

Model-View-Controller (MVC) Design Pattern

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

Lecture 23

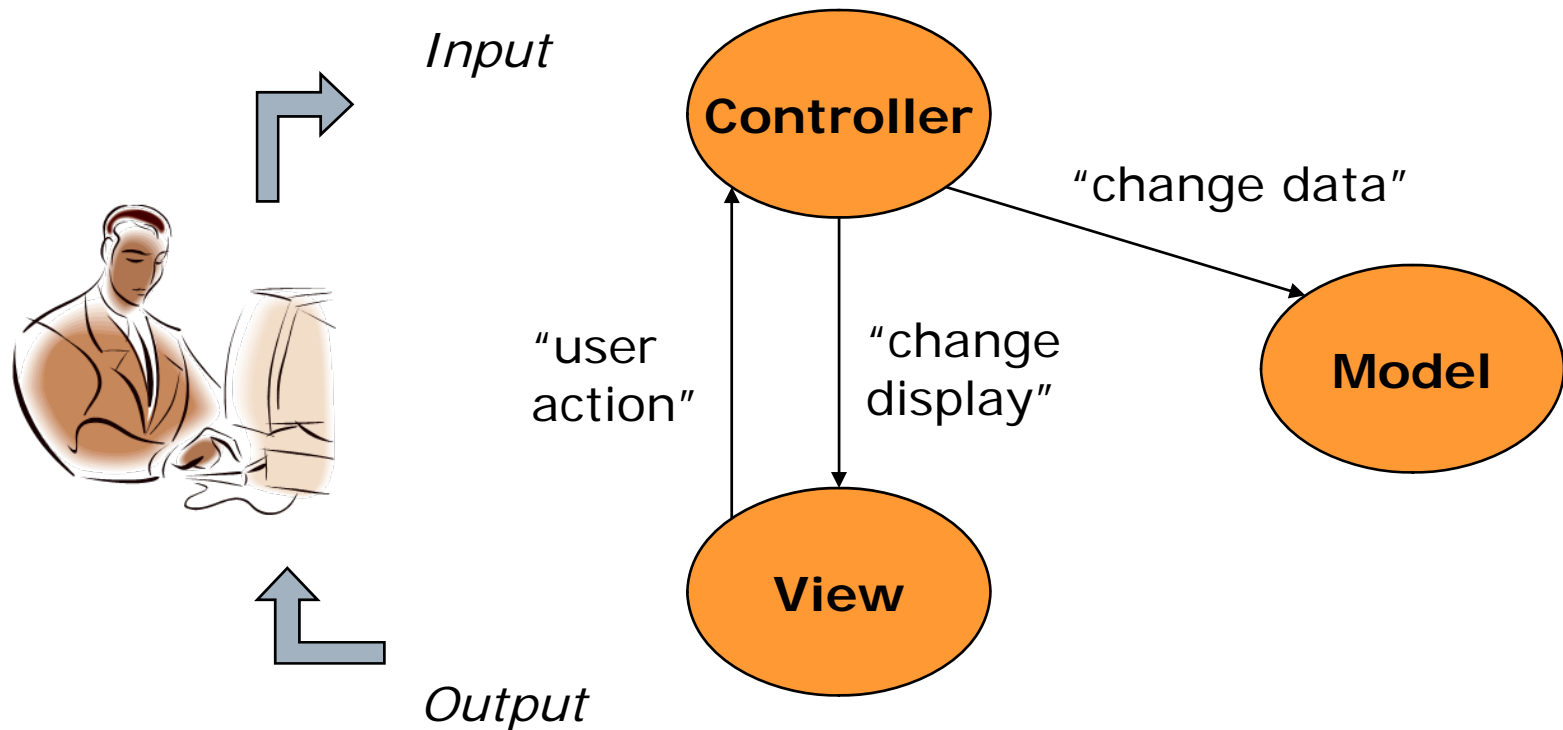
Motivation

- 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 separately designable/evolvable
- Example: grade distribution in class
 - Displayed as both pie chart and/or bar chart
- Anti-example: see BigBlob
 - Presentation, logic, and state all mixed together

