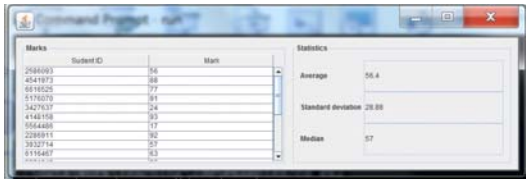
Model/View Applications in Swing

Some slides are based on Dr Ian Warren's slides



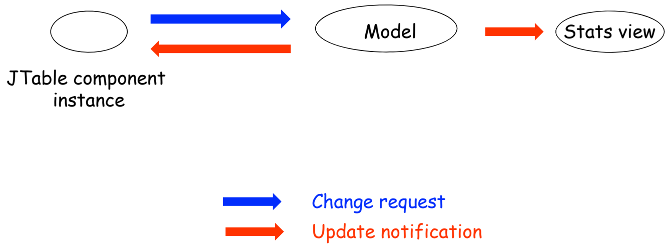
Marks

Student ID	Mark
2586093	56
4541973	88
6818525	77
5176070	81
3427637	24
4148158	93
5564486	17
2286911	92
3932714	57
6116467	63

Statistics

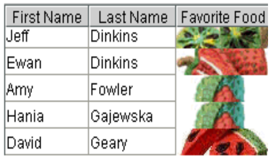
Average	56.4
Standard deviation	28.88
Median	57

- Many applications provide multiple views of the same set of data.
- All the views should be derived from the same set of data.
  - The change to the data should be reflected by all the views.
- The data that are used to derived the views are called the data model.
- The presentation of the data is called the view.

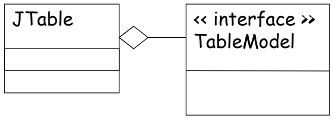


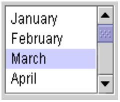
Model/view GUIs with Swing

- Contemporary GUI frameworks, like Swing, are based on a separable model architecture
- In Swing all components (JComponents) have separate models to represent the data to be shown by the components

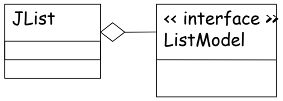


First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	

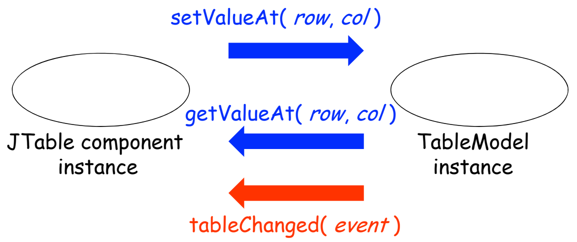




January
February
March
April



- The data model provides an interface for the Swing component to retrieve/update the data in the data model
- The data model's interface also allows the views to register their interest with the model to allow the views to be notified of the changes to the data model



Implementing TableModel

- TableModel is an interface defined in Java.
- AbstractTableModel provides a partial implementation of the TableModel interface.
- In your implementation, you should implement your own data model by extending the AbstractTableModel

```
public class MyTableModel extends AbstractTableModel
```

Storing data in the model

- In MyTableModel, you should define data structures for holding the data to be presented by the JTable view
- In the code below, records holds the data; colNames stores the names of the columns to be shown in the table.
  - It's natural to use a two dimensional array to hold the values in a table. But, you can use other type of structure as well.

```
String[][] records;
String[] colNames;

public MyTableModel(String[][] records, String[] colNames) {
        this.records = records;
        this.colNames = colNames;
}
```

- AbstractTableModel defined some methods
  - Through these methods, the view can interact with the model
- getColumnCount
  - Returns the number of columns in the model.
  - A JTable uses this method to determine how many columns it should create and display by default.

```
@Override
public int getColumnCount() {
        return colNames.length;
}
```

- getRowCount
  - Returns the number of rows in the model.
  - A JTable uses this method to determine how many rows it should display.

```
@Override
public int getRowCount() {
        // assume there are 20 rows
        return 20;
}
```

