PROGRAMMING IN JAVA

Lecture 13

Java GUI Programming and Event-Driven Design

Swing Components, Layout Managers, Events, Dialogs, Tables, Menus, Desktop Application Design, and Historical Applet Concepts

Lecturer: Arshyn Altybay

Department of Computer Engineering,
Almaty Technological University, Almaty, Kazakhstan

1. Learning Objectives

After completing this lecture, students should be able to:

1. explain the difference between console and graphical applications;
2. explain event-driven programming;
3. distinguish AWT from Swing;
4. create a top-level Swing window with `JFrame`;
5. use common components such as `JLabel`, `JButton`, and `JTextField`;
6. organize components using layout managers;
7. respond to user actions using listeners and lambda expressions;
8. use panels to create modular interfaces;
9. work with check boxes, radio buttons, combo boxes, and lists;
10. display message and input dialogs;
11. build menus and menu items;
12. display tabular information using `JTable`;
13. use `JFileChooser`;
14. explain the role of the Swing Event Dispatch Thread;
15. structure a small desktop application using separation of concerns;
16. explain why Java applets are now a historical topic rather than a current deployment technology.

2. Introduction to Graphical User Interfaces

A console application communicates mainly through textual input and output:

Illustration

```
User <--> Console <--> Program
```

A graphical user interface (GUI) uses visual components such as:

- windows;
- buttons;
- labels;
- text fields;
- lists;
- tables;
- menus;
- dialogs.

Instead of executing strictly from top to bottom and waiting only for console input, a GUI program spends much of its time waiting for **events**.

3. Event-Driven Programming

An **event** is an action or occurrence detected by the program.

Examples include:

- a button click;
- keyboard input;
- mouse movement;
- selection of a menu item;
- closing a window;
- selecting an item from a list.

The general model is:

Illustration

```
User action
    |
    v
```

```
Event generated
    |
    v
Listener receives event
    |
    v
Application executes handler code
```

This approach is called **event-driven programming**.

4. AWT and Swing

Java provides several GUI-related APIs.

The older Abstract Window Toolkit is known as **AWT**.

Swing was built on top of AWT and provides a richer set of desktop GUI components.

Important packages include:

```
java.awt
java.awt.event
javax.swing
```

Examples of Swing classes:

- `JFrame`
- `JPanel`
- `JLabel`
- `JButton`
- `JTextField`
- `JTextArea`
- `JCheckBox`
- `JRadioButton`
- `JComboBox`
- `JList`
- `JTable`
- `JMenuBar`
- `JOptionPane`

5. AWT vs Swing

Feature	AWT	Swing
Package	`java.awt`	`javax.swing`
Component style	more native-peer based	mostly lightweight Java components
Component set	basic	richer
Top-level window	`Frame`	`JFrame`
Button	`Button`	`JButton`
Label	`Label`	`JLabel`

Swing still uses important AWT concepts such as layout managers, events, colors, fonts, and graphics.

6. The Swing Event Dispatch Thread

Swing is primarily single-threaded.

Most Swing component creation and updates should occur on the **Event Dispatch Thread (EDT)**.

A common program entry pattern is:

```
SwingUtilities.invokeLater(() -> {
    // create and show GUI
});
```

This schedules GUI construction on the EDT.

Long-running work should not block the EDT because the interface would become unresponsive.

Example 1. First Swing Window

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "My First GUI"
                );

        frame.setSize(400, 250);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

This program creates a simple desktop window.

Important operations:

- `setSize()` sets width and height;
- `setDefaultCloseOperation()` defines close behavior;
- `setLocationRelativeTo(null)` centers the window;
- `setVisible(true)` displays it.

7. JLabel

A `JLabel` displays non-editable text or an icon.

Example:

```
JLabel label =
        new JLabel(
            "Welcome to Java!"
        );
```

Labels are commonly used for:

- field descriptions;
- titles;
- status messages;
- results.

