Savitribai Phule Pune University
(Formerly University of Pune)

T.Y.B.Sc.(Computer Science)

With

Major: Computer Science

(Faculty of Science and Technology)

(For Colleges Affiliated to Savitribai Phule Pune University)

Choice Based Credit System (CBCS) Syllabus Under National Education Policy (NEP)

To be implemented from Academic Year 2026-2027

Title of the Course: B.Sc.(Computer Science)

Savitribai Phule Pune University

(Formerly University of Pune)

T.Y.B.Sc.(Computer Science) Semester V

WORKBOOK

Course Code: CS-306-MJ-P

Lab Course based on

CS-301-MJ-T (Core Java)

&

CS-303-MJ-T (Web Technology-I)

WORKBOOK

T.Y.B.Sc. (Computer Science)

Core Java & Web Technology-I

Course Type: Major Core Course Code: **CS-306-MJ-P**

Course Title: **(Lab Course based on CS-301-MJ-T (Core Java)**

& CS-303-MJ-T (Web Technology-I))

SEMESTER V

Student Name:			
College:			
Roll No:		**Exam Seat No:**	
Year:		**Division:**	

BOARD OF STUDIES

1. Dr. Patil Ranjeet
2. Dr. Joshi vinayak
3. Dr. Wani Vilas
4. Dr. Mulay Prashant
5. Dr. Sardesai Anjali
6. Dr. Shelar Madhukar
7. Dr. Bharambe Manisha
8. Dr. Deshpande Madhuri
9. Dr. Kardile Vilas
10. Dr. Korpal Ritambhara
11. Dr. Gangurde Rajendra
12. Dr. Manza Ramesh
13. Dr. Bachav Archana
14. Dr. Bhat Shridhar

Co-ordinators

- **Dr. Prashant Mulay**

Annasaheb Magar College, Hadapsar , Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

- **Dr. Sardesai Anjali**

Modern College of Arts, Science and Commerce , Shivajinagar, Pune.
Member, BOS Computer Science, Savitribai Phule Pune University

CS-301-MJ-T (Core Java)

Editor

Mrs Dipali N. Shilvant	Jaikranti College of Computer Science and Management Studies, Katraj, Pune

Prepared by:

Mr. Prashant Deshmukh	PVG's College of Science and Commerce, Pune
Mrs. Shital T. Alhat	RJPM's ACS College, Landewadi, Bhosari, Pune
Mrs. Pallavi A. Joshi	Kaveri College of ASC, Pune
Mrs. Nayana Joshi	Vishwakarma College of ACS, Pune
Mrs. Swati Jadhav	Christ College, Pune
Mrs. A. L. Taskar	K.T. H. M, College, Nasik
Mrs. Rajshri Rahane	K.T. H. M, College, Nasik

CS-303-MJ-T (Web Technology-I)

Editor

Dr. A. B. Nimbalkar	Waghire College, Saswad, Pune. Member, BOS Computer Application, Savitribai Phule Pune University

Prepared by:

Dr. G. A. Kudale	Vidya Pratishthan's Arts, Science and Commerce College, M.I.D.C., Baramati, Pune-Maharashtra
Dr.C.D.Sonawane	ASM's College of Commerce, Science and Information Technology,Pimpri,Pune-18

Table of Contents for CS-306-MJ-P

INTRODUCTION

About the Workbook

This workbook is intended to be used by **T.Y.B.Sc. (Computer Science) Semester-V students** for the Practical Course based on: **CS-301-MJ-T : Core Java & CS-303-MJ-T : Web Technology-I.**

The practical course is designed to provide hands-on experience in Core Java Programming and Modern Web Technologies. The laboratory component plays a crucial role in understanding theoretical concepts through practical implementation.

This workbook is divided into two major sections:

Section I – Core Java

The first section contains assignments related to Java Programming including:

- Java Environment and Tools
- Classes and Objects
- Inheritance and Interfaces
- Exception Handling
- File Handling
- GUI Development using JavaFX

Section II – Web Technology-I

The second section contains assignments related to modern web technologies including:

- HTML5
- CSS3
- JavaScript
- DOM Manipulation
- ES6+ Features
- Asynchronous Programming
- Node.js
- REST APIs
- File System Operations

The study of programming technologies is incomplete without practical implementation. Laboratory work helps students understand concepts thoroughly and develop problem-solving abilities, logical thinking, debugging skills, and software development practices.

This workbook provides a comprehensive collection of practical assignments, exercises, mini-projects, and experiments that demonstrate the applicability of programming concepts in real-world scenarios.

Objectives of the Workbook

The objectives of this workbook are:

• To define the scope and coverage of the practical course.

• To bring uniformity in the conduct of practical sessions across affiliated colleges.

• To facilitate continuous assessment of students.

• To provide structured laboratory exercises aligned with the university syllabus.

• To encourage systematic learning through experimentation.

• To provide ready reference material for students while working in the laboratory.

• To cater to the learning needs of both slow-paced and advanced learners.

• To promote self-learning and independent problem-solving skills.

• To encourage the development of practical software development competencies.

• To bridge the gap between theoretical knowledge and practical implementation.

How to Use this Workbook?

Each assignment contains a collection of practical exercises designed to strengthen understanding of concepts.

The assignments are organized into:

Set A – Basic Experiments

These experiments cover the fundamental concepts and core implementations.

Set A is compulsory for all students.

Set B – Intermediate Experiments

These experiments extend the concepts learned in Set A and introduce additional features and variations.

Students should complete these experiments based on laboratory time availability.

Set C – Advanced Exercises and Mini Projects

These exercises are intended for self-learning and advanced practice.

Students are encouraged to solve these problems independently in the laboratory or at home to gain deeper understanding of the subject.

Instructions to Students

Please read the following instructions carefully and follow them throughout the practical course.

General Instructions

• Students must carry the workbook during every practical session.
• Students should read the relevant theory and prepare the assignment before coming to the laboratory.
• Students should maintain a separate practical journal as prescribed by the university.
• Students should complete the assigned experiments during the allotted practical session.
• Students should get their work verified by the instructor after successful execution.
• Students should spend additional time in the laboratory and at home to complete additional exercises and mini-projects.
• Students should maintain proper documentation of all practical work.
• Students should follow laboratory discipline and safety guidelines.

Assessment Scheme

Students will be assessed for each experiment on a scale of 0 to 5.

Performance Level	Marks
Not Done	0
Incomplete	1
Late Completion	2
Needs Improvement	3
Complete	4
Well Done	5

Assessment will be based on:

• Program Logic
• Program Execution
• Correct Output
• Understanding of Concepts
• Viva Performance
• Journal Maintenance

Instructions to the Practical In-Charge

The Practical In-Charge should:

• Explain the assignment objectives and related concepts.
• Demonstrate important programming techniques and tools.
• Assign appropriate experiments based on students' learning progress.
• Ensure that Set A experiments are completed by all students.
• Encourage students to complete additional exercises and mini-projects.
• Verify and evaluate each practical experiment.
• Maintain records of practical performance.
• Conduct regular viva examinations.
• Provide guidance for project-based learning activities.
• Encourage problem-solving and analytical thinking.

Instructions to Laboratory Administrator

The Laboratory Administrator should ensure:

- Availability of adequate hardware and software resources.
- Proper functioning of computer systems.
- Availability of required development tools and compilers.
- Proper network configuration where necessary.
- Regular maintenance of laboratory equipment.
- Backup and recovery mechanisms.
- Safe and secure laboratory environment.

Practical Examination Guidelines

During Internal Assessment

- Students must execute assigned programs independently.
- Journal should be completed and certified.
- Viva examination should be conducted regularly.

During University Examination

- Students must solve the assigned practical problem.
- Journal should be produced during examination.
- Internet access should not be provided.
- External storage devices should not be allowed.
- Students must explain program logic during viva.

Hardware and Software Requirements

Operating System

- Linux

Software Requirements

For Core Java

- JDK 21 (or later)
- Eclipse IDE
- NetBeans IDE
- IntelliJ IDEA Community Edition

For Web Technology-I

- Visual Studio Code
- Sublime Text
- Atom Editor
- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Runtime Environment

- Node.js (Version 18.x or Higher)
- npm (Node Package Manager)

Laboratory Outcomes

Upon successful completion of this practical course, students will be able to:

- Develop object-oriented applications using Java.
- Implement exception handling and file handling techniques.
- Design graphical user interfaces using JavaFX.
- Develop responsive websites using HTML5 and CSS3.
- Create interactive web applications using JavaScript and DOM.
- Utilize modern ES6+ programming features.
- Implement asynchronous programming techniques.
- Develop server-side applications using Node.js.
- Build RESTful APIs and perform CRUD operations.
- Apply programming concepts to solve real-world problems effectively.

Assignment Completion Sheet

Section I Core Java

Sr. No		Marks	Signature
1	An Introduction to Java		
2	Objects and Classes		
3	Inheritance and Interface		
4	Exception and File Handling		
5	User Interface with JavaFX		
a. Total out of 30			

Section II Web Technology I

Sr. No.	Assignment Name	Marks	Signature
1.	Frontend Foundations (HTML5 & CSS3)		
2.	Interactive Web Logic (Core JavaScript & DOM)		
3.	Modern Scripting with ES6+		
4.	Asynchronous Programming in JavaScript		
5.	Server-Side Basics with Node.js		
6.	File System and API Fundamentals		
b. Total out of 30			
a+b (Marks out of 60 to be converted to out of 15)			

This is to certify that **Mr/Ms** _____
University Exam Seat Number _____have successfully and satisfactorily completed and submitted the course work for **T.Y.B.Sc.(CS)** Lab course CS-306-MJ-P Semester V prescribed by Savitribai Phule Pune University during the academic year _____.

Instructor **Head of Department**

Internal Examiner **External Examiner**

Section I

Java-I

Assignment 1: An Introduction to Java (Total slots = 1)

Objectives

- Introduction to the java environment
- Use of java tools like java, javac, jdb and javadoc
- Use of IDE – Eclipse (demo)
- Use of Control Statement and Iterative Statement and Array.

Reading

You should read the following topics before starting this exercise

- Creating, compiling and running a java program.
- The java virtual machine.
- Java tools like javac, java, javadoc, javap and jdb.
- Java keywords
- Syntax of class.

Ready Reference Java Tools

1. javac:- javac is the java compiler which compiles .java file into .class file(i.e. bytecode). If the program has syntax errors, javac reports them. If the program is error-free, the output of this command is one or more .class files.

Syntax: javac fileName.java

2. java:- This command starts Java runtime environment, loads the specified .class file and executes the main method.

Syntax: java fileName

3. javadoc:- javadoc is a utility for generating HTML documentation directly from comments written in Java source code.Javadoc comments have a special form but seems like an ordinary multiline comment to the compiler.

Syntax of the comment:

/** A sample doc comment */

Syntax: javadoc [options] [packagenames] [sourcefiles] [@files]

Where,

packagenames: A series of names of packages, separated by spaces

sourcefiles: A series of source file names, separated by spaces

@files: One or more files that contain packagenames and sourcefiles in any

order, one name per line.

Javadoc creates the HTML documentation on the basis of the javadoc tags used in the source code files. These tags are described in the table below:

Tag	**Syntax**	**Description**
@see	@see reference	Allows you to refer to the documentation in other classes.
@author	@author authorinformation	Author-information contains author name, and / or author email address or any other appropriate information.
@version	@version versioninformation	Specifies the version of the program
@since	@since version	This tag allows you to indicate the version of this code that began using a particular feature.
@param	@param name	description This is used for method documentation. Here, name is the identifier in the method parameter list, and description is text that can describe the parameter.
@return	@return description	This describes the return type of a method.
@throws	@throws classname description	This is used when we handle Exceptions. It describes a particular type of exception that can be thrown from the method call.
@deprecated	@deprecated description	The deprecated tag suggests that this feature is no longer supported. A method that is marked @deprecated causes the compiler to issue a warning if it is used

4. jdb: - jdb helps you find and fix bugs in Java language programs. This debugger has limited functionality.

Syntax: jdb [options] [class] [arguments]

options : Command-line options.

class : Name of the class to begin debugging.

arguments : Arguments passed to the main() method of class.

After starting the debugger, the jdb commands can be executed. The important jdb commands are:

i. help, or?: The most important jdb command, help displays the list of recognized commands with a brief description.

ii. run: After starting jdb, and setting any necessary breakpoints, you can use this command to start the execution of the debugged application.

cont: Continues execution of the debugged application after a breakpoint, exception, or step.

iv. print: Displays Java objects and primitive values. For variables or fields of primitive types, the actual value is printed. For objects, a short description is printed.

Examples:

print MyClass.myStaticField

print myObj.myInstanceField

print i + j + k

print myObj.myMethod() //if myMethod returns non-null

v. dump: For primitive values, this command is identical to print. For objects, it prints the current value of each field defined in the object. Static and instance fields are included.

vi. next: The next command advances execution to the next line in the current stack frame.

vii. step: The step commands advances execution to the next line whether it is in the current stack frame or a called method.

Breakpoints can be set in jdb at line numbers, constructors, and the beginning of a method.

Example:

stop at MyClass:10 //sets breakpoint at instruction at line 10 of the source file containing MyClass

stop in MyClass.display // sets breakpoint at beginning of method display in MyClass

stop in MyClass.<init> //sets breakpoint at default constructor of MyClass

stop in MyClass.<init(int)> //sets breakpoint at parameterized constructor with int as Parameter

5. javap: - The javap tool allows you to query any class and find out its list of methods and constants.

javap [options] class

Example: javap java.lang.String It is a disassembler which allows the bytecodes of a class file to be viewed when used with a classname and the –c option.
javap -c class

Setting CLASSPATH

The classpath is the path that the Java runtime environment searches for classes and other resource files. The class path can be set using either the –classpath option or by setting the CLASSPATH environment variable.
The -classpath option is preferred because you can set it individually for each application without affecting other applications and without other applications modifying its value.
The default value of the class path is ".", meaning that only the current directory is searched. Specifying either the CLASSPATH variable or the -cp command line switch overrides this value.

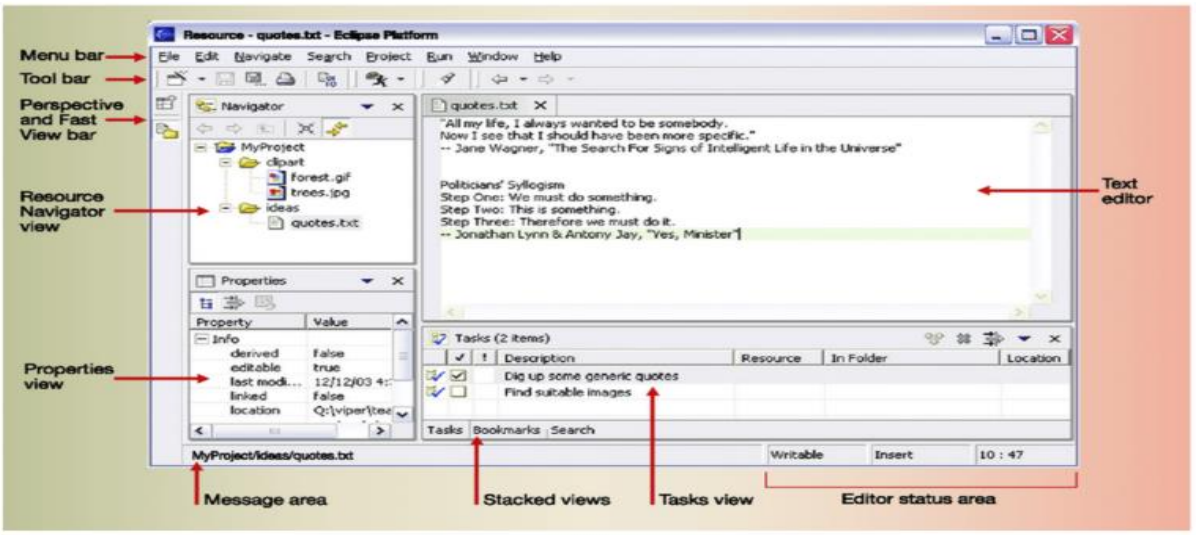
javac –classpath \myProg\myPackage; \myProg\otherclasses Or
CLASSPATH= classpath1;classpath2...
export CLASSPATH

Example
CLASSPATH=.:/usr/local/classes.jar:/home/user1/myclasses
export CLASSPATH
To retain the existing classpath setting, use $CLASSPATH in the new list.
export CLASSPATH=$CLASSPATH:/home/user1/myclasses

About Eclipse

Eclipse is a popular IDE (Integrated Development Environment) for java programming. It contains a base workspace and an extensible plug-in system for customizing the environment.
The latest Eclipse version 4.5 was released in 2015. The Eclipse IDE is also available as an IDE for other languages, ranging from C, C++ to Lua, Python, Perl and PHP.
It provides an editor, debugger, source control and other tools.

The GUI looks as shown:



Steps to run a java program using Eclipse:

1. Select workspace
2. Create a new project
3. Project name appears in package explorer, src folder contains source code files
4. Create class (New->Class)
5. Run your program (right click on the class and select Run As -> Java Application)

Self Activity (using IDE and editor)

Sample program 1 /* Program to generate documentation*/

```
/** This program demonstrates javadoc */
public class MyClass
{
int num;
/** Default constructor */
public MyClass()
{
num=0;
}
/**
Member function
@param x Represents the new value of num
@return void No return value */
public void assignValue(int x)
{
```

```
num = x;
}
}
```

.Type the following command: javadoc MyClass.java.

See the HTML documentation file

MyClass.html.

Sample program 2 : Program for while loop.

```
public class Examplewhile
{
public static void main(String[] args)
{
int i =1; // initalization of variable
while(i<=10) // check the condition
{
System.out.println(i);
i++; //increment variable by 1
}
}
}
```

Sample Program 3 : Program do while loop.

```
public class DoWhileDemo
{
public static void main(String[] args)
{
int cnt = 1;
do
{
System.out.println("Count is: " + cnt);
cnt++;
} while (cnt < 11);
}
}
```

Sample Program 4 : Program for loop.

```
public class ForDemo
{
public static void main(String[] args)
{
for(int i=1; i<=10; i++)
{
System.out.println("Value of i = : " + i +" ");
}
}
}
```

Sample Program 5 : Program for switch_case .

```
public class DemoSwitch
{
public static void main(String args[])
{
char ch = 'S';
switch (ch)
{
case 'S':
System.out.println("Sunday");
break;
case 'M':
System.out.println("Monday");
break;
case 'T':
System.out.println("Tuesday");
break;
case 'W':
System.out.println("Wednesday");
break;
case 't':
System.out.println("Thursday");
break;
```

```
case 'F':
System.out.println("Friday");
break;
case 's':
System.out.println("Saturday");
break;
}
}
}
```

Sample Program 6 : Program for 1D Array .

```
class Demo_onedimarr
{
public static void main(String args[])
{
int a[]={100,200,300,400,500}; //declaration and initialization
System.out.println("Array elements are :\n");
for(int i=0;i<a.length;i++)
{
System.out.print(" "+a[i]);
}
}
}
```

Sample Program 7 : Program for Multidimensional Array

```
class Demo_multdimarr
{
public static void main(String args[])
{
int arr[][]={{11,22,33},{44,55,66},{77,88,99}}; //declaring and initializing 2D array
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
```

```
{
System.out.print(arr[i][j]+" "); //display 2D array
}
System.out.println();
}
}
}
```

.

