ID2212 Network Programming with Java
Lecture 2

GUI Programming in Java

Leif Lindbäck, Vladimir Vlassov

KTH/ICT/SCS

HT 2015

User Interface in a Distributed Application

- A (graphical) user interface, (G)UI, needs to be constructed so that the user:

  – knows the current state of the application;

  – knows whether a command given to the application has been received;

  – knows that the application is working on a task and not simply "hung";

  – can always control the flow of the program, e.g. can cancel the task.

- Specific issue: long communication latency

Responsive (G)UI

- A responsive UI as an interface that is non-blocking while performing some time-consuming task such as
    - extensive calculations
    - networking with sockets or RMI or CORBA
    - file I/O (loading images, for example), etc.
- Responsive UIs can be implemented by using threads.
    - Move a time-consuming task out of the main thread, so that the GUI comes up faster;
    - Move a time-consuming task out of the event dispatching thread, so that the GUI remains responsive, i.e. not "frozen";
    - In a listener, create and start a thread to perform the corresponding task; the control flow returns to GUI
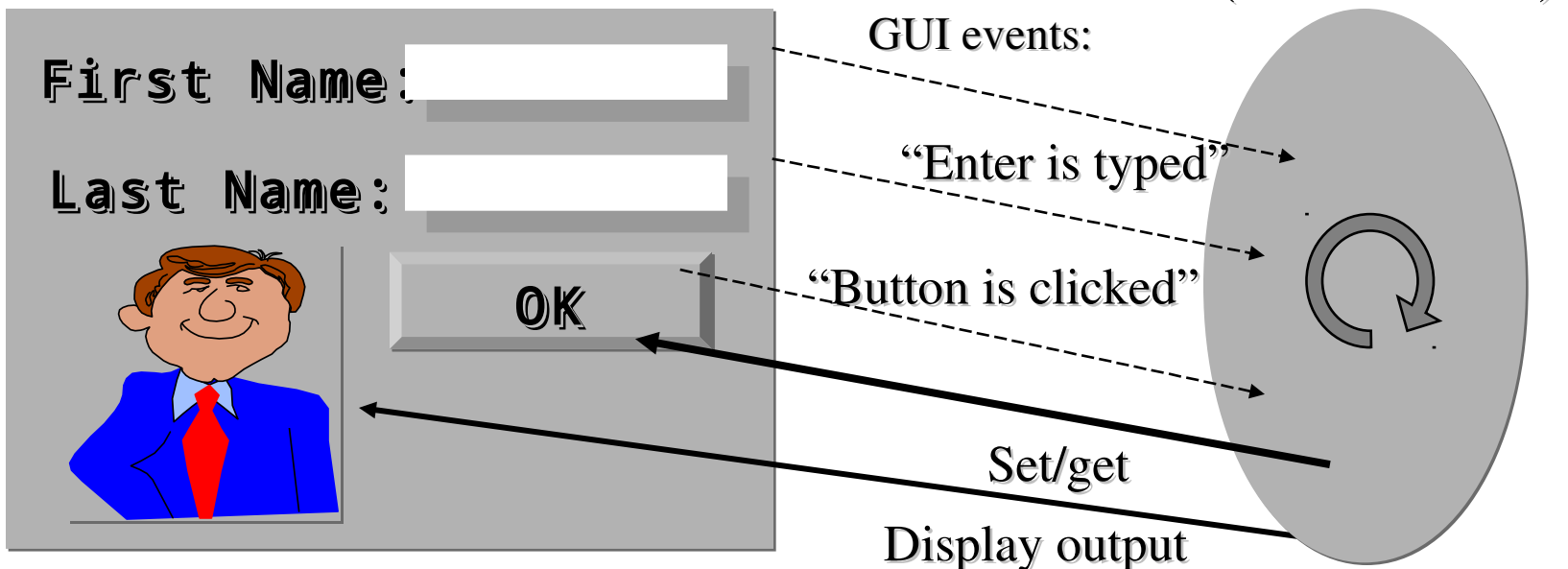
Contents

- AWT Packages and classes
- Constructing GUI
- AWT events, listeners and adapters
- Layout Managers and container attributes
- Overview of Swing
- Applets
- JavaBeans

GUI: Graphical User Interface

- GUI allows the user:
  - To control an application in a convenient way,
  - To input data, to display output.

GUI objects (widgets)

Program logic
(Event listeners)

GUI events:

First Name:

Last Name:

"Enter is typed"

"Button is clicked"

OK

Set/get

Display output

Java GUI APIs and Tools

- APIs for GUI in Java Platform SE:
    - *Java AWT*: Abstract Window Toolkit (basic GUI classes)
    - *Java Swing*
        - Lightweight GUI framework.
    - *JavaBean* API supports the development of JavaBeans.
        - A JavaBean is a reusable software component that can be manipulated visually in a builder tool, and can provide GUI.

- IDE (Integrated Development Environments), such as Eclipse, NetBeans, JBuilder from Borland, VisualAge from IBM

AWT: Abstract Window Toolkit

`java.awt`

Overview

AWT: Abstract Window Toolkit

- ***AWT***: Abstract Window Toolkit is a collection of Java packages that represent
    - GUI containers (frame, panel, etc.)
    - GUI components (button, text-field, etc.)
    - Basic GUI attributes (colors, fonts, cursor, layout, etc.)
    - GUI events and event listener interfaces
    - AWT was developed for building GUI without having to learn many of the details of the underlying windowing system.

Java AWT Widget ↔ Peer ↔ Native code ↔ X-windows
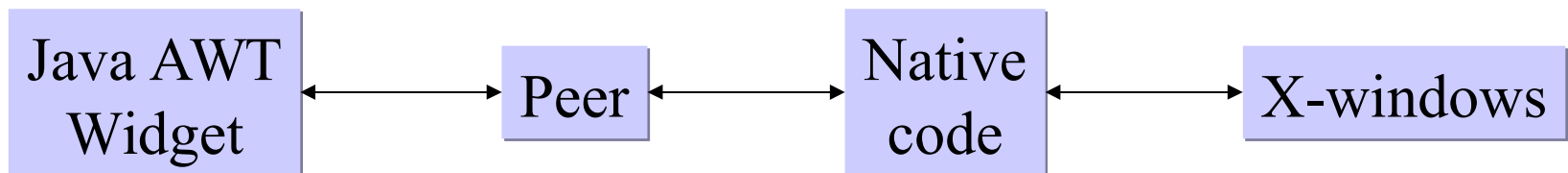
AWT Containers

