MVC and Swing

IAT 351

Week 7 Lecture/tutorial

22.02.2008

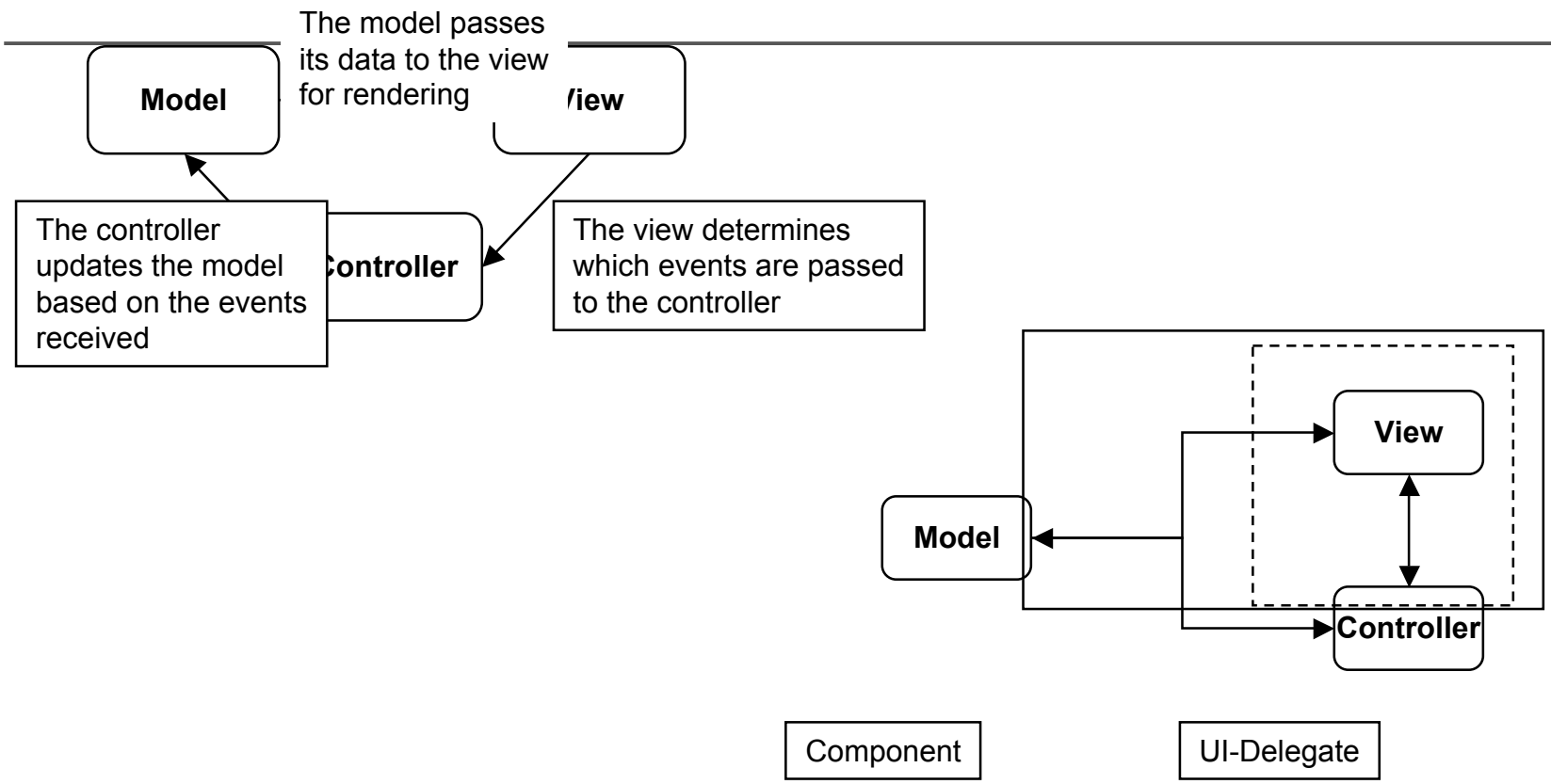
Lyn Bartram

SCHOOL OF INTERACTIVE
ARTS + TECHNOLOGY

Assignment 3

- Goal: implement different interaction techniques for the same interactive tasks in 2 different contexts
  - Regular desktop
  - Small mobile device
- Use model-view-controller architecture
- Have some FUN designing the interactions
- Due next week

