**PRACTICAL-1**

**AIM:** Install JDK and writer a java program to print your name.

**Step 1: Install JDK:**

**For Windows:**

1. **Go to** [**Oracle JDK Download**](https://www.oracle.com/java/technologies/javase-downloads.html)**.**
2. **Download the latest JDK version (e.g., Java 21 or 17 LTS).**
3. **Run the installer and follow the instructions.**
4. **After installation, set the environment variable:**
   * **Search for "Environment Variables" → Edit system variables**
   * **Under System variables, add:**
     + **JAVA\_HOME → path to your JDK installation (e.g., C:\Program Files\Java\jdk-21)**
     + **Add ;%JAVA\_HOME%\bin to your Path variable.**

**Verify Installation:**

**Open Command Prompt and run:**

**java -version**

**javac -version**

**Step 2: Write a Java Program:**

1. **Open Notepad or any text editor.**
2. **Type the following code:**

**public class MyName {**

**public static void main(String[ ] args) {**

**System.out.println("M.J.J.B.");**

**}**

**}**

1. **Save the file as MyName.java.:**

**Step 3: Compile and Run the Program:**

**Open Command Prompt, navigate to the folder where your file is saved, and run:**

**javac MyName.java**

**java MyName**

**✅ Output:**

**M.J.J.B.**

**PRACTICAL-2**

**AIM:** Create three variables to store marks of three subjects and generate mark sheet.(use if condition or switch case)

**INPUT:**

public class MarkSheet {

public static void main(String[] args) {

int math = 85;

int science = 78;

int english = 92;

int total = math + science + english;

double percentage = total / 3.0;

System.out.println("----------- MARK SHEET -----------");

System.out.println("Math : " + math);

System.out.println("Science : " + science);

System.out.println("English : " + english);

System.out.println("----------------------------------");

System.out.println("Total : " + total);

System.out.printf("Percentage: %.2f%%\n", percentage);

if (math < 35 || science < 35 || english < 35) {

System.out.println("Result : FAIL");

} else {

System.out.println("Result : PASS");

if (percentage >= 85) {

System.out.println("Grade : A");

} else if (percentage >= 70) {

System.out.println("Grade : B");

} else if (percentage >= 50) {

System.out.println("Grade : C");

} else {

System.out.println("Grade : D");

}

}

}

}

**OUTPUT:**

E:\>java MarkSheet

----------- MARK SHEET -----------

Math : 85

Science : 78

English : 92

----------------------------------

Total : 255

Percentage: 85.00%

Result : PASS

Grade : A

**PRACTICAL-3**

**AIM:** Write a program in Java to reverse the digits of a number using while loop.

**INPUT:**

public class ReverseNumber {

public static void main(String[] args) {

int number = 12345;

int reverse = 0;

System.out.println("Original Number: " + number);

while (number != 0) {

int digit = number % 10;

reverse = reverse \* 10 + digit;

number = number / 10;

}

System.out.println("Reversed Number: " + reverse);

}

}

**OUTPUT:**

E:\>java ReverseNumber

Original Number: 12345

Reversed Number: 54321

**PRACTICAL-4**

**AIM:** Writer a program to demonstrate use of wrapper class.

**INPUT:**

public class WrapperDemo {

public static void main(String[] args) {

System.out.println("Demonstrating Wrapper Class Usage");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// Autoboxing (primitive to object)

int a = 10;

Integer aObj = a;

double d = 15.75;

Double dObj = d;

char c = 'X';

Character cObj = c;

System.out.println("Autoboxing (primitive to object):");

System.out.println("int a = " + a + " → Integer aObj = " + aObj);

System.out.println("double d = " + d + " → Double dObj = " + dObj);

System.out.println("char c = " + c + " → Character cObj = " + cObj);

System.out.println();

// Unboxing (object to primitive)

Integer x = new Integer(25);

int xVal = x;

Double y = new Double(99.99);

double yVal = y;

Character z = new Character('Z');

char zVal = z;

System.out.println("Unboxing (object to primitive):");

System.out.println("Integer x = " + x + " → int xVal = " + xVal);

System.out.println("Double y = " + y + " → double yVal = " + yVal);

System.out.println("Character z = " + z + " → char zVal = " + zVal);

