Distributed Programming in Java

Session: 6

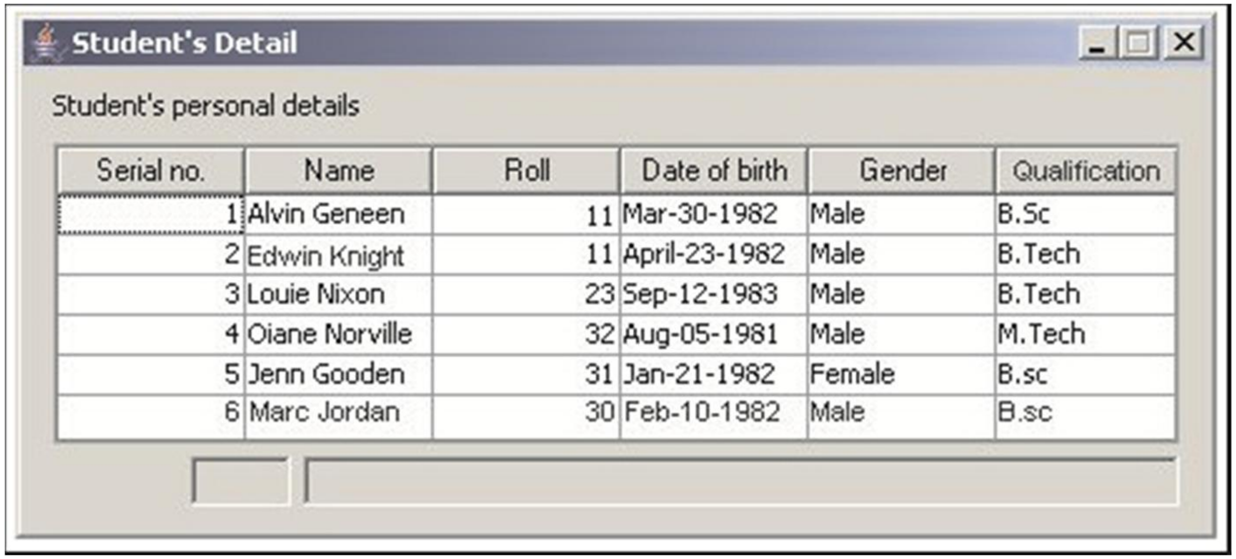
Swing Advanced Components

Objectives

- Explain JTable
- Explain how to create a JTable with a custom data model
- Explain event handing of JTable
- Explain JTree
- Explain how to add, configure, and display a JTree
- Explain event handing of JTree
- Explain concept of Drag and Drop
- Explain TransferHandler class
- Explain how to import and export data

JTable

- A `JTable` is a component which displays data in a two dimensional format.
- A `JTable` is similar to a spread-sheet in appearance.
- A `JTable` does not support scrolling inherently, a `JScrollPane` is used to provide the scrolling facility.
- A `JTable` has two distinct parts:
  - **Column Header**: A single dimensional row of column header.
  - **Data**: A two dimensional rows and columns of data.
- Figure shows a `JTable`.



Student's personal details

Serial no.	Name	Roll	Date of birth	Gender	Qualification
1	Alvin Geneen	11	Mar-30-1982	Male	B.Sc
2	Edwin Knight	11	April-23-1982	Male	B.Tech
3	Louie Nixon	23	Sep-12-1983	Male	B.Tech
4	Oiane Norville	32	Aug-05-1981	Male	M.Tech
5	Jenn Gooden	31	Jan-21-1982	Female	B.sc
6	Marc Jordan	30	Feb-10-1982	Male	B.sc

