Experiment 1:

|  |
| --- |
| Working with user defined classes creating objects and using different types of constructors |

// User-defined class

class Person {

private String name;

private int age;

private String address;

// Default constructor (no arguments)

public Person() {

this.name = "Unknown";

this.age = 0;

this.address = "Unknown";

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

}

// Parameterized constructor (with name and age)

public Person(String name, int age) {

this.name = name;

this.age = age;

this.address = "Unknown"; // Initialize address with a default value

System.out.println("Parameterized constructor (name, age) called");

}

// Parameterized constructor (with name, age, and address)

public Person(String name, int age, String address) {

this.name = name;

this.age = age;

this.address = address;

System.out.println("Parameterized constructor (name, age, address) called");

}

// Copy constructor (creates a new object from an existing object)

public Person(Person other) {

this.name = other.name;

this.age = other.age;

this.address = other.address;

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

}

// Method to display object information

public void displayInfo() {

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

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

System.out.println("Address: " + address);

}

public String getName() {

return name;

}

}

public class UserDefinedClassExample {

public static void main(String[] args) {

// Creating objects of the Person class using different constructors

// 1. Using the default constructor

Person person1 = new Person();

System.out.println("\n--- Person 1 ---");

person1.displayInfo();

// 2. Using the parameterized constructor with name and age

Person person2 = new Person("Alice", 30);

System.out.println("\n--- Person 2 ---");

person2.displayInfo();

// 3. Using the parameterized constructor with name, age, and address

Person person3 = new Person("Bob", 25, "123 Main St");

System.out.println("\n--- Person 3 ---");

person3.displayInfo();

// 4. Using the copy constructor

Person person4 = new Person(person3); // Create a copy of person3

System.out.println("\n--- Person 4 (Copy of Person 3) ---");

person4.displayInfo();

//Demonstrate the use of getName()

System.out.println("\n--- Person 1 getName() ---");

System.out.println("Person 1's Name: " + person1.getName());

}

}

Experiment 2:

|  |
| --- |
| Program to perform simple String operations |

public class StringOperations {

public static void main(String[] args) {

// Sample strings

String str1 = "Hello";

String str2 = "World";

String str3 = " Hello World ";

String str4 = "hello world";

System.out.println("String Operations:");

// 1. Concatenation

System.out.println("\n1. Concatenation:");

System.out.println("str1 + str2: " + str1 + str2); // Using + operator

System.out.println("str1.concat(str2): " + str1.concat(str2)); // Using concat() method

// 2. Length

System.out.println("\n2. Length:");

System.out.println("str1.length(): " + str1.length());

// 3. Case Conversion

System.out.println("\n3. Case Conversion:");

System.out.println("str1.toUpperCase(): " + str1.toUpperCase());

System.out.println("str1.toLowerCase(): " + str1.toLowerCase());

// 4. Trimming

System.out.println("\n4. Trimming:");

System.out.println("str3: \"" + str3 + "\"");

System.out.println("str3.trim(): \"" + str3.trim() + "\""); // Remove leading and trailing spaces

// 5. Substring

System.out.println("\n5. Substring:");

System.out.println("str1.substring(2): " + str1.substring(2)); // Get substring from index 2

System.out.println("str1.substring(1, 4): " + str1.substring(1, 4)); // Get substring from index 1 to 3

// 6. Character at Index

System.out.println("\n6. Character at Index:");

System.out.println("str1.charAt(0): " + str1.charAt(0)); // Get the character at index 0

// 7. String Comparison

System.out.println("\n7. String Comparison:");

System.out.println("str1.equals(str4): " + str1.equals(str4)); // Case-sensitive comparison

System.out.println("str1.equalsIgnoreCase(str4): " + str1.equalsIgnoreCase(str4)); // Case-insensitive

System.out.println("str1.compareTo(str4): " + str1.compareTo(str4)); //Lexicographical comparison

System.out.println("str1.compareToIgnoreCase(str4): " + str1.compareToIgnoreCase(str4)); // Lexicographical and Case-insensitive

// 8. Checking if a String Contains another String

System.out.println("\n8. Checking if a String Contains another String:");