}

}

**OUTPUT:**

E:\>java WrapperDemo

Demonstrating Wrapper Class Usage

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Autoboxing (primitive to object):

int a = 10 → Integer aObj = 10

double d = 15.75 → Double dObj = 15.75

char c = X → Character cObj = X

Unboxing (object to primitive):

Integer x = 25 → int xVal = 25

Double y = 99.99 → double yVal = 99.99

Character z = Z → char zVal = Z

**PRACTICAL-5**

**AIM:** Write a program in Java to perform addition of two matrix.

**INPUT:**

public class MatrixAddition {

public static void main(String[ ] args) {

// Define two 2x2 matrices

int[ ][ ] A = {

{1, 2},

{3, 4}

};

int[ ][ ] B = {

{5, 6},

{7, 8}

};

// Create a result matrix

int[ ][ ] sum = new int[2][2];

// Perform addition

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

sum[i][j] = A[i][j] + B[i][j];

}

}

System.out.println("Matrix A + Matrix B =");

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

System.out.print(sum[i][j] + "\t");

}

System.out.println( );

}

}

}

**OUTPUT:**

E:\>java ReverseNumber

Original Number: 12345

Reversed Number: 54321

**PRACTICAL-6**

**AIM:** Using command line argument perform addition of two values.

**INPUT:**

public class AddArgs {

public static void main(String[ ] args) {

if (args.length < 2) {

System.out.println("Please provide two numbers as command line arguments.");

return;

}

int num1 = Integer.parseInt(args[0]);

int num2 = Integer.parseInt(args[1]);

int sum = num1 + num2;

System.out.println("First Number : " + num1);

System.out.println("Second Number : " + num2);

System.out.println("Sum : " + sum);

}

}

**OUTPUT:**

E:\>java AddArgs 15 25

First Number : 15

Second Number : 25

Sum : 40

**PRACTICAL-7**

**AIM:** Write a program in Java to demonstrate use of this keyword.

**INPUT:**

public class ThisDemo {

int id;

String name;

// Constructor using 'this' keyword

ThisDemo(int id, String name) {

this.id = id; // refers to instance variable

this.name = name;

}

void display( ) {

System.out.println("ID : " + this.id);

System.out.println("Name : " + this.name);

}

public static void main(String[ ] args) {

ThisDemo student = new ThisDemo(101, " M.J.J.B.");

student.display( );

}

}

**OUTPUT:**

E:\>java ThisDemo

ID : 101

Name : M.J.J.B.

**PRACTICAL-8**

**AIM:** Write a program in java to demonstrate use of default constructor, copy constructor and parameterized constructor.

**INPUT:**

public class ConstructorDemo {

int id;

String name;

// Default constructor

ConstructorDemo ( ) {

this.id = 0;

this.name = "Default Name";

System.out.println("Default Constructor called");

}

// Parameterized constructor

ConstructorDemo(int id, String name) {

this.id = id;

this.name = name;

System.out.println("Parameterized Constructor called");

}

// Copy constructor

ConstructorDemo(ConstructorDemo other) {

this.id = other.id;

this.name = other.name;

System.out.println("Copy Constructor called");

}

// Method to display data

void display( ) {

System.out.println("ID : " + this.id);

System.out.println("Name : " + this.name);

}

public static void main(String[ ] args) {

// Using default constructor

ConstructorDemo student1 = new ConstructorDemo();

student1.display( );

// Using parameterized constructor

ConstructorDemo student2 = new ConstructorDemo(101, "Alice");

student2.display( );

// Using copy constructor

ConstructorDemo student3 = new ConstructorDemo(student2);

student3.display( );

}

}

**OUTPUT:**

E:\>java ConstructorDemo

Default Constructor called

ID : 0

Name : Default Name

Parameterized Constructor called

ID : 101

Name : Alice

Copy Constructor called

ID : 101

Name : Alice

**PRACTICAL-9**

**AIM:** Write a program in Java to demonstrate use of final keyword at variable level and class level.

**INPUT:**

// Final class

final class FinalClass {

// Final variable

final int id = 100;

// Constructor

FinalClass( ) {

System.out.println("FinalClass Constructor called");

