1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 26: Using JList and JTable with models

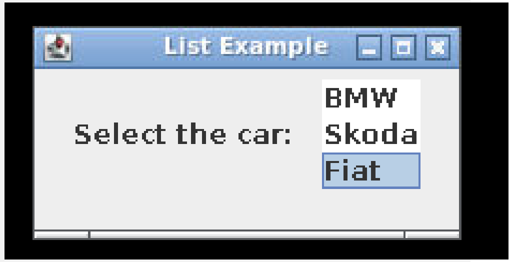
Models for JList

- There are different pre-defined models for `JList`:
  - **`ListModel`**: stores the information about the data items displayed in the list and the list states. To initialize a `ListModel`, you must either:
    - Use the class **`DefaultListModel`** — everything is taken care of for you.
    - Extend the class **`AbstractListModel`** — you manage the data and invoke the "fire" methods; you must implement `getSize()` and `getElementAt()` methods inherited from `ListModel` interface
    - Implement the `ListModel` interface — you manage everything
  - **`ListSelectionModel`**: manages the selection of list data items

Initialise JList using DefaultListModel

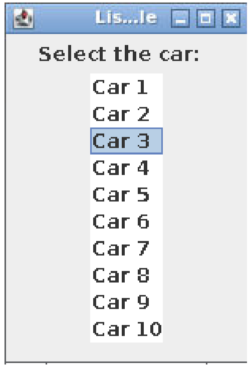
- You can use `DefaultListModel` to add items and initialise `JList`

```
...
class Car {
  private String make;
  public Car(String make) { this.make = make; }
  @Override
  public String toString() { return make; }
}
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    cars.addElement(new Car ("BMW"));
    cars.addElement(new Car ("Skoda"));
    cars.addElement(new Car ("Fiat"));
    JList<Car> list = new JList<>(cars);
...
```



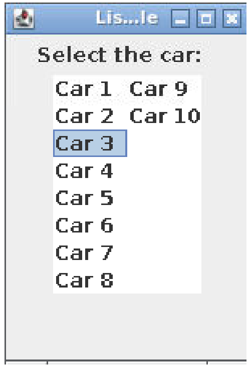
Using different JList layouts

- Different layouts can be chosen for `JList` using method **`setLayoutOrientation()`**



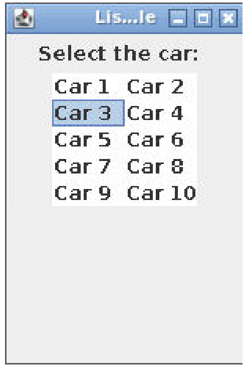
```
Select the car:
Car 1
Car 2
Car 3
Car 4
Car 5
Car 6
Car 7
Car 8
Car 9
Car 10
```

`VERTICAL`



```
Select the car:
Car 1  Car 9
Car 2  Car 10
Car 3
Car 4
Car 5
Car 6
Car 7
Car 8
```

`VERTICAL_WRAP`



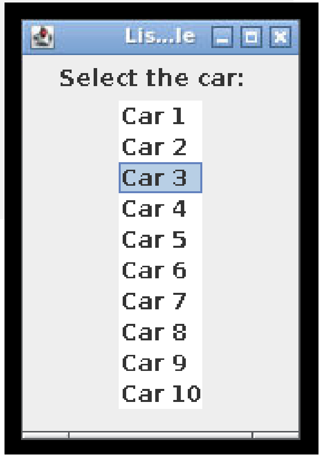
```
Select the car:
Car 1  Car 2
Car 3  Car 4
Car 5  Car 6
Car 7  Car 8
Car 9  Car 10
```

`HORIZONTAL_WRAP`

Vertical JList layout example

- `JList.VERTICAL` inidcates vertical layout in a single column (default layout)

```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(JList.VERTICAL);
...
```



