Graphical User Interface in Java

Swing Library and Event Handling

Zbigniew Dendzik

March 2026

Lecture Plan

- Introduction to Swing library
- Basic GUI components
- Creating windows and panels
- Event handling – ActionListener
- Layout Managers
- Calculator – comprehensive example
- Serialization and encryption
- Practical applications

Swing Library

Swing – library for creating graphical interfaces in Java. **Main package:**

`javax.swing`

- Integral part of Java platform
- Advanced GUI components
- Platform independence (“Write once, run anywhere”)
- Rich set of ready-made components

Basic components:

- `JFrame` – main application window
- `JPanel` – panel for grouping components
- `JButton` – button
- `JTextField` – text field
- `JLabel` – text label

Swing Application Architecture

Basic structure:

1. **JFrame** – main window
2. **Container** – component container
3. **JPanel** – grouping panel
4. **Components** – buttons, text fields

Hierarchy:

$$\text{JFrame} \rightarrow \text{Container} \rightarrow \text{JPanel} \rightarrow \text{Components}$$

Key concepts:

- Component creation
- Component layout
- Event listener registration
- User event handling

Simple Example – Structure

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

public class Demo implements ActionListener
{
    JTextField t1;
    JButton b1;
    JButton b2;

    public void actionPerformed(ActionEvent e)
    {
        // Event handling
    }

    void init()
    {
        // Interface creation
```

Creating Components

```
void init()
{
    // Create components
    t1 = new JTextField(6);
    b1 = new JButton("^2");
    b2 = new JButton("clear");

    // Create panel
    JPanel p = new JPanel();
    p.add(t1);
    p.add(b1);
    p.add(b2);

    // Create window
    JFrame f = new JFrame();
    Container c = f.getContentPane();
    c.add(p);
    f.pack();
```

Event Handling – ActionListener

Event – user action (click, key press). **ActionListener interface:**

- Implement `actionPerformed(ActionEvent e)` method
- Register listener: `component.addActionListener(this)`
- Identify event source: `e.getSource()`

Pattern:

1. Class implements ActionListener interface
2. Components register listener
3. actionPerformed() method handles events
4. Source identification and appropriate response

ActionListener Implementation

```
public void actionPerformed(ActionEvent e)
{
    Object target = e.getSource();

    if(target == b1 || target == t1)
    {
        int k = Integer.parseInt(t1.getText());
        t1.setText(Integer.toString(k * k));
        t1.requestFocus();
    }
    else if(target == b2)
    {
        t1.setText("");
        t1.requestFocus();
    }
}
```

Listener Registration

```
void init()
{
    t1 = new JTextField(6);
    b1 = new JButton("^2");
    b2 = new JButton("clear");

    // Register listeners
    t1.addActionListener(this);
    b1.addActionListener(this);
    b2.addActionListener(this);

    JPanel p = new JPanel();
    p.add(t1);
    p.add(b1);
    p.add(b2);

    // ...
}
```

Launching Swing Application

```
public static void main(String[] args)
{
    // Up to Java 1.4
    // new Demo().init();

    // From Java 1.5
    SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            new Demo().init();
        }
    });
}
```

Why SwingUtilities.invokeLater()?

- Ensures safe GUI creation in Event Dispatch Thread

Layout Managers

Layout Manager – controls component placement. **Main managers:**

- **FlowLayout** – components in row, left to right
- **BorderLayout** – 5 regions (North, South, East, West, Center)
- **GridLayout** – grid with equal cells
- **GridBagLayout** – flexible grid (most advanced)

GridBagLayout:

- Most flexible and powerful
- Requires GridBagConstraints
- Control over position, size, margins
- Ideal for complex interfaces

GridBagLayout – Initialization

```
void init()
{
    JFrame f = new JFrame();
    Container c = f.getContentPane();

    // Create layout manager
    GridBagLayout gbl = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();

    // Default settings
    gbc.fill = GridBagConstraints.HORIZONTAL;

    // Set manager
    c.setLayout(gbl);

    // Add components...
}
```

GridBagConstraints – Parameters

```
// Grid position
gbc.gridx = 0;          // column
gbc.gridy = 0;          // row