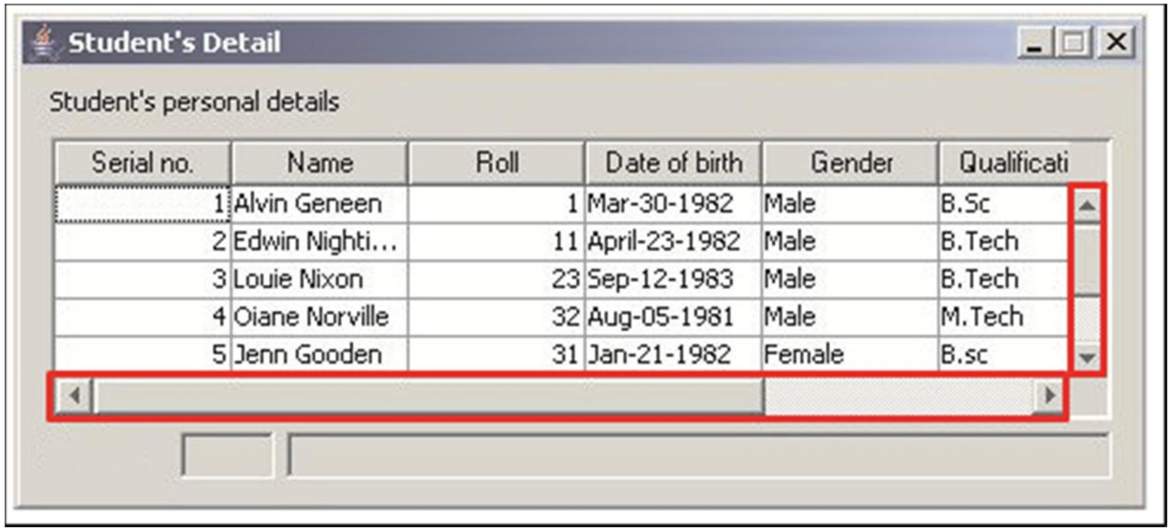
Constructors of JTable

A `JTable` is created using any one of the constructors mentioned:

- `public JTable()`
- `public JTable(int numRows, int numColumns)`
- `public JTable(Object[][] data, Object[] columns)`
- `public JTable(Vector data, Vector columns)`
- `public JTable(TableModel model)`

Displaying a JTable [1-2]

- To add and display a JTable, perform the following steps:
  - Instantiate a JTable object.
  - Instantiate a JScrollPane and pass the JTable object as a parameter.
  - Add the JScrollPane to the container.
  - The JScrollPane manages the vertical scrolling properly by default.
  - Figure displays the JTable with JScrollPane.



Student's personal details

Serial no.	Name	Roll	Date of birth	Gender	Qualificati
1	Alvin Geneen	1	Mar-30-1982	Male	B.Sc
2	Edwin Nighti...	11	April-23-1982	Male	B.Tech
3	Louie Nixon	23	Sep-12-1983	Male	B.Tech
4	Oiane Norville	32	Aug-05-1981	Male	M.Tech
5	Jenn Gooden	31	Jan-21-1982	Female	B.sc

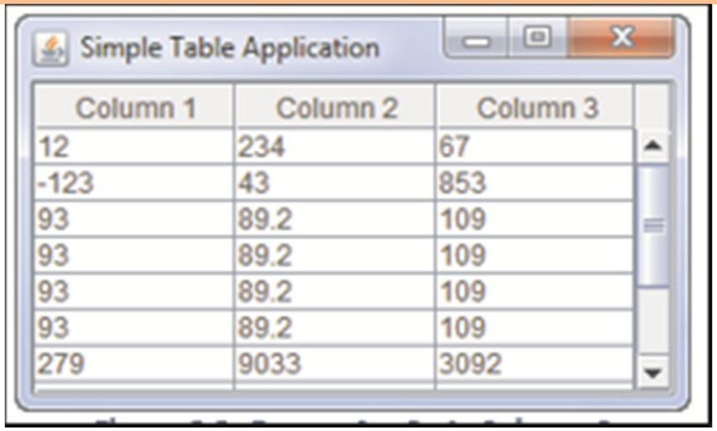
Displaying a JTable [2-2]

- Code Snippet shows how to add the table to the frame.

Code Snippet

```
JFrame frmReports;
JTable tblReports;
Vector vecData, vecColumns;
JScrollPane scpScroller;
// Creates a table with the specified data and columns
    tblReports = new JTable(data, columns);
// Provides the table to the scrollpane to facilitate scrolling
    scpScroller = new JScrollPane(tblReports);
 // Adds the scrollpane to the frame
    frmReports.getContentPane().add(scpScroller);
```

- Output:



Column 1	Column 2	Column 3
12	234	67
-123	43	853
93	89.2	109
93	89.2	109
93	89.2	109
93	89.2	109
279	9033	3092

Drawbacks of JTable

- The main drawbacks are as follows:

1
- All columns are of the same width by default.