Example 2. Window with a Label

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import javax.swing.SwingConstants;
4  import javax.swing.SwingUtilities;
5
6  public class Main {
7
8      private static void createGui() {
9
10         JFrame frame =
11                 new JFrame(
12                     "Label Example"
13                 );
14
15         JLabel label =
16                 new JLabel(
17                     "Programming in Java",
18                     SwingConstants.CENTER
19                 );
20
21         frame.add(label);
22
23         frame.setSize(400, 200);
24
25         frame.setDefaultCloseOperation(
```

Example 2 (continued)

```
26              JFrame.EXIT_ON_CLOSE
27          );
28
29          frame.setLocationRelativeTo(null);
30          frame.setVisible(true);
31      }
32
33      public static void main(String[] args) {
34
35          SwingUtilities.invokeLater(
36              Main::createGui
37          );
38      }
39  }
```

8. JButton and Action Events

A `JButton` represents a clickable button.

Example:

```
JButton button =
        new JButton("Click Me");
```

A button normally generates an `ActionEvent`.

A listener can be attached using:

```
button.addActionListener(
    event -> {
        // handle click
    }
);
```

Example 3. Button Click Counter

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import java.awt.FlowLayout;

public class Main {

    private static int count = 0;

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Counter"
                );

        JButton button =
                new JButton(
                    "Clicked 0 times"
                );

        button.addActionListener(
            event -> {

                count++;

                button.setText(
                    "Clicked " +
                    count +
                    " times"
                );
            }
        );

        frame.setLayout(
            new FlowLayout()
        );

        frame.add(button);

        frame.setSize(300, 150);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
```

Example 3 (continued)

```
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

9. JTextField

A `JTextField` is used for single-line text input.

Example:

```
JTextField field =
        new JTextField(20);
```

Useful methods include:

- `getText()`
- `setText()`
- `setEditable()`

Example 4. Greeting Form

```
import javax.swing.*;
import java.awt.FlowLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Greeting"
```

Example 4 (continued)

```
                );

        JLabel nameLabel =
                new JLabel("Name:");

        JTextField nameField =
                new JTextField(15);

        JButton button =
                new JButton("Greet");

        JLabel result =
                new JLabel(" ");

        button.addActionListener(
            event -> {

                String name =
                    nameField
                        .getText()
                        .trim();

                if (name.isEmpty()) {
                    result.setText(
                        "Enter a name."
                    );
                } else {
                    result.setText(
                        "Hello, " +
                        name + "!"
                    );
                }
            }
        );

        frame.setLayout(
            new FlowLayout()
        );

        frame.add(nameLabel);
        frame.add(nameField);
        frame.add(button);
        frame.add(result);

        frame.setSize(350, 180);
```

Example 4 (continued)

```
56
57          frame.setDefaultCloseOperation(
58              JFrame.EXIT_ON_CLOSE
59          );
60
61          frame.setLocationRelativeTo(null);
62          frame.setVisible(true);
63      }
64
65      public static void main(String[] args) {
66
67          SwingUtilities.invokeLater(
68              Main::createGui
69          );
70      }
71  }
```

10. JTextArea and JScrollPane

A `JTextArea` supports multi-line text.

Example:

```
JTextArea area =
        new JTextArea(10, 30);
```

A text area is often placed inside a scroll pane:

```
JScrollPane scrollPane =
        new JScrollPane(area);
```

Example 5. Simple Notes Window

```
1  import javax.swing.*;
2
3  public class Main {
4
5      private static void createGui() {
6
7          JFrame frame =
```

Example 5 (continued)

```
                new JFrame("Notes");

        JTextArea area =
                new JTextArea(
                    15, 40
                );

        area.setLineWrap(true);
        area.setWrapStyleWord(true);

        frame.add(
            new JScrollPane(area)
        );

        frame.pack();

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

11. Layout Managers

A layout manager controls component positioning and sizing.

Using layout managers is normally better than fixed coordinates.

Important layout managers include:

- `FlowLayout`
- `BorderLayout`
- `GridLayout`

- `BoxLayout`
- `CardLayout`

12. FlowLayout

`FlowLayout` places components from left to right and moves to a new row when necessary.

It is simple and useful for small toolbars and button groups.

```
panel.setLayout(
    new FlowLayout()
);
```

13. BorderLayout

`BorderLayout` divides a container into five regions:

Illustration

```
        NORTH
WEST    CENTER    EAST
        SOUTH
```

Example:

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

frame.add(
    form,
    BorderLayout.CENTER
);
```

Example 6. BorderLayout

```
import javax.swing.*;
import java.awt.BorderLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "BorderLayout"
                );

        frame.add(
            new JButton("North"),
            BorderLayout.NORTH
        );

        frame.add(
            new JButton("South"),
            BorderLayout.SOUTH
        );

        frame.add(
            new JButton("West"),
            BorderLayout.WEST
        );

        frame.add(
            new JButton("East"),
            BorderLayout.EAST
        );

        frame.add(
            new JButton("Center"),
            BorderLayout.CENTER
        );

        frame.setSize(450, 250);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
```

Example 6 (continued)

```
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

14. GridLayout

`GridLayout` arranges components in equally sized rows and columns.

