# EXERCISE 4

## GUI Programming with AWT and Swing

In Java, GUI-based programs are implemented by using the classes from the standrad `javax.swing` and `java.awt` packages.

- Java 1.0 (1996) AWT (AbstractWindowToolkit): package `java.awt.*`
- Java 1.1 (1998) JFC/Swing as Standard GUI: package `javax.swing.*`

In older versions of Java, we had only `AWT` classes to build GUI-based programs. These classes are still available, but it is generally preferable to use `Swing` classes. [2]
There are two main advantages in using `Swing` classes over the `AWT` classes:

- First, `Swing` classes provide greater compatibility across different operating systems. The `Swing` classes are fully implemented in Java, they are called lightweight classes. The `AWT` classes are implemented by using the native GUI objects of the operating systems (so, behaviour is dependent on the operating system), they are called heavyweight classes.
- Second, `Swing` classes support many new functionalities not supported by the `AWT` classes. As a general rule, it is best not to mix the counterparts (e.g. `Swing JButton` and `AWT Button`) in the same program. Elements of the `Swing` classes start with `J...`.

Futhermore, Java provides JavaFX for developing GUI-based programs for desktop, web or mobile applications.

## GUI Programming Basics - An Introducing Example

There are two key aspects in GUI programming: first, the placement of GUI objects on the content pane of a frame, and second, the handling of events generated by these GUI objects.

### GUI Object Placement

We have two possibilities when placing components on a frame's content pane, either using a Layout Manager or using none, which is called absolute positioning.

**Algorithm 1** A Window to the World!

```java
import javax.swing.JFrame;
public class HelloSwingFrame {
    public static void main(String[] args) {
        JFrame f = new JFrame("A Window to the World!");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300,200);
        f.setVisible(true);
    }
}
```

`JFrame`, `JWindow`, and `JDialog` produce by using `getContentPane()` a `Container`. The layout of this `Container` can be changed via `setLayout(LayoutManager)`.

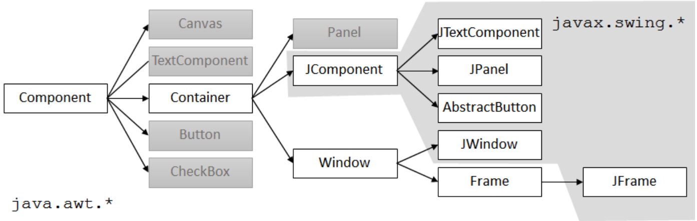


Figure 0.1: Class Hierarchy - AWT and Swing Classes

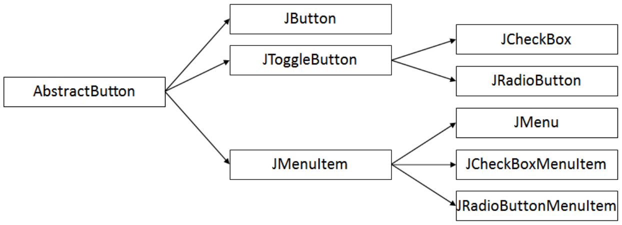


Figure 0.2: Class Hierarchy - AbstractButton Classes

**Algorithm 2** JButton

```java
import javax.swing.JButton;
import javax.swing.JFrame;

public class Exercise_JButton {
    public static void main(String[] args) {
        JFrame f = new JFrame("The Window to the World!");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JButton("I am JButton!"));
        f.setSize(300,200);
        f.setVisible(true);
    }
}
```

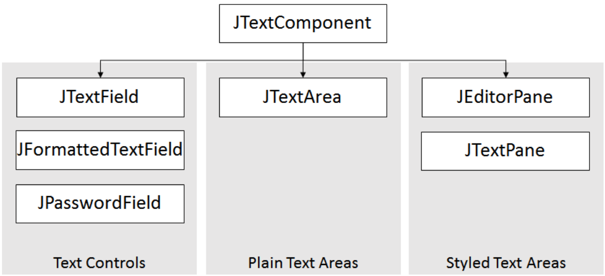


Figure 0.3: Class Hierarchy - JTextComponent Classes

**Algorithm 3** JTextField

```java
import javax.swing.JFrame;
import javax.swing.JTextField;

public class Exercise_JTextField {
    public static void main(String[] args) {
        JFrame f = new JFrame("The Window to the World!");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JTextField("I am a JTexfield!",60));
        f.setSize(300,200);
        f.setVisible(true);
    }
}
```

### Event-driven Programming

For effective GUI-programming it is necessary to understand event-driven programming. An event will occur if the user interacts with a GUI object. For instance, if the user clicks a button, an event will be triggered. In event-driven programs, we implement objects corresponding to these events by defining event-driven programming.

An action such as clicking a button is called **event**. The mechanism to process events is called **event handling**. Java uses the concept known as the **delegation based event model**. Event handling in Java is implemented by two types of objects: event source objects, and event listener objects.

**Event source object (or simple event)** A GUI object, such as a button, is called an event source. It generates events. So, if a user clicks a `JButton` object, it will generate an action event. When an event is generated, the system notifies the corresponding event listener objects.