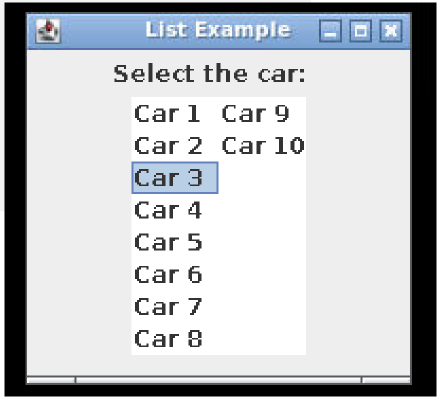
Select the car:

Car 1
Car 2
Car 3
Car 4
Car 5
Car 6
Car 7
Car 8
Car 9
Car 10

Vertical wrap JList layout example

- `JList.VERTICAL_WRAP` indicates "newspaper style" layout with cells flowing horizontally, then vertically

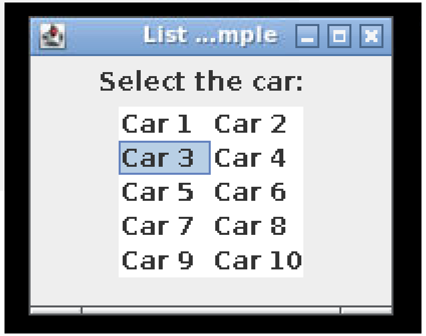
```java
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(JList.VERTICAL_WRAP);
...
```



Vertical wrap JList layout example

- `JList.VERTICAL_WRAP` indicates “newspaper style” layout with cells flowing horizontally, then vertically

```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
...
```

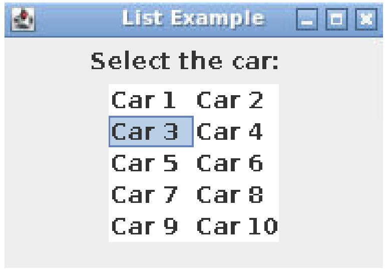


Select the car:

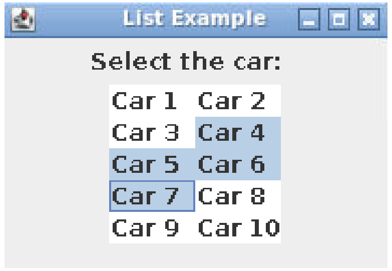
```
Car 1  Car 2
Car 3  Car 4
Car 5  Car 6
Car 7  Car 8
Car 9  Car 10
```

Using different JList selection modes

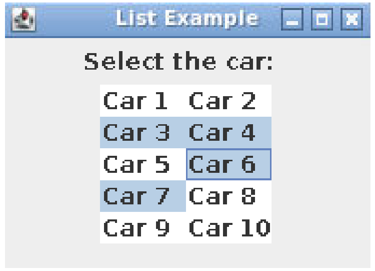
- Different selection modes can be chosen for `JList` using method **`setSelectionMode()`**



SINGLE_SELECTION



SINGLE_INTERVAL_SELECTION

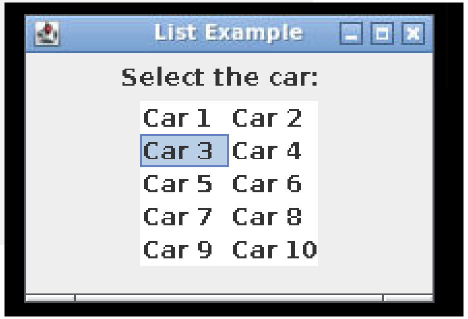


MULTIPLE_INTERVAL_SELECTION

Single selection example

- `ListSelectionModel.SINGLE_SELECTION` indicates that only one item can be selected at a time

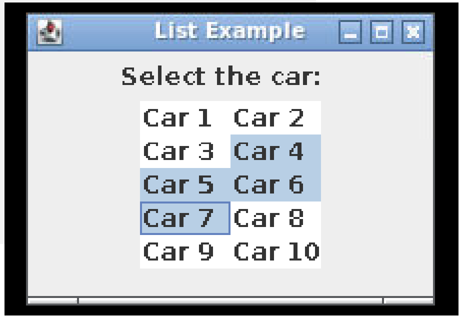
```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(
      ListSelectionModel.SINGLE_SELECTION);
...
```



