

CS 349

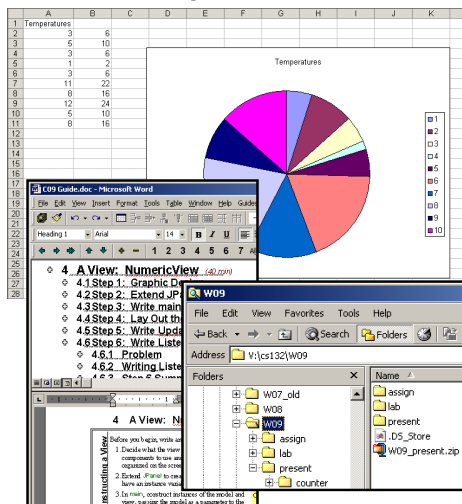
07 Model-View-Controller

Byron Weber Becker
Winter 2012



Applications with Multiple Views

- Two (or more!) views is normal.
Examples:
- MS Word
 - outline view, normal view, map
 - often at the same time
- Excel
 - table, chart
- Windows Explorer
 - folder view, file view, address



Design Considerations

- When one view changes, the other(s) should change as well.
- The user interface probably changes more and faster than the underlying application
 - Many of the changes to the most recent version of Office were to the UI
 - Excel's underlying functions and data structures are probably very similar to Visicalc, the original spreadsheet
- How do we design software to support these observations?



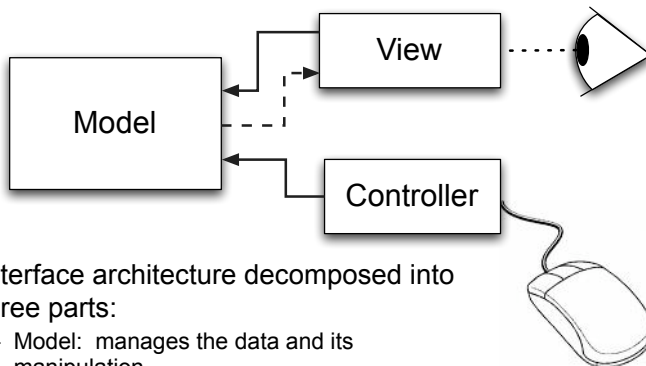
Possible Design

Spreadsheet
-Cell [] [] cells
+void setCell(int row, int col, Object data)
+Object getCell(int row, int col)
-void paintGraph(Graphics g)
-void paintTable(Graphics g)
+void paint(Graphics g)

- Issues with bundling everything together:
 - What if we want to display data from a different type of source (eg: a database)?
 - What if we want to add new ways to view the data?
- Primary Issue: Data and its presentation are tightly coupled



Solution: Model-View-Controller



- Interface architecture decomposed into three parts:
 - Model: manages the data and its manipulation
 - View: manages the presentation of the data
 - Controller: handles user interaction

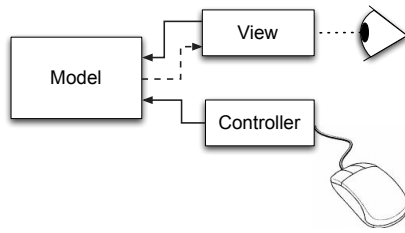


MVC background

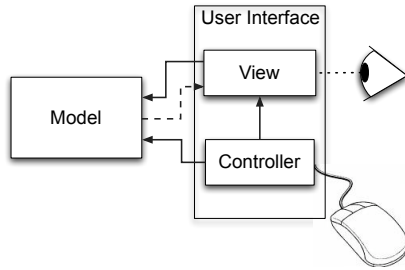
- History
 - Developed in Smalltalk-80 in 1979 by Trygve Reenskaug
 - Now the standard design pattern for graphical user interfaces.
- Used at many levels
 - Overall application design
 - Individual components
 - eg: JTable
- Will feel very familiar because the Listener model we've already examined is very similar.



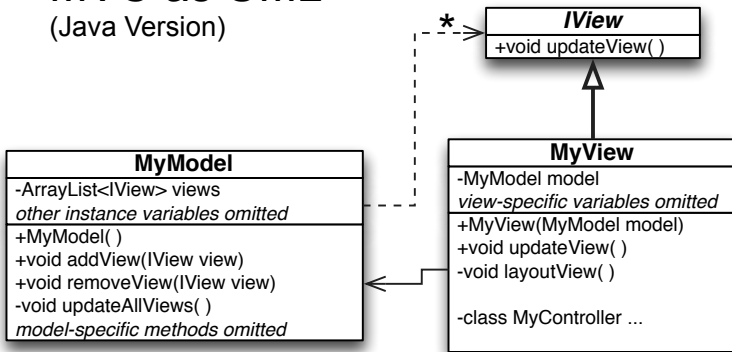
- MVC in Theory
 - View and Controller both refer to Model directly.
 - Model uses a listener-style (observer) design pattern to inform view of changes.



- MVC in Practice
 - Model is very loosely coupled with UI using the observer pattern (listeners in Java)
 - The View and Controller are tightly coupled.
 - Why?