2
- Ellipsis is used for displaying data if the column width is smaller.

3
- Horizontal scrolling will not be activated because of ellipsis.

- Customization of the table can be a solution to the problems mentioned.

Solutions to Drawbacks

- The `JTable` makes all the columns of same width by default.
- If the width of a column is smaller than the minimum required, an ellipsis is used.
- When you create a `JTable` with arrays or vector it has one major drawback.
- If the data changes, the new data is not reflected in the table.

Methods of JTable [1-3]

- **`print()`**
  - The `print()` method is used to print the columns and rows of the table. This method uses the default print mode `JTable.FIT_WIDTH`.
  - A modal print dialog box with a button labeled 'Abort' is displayed throughout the duration of printing.
  - Code Snippet shows how to print the contents of the table with the default print mode.

Code Snippet

```
JTable tblReports;
String[][] data;
String[] columns;
// Initialize the columns and data array
. . .
   tblReports = new JTable(data, columns);
// Prints the content of the table with default mode
    try
    {  table.print(); }
    catch(PrintingException ex)
    { System.out.println("Exception : " + ex.getMessage()); }
```

Methods of JTable [2-3]

- **`print(JTable.PrintMode printMode)`**
  - Specifies the mode of printing, the valid values are `JTable.FIT_WIDTH` and `JTable.FIT_NORMAL`.
  - Code Snippet shows how to print the contents of the table with the specified mode to print in normal width.

Code Snippet

```
JTable tblReports;
String[][] data;
String[] columns;
// Initialize the columns and data array
     . . .
   tblReports = new JTable(data, columns);
// Print the contents of the table with the current size
   try
   {  table.print(JTable.FIT_NORMAL); }
   catch(PrintingException ex)
   { System.out.println("Exception : " + ex.getMessage()); }
```

Methods of JTable [3-3]

- `print(JTable.PrintMode, MessageFormat messageformat)`
  - Method uses the specified print mode.
  - A `MessageFormat` object is used to specify the text to be printed as header and footer.
  - Code Snippet shows how to print the contents of the table with the specified mode and with header and footer.

Code Snippet

```
JTable tblReports; String[][] data;
String[] columns;
// Initialize the columns and data array
   . . .
   tblReports = new JTable(data, columns);
// Print the contents of the table with the current size
   try {
     // Create a header to print "Page 1", "Page 2" and so on
        MessageFormat header = new MessageFormat("Page {0}");
    // Create a footer to print "- 0 –", "- 1-" and so on
       MessageFormat footer = new MessageFormat("- {0} -");
       table.print(JTable.PrintMode.FIT_WIDTH, header, footer);
   }
    catch(PrintingException ex)
   { System.out.println("Exception : " + ex.getMessage()); }
```

Creating a Custom JTable

- A custom data model for `JTable` can be an object of any class which implements the `javax.swing.table.TableModel` interface.
- Generally, you do not implement this interface directly.
- Two default implementations of `TableModel` are:
  - `AbstractTableModel`: The abstract class `AbstractTableModel` provides an abstract implementation of `TableModel` interface.
  - `DefaultTableModel`: The `DefaultTableModel` is a concrete implementation of the `TableModel` interface which uses vectors to store the data and columns. All the methods of the `TableModel` interface are implemented.

AbstractTableModel Interface [1-3]

- Typically the `AbstractTableModel` is sub classed to provide a concrete implementation.
- Generally, the data model class constructor receives the two-dimensional array data and the single-dimensional array columns.
- To reflect the changes in the table you invoke the `fireTableDataChanged()` method of the `AbstractTableModel` class.
- You need to provide the implementation of the following methods only in case of `AbstractTableModel`:
  - `public int getRowCount():` Returns the number of rows. In case the data is a two dimensional array, the attribute length is returned. If data is a vector, the `size()` method is used to return the number of elements in the vector.
  - `public int getColumnCount():` Returns the number of columns. In case the columns is a single dimensional array, the attribute length is returned.
  - `public Object getValueAt(int row, int column):` Returns the value of the element at the specified row and column.

AbstractTableModel Interface [2-3]

- Code Snippet shows how to provide a concrete implementation for a custom table model and add new data.