- GUI containers are used to hold GUI components (widgets) :
  - Titled framed windows: `Frame`
  - Popup windows:
    - `Dialog`
    - `FileDialog`
    - `JFileChooser`, `JColorChooser` in Swing
  - Child sub-windows:
    - `Panel,`
    - `Applet,`
    - `ScrollPane`
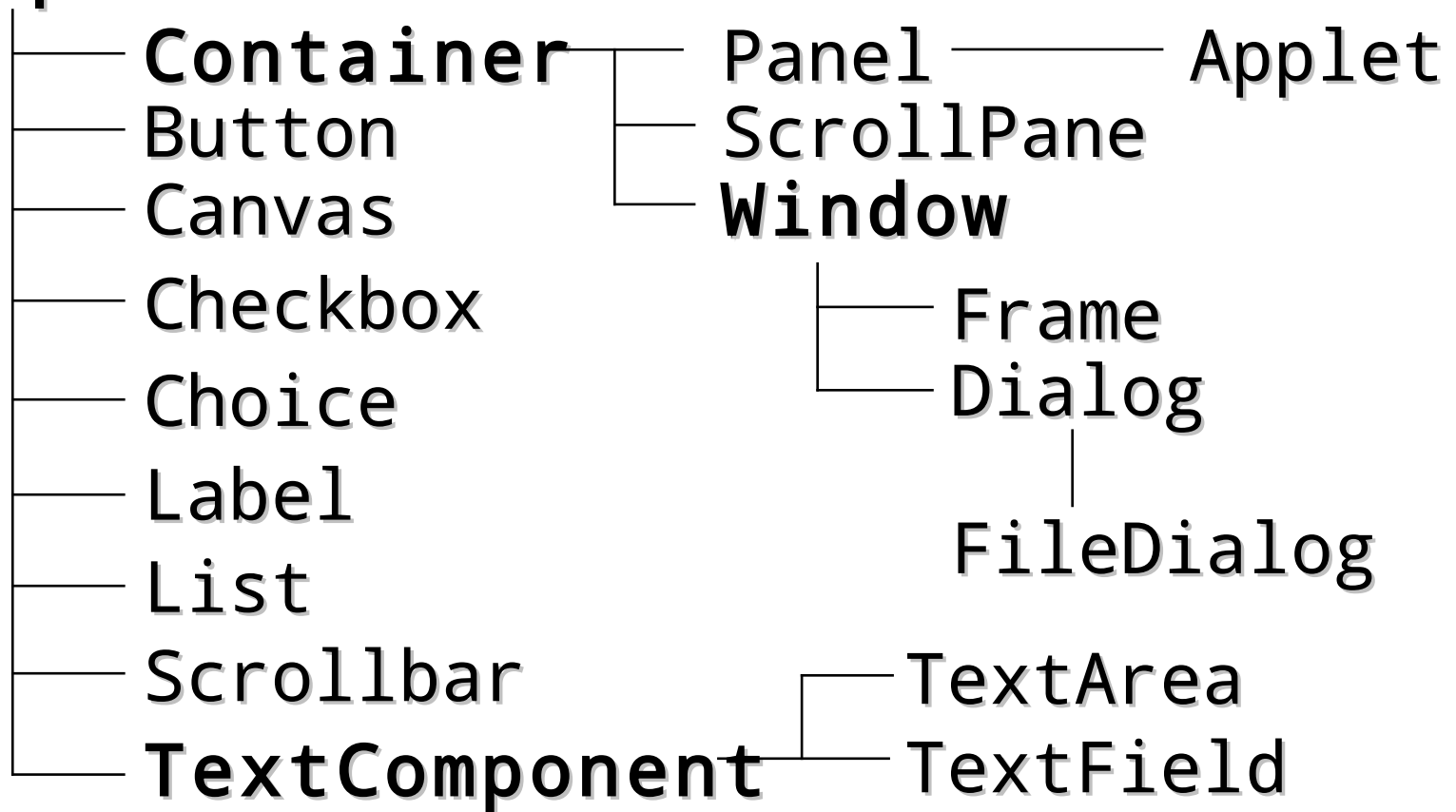
AWT Components

- Buttons
  - used to invoke some actions in an application.

- Selectable components
  - provide several choices to the user in which one or more items (options) can be selected:
    - Pulldown/pop-up menu: `Choice`
      - `JPopupMenu` in Swing
    - Scrolling list of choices: `List`
    - Menus in a menu bar frame : `Menu`, `MenuItem`, `MenuBar`
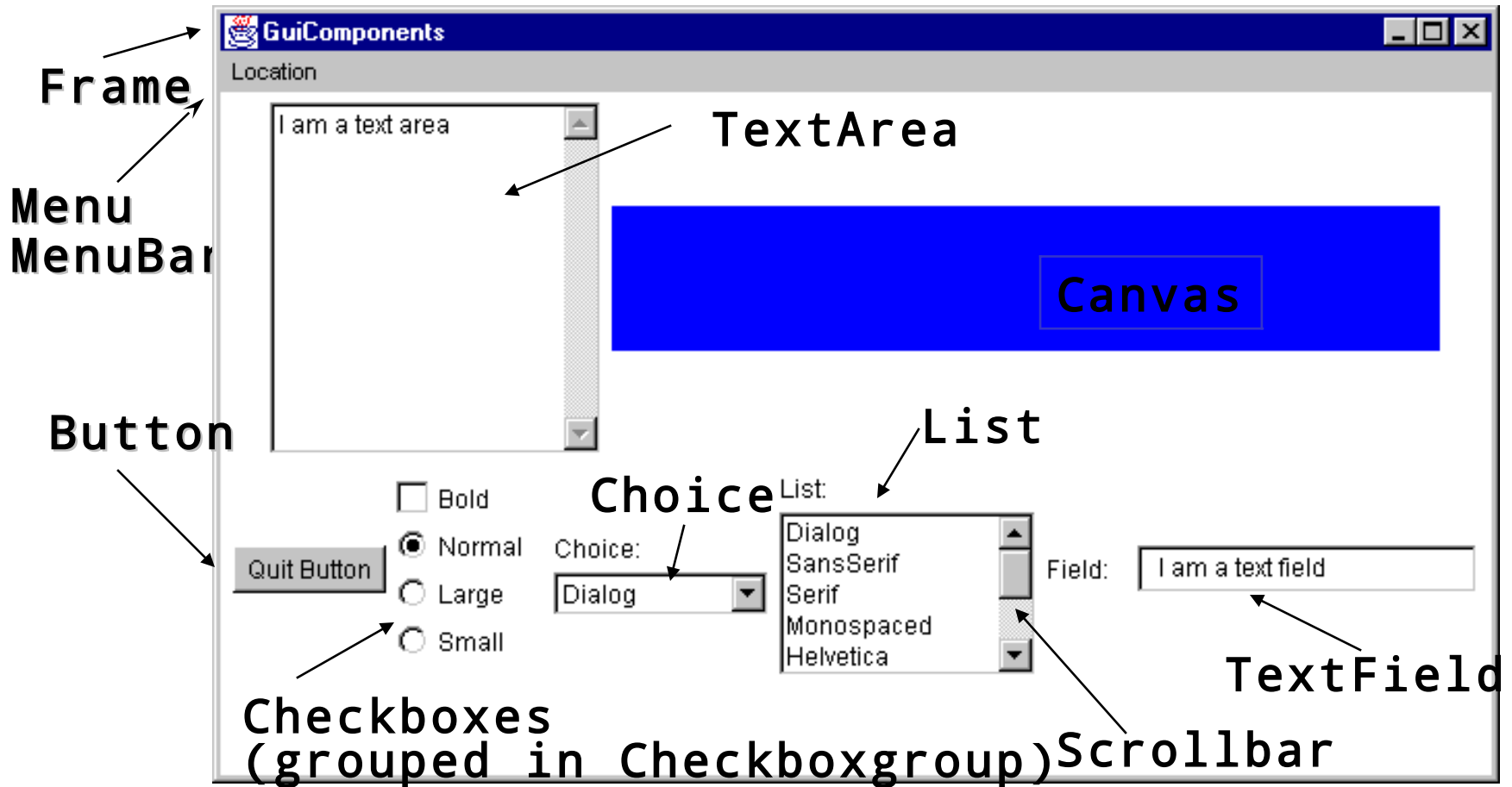   - On/off switch element: `Checkbox`

AWT Components (cont'd)

- Text components
  - used to input, edit and output text information:
    - Labels and messages: `Label`
    - An editable text string: `TextField`
    - A scrolling text area: `TextArea`

- Drawing components
  - `Canvas` provides a drawing surface for displaying graphical output.