System.out.println("str1.contains(\"ell\"): " + str1.contains("ell"));

// 9. Replacing characters

System.out.println("\n9. Replacing characters:");

System.out.println("str1.replace('l', 'p'): " + str1.replace('l', 'p'));

// 10. Splitting a string

String str5 = "apple,banana,orange";

System.out.println("\n10. Splitting a string:");

String[] fruits = str5.split(",");

for (String fruit : fruits) {

System.out.println(fruit);

}

// 11. Checking if a string starts or ends with a specific string

System.out.println("\n11. startsWith() and endsWith():");

System.out.println("str1.startsWith(\"He\"): " + str1.startsWith("He"));

System.out.println("str1.endsWith(\"lo\"): " + str1.endsWith("lo"));

// 12. Checking if the string is empty

String str6 = "";

String str7 = " "; //not empty because it has a space

System.out.println("\n12. isEmpty():");

System.out.println("str6.isEmpty(): " + str6.isEmpty());

System.out.println("str7.isEmpty(): " + str7.isEmpty());

}

}

Experiment 3:

|  |
| --- |
| Program to implement Inheritance using super keyword and super() call |

// Base class (Parent class)

class Animal {

private String name;

private int age;

public Animal(String name, int age) {

this.name = name;

this.age = age;

System.out.println("Animal constructor called");

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public void eat() {

System.out.println(name + " is eating.");

}

public void sleep() {

System.out.println(name + " is sleeping.");

}

@Override

public String toString() {

return "Animal{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

// Derived class (Child class) inheriting from Animal

class Dog extends Animal {

private String breed;

public Dog(String name, int age, String breed) {

super(name, age); // Calling the constructor of the superclass (Animal)

this.breed = breed;

System.out.println("Dog constructor called");

}

public String getBreed() {

return breed;

}

public void bark() {

System.out.println(getName() + " is barking."); // Using getName() from the superclass

}

// Overriding the eat() method of the superclass

@Override

public void eat() {

super.eat(); // Calling the eat() method of the superclass

System.out.println(getName() + " is eating dog food.");

}

@Override

public String toString() {

return "Dog{" +

"breed='" + breed + '\'' +

"} " + super.toString(); // Calling toString() of the superclass

}

}

// Main class to demonstrate inheritance

public class InheritanceExample {

public static void main(String[] args) {

// Creating an object of the derived class (Dog)

Dog myDog = new Dog("Buddy", 3, "Golden Retriever");

// Accessing methods and fields from the superclass (Animal)

System.out.println("Dog's name: " + myDog.getName());

System.out.println("Dog's age: " + myDog.getAge());

System.out.println("Dog's breed: " + myDog.getBreed());

myDog.eat(); // Calling the overridden eat() method

myDog.sleep(); // Calling the inherited sleep() method

myDog.bark(); // Calling the method specific to the Dog class

System.out.println(myDog); // Calling the toString() method

}

}

Experiment 4:

|  |
| --- |
| Program to create and Implement user defined interface |

// Define a user-defined interface

interface Shape {

// Abstract methods (no implementation)

double getArea();

double getPerimeter();

String getName(); // Added getName() for better illustration

}

// Class implementing the Shape interface

class Circle implements Shape {

private double radius;

private String name;

public Circle(double radius, String name) {

this.radius = radius;

this.name = name;

}

@Override

public double getArea() {

return Math.PI \* radius \* radius;

}

@Override

public double getPerimeter() {

return 2 \* Math.PI \* radius;

}

@Override

public String getName() {

return name;

}

}

// Class implementing the Shape interface

class Rectangle implements Shape {

private double length;

private double width;

private String name;

public Rectangle(double length, double width, String name) {

this.length = length;

this.width = width;

this.name = name;

}

@Override

public double getArea() {

return length \* width;

}

@Override

public double getPerimeter() {

return 2 \* (length + width);

}

@Override

public String getName() {

return name;

}

}

// Main class to demonstrate interface usage

public class InterfaceExample {

public static void main(String[] args) {

// Create objects of classes that implement the Shape interface

Circle circle = new Circle(5, "Circle");

Rectangle rectangle = new Rectangle(4, 6, "Rectangle");

// Create an array of Shape objects. This is possible because both

// Circle and Rectangle implement the Shape interface.