Code Snippet

```
import javax.swing.*;
import javax.swing.table.*;
public class ReportTable extends AbstractTableModel  {
 // Arrays to hold data and columns
    String[][] data;
    String[] columns;
    . . .
  // Construct to initialize the model with data and columns
     public ReportTable (String[][] data, String[] columns) {
         // Initialize the data and columns
            this.data = data;
            this.columns = columns;
    }
    // A user defined method to set new data
   // This method should be called subsequently on data change
```

AbstractTableModel Interface [3-3]

```
public void setData(Object[][] data) {
 this.data= data;
 fireTableDataChanged();
 }
 public int getRowCount() {
  return data.length;     }

 public int getColumnCount() {
  return column.length;
 }
 public Object getValueAt(int row, int column) {
   return data(row,column);
 }
```

Methods of JTable [1-2]

- Once your model class is ready, perform the following steps:
- **Instantiate data model class**
  - Before you instantiate the data model class object, you create the two-dimensional data array, and the single-dimensional column array.
  - Next you declare and instantiate the data model class object and send the data and column array as arguments.
  - Code Snippet shows how to create the data and columns array and instantiate the data model.

Code Snippet

```
ReportTable reportTable;
String[][] data;
String[] columns;
// Initialize data and columns
   String[][] data = { {…}, {…} };
   String[] columns = {"…", "…"};
   . . .
// Instantiate the data model reportTable with the data and columns
from the arrays
   reportTable = new ReportTable(data, columns);
```

Methods of JTable [2-2]

- **Instantiate `JTable` with data model object**
  - Once the data model object is ready, you declare the table object and instantiate it by passing the data model object as an argument in the constructor.
  - Code Snippet shows how to declare a table and instantiate it with a table model object.

Code Snippet

```
JTable tblReports;
ReportTable reportTable;
String[][] data;
String[] columns;
// Instantiates the table tblreports with the data model object
   tblReports = new JTable(reportTable);
      . . .
// Whenever the data Changes replace the new data into the two
dimensional array
// Assigns the new data to the model by invoking setData()
```

Event Handling Using MouseListener

To handle the mouse click event, perform the following steps:

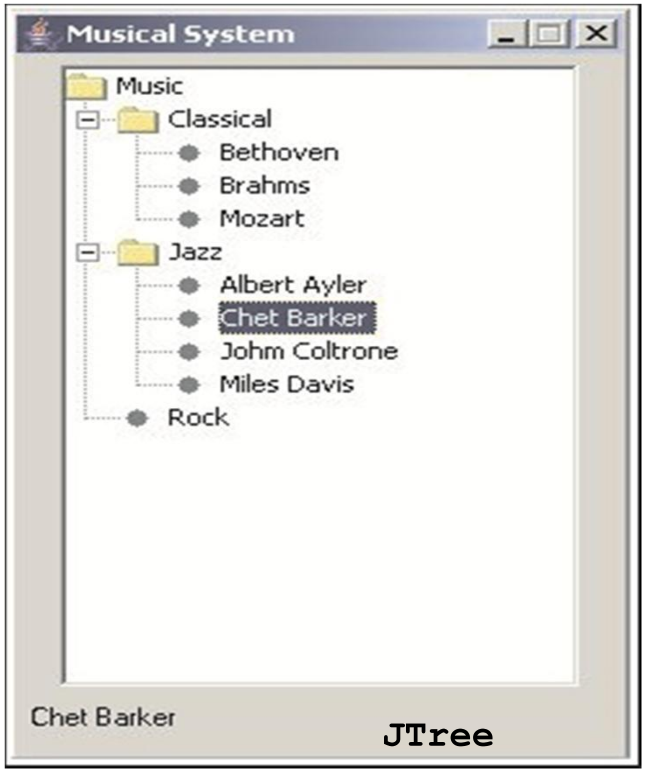
- Register a `MouseListener`.
- Add the action code in `mouseClicked()` method.
- Code Snippet register the mouse listener to the reports table `tb1Reports`.