- Scrollbar implements the metaphor of an elevator on a cable.
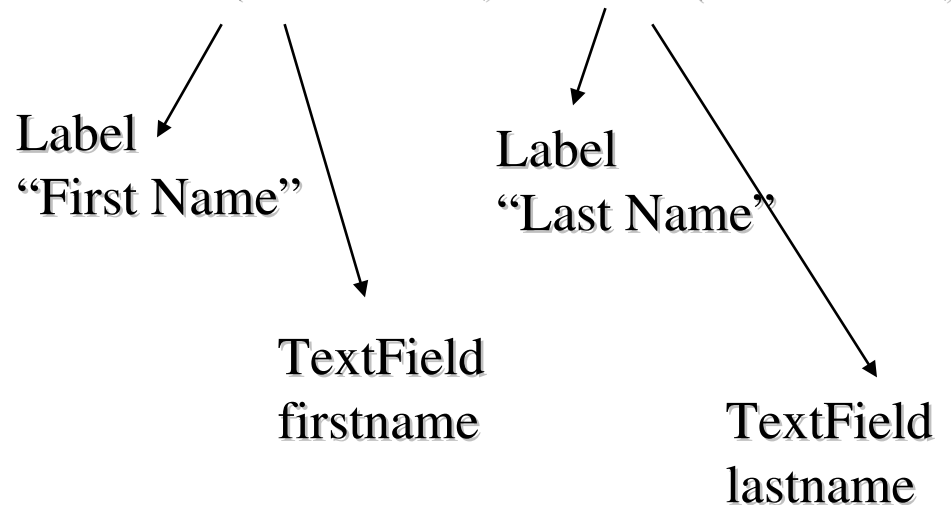
Component classes of AWT

```
Component
        ┌──── Container ──┬── Panel ──────── Applet
        ├─── Button       ├── ScrollPane
        ├─── Canvas       └── Window
        │                         ┌─── Frame
        ├─── Checkbox             │
        │                         └── Dialog
        ├─── Choice                     │
        │                               │
        ├─── Label               FileDialog
        │
        ├─── List
        │                        ┌── TextArea
        ├─── Scrollbar           │
        └─── TextComponent ──────┴── TextField
```

AWT components

Frame

Menu
MenuBar

Button

TextArea

Canvas

List

Choice

TextField

Checkboxes
(grouped in Checkboxgroup) **Scrollbar**

Nesting of GUI Containers and Components

Frame (root window)

Canvas   Button "OK"   Panel (a sub-window)   Panel (a sub-window)

First Name:

Last Name:

OK

Label
"First Name"

Label
"Last Name"

TextField
firstname

TextField
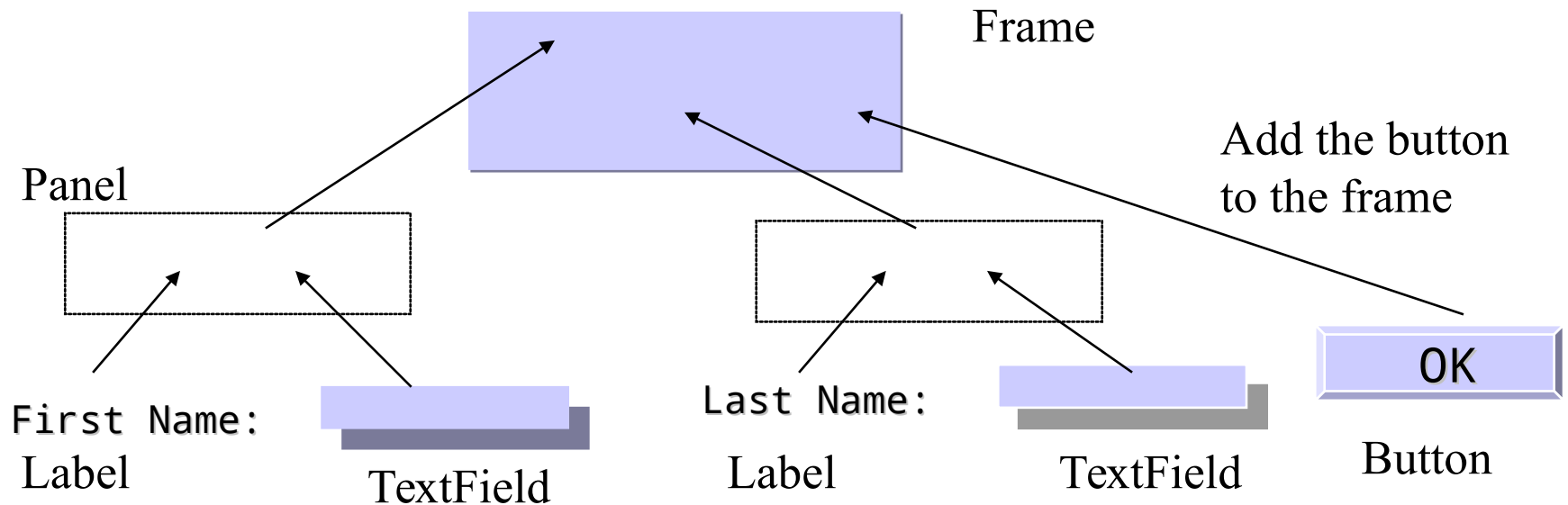lastname
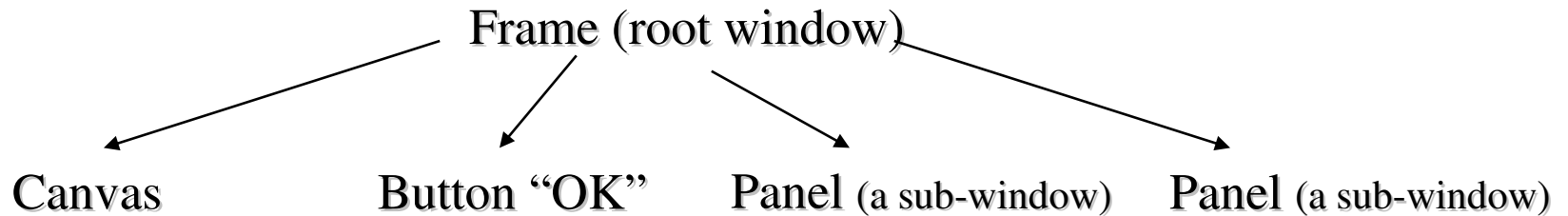
Constructing a GUI

- GUI is constructed by
  - Constructors (create a component object - a widget)
  - Setters (set attributes for the component, e.g. colors, size, labels, etc.)
  - Adders (add child components to the parent container, register event listeners)

Top-Down (Bottom-Up) Construction

Frame

Panel

Add the button
to the frame

First Name:
Label

TextField

Last Name:
Label

TextField

OK

Button

Constructing a GUI

- Create a root window (frame)
  ```
  Frame f = new Frame();
  ```
- Change/set attributes
  ```
  f.setLayout(new FlowLayout());
  ```
- Construct and add child subcomponents to the frame
  ```
  Panel p = new Panel();
  p.add(new Label("First name:"));
  TextField tffn = new TextField("", 30);
  tffn.addActionListener(this);
  p.add(tffn);
  f.add(p);
  ```
- Pack the frame and make it visible
  ```
  f.pack();
  f.setVisible(true);
  ```

Constructing GUI in an Applet

- The `Applet` class extends the `Panel` class, which is a GUI container, therefore an applet can be directly used as a root container to be filled in with GUI elements.

- The applet's GUI is usually created in the appelt's `init` method by instantiating and adding GUI components to the applet.

Events and Listeners

`java.awt.event`

- AWT events

- Event listeners and adapters

- Event delivery

AWT Events

- GUI-controlled Java application (applet) is event-driven.

    - Events propagate notifications of state change or commands from a source object (a GUI component) to one or more target objects (event listeners) via method invocation on listeners' interfaces.

- Examples of AWT Events:

    - Mouse actions, e.g.

