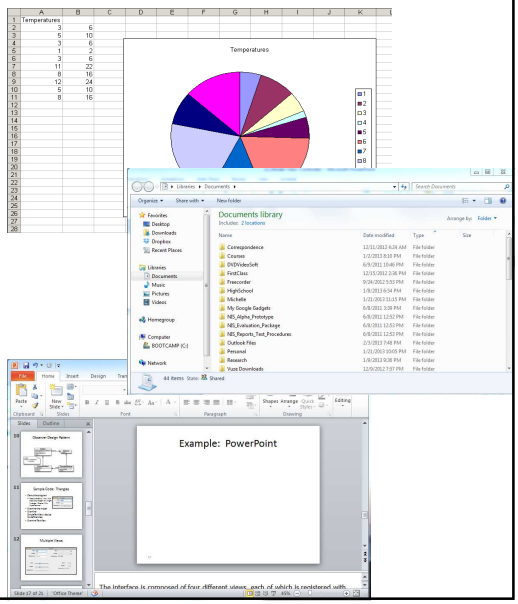
# Model-View-Controller

1

# 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




2

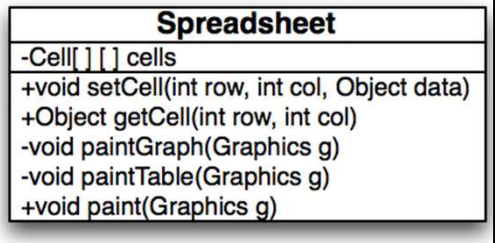
# Design Considerations

- 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?

3

# Possible Design

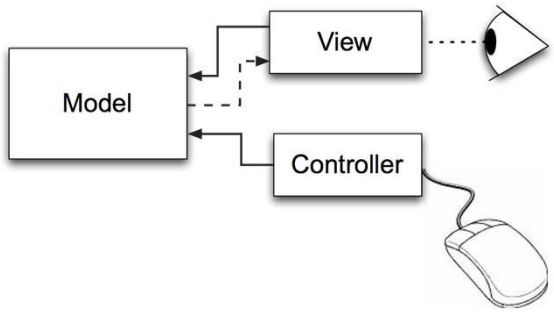
- Issues with bundling everything together:
  - What if we want to display data from a different type of source (eg: a database)?
  - What if we want to add new ways to view the data?
- Primary Issue: Data and its presentation are tightly coupled




4

# 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: manages user interaction



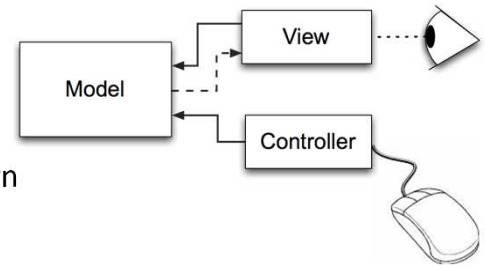

5

# MVC background

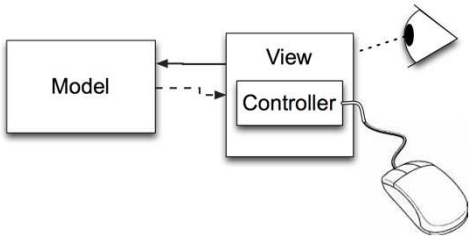
- History
  - Developed in Smalltalk-80 in 1979 by Trygve Reenskaug
  - Now the standard design pattern for graphical user interfaces.
- Used at many levels
  - Overall application design
  - Individual components
    - eg: JTable

6

- MVC in Theory
  - View and Controller both refer to Model directly
  - Model uses the observer design pattern to inform view of changes



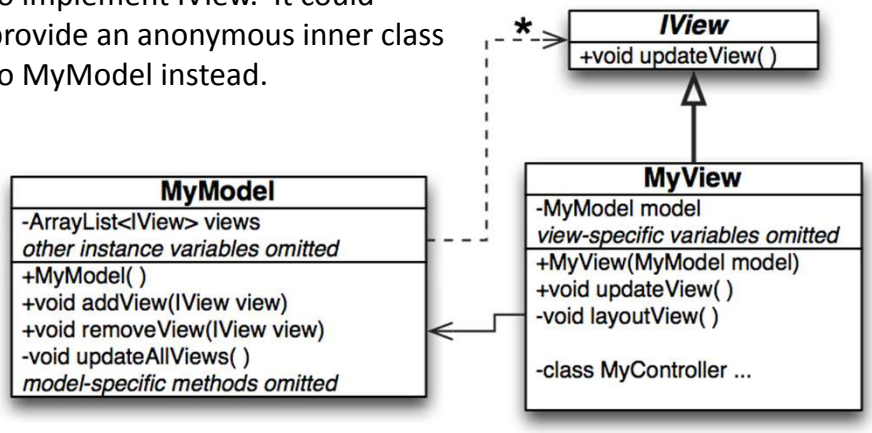

- MVC in Practice
  - Model is very loosely coupled with UI using the observer pattern
  - The View and Controller are tightly coupled
  - Why?