- getValueAt(int rowIndex, int columnIndex)
  - Returns the value for the cell at columnIndex and rowIndex.
  - It is called by the runtime to retrieve value to fill the table

```
@Override
public Object getValueAt(int row, int col) {
        return records[row][col];
}
```

- setValueAt(Object aValue, int rowIndex, int columnIndex)
  - This method sets the value of the cell at the specified row and column
  - If you allow the data to be modified through the table (i.e. by editing a cell in the table), the runtime calls this method after user modifies a cell’s value

```
@Override
public void setValueAt(Object value, int row, int col) {
        records[row][col] = (String)value;
        …
}
```

- getColumnName(int column)
  - If you want to display names for the columns in the table, you should override this method.
  - Otherwise, the column names would be A, B, C, … Z, AA, AB, etc (i.e. the default name for a spread sheet)
  - This method is called by the runtime when the table is shown.

```
@Override
public String getColumnName(int column) {
        return colNames[column];
}
```

- isCellEditable(int rowIndex, int columnIndex)
  - This method returns a Boolean value
  - True means the cell at the given row and column can be modified. Otherwise, the cell cannot be modified.
  - The runtime calls this method to determine whether a cell in the table can be modified.

```
@Override
public boolean isCellEditable(int row, int col) {
        if (col == 1)  // only the second column can be modified
                return true;
        else
                return false;
}
```

- In an application, multiple parties might be interested in knowing the changes to the data model.
  - These parties would register as listeners of the data model.
  - When the data model changes, the listeners need to be notified.
- Writing code to notify the changes to the listeners is tedious.
- The AbstractTableModel implements a set of fireXXX methods.
  - You should use these methods in your program.
- These methods are used to notify the listeners who want to know the data in the data model have been changed.
  - As a result of the notification, the listeners will modify their presentation of data.

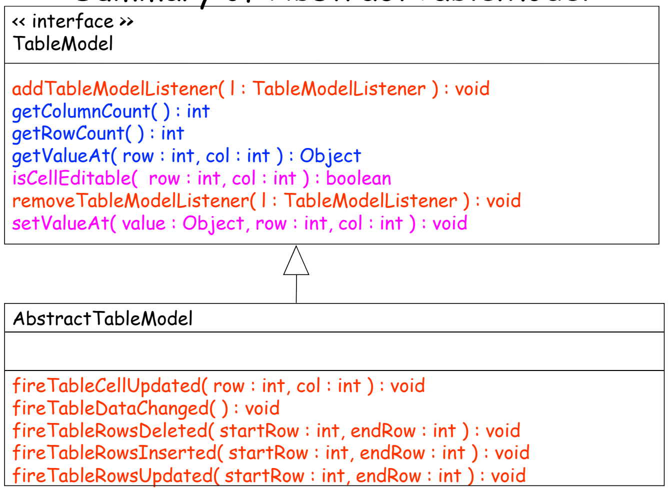
- fireTableCellUpdated(int row, int column)
  - Notifies all listeners that the value of the cell at [row, column] has been updated.
- fireTableDataChanged()
- fireTableRowsUpdated(int firstRow, int lastRow)
- fireTableRowsDeleted(int firstRow, int lastRow)
- fireTableRowsInserted(int firstRow, int lastRow)

```
@Override
public void setValueAt(Object value, int row, int col) {
        records[row][col] = (String)value;
        fireTableCellUpdated(row, col);
}
```

- addTableModelListener(TableModelListener l)
  - Adds a listener to the list that's notified each time a change to the data model occurs.

```
myTableModel = new MyTableModel(records, colNames);
myTableModel. addTableModelListener(this);
```

Summary of AbstractTableModel



« interface »
TableModel

addTableModelListener(l : TableModelListener) : void
getColumnCount( ) : int
getRowCount( ) : int
getValueAt(row : int, col : int) : Object
isCellEditable(row : int, col : int) : boolean
removeTableModelListener(l : TableModelListener) : void
setValueAt(value : Object, row : int, col : int) : void