Model-View-Controller Pattern

- Model
 - The data (ie 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 (ie interactions with view) into operations on the model
 - Example user actions: button clicks, menu selections

Basic Interactions in MVC



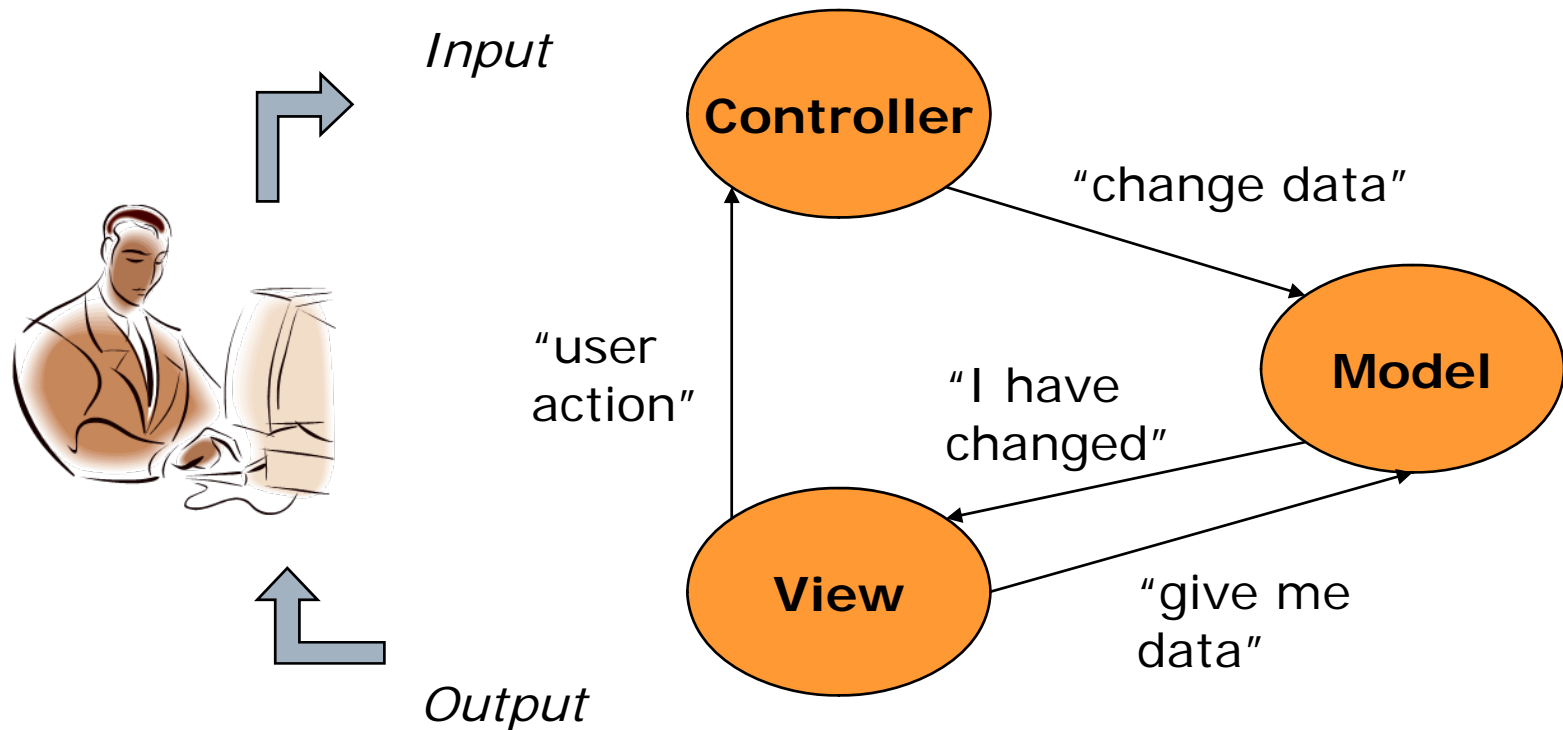
Implementing Basic MVC in Swing

- Mapping of classes to MVC parts
 - View is a Swing widget (like a JFrame & JButtons)
 - Controller is an ActionListener
 - Model is an ordinary Java class (or database)
- Alternative mapping
 - View is a Swing widget and includes (inner) ActionListener(s) 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 ActionListener?
 - Regardless, model and view are completely decoupled (linked only by controller)

Mechanics of Basic MVC

- Setup
 - Instantiate model
 - Instantiate view
 - Has reference to a controller, initially null
 - Instantiate controller with references to both
 - Controller registers with view, so view now has a (non-null) reference to controller
- Execution
 - View recognizes event
 - View calls appropriate method on controller
 - Controller accesses model, possibly updating it
 - If model has been changed, view is updated (via the controller)
- Example: CalcMVC
 - CalcModel, CalcView, CalcController
 - Note: View includes (gratuitous) reference to model
 - Note 2: The example code has a bug! Can you find it?

Extended Interactions in MVC



Extended Pattern

- Background: Observer pattern
 - One object is notified of changes in another
 - In extended MVC, view is an observer of model
- Application within MVC
 - Asynchronous model updates
 - Model changes independent of user actions
 - Associated view must be notified of change in order to know that it must update
 - A model may have multiple views
 - But a view has one model
 - All views have to be updated when model changes

Mechanics of Extended MVC

- Setup
 - Instantiate model
 - Has reference to view, initially null
 - Instantiate view with reference to model
 - View registers with model
 - Instantiate controller with references to both
 - Controller registers with view
- Execution
 - View recognizes event
 - View calls appropriate method on controller
 - Controller accesses model, possibly updating it
 - If model has been changed, it notifies all registered views
 - Views then query model for the nature of the change, rendering new information as appropriate

