What is a GUI?

- Instead of asking the user "What do you want?" and "How should we do it?"
- A GUI tells the user "What we can do" and then asks for "What should we do?"
- Pioneers: Gremlin, Apple
Java GUI components (in java.awt)

- **Label**: area where non-editable text is displayed
- **Button**: area which triggers an event when clicked
- **TextField**: area where user inputs text; can also display text
- **Choice**: pull-down list of items
- **CheckBox**: boolean GUI component
- **List**: area where a list of items is displayed
- **Panel**: a "container" in which components can be displayed
Label

- area where non-editable ext is displayed

Constructors:

- `public Label()`
- `public Label(String s)`
- `public Label(String s, int alignment)`

Methods:

- `public String getText()`
- `public void setText(String s)`

Can be added to a container such as an "applet" using

```
container.add(labelname)
```

(See Figure 10.5)
Push Button

- area which triggers an event when clicked
- generate "ActionEvent"

Constructors:  
- public Button()
- public Button( String s )

Methods:  
- public String getLabel()
- public void setLabel( String s )
- public String getActionCommand()
- public Synchronized void addActionListener( ActionListener l )

To process an event, an "event listener" has to be registered, and an "event handler" has to be implemented.

When a button is clicked, method "actionperformed" of the registered event-listener object is called.

(See Figure 10.8)
**TextField**

- single line area which receives user input
- can also display text
- generate "ActionEvent"

Constructors:

```java
public TextField()
public TextField( int columns )
public TextField( String s )
public TextField( String s, int columns )
```

Methods:

```java
public void getEchoChar( char c )
    // c: masking character
public void setEditable( boolean b )
```

To get the input of the user, use

```java
ActionEvent.getActionCommand()
```

(See Figure 10.10)
Choice Buttons

- user selects from a pull-down (drop-down) list
- saves space
- generate "ItemEvent"

Constructors: public Choice()

Methods: public void addItem( String item )
          public void getItem( int Index )
          public int getItemCount( )

implements "ItemListener"

When an item is selected, method "itemStateChanged" of the registered event-listener object is called.

(See Figure 10.12)
Checkboxes and Radio Buttons

- checkbox: all can be checked or unchecked
- Radio: only one at a time can be checked
- generate "ItemEvent"

Constructors:
- public Checkbox(String s)
- public Checkbox(String s, CheckboxGroup c, boolean state)
- public CheckboxGroup()

Methods: similar to Choice

implements "ItemListener"

When an item is selected, method "itemStateChanged" of the registered event-listener object is called.

(See Figures 10.14, 10.15)
List

- a scrolling list of text items
- user can choose either one item or multiple items
- generate both item events (single click) and action events (double clicks)

 Constructors:

```java
public List()
public List( int rows )
public List( int rows, boolean multipleMode )
```

Methods:

```java
public void add( String item )
public void add( String item, int index )
public void addActionListener( int Index )
public void addItemListener( )
public synchronized String[] getSelectedItems() 
```

(see Figures 10.16, 10.17)
Event-Driven Interaction

- **request mode**: input under control of application program
- **sample mode**: input devices and program operate concurrently
- **event mode**: input devices initiate data entry and control processing
  - Mouse
  - Keyboard

**Mouse Event**

- can be trapped for any GUI component derived from *component*

- Event listener:
  - MouseListener
  - MouseMotionListener
Event-Listener Interfaces

- public void `mousePressed` (MouseEvent e)
- public void `MouseClicked` (MouseEvent e)
- public void `mouseReleased` (MouseEvent e)
- public void `mouseEntered` (MouseEvent e)
- public void `mouseExited` (MouseEvent e)
- public void `mouseDragged` (MouseEvent e)
- public void `mouseMoved` (MouseEvent e)

Called automatically if event-listener is registered for a GUI component (See Figure 10.19)

use "event-listener adaptor" to override (define) method(s) needed for event-handling (see Figure 10.21)
How to preventing "update" from erasing the screen?

- By adding the following line in the applet.

  ```java
  public void update( Graphics g ) { paint( g ); }
  ```

This allows everything the user draws to remain on the screen.
**layout Manager**

- arranging GUI components on a container for presentation purpose

- FlowLayout (default for **applets** and **panels**)

```
[   ]  [   ]  [   ]
  (sequential)
```

- BorderLayout

```
[ N ]
[ W C E ]
[ S ]
```

- GridLayout
• use the container method "setLayout" to set layout manager

        container.setLayout( new FlowLayout( align ) );
        container.validate();

• each container can have only one layout manager of all the GUI components

Constructors:

public FlowLayout( )
public FlowLayout( int alignment )
public FlowLayout( int alignment,
                  int horizontalGap,
                  int verticalGap )

public BorderLayout( )
public BorderLayout( int horizontalGap,
                    int verticalGap )

public GridLayout( int rows, int columns )
public GridLayout( int rows, int columns,
                  int h, int v )
Defining Simple GUI Using Panels

- **Panel** inherits from "container"

- **Panels** can have components added to them

```
panel.setLayout( new GridLayout( 2, 8 ));
... (add GUI components to panel)

applet.setLayout( new BorderLayout() );
applet.add( panel, BorderLayout.EAST );
```

(See Figure 10.32)
GUI for Java Applications

Facts about Java applications

- Not as concentrated as applets
  (applets: how Java will be used in practice)

- Java applications are more trustworthy than applets
  (applets may be downloaded from suspect Web sites)

- Have more privileges than applets
  (such as with freer access to the local user’s files)

- Can not use the applet classes, but can use all the other classes

- Basic windowing unit is the "frame".