// Size
gbc.gridwidth = 5;      // width in columns
gbc.gridheight = 1;     // height in rows

// Padding
gbc.ipadx = 0;          // internal
gbc.ipady = 5;          // internal

// External margins
gbc.insets = new Insets(5, 5, 0, 5); // top, left, bottom,
    right

// Apply constraints
gbl.setConstraints(component, gbc);
```

Example – Adding Text Field

```
t1 = new JTextField(15);
t1.addActionListener(this);
t1.setHorizontalAlignment(JTextField.RIGHT);

gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 5;
gbc.ipadx = 0;
gbc.ipady = 5;
gbc.insets = new Insets(5, 5, 0, 5);

gbl.setConstraints(t1, gbc);
c.add(t1);
```

Example – Adding Button

```
b1 = new JButton("1");
b1.addActionListener(this);
b1.setFocusable(false);

gbc.gridx = 0;
gbc.gridy = 1;
gbc.gridwidth = 1;
gbc.ipadx = 0;
gbc.ipady = 0;
gbc.insets = new Insets(5, 5, 0, 0);

gbl.setConstraints(b1, gbc);
c.add(b1);
```

Component Properties

JTextField:

- `getText()` – get text
- `setText(String)` – set text
- `setHorizontalAlignment()` – text alignment
- `setEditable(boolean)` – editable or not
- `requestFocus()` – set focus

JButton:

- `getText()` – get label
- `setText(String)` – set label
- `setFocusable(boolean)` – can receive focus
- `setToolTipText(String)` – tooltip on hover

Calculator – Architecture

Calculator elements:

- Display (JTextField)
- Digit buttons (0-9)
- Operation buttons (+, -, *, /)
- Equals button (=)
- Additional buttons (C, ., +/-,

State variables:

- `double buf` – buffer for first number
- `double x` – calculation result
- Operation flags

Calculator – Handling Digits

```
public void actionPerformed(ActionEvent e)
{
    Object target = e.getSource();

    // Handle digit buttons
    if(target == b1) // button "1"
    {
        t1.setText(t1.getText() +
                   ((JButton)target).getText());
        t1.requestFocus();
    }

    // Similarly for other digits...
}
```

Calculator – Handling Operations

```
// Handle "+" button
else if(target == bplus)
{
    buf = Double.parseDouble(t1.getText());
    t1.setText("");
    t1.requestFocus();
}

// Handle "=" button
else if(target == brow || target == t1)
{
    x = Double.parseDouble(t1.getText());
    x = buf + x;
    t1.setText(Double.toString(x));
    t1.requestFocus();
}
```

Error Handling

Common GUI exceptions:

- `NumberFormatException` – invalid number format
- `ArithmeticException` – division by zero

Best practices:

- Always wrap parsing in try-catch
- Inform user about error
- Reset calculator state after error
- Validate input data

Error Handling Example

```
try
{
    x = Double.parseDouble(t1.getText());
    x = buf + x;
    t1.setText(Double.toString(x));
}
catch(NumberFormatException ex)
{
    t1.setText("Error");
    buf = 0;
}
catch(ArithmeticException ex)
{
    t1.setText("Error: div by 0");
    buf = 0;
}
```

Mathematical Functions

java.lang.Math class:

- `Math.sqrt(double)` – square root
- `Math.pow(double, double)` – power
- `Math.sin(double)`, `Math.cos(double)` – trigonometric functions
- `Math.abs(double)` – absolute value
- `Math.PI` – constant
- `Math.E` – constant e

double String conversion:

- `Double.parseDouble(String)` – String $\rightarrow$ double
- `Double.toString(double)` – double $\rightarrow$ String

KeyListener – Keyboard Handling

KeyListener interface:

- `keyPressed(KeyEvent e)` – key pressed
- `keyReleased(KeyEvent e)` – key released
- `keyTyped(KeyEvent e)` – character typed

Applications:

- Calculator keyboard handling
- Input character validation
- Keyboard shortcuts

Registration:

- `component.addKeyListener(this)`
- Class must implement KeyListener

KeyListener Example