        - a right mouse button is down (`MouseEvent`)

        - a GUI button is clicked (`ActionEvent`)

        - a menu item is chosen (`ItemEvent`)

    - Keyboard actions, e.g. a key is pressed (`KeyEvent`)

- Classes of AWT events are specified in
  `java.awt.event`

AWT Event Listeners

- A listener must be registered at a source:

```
Button button = new Button("Start"); // source of
    events
button.addActionListener(new FooListener());
```

- The listener must implement the appropriate event listener interface:

```
class FooListener extends Foo implements
    ActionListener
{
  ...
  public void actionPerformed(ActionEvent e) {
  ... // perform some action on event e
  }
}
```

- Event listener interfaces are specified in `java.awt.event`

Event Listeners (cont'd)
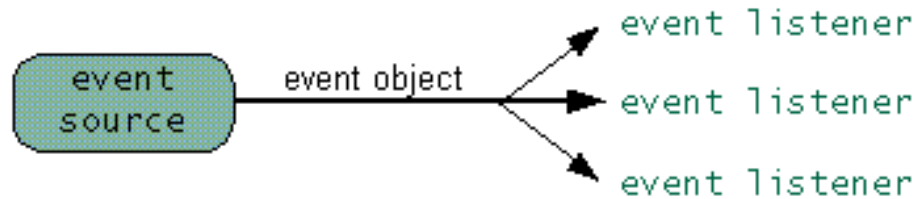
Click a button, press Return while typing in a text field, or choose a menu item	`ActionListener`
Close a frame (main window)	`WindowListener`
Press a mouse button while the cursor is over a component	`MouseListener`
Move the mouse over a component	`MouseMotionListener`
Component becomes visible	`ComponentListener`
Component gets the keyboard focus	`FocusListener`
Table or list selection changes	`ListSelectionListener`

One Listener –Multiple Sources

- One and the same event listener object can be linked to multiple event sources via different interfaces or the same interface.

```
TextField tf = new TextField("", 30);
Button b = new Button("Start");
tf.addActionListener(this); // on Enter
tf.addTextListener(this); // when text changes
b.addActionListener(this); // on click
```

b —— ActionListener.actionPerformed ——→ **'this'**

tf —— TextListener.textValueChanged ——→ **'this'**
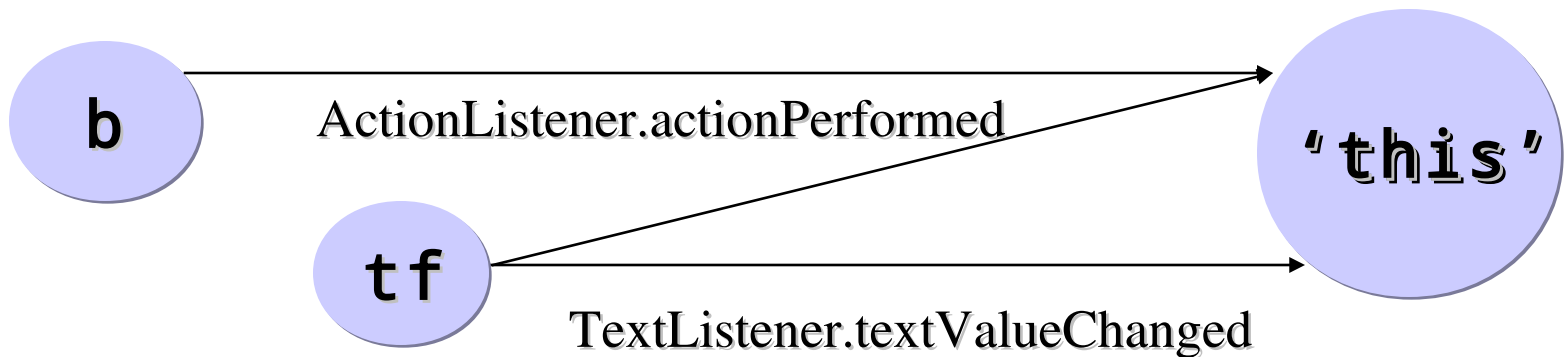
Using an Adapter Class

- Use of AWT adapter classes simplifies implementation of listener interfaces.

- There is an adapter class for each AWT interface in `java.awt.event`

  - An adapter class FooAdapter implements FooListener interface for FooEvent

  - Default implementation: do nothing

Using an Adapter Class (cont'd)

- Example:
  - The **MouseAdapter** class implements all five methods of **MouseListener** interface that receives **MouseEvent**
  - The adapter can be used as follows:

```
addMouseListener(new MouseAdapter {
  public void mousePressed () {
    //override the method
    }
});
```
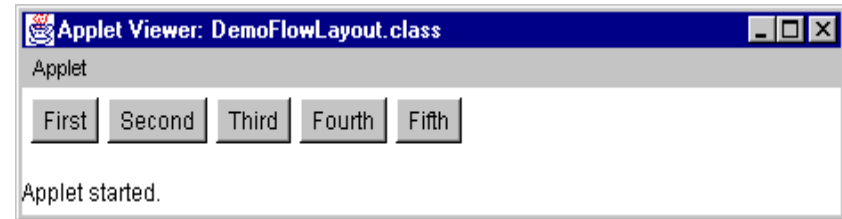
Layout Managers and Container Attributes

Layout Managers

- Layout manager allows organizing the layout of the GUI elements in a container with this layout.
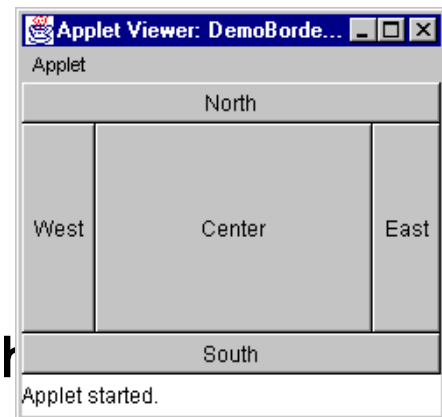
- **`FlowLayout`**

  - From left to right, top to bottom.

  - The order of adds defines the order of components in the container.

- **`BorderLayout`**

  - "North", "South", "East", "West", "Center"

```
frame.add(b = new Button(), "South
```

Layout Managers (cont'd)

- **`GridLayout`**
  - A grid of equal-sized rectangles for components.
  - **`public GridLayout(int rs,int cs)`**
    - where: **`rs`**, **`cs`** - Number of rows and columns.

      0 – "any number".

- **`GridBagLayout`**
  - An irregular grid of components that uses constraints to arrange components.

Layout Managers (cont'd)

- ## CardLayout
  - Allows adding "cards" (sub-windows) and flipping through the "cards"
  - A container with CardLayout may contain several "cards" (sub-windows). One is shown at a time.
  - Flip through the cards sequentially, or show a card specified by its name
  - For example:

```
CardLayout cl = (CardLayout) mainframe.getLayout();
cl.first(mainframe); // show the first card
cl.next(mainframe); // show the second (next) card
cl.show(mainframe, "Card5"); // show the card
  "Card5"
```

Default Layout Managers, Setting a New Layout

Subclasses of `Container`	Default Layout Manager
`Panel, Applet`	`FlowLayout`
`Window`	`BorderLayout`
`Dialog`	`BorderLayout`
`Frame`	`BorderLayout`

- Setting a new layout manager:
  ```
  setLayout(new FlowLayout());
  Panel p = new Panel(new BorderLayout());
  ```

Set / Get Container's Attributes

- Size: `setSize(int, int)`
- Location: `setLocation(int, int)`
- Layout manager: `setLayout(LayoutManager)`
- Title (for a Frame): `setTitle(String)`
- Colors:
  - `setBackground(Color);`
  - `setForeground(Color)`
- Visibility: `setVisible(boolean)`
- Cursor: `setCursor(Cursor)`
- Font: `setFont(Font)`
- Event Listeners:
  - `addWindowListener(WindowListener)`
  - `addFocusListener(FocusListener)`, etc.

