How many of you…

have implemented a command-line user interface?
How many of you…

have implemented a **graphical user** interface?

- HTML/CSS
- Java Swing
- .NET Framework
- Mozilla’s XUL
- Mobile platform (iOS, Android, Blackberry, …)
- Something else?

---

What’s the difference?

**Command-line model** (e.g., UNIX shell, DOS)

- Interaction controlled by system
- User queried when input is needed

**Event-driven model** (e.g., GUIs)

- Interaction controlled by the user
- System waits for user actions and then reacts
- More complicated programming and architecture
- Need to build the “look” and “feel” of interface
Component/Container Model

Component (aka widget, control, etc.)

- Encapsulation of an interactive element
- Drawn using the 2D graphics library
- Low-level input event processing
- Repaint management

- In OOP systems, each component is implemented as a sub-class of a base “Component” class

Examples of Components

- Button
- Checkbox
- Radio button
- Text box
- Combo box (drop-down list)
- List box
- Scrollbar
- Slider
- Menu
- Menu item
- NumericPicker
- DateTimePicker
- …
Java Swing Components

.NET Framework Controls
HTML Form Controls

Component/Container Model

Container

• Component that contains one or more other components
• Creates the structure of the user interface
• Manages child components
  • Layout, painting, event dispatch
• Some have interactive features (e.g., tab panel)
Container Structure

Label

Textbox

Buttons

Panels

Container Structure

Label

Textbox

Buttons

Panels
Container Structure

Containers specify layout of their children

Layout

Containers specify layout of their children
Layout

Containers specify layout of their children

```
Window
  Panel
    Label
    Textbox
    Panel
```

“Feel”: Events

User input is modeled as “events” that must be handled by the system

Examples?
  - Mouse
    button down, button up, button clicked, entered, exited, moved, dragged
  - Keyboard
    key down, key up, key pressed
  - Window
    movement, resizing
  - Touchscreen
    Touching, swiping, dragging, pinching
Anatomy of an Event

An event encapsulates the information needed for handlers to react to the input

- Event type (mouse button down, key up, etc.)
- Event target (component in which event occurred)
- Timestamp
- Modifiers (Ctrl, Shift, Alt, etc.)
- Type-specific content
  - Mouse: x,y coordinates, # clicks
  - Keyboard: key code

Event Handlers

Events are dispatched to components

- Application developers can specify code to be executed when the event occurs (callbacks)
- Built-in components will have code to handle most keyboard and mouse events
  - Buttons handle mouse up/down to change graphic
  - Text boxes update their contents on key press
- Built-in components often generate new “high-level” events from combinations of low-level events
  - Text boxes generate “change” events when contents changes and focus is lost
  - Sliders generate “change” events when thumb is dragged
Event Loop

Input Devices → Event Queue → Event Loop

while(!done) {
  evt = dequeue_event();
  dispatch_event(evt);
  repaint_screen();
}

Blocks until an event arrives

Exists in every application
Usually handled for you by UI framework
Event Loop

Input Devices → Event Queue → Event Loop

Mouse up (10,20)
key down ('h')
key up ('h')
key down ('i')

while(!done) {
    evt = dequeue_event();
    dispatch_event(evt);
    repaint_screen();
}

Most of the work happens here

Dispatching Events

mouse down (10,50)

Window
  Panel
    Label
    Textbox
    Panel
      Button
      Button

function onMouseDown(evt) {
    // do something...
}
Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}

Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}
Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}

Dispatching Events

mouse down (10,50)

function onMouseDown(evt) {
    // do something...
}
MODEL VIEW CONTROLLER (MVC)

- Architecture for interactive apps
- Partitions application in a way that is
  - Scalable
  - Maintainable

MVC

- Architectural design pattern which works to separate data and UI for a more cohesive and modularized system

- Presented by Trygve Reenskaug in 1979
- First used in the Smalltalk-80 framework
  - Used in making Apple interfaces (Lisa and Macintosh)
MVC

- Model represents the data model
  - “Manages behavior and data of the application domain”
- View represents the screen(s) shown to the user
  - “Manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application”
- Controller represents interactions from the user that changes the data and the view
  - “Interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate”

Example Application

Blue circles: 4
Cardinal squares: 2
**Model**

Information the app is trying to manipulate  
Representation of real world objects  
- Circuit for a CAD program  
- Shapes in a drawing program  
- List of people in a contact management program

**View**

Implements a visual display of the model  
May have multiple views  
- E.g., shape view and numeric view
Multiple Views

- Implements a visual display of the model
- May have multiple views
  - E.g., shape view and numeric view

Any time the model is changed, each view must be notified so that it can update later.
Controller

- Receives all input events from the user
- Decides what they mean and what to do
  - Communicates with view to determine the objects being manipulated (e.g., selection)
  - Calls model methods to make changes to objects
Controller

Click

Blue circles: 3
Cardinal squares: 2

Controller

Click

Blue circles: 3
Cardinal squares: 2
Controller

Combining View & Controller

- View and controller are tightly intertwined
  - Lots of communication between the two
  - E.g. determine what was clicked on
- Almost always occur in pairs
  - i.e., for each view, need a separate controller
- Many architectures combine into a single unit
Why MVC?

