SUDDHANANDA SCHOOL OF MANAGEMENT & COMPUTER SCIENCE

LECTURE NOTES ON

OBJECT ORIENTED PROGRAMMING USING JAVA (OOPS)

(MCPC1005)

UNIT – I

INTRODUCTION TO JAVA AND OBJECT-ORIENTED PROGRAMMING:

INTRODUCTION TO JAVA

Java is a high-level, object-oriented, platform-independent programming language developed by **James Gosling** and his team at **Sun Microsystems** in 1995.

Java was designed with the principle:

"Write Once, Run Anywhere (WORA)"

This means a Java program can run on any platform that has a Java Virtual Machine (JVM).

FEATURES OF JAVA

1. Simple

Java syntax is similar to C and C++ and is easy to learn.

Example

```
class Hello {
   public static void main(String args[]) {
      System.out.println("Hello World");
   }
}
```

2. Object-Oriented

Java follows Object-Oriented Programming concepts such as:

- Class
- Object
- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

3. Platform Independent

Java source code is converted into bytecode.

Bytecode can run on any operating system using JVM.

Process:

Java Program

↓

Compiler

↓

Byte Code

↓

JVM

↓

Output

4. Secure

Java provides security through:

- Bytecode verification
- Security manager
- No pointer manipulation

5. Robust

Java provides:

- Exception Handling
- Garbage Collection

which make programs more reliable.

6. Multithreaded

Java supports execution of multiple tasks simultaneously.

Example:

- Downloading files while listening to music.

7. Distributed

Java supports network programming.

Applications:

- Web applications
- Client-server systems

HISTORY OF JAVA

1991

Project Green started at Sun Microsystems.

1995

Java officially released.

Creator

James Gosling

He is known as the "Father of Java."

JAVA ENVIRONMENT

Java environment consists of:

1. JDK
2. JRE
3. JVM

1. JVM (Java Virtual Machine)

JVM executes Java bytecode.

Functions:

- Loads class files
- Executes bytecode
- Provides runtime environment

2. JRE (Java Runtime Environment)

Contains:

- JVM
- Libraries
- Supporting files

Used to run Java applications.

3. JDK (Java Development Kit)

Contains:

- JRE
- Compiler
- Development tools

Used for Java development.

Difference Between JVM, JRE and JDK

Feature	JVM	JRE	JDK
Executes Program	Yes	Yes	Yes
Contains Compiler	No	No	Yes
Used for Development	No	No	Yes

OBJECT ORIENTED PROGRAMMING (OOP)

Object-Oriented Programming is a programming methodology based on objects and classes.

It organizes software into reusable components.

ADVANTAGES OF OOP

1. Reusability
2. Security
3. Easy Maintenance
4. Modularity
5. Flexibility

BASIC CONCEPTS OF OOP

1. Class
2. Object
3. Inheritance
4. Polymorphism
5. Encapsulation
6. Abstraction

CLASS

A class is a blueprint or template for creating objects.

It defines:

- Data members
- Methods

Syntax

```
class Student {
   int roll;
   String name;
}
```

OBJECT

An object is an instance of a class.

Example

Student s1 = new Student();

Here:

- Student = Class
- s1 = Object

Example Program of Class and Object

```
class Student {
```

```
    int roll = 101;
    String name = "Alisha";

    void display() {
        System.out.println(roll + " " + name);
    }
}

class Demo {
    public static void main(String args[]) {
        Student s = new Student();
        s.display();
    }
}
```

Output

101 Alisha

CHARACTERISTICS OF OBJECT

Every object has:

1. State

Represents data.

Example:

- Name
- Age

2. Behavior

Represents functionality.

Example:

- Display()
- Calculate()

3. Identity

Unique identification of object.

METHODS IN JAVA

Methods define behavior of an object.

Syntax

```
returnType methodName() {
}
```

Example

```
void display() {
   System.out.println("Hello");
}
```

CONSTRUCTOR

A constructor is a special method used to initialize objects.

Characteristics:

- Same name as class
- No return type

Default Constructor

```
class Student {
   Student() {
      System.out.println("Constructor Called");
   }
}
```

Parameterized Constructor

```
class Student {
   Student(String n) {
      System.out.println(n);
   }
}
```

Example

```
class Student {
   Student(String name) {
      System.out.println("Name = " + name);
   }

   public static void main(String args[]) {
      Student s = new Student("Alisha");
   }
}
```

Output:

Name = Alisha

THIS KEYWORD

The this keyword refers to the current object.

