Section 4!

Presenting, for your educational enrichment…

Fun With Testing II
Model-View-Controller
And other sundries

CSE 331, 10/13/11 *TA: Krysta Yousoufian*

Announcements

- **Affiliates Career Fair. BE THERE!**
  - Today, 10-3, Atrium + Gates Commons
- Subclipse
  - Don't actually have to use Team Sync view
  - Package Explorer: right-click project under version control, use "Team" menu
- Assignment 1
  - Lenient grading on many items
  - Will send out list of "-0" items. READ IT. Incorporate it in A3.
- Assignment 3: questions?

A1 Unit Test Pitfalls

- DO: comment tests – but not with JavaDoc
- DO: use descriptive names
  - Not "testGetItem1, testGetItem2, …"
- DO: test ALL non-constructor methods
  - Even accessors for now.
  - Black-box tester doesn't know if accessors are simple inside
- DO: test common cases

A1 Unit Test Pitfalls

- DO: split up tests into separate methods.
    - Easier to see what failed
    - Separate test method for each input condition
    - Separate test method for each method being tested
    - DON'T write one test method for each class or method
    - A1: lots of people's tests were too long

A1 Unit Test Pitfalls

- Caveat to splitting up tests
  - Sometimes one method can't be tested without another.
  - If foo() has to be tested by calling bar() but not vice versa, still write testFoo and testBar methods.
  - If setFoo() and getFoo() can't be separated (except when testing getFoo's initial value), just write one (or several) testChangeFoo methods.
- "Test smarter, not harder" – bigger != better
- Gold star: store parameter values in local variables or class fields instead of hard-coding

Pitfalls: Checking Values

- DO: check values with assert()
- DON'T: use println().
- DON'T: write methods that never assert() or fail()
    - … *unless* testing for an exception with "expected" option
    - Tests that don't assert() anything are usually not useful
- DO: use the right assert() for the occasion
    - assertTrue(a) instead of assertEquals(true, a)
    - assertEquals(a, b) instead of assertTrue(a == b)
    - Check values inside assert instead of using if, assertTrue(true), and fail()

Pitfalls: What to Test

Test all edge cases, input conditions, etc.

- Methods returning boolean: test false AND true

  `isEmpty(), matches()`

- Accessor/mutator: test default value, after changing to multiple values

  `hasDiscount():` original value, set true, set false

- Test all combinations of input/state categories (within reason…)

  `getTotal()`: discount [was | wasn't] requested, cart [does | doesn't] have enough items

- Test getItem(), totalQuantity(), getTotal() after adding multiple items, replacing item

A3 Expectations

- Stricter than A0 – check those -0 points
- Always throw exceptions on bad input
  - **Don't** force bad input into good input (e.g. price of 0 to $0.01)
- Write descriptive comments with JavaDoc tags
- Agree with partner on coding conventions
  - Ideally, official Java coding conventions
  - May lose points if significantly different styles
- Tests should be:
  - Thorough: ALL methods, common cases, edge/invalid/boundary conditions
  - Well-documented: non-Javadoc comments, good method names
  - Well-organized: separate method for each scenario
  - Well-organized: test suite with one test class for each regular class

Test Your Testing Skillz…

Schedule Class

- http://www.cs.washington.edu/education/courses/cse331/11au/sections/schedule/Schedule.html

