Advanced Programming
Graphical User Interface (GUI)
Human-Machine Interfaces

The ways in which a software system interacts with its users.

- Command Line
- Graphical User Interface - GUI
- Touch User Interface - TUI
- Multimedia (voice, animation, etc.)
- Intelligent (gesture recognition, conversational, etc.)
Graphical User Interfaces

Visual communication between software and users.

- **AWT** (Abstract Windowing Toolkit)
- **Swing** – part of JFC (Java Foundation Classes)
- **SWT** (IBM)
- Java FX
- XUL
- ...
  - Java 2D
  - Java 3D
The Stages of Creating a GUI Application

- **Design**
  - Create the **containers**
  - Create and arrange the **components**

- **Functionality**
  - Define the user-components **interaction**
  - Attach **actions** to components
  - Create the action **handlers**

- **Considerations**
  - Programatic – Declarative – Visual
  - Separation between the GUI and application logic
import java.awt.*;
public class AWTExample {
    public static void main (String args []) {
        // Create the window (frame)
        Frame f = new Frame("O fereastra");

        // Set the layout of the frame
        f.setLayout (new FlowLayout());

        // Create the components
        Button b1 = new Button("OK");
        Button b2 = new Button("Cancel");

        // Add the components to the frame
        f.add(b1);
        f.add(b2);
        f.pack();

        // Show the frame
        f.setVisible(true);
    }
}
# AWT Components

- Button
- Canvas
- Checkbox
- CheckBoxGroup
- Choice
- Container
- Label
- List
- Scrollbar
- TextComponent
- TextField
- TextArea

AWT Components are *platform-dependent*, each of them having an underlying *native peer*. 
AWT Infrastructure

- **Component**
  - A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Properties common to all components are:
    
    $location, x, y, size, height, width, bounds, foreground, background, font, visible, enabled,...$

- **Container**
  - A generic component containing other components.

- **LayoutManager**
  - The interface for classes that know how to lay out Containers.

- **AWTEvent**
  - The root event class for all AWT events
Frames and Panels

```java
Frame f = new Frame("Hello Frame");
// Add a button on the frame
f.add(new Button("Hello"));

// Create a panel
Panel panel = new Panel();
panel.add(new Label("Name:"));
panel.add(new TextField());

// Add the panel on the frame
f.add(panel);

class MyFrame extends Frame {
    // Constructor
    public MyFrame(String title) {
        super(title);
        ...
    }
}
...
MyFrame f = new MyFrame("My very special frame");
f.setVisible(true);
```
import java.awt.*;
public class TestLayout {
    public static void main ( String args []){
        Frame f = new Frame("Grid Layout");
        f.setLayout (new GridLayout (3, 2));
        Button b1 = new Button (" Button 1");
        Button b2 = new Button ("2");
        Button b3 = new Button (" Button 3");
        Button b4 = new Button ("Long - Named Button 4");
        Button b5 = new Button (" Button 5");
        f.add(b1); f.add (b2); f. add(b3); f.add(b4); f.add(b5);
        f.pack ();
        f.setVisible(true);
    }
}

Frame f = new Frame("Flow Layout");
f.setLayout (new FlowLayout ());
A layout manager is an object that controls the size and arrangement (position) of components inside a container.

Each Container object has a layout manager.

All classes that instantiate objects for managing positioning implements LayoutManager interface.

Upon instantiation of a container it is created an implicit layout manager associated with it:

- frames: BorderLayout
- panels: FlowLayout
“Classical” Layout Managers

FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout

Setting a layout manager

container.setLayout(new FlowLayout());

Controlling the dimensions of the components

preferredSize, minimumSize, maximumSize

Absolute positioning

container.setLayout(null);
Button b = new Button("Buton");
b.setSize(10, 10);
b.setLocation(0, 0);
container.add(b);
import java.awt.*;
public class TestBorderLayout {
    public static void main ( String args []) {

        Frame f = new Frame (" Border Layout ");
        // This is the default for frames
        f.setLayout (new BorderLayout());

        f.add(new Button(" North "), BorderLayout.NORTH);
        f.add(new Button(" South "), BorderLayout.SOUTH);
        f.add(new Button(" East "), BorderLayout.EAST);
        f.add(new Button(" West "), BorderLayout.WEST);
        f.add(new Button(" Center "), BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }
}
GridBagLayout