}

// Method to display ID

void display( ) {

System.out.println("ID: " + id);

}

}

public class FinalKeywordDemo {

public static void main(String[ ] args) {

// Trying to modify final variable (This will give a compile-time error)

// id = 200; // Uncommenting this line will cause an error

// Creating an object of FinalClass

FinalClass obj = new FinalClass( );

obj.display( );

}

}

**OUTPUT:**

E:\>java FinalKeywordDemo

FinalClass Constructor called

ID: 100

**PRACTICAL-10**

**AIM:** Write a program in Java to demonstrate use of static keyword.

**INPUT:**

public class StaticKeywordDemo {

// Static variable

static int count = 0;

// Constructor

StaticKeywordDemo( ) {

count++; // Increment the static variable each time an object is created

}

// Static method

static void displayCount( ) {

System.out.println("Count: " + count);

}

public static void main(String[ ] args) {

// Display count before creating objects

StaticKeywordDemo.displayCount( );

// Create objects

StaticKeywordDemo obj1 = new StaticKeywordDemo( );

StaticKeywordDemo obj2 = new StaticKeywordDemo( );

StaticKeywordDemo obj3 = new StaticKeywordDemo( );

// Display count after creating objects

StaticKeywordDemo.displayCount( );

}

}

**OUTPUT:**

E:\>java StaticKeywordDemo

Count: 0

Count: 3

**PRACTICAL-11**

**AIM:** Develop minimum 4 program based on variation in methods i.e. passing by value, passing by reference, returning values and returning objects from methods.

**INPUT:**

✅ 1. **Passing by Value**:

public class PassByValueDemo {

// Method to demonstrate passing by value

static void modifyValue(int num) {

num = num + 10; // Modify the value

System.out.println("Inside modifyValue, num = " + num);

}

public static void main(String[] args) {

int number = 5;

System.out.println("Before modifyValue, number = " + number);

modifyValue(number); // Pass by value

System.out.println("After modifyValue, number = " + number); // Original value remains unchanged

}

}

**OUTPUT:**

E:\>java PassByValueDemo

Before modifyValue, number = 5

Inside modifyValue, num = 15

After modifyValue, number = 5

**INPUT:**

✅ 2. **Passing by Reference**:

public class PassByReferenceDemo {

// Method to demonstrate passing by reference

static void modifyArray(int[ ] arr) {

arr[0] = 99; // Modify the first element of the array

System.out.println("Inside modifyArray, arr[0] = " + arr[0]);

}

public static void main(String[] args) {

int[ ] numbers = {1, 2, 3};

System.out.println("Before modifyArray, arr[0] = " + numbers[0]);

modifyArray(numbers); // Pass by reference

System.out.println("After modifyArray, arr[0] = " + numbers[0]); // Original array is modified

}

}

**OUTPUT:**

E:\>java PassByReferenceDemo

Before modifyArray, arr[0] = 1

Inside modifyArray, arr[0] = 99

After modifyArray, arr[0] = 99

**INPUT:**

✅ 3. **Returning Values**:

public class ReturnValueDemo {

// Method to return a value

static int addNumbers(int a, int b) {

return a + b; // Return the sum

}

public static void main(String[ ] args) {

int sum = addNumbers(10, 20); // Get the returned value

System.out.println("The sum of 10 and 20 is: " + sum);

}

}

**OUTPUT:**

E:\>java ReturnValueDemo

The sum of 10 and 20 is: 30

**INPUT:**

✅ 4. **Returning Objects**:

class Person {

String name;

Person(String name) {

this.name = name;

}

void display( ) {

System.out.println("Name: " + name);

}

}

public class ReturnObjectDemo {

// Method to return an object

static Person createPerson(String name) {

return new Person(name); // Return a new Person object

}

public static void main(String[ ] args) {

Person p = createPerson("John Doe"); // Get the returned object

p.display( ); // Call method on the returned object

}

}

**OUTPUT:**

E:\>java ReturnObjectDemo

Name: John Doe

**PRACTICAL-12**

**AIM:** Write a program in Java to demonstrate single inheritance, multilevel inheritance and hierarchical inheritance.

**INPUT:**

**✅ 1. Single Inheritance:**

class Animal {

void sound( ) {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark( ) {

System.out.println("Dog barks");