AbstractTableModel

fireTableCellUpdated(row : int, col : int) : void
fireTableDataChanged( ) : void
fireTableRowsDeleted(startRow : int, endRow : int) : void
fireTableRowsInserted(startRow : int, endRow : int) : void
fireTableRowsUpdated(startRow : int, endRow : int) : void

Using JTable

- JTable is useful for displaying, navigating, and editing tabular data.
  - It is the view of the data.
  - It is a very complicated class. We will only see a tiny portion of the features.
- When you create a JTable, you can associate a data model to the table.

```
JTable tableView = new JTable(tableModel);
```

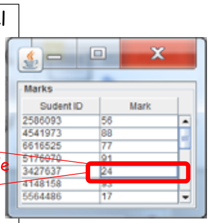
```
JTable tableView = new JTable();
tableView.setModel(tableModel);
```

- When you associate a data model with a JTable, the table is automatically added as a listener of the data model.
  - If the data is changed, the cells in the table will also be changed accordingly.
- In the data model, if you have specified a cell as editable, you can enter the value of the cell in the table. The new value will be propagated to the data model.
  - If the cell at [row, col] is editable, i.e. isCellEditable(row, col) returns true, you can change the value of the cell at [row, col] in the table.
  - The runtime calls the setValueAt method of the data model to make the change to the data model.

```
public class MyTableModel extends AbstractTableModel

...
@Override
public boolean isCellEditable(int row, int col) {
        if (col == 1) return true;
        else return false;
}

@Override
public void setValueAt(Object value, int row, int col) {
        records[row][col] = (String)value;
        fireTableCellUpdated(row, col);
}
```

1. check whether selectable

2. change the value in the model



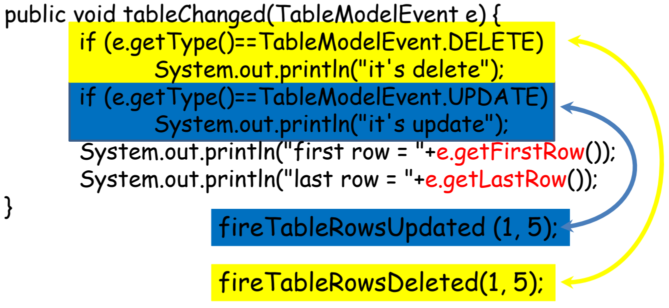
Marks

Sudent ID	Mark
2586093	56
4541973	88
6616525	77
5176070	91
3427637	24
4148158	93
5564486	17

```
// Make each cell selectable.
tableView.setRowSelectionAllowed(false);
```

TableModelListener

- TableModelListener is an interface which should be implemented by all the classes that want to be notified of the changes to the data model.
  - The interface has a tableChanged method which specifies the operations that the listeners want to carry out once being notified of the changes to the model.
- Object needs to register with a table model using the model's addTableModelListener method.
- The table model will notify the listeners of the changes to the model by calling one of the fireXXX methods.
- Once the listener is notified, the listener executes its tableChanged method.

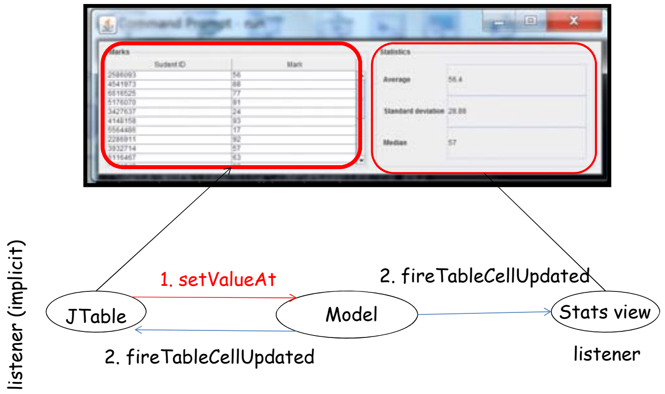


```
public class MyTableModelListener implements TableModelListener


@Override
public void tableChanged(TableModelEvent e) {
        if (e.getType()==TableModelEvent.DELETE)
                System.out.println("it's delete");
        if (e.getType()==TableModelEvent.UPDATE)
                System.out.println("it's update");
        System.out.println("first row = "+e.getFirstRow());
        System.out.println("last row = "+e.getLastRow());
}
                        fireTableRowsUpdated (1, 5);

                        fireTableRowsDeleted(1, 5);
```