Code Snippet

```
// Register the mouse listener to the reports table
tblReports.addMouseListener(new MouseAdapter() {
  // Provide implementation to the mouse clicked event
    public void mouseClicked(MouseEvent me) {
        // Add the code
            . . .
            . . .
        }
    });
```

JTree

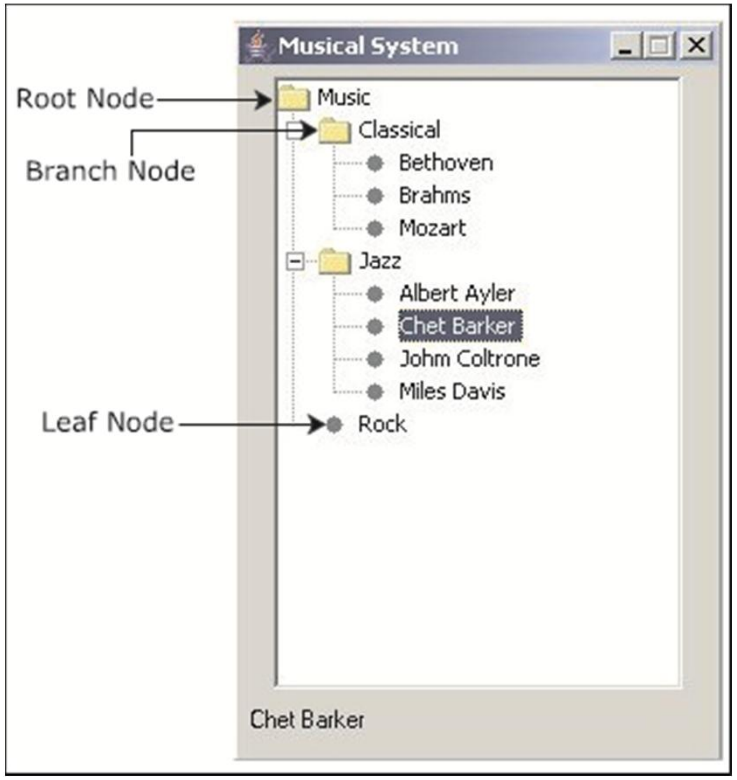
- A JTree is a component which displays its data in a hierarchical way.
- A JTree can be used to represent a complex tree like structure of hierarchical data.
  - Hierarchical data cannot be easily represented in terms of rows and columns.
  - They can be best represented in a tree like structure.
- A JTree can be used to display:
  - The file system on an operating system
  - Organizational hierarchy
  - Ancestry details



JTree

Structure of JTree

- A `JTree` displays data vertically, each row contains only one item called node.
- Every `JTree` has a root node at the top, all other nodes descend from the root node.
- Initially, except the root node, all branch nodes are in a collapsed state.
- You typically click the root node to expand the tree, and then the branch node.
- Nodes which have children are called branch nodes; they expand and collapse.
- A node which does not have children is called a leaf node, it cannot expand and collapse.



Constructing a JTree

- To create a `JTree`, perform the following steps:
    - Create an instance of `DefaultMutableTreeNode`
    - Create an instance of a `JTree`
    - Code Snippet creates the instance of a `JTree` and represents a root node.

Code Snippet

```
DefaultMutableTreeNode dmtnRoot;
JTree treTree;
// Creates an instance of a JTree with the root node as an
// argument
   treTree  = new JTree(dmtnRoot);
```

Configuring and Displaying JTree [1-2]

To add, configure, and display a `JTree`, perform the following steps:

1
- Create instances of `DefaultMutableTreeNode` to represent the branch nodes and leaf nodes.

2
- Add the branch nodes and leaf nodes to the root node.

3
- Create a `JScrollPane` and send the tree instance as an argument.

4
- Add the `JScrollPane` to the container.

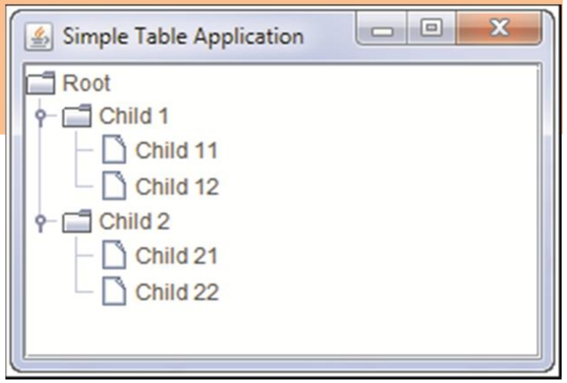
Configuring and Displaying JTree [2-2]

- Code Snippet shows how to add, configure, and display a `JTree`.