GridBagLayout gridBag = new GridBagLayout();
container.setLayout(gridBag);

GridBagConstraints c = new GridBagConstraints();
//Define the constraints
  c.fill = GridBagConstraints.HORIZONTAL;
  c.gridx = 0;
  c.gridy = 0;

  • gridx, gridy
  • gridwidth, gridheight
  • fill
  • insets
  • anchor
  • weightx, weighty

  ...

gridBag.setConstraints(componenta, c);
container.add(componenta);
User Interactions

*Event-Driven Programming*

**Event:** clicking a button, altering the text, checking an option, closing a frame, etc.

**Source:** the component that generates an event.

**Listener:** the responsible for receiving and handling (consuming) events.
**Observer Design Pattern**

Observing the state of an entity within a system

*(Publish-Subscribe)*

```
Subject
- attach(Observer)
- detach(Observer)
- notify()

Observers
- for all o in observers {
  - o->update()
}

Observer
- update()

ConcreteSubject
- subjectState
- getState()
- setState()

ConcreteObserver
- observerState
- update()
```

```c
return subjectState = subject->getState()
```
Button - ActionEvent - ActionListener

class MyFrame extends Frame {
    public MyFrame ( String title ) {
        super (title);
        setLayout (new FlowLayout ());
        setSize (200, 100);
        Button b1 = new Button ("OK");
        Button b2 = new Button ("Cancel");
        add(b1); add(b2);
        MyButtonListener listener = new MyButtonListener (this);
        b1.addActionListener ( listener );
        b2.addActionListener ( listener );
        // The events generated by the two buttons
        // are intercepted by the listener object
    }
}

class MyButtonListener implements ActionListener {
    private MyFrame frame;
    public MyButtonListener (MyFrame frame) {
        this.frame = frame;
    }
    // ActionListener interface has only one method
    public void actionPerformed (ActionEvent e) {
        frame.setTitle ("You pressed the button " + e.getActionCommand());
    }
}
Using Anonymous Classes

class MyFrame extends Frame {
    public MyFrame ( String title ) {
        ...
        button.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                MyFrame.this.setTitle( "You pressed the button " + e.getActionCommand() );
            }
        });
        ...
    }
}

Using Lambda Expressions

    ...
    button.addActionListener( (ActionEvent e) -> {
        MyFrame.this.setTitle( "You pressed the button " + e.getActionCommand() );
    });
    ...
}
Using Method References

```java
class MyFrame extends Frame {
    public MyFrame ( String title ) {
        ...
        button.addActionListener( this::onButtonPressed );
        checkbox.addItemListener( this::onItemChanged );
        ...
    }

    //Your own, suggestively called, methods
    private void onButtonPressed(ActionEvent e) {
        this.setTitle("You pressed the button");
    }

    private void onItemChanged(ItemEvent e) {
        this.setTitle("Checkbox state: " + check.getState());
    }
}
```
## Event Types

<table>
<thead>
<tr>
<th>Low-level</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ComponentEvent</strong></td>
<td><strong>ActionEvent</strong></td>
</tr>
<tr>
<td>hiding, moving, resizing,</td>
<td>pressing a button, pressing 'enter' in a</td>
</tr>
<tr>
<td>displaying components</td>
<td>text editing component, etc.</td>
</tr>
<tr>
<td><strong>ContainerEvent</strong></td>
<td><strong>AdjustmentEvent</strong></td>
</tr>
<tr>
<td>adding, removing components</td>
<td>adjusting the value of a scrollbar, etc.</td>
</tr>
<tr>
<td>in/from a container</td>
<td></td>
</tr>
<tr>
<td><strong>FocusEvent</strong></td>
<td><strong>ItemEvent</strong></td>
</tr>
<tr>
<td>getting, losing the focus</td>
<td>changing the state of a componente:</td>
</tr>
<tr>
<td></td>
<td>selecting some items in a list, selecting</td>
</tr>
<tr>
<td></td>
<td>or deselecting a checkbox, etc.</td>
</tr>
<tr>
<td><strong>KeyEvent</strong></td>
<td><strong>TextEvent</strong></td>
</tr>
<tr>
<td>pressing, releasing a key</td>
<td>changing the text in a component</td>
</tr>
<tr>
<td><strong>MouseEvent</strong></td>
<td></td>
</tr>
<tr>
<td>mouse clicking, dragging,</td>
<td>...</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
<tr>
<td><strong>WindowEvent</strong></td>
<td></td>
</tr>
<tr>
<td>frame minimizing, resizing,</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
# Component-Listener Relationship

