MVC

- THE classic design pattern
- Used for data-driven user applications
- Such apps juggle several tasks:
  - **Loading** and **storing** the **data** – getting it in/out of storage on request
  - **Constructing** the **user interface** – what the user sees
  - **Interpreting** **user actions** – deciding whether to modify the UI or data
- These tasks are largely independent of each other
- Model, View, and Controller each get one task
Model
talks to data source to retrieve and store data

Which database tables is the requested data stored in?

What SQL query will get me the data I need?
View

asks model for data and presents it in a user-friendly format

Would this text look better blue or red? In the bottom corner or front and center?

Should these items go in a dropdown list or radio buttons?
Controller

listens for the user to change data or state in the UI, notifying the model or view accordingly

The user just clicked the “hide details” button. I better tell the view.

The user just changed the event details. I better let the model know to update the data.
MVC: Summary

Model
talks to data source to retrieve and store data

Controller
listens for the user to change data or state in the UI, notifying the model or view accordingly

View
asks model for data and presents it in a user-friendly format
Communication Flow

Aside: HW 8

- Applying your generic graph & Dijkstra’s to campus map data
- Given a list of buildings, and walking paths
- Produce routes from one building to another on the walking paths
- Command-line interface now, GUI in HW9

DEMO
Aside: HW 8 Data Format

- List of buildings (abbrev, long name, loc in pixels):
  BAG  Bagley Hall (East Entrance)  1914.51031709.8816
  BAG (NE) Bagley Hall (Northeast Entrance)  1878.37861661.4083
  BGR  By George  1671.54991258.4333

- List of paths (between two pixels, dist in feet):
  1903.7201,1952.4322
  1906.1864,1939.0633: 26.583482327919597
  1897.9472,1960.0194: 20.597253035175832
  1915.7143,1956.5: 26.68364745009741
  2337.0143,806.8278
  2346.3446,817.55768: 29.685363221542797
  2321.6193,788.16714: 49.5110360968527
  2316.4876,813.59229: 44.65826043418031

- Remember (0,0) is in the upper left (not lower)
Aside: HW 8 Output

- List of walking directions between two given points
  - Distance in feet
  - Directions:
MVC in HW8

- Model stores graph, performs Dijkstra’s
- View shows results to users in text format
- Controller takes user commands and uses view to show results
- View and Controller changed in HW9, but Model stays the same
Benefits of MVC

● Organization of code
  ● Maintainable, easy to find what you need

● Ease of development
  ● Build and test components independently
  ● Different people work on different parts at the same time, designers can work on the view even if they don’t understand code

● Flexibility
  ● Swap out views for different presentations of the same data (ex: calendar daily, weekly, or monthly view)
  ● Swap out models to change data storage without affecting user
Communication Flow & User Interaction

Model <-> View

Controller

User

User looks at view

User interacts with controller
Communication Flow & User Interaction

- If the user only interacts with controller, then how to update view, model?
  - Callbacks

- Remember callbacks are different than calls
  - Think synchronous and asynchronous
  - Not blocking & non-blocking
Callbacks

- **Synchronous callbacks:**
  - Examples: HashMap calls its client’s hashCode, equals
  - Useful when the callback result is needed immediately by the library

- **Asynchronous callbacks:**
  - Examples: GUI listeners
  - *Register* to indicate interest and where to call back
  - Useful when the callback should be performed later, when some interesting event occurs
Asynchronous callbacks

- Asynchronous callbacks:
  - Examples: GUI listeners
  - Register to indicate interest and where to call back
  - Useful when the callback should be performed later, when some interesting event occurs

Client

Library

Register

Return

Event

Callback

Return, so library can finish processing event
Asynchronous callbacks

- Calendar asynchronous callback demo
  - Form’s calendar registers to receive click events by adding the interaction method to calendar’s list of methods to call when it’s clicked.
    ```csharp
    this.calendar1.DateChanged +=
    new Forms.DateRangeEventHandler(
        this.calendar1_DateChanged
    );
    ```
  - When calendar is clicked it alerts everyone who signed up to be notified of the click.
  - The callback is executed
    ```csharp
    private void monthCalendar1_DateChanged(
        object sender,
        DateRangeEventArgs e) {
        MessageBox.Show("Calendar clicked: " + e.Start);
    }
    ```
User adds comment, AJAX sends to server, waits for callback

Server starts work to save comment

Browser shows page

Server

Return HTTP OK

Callback to browser

Browser updates UI to show success

When save to database is complete
Callbacks & MVC

- Controller utilizes callbacks to respond to user events, update the model
- View uses callbacks to update when the model changes
- Callbacks are used very commonly outside MVC as well, especially in distributed systems
MVC in industry

Image stitcher demo
http://research.microsoft.com/en-us/um/redmond/groups/ivm/ice/

Ruby on Rails / Django enforce programmatically

- models, views, and controllers folders

Lab Specimens
http://code.google.com/p/lab-specimen-transport-system/

BeatTide
http://beattide.herokuapp.com/
https://github.com/djmailhot/BeatTide/tree/master/app
MVC Example – Traffic Signal

- Regulate valid traffic movements (i.e. don't run cars into each other)
- Detect cars waiting to enter intersection
- Detect pedestrians waiting to cross street.
- Traffic lights to direct car traffic
- Pedestrian signals to direct peds to cross
- Manual override for particular lights (i.e. disable traffic signals for a parade)
- External timer which triggers changes in light at set interval
MVC Example – Traffic Signal

- Model:
  - stores current state of traffic flow
  - stores whether cars and pedestrians who are waiting
  - "Java" interface:
    - getCurrentTrafficDirection()
    - carWaiting(direction)
    - pedWaiting(direction)
    - timeStep() // May skip a light cycle
  - Implements Observable
MVC Example – Traffic Signal

- Views:
  - CarLight
    - Each instance knows what direction it is associated with.
    - Observes TrafficModel
  - PedLight
    - same as CarLight, but for pedestrians
MVC Example – Traffic Signal

- **Controllers:**
  - PedButton
    - Is aware of what TrafficModel it controls, and its direction
    - When triggered, calls pedWaiting (direction) on that TrafficModel
  - CarDetector
    - is aware of TrafficModel and direction
    - When triggered, calls carWaiting (direction)
MVC Example – Traffic Signal

- Controllers (cont’d):
  - LightSwitch:
    - aware of what light it controls
    - when triggered, enables or disables the light

- Timer:
  - Somehow regulates time (how is not important)
  - aware of a TrafficModel
  - calls timeStep() at a regular interval
MVC Example – Registration

- Registration system with web interface
- Advisors create classes, set space, time, restrictions
- Professors can see who’s signed up for their class
- Students can sign up for classes, see available classes, see what they’ve signed up for
- Administrators can place holds on student registration
- Professors can be notified when a student drops
- Students can be notified when a spot is available in a class they want
MVC Example – Wrapup

- Did you imagine a push or a pull model (or both)?
- What would change for interaction with an API, or mobile app?
- Now advisors can see what students are registered for, change their registration, what changes?