Code Snippet

```
JFrame frmTree;
DefaultMutableTreeNode dmtnBranch1, dmtnBranch2;
DefaultMutableTreeNode dmtnLeaf1, dmtnLeaf2;
DefaultMutableTreeNode dmtnRoot; JTree treTree;
    . . .
// Adds the branch-nodes to the root node
   dmtnRoot.add(dmtnBranch1);
   dmtnRoot.add(dmtnBranch2);
// Adds the leaf-nodes to the branch node
   dmtnBranch1.add(dmtnLeaf1);
   dmtnBranch2.add(dmtnLeaf2);
// Creates a JScrollPane with the tree instance as an argument
   JScrollPane scpScroller  = new JScrollPane(treTree);
// Adds the JScrollPane to the frame
   frmTree.getContentPane().add(scpScroller);
```

- Output:



Event Handling Using TreeSelectionListener

- When a node is selected the `JTree` fires `valueChanged()` event.
- A `TreeSelectionListener` object is registered to handle this event.
- To handle the node selection event, perform the following steps:
  - Register a `TreeSelectionListener`
  - Add code in the `valueChanged`() method
- Code Snippet shows how to register tree selection listener to the `JTree` and handle the node selection event of the `JTree`.

Code Snippet

```
JTree treTree;
// Registers the tree selection listener to the tree
   treTree.addTreeSelectionListener(new TreeSelectionListener() {
   // Handles value change event
       public void valueChanged(TreeSelectionEvent e) {
         // Add the code to handle the event
   }});
   frmTree.getContentPane().add(scpScroller);
```

Event Handling Using TreeExpansionListeners

- To handle the tree expansion event, perform the steps:
  - Register a `TreeExpansionListener`
  - Add code in the `treeExpanded()` method
  - Add code in the `treeCollapsed()` method
- Code Snippet shows how to register tree expansion listener to the `JTree` and handle the node expansion and collapse event of the `JTree`.

Code Snippet

```
JTree treTree;
. . .
treTree.addTreeExpansionListener(new TreeExpansionListener() {
    // Handles the node expansion event
       public void treeExpanded(TreeExpansionEvent e)
       {
          // Action code for the event tree expanded
             . . .
       }
      public void treeCollapsed(TreeExpansionEvent evt)
      {
         JTree tree = (JTree) evt.getSource();
         TreePath path = evt.getPath();
         System.out.println("treeCollapsed");
      }
}});
```

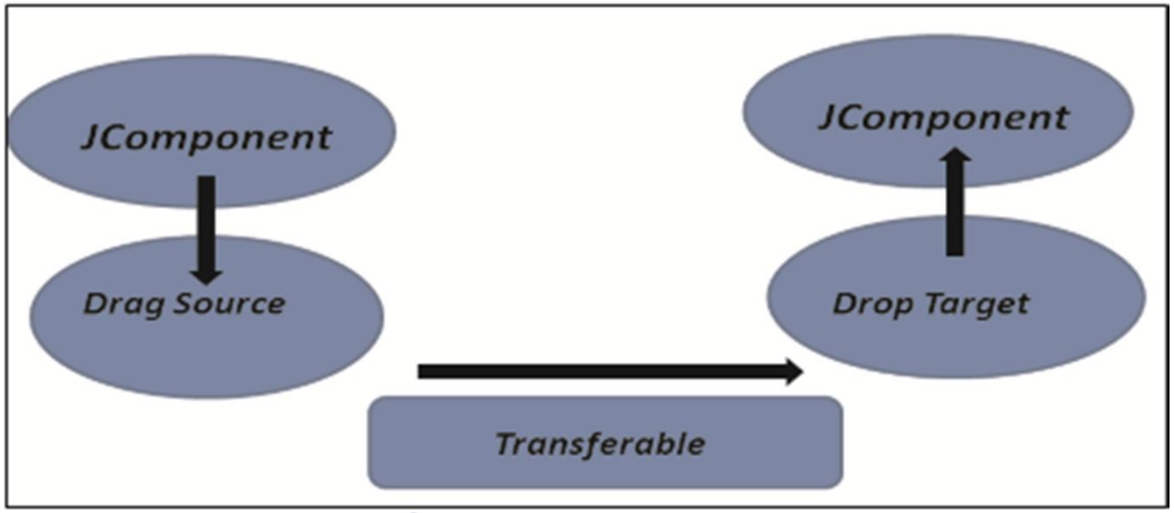
Transfer of Data in Swing

- Modern GUI applications provide the ability to support transfer of data between components in the application.