- Can use "menus" and "file dialog boxes"
Converting a Java Application into an Applet

- Create an HTML page with an "applet" tag to invoke the applet.

- Delete the "main" method.

- Alter the class header so that it extends "Applet" rather than "Frame".

- Add the "import" for the library class "Applet".

- Change the name of the constructor method from the name of the class to "init".

- Add an invocation of a method to set the layout of widgets in the applet window. Typically

  ```java
  setLayout( new BorderLayout() );
  ```
Converting a Java Applet to an Application

- Change the name of the "init" method to the name of the class and delete the work "void" in the header for this method
  (This is now the constructor method for the class)

- Alter the class header so that it extends "Frame" rather than "Applet"

- Create a new method called "main", with the header "public static void main( String args[] )". This method should create a "Frame" object as an instance of the class

- Delete the "import" for the class "Applet"

- Add a method to handle the "WINDOW_DESTROY" event

- Add an invocation for a method to set the layout of widgets in the frame. Typically

  ```java
  setLayout( new FlowLayout() );
  ```
**TextArea**

- Display multiple-line text or input multiple-line text (See Figure 11.3)

- Actually, facilities to cut, copy, paste and delete text are all available using a mouse.
Canvases

- Drawback with drawing in the applet window: can get in the way of the GUI components shown in the same window
- Dedicated, separate drawing area within a window
- There is a "paint" method that will be automatically called (can be used like the "paint" method in an applet)

Constructors: public `Canvas()`

Methods: public void `paint(Graphics g)`

- "paint" is called to repaint the canvas with background color. So should be overridden to draw on the canvas.
- A canvas should be created as a subclass of the library class "Canvas"
Example

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

public class CanvasDemo extends Applet {
    private MyCanvas canvas = new MyCanvas();

    public void init() {
        canvas.setBackground( color.gray );
        canvas.setSize( 200, 100 );
        add( canvas );
    }
}