Sample Program 8 : Program to define a class and an object of the class

```
public class MyClass
{
int num;
public MyClass()
{
num=0;
}
public MyClass(int num)
{
this.num = num;
}
public static void main(String[] args)
{
MyClass m1 = new MyClass();
if(args.length > 0)
{
int n = Integer.parseInt(args[0]);
MyClass m2 = new MyClass(n);
System.out.println(m1.num);
System.out.println(m2.num);
}
else
System.out.println("Insufficient arguments");
}
}
```

Pass one command line argument to the above program and execute it.

Lab Assignment

Set A

a) Using javap, view the methods of the following classes from the lang package: java.lang.Object , java.lang.String and java.util.Scanner. and also Compile sample program 8. Type the following command and view the bytecodes. javap -c MyClass.

b) Write a program to calculate perimeter and area of rectangle.
(hint : area = length * breadth , perimeter=2*(length+breadth))

c) Write a menu driven program to perform the following operations
i. Calculate the volume of the cylinder. (hint : Volume: π × r2 × h)
ii. Find the factorial of a given number.
iii. Check if the number is Armstrong or not.
iv. Exit

d) Write a java program to generate the fibonacci series up to n numbers.

e) Write a program to check if a given number is prime or not.

f) Write a program to accept the array element and display in reverse order.

Set B

a) Write a java program to display the system date and time in various formats shown Below:
Current date is : 31/08/2021
Current date is : 08-31-2021
Current date is : Tuesday August 31 2021
Current date and time is : Fri August 31 15:25:59 IST 2021
Current date and time is : 31/08/21 15:25:59 PM +0530
Current time is : 15:25:59
Current week of year is : 35
Current week of month : 5
Current day of the year is : 243
Note: Use java.util.Date and java.text.SimpleDateFormat class

b) Define a class MyNumber having one private int data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value (Use this). Write methods isNegative, isPositive, isZero, isOdd, isEven. Create an object in main. Use command line arguments to pass a value to the object (Hint : convert string argument to integer) and perform the above tests. Provide javadoc comments for all constructors and methods and generates the html help file.

c) Write a menu driven program to perform the following operations on multidimensional array ie matrix :

i. Addition ii. Multiplication

iii. Transpose of any matrix. iv. Exit

d) Write a program to display the 1 to 15 tables.

(1 * 1 = 1 2 * 1 = 2 ..... 15 * 1 = 15

1 * 2 = 2 2 * 2 = 4 15 * 2 = 30

1 * 3 = 3 2 * 3 = 6 … 15 * 3 = 45

.....

1 * 10 = 10 2 * 10 = 20 …. 15 * 10 = 150)

Set C

a) What is the difference between implicit type conversion and explicit type conversion in Java? Demonstration with examples.

b) Which data type should be used for storing a person's age, annual salary, and grade respectively? Justify your answer.

c) What modifications are required to handle invalid user choices in a menu-driven application?

d) Compare the execution behavior of for, while, and do-while loops.

e) What modifications are required to find the second largest element in an array?

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 2: Objects and Classes (Total Slots = 1)

1. Objectives

By the end of this chapter, students should be able to:

- Encapsulate data and behavior within a user-defined class.
- Differentiate between instance and static members of a class.
- Apply Constructor Overloading to initialize objects in multiple ways.
- Utilize this keyword to reference current object instances.
- Organize code using Packages to improve modularity and access control.
- Manipulate Strings and StringBuffer effectively for memory efficiency.
- Understand the Object class as the root of the Java hierarchy.

2. Reading Topics

For a deep understanding, students should refer to the following:

- The Concept of 'state' and 'behavior': How variables represent state and methods represent behavior.
- Memory Allocation: Difference between Stack memory (references) and Heap memory (actual objects).
- Garbage Collection: Briefly understand how Java manages memory for objects that are no longer referenced.
- Immutable vs Mutable: Why String is immutable and when to use StringBuffer or StringBuilder.
- Access Modifiers Matrix: A clear understanding of which members are visible where (especially the protected and default distinction).

2.1 Defining Your Own Classes

A class is a blueprint or template from which objects are created. It encapsulates data (fields) and behavior (methods).

General form of a class:

```
class ClassName {
    // 1. Instance Variables
    accessSpecifier type variable1;
    accessSpecifier type variable2;
    // 2. Constructor (Used to initialize objects)
    ClassName([parameter-list]) {
        // initialization code
    }
    // 3. Methods
```

```
    accessSpecifier returnType methodName1([parameter-list]) {
        // body of method
        return value; // if returnType is not void
    }
    accessSpecifier returnType methodName2([parameter-list]) {
        // body of method
    }
}
```

Note: [] indicates optional

- **Object:** An instance of a class. Created using the new keyword.

  Syntax:

  Option A: Declaration and Instantiation (Two Steps)

  This is useful when you want to declare a variable now but create the object later based on logic.

  ```
  ClassName referenceName;          // 1. Declaration (creates a null reference)
  referenceName = new ClassName();  // 2. Instantiation & Initialization
  ```

  Option B: Combined Declaration and Instantiation (One Step)

  This is the most common approach.

  ```
  ClassName referenceName = new ClassName();
  ```

Example : Display the details of student

```
class Student {
    // Variables
    private int rollNo;
    private String name;
    // Parameterless Constructor
    public Student() {
        this.rollNo = 0;
        this.name = null;
      }
    // Parameterised Constructor
    public Student(int rollNo, String name) {
        this.rollNo = rollNo;
        this.name = name;
    }
    // Method
```

```
    public void display() {
        System.out.println("Roll No: " + rollNo + ", Name: " + name);
    }
    public static void main(String[] args) {
            // In the Main method:
            Student s1 = new Student();
            Student s2 = new Student(101, "Prashant");
            s1.display();
            s2.display();
        }
}
```

Output:
Roll No: 0, Name: null
Roll No: 101, Name: Prashant

2.2 Access Specifiers

Access specifiers (also called access modifiers) define the **scope and visibility** of classes, variables, methods, and constructors in Java.

Types of Access Specifiers

Specifier	Scope / Visibility
public	Accessible from any class, anywhere in the program.
private	Accessible only within the same class.
protected	Accessible within the same package and also in subclasses (even in different packages).
default (no specifier)	Accessible only within the same package.

Example:

```
class AccessExample {
    public int a = 10;        // public variable
    private int b = 20;       // private variable
    protected int c = 30;     // protected variable
    int d = 40;               // default access
    public void display() {
        System.out.println("Public: " + a);
        System.out.println("Private: " + b);
        System.out.println("Protected: " + c);
        System.out.println("Default: " + d);
    }
}
public class TestAccess {
    public static void main(String[] args) {
        AccessExample obj = new AccessExample();
        System.out.println("Public: " + obj.a);
```

```
        // System.out.println(obj.b);  // Error: private not accessible
        System.out.println("Protected: " + obj.c);
        System.out.println("Default: " + obj.d);
        obj.display();  // Accessing private member through public method
    }
}
```

Output:

```
Public: 10
Protected: 30
Default: 40
Public: 10
Private: 20
Protected: 30
Default: 40
```

2.3 Array of Objects

Just like an array of integers, Java allows you to create an array that stores **references to objects**.

An array of objects stores references (memory addresses) of objects of a class.

.**Syntax :**

```
ClassName[ ] arrayName = new ClassName[size];
```

Important Note

When you create an array of objects:

```
Student[] arr = new Student[5];
```

This only creates an array capable of holding 5 Student object references.
It does **NOT** create the actual Student objects.

Each element must be initialized individually:

```
arr[0] = new Student();
```

Example Program

```
class Student {
    int roll;
    String name;
    void setData(int r, String n) {
        roll = r;
        name = n;
    }
    void display() {
        System.out.println("Roll: " + roll + ", Name: " + name);
    }
}
public class ArrayOfObjectsExample {
    public static void main(String[] args) {
        // Creating array of object references
        Student[] arr = new Student[3];
        // Initializing each object
        for (int i = 0; i < 3; i++) {
```

```
        arr[i] = new Student();
    }
    // Assigning values
    arr[0].setData(1, "Rahul");
    arr[1].setData(2, "Sneha");
    arr[2].setData(3, "Amit");
    // Displaying data
    for (int i = 0; i < 3; i++) {
        arr[i].display();
    }    }    }
```

Output:

Roll: 1, Name: Rahul
Roll: 2, Name: Sneha
Roll: 3, Name: Amit

2.4 Constructors & Overloading

1. Constructor:

A constructor is a special method used to initialize objects.

- It has the same name as the class.
- It has no return type (not even void).
- It is automatically called when an object is created using the new keyword.

Types of Constructors

1. **Parameterless Constructor (Default Constructor)** - Does not take any arguments.
2. **Parameterized Constructor** - Takes parameters to initialize object data**.**

Example

```
class Student {
    int roll;
    String name;
    // Parameterless Constructor
    Student() {
        roll = 0;
        name = "Unknown";
    }
    // Parameterized Constructor
    Student(int r, String n) {
        roll = r;
        name = n;
    }
    void display() {
        System.out.println("Roll: " + roll + ", Name: " + name);
    }
}
public class ConstructorExample {
    public static void main(String[] args) {
        Student s1 = new Student();              // Calls parameterless constructor
        Student s2 = new Student(101, "Prashant"); // Calls parameterized constructor
```

```
        s1.display();
        s2.display();
    }
}
```

Output:
Roll: 0, Name: Unknown
Roll: 101, Name: Prashant

2. Constructor Overloading

When a class has **more than one constructor with different parameters**, it is called **constructor overloading**.

In the above example:

- Student()
- Student(int r, String n)

Both constructors have different parameter lists → this is **constructor overloading**.

2.5 'this' Keyword & Static Members

1. this Keyword

- this refers to the current object.
- It is mainly used to differentiate instance variables from local variables or parameters when they have the same name.
- It can also be used to call another constructor in the same class.

2. Static Members

a) Static Field (Class Variable)

- Declared using the static keyword.
- Shared by all objects of the class.
- Only one copy exists for the entire class.

b) Static Method

- Belongs to the class, not to objects.
- Can be called without creating an instance.
- Can access only static data directly.

c) Static Block

- Declared using static { }.
- Executes only once when the class is loaded into memory.
- Used for initializing static variables.

Example Program

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

```
    static String college = "ABC College";  // static field
    // Constructor using 'this' keyword
    Student(int roll, String name) {
        this.roll = roll;   // refers to instance variable
        this.name = name;
    }
    // Static method
    static void changeCollege() {
        college = "XYZ College";
    }
    void display() {
        System.out.println(roll + " " + name + " " + college);
    }
    // Static block
    static {
        System.out.println("Static block executed.");
    }
}
public class ThisStaticExample {
    public static void main(String[] args) {
        Student.changeCollege();  // calling static method without object
        Student s1 = new Student(101, "Prashant");
        Student s2 = new Student(102, "Rahul");
        s1.display();
        s2.display();
    }
}
```

Output:

```
Static block executed.
101 Prashant XYZ College
102 Rahul XYZ College
```

2.6 Predefined Classes

1. Object Class

- The Object class is the root of the class hierarchy in Java.
- Every class in Java directly or indirectly inherits from the Object class.

Important Methods of Object Class

- equals() → Compares two objects.
- toString() → Returns string representation of an object.
- hashCode() → Returns hash code value of an object.
- getClass() → Returns runtime class of an object.

Overriding toString() Method

The toString() method returns a string representation of an object.

Syntax

```
public String toString() {
        // return string representation of the object
}
```

Example

```
class Student {
   private int rollNumber;
   private String name;
   Student(int rollNumber, String name) {
      this.rollNumber = rollNumber;
      this.name = name;
   }
   @Override
   public String toString() {
      return "RollNumber = " + rollNumber + ", Name = " + name;
   }
}
public class Test {
   public static void main(String[] args) {
      Student s1 = new Student(101, "Prashant");
      System.out.println(s1);   // Automatically calls toString()
   }
}
```

Output:

RollNumber = 101, Name = Prashant

2. String Class

- Represents an **immutable sequence of characters**.
- Once created, the value of a String object cannot be changed.
- Located in java.lang package.

Example:

String str1 = "Hello";

OR

String str2=new String(“Hello”);

String Class Methods:

Method	**Purpose**
int length()	Returns the length of the string.
char charAt(int index)	Returns character at given index.
String substring(int begin)	Returns substring from given index to end.
String substring(int begin, int end)	Returns substring between given indices.

boolean equals(String s)	Compare two strings (case-sensitive).
boolean equalsIgnoreCase(String s)	Compare two strings (case-insensitive).
String toUpperCase()	Converts string to uppercase.
String toLowerCase()	Converts string to lowercase.
String trim()	Removes leading and trailing spaces.
String replace(char old, char new)	Replaces characters in string.
boolean contains(String s)	Checks if the string contains given text.
int indexOf(String s)	Returns index of first occurrence.

3.StringBuffer Class

- Represents a **mutable sequence of characters**.
- Object value can be modified after creation.
- Thread-safe (synchronized).
- Used when frequent modifications are required.

Example:

```
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");
System.out.println(sb);
```

StringBuffer Class Methods:

Method	**Purpose**
StringBuffer append(String s)	Adds string at the end.
StringBuffer insert(int offset, String s)	Inserts string at specified position.
StringBuffer replace(int start, int end, String s)	Replaces characters between indices.
StringBuffer delete(int start, int end)	Deletes characters between indices.
StringBuffer reverse()	Reverses the string.
int length()	Returns length of string.
int capacity()	Returns current buffer capacity.
char charAt(int index)	Returns character at given index.
void setCharAt(int index, char ch)	Changes character at specified index.

2.7 Packages

1. Introduction to Packages

A **package** in Java is a collection of related classes and interfaces. It is used to organize classes into namespaces, making large projects easier to manage and maintain.

Packages help to:

- Avoid name conflicts
- Provide access protection
- Organize related classes and interfaces logically
- Improve code reusability

The **Java API** is organized into several predefined packages.

The most commonly used packages are:

Package	Purpose / Use
java.lang	Basic classes like String, Math, System (auto imported).
java.util	Utility classes & collections (ArrayList, Scanner, Date).
java.io	Input and output operations, file handling.
java.net	Networking (Socket, URL, ServerSocket).
java.sql	Database connectivity (JDBC).
java.awt	GUI components (Button, Frame, Label).

Note: The java.lang package is automatically available in all Java programs. No need to import it explicitly.

2. Creating a User-Defined Package

To create a user-defined package, use the package statement at the **top of the source file**. It must be the **first line** of the program (except comments).

Syntax:

```
package packageName;
```

After writing the package statement, save the file inside a directory with the same name as the package.

Example: Creating a Package

```
package mypackage;
public class Demo {
   public void display() {
      System.out.println("This is a user-defined package.");
   }
}
```

Save this file as:

```
mypackage/Demo.java
```

3. Accessing a Package

To use classes from another package, use the import statement.

Syntax:

import packageName.*; // Imports all classes from the package

import packageName.className; // Imports only a specific class

Example: Accessing a Package

```
import mypackage.Demo;
public class Test {
    public static void main(String[] args) {
        Demo obj = new Demo();
        obj.display();    }
}
```

4. Package Hierarchy (Subpackages)

A package can contain subpackages. In such cases, the package name must be written using dot (.) notation.

Example of Hierarchical Package:

project.sourcecode.java

Here:

- project → main package
- sourcecode → subpackage inside project
- java → subpackage inside sourcecode

Example Code:

```
package project.sourcecode.java;
public class Sample {
    public void show() {
        System.out.println("Hierarchical package example.");
    }
```

```
}
Directory structure:
project/
  sourcecode/
    java/
      Sample.java
```

Access Rules:

The access rules for members for class are given in the table below.

Accessible to	**public**	**protected**	**private**	**none**
Same class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	No	Yes
Sub class (in other package)	Yes	Yes	No	No
Non sub class in other package	Yes	No	No	No

2.8 Wrapper Classes

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

These provide a way to use primitive data types (boolean, byte, char, short, int, long, float, double) as objects.

Examples: Boolean, Byte, Character, Short, Integer, Long, Float, Double

Method	**Purpose**
Byte.parseByte	Returns byte equivalent of a String
Short.parseShort	Returns the short equivalent of a String
Integer.parseInt	Returns the int equivalent of a String
Long.parseLong	Returns the long equivalent of a String
Float.parseFloat	Returns the float equivalent of a String
Double.parseDouble	Returns the double equivalent of a String

```
//Program to demonstrate Wrapping
    public class WrapperExample
    {
        public static void main(String args[])
```

```
        {
                //Converting int into Integer
                int a=20;
                Integer i = Integer.valueOf(a);//converting int into Integer
                Integer j=a; //autoboxing, now compiler will write Integer.valueOf(a)
    internally
               System.out.println(a+" "+i+" "+j);
        }
    }
```

Output: 20 20 20

//Program to demonsrtrate unwrapping

```
    public class WrapperExample2
    {
        public static void main(String args[])
        {
            //Converting Integer to int
            Integer a=new Integer(3);
            int i = a.intValue(); //converting Integer to int
            int j = a; //unboxing, now compiler will write a.intValue() internally
            System.out.println(a+" "+i+" "+j);
        }
    }
```

Output: 3 3 3

Self-Activity

Execute all Sample Programs

Sample Program 1 : To demonstrate Constructors and this keyword.

```
class MyRectangle
{
        int length; int breadth;
       MyRectangle() //Default constructor
      {
          length = 0; breadth= 0;
      }
     MyRectangle (int length, int breadth) // parameterised constructor
     {
          this.length = length; this.breadth = breadth;
     }
     int area()
     {
          return length*breadth;
     }
}
public class RectangleTest
{
    public static void main (String [] args)
   {
        MyRectangle r1 = new MyRectangle();
       System.out.println("Area = " +r1.area());
       MyRectangle r2 = new MyRectangle(10,20);
```

```
        System.out.println(“Area = ” +r2.area());
    }
}
```

Sample program 2: To create objects, demonstrate use of toString and static keyword.

```
class Student
{
        int rollNumber;
       String name;
       static String classTeacher;
      Student (int r, String n)
     {
          rollNumber = r;
         name = n;
    }
    static void assignTeacher (String name)
    {
        classTeacher = name;
    }
   public StringtoString()
  {
      return "[" + rollNumber + "," + name + "," +classTeacher +
   }
  public static void main(String[]args)
 {
       Student s1 = new Student(1,"A");
      Student s2 = new Student(2,"B");
     Student. assignTeacher(“ABC”);
     System.out.println(s1);
    System.out.println(s2);
 }
}
```

Sample program 3: To create and use packages.

```
//A.java
        package P1;
        public class A
        {
                public void display()
                {
                        System.out.println("IndisplayofA");
                }
        }
        //B.java
        package P1.P2; //P2 is a subpackage of P1
        public class B
        {
                public void display()
                {
                        System.out.println("IndisplayofB");
                }
```

```
}
//PackageTest.java
import P1.*;
import P1.P2.*;
class PackageTest
{
        public static void main (String args[])
        {
                A obj1 = new A();
                obj1.display();
                B obj2 = new B();
                Obj2.display();
        }
}
```

Create folder P1.Save A.java in folder P1.Create folder P2 in side P1.Save B.java in P2.

