HE FOLOWING POINTS FOR ADVANCED JAVA PRACTICAL EXAM:

1. ***BELOW GIVEN ARE A FEW SAMPLE QUESTIONS. PLEASE BE INFORMED THAT SIMILAR QUESTIONS CAN BE EXPECTED.***

2. ***ALL PROGRAMS DONE IN LAB EXPERIMENTS CAN BE EXPECTED.***

3. ***THERE WILL BE ONLY 1 PROGRAM FOR EXECUTION (1HOUR-15MARKS) and ORAL EXAM BASED ON ENTIRE SYLLABUS FOR 10 MARKS. SINCE THE ORAL EXAM WILL BE CONDUCTED BY THE EXTERNAL EXAMINER KINDLY DO NOT EXPECT INTERNAL FACULTIES TO GIVE YOU SAMPLE QUESTIONS FOR VIVA.***

General questions for GUI PROGRAMMING:

For any application design a GUI with 1 frame in netbeans. Accept the values from the user. Perform the given operation and store the values in database.

Eg:

1. Create a GUI application with two text boxes labeled "Celsius" and "Fahrenheit". Depending upon what value is entered in the 1st textfield the application should convert the value in farhenite and display in the 2nd textfield. Also, store original and converted values in database.

   Code:

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

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TemperatureConverterApp extends Application {

   private TextField celsiusTextField;
   private TextField fahrenheitTextField;

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

   @Override
   public void start(Stage primaryStage) {
      primaryStage.setTitle("Temperature Converter");

      Label celsiusLabel = new Label("Celsius:");
      celsiusTextField = new TextField();

      Label fahrenheitLabel = new Label("Fahrenheit:");
      fahrenheitTextField = new TextField();

      Button convertButton = new Button("Convert");
      convertButton.setOnAction(e -> convertAndStore());

      VBox layout = new VBox(10);
      layout.getChildren().addAll(celsiusLabel,          celsiusTextField,          fahrenheitLabel,
fahrenheitTextField, convertButton);

      Scene scene = new Scene(layout, 250, 150);
      primaryStage.setScene(scene);
```

```
        primaryStage.show();
    }

    private void convertAndStore() {
        try {
            double celsiusValue = Double.parseDouble(celsiusTextField.getText());
            double fahrenheitValue = (celsiusValue * 9 / 5) + 32;

            fahrenheitTextField.setText(String.valueOf(fahrenheitValue));

            // Store values in the database
            storeInDatabase(celsiusValue, fahrenheitValue);
        } catch (NumberFormatException e) {
            fahrenheitTextField.setText("Invalid input");
        }
    }

    private void storeInDatabase(double celsius, double fahrenheit) {
        String url = "jdbc:sqlite:temperature.db";

        try (Connection connection = DriverManager.getConnection(url);
             PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO
temperature (celsius, fahrenheit) VALUES (?, ?)")) {

            preparedStatement.setDouble(1, celsius);
            preparedStatement.setDouble(2, fahrenheit);

            preparedStatement.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

2. Build a Java GUI form for a flight booking system. Use a combo box to allow users to select their preferred destination. Fetch the available flights based on the selected destination from the database and show it to the user.

   Code:

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class FlightBookingApp extends Application {

    private ComboBox<String> destinationComboBox;
    private ListView<String> flightListView;

    private static final String DB_URL = "jdbc:sqlite:flights.db";

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

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Flight Booking System");

        destinationComboBox = new ComboBox<>();
        flightListView = new ListView<>();

        Label destinationLabel = new Label("Select Destination:");
        Label flightsLabel = new Label("Available Flights:");

        // Fetch destinations from the database
        fetchDestinations();

        destinationComboBox.setOnAction(e -> fetchFlights());

        VBox layout = new VBox(10);
        layout.getChildren().addAll(destinationLabel,        destinationComboBox,        flightsLabel,
flightListView);

        Scene scene = new Scene(layout, 300, 250);
        primaryStage.setScene(scene);

        primaryStage.show();
    }

    private void fetchDestinations() {
        try (Connection connection = DriverManager.getConnection(DB_URL);
             PreparedStatement    preparedStatement    =    connection.prepareStatement("SELECT
DISTINCT destination FROM flights");
             ResultSet resultSet = preparedStatement.executeQuery()) {

            ObservableList<String> destinations = FXCollections.observableArrayList();

            while (resultSet.next()) {
                destinations.add(resultSet.getString("destination"));
                }

                destinationComboBox.setItems(destinations);

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        private void fetchFlights() {
            String selectedDestination = destinationComboBox.getValue();

            if (selectedDestination != null) {
                try (Connection connection = DriverManager.getConnection(DB_URL);
                     PreparedStatement preparedStatement = connection.prepareStatement("SELECT
    flight_number FROM flights WHERE destination = ?");
                ) {
                    preparedStatement.setString(1, selectedDestination);

                    ResultSet resultSet = preparedStatement.executeQuery();

                    ObservableList<String> flights = FXCollections.observableArrayList();

                    while (resultSet.next()) {
                        flights.add(resultSet.getString("flight_number"));
                    }

                    flightListView.setItems(flights);

                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

QUESTIONS BASED ON DIFFERENT APPLICATIONS WITH DIFFERENT GUI COMPONENTS (radiobutton, combobox, checkbox, dropdown, passwordfield etc) CAN BE EXPECTED HERE.

Collections, GENERICS, REFLECTIONS, STREAMS :

Stacks

1. Implement a Java program to evaluate a postfix expression using a stack. The expression will be given as a string, and the program should return the result of the expression.

Save file as: PostfixEvaluator.java
Code:

```
import java.util.Stack;

public class PostfixEvaluator {

    public static int evaluatePostfix(String postfixExpression) {
        Stack<Integer> operandStack = new Stack<>();

        for (char c : postfixExpression.toCharArray()) {
            if (Character.isDigit(c)) {
                // Operand: Push onto the stack
                operandStack.push(Character.getNumericValue(c));
            } else if (isOperator(c)) {
                // Operator: Pop operands, perform operation, and push result back onto the stack
                int operand2 = operandStack.pop();
                int operand1 = operandStack.pop();
                int result = performOperation(operand1, operand2, c);
                operandStack.push(result);
            }
            // Ignore other characters (e.g., spaces)
        }

        // The final result should be at the top of the stack
        return operandStack.pop();
    }

    private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static int performOperation(int operand1, int operand2, char operator) {
        switch (operator) {
            case '+':
                return operand1 + operand2;
            case '-':
                return operand1 - operand2;
            case '*':
                return operand1 * operand2;
            case '/':
                if (operand2 == 0) {
                    throw new ArithmeticException("Division by zero");
                }
                return operand1 / operand2;
            default:
                throw new IllegalArgumentException("Invalid operator: " + operator);
        }
    }

    public static void main(String[] args) {
        // Example usage:
```

```
        String postfixExpression = "23*5+";
        int result = evaluatePostfix(postfixExpression);

        System.out.println("Postfix Expression: " + postfixExpression);
        System.out.println("Result: " + result);
    }
}
```

2. Develop a Java program to convert an infix expression (with parentheses and operators) to a postfix expression using a stack.

   Save file as: InfixToPostfixConverter.java
   Code:

```
import java.util.Stack;

public class InfixToPostfixConverter {

    public static String infixToPostfix(String infixExpression) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> operatorStack = new Stack<>();

        for (char c : infixExpression.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                postfix.append(c);
            } else if (c == '(') {
                operatorStack.push(c);
            } else if (c == ')') {
                while (!operatorStack.isEmpty() && operatorStack.peek() != '(') {
                    postfix.append(operatorStack.pop());
                }
                if (!operatorStack.isEmpty() && operatorStack.peek() == '(') {
                    operatorStack.pop(); // Pop the '('
                }
            } else {
                while          (!operatorStack.isEmpty()          &&          precedence(c)          <=
precedence(operatorStack.peek())) {
                    postfix.append(operatorStack.pop());
                }
                operatorStack.push(c);
            }
        }

        while (!operatorStack.isEmpty()) {
            postfix.append(operatorStack.pop());
        }

        return postfix.toString();
    }

    private static int precedence(char operator) {
        switch (operator) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
            default:
                return -1;
        }
    }

    public static void main(String[] args) {
        // Example usage:
        String infixExpression = "a+b*(c^d-e)^(f+g*h)-i";
        String postfixExpression = infixToPostfix(infixExpression);

        System.out.println("Infix Expression: " + infixExpression);
        System.out.println("Postfix Expression: " + postfixExpression);
    }
}
```

3. Create a Java program to sort a stack in ascending order. You can only use one additional stack to assist in sorting, and the original stack should be modified to be in sorted order.

Save file as: SortStack.java
Code:

```
import java.util.Stack;

public class SortStack {

    public static void sortStack(Stack<Integer> stack) {
        if (stack == null || stack.isEmpty()) {
            return;
        }

        Stack<Integer> auxStack = new Stack<>();

        while (!stack.isEmpty()) {
            int temp = stack.pop();

            // Move elements from the auxStack to the stack until the correct position for temp is found
            while (!auxStack.isEmpty() && auxStack.peek() > temp) {
                stack.push(auxStack.pop());
            }

            // Place temp in the correct position
            auxStack.push(temp);
        }
        // Copy elements back to the original stack
        while (!auxStack.isEmpty()) {
            stack.push(auxStack.pop());
        }
    }

    public static void main(String[] args) {
        // Example usage:
        Stack<Integer> stack = new Stack<>();
        stack.push(5);
        stack.push(2);
        stack.push(8);
        stack.push(1);
        stack.push(3);

        System.out.println("Original Stack: " + stack);
        sortStack(stack);
        System.out.println("Sorted Stack: " + stack);
    }
}
```

4. Given an array of integers, write a Java program to find the next greater element for each element in the array. Implement this using a stack to efficiently find the next greater element.

