**Section 1: Defining, Implementing, and Importing Packages**

1. Create a Java package utilities that contains a class MathUtils with a method add(int a, int b) to return the sum of two numbers. Demonstrate the use of this package in another class.

utilities.java

package utilities;

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

}

Main.java

import utilities.MathUtils;

public class Main {

public static void main(String[] args) {

MathUtils mu = new MathUtils();

System.out.println("Sum: " + mu.add(5, 10));

}

}

1. Define a package shapes containing an interface Shape with methods double area() and double perimeter(). Implement the interface in classes Circle and Rectangle.

shapes.java

package shapes;

public interface Shape {

double area();

double perimeter();

}

shapes.java

package shapes;

public class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

public double area() {

return Math.PI \* radius \* radius;

}

public double perimeter() {

return 2 \* Math.PI \* radius;

}

}

Rectangle.java

package shapes;

public class Rectangle implements Shape {

private double length, width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

public double area() {

return length \* width;

}

public double perimeter() {

return 2 \* (length + width);

}

}

ShapeTest.java

import shapes.\*;

public class ShapeTest {

public static void main(String[] args) {

Shape circle = new Circle(5);

Shape rectangle = new Rectangle(4, 6);

System.out.println("Circle Area: " + circle.area());

System.out.println("Circle Perimeter: " + circle.perimeter());

System.out.println("Rectangle Area: " + rectangle.area());

System.out.println("Rectangle Perimeter: " + rectangle.perimeter());

}

}

1. Write a program to import classes from java.util and use ArrayList to store and display a list of integers.

import java.util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

System.out.println("Numbers: " + numbers);

}

}

**Section 2: Interfaces**

**Functional Interface and Lambda Expression**

1. Define a functional interface Calculator with a method int compute(int a, int b). Use a lambda expression to provide implementation for addition, subtraction, and multiplication.

@FunctionalInterface

interface Calculator {

int compute(int a, int b);

}

public class LambdaCalculator {

public static void main(String[] args) {

Calculator add = (a, b) -> a + b;

Calculator sub = (a, b) -> a - b;

Calculator mul = (a, b) -> a \* b;

System.out.println("Addition: " + add.compute(5, 3));

System.out.println("Subtraction: " + sub.compute(5, 3));

System.out.println("Multiplication: " + mul.compute(5, 3));

}

}

1. Write a Java program to sort a list of strings in descending order using a lambda expression.

import java.util.\*;

public class SortStringsDescending {

public static void main(String[] args) {

List<String> list = Arrays.asList("Apple", "Orange", "Banana");

list.sort((a, b) -> b.compareTo(a));

System.out.println(list);

}

}

1. Implement a method reference in a program to find the square of a number using a static method.

interface SquareFinder {

int findSquare(int num);

}

public class MethodReferenceExample {

public static int square(int num) {

return num \* num;

}

public static void main(String[] args) {

SquareFinder sf = MethodReferenceExample::square;

System.out.println("Square of 5: " + sf.findSquare(5));

}

}

**Section 3: Errors and Exceptions**

**Exception Handling**

1. Write a program that demonstrates the difference between try-catch and try-catch-finally blocks by dividing two numbers and handling ArithmeticException.

public class TryCatchExample {

public static void main(String[] args) {

// Try-Catch

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

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

}

// Try-Catch-Finally

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

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

} finally {

System.out.println("Finally block executed");

}

}

}

1. Create a custom exception InvalidAgeException that is thrown when a user's age is less than 18. Write a program to demonstrate its use.

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class CustomExceptionDemo {

public static void main(String[] args) {

try {

checkAge(16);

} catch (InvalidAgeException e) {

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

}

}

public static void checkAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age must be 18 or above");

}

}

}

**Built-in and Custom Exceptions**

1. Create a custom exception InvalidAgeException that is thrown when a user's age is less than 18. Handle the exception and log the error to a file using java.util.logging.

import java.io.IOException;

import java.util.logging.\*;

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class CustomExceptionLogging {

private static final Logger logger = Logger.getLogger(CustomExceptionLogging.class.getName());

public static void main(String[] args) throws IOException {

FileHandler fh = new FileHandler("error.log", true);

logger.addHandler(fh);

logger.setLevel(Level.SEVERE);

SimpleFormatter formatter = new SimpleFormatter();

fh.setFormatter(formatter);

try {

checkAge(16);

} catch (InvalidAgeException e) {

logger.severe(e.getMessage());

}

}

public static void checkAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age must be 18 or above");

}

}

}

1. Demonstrate the use of throw and throws keywords in a program that calculates the factorial of a number. Throw an exception if the input number is negative.

class NegativeNumberException extends Exception {

public NegativeNumberException(String message) {

super(message);

}

}

public class FactorialThrowsExample {

public static long factorial(int n) throws NegativeNumberException {

if (n < 0) {

throw new NegativeNumberException("Number cannot be negative");

}

long fact = 1;

for (int i = 1; i <= n; i++) {

fact \*= i;

}

return fact;

}

public static void main(String[] args) {

try {

System.out.println("Factorial: " + factorial(5));

System.out.println("Factorial: " + factorial(-3));

} catch (NegativeNumberException e) {

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

}

}

}