1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 25: Models in Swing

References and learning objectives

- Today’s sessions are mostly based on Oracle documentation:
    - https://docs.oracle.com/javase/tutorial/uiswing
- After today’s session, you should be able to:
    - Use Swing models in your Java GUI implementation
    - Implement custom functionalities in `JList` and `JTable` components

Swing models

- Models store the state of the component (e.g., mnemonics, whether it is enabled, selected, etc.) and data (e.g., items displayed in a list)
    - Most of the Swing components have predefined models
- Some components, such as lists, have multiple models
    - For example, `JList` uses `ListModel` and also `ListSelectionModel`
- For simple components (e.g., buttons) you would normally interact with the component directly, whereas for more complex components, such as lists and tables, interacting with models is a better choice

Why to use models?

- Models allow the separation of data from the view and controller if the MVC pattern is applied
- Default models can be extended and thus provide custom functionalities and flexibility in deciding how data is stored and retrieved
- Models automatically propagate changes to all registered listeners, allowing the view (i.e., GUI) to be updated

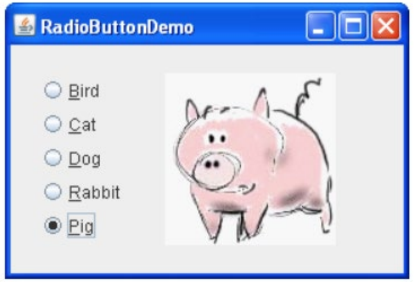
Using models vs. components directly

- There are different ways to achieve the same outcome in Java

```java
JRadioButton pigButton = new JRadioButton("Pig");
pigButton.setMnemonic(KeyEvent.VK_P);
pigButton.setActionCommand("Pig");
pigButton.setSelected(true);

// Use the component directly
System.out.println(pigButton.isSelected());

// Use the model
DefaultButtonModel model = (DefaultButtonModel)pigButton.getModel();
System.out.println(model.isSelected());
```

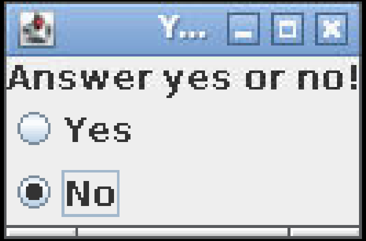


- Most component classes inherited from `JComponent` have a model by default, and it can be accessed using method `getModel()`

Interact with radio buttons directly

```java
import javax.swing.*;
public class YesNoButtonExample {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Yes or No?");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel();
    BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.Y_AXIS);
    panel.setLayout(boxlayout);
    JLabel question = new JLabel("Answer yes or no!");
    ButtonGroup group = new ButtonGroup();
    JRadioButton yes = new JRadioButton("Yes");
    JRadioButton no = new JRadioButton("No");
    group.add(yes); group.add(no);
    panel.add(question);
    panel.add(yes); panel.add(no);
    frame.add(panel);
    frame.pack();
    frame.setVisible(true);
    no.setSelected(true);
    System.out.println("Yes selected: "+yes.isSelected());
    System.out.println("No selected: "+no.isSelected());
  }
}
```

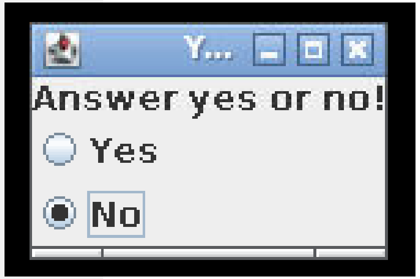
```
$ java YesNoButtonExample
Yes selected: false
No selected: true
```