Example

```
class Student {
   int roll;


   Student(int roll) {
      this.roll = roll;
   }
}
```

GARBAGE COLLECTION

Java automatically removes unused objects.

This process is called Garbage Collection.

Benefits:

1. Memory management
2. Improved performance
3. Reduced memory leaks

IDENTIFIERS IN JAVA

Identifiers are names used for:

- Variables
- Methods
- Classes

Valid Examples

student

studentName

_roll

Invalid Examples

1student

class

int

VARIABLES IN JAVA

A variable stores data.

Local Variable

Declared inside method.

```
void show() {
```

```
    int x = 10;
}
```

Instance Variable

Declared inside class but outside method.

```
class Test {
    int x = 10;
}
```

Static Variable

Shared among all objects.

```
static int count;
```

DATA TYPES IN JAVA

Primitive Data Types

Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
boolean	1 bit

Example

```
int age = 25;
double salary = 25000.50;
char grade = 'A';
```

OPERATORS IN JAVA

Operators perform operations on data.

Arithmetic Operators

+

-

*

/

%

Example:

```
int a = 10;
int b = 5;
System.out.println(a+b);
```

Relational Operators

==

!=

>

<

>=

<=

Logical Operators

&&

||

!

CONTROL STATEMENTS

Used for decision making.

if Statement

```
if(age >= 18)
{
  System.out.println("Eligible");
}
```

if-else Statement

```
if(age >=18)
{
  System.out.println("Adult");
}
else
{
  System.out.println("Minor");
}
```

switch Statement

```
switch(day)
{
 case 1:
   System.out.println("Monday");
 break;
}
```

LOOPS IN JAVA

for Loop

```
for(int i=1;i<=5;i++)
{
   System.out.println(i);
}
```

while Loop

```
while(i<=5)
{
   System.out.println(i);
   i++;
}
```

do-while Loop

```
do
{
   System.out.println(i);
}while(i<=5);
```

ARRAY IN JAVA

Array stores multiple values of same type.

Example

```
int marks[] = {10,20,30,40};
```

Advantages of Arrays

1. Easy storage
2. Fast access
3. Memory efficient

UNIT – II

INHERITANCE, POLYMORPHISM, PACKAGES AND INTERFACES

INTRODUCTION TO INHERITANCE

Inheritance is one of the most important features of Object-Oriented Programming.

Inheritance allows one class to acquire the properties and methods of another class.

The class whose properties are inherited is called:

Parent Class / Super Class / Base Class

The class which inherits the properties is called:

Child Class / Sub Class / Derived Class

NEED FOR INHERITANCE

1. Code Reusability
2. Reduces Program Length
3. Improves Maintainability
4. Faster Development
5. Supports Hierarchical Classification

REAL LIFE EXAMPLE OF INHERITANCE

Consider:

- Animal → Parent Class
- Dog → Child Class

A dog automatically inherits common properties of an animal such as:

- Eating
- Breathing

- Sleeping

SYNTAX OF INHERITANCE

```
class Parent
{
  // members
}

class Child extends Parent
{
  // inherited members
}
```

SIMPLE INHERITANCE PROGRAM

```
class Animal
{
  void eat()
  {
    System.out.println("Animal is eating");
  }
}

class Dog extends Animal
{
  void bark()
  {
    System.out.println("Dog is barking");
  }
}

class Test
{
  public static void main(String args[])
  {
    Dog d = new Dog();

    d.eat();
```

```
    d.bark();
  }
}
```

Output

Animal is eating

Dog is barking

TYPES OF INHERITANCE IN JAVA

Java supports:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

Java does not support multiple inheritance through classes.

1. SINGLE INHERITANCE

One child class inherits one parent class.

A

|

B

Example

```
class A
{
  void show()
  {
    System.out.println("Class A");
  }
}

class B extends A
{
}
```

2. MULTILEVEL INHERITANCE

A class inherits another class which further inherits another class.

A

|

B

|

C

Example

```
class A
{
  void display()
  {
    System.out.println("Class A");
  }
}

class B extends A
{
}

class C extends B
{
}
```

3. HIERARCHICAL INHERITANCE

Multiple child classes inherit from one parent class.

```
    A
   / \
  B   C
```

Example

```
class A
{
  void show()
  {
    System.out.println("Parent");
  }
}

class B extends A
{
}
```

```
class C extends A
{
}
```

WHY JAVA DOES NOT SUPPORT MULTIPLE INHERITANCE?

