CHAPTER 2

■ ■ ■

Swing Components

QUESTIONS AND EXERCISES

1. What is the superclass of all Swing components?

   Answer:

   `javax.swing.JComponent`

2. What class in Swing represents a button? What are the advantages of creating buttons in Swing using an `Action` object?

   Answer:

   An instance of the `javax.swing.JButton` class represents a button in Swing.

   `javax.swing.Action` is an interface. An `Action` encapsulates the state and the behavior of a button. You set the text, icon, mnemonic, tool tip text, other properties, and the `ActionListener` in an `Action`, and use the same `Action` to create many instances of the button. One obvious benefit of doing this is that if you want to enable/disable all buttons, you do not need to enable/disable all of them separately. Rather, you set the `enabled` property in the `Action` and it will enable/disable all buttons. Typically, you use an Action to create a menu item, a toolbar item, and a button where all three represent the same action but they are placed at different locations in the same application.

3. Describe a few uses of the `JPanel` Swing component.

   Answer:

   A `JPanel` is a container that can contain other components. You can set its layout manager, border, and background color. Typically, you use a `JPanel` to group related components

and add it to another container such as to the content pane of a `JFrame`. Note that a `JPanel` is a container, but not a top-level container, whereas a `JFrame` is a top-level container. Therefore, you cannot display a `JPanel` by itself in a Swing application, unless you add it to a top-level container. Sometimes, a `JPanel` is inserted between two components to create a gap. You can also use a `JPanel` as a canvas for drawing such as for drawing lines, rectangles, circles, etc. The default layout manager for a `JPanel` is `FlowLayout`.

4. What Swing component will you use to display an image?

   Answer:

   `javax.swing.JLabel`

5. Create a login form that lets users enter a user ID and a password. The form should have two buttons labeled `Login` and `Cancel`. When the `Login` button is clicked, a dialog box should be displayed to let the user know about the login status. The valid user ID and password are `jdk9` and `letmein`, respectively. The application should exit when the `Cancel` button is clicked.

   Solution:

```
// LoginFrame.java
package com.jdojo.swing.component.exercises;

import java.awt.Container;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.util.Arrays;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import static javax.swing.WindowConstants.EXIT_ON_CLOSE;

public class LoginFrame extends JFrame {
    private final JLabel uidLabel = new JLabel("User ID:");
    private final JLabel pwdLabel = new JLabel("Password:");

    private final JTextField uidField = new JTextField(20);
    private final JPasswordField pwdField = new JPasswordField(20);
```

```
    private final JButton loginBtn = new JButton("Login");
    private final JButton cancelBtn = new JButton("Cancel");

    public LoginFrame() {
        super("Login");
        this.initFrame();
    }

    private void initFrame() {
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container contentPane = this.getContentPane();
        contentPane.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);

        // Add components
        gbc.gridx = 0;
        gbc.gridy = 0;
        contentPane.add(uidLabel, gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;
        contentPane.add(uidField, gbc);

        gbc.gridx = 0;
        gbc.gridy = 1;
        contentPane.add(pwdLabel, gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        contentPane.add(pwdField, gbc);

        JPanel btnPanel = new JPanel();
        btnPanel.add(loginBtn);
        btnPanel.add(cancelBtn);
        gbc.gridx = 1;
        gbc.gridy = 2;
        gbc.anchor = GridBagConstraints.EAST;
        contentPane.add(btnPanel, gbc);

        // Add action listener to buttons
        loginBtn.addActionListener(this::login);
        cancelBtn.addActionListener(this::cancel);
    }

    private void login(ActionEvent e) {
        String correctUid = "jdk9";
        char[] correctPwd = "letmein".toCharArray();

        String uid = uidField.getText();
        char[] pwd = pwdField.getPassword();

        // Verify the user id and password
        if (uid.equalsIgnoreCase(correctUid) && Arrays.equals(pwd, correctPwd)) {
            JOptionPane.showMessageDialog(this, "Login successful.");
```

```
        } else {
            JOptionPane.showMessageDialog(this, "Invalid User ID/Password.");
        }
    }

    private void cancel(ActionEvent e) {
        System.exit(0);
    }

    public static void main(String[] args) {
        LoginFrame frame = new LoginFrame();
        frame.pack();
        frame.setVisible(true);
    }
}
```

6. Create a plain text editor like Notepad on Windows. The user should be able to load the contents of a text file in the editor, modify the contents, and save the changes to the file. Use menus, toolbars, `JTextArea`, and other Swing components to develop this editor.

   Solution:

