CHENNAI INSTITUTE OF TECHNOLOGY
(Autonomous)

REGULATION 2022

LAB MANUAL

CS3203- INTRODUCTION TO JAVA PROGRAMMING

II SEMESTER

COMMON TO CSE, IT, AI&DS, AI&ML, CSBS, CZ BRANCHES

Name of the Student :

Register Number :

Year / Semester / Section :

Batch :

CONTENT

EXP. NO	DATE OF THE EXP	TITLE	CO Mapped	PO Mapped	MARKS	SIGN OF THE FACULTY
1.		Sequential search	CO1	PO1,PO2,PO3		
2.		Binary search	CO1	PO1,PO2,PO3		
3.		Selection Sort	CO1	PO1,PO2,PO3		
4.		Insertion Sort	CO1	PO1,PO2,PO3		
5.		Stack data structures using classes and objects.	CO1	PO1,PO2,PO3		
6.		Queue data structures using classes and objects.	CO1	PO1,PO2,PO3		
7.		Printing Unique Values	CO1	PO1,PO2,PO3		
8.		Inheritance-Vehicle	CO2	PO1,PO2,PO3		
9.		Inheritance- Employee details	CO2	PO1,PO2,PO3		
10.		Inheritance- Shapes	CO2	PO1,PO2,PO3		
11.		Exception handling	CO3	PO1,PO2,PO3		

12.		User defined exception class	CO3	PO1,PO2,PO3		
13.		Reading and Writing Files	CO3	PO1,PO2,PO3		
14.		Creating threads	CO4	PO1,PO2,PO3		
15.		Multi-threaded application - generating a random integer	CO4	PO1,PO2,PO3		
16.		mapping the words of the numbers using Map Interface	CO5	PO1,PO2,PO3		
17.		Connect with a database Using JDBC	CO5	PO1,PO2,PO3		

EX.NO: 1	SEQUENTIAL SEARCH
DATE:	

AIM:

To write a java program for Solving problems by using sequential search.

ALGORITHM:

Step 1: Start with an input array and a target element.
Step 2: Iterate through each element of the array sequentially, starting from the first element.
Step 3: Compare each element with the target element.
Step 4: If a match is found, return the index of the element.
Step 5: If no match is found after iterating through all elements, return -1.
Step 6: Print the result indicating whether the target element was found or not, along with its index if found.
Step 7: Stop the program.

PROGRAM:

```
package PGM;


import java.util.Scanner;

public class SequentialSearch {

    // Sequential search method
    public static int sequentialSearch(int[] array, int target) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == target) {
                return i; // Return the index where the target is found
            }
        }
        return -1; // Return -1 if target is not found in the array
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Ask the user to input the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        int[] array = new int[size];
```

```
        // Ask the user to input the elements of the array
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            array[i] = scanner.nextInt();
        }

        // Ask the user to input the target element to search for
        System.out.print("Enter the target element to search for: ");
        int target = scanner.nextInt();

        scanner.close(); // Close the scanner

        int index = sequentialSearch(array, target);

        if (index != -1) {
            System.out.println("Element " + target + " found at index " +
index);
        } else {
            System.out.println("Element " + target + " not found in the
array");
        }
    }
}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the elements of the array:
Element 1: 5
Element 2: 22
Element 3: 34
Element 4: 21
Element 5: 34
Enter the target element to search for: 34
Element 34 found at index 2


Enter the size of the array: 5
Enter the elements of the array:
Element 1: 2
Element 2: 8
Element 3: 6
Element 4: 4
Element 5: 5
Enter the target element to search for: 7
Element 7 not found in the array
```

INFERENCE:

The time complexity of sequential search is O(n), where n is the number of elements in the array. This means that the time taken to search increases linearly with the size of the array.

RESULT:

The program was successfully executed and result is obtained

EX.NO:2	BINARY SEARCH
DATE:	

AIM:

To write a java program to search a number using binary search algorithm.

ALGORITHM:

Step 1: Import Required Packages:

- Import **java.util.Arrays** for array sorting and **java.util.Scanner** for user input.

Step 2: Define Binary Search Method:

- Create a method named **binarySearch** that takes two parameters: the sorted array and the target element.
- Initialize **left** and **right** pointers to the start and end of the array, respectively.
- Use a while loop to divide the search space in half until the target element is found or the search space is exhausted.
- Check if the target element is at the midpoint and return its index if found.
- Update the **left** and **right** pointers based on whether the target is greater or smaller than the midpoint.
- If the target element is not found, return -1.

Step 3: Define Main Method:

- Create the **main** method.
- Create a **Scanner** object to take user input.
- Prompt the user to enter the size of the array.
- Create an array of the specified size and prompt the user to enter each element.
- Prompt the user to enter the target element to search for.
- Close the **Scanner** object after taking input.