Multiple inheritance can create ambiguity.

Example:

If two parent classes contain the same method, Java cannot decide which method to execute.

This problem is known as:

Diamond Problem

Therefore Java avoids multiple inheritance through classes.

SUPER KEYWORD

The super keyword refers to parent class object.

Uses:

1. Access Parent Class Variable
2. Access Parent Class Method
3. Invoke Parent Constructor

Example

```
class Animal
{
  String color="White";
}

class Dog extends Animal
{
  String color="Black";

  void display()
  {
    System.out.println(super.color);
    System.out.println(color);
  }
}
```

Output:

White

Black

METHOD OVERRIDING

Method overriding occurs when child class provides its own implementation of parent class method.

Example

```
class Animal
{
  void sound()
  {
    System.out.println("Animal Sound");
  }
}

class Dog extends Animal
{
  void sound()
  {
    System.out.println("Bark");
  }
}
```

ADVANTAGES OF METHOD OVERRIDING

1. Runtime Polymorphism
2. Flexibility
3. Better Code Design

POLYMORPHISM

Polymorphism means:

"One Interface, Multiple Forms"

The same method behaves differently in different situations.

TYPES OF POLYMORPHISM

1. Compile Time Polymorphism
2. Runtime Polymorphism

1. COMPILE TIME POLYMORPHISM

Achieved using:

Method Overloading

Compiler decides which method to call.

METHOD OVERLOADING

Same method name but different parameters.

Example

```
class Addition
{
  void add(int a,int b)
  {
    System.out.println(a+b);
  }

  void add(int a,int b,int c)
  {
    System.out.println(a+b+c);
  }
}
```

Output:

30

60

Advantages

1. Improves Readability
2. Easy Programming

2. RUNTIME POLYMORPHISM

Achieved using:

Method Overriding

Decision taken during runtime.

Example

```
Animal a = new Dog();
a.sound();
```

Output:

Bark

DYNAMIC METHOD DISPATCH

Dynamic Method Dispatch is the process of resolving overridden methods during runtime.

ABSTRACT CLASS

An abstract class is a class that cannot be instantiated.

It contains:

- Abstract methods
- Concrete methods

Syntax

```
abstract class Shape
{
  abstract void draw();
}
```

Example

```
abstract class Shape
{
  abstract void draw();
}

class Circle extends Shape
{
  void draw()
  {
    System.out.println("Drawing Circle");
  }
}
```

Advantages of Abstract Class

1. Achieves Abstraction
2. Improves Security
3. Reduces Complexity

INTERFACE

An interface is a collection of abstract methods.

It provides 100% abstraction.

Syntax

```
interface Shape
{
  void draw();
}
```

Example

```
interface Shape
{
  void draw();
}

class Circle implements Shape
{
  public void draw()
  {
    System.out.println("Drawing Circle");
  }
}
```

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract Class	**Interface**
Contains abstract and normal methods	Contains abstract methods
Uses abstract keyword	Uses interface keyword
Supports constructors	No constructors
Partial abstraction	Complete abstraction

MULTIPLE INHERITANCE USING INTERFACE

Java supports multiple inheritance through interfaces.

Example

```
interface A
{
  void show();
}

interface B
{
  void display();
}

class Test implements A,B
```

```
{
  public void show()
  {
    System.out.println("Show");
  }

  public void display()
  {
    System.out.println("Display");
  }
}
```

PACKAGE IN JAVA

A package is a collection of related classes and interfaces.

Packages help organize Java programs.

ADVANTAGES OF PACKAGES

1. Code Organization
2. Reusability
3. Security
4. Easy Maintenance

TYPES OF PACKAGES

1. Built-in Packages
2. User-defined Packages

BUILT-IN PACKAGES

Provided by Java.

Examples:

- java.lang
- java.util
- java.io
- java.net

java.lang PACKAGE

Contains:

- String
- Math

- System

Automatically imported.

java.util PACKAGE

Contains:

- Scanner
- Date
- ArrayList

USER DEFINED PACKAGE

Created by programmer.

Example

package mypack;

public class Test
{
  public void display()
  {
    System.out.println("Package Example");
  }
}

IMPORT KEYWORD

Used to access package classes.

