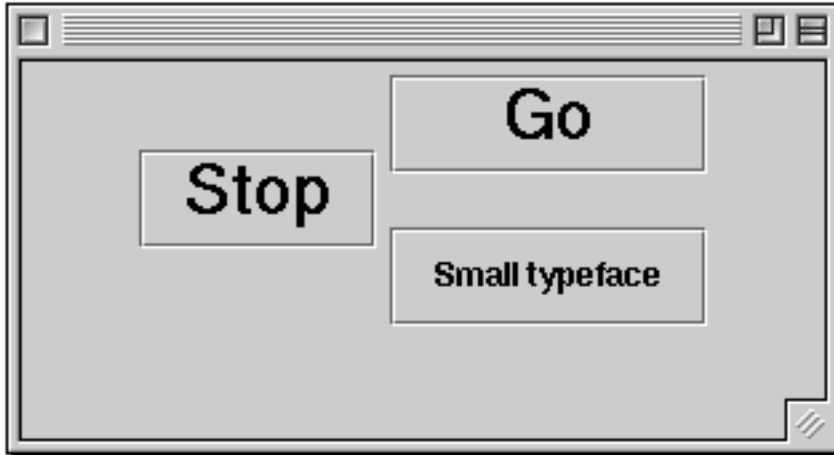
**What is contained within what, etc.**

**e.g., a button may be in a panel  
that is itself in a panel that is in a frame**

**Sidenote:**

**An applet is a panel that is contained  
in a web browser's frame**

For example



The two right buttons are in a **GridLayout** panel  
The frame has a **FlowLayout** and contains  
the panel and the left button

## The relevant code:

```
JFrame f = new JFrame();
Container cp = f.getContentPane();
cp.setLayout(new FlowLayout());
JButton b1 = new JButton("Stop");
cp.add(b1);
JPanel p = new JPanel();
p.setLayout(new GridLayout(2, 1, 0, 20));
JButton b2 = new JButton("Go");
p.add(b2);
JButton b3 = new JButton("Small typeface");
p.add(b3);
cp.add(p);
```

By the way...

The default layout for **JPanel** is **FlowLayout**

The default layout for **JFrame** is **BorderLayout**

Layout can be specified as a parameter for a new panel:

```
JPanel p = new JPanel(new GridLayout(2, 1))
```

# Components and containers

Swing components are objects  
that are instances of the **JComponent** class

**JComponents** and **JFrames** are objects  
that are instances of the **Container** class

**Containers** are objects  
that are instances of the **Component** class,  
can contain **Components**, and have layouts

# Rules of containment

Every component must always be  
in some container or another

No component can be in more  
than one container at a time

Components can be removed from  
and added to containers freely

Layout composition:

Every **Container** has a layout  
and any **Container**  
can contain other **Containers**  
*each* with their own layouts

# Some other GUI components

**Labels – a fixed piece of text/icon**

```
JLabel lab = new JLabel("Enter key phrase");
```

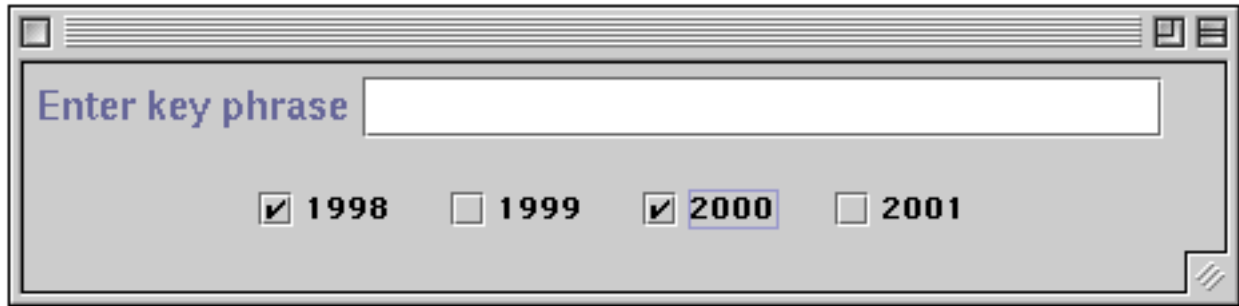
**Checkboxes – buttons that hold state**

```
JCheckBox cb3 = new JCheckBox("2000");
```

**Text fields – variable lines of text**

```
JTextField tf = new JTextField(20);
```

# An example

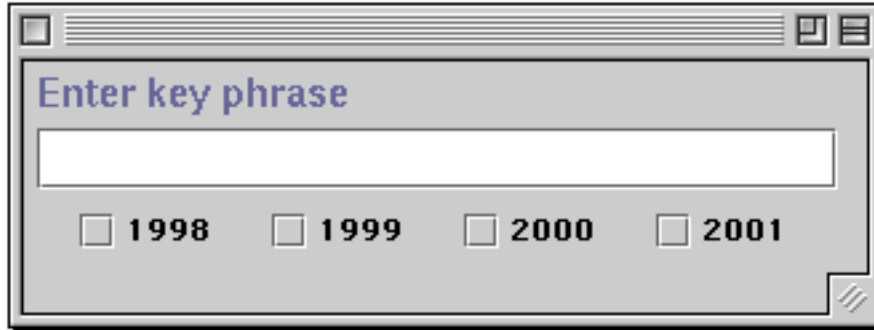


Enter key phrase

1998     1999     2000     2001

The image shows a classic web browser window with a title bar and standard window controls. The main content area contains a text input field with the placeholder text "Enter key phrase". Below the input field, there are four radio buttons, each followed by a year: 1998, 1999, 2000, and 2001. The radio button for the year 2000 is selected, indicated by a checkmark inside the box and a blue highlight around the text "2000".

# Same example, resized



## Homework #2

Create this application

Due to your TA by 3:00pm Monday Sept. 13, 1999

Enter key phrase

1998    1999    2000    2001

Enter key phrase

98    1999    2000    2

Enter key phrase

1998    1999    2000    2001

# Another example



# Reading the API documentation

**Full but cryptic information is provided about classes, their constants, variables, constructors, and methods**

**Available online, in 'straight' printed form, and in the Chan series**