Step 4: Sort the Array:

- Use **Arrays.sort()** to sort the array in ascending order.
- Binary search requires a sorted array for correctness.

Step 5: Perform Binary Search:

- Call the **binarySearch** method with the sorted array and target element.
- Store the result (index of the target element or -1) in a variable.

Step 6: Print Result:

- Check if the index returned by the binary search method is not equal to -1.
- If so, print the index where the target element is found.
- Otherwise, print a message indicating that the target element is not found in the array.

PROGRAM:

```
import java.util.Arrays;
import java.util.Scanner;

public class BinarySearch {

    // Binary search method
```

```
public static int binarySearch(int[] array, int target) {
    int left = 0;
    int right = array.length - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if target is present at mid
        if (array[mid] == target) {
            return mid;
        }

        // If target is greater, ignore left half
        if (array[mid] < target) {
            left = mid + 1;
        }
        // If target is smaller, ignore right half
        else {
            right = mid - 1;
        }
    }

    // If target is not found in the array
    return -1;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Take input for the size of the array
    System.out.print("Enter the size of the array: ");
    int size = scanner.nextInt();

    // Create an array of the specified size
    int[] array = new int[size];

    // Take input for each element of the array
    System.out.println("Enter the elements of the array in sorted order:");
    for (int i = 0; i < size; i++) {
        System.out.print("Element " + (i + 1) + ": ");
        array[i] = scanner.nextInt();
    }

    // Take input for the target element to search for
    System.out.print("Enter the target element to search for: ");
    int target = scanner.nextInt();

    scanner.close();

    // Sort the array (binary search requires a sorted array)
    Arrays.sort(array);

    // Perform binary search
    int index = binarySearch(array, target);

    if (index != -1) {
```

```
            System.out.println("Element " + target + " found at index " + index);
        } else {
            System.out.println("Element " + target + " not found in the array");
        }
    }
}
```

OUTPUT

```
Enter the size of the array: 5
Enter the elements of the array in sorted order:
Element 1: 10
Element 2: 20
Element 3: 30
Element 4: 40
Element 5: 50
Enter the target element to search for: 20
Element 20 found at index 1
```

INFERENCE:

The program shows the implementation of a fundamental searching algorithm in Java, demonstrating principles of efficiency, user interaction, and robustness. It provides users with a tool to efficiently search for target elements in sorted arrays, making it a valuable utility for various applications.

RESULT

The program was successfully executed and result is obtained

EX.NO: 3	Selection Sort
DATE:	

AIM:

To write a java program to sort list of numbers using Selection Sort (Quadratic sorting) algorithm

ALGORITHM:

Step 1: Start with the input array.

Step 2: Iterate through each element of the array.

Step 3: Find the minimum element in the unsorted part.

Step 4: Swap the minimum element with the first unsorted element.

Step 5: Repeat steps 2-4 until the entire array is sorted.

PROGRAM:

```
// Selection sort method
    public static void selectionSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap the found minimum element with the first element
            int temp = array[minIndex];
            array[minIndex] = array[i];
            array[i] = temp;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
// Ask the user to input the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        int[] array = new int[size];

        // Ask the user to input the elements of the array
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            array[i] = scanner.nextInt();
        }

        // Perform selection sort on the array
        selectionSort(array);

        // Print the sorted array
        System.out.println("Sorted array:");
        for (int i = 0; i < size; i++) {
            System.out.print(array[i] + " ");
        }

        scanner.close(); // Close the scanner
    }
}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the elements of the array:
Element 1: 5
Element 2: 3
Element 3: 7
Element 4: 2
Element 5: 1
Sorted array:
1 2 3 5 7
```

INFERENCE:

The program uses Selection Sort, which takes longer to sort larger arrays. As the array size increases, the time taken to sort it grows significantly. Other sorting methods like Merge Sort or Quick Sort are faster for larger arrays.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 4	Insertion Sort
DATE:	

AIM:

To write a java program to sort list of numbers using insertion Sort (Quadratic sorting) algorithm

ALGORITHM:

Step 1: Initialization: Start with the input array.
Step 2: Iteration: Iterate through each element of the array, starting from the second element.
Step 3: Insertion: For each element, compare it with the elements before it and insert it into the correct position among the already sorted elements.
Step 4: Repeat: Continue this process until all elements are sorted.
Step 5: Stop: Once all elements are sorted, the array is completely sorted.