Example

import java.util.Scanner;

ACCESS SPECIFIERS IN JAVA

Access specifiers control visibility.

TYPES OF ACCESS SPECIFIERS

1. Private
2. Default
3. Protected
4. Public

1. PRIVATE

Accessible only within same class.

private int age;

2. DEFAULT

Accessible within same package.

int age;

3. PROTECTED

Accessible within package and subclasses.

protected int age;

4. PUBLIC

Accessible everywhere.

public int age;

COMPARISON OF ACCESS SPECIFIERS

Modifier	Same Class	Same Package	Subclass	Outside Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

OBJECT CLASS

Every Java class directly or indirectly inherits from Object class.

Important Methods:

1. toString()
2. equals()
3. hashCode()
4. getClass()

APPLICATIONS OF INHERITANCE AND POLYMORPHISM

1. Banking Systems
2. Employee Management Systems
3. Hospital Management Systems
4. Online Shopping Applications
5. Mobile Applications

ADVANTAGES OF OOP CONCEPTS

Inheritance

- Code Reusability
- Easy Maintenance

Polymorphism

- Flexibility
- Dynamic Behavior

Interface

- Multiple Inheritance Support
- Complete Abstraction

Packages

- Organized Programs
- Reusable Components

UNIT – III

EXCEPTION HANDLING, MULTITHREADING AND INPUT/OUTPUT STREAMS:

INTRODUCTION TO EXCEPTION HANDLING

During program execution, unexpected situations may occur that interrupt the normal flow of a program. These situations are called **Exceptions**.

An exception is an event that occurs during the execution of a program and disrupts the normal execution process.

Example:

- Dividing a number by zero
- Accessing an invalid array index
- Opening a file that does not exist

Java provides a powerful mechanism called **Exception Handling** to handle such errors gracefully.

NEED FOR EXCEPTION HANDLING

1. Prevents abnormal termination of program.
2. Maintains normal program flow.
3. Improves program reliability.
4. Helps in debugging.
5. Provides meaningful error messages.

EXCEPTION HIERARCHY IN JAVA

Object

|

Throwable

/ \

Error Exception

|

RuntimeException

TYPES OF EXCEPTIONS

1. Checked Exceptions

Checked at compile time.

Examples:

- IOException
- SQLException
- FileNotFoundException

2. Unchecked Exceptions

Occur during runtime.

Examples:

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NullPointerException

COMMON EXCEPTIONS IN JAVA

ArithmeticException

Occurs when illegal arithmetic operation is performed.

Example

class Test

```
{
  public static void main(String args[])
  {
    int a=10,b=0;
    System.out.println(a/b);
  }
}
```

Output:

ArithmeticException

ArrayIndexOutOfBoundsException

Occurs when array index exceeds array size.

Example

```
int arr[]={10,20,30};
System.out.println(arr[5]);
```

NullPointerException

Occurs when object reference is null.

Example

```
String s=null;
System.out.println(s.length());
```

EXCEPTION HANDLING KEYWORDS

Java provides five keywords:

1. try
2. catch
3. finally
4. throw
5. throws

TRY BLOCK

The code that may generate exception is placed inside try block.

Syntax

```
try
{
  // risky code
}
```

CATCH BLOCK

Handles the exception generated in try block.

Syntax

```
catch(Exception e)
{
  // handling code
}
```

Example of try-catch

```
class Demo
{
  public static void main(String args[])
  {
    try
    {
      int a=10/0;
    }
    catch(ArithmeticException e)
    {
      System.out.println("Division by Zero");
    }
  }
}
```

Output:

Division by Zero

FINALLY BLOCK

Finally block executes whether exception occurs or not.

Syntax

```
finally
{
  // cleanup code
}
```

Example

```
try
{
  int a=10/0;
```

```
}
catch(Exception e)
{
  System.out.println("Error");
}
finally
{
  System.out.println("Always Executes");
}
```

Output:

Error

Always Executes

THROW KEYWORD

Used to explicitly create an exception.

Example

```
throw new ArithmeticException("Invalid Operation");
```

THROWS KEYWORD

Used to declare exceptions.

Example

```
void display() throws IOException
{
}
```

USER DEFINED EXCEPTION

Programmer can create custom exceptions.

Example

```
class AgeException extends Exception
{
  AgeException(String msg)
  {
    super(msg);
  }
}
```

MULTIPLE CATCH BLOCKS

A single try block can have multiple catch blocks.