Applets: Downloadable Web clients

`java.applet.Applet`

Applet

- An applet is a subclass of `java.applet.Applet` that is a subclass of `java.awt.Panel`
  - loaded by a web-browser when an activated html page contains a reference to it – an applet tag.
  - can open a TCP connection (do RMI) to the host from which the applet has been downloaded;
  - expose network methods, e.g. `getImage`

Applet Methods

- Applet includes methods called by the environment where the applet runs:
  - `init()`
    - called once to initialize the applet, e.g. to build its GUI
  - `start()`
    - called each time when the applet's page is (re)open
  - `stop()`
    - called each time when the applet's page is closed
  - `paint(Graphics g)`
    - called when the applet must be repainted (resized, etc)
  - `destroy()` etc.
- The methods can be overridden, if needed.

Applet Tag in HTML Example

```
<HTML>
<HEAD>
<TITLE> This is a html page for running an
   applet </TITLE>
</HEAD>
<BODY>
   <APPLET CODE = "WebApplet.class"
     WIDTH = 500
     HEIGHT = 500>
   <PARAM NAME="image" VALUE="jim.gif">
   </APPLET>
</BODY>
</HTML>
```

Example of Applet

- Draws a line that follows a mouse pressed and dragged.
- No communication with the server

- **See also Applet examples in Lab 0 and Lab 1**

```java
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Scribble extends Applet
    implements MouseListener, MouseMotionListener {
  private int last_x, last_y;

  //Initialize the applet
  public void init(){
    this.addMouseListener(this);
    this.addMouseMotionListener(this);
    this.setSize(400, 300);
  }
  public void mousePressed(MouseEvent e) {
    last_x = e.getX();
    last_y = e.getY();
  }
  public void mouseDragged(MouseEvent e) {
    Graphics g = this.getGraphics();
    int x = e.getX();
    int y = e.getY();
    g.drawLine(last_x,last_y,x,y);
    last_x = x;
    last_y = y;
  }
  public void mouseReleased(MouseEvent e) {;}
  public void mouseClicked(MouseEvent e) {;}
  public void mouseEntered(MouseEvent e) {;}
  public void mouseMoved(MouseEvent e) {;}
  public void mouseExited(MouseEvent e) {;}
}
```
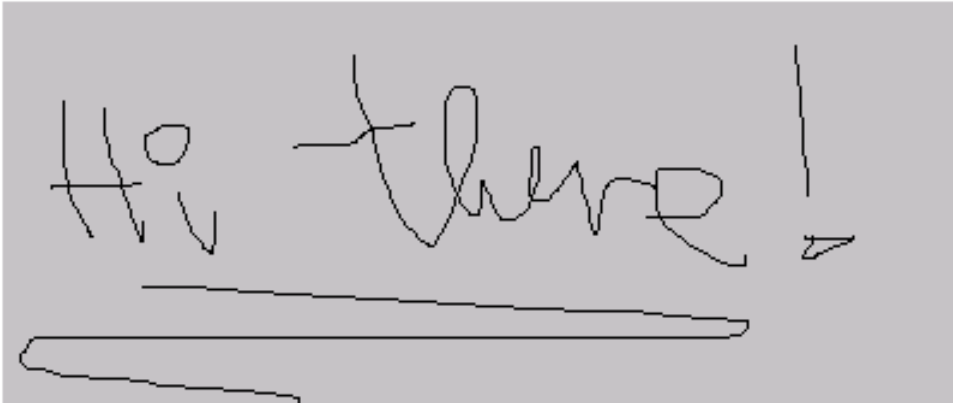
Example (cont'd)

- A source and a view of the HTML page with the Scribble applet

Scribble will appear below in a Java enabled browser.


```html
<html>
<head>
<title>
HTML Test Page
</title>
</head>
<body>
Scribble will appear below
in a Java enabled browser.<br>
<applet
   codebase = "."
   code      = "Scribble.class"
   name      = "TestScribble "
   width     = "400"
   height    = "300"
   hspace    = "0"
   vspace    = "0"
   align     = "middle"
>
</applet>
</body>
</html>
```

Applet Example 2

- Develop an applet that counts how many times the basic methods of a applet (init, start, paint, stop) are invoked.

```java
import java.applet.Applet;
import java.awt.*;
public class StarterApplet extends Applet {

  private int InitCount = 0;
  private int StartCount = 0;
  private int StopCount = 0;
  private int PaintCount = 0;

  public void init() {
    resize( 300, 100 );
    InitCount++;
  }
  public void start() {
    StartCount++;
  }
  public void stop() {
    StopCount++;
  }
  public void paint( Graphics g ) {
    PaintCount++;
    g.drawString( " Inits: " + InitCount +
        " Starts: " + StartCount +
        " Stops: " + StopCount +
        " Paints: " + PaintCount, 30, 50 );
  }
}
```

Example 2 (cont)

- Applet and application combined in one class

```java
import java.awt.Frame;
import java.awt.event.*;

public class StarterCombined extends StarterApplet {
  public static void main( String args[] ) {
      Frame f = new Frame( "Starter Application" );
      f.addWindowListener(new WindowAdapter()
      {
          public void windowClosing(WindowEvent e)
          {
              System.exit(0);
          }
      });
      StarterCombined applet = new StarterCombined();
      applet.init();
      f.add( "Center", applet );
      f.setSize( 300, 100 );
      f.setVisible(true);
      applet.start();
  }
}
```

Some Network Methods of the `Applet` Class

- Getting audio and image files:
  - `Image getImage(URL)`
  - `Image getImage(URL, String)`
  - `AudioClip getAudioClip(URL)`
  - `AudioClip getAudioClip(URL, String)`
  - `play(URL)`
  - `play(URL, String)`
- Locate the applet or the document in which the applet is embedded:
  - `URL getCodeBase()`
  - `URL getDocumentBase()`

Applet Context

*   The applet context represents an applet's environment, e.g. a browser or an applet viewer:

    ```
    AppletContext ac = getAppletContext();
    ```

*   Useful methods:

    ```
    showDocument(URL)
    ```
    – replace the current Web with the given URL.

    ```
    showDocument(URL url, String target)
    ```
    – Show the Web page indicated by url. The target argument indicates where to display the frame, for instance, in the current frame or in a new navigator.

    ```
    getApplet(String)
    ```
    – returns the applet with the given name

*   See Example 3.13 (on the Examples page) `ShowImage1.java`

Named Applets

- An applet can be named in the APPLET (EMBED) tag:

```
<APPLET CODEBASE="../MyClasses" CODE=TalkApplet.class
   WIDTH=793 HEIGHT=130 NAME="Top">
<PARAM NAME=Partner VALUE="Bottom">
</APPLET>
<HR>
<APPLET CODEBASE="../MyClasses" CODE=TalkApplet.class
   WIDTH=793 HEIGHT=130 NAME="Bottom">
<PARAM NAME=Partner VALUE="Top">
</APPLET>
```

- Named applets embedded in the same Web page can get references to each other via the shared Applet Context:

```
TalkApplet a = (TalkApplet)(getAppletContext().
    getApplet(getParameter("Partner")));
```

# Interaction of Applets via the Applet Context

- Applets on the same Web page share the Applet Context (JVM) and can interact with each other via

  - Method invocation. The methods should be declared as synchronized.
  - Static class variables;
  - Piped connection.

- An applet gets reference to another applet by name.

- See Example 4.5 (on the Examples page) TalkApplet.java

Swing: Advanced Window Toolkit

`javax.swing`

What Is Swing?

- *Java Swing* – is an API that provides a set of extensible GUI components to develop powerful Java front ends for commercial applications more rapidly.