- The ability to transfer the data can be achieved through two ways:
  - Drag and Drop (DND) support.
  - Clipboard transfer through Cut or Copy or Paste (CCP).

Drag and Drop (DND)

- The DND can be used to transfer binary data or graphical objects.
- For example, consider a situation where an image being dragged between two Java applications, then data transfer is of binary.
- Similarly, if a component is dragged, then the transfer is of graphical components.
- The data is transferred using a drag board, which is a clipboard that is used specifically for Drag and Drop operations.
- Figure shows the drag and drop flow.



DND Support for Swing Components [1-2]

- The DND support is provided by most Swing components in-built.
- Once the `setDragEnabled(True)` method is invoked on a component it supports the drag gesture.
- The following is a list of components that support the drag gesture:
    - `JColorChooser`
    - `JEditorPane`
    - `JFileChooser`
    - `JFormattedTextField`
    - `JList`
    - `JTable`
    - `JTextArea`
    - `JTextField`
    - `JTextPane`
    - `JTree`

DND Support for Swing Components [2-2]

- Similarly, the components that provide support for drop are as follows:
  - `JEditorPane`
  - `JFormattedTextField`
  - `JPasswordField`
  - `JTextArea`
  - `JTextField`
  - `JTextPane`
  - `JColorChooser`
- Some components calculates the drop location for displaying the dragged component.
- Example: Components such as `JList`, `JTable`, and `JTree` allows the user to specify a drop mode to handle component details.
- A source can support one or more targets.
- A target can accept one or more transfer modes.

TransferHandler Class

- Provides an easy mechanism for transferring data between components.
- Used to perform DND operations by performing drag from a component and drop to a component.
- Used to work with CCP operations that performs data operation through clipboard.
- Provides a constructor method that implements the default behavior for the data to be transferred by specifying the property name.
- Some of the constructor methods are:
  - `TransferHandler()`
  - `TransferHandler(String property)`

Built-in DND Mechanism [1-2]

- Most of the Swing components are provided with the default implementation of the transfer handler.
- Thus, you need to turn the drag and drop facility ON to perform DND operations on them.
- Some of the methods that are supported by the Swing components to enable built-in data transfer are as follows:
  - `setDragEnabled(boolean)`
  - `setDropMode(DropMode)`
  - `setTransferHandler(TransferHandler)`
- Code Snippet demonstrates the drag of item from `JList` into a `JTextField` instance.

Code Snippet

```
public class DragAndDropDemo extends JFrame {
JTextField field;
JButton button;
 public DragAndDropDemo() {
   setTitle("Built-in Drag and Drop");
   setLayout(new GridLayout(1, 2));
```

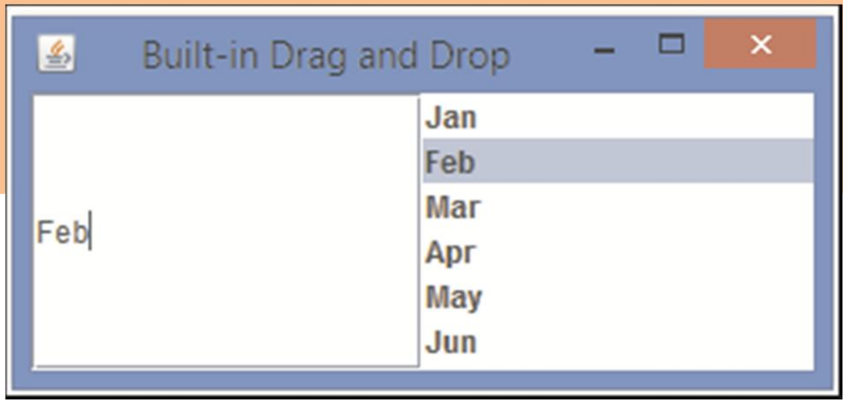
Built-in DND Mechanism [2-2]

```
// Creates an array and initializes the JList
   String[] month = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul"};
   JList l = new JList(month);
   field = new JTextField();
   field.setBounds(30, 50, 50, 25);
   add(field);
   add(l);

// Enables the drag on JList
    l.setDragEnabled(true);

// Sets the transferable property for the text box
    field.setTransferHandler(new TransferHandler("text"));
    setSize(330, 150);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
 }
 public static void main(String[] args) {
       new DragAndDropDemo();
 }
}
```