Lab Assignments:

Set A

a. Create an employee class (id,name,deptname,salary). Define a default and parameterized constructor. Use 'this' keyword to initialize instance variables. Keep a count of objects created. Create objects using parameterized constructors and display the object count after each object is created. (Use static member and method). Also display the contents of each object.

b. Write a program to display the Employee(Empid, Empname, Empdesignation, Empsal) information using toString().

c. Define a class MyNumber having one private int data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value (Use this). Write methods isNegative, isPositive, isZero, isOdd, isEven. Create an object in main. Use command line arguments to pass a value to the Object.

d. Write a program to define class Person with data members as Personname,Aadharno, Panno. Accept information for 5 objects and display appropriate information (use this keyword).

e. Create a class Sphere, to calculate the volume and surface area of the sphere.
(Hint : Surface area=4*3.14(r*r), Volume=(4/3)3.14(r*r*r))

Set B

a. Write a Java program to create a Package "SY" which has a class SYMarks (members ComputerTotal, MathsTotal, and ElectronicsTotal). Create another package TY which has a class TYMarks (members Theory, Practicals). Create n objects of Student class (having rollNumber, name, SYMarks and TYMarks). Add the marks of SY and TY computer subjects and calculate the Grade('A' for >= 70, 'B' for >= 60, 'C'for >= 50 , Pass Class for > =40 else "FAIL") and display the result of the student in proper format.

b.

c. Define a class CricketPlayer (name,no_of_innings, no_of_times_notout total total runs, bat_avg). Create an array of n player objects .Calculate the batting average for each player using static method avg(). Define a static sort method which sorts the array on the basis of average. Display the player details in sorted order.

d. Define a "Clock" class that does the following ;

a. Accept Hours, Minutes and Seconds
b. Check the validity of numbers
c. Set the time to AM/PM mode
Use the necessary constructors and methods to do the above task

e. Write a package for Operation, which has two classes, Addition and Maximum. Addition has two methods add () and subtract (), which are used to add two integers and subtract two, float values respectively. Maximum has a method max () to display the maximum of two Integers

Set C

a. Write a package for String operation which has two classes Con and Comp. Con class has to concatenate two strings and comp class compares two strings. Also display proper messages on execution.

b. Create four member variables for Customer class. Assign public, private, protected and default access modifiers respectively to these variables. Try to access these variables from other classes (Same package and Different package)

c. Write a program to create a package name student. Define class StudentInfo with a method to display information about students such as rollno, class, and percentage. Create another class StudentPer with a method to find the percentage of the student. Accept student details like rollno, name, class and marks of 6 subjects from the user.

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 3: Inheritance and Interfaces (Total Slots = 1)

Objectives

To improve inheritance in java.

To define abstract classes.

To define and use interfaces and functional interfaces

Reading

You should read the following topics before starting this exercise:

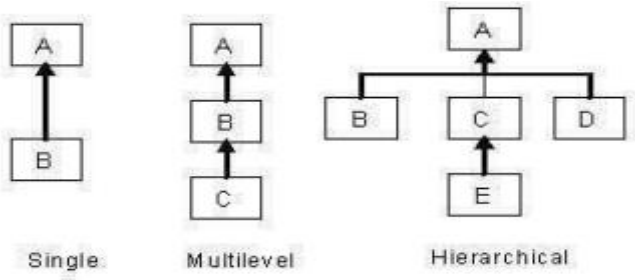
- Concept of inheritance.
- Use of extends keyword.
- Concept of abstract class.
- Defining an interface.
- Use of implements keyword.

Ready Reference

Inheriting a class:

The syntax to create sub class is:
class SubClassName extends SuperClassName
{
//class body
}
Example :
class Manager extends Employee
{
//code
}

Types of Inheritance



Access in subclass

The following members can be accessed in a subclass:

i. public or protected superclass members.

ii. Members with no specifier if the subclass is in the same package.

The"super"keyword

It is used for three purposes:

i. Invoking superclass constructor-super(arguments)

ii. Accessing superclass members–super.member

iii. Invoking superclass methods–super.method(arguments)

```
Example:
class A
{
protected int num;
A(int num)
{
this.num=num;
}
}
class B extends A
{
int num;
B(int a, int b)
{
super(a); //should be the first line in the subclass constructor
this.num=b;
}
void display()
{
System.out.println("In A, num = "+super.num);
System.out.println("In B, num = "+num);
}
}
```

Overriding methods

Redefining superclass methods in a subclass is called overriding. The signature of the subclass method should be the same as the superclass method.

Syntax

```
class A
{
void method1(int num)
{
//code
}
}
class B extends A
{
void method1(int x)
{
//code
}
}
```

Dynamic binding

When over-riding is used ,the method call is resolved during run-time i.e. depending on the object type, the corresponding method will be invoked.

Example :

```
A ref;
ref = new A();
ref.method1(10); //calls method of class
A ref = new B();
ref.method1(20); //calls method of classB
```

Abstract class

An abstract class is a class which cannot be instantiated. It is only used to create subclasses. A class which has abstract methods must be declared abstract.

An abstract class can have data members, constructors, method definitions and method declarations.

Syntax :

```
abstract class ClassName
{
...
}
```

Abstract method

An abstract method is a method which has no definition. The definition is provided by the subclass.

Syntax :

```
abstract returnTypemethod(arguments);
```

Interface

An interface is a pure abstract class i.e. it has only abstract methods and final variables.

An interface can be implemented by multiple classes.

Syntax :

```
interface InterfaceName
{
//abstract methods
//final variables
}
```

Example:

```
interface MyInterface
{
voidmethod1();
```

```
void method2();
int size=10; //finalandstatic
}
class MyClassimplements MyInterface
{
//definemethod1andmethod2
}
```

Marker Interface :

An interface that does not contain methods, fields, and constants is known as marker interface. In other words, an empty interface is known as marker interface or tag interface. It delivers the run-time type information about an object. It is the reason that the JVM and compiler have additional information about an object.

The Serializable and Cloneable interfaces are examples of marker interfaces. In short, it indicates a signal or command to the JVM. The declaration of marker interface is the same as interface in Java but the interface must be empty.

Example:

```
public interface Serializable
{
.....
}
```

FunctionalInterfaces

A functional interface in Java is an interface that contains only a single abstract (unimplemented) method. A functional interface can contain default and static methods which do have an implementation, in addition to the single unimplemented method. Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve a functional programming approach.

Syntax :

```
public interface MyFunctionalInterface()
{
// abstract method
public void functionalMethod();
}
```

Example :

The below interface is a functional interface in Java, since it only contains a single non-

```
implemented method execute()
public interface MyFunctionalInterface2
{
public void execute();
public default void print(String text)
{
System.out.println(text);
}
public static void print(String text, PrintWriter writer) throws IOException
{
writer.write(text);
}
}
```

FunctionalInterfaces Can Be Implemented by a Lambda Expression

lambda Expression

A Java lambda expression is thus a function which can be created without belonging to any class. Java lambda expressions are commonly used to implement simple event listeners / callbacks, or in functional programming with the Java Streams API.

Syntax :

lambda operator -> body

where lambda operator can be :

Zero parameter :

```
() -> System.out.println("Zero parameter lambda");
```

One parameter :

```
(t) -> System.out.println("One parameter: " + t);
```

Multiple parameters :

```
(t1, t2) -> System.out.println("Multiple parameters: " + t1 + ", " + t2);
```

OR

```
MyFunctionalInterfacefunctionalInterface = () ->
{
// basic functionality logic goes here
}
```

Example :

```
MyFunctionalInterface lambda = () ->
{
System.out.println("Executing...");
}
```

A Java lambda expression implements a single method from a Java interface. In order to know what method the lambda expression implements, the interface can only contain a single unimplemented method i.e. functional interface.

Self-Activity

Execute all the sample programs

Sample program 1 : To demonstrate Inheritance concept.

```
class Parent
{
public void p1()
{
System.out.println("Parent method");
}
}
public class Child extends Parent
{
public void c1()
{
System.out.println("Child method");
}
public static void main(String[] args)
{
Child cobj = new Child();
cobj.c1(); //method of Child class
cobj.p1(); //method of Parent class
}
}
```

Sample program 2 : To demonstrate Abstract class.

```
abstract class Bank
{
abstract int getRateOfInterest();
}
class SBI extends Bank
{
int getRateOfInterest()
{
return 7;
}
}
class PNB extends Bank
{
int getRateOfInterest()
{
return 8;
}
}
class TestBank
{
public static void main(String args[])
{
Bank b;
b=new SBI();
```

```
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}
}
```

Sample Program 3 : To illustrate Interfaces.

```
interface Result
{
finalstatic float pi-3.14f;
float compute(float x, float y);
}
class Addition implements Result
{
public float compute(float x, float y)
{
return (x+y);
}
}
class CircleArea implements Result
{
public float compute(float x, float y)
{
return (pi*x*x);
}
}
class InterfaceDemo
{
public static void main(String args[])
{
Addition a = new Addition();
CircleArea c = new CircleArea();
Result res;
res = a;
System.out.println(“Result of Addition =”+res.compute(15.2f,10));
res = c;
System.out.println(“Result of CircleArea=”+res.compute(10,2));
}
}
```

Sample program 4 : To demonstrate inheritance and interfaces.

```
interface Shape
{
double area();
}
class Circle implements Shape
{
double radius;
Circle(double radius)
```

```
{
this.radius=radius;
}
public double area()
{
return java.util.Math.PI*radius*radius;
}
class CylinderextendsCircle
{
double height;
Cylinder(double radius ,double height)
{
super(radius);
this.height=height;
}
public double area() //overriding
{
return java.util.Math.PI*radius*radius*height;
}
}
public class Test
{
public static void main (String []args)
{
Shape s;
s = new Circle(5.2);
System.out.println(“Area of circle =”+s.area());
s = new Cylinder(5,2.5);
System.out.println(“Area of cylinder=”+s.area());
} }
```

Sample program 5 : program to display of square of given number using Function Interface. (lambda expression)

```
@FunctionalInterface
interface PrintNumber
{
public void print(int num1);
}
public class PrintSquare
{
public static void main(String[] a)
{
PrintNumber p = n -> System.out.println("Square is: "+ n*n);
p.print(25);
}
}
```

NOTE : (JAVA 8 or above version)

Lab Assignments:

Set A

a) Write a program for multilevel inheritance such that the country is inherited from the continent. State is inherited from the country. Display the place, state, country and continent.

b) Write a Java program that defines an interface Calculator containing methods add() and subtract() and implements it in class SimpleCalc.

c) Define an abstract class Staff with protected membersid and name. Define a parameterized constructor. Define one subclass OfficeStaff with member department. Create n objects of OfficeStaff and display all details.

d) Define an interface "Operation" which has methods area(),volume().Define a constant PI having a value 3.142.Create a class cylinder which implements this interface (members – radius, height) Create one object and calculate the area and volume.

e) Write a Java program to implement single inheritance using classes Vehicle and Car. Include appropriate data members and methods to display vehicle information.

Set B

a) Create an abstract class "order" having members id,description.Create two subclasses "Purchase Order" and "Sales Order" having members customer name and Vendor name respectively.Define methods accept and display in all cases. Create 3 objects each of Purchase Order and Sales Order and accept and display details.

b) Write a program to using marker interface create a class product(product_id, product_name, product_cost, product_quantity) define a default and parameterized constructor. Create objects of class product and display the contents of each object and Also display the object count.

c) Write a Java program to create a super class Vehicle having members Company and Price. Derive two different classes LightMotorVehicle (mileage) and HeavyMotorVehicle (capacity_in_tons). Accept the information for "n" vehicles and display the information in appropriate form. While taking data, ask user about the type of vehicle first.

Set C

a) Create an interface Department containing attributes deptName and deptHead. It also

has abstract methods for printing the attributes. Create a class hostel containing hostelName, hostelLocation and numberOfRooms. The class contains method printing the attributes. Then write Student class extending the Hostel class and implementing the Department interface. This class contains attributes studentName, regNo, electiveSubject and avgMarks. Write a suitable printData method for this class. Also, implement the abstract methods of the Department interface. Write a driver class to test the Student class. The program should be menu driven containing the options:

i. Admit new student ii. Migrate a student

iii. Display details of a student

For the third option, a search is to be made on the basis of the entered registration Number.

b) Create an interface "CreditCardInterface" with methods: viewCreditAmount(), useCard(), payCard(), and increaseLimit(). Create a class "SolverCardCustomer" (name, cardnumber (16digit), creditamount-initialized to 0, creditLimit-set to 50,000) which implements above interface. Inherit class GoldCardCustomer from SilverCardCustomer having same methods but creditLimit of 1,00,000. Create an object of each class and perform operations. Display appropriate messages for success or failure of transaction. (Use method overloading)

i)useCard() method increase the creditAmount by a specific amount uptocreaditLimit.

ii)payCreadit() reduces the creditAmount by a specific amount.

iii)increaseLimit() increases the creaditLimit for GoldCardCustomers (only 3 times, not more than 5000 rupees each time.)

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 4: Exception Handling and File Handling (Total Slots = 1)

Objectives

• Demonstrate exception handling mechanism in java
• Defining user defined exception classes.
• Performing Input/Output operations using console and files.

Reading

You should read the following topics before starting this exercise:

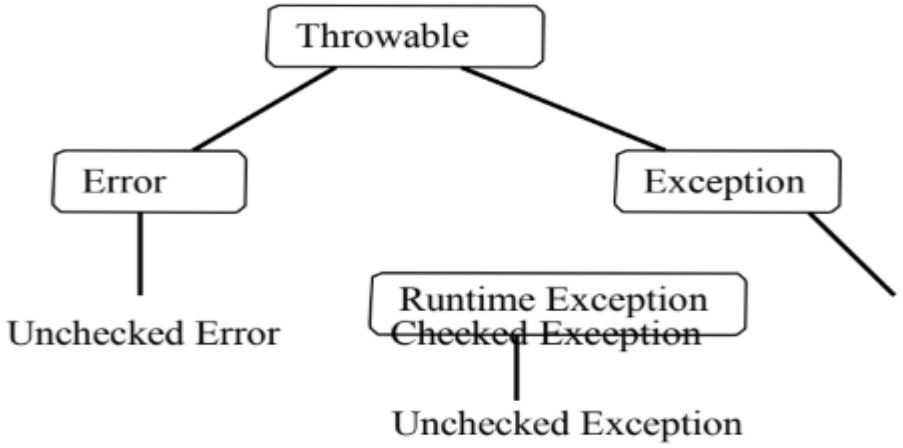
▪ Concept of Exception and Exception class hierarchy.
▪ Use of try, catch, throw, throws and finally keywords
▪ Defining user defined exception classes
▪ Concept of streams and Types of streams
▪ Byte and Character stream classes and File class .

Ready Reference

Exception : An exception is an abnormal condition that arises in a code at run time.
When an exception occurs :
1. An object representing that exception is created.
2. The method may handle the exception itself.
3. If the method cannot handle the exception, it "throws" this exception object to the method which called it.
4. The exception is "caught" and processed by some method or finally by the default java exception handler.

Predefined Exception classes

Java provides a hierarchy of Exception classes which represent an exception type



Exception Handling keywords

Exception handling in java is managed using 5 keywords :

try , catch , throw, throws, finally

```
Syntax :
try
{
// code that may cause an exception
}
```

```
catch (ExceptionType1 object)
{
// handle the exception
}
catch (ExceptionType2 object)
{
// handle the exception
}
finally
{
// this code is always executed
}
Example:
try
{
int a = Integer.parseInt(args[0]);
...
}
catch(NumberFormatException e)
{
System.out.println(“ Exception Caught” ) ;
}
```

Note: try-catch blocks can be nested.

throw keyword :

The throw keyword is used to throw an exception object or to rethrow an exception.

throw exceptionObject;

Example:

catch(NumberFormatException e)

{

System.out.println(“Caught and rethrown”) ;

throw e;

}

We can explicitly create an exception object and throw it.

For Example:

throw new NumberFormatException();

throws keyword:

If the method cannot handle the exception, it must declare a list of exceptions it may cause.

This list is specified using the throws keyword in the method header.

All checked exceptions must be caught or declared.

Syntax:

returnType methodName(arguments) throws ExceptionType1

[,ExceptionType2...]

{

}

Example:

//method body

void acceptData() throws IOException

{

//code

}

Exception Types: There are two types of Exceptions,

1. Checked exceptions

2. Unchecked exceptions

Checked exceptions must be caught or rethrown. Unchecked exceptions do not have to be caught.

Unchecked Exceptions:

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	The environment or application is in an incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to index outside the bounds of a string.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.

UnsupportedOperationException	An unsupported operation was encountered.

Checked Exceptions:

Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the Cloneable interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

User Defined Exceptions:

A user defined exception class can be created by extending the Exception class.

```
class UserDefinedException extends Exception
{
//code
}
```

When that exception situation occurs, an object of this exception class can be created and thrown.

For example, if we are accepting an integer whose valid values are only positive, then we can throw an “InvalidNumberException” for any negative value entered.

```
class NegativeNumberException extends Exception
{
NegativeNumberException(int n)
{
}
...
System.out.println("Negative input " + n);
}
```

```
public static void main(String[] args)
{
int num = Integer.parseInt(args[0]);
if( num < 0)
throw new NegativeNumberException(num);
else
}
//process num
```

java.io.File class

This class supports a platform-independent definition of file and directory names.
It also provides methods to list the files in a directory, to check the existence, readability, writeability, type, size, and modification time of files and directories, to make new directories, to rename files and directories, and to delete files and directories.

Constructors :

```
public File(String path);
public File(String path, String name);
public File(File dir, String name);
```

Example :

```
File f1=new File("/home/java/a.txt");
```

Methods

- boolean canRead()- Returns True if the file is readable.
- boolean canWrite()- Returns True if the file is writeable.
- String getName()- Returns the name of the File with any directory names omitted.
- boolean exists()- Returns true if file exists
- String getAbsolutePath()- Returns the complete filename.

Otherwise, if the File is a relative file specification, it returns the relative filename appended to the current working directory.

- String getParent() - Returns the directory of the File. If the File is an absolute specification.
- String getPath() - Returns the full name of the file, including the directory name.
- boolean isDirectory() - Returns true if File Object is a directory
- boolean isFile() - Returns true if File Object is a file
- long lastModified() - Returns the modification time of the file (which should be used

for comparison with other file times only, and not interpreted as any particular time format).

▪ long length() - Returns the length of the file.

▪ boolean delete() - deletes a file or directory. Returns true after successful deletion of a file.

▪ boolean mkdir () - Creates a directory.

▪ boolean renameTo(File dest) - Renames a file or directory. Returns true after successful renaming

Example :- **Checking file existence**

```
import java.io.File;
class FileTest
{
public static void main(String args[ ])
{
File f1=new File ("data.txt");
if (f1.exists())
System.out.println ("File Exists");
else
System.out.println ("File Does Not Exists");
}
}
```

Directories

A directory is a File that contains a list of other files & directories. When you create a File object and it is a directory, the isDirectory() method will return true. In this case list method can be used to extract the list of other files & directories inside.

The forms of list() method is

public String[ ] list()

public String[ ] list(FilenameFilter filter)

Example : Using a list( )

```
String dirname = "/javaprg/demo";
File f1 = new File (dirname);
if (f1.isDirectory())
{
String s[] = f1.list();
for (int i=0; i<s.length; i++)
{
File f = new File (dirname+"/"+s[i]);
if (f.isDirectory())
System.out.println(s[i]+ " is a directory");
else
System.out.println(s[i]+ " is a File");
```

```
}
}
```

Streams

A stream is a sequence of bytes. When writing data to a stream, the stream is called an output stream. When reading data from a stream, the stream is called an input stream. If a stream has a buffer in memory, it is a buffered stream. Binary Streams contain binary data. Character Streams have character data and are used for storing and retrieving text.