- 100% pure Java, GUI component kit

- The Swing Stack:
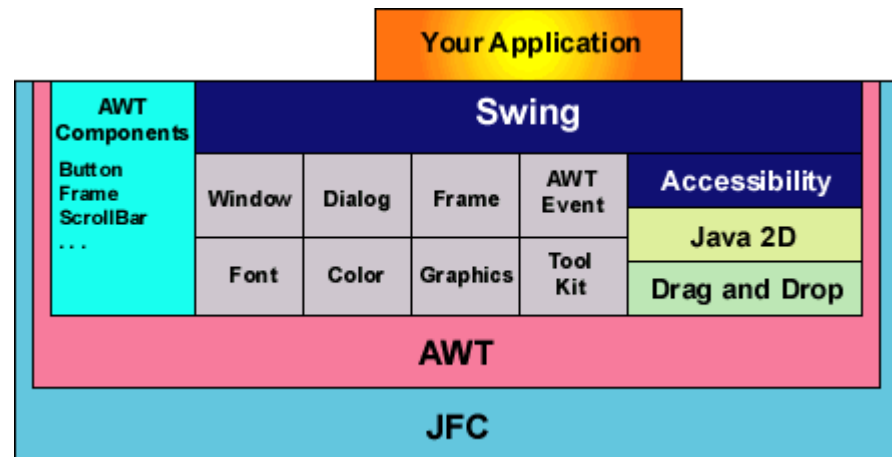
Swing Components

```
Container
JComponent
JSplitpane
         JButton
         JLabel
         JCheckbox
         JProgressBar
         JTable
         JList
         JScrollbar
         JTextComponent
```

```
                          JPanel          JApplet
                          JScrollPane
                          JDesktopPane
                             Window
                                   JFrame
                                   JDialog
```

```
                             JTextArea
                             JTextField
```

Swing components (1/2)

- See Swing component and container gallery (A Visual Index to the Swing Components) at

`http://download.oracle.com/javase/tutorial/uiswing/components /index.html`

- Basic Controls:
  - **JButton**, **JCheckBox**, **JRadioButton**
  - **JComboBox** –Buttons that bring up menus of choices.
  - **JMenuBar**, **JMenu**, **JRadioButtonMenuItem**, etc.
  - **JList**, **JTextField**
  - **JSlider** lets choose one of a continuous range of values.

Swing Components (2/2)

- Uneditable information displays:
  - `JLabel`
  - `JProgressBar` displays progress toward a goal.
  - `JToolTip` brings up a small window that describes another component.
- Editable displays of formatted information:
  - `JColorChooser`, `JFileChooser`
  - `JPasswordField`, `JTextArea`, `JEditorPane`, `JTextPane`
  - `JTable`
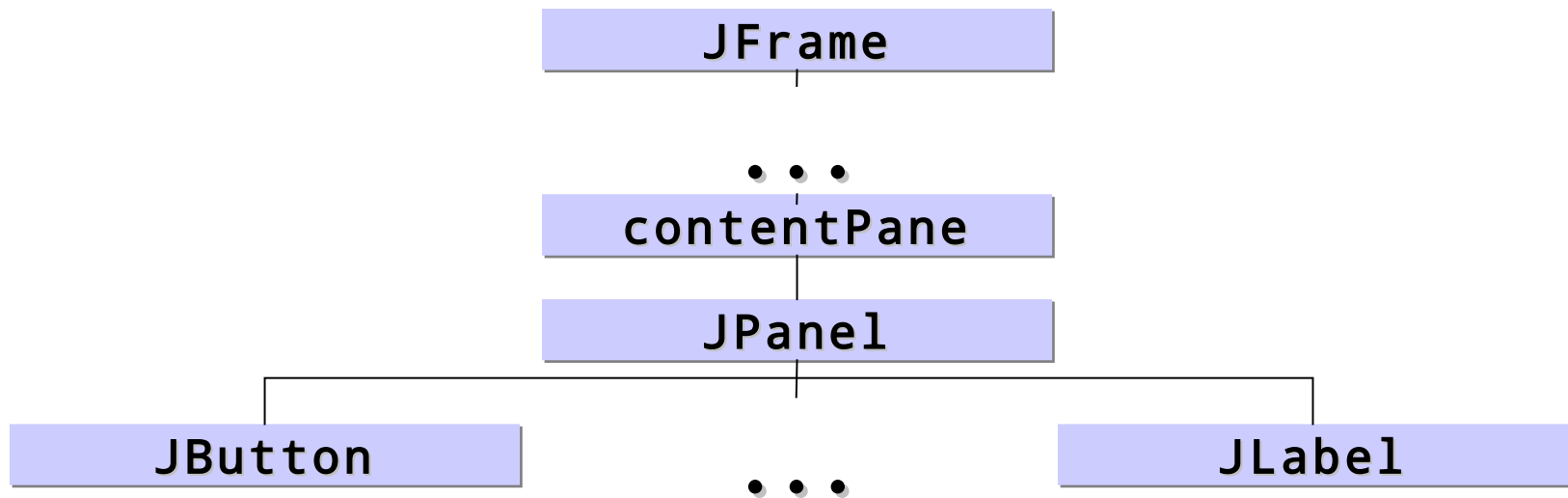 - `JTree` displays hierarchical data.

Swing containers (1/3)

- Top-Level Containers: `JFrame`, `JDialog`, `JApplet`
  - `frame.getContentPane().add(child);`
  - The content pane should be the parent of any children of a top-level container.

- General-Purpose Containers
  - `JPanel`
  - `JScrollPane` provides scroll bars around a large or growable component.
  - `JSplitPane` displays two components in a specified amount of space.
  - `JTabbedPane` contains multiple components but shows one at a time.
  - `JToolBar` holds a group of components (usually buttons) in a row or column, allowing the user to drag the tool bar into different locations.

Swing Containers (2/3)