The Model-View-Controller Architecture



With Swing, the view and the controller are combined into a UI-Delegate object

Tables

- Tables are used to display data in a spreadsheet fashion
- The `JTable` is oriented toward displaying database records in which each row displays a row in the database, and each column displays a different record's values for the same field
- So key concepts in the model of a table:
  - Cell
  - Row, column
  - Value(s) in each

Class JTable

- JTable component presents data in a 2D table format
- The JTable has many features that make it possible to customize its rendering and editing but provides defaults for these features.
- A JTable consists of:
  - Rows of data
  - Columns of data
  - Column headers
  - An editor, if you want cells to be editable

Class JTable

- A `JTable` consists of:
  - A `TableModel`, usually a subclass of `AbstractTableModel`, which stores the table's data
  - A `TableColumnModel`, usually `DefaultTableColumnModel`, which controls the behavior of the table's columns and gives access to the `TableColumn`s
  - A `ListSelectionModel`, usually `DefaultListSelectionModel`, which keeps track of the `JTable`'s currently selected row(s)
  - A `TableCellRenderer`, usually an instance of `DefaultCellRenderer`
  - Multiple `TableColumn`s, which store graphical information about each column
  - A `JTableHeader` which displays column headers

Class `JTable`

- Steps in creating and using `JTable`
  1. Create a `JTable` (there are 7 different constructors)
  2. Create a `JScrollPane` that can be used to scroll around the `JTable` via `createScrollPaneForTable()`
  3. Place the `JTable` within a container
  4. Control whether grid lines should be drawn via `setShowGrid()`
  5. Specify a default value for a cell via `setValueAt()`
  6. Get the value for a cell via `getValueAt()`
  7. Make individual cells selectable via `setCellSelectionEnabled()`
  8. Find out which cells are selected via the `JTable`'s `ListSelectionModel` and the `TableColumnModel`'s `ListSelectionModel`
  9. Add new rows and columns via the `JTable`'s `TableModel`

Class

`AbstractTableModel`

- `AbstractTableModel` is an *abstract class* that implements most of the `TableModel` interface
- The `TableModel` methods that are not implemented are `getRowCount()` , `getColumnCount()` , and `getValueAt()`
- Steps in creating and using `AbstractTableModel`
  - Create an `AbstractTableModel` subclass
  - Implement the `getRowCount()` , `getColumnCount()` , and `getValueAt()` methods
  - Instantiate an instance of the subclass
  - Create a `JTable` using the subclass via `new JTable( model )`

Class

AbstractTableModel

- To set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel()
{
  public int    getColumnCount() { return 10; }
  public int    getRowCount()    { return 10;}
  public Object getValueAt(int row, int col)
  { return new Integer(row*col); }
};
JTable table = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(table);
```

Class

`DefaultTableModel`

- `DefaultTableModel` is the JFC's default subclass of the abstract `AbstractTableModel` class
- If a `JTable` is created and no `TableModel` is specified, the `JTable` creates an instance of `DefaultJTableModel` and uses it to hold the table's data
- If you have complex data, you may prefer to extend the `AbstractTableModel` yourself
- Steps in creating and using `DefaultTableModel`
  - Create a `DefaultTableModel` (there are 6 different constructors)

    ```
    DefaultTableModel( Vector data, Vector columnIDs)
    ```
  - Create a `JTable` using the `DefaultTableModel` via `new JTable(model)`

Class

`DefaultTableModel`

- Steps in creating and using `DefaultTableModel`
  - Define a `TableModelListener` to receive `TableModelEvent`s when the model changes, or when one or more cell's contents change
  - Add a row to the `DefaultTableModel` via `addRow()`
  - Add a column to the `DefaultTableModel` via `addColumn()`
  - Get the current value of a cell in a `DefaultTableModel` via `getValueAt()`
  - Move one or more rows via `moveRow()`
  - Load a new set of data into a `DefaultTableModel` via `setDataVector()`
  - Get the number of rows or columns in a `DefaultTableModel` via `getRowCount()` and `getColumnCount()`

Class

`TableColumn`

- A `TableColumn` contains the graphical attributes for a single column of data in a `JTable`'s model
- It stores information about the column header, the column height and width, and how cells in the column should be drawn and edited
- Steps in creating and using `TableColumn`
  - `TableColumn`s are created automatically when columns are added to the table model. They are accessed via the table column model via `getColumn()`
  - Specify the `TableCellEditor` to use when editing the `TableColumn`'s cells