}

}

public class SingleInheritance {

public static void main(String[ ] args) {

Dog d = new Dog( );

d.sound( ); // Inherited method

d.bark ( ); // Own method

}

}

**OUTPUT:**

E:\>java SingleInheritance

Animal makes sound

Dog barks

**INPUT:**

**✅ 2. Multilevel Inheritance**

class Animal {

void sound( ) {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark( ) {

System.out.println("Dog barks");

}

}

class Puppy extends Dog {

void weep( ) {

System.out.println("Puppy weeps");

}

}

public class MultilevelInheritance {

public static void main(String[ ] args) {

Puppy p = new Puppy( );

p.sound( ); // From Animal

p.bark( ); // From Dog

p.weep( ); // Own method

}

}

**OUTPUT:**

E:\>java MultilevelInheritance

Animal makes sound

Dog barks

Puppy weeps

**INPUT:**

**✅ 3. Hierarchical Inheritance**

class Animal {

void sound( ) {

System.out.println("Animal makes sound");

}

}

class Dog extends Animal {

void bark( ) {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

void meow( ) {

System.out.println("Cat meows");

}

}

public class HierarchicalInheritance {

public static void main(String[ ] args) {

Dog d = new Dog( );

Cat c = new Cat( );

d.sound( ); // Inherited

d.bark( ); // Dog's own

c.sound( ); // Inherited

c.meow( ); // Cat's own

}

}

**OUTPUT:**

E:\>java HierarchicalInheritance

Animal makes sound

Dog barks

Animal makes sound

Cat meows

**PRACTICAL-13**

**AIM:** Write a program in Java in which a subclass constructor invokes the constructor of the super class and instantiate the values.

**INPUT:**

class Person {

String name;

int age;

// Superclass constructor

Person(String name, int age) {

this.name = name;

this.age = age;

System.out.println("Person Constructor Called");

}

void displayPersonInfo( ) {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

}

class Student extends Person {

String course;

// Subclass constructor invoking superclass constructor

Student(String name, int age, String course) {

super(name, age); // Call to superclass constructor

this.course = course;

System.out.println("Student Constructor Called");

}

void displayStudentInfo( ) {

displayPersonInfo( ); // Call method from superclass

System.out.println("Course: " + course);

}

}

public class SuperConstructorDemo {

public static void main(String[ ] args) {

Student s = new Student("M.J.J.B.", 20, "Computer Science");

s.displayStudentInfo( );

}

}

**OUTPUT:**

E:\>java SuperConstructorDemo

Person Constructor Called

Student Constructor Called

Name: M.J.J.B.

Age: 20

Course: Computer Science

**PRACTICAL-14**

**AIM:** Write a program that illustrates interface inheritance. Interface P12 inherits from both P1 and P2. Each interface declares one constant and one method. The class Q implements P12. Instantiate Q and invoke each of its methods. Each method displays one of the constants.

**INPUT:**

interface P1 {

int A = 10; // Constant

void displayA( ); // Method

}

interface P2 {

int B = 20; // Constant

void displayB( ); // Method

}

interface P12 extends P1, P2 {

int C = 30; // Constant in extended interface

void displayC( ); // Method in extended interface

}

class Q implements P12 {

public void displayA( ) {

System.out.println("P1 Constant A = " + A);

}

public void displayB( ) {

System.out.println("P2 Constant B = " + B);

}

public void displayC ( ) {

System.out.println("P12 Constant C = " + C);

}

}

public class InterfaceInheritanceDemo {

public static void main(String[ ] args) {

Q obj = new Q( );

obj.displayA( );

obj.displayB( );

obj.displayC( );

}

}

**OUTPUT:**

E:\>java InterfaceInheritanceDemo

P1 Constant A = 10

P2 Constant B = 20

P12 Constant C = 30

**PRACTICAL-15**

**AIM:** Write a program in Java to demonstrate implementation of multiple inheritance using interfaces.

**INPUT:**

// First interface

interface Writer {

void write( );

}

// Second interface

interface Reader {

void read( );

}

// Class implementing both interfaces (Multiple Inheritance via Interfaces)

class Person implements Writer, Reader {

public void write( ) {

System.out.println("M.J.J.B. is Writing...");

}

public void read( ) {

System.out.println("M.J.J.B. is Reading...");

}

}

public class MultipleInheritanceDemo {

public static void main(String[ ] args) {

Person obj = new Person();

obj.read( ); // Call from Reader interface

obj.write( ); // Call from Writer interface

}

}

**OUTPUT:**

E:\>java MultipleInheritanceDemo

M.J.J.B. is Reading...