   Save file as: NextGreaterElement.java
   Code:

```
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;

public class NextGreaterElement {

    public static int[] findNextGreaterElements(int[] nums) {
        if (nums == null || nums.length == 0) {
            return new int[0];
        }

        int[] result = new int[nums.length];
        Deque<Integer> stack = new ArrayDeque<>();
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            // Check for elements with no next greater element
            while (!stack.isEmpty() && nums[i] > stack.peek()) {
                nextGreaterMap.put(stack.pop(), nums[i]);
            }

            stack.push(nums[i]);
        }
        // For elements with no next greater element, mark as -1
        while (!stack.isEmpty()) {
            nextGreaterMap.put(stack.pop(), -1);
        }

        for (int i = 0; i < nums.length; i++) {
            result[i] = nextGreaterMap.get(nums[i]);
        }

        return result;
    }

    public static void main(String[] args) {
        // Example usage:
        int[] nums = {4, 2, 7, 3, 1, 9, 8};
        int[] result = findNextGreaterElements(nums);

        System.out.println("Next Greater Elements:");
        for (int i = 0; i < nums.length; i++) {
            System.out.println(nums[i] + " --> " + result[i]);
        }
    }
}
```

Queue

5. Write a Java program to store the first N binary numbers using a queue.
   Save code as: BinaryNumbersGenerator.java
   Code:

```
import java.util.LinkedList;
import java.util.Queue;

public class BinaryNumbersGenerator {

    public static void generateBinaryNumbers(int n) {
        if (n <= 0) {
            System.out.println("Invalid input");
            return;
        }

        Queue<String> queue = new LinkedList<>();
        queue.offer("1"); // Start with the first binary number

        for (int i = 0; i < n; i++) {
            String current = queue.poll();
            System.out.println(current);

            // Enqueue the next binary numbers by appending '0' and '1'
            queue.offer(current + "0");
            queue.offer(current + "1");
        }
    }

    public static void main(String[] args) {
        // Example usage:
        int N = 5;
        System.out.println("First " + N + " binary numbers are:");
        generateBinaryNumbers(N);
    }
}
```

6. Given an array and an integer k, find the maximum element for each and every contiguous subarray of size k. Implement this using a queue.

   Save file as: MaximumInSubarrays.java
   Code:

```
import java.util.ArrayDeque;
import java.util.Deque;

public class MaximumInSubarrays {

    public static void printMaxInSubarrays(int[] nums, int k) {
        if (nums == null || nums.length == 0 || k <= 0 || k > nums.length) {
            System.out.println("Invalid input");
            return;
        }

        Deque<Integer> deque = new ArrayDeque<>();

        // Process the first k elements separately
        for (int i = 0; i < k; i++) {
            while (!deque.isEmpty() && nums[i] >= nums[deque.peekLast()]) {
                deque.pollLast();
            }
            deque.offerLast(i);
        }

        // Process the remaining elements
        for (int i = k; i < nums.length; i++) {
            System.out.print(nums[deque.peekFirst()] + " ");

            // Remove elements that are out of the current window
            while (!deque.isEmpty() && deque.peekFirst() <= i - k) {
                deque.pollFirst();
            }

            // Remove smaller elements from the back of the deque
            while (!deque.isEmpty() && nums[i] >= nums[deque.peekLast()]) {
                deque.pollLast();
            }
            deque.offerLast(i);
        }

        // Print the maximum for the last subarray
        System.out.println(nums[deque.peekFirst()]);
    }

    public static void main(String[] args) {
        // Example usage:
        int[] nums = {1, 3, -1, -3, 5, 3, 6, 7};
        int k = 3;

        System.out.println("Maximum elements in each subarray of size " + k + ":");
        printMaxInSubarrays(nums, k);
    }
}
```

7. Simulate the hot potato game using a queue in Java. Given a queue of names and a number, pass the "potato" around, and remove the person holding the "potato" after the specified number of passes.

   Save file as: HotPotatoGame.java

   Code:

```
import java.util.LinkedList;
import java.util.Queue;

public class HotPotatoGame {

    public static String hotPotato(Queue<String> players, int passes) {
        while (players.size() > 1) {
            for (int i = 0; i < passes - 1; i++) {
                // Pass the "potato" by dequeuing and enqueuing
                String currentPlayer = players.poll();
                players.offer(currentPlayer);
            }

            // Remove the person holding the "potato"
            System.out.println(players.poll() + " is eliminated!");
        }

        // Return the last remaining player (the winner)
        return players.poll();
    }

    public static void main(String[] args) {
        // Example usage:
        Queue<String> players = new LinkedList<>();
        players.offer("Alice");
        players.offer("Bob");
        players.offer("Charlie");
        players.offer("David");
        players.offer("Eva");

        int passes = 3;

        System.out.println("Starting Hot Potato Game with players: " + players);
        String winner = hotPotato(players, passes);
        System.out.println("The winner is: " + winner);
    }
}
```

8. Write a Java program to generate all possible numbers with a given set of digits using a queue.

Save file as: GenerateNumbersWithDigits.java

Code:

```
import java.util.LinkedList;
import java.util.Queue;

public class GenerateNumbersWithDigits {

    public static void generateNumbers(int[] digits, int n) {
        if (digits == null || digits.length == 0 || n <= 0) {
            System.out.println("Invalid input");
            return;
        }

        Queue<String> queue = new LinkedList<>();

        // Enqueue each digit as a starting point
        for (int digit : digits) {
            queue.offer(String.valueOf(digit));
        }

        // Perform BFS traversal to generate all possible numbers
        while (!queue.isEmpty() && n > 0) {
            int size = queue.size();

            for (int i = 0; i < size; i++) {
                String current = queue.poll();

                // Print the current number
                System.out.print(current + " ");

                // Enqueue the next possible numbers by appending each digit
                for (int digit : digits) {
                    queue.offer(current + digit);
                }
            }

            n--;
        }

        System.out.println();
    }

    public static void main(String[] args) {
        // Example usage:
        int[] digits = {1, 2, 3};
        int n = 5;

        System.out.println("All possible numbers with digits " + arrayToString(digits) +
                " up to length " + n + ":");
        generateNumbers(digits, n);
    }

    // Utility method to convert an array to a string
    private static String arrayToString(int[] array) {
        StringBuilder sb = new StringBuilder("[");
        for (int i = 0; i < array.length; i++) {
            sb.append(array[i]);
            if (i < array.length - 1) {
                sb.append(", ");
            }
        }
        sb.append("]");
        return sb.toString();
    }
}
```

HASHSET

9. Given an unsorted array of integers, write a Java program to find the length of the longest consecutive elements sequence using a HashSet.
   Save file as: LongestConsecutiveSequence.java
   Code:

```
import java.util.HashSet;

public class LongestConsecutiveSequence {

    public static int longestConsecutive(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        HashSet<Integer> numSet = new HashSet<>();
        for (int num : nums) {
            numSet.add(num);
        }

        int longestStreak = 0;
        for (int num : numSet) {
            // If the current number is the start of a sequence, find the length of the consecutive sequence
            if (!numSet.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;

                while (numSet.contains(currentNum + 1)) {
                    currentNum++;
                    currentStreak++;
                }

                // Update the longest streak if the current streak is longer
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }

    public static void main(String[] args) {
        // Example usage:
        int[] nums = {100, 4, 200, 1, 3, 2};
        System.out.println("Length of the longest consecutive sequence: " +
longestConsecutive(nums)); // Output: 4
    }
}
```

10. Given two strings, determine if they are isomorphic. Two strings are isomorphic if the characters in one string can be mapped to the characters in another string using a one-to-one mapping. Use a HashSet for efficient character mapping.

Save file as: IsomorphicStrings.java
Code:

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class IsomorphicStrings {

    public static boolean areIsomorphic(String s, String t) {
        if (s.length() != t.length()) {
            return false;
        }

        Map<Character, Character> charMapping = new HashMap<>();
        Set<Character> mappedChars = new HashSet<>();

        for (int i = 0; i < s.length(); i++) {
            char charS = s.charAt(i);
            char charT = t.charAt(i);

            // If charS is already mapped, check if the mapping is correct
            if (charMapping.containsKey(charS)) {
                if (charMapping.get(charS) != charT) {
                    return false; // Incorrect mapping
                }
            } else {
                // If charS is not mapped, check if charT is already mapped to another character
                if (mappedChars.contains(charT)) {
                    return false; // charT is already used for another character
                }

                // Map charS to charT
                charMapping.put(charS, charT);
                mappedChars.add(charT);
            }
        }

        return true;
    }

    public static void main(String[] args) {
        // Example usage:
        String s1 = "egg";
        String t1 = "add";
        System.out.println(s1 + " and " + t1 + " are isomorphic: " + areIsomorphic(s1, t1)); //
Output: true

        String s2 = "foo";
        String t2 = "bar";
        System.out.println(s2 + " and " + t2 + " are isomorphic: " + areIsomorphic(s2, t2)); //
Output: false

        String s3 = "paper";
        String t3 = "title";
        System.out.println(s3 + " and " + t3 + " are isomorphic: " + areIsomorphic(s3, t3)); //
Output: true
    }
}
```

11. Write a Java program to determine if a given number is a happy number. A number is a happy number if, after repeatedly replacing it with the sum of the square of its digits, it eventually reaches 1. Use a HashSet to detect cycles. []

    Save file as: HappyNumberChecker.java
    Code:

```
import java.util.HashSet;

public class HappyNumberChecker {
    public static boolean isHappy(int n) {
        HashSet<Integer> seenNumbers = new HashSet<>();

        while (n != 1 && !seenNumbers.contains(n)) {
            seenNumbers.add(n);
            n = getNextSumOfSquares(n);
        }

        return n == 1;
    }

    private static int getNextSumOfSquares(int n) {
        int sum = 0;
        while (n > 0) {
            int digit = n % 10;
            sum += digit * digit;
            n /= 10;
        }
        return sum;
    }

    public static void main(String[] args) {
        // Example usage:
        int number1 = 19;
        System.out.println(number1 + " is a happy number: " + isHappy(number1)); // Output: true

        int number2 = 22;
        System.out.println(number2 + " is a happy number: " + isHappy(number2)); // Output: false
    }
}
```

12. Given an array of strings, write a Java program to group the anagrams together using a HashSet.

Save file as: GroupAnagrams.java
Code:

```
import java.util.*;

public class GroupAnagrams {

    public static List<List<String>> groupAnagrams(String[] strs) {
        if (strs == null || strs.length == 0) {
            return new ArrayList<>();
        }

        // Create a map to store anagrams
        Map<String, List<String>> anagramMap = new HashMap<>();

        for (String str : strs) {
            // Sort each string to identify anagrams
            char[] charArray = str.toCharArray();
            Arrays.sort(charArray);
            String sortedStr = new String(charArray);

            // Check if the sorted string is already in the map
            // If yes, add the original string to the corresponding list
            // If no, create a new list and add the original string to it
            anagramMap.computeIfAbsent(sortedStr, k -> new ArrayList<>()).add(str);
        }

        // Convert map values to a list of lists
        return new ArrayList<>(anagramMap.values());
    }

    public static void main(String[] args) {
        // Example usage:
        String[] strings = {"eat", "tea", "tan", "ate", "nat", "bat"};
        List<List<String>> result = groupAnagrams(strings);

        // Display the grouped anagrams
        for (List<String> group : result) {
            System.out.println(group);
        }
    }
}
```

13. Given an array of integers and an integer k, write a Java program to check if there are any duplicate elements within k distance using a HashSet.
    Save file as: DuplicateWithinKDistance.java
    Code:

```
import java.util.HashSet;

public class DuplicateWithinKDistance {

    public static boolean containsNearbyDuplicate(int[] nums, int k) {
        if (nums == null || nums.length == 0 || k <= 0) {
            return false; // Invalid input
        }

        HashSet<Integer> set = new HashSet<>();

        for (int i = 0; i < nums.length; i++) {
            if (i > k) {
                set.remove(nums[i - k - 1]); // Remove element that is outside the window
            }

            if (!set.add(nums[i])) {
                return true; // Found a duplicate within distance k
            }
        }
        return false;
    }

    public static void main(String[] args) {
        // Example usage:
        int[] nums1 = {1, 2, 3, 1};
        int k1 = 3;
        System.out.println("Array contains duplicate within distance " + k1 + ": " +
                containsNearbyDuplicate(nums1, k1)); // Output: true

        int[] nums2 = {1, 0, 1, 1};
        int k2 = 1;
        System.out.println("Array contains duplicate within distance " + k2 + ": " +
                containsNearbyDuplicate(nums2, k2)); // Output: true

        int[] nums3 = {1, 2, 3, 1, 2, 3};
        int k3 = 2;
        System.out.println("Array contains duplicate within distance " + k3 + ": " +
                containsNearbyDuplicate(nums3, k3)); // Output: false
    }
}
```

14. Find the Single Number: Given an array of integers where every element appears twice except for one, write a Java program to find the single non-duplicated number using a HashSet.

Save file as: SingleNumberFinder.java
Code:

```
import java.util.HashSet;

public class SingleNumberFinder {

    public static int findSingleNumber(int[] nums) {
        HashSet<Integer> set = new HashSet<>();

        for (int num : nums) {
            // If the number is not in the set, add it; if it's already in the set, remove it
            if (!set.add(num)) {
                set.remove(num);
            }
        }

        // At this point, the set should contain only the single non-duplicated number
        // If the array is valid, the set should have only one element
        if (set.size() == 1) {
            return set.iterator().next();
        } else {
            throw new IllegalArgumentException("Invalid input array");
        }
    }
    public static void main(String[] args) {
        // Example usage:
        int[] nums = {4, 2, 1, 2, 1};

        try {
            int result = findSingleNumber(nums);
            System.out.println("The single non-duplicated number is: " + result);
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

15. Valid Sudoku: Write a Java program to determine if a 9x9 Sudoku board is valid. The board is valid if each row, each column, and each of the 9 3x3 sub-grids that compose the board contains all of the digits from 1 to 9 without repetition. Use HashSet to check for duplicates.

    Save file as: ValidSudoku.java
    Code:

```
import java.util.HashSet;


public class ValidSudoku {

    public boolean isValidSudoku(char[][] board) {

        for (int i = 0; i < 9; i++) {

            HashSet<Character> rowSet = new HashSet<>();

            HashSet<Character> colSet = new HashSet<>();

            HashSet<Character> gridSet = new HashSet<>();


            for (int j = 0; j < 9; j++) {

                // Check rows

                if (board[i][j] != '.' && !rowSet.add(board[i][j]))

                    return false;


                // Check columns

                if (board[j][i] != '.' && !colSet.add(board[j][i]))

                    return false;


                // Check sub-grids
            int rowIndex = 3 * (i / 3);
            int colIndex = 3 * (i % 3);
            if (board[rowIndex + j / 3][colIndex + j % 3] != '.' &&
                    !gridSet.add(board[rowIndex + j / 3][colIndex + j % 3]))
                return false;
        }
    }
    return true;
}

public static void main(String[] args) {
    char[][] sudokuBoard = {
        {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
        {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
        {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
        {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
        {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
        {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
        {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
        {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
        {'.', '.', '.', '.', '8', '.', '.', '7', '9'}
    };

    ValidSudoku validator = new ValidSudoku();
    boolean isValid = validator.isValidSudoku(sudokuBoard);

    if (isValid) {
        System.out.println("The Sudoku board is valid.");
    } else {
        System.out.println("The Sudoku board is invalid.");
    }
}
```

}

16. You are given strings J representing the types of stones that are jewels, and S representing the stones you have. Each character in S is a type of stone you have. Write a Java program to determine how many of the stones you have are also jewels using a HashSet.

Save file as: JewelAndStoneCounter.java
Code:

```java
import java.util.HashSet;

public class JewelAndStoneCounter {
    public static int countJewelsInStones(String J, String S) {
        HashSet<Character> jewelSet = new HashSet<>();

        // Add each jewel type to the HashSet
        for (char jewel : J.toCharArray()) {
            jewelSet.add(jewel);
        }

        int jewelCount = 0;

        // Check each stone and increment count if it's a jewel
        for (char stone : S.toCharArray()) {
            if (jewelSet.contains(stone)) {
                jewelCount++;
            }
        }

        return jewelCount;
    }

    public static void main(String[] args) {
        // Example usage:
        String jewels = "aA";
        String stones = "aAAbbbb";

        int result = countJewelsInStones(jewels, stones);

        System.out.println("Number of jewels in stones: " + result);
    }
}
```

17.Create a class representing an employee with attributes like name, age, and department.

Use an Array List to store a collection of employees.

Implement methods to add new employees, remove employees by name, and display the list of employees in a specific department.

Save file as : EmployeeManagementExample.java

Code:

```
import java.util.ArrayList;

import java.util.Iterator;

import java.util.Scanner;


class Employee {

    private String name;

    private int age;

    private String department;


    public Employee(String name, int age, String department) {

        this.name = name;

        this.age = age;

        this.department = department;

    }


    public String getName() {

        return name;

    }


    public int getAge() {

        return age;

    }


    public String getDepartment() {

        return department;

    }
    @Override
    public String toString() {
        return "Employee{name='" + name + "', age=" + age + ", department='" + department + "'}";
    }
}

class EmployeeManager {
    private ArrayList<Employee> employeeList;

    public EmployeeManager() {
        employeeList = new ArrayList<>();
    }

    public void addEmployee(Employee employee) {
        employeeList.add(employee);
    }

    public void removeEmployeeByName(String name) {
        Iterator<Employee> iterator = employeeList.iterator();
        while (iterator.hasNext()) {
            Employee employee = iterator.next();
            if (employee.getName().equals(name)) {
                iterator.remove();
            }
        }
    }

    public void displayEmployeesInDepartment(String department) {
        System.out.println("Employees in department '" + department + "':");
        for (Employee employee : employeeList) {
            if (employee.getDepartment().equals(department)) {
                System.out.println(employee);
            }
        }
    }

    public void displayAllEmployees() {
        System.out.println("All Employees:");
        for (Employee employee : employeeList) {
            System.out.println(employee);
        }
    }
}

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

        int choice;
        do {
            System.out.println("Menu:");
            System.out.println("1. Add Employee");
            System.out.println("2. Remove Employee by Name");
            System.out.println("3. Display Employees in a Department");
            System.out.println("4. Display All Employees");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character
```

```
switch (choice) {
  case 1:
    System.out.print("Enter employee name: ");
    String name = scanner.nextLine();
    System.out.print("Enter employee age: ");
    int age = scanner.nextInt();
    scanner.nextLine(); // Consume the newline character
    System.out.print("Enter employee department: ");
    String department = scanner.nextLine();

    Employee newEmployee = new Employee(name, age, department);
    employeeManager.addEmployee(newEmployee);
    System.out.println("Employee added successfully!\n");
    break;

  case 2:
    System.out.print("Enter the name of the employee to remove: ");
    String employeeNameToRemove = scanner.nextLine();
    employeeManager.removeEmployeeByName(employeeNameToRemove);
    System.out.println("Employee removed successfully!\n");
    break;

  case 3:
    System.out.print("Enter the department to display employees: ");
    String departmentToDisplay = scanner.nextLine();
    employeeManager.displayEmployeesInDepartment(departmentToDisplay);
    System.out.println();
    break;

  case 4:
    employeeManager.displayAllEmployees();
```

```
                System.out.println();
                break;

              case 5:
                System.out.println("Exiting the program. Goodbye!");
                break;

              default:
                System.out.println("Invalid choice. Please enter a valid option.\n");
            }
        } while (choice != 5);

        scanner.close();
    }

}
```

18.Simulate a scenario where a set of sensors continuously collect data.