Example:

```
new GridLayout(3, 2)
```

This is useful for simple forms.

Example 7. Login Form

```
import javax.swing.*;
import java.awt.GridLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame("Login");

        JPanel panel =
                new JPanel(
                    new GridLayout(
                        3, 2, 8, 8
                    )
                );

        JTextField username =
                new JTextField();
```

Example 7 (continued)

```
20
21          JPasswordField password =
22                  new JPasswordField();
23
24          JButton login =
25                  new JButton("Login");
26
27          JLabel status =
28                  new JLabel(" ");
29
30          panel.add(
31              new JLabel("Username:")
32          );
33
34          panel.add(username);
35
36          panel.add(
37              new JLabel("Password:")
38          );
39
40          panel.add(password);
41
42          panel.add(login);
43          panel.add(status);
44
45          login.addActionListener(
46              event -> {
47
48                  if (username
49                          .getText()
50                          .equals("admin")) {
51
52                      status.setText(
53                          "Accepted"
54                      );
55
56                  } else {
57
58                      status.setText(
59                          "Rejected"
60                      );
61                  }
62              }
63          );
64
```

Example 7 (continued)

```
65          frame.add(panel);
66          frame.pack();
67
68          frame.setDefaultCloseOperation(
69              JFrame.EXIT_ON_CLOSE
70          );
71
72          frame.setLocationRelativeTo(null);
73          frame.setVisible(true);
74      }
75
76      public static void main(String[] args) {
77
78          SwingUtilities.invokeLater(
79              Main::createGui
80          );
81      }
82  }
```

15. Using `JPanel`

A `JPanel` is a general-purpose container.

Panels are essential for building modular interfaces.

A frame may contain:

Illustration

```
JFrame
  |
  +-- topPanel
  |
  +-- formPanel
  |
  +-- buttonPanel
  |
  +-- statusPanel
```

Each panel can have its own layout manager.

16. Check Boxes

A `JCheckBox` represents an independent on/off option.

Example:

```
JCheckBox javaBox =
        new JCheckBox("Java");
```

To check its state:

```
javaBox.isSelected()
```

17. Radio Buttons and `ButtonGroup`

Radio buttons are used when the user should select one option from a group.

Example:

```
JRadioButton male =
        new JRadioButton("Male");

JRadioButton female =
        new JRadioButton("Female");

ButtonGroup group =
        new ButtonGroup();

group.add(male);
group.add(female);
```

The `ButtonGroup` makes the choices mutually exclusive.

Example 8. Course Selection

```
1 import javax.swing.*;
2 import java.awt.FlowLayout;
3
4 public class Main {
5
6     private static void createGui() {
7
```

Example 8 (continued)

```
8           JFrame frame =
9                   new JFrame(
10                      "Course Selection"
11                  );
12
13          JCheckBox java =
14                  new JCheckBox("Java");
15
16          JCheckBox database =
17                  new JCheckBox(
18                      "Databases"
19                  );
20
21          JButton button =
22                  new JButton("Show");
23
24          JLabel result =
25                  new JLabel(" ");
26
27          button.addActionListener(
28              event -> {
29
30                  String text = "Selected:";
31
32                  if (java.isSelected()) {
33                      text += " Java";
34                  }
35
36                  if (database.isSelected()) {
37                      text += " Databases";
38                  }
39
40                  result.setText(text);
41              }
42          );
43
44          frame.setLayout(
45              new FlowLayout()
46          );
47
48          frame.add(java);
49          frame.add(database);
50          frame.add(button);
51          frame.add(result);
52
```

Example 8 (continued)

```
53          frame.setSize(400, 160);
54
55          frame.setDefaultCloseOperation(
56              JFrame.EXIT_ON_CLOSE
57          );
58
59          frame.setLocationRelativeTo(null);
60          frame.setVisible(true);
61      }
62
63      public static void main(String[] args) {
64
65          SwingUtilities.invokeLater(
66              Main::createGui
67          );
68      }
69  }
```

18. JComboBox

A combo box allows one item to be selected from a drop-down list.