MVC as UML (Java Version)

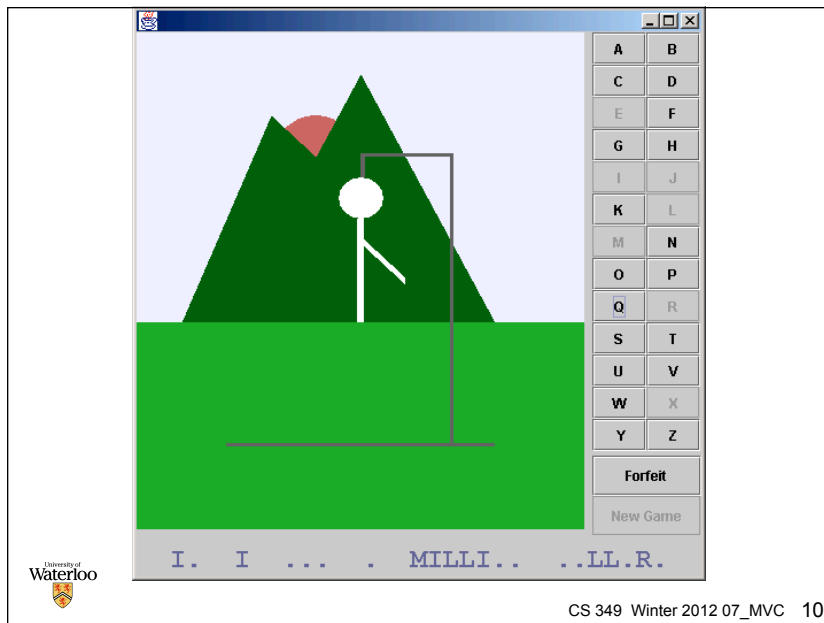


Note that MyView does not need to implement IView. It could provide an anonymous inner class to MyModel instead.

MVC Rationale 1: Multiple Views

Separation of concerns enables multiple, simultaneous views of the data.

- Given the same set of data, we may want to render it in multiple ways:
 - a table of numbers
 - a pie chart
 - a line graph
 - an audio stream
 - ...
- A separate model makes it easier for different UI components to use the same data
 - Each view is unencumbered by the details of the other views
 - Reduces dependencies on the GUI that could change



MVC Rationale 2: Alt. Interactions

Separation of concerns enables alternative forms of interactions with the same underlying data.

- Data and how it is manipulated (the model) will remain fairly constant over time.
 - Consider a stable application like...
- How we present and manipulate that data (view and controller) via the user interface will likely change more often than the underlying model.



CS 349 Winter 2012 07_MVC 11

MVC Rationale 3: Code Reuse

Separation of concerns enables programmers to more easily use a stock set of controls to manipulate their unique application data.

- Example: JTable
 - Because the model is separated out, it can be used to manipulate many kinds of data stored in many different ways.
 - More time and attention can be given to JTable itself to make it more robust and versatile.



CS 349 Winter 2012 07_MVC 12

MVC Rationale 4: Testing

Separation of concerns enables one to more easily develop and test data-specific manipulations that are independent of the user interface



Process

- Set up the infrastructure
 - Write three classes:
 - the model
 - one or more view/controller classes (extends JComponent or JPanel)
 - a class containing the main method
 - In the main method:
 - create an instance of the model
 - create instances of the views/controllers, adding them the model
 - display the view(s) in a frame

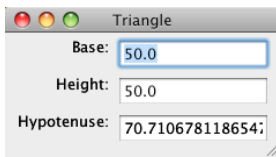


Process (cont.)

- Build and test the model
 - Design, implement, and test the model
 - add commands used by controllers to change the model
 - add queries used by the view to update the display
 - Call `updateAllViews` just before exiting any public method that changes the model
- Build the Views and Controllers
 - Design the user interface as one or more views. For each view:
 - Construct components
 - Lay the components out in the view
 - Write and register appropriate controllers for each component
 - Write `updateView` to get and display info from the model; register it with the model.



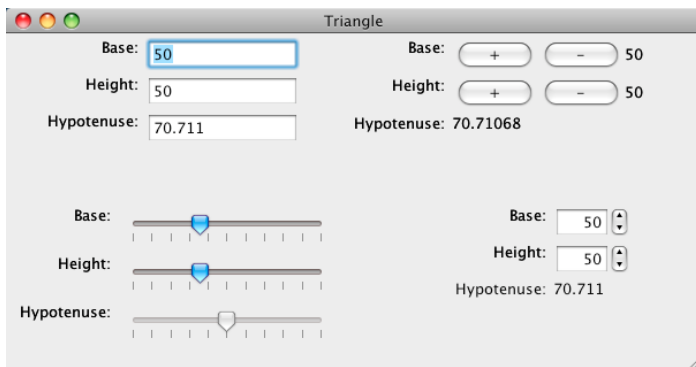
Demo: Triangles



- Demo the program
- Examine the model
- Examine SimpleTextView; discuss its deficiencies
- Examine TextView

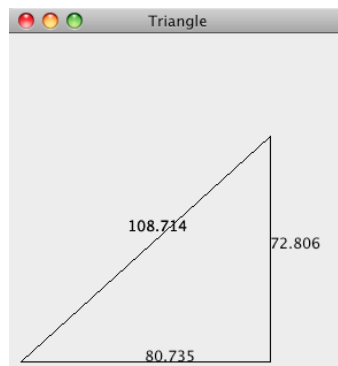


Multiple Views



A Graphical View

- Click the right side to select
- Once selected:
 - show “handles” for dragging
 - drag the right side to change the base
 - drag the apex to change both the base and the height
 - adjust cursor when over the right side or the apex



• Topic of future lecture