Model-View-Controller

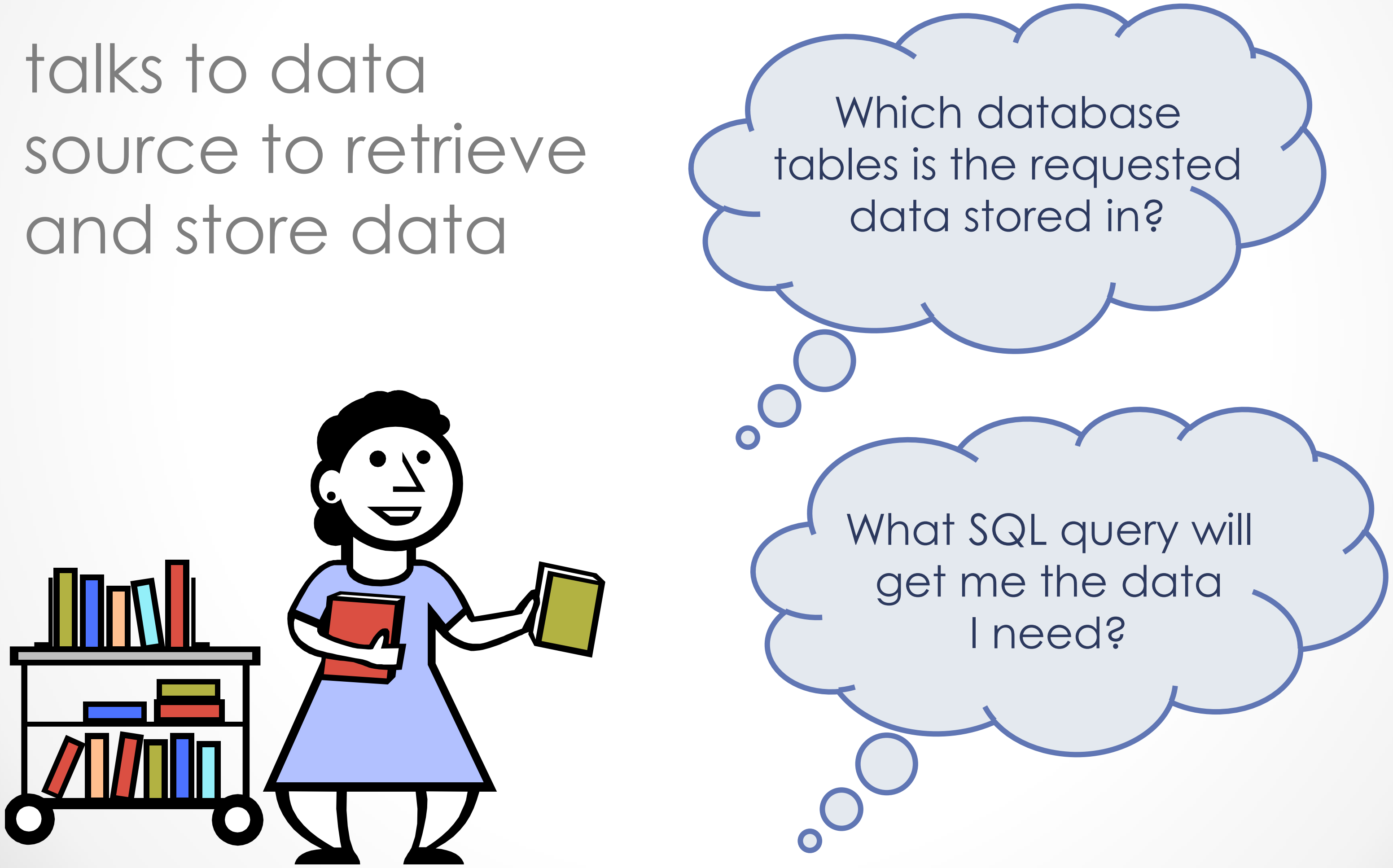
(or Model-View-Presenter)

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

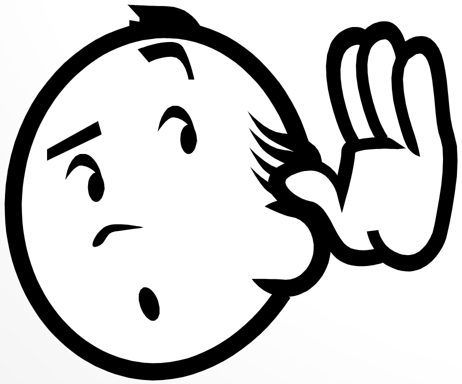
asks model for data and presents it in a user-friendly format

