CST8221 – Java Application Programming

Unit 5, Part 2- Actions and Toolbars

Swing provides two set of components that allow the programmer to attach menu-like system to their applications: menus and toolbars. In this unit we will discuss toolbars. Toolbars are a new addition to Swing. Toolbars allow the programmer to group buttons, combo boxes, and other components together; these tools can assist the user in performing many common tasks. You can add any component to a Swing toolbar, even non-Swing components. In addition, Swing allows the toolbar to be dragged from the frame and positioned inside a child window for convenience.

Introduction to Actions

As you have already experienced it is common to have multiple ways to activate the same command in an application GUI. For example, the user can choose certain functionality through a menu item, keystroke, or a button on a toolbar. One way to achieve this is to link the same listener to all event sources. Then the command will be handled in a uniform way, no matter whether it has been caused by a button click, a menu selection, or a key combination pressed.
The Swing API provides a very useful mechanism to encapsulate commands and to attach them to multiple event sources: the ***Action*** interface. An *action* object is an *ActionListener* object that encapsulates: A description of the command as text string and optional icon image; and parameters that are necessary to carry out the command. In other words, The Action interface provides a useful extension to the *ActionListener* interface in cases where the same functionality may be accessed by several controls.
The *Action* interface has the following methods:

```
void actionPerformed(ActionEvent e)
Object getValue(String key)
void putValue(String key, Object value)
boolean isEnabled()
void setEnabled(boolean b)
void addPropertyChangeListener(PropertyChangeListener listener)
void removePropertyChangeListener(PropertyChangeListener listener)
```

The first method is the familiar method of the *ActionListener* interface inherited by the *Action* interface. Therefore, you can use the *Action* object whenever an *ActionListener* object is required. When an action is disabled the options it represents will be grayed out.
The *putValue()* and *getValue()* methods allow you to store and retrieve arbitrary name/value pairs in the action object. There are a few predefined strings which can be used to store action commands, action names, mnemonic, tooltips, and icons. For example,

```
putValue(Action.ACTION_COMMAND_KEY, “Blue)
```

will set the action command of the component to “Blue”.

Note that Action is an interface, not a class. This means that any class implementing this interface must implement the seven methods of the interface. Fortunately, Swing provides a class called ***AbstractAction*** that implements all methods except for *actionPerformed().* That class takes care of storing all names/values pairs and managing the property change listeners. This means that all you need to do to create an action class is to extend *AbstractAction* and implement an *actionPerformed()* method. See the code example for how to create and use an *Action* object. Also visit the link below for more details
https://docs.oracle.com/javase/tutorial/uiswing/misc/action.html

Introduction to Swing Toolbars

A toolbar is a container (***JToolBar***) that groups several components (usually buttons with icons) into a row or column. Often, tool bars provide easy access to functionality that is also in menus. In this case the easiest way to create toolbars and menus with the same functionality is to use Action objects. The default layout manager of *JToolBar* is *BoxLayout*. The toolbar can be position vertically or horizontally in the west, east, north, south border sides of a frame. By default, the user can drag the tool bar to another side of its container or out into a window of its own. For the drag behavior to work correctly, the tool bar must be in a container that uses the *BorderLayout* layout manager. The component that the tool bar affects is generally in the center of the container. The tool bar must be the only other component in the container, and it must not be in the center. The "drag" feature of the toolbar is controlled by a method *setFloatable(boolean).* If the parameter of this method is false, the toolbar cannot be dragged and it will stay in one position only.

To program a toolbar involve a very simple process. Once you create a toolbar object

```
JToolBar bar = new JToolBar();
```

you have to populate toolbar by calling its *add()* method adding different components (usually action objects or buttons). If you want to separate a group of components, you can use the *addSeparator()* method. The final step is to add the toolbar to the frame, for example

```
frame.add(bar, BorderLayout.NORTH);
```

Buttons or actions are the most common components inside toolbars, but there is no restriction preventing you from adding other components like combo box or text field.

A disadvantage of the toolbars is that the user is often confused by the meaning of the fancy icons lined up in the toolbars. To solve the problem you should use tooltips.

The code example *ToolBarDemo* demonstrates how to crate toolbars using Actions. Follow the link below and you can find additional information about toolbars.

https://docs.oracle.com/javase/tutorial/uiswing/components/toolbar.html

Introduction to JavaFX Toolbars

The process of building application toolbars with JavaFX is strikingly similar. Of course, there are some differences. The most notable is that there is not an Action class in JavaFX. Also, the "floatable" behavior must be implemented in a different way. See the Unit 5 JavaFX toolbar code examples for details. Also visit the following links for more details:

http://www.java2s.com/Tutorials/Java/javafx.scene.control/ToolBar/0040__ToolBar.ToolBar_.htm

https://docs.oracle.com/javafx/2/ui_controls/ColorPickerSample.java.html

https://docs.oracle.com/javafx/2/ui_controls/color-picker.htm#BABHFGHA