MVC:
Model View Controller

Lecture 24
Motivation

- Basic parts of any application:
  - Data being manipulated
  - A user-interface through which this manipulation occurs

- The data is logically independent from how it is displayed to the user
  - Display should be decoupled from content
  - Single-point-of-control over change

- Example: grade distribution in class
  - Could be displayed as a pie chart, or a bar chart, or a cumulative fraction plot, or...
Architecture: Desktop App

- User Interface
  - Graphical events (mouse moves, button pushed)
- Application
  - Processing, Calculating
- Data
  - Persistence, Transactions, Triggers
Model-View-Controller Pattern

- **Model**
  - The data *(i.e. state)*
  - Methods for accessing and modifying state

- **View**
  - Renders contents of model for user
  - When model changes, view must be updated

- **Controller**
  - Translates user actions *(i.e. interactions with view)* into operations on the model
  - Example user actions: button clicks, menu selections
Basic Interactions in MVC

- **Input**: "user action"
- **Controller**: "change data" ➔ "new state"
- **Model**: "change display"
- **View**: Output
Implementing Basic MVC in Swing

- Mapping of classes to MVC parts
  - View is a Swing widget (JFrame, JButton, etc.)
  - Controller is an event handler (ActionListener)
  - Model is an ordinary Java class (or database)

- Alternative mapping
  - View is a Swing widget and includes (inner) class(es) as event handlers
  - Controller is an ordinary Java class with "business logic", invoked by event handlers in view
  - Model is an ordinary Java class (or database)

- Difference: Where is the event listener?
  - Regardless, model and view are completely decoupled (linked only by controller)
Example: Simple MVC GUI Demo

Controller

Model

JFrame

View

Swing Components

ActionListener

extends

implements

“instance of this class holds a reference to instance of that class”
Wiring Parts Together

```java
void actionPerformed(ActionEvent e) {
    ...
}

void multiplyBy(String arg) {
    ...
}
```
public class CalcView extends JFrame {
    private JButton multiplyBtn = new JButton("X");

    public void register(ActionListener x) {
        multiplyBtn.addActionListener(x);
    }
}

public class CalcController {
    ...
    view.register(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            ...
        }
    });
}
Basic MVC in JavaScript

- Mapping of classes to MVC parts
  - View is an HTML page
  - Controller is event handler, an ordinary JavaScript function
  - Model is an ordinary JavaScript object

- Alternative mapping
  - Separate event handler(s) from controller
  - Controller is an ordinary object with “business logic”, invoked by event handlers
  - Model is an ordinary object

- Difference: Where is the event listener?
  - Regardless, model and view are completely decoupled (linked only by controller)
Wiring Parts Together

Calculator.html

ActionListener.js

```javascript
function multiplyListener (event) {
  ...
}
```

CalculatorState.js

```javascript
function multiplyBy (arg) {
  ...
}
```
Registering an Event Handler

- Three techniques, ordered from:
  - Oldest (most brittle, most universal) to
  - Newest (most general, least standard)

1. Inline (link in HTML itself)
   \(<a\ href="page.html"\ onclick="foo()">…\>

2. Direct (link in JavaScript)
   \(\text{var } e = \ldots \text{//find source element in tree}\)
   \(e.\text{onclick} = \text{foo};\)

3. Chained (In JavaScript, browser differences)
   \(\text{var } e = \ldots \text{//find source element in tree}\)
   \(e.\text{addEventListener("click", foo, false);}\)
Basic MVC in Objective-C
Implementing MVC in XCode
Implementing MVC in XCode
Basic Web App Skeleton: 3-Tier

User Interface

http

HTML, CSS, Javascript

Application

......

SQL

Data

http

HTML, CSS, Javascript

Application

......

SQL

Data

http

HTML, CSS, Javascript

Computer Science and Engineering - The Ohio State University
MVC in a Web Application

- **Model**
  - Database (table with rows)
  - Classes that wrap database operations (class with instances)

- **View**
  - HTML (+ CSS, JavaScript) files rendered by client's browser
  - Skeleton files used by server to generate these HTML files

- **Controller**
  - Receives HTTP requests via web server
  - Orchestrates activity (model and view)
MVC with Rails
MVC with Rails

Ruby on Rails
Web Applications

HTTP, RSS, ATOM or SOAP

Apache, WeBrick or Lighttpd

Invokes FastCGI, mod_ruby or CGI processor

Handles some validations

MySQL, PostgreSQL or Oracle

MySQL, PostgreSQL or Oracle

CRUDs

Responds

Active Record

Queries

Data or Errors

Database

Active Record

Redirects

Controller

Delegates

Action WebServices

Delivers

Responds

Controller

Renders

Action View

Displays

XHTML, CSS, JS & images, XML

XML response

Requests

Web Server

Forwards

Dispatcher

Browser or client

Responds

Action Mailer

Action WebServices

Action View

Computer Science and Engineering  The Ohio State University
Directory Structure of Rails

depot/
  ....../app
  .........../controllers
  .........../helpers
  .........../models
  .........../views
  ............../layouts
  ....../components
  ....../config
  ....../db
  ....../doc
  ....../lib
  ....../log
  ....../public
  ....../script
  ....../test
  ....../tmp
  ....../vendor
  ....../README
  ....../Rakefile
"Convention Over Configuration"

- Use naming & location conventions to wire components together *implicitly*
- Explicit routing too, based on *names* and pattern matching
- Contrast with:
  - Configuration files (e.g., XML)
  - Configuration code (e.g., Swing register listener)
  - Configuration tools (e.g., IDEs to connect GUI widgets to code snippets)
Wiring Parts Together in Rails

- Example: Event → Controller wiring
  - HTTP GET request for URL /say/hello gets routed to controller:
    - Class called SayController
    - File say_controller.rb in app/controllers
    - Method hello

- Example: Controller → View wiring
  - HTTP response formed from:
    - File app/views/say/hello.html.erb

- Example: Model → Database wiring
  - Class Order maps to database table "orders"
  - Attributes of Order map to columns of table
  - Instances of Order map to a rows of table
Summary

- Programming Patterns
  - Common idioms for solving categories of problems
  - Example: Observer pattern, MVC
- Separation of concerns
  - Decouple state from business logic
  - Decouple business logic from display
- Rails: Convention over configuration
  - Parts are wired together based on naming and structuring conventions
  - Defaults can always be overridden (but better not to fight!)