M.J.J.B. is Writing...

**PRACTICAL-16**

**AIM:** Describe abstract class called Shape which has three subclasses say Triangle, Rectangle, and Circle. Define one method area() in the abstract class and override this area() in these three subclasses to calculate for specific object i.e. area() of Triangle subclass should calculate area of triangle etc. Same for Rectangle and Circle.

**INPUT:**

abstract class Shape {

abstract void area( ); // Abstract method

}

// Subclass for Triangle

class Triangle extends Shape {

double base = 10.0;

double height = 5.0;

void area( ) {

double result = 0.5 \* base \* height;

System.out.println("Area of Triangle = " + result);

}

}

// Subclass for Rectangle

class Rectangle extends Shape {

double length = 8.0;

double breadth = 4.0;

void area( ) {

double result = length \* breadth;

System.out.println("Area of Rectangle = " + result);

}

}

// Subclass for Circle

class Circle extends Shape {

double radius = 7.0;

void area( ) {

double result = Math.PI \* radius \* radius;

System.out.println("Area of Circle = " + result);

}

}

public class ShapeDemo {

public static void main(String[ ] args) {

Shape s;

s = new Triangle( );

s.area( );

s = new Rectangle( );

s.area( );

s = new Circle( );

s.area( );

}

}

**OUTPUT:**

E:\>java ShapeDemo

Area of Triangle = 25.0

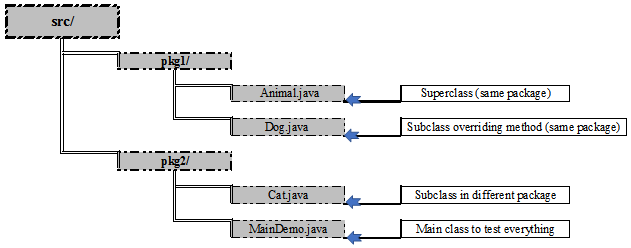
Area of Rectangle = 32.0

Area of Circle = 153.93804002589985

**PRACTICAL-17**

**AIM:** Write an application that illustrates method overriding in the same package and different packages. Also demonstrate accessibility rules in inside and outside packages

**📂 Project Structure (Simulating Packages):**

****

📄 pkg1/Animal.java

**INPUT:**

package pkg1;

public class Animal {

public void speak( ) {

System.out.println("Animal speaks (public)");

}

protected void eat( ) {

System.out.println("Animal eats (protected)");

}

void sleep( ) {

System.out.println("Animal sleeps (default)");

}

private void run( ) {

System.out.println("Animal runs (private)");

}

public void callPrivate( ) {

run( ); // private method is accessible within class

}

}

📄 pkg1/Dog.java (Same package as Animal)

**INPUT:**

package pkg1;

public class Dog extends Animal {

@Override

public void speak( ) {

System.out.println("Dog barks (Overridden in same package)");

}

@Override

protected void eat( ) {

System.out.println("Dog eats bones (Overridden)");

}

@Override

void sleep( ) {

System.out.println("Dog sleeps a lot (Overridden)");

}

// Cannot override private method

}

📄 pkg2/Cat.java (Different package)

**INPUT:**

package pkg2;

import pkg1.Animal;

public class Cat extends Animal {

@Override

public void speak( ) {

System.out.println("Cat meows (Overridden in different package)");

}

@Override

protected void eat( ) {

System.out.println("Cat eats fish (Overridden with protected access)");

}

// run( ) is private, not accessible either

}

📄 MainDemo.java

**INPUT:**

import pkg1.\*;

import pkg2.\*;

public class MainDemo {

public static void main(String[ ] args) {

System.out.println("Same Package (Dog):");