class MyCanvas extends Canvas {

    public void paint( Graphics g ) {
        g.drawString( "I am in a canvas", 20, 20 );
    }
}
```

- 316 -

- Applet Viewer: ConvasDemo.class

<table>
<thead>
<tr>
<th>Applet</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.jpg" alt="Canvas" /></td>
</tr>
<tr>
<td>I am in a canvas</td>
</tr>
<tr>
<td>Applet started.</td>
</tr>
</tbody>
</table>

- Coordinates specified within "paint" are relative to the upper-left corner of the canvas, not the window.

- The size of the canvas has to be explicitly defined

- Canvases do not generate any events but are capable of recognizing mouse events

- Canvas can be (and, should be) customized to be a self-contained drawing class

(see Figures 11.4, 11.5)
Frames

• An application has to create its own window area called "frame".

  (an applet extends "Applet", an application extends "Frame")

• A "Frame" is a window with a title bar and a border.

  (The title and border of an applet you see when using an appletviewer to invoke an applet are those of the appletviewer’s)

Constructors:  
  public Frame() //invisible initially
  public Frame( String title )

Methods:  
  public void setTitle( String title )
  public void dispose( )
  public void setMenuBar( MenuBar mb )

• An application must terminate itself
  (no need for an applet to deal with this issue)
Example

public class MyFrame extends Frame {
    
    public MyFrame ( String s ){
        super( s );
        setSize( 275, 150 );
        addWindowListener( new CloseWindow() );
    }
}

public class CloseWindow extends WindowAdapter {

    public void windowClosing( WindowEvent e ) { 
        System.exit( 0 );
    }
}

******************************************************************
Converting an Applet into an Application

• An applet can run both as an applet and an application

(see Figure 11.10)
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class MyFrame extends Applet
    implements ActionListener {
    private ColorFrame f;
    private Button showFrame;

    public void init() {
        showFrame = new Button( "Show frame" );
        add( showFrame );
        showFrame.addActionListener( this );
        f = new ColorFrame( "Select a color" );
    }

    public void actionPerformed( ActionEvent e ) {
        f.setVisible( true );
    }

    public static void main( String args[] ) {
        ....
    }
}
## Input and Interaction

### Logical Classes of devices and techniques

<table>
<thead>
<tr>
<th>Logical Device</th>
<th>Function</th>
<th>Prototype (Physical device)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard (string)</td>
<td>Input a character string</td>
<td>alphanumeric keyboard</td>
</tr>
<tr>
<td>Locator</td>
<td>Indicate a position and/or orientation</td>
<td>Tablet, mouse, joystick</td>
</tr>
<tr>
<td>Pick</td>
<td>Select a displayed entity</td>
<td>Light pen</td>
</tr>
<tr>
<td>Choice</td>
<td>Select from a set of actions or choices</td>
<td>PFK, mouse</td>
</tr>
<tr>
<td>Dial (Valuator)</td>
<td>Input an analog value (number)</td>
<td>Slidebar, potentiometer</td>
</tr>
</tbody>
</table>
Physical Input Devices

Keyboard:
- each keystroke causes a 7-bit code (ASCII, EBCDIC, etc) to be stored in a character register, the CPU is interrupted
- the interpretation of the code is determined by the CPU program
- characters might be accumulated into a buffer until a terminating character is typed, thereby give the application program a character string input device

Tablet:
- A flat surface and a stylus or hand cursor
- electrical sensing mechanism is used to measure the stylus or cursor position (within half inch of the tablet surface)
- downward pressure on the stylus closes a microswitch (pressure-sensitive switch) and interrupts the computer
**Mouse**: --- most commonly used

- using mechanical detector
  or optical detector to
  measure motion

- mechanical mice measure
distance by turning a ball
(at the bottom) and consequently a pair of encoders.
The encoders measure motion in two directions.

- optical mice measure distance traveled by counting lines on a
  special pad

- a relative device, has no absolute origin,
  report only changes from their former position

- can be used to specify an arbitrarily large change in position

- the application program can reposition the cursor anywhere on
  the screen
**Light pen:**

- the pen sees the sharp burst of fluorescent light emitted when the electron beam is actually bombarding the phospher, not sensitive to the more prolonged phosphorescence or to ambient room light
- pen’s output connected to image display system (IDS); when IDS is interrupted, it sends CPU the contents of the $X$ and $Y$ registers which are the $x$ and $y$ coordinates of the pixel detected (hence, on a raster-scan display, light pen implements a locator)
- popularity of light pens will decrease in the future
**Valuator**: (prototype: potentiometer)

- can be rotary or slide potentiometers (slide bars)
  (e.g., volume, balance, and tone controls on a stereo set)

![Diagram of a Valuator](image)

- by rotating the dial (for rotary potentiometer) or moving the slider (for slide potentiometer), scalar can be generated and placed in DPU device registers and then read by CPU
Input Modes:

- Defined by the relationship between the measure process and the trigger
  Measure: what the device returns to the user program
  Trigger: a physical input on the device

- The display processing unit contains a number of registers (buffers). Once initialized, input devices store appropriate values in these registers

**Request mode:** The measure of the device is not returned until the device is triggered

**Sample mode:** Input provides immediate input
No trigger is needed

**Event mode:** when a device is triggered, the device measure with the identifier for the device is placed in an "event queue"
(but application program is not interrupted)
Menus: a simple example

Fixed menu items with each of them serving a simple purpose

Advantages: simple to implement

Disadvantages: occupying a large area of screen and not leaving enough room for much else

Alternatives: pull down menus, pop up menus
**Pull-Down Menus:**

- A small menu is always on display, usually at the top of the screen. When a user makes a selection, the single item is replaced with a longer menu, which disappears after the selection has been done.

  e.g.

<table>
<thead>
<tr>
<th></th>
<th>Color</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Purple</td>
<td></td>
</tr>
<tr>
<td></td>
<td>White</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Black</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Red</td>
<td>Style</td>
</tr>
</tbody>
</table>

  If "color" is picked

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blue</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Black</td>
<td></td>
</tr>
<tr>
<td></td>
<td>White</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Purple</td>
<td></td>
</tr>
</tbody>
</table>

  After a color has been selected

<table>
<thead>
<tr>
<th></th>
<th>Color</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pop-Up Menus:

- Always hidden. It appears in response to a particular user action (such as pressing the right button of the mouse). It stays visible only until the user has made a choice.
- Sometimes a choice may cause another menu to appear.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td></td>
</tr>
<tr>
<td>Edit</td>
<td>Undo</td>
</tr>
<tr>
<td>Go</td>
<td>Copy</td>
</tr>
<tr>
<td></td>
<td>Paste</td>
</tr>
<tr>
<td></td>
<td>Find</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation: The menu corresponding to "Edit" disappears after one of its items is selected.
Java's Menu Components

Menu component hierarchy:

(see Figures 11.11, 11.14)
Sliders and Scrollbars:

- Offer the user a choice from a continuous range (instead of a choice of discrete options offered by menus)

Example of a Scrollbar:

Implementation: the shaded area of the bar is moved when a button is pressed and moved, and stopped when the button is released

\[ v = v_1 + \frac{(x - x_1)}{(x_2 - x_1)} \]

(see Figure 11.13)
Dialogue Panel:

- menu selection can be used for many instructions, but typing information such as file names, often is necessary

Java Dialog boxes

- Gather (show) information from (to) the user
- Can either be modal or modeless

(see Figure 11.11)
Eight Golden Rules of Interface Design:
(Designing the User Interface, B. Shneiderman, 1998, Addison-Wesley)

1. Strive for consistency

2. Enable frequent users to use shortcut

3. Offer informative feedback

4. Design dialogs to yield closure

5. Offer error prevention and simple error handling

6. Permit easy reversal of action

7. Support internal locus of control

8. Reduce short-term memory load