- Special-Purpose Containers:
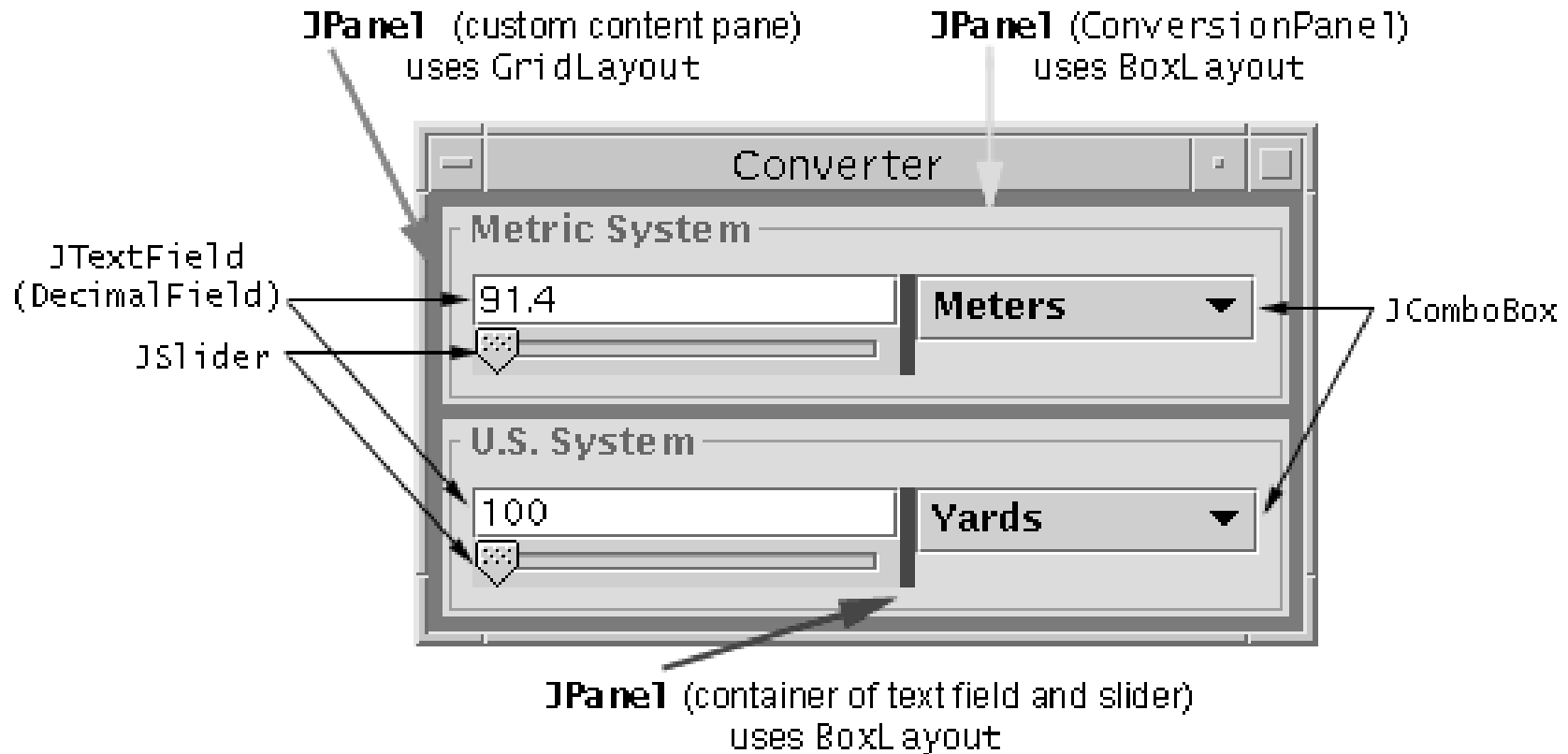 - `JInternalFrame` looks like a frame, but must appear within another window (`JDesktopPane`).
  - `JLayeredPane` provides a third dimension, depth, for positioning components. You specify the position and size of each component.
  - `JDesktopPane` is used to create a multiple-document interface or a virtual desktop.

Swing Containers (3/3)

`frame.getContentPane().add(panel);`

View of Some Swing Components

JPanel (custom content pane) uses GridLayout

JPanel (ConversionPanel) uses BoxLayout

JTextField (DecimalField)

JSlider

Converter

Metric System

91.4   Meters

U.S. System

100   Yards

JComboBox

JPanel (container of text field and slider) uses BoxLayout

Look-and-Feels

- The Java look-and-feel (Metal): "javax.swing.plaf.metal.MetalLookAndFeel"

- The CDE/Motif look-and-feel: "com.sun.java.swing.plaf.motif.MotifLookAndFeel"

- The Windows look-and-feel: "com.sun.java.swing.plaf.windows.WindowsLookAndFeel"

- Make a program use the Java look & feel:

```
public static void main(String[] args) {
  try {
    UIManager.setLookAndFeel( UIManager.getCrossPlatformLookAndFeelClassName());
  } catch (Exception e) { };
  new SwingApplication(); //Create and show the GUI.
}
```

- Setting a specific look & feel, e.g. the Windows Look & Feel:

```
 try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
 } catch (Exception e) { };
```

Overview of JavaBeans

"

`http://docs.oracle.com/javase/tutorial/javabeans/`

A Component Model

- A component model
  - includes a component architecture (specification and a set of APIs)
  - allows defining software components to be combined together to create an application.
- Two major groups of elements:
  - Containers
    - Used to hold assembly of related components
    - Provide context for components to interact with one another
  - Components
    - Vary in size and capabilities
    - Can be containers

Java Component Architectures

- *JavaBeans* – a first Java component architecture
  - Portable, self-described, reusable software components that can be visually manipulated, customized and combined with other Java components in a builder tool (bean container) such as Eclipse or Sun's NetBeans.
  - Have attributes, expose public methods and may fire events.
  - Example: AWT components, e.g. Button, TextField, etc.

- *Enterprise Java Beans (EJB)* – a component architecture for enterprise applications
  - Different from JavaBeans: EJBs are deployable
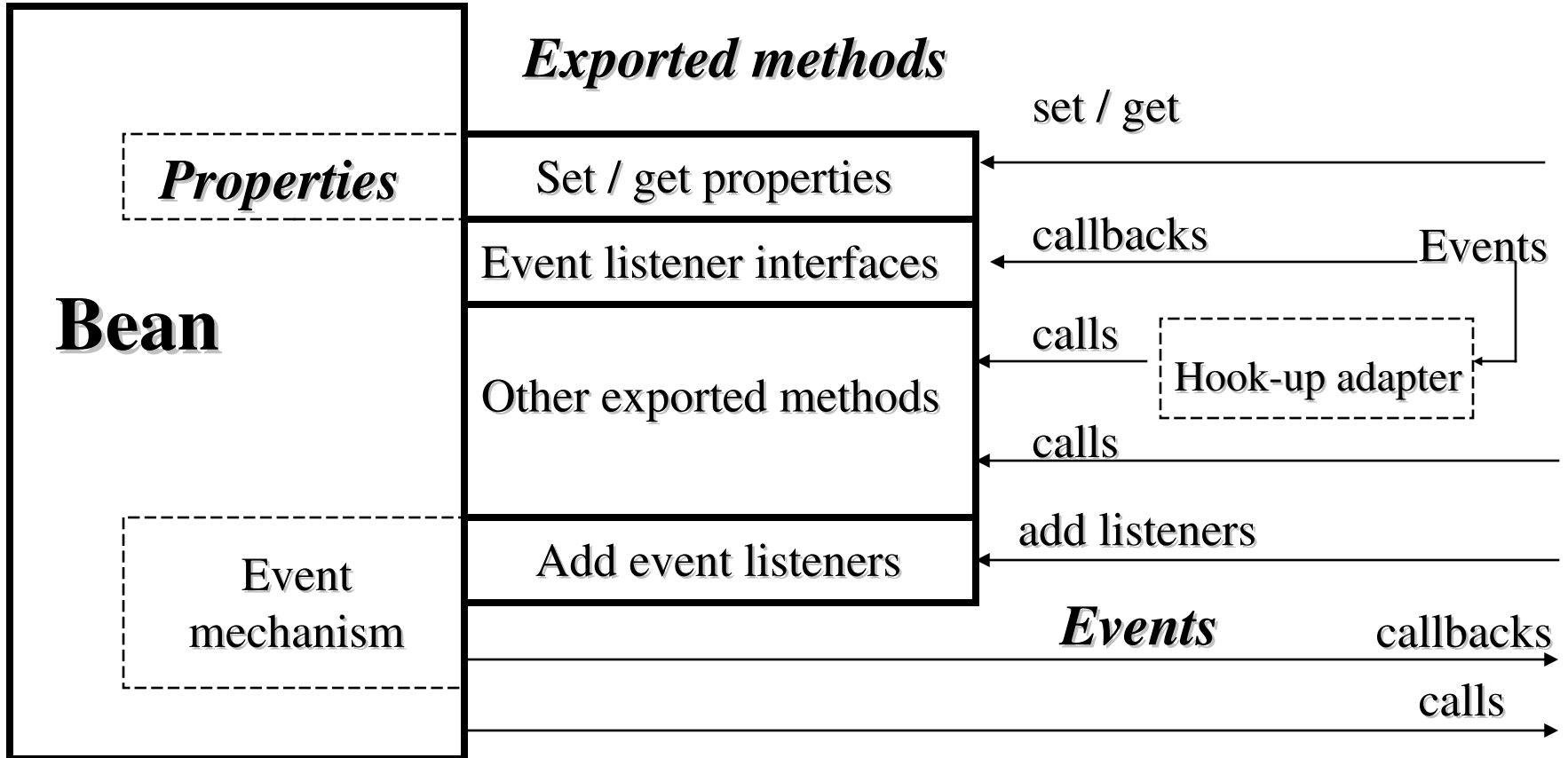 - An EJB can be either Session or Message-Driven

A JavaBean Can

- Export public methods
- Raise and catch events of proper types
- Allow queries about its capabilities - introspection
- Support persistent properties (can be stored and restored)
- Support component editors for builders to allow users to customize component behavior
- Be a container for other beans
  - Hierarchical components
  - Java compound documents

Bean Properties, Methods and Events

- A Bean communicates via:
  - *Properties* it exposes.
    - Named attributes that can be read or written by calling the appropriate set and get methods on the bean.
  - *Methods* it exports.
    - By default all of the bean's public methods are exported
  - *Events* it fires or/and listens.
    - The bean can register listeners for its events via public methods add<EventType>Listener(<EventType>Listener)
    - A bean instance can be registered as a listener of events fired by other components:
      - directly: implementing appropriate event listener interfaces
      - indirectly: via hook-up event adapters.