**Event listener object (or simple event listener)** An event listener object is an object that handles generated events. Such an object includes a method that gets executed in response to generated events. There are several kinds of events, for instance, action event, changing event, window event, list selection event, ...

A single object can be both, an event source and an event listener.

**The Interface ActionListener** Each event is represented by a class, e.g. an action event by the class `ActionEvent`. An event can be handled by one or more listeners. An object that can be registered as an action listener must be an instance of a class that is declared for this purpose. Consequently, we must associate event listeners to the event sources. `JComponents` provide similar methods for adding or deleting listeners for an event (addXXXListener(XXXEvent); removeXXXListener();)

When an event source generates an event, the system checks for matching associated listeners. If there is no matching listener, the event will be ignored.

**Algorithm 4** The Interface ActionListener

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ExerciseActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("You clicked: " + arg0.getActionCommand());
    }

    public static void main(String[] args) {
        JFrame jf = new JFrame("The Window to the World!");
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton jb = new JButton("Click Me!");
        jf.add(jb);
        ExerciseActionListener eal = new ExerciseActionListener();
        jb.addActionListener(eal);
        jf.setSize(300,200);
        jf.setVisible(true);
        jf.pack();
    }
}
```

A single listener can be associated to multiple event sources, and multiple listeners can be associated to a single event source.

**Making a Frame the Event Listener** Instead of creating a separate event listener for each single event source, it is more common to let a frame be the event listener of the GUI objects it contains. We can declare a subclass of `JFrame` that implements the `ActionListener` interface.

**Algorithm 5** Making a Frame the Event Listener

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ExerciseJButtonFrameHandler extends JFrame implements ActionListener {
   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 200;
   private static final int FRAME_X_ORIGIN = 150;
   private static final int FRAME_Y_ORIGIN = 250;
   private static final long serialVersionUID = 1L; // otherwise warning
   private JButton btn_Ok;
   private JButton btn_Cancel;

   public ExerciseJButtonFrameHandler() {
      Container contentPane = this.getContentPane();
      // set the frame properties
      this.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      this.setResizable(true);
      this.setTitle("Exercise Frame as ActionListener");
      this.setLocation(FRAME_X_ORIGIN, FRAME_Y_ORIGIN);
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // set the layout manager
      contentPane.setLayout(new FlowLayout());
      // create and place two buttons on the frame's content pane
      this.btn_Ok = new JButton("Ok");
      this.btn_Cancel = new JButton("Cancel");
      contentPane.add("btn_Ok",btn_Ok);
      contentPane.add("bnt_Cancel", btn_Cancel);
      // register the frame as an action listener of the two buttons
      this.btn_Ok.addActionListener(this);
      this.btn_Cancel.addActionListener(this);
   }

   @Override
   public void actionPerformed(ActionEvent e) {
      JButton btn_Clicked = (JButton) e.getSource();
      String txt = btn_Clicked.getText();
      this.setTitle("You clicked " + txt);
   }

   public static void main(String[] args) {
      ExerciseJButtonFrameHandler myf = new ExerciseJButtonFrameHandler();
      myf.setVisible(true);
   }
}
```

### Types of Events and Listeners

| Types | Components |
|---|---|
| ActionEvent, ActionListener, addActionListener(ActionListener), removeActionListener() | AbstractButton and children (JButton, JRadioButton, JCheckboxButton, JMenuItem, ...) |
| ItemEvent, ItemListener, addItemListener(ItemListener), removeItemListener() | JCheckBox, JComboBox, JList, ... |
| MouseEvent, MouseListener, addMouseListener(MouseListener), removeMouseListener() | Components and children |
| TextEvent, TextListener, addTextListener(TextListener), removeTextListener() | Children of JTextComponent (JTextArea, JTextField, ...) |
| WindowEvent, WindowListener, addWindowListener(WindowListener), removeWindowListener() | Window and children (JFame, JDialog, JFileDialog, ...) |
| ... | |

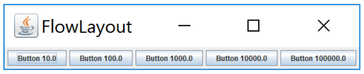
## Layout Managers

A JPanel is used as container to hold other JComponents (JPanel, JButton, JTextField, ...). A JPanel draws its background and uses a LayoutManager to order components in the JPanel. There are several layout managers, for instance: FlowLayout, BorderLayout, GridLayout, BoxLayout, ... [1]

java.awt.FlowLayout    In this layout, GUI components are placed in left-to-right-order.

**Algorithm 6** Flow Layout

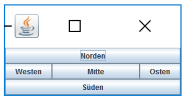
```
package exercise7;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Exercise_FlowLayout extends JPanel {
	private static final long serialVersionUID = 1L; // otherwise warning

	public Exercise_FlowLayout() {
		for(int i = 1; i <= 5; ++i) {
			add(new JButton("Button "+(Math.pow(10, i))));
			}
	}

	public static void main(String[] args) {
		JFrame jf = new JFrame("FlowLayout");
		jf.add(new Exercise_FlowLayout());
		jf.pack();
		jf.setVisible(true);
	}

}
```



java.awt.BorderLayout    In this layout, GUI components are placed into five regions: center, north, south, east, and west.