Example

```
try
{
}
catch(ArithmeticException e)
{
}
catch(ArrayIndexOutOfBoundsException e)
{
}
```

NESTED TRY BLOCK

A try block inside another try block.

Example

```
try
{
  try
  {
  }
  catch(Exception e)
  {
  }
}
catch(Exception e)
{
}
```

ADVANTAGES OF EXCEPTION HANDLING

1. Program Continuity
2. Better Error Management
3. Improved Reliability
4. Easier Debugging

MULTITHREADING IN JAVA

Multithreading is the process of executing multiple threads simultaneously.

A thread is the smallest unit of execution in a program.

NEED FOR MULTITHREADING

1. Better CPU utilization
2. Faster execution
3. Concurrent task processing
4. Improved performance

REAL LIFE EXAMPLES

1. Downloading files while browsing.
2. Listening to music while editing documents.
3. Running multiple applications simultaneously.

THREAD LIFE CYCLE

A thread passes through various states:

New

|

Runnable

|

Running

|

Blocked

|

Terminated

CREATING THREADS IN JAVA

Two methods:

1. Extending Thread Class
2. Implementing Runnable Interface

METHOD 1: EXTENDING THREAD CLASS

Example

```
class MyThread extends Thread
{
  public void run()
  {
    System.out.println("Thread Running");
```

```
  }
}

class Test
{
  public static void main(String args[])
  {
    MyThread t=new MyThread();
    t.start();
  }
}
```

METHOD 2: IMPLEMENTING RUNNABLE INTERFACE

Example

```
class MyThread implements Runnable
{
  public void run()
  {
    System.out.println("Thread Running");
  }
}
```

DIFFERENCE BETWEEN start() AND run()

start()	**run()**
Creates new thread	Executes normally
Multithreading occurs	No multithreading

THREAD METHODS

Common methods:

1. start()
2. run()
3. sleep()
4. join()
5. stop() (deprecated)

sleep() METHOD

Temporarily pauses thread execution.

Example

Thread.sleep(1000);

Meaning:

- Pause for 1 second

join() METHOD

One thread waits for another thread to complete.

THREAD PRIORITY

Java allows assigning priority to threads.

Range:

1 to 10

Constants:

Thread.MIN_PRIORITY

Thread.NORM_PRIORITY

Thread.MAX_PRIORITY

SYNCHRONIZATION

Synchronization prevents multiple threads from accessing shared resources simultaneously.

Purpose:

- Avoid data inconsistency

Example

```
synchronized void display()
{
}
```

ADVANTAGES OF MULTITHREADING

1. Faster Execution
2. Better Resource Utilization
3. Improved Performance
4. Concurrent Processing

INPUT/OUTPUT (I/O) STREAMS

Java provides I/O streams for reading and writing data.

I/O stands for:

Input / Output

STREAM

A stream is a sequence of data.

Two types:

1. Input Stream
2. Output Stream

INPUT STREAM

Used to read data.

Examples:

- Keyboard input
- File input

OUTPUT STREAM

Used to write data.

Examples:

- Monitor output
- File output

BYTE STREAMS

Used for binary data.

Classes:

1. InputStream
2. OutputStream

CHARACTER STREAMS

Used for character data.

Classes:

1. Reader
2. Writer

FILE HANDLING IN JAVA

File handling allows storing data permanently.

Operations:

1. Create File
2. Read File
3. Write File
4. Delete File

FILE CLASS

Used for file operations.

Example

```
File f=new File("data.txt");
```

WRITING TO FILE

Example

```
FileWriter fw=new FileWriter("data.txt");


fw.write("Hello Java");


fw.close();
```

READING FROM FILE

Example

```
FileReader fr=new FileReader("data.txt");


int ch;


while((ch=fr.read())!=-1)
{
  System.out.print((char)ch);
}
```

BUFFERED STREAMS

Provide faster file operations.

Classes:

1. BufferedReader
2. BufferedWriter

BUFFEREDREADER

Used to read text efficiently.

Example

BufferedReader br =

new BufferedReader(

new InputStreamReader(System.in));

SCANNER CLASS

Scanner class reads user input.

Package:

import java.util.Scanner;

Example

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

SERIALIZATION

Serialization converts object into byte stream.

