RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

# Course Name: Object Oriented Programming
# Course Code: CS36
# Credits: 3:0:0

## Term: September – December 2020

Faculty:
Dr. Geetha J
Hanumantharaju R

RAMAIAH
Institute of Technology

# Unit-5

**Event Handling, Introducing Swing:**

Two Event Handling Mechanisms, The Delegation Event Model, Event Classes, Sources of Events, Event Listener Interfaces, Using the Delegation Event Model, Adapter Classes, Inner Classes. Swing: Introducing Swing. **Lambda Expressions:** Fundamentals, Block Lambda expressions, Passing Lambda Expressions as Argument, Lambda Expressions and Exceptions, Method References.

# Swings

**Java Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*.

It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

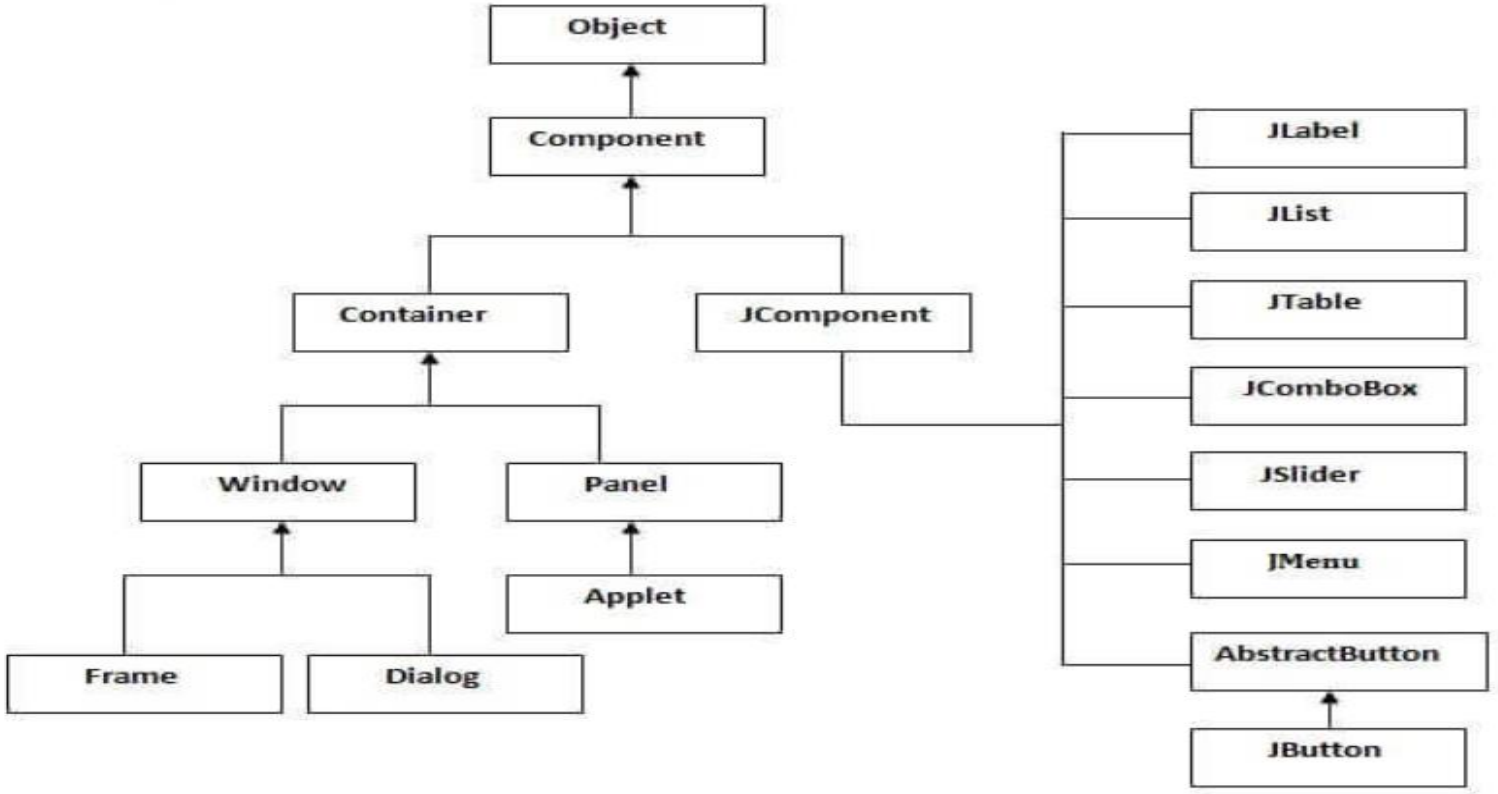
Unlike AWT, Java Swing provides platform-independent and lightweight components.

The **javax.swing package** provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

# Difference between AWT and Swing

| No. | Java AWT | Java Swing |
|---|---|---|
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| 4) | AWT provides **less components** than Swing. | Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing **follows MVC**. |

# Hierarchy of Java Swing classes



# Commonly used methods of component class

| Method | Description |
|---|---|
| public void add(Component c) | add a component on another component. |
| public void setSize(int width,int height) | sets size of the component. |
| public void setLayout(LayoutManager m) | sets the layout manager for the component. |
| public void setVisible(boolean b) | sets the visibility of the component. It is by default false. |

# Lambda Expressions

- Lambda expression is a new and important feature of Java which was included in Java SE 8.
- It provides a clear and concise way to represent one method interface using an expression.
- It is very useful in collection library. It helps to iterate, filter and extract data from collection.
- The Lambda expression is used to provide the implementation of an interface which has functional interface.
- It saves a lot of code.
- In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.
- Java lambda expression is treated as a function, so compiler does not create .class file.

# Functional Interface

- Lambda expression provides implementation of *functional interface*.
- An interface which has only one abstract method is called functional interface.
- Java provides an anotation @*FunctionalInterface*, which is used to declare an interface as functional interface.

# Why use Lambda Expression

1. To provide the implementation of Functional interface.
2. Less coding.

# Java Lambda Expression Syntax

(argument-list) -> {body}

Java lambda expression is consisted of three components.

**1) Argument-list:** It can be empty or non-empty as well.

**2) Arrow-token:** It is used to link arguments-list and body of expression.

**3) Body:** It contains expressions and statements for lambda expression.

# Java Lambda Expression Syntax

**No Parameter Syntax**

```
() -> {
//Body of no parameter lambda
}
```

**One Parameter Syntax**

```
(p1) -> {
//Body of single parameter lambda
}
```

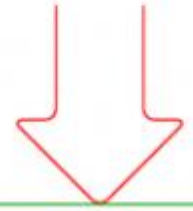
**Two Parameter Syntax**

```
1.(p1,p2) -> {
2.//Body of multiple parameter lambda

3.}
```

# Java Lambda Expression Syntax

```
Arrays.sort(dogArray, new Comparator<Dog>() {
    @Override
    public int compare(Dog o1, Dog o2) {
        return o1.getWeight() - o2.getWeight();
    }
});
```



Lambda Expression

```
Arrays.sort(dogArray, (m, n) -> m.getWeight() - n.getWeight());
```

# Thank you