Use a HashMap to store sensor data, where the sensor ID is the key, and the data is the value.

Implement a method to display the average data value for all sensors.

Code: save file as SensorSimulation.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

class SensorDataCollection {
    private Map<String, Double> sensorData;
```

```
public SensorDataCollection() {
    sensorData = new HashMap<>();
}

// Simulate collecting data from sensors
public void collectData() {
    // Simulating data collection for 5 sensors
    Random random = new Random();
    for (int i = 1; i <= 5; i++) {
        String sensorID = "Sensor" + i;
        double dataValue = random.nextDouble() * 100; // Random data value between 0 and 100
        sensorData.put(sensorID, dataValue);
    }
}

// Calculate average data value for all sensors
public double calculateAverage() {
    double total = 0;
    for (double data : sensorData.values()) {
        total += data;
    }
    return sensorData.isEmpty() ? 0 : total / sensorData.size();
}

public void displaySensorData() {
    System.out.println("Sensor Data:");
    for (Map.Entry<String, Double> entry : sensorData.entrySet()) {
        System.out.println("Sensor ID: " + entry.getKey() + ", Data: " + entry.getValue());
    }
}
```

```
}


public class SensorSimulation {
    public static void main(String[] args) {
        SensorDataCollection sensorCollection = new SensorDataCollection();


        // Simulate data collection
        sensorCollection.collectData();


        // Display collected sensor data
        sensorCollection.displaySensorData();


        // Calculate and display the average data value for all sensors
        double average = sensorCollection.calculateAverage();
        System.out.println("Average data value for all sensors: " + average);
    }
}
```

19.Create a program to manage student course registrations.

Use a HashSet to store unique student IDs for a specific course.

Implement methods to add and remove students from the course and display the current list of registered students.

Code:save file as CourseRegistration.java

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;


public class CourseRegistration {
```

```
private Set<String> studentIDs;


public CourseRegistration() {
    studentIDs = new HashSet<>();
}


public void addStudent(String studentID) {
    if (studentIDs.add(studentID)) {
        System.out.println("Student with ID " + studentID + " added to the course.");
    } else {
        System.out.println("Student with ID " + studentID + " is already registered in the course.");
    }
}


public void removeStudent(String studentID) {
    if (studentIDs.remove(studentID)) {
        System.out.println("Student with ID " + studentID + " removed from the course.");
    } else {
        System.out.println("Student with ID " + studentID + " is not registered in the course.");
    }
}


public void displayRegisteredStudents() {
    System.out.println("Registered Students:");
    for (String studentID : studentIDs) {
        System.out.println("Student ID: " + studentID);
    }
}


public static void main(String[] args) {
    CourseRegistration course = new CourseRegistration();
```

```
Scanner scanner = new Scanner(System.in);

int choice;
do {
    System.out.println("\nCourse Registration Menu:");
    System.out.println("1. Add Student");
    System.out.println("2. Remove Student");
    System.out.println("3. Display Registered Students");
    System.out.println("4. Exit");
    System.out.print("Enter your choice: ");
    choice = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    switch (choice) {
        case 1:
            System.out.print("Enter student ID to add: ");
            String addID = scanner.nextLine();
            course.addStudent(addID);
            break;
        case 2:
            System.out.print("Enter student ID to remove: ");
            String removeID = scanner.nextLine();
            course.removeStudent(removeID);
            break;
        case 3:
            course.displayRegisteredStudents();
            break;
        case 4:
            System.out.println("Exiting the program. Goodbye!");
            break;
        default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

20.Model a hospital patient queue system using a queue.

Simulate patients arriving at the hospital and being added to the queue.

Implement methods to process patients (dequeue) and display the current queue.

Code: save as HospitalQueue.java

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class HospitalQueue {

    public static void main(String[] args) {
        Queue<String> patientQueue = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);

        int choice;
        do {
            System.out.println("\nHospital Queue Management:");
            System.out.println("1. Add Patient");
            System.out.println("2. Process Next Patient");
            System.out.println("3. Display Queue");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
```

```
choice = scanner.nextInt();
scanner.nextLine(); // Consume newline

switch (choice) {
    case 1:
        System.out.print("Enter patient's name to add: ");
        String patientName = scanner.nextLine();
        patientQueue.offer(patientName);
        System.out.println(patientName + " has been added to the queue.");
        break;
    case 2:
        if (!patientQueue.isEmpty()) {
            String processedPatient = patientQueue.poll();
            System.out.println("Processing patient: " + processedPatient);
        } else {
            System.out.println("No patients in the queue.");
        }
        break;
    case 3:
        System.out.println("Current Patient Queue:");
        if (patientQueue.isEmpty()) {
            System.out.println("The queue is empty.");
        } else {
            for (String patient : patientQueue) {
                System.out.println(patient);
            }
        }
        break;
    case 4:
        System.out.println("Exiting the program. Goodbye!");
        break;
```

```
            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

21.Create an Array List of integers. Perform the following operations:

Add elements at the end.

Insert an element at a specific index.

Remove an element at a specific index.

Print the Array List after each operation.

Code: save code as ArrayListOperations.java

```
import java.util.ArrayList;

import java.util.Scanner;
```

```
public class ArrayListOperations {
    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nArrayList Operations:");
            System.out.println("1. Add element at the end");
            System.out.println("2. Insert element at specific index");
            System.out.println("3. Remove element at specific index");
            System.out.println("4. Print ArrayList");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter the element to add at the end: ");
                    int elementToAdd = scanner.nextInt();
                    arrayList.add(elementToAdd);
                    System.out.println("Element added successfully.");
                    break;
                case 2:
                    System.out.print("Enter the index to insert the element: ");
                    int insertIndex = scanner.nextInt();
                    System.out.print("Enter the element to insert: ");
                    int elementToInsert = scanner.nextInt();
                    if (insertIndex >= 0 && insertIndex <= arrayList.size()) {
                        arrayList.add(insertIndex, elementToInsert);
                        System.out.println("Element inserted successfully.");
                    } else {
                        System.out.println("Invalid index. Element not inserted.");
                    }
                    break;
                case 3:
                    System.out.print("Enter the index to remove the element: ");
                    int removeIndex = scanner.nextInt();
                    if (removeIndex >= 0 && removeIndex < arrayList.size()) {
                        arrayList.remove(removeIndex);
                        System.out.println("Element removed successfully.");
                    } else {
                        System.out.println("Invalid index. Element not removed.");
                    }
                    break;
                case 4:
                    System.out.println("ArrayList: " + arrayList);
                    break;
                case 5:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
                    break;
            }
        } while (choice != 5);

        scanner.close();
      }
}
```

22.Create a HashSet of strings.

Add elements to the HashSet.

Demonstrate the use of methods like contains, remove, and size.

Print the final HashSet.

Code: save as HashSetOperations.java

```
import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;


public class HashSetOperations {

  public static void main(String[] args) {

    Set<String> hashSet = new HashSet<>();

    Scanner scanner = new Scanner(System.in);

    int choice;


    do {

      System.out.println("\nHashSet Operations:");

      System.out.println("1. Add element");

      System.out.println("2. Check if element exists");

      System.out.println("3. Remove element");

      System.out.println("4. Check size of HashSet");

      System.out.println("5. Print HashSet");

      System.out.println("6. Exit");

      System.out.print("Enter your choice: ");

      choice = scanner.nextInt();

      scanner.nextLine(); // Consume the newline character


      switch (choice) {

        case 1:

          System.out.print("Enter the element to add: ");

          String elementToAdd = scanner.nextLine();
    hashSet.add(elementToAdd);
    System.out.println("Element added successfully.");
    break;
case 2:
    System.out.print("Enter the element to check: ");
    String elementToCheck = scanner.nextLine();
    if (hashSet.contains(elementToCheck)) {
        System.out.println("Element exists in HashSet.");
    } else {
        System.out.println("Element does not exist in HashSet.");
    }
    break;
case 3:
    System.out.print("Enter the element to remove: ");
    String elementToRemove = scanner.nextLine();
    if (hashSet.remove(elementToRemove)) {
        System.out.println("Element removed successfully.");
    } else {
        System.out.println("Element not found in HashSet.");
    }
    break;
case 4:
    System.out.println("Size of HashSet: " + hashSet.size());
    break;
case 5:
    System.out.println("HashSet: " + hashSet);
    break;
case 6:
    System.out.println("Exiting the program.");
    break;
default:
```

```
                System.out.println("Invalid choice. Please enter a valid option.");
                break;
            }
        } while (choice != 6);

        scanner.close();
    }
}
```

23.Implement a HashMap to store student names and their corresponding grades.

Allow the user to add entries, remove entries, and display the current content of the HashMap.

Code: save as StudentGrades.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class StudentGrades {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Integer> studentGrades = new HashMap<>();

        while (true) {
            System.out.println("\nStudent Grades Menu:");
            System.out.println("1. Add student entry");
            System.out.println("2. Remove student entry");
            System.out.println("3. Display student grades");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline
```

```
switch (choice) {
    case 1:
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();
        System.out.print("Enter student grade: ");
        int grade = scanner.nextInt();
        studentGrades.put(name, grade);
        System.out.println("Student entry added.");
        break;
    case 2:
        System.out.print("Enter student name to remove: ");
        String nameToRemove = scanner.nextLine();
        if (studentGrades.containsKey(nameToRemove)) {
            studentGrades.remove(nameToRemove);
            System.out.println("Student entry removed.");
        } else {
            System.out.println("Student not found in the records.");
        }
        break;
    case 3:
        if (studentGrades.isEmpty()) {
            System.out.println("No student records available.");
        } else {
            System.out.println("Student Grades:");
            for (Map.Entry<String, Integer> entry : studentGrades.entrySet()) {
                System.out.println(entry.getKey() + ": " + entry.getValue());
            }
        }
        break;
    case 4:
```

```
                    System.out.println("Exiting program. Goodbye!");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
            }
        }
    }
}
```

24.Create two sets of integers.