7

# MVC as UML (Java Version)

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



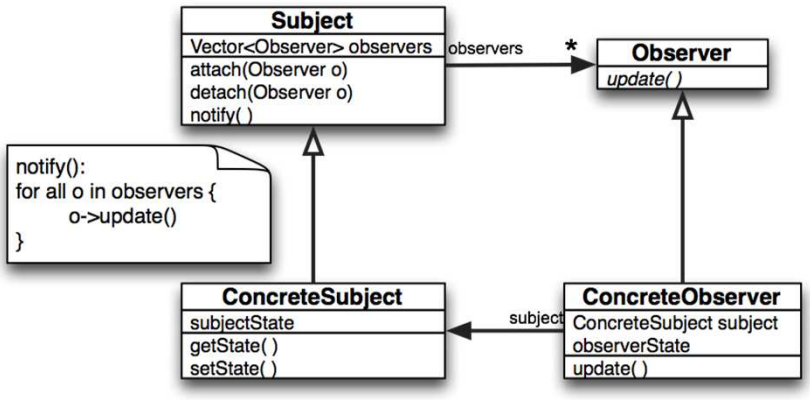

8

# Observer Design Pattern

- MVC is an instance of the Observer design pattern
- Provides a well-defined mechanism that allows objects to communicate without knowing each others' specific types
  - Promotes loose coupling
- AKA "listener" and "publish-subscribe"
  - Delegates in C#
- Examples in Java
  - ActionListener, PropertyChangeListener, WindowListener…

9

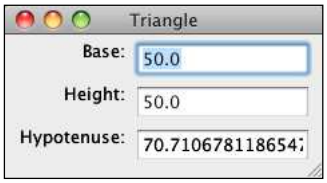
# Observer Design Pattern




10

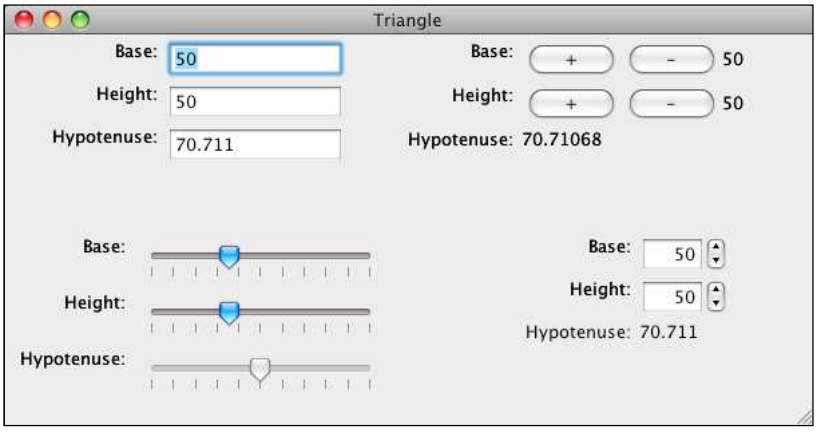
# Sample Code: Triangles

- Demo the program
  - Requirements: Vary the base and height of a right triangle. Display the hypotenuse.
- Examine the model
- Examine SimpleTextView; discuss its deficiencies
- Examine TextView




11

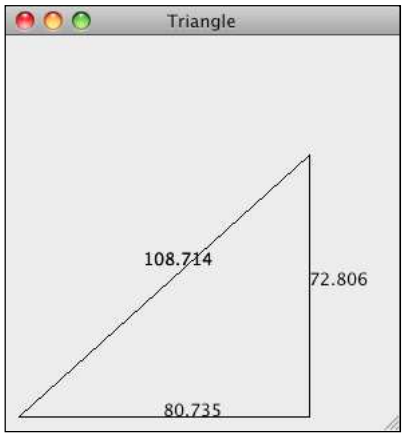
# Multiple Views




12

# 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
- Similar to A02




13

# 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 to them the model
    - display the view(s) in a frame

14

# 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's data
- Build the Views and Controllers
  - Design the UI 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.