PROGRAM:

```
public class InsertionSort {

    // Insertion sort method
    public static void insertionSort(int[] array) {
        int n = array.length;
        for (int i = 1; i < n; ++i) {
            int key = array[i];
            int j = i - 1;

            // Move elements of array[0..i-1], that are greater than key,
            // to one position ahead of their current position
            while (j >= 0 && array[j] > key) {
                array[j + 1] = array[j];
                j = j - 1;
            }
            array[j + 1] = key;
        }
    }

    public static void main(String[] args) {
        int[] array = {12, 11, 13, 5, 6};
```

```
        System.out.println("Array before sorting:");
        printArray(array);

        insertionSort(array);

        System.out.println("Array after sorting:");
        printArray(array);
    }

    // Utility method to print an array
    public static void printArray(int[] array) {
        for (int i = 0; i < array.length; ++i) {
            System.out.print(array[i] + " ");
        }
        System.out.println();
    }
}
```

OUTPUT:

```
Array before sorting:
12 11 13 5 6
Array after sorting:
5 6 11 12 13
```

INFERENCE:

The program successfully sorts an array of integers using the Insertion Sort algorithm. It starts with an unsorted array {12, 11, 13, 5, 6} and rearranges it into a sorted array {5, 6, 11, 12, 13}. This demonstrates the effectiveness of Insertion Sort in arranging elements in ascending order.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 5	STACK DATA STRUCTURES USING CLASSES AND OBJECTS
DATE:	

AIM:

To write a java program to demonstrate Stack data structure using classes and objects.

ALGORITHM:

Step 1: Create a class for either Stack.
Step 2: Include methods for push/pop (Stack).
Step 3: Use arrays to store elements in both data structures.
Step 4: Keep track of the top (Stack) indices.
Step 5: Add a method to check if the data structure is empty.

PROGRAM:

```
STACK:(program 1)
class Stack {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public Stack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }

    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack overflow");
            return;
        }
        stackArray[++top] = value;
    }

    public int pop() {
```

```
        if (isEmpty()) {
            System.out.println("Stack underflow");
            return -1;
        }
        return stackArray[top--];
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty");
            return -1;
        }
        return stackArray[top];
    }

    public boolean isEmpty() {
        return (top == -1);
    }

    public boolean isFull() {
        return (top == maxSize - 1);
    }
}

public class StackExample {
    public static void main(String[] args) {
        Stack stack = new Stack(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Top element: " + stack.peek());

        System.out.println("Popped element: " + stack.pop());
        System.out.println("Popped element: " + stack.pop());
    }
}
```

OUTPUT:

```
Top element: 30
```

```
Popped element: 30
Popped element: 20
```

INFERENCE:

The Stack program demonstrates Last In, First Out (LIFO) behavior, common in computer science for managing data.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 6	QUEUE DATA STRUCTURES USING CLASSES AND OBJECTS
DATE:	

AIM:

To write a java program to demonstrate queue data structure using classes and objects.

ALGORITHM:

Step 1: Create a class for Queue.
Step 2: Include methods for enqueue/dequeue (Queue).
Step 3: Use arrays to store elements.
Step 4: Keep track of the front/rear (Queue) indices.
Step 5: Add a method to check if the data structure is empty.

PROGRAM:

```
class Queue {
    private int maxSize;
    private int[] queueArray;
    private int front;
    private int rear;
    private int nItems;

    public Queue(int size) {
        maxSize = size;
        queueArray = new int[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }

    public void insert(int value) {
        if (isFull()) {
            System.out.println("Queue is full");
            return;
        }
        if (rear == maxSize - 1) {
            rear = -1;
        }
        queueArray[++rear] = value;
        nItems++;
    }

    public int remove() {
        if (isEmpty()) {
```

```
            System.out.println("Queue is empty");
            return -1;
        }
        int temp = queueArray[front++];
        if (front == maxSize) {
            front = 0;
        }
        nItems--;
        return temp;
    }

    public int peekFront() {
        return queueArray[front];
    }

    public boolean isEmpty() {
        return (nItems == 0);
    }

    public boolean isFull() {
        return (nItems == maxSize);
    }
}

public class QueueExample {
    public static void main(String[] args) {
        Queue queue = new Queue(5);

        queue.insert(10);
        queue.insert(20);
        queue.insert(30);

        System.out.println("Front element: " + queue.peekFront());

        System.out.println("Removed element: " + queue.remove());
        System.out.println("Removed element: " + queue.remove());
    }
}
```

INFERENCE:

The Queue program illustrates First In, First Out (FIFO) behavior, common in computer science for managing data.