The two main types of Streams are ByteStream and CharacterStream.

There are four top level abstract stream classes: InputStream, OutputStream, Reader, and Writer.

1. InputStream. A stream to read binary data.
2. OutputStream. A stream to write binary data.
3. Reader. A stream to read characters.
4. Writer. A stream to write characters.

ByteStream Classes

a. InputStream Methods-

1. int read () - Returns an integer representation of next available byte of input.-1 is returned at the stream end.
2. int read (byte buffer[ ]) - Read up to buffer.length bytes into buffer & returns actual number of bytes that are read. At the end returns –1.
3. int read(byte buffer[ ], int offset, int numbytes) - Attempts to read up to numbytes bytes into buffer starting at buffer[offset]. Returns actual number of bytes that are read. At the end returns –1.
4. void close() - to close the input stream
5. void mark(int numbytes) - places a mark at current point in input stream & remain valid till number of bytes are read.
6. void reset() - Resets pointer to previously set mark/ goes back to stream beginning.
7. long skip(long numbytes) - skips number of bytes.
8. int available() - Returns number of bytes currently available for reading.

b. OutputStream Methods-

1.void close() - to close the OutputStream

2. void write (int b) - Writes a single byte to an output stream.

3. void write(byte buffer[ ]) - Writes a complete array of bytes to an output stream.

4.void write (byte buffer[ ], int offset, int numbytes) - Writes a sub range of numbytes bytes from the array buffer, beginning at buffer[offset].

5. void flush() - clears the buffer.

The following table lists the Byte Stream classes

Stream Class	**Meaning**
BufferedInputStream	Buffered input stream.
BufferedOutputStream	Buffered output stream.
ByteArrayInputStream	Input stream that reads from a byte array.
ByteArrayOutputStream	Output stream that writes to a byte array.
DataInputStream	An input stream that contains methods for reading the Java standard data types.
DataOutputStream	An output stream that contains methods for writing the Java standard data types.
FileInputStream	Input stream that reads from a file.
FileOutputStream	Output stream that writes to a file.
FilterInputStream	Implements InputStream.
FilterOutputStream	Implements OutputStream.
InputStream	Abstract class that describes stream input.
OutputStream	Abstract class that describes stream output.
PipedInputStream	Input pipe.
PipedOutputStream	Output pipe.
PrintStream	Output stream that contains print( ) and println( ).
RandomAccessFile	Supports random access file I/O.
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read.

CharacterStream Classes

Reader : Reader is an abstract class that defines Java's method of streaming character input.

All methods in this class will throw an IOException.

Methods in this class-

1. int read () - Returns an integer representation of next available character from

invoking stream. -1 is returned at the stream end.

2. int read (char buffer[ ]) - Read up to buffer.length chacters to buffer & returns actual number of characters that are successfully read. At the end returns –1.

3. int read(char buffer[ ], int offset, int numchars) - Attempts to read up to numchars into buffer starting at buffer[offset]. Returns actual number of characters that are read. At the end returns –1.

4. void close() - to close the input stream.

5. void mark(int numchars) - places a mark at current point in input stream & remain valid till number of characters are read.

6. void reset() - Resets pointer to previously set mark/ goes back to stream beginning.

7. long skip(long numchars) - skips number of characters.

8. int available() - Returns number of bytes currently available for reading.

Writer : Is an abstract class that defines streaming character output. All the methods in this class returns a void value & throws an IOException.

Methods in this class-

1. void close() - to close the OutputStream

2. void write (int ch) - Writes a single character to an output stream.

3. void write(char buffer[ ]) - Writes a complete array of characters to an output stream.

4. void write (char buffer[ ], int offset, int numchars) - Writes a sub range of numchars from the array buffer, beginning at buffer[offset].

5. void write(String str) - Writes str to output stream.

6. void write(String str, int offset, int numchars) - Writes a subrange of numchars from string beginning at offset.

7. void flush() - clears the buffer.

The following table lists the Character Stream classes

Stream Class	**Meaning**
BufferedReader	Buffered input character stream.
BufferedWriter	Buffered output character stream.
CharArrayReader	Input stream that reads from a character array.
CharArrayWriter	Output stream that writes to a character array.
FileReader	Input stream that reads from a file.
FileWriter	Output stream that writes to a file.
FilterReader	Filtered reader.

FilterWriter	Filtered writer.
InputStreamReader	Input stream that translates bytes to characters.
LineNumberReader	Input stream that counts lines.
OutputStreamWriter	Output stream that translates characters to bytes.
PipedReader Input	pipe PipedWriter Output pipe.
PrintWriter Output	stream that contains print( ) and println( ).
PushbackReader	Input stream that allows characters to be returned to the input stream.
Reader	Abstract class that describes character stream input.
StringReader	Input stream that reads from a string.
StringWriter	Output stream that writes to a string.
Writer	Abstract class that describes character stream output.

Random Access File

Random access files permit nonsequential, or random, access to a file's contents. To access a file randomly, you open the file, seek a particular location, and read from or write to that file. When opening a file using a RandomAccessFile, you can choose whether to open it read-only or read write.

RandomAccessFile (File file, String mode) throws FileNotFoundException

RandomAccessFile (String filePath, String mode) throws FileNotFoundExcep tion

The value of mode can be one of these:

"r" Open for reading only.

"rw" Open for reading and writing.

Methods :

1. position – Returns the current position
2. position(long) – Sets the position
3. read(ByteBuffer) – Reads bytes into the buffer from the stream
4. write(ByteBffer) – Writes bytes from the buffer to the stream
5. truncate(long) – Truncates the file (or other entity) connected to the stream

Example:

```
File f = new File("data.dat"); //Open the file for both reading and writing
RandomAccessFile rand = new RandomAccessFile(f,"rw");
rand.seek(f.length()); //Seek to end of file
rand.writeBytes("Append this line at the end"); //Write end of file
```

```
rand.close();
System.out.println("Write-Successful");
```

Self Activity

Sample program 1 : To demonstrate exceptions.

```
class NegativeNumberException extends Exception
{
NegativeNumberException(int n)
{
System.out.println(“Negative input : “ + n);
}
}
public class ExceptionTest
{
public static void main( String args[] )
{
int num, i, sum=0;
try
{
num = Integer.parseInt(args[0]);
if(num < 0)
throw new NegativeNumberException(num);
for(i=0; i<num; i++)
sum = sum+i;
}
catch(NumberFormatException e)
{
System.out.println(“Invalid format”);
}
catch(NegativeNumberException e)
{
}
finally
{
System.out.println(“The sum is : “+sum);
}
} // end main
} // end class
```

Note : Compile and run the program for different inputs like abc, -3 and 10

Sample program 2: /* Program to count occurrences of a string within a text file*/

```
import java.io.*;
import java.util.*;
public class TextFileReadApp
{
public static void main (String arg[])
{
File f = null; // Get the file from the argument line.
if (arg.length > 0)
f = new File (arg[0]);
if (f == null || !fe.exists ())
{
```

```
System.exit(0);
}
String string_to_find = arg[1];
int num_lines = 0;
try
{
FileReader file_reader = new FileReader (f);
BufferedReader buf_reader = new BufferedReader (file_reader);
// Read each line and search string

do
{
String line = buf_reader.readLine ();
if (line == null)
break;
if (line.indexOf(string_to_find) != -1)
num_lines++;
} while (true);
buf_reader.close ();
}
catch (IOException e)
{
System.out.println ("IO exception =" + e );
}
System.out.println ("No of lines containing " +

string_to_find + " = " + num_lines);

} // main
} //class TextFileReadApp
```

Sample program 3 :/* Program to write and read primitive types to a file */

```
import java.io.*;
class PrimitiveTypes
{
public static void main(String args[]) throws IOException
{
FileOutputStream fos = new FileOutputStream("info.dat");
DataOutputStream dos = new DataOutputStream(fos);
dos.writeInt(25);
dos.writeBoolean(true);
dos.writeChar('A');
dos.writeDouble(5.45);
fos.close();
FileInputStream fis = new FileInputStream("info.dat") ;
DataInputStream dis = new DataInputStream(fis);
int num = dis.readInt();
boolean b = dis.readBoolean();
char ch = dis.readChar();
double dbl = dis.readDouble();
System.out.println("Int- "+num +"\nBoolean- "+b);
System.out.println("\nCharacter- "+ch+"\nDouble- "+dbl);
```

```
fis.close();
} //main
} //class
```

Sample program 4 : /* Program to read integers from a file using Scanner class*/

```
import java.io.*;
import java.util.*;
class ReadIntegers
{
public static void main(String args[]) throws IOException
{
FileReader file = new FileReader("numbers.txt");
Scanner sc = new Scanner(file);
int sum=0, num;
while(sc.hasNext())
{
num = sc.nextInt();
System.out.println("Number = "+ num);
sum = sum+num;
}
System.out.println("The sum = "+ sum); file.close();
}//main
} //class
```

Lab Assignments:

Set A

1] Write a Java program to store five integers in an array and display an element using an index. Handle the predefined exception **ArrayIndexOutOfBoundsException** for invalid index access.

2] Define a class Student with data members student_name, student_rollno, total_lectures, attended_lectures. Create an object of Student. If attendance percentage is **less than 75%**, throw a user-defined exception **"Student is Not Eligible for Exam"**, otherwise display student details.

3] Define a class BankAccount with data members acc_no, acc_holder_name, balance.
Create an object of BankAccount. If the withdrawal amount is **greater than balance**, throw a user-defined exception **"Insufficient Balance"**, otherwise perform withdrawal and display account details.

4] Write a Java program to accept a file name from the user and count the number of **characters, words, and lines** present in the file.

Set B

a) Write a program to read book information (bookid, bookname, bookprice, bookqty) in

file "book.dat". Write a menu driven program to perform the following operations

using Random access file:

i. Search for a specific book by name.

ii. Display all book and total cost

b) Define class EmailId with members ,username and password. Define default and parameterized constructors. Accept values from the command line Throw user defined exceptions – “InvalidUsernameException” or “InvalidPasswordException” if the username and password are invalid.

c) Define a class MyDate (day, month, year) with methods to accept and display a MyDate object. Accept date as dd, mm, yyyy. Throw user defined exception “InvalidDateException” if the date is invalid.
Examples of invalid dates : 03 15 2019, 31 6 2000, 29 2 2021

Set C

a) Write a menu driven program to perform the following operations on a set of integers as shown in the following figure. A load operation should generate 10 random integers (2 digit) and display the number on screen. The save operation should save the number to a file “number.txt”. The short menu provides various operations and the result is displayed on the screen.

b) Write a java program to accept Employee name from the user and check whether it is valid or not. If it is not valid then throw user defined Exception “Name is Invalid” otherwise display it.(Name should contain only characters)

Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Assignment 5 User Interface with JavaFX (Total Slots = 2)

Objectives

1. Understanding Application Architecture
2. Practicing how to organize UI elements using different "Panes"
3. Gaining hands-on experience with common components to build interactive forms.
4. Learning how to capture and respond to user actions

You should read the following topics before starting this exercise

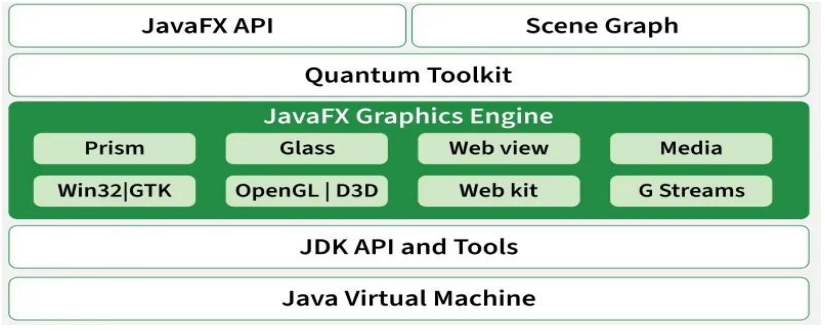
1. Know about JavaFX architecture.
2. Life cycle of JavaFX application
3. Common JavaFX Layouts
4. Containers and Components, charts
5. Event handling

JavaFX is a modern framework used to build rich desktop applications in Java. It is designed as a replacement for older GUI tools like Swing and AWT, offering better performance, modern UI JavaFX provides a complete API with a rich set of classes and interfaces to build GUI applications with rich graphics.

The important packages of this API are –

- **javafx.animation**: Contains classes to add transition-based animations such as fill, fade, rotate, scale and translate, to the JavaFX nodes.
- **javafx.application**: Contains a set of classes responsible for the JavaFX application life cycle.
- **javafx.css**: Contains classes to add CSS–like styling to JavaFX GUI applications.
- **javafx.event**: Contains classes and interfaces to deliver and handle JavaFX events.
- **javafx.geometry**: Contains classes to define 2D objects and perform operations on them.
- **javafx.stage**: This package holds the top-level container classes for JavaFX applications.
- **javafx.scene**: This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. There are several components that support this rich API of JavaFX.

JavaFX Architecture



JavaFX API: Top layer providing classes and packages for animations, UI controls, CSS styling, scene graph, events, media and application lifecycle.

Scene Graph: Core of GUI; hierarchical tree of nodes (root, branch, leaf) representing visual components.

Quantum Toolkit: Connects Prism (graphics engine) and Glass (windowing toolkit) to the API.

Prism: Hardware-accelerated 2D/3D graphics engine; falls back to software rendering if GPU is unavailable.

Glass Windowing Toolkit: Platform-dependent layer managing windows, events and OS surfaces.

WebView: Embeds web content (HTML5, CSS, JavaScript) and allows Java-JavaScript interaction.

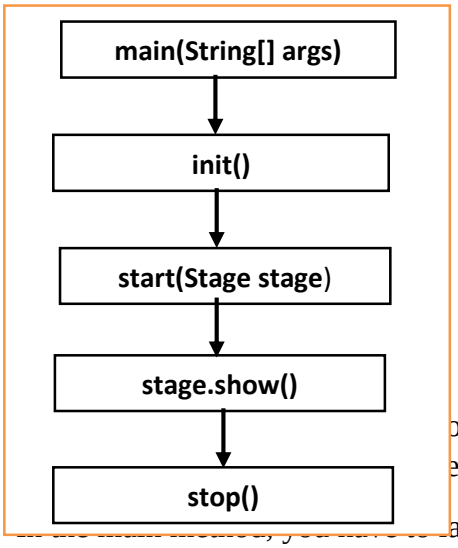
Media Engine: High-performance multimedia pipeline (GStreamer) for audio and video playback.

The basic structure of a JavaFX application follows a theater metaphor, where the user interface is organized into a hierarchical container system comprising a Stage, a Scene, and a Scene Graph.

1. **Stage (The Window):** Main window of the application. The top-level container that represents the application window. The JavaFX platform automatically creates a "primary stage" when the application starts, though developers can create additional stages for multiple windows.
2. **Scene (The Content):** A container that holds all visual content for a specific "screen" or view within the stage. A stage can only display one scene at a time, but scenes can be swapped to change what the user sees.
3. **Scene Graph (The Tree Structure):** A hierarchical tree of all visual elements (nodes) within a scene. It begins with a Root Node (usually a layout pane) which contains children like buttons, text, and other shapes.

Life Cycle of a JavaFX Application

A JavaFX application goes through four main phases, managed by the javafx.application.Application class:



init(): Runs first on a non-GUI thread; used for initialization (e.g., loading resources).

start(Stage primaryStage): Main entry point; create scenes, UI, event handlers and show the stage.

Running: Application event loop handles user interactions and animations.

stop(): Called on exit; used for cleanup and resource release.

ou abstract
e w
launch the application using the launch() method. This method internally calls the start() method of the Application class as shown in the following program.

Sample Program 1 : Basic JavaFX Program to display Hello World!

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
```

```
    @Override
    public void start(Stage primaryStage) {
        //  Create a UI element
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(event -> System.out.println("Hello World!"));

        //  Set up the layout (Root Node)
        StackPane root = new StackPane();
        root.getChildren().add(btn);

        //  Create the Scene with the layout
        Scene scene = new Scene(root, 300, 250);

        //  Configure and show the Stage
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Output:

Common JavaFX Layouts

In JavaFX, layouts (also known as Layout Panes) are container classes that manage the positioning and sizing of user interface elements (nodes). They automatically adjust the arrangement of child nodes when a window is resized based on predefined rules.

The Root Node: Every JavaFX scene must have one root node, which is typically a layout pane that holds all other components.

Package: All layout classes belong to the javafx.scene.layout package.

Base Class: The Pane class is the superclass of all layout panes.

Dynamic vs. Static:

Dynamic Layouts: Preferred for responsive applications; they automatically reposition and resize nodes.

Static Layouts (Absolute Positioning): Use the base Pane for manual X-Y coordinate placement.