## many-to-many

<table>
<thead>
<tr>
<th>Component</th>
<th>Listener</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>ComponentListener</td>
</tr>
<tr>
<td></td>
<td>FocusListener</td>
</tr>
<tr>
<td></td>
<td>KeyListener</td>
</tr>
<tr>
<td></td>
<td>MouseListener</td>
</tr>
<tr>
<td>Container</td>
<td>ContainerListener</td>
</tr>
<tr>
<td>Window</td>
<td>WindowListener</td>
</tr>
<tr>
<td>Button List</td>
<td>ActionListener</td>
</tr>
<tr>
<td>MenuItem</td>
<td></td>
</tr>
<tr>
<td>List</td>
<td></td>
</tr>
<tr>
<td>TextField</td>
<td></td>
</tr>
<tr>
<td>Choice List</td>
<td>ItemListener</td>
</tr>
<tr>
<td>Checkbox List</td>
<td></td>
</tr>
<tr>
<td>Scrollbar</td>
<td>AdjustmentListener</td>
</tr>
<tr>
<td>TextField</td>
<td>TextListener</td>
</tr>
<tr>
<td>TextArea</td>
<td></td>
</tr>
</tbody>
</table>
## Handler Methods

<table>
<thead>
<tr>
<th>Listener Type</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent e)</td>
</tr>
<tr>
<td>ItemListener</td>
<td>itemStateChanged(ItemEvent e)</td>
</tr>
<tr>
<td>TextListener</td>
<td>textValueChanged(TextEvent e)</td>
</tr>
<tr>
<td>MouseListener</td>
<td>mouseClicked(MouseEvent e)</td>
</tr>
<tr>
<td></td>
<td>mouseEntered(MouseEvent e)</td>
</tr>
<tr>
<td></td>
<td>mouseExited(MouseEvent e)</td>
</tr>
<tr>
<td></td>
<td>mousePressed(MouseEvent e)</td>
</tr>
<tr>
<td></td>
<td>mouseReleased(MouseEvent e)</td>
</tr>
<tr>
<td>MouseMotionListener</td>
<td>mouseDragged(MouseEvent e)</td>
</tr>
<tr>
<td></td>
<td>mouseMoved(MouseEvent e)</td>
</tr>
<tr>
<td>WindowListener</td>
<td>windowActivated(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowClosed(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowClosing(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowDeactivated(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowDeiconified(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowIconified(WindowEvent e)</td>
</tr>
<tr>
<td></td>
<td>windowOpened(WindowEvent e)</td>
</tr>
</tbody>
</table>

...
Using Adapters

class MyFrame extends Frame implements WindowListener {
    public MyFrame (String titlu) {
        super (titlu);
        this.addWindowListener(this);
    }
    // We are interested only in one of method of WindowListener
    public void windowOpened (WindowEvent e) {}  
    public void windowClosing (WindowEvent e) {
        // Terminate the program
        System.exit (0);
    }
    public void windowClosed (WindowEvent e) {} 
    public void windowIconified (WindowEvent e) {} 
    public void windowDeiconified (WindowEvent e) {} 
    public void windowActivated (WindowEvent e) {} 
    public void windowDeactivated (WindowEvent e) {} 
}