• Mixing all pieces in one place will not scale
  • Model may have more than one view
    • Each is different and needs update when model changes
  • Separation eases maintenance and extensibility
    • Easy to add a new view later
    • Model can be extended, but old views still work
    • Views can be changed later (e.g., add 3D)

Android: Getting Started & Main Tools

• Getting and installing Android Studio:
  • http://developer.android.com/training/basics/firstapp/index.html

• Gradle: build toolkit (manages dependencies)
• API: Application Programming Interface
• SDK: Software Development Kit
• Activity: single/focused “thing” a user can do
• Fragment: reusable behavior/portion of user interface in activity
• SDK Manager: manage platforms/tools/components
• AVD Manager: Android Virtual Device manager
• adb: Android Debug Bridge
Android Versions

<table>
<thead>
<tr>
<th>Code name</th>
<th>Version</th>
<th>API level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lollipop</td>
<td>5.0</td>
<td>21</td>
</tr>
<tr>
<td>KitKat</td>
<td>4.4.x</td>
<td>19</td>
</tr>
<tr>
<td>Jelly Bean</td>
<td>4.1.x, 4.2.x, 4.3.x</td>
<td>16 - 18</td>
</tr>
<tr>
<td>Ice Cream Sandwich</td>
<td>4.0.3, 4.0.4</td>
<td>15</td>
</tr>
<tr>
<td>Ice Cream Sandwich</td>
<td>4.0.1, 4.0.2</td>
<td>14</td>
</tr>
<tr>
<td>Honeycomb</td>
<td>3.x</td>
<td>11-13</td>
</tr>
<tr>
<td>Froyo, Gingerbread</td>
<td>2.2.x - 2.3.x</td>
<td>8-10</td>
</tr>
<tr>
<td>Cupcake, Donut, Eclair</td>
<td>1.x - 2.1.x</td>
<td>1-7</td>
</tr>
</tbody>
</table>

Target API Compiled SDK

Minimum API

< ~8% users

Package Content (Eclipse View)

- All source code here
- Java code for our activity
- All non-code resources
- Images
- Generated Java code
- Helps link resources to Java code
- Layout of the activity
- Strings used in the program
- Android Manifest
App Structure/Files

- AndroidManifest.xml: essential info about your app
- R class: definitions of all resources; namespace (R = resources)
- Folders: drawable (images), layout (xml for activity user interface), menu (menu management)
- Intent: abstract description of operation to be performed
  - startActivity
  - putExtra + getStringExtra
- Log messages: debugging

R Class

- Auto-generated: you shouldn’t edit it
- Contains IDs of the project resources
- Enforces good software engineering
- Use findViewById and Resources object to get access to the resources
  - Ex. Button b = (Button) findViewById(R.id.button1)
  - Ex. getResources().getString(R.string.hello);
Mini Project Tips

• Map view + marker:
  • http://www.vogella.com/tutorials/AndroidGoogleMaps/article.html
• Location: next slide
• Accelerometer: similar to LocationManager (SensorManager):
  • http://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--mobile-22125
• Second activity + intent: see before
• List view:
  • http://www.vogella.com/tutorials/AndroidListView/article.html

GPS

Context theContext = this.getApplicationContext();
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        String newLatitude =
            Double.toString(location.getLatitude());
        String newLongitude =
            Double.toString(location.getLongitude());
        Toast.makeText(YourClass.this, "New location is detected: " + count + "->(" + newLatitude + "," +
            newLongitude + ")", Toast.LENGTH_SHORT).show();
        count++;
    }
};
Permissions

- `<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>`

Android Manifest (Example)

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name="HelloAndroid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```
Activity

- An Android activity is focused on a single thing a user can do.
- Most applications have multiple activities

HelloAndroid.java

```java
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android – by hand");
        setContentView(tv);
    }
}
```

Set the view “by hand” – from the program

Inherit from the Activity Class
/res/layout/main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>
```

Further redirection to
/res/values/strings.xml

/res/values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloAndroid – by resources!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```
HelloAndroid.java

package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
public class HelloAndroid extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
Set the layout of the view as described in the main.xml layout