MOBILE COMPUTING

CSE 40814/60814
Spring 2021

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

Container Structure

Label
Textbox
Panels
Buttons
Container Structure

Containers specify layout of their children

Layout

Containers specify layout of their children
Layout

Containers specify layout of their children

“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 content changes and focus is lost
  - Sliders generate “change” events when thumb is dragged
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();
}

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();
}

Blocks until an event arrives
Event Loop

Most of the work happens here

Dispatching Events

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

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

Dispatching Events

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

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

Dispatching Events

```javascript
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**: data model
  - manages behavior and data of the application domain

- **View**: 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**: 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**

![Image of a game interface with blue circles and red squares]

- **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
- Anytime the model is changed, each view must be notified so that it can update *later*

**View**

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

- Blue circles: 3
- Cardinal squares: 2
Controller

Blue circles: 3
Cardinal squares: 2

Click
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 that into a single unit

Combining View & Controller
One Model, Many Controllers

One Model, Many Controllers

MVC Feedback Loop

MVC Feedback Loop
Android View Class

- The View class is the Android’s most basic component from which users interfaces can be created. This element is similar to the Swing JComponent class for Java apps.
  - A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
  - Widgets are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.
  - Layouts are invisible containers used for holding other Views and nested layouts.
Graphical UI – XML Layout

Actual UI displayed by the app

Text version: activity_main.xml file

Examples of UI Components

Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.

Relative Layout
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

Table Layout
A TableLayout is a ViewGroup that places elements using a row & column disposition.
Examples of UI Components

- **DatePicker**
  - A `DatePicker` is a widget that allows the user to select a month, day and year.

- **Form Controls**
  - Includes a variety of typical form widgets, like:
    - image buttons
    - text fields
    - checkboxes
    - radio buttons

- **Widgets**

- **GalleryView**

- **TabWidget**

---

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)