Shape[] shapes = new Shape[2];

shapes[0] = circle;

shapes[1] = rectangle;

// Iterate through the array and use the interface methods

for (Shape shape : shapes) {

System.out.println("Shape: " + shape.getName());

System.out.println("Area: " + shape.getArea());

System.out.println("Perimeter: " + shape.getPerimeter());

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

}

}

}

Experiment 5:

|  |
| --- |
| Program on creating own package and using built in packages. |

// Creating a custom package named 'arithmetic'

package arithmetic;

// Class inside the custom package for basic arithmetic operations

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public double divide(int a, int b) {

if (b == 0) {

return Double.NaN; // Handle division by zero

}

return (double) a / b;

}

}

// Main class demonstrating package usage

import arithmetic.Calculator; // Importing the custom class

import java.util.Scanner; // Importing a class from the built-in java.util package

public class ArithmeticPackageExample {

public static void main(String[] args) {

// Using a class from the custom package

Calculator calculator = new Calculator();

// Using a class from the built-in java.util package

Scanner scanner = new Scanner(System.in);

System.out.println("Enter two numbers:");

System.out.print("Number 1: ");

int num1 = scanner.nextInt();

System.out.print("Number 2: ");

int num2 = scanner.nextInt();

System.out.println("Performing arithmetic operations using custom package:");

System.out.println("Addition: " + calculator.add(num1, num2));

System.out.println("Subtraction: " + calculator.subtract(num1, num2));

System.out.println("Multiplication: " + calculator.multiply(num1, num2));

System.out.println("Division: " + calculator.divide(num1, num2));

scanner.close();

}

}

Experiment 6 :

|  |
| --- |
| Java program on Lambda expression to perform arithmetic operations |

import java.util.function.BiFunction;

public class ArithmeticOperations {

public static void main(String[] args) {

// Lambda expression for addition

BiFunction<Integer, Integer, Integer> addition = (a, b) -> a + b;

// Lambda expression for subtraction

BiFunction<Integer, Integer, Integer> subtraction = (a, b) -> a - b;

// Lambda expression for multiplication

BiFunction<Integer, Integer, Integer> multiplication = (a, b) -> a \* b;

// Lambda expression for division

BiFunction<Integer, Integer, Double> division = (a, b) -> {

if (b == 0) {

System.out.println("Error: Cannot divide by zero.");

return Double.NaN; // Not-a-Number

}

return (double) a / b;

};

int num1 = 10;

int num2 = 5;

System.out.println("Performing arithmetic operations:");

System.out.println(num1 + " + " + num2 + " = " + operate(num1, num2, addition));

System.out.println(num1 + " - " + num2 + " = " + operate(num1, num2, subtraction));

System.out.println(num1 + " \* " + num2 + " = " + operate(num1, num2, multiplication));

System.out.println(num1 + " / " + num2 + " = " + operate(num1, num2, division));

int num3 = 8;

int num4 = 0;

System.out.println(num3 + " / " + num4 + " = " + operate(num3, num4, division));

}

// A generic method to perform an operation using a BiFunction

public static <T, U, R> R operate(T operand1, U operand2, BiFunction<T, U, R> operation) {

return operation.apply(operand1, operand2);

}

}

Experiment 7:

|  |
| --- |
| Java Program to perform operations on Text Files |

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class TextFileOperations {

public static void main(String[] args) {

String filename = "my\_text\_file.txt";

// Write some data to the file

writeDataToFile(filename, List.of("This is the first line.",

"This is the second line.",

"And this is the third line."));

// Read data from the file

List<String> lines = readDataFromFile(filename);

if (lines != null) {

System.out.println("\nContents of the file:");

for (String line : lines) {

System.out.println(line);

}

}

// Append more data to the file

appendDataToFile(filename, List.of("This line is appended.",

"Another appended line."));

// Read data again to see the appended content

List<String> updatedLines = readDataFromFile(filename);

if (updatedLines != null) {

System.out.println("\nContents of the file after appending:");

for (String line : updatedLines) {

System.out.println(line);