Example:

```
String[] courses = {
    "Java",
    "Databases",
    "Networks"
};

JComboBox<String> combo =
        new JComboBox<>(courses);
```

To obtain the selected item:

```
String value =
        (String)
        combo.getSelectedItem();
```

Example 9. Combo Box

```
import javax.swing.*;
import java.awt.FlowLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Course"
                );

        JComboBox<String> combo =
                new JComboBox<>(
                    new String[] {
                        "Java",
                        "Databases",
                        "Networks"
                    }
                );

        JLabel result =
                new JLabel(
                    "Java"
                );

        combo.addActionListener(
            event -> {

                result.setText(
                    String.valueOf(
                        combo
                            .getSelectedItem()
                    )
                );
            }
        );

        frame.setLayout(
            new FlowLayout()
        );

        frame.add(combo);
        frame.add(result);
```

Example 9 (continued)

```
45
46          frame.setSize(350, 150);
47
48          frame.setDefaultCloseOperation(
49              JFrame.EXIT_ON_CLOSE
50          );
51
52          frame.setLocationRelativeTo(null);
53          frame.setVisible(true);
54      }
55
56      public static void main(String[] args) {
57
58          SwingUtilities.invokeLater(
59              Main::createGui
60          );
61      }
62  }
```

19. JList

A `JList<E>` displays a list of values.

Example:

```
JList<String> list =
        new JList<>(
            new String[] {
                "Aruzhan",
                "Dias",
                "Ali"
            }
        );
```

It can support single or multiple selection.

20. Dialogs with `JOptionPane`

`JOptionPane` provides simple standard dialogs.

Message dialog:

```
JOptionPane.showMessageDialog(
    null,
    "Operation completed."
);
```

Input dialog:

```
String name =
    JOptionPane.showInputDialog(
        "Enter your name:"
    );
```

Confirmation dialog:

```
int result =
    JOptionPane.showConfirmDialog(
        null,
        "Delete record?"
    );
```

Example 10. Input and Message Dialog

```
import javax.swing.JOptionPane;

public class Main {

    public static void main(String[] args) {

        String name =
            JOptionPane.showInputDialog(
                "Enter your name:"
            );

        if (name != null) {

            JOptionPane
                .showMessageDialog(
                    null,
                    "Hello, " +
                    name + "!"
                );
        }
    }
}
```

21. Menus

Important Swing menu classes include:

- `JMenuBar`
- `JMenu`
- `JMenuItem`
- `JCheckBoxMenuItem`
- `JRadioButtonMenuItem`

Example 11. Simple Menu

```
import javax.swing.*;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Menu Example"
                );

        JMenuBar bar =
                new JMenuBar();

        JMenu fileMenu =
                new JMenu("File");

        JMenuItem exitItem =
                new JMenuItem("Exit");

        exitItem.addActionListener(
            event ->
                frame.dispose()
        );

        fileMenu.add(exitItem);
        bar.add(fileMenu);

        frame.setJMenuBar(bar);

        frame.setSize(400, 250);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
```

Example 11 (continued)

```
        );
    }
}
```

22. JTable

A `JTable` displays data in rows and columns.

Simple example data:

```
String[] columns = {
    "ID",
    "Name",
    "GPA"
};

Object[][] data = {
    {1, "Aruzhan", 3.9},
    {2, "Dias", 3.5}
};
```

Create a table:

```
JTable table =
        new JTable(
            data,
            columns
        );
```

Tables are normally placed inside a `JScrollPane`.

Example 12. Student Table

```
import javax.swing.*;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
```

Example 12 (continued)

```
                    "Students"
                );

        String[] columns = {
            "ID",
            "Name",
            "GPA"
        };

        Object[][] data = {
            {1, "Aruzhan", 3.9},
            {2, "Dias", 3.5},
            {3, "Ali", 3.7}
        };

        JTable table =
                new JTable(
                    data,
                    columns
                );

        frame.add(
            new JScrollPane(table)
        );

        frame.setSize(450, 220);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

23. Table Models

For real applications, a table is usually backed by a model.

A common class is:

```
DefaultTableModel
```

For more control, a developer may extend:

```
AbstractTableModel
```

The model approach separates the data representation from the visual table.

Example 13. Adding Rows with `DefaultTableModel`

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.BorderLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Student Table"
                );

        DefaultTableModel model =
            new DefaultTableModel(
                new String[] {
                    "ID",
                    "Name",
                    "GPA"
                },
                0
            );

        JTable table =
                new JTable(model);

        JButton add =
                new JButton(
                    "Add Example"
```

Example 13 (continued)

```
                );

        add.addActionListener(
            event -> {

                model.addRow(
                    new Object[] {
                        1001,
                        "Aruzhan",
                        3.9
                    }
                );
            }
        );

        frame.add(
            new JScrollPane(table),
            BorderLayout.CENTER
        );

        frame.add(
            add,
            BorderLayout.SOUTH
        );

        frame.setSize(450, 250);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

24. JFileChooser

A `JFileChooser` allows users to select files using a standard graphical dialog.

Example:

```
JFileChooser chooser =
        new JFileChooser();

int result =
        chooser.showOpenDialog(frame);
```

If the user approves:

```
if (result ==
        JFileChooser.APPROVE_OPTION) {

    File file =
        chooser.getSelectedFile();
}
```

Example 14. Open a Text File

```
1  import javax.swing.*;
2  import java.io.File;
3  import java.io.IOException;
4  import java.nio.file.Files;
5
6  public class Main {
7
8      private static void createGui() {
9
10         JFrame frame =
11                 new JFrame(
12                     "Text Viewer"
13                 );
14
15         JTextArea area =
16                 new JTextArea();
17
18         JButton open =
19                 new JButton("Open");
20
21         open.addActionListener(
```

Example 14 (continued)

```
22              event -> {
23
24                  JFileChooser chooser =
25                          new JFileChooser();
26
27                  int result =
28                      chooser.showOpenDialog(
29                          frame
30                      );
31
32                  if (result ==
33                      JFileChooser
34                          .APPROVE_OPTION) {
35
36                      File file =
37                          chooser
38                              .getSelectedFile();
39
40                      try {
41
42                          area.setText(
43                              Files.readString(
44                                  file.toPath()
45                              )
46                          );
47
48                      } catch (
49                              IOException e) {
50
51                          JOptionPane
52                              .showMessageDialog(
53                                  frame,
54                                  "Cannot read file."
55                              );
56                      }
57                  }
58              }
59          );
60
61          frame.add(
62              new JScrollPane(area)
63          );
64
65          frame.add(
66              open,
```

Example 14 (continued)

```
67              java.awt.BorderLayout.SOUTH
68          );
69
70          frame.setSize(550, 350);
71
72          frame.setDefaultCloseOperation(
73              JFrame.EXIT_ON_CLOSE
74          );
75
76          frame.setLocationRelativeTo(null);
77          frame.setVisible(true);
78      }
79
80      public static void main(String[] args) {
81
82          SwingUtilities.invokeLater(
83              Main::createGui
84          );
85      }
86  }
```

25. Mouse and Keyboard Events

Java provides many listener interfaces.

Examples include:

- `ActionListener`
- `MouseListener`
- `MouseMotionListener`
- `KeyListener`
- `WindowListener`
- `ItemListener`

For many high-level Swing controls, action-oriented listeners are preferable to low-level keyboard or mouse handling.

26. Adapter Classes

Some listener interfaces contain several methods.

Adapter classes provide empty implementations, allowing the programmer to override only the needed methods.

Examples:

- `MouseAdapter`
- `WindowAdapter`
- `KeyAdapter`

Example 15. Mouse Event

```
import javax.swing.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Mouse Example"
                );

        JLabel label =
                new JLabel(
                    "Click here",
                    SwingConstants.CENTER
                );

        label.addMouseListener(
            new MouseAdapter() {

                @Override
                public void mouseClicked(
                        MouseEvent e) {

                    label.setText(
                        "Clicked at " +
                        e.getX() +
                        ", " +
                        e.getY()
```

Example 15 (continued)

```
                    );
                }
            }
        );

        frame.add(label);

        frame.setSize(400, 250);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

27. Validation in GUI Applications

GUI input must be validated just like console input.

Example:

```
String text =
        gpaField
            .getText()
            .trim();

try {

    double gpa =
        Double.parseDouble(text);

    if (gpa < 0 || gpa > 4) {
```

```
            throw new
                IllegalArgumentException(
                    "GPA must be 0..4"
                );
        }

    } catch (
            NumberFormatException e) {

        JOptionPane.showMessageDialog(
            frame,
            "Enter a valid number."
        );
    }
```

The interface should provide clear feedback rather than terminate.

28. Separation of Concerns in GUI Design

A common beginner mistake is to put all logic into one `JFrame` class.

A better organization separates:

- domain objects;
- application/service logic;
- persistence;
- graphical presentation.

Example:

Illustration

```
model/
  Student.java

service/
  StudentService.java

repository/
  StudentRepository.java

ui/
  StudentFrame.java
```

```
app/
  Main.java
```

The GUI should coordinate user interaction, not contain every business rule.