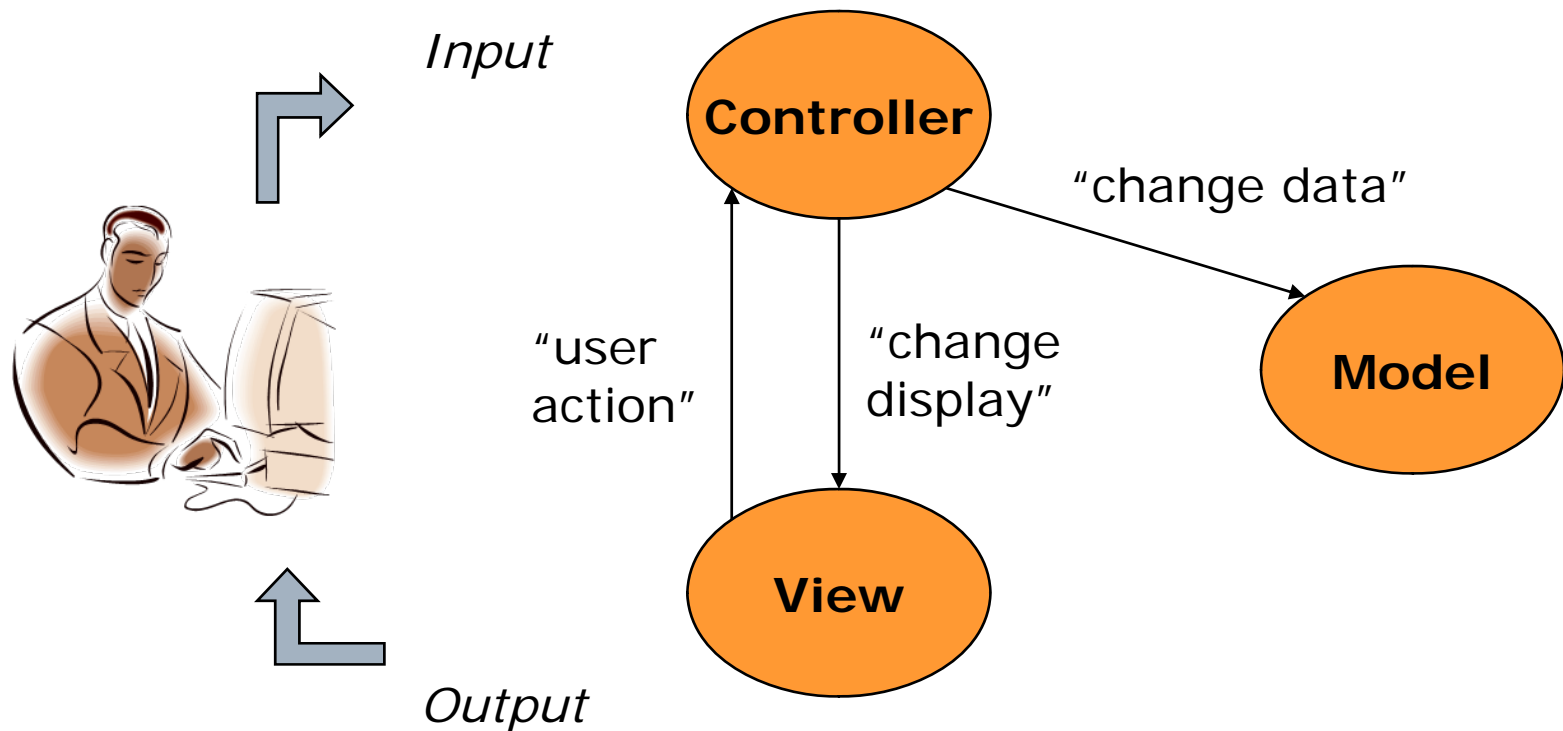
Problems with Classic MVC

- Controller might need to produce its own output
 - eg Popup menu
- Some state is shared between controller and view, but does not belong in model
 - eg Selection (highlighted text)
- Direct manipulation means that user can interact (control) visual elements (views)
 - eg Scrollbar
- Overall issue: Input and output are often intermingled in a GUI
 - Result: View and controller are tightly coupled