**Algorithm 7** Border Layout

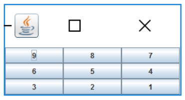
```
package exercise7;

import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Exercise_BorderLayout extends JPanel {
	private static final long serialVersionUID = 1L; // otherwise warning

	public Exercise_BorderLayout() {
		setLayout(new BorderLayout());
		add(new JButton("Norden"),BorderLayout.NORTH);
		add(new JButton("Westen"),BorderLayout.WEST);
		add(new JButton("Osten"),BorderLayout.EAST);
		add(new JButton("Süden"),BorderLayout.SOUTH);
		add(new JButton("Mitte"),BorderLayout.CENTER);
	}

	public static void main(String[] args) {
		JFrame jf = new JFrame("BorderLayout");
		jf.add(new Exercise_BorderLayout());
		jf.pack();
		jf.setVisible(true);

	}

}
```



`java.awt.GridLayout` In this layout, GUI components are placed on equal-size $N \times M$ grids. The components are placed in top-to-bottom, left-to-right order.

**Algorithm 8** Grid Layout

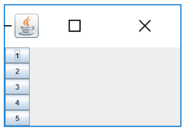
```java
package exercise7;

import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Exercise_GridLayout extends JPanel {
      private static final long serialVersionUID = 1L; // otherwise warning

      public Exercise_GridLayout(){
             setLayout(new GridLayout(3,3));
             for (int i = 9; i >= 1; --i){
             add(new JButton(new Integer(i).toString()));
             }
      }

      public static void main(String[] args) {
             JFrame jf = new JFrame("GridLayout");
             jf.add(new Exercise_GridLayout());
             jf.pack();
             jf.setVisible(true);
      }

}
```



`java.awt.BoxLayout` In this layout, GUI components are placed either vertically or horizontally.

**Algorithm 9** Box Layout

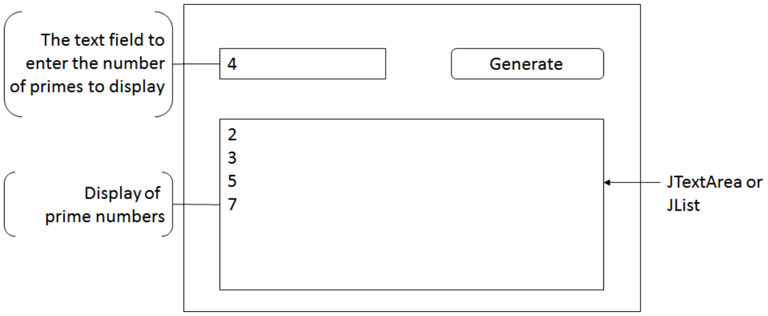
```java
package exercise7;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Exercise_BoxLayout extends JPanel {
      private static final long serialVersionUID = 1L; // otherwise warning

      public Exercise_BoxLayout() {
             this(BoxLayout.Y_AXIS); // this(BoxLayout.X_AXIS);
      }

      public Exercise_BoxLayout(int direction) {
             setLayout(new BoxLayout(this, direction));
             for(int i = 1; i <=5; ++i){
             add(new JButton(new Integer(i).toString()));
             }
      }

      public static void main(String[] args) {
             JFrame jf = new JFrame("BoxLayout");
             jf.add(new Exercise_BoxLayout());
             jf.pack();
             jf.setVisible(true);
      }

}
```



## Do It

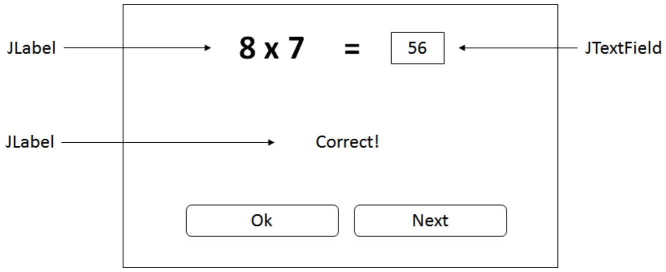
1. *Prime Numbers.*

   Using the frame layout shown, write a GUI-based program that displays $n$ prime numbers, where $n$ is a value entered by the user in the text field. If the user clicks the “Generate” button, the computed prime numbers will be displayed in the text area above.



2. *Learning Arithmetics.*

   Write a TeachArithmeticFrame class that teaches children arithmetic. The frame uses a JLabel for a problem and a JTextField for the user’s answer. If the user presses the Enter key (while the JTextField object is active) or clicks the “Ok” button, a message, stating wether the user’s answer was correct or not, will be displayed. If the user clicks the “Next” button, there will be a new problem displayed. The numbers are limited to two digits. Define a helper class that is able to generate problems.



## Vocabulary

In the following, we define several terms, which are useful for further understanding.

**absolute_positioning** GUI objects can be placed on the content pane without using any layout manager.

**layout_manager** The layout manager determines the placement of GUI objects.

## References

[1] *Java Tutorial: Creating a GUI With JFC/Swing*, 2016.

[2] C.Thomas Wu. *An Introduction to Object-Oriented Programming with Java.* McGraw Hill Higher Education, 2010.