Dog dog = new Dog( );

dog.speak( ); // public - accessible

dog.eat ( ); // protected - accessible in same package

dog.sleep( ); // default - accessible in same package

dog.callPrivate( ); // private method accessed via public method

System.out.println("\nDifferent Package (Cat):");

Cat cat = new Cat( );

cat.speak( ); // public - accessible

cat.eat( ); // protected - accessible via subclass

}

}

**OUTPUT:**

E:\>javac pkg1\\*.java pkg2\\*.java MainDemo.java

E:\>java MainDemo

Same Package (Dog):

Dog barks (Overridden in same package)

Dog eats bones (Overridden)

Dog sleeps a lot (Overridden)

Animal runs (private)

Different Package (Cat):

Cat meows (Overridden in different package)

Cat eats fish (Overridden with protected access)

**PRACTICAL-18**

**AIM:** Write a program in Java to demonstrate multiple try block and multiple catch exception and include „divide by zero‟ and „Arithmetic exception‟.

**INPUT:**

public class MultipleTryCatchDemo {

public static void main(String[ ] args) {

System.out.println("M.J.J.B. :: Multiple Try-Catch Demo\n");

try {

int a = 10, b = 0;

int result = a / b; // This will cause divide by zero

System.out.println("Result = " + result);

} catch (ArithmeticException e) {

System.out.println("Caught ArithmeticException: " + e.getMessage());

}

try {

int[ ] arr = new int[3];

arr[5] = 100; // This will throw ArrayIndexOutOfBoundsException

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Caught ArrayIndexOutOfBoundsException: " +e.getMessage( ));

}

try {

String str = null;

System.out.println(str.length( )); // NullPointerException

} catch (NullPointerException e) {

System.out.println("Caught NullPointerException: " + e.getMessage());

}

System.out.println("\nProgram executed successfully.");

}

}

**OUTPUT:**

E:\>java MultipleTryCatchDemo

M.J.J.B. :: Multiple Try-Catch Demo

Caught ArithmeticException: / by zero

Caught ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

Caught NullPointerException: Cannot invoke "String.length()" because "str" is null

Program executed successfully.

**PRACTICAL-19**

**AIM:** Write a program in java to demonstrate use defined exception.

**INPUT:**

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class UserDefinedExceptionDemo {

static void validateAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age is less than 18. Not eligible to vote.");

} else {

System.out.println("M.J.J.B. :: Age is valid. Eligible to vote.");

}

}

public static void main(String[ ] args) {

try {

int inputAge = 16; // try changing this to 20 for valid case

System.out.println("M.J.J.B. :: Checking eligibility for age: " + inputAge);

validateAge(inputAge);

} catch (InvalidAgeException e) {

System.out.println("Exception Caught: " + e.getMessage( ));

}

System.out.println("M.J.J.B. :: Program completed.");

}

}

**OUTPUT:**

E:\>java UserDefinedExceptionDemo

M.J.J.B. :: Checking eligibility for age: 16

Exception Caught: Age is less than 18. Not eligible to vote.

M.J.J.B. :: Program completed.

===========================================

E:\>java UserDefinedExceptionDemo

M.J.J.B. :: Checking eligibility for age: 20

M.J.J.B. :: Age is valid. Eligible to vote.

M.J.J.B. :: Program completed.

**PRACTICAL-20**

**AIM:** Write a small application in Java to develop Banking Application in which user deposits the amount Rs. 2000.00 and then start withdrawing of Rs. 1500.00, Rs. 400.00 and it throws exception "Not Sufficient Fund" when user withdraws Rs. 500 thereafter.

**INPUT:**

// User-defined exception

class NotSufficientFundException extends Exception {

public NotSufficientFundException(String message) {

super(message);

}

}

class BankAccount {

private double balance;

// Constructor to deposit initial amount

public BankAccount(double initialBalance) {

balance = initialBalance;

System.out.println("M.J.J.B. :: Account created with balance: ₹" + balance);

}

// Method to withdraw amount

public void withdraw(double amount) throws NotSufficientFundException {

if (amount > balance) {

throw new NotSufficientFundException("Withdrawal of ₹" + amount + " failed: Not Sufficient Fund.");

} else {

balance -= amount;

