CSE 1030

Yves Lespérance

Lecture Notes

Week 7 — Implementing Graphical User Interfaces

Recommended Readings:
Van Breugel & Roumani Ch. 6 and Savitch Ch. 17 and 18, and Sec. 13.2

Graphical User Interfaces

Graphical user interfaces (GUIs) allow the user to interract with an application in a more natural and efficient way.

To implement GUIs in Java, we generally use classes from the Java libraries, especially `javax.swing`. The classes in these libraries are complex, with many components and methods, and are organized in complex hierarchies.

The Model-View-Controller Pattern

Helps organize a GUI program.

The *model* represents the data and supports its manipulation.

The *view* specifies the graphical representation of the model.

The *controller* translates the user's interactions with the view (e.g. selecting a menu item, pressing a button, dragging the mouse, etc.) into actions on the model.

E.g. a window with dynamic menu, from Van Breugel & Roumani's.

Can define the `View` class as a subclass of `JFrame`.

A `JFrame` is a kind of window component. It has a `JMenuBar` and a `title` (inherited from parent `Frame`).

We can add `JMenu`s to the `JMenuBar`.

Then, we can add `JMenuItem`s to a `JMenu`.

Most GUI elements are `Component`s. Some components are `Container`s that can contain other components, e.g. a `Frame`.

Event-Driven Programming

User interactions with components of a GUI (e.g. selecting a menu item, pressing a button, draging the mouse, etc.) manifest themselves as *events*.

A program can set up *listeners* that watch for a particular class of events in a particular component, and execute an appropriate reaction when the event occurs, for instance to update the model and then the view.

To react to `ActionEvent`s (e.g. a menu item being selected), one adds action listeners to the associated components.

An action listener is an instance of a class that implements the `ActionListener` interface. To do this, it must provide the method

```
public void actionPerformed(ActionEvent event)
```

Can make the `Controller` class implement the `ActionListener` interface. It defines the `actionPerformed` method to handle the events we want to deal with. When such an event occurs it calls the appropriate method to update the model.

Commonly Used Component Classes

`JLabel`: used to display uneditable text; the text is generated by the program.

`JTextField`: used to display a boxed area where the user can enter text; an action event is triggered when the user hits “enter/return”.

`JPasswordField`: similar to `JTextField`, but text entered is not shown.

`JButton`: displays a boxed area which triggers an action event when clicked on.

`JCheckBox`: displays a box that is either selected or not selected; can react to event with `ItemListener`.

`JRadioButton`: like `JCheckBox`, but it does not allow multiple selections in a group of related buttons; can react to event with `ItemListener`.

Commonly Used Component Classes (cont.)

`JComboBox`: displays a drop-down list of items from which user can make a selection by clicking an item; react with `ItemListener`.

`JList`: displays a list of items, from which the user can select several by clicking the mouse once; double clicking an item generates an action event.

`JTextArea`: displays many lines of uneditable text; if the size of the text is larger than the `JTextArea`, scroll bars are automatically generated.

`JPanel`: a subcontainer, where GUI components can be put (more about this later).

See `GUIComponentsEg` which illustrates how many of these are used. Also shows different ways of defining event listeners and associating them with components.

Layout Managers

Organizing items/components on the screen requires layout managers. Java provides a variety of classes for this purpose, all of which implement the interface `LayoutManager`.

Layout managers automatically rearrange the layout according to their type when the window is resized!

Commonly Used Layout Managers

`FlowLayout`: components are placed left to right, row after row, in the order of addition.

`BorderLayout`: objects are placed in 5 possible places: North, S, W, E, or Center.

`GridLayout`: objects are placed in a grid/2-dimensional array; you specify desired number of rows & columns.

`CardDeckLayout`: objects are placed on different stacked “cards” of a deck.

Can use `JPanel` containers to make more complex GUIs. Each `JPanel` can have a different layout manager.

`GUIComponentsEg` illustrates use of `FlowLayout`.

See `GUIBorderLayoutEg` for example using `BorderLayout`.

`GUIPanelEg` illustrates use of `JPanel` to build more complex GUIs.

Drawing

Can also draw various shapes or display strings. Done by defining the component’s `void paint(Graphics g)` method.

`Graphics` class provides various methods for drawing.

`drawRect(int x, int y, int width, int height)`: draws an empty rectangle; (x,y) are the coordinates of the upper left corner; all arguments are in pixels.

`fillRect(int x, int y, int width, int height)`: draws a filled rectangle.

`drawOval(int x, int y, int width, int height)`: draws an oval.

`drawString(String str, int x, int y)`: draws a string; (x,y) is the baseline of the leftmost charcter.

Colours

Colours are represented by instances of the `Color` class. These are defined using RGB values. There are many constants for commonly used colours, e.g. `Color.blue`.

When drawing, can set the colour to be used using the `Graphics` class's `setColor(Color c)` method.

`GraphicsShowColorsEg` illustrates drawing and the use of colours..