Purpose:

- Store objects permanently
- Transfer objects over network

DESERIALIZATION

Converts byte stream back into object.

ADVANTAGES OF I/O STREAMS

1. Permanent Data Storage
2. Efficient File Processing
3. Supports Large Data Handling

APPLICATIONS OF EXCEPTION HANDLING, MULTITHREADING AND I/O

Exception Handling

- Banking Systems
- ATM Software
- Online Transactions

Multithreading

- Games
- Operating Systems
- Web Servers

I/O Streams

- File Management Systems

- Database Applications
- Data Processing Software

UNIT – IV

APPLET, AWT, SWING AND EVENT HANDLING:

INTRODUCTION TO GRAPHICAL USER INTERFACE (GUI)

A Graphical User Interface (GUI) allows users to interact with applications using graphical components such as:

- Buttons
- Text Fields
- Menus
- Check Boxes
- Radio Buttons
- Windows

Instead of typing commands, users can perform operations by clicking and selecting options.

Java provides GUI development through:

1. AWT (Abstract Window Toolkit)
2. Swing

ADVANTAGES OF GUI

1. Easy to use
2. User-friendly
3. Attractive interface
4. Faster interaction
5. Better user experience

APPLET IN JAVA

An Applet is a small Java program that runs inside a web browser or applet viewer.

Applet programs do not have a main() method.

They are executed using browser support or applet viewer.

FEATURES OF APPLET

1. Platform independent
2. Dynamic execution
3. Interactive applications
4. Graphical user interface support

LIFE CYCLE OF APPLET

Every applet follows a specific life cycle.

Important methods are:

1. init()
2. start()
3. paint()
4. stop()
5. destroy()

APPLET LIFE CYCLE DIAGRAM

init()

↓

start()

↓

paint()

↓

stop()

↓

destroy()

1. init() METHOD

Executed only once.

Purpose:

- Initialize variables
- Load resources

Example

```
public void init()
{
  setBackground(Color.yellow);
}
```

2. start() METHOD

Called after init().

Used when applet becomes active.

3. paint() METHOD

Used for drawing output on screen.

Example

```
public void paint(Graphics g)
{
  g.drawString("Hello Java Applet",50,50);
```

4. stop() METHOD

Executed when applet becomes inactive.

Example:

- User switches browser tab

5. destroy() METHOD

Called before applet is removed from memory.

Used for:

- Resource cleanup

SIMPLE APPLET PROGRAM

```
import java.applet.*;
import java.awt.*;

public class DemoApplet extends Applet
{
  public void paint(Graphics g)
  {
    g.drawString("Welcome to Java Applet",50,50);
  }
}
```

LIMITATIONS OF APPLET

1. Requires browser support
2. Security restrictions
3. Less commonly used today

AWT (ABSTRACT WINDOW TOOLKIT)

AWT is Java's first GUI toolkit.

It provides classes for creating windows and graphical components.

Package:

```
import java.awt.*;
```

FEATURES OF AWT

1. Platform dependent
2. Uses native operating system components
3. Lightweight GUI support

IMPORTANT AWT COMPONENTS

1. Label

2. Button
3. TextField
4. TextArea
5. Checkbox
6. Choice
7. List

LABEL

Used to display text.

Example

Label l = new Label("Enter Name");

BUTTON

Used to perform actions.

Example

Button b = new Button("Submit");

TEXTFIELD

Used to accept single-line input.

Example

TextField tf =

new TextField(20);

TEXTAREA

Used to accept multiple lines of text.

Example

TextArea ta =

new TextArea();

CHECKBOX

Allows selection of options.

Example

Checkbox cb =

new Checkbox("Java");

CHOICE

Provides drop-down menu.

Example

Choice c = new Choice();

LIST

Displays multiple selectable items.

Example

List l = new List();

AWT CONTAINERS

Containers hold GUI components.

Examples:

1. Frame
2. Panel
3. Dialog

FRAME

Frame is a top-level window.

Example

Frame f =

new Frame("My Window");

SIMPLE AWT PROGRAM

```
import java.awt.*;


class MyFrame
{
  MyFrame()
  {
    Frame f =
    new Frame("AWT Example");


    Button b =
    new Button("Click");


    b.setBounds(100,100,80,30);


    f.add(b);


    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
  }
```

```
  public static void main(String args[])
  {
    new MyFrame();
  }
}
```

LAYOUT MANAGERS

Layout Managers control placement of components.