Writing a JTable application

- Write the code for implementing the data model
  - Subclass the AbstractTableModel
  - Must implement getColumnCount, getRowCount, getValueAt
  - Implement isCellEditable and setValueAt if you want to update the model through the table
  - Call the fireXXX methods when notifying the listeners for changes to the data
- Write the code for implementing the listeners of the data model
  - Implement the TableModelListener interface
  - Implement the tableChanged method

- Write the code that uses JTable
  - Create a data model object
  - Create a JTable object
  - Associate the data model object with the JTable object
  - Create listeners of the data model (if needed)
  - Register the listeners with the model
  - Layout the application
    - JTable is normally placed in a JScrollPane



Marks

Sudent ID	Mark
2586093	56
4541873	88
6616525	77
5176070	81
3427637	24
4146158	93
5564486	17
2286911	92
3932714	57
6116467	63

Statistics

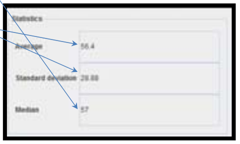
Average 56.4

Standard deviation 28.88

Median 57

```
public class MyTableModel extends AbstractTableModel
String[][] records;
String[] colNames;

public MyTableModel(String[][] records, String[] colNames)
public String[][] getData()
public String getColumnName(int column)
public int getColumnCount()
public int getRowCount()
public Object getValueAt(int row, int col)
public boolean isCellEditable(int row, int col)
public void setValueAt(Object value, int row, int col) {
        …
        fireTableCellUpdated(row, col);
}
```

```
public class StatisticsPanel extends JPanel implements TableModelListener
private JTextField averageField, stdDevField, medianField;
private MyTableModel tabMod;

public StatisticsPanel(MyTableModel tabMod)

public void paintComponent(Graphics g) {
        super.paintComponent(g);
        String[][] records = tabMod.getData();
        double stdDev = getStdDev(records);
        double average = getAverage(records);
        int median = getMedian(records);

        averageField.setText(formatNumber(average));
        stdDevField.setText(formatNumber(stdDev));
        medianField.setText(formatNumber(median));
}

public void tableChanged(TableModelEvent e) {
        repaint();
}
```



Statistics

Average 56.4

Standard deviation 28.88

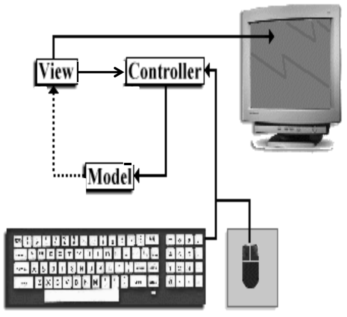
Median 57

calls

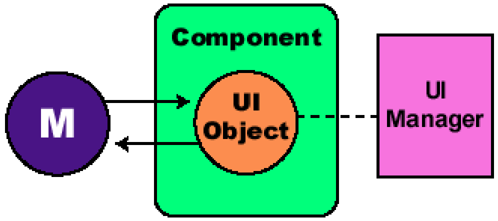
```
public class Main extends JFrame

private MyTableModel tableModel;
private JTable tableView;
private String[] columnNames = { "Sudent ID", "Mark" };
private String[][] records;

public Main() {
        … // code for reading data from a file to populate records
        // create a model
        tableModel = new MyTableModel(records, columnNames);
        // create a table and associate the model with the table
        tableView = new JTable(tableModel);
        // create a model listener
        StatisticsPanel statsView = new StatisticsPanel(tableModel);
        // register the listener with the model
        tableModel.addTableModelListener(statsView);
        setupPane(tableView, statsView);
}
```