Single interval selection example

- `ListSelectionModel.SINGLE_INTERVAL_SELECTION` indicates that only one contiguous interval can be selected at a time

```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(
      ListSelectionModel.SINGLE_INTERVAL_SELECTION);
...
```

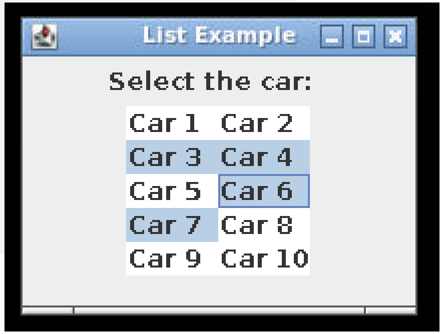


UNIVERSITY OF ABERDEEN

Multiple interval selection example

- `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION` indicates that items can be selected freely (default mode)

```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<10; i++) {
      cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    list.setLayoutOrientation(
      ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
...
```



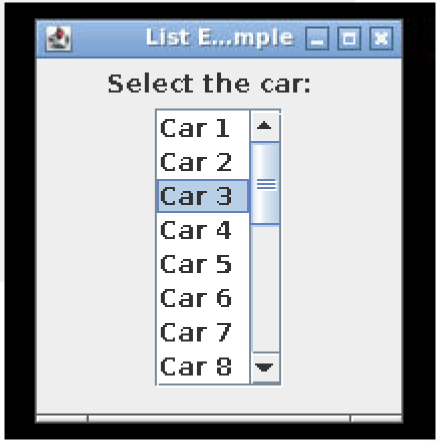
Using scroll bars with lists

- Some elements such as `JLists` can have many items and possible not fit in the visible area reserved for them
    - Scroll bars can be enabled by adding the component to a scroll pane
- A **`JScrollPane`** object provides a scrollable view of a component
    - You can add `JList` object (or any other `JComponent`) to `JScrollPane` object by passing it as a parameter to the constructor
    - Additional parameters can be used to further control the behavior of the scroll bar (e.g., whether the scroll bar is always visible, or only if the content does not fit in the visible area)

List with scroll bar example (1)

- Simply create a scroll pane with the list as a parameter, and add it to the panel

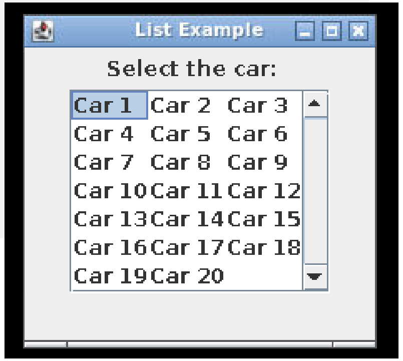
```
...
    DefaultListModel<Car> cars = new DefaultListModel<>();
    for(int i=0; i<20; i++) {
        cars.addElement(new Car("Car " + (i+1)));
    }
    JList<Car> list = new JList<>(cars);
    JScrollPane listPane = new JScrollPane(list);
    panel.add(label);
    panel.add(listPane);
...
```



List with scroll bar example (2)

- Force vertical scroll bar to be visible always using another constructor with parameters for vertical and horizontal scroll bars

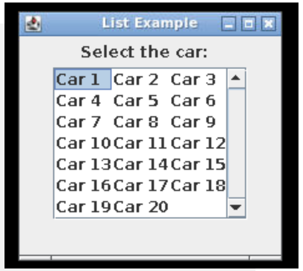
```
...
    JList<Car> list = new JList<>(cars);
    list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
    list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
    JScrollPane listPane = new JScrollPane(list,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    panel.add(label);
    panel.add(listPane);
...
```