}

}

}

// Method to write data to a text file

public static void writeDataToFile(String filename, List<String> data) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {

for (String line : data) {

writer.write(line);

writer.newLine(); // Add a newline character after each line

}

System.out.println("Data successfully written to: " + filename);

} catch (IOException e) {

System.err.println("Error writing to file: " + e.getMessage());

}

}

// Method to read data from a text file

public static List<String> readDataFromFile(String filename) {

List<String> lines = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

String line;

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

lines.add(line);

}

return lines;

} catch (IOException e) {

System.err.println("Error reading from file: " + e.getMessage());

return null;

}

}

// Method to append data to a text file

public static void appendDataToFile(String filename, List<String> data) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename, true))) { // The 'true' argument enables append mode

for (String line : data) {

writer.write(line);

writer.newLine(); // Add a newline character after each line

}

System.out.println("Data successfully appended to: " + filename);

} catch (IOException e) {

System.err.println("Error appending to file: " + e.getMessage());

}

}

}

Experiment 8:

|  |
| --- |
| Program to use built in exceptions and to create and handle User Defined Excpetions |

import java.util.InputMismatchException;

import java.util.Scanner;

// User-defined exception for negative numbers

class NegativeNumberException extends Exception {

public NegativeNumberException(String message) {

super(message);

}

}

// User-defined exception for division by zero (optional, as ArithmeticException already exists, but for demonstration)

class DivisionByZeroException extends Exception {

public DivisionByZeroException(String message) {

super(message);

}

}

public class ArithmeticOperationsWithExceptions {

// Function to perform addition

public static int add(int a, int b) {

return a + b;

}

// Function to perform subtraction

public static int subtract(int a, int b) {

return a - b;

}

// Function to perform multiplication

public static int multiply(int a, int b) {

return a \* b;

}

// Function to perform division

public static double divide(int a, int b) throws DivisionByZeroException {

if (b == 0) {

throw new DivisionByZeroException("Cannot divide by zero."); //Use user defined exception

}

return (double) a / b;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter the first number: ");

int num1 = scanner.nextInt();

if (num1 < 0) {

throw new NegativeNumberException("First number cannot be negative.");

}

System.out.print("Enter the second number: ");

int num2 = scanner.nextInt();

if (num2 < 0) {

throw new NegativeNumberException("Second number cannot be negative.");

}

System.out.println("Performing arithmetic operations:");

System.out.println("Addition: " + add(num1, num2));

System.out.println("Subtraction: " + subtract(num1, num2));

System.out.println("Multiplication: " + multiply(num1, num2));

try {

System.out.println("Division: " + divide(num1, num2));

} catch (DivisionByZeroException e) {

System.err.println("Error during division: " + e.getMessage()); // Handle user defined Exception

}

} catch (InputMismatchException e) {

System.err.println("Error: Invalid input. Please enter integers only.");

} catch (NegativeNumberException e) {

System.err.println("Error: " + e.getMessage()); // Handle user defined Exception

} finally {

scanner.close();

System.out.println("Scanner closed.");

}

System.out.println("End of program.");

}

}

Experiment 9:

|  |
| --- |
| Program on Performming parallel tasks using multithreading |

class Task implements Runnable {

private String taskName;

public Task(String name) {

this.taskName = name;

}

@Override

public void run() {

System.out.println("Starting task: " + taskName + " in thread: " + Thread.currentThread().getName());

// Simulate some work

try {

Thread.sleep((long) (Math.random() \* 5000)); // Sleep for a random duration (up to 5 seconds)

} catch (InterruptedException e) {

System.out.println("Task: " + taskName + " interrupted.");

Thread.currentThread().interrupt();

}

System.out.println("Finished task: " + taskName + " in thread: " + Thread.currentThread().getName());

}

}

public class ParallelTasks {

public static void main(String[] args) {

// Create multiple tasks

Task task1 = new Task("Data Processing");

Task task2 = new Task("File Upload");

Task task3 = new Task("Report Generation");

// Create threads for each task

Thread thread1 = new Thread(task1);

Thread thread2 = new Thread(task2);