Model-View-Controller architecture (MVC)

- The Model-View-Controller architecture (MVC) decomposes a user-interface design into three separate parts.
- A model that represents the data for the application.
- The view that is the visual representation of that data.
- A controller that takes user input on the view and translates that to changes in the model.



- In practice, the view and controller require a tight coupling
  - Some of the components, e.g. a table cell, allow you enter data while others, e.g. button, don't.
  - So, you must know the specifics of the view in order to make the controller work
- Java Swing collapsed the view and control to a user-interface component.
  - A model and a UI component
  - Called separable model architecture

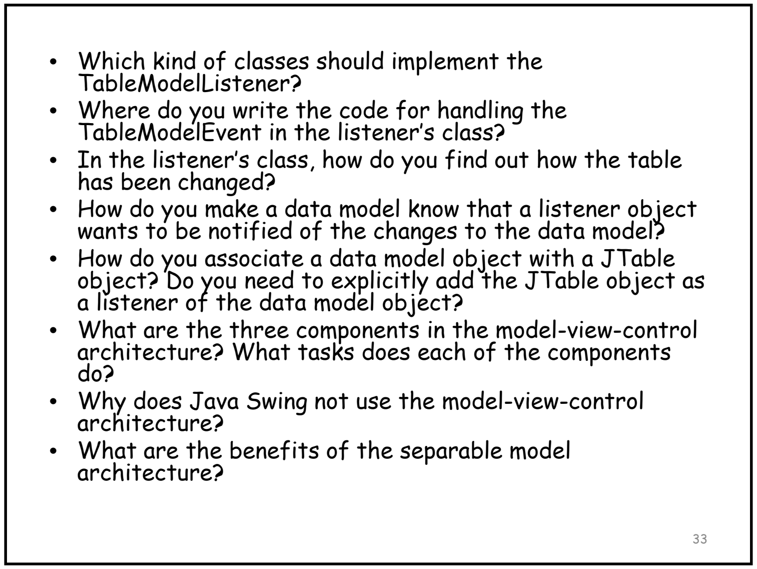


benefits of the separable model arch.

- Consistency of the views
  - All registered listeners (views) of a model are **guaranteed** to be notified when the model changes its state
- Flexibility
  - Views can easily be added to (and removed from) a model at run-time
- Maintainability
  - It's very easy to add new views
    - All that's required is to write a new implementation of the listener interface
    - Existing model and view classes need not be changed
  - Changes to the code implementing the model does not affect the code implementing view and vice versa
    - The views obtain/modify the data through the methods defined by the interface which the model must implements

reviews

- Understand the methods defined or inherited in AbstractTableModel.
- When you implement your data model as a subclass of AbstractTableModel, which methods in AbstractTableModel must be implemented? Which methods should be implemented if you want to update the data through the table that your model is associated with?
- In your implementation of the data model, can you implement other methods apart from the ones defined in AbstractTableModel?
- How does a model notify its listeners that the data in the model have changed?
- In your implementation, where do you place the code that make the model notify the listeners of the changes to the data?



- Which kind of classes should implement the TableModelListener?
- Where do you write the code for handling the TableModelEvent in the listener's class?
- In the listener's class, how do you find out how the table has been changed?
- How do you make a data model know that a listener object wants to be notified of the changes to the data model?
- How do you associate a data model object with a JTable object? Do you need to explicitly add the JTable object as a listener of the data model object?
- What are the three components in the model-view-control architecture? What tasks does each of the components do?
- Why does Java Swing not use the model-view-control architecture?
- What are the benefits of the separable model architecture?