FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 1 – SUPPORT PRESENTATION (PART 2)

Outline

- Graphical User Interfaces Development using Swing
- Java Serialization

Graphical User Interfaces Development using Swing

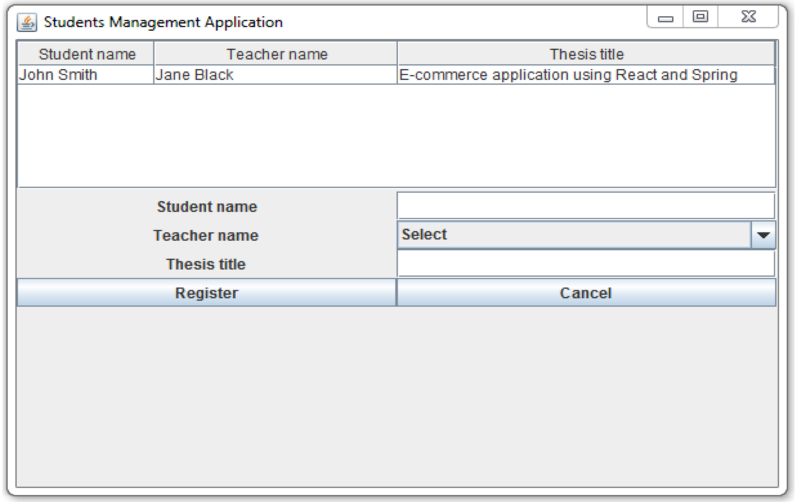
Graphical User Interfaces Development using Swing

- SWING API [Link]
  - Is part of the Java Foundation Classes (JFC)
  - Offers facilities to write applications with a graphical user interface
  - Includes 17 packages consisting of classes and interfaces
- **javax.swing** - Is the most important package from Swing

Component Type	**Examples**
Atomic components	JLabel, JButton, JCheckBox, JRadioButton, JToggleButton, JScrollBar, JSlider
Complex components	JTable, JTree, JComboBox, JList, JFileChooser, JColorChooser, JOptionPane
Text components	JTextField, JPasswordField, JTextArea, JEditorPane, JTextPane
Menus	JMenuBar, JMenu, JPopupMenu, JMenuItem, CheckboxMenuItem , JRadioButtonMenuItem
Intermediate containers	JPanel, JTabbedPane, JDesktopPane
Top level containers	JFrame, JDialog

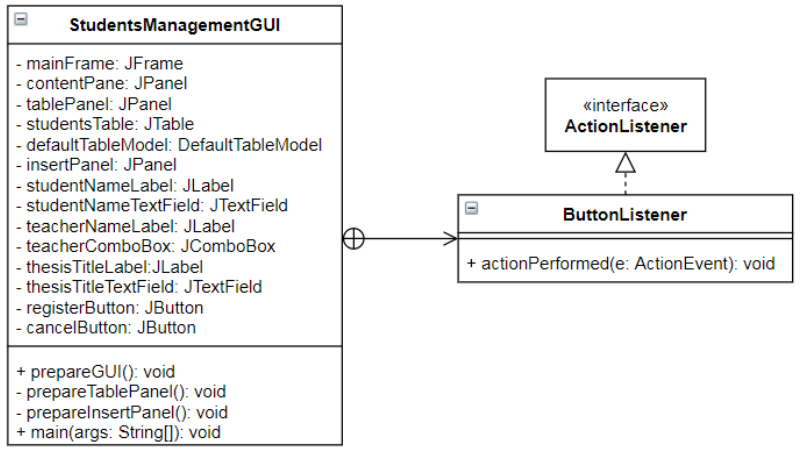
Graphical User Interfaces Development using Swing

- Example – students management application



Students Management Application

Student name	Teacher name	Thesis title
John Smith	Jane Black	E-commerce application using React and Spring



StudentsManagementGUI
- mainFrame: JFrame
- contentPane: JPanel
- tablePanel: JPanel
- studentsTable: JTable
- defaultTableModel: DefaultTableModel
- insertPanel: JPanel
- studentNameLabel: JLabel
- studentNameTextField: JTextField
- teacherNameLabel: JLabel
- teacherComboBox: JComboBox
- thesisTitleLabel:JLabel
- thesisTitleTextField: JTextField
- registerButton: JButton
- cancelButton: JButton
+ prepareGUI(): void
- prepareTablePanel(): void
- prepareInsertPanel(): void
+ main(args: String[]): void
«interface»
ActionListener
ButtonListener
+ actionPerformed(e: ActionEvent): void

GOOD TO KNOW – TOP-LEVEL CONTAINERS [Link]

The graphical components must be included in a containment hierarchy having a top-level container (e.g. JFrame, JDialog) as root. In particular, the graphical components will be contained in the content pane of the top-level container. A menu bar can be included in a top-level container, but it will reside outside the content pane. To create and set up a frame, the following steps should be performed:

- Create the frame by instantiating the *JFrame* class.
- Create components and add them to the frame's content pane.
- Size the frame manually (using the *setSize* method), or automatically (using the pack method).
- Show the frame onscreen (using the *setVisible* method).