OUTPUT:

```
Front element: 10
Removed element: 10
Removed element: 20
```

RESULT:

The program was successfully executed and result is obtained

EX.NO: 7	Printing Unique Values
DATE:	

AIM:

To write a java program to Printing Unique Values

ALGORITHM:

Step 1: Create a HashSet to store unique values.
Step 2: Iterate through each element of the array.
Step 3: For each element, check if it exists in the HashSet.
Step 4: If it doesn't exist, add it to the HashSet and print it.
Step 5: If it already exists, skip it.
Step 6: Continue this process until all elements are processed.
Step 7: Stop

PROGRAM:

```
import java.util.HashSet;

public class UniqueValues {

    // Method to print unique values from an array
    public static void printUnique(int[] array) {
        HashSet<Integer> uniqueSet = new HashSet<>();

        System.out.println("Unique values:");
        for (int i = 0; i < array.length; i++) {
           if (!uniqueSet.contains(array[i])) {
                System.out.println(array[i]);
                uniqueSet.add(array[i]);
            }
        }
    }

    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 2, 3, 5};

        printUnique(array);
    }
}
```

OUTPUT:

```
Unique values:
1
```

2
3
4
5

INFERENCE:

The program quickly finds and prints unique values from an array. It uses a HashSet to store unique elements and prints each unique value only once. This approach ensures simplicity and efficiency in extracting distinct elements from the array.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 8	Inheritance-Vehicle
DATE:	

AIM:

To write a java program to demonstrate Inheritance.

ALGORITHM:

Step 1: Begin by creating a class with basic features.
Step 2: Create another class that "extends" or "inherits" from the first class.
Step 3: The new class automatically gains all the features of the first class.
Step 4: Add new features or change existing ones in the new class if needed.
Step 5: Utilize inheritance to share and reuse code easily.

PROGRAM:

```
// Parent class
class Vehicle {
    void display() {
        System.out.println("This is a vehicle.");
    }
}

// Child class inheriting from Vehicle
class Car extends Vehicle {
    void displayCar() {
        System.out.println("This is a car.");
    }
}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();

        car.display(); // Inherited method from Vehicle class
        car.displayCar(); // Method specific to Car class
    }
}
```

OUTPUT:

```
This is a vehicle.
This is a car.
```

INFERENCE:

Inheritance lets one class use features (methods and attributes) from another class, promoting code reuse and organization. It simplifies code maintenance by allowing specialized classes to inherit common features from a parent class.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 9	Inheritance- Employee details
DATE:	

AIM:

To write a java program to create a simple Java program to demonstrate inheritance with an "Employee" class and a "Manager" class that inherits from it.

ALGORITHM:

Step 1: Define a class with common attributes (name, age) and a method to display employee details
Step 2: Define another class that inherits from the Employee class and adds a new attribute (department).
Step 3: Create an object of the Manager class and initialize it with employee details.
Step 4: Call the method to display manager details, which also invokes the parent class's method to display common employee details.

PROGRAM:

```
class Employee {
    String name;
    int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

class Manager extends Employee {
    String department;

    public Manager(String name, int age, String department) {
        super(name, age);
```

```
        this.department = department;
    }

    public void displayManagerDetails() {
        super.displayDetails();
        System.out.println("Department: " + department);
    }
}

public class Main {
    public static void main(String[] args) {
        Manager manager = new Manager("John", 35, "HR");
        manager.displayManagerDetails();
    }
}
```

OUTPUT:

```
Name: John
Age: 35
Department: HR
```

INFERENCE:

Inheritance simplifies code organization by allowing specialized classes like Manager to inherit common attributes and methods from a parent class like Employee, promoting code reuse. This hierarchical structure facilitates easy maintenance and extension of employee details, enhancing modularity in programming.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 10	Inheritance- Shapes
DATE:	

AIM:

To write a simple Java program to demonstrate inheritance with shapes.

ALGORITHM:

Step 1: Create a class with common attributes and methods shared among different shapes.
Step 2: Define classes that inherit from the Shape class and add specific attributes and methods for each shape.
Step 3: Create objects of both Circle and Rectangle classes, initializing them with shape-specific details.
Step 4: Call methods to display shape details, which utilize inheritance to access common attributes and methods from the parent class.

PROGRAM:

```
class Shape {
    String name;

    public Shape(String name) {
        this.name = name;
    }

    public void displayDetails() {
        System.out.println("Shape: " + name);
    }
}

class Circle extends Shape {
    double radius;

    public Circle(String name, double radius) {
        super(name);
        this.radius = radius;
    }

    public void displayCircleDetails() {
        super.displayDetails();
```

```
        System.out.println("Radius: " + radius);
    }
}

class Rectangle extends Shape {
    double length;
    double width;

    public Rectangle(String name, double length, double width) {
        super(name);
        this.length = length;
        this.width = width;
    }

    public void displayRectangleDetails() {
        super.displayDetails();
        System.out.println("Length: " + length);
        System.out.println("Width: " + width);
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle("Circle", 5.0);
        circle.displayCircleDetails();

        Rectangle rectangle = new Rectangle("Rectangle", 4.0, 3.0);
        rectangle.displayRectangleDetails();
    }
}
```

OUTPUT:

```
Shape: Circle
Radius: 5.0
Shape: Rectangle
Length: 4.0
Width: 3.0
```

INFERENCE:

Inheritance simplifies the management of shapes by allowing specialized classes like Circle and Rectangle to inherit common attributes and methods from a parent class like Shape. This promotes code reuse and organization, facilitating easy maintenance and extension of shape-related functionality.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 11	EXCEPTION HANDLING
DATE:	

AIM: To write a java program divide by zero to cause an ArithmeticException. Use try, catch and finally blocks to throw an exception, catches the exception and provide a way to handle it. Use finally block regardless of whether an exception occurred or not.