Add a list selection listener

- To process list selection events, register `ListSelectionListener` using `ListSelectionModel`
  - Check the event's `getValueIsAdjusting()` to make sure your listener is not reacting on wrong type of events

```
...
    ListSelectionModel lsModel = list.getSelectionModel();
    lsModel.addListSelectionListener(new ListSelectionListener() {
      @Override
      public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting() == false) {
          System.out.println("Item " + list.getSelectedIndex() +
                                     " selected");
        }
      }
    });
...
```



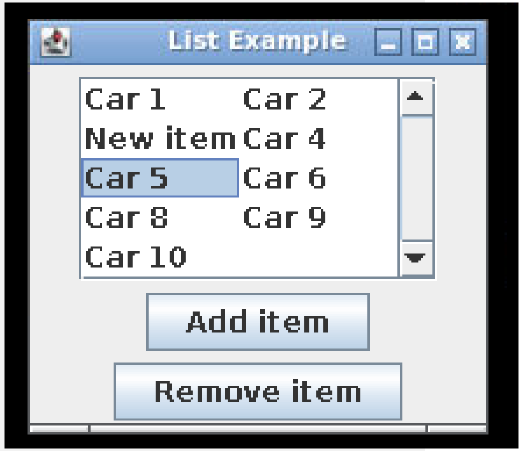
```
$ java ListSelectionExample3
Item 2 selected
Item 16 selected
Item 0 selected
```

Adding and removing items

- List items can be removed and added dynamically using **`remove(index)`** and **`insertElementAt(item,index)`** methods of `ListModel` object
  - Note that the methods do not check if the index is valid: you need to make sure you are not e.g., removing from an empty list, or adding beyond the end of the list!

Example of adding and removing items

```
...
    JButton removeButton = new JButton("Remove item");
    JButton addButton = new JButton("Add item");
    removeButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        cars.remove(list.getSelectedIndex());
      }
    });
    addButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        cars.insertElementAt(new Car("New item"),
                                        list.getSelectedIndex());
      }
    });
...
```



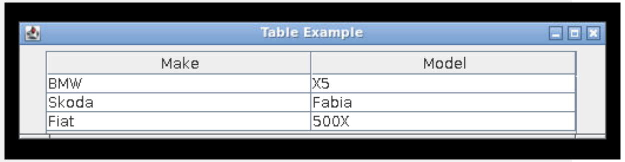
Note that exception is thrown, if you try to remove an item when none is selected!

Using component JTable

- `JTable` class allows you to create tabular views of your data
  - A `JTable` instance presents the user with a group of items arranged in a form of a table with rows and columns
  - User can be optionally also allowed to edit the table data
  - Tables can get complex, and we will just look at the basics
- A table can be initialised directly by passing the column names and data to the `JTable` constructor
  - All the cells will be editable, and data will be treated as `Strings`
  - This method is only suitable if you have the data available in advance

Simple example of using JTable directly

```java
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class SimpleTableExample {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Table Example");
    JPanel panel = new JPanel();
    String cols[] = {“Make", "Model"};
    String cars[][] = { {"BMW", "X5"},
                        {"Skoda", "Fabia"},
                        {"Fiat", "500X"} };
    JTable table = new JTable(cars, cols);
    JScrollPane sp = new JScrollPane(table);
    panel.add(sp);
```



Make	Model
BMW	X5
Skoda	Fabia
Fiat	500X

```java
    frame.add(panel);
    frame.setSize(500,100);
    frame.setVisible(true);
  }
}
```

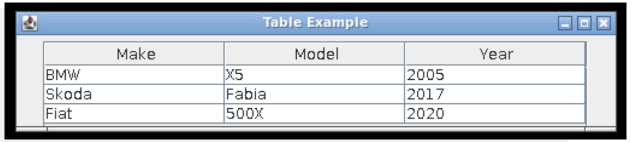
UNIVERSITY OF ABERDEEN

Models for JTable

- Same as with JList: to use a table model you must either:
  - Use the class **`DefaultTableModel`:** everything is taken care of for you
  - Extend the class **`AbstractTableModel`:** you manage the data and invoke the "fire" methods
    - You must implement `getRowCount()`, `getColumnCount()`, and `getValueAt()` methods inherited from the `TableModel` interface.
  - Implement the interface **`TableModel`**: you manage everything