Main Layout Pane

1. HBox
2. VBox
3. BorderPane
4. GridPane
5. FlowPane
6. StackPane

1. HBox(Horizontal Layout) :

Arranges nodes in a **single horizontal row**

Sample Program 2:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxDemo extends Application {
   @Override
   public void start(Stage stage) {
      HBox hbox = new HBox(20);

      Button b1 = new Button("One");
      Button b2 = new Button("Two");
      Button b3 = new Button("Three");
      hbox.getChildren().addAll(b1, b2, b3);
      Scene scene = new Scene(hbox, 300, 200);
      stage.setTitle("HBox Example");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) { launch(); }
}
```

2. VBox (Vertical Layout)

Sample Program 3:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
```

```
public class VBoxDemo extends Application {
   @Override
   public void start(Stage stage) {
      VBox vbox = new VBox(15);
      Button b1 = new Button("One");
      Button b2 = new Button("Two");
      Button b3 = new Button("Three");
      vbox.getChildren().addAll(b1, b2, b3);
      Scene scene = new Scene(vbox, 300, 200);
      stage.setTitle("VBox Example");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) { launch(); }
}
```

3. BorderPane

Sample Program 4:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class BorderPaneDemo extends Application {
   @Override
   public void start(Stage stage) {
      BorderPane bp = new BorderPane();
      bp.setTop(new Button("Top"));
      bp.setBottom(new Button("Bottom"));
      bp.setLeft(new Button("Left"));
      bp.setRight(new Button("Right"));
      bp.setCenter(new Button("Center"));

     Scene scene = new Scene(bp, 400, 300);
      stage.setTitle("BorderPane Example");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) { launch(); }
}
```

4. GridPane

Sample Program 5:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class GridPaneDemo extends Application {
   @Override
   public void start(Stage stage) {
      GridPane grid = new GridPane();
      grid.setHgap(10);
      grid.setVgap(10);
      grid.add(new Button("1"), 0, 0);
      grid.add(new Button("2"), 1, 0);
      grid.add(new Button("3"), 0, 1);
      grid.add(new Button("4"), 1, 1);

      Scene scene = new Scene(grid, 300, 200);
      stage.setTitle("GridPane Example");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) { launch(); }
}
```

5. FlowPane

Sample Program 6:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class FlowPaneDemo extends Application {
   @Override
   public void start(Stage stage) {
      FlowPane flow = new FlowPane(10, 10);
      for (int i = 1; i <= 6; i++) {
         flow.getChildren().add(new Button("Btn " + i));
      }

      Scene scene = new Scene(flow, 300, 200);
      stage.setTitle("FlowPane Example");
      stage.setScene(scene);
      stage.show();
   }
```

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

6. StackPane

Sample Program 7:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class StackPaneDemo extends Application {
    @Override
    public void start(Stage stage) {
        StackPane stack = new StackPane();
        Button bottom = new Button("Bottom");
        Button top = new Button("Top");

        stack.getChildren().addAll(bottom, top);
        Scene scene = new Scene(stack, 300, 200);
        stage.setTitle("StackPane Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) { launch(); }
}
```

7. AnchorPane

Sample Program 8

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

public class AnchorPaneDemo extends Application {
    @Override
    public void start(Stage stage) {
        AnchorPane anchor = new AnchorPane();
        Button btn = new Button("Anchored");
        AnchorPane.setTopAnchor(btn, 20.0);
```

```
        AnchorPane.setLeftAnchor(btn, 30.0);
        anchor.getChildren().add(btn);
        Scene scene = new Scene(anchor, 300, 200);
        stage.setTitle("AnchorPane Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) { launch(); }
}
```

Containers and Components

In JavaFX, the user interface is built using a hierarchy of nodes within a scene graph. These nodes are generally categorized into containers (panes) that manage layout and components (controls) that provide functionality.

Containers, often referred to as Panes, are responsible for the positioning and sizing of their child nodes. They reside in the javafx.scene.layout package.

Components, typically called Controls, are individual interactive elements that users see and touch. Most are located in the javafx.scene.control package.

Control	Description	Key Handling Methods
Label	Displays non-editable text or images.	setText(String), setGraphic(Node), setFont(Font).
Button	A clickable component to trigger actions.	setOnAction(EventHandler), setGraphic(Node), setDisable(boolean).
TextField	Single-line text input.	getText(), setText(String), setPromptText(String).
TextArea	Multi-line text input.	appendText(String), setWrapText(boolean), setPrefRowCount(int).
CheckBox	A tri-state selection box (on, off, indeterminate).	isSelected(), setSelected(boolean), selectedProperty().
RadioButton	Used for single-selection within a group.	setToggleGroup(ToggleGroup), isSelected(), selectedProperty().
ComboBox	A dropdown list for selecting one item.	getItems(), getValue(), setEditable(boolean).
ListView	Displays a scrollable list	getSelectionModel(), setItems(ObservableList), setPlaceholder(Node).

	of items.	
Slider	Selects a numeric value from a bounded range.	getValue(), setValue(double), valueProperty().
DatePicker	Allows text entry or calendar-based date selection.	getValue(), setValue(LocalDate), setShowWeekNumbers(boolean).
TableView	Visualizes data in rows and columns.	getColumns(), setItems(ObservableList), getSelectionModel().
ProgressBar	Visualizes task progress as a horizontal bar.	setProgress(double), progressProperty()

- **Menu Components**: MenuBar, Menu, and MenuItem are used to build top-level application menus.

Sample Program 9

This program demonstrates how to take user input from a TextField and display it in a Label using a vertical layout (VBox).

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class InputDemo extends Application {
   @Override
   public void start(Stage stage) {
      // UI Controls
      Label instruction = new Label("Enter your name:");
      TextField nameField = new TextField();
      Button submitBtn = new Button("Submit");
      Label outputLabel = new Label();

      // Handle the button click
      submitBtn.setOnAction(e -> {
         String name = nameField.getText();
         outputLabel.setText("Welcome, " + name + "!");
      });

      // Layout: VBox (Vertical Box) with 10px spacing
      VBox layout = new VBox(10);
      layout.setAlignment(Pos.CENTER);
      layout.getChildren().addAll(instruction, nameField, submitBtn, outputLabel);
```

```
        stage.setScene(new Scene(layout, 300, 200));
        stage.setTitle("User Input Demo");
        stage.show();
    }

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

Sample Program 10

The following JavaFX program demonstrates the use of common UI components including Button, Label, Text, TextArea, CheckBox, RadioButton, ListView, ComboBox, and Menu.

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class JavaFXShowcase extends Application {

    @Override
    public void start(Stage primaryStage) {
        // 1. Label and Text
        Label label = new Label("This is a Label (Non-editable text):");
        Text textGraphic = new Text("This is a Text object (Visual node).");

        // 2. Button
        Button button = new Button("Click Me");
        button.setOnAction(e -> System.out.println("Button Clicked!"));

        // 3. TextArea (Multi-line input)
        TextArea textArea = new TextArea("This is a TextArea.\nYou can write
multiple lines here.");
        textArea.setPrefRowCount(3);

        // 4. CheckBox
        CheckBox checkBox = new CheckBox("I agree to terms");

        // 5. RadioButtons (Requires a ToggleGroup to ensure single selection)
        ToggleGroup group = new ToggleGroup();
        RadioButton rb1 = new RadioButton("Option A");
```

```
        rb1.setToggleGroup(group);
        RadioButton rb2 = new RadioButton("Option B");
        rb2.setToggleGroup(group);

        // 6. ListView (Scrollable list)
        ListView<String> listView = new
    ListView<>(FXCollections.observableArrayList("Item 1", "Item 2", "Item 3"));
        listView.setPrefHeight(100);

        // 7. ComboBox (Dropdown list)
        ComboBox<String> comboBox = new
    ComboBox<>(FXCollections.observableArrayList("Choice 1", "Choice 2", "Choice
    3"));
        comboBox.setPromptText("Select an option");

        // 8. Menus (MenuBar -> Menu -> MenuItem)
        MenuBar menuBar = new MenuBar();
        Menu fileMenu = new Menu("File");
        MenuItem exitItem = new MenuItem("Exit");
        exitItem.setOnAction(e -> System.exit(0));
        fileMenu.getItems().add(exitItem);
        menuBar.getMenus().add(fileMenu);

        // Layout: Organize components vertically
        VBox layout = new VBox(10); // 10px spacing
        layout.setPadding(new Insets(15));
        layout.getChildren().addAll(menuBar, label, textGraphic, button, textArea,
        checkBox, rb1, rb2, listView, comboBox);

        // Scene and Stage
        Scene scene = new Scene(layout, 400, 600);
        primaryStage.setTitle("JavaFX UI Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
      }
      public static void main(String[] args)
      {
        launch(args);
      }
}
```

JavaFX Charts

JavaFX provides a robust API for creating data visualizations through the javafx.scene.chart package. The API is divided into two main categories: **PieChart** (no-axis) and **XYChart** (two-axis charts like Bar, Line, and Area).

Common JavaFX Chart Types

- **PieChart**: Used for representing portions of a whole.
- **BarChart**: Displays grouped data using rectangular bars; can be horizontal or vertical.
- **LineChart**: Connects data points with lines to show trends over time or categories.
- **AreaChart**: Similar to a line chart but fills the area between the line and the axis with color.

- **BubbleChart**: Represents three-dimensional data using the (X, Y) position and the bubble's radius.
- **ScatterChart**: Plots individual data points to show relationships or distributions.
- **StackedBarChart / StackedAreaChart**: Variation of bar/area charts where different groups are stacked to show cumulative totals.

General Steps to Create a JavaFX Chart

1. **Define the Axes**
   For 2D charts (like Bar, Line, or Scatter), you must first define the X and Y axes.
   - **Numerical Data**: Use NumberAxis. You can set range and tick units in the constructor: new **NumberAxis(lower, upper, unit).**
   - **Categorical Data**: Use CategoryAxis for non-numerical labels like months or names.
   - **Labels**: Use the **setLabel(String)** method to title each axis.
2. **Instantiate the Chart Class**
   Create the specific chart object and pass your defined axes to the constructor.
   - **Examples**: **new LineChart(xAxis, yAxis), new BarChart(xAxis, yAxis), or new PieChart()** (Pie charts do not require separate axes).
2. **Prepare and Add Data**
   Data is typically managed through the **XYChart.Series** and **XYChart.Data** classes for 2D charts, or **PieChart.Data** for circular charts.
   - **Create a Series**: **XYChart.Series series = new XYChart.Series();.**
   - **Set Series Name**: Use **series.setName("Name")** for the legend.
   - **Add Data Points**: Use **series.getData().add(new XYChart.Data(xValue, yValue))** to populate the series.
   - **Bind to Chart**: Add the series to the chart using **chart.getData().add(series)** or **addAll(series1, series2).**
2. **Configure UI and Launch**
   Add the chart to a layout pane (like Group, VBox, or StackPane), set the scene, and show the stage.
   - **Layout**: Group root = new Group(chart);.
   - **Scene**: Scene scene = new Scene(root, width, height);.
   - **Stage**: primaryStage.setScene(scene); followed by primaryStage.show();

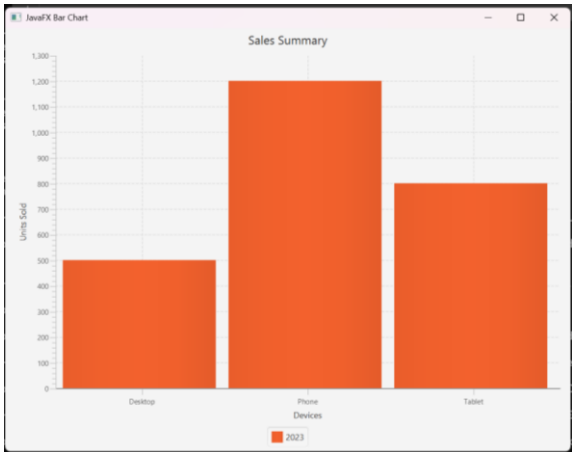
Methods & Properties

1.	setTitle(String)	Sets the title text for the chart.
2.	getData()	Retrieves the ObservableList of data series for the chart.
3.	setAnimated(boolean)	Toggles whether data changes are animated (default is often true).
4.	setLegendVisible(boolean)	Determines if the chart's legend should be displayed.
5.	setLabel(String)	Used on axis objects (e.g., NumberAxis or CategoryAxis) to name the axes.

Sample Program 11 : Bar chart

```
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class BarChartExample extends Application {
   @Override
   public void start(Stage stage) {
      stage.setTitle("JavaFX Bar Chart");
      CategoryAxis xAxis = new CategoryAxis();
      xAxis.setLabel("Devices");
      NumberAxis yAxis = new NumberAxis();
      yAxis.setLabel("Units Sold");

      BarChart<String, Number> barChart = new BarChart<>(xAxis, yAxis);
      barChart.setTitle("Sales Summary");
      XYChart.Series<String, Number> series = new XYChart.Series<>();
      series.setName("2023");
      series.getData().add(new XYChart.Data<>("Desktop", 500));
      series.getData().add(new XYChart.Data<>("Phone", 1200));
      series.getData().add(new XYChart.Data<>("Tablet", 800));
      barChart.getData().add(series);
      stage.setScene(new Scene(barChart, 800, 600));
      stage.show();
   }
   public static void main(String[] args) { launch(args); }
}
```

Output

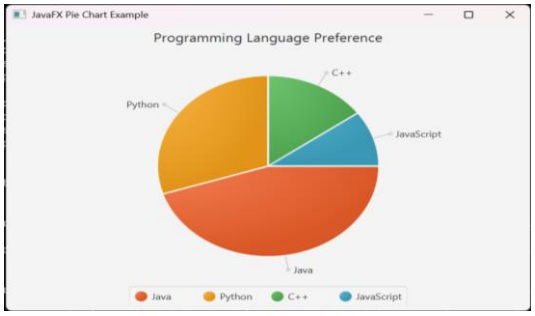


Sample program 12 : Pie chart

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.chart.PieChart;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class SimplePieChart extends Application {
    @Override
    public void start(Stage stage) {
        // 1. Prepare Data
        ObservableList<PieChart.Data> pieChartData =
                FXCollections.observableArrayList(
                new PieChart.Data("Java", 45),
                new PieChart.Data("Python", 30),
                new PieChart.Data("C++", 15),
                new PieChart.Data("JavaScript", 10));
        // 2. Create Chart
        PieChart pieChart = new PieChart(pieChartData);
        pieChart.setTitle("Programming Language Preference");
        // 3. Set up Layout and Scene
        StackPane root = new StackPane(pieChart);
        Scene scene = new Scene(root, 600, 400);
```

```
        // 4. Configure and Show Stage
        stage.setTitle("JavaFX Pie Chart Example");
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Output



Event Handling: Adding and Removing Event Handlers

An event is a notification about a change. It encapsulates the state changes in the event source. Registered event filters and event handlers within the application receive the event and provide a response.

JavaFX provides support to handle events through the base class "**Event**" which is available in the package **javafx.event.**

Examples of Events:

- **Action Event** — widely used to indicate things like when a button is pressed.
  - Class:- ActionEvent
  - Actions:- button pressed.
- **Mouse Event** — occurs when mouse is clicked
  - Class:- MouseEvent
  - Actions:- mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target.
- **Drag Event** — occurs when the mouse is dragged.
  - Class:- DragEvent
  - Actions:- drag entered, drag dropped, drag entered target, drag exited target,
  - drag over.
  - Key Event — indicates that a keystroke has occurred.
- Class:- KeyEvent
- Actions:- Key pressed, key released and key typed.
- Window Event:

- Class:- WindowEvent
- Actions:- window hiding, window shown, window hidden, window showing.
- Scroll Event — indicates scrolling by mouse wheel, track pad, touch screen, etc...
- TouchEvent — indicates a touch screen action

Three methods for Event Handling:

1. Convenience Methods:

- setOnKeyPressed(eventHandler);
- setOnMouseClicked(eventHandler);

2. Event Handler/Filter Registration Methods:

- addEventHandler(eventType, eventHandler);
- addEventFilter(eventType, eventFilter);

3. Event Dispatcher Property (lambda expression).

Adding Event-Handler to a node:

To register the event handler for a node, **addEventHandler()** method is used.

Syntax:

```
node.addEventHandler (<Event_Type>, new EventHandler<Event-Type>()
{
public void handle(<Event-Type> e)
{
//Handling Code
});
```

Where,
First argument is the type of event that is generated.
Second argument is the filter to handle the event.

Removing Event-Filter:

We can remove an event handler on a node using removeEventHandler() method.

Syntax:

```
node.removeEventHandler(<EventType>, handler);
```

A node can register for more than one Event Filters and Handlers.

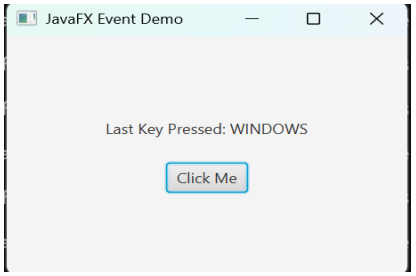
The interface javafx.event.EventHanler must be implemented by all the event filters and

event handlers.

Sample Program 13: This program demonstrates the three primary event types—**Action**, **Mouse**, and **Key**—using lambda expressions.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.geometry.Pos;

public class JavaFXEventDemo extends Application {

    @Override
    public void start(Stage primaryStage) {
        // 1. Create UI components
        Label label = new Label("Interact with the UI!");
        Button btn = new Button("Click Me");

        // 2. Action Event: Triggered when the button is clicked
        btn.setOnAction(e -> {
            label.setText("Status: Button Clicked!");
        });

        // 3. Mouse Events: Change button style on hover
        btn.setOnMouseEntered(e -> btn.setStyle("-fx-background-color: lightblue;"));
        btn.setOnMouseExited(e -> btn.setStyle(""));

        // 4. Layout setup
        VBox root = new VBox(20, label, btn);
        root.setAlignment(Pos.CENTER);
        Scene scene = new Scene(root, 300, 200);

        // 5. Key Event: Detect key presses on the entire scene
        scene.setOnKeyPressed(e -> {
            label.setText("Last Key Pressed: " + e.getCode());
        });

        primaryStage.setTitle("JavaFX Event Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Output:

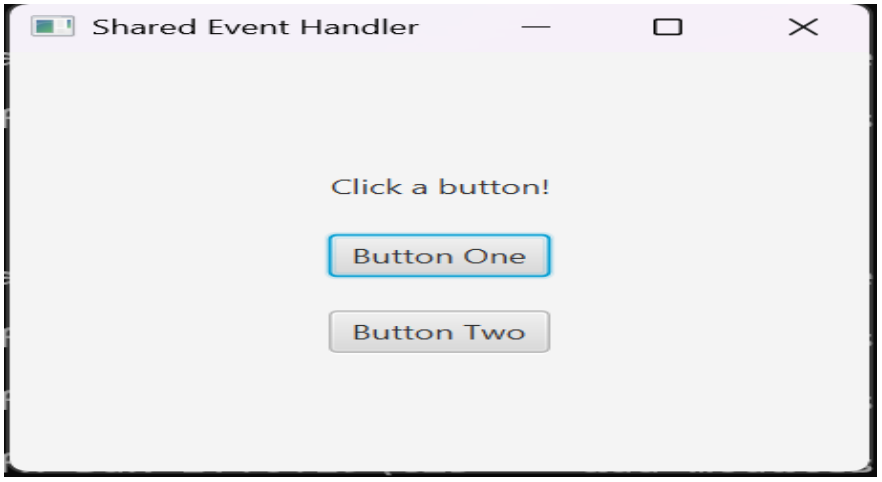


Sample Program 14: program using a single **Anonymous Inner Class** to manage multiple buttons:

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SharedHandlerDemo extends Application {

    private Button btn1;
    private Button btn2;
    private Label label;

    @Override
    public void start(Stage primaryStage) {
        label = new Label("Click a button!");
        btn1 = new Button("Button One");
        btn2 = new Button("Button Two");

        // 1. Create a single shared event handler
        EventHandler<ActionEvent> sharedHandler = new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                // Use getSource() to identify which button was clicked
                if (event.getSource() == btn1) {
                    label.setText("Status: First Button Clicked!");
                } else if (event.getSource() == btn2) {
                    label.setText("Status: Second Button Clicked!");
                }
            }
        };
// 2. Register the same handler instance to both buttons
        btn1.setOnAction(sharedHandler);
        btn2.setOnAction(sharedHandler);
```

```
        VBox root = new VBox(20, label, btn1, btn2);
        root.setAlignment(Pos.CENTER);
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Shared Event Handler");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Output:



Lab Assignments:

SET A

a. Write a JavaFX application that changes the background color of a window based on a combination of keys pressed (e.g., Ctrl+Alt+O changes the background to orange).

b. Write a JavaFX application that simulates a shopping cart. Add items with buttons, and implement action listeners to update the cart when an item is added.

c. Write a JavaFX login form. Implement an action listener on the "Login" button to validate user credentials.

d. Write a JavaFX program that takes a numeric input from a user via a TextField and displays whether it is a prime number in an output field upon clicking a "Process" button.

e. Write a JavaFX program to build a simple calculator with digit buttons (0-9) and basic arithmetic operator buttons (+, -, *, /). Implement action listeners for each button (digits and operators) to perform calculations.

SET B

a. Build a PieChart that displays a breakdown of expenses. Implement an event handler so that when a user clicks a slice of the pie, a Label updates to show the exact percentage or value of that category.

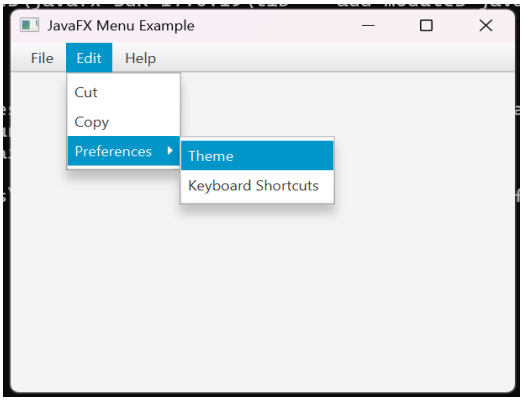
b. Design a screen to handle the Mouse Events such as MOUSE_MOVED and MOUSE_CLICK and display the position of the Mouse_Click in a TextField.

c. Creating an application with multiple buttons (e.g., Red, Green, Blue) that print their respective color or change the background color when clicked.