System.out.println("M.J.J.B. :: Withdrawn ₹" + amount + " successfully. Remaining Balance: ₹" + balance);

}

}

}

public class BankingApp {

public static void main(String[ ] args) {

try {

BankAccount account = new BankAccount(2000.00);

account.withdraw(1500.00);

account.withdraw(400.00); // ₹100 left

account.withdraw(500.00); // This should throw exception

} catch (NotSufficientFundException e) {

System.out.println("M.J.J.B. :: Exception Caught - " + e.getMessage ( ) );

}

System.out.println("M.J.J.B. :: Transaction completed.");

}

}

**OUTPUT:**

E:\>java BankingApp

M.J.J.B. :: Account created with balance: ₹2000.0

M.J.J.B. :: Withdrawn ₹1500.0 successfully. Remaining Balance: ₹500.0

M.J.J.B. :: Withdrawn ₹400.0 successfully. Remaining Balance: ₹100.0

M.J.J.B. :: Exception Caught - Withdrawal of ₹500.0 failed: Not Sufficient Fund.

M.J.J.B. :: Transaction completed.

**PRACTICAL-21**

**AIM:** Write a program that executes two threads. One thread displays “Thread1” every 3,000 milliseconds, and the other displays “Thread2” every 5,000 milliseconds. Create the threads by extending the Thread class.

**INPUT:**

class ThreadOne extends Thread {

public void run ( ) {

try {

while (true) {

System.out.println("M.J.J.B. :: Thread1 is running");

Thread.sleep(3000); // 3 seconds

}

} catch (InterruptedException e) {

System.out.println("M.J.J.B. :: Thread1 Interrupted");

}

}

}

class ThreadTwo extends Thread {

public void run( ) {

try {

while (true) {

System.out.println("M.J.J.B. :: Thread2 is running");

Thread.sleep(5000); // 5 seconds

}

} catch (InterruptedException e) {

System.out.println("M.J.J.B. :: Thread2 Interrupted");

}

}

}

public class ThreadDemo {

public static void main(String[ ] args) {

ThreadOne t1 = new ThreadOne( );

ThreadTwo t2 = new ThreadTwo( );

System.out.println("M.J.J.B. :: Starting Threads...\n");

t1.start();

t2.start();

}

}

**OUTPUT:**

E:\>java ThreadDemo

M.J.J.B. :: Starting Threads...

M.J.J.B. :: Thread1 is running

M.J.J.B. :: Thread2 is running

M.J.J.B. :: Thread1 is running

M.J.J.B. :: Thread1 is running

M.J.J.B. :: Thread2 is running

...

**PRACTICAL-22**

**AIM:** Write a program in Java to demonstrate use of synchronization of threads when multiple threads are trying to update common variable.

**INPUT:**

class Account {

private int balance = 1000;

// Synchronized method to update balance

public synchronized void deposit(String threadName, int amount) {

System.out.println("M.J.J.B. :: " + threadName + " is trying to deposit ₹" + amount);

int newBalance = balance + amount;

try {

Thread.sleep(100); // Simulate delay

} catch (InterruptedException e) {

System.out.println("M.J.J.B. :: " + threadName + " was interrupted.");

}

balance = newBalance;

System.out.println("M.J.J.B. :: " + threadName + " updated balance: ₹" + balance);

}

}

class DepositThread extends Thread {

Account account;

String threadName;

int amount;

DepositThread(Account acc, String name, int amt) {

account = acc;

threadName = name;

amount = amt;

}

public void run( ) {

account.deposit(threadName, amount);

}

}

public class SyncDemo {

public static void main(String[ ] args) {

Account sharedAccount = new Account();

DepositThread t1 = new DepositThread(sharedAccount, "Thread1", 500);

DepositThread t2 = new DepositThread(sharedAccount, "Thread2", 800);

DepositThread t3 = new DepositThread(sharedAccount, "Thread3", 200);

System.out.println("M.J.J.B. :: Starting synchronized deposit threads...\n");

t1.start( );

t2.start( );

t3.start( );

}

}

**OUTPUT:**

E:\>java SyncDemo

M.J.J.B. :: Starting synchronized deposit threads...