Example 16. Small Student Registration GUI

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.BorderLayout;
import java.awt.GridLayout;

public class Main {

    private static void createGui() {

        JFrame frame =
                new JFrame(
                    "Student Registration"
                );

        JTextField idField =
                new JTextField();

        JTextField nameField =
                new JTextField();

        JTextField gpaField =
                new JTextField();

        JPanel form =
                new JPanel(
                    new GridLayout(
                        3, 2, 8, 8
                    )
                );

        form.add(
            new JLabel("ID:")
        );
        form.add(idField);

        form.add(
            new JLabel("Name:")
        );
        form.add(nameField);
```

Example 16 (continued)

```
40
41          form.add(
42              new JLabel("GPA:")
43          );
44          form.add(gpaField);
45
46          DefaultTableModel model =
47              new DefaultTableModel(
48                  new String[] {
49                      "ID",
50                      "Name",
51                      "GPA"
52                  },
53                  0
54              );
55
56          JTable table =
57                  new JTable(model);
58
59          JButton addButton =
60                  new JButton(
61                      "Add Student"
62                  );
63
64          addButton.addActionListener(
65              event -> {
66
67                  try {
68
69                      int id =
70                          Integer.parseInt(
71                              idField
72                                  .getText()
73                                  .trim()
74                          );
75
76                      String name =
77                          nameField
78                              .getText()
79                              .trim();
80
81                      double gpa =
82                          Double.parseDouble(
83                              gpaField
84                                  .getText()
```

Example 16 (continued)

```
                                .trim()
                        );

                    if (name.isEmpty()) {

                        throw new
                            IllegalArgumentException(
                                "Name is required."
                            );
                    }

                    if (gpa < 0 ||
                        gpa > 4) {

                        throw new
                            IllegalArgumentException(
                                "GPA must be 0..4."
                            );
                    }

                    model.addRow(
                        new Object[] {
                            id,
                            name,
                            gpa
                        }
                    );

                    idField.setText("");
                    nameField.setText("");
                    gpaField.setText("");

                } catch (
                    NumberFormatException e) {

                    JOptionPane
                        .showMessageDialog(
                            frame,
                            "ID and GPA must be numeric."
                        );

                } catch (
                    IllegalArgumentException e) {

                    JOptionPane
```

Example 16 (continued)

```
                        .showMessageDialog(
                            frame,
                            e.getMessage()
                        );
                }
            }
        );

        frame.add(
            form,
            BorderLayout.NORTH
        );

        frame.add(
            new JScrollPane(table),
            BorderLayout.CENTER
        );

        frame.add(
            addButton,
            BorderLayout.SOUTH
        );

        frame.setSize(550, 350);

        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE
        );

        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(
            Main::createGui
        );
    }
}
```

This example combines:

- forms;
- layout managers;

- button events;
- input validation;
- exception handling;
- table models;
- dialogs.

29. Long-Running Work and GUI Responsiveness

A long task executed directly inside an event handler can freeze the user interface.

Example of a poor idea:

```
button.addActionListener(
    event -> {
        // very long calculation here
    }
);
```

The EDT cannot process repainting or additional input while it is blocked.

For long-running tasks, Swing provides tools such as:

```
SwingWorker
```

General multithreading is studied in the next lecture.

Example 17. Basic `SwingWorker` Idea

```
1  import javax.swing.*;
2
3  public class Main {
4
5      private static void createGui() {
6
7          JFrame frame =
8                  new JFrame(
9                      "Background Task"
10                 );
11
12         JButton button =
13                 new JButton(
14                     "Start"
15                 );
```

Example 17 (continued)

```
16
17          JLabel status =
18                  new JLabel(
19                      "Ready"
20                  );
21
22          button.addActionListener(
23              event -> {
24
25                  button.setEnabled(false);
26                  status.setText(
27                      "Working..."
28                  );
29
30                  SwingWorker<String, Void>
31                      worker =
32                      new SwingWorker<>() {
33
34                      @Override
35                      protected String
36                      doInBackground()
37                              throws Exception {
38
39                          Thread.sleep(2000);
40
41                          return "Completed";
42                      }
43
44                      @Override
45                      protected void done() {
46
47                          try {
48
49                              status.setText(
50                                  get()
51                              );
52
53                          } catch (
54                              Exception e) {
55
56                              status.setText(
57                                  "Failed"
58                              );
59                          }
60
```

Example 17 (continued)