SET C

a. Develop a GUI that allows a user to select a CSV or text file via a FileChooser. The program must parse the data and display it as one of four chart types (e.g., Area, Bar, Line, or Stacked) selectable from a Menu

b. Develop a menu and submenu as shown below



Assignment Evaluation

0: Not Done		1: Incomplete		2:Late Complete	
3: Needs Improvement		4: Complete		5: Well Done	

Practical In-charge

Section II

CS-303-MJ-T: Web Technology-I.

CS-303-MJ-T : Web Technology-I.

1. FRONTEND FOUNDATIONS (HTML5 & CSS3) (Total Slots = 1)

Assignment No. 1

Title Frontend Foundations (HTML5 & CSS3)

Duration 01 Practical Slot

Course Outcome Mapping

CO4: Create responsive, multi-column web layouts using Flexbox and Media Queries.

1 Aim

To design and develop structured, responsive, and visually appealing web pages using HTML5 and CSS3.

2 Learning Objectives

After completing this experiment, students will be able to:

1. Design webpages using HTML5.
2. Create forms and multimedia webpages.
3. Apply CSS styling techniques.
4. Use JavaScript for validation and DOM manipulation.
5. Develop responsive web pages.
6. Create dynamic client-side applications.
7. Build mini web projects using HTML, CSS, and JavaScript.

3 Introduction to HTML5

HTML (HyperText Markup Language) is the standard language used to create web pages.

HTML5 is the latest version providing:

- Semantic Elements
- Audio and Video Support
- Form Enhancements
- Canvas Graphics
- Improved Accessibility

4 Structure of HTML5 Document

```
<!DOCTYPE html>
<html>
<head>
   <title>My Web Page</title>
</head>
<body>
<h1>Welcome to HTML5</h1>
</body>
</html>
```

5 Basic HTML Elements

Tag	Purpose
<html>	Root element
<head>	Metadata
<title>	Page title
<body>	Visible content
<h1> - <h6>	Headings
<p>	Paragraph
<a>	Hyperlink
<img>	Image
<table>	Table

<form>	Form

6 Semantic HTML Elements

Semantic elements describe the meaning of content.

Element	Purpose
<header>	Page header
<nav>	Navigation
<section>	Content section
<article>	Independent content
<aside>	Sidebar
<footer>	Footer

Example

```
<header>
   <h1>College Website</h1>
</header>
<nav>
   Home | About | Contact
</nav>
<section>
   Welcome to College Portal
</section>
<footer>
   Copyright 2025
</footer>
```

7 HTML Forms : Forms collect user information.

Common Form Controls

- Text Box
- Password Box
- Radio Button
- Checkbox
- Drop-down List
- Submit Button

8 Tables in HTML

Example

```
<table border="1">
<tr>
   <th>Roll No</th>
   <th>Name</th>
</tr>
<tr>
   <td>101</td>
   <td>Rahul</td>
</tr>
</table>
```

9 Multimedia in HTML5

Audio

```
<audio controls>
   <source src="song.mp3">
</audio>
```

Video

```
<video width="400" controls>
   <source src="movie.mp4">
</video>
```

10 Introduction to CSS3 :CSS (Cascading Style Sheets) controls presentation and appearance of web pages.

Advantages

- Improved Design

- Better User Experience
- Responsive Layouts
- Reduced Code Duplication

11 CSS Syntax

```
selector
{
   property:value;
}
```

Example

```
h1
{
   color:blue;
}
```

12 CSS Selectors

Element Selector

```
p
{
color:red;
}
```

Class Selector

```
.title
{
color:green;
}
```

ID Selector

```
#header
{
background:yellow;
}
```

13 CSS Box Model

Every HTML element consists of:

```
Margin
 └── Border
     └── Padding
         └── Content
```

14 Flexbox Layout

Flexbox provides flexible responsive layouts.

Example

```
.container
{
display:flex;
justify-content:space-around;
}
```

15 CSS Grid Layout

Grid provides two-dimensional layout control.

Example

```
.container
{
display:grid;
grid-template-columns:
1fr 1fr 1fr;
}
```

16 Media Queries

Used for responsive web design.

Example

```
@media(max-width:768px)
{
.container
{
flex-direction:column;
}
}
```

SET A – Basic Programs

Program A1: HTML Page Structure

Aim :-To create a basic HTML5 webpage containing a heading, paragraph, and image.

Theory :HTML (HyperText Markup Language) is used to create web pages. Every HTML document contains:

- <!DOCTYPE html> → Defines HTML5 document.
- <html> → Root element.
- <head> → Contains page information.
- <body> → Contains visible content.

Step-by-Step Procedure

Step 1: Create HTML File

Create a file named: index.html

Step 2: Write HTML Structure

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Welcome to Web Technology</h1>
  <p>
    This is my first HTML webpage.
  </p>
  <img src="https://via.placeholder.com/300"alt="Sample Image">
</body>
</html>
```

Step 3: Save and Run

1. Save the file.
2. Open in browser.

Output

Welcome to Web Technology
This is my first HTML webpage.
[Image Displayed]

Program A2: Student Registration Form

Aim : To create a student registration form using HTML.

Step-by-Step Procedure

Step 1: Create File : registration.html

Step 2: Write Code

```
<!DOCTYPE html>
<html>
<head>
  <title>Student Registration</title>
</head>
<body>
<h2>Student Registration Form</h2>
```

```
<form>
   Name:<input type="text"><br><br>
   Email:<input type="email"><br><br>
   Password:<input type="password"><br><br>
 Gender:<input type="radio" name="gender"> Male<input type="radio" name="gender"> Female
   <br><br>
   Course:<select>
      <option>BCA</option>
      <option>BSc CS</option>
      <option>BSc IT</option>
   </select>
   <br><br>
   <input type="submit" value="Register">
</form>
</body>
</html>
```

Output

Student Registration Form

Name : _____
Email : _____
Password : _____

Gender : Male Female

Course : [Dropdown]

[Register]

Program A3: HTML Multimedia

Aim: To embed audio, video and YouTube video in webpage.

Step-by-Step Procedure

Step 1: Create File: multimedia.html

Step 2: Write Code

```
<!DOCTYPE html>
<html>
<head>
   <title>Multimedia</title>
</head>
<body>
<h2>Audio Example</h2>
<audio controls> <source src="audio.mp3" type="audio/mp3">
</audio>
<h2>Video Example</h2>
<video width="400" controls><source src="video.mp4" type="video/mp4">
</video>
<h2>YouTube Video</h2>
<iframe width="560"
      height="315"
      src="https://www.youtube.com/embed/tgbNymZ7vqY">
</iframe>
</body> </html>
```

Output

Audio Player

Video Player

Embedded YouTube Video

Program A4: Basic CSS Styling
Aim : To apply CSS properties on webpage elements.
Step-by-Step Procedure
Step 1: Create File : style.html
Step 2: Write Code

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
   background-color: lightblue;
} h1{
   color: darkblue;
   border:2px solid black;
   padding:10px;
   margin:20px;
}p{
   font-size:20px;
} </style>
</head>
<body>
<h1>CSS Example</h1>
<p>Learning CSS Styling.
</p>
</body>
</html>
```

Output
Background Color Applied

Styled Heading
Border Added
Padding Added
Margin Added

SET B – Moderate Programs
Program B1: Semantic Webpage
Aim : To create a webpage using semantic HTML tags.
Code

```
<!DOCTYPE html>
<html>
<head>
<title>Semantic Page</title>
</head>
<body>
<header>
   <h1>College Website</h1>
</header>
<nav>
   <a href="#">Home</a> |
```

```
    <a href="#">About</a>
</nav>
<section>
    <article>
        <h2>News</h2>
        <p>Latest college updates.</p>
    </article>
</section>
<footer>
    Copyright 2026
</footer>
</body>
</html>
```

Output

Header

Navigation Menu

News Section

Footer

Program B2: Responsive Layout using Flexbox

Aim : To create a responsive layout using Flexbox.

Code

```
<!DOCTYPE html>
<html>
<head>
<style>
.container{
    display:flex;
    justify-content:space-around;
    align-items:center;
}
.card{
    border:1px solid black;
    padding:20px;
    width:200px;
}
</style>
</head>
<body>
<nav>
    <h2>Navigation Bar</h2>
</nav>
<div class="container">
<div class="card">
Card 1
</div>
<div class="card">

Card 2
</div>
<div class="card">
Card 3
```

```
</div>
</div>
</body>
</html>
```

Output
Navigation Bar

Card 1 Card 2 Card 3

Program B3: JavaScript Form Validation
Aim : To validate form fields using JavaScript.
Code

```
<!DOCTYPE html>
<html>
<head>
<title>Validation</title>
<script>
function validate(){
var email =document.getElementById("email").value;
var pass =document.getElementById("pass").value;
if(email==""){
   alert("Email Required");
   return false;
}
if(pass.length<6){
   alert("Password must contain 6 characters");
   return false;
}
}
</script>
</head>
<body>
<form onsubmit="return validate()">
Email: <input type="text" id="email">
<br><br>
Password:<input type="password" id="pass">
<br><br>
<input type="submit">
</form>
</body>
</html>
```

Output
Invalid Data → Error Message

Valid Data → Form Submitted

Program B4: DOM Manipulation
Aim: To manipulate webpage content using DOM.
Code

```
<!DOCTYPE html>
<html>
```

```
<head>
<script>
function changeContent(){
document.getElementById("msg").innerHTML= "Text Changed Successfully";
document.body.style.backgroundColor = "yellow";
var p=document.createElement("p");
p.innerHTML="New Paragraph Added";
document.body.appendChild(p);
}
</script>
</head>
<body>
<h2 id="msg">
Original Text
</h2>
<button onclick="changeContent()">
Click Here
</button>
</body>
</html>
```

Output

Original Text

[Click Here]

After Click:

Text Changed Successfully

Background Color Changed

New Paragraph Added

SET C – Advanced Programs

Program C1: Responsive Website using Media Queries

Aim: To create a responsive webpage using media queries.

Code

```
<!DOCTYPE html>
<html>
<head>
<style>
.container{
display:grid;
grid-template-columns:
1fr 1fr 1fr;
gap:10px;
}
.box{
background:skyblue;
padding:20px;
}
@media(max-width:768px){
.container{
```

```
grid-template-columns:
1fr 1fr;
}
}
@media(max-width:480px){
.container{
grid-template-columns:
1fr;
}
}
</style>
</head>
<body>
<div class="container">
<div class="box">Box 1</div>
<div class="box">Box 2</div>
<div class="box">Box 3</div>
</div>
</body>
</html>
```

Output

Desktop → 3 Columns

Tablet → 2 Columns

Mobile → 1 Column

Program C2: Interactive Image Gallery

Aim: To create an image gallery with image preview.

Code

```
<!DOCTYPE html>
<html>
<head>
<script>
function showImage(img){
document.getElementById("main")
.src=img.src;
}
</script>
</head>
<body>
<img id="main"
src="img1.jpg"
width="400">
<br><br>
<img src="img1.jpg"
width="100"
onclick="showImage(this)">
<img src="img2.jpg"
width="100"
onclick="showImage(this)">
<img src="img3.jpg"
width="100"
onclick="showImage(this)">
```

```
</body>
</html>
```

Output

Large Preview Image

Thumbnail Images

Click Thumbnail

Large Image Changes

Program C3: Dynamic To-Do List

Aim: To create a To-Do List using JavaScript.

Code

```
<!DOCTYPE html>
<html>
<body>
<input type="text" id="task">
<button onclick="addTask()">
Add
</button>
<ul id="list"></ul>
<script>
function addTask(){
let task= document.getElementById("task").value;
let li=document.createElement("li");
li.innerHTML= task + " <button onclick='this.parentElement.remove()'>Delete</button>";
document.getElementById("list") .appendChild(li);
}
</script>
</body>
</html>
```

Output

Enter Task

Add Task

Delete Task

Dynamic Task List

Program C4: Student Result Calculator

Aim : To calculate Total, Percentage and Grade.

Code

```
<!DOCTYPE html>
<html>
<body>
<input type="number" id="m1" placeholder="Subject 1"><br><br>
<input type="number" id="m2" placeholder="Subject 2"><br><br>
```

```
<input type="number" id="m3" placeholder="Subject 3"><br><br>
<input type="number" id="m4" placeholder="Subject 4"><br><br>
<input type="number" id="m5" placeholder="Subject 5"><br><br>
<button onclick="calculate()">
Calculate
</button>
<h3 id="result"></h3>
<script>
function calculate(){
let total=Number(m1.value)+ Number(m2.value)+Number(m3.value)+Number(m4.value)+
Number(m5.value);
let percentage= total/5;
let grade;
if(percentage>=75)
grade="A";
else if(percentage>=60)
grade="B";
else if(percentage>=50)
grade="C";
else
grade="Fail";
document.getElementById("result")
.innerHTML=
"Total = "+total+
"<br>Percentage = "+percentage+
"<br>Grade = "+grade;
}
</script>
</body>
</html>
```

Output

Marks Entered

Total = 425

Percentage = 85%

Grade = A

Exercises:

1. Create Resume Webpage.
2. Develop Department Website.
3. Create Event Registration Form.
4. Design E-Commerce Homepage.
5. Create Hotel Website.
6. Design Restaurant Website.
7. Create Online Examination Form.
8. Develop Faculty Profile Page.
9. Design Library Portal.
10. Create Responsive News Portal.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge

2. INTERACTIVE WEB LOGIC (CORE JAVASCRIPT & DOM) (Total Slots = 1)

Assignment No. 2
Title : Interactive Web Logic (Core JavaScript & DOM)
Duration :01 Practical Slot
Course Outcome Mapping
CO5: Perform dynamic DOM manipulation and client-side form validation.
1.Aim : To understand JavaScript fundamentals, DOM manipulation, event handling, and client-side form validation for developing interactive web applications.
2 Learning Objectives
After completing this experiment, students will be able to:

1. Understand ES6 features such as let, const, and var.
2. Write concise functions using Arrow Functions.
3. Use Template Literals and Destructuring Assignments.
4. Work with Spread Operators and JavaScript Modules.
5. Develop logic-based programs such as Prime Number and Fibonacci Series.
6. Implement asynchronous programming using Promises.
7. Use Async/Await for data processing.
8. Perform sorting operations and create visual outputs.
9. Build modern JavaScript applications using ES6+ concepts.
10. Apply advanced JavaScript techniques in real-world web development projects.

3 Introduction to JavaScript
JavaScript is a lightweight, interpreted scripting language used to make web pages interactive and dynamic.
Features of JavaScript

- Lightweight
- Object-Based
- Platform Independent
- Event Driven
- Client-Side Execution
- Supports DOM Manipulation

Applications

- Form Validation
- Dynamic Web Pages
- Interactive Menus
- Animations
- API Integration

4 JavaScript in HTML
Internal JavaScript

```
<script>
document.write("Welcome to JavaScript");
</script>
```

External JavaScript <script src="script.js"></script>
5 Variables in JavaScript
Using var var name = "Rahul";
Using let : let age = 20;
Using const : const PI = 3.14;
6 Data Types

Data Type	Example
Number	100
String	"Hello"
Boolean	true
Array	[10,20,30]

Object	{name:"Rahul"}
Null	null
Undefined	undefined

7 Operators

Arithmetic Operators

+ - * / %

Comparison Operators

== === != > <

Logical Operators

&& || !

8 Decision Making

if Statement

```
if(age>=18)
{
   document.write("Eligible");
}
```

if-else Statement

```
if(marks>=40)
{
   document.write("Pass");
}
else
{
   document.write("Fail");
}
```

9 Loops

for Loop

```
for(let i=1;i<=5;i++)
{
   console.log(i);
}
```

while Loop

```
let i=1;

while(i<=5)
{
   console.log(i);
   i++;
}
```

10 Functions

Function Declaration

```
function add(a,b)
{
   return a+b;
}
```

Function Call

```
let result=add(10,20);
```

11 Events in JavaScript : Events occur when users interact with web pages.

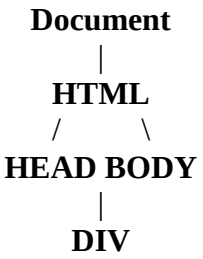
Common Events

Event	Description
onclick	Mouse click
ondblclick	Double click
onmouseover	Mouse over
onmouseout	Mouse out
onkeypress	Key pressed

onsubmit	Form submitted
onchange	Value changed

12 Document Object Model (DOM)

DOM represents an HTML document as a tree structure.



DOM allows JavaScript to:

- Access Elements
- Modify Content
- Change Styles
- Add New Elements
- Remove Elements

13 DOM Methods

getElementById()

document.getElementById("demo");

getElementsByClassName()

document.getElementsByClassName("title");

querySelector()

document.querySelector(".box");

SET A – Basic Programs

Program A1: Demonstrate let, const and var

Aim : To understand the difference between var, let, and const in JavaScript.

Theory

var

- Function scoped
- Can be redeclared
- Can be reassigned

let

- Block scoped
- Cannot be redeclared
- Can be reassigned

const

- Block scoped
- Cannot be redeclared
- Cannot be reassigned

Step-by-Step Procedure

Step 1

Create a file named:

var-let-const.html

Step 2

Write the following code.

```
<!DOCTYPE html>
<html>
<head>
<title>Var Let Const</title>
</head>
<body>
<h2>Open Console to View Output</h2>
```

```
<script>
var a = 10;
var a = 20;
let b = 30;
b = 40;
const c = 50;
console.log("var =", a);
console.log("let =", b);
console.log("const =", c);
</script>
</body>
</html>
```

Step 3

Open browser and press F12.

Output

```
var = 20
let = 40
const = 50
```

Program A2: Arrow Functions

Aim : To demonstrate Arrow Functions in ES6.

Theory

Arrow Functions provide a shorter syntax for writing functions.

Traditional Function:

```
function add(a,b){
 return a+b;
}
```

Arrow Function:

```
const add=(a,b)=>a+b;
```

Step-by-Step Procedure

Step 1

Create file:

arrow-function.html

Step 2

Write code.

```
<!DOCTYPE html>
<html>
<body>
<h2>Arrow Function Example</h2>
<script>
const add = (a,b) => a+b;
let result = add(10,20);
document.write("Addition = " + result);
</script>
</body>
</html>
```

Output

```
Addition = 30
```

Program A3: Template Literals
Aim: To display formatted strings using Template Literals.
Theory
Template literals use backticks ( ).
Syntax:
`Hello ${name}`
Step-by-Step Procedure
Step 1
Create file:
template-literal.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let name = "Rahul";
let course = "BSc Computer Science";
let message =
`Student Name: ${name}
Course: ${course}`;
document.write(message);
</script>
</body>
</html>
```

Output
Student Name: Rahul
Course: BSc Computer Science

Program A4: Swap Numbers using Destructuring
Aim: To swap two numbers using Destructuring Assignment.

Theory
Destructuring allows unpacking values from arrays.
Syntax:
[a,b] = [b,a];

Step-by-Step Procedure
Step 1
Create file:
swap.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let a = 10;
let b = 20;
[a,b] = [b,a];
document.write("A = " + a);
document.write("<br>");
document.write("B = " + b);
```

```
</script>
</body>
</html>
```

Output
A = 20
B = 10

SET B – Moderate Programs

Program B1: Merge Arrays using Spread Operator
Aim : To merge multiple arrays using Spread Operator.