Using JTable via custom table model

```java
import javax.swing.*;
import javax.swing.table.AbstractTableModel;
class MyTableModel extends AbstractTableModel {
  private String[] columnNames = {"Model", "Make", "Year"};
  private Object[][] data = {{"BMW", "X5", Integer.valueOf(2005)},
                             {"Skoda", "Fabia", Integer.valueOf(2017)},
                             {"Fiat", "500X", Integer.valueOf(2020)}};
  public int getRowCount() {
    return data.length;
  }
  public int getColumnCount() {
    return columnNames.length;
  }
  public String getColumnName(int col) {
    return columnNames[col];
  }
  public Object getValueAt(int row, int col) {
    return data[row][col];
  }
}
```



Make	Model	Year
BMW	X5	2005
Skoda	Fabia	2017
Fiat	500X	2020

```java
public class CustomTableModelExample {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Table Example");
    JPanel panel = new JPanel();
    MyTableModel model = new MyTableModel();
    JTable table = new JTable(model);
    JScrollPane sp = new JScrollPane(table);
    panel.add(sp);
    frame.add(panel);
    frame.setSize(500,100);
    frame.setVisible(true);
  }
}
```

Using tables with selection listener

- Interface `ListSelectionListener` can be implemented to listen selection events, such as user selecting a cell

```
...
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
...
    ListSelectionModel lsModel = table.getSelectionModel();
    lsModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    lsModel.addListSelectionListener(new ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        if(e.getValueIsAdjusting()) {
          System.out.println("Selected: " + table.getValueAt(
                table.getSelectedRows()[0],
                table.getSelectedColumns()[0]).toString());
        }
      }
    });
...
```

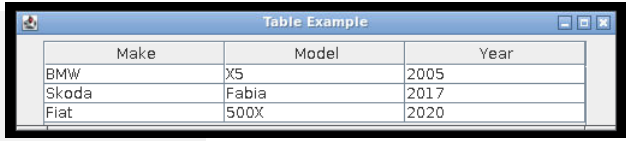


Table Example

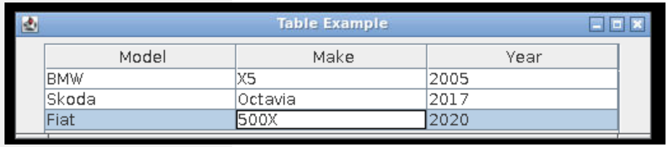
Make	Model	Year
BMW	X5	2005
Skoda	Fabia	2017
Fiat	500X	2020

```
$ java CustomTableModelExample
Selected: BMW
Selected: Fabia
Selected: 500X
Selected: Skoda
Selected: BMW
Selected: 2017
Selected: Fiat
```

Using editable cells with a custom model

- To make cells editable with a model extended from `AbstractTableModel`, you need to implement **`isCellEditable()`** and **`setValueAt()`** methods

```
...
class MyTableModel extends AbstractTableModel {
...
  public boolean isCellEditable(int row, int col) {
    return true;
  }
  public void setValueAt(Object value, int row, int col) {
    data[row][col] = value;
    fireTableCellUpdated(row, col);
    System.out.println("Cell (" + row + "," + col +
      ") edited with value: " + value.toString());
  }
...
```



Model	Make	Year
BMW	X5	2005
Skoda	Octavia	2017
Fiat	500X	2020

```
$ java CustomTableModelExample2
Cell (1,1) edited with value: Octavia
Selected: 500X
```

Summary

- In Swing, most `JComponent` classes have pre-defined models for storing data related to components
  - Models help to separate the view and the related data
  - Models often make it easier to implement custom functionality in GUI elements
- Models are especially useful for complex GUI components, such as lists and tables
  - `JList` and `JTable` components have multiple predefined models, and it is possible to implement custom models as well

Questions, comments?

1495 UNIVERSITY OF ABERDEEN