```
61                          button.setEnabled(true);
62                      }
63                  };
64
65                  worker.execute();
66              }
67          );
68
69          JPanel panel =
70                  new JPanel();
71
72          panel.add(button);
73          panel.add(status);
74
75          frame.add(panel);
76
77          frame.setSize(350, 150);
78
79          frame.setDefaultCloseOperation(
80              JFrame.EXIT_ON_CLOSE
81          );
82
83          frame.setLocationRelativeTo(null);
84          frame.setVisible(true);
85      }
86
87      public static void main(String[] args) {
88
89          SwingUtilities.invokeLater(
90              Main::createGui
91          );
92      }
93  }
```

30. Historical Topic: Java Applets

An **applet** was a small Java program designed to run inside an applet container, historically often associated with web browsers.

A traditional applet used a lifecycle such as:

- initialization;
- start;
- execution;
- stop;
- destruction.

Historically, classes such as:

Illustration

```
java.applet.Applet
javax.swing.JApplet
```

were used.

30.1 Current Status

Applets must now be treated only as a historical curriculum topic.

Modern web browsers do not support the old Java browser plug-in model.

The Java deployment stack, including the browser plug-in and Applet Viewer, was removed from the JDK earlier, and the Applet API itself was removed in JDK 26.

Therefore modern JDK 26 code cannot compile against `java.applet.Applet` or `javax.swing.JApplet` because those APIs are no longer part of the platform.

There is no direct replacement for the Applet API.

31. Historical Applet Structure

For historical understanding only, old source code commonly looked conceptually like this:

Illustration

```
public class HelloApplet extends Applet {

```

```
    public void init() {
        // initialization
    }

    public void start() {
        // applet becomes active
    }

    public void stop() {
        // applet becomes inactive
    }

    public void destroy() {
        // cleanup
    }
}
```

Important: This is a historical illustration only. It is not intended to compile on current JDK 26 installations.

For practical GUI development in this course, use Swing desktop applications.

32. Swing Desktop Applications vs Historical Applets

Feature	Swing Desktop Application	Historical Applet
Entry model	`main()` and window creation	applet lifecycle methods
Runs independently	Yes	No, required applet environment
Modern JDK support	Yes	No
Browser plug-in required	No	Historically yes
Recommended for practice	Yes	No

33. GUI Design Guidelines

When designing a Swing application:

1. construct and update Swing components on the EDT;
2. use layout managers rather than hard-coded coordinates;
3. divide large windows into panels;
4. validate all user input;
5. show clear error messages;
6. keep business logic outside GUI classes when practical;
7. use models for tables and other data-rich components;
8. avoid long-running work on the EDT;
9. use meaningful labels and button names;
10. keep the interface consistent and simple.

34. Common Mistakes

34.1 Mistake 1: Using `null` Layout Everywhere

Absolute positioning may break when:

- fonts change;
- window size changes;
- operating system appearance changes;
- text length changes.

Prefer layout managers.

34.2 Mistake 2: Putting All Code in One JFrame Class

Separate domain, service, persistence, and UI responsibilities.

34.3 Mistake 3: Ignoring Input Validation

Text fields contain strings. Numeric conversion can fail.

34.4 Mistake 4: Blocking the EDT

Long calculations or file operations can freeze the GUI.

34.5 Mistake 5: Forgetting the Close Operation

A frame should usually specify an appropriate close behavior.

34.6 Mistake 6: Treating Applets as Current Java Technology

Applets are obsolete and removed from current JDK 26. They should only be studied historically.

35. Summary

In this lecture, we learned that:

- GUI applications are event-driven;
- Swing provides Java desktop GUI components;
- `JFrame` represents a top-level Swing window;
- labels, buttons, text fields, and text areas are basic components;
- layout managers organize components;
- listeners respond to events;
- panels support modular window construction;
- dialogs provide convenient user interaction;
- tables display structured row-column data;
- menus provide application commands;
- file choosers support graphical file selection;
- Swing code should respect the Event Dispatch Thread;
- long-running work should not block the EDT;
- applets are now historical and are removed from JDK 26.

36. Review Questions

1. What is a GUI?
2. What is event-driven programming?
3. What is an event?
4. What is an event listener?
5. What is AWT?
6. What is Swing?
7. What is the difference between AWT and Swing?
8. What is `JFrame`?
9. What is `JPanel`?
10. What is `JLabel`?
11. What is `JButton`?
12. What is `JTextField`?
13. What is `JTextArea`?
14. Why is `JScrollPane` useful?
15. What is a layout manager?
16. What does `FlowLayout` do?
17. What does `BorderLayout` do?
18. What does `GridLayout` do?
19. What is an `ActionListener`?
20. How can a lambda expression handle a button click?
21. What is a check box?
22. What is a radio button?
23. Why is `ButtonGroup` used?
24. What is `JComboBox`?
25. What is `JList`?
26. What is `JOptionPane`?
27. What is `JTable`?
28. Why are table models useful?
29. What is `JFileChooser`?
30. What is the Event Dispatch Thread?
31. Why should long tasks not run directly on the EDT?
32. What is `SwingWorker`?
33. What is separation of concerns in GUI design?
34. What was a Java applet?
35. Why are Java applets considered obsolete?

36. What happened to the Applet API in JDK 26?

37. Programming Exercises

Beginner Level

1. Create an empty `JFrame`.
2. Add a centered `JLabel`.
3. Add a `JButton` that changes its own text.
4. Create a click counter.
5. Create a form with one `JTextField`.
6. Display entered text in a `JLabel`.
7. Create a multi-line notes application using `JTextArea`.
8. Create a small window using `FlowLayout`.
9. Create a window using `GridLayout`.
10. Create a window using `BorderLayout`.
11. Create two check boxes and display their selected states.
12. Create a group of three radio buttons.
13. Create a combo box for selecting a course.
14. Show a message dialog.
15. Ask for a name using an input dialog.

Intermediate Level

1. Create a login form with validation.
2. Create a simple calculator GUI.
3. Create a temperature converter.
4. Create a GPA calculator.
5. Create a student registration form.
6. Display students in a `JTable`.
7. Add rows dynamically using `DefaultTableModel`.
8. Add a menu bar with File and Help menus.
9. Add an Exit menu item.
10. Create a text viewer using `JFileChooser`.
11. Create a text editor with Open and Save actions.
12. Add input validation with dialog messages.
13. Build a form using several nested `JPanel` objects.
14. Add mouse-click handling with `MouseAdapter`.
15. Create a small background task using `SwingWorker`.

Advanced Level

1. Build a GUI-based Student Management System.
2. Create add, edit, delete, and search operations in a student table.
3. Connect a Swing form to the file-based repository from Lecture 12.
4. Build a contact-management desktop application.
5. Create a small inventory-management GUI.
6. Create a course-registration GUI with validation.
7. Separate model, service, repository, and UI packages.
8. Create a table model for a custom domain class.
9. Build a GUI log viewer with file selection.
10. Create a multi-panel dashboard using menus and dialogs.

38. Homework

Task: Develop a Swing-Based Student Management Application

Create the following project structure:

Illustration