Thread thread3 = new Thread(task3);

// Start the threads

System.out.println("Starting parallel tasks...");

thread1.start();

thread2.start();

thread3.start();

System.out.println("Main thread continues to execute...");

}

}

Experiment 10:

|  |
| --- |
| Program based on List Interface |

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

public class ListExample {

public static void main(String[] args) {

// Using ArrayList (dynamically resizable array)

List<String> arrayList = new ArrayList<>();

System.out.println("ArrayList Example:");

performListOperations(arrayList);

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

// Using LinkedList (doubly-linked list)

List<String> linkedList = new LinkedList<>();

System.out.println("LinkedList Example:");

performListOperations(linkedList);

}

public static void performListOperations(List<String> list) {

// Adding elements

list.add("Apple");

list.add("Banana");

list.add("Orange");

System.out.println("Initial list: " + list);

// Adding an element at a specific index

list.add(1, "Mango");

System.out.println("List after adding at index 1: " + list);

// Accessing an element by index

String elementAtIndex2 = list.get(2);

System.out.println("Element at index 2: " + elementAtIndex2);

// Checking if an element exists

boolean containsBanana = list.contains("Banana");

System.out.println("Does the list contain 'Banana'? " + containsBanana);

// Finding the index of an element

int indexOfOrange = list.indexOf("Orange");

System.out.println("Index of 'Orange': " + indexOfOrange);

// Getting the size of the list

int size = list.size();

System.out.println("Size of the list: " + size);

// Iterating through the list using a for-each loop

System.out.println("Elements in the list:");

for (String fruit : list) {

System.out.println(fruit);

}

// Removing an element by value

list.remove("Banana");

System.out.println("List after removing 'Banana': " + list);

// Removing an element by index

list.remove(0);

System.out.println("List after removing element at index 0: " + list);

// Clearing the list

list.clear();

System.out.println("List after clearing: " + list);

// Checking if the list is empty

boolean isEmpty = list.isEmpty();

System.out.println("Is the list empty? " + isEmpty);

}

}

Experiment 11: Java program to implement TCP client server

import java.io.\*;

import java.net.\*;

public class TCPClientServer {

// Server program

public static class TCPServer {

public static void main(String[] args) {

final int PORT = 12345; // Port number for the server

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Server started. Listening on port " + PORT);

while (true) {

try (Socket clientSocket = serverSocket.accept(); // Accept incoming client connection

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

System.out.println("Client connected from: " + clientSocket.getInetAddress());

// Read message from the client

String message = in.readLine();

System.out.println("Received from client: " + message);

// Send a response back to the client

String response = "Server received: " + message;

out.println(response);

System.out.println("Sent to client: " + response);

} catch (IOException e) {

System.err.println("Exception in server: " + e.getMessage());

}

// The client socket and streams are automatically closed due to try-with-resources

}

// The server socket is automatically closed due to try-with-resources

} catch (IOException e) {

System.err.println("Error starting server: " + e.getMessage());

}

System.out.println("Server stopped."); //This line will never be executed as the server runs in an infinite loop.

}

}

// Client program

public static class TCPClient {

public static void main(String[] args) {

final String SERVER\_ADDRESS = "localhost"; // Or the actual server IP address

final int PORT = 12345;

try (Socket socket = new Socket(SERVER\_ADDRESS, PORT); // Connect to the server

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))) {

System.out.println("Connected to server: " + SERVER\_ADDRESS + " on port " + PORT);

String userInput;

System.out.println("Type your message (or type 'exit' to close connection):");

while ((userInput = stdIn.readLine()) != null) {

out.println(userInput); // Send message to the server

System.out.println("Sent to server: " + userInput);

if ("exit".equalsIgnoreCase(userInput)) {

break; // Exit the loop if user enters "exit"

}

String serverResponse = in.readLine(); // Read response from the server

System.out.println("Received from server: " + serverResponse);

System.out.println("Type your message (or type 'exit' to close connection):");

}

System.out.println("Closing connection.");

} catch (IOException e) {

System.err.println("Exception in client: " + e.getMessage());

}

System.out.println("Client stopped.");

}

}

}