Perform set operations like union, intersection, and difference.

Display the results of each operation.

Code: save file as class SetOperations ????**not menu driven**

```
import java.util.HashSet;
import java.util.Set;

public class SetOperations {
    public static void main(String[] args) {
        // Creating two sets of integers
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        // Adding elements to set1
        set1.add(1);
        set1.add(2);
        set1.add(3);
        set1.add(4);

        // Adding elements to set2
        set2.add(3);
        set2.add(4);

        set2.add(5);

        set2.add(6);


        // Union of two sets

        Set<Integer> union = new HashSet<>(set1);

        union.addAll(set2);


        // Intersection of two sets

        Set<Integer> intersection = new HashSet<>(set1);

        intersection.retainAll(set2);


        // Difference of two sets (elements in set1 but not in set2)

        Set<Integer> difference = new HashSet<>(set1);

        difference.removeAll(set2);


        // Displaying the results

        System.out.println("Set 1: " + set1);

        System.out.println("Set 2: " + set2);

        System.out.println("Union: " + union);

        System.out.println("Intersection: " + intersection);

        System.out.println("Difference (Set 1 - Set 2): " + difference);

    }

}
```

//menu driven

Save file as SetOperationsMenu.java

Code:

```
import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;
```

```
public class SetOperationsMenu {
  public static void main(String[] args) {
    Set<Integer> set1 = new HashSet<>();
    Set<Integer> set2 = new HashSet<>();
    Scanner scanner = new Scanner(System.in);
    int choice;

    do {
      System.out.println("\nSet Operations Menu:");
      System.out.println("1. Add element to Set 1");
      System.out.println("2. Add element to Set 2");
      System.out.println("3. Perform Union");
      System.out.println("4. Perform Intersection");
      System.out.println("5. Perform Difference (Set 1 - Set 2)");
      System.out.println("6. Display Sets");
      System.out.println("7. Exit");
      System.out.print("Enter your choice: ");
      choice = scanner.nextInt();

      switch (choice) {
        case 1:
          System.out.print("Enter element to add to Set 1: ");
          int element1 = scanner.nextInt();
          set1.add(element1);
          System.out.println("Element added to Set 1.");
          break;
        case 2:
          System.out.print("Enter element to add to Set 2: ");
          int element2 = scanner.nextInt();
          set2.add(element2);
```

```
            System.out.println("Element added to Set 2.");
            break;
        case 3:
            Set<Integer> union = new HashSet<>(set1);
            union.addAll(set2);
            System.out.println("Union: " + union);
            break;
        case 4:
            Set<Integer> intersection = new HashSet<>(set1);
            intersection.retainAll(set2);
            System.out.println("Intersection: " + intersection);
            break;
        case 5:
            Set<Integer> difference = new HashSet<>(set1);
            difference.removeAll(set2);
            System.out.println("Difference (Set 1 - Set 2): " + difference);
            break;
        case 6:
            System.out.println("Set 1: " + set1);
            System.out.println("Set 2: " + set2);
            break;
        case 7:
            System.out.println("Exiting the program.");
            break;
        default:
            System.out.println("Invalid choice. Please enter a valid option.");
            break;
    }
} while (choice != 7);

scanner.close();
```

```
    }
}
```

25.Implement a queue using the built-in collections module or using a list.

Perform enqueue and dequeue operations.

Display the queue after each operation.

Save file as QueueImplementation.java

Code:

```
import java.util.ArrayList;
import java.util.Scanner;

public class QueueImplementation {
    private ArrayList<Integer> queue = new ArrayList<>();

    public void enqueue(int item) {
        queue.add(item);
        System.out.println("Enqueued: " + item);
        displayQueue();
    }

    public void dequeue() {
        if (!queue.isEmpty()) {
            int item = queue.remove(0);
            System.out.println("Dequeued: " + item);
        } else {
            System.out.println("Queue is empty");
        }
        displayQueue();
    }

    public void displayQueue() {
```

```
    System.out.println("Queue: " + queue);
}

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

    do {
        System.out.println("\nQueue Operations:");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Display Queue");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter element to enqueue: ");
                int item = scanner.nextInt();
                queueObj.enqueue(item);
                break;
            case 2:
                queueObj.dequeue();
                break;
            case 3:
                queueObj.displayQueue();
                break;
            case 4:
                System.out.println("Exiting the program.");
```

```
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
                    break;
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

26.Implement a stack using a list.

Demonstrate push and pop operations.

Display the stack after each operation.

Save filr as : StackImplementation.java

Code:

```
import java.util.ArrayList;
import java.util.Scanner;

public class StackImplementation {
    private ArrayList<Integer> stack = new ArrayList<>();

    public void push(int item) {
        stack.add(item);
        System.out.println("Pushed: " + item);
        displayStack();
    }

    public void pop() {
        if (!stack.isEmpty()) {
```

```
        int item = stack.remove(stack.size() - 1);
        System.out.println("Popped: " + item);
    } else {
        System.out.println("Stack is empty");
    }
    displayStack();
}

public void displayStack() {
    System.out.println("Stack: " + stack);
}

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

    do {
        System.out.println("\nStack Operations:");
        System.out.println("1. Push");
        System.out.println("2. Pop");
        System.out.println("3. Display Stack");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter element to push: ");
                int item = scanner.nextInt();
                stackObj.push(item);
                break;
              case 2:
                stackObj.pop();
                break;
              case 3:
                stackObj.displayStack();
                break;
              case 4:
                System.out.println("Exiting the program.");
                break;
              default:
                System.out.println("Invalid choice. Please enter a valid option.");
                break;
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

27.Create a list of random integers.

Use a sorting algorithm (e.g., quicksort or merge sort) to sort the list.

Display the sorted list.

Save as : MergeSortMenu.java

Code for merge sort

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
```

```
public class MergeSortMenu {
    public static void main(String[] args) {
        List<Integer> randomList = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nMerge Sort Menu:");
            System.out.println("1. Generate Random List");
            System.out.println("2. Sort List (Merge Sort)");
            System.out.println("3. Display Sorted List");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter the size of the list: ");
                    int size = scanner.nextInt();
                    randomList = generateRandomList(size);
                    System.out.println("Random List generated.");
                    break;
                case 2:
                    if (randomList.isEmpty()) {
                        System.out.println("Please generate a list first.");
                    } else {
                        mergeSort(randomList, 0, randomList.size() - 1);
                        System.out.println("List sorted using Merge Sort.");
                    }
                    break;
                case 3:
                    if (randomList.isEmpty()) {
                        System.out.println("Please generate and sort the list first.");
                    } else {
                        System.out.println("Sorted List:");
                        System.out.println(randomList);
                    }
                    break;
                case 4:
                    System.out.println("Exiting the program.");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
                    break;
            }
        } while (choice != 4);

        scanner.close();
    }

    public static List<Integer> generateRandomList(int size) {
        List<Integer> randomList = new ArrayList<>();
        Random random = new Random();

        for (int i = 0; i < size; i++) {
            randomList.add(random.nextInt(100)); // Generating random integers up to 100, change as
needed
        }

        return randomList;
}
```

```
public static void mergeSort(List<Integer> arr, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

public static void merge(List<Integer> arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    List<Integer> leftArray = new ArrayList<>(arr.subList(left, mid + 1));
    List<Integer> rightArray = new ArrayList<>(arr.subList(mid + 1, right + 1));

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (leftArray.get(i) <= rightArray.get(j)) {
            arr.set(k, leftArray.get(i));
            i++;
        } else {
            arr.set(k, rightArray.get(j));
            j++;
        }
        k++;
    }

    while (i < n1) {
            arr.set(k, leftArray.get(i));
            i++;
            k++;
        }

        while (j < n2) {
            arr.set(k, rightArray.get(j));
            j++;
            k++;
        }
    }
}
```

//Code for quick sort

Save as QuickSortMenu.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class QuickSortMenu {
    public static void main(String[] args) {
        List<Integer> randomList = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\nQuick Sort Menu:");
            System.out.println("1. Generate Random List");
            System.out.println("2. Sort List (Quick Sort)");
System.out.println("3. Display Sorted List");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();

switch (choice) {
  case 1:
    System.out.print("Enter the size of the list: ");
    int size = scanner.nextInt();
    randomList = generateRandomList(size);
    System.out.println("Random List generated.");
    break;
  case 2:
    if (randomList.isEmpty()) {
      System.out.println("Please generate a list first.");
    } else {
      quickSort(randomList, 0, randomList.size() - 1);
      System.out.println("List sorted using Quick Sort.");
    }
    break;
  case 3:
    if (randomList.isEmpty()) {
      System.out.println("Please generate and sort the list first.");
    } else {
      System.out.println("Sorted List:");
      System.out.println(randomList);
    }
    break;
  case 4:
    System.out.println("Exiting the program.");
    break;
```

```
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
                    break;
            }
        } while (choice != 4);

        scanner.close();
    }

    public static List<Integer> generateRandomList(int size) {
        List<Integer> randomList = new ArrayList<>();
        Random random = new Random();

        for (int i = 0; i < size; i++) {
            randomList.add(random.nextInt(100)); // Generating random integers up to 100, change as
needed
        }

        return randomList;
    }

    public static void quickSort(List<Integer> arr, int low, int high) {
        if (low < high) {
            int partitionIndex = partition(arr, low, high);
            quickSort(arr, low, partitionIndex - 1);
            quickSort(arr, partitionIndex + 1, high);
        }
    }

    public static int partition(List<Integer> arr, int low, int high) {
        int pivot = arr.get(high);
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr.get(j) <= pivot) {
                i++;
                int temp = arr.get(i);
                arr.set(i, arr.get(j));
                arr.set(j, temp);
            }
        }

        int temp = arr.get(i + 1);
        arr.set(i + 1, arr.get(high));
        arr.set(high, temp);

        return i + 1;
    }
}
```

28.Design a generic class for an inventory system that can handle different types of items (e.g., electronics, groceries).