An adapter class provides the default implementation of all methods in an event listener interface.

```java
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```
Menus
Swing

- **Extends** the core concepts and mechanisms of AWT; *we still have components, containers, layout managers, events and event listeners.*

- **Replaces completely** the AWT component set, providing a [new set of components](#), capable of sorting, printing, drag and drop and other “cool” features.

- Brings **portability** to the GUI level; no more *native peers*, all components are “pure”.

- Based on **Separable Model-and-View** design pattern.

- "**Component Oriented Programming**"
Java Foundation Classes

The Java Foundation Classes (JFC) are a comprehensive set of GUI components and services which dramatically simplify the development and deployment of commercial-quality desktop and Internet/Intranet applications.

- Swing
- Look-and-Feel
- Accessibility API
- Java 2D API
- Drag-and-Drop
- Internationalization
Swing Components

- **Atomic Components**
  - JLabel, JButton, JCheckBox, JRadioButtom, JToggleButton, JScrollBar, JSlider, JProgressBar, JSeparator

- **Complex Components**
  - JTable, JTextTree, JComboBox, JSpinner, JList, JFileChooser, JColorChooser, JOptionPane

- **Text Editing Components**
  - JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane, JTextPane

- **Menus**
  - JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem

- **Intermediate Containers**
  - JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, JToolBar

- **High-Level Containers**
  - JFrame, JDialog, JWindow, JInternalFrame, JApplet
Similarities and Differences with AWT

"J" Convention

java.awt.Frame – javax.swing.JFrame
java.awt.Button – javax.swing.JButton
java.awt.Label – javax.swing.JLabel

New Layout Managers

BoxLayout, SpringLayout, GroupLayout, OverlayLayout, etc.

HTML Aware Components

JButton simple = new JButton("Dull text");

JButton html = new JButton("<html><u>Cool</u> <i>text</i></html>"
Frame f = new Frame();
f.setLayout(new FlowLayout());
f.add(new Button("OK"));

JFrame jf = new JFrame();
jf.getContentPane().setLayout(new FlowLayout());
jf.getContentPane().add(new JButton("OK"));
Internal Frames

GUI applications can be designed either as:

- **SDI** (Single Document Interface) or
- **MDI** (Multiple Document Interface)

Multiple-document interface (MDI) applications enable you to display multiple documents at the same time, with each document displayed in its own window.

**JInternalFrame**

**DesktopPane**
**JComponent**

**JComponent** is the base class for all Swing components, except top-level containers: JFrame, JDialog, JApplet.

**JComponent extends Container**

- Support for **tool tips** - setToolTip
- Support for **borders** - setBorder
- Enhanced support for **sizing and positioning**
  - setPreferredSize, ...
- **Opacity** control - setOpaque
- **Keyboard bindings**
- “Pluggable” **look and feel**
- Double-Buffering, Support for accessibility, etc.
Swing Architecture

Swing architecture is “rooted” in the MVC design:

- **Model** – the data for the application
- **View** – the visual representation of the data
- **Controller** – takes user input on the view and translates that to changes in the model.

Separable Model Architecture

Model + (Presentation, Control)
<table>
<thead>
<tr>
<th>Component Class</th>
<th>Interface describing the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>JList</td>
<td>ListModel, ListSelectionModel</td>
</tr>
<tr>
<td>JTable</td>
<td>TableModel, TableColumnModel, ListSelectionModel</td>
</tr>
<tr>
<td>JTree</td>
<td>TreeModel, TreeSelectionModel</td>
</tr>
<tr>
<td>JEditorPane, JTextPane</td>
<td>Document</td>
</tr>
<tr>
<td>JTextField</td>
<td></td>
</tr>
</tbody>
</table>

Creating a model involves either implementing the interface or extending an abstract support class

JList - ListModel, DefaultListModel, AbstractListModel
Components are created based on a model:

The model could be a standard data structure:

```java
Object elements[] = {"Unu", "Doi", new Integer(3), 4.0};
JList list = new JList(elements);
```

or a model instance specific to that component:

```java
DefaultListModel model = new DefaultListModel();
model.addElement("Unu");
model.addElement("Doi");
model.addElement(new Integer(3));
model.addElement(4.0);
JList list = new JList(model);
```
Example: *JTable*

class MyTableModel extends AbstractTableModel {
    private String[] columns = {"Nume", "Varsta", "Student"};
    private Object[][] elements = {
        {"Ionescu", new Integer(20), Boolean.TRUE},
        {"Popescu", new Integer(80), Boolean.FALSE}};

    public int getColumnCount() {
        return columns.length;
    }

    public int getRowCount() {
        return elements.length;
    }

    public Object getValueAt(int row, int col) {
        return elements[row][col];
    }

    public String getColumnName(int col) {
        return columns[col];
    }

    public boolean isCellEditable(int row, int col) {
        // Doar numele este editabil
        return (col == 0);
    }
}
## Customizing the View

The SwingSet2 application allows for customization of the view in various ways. You can adjust settings such as reordering allowed, row selection, horizon and vertical lines, inter-cell spacing, row height, selection mode, auto-size mode, and printing options.

### Example Table

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Favorite Color</th>
<th>Favorite Movie</th>
<th>Favorite Number</th>
<th>Favorite Food</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike</td>
<td>Albers</td>
<td>Green</td>
<td>Brazil</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Mark</td>
<td>Andrews</td>
<td>Blue</td>
<td>Curse of the Demon</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Brian</td>
<td>Beck</td>
<td>Black</td>
<td>The Blues Brothers</td>
<td>2,718</td>
<td></td>
</tr>
<tr>
<td>Lara</td>
<td>Bunni</td>
<td>Red</td>
<td>Airplane (the whole)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Roger</td>
<td>Brinkley</td>
<td>Blue</td>
<td>The Man Who Know...</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Brent</td>
<td>Christian</td>
<td>Black</td>
<td>Blade Runner (Dir...</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Mark</td>
<td>Davidson</td>
<td>Dark Green</td>
<td>Brazil</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Jeff</td>
<td>Dinkins</td>
<td>Blue</td>
<td>The Lady Vanishes</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Ewan</td>
<td>Dinkins</td>
<td>Yellow</td>
<td>A Bug’s Life</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Amy</td>
<td>Fowler</td>
<td>Violet</td>
<td>Reservoir Dogs</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Hania</td>
<td>Gajewiska</td>
<td>Purple</td>
<td>Jules et Jim</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Press Shift-F10 to activate popup menu.
The **CellRenderer** Concept

A **renderer** is responsible for displaying the content of components, for example: the appearance of items in a list.

```java
class MyCellRenderer extends JLabel implements ListCellRenderer {
    public MyCellRenderer() {
        setOpaque(true);
    }

    public Component getListCellRendererComponent(
            JList list, Object value, int index,
            boolean isSelected, boolean cellHasFocus) {
        setText(value.toString());
        setBackground(isSelected ? Color.red : Color.white);
        setForeground(isSelected ? Color.white : Color.black);
        return this;
    }
}

...  
list.setCellRenderer(new MyCellRenderer());
```
The **CellEditor** Concept

An **editor** is responsible for the editing of individual items of components, such as the cells in a table.

```java
public class MyCellEditor extends AbstractCellEditor
    implements TableCellEditor {

    public Component getTableCellEditorComponent(...) {
        //Returns the component
        //responsible for editing
        ...
    }

    public Object getCellEditorValue() {
        // Returns the cell value
        ...
    }
}
```
Swing Containers

- **High-Level Containers**
  - JFrame, JDialog, JApplet

- **Intermediate Containers**
  - JPanel
  - JScrollPane
  - JTabbedPane
  - JSplitPane
  - JLayeredPane
  - JDesktopPane
  - JRootPane
Look and Feel

The architecture of Swing is designed so that you may change the "look and feel" (L&F) of your application's GUI. "Look" refers to the appearance of GUI widgets and "feel" refers to the way the widgets behave.

- javax.swing.plaf.metal.MetalLookAndFeel
- com.sun.java.swing.plaf.windows.WindowsLookAndFeel
- com.sun.java.swing.plaf.mac.MacLookAndFeel
- com.sun.java.swing.plaf.motif.MotifLookAndFeel
- com.sun.java.swing.plaf.gtk.GTKLookAndFeel
- ...

UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

SwingUtilities.updateComponentTreeUI(f);
f.pack();
The Java Tutorial

- Trail: Graphical User Interfaces
  
  http://docs.oracle.com/javase/tutorial/ui/index.html

- Trail: Creating a GUI With JFC/Swing
  
  http://docs.oracle.com/javase/tutorial/uiswing/index.html