To get the content pane of a JFrame component, the method getContentPane defined in the *JFrame* class is used. There are 2 approaches for setting the content pane of a JFrame component:

1) Use the method *getContentPane*() defined in the *JFrame* class to get the frame's content pane and add various components to it: **mainFrame.getContentPane().add(tablePanel);**

Note: mainframe.add(tablePanel) can also be used as the add method has been overridden and it actually adds tablePanel to the frame's content pane

2) Use the JFrame's setContentPane method to make another component the content pane of the frame:

```
JPanel contentPanePanel = new JPanel();
// add other graphical components to contentPanePane
  …
mainFrame.setContentPane(contentPanePanel);
```

Graphical User Interfaces Development using Swing

- Example – students management application

```
private void prepareGUI() {
        mainFrame = new JFrame("Students Management Application");
        mainFrame.setSize(500, 500);
        mainFrame.addWindowListener(new WindowAdapter() {
              @Override
               public void windowClosing(WindowEvent e) {
                    System.exit(0);
               }
          });
        contentPane = new JPanel(new GridLayout(2, 1));
        prepareTablePanel();
        prepareInsertPanel();
        mainFrame.setContentPane(contentPane);
        mainFrame.setVisible(true);
}
```

The *setSize* method is used to explicitly set the size of the frame.

Adds a window listener to receive window events from the *JFrame* component. In this case, once the frame is closed, the application is exited.

GridLayout is used as the *Layout Manager of* the *contentPane JPanel* which serves as the content pane of the frame. *GridLayout* origanizes the graphical components on rows and columns, setting the same size for all components.

The *setVisible(true)* method is used to make the frame appear onscreen.

The *contentPane JPanel* component is set as the content pane of the frame. All other graphical components will be contained in it.

```
private void prepareTablePanel(){
        tablePanel = new JPanel();
        tablePanel.setLayout(new BoxLayout(tablePanel, BoxLayout.PAGE_AXIS));
        defaultTableModel = new DefaultTableModel();
        defaultTableModel.addColumn("Student name");
        defaultTableModel.addColumn("Teacher name");
        defaultTableModel.addColumn("Thesis title");
        defaultTableModel.addRow(new Object[] {"John Smith", "Jane Black", "E-
                      commerce application using React and Spring"});
        studentsTable = new JTable(defaultTableModel);
        JScrollPane scrollPane = new JScrollPane(studentsTable);
        studentsTable.setFillsViewportHeight(true);
        this.tablePanel.add(scrollPane);
        contentPane.add(tablePanel);
    }
```

BoxLayout is used as the Layout Manager of tablePanel in order to organize the contained graphical components on top of each other.

A *DefaultTableModel* object uses a Vector of Vectors to store the cell value objects of a *JTable* component.

The *JScrollPane* object is used as a container for the *JTable Component* and automatically places the table's header at the top and they remain visible even when the table data is scrolled. If a *JScrollPane* is not used, then the table's header must be manually placed in the container.

The *tablePanel* is added to the content pane of the frame.

Graphical User Interfaces Development using Swing

- Layout Managers are used to organize graphical components in containers. The following Layout Managers can be used [Link]:

a) BorderLayout – places the components in 5 areas: top, bottom, left, right, and centre.

b) BoxLayout – places the components on a row or on a column.

c) CardLayout – enables the implementation of an area that contains different components at different times.

d) FlowLayout – places the components in a single row.

e) GridBagLayout – places the components in a grid of cells, allowing the spanning and sizing of components over multiple cells.

f) GridLayout – sets equal sizes for the components and places them in the requested number of rows and columns.

Java Serialization

Java Serialization

- Persist an object state even after the program is not running
- Object whose class implements the Serializable interface
  - Into a sequence of bytes that can be written to disk and later restored to recreate the original object
- Mechanism for implementing a lightweight persistence
  - The user must explicitly serialize and deserialize the objects in a program
- A serialized object can be transmitted over the network
- The transient keyword can be used to turn off serialization for a field
- Static fields are not serializable

Java Serialization

Serialization / **Deserialization**

```
public class SerializationOperations {
    public static void main(String[] args) throws ParseException,
                        IOException, ClassNotFoundException {

        FileOutputStream fileOutputStream = new
                        FileOutputStream("john_doe.txt");
        ObjectOutputStream objectOutputStream = new
                         ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(user);
        objectOutputStream.flush();
        objectOutputStream.close();

        FileInputStream fileInputStream = new
                                FileInputStream("john_doe.txt");
        ObjectInputStream objectInputStream = new
                             ObjectInputStream(fileInputStream);
        User restoredUser = (User) objectInputStream.readObject();
        objectInputStream.close();
        System.out.println(restoredUser.toString());
    }
}
```

Used to remember versions of the Serializable class to verify that a loaded class and the serialized object are compatible.

```
public class User implements Serializable {
    private static final long
            serialVersionUID = 1L;
    private String username;
    private transient String password;
    private String name;
    private LocalDate birthdate;
    // get and set methods…
```