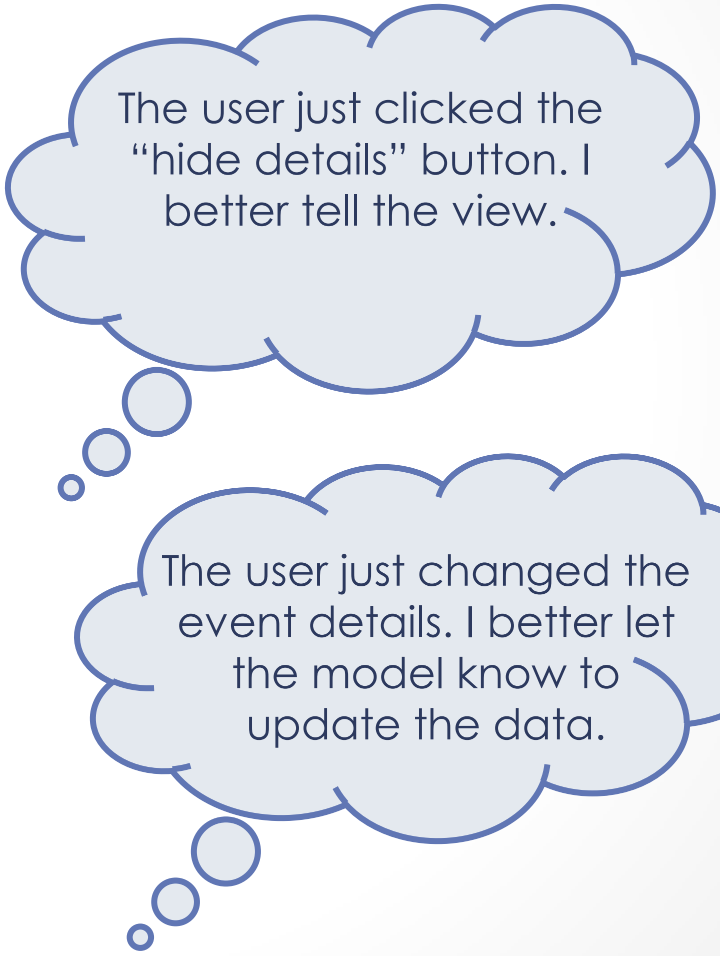
Would this text look better blue or red? In the bottom corner or front and center?

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

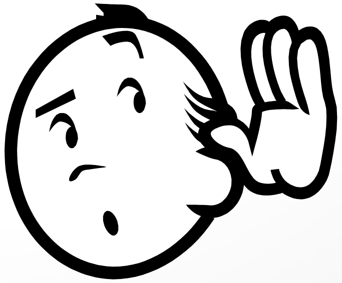
talks to data source to retrieve and store data

View

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

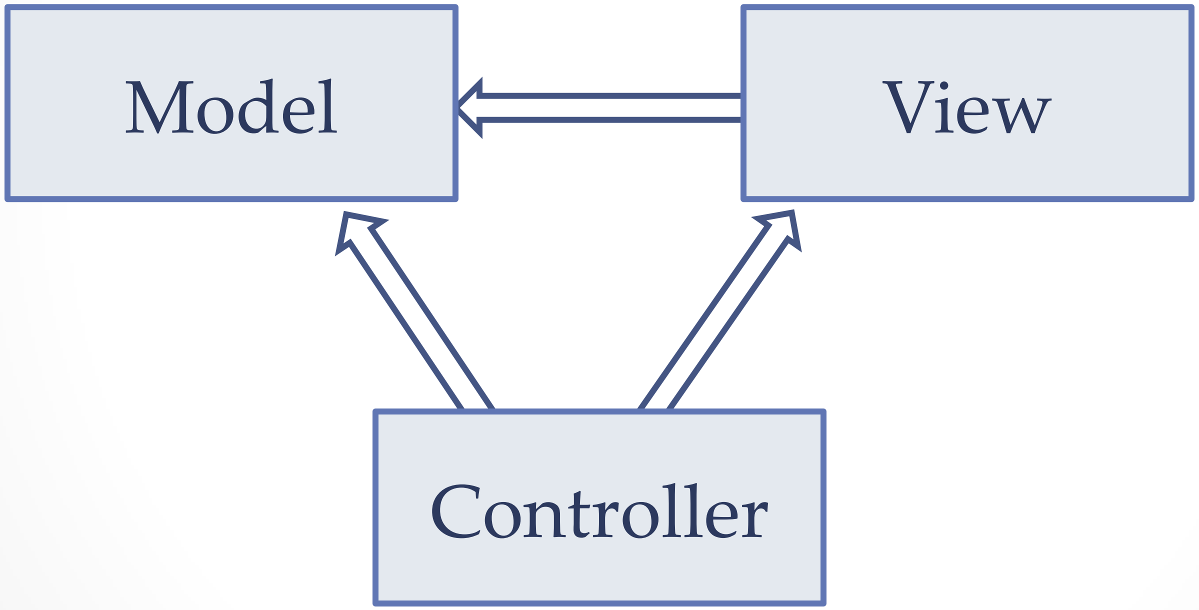
Controller

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



Communication Flow

Taken from http://msdn.microsoft.com/en-us/library/ff649643.aspx



What do you think are the benefits of MVC?

Benefits of MVC

- Organization of code
  - Maintainable, easy to find what you need
- Ease of development
  - Build and test components independently
- 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