Theory
Spread Operator (...) expands array elements.
Example:
let arr3=[...arr1,...arr2];

Step-by-Step Procedure
Step 1
Create file:
merge-array.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let arr1 = [1,2,3];
let arr2 = [4,5,6];
let arr3 = [...arr1,...arr2];
document.write(arr3);
</script>
</body>
</html>
```

Output
1,2,3,4,5,6

Program B2: Module-Based Calculator
Aim : To perform arithmetic operations using JavaScript modules.

Theory
Modules allow code reusability using:
export
import

Step 1: Create Module File
calculator.js

```
export function add(a,b){
   return a+b;
}

export function sub(a,b){
   return a-b;
```

```
}
export function mul(a,b){
   return a*b;
}
export function div(a,b){
   return a/b;
}
```

Step 2: Create Main File
index.html

```
<!DOCTYPE html>
<html>
<body>
<script type="module">
import {add,sub,mul,div}
from './calculator.js';
document.write("Addition = " + add(10,5));
</script>
</body>
</html>
```

Output
Addition = 15

Program B3: Prime Number Checker
Aim: To check whether a number is prime or not.

Theory
A prime number is divisible only by 1 and itself.
Examples:
2,3,5,7,11...

Step-by-Step Procedure
Step 1
Create file:
prime.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let num = 17;
let prime = true;
for(let i=2;i<num;i++)
{
   if(num%i==0)
   {
      prime = false;
      break;
   }
}

if(prime)
   document.write("Prime Number");
```

```
else
   document.write("Not Prime");
</script>
</body>
</html>
```

Output
Prime Number

Program B4: Fibonacci Series Generator
Aim : To generate Fibonacci Series.

Theory
Fibonacci Sequence:
0 1 1 2 3 5 8 13...
Formula:
Next = Previous + Current

Step-by-Step Procedure
Step 1
Create file:
fibonacci.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let n = 10;
let a = 0;
let b = 1;
document.write(a + " " + b + " ");
for(let i=2;i<n;i++)
{
   let c = a+b;
   document.write(c + " ");
   a = b;
   b = c;
}
</script>
</body>
</html>
```

Output
0 1 1 2 3 5 8 13 21 34

SET C – Advanced Programs

Program C1: Promise-Based Login Simulation
Aim : To simulate login using JavaScript Promises.

Theory
Promises handle asynchronous operations.
States:

- Pending

- Resolved
- Rejected

Step-by-Step Procedure
Step 1
Create file:
login-promise.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let login =
new Promise((resolve,reject)=>{
let success = true;
if(success)
{
   resolve("Login Successful");
}
else
{
   reject("Login Failed");
}

});

login
.then(result =>
document.write(result))
.catch(error =>
document.write(error));
</script>
</body>
</html>
```

Output
Login Successful

Program C2: Async/Await Data Fetch Simulation
Aim: To simulate data fetching using Async/Await.

Theory
Async/Await simplifies asynchronous programming.

Step-by-Step Procedure
Step 1
Create file:
async-await.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
```

```
function fetchData(){
return new Promise(resolve=>{
setTimeout(()=>{
resolve("Student Data Loaded");
},2000);
});
}
async function displayData(){
let data =await fetchData();
document.write(data);
}
displayData();
</script>
</body>
</html>
```

Output

(wait 2 seconds)

Student Data Loaded

Program C3: Sorting Visualizer

Aim: To sort numbers and display before and after sorting.

Theory

JavaScript provides:
sort()
method.

Step-by-Step Procedure

Step 1

Create file:
sorting.html

Step 2

Write code.

```
<!DOCTYPE html>
<html>
<body>
<script>
let numbers =[50,20,90,10,30];
document.write("Before Sorting: "+ numbers);
numbers.sort((a,b)=>a-b);
document.write("<br><br>");
document.write("After Sorting: "+ numbers);

</script>
</body>
</html>
```

Output

Before Sorting:
50,20,90,10,30

After Sorting:
10,20,30,50,90

Program C4: Pattern Generator
Aim: To generate number and star patterns using JavaScript.

Theory
Nested loops are used to generate patterns.

Step-by-Step Procedure
Step 1
Create file:
pattern.html
Step 2
Write code.

```
<!DOCTYPE html>
<html>
<body>
<pre id="output"></pre>
<script>
let pattern = "";
for(let i=1;i<=5;i++)
{
   for(let j=1;j<=i;j++)
   {
      pattern += "* ";
   }

   pattern += "\n";
}
document.getElementById("output")
.innerText = pattern;
</script>
</body>
</html>
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

Exercises:

1. Create Digital Clock.
2. Design Image Gallery.
3. Create Quiz Application.
4. Develop Student Attendance System.
5. Design Online Voting System.
6. Create Age Calculator.
7. Develop BMI Calculator.

8. Create Dynamic Resume Builder.
9. Design Shopping Cart Interface.
10. Develop Online Feedback Form.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge

3. MODERN SCRIPTING WITH ES6+ (Total Slots = 1)

Assignment No. 3
Title : Modern Scripting with ES6+
Duration : 01 Practical Slot
Course Outcome Mapping
CO6: Apply ES6+ features for writing cleaner, modular, and efficient JavaScript programs.

1 Aim

To understand and implement modern JavaScript (ES6+) features such as let, const, template literals, arrow functions, destructuring, spread operators, classes, modules, and other advanced scripting techniques.

2 Learning Objectives

After completing this experiment, students will be able to:

1. Install and configure Node.js environment.
2. Execute JavaScript programs outside the browser.
3. Work with command-line arguments in Node.js.
4. Create and manage HTTP servers.
5. Serve HTML content through Node.js.
6. Use built-in modules such as http and path.
7. Understand Node.js Event Loop and asynchronous callbacks.
8. Develop route-based web applications.
9. Create REST APIs returning JSON data.
10. Build URL-based calculator services.
11. Implement middleware concepts and request logging.
12. Develop basic backend applications using Node.js.

3 Introduction to ES6+

ECMAScript 2015 (ES6) introduced modern features to JavaScript that improved code readability, maintainability, and performance.

Advantages of ES6+

- Cleaner Syntax
- Better Variable Management
- Object-Oriented Support
- Modular Development
- Improved Function Handling
- Enhanced Data Manipulation

4 let and const

let

Used for variables whose values can change.

```
let age = 20;
age = 21;
```

Characteristics

- Block Scoped
- Reassignable
- Not Redeclarable

const

Used for constants.

```
const PI = 3.14159;
```

Characteristics

- Block Scoped
- Cannot be reassigned
- Must be initialized

5 Template Literals : Template literals simplify string creation.

Syntax : `text ${variable}`

Example : let name = "Rahul";
console.log(
`Welcome ${name}`);
Output : Welcome Rahul

6 Arrow Functions

Arrow functions provide shorter syntax.

Traditional Function

```
function add(a,b)
{
return a+b;
}
```

Arrow Function

```
const add =
(a,b)=>a+b;
```

7 Default Parameters

Parameters can have default values.

```
function greet(
name="Student")
{
return
`Hello ${name}`;
}
```

Output

Hello Student

8 Rest Operator

Collects multiple arguments into a single array.

Syntax

```
function demo(...args)
{
}
```

Example

```
function sum(...numbers)
{
return numbers.reduce(
(a,b)=>a+b);
}
```

9 Spread Operator : Expands arrays and objects.

Example

```
let arr1=
[10,20];
let arr2=
[...arr1,30,40];
```

Output :[10,20,30,40]

10 Destructuring

Extract values from arrays or objects.

Array Destructuring

```
const marks=
[80,90,95];
const [a,b,c]=marks;
```

Object Destructuring

```
const student=
{
name:"Rahul",
age:20
};
```

```
const {name,age}
=student;
```

11 Classes in ES6

Syntax

```
class Student
{
constructor(name)
{
this.name=name;
}
display()
{
console.log(
this.name);
}
}
```

12 Modules

Modules allow code organization.

Export Module

```
export function add(a,b)
{
return a+b;
}
```

Import Module

```
import {add}
from './math.js';
```

UNIT 3: Node.js Fundamentals

SET A – Basic Programs

Program A1: Node.js Hello World Program

Aim : To write and execute a simple Hello World program using Node.js.

Theory

Node.js is a JavaScript runtime environment that allows JavaScript code to run outside the browser.

Step-by-Step Procedure

Step 1

Create a file named:
hello.js

Step 2

Write the following code:

```
console.log("Hello World from Node.js");
```

Step 3

Open Command Prompt or Terminal.

Step 4

Run the program:

```
node hello.js
```

Output

Hello World from Node.js

Program A2: Display System Date and Time
Aim: To display current system date and time using Node.js.

Step-by-Step Procedure
Step 1
Create file:
datetime.js
Step 2
Write code:
let currentDate = new Date();
console.log("Current Date and Time:");
console.log(currentDate);
Step 3
Execute:
node datetime.js

Output
Current Date and Time:
Mon Jun 15 2026 10:30:20 GMT+0530

Program A3: Create and Run a Simple Node.js Script Outside Browser
Aim: To execute JavaScript code outside the browser using Node.js.

Theory
Node.js allows JavaScript execution directly from the command line.

Step-by-Step Procedure
Step 1
Create file:
addition.js
Step 2
Write code:
let a = 10;
let b = 20;
let sum = a + b;
console.log("Sum =", sum);
Step 3
Run:
node addition.js

Output
Sum = 30

Program A4: Print Command Line Arguments
Aim: To accept and display command line arguments.

Theory
Node.js provides:
process.argv
for accessing command-line arguments.

Step-by-Step Procedure
Step 1
Create file:

arguments.js

Step 2

Write code:

```
console.log("Arguments:");
for(let i=2;i<process.argv.length;i++)
{
   console.log(process.argv[i]);
}
```

Step 3

Execute:

```
node arguments.js Rahul BCA Pune
```

Output

```
Arguments:
Rahul
BCA
Pune
```

SET B – Moderate Programs

Program B1: Basic HTTP Server

Aim: To create an HTTP server that displays "Welcome to Node.js".

Theory

The http module is used to create web servers.

Step-by-Step Procedure

Step 1

Create file:

server.js

Step 2

Write code:

```
const http = require("http");
const server = http.createServer((req,res)=>{
   res.write("Welcome to Node.js");
   res.end()
});
server.listen(3000);
console.log("Server Running at Port 3000");
```

Step 3

Run:

```
node server.js
```

Step 4

Open browser:

http://localhost:3000

Output

Welcome to Node.js

Program B2: Server Returning HTML Page

Aim: To return an HTML page from Node.js server.

Step-by-Step Procedure
Step 1
Create file:
htmlserver.js
Step 2
Write code:

```
const http = require("http");
const server = http.createServer((req,res)=>{
res.writeHead(200,
{
"Content-Type":"text/html"});
res.write(`
<h1>Welcome</h1>
<p>This page is served using Node.js</p>
`);
res.end();
});
server.listen(3000);
console.log("Server Started");
```

Output
Welcome

This page is served using Node.js

Program B3: Demonstrate Path Module
Aim: To demonstrate usage of Path Module.

Theory
The path module provides utilities for working with file paths.

Step-by-Step Procedure
Step 1
Create file:
pathdemo.js
Step 2
Write code:

```
const path = require("path");
console.log("File Name:",path.basename(__filename));
console.log("Directory:",path.dirname(__filename));
console.log("Extension:",path.extname(__filename));
```

Step 3
Run:
node pathdemo.js

Output
File Name: pathdemo.js

Directory: D:\NodePrograms

Extension: .js

Program B4: Event Loop and Callback Execution Order

Aim: To understand callback execution and Node.js event loop.

Theory
Node.js executes synchronous code first and asynchronous callbacks later.

Step-by-Step Procedure
Step 1
Create file:
eventloop.js
Step 2
Write code:

```
console.log("Start");
setTimeout(()=>{
console.log("Callback Executed");
},0);
console.log("End");
```

Step 3
Run:
node eventloop.js

Output

```
Start
End
Callback Executed
```

SET C – Advanced Programs

Program C1: Mini Web Server
Aim: To create a web server providing different responses for different routes.

Step-by-Step Procedure
Step 1
Create file:
miniwebserver.js
Step 2
Write code:

```
const http = require("http");
const server = http.createServer((req,res)=>{
if(req.url==="/home")
{
   res.end("Welcome to Home Page");
}
else if(req.url==="/about")
{
   res.end("About Us Page");
}
else if(req.url==="/contact")
{
   res.end("Contact Us Page");
}
else
{
   res.end("Page Not Found");
}
```

```
});
server.listen(3000);
console.log("Server Running");
```

URLs to Test
http://localhost:3000/home

http://localhost:3000/about

http://localhost:3000/contact

Output
/home → Welcome to Home Page

/about → About Us Page

/contact → Contact Us Page

Program C2: Simple REST API
Aim: To create a REST API returning student JSON data.

Theory
REST APIs commonly return JSON data.

Step-by-Step Procedure
Step 1
Create file:
studentapi.js
Step 2
Write code:

```
const http = require("http");
const server = http.createServer((req,res)=>{
res.writeHead(200,
{
"Content-Type":"application/json"
});
let student =
{
id:101,name:"Rahul",course:"BCA"};
res.end(JSON.stringify(student));
});
server.listen(3000);
console.log("API Running");
```

Output

```
{
  "id":101,
  "name":"Rahul",
  "course":"BCA"
}
```

Program C3: Node.js Calculator Server
Aim: To perform arithmetic operations using URL parameters.

Step-by-Step Procedure

Step 1
Create file:
calculatorserver.js
Step 2
Write code:

```
const http = require("http");
const url = require("url");
const server = http.createServer((req,res)=>{
let q =url.parse(req.url,true).query;
let a =parseInt(q.a);
let b =parseInt(q.b);
let sum = a+b;
res.end("Result = " + sum);
});
server.listen(3000);
console.log("Calculator Server Running");
```

URL Example
http://localhost:3000/?a=10&b=20

Output
Result = 30

Program C4: Logging Middleware Simulation
Aim: To simulate middleware that logs request time and URL.

Theory
Middleware performs operations before sending a response.

Step-by-Step Procedure
Step 1
Create file:
middleware.js
Step 2
Write code:

```
const http = require("http");
const server = http.createServer((req,res)=>{
console.log("Time:",new Date().toLocaleString());
console.log("URL:",req.url);
res.end("Request Logged");
});
server.listen(3000);
console.log("Server Running");
```

Output
Browser
Request Logged
Console
Time: 15/06/2026, 11:45:30 AM

URL: /home

Exercises:

1. Employee Management System.
2. Product Inventory Application.
3. Student Result Analyzer.
4. Expense Tracker.
5. Online Library System.
6. Banking Record Manager.
7. Shopping Cart Application.
8. Faculty Information System.
9. Attendance Management System.
10. Online Course Portal.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge

4. ASYNCHRONOUS PROGRAMMING IN JAVASCRIPT (Total Slots = 1)

Assignment No. 4
Title: Asynchronous Programming in JavaScript
Duration :01 Practical Slot
Course Outcome Mapping
CO7: Implement asynchronous logic and handle real-world data fetching scenarios.

1 Aim

To understand and implement asynchronous programming concepts using Callbacks, Promises, Async/Await, Fetch API, and JSON for developing modern web applications.

2 Learning Objectives

After completing this experiment, students will be able to:

1. Differentiate between synchronous and asynchronous programming.
2. Implement callback functions.
3. Understand callback hell problems.
4. Create and consume promises.
5. Implement promise chaining.
6. Use async and await keywords.
7. Fetch data from APIs.
8. Handle JSON data.
9. Develop real-world asynchronous applications.

3 Introduction to Asynchronous Programming

JavaScript is a single-threaded language. To perform long-running operations without blocking execution, asynchronous programming is used.

Examples

- Loading web pages
- API requests
- Database operations
- File uploads
- Timers

4 Synchronous Programming

Tasks execute one after another.

Example

```
console.log("Task 1");
console.log("Task 2");
console.log("Task 3");
```

Output

```
Task 1
Task 2
Task 3
```

5 Asynchronous Programming

Tasks can execute independently without blocking other operations.

Example

```
console.log("Start");
setTimeout(()=>
{
console.log("Task");
},2000);
console.log("End");
```

Output

```
Start
End
Task
```

6 Callback Functions

A callback is a function passed as an argument to another function.

Example

```
function greet(name,
callback)
{
console.log(
"Hello "+name);
callback();
}
function goodbye()
{
console.log(
"Goodbye");
}
greet(
"Rahul",
goodbye);
```

Output

```
Hello Rahul
Goodbye
```

7 Callback Hell

Nested callbacks create complex and difficult-to-maintain code.

Example

```
task1(function()
{
task2(function()
{
task3(function()
{
});
});
});
```

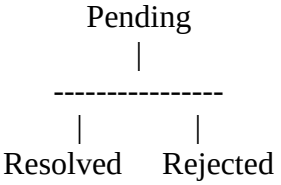
Problems

- Difficult debugging
- Poor readability
- Maintenance issues

8 Promises

A Promise represents the eventual completion or failure of an asynchronous operation.

States of Promise



9 Creating Promise

Syntax

```
let promise=
new Promise(
(resolve,reject)=>
{
});
```

10 Promise Example

```
let promise=
new Promise(
(resolve,reject)=>
{
let success=true;
if(success)
resolve("Success");
else
reject("Failed");
});
promise
.then(result=>
console.log(result))
.catch(error=>
console.log(error));
```

Output

Success

11 Promise Chaining Example

```
Promise.resolve(10)
.then(num=>num*2)
.then(num=>num+5)
.then(result=>
console.log(result));
```

Output

25

12 Async and Await

Introduced in ES8.

Advantages

- Cleaner code
- Better readability
- Easier debugging

Example

```
async function show()
{
return "Hello";
}
show()
.then(result=>
console.log(result));
```

13 Await Example

```
async function demo()
{
let result=
await Promise.resolve(
"Data Loaded");
console.log(result);
}
demo();
```

Output

Data Loaded

14 JSON

JSON (JavaScript Object Notation) is a lightweight format used for data exchange.

Example

```
{
"name":"Rahul",
"age":20
}
```

15 JSON Methods

Convert Object to JSON

JSON.stringify()

Convert JSON to Object

JSON.parse()

16 Fetch API

Used to retrieve resources from servers.

Syntax

```
fetch(url)
.then(response=>response.json())
.then(data=>console.log(data));
```

Experiment 1 :setTimeout Demonstration

Program

```
<script>
setTimeout(
()=>{
document.write("Executed");
},3000);
</script>
```

Output : Message displayed after 3 seconds.