ALGORITHM:

Step 1: Import the Scanner class at the beginning of your code to enable user input.
Step 2: Prompt the user to enter a value for d using a Scanner object. Store the user input in the variable d.
Step 3: Wrap the division operation 42 / d inside a try block. If the value of d is zero, it will cause an ArithmeticException to be thrown.
Step 4: Catch the ArithmeticException and handle it appropriately. Print a message indicating the occurrence of the ArithmeticException.
Step 5: Use a finally block to execute cleanup code. Print a message indicating the execution of the finally block. Optionally, you can also print the value of a to see if it was modified in the try block.
Step 6: Print a message indicating that the program has moved beyond the try-catch-finally block.

PROGRAM:

```
class ExcepHandlingbyzero {
public static void main(String args[]) {
	int d, a = 0;
try { // monitor a block of code.
	Scanner s = new Scanner(System.in);
	System.out.println("Enter D:");
	d=s.nextInt();

a = 42 / d;
System.out.println("D is not 0	"+a);
}


catch (ArithmeticException e)
{ // catch divide-by-zero error
	System.out.println(" ArithmeticException.  "+e);
}
finally

{
	System.out.println("finally."+a);
}
System.out.println("After catch and finally statement.");
}
```

```
}
```

OUTPUT:

```
Enter D:
0
 ArithmeticException.  java.lang.ArithmeticException: / by zero
finally.0
After catch and finally statement.
```

INFERENCE:

The program demonstrates best practices for exception handling in Java, including the use of try-catch-finally blocks to handle exceptional conditions, resource management, and providing feedback to the user during program execution.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 12	USER DEFINED EXCEPTION CLASS
DATE:	

AIM: To **write** a simple Java program to create a user defined exception class.

ALGORITHM:

Step 1: **Define Custom Exception Class:**
Create a class MyOwnException that extends Exception.
Define a constructor that accepts a message and passes it to the superclass constructor using super(msg).
Step 2: **Define Method with Exception Handling:**
Create a method employeeAge that takes an integer age as input and throws MyOwnException.
Check if the age is less than 0.
If it is less than 0, throw a MyOwnException with a message "Age can't be less than zero".
If it is non-negative, print "Input is valid!!".
Step 3: **Handle Exception in Main Method:**
In the main method, call the employeeAge method with a negative age (-2) inside a try-catch block.
If a MyOwnException is caught, print the stack trace using e.printStackTrace().
Step 4: **Exception Thrown and Caught:**
When the program is executed with a negative age, it throws a MyOwnException.
The catch block catches the MyOwnException and prints the stack trace of the exception.

PROGRAM:

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

class EmployeeTest
{
	static void employeeAge(int age) throws
	MyOwnException { if(age < 0)
	throw new MyOwnException("Age can't be less than zero");
	else
	System.out.println("Input is valid!!");
	}
	public static void main(String[] args)
	{
	try { employeeAge(-2);
	}
	catch (MyOwnException e)
	{
	e.printStackTrace();
	}
	}
	}
```

OUTPUT:

```
sample.MyOwnException: Age can't be less than zero
	at sample.EmployeeTest.employeeAge(EmployeeTest.java:7)
	at sample.EmployeeTest.main(EmployeeTest.java:13)
```

INFERENCE:

The program demonstrates best practices for exception handling in Java, including the use of custom exception classes, throwing and catching exceptions, and printing stack traces for debugging purposes. These concepts are fundamental for writing robust and reliable Java applications that gracefully handle exceptional conditions.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 13	Reading and Writing Files
DATE:	

AIM: To write a simple Java program to create an input stream to read data from a file.

ALGORITHM:

Step 1: Initialize sourceStream to read from "testfile.txt"
Step 2: Initialize desStream to write to "writefile.txt"
Step 3: Enter try block:Loop until end of source file is reached:
    i. Read a character from source file and store it in temp
    ii. If temp is not -1 (end of file):
        - Write temp to destination file
Step 4: Close sourceStream and desStream in finally block

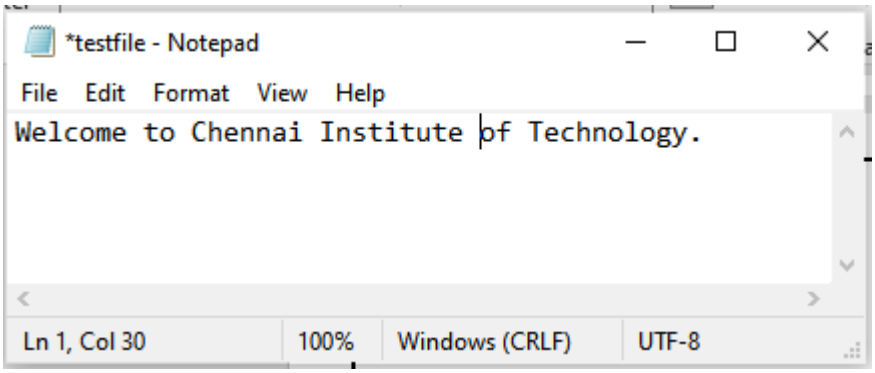
PROGRAM:

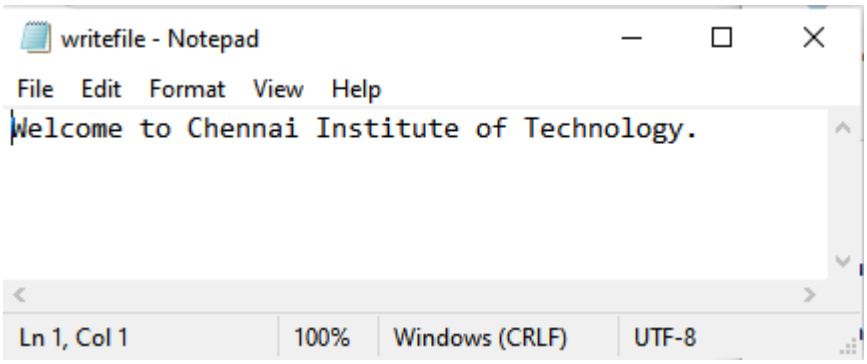
```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileReadWrite {
	public static void main(String[] args) throws IOException {
		FileReader sourceStream = null;
		FileWriter desStream = null;
		try
		{
		sourceStream = new FileReader("E:\\CIT\\Java\\Fileconcept\\testfile.txt");
		desStream = new FileWriter("E:\\CIT\\Java\\Fileconcept\\writefile.txt");
		int temp;
		while((temp = sourceStream.read()) != -1)
		{
			desStream.write((char)temp);
		}
		}
		finally
		{
		//Closing stream as no longer in use
		if(sourceStream != null)
		sourceStream.close();
		desStream.close();

		}
	}
}
```

OUTPUT:





INFERENCE:

The code reads characters from the source file one by one using sourceStream.read() and writes them to the destination file using desStream.write((char)temp). This approach is suitable for small to medium-sized files, but for large files, it may be more efficient to use buffered readers and writers for better performance.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 14	CREATING THREADS
DATE:	

AIM:

To write a java program that creates three threads. First thread displays “Good Morning” every one second, the second thread displays “Hello” every two seconds and the third thread displays “Welcome” every three seconds.

ALGORITHM:

Step 1: Create a class DisplayMessage that extends Thread.
Step 2: Include instance variables to store the message to be displayed and the interval at which it should be displayed.
Step 3: Override the run method to continuously display the message at the specified interval.
Step 4: Create three instances of the DisplayMessage class, each representing a thread.
Step 5: Initialize each instance with a message and the corresponding interval (1 second, 2 seconds, and 3 seconds).
Step 6: Call the start method on each thread instance to start execution.
Step 7: Each thread will run concurrently and continuously display its message at the specified intervals until the program is terminated.

PROGRAM:

```
class DisplayMessage extends Thread {
    private String message;
    private int interval;

    public DisplayMessage(String message, int interval) {
        this.message = message;
        this.interval = interval;
    }

    public void run() {
        try {
            while (true) {
                System.out.println(message);
                Thread.sleep(interval); // Sleep for the specified interval
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + message + " interrupted.");
        }
    }
}

public class CreatingThread {
    public static void main(String[] args) {
        DisplayMessage thread1 = new DisplayMessage("Good Morning", 1000); // 1 second
        DisplayMessage thread2 = new DisplayMessage("Hello", 2000); // 2 seconds
```

```
        DisplayMessage thread3 = new DisplayMessage("Welcome", 3000); // 3 seconds

        // Start the threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

OUTPUT:

```
Good Morning
Hello
Welcome
Good Morning
Hello
Good Morning
Welcome
Good Morning
Hello
Good Morning
Good Morning
Hello
Welcome
Good Morning
Good Morning
Hello
Good Morning
Welcome
Good Morning
Hello
Good Morning
Good Morning
Welcome
Hello
...
```

INFERENCE:

The program properly manages system resources by ensuring that threads are started and stopped appropriately. Resources are released when the program terminates or when threads are interrupted.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 15	MULTI-THREADED APPLICATION - GENERATING A RANDOM INTEGER
DATE:	

AIM: To write a java program that implements a multi-threaded application.