Interact with radio buttons via model

```java
import javax.swing.*;
public class YesNoButtonExample2 {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Yes or No?");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel();
    BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.Y_AXIS);
    panel.setLayout(boxlayout);
    JLabel question = new JLabel("Answer yes or no!");
    ButtonGroup group = new ButtonGroup();
    JRadioButton yes = new JRadioButton("Yes");
    JRadioButton no = new JRadioButton("No");
    DefaultButtonModel yesModel = (DefaultButtonModel)yes.getModel();
    DefaultButtonModel noModel = (DefaultButtonModel)no.getModel();
    group.add(yes); group.add(no);
    panel.add(question);
    panel.add(yes); panel.add(no);
    frame.add(panel);
    frame.pack();
    frame.setVisible(true);
    no.setSelected(true);
    System.out.println("Yes selected: "+yesModel.isSelected());
    System.out.println("No selected: "+noModel.isSelected());
  }
}
```

```
$ java YesNoButtonExample
Yes selected: false
No selected: true
```



Defining custom button model

```java
import javax.swing.*;
class CustomButtonModel extends JToggleButton.ToggleButtonModel {
  private AbstractButton button;
  private String text;
  CustomButtonModel(AbstractButton button) {
    this.button = button;
    text = button.getText();
  }
  public void printStatus() {
    System.out.println(text + " selected: " + isSelected());
  }
  @Override
  public void setSelected(boolean b) {
    if(b) {
      button.setText(text + " (currently enabled)");
    }
    else {
      button.setText(text + " (currently disabled)");
    }
    super.setSelected(b);
  }
}
```

Custom radio button model should inherit toggle button model

New method for additional functionality

Overriden method for additional functionality

Using custom button model (1)

```
import javax.swing.*;
class CustomButtonModel extends JToggleButton.ToggleButtonModel {
  private AbstractButton button;
  private String text;
  CustomButtonModel(AbstractButton button) {
    thi
    tex
  }
  publi
    Sys
  }
  @Over
  publi
    if(
      b
    }
    els
      b
    }
    sup
  }
}
```

```
public class YesNoButtonExample2 {
  public static void main(String[] args) {
...
    JRadioButton yes = new JRadioButton("Yes");
    JRadioButton no = new JRadioButton("No");
    CustomButtonModel yesModel = new CustomButtonModel(yes);
    CustomButtonModel noModel = new CustomButtonModel(no);
    yes.setModel(yesModel);
    no.setModel(noModel);
...
    yes.setSelected(false);
    no.setSelected(true);
    yesModel.printStatus();
    noModel.printStatus();
  }
}
```

Instantiate custom models and assign to radio button objects

Using custom button model (2)

```
import javax.swing.*;
class CustomButtonModel extends JToggleButton.ToggleButtonModel {
  private AbstractButton button;
  private String text;
  Custo
    thi
    tex
  }
  publi
    Sys
  }
  @Over
  publi
    if(
      b
    }
    els
      b
    }
    sup
  }
}
```

```
public class YesNoButtonExample2 {
  public static void main(String[] args) {
...
    JRadioButton yes = new JRadioButton("Yes");
    JRadioButton no = new JRadioButton("No");
    CustomButtonModel yesModel = new CustomButtonModel(yes);
    CustomButtonModel noModel = new CustomButtonModel(no);
    yes.setModel(yesModel);
    no.setModel(noModel);
...
    yes.setSelected(false);
    no.setSelected(true);
    yesModel.printStatus();
    noModel.printStatus();
  }
}
```

Use custom method `printStatus()`

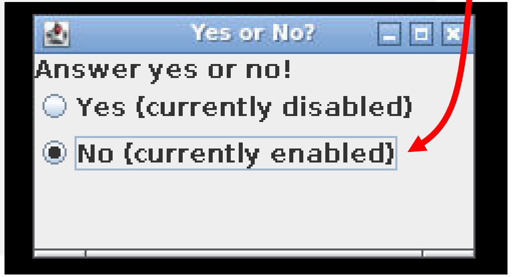
Using custom button model (3)

```
import javax.swing.*;
class CustomButtonModel extends JToggleButton.ToggleButtonModel {
  private AbstractButton button;
  private String text;
  Custo
    thi
    tex
  }
  publi
    Sys
  }
  @Over
  publi
    if(
      b
    }
    els
      b
    }
    sup
  }
}
```

```
public class YesNoButtonExample2 {
  public static void main(String[] args) {
...
    JRadioButton yes = new JRadioButton("Yes");
    JRadioButton no = new JRadioButton("No");
    CustomButtonModel yesModel = new CustomButt
    CustomButtonModel noModel = new CustomButton
    yes.setModel(yesModel);
    no.setModel(noModel);
...
    yes.setSelected(false);
    no.setSelected(true);
    yesModel.printStatus();
    noModel.printStatus();
  }
}
```

```
$ java YesNoButtonExample3
Yes selected: false
No selected: true
█
```