Types:

1. FlowLayout
2. BorderLayout
3. GridLayout
4. CardLayout
5. GridBagLayout

FLOWLAYOUT

Places components from left to right.

Example

```
setLayout(new FlowLayout());
```

BORDERLAYOUT

Divides container into:

- North
- South
- East
- West
- Center

GRIDLAYOUT

Arranges components in rows and columns.

Example

```
setLayout(new GridLayout(2,2));
```

SWING

Swing is an advanced GUI toolkit developed as an extension of AWT.

Package:

```
import javax.swing.*;
```

FEATURES OF SWING

1. Platform independent

2. Rich GUI components
3. Lightweight components
4. Better appearance

ADVANTAGES OF SWING OVER AWT

AWT	**Swing**
Platform dependent	Platform independent
Fewer components	More components
Heavyweight	Lightweight
Less flexible	Highly flexible

IMPORTANT SWING COMPONENTS

1. JFrame
2. JButton
3. JLabel
4. JTextField
5. JTextArea
6. JCheckBox
7. JComboBox

JFRAME

Acts as main window.

Example

```
JFrame f =
new JFrame();
```

JBUTTON

Creates button component.

Example

```
JButton b =
new JButton("Submit");
```

JLABEL

Displays text.

Example

```
JLabel l =
new JLabel("Name");
```

JTEXTFIELD

Accepts user input.

Example

```
JTextField tf =
new JTextField();
```

SIMPLE SWING PROGRAM

```
import javax.swing.*;


class SwingDemo
{
  public static void main(String args[])
  {
    JFrame f =
    new JFrame("Swing");


    JButton b =
    new JButton("Click");


    b.setBounds(100,100,100,40);


    f.add(b);


    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```

EVENT HANDLING

An event is an action performed by the user.

Examples:

- Mouse click
- Keyboard press
- Button click

- Window closing

Event handling is the process of responding to user actions.

EVENT DELEGATION MODEL

Java follows Event Delegation Model.

Components:

1. Source
2. Event Object
3. Listener

SOURCE

Component that generates event.

Examples:

- Button
- TextField

EVENT OBJECT

Stores information about event.

Examples:

- ActionEvent
- MouseEvent
- KeyEvent

LISTENER

Receives event and processes it.

Examples:

- ActionListener
- MouseListener
- KeyListener

ACTION LISTENER

Used for button click events.

Method:

actionPerformed()

EXAMPLE OF EVENT HANDLING

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


class Demo extends Frame
implements ActionListener
```

```
{
  Button b;

  Demo()
  {
    b=new Button("Click");

    b.addActionListener(this);

    add(b);

    setSize(300,300);
    setVisible(true);
  }

  public void actionPerformed(ActionEvent e)
  {
    System.out.println("Button Clicked");
  }

  public static void main(String args[])
  {
    new Demo();
  }
}
```

MOUSE EVENTS

Generated when mouse is used.

Methods:

1. mouseClicked()
2. mousePressed()
3. mouseReleased()
4. mouseEntered()
5. mouseExited()

KEY EVENTS

Generated when keyboard keys are pressed.

Methods:

1. keyPressed()
2. keyReleased()
3. keyTyped()

WINDOW EVENTS

Generated when window operations occur.

Examples:

- Open
- Close
- Minimize

ADAPTER CLASSES

Adapter classes provide default implementation of listener interfaces.

Examples:

1. MouseAdapter
2. KeyAdapter
3. WindowAdapter

ADVANTAGES OF EVENT HANDLING

1. Interactive applications
2. Better user experience
3. Responsive software
4. Dynamic interfaces

MENU IN JAVA

Menus provide list of commands.

Components:

1. MenuBar
2. Menu
3. MenuItem

Example

MenuBar mb =

new MenuBar();

Menu file =

new Menu("File");

MenuItem save =

new MenuItem("Save");

DIALOG BOXES

Dialog boxes display messages.

Types:

1. Message Dialog
2. Confirm Dialog
3. Input Dialog

Example

JOptionPane.showMessageDialog

(null,"Welcome");

APPLICATIONS OF GUI PROGRAMMING

1. Banking Systems
2. Library Management
3. Student Management Systems
4. Hospital Management
5. Online Examination Systems

ADVANTAGES OF JAVA GUI

1. Easy interaction
2. Professional appearance
3. Platform independence
4. User-friendly applications