Experiment 2:Callback Function

Program

```
<script>
function process(callback)
{
document.write("Processing");
callback();
}
process(
function()
{
document.write("<br>Done");
});
</script>
```

Output

```
Processing
Done
```

Experiment 3:Promise Example

Program

```
<script>
let promise=
new Promise(
(resolve,reject)=>
{
resolve(
"Promise Resolved");
});
promise.then(
```

```
result=>
document.write(
result));
</script>
```

Output : Promise Resolved

Experiment 4:Promise Rejection

Program

```
<script>
let promise=
new Promise(
(resolve,reject)=>
{
reject("Error");
});
promise.catch(
error=>
document.write(
error));
</script>
```

Output: Error

Experiment 5: Promise Chaining

Program

```
<script>
Promise.resolve(5)
.then(
num=>num*2)
.then(
num=>num+10)
.then(
result=>
document.write(
result));
</script>
```

Output:20

Experiment 6:Async Function

Program

```
<script>
async function test()
{
return "Async Result";
}
test()
.then(
result=>
document.write(
result));
</script>
```

Output: Async Result

Experiment 7:Async Await

Program

```
<script>
async function load()
{
let result=
await Promise.resolve(
```

```
"Data Loaded");
document.write(
result);
}
load();
</script>
```
Output: Data Loaded

Experiment 8: JSON Object Example

Program

```
<script>
let student=
{
name:"Amit",
age:21
};
let json=
JSON.stringify(
student);
document.write(
json);
</script>
```
Output:{"name":"Amit","age":21}

Experiment 9: JSON Parsing

Program

```
<script>
let data=
'{"name":"Rahul"}';
let obj=
JSON.parse(data);
document.write(
obj.name);
</script>
```
Output : Rahul

Experiment 10 :Fetch API Example

Program

```
<script>
fetch(
'https://jsonplaceholder.typicode.com/users')
.then(
response=>
response.json())
.then(
data=>
console.log(data));
</script>
```
Output

User data displayed in browser console.

Experiment 11

Weather Information Application

Features

- Fetch weather data
- Display temperature
- Display city information

Technologies

- Fetch API
- JSON

Experiment 12

Student Data Dashboard

Features

- Fetch student records
- Display dynamically
- Update webpage without reload

Mini Project

Real-Time Data Dashboard

Functionalities

1. Fetch Data
2. Display Records
3. Auto Refresh
4. Search Functionality
5. Dynamic Updates

Technologies

- HTML5
- CSS3
- JavaScript
- Fetch API
- JSON

Exercises

1. Movie Information App.
2. Currency Converter.
3. News Fetching Portal.
4. Student Record Dashboard.
5. Employee Management Portal.
6. COVID Statistics Viewer.
7. Stock Market Tracker.
8. Weather Forecast System.
9. Sports Score Dashboard.
10. Online Book Information Portal.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge

5. SERVER-SIDE BASICS WITH NODE.JS (Total Slots = 1)

Assignment No. 5

Title :Server-Side Basics with Node.js

Duration : 01 Practical Slot

Course Outcome Mapping

CO7: Implement asynchronous logic and build basic server-side applications and REST APIs.

1 **Aim :** To understand Node.js fundamentals and develop server-side applications using built-in modules and the Express framework.

2 Learning Objectives

After completing this experiment, students will be able to:

1. Understand Node.js architecture.
2. Install and use Node.js and npm.
3. Work with Node.js modules.
4. Create web servers using HTTP module.
5. Implement routing.
6. Develop applications using Express.js.
7. Use middleware functions.
8. Create RESTful web services.
9. Build simple server-side applications.

3 Introduction to Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside the browser.

Developed By

Ryan Dahl

Features

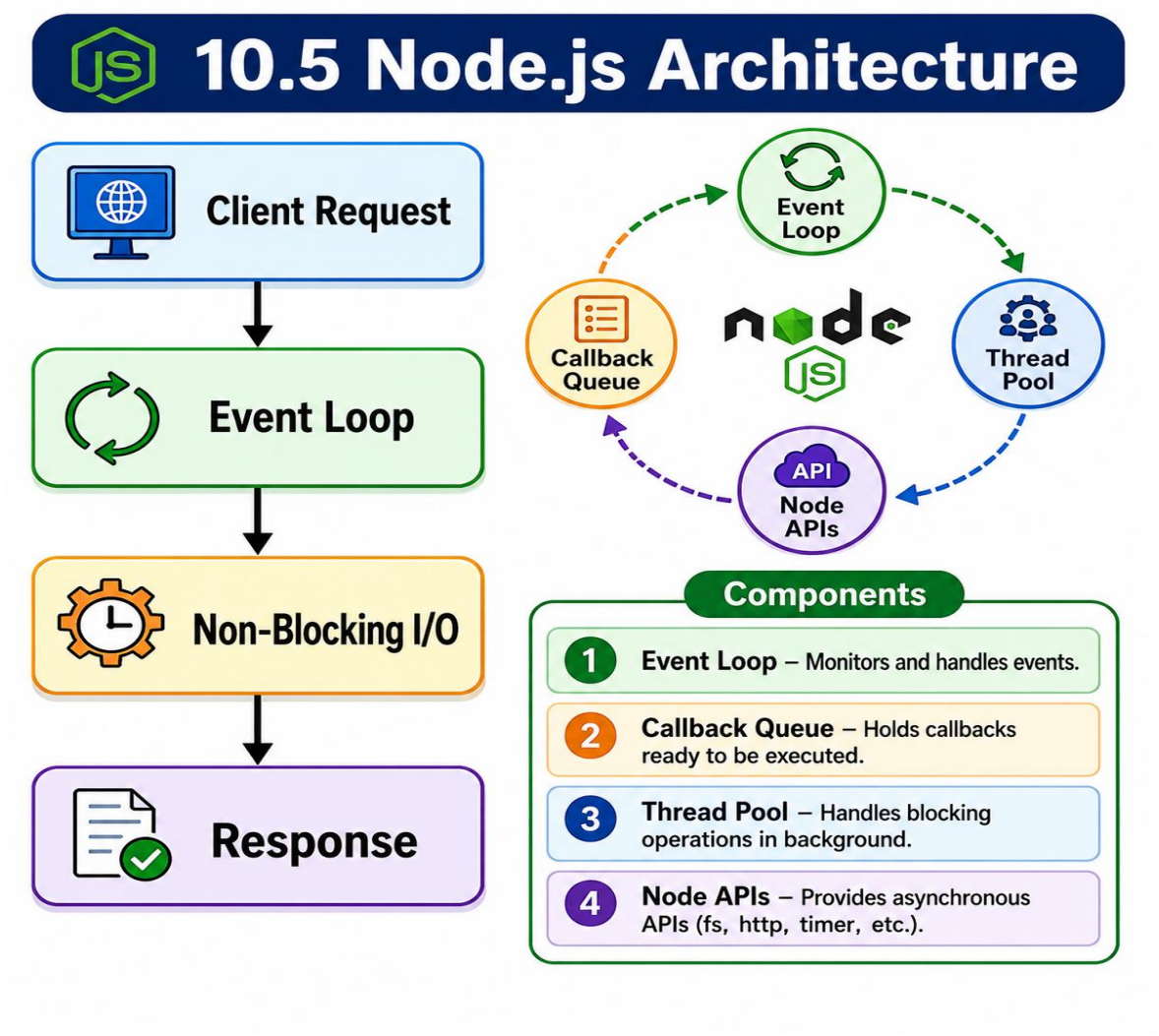
- Event Driven
- Non-Blocking I/O
- Fast Execution
- Scalable Applications
- Cross Platform
- Open Source

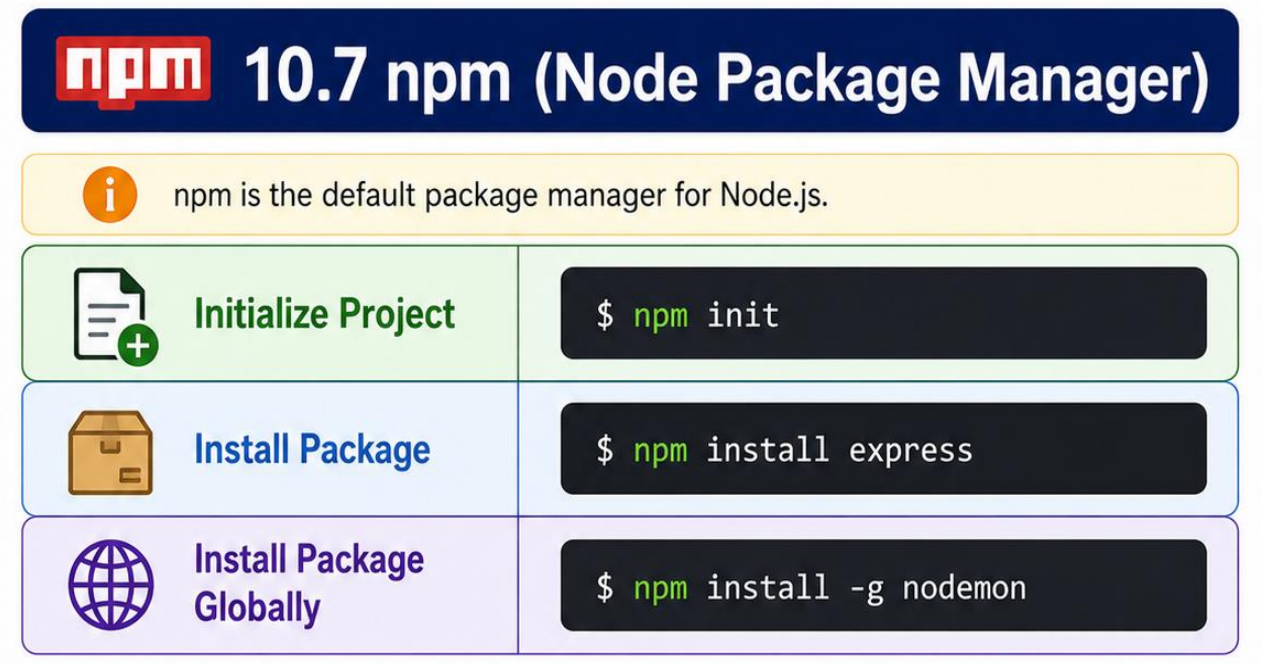
4 Why Node.js?

Advantages

- Uses JavaScript on both client and server side
- High performance
- Efficient handling of concurrent requests

- Large package ecosystem
- Easy API development





8 Node.js Modules

Modules help organize code into reusable components.

Types

Built-in Modules

- http
- fs
- path
- os

User Defined Modules

Custom modules created by developers.

Third-Party Modules

Installed through npm.

9 Creating Modules

math.js

```
function add(a,b)
{
    return a+b;
}
module.exports=
{
    add
};
```

app.js

```
const math = require('./math');
console.log(math.add(10,20));
```

10 HTTP Module

Used to create web servers.

Syntax

```
const http = require('http');
```

Set A – Basic Programs

Experiment A1 :Node.js Installation Verification

Program

```
console.log("Node.js Installed Successfully");
```

Execution

```
node app.js
```

Output

```
Node.js Installed Successfully
```

Experiment A2: Simple HTTP Server

Program

```
const http =require('http');
http.createServer((req,res)=>
{
res.write("Welcome to Node.js");
res.end();
}).listen(3000);
console.log("Server Running");
```

Output

```
Server Running
```

Browser Output:

```
Welcome to Node.js
```

Experiment A3 : Display Current Date and Time

Program

```
const http =require('http');
http.createServer((req,res)=>
{
res.write(new Date().toString());
res.end();
}).listen(3000);
```

Experiment A4 : User Defined Module

math.js

```
exports.add=(a,b)=>a+b;
```

app.js

```
const math=require('./math');
console.log(math.add(10,20));
```

Output

```
30
```

Set B – Moderate Programs

Experiment B5 File Server

Program

```
const http= require('http');
const fs= require('fs');
http.createServer( (req,res)=>
{
fs.readFile('index.html',(err,data)=>
{
res.write(data);
res.end();
});
}).listen(3000);
```

Experiment B6 :Routing Example

Program

```
const http=require('http');
http.createServer((req,res)=>
{
if(req.url==="/")
{
res.end("Home Page");
}
else if(req.url==="/about")
{
res.end("About Page");
}
else
{
res.end("404 Error");
}
}).listen(3000);
```

11 Express Framework

Express.js is a lightweight framework for building web applications and APIs.

Installation

```
npm install express
```

Import Express

```
const express= require('express');
```

Experiment B7 : First Express Application

Program

```
const express=require('express');
const app=express();
app.get('/',(req,res)=>
{
res.send(
"Welcome Express");
});
app.listen(3000);
```

Output :Welcome Express

Experiment B8 :Multiple Routes

Program

```
const express=require('express');
const app=express();
```

```
app.get('/',(req,res)=>
{
res.send("Home");
});
app.get('/about',(req,res)=>
{
res.send("About");
});
app.listen(3000);
```

12 Middleware

Middleware functions execute between request and response.

Uses

- Authentication
- Logging
- Validation
- Error Handling

Set C – Advanced Programs

Experiment C9 :Middleware Example

Program

```
const express= require('express');
const app=express();
app.use((req,res,next)=>
{
console.log("Request Received");
next();
});
app.get('/',(req,res)=>
{
res.send("Home Page");
});
app.listen(3000);
```

Experiment C10: Student Information Server

Features

- Display Student Details
- Send JSON Response

Program

```
const express=require('express');
const app=express();
app.get('/student',(req,res)=>
{
res.json(
{
name:"Rahul",course:"BSc CS"});
});
app.listen(3000);
```

Output

```
{
"name":"Rahul",
"course":"BSc CS"
}
```

Experiment C11 :REST API Example

Program

```
const express=require('express');
const app=express();
app.get('/api/students',(req,res)=>
```

```
{
res.json(
[
{
id:1,
name:"Rahul"
},
{
id:2,
name:"Neha"
}
]);
});
app.listen(3000);
```

Experiment C12 : Student Management Server

Features

1. Add Student
2. View Student
3. Update Student
4. Delete Student

Technologies

- Node.js
- Express.js

Exercises:

1. Employee Information API.
2. Library Management Server.
3. Hospital Information Server.
4. Banking API.
5. Inventory Management API.
6. Attendance Management System.
7. Student Result API.
8. Online Quiz Server.
9. Product Catalog API.
10. Event Management System.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge

6. FILE SYSTEM & API FUNDAMENTALS (Total Slots = 1)

Assignment No.6
Title : File System & API Fundamentals
Duration : 01 Practical Slot
Course Outcome Mapping
CO7: Implement asynchronous logic and build basic server-side applications and REST APIs.
1 Aim : To understand file system operations and develop RESTful APIs using Node.js and Express for real-world application development.
2 Learning Objectives
After completing this experiment, students will be able to:

1. Perform file operations using Node.js File System Module.
2. Read, write, append, and delete files.
3. Manage directories.
4. Work with JSON data.
5. Understand REST API concepts.
6. Implement CRUD operations.
7. Develop RESTful APIs using Express.
8. Test APIs using browser and API testing tools.

3 Introduction to File System Module
Node.js provides the **File System (fs)** module to perform operations on files and directories.
Importing fs Module

```
const fs = require('fs');
```

Features

- Create Files
- Read Files
- Write Files
- Append Data
- Rename Files
- Delete Files
- Create Directories
- Remove Directories

4 Synchronous and Asynchronous File Operations
Synchronous : Execution waits until operation completes.

```
fs.writeFileSync("data.txt","Hello");
```

Asynchronous : Execution continues without waiting.

```
fs.writeFile(
"data.txt",
"Hello",
(err)=>
{
if(err)
throw err;
});
```

5 Reading Files
Syntax

```
fs.readFile(
filename,
encoding,
callback);
```

Example

```
fs.readFile(
'student.txt',
'utf8',
(err,data)=>
```

```
{
console.log(data);
});
```

6 Writing Files

Example

```
fs.writeFile(
'student.txt',
'TYBSc Computer Science',
(err)=>
{
if(err)
throw err;
console.log(
"File Saved");
});
```

7 Appending Data

Example

```
fs.appendFile(
'student.txt',
'\nRoll No: 101',
(err)=>
{
if(err)
throw err;
});
```

8 Deleting Files

Example

```
fs.unlink(
'student.txt',
(err)=>
{
if(err)
throw err;
console.log(
"File Deleted"); });
```

9 Directory Operations

Create Directory

```
fs.mkdir('Students',
(err)=>
{
if(err)
throw err;
});
```

Remove Directory

```
fs.rmdir(
'Students',
(err)=>
{
if(err)
throw err;
});
```

10 JSON Data Handling

JSON is widely used for data exchange.

Example JSON

```
{
"name":"Rahul",
"course":"BSc CS"
}
```

Convert Object to JSON

JSON.stringify();

Convert JSON to Object

JSON.parse();

11 REST API Fundamentals

REST stands for:

Representational State Transfer

REST APIs allow communication between applications over HTTP.

12 HTTP Methods

Method	Operation
GET	Retrieve Data
POST	Insert Data
PUT	Update Data
DELETE	Remove Data

13 CRUD Operations

CRUD represents:

Operation	Description
Create	Add Data
Read	View Data
Update	Modify Data
Delete	Remove Data

Set A – Basic Programs

Experiment 1: Create a File

Program

```
const fs =
require('fs');
fs.writeFile(
'student.txt',
'Student Record',
(err)=>
{
if(err)
throw err;
console.log(
"File Created");
});
```

Output

File Created

Experiment 2 Read a File

Program

```
const fs =require('fs');
fs.readFile('student.txt','utf8',(err,data)=>
{
console.log(data);
});
```

Output

Student Record

Experiment 3 Append Data
Program

```
const fs =
require('fs');
fs.appendFile(
'student.txt',
'\nRoll No:101',
(err)=>
{
console.log(
"Data Added");
});
```

Output

Data Added

Experiment 4 Delete a File
Program

```
const fs =
require('fs');
fs.unlink(
'student.txt',
(err)=>
{
console.log(
"File Deleted");
});
```

Output

File Deleted

Set B – Moderate Programs

Experiment 5 :Create Directory
Program

```
const fs =
require('fs');
fs.mkdir(
'StudentData',
(err)=>
{
console.log(
"Folder Created");
});
```

Output

Folder Created

Experiment 6 Rename File
Program

```
const fs =require('fs');
fs.rename('old.txt','new.txt',(err)=>
{
console.log("File Renamed");
});
```

Output : File Renamed

Experiment 7:JSON Data Example

Program

```
const student={
name:"Rahul",
course:"BSc CS"
};
console.log(JSON.stringify(student));
```

Output

```
{"name":"Rahul","course":"BSc CS"}
```

Experiment 8 :Express REST API Setup

Program

```
const express=require('express');
const app=express();
app.listen(3000);
```

Output : Server Started

Set C – Advanced Programs

Experiment 9 :GET API

Program

```
const express=require('express');
const app=express();
app.get('/students',(req,res)=>
{
res.json(
[
{
id:1,
name:"Rahul"
}
]);
});
app.listen(3000);
```

Output

```
[
{
"id":1,"name":"Rahul"
}
]
```

Experiment 10 :POST API

Program

```
app.post(
'/students',
(req,res)=>
{
res.send(
"Student Added");
});
```

Output: Student Added

Experiment 11:PUT API

Program

```
app.put(
'/students/:id',
(req,res)=>
{
res.send(
"Student Updated");
});
```

Output: Student Updated

Experiment 12:DELETE API

Program

```
app.delete(
'/students/:id',
(req,res)=>
{
res.send(
"Student Deleted");
});
```

Output: Student Deleted

Exercises:-

1. Employee Management REST API.
2. Library Management API.
3. Hospital Information API.
4. Inventory Management System.
5. Banking Transaction API.
6. Product Catalog API.
7. Event Registration API.
8. Student Attendance API.
9. Faculty Information API.
10. Online Examination API.

Assignment Evaluation:

0: Not Done		**1:Incomplete**		**2:Late Completion**	
3:Needs Improvement		**4:Complete**		**5:Well Done**	

Practical In-Charge