   The following is the source code for a simple notepad that lets you edit a text file. You can enhance it as needed.

```
// MyNotepad.java
package com.jdojo.swing.component.exercises;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JTextArea;
import static javax.swing.WindowConstants.EXIT_ON_CLOSE;

public class MyNotepad extends JFrame {
    private final JTextArea contents = new JTextArea(30, 80);
```

```
    private File sourceFile;

    public MyNotepad() {
        super("Notepad");
        this.initFrame();
    }

    private void initFrame() {
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container contentPane = this.getContentPane();

        contentPane.add(contents);
        this.setJMenuBar(this.getAppMenuBar());
    }

    private JMenuBar getAppMenuBar() {
        // Construct the menu and its items
        JMenuItem openFile = new JMenuItem(this.getOpenFileAction());
        JMenuItem saveFile = new JMenuItem(this.getSaveFileAction());
        JMenuItem closeFile = new JMenuItem(this.getCloseFileAction());
        JMenuItem exitApp = new JMenuItem(this.getExitAppAction());

        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic('F');
        fileMenu.add(openFile);
        fileMenu.add(saveFile);
        fileMenu.add(closeFile);
        fileMenu.addSeparator();
        fileMenu.add(exitApp);

        JMenuBar menuBar = new JMenuBar();
        menuBar.add(fileMenu);

        return menuBar;
    }

    private Action getOpenFileAction() {
        class OpenFileAction extends AbstractAction {
            OpenFileAction(String action) {
                super(action);
                this.putValue(SHORT_DESCRIPTION, "Open a text file");
                this.putValue(MNEMONIC_KEY, KeyEvent.VK_O);
            }

            @Override
            public void actionPerformed(ActionEvent e) {
                openFile();
            }
        }

        return new OpenFileAction("Open");
    }
```

```
    private Action getSaveFileAction() {
        class SaveFileAction extends AbstractAction {
            SaveFileAction(String action) {
                super(action);
                this.putValue(SHORT_DESCRIPTION, "Save the contents");
                this.putValue(MNEMONIC_KEY, KeyEvent.VK_S);
            }

            @Override
            public void actionPerformed(ActionEvent e) {
                saveFile();
            }
        }

        return new SaveFileAction("Save");
    }

    private Action getCloseFileAction() {
        class CloseFileAction extends AbstractAction {
            CloseFileAction(String action) {
                super(action);
                this.putValue(SHORT_DESCRIPTION, "Close the file");
                this.putValue(MNEMONIC_KEY, KeyEvent.VK_C);
            }

            @Override
            public void actionPerformed(ActionEvent e) {
                closeFile();
            }
        }

        return new CloseFileAction("Close");
    }

    private Action getExitAppAction() {
        class ExitAppAction extends AbstractAction {
            ExitAppAction(String action) {
                super(action);
                this.putValue(SHORT_DESCRIPTION, "Exit application");
                this.putValue(MNEMONIC_KEY, KeyEvent.VK_E);
            }

            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        }

        return new ExitAppAction("Exit");
    }

    private void openFile() {
        JFileChooser fileChooser = new JFileChooser();
```

```
        int option = fileChooser.showOpenDialog(this);
        if (option != JFileChooser.APPROVE_OPTION) {
            return;
        }

        // Read the contents of the selected file into the text area
        File selectedFile = fileChooser.getSelectedFile();
        try (FileReader fr = new FileReader(selectedFile)) {
            contents.read(fr, null);

            // Save the source file for later use
            this.sourceFile = selectedFile;
            updateTitle();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void saveFile() {
        // If the contents are not from a file, prompt the user for a file
        if (this.sourceFile == null) {
            JFileChooser fileChooser = new JFileChooser();
            int option = fileChooser.showSaveDialog(this);
            if (option != JFileChooser.APPROVE_OPTION) {
                return;
            }
            sourceFile = fileChooser.getSelectedFile();
        }

        // Write the contents to the file
        try (FileWriter fw = new FileWriter(sourceFile)) {
            contents.write(fw);
            updateTitle();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void closeFile() {
        sourceFile = null;
        contents.setText("");
    }

    private void updateTitle() {
        String title = "Notepad" +
                (sourceFile == null ? "" : " - " + sourceFile.getAbsolutePath());
        this.setTitle(title);
    }
```

```
    public static void main(String[] args) {
        MyNotepad frame = new MyNotepad();
        frame.pack();
        frame.setVisible(true);
    }
}
```

7. What Swing component would you use to display HTML pages in your application?

   Answer:

   `javax.swing.JEditorPane`

8. What Swing component would you use to let users edit styled documents in RTF format?

   Answer:

   `javax.swing.JTextPane`

9. Swing provides the following components to let user select zero or more items from a list of items: `JToggleButton`, `JCheckBox`, `JRadioButton`, `JComboBox`, and `JList`. Which of these components are suitable to present multiple choice items such as hobbies? Which one of these components will you use if you have a list of 50 states and want to enable users to select only one?

   Answer:

   One of the following component can be used to present multiple choice items: `JToggleButton`, `JCheckBox`, and `JList`.

   If you have a list of 50 states to display from which only one may be selected by the user, you should use a `JComboBox`.

10. When do you need to use a `JScrollPane` component?

    Answer:

When you have a component bigger than the available space in the container, you display the component inside a `JScrollPane`. A `JScrollPane` provides a viewport in which you can view the component. You can use vertical and horizontal scroll bars provided by the `JScrollPane` to scroll through all parts of the components.

11. Suppose you have to let the user select an integer between 1 and 10. Which Swing component will you use, `JSpinner` or `JSlider`, provided you do not have a space constraint?

    Answer:

    Both will work. Because there is no space limitation, a `JSlider` is a better option.

12. When do you use a `JFileChooser` component?

    Answer:

    You use a `JFileChooser` to let the user select a file (or files) from the file system.

13. What is the fully qualified name of the class that represents a color in Swing applications?

    Answer:

    `java.awt.Color`

14. What is double buffering?

    Answer:

    Different techniques can be used to paint a component on the screen. If a component is painted directly on the screen, it is known as an onscreen painting. If a component is painted using an off-screen buffer and that buffer is copied on to the screen in one step, it is called double buffering.

15. Suppose that you are developing a `JFrame` that will let the users enter and save some data. The data must be saved before the `JFrame` is closed. What event will you capture to write this logic?

Answer:

You will need to handle the window closing event, like so:

```
// Use the WindowAdapter class to intercept only the window closing event
frame.addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("JFrame is closing.");
    }
});
```