- Output:



Import and Export Data Using TransferHandler Class [1-2]

- Export methods are used for exporting data from a component.
- They are invoked for the drag gesture and copy action when the component is the source of the operation.
- Some of the commonly used methods are:
  - `getSourceActions(JComponent)`
  - `createTransferable(JComponent)`
  - `exportDone(JComponent, Transferable, int)`
- Code Snippet shows an example of `TransferHandler` methods for Data export.

Code Snippet

```
int getSourceActions(JComponent a)
{  return COPY_OR_MOVE; }

Transferable createTransferable(JComponent a) {
   return new StringSelection(a.getSelection());
}
void exportDone(JComponent a, Transferable t1, int act) {
  if (act == MOVE) {  c.removeSelection(); }
}
```

Import and Export Data Using TransferHandler Class [2-2]

- Import methods are used for importing data into a component.
- They are invoked for the drop gesture, or the paste action, when a component is the target of the operation.
- Some methods for importing data are:
  - `canImport(TransferHandler.TransferSupport)`
  - `importData(TransferHandler.TransferSupport)`

Drag Gesture

- Drag and drop is initiated with a gesture, which is a mouse down followed by mouse drags.
- Drag gesture recognizers fire events when drag gestures are detected in an associated component.
- The `DragSourceListener` defines the event interface for initiators of Drag and Drop operations and trace user's gestures.
- Some of the drag gesture methods where drag events fired are as follows:
  - `public void dragDropEnd(DragSourceDropEvent e){}`
  - `public void dragEnter(DragSourceDragEvent e){}`
  - `public void dragExit(DragSourceEvent e){}`
  - `public void dropActionChanged(DragSourceDragEvent e){}`

Implementation of DND

- Before a DND operation is fired, it is necessary to create Drag sources, Drop targets, drag gesture recognizers, and transferables.
- Drag source listeners initiate the drag. Drop target listeners handle drop events. Programmer intervention is needed only for wrapping data in transferables and handling drop.
- The objects involved in as drag and drop operations are as follows:
  - The `startDrag()` method is initiated by the Drag source.
  - Drag gesture recognizers are created using the `createDragGestureRecognizer()` or the `createDefaultDragGestureRecognizer()`.
  - Drop target is linked to a component and a target listener. Whenever target event occur the listener is notified.
  - Transferables wrap data to be transferred.
  - The transfer modes supported by the gesture source are defined, and the data to be transferred is placed onto the dragboard in the handler of the drag event.
  - Drag listeners are notified of the drag gesture. The listeners react by initiating drag by invoking `DragSource.startDrag()`.
  - Source listeners react to events that occur after the drag is initiated.
  - Drop target listeners respond to drop events and handle the actual drop of data.
  - After a drag and drop event is started, any node or scene the mouse moves over is a potential target for the drop. The target object that receives the data is specified by implementing the `Drag_Over` event handler.

Summary

- The JTable is a class that allows data to be displayed and edited in table form. It is simply the user's view of data and does not contain data or cache data.
- The JTable filter and sort through data. This is enabled by the RowSorter() method columns in a JTable may be arranges differently in the view than in the modal, but this does not affect the data in any way.
- The JTree class displays a set of hierarchical data as an outline or a tree. The tree had nodes, branches and leaves. Each data row that is displayed is a node.
- Nodes are non leaf nodes that display all the children when the parent is expanded. A collapsed node hides all its children. Nodes that have children are Branch nodes.
- The ability to transfer data between components in an application is very essential. The DND feature of Java swing gives users this ability. While most of the JComponent support DND inherently, some of them needs the DND to be invoked.
- A DND operation occurs between two objects referred to as the gesture source and the gesture target. The type of transfer between the source and target is specified by the transfer Modes such as Copy, Move, and Link.