```
public class Kalk implements ActionListener, KeyListener
{
    public void keyPressed(KeyEvent e)
    {
        int code = e.getKeyCode();

        if(code >= KeyEvent.VK_0 && code <= KeyEvent.VK_9)
        {
            // Handle digits
        }
        else if(code == KeyEvent.VK_ENTER)
        {
            // Perform calculation
        }
    }

    public void keyReleased(KeyEvent e) { }
    public void keyTyped(KeyEvent e) { }
```

Object Serialization

Serialization – writing object to byte stream. **Key classes:**

- `ObjectOutputStream` – writing objects
- `ObjectInputStream` – reading objects
- `Serializable` – interface marking serializability

Applications:

- Saving application state
- Sending objects over network
- Persistent data storage

Serialization Example

```
// Write object
ObjectOutputStream out = new ObjectOutputStream(
    new FileOutputStream("data.ser"));
out.writeObject(myObject);
out.flush();
out.close();

// Read object
ObjectInputStream in = new ObjectInputStream(
    new FileInputStream("data.ser"));
MyClass obj = (MyClass)in.readObject();
in.close();
```

DES Encryption

DES (Data Encryption Standard) – symmetric encryption algorithm. **Key classes:**

- `javax.crypto.Cipher` – encryption/decryption
- `javax.crypto.KeyGenerator` – key generation
- `javax.crypto.SecretKey` – symmetric key
- `CipherOutputStream` – encrypting stream
- `CipherInputStream` – decrypting stream

Steps:

1. Generate key from password
2. Initialize cipher
3. Encrypt/decrypt data

DES Key Generation

```
String passwd = "secretpassword";
byte[] b = passwd.getBytes();

// Generate key
KeyGenerator kgen = KeyGenerator.getInstance("DES");
kgen.init(new SecureRandom(b));

SecretKey key = kgen.generateKey();
```

Encryption with Serialization

```
// Initialize cipher for encryption
Cipher c1 = Cipher.getInstance("DES");
c1.init(Cipher.ENCRYPT_MODE, key);

// Write encrypted object
ObjectOutputStream out = new ObjectOutputStream(
    new CipherOutputStream(
        new FileOutputStream("file.txt"), c1));
out.writeObject(msg);
out.flush();
out.close();
```

Decryption with Serialization

```
// Initialize cipher for decryption
Cipher c2 = Cipher.getInstance("DES");
c2.init(Cipher.DECRYPT_MODE, key);

// Read encrypted object
ObjectInputStream in = new ObjectInputStream(
    new CipherInputStream(
        new FileInputStream("file.txt"), c2));
String str = (String)in.readObject();
in.close();

System.out.println("Read: " + str);
```

Application – Electronic Diary

Requirements:

- Graphical interface (Swing)
- Text area for entries (JTextArea)
- Automatic entry dating
- Save to encrypted file (DES)
- Password protection

Extensions:

- Bookmark system for entries
- Full-text search
- Adding photos to notes
- Export to PDF

Look and Feel

Look and Feel – interface appearance and behavior. **Available styles:**

- Metal (default Java)
- System (native to operating system)
- Nimbus (modern)
- Motif, GTK+

Setting:

- Before creating GUI components
- `UIManager.setLookAndFeel()`

Setting Look and Feel

```
void init()
{
    try
    {
        UIManager.setLookAndFeel(
            UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e)
    {
        // Use default
    }

    // Create interface...
}
```

Additional Swing Components

Useful components:

- `JLabel` – text label
- `JTextArea` – multi-line text field
- `JCheckBox` – checkbox
- `JRadioButton` – radio button
- `JComboBox` – dropdown list
- `JList` – list
- `JMenu`, `JMenuBar` – menu
- `JScrollPane` – scrollable panel
- `JDialog` – dialog window
- `JFileChooser` – file chooser

Summary

- **Swing** – GUI library in Java
- **JFrame** – main application window
- **ActionListener** – event handling
- **Layout Managers** – component placement
- **GridBagLayout** – flexible layout
- **KeyListener** – keyboard handling
- **Serialization** – object storage
- **DES encryption** – data protection