First thread generates a random integer every 1 second and if the value is even, the second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of the cube of the number.

ALGORITHM:

Step 1: Define Thread Classes:
- •Create a class for each thread (RandomNumberGenerator, SquareCalculator, and CubeCalculator).
- •Each thread class extends the Thread class or implements the Runnable interface.
- •Implement the run() method in each thread class to define its behavior.

Step 2: Random Number Generation:
- •In the RandomNumberGenerator thread, generate a random integer between 0 and 99 every 1 second.
- •If the generated number is even, set a flag (isEvenGenerated) to true and notify the SquareCalculator thread.
- •If the generated number is odd, set another flag (isOddGenerated) to true and notify the CubeCalculator thread.

Step 3: Square Calculation:
- •In the SquareCalculator thread, wait for the isEvenGenerated flag to become true.
- •Once the flag becomes true, calculate the square of the generated number and print it.

Step 4: Cube Calculation:
- •In the CubeCalculator thread, wait for the isOddGenerated flag to become true.
- •Once the flag becomes true, calculate the cube of the generated number and print it.

Step :5 Synchronization:
- •Use synchronization to coordinate the execution of threads and prevent race conditions.
- •Synchronize access to shared variables (isEvenGenerated and isOddGenerated) using a common lock object.

Step 6: Main Class:
- •Create instances of the three thread classes (RandomNumberGenerator, SquareCalculator, and CubeCalculator).
- •Start each thread using the start() method.

PROGRAM:

```
package PGM;


import java.util.Random;

class RandomNumberGenerator extends Thread {
    public void run() {
```

```
        Random random = new Random();
        try {
            while (true) {
                int randomNumber = random.nextInt(100); // Generate a random integer
between 0 and 99
                System.out.println("Generated number: " + randomNumber);
                if (randomNumber % 2 == 0) {
                    synchronized (Main.lock) {
                        Main.number = randomNumber;
                        Main.isEvenGenerated = true;
                        Main.lock.notifyAll(); // Notify waiting threads
                    }
                } else {
                    synchronized (Main.lock) {
                        Main.number = randomNumber;
                        Main.isOddGenerated = true;
                        Main.lock.notifyAll(); // Notify waiting threads
                    }
                }
                Thread.sleep(1000); // Sleep for 1 second
            }
        } catch (InterruptedException e) {
            System.out.println("Random number generator thread interrupted.");
        }
    }
}

class SquareCalculator extends Thread {
    public void run() {
        try {
            while (true) {
                synchronized (Main.lock) {
                    while (!Main.isEvenGenerated) {
                        Main.lock.wait(); // Wait for even number to be generated
                    }
                    int square = Main.number * Main.number;
                    System.out.println("Square of " + Main.number + " is: " + square);
                    Main.isEvenGenerated = false; // Reset flag
                }
            }
        } catch (InterruptedException e) {
            System.out.println("Square calculator thread interrupted.");
        }
    }
}

class CubeCalculator extends Thread {
    public void run() {
        try {
            while (true) {
                synchronized (Main.lock) {
                    while (!Main.isOddGenerated) {
                        Main.lock.wait(); // Wait for odd number to be generated
                    }
                    int cube = Main.number * Main.number * Main.number;
                    System.out.println("Cube of " + Main.number + " is: " + cube);
```

```
                    Main.isOddGenerated = false; // Reset flag
                }
            }
        } catch (InterruptedException e) {
            System.out.println("Cube calculator thread interrupted.");
        }
    }
}

public class Main {
    public static final Object lock = new Object();
    public static int number;
    public static boolean isEvenGenerated = false;
    public static boolean isOddGenerated = false;

    public static void main(String[] args) {
        RandomNumberGenerator randomNumberGenerator = new RandomNumberGenerator();
        SquareCalculator squareCalculator = new SquareCalculator();
        CubeCalculator cubeCalculator = new CubeCalculator();

        randomNumberGenerator.start();
        squareCalculator.start();
        cubeCalculator.start();
    }
}
```

OUTPUT:

```
Generated number: 84
Square of 84 is: 7056
Generated number: 69
Cube of 69 is: 328509
Generated number: 62
Square of 62 is: 3844
Generated number: 8
Square of 8 is: 64
Generated number: 58
Square of 58 is: 3364
...
```

INFERENCE:

Inter-Thread Communication:
The threads communicate with each other through shared variables (isEvenGenerated and isOddGenerated) and a common lock object.
The RandomNumberGenerator thread notifies the SquareCalculator or CubeCalculator thread when an even or odd number is generated, respectively.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 16	MAPPING THE WORDS OF THE NUMBERS USING MAP
DATE:	

AIM: To create a java program for getting a list of numbers, print the list using for loop and map the words of the numbers using Map interface.