A Bean Interface

Bean

Properties

Event mechanism

Exported methods

Set / get properties	set / get
Event listener interfaces	callbacks → Events
Other exported methods	calls → Hook-up adapter
	calls
Add event listeners	add listeners

Events — callbacks

calls

Bean Properties

- Properties are public/private attributes that can be exposed by set / get methods (accessors).

- The name convention: public methods named setFoo and getFoo indicate a property named Foo.

- A *simple property* represents a single value.

- An *indexed property* represents an array of values.
  - The property can be accessed at once.
  - An element of the property can be accessed by index.

- A *bound property*
  - Notifies other objects when its value changes via a `PropertyChange` event that contains the property name, old and new value.

- A *constrained property*
  - Notifies other objects when its value changes via a `VetoableChange` event and allows a listener to veto the change by throwing a `PropertyVetoException`.
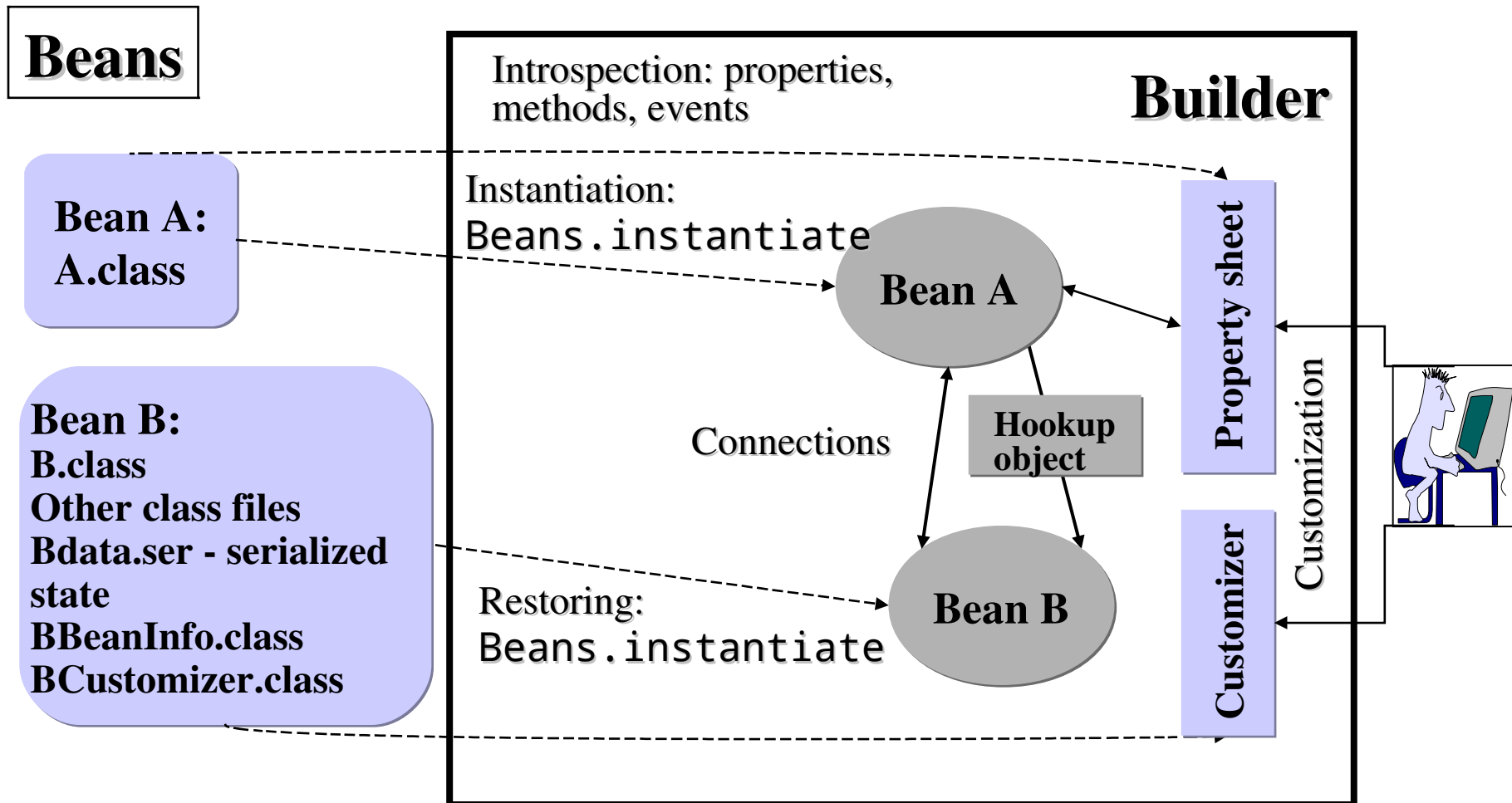
Bean Event Mechanism

- Based on Java event delegation model
- Convention: A Bean indicates that it can fire FooEvent if
  - it contains a pair of methods:
    ```
    public void addFooListener(FooListener a)
    public void removeFooListener(FooListener a)
    ```
  - and it declares or imports the FooListener interface and the FooEvent class.
- Use of a hookup event adapter:
  - If a listener does have the FooListener interface, it can be connected to the bean via a hookup event adapter that implements the interface and invokes appropriate methods on the real listener (i.e. the bean).

Developing Beans with a RAD tool

- Visual development using property sheets, palettes, and design-time drag-and-drop behavior.

  - A Bean class must adhere to a set of name conventions and design patterns for properties it exposes, events it fires

  - A builder tool uses introspection based on reflection to learn about the properties, events, and methods supported by a target Java Bean.

  - The tool relies on design patterns specified in the JavaBeans Specification

Beans in an Application Builder

Beans

Builder

Introspection: properties, methods, events

Instantiation:
`Beans.instantiate`

Bean A:
A.class

Bean A

Property sheet

Bean B:
B.class
Other class files
Bdata.ser - serialized state
BBeanInfo.class
BCustomizer.class

Connections

Hookup object

Customization

Restoring:
`Beans.instantiate`

Bean B

Customizer

Saving and Restoring Beans

- ***Java Object Serialization*** provides persistence: saving Beans to streams (files, network streams)
  - The serialized state of the Bean can been restored at design-time (by a builder tool) and at run-time.
  - Loading a serialized state of a Bean to a builder tool, customization of the Bean and saving its state for further use.

- All files related to a Bean can be stored into a JAR file that normally contains:
  - All classes files related to the Bean: <BeanName>.class, others.
  - Supporting classes: <BeanName>BeanInfo.class, <BeanName>Customizer.class
  - Other files: images, audio, etc.
  - A serialized state of the Bean to be restored: <BeanName>Data.ser