The text changes when the button is toggled



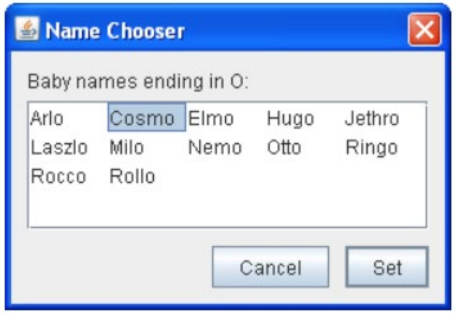
Answer yes or no!

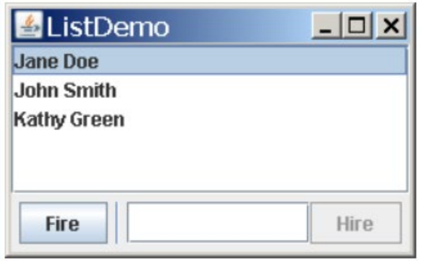
○ Yes (currently disabled)

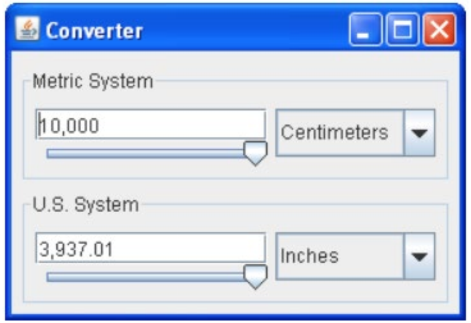
● No (currently enabled)

Using models for complex interaction

- The benefits of using models with simple components like `JButton` are usually limited, but with complex components, models are essential
  - With components such as **`JList`** and **`JTable`**, models allow more complex functionality and interaction
  - Models can also be beneficial for interaction between components



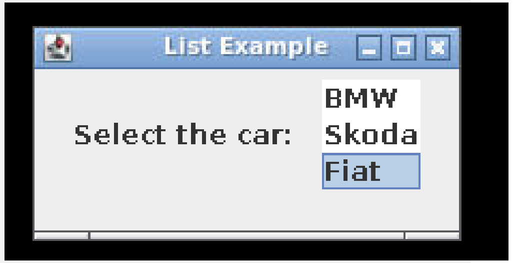




Simple example of using JList directly

- A `JList` instance presents the user with a group of items, displayed in one or more columns, to choose from

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class SimpleListExample {
  public static void main(String[] args) {
    JFrame frame = new JFrame("List Example");
    JPanel panel = new JPanel();
    JLabel label = new JLabel("Select the car:  ");
    String cars[]= {"BMW", "Skoda", "Fiat"};
    Jlist<String> list = new Jlist<>(cars);
    list.setSelectedIndex(2);
    panel.add(label);
    panel.add(list);
    frame.add(panel);
    frame.setSize(300,200);
    frame.setVisible(true);
  }
}
```

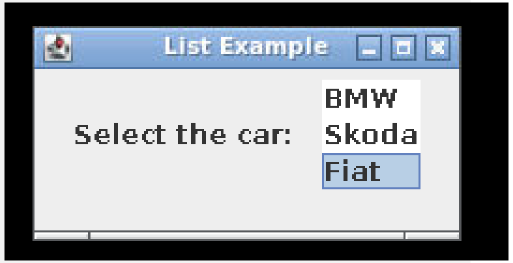


Note you need to define the type of items in `JList` (in this case, `String`)

Using JList directly with your own class

- You can store instances of your own class in `JList`, but you need to override `toString()` method to control how the items are displayed

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



Questions, comments?