M.J.J.B. :: Thread1 is trying to deposit ₹500

M.J.J.B. :: Thread1 updated balance: ₹1500

M.J.J.B. :: Thread2 is trying to deposit ₹800

M.J.J.B. :: Thread2 updated balance: ₹2300

M.J.J.B. :: Thread3 is trying to deposit ₹200

M.J.J.B. :: Thread3 updated balance: ₹2500

**PRACTICAL-23**

**AIM:** Write a program in java to use String class and compare two strings.

**INPUT:**

public class StringCompareDemo {

public static void main(String[ ] args) {

String str1 = "M.J.J.B.";

String str2 = "M.J.J.B.";

String str3 = new String("M.J.J.B.");

System.out.println("M.J.J.B. :: String Comparison Program Started");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// Using equals( )

System.out.println("M.J.J.B. :: Using equals( ): " + str1.equals(str3));

// Using ==

System.out.println("M.J.J.B. :: Using == : " + (str1 == str2));

System.out.println("M.J.J.B. :: Using == with new String : " + (str1 == str3));

// Using compareTo( )

int result = str1.compareTo(str3);

System.out.println("M.J.J.B. :: Using compareTo( ): " + result);

System.out.println("M.J.J.B. :: Program Ended.");

}

}

**OUTPUT:**

E:\>java StringCompareDemo

M.J.J.B. :: String Comparison Program Started

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

M.J.J.B. :: Using equals( ): true

M.J.J.B. :: Using == : true

M.J.J.B. :: Using == with new String : false

M.J.J.B. :: Using compareTo( ): 0

M.J.J.B. :: Program Ended.

**PRACTICAL-24**

**AIM:** Write a program in java to use StringBuffer class and perform concatenation of two Strings.

**INPUT:**

public class StringBufferDemo {

public static void main(String[ ] args) {

String str1 = "M.J.J.B.";

String str2 = " welcomes you!";

System.out.println("M.J.J.B. :: StringBuffer Concatenation Started");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

StringBuffer sb = new StringBuffer(str1);

sb.append(str2); // Concatenation

System.out.println("M.J.J.B. :: After Concatenation: " + sb.toString());

System.out.println("M.J.J.B. :: Program Ended.");

}

}

**OUTPUT:**

E:\>java StringBufferDemo

M.J.J.B. :: StringBuffer Concatenation Started

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

M.J.J.B. :: After Concatenation: M.J.J.B. welcomes you!

M.J.J.B. :: Program Ended.

**PRACTICAL-25**

**AIM:** Write a program in Java to create, write, modify, read operations on a Text file.

**INPUT:**

import java.io.\*;

public class FileDemo {

public static void main(String[ ] args) {

String filePath = "MJJB\_File.txt";

// 1. Create & Write

try {

FileWriter writer = new FileWriter(filePath);

writer.write("M.J.J.B. :: Welcome to File Handling in Java.\n");

writer.write("M.J.J.B. :: This is a demo file.\n");

writer.close( );

System.out.println("M.J.J.B. :: File Created and Data Written.");

} catch (IOException e) {

System.out.println("M.J.J.B. :: Error writing file: " + e.getMessage( ));

}

// 2. Modify (Append)

try {

FileWriter writer = new FileWriter(filePath, true); // 'true' enables appending

writer.write("M.J.J.B. :: This line is appended.\n");

writer.close( );

System.out.println("M.J.J.B. :: File Modified Successfully.");

} catch (IOException e) {

System.out.println("M.J.J.B. :: Error modifying file: " + e.getMessage());

}

// 3. Read

try {

BufferedReader reader = new BufferedReader(new FileReader(filePath));

String line;

System.out.println("\nM.J.J.B. :: Reading File Contents:");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

while ((line = reader.readLine( )) != null) {

System.out.println(line);

}

reader.close( );

} catch (IOException e) {

System.out.println("M.J.J.B. :: Error reading file: " + e.getMessage());

}

}

}

**OUTPUT:**

E:\>java FileDemo

M.J.J.B. :: File Created and Data Written.

M.J.J.B. :: File Modified Successfully.

M.J.J.B. :: Reading File Contents:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

M.J.J.B. :: Welcome to File Handling in Java.

M.J.J.B. :: This is a demo file.

M.J.J.B. :: This line is appended.