```
model/
  Student.java

repository/
  StudentRepository.java

service/
  StudentService.java

ui/
  StudentFrame.java

app/
  Main.java
```

Requirements:

1. The main window must use `JFrame`.
2. Use at least three different layout managers.
3. Provide fields for:
   - student ID;
   - name;
   - age;
   - GPA.
4. Provide buttons for:
   - Add;
   - Update;
   - Delete;
   - Clear;
   - Search.
5. Display student records in a `JTable`.
6. Validate all user input.
7. Show validation errors using `JOptionPane`.
8. Use a menu bar with:

- File;
- Help;
- Exit;
- About.

9. Load student data from a file when the application starts.
10. Save student data after modifications.
11. Use `JFileChooser` for at least one import or export action.
12. Keep persistence logic outside the GUI class.
13. Build the GUI on the Event Dispatch Thread.

Submission requirements:

- all Java source files;
- screenshots of the running GUI;
- sample student data file;
- a short class diagram;
- explanation of the event-handling mechanism;
- explanation of the selected layout managers;
- explanation of how input validation is implemented;
- explanation of how model, service, repository, and UI responsibilities are separated.

39. Key Terms

GUI
: Graphical User Interface.

Event-driven programming
: Programming model in which execution responds to events.

Event
: An action or occurrence recognized by the application.

Listener
: Object that receives and handles events.

AWT
: Abstract Window Toolkit, an older Java GUI toolkit and event infrastructure.

Swing
: Java desktop GUI toolkit built on AWT.

`JFrame`
: Top-level Swing window.

`JPanel`
: General-purpose Swing container.

Layout manager
: Object that arranges components in a container.

Event Dispatch Thread
: Swing thread responsible for event processing and most GUI updates.

`JTable`
: Swing component for tabular data.

`JOptionPane`
: Swing utility for standard dialogs.

`JFileChooser`
: Swing file selection dialog.

`SwingWorker`
: Swing helper for background work with EDT-safe completion processing.

Applet
: Historical Java program type formerly embedded in an applet environment; removed from JDK 26.