Save code as InventorySystemMenu.java

CODE:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class Inventory<T> {
    private Map<T, Integer> items;
    public Inventory() {
        this.items = new HashMap<>();
    }

    public void addItem(T item, int quantity) {
        items.put(item, items.getOrDefault(item, 0) + quantity);
        System.out.println(quantity + " " + item.getClass().getSimpleName() + "(s) added to inventory.");
    }

    public void removeItem(T item, int quantity) {
        if (items.containsKey(item)) {
            int currQuantity = items.get(item);
            if (currQuantity >= quantity) {
                items.put(item, currQuantity - quantity);
                System.out.println(quantity + " " + item.getClass().getSimpleName() + "(s) removed from
inventory.");
            } else {
                System.out.println("Insufficient quantity of " + item.getClass().getSimpleName() + " in
inventory.");
            }
        } else {
            System.out.println(item.getClass().getSimpleName() + " not found in inventory.");
        }
    }

    public void displayInventory() {
        System.out.println("Inventory:");
        for (Map.Entry<T, Integer> entry : items.entrySet()) {
            System.out.println(entry.getValue() + " " + entry.getKey().getClass().getSimpleName() + "(s)");
        }
    }
}
```

```
class Electronics {
  // Electronics Class (Sample)
}

class Groceries {
  // Groceries Class (Sample)
}

public class InventorySystemMenu {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Inventory<Electronics> electronicsInventory = new Inventory<>();
    Inventory<Groceries> groceriesInventory = new Inventory<>();

    int choice;
    do {
      System.out.println("\nInventory System Menu:");
      System.out.println("1. Add Electronics Item");
      System.out.println("2. Remove Electronics Item");
      System.out.println("3. Add Groceries Item");
      System.out.println("4. Remove Groceries Item");
      System.out.println("5. Display Electronics Inventory");
      System.out.println("6. Display Groceries Inventory");
      System.out.println("7. Exit");
      System.out.print("Enter your choice: ");
      choice = scanner.nextInt();

      switch (choice) {
        case 1:
          electronicsInventory.addItem(new Electronics(), getQuantity(scanner));
```

```
                break;
            case 2:
                electronicsInventory.removeItem(new Electronics(), getQuantity(scanner));
                break;
            case 3:
                groceriesInventory.addItem(new Groceries(), getQuantity(scanner));
                break;
            case 4:
                groceriesInventory.removeItem(new Groceries(), getQuantity(scanner));
                break;
            case 5:
                electronicsInventory.displayInventory();
                break;
            case 6:
                groceriesInventory.displayInventory();
                break;
            case 7:
                System.out.println("Exiting the program.");
                break;
            default:
                System.out.println("Invalid choice. Please enter a valid option.");
                break;
        }
    } while (choice != 7);

    scanner.close();
}

private static int getQuantity(Scanner scanner) {
    System.out.print("Enter quantity: ");
    return scanner.nextInt();
    }
}
```

29.Implement methods to add items to the inventory, remove items, and display the current inventory for a specific item type.

Save file as: InventorySystemMenuDriven.java

Code:

```
import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;


class Inventory<T> {

    private Map<T, Integer> items;


    public Inventory() {

        this.items = new HashMap<>();

    }


    public void addItem(T item, int quantity) {

        items.put(item, items.getOrDefault(item, 0) + quantity);

        System.out.println(quantity + " " + item.getClass().getSimpleName() + "(s) added to inventory.");

    }


    public void removeItem(T item, int quantity) {

        if (items.containsKey(item)) {

            int currQuantity = items.get(item);

            if (currQuantity >= quantity) {

                items.put(item, currQuantity - quantity);

                System.out.println(quantity + " " + item.getClass().getSimpleName() + "(s) removed from
inventory.");

            } else {
                System.out.println("Insufficient quantity of " + item.getClass().getSimpleName() + " in
inventory.");
            }
        } else {
            System.out.println(item.getClass().getSimpleName() + " not found in inventory.");
        }
    }

    public void displayInventory() {
        System.out.println("Inventory:");
        for (Map.Entry<T, Integer> entry : items.entrySet()) {
            System.out.println(entry.getValue() + " " + entry.getKey().getClass().getSimpleName() + "(s)");
        }
    }

    public int getQuantityForItem(T item) {
        return items.getOrDefault(item, 0);
    }
}

class Electronics {
    // Electronics Class (Sample)
}

class Groceries {
    // Groceries Class (Sample)
}

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

```
Inventory<Electronics> electronicsInventory = new Inventory<>();
Inventory<Groceries> groceriesInventory = new Inventory<>();

int choice;
do {
  System.out.println("\nInventory System Menu:");
  System.out.println("1. Add Electronics Item");
  System.out.println("2. Remove Electronics Item");
  System.out.println("3. Display Electronics Inventory");
  System.out.println("4. Add Groceries Item");
  System.out.println("5. Remove Groceries Item");
  System.out.println("6. Display Groceries Inventory");
  System.out.println("7. Exit");
  System.out.print("Enter your choice: ");
  choice = scanner.nextInt();

  switch (choice) {
    case 1:
      electronicsInventory.addItem(new Electronics(), getQuantity(scanner));
      break;
    case 2:
      electronicsInventory.removeItem(new Electronics(), getQuantity(scanner));
      break;
    case 3:
      electronicsInventory.displayInventory();
      break;
    case 4:
      groceriesInventory.addItem(new Groceries(), getQuantity(scanner));
      break;
    case 5:
      groceriesInventory.removeItem(new Groceries(), getQuantity(scanner));
```

```
                break;
            case 6:
                groceriesInventory.displayInventory();
                break;
            case 7:
                System.out.println("Exiting the program.");
                break;
            default:
                System.out.println("Invalid choice. Please enter a valid option.");
                break;
          }
        } while (choice != 7);

        scanner.close();
    }

    private static int getQuantity(Scanner scanner) {
        System.out.print("Enter quantity: ");
        return scanner.nextInt();
    }
}
```

30.Develop a program that works with generic collections (e.g., Array List, HashMap). Implement a method that accepts a generic collection and performs an operation such as sorting or filtering based on the type of elements.

Save file as: CollectionOperations.java

Code:for hash map

```
import java.util.*;
import java.util.function.Predicate;

public class CollectionOperations {

    // Method to sort the generic collection
```

```
public static <T extends Comparable<? super T>> void sortCollection(Collection<T> collection) {
    List<T> list = new ArrayList<>(collection);
    Collections.sort(list);
    collection.clear();
    collection.addAll(list);
}

// Method to filter the generic collection
public static <T> void filterCollection(Collection<T> collection, Predicate<T> predicate) {
    collection.removeIf(element -> !predicate.test(element));
}

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

    while (true) {
        System.out.println("Select an operation:");
        System.out.println("1. Sort collection");
        System.out.println("2. Filter collection");
        System.out.println("3. Exit");

        int choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                performSortOperation(scanner);
                break;
            case 2:
                performFilterOperation(scanner);
                break;
            case 3:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please enter again.");
        }
    }
}

private static void performSortOperation(Scanner scanner) {
    System.out.println("Enter the elements separated by spaces:");
    String[] elements = scanner.nextLine().split("\\s+");

    List<String> collection = new ArrayList<>(Arrays.asList(elements));
    sortCollection(collection);

    System.out.println("Sorted collection: " + collection);
}

private static void performFilterOperation(Scanner scanner) {
    System.out.println("Enter the elements separated by spaces:");
    String[] elements = scanner.nextLine().split("\\s+");

    List<String> collection = new ArrayList<>(Arrays.asList(elements));

    System.out.println("Enter the condition for filtering (e.g., length > 5):");
    String condition = scanner.nextLine();

    Predicate<String> predicate = createPredicate(condition);
    filterCollection(collection, predicate);

    System.out.println("Filtered collection: " + collection);
}

private static Predicate<String> createPredicate(String condition) {
    return s -> {
        // Implement the logic to evaluate the condition
        // For simplicity, assume filtering by length
        String[] parts = condition.split("\\s+");
        if (parts.length != 3) {
            System.out.println("Invalid condition format");
            return false;
        }

        String operator = parts[1];
        int length = Integer.parseInt(parts[2]);

        switch (operator) {
            case ">":
                return s.length() > length;
            case "<":
                return s.length() < length;
            case "==":
                return s.length() == length;
            default:
                System.out.println("Invalid operator");
                return false;
        }
    };
}
```

```
}
```

Code : For array list

Save file as: GenericCollectionOperations.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;
import java.util.function.Predicate;

public class GenericCollectionOperations {

  // Method to perform sorting on a generic collection
  public static <T extends Comparable<? super T>> void sortCollection(List<T> collection) {
    Collections.sort(collection);
  }

  // Method to perform filtering on a generic collection
  public static <T> void filterCollection(List<T> collection, Predicate<T> predicate) {
    collection.removeIf(element -> !predicate.test(element));
  }

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

    while (true) {
      System.out.println("Select an operation:");
      System.out.println("1. Sort collection");
      System.out.println("2. Filter collection");
      System.out.println("3. Exit");
```

```
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    performSortOperation(scanner);
                    break;
                case 2:
                    performFilterOperation(scanner);
                    break;
                case 3:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please enter again.");
            }
        }
    }

    private static <T extends Comparable<? super T>> void performSortOperation(Scanner scanner) {
        System.out.println("Enter elements separated by spaces:");
        String[] elements = scanner.nextLine().split("\\s+");

        List<String> stringCollection = new ArrayList<>(List.of(elements));
        sortCollection(stringCollection);