ALGORITHM:

Step 1: Define Object Class:

- Create a class (e.g., **Student**) to represent the objects in the list.
- Include properties (e.g., **id** and **name**) and a constructor to initialize them.
- Include getter methods to access the properties.

Step 2: Create List of Objects:

- Instantiate a list (e.g., **ArrayList<Student>**) to store objects of the defined class.
- Add objects to the list, providing values for their properties.

Step 3: Create Map:

- Instantiate a map (e.g., **HashMap<Integer, String>**) to store converted values.
- Define the types for keys and values based on the properties of the object class.

Step 4: Convert List to Map:

- Iterate over each object in the list.
- Extract the key-value pairs from the object's properties.
- Add the key-value pairs to the map using the **put** method.

Step 5: Print Map:

- After converting the list to a map, print the map to verify the conversion.

PROGRAM:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.HashMap;

// create a list
class Student {

	// id will act as Key
	private Integer id;

	// name will act as value
	private String name;

	// create curstuctor for reference
	public Student(Integer id, String name)
	{

		// assign the value of id and name
		this.id = id;
		this.name = name;
	}
```

```
        // return private variable id
        public Integer getId()
        {
                return id;
        }

        // return private variable name
        public String getName()
        {
                return name;
        }
}
// main class and method
public class ListEx2 {
        // main Driver
        public static void main(String[] args)
        {
                // create a list
                List<Student> lt = new ArrayList<Student>();

                // add the member of list
                lt.add(new Student(1, "One"));
                lt.add(new Student(2, "Two"));
                lt.add(new Student(3, "Three"));

                // create map with the help of
                // Object (stu) method
                // create object of Map class
                Map<Integer, String> map = new HashMap();

                // put every value list to Map
                for (Student stu : lt) {
                        map.put(stu.getId(), stu.getName());
                }

                // print map
                System.out.println("Map : " + map);
        }
}
```

OUTPUT:

```
Map : {1=One, 2=Two, 3=Three}
```

INFERENCE:

The program demonstrates fundamental concepts of object-oriented programming, collection handling, iterative processing, and data mapping in Java. It showcases how to efficiently convert data from one data structure (list) to another (map) while preserving the integrity of the data.

RESULT:

The program was successfully executed and result is obtained

EX.NO: 17	CONNECT WITH A DATABASE USING JDBC
DATE:	

AIM: To demonstrate the use of JDBC to connect to a database, execute queries, retrieve data, and close resources properly.

ALGORITHM:

Step 1: Load and Register JDBC Driver:

- Use **Class.forName()** to load and register the JDBC driver. For example, for MySQL, you can use **Class.forName("com.mysql.jdbc.Driver")**.

Step 2: Establish Database Connection:

- Use **DriverManager.getConnection()** to establish a connection to the database. Provide the JDBC URL, username, and password as parameters.

Step 3: Create Statement:

- Create a **Statement** object using the **createStatement()** method of the **Connection** object.

Step 4: Execute Query:

- Use the **executeQuery()** method of the **Statement** object to execute a SQL query. Pass the SQL query as a parameter.

Step 5: Process Result Set:

- Use a **while** loop to iterate over the **ResultSet** returned by the query execution.
- Inside the loop, use methods like **getInt()**, **getString()**, etc., to retrieve data from each row of the result set.

Step 6: Close Resources:

- Close the **ResultSet**, **Statement**, and **Connection** objects in the **finally** block to release database resources.

PROGRAM:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcConnect {

    // Database connection details
    private static final String JDBC_URL = "jdbc:mysql://localhost:3306/mydatabase";
    private static final String USERNAME = "username";
    private static final String PASSWORD = "password";

    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
```

```
        try {
            // Load and register the JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            // Establish the connection
            connection = DriverManager.getConnection(JDBC_URL, USERNAME, PASSWORD);

            // Create a statement
            statement = connection.createStatement();

            // Execute a query
            String query = "SELECT * FROM my_table";
            resultSet = statement.executeQuery(query);

            // Process the result set
            while (resultSet.next()) {
                // Retrieve data from each row
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");

                // Process retrieved data (print here)
                System.out.println("ID: " + id + ", Name: " + name);
            }
        } catch (ClassNotFoundException e) {
            System.err.println("Error loading JDBC driver: " + e.getMessage());
        } catch (SQLException e) {
            System.err.println("Error executing SQL query: " + e.getMessage());
        } finally {
            // Close resources
            try {
                if (resultSet != null) {
                    resultSet.close();
                }
                if (statement != null) {
                    statement.close();
                }
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.err.println("Error closing resources: " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

ID: 1, Name: John
ID: 2, Name: Alice
ID: 3, Name: Bob

INFERENCE:

This program loads and registers the JDBC driver, establishes a connection to the database, creates a statement, executes a query, processes the result set to retrieve data, and finally closes all resources properly to avoid memory leaks.

RESULT:

The program was successfully executed and result is obtained.