```
JCheckBox cbox = new JCheckBox()
DefaultCellEditor editor = new DefaultCellEditor(cbox)
tableColumn.setCellEditor(editor)
```

  - Change the column header via `setHeaderValue()`

Class

`DefaultTableColumnModel`

- `DefaultTableColumnModel` is the JFC's default implementation of the `TableColumnModel` interface
- This class is used to keep track of information about table columns. It gives access to `TableColumn`s and keeps track of general characteristics of columns, like column margins and widths. It also contains a `ListSelectionModel` that it uses to keep track of which columns are currently selected
- Steps in creating and using `DefaultTableColumnModel`
  - You will usually let the `JTable` create it
  - Specify the selection mode for the `DefaultTableColumnModel` via `setSelectionMode()`

Class

`DefaultTableColumnModel`

- Steps in creating and using `DefaultTableColumnModel`
  - Define a `ColumnModelListener` to receive `TableColumnModelEvent`s when a column is added, removed, moved, margins are changed, or the column selection state changes
  - Get the currently selected columns via `getSelectedColumns()`

Class

`JTableHeader`

- A `JTableHeader` is a companion to `JTable` and contains the graphical representation of the table's column headers
- A `JTableHeader` does not display by default but will display if you place a `JTable` into a `JScrollPane` created using the `createScrollPaneForTable()` method
- A `JTableHeader` draws itself using information from the `TableColumnModel` associated with the `JTable`
- Steps in creating and using `JTableHeader`
  - You will usually let the `JTable` create it
  - Change the `TableCellRenderer` used to draw a column's header via `setHeaderRenderer()`
  - Enable/Disable column reordering via `setReorderingAllowed()`

Class

`DefaultCellEditor`

- `DefaultCellEditor` is an editor that can be used with a `JTable` or a `JTree` to edit table cells and tree nodes
- It can edit in one of three ways: as a text field, as a check box, or as a combo box
- Steps in creating and using a `DefaultCellEditor`
  - Create a component to be used by the `DefaultCellEditor` and set its properties
  - Create a `DefaultCellEditor` using the component you just created
  - Specify how many mouse clicks it takes to start the editor via `setClickCountToStart()` (default is 2)
  - Define a `CellEditorListener` to receive `ChangeEvent`s when a cell editing session ends

JTable

`TableModel` interface methods and descriptions.

Method	Description
`void addTableModelListener( TableModelListener listener )`	Add a **`TableModelListener`** to the **`TableModel`**. The **`TableModel`** will notify the **`TableModelListener`** of changes in the **`TableModel`**.
`void removeTableModelListener ( TableModelListener listener )`	Remove a previously added **`TableModelListener`** from the **`TableModel`**.
`Class getColumnClass( int columnIndex )`	Get the **`Class`** object for values in the column with specified **`columnIndex`**.
`int getColumnCount()`	Get the number of columns in the **`TableModel`**.
`String getColumnName( int columnIndex )`	Get the name of the column with the given **`columnIndex`**.
`int getRowCount()`	Get the number of rows in the **`TableModel`**.

JTable

`TableModel` interface methods and descriptions.

`Object getValueAt( int rowIndex, int columnIndex )`	Get an **`Object`** reference to the value stored in the **`TableModel`** at the given row and column indices.
`void` **`setValueAt( Object value, int rowIndex, int columnIndex )`**	Set the value stored in the **`TableModel`** at the given row and column indices.
`boolean isCellEditable( int rowIndex, int columnIndex )`	Return **`true`** if the cell at the given row and column indices is editable.

- Java Documentation
- http://java.sun.com/j2se/1.5.0/docs/api/

- Look at Jtable

- How to use tables in Swing