        System.out.println("Sorted collection: " + stringCollection);
    }
```

```
private static <T> void performFilterOperation(Scanner scanner) {
    System.out.println("Enter elements separated by spaces:");
    String[] elements = scanner.nextLine().split("\\s+");

    List<String> stringCollection = new ArrayList<>(List.of(elements));

    System.out.println("Enter the condition for filtering (e.g., length > 5):");
    String condition = scanner.nextLine();

    Predicate<String> predicate = createPredicate(condition);

    filterCollection(stringCollection, predicate);

    System.out.println("Filtered collection: " + stringCollection);
}

private static Predicate<String> createPredicate(String condition) {
    return s -> {
        // For simplicity, assume filtering by length
        String[] parts = condition.split("\\s+");
        if (parts.length != 3) {
            System.out.println("Invalid condition format");
            return false;
        }

        String operator = parts[1];
        int length = Integer.parseInt(parts[2]);

        switch (operator) {
            case ">":
                return s.length() > length;
            case "<":
                return s.length() < length;
            case "==":
                return s.length() == length;
            default:
                System.out.println("Invalid operator");
                return false;
        }
    };
  }
}
```

31.Design a generic class to schedule events with different data types for event details (e.g., event name, date, location).

Implement methods to add events, remove events, and display the schedule for a specific type of event.

Save file as: EventSchedulerDemo.java

Code:

```
import java.util.*;

class Event<T> {
  private Map<T, List<String>> eventSchedule;

  public Event() {
    this.eventSchedule = new HashMap<>();
  }

  public void addEvent(T eventType, String details) {
    if (!eventSchedule.containsKey(eventType)) {
      eventSchedule.put(eventType, new ArrayList<>());
    }
    eventSchedule.get(eventType).add(details);
    }

    public void removeEvent(T eventType, int index) {
        if (eventSchedule.containsKey(eventType)) {
            List<String> events = eventSchedule.get(eventType);
            if (index >= 0 && index < events.size()) {
                events.remove(index);
                System.out.println("Event removed successfully.");
            } else {
                System.out.println("Invalid event index.");
            }
        } else {
            System.out.println("No events found for the given type.");
        }
    }

    public void displaySchedule(T eventType) {
        if (eventSchedule.containsKey(eventType)) {
            System.out.println("Schedule for " + eventType + " events:");
            List<String> events = eventSchedule.get(eventType);
            for (int i = 0; i < events.size(); i++) {
                System.out.println(i + ": " + events.get(i));
            }
        } else {
            System.out.println("No events found for the given type.");
        }
    }
}

public class EventSchedulerDemo {
    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
Event<String> eventScheduler = new Event<>();

while (true) {
  System.out.println("Select an operation:");
  System.out.println("1. Add Event");
  System.out.println("2. Remove Event");
  System.out.println("3. Display Schedule");
  System.out.println("4. Exit");

  int choice = scanner.nextInt();
  scanner.nextLine(); // Consume newline

  switch (choice) {
    case 1:
      System.out.println("Enter event type:");
      String eventType = scanner.nextLine();
      System.out.println("Enter event details:");
      String eventDetails = scanner.nextLine();
      eventScheduler.addEvent(eventType, eventDetails);
      break;
    case 2:
      System.out.println("Enter event type to remove:");
      String typeToRemove = scanner.nextLine();
      System.out.println("Enter index of event to remove:");
      int indexToRemove = scanner.nextInt();
      scanner.nextLine(); // Consume newline
      eventScheduler.removeEvent(typeToRemove, indexToRemove);
      break;
    case 3:
      System.out.println("Enter event type to display:");
```

```
                    String typeToDisplay = scanner.nextLine();
                    eventScheduler.displaySchedule(typeToDisplay);
                    break;
                case 4:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Please enter again.");
            }
        }
    }
}
```

32.Develop a program that manages an employee hierarchy using generics.

Create a generic class for employees with attributes like name, ID, and salary.

Implement methods to add employees, remove employees, and display the hierarchy.

Save file as: EmployeeHierarchy.java

Code:

```
import java.util.ArrayList;
import java.util.List;
public class EmployeeHierarchy<T> {

    private Employee<T> root;

    public EmployeeHierarchy(Employee<T> root) {
        this.root = root;
    }

    public void addEmployee(Employee<T> employee, Employee<T> manager) {
```

```
    if (manager == null) {
        root = employee;
    } else {
        manager.addSubordinate(employee);
    }
    employee.setManager(manager);
}

public void removeEmployee(Employee<T> employee) {
    if (employee == root) {
        root = null;
    } else {
        Employee<T> manager = employee.getManager();
        manager.getSubordinates().remove(employee);
    }
}

public void displayHierarchy() {
    displayHierarchy(root, 0);
}

private void displayHierarchy(Employee<T> employee, int level) {
    for (int i = 0; i < level; i++) {
        System.out.print("  ");
    }
    System.out.println(employee);
    for (Employee<T> subordinate : employee.getSubordinates()) {
        displayHierarchy(subordinate, level + 1);
    }
}
```

```
public static class Employee<T> {
    private String name;
    private int id;
    private double salary;
    private T manager;
    private List<Employee<T>> subordinates;

    public Employee(String name, int id, double salary) {
        this.name = name;
        this.id = id;
        this.salary = salary;
        this.manager = null;
        this.subordinates = new ArrayList<>();
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }

    public double getSalary() {
        return salary;
    }

    public void setManager(Employee<T> manager) {
        this.manager = (T) manager;
    }
```

```
        public Employee<T> getManager() {
            return (EmployeeHierarchy.Employee<T>) manager;
        }

        public void addSubordinate(Employee<T> subordinate) {
            subordinates.add(subordinate);
        }

        public List<Employee<T>> getSubordinates() {
            return subordinates;
        }

        @Override
        public String toString() {
            return "Employee{" +
                    "name='" + name + '\'' +
                    ", id=" + id +
                    ", salary=" + salary +
                    ", manager=" + (manager != null ? ((EmployeeHierarchy.Employee<T>)
manager).getName() : "null") +
                    '}';
        }
    }

    public static class Manager<T> extends Employee<T> {

        public Manager(String name, int id, double salary) {
            super(name, id, salary);
            //TODO Auto-generated constructor stub
        }
        // Additional fields and methods specific to managers
    }

    public static void main(String[] args) {
        Employee<String> ceo = new Employee<>("John Doe", 1, 100000);
        Employee<String> manager1 = new Manager<>("Jane Doe", 2, 80000);
        Employee<String> manager2 = new Manager<>("Mike Smith", 3, 70000);

        EmployeeHierarchy<String> hierarchy = new EmployeeHierarchy<>(ceo);
        hierarchy.addEmployee(manager1, ceo);
        hierarchy.addEmployee(manager2, ceo);
        hierarchy.addEmployee(new Employee<>("Alice Johnson", 4, 50000), manager1);
        hierarchy.addEmployee(new Employee<>("Bob Lee", 5, 45000), manager2);

        hierarchy.displayHierarchy();
    }
}
```

33.Design a generic class to handle medical records with different types of patient data.

Implement methods to add patient records, update medical information, and display patient records for a specific condition.

Save file as: MedicalRecordManager.java

Code:

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

class Patient {
    private String name;
    }

    private int age;
    private String condition;
    private Map<String, String> medicalInfo; // Store medical information as key-value pairs

    public Patient(String name, int age, String condition) {
        this.name = name;
        this.age = age;
        this.condition = condition;
        this.medicalInfo = new HashMap<>();
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getCondition() {
        return condition;
    }

    public void updateMedicalInfo(String key, String value) {
        medicalInfo.put(key, value);
    }

    public Map<String, String> getMedicalInfo() {
        return medicalInfo;
    }
}
```

```
public class MedicalRecordManager {
  private List<Patient> patients;

  public MedicalRecordManager() {
    this.patients = new ArrayList<>();
  }

  public void addPatientRecord(Patient patient) {
    patients.add(patient);
  }

  public void updateMedicalInformation(String patientName, String key, String value) {
    for (Patient patient : patients) {
      if (patient.getName().equalsIgnoreCase(patientName)) {
        patient.updateMedicalInfo(key, value);
        return;
      }
    }
    System.out.println("Patient not found.");
  }

  public void displayPatientRecordsForCondition(String condition) {
    System.out.println("Patients with condition " + condition + ":");
    for (Patient patient : patients) {
      if (patient.getCondition().equalsIgnoreCase(condition)) {
        System.out.println("Name: " + patient.getName() + ", Age: " + patient.getAge());
        System.out.println("Medical Information:");
        for (Map.Entry<String, String> entry : patient.getMedicalInfo().entrySet()) {
          System.out.println(entry.getKey() + ": " + entry.getValue());
        }
```

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

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

    int choice = 0;
    while (choice != 4) {
        System.out.println("Menu:");
        System.out.println("1. Add Patient Record");
        System.out.println("2. Update Medical Information");
        System.out.println("3. Display Patient Records for a Condition");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        switch (choice) {
            case 1:
                System.out.print("Enter patient name: ");
                String name = scanner.nextLine();
                System.out.print("Enter patient age: ");
                int age = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                System.out.print("Enter patient condition: ");
                String condition = scanner.nextLine();

                Patient newPatient = new Patient(name, age, condition);
                recordManager.addPatientRecord(newPatient);

                System.out.println("Patient record added.");

                break;

              case 2:

                System.out.print("Enter patient name: ");

                String patientName = scanner.nextLine();

                System.out.print("Enter medical information key: ");

                String key = scanner.nextLine();

                System.out.print("Enter medical information value: ");

                String value = scanner.nextLine();


                recordManager.updateMedicalInformation(patientName, key, value);

                break;

              case 3:

                System.out.print("Enter condition to display records: ");

                String conditionToDisplay = scanner.nextLine();

                recordManager.displayPatientRecordsForCondition(conditionToDisplay);

                break;

              case 4:

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

                break;

              default:

                System.out.println("Invalid choice. Please enter a number between 1 and 4.");

            }

        }

        scanner.close();

    }

}
```

34.Create a generic class to manage a sports league with different types of teams.

Implement methods to add teams, update match results, and display the standings for a specific type of sport.

Save code as: FootballLeagueManager.java

Code:

```java
import java.util.*;


class Team {

    private String name;

    private int points;


    public Team(String name) {

        this.name = name;

        this.points = 0;

    }


    public String getName() {

        return name;

    }


    public int getPoints() {

        return points;

    }


    public void updatePoints(int points) {

        this.points += points;

    }

}


class SportsLeague {

    private List<Team> teams;
```

```
public SportsLeague() {
    teams = new ArrayList<>();
}

public void addTeam(Team team) {
    teams.add(team);
}

public void updateMatchResult(String team1, String team2, int pointsTeam1, int pointsTeam2) {
    Team t1 = findTeamByName(team1);
    Team t2 = findTeamByName(team2);

    if (t1 != null && t2 != null) {
        t1.updatePoints(pointsTeam1);
        t2.updatePoints(pointsTeam2);
    } else {
        System.out.println("One or both teams not found.");
    }
}

public void displayStandings() {
    System.out.println("League Standings:");
    teams.sort(Comparator.comparingInt(Team::getPoints).reversed());

    for (Team team : teams) {
        System.out.println(team.getName() + " - Points: " + team.getPoints());
    }
}

private Team findTeamByName(String teamName) {
    for (Team team : teams) {
            if (team.getName().equalsIgnoreCase(teamName)) {
                return team;
            }
        }
        return null;
    }
}

public class FootballLeagueManager {
    public static void main(String[] args) {
        SportsLeague leagueManager = new SportsLeague();

        Scanner scanner = new Scanner(System.in);

        int choice = 0;
        while (choice != 4) {
            System.out.println("Menu:");
            System.out.println("1. Add Team");
            System.out.println("2. Update Match Result");
            System.out.println("3. Display Standings");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter team name: ");
                    String teamName = scanner.nextLine();

                    Team newTeam = new Team(teamName);
                    leagueManager.addTeam(newTeam);
                    System.out.println("Team added.");
                    break;
                case 2:
                    System.out.print("Enter first team name: ");
                    String team1 = scanner.nextLine();
                    System.out.print("Enter second team name: ");
                    String team2 = scanner.nextLine();
                    System.out.print("Enter points for team 1: ");
                    int pointsTeam1 = scanner.nextInt();
                    System.out.print("Enter points for team 2: ");
                    int pointsTeam2 = scanner.nextInt();

                    leagueManager.updateMatchResult(team1, team2, pointsTeam1, pointsTeam2);
                    break;
                case 3:
                    leagueManager.displayStandings();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a number between 1 and 4.");
            }
        }
        scanner.close();
    }
}
```

35.Create a class student with private members attendance and marks. Create a class teacher who sets the values for marks and attendance. Finally create a class parent who creates a reflection of methods to know the values of marks and attendance of the student.

Save file as: SchoolManagement.java

Code:

```
import java.util.Scanner;

class Student {
    private int attendance;
    private int marks;

    public void setAttendance(int attendance) {
        this.attendance = attendance;
    }

    public void setMarks(int marks) {
        this.marks = marks;
    }

    public int getAttendance() {
        return attendance;
    }

    public int getMarks() {
        return marks;
    }
}

class Teacher {
    public void setStudentValues(Student student, int attendance, int marks) {
        student.setAttendance(attendance);
        student.setMarks(marks);
    }
}
```

```
class Parent {
    public int viewAttendance(Student student) {
        return student.getAttendance();
    }

    public int viewMarks(Student student) {
        return student.getMarks();
    }
}

public class SchoolManagement {
    public static void main(String[] args) {
        Student student = new Student();
        Teacher teacher = new Teacher();
        Parent parent = new Parent();

        Scanner scanner = new Scanner(System.in);

        int choice = 0;
        while (choice != 4) {
            System.out.println("Menu:");
            System.out.println("1. Set Attendance and Marks by Teacher");
            System.out.println("2. View Attendance by Parent");
            System.out.println("3. View Marks by Parent");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
```

```
            case 1:
                System.out.print("Enter attendance: ");
                int attendance = scanner.nextInt();
                System.out.print("Enter marks: ");
                int marks = scanner.nextInt();

                teacher.setStudentValues(student, attendance, marks);
                System.out.println("Attendance and marks set for the student.");
                break;
            case 2:
                int studentAttendance = parent.viewAttendance(student);
                System.out.println("Student Attendance: " + studentAttendance);
                break;
            case 3:
                int studentMarks = parent.viewMarks(student);
                System.out.println("Student Marks: " + studentMarks);
                break;
            case 4:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
          }
        }
        scanner.close();
    }
}
```

36.WAP to create a list of products with data fields: id, name, price. Create a stream on the product list and generate an optimized code to filter data (e.g. price>30k) based on the product prize. Display the products whose cost is 30k using forEach(). Find the total cost of all products using reduce(). Find the product with minimum and maximum cost. Fint the count of products with prize<30k. Convert the list to map(id,name).

Save file as: ProductStreamExample.java

Code:

```java
import java.util.*;

import java.util.stream.Collectors;


class Product {

    private int id;

    private String name;

    private double price;


    public Product(int id, String name, double price) {

        this.id = id;

        this.name = name;

        this.price = price;

    }


    public int getId() {

        return id;

    }


    public String getName() {

        return name;

    }


    public double getPrice() {

        return price;

    }

}
```

```
public class ProductStreamExample {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product(1, "Product A", 25000));
        products.add(new Product(2, "Product B", 35000));
        products.add(new Product(3, "Product C", 28000));
        products.add(new Product(4, "Product D", 40000));
        products.add(new Product(5, "Product E", 20000));

        // Filter products with price > 30k and display them
        System.out.println("Products with price > 30k:");
        products.stream()
                .filter(product -> product.getPrice() > 30000)
                .forEach(product -> System.out.println(product.getName()));

        // Calculate total cost of all products
        double totalCost = products.stream()
                .mapToDouble(Product::getPrice)
                .sum();
        System.out.println("\nTotal cost of all products: " + totalCost);

        // Find product with minimum cost
        Optional<Product> minProduct = products.stream()
                .min(Comparator.comparingDouble(Product::getPrice));
        minProduct.ifPresent(product -> System.out.println("\nProduct with minimum cost: " +
product.getName()));

        // Find product with maximum cost
        Optional<Product> maxProduct = products.stream()
                .max(Comparator.comparingDouble(Product::getPrice));
        maxProduct.ifPresent(product -> System.out.println("Product with maximum cost: " +
product.getName()));
```

```
        // Count products with price < 30k
        long countLessThan30k = products.stream()
                .filter(product -> product.getPrice() < 30000)
                .count();
        System.out.println("\nNumber of products with price < 30k: " + countLessThan30k);

        // Convert list to map (id, name)
        Map<Integer, String> idNameMap = products.stream()
                .collect(Collectors.toMap(Product::getId, Product::getName));
        System.out.println("\nProduct Map (id, name): " + idNameMap);
    }
}
```

Save file as: ProductStreamMenu.java

Code:

```
import java.util.*;
import java.util.stream.Collectors;

class Product {
    private int id;
    private String name;
    private double price;

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

public class ProductStreamMenu {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        int choice = 0;
        while (choice != 7) {
            System.out.println("Menu:");
            System.out.println("1. Add Product");
            System.out.println("2. Filter Products with price > 30k");
            System.out.println("3. Display Products with price = 30k");
            System.out.println("4. Total cost of all products");
            System.out.println("5. Product with Minimum and Maximum cost");
            System.out.println("6. Count of products with price < 30k");
            System.out.println("7. Convert list to Map (id, name)");
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
```

```
scanner.nextLine(); // Consume newline


switch (choice) {
  case 1:
    System.out.print("Enter product name: ");
    String productName = scanner.nextLine();
    System.out.print("Enter product price: ");
    double productPrice = scanner.nextDouble();
    int newId = products.size() + 1;


    Product newProduct = new Product(newId, productName, productPrice);
    products.add(newProduct);
    System.out.println("Product added.");
    break;
  case 2:
    List<Product> filteredProducts = products.stream()
        .filter(product -> product.getPrice() > 30000)
        .collect(Collectors.toList());
    System.out.println("Products with price > 30k:");
    filteredProducts.forEach(product -> System.out.println(product.getName()));
    break;
  case 3:
    List<Product> price30kProducts = products.stream()
        .filter(product -> product.getPrice() == 30000)
        .collect(Collectors.toList());
    System.out.println("Products with price = 30k:");
    price30kProducts.forEach(product -> System.out.println(product.getName()));
    break;
  case 4:
    double totalCost = products.stream()
        .mapToDouble(Product::getPrice)
```

```
                    .sum();
                System.out.println("Total cost of all products: " + totalCost);
                break;
            case 5:
                Optional<Product> minProduct = products.stream()
                    .min(Comparator.comparingDouble(Product::getPrice));
                minProduct.ifPresent(product -> System.out.println("Product with minimum cost: " +
product.getName()));

                Optional<Product> maxProduct = products.stream()
                    .max(Comparator.comparingDouble(Product::getPrice));
                maxProduct.ifPresent(product -> System.out.println("Product with maximum cost: " +
product.getName()));
                break;
            case 6:
                long countLessThan30k = products.stream()
                    .filter(product -> product.getPrice() < 30000)
                    .count();
                System.out.println("Number of products with price < 30k: " + countLessThan30k);
                break;
            case 7:
                Map<Integer, String> idNameMap = products.stream()
                    .collect(Collectors.toMap(Product::getId, Product::getName));
                System.out.println("Product Map (id, name): " + idNameMap);
                break;
            case 8:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 8.");
        }
    }
```

```
        scanner.close();
    }
}
```