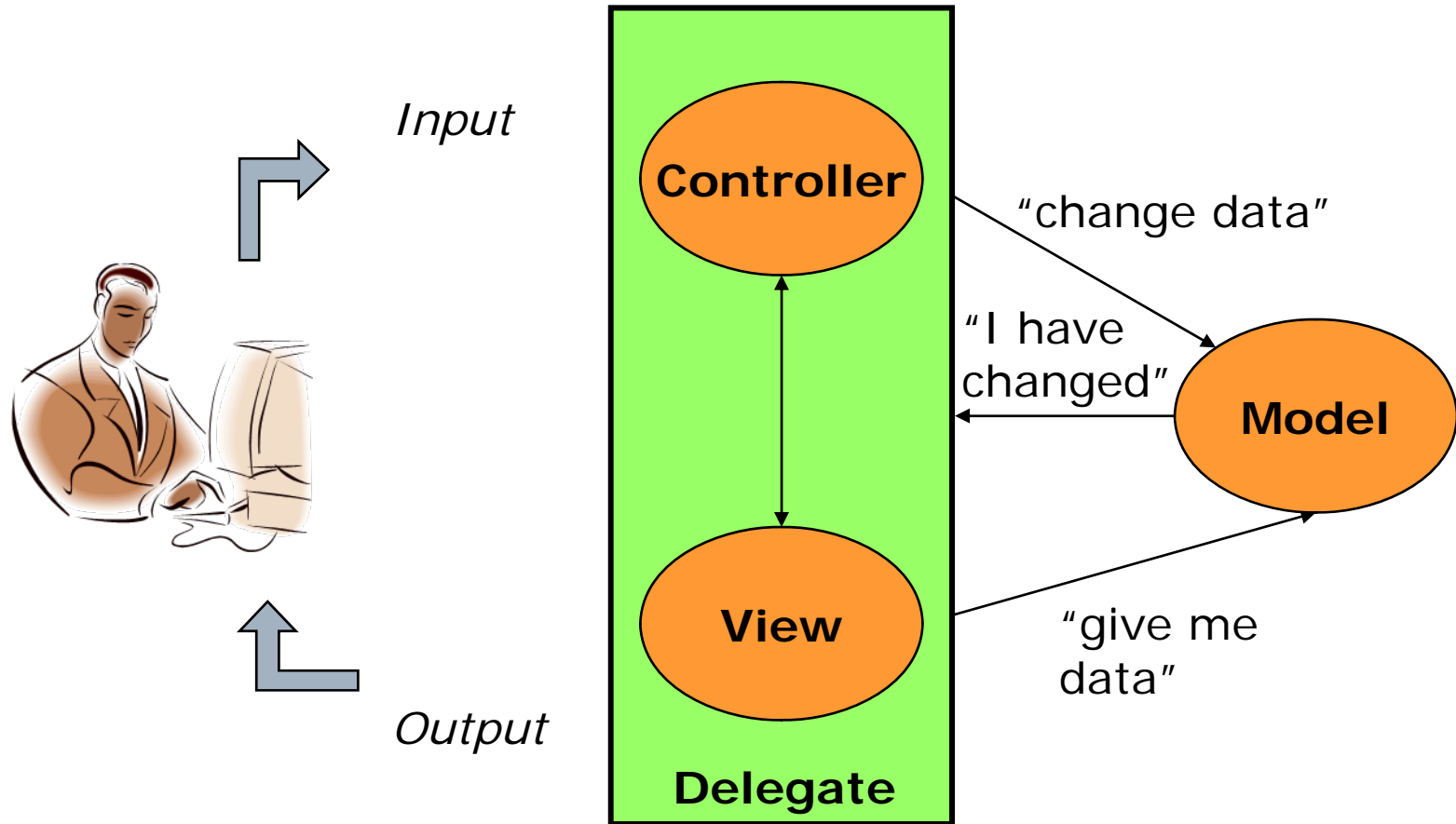
Delegate-Model Pattern

- Model
 - Data, same as before
- Delegate
 - Responsible for both input and output
 - A combination of both view and controller
- Many other names
 - UI-Model
 - Document-View

Basic Interactions in Delegate Model



Basic Interactions in Delegate Model



Mechanics of Delegate Model

- Setup
 - Instantiate model
 - As with MVC, model does not know/care about UI
 - Instantiate delegate with reference to model
- Execution
 - Delegate recognizes event and executes appropriate handler for the event
 - Delegate accesses model, possibly updating it
 - If model has been changed, UI is updated
- Example: CalcV3
 - CalcModel, CalcViewController
 - Note: CalcModel is exactly the *same* as with CalcMVC

Notes

- Litmus test: Swapping out user interface
 - Can the model be used, without modification, by a completely different UI?
 - eg Swing vs console text interface
- Model can be easily tested with JUnit
- Model actions should be quick
 - GUI is frozen while model executes
 - Alternative: multithreading, which gets much more complicated

Supplemental Reading

- Sun Developer Network
 - “Java SE Application Design with MVC”
 - <http://java.sun.com/developer/technicalArticles/javase/mvc/>
- OnJava article
 - “A Generic MVC Model in Java”
 - <http://www.onjava.com/pub/a/onjava/2004/07/07/genericmvc.html>

Summary

- Motivation: Information hiding
 - Data (state) vs user interface
 - State should be agnostic of user interface
- Model-View-Controller
 - Model contains state (data)
 - View displays model to user (presentation)
 - Controller modifies model (business logic)
- UI-Model
 - Allows for tight coupling between view and controller
 - Preserves most significant separation