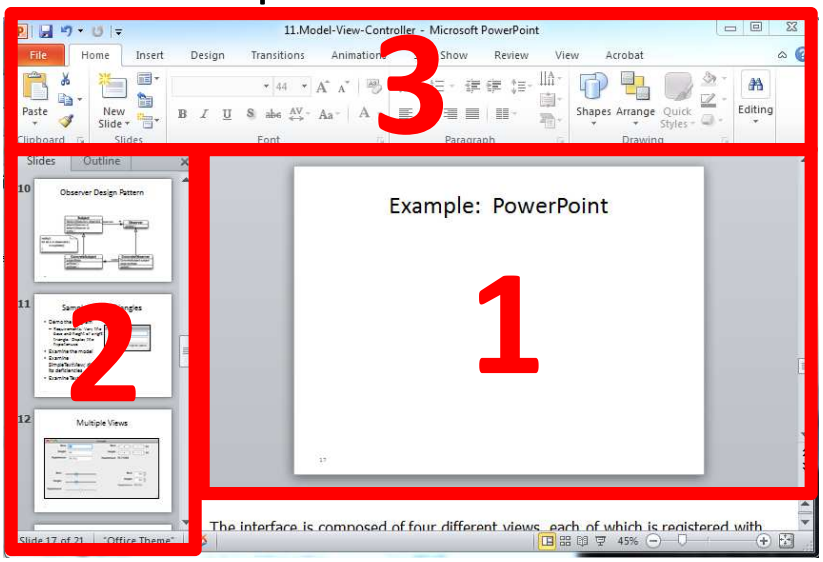
15

# 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

16

# Example: PowerPoint




17

# 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.

18

# 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.

19

# 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

20

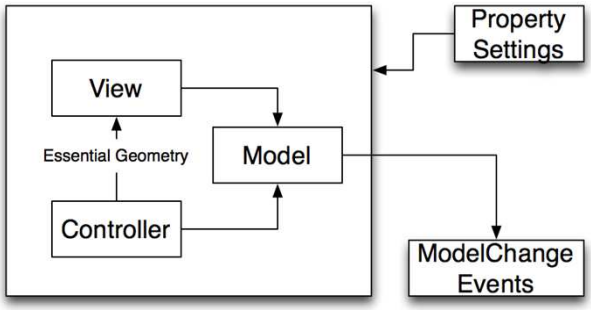
# MVC and Java

- Observers: Java's Listeners are a more complex version of the Observer pattern
  - Used on a smaller model: a component
  - Multiple "update" methods
  - "Update" methods have parameters
- Controllers:
  - Implemented in Java using inner classes
  - Generally an instance of a Listener
- MVC can occur at multiple granularities
  - And does in Java!

21

# Simple Widgets

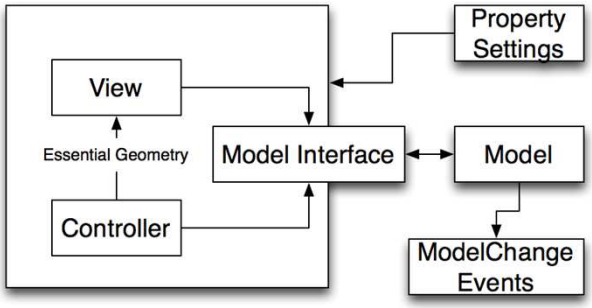
- Modern widget toolkits use MVC throughout
- Simple widgets usually contain a default model within themselves
- Examples: buttons, checkboxes, scrollbars, ...




22

# Complex Widgets

- More complex widgets expect the application to implement model interface or extend an abstract class
- Examples: table and tree widgets




23

# Table Model

```
public interface TableModel {
int getColumnCount();
String getColumnName(int columnIndex);
Class<?> getColumnClass(int columnIndex);

int getRowCount();

Object getValueAt(int rowIndex, int columnIndex);
void setValueAt(Object aValue, int rowIndex, int columnIndex);
boolean isCellEditable(int rowIndex, int columnIndex);

void addTableModelListener(TableModelListener l);
void removeTableModelListener(TableModelListener l);
}
```

24

# Table Model Listener

```
public interface TableModelListener {
void tableChanged(TableModelEvent e);
}

public class TableModelEvent {
TableModelEvent(TableModel source) { ... }
TableModelEvent(TableModel source, int row)  { ... }
TableModelEvent(TableModel source, int firstRow, int lastRow)  { ... }
TableModelEvent(TableModel source, int firstRow, int lastRow, int column)  { ... }
TableModelEvent(TableModel source, int firstRow, int lastRow,
        int column, int type)  { ... }

int getColumn() { ... }
int getFirstRow() { ... }
int getLastRow() { ... }
int getType() { ... }  // one of INSERT, UPDATE, DELETE
}
```

25

# Customizing JTable

- JTable has a TableColumnModel with information about each column in the table.
- The TableColumnModel has methods like:
  - void addColumn(TableColumn aColumn)
  - TableColumn getColumn(int columnIndex)
  - int getColumnCount()
  - int[ ] getSelectedColumns()
  - void moveColumn(int columnIndex, int newIndex)
- TableColumnModel is actually an interface;  there is a DefaultTableColumnModel that implements the standard column-handler.  It chooses appropriate TableColumn instances based on the type of data returned by getColumnClass() in the TableModel.

26

# Customizing JTable

- The TableColumn has information about a single column.
- It has methods like:
  - void setCellRenderer(TableCellRenderer cellRenderer)
  - void setCellEditor(TableCellEditor cellEditor)
  - void setPreferredWidth(int preferredWidth)
  - void setMinWidth(int minWidth)
  - void setResizable(boolean isResizable)
  - void setHeaderRenderer(TableCellRenderer headerRenderer)

27