2. Search an Element

Write a program to:

· Create an ArrayList of integers.

· Ask the user to enter a number.

· Check if the number exists in the list.

```
import java.util.*;

public class SearchInArrayList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        System.out.print("Enter a number to search: ");
        int num = sc.nextInt();
        if (list.contains(num)) {
            System.out.println(num + " exists in the list.");
        } else {
            System.out.println(num + " does not exist in the list.");
        }
    }
}
```

3. Remove Specific Element

Write a program to:

- **Create an ArrayList of Strings.**
- **Add 5 fruits.**
- **Remove a specific fruit by name.**
- **Display the updated list.**

```
import java.util.*;


public class RemoveFruit {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");
        fruits.add("Grapes");
        System.out.println("Fruits List: " + fruits);
        System.out.print("Enter fruit name to remove: ");
        String fruit = sc.nextLine();
        fruits.remove(fruit);
        System.out.println("Updated List: " + fruits);
    }
}
```

4. Sort Elements

Write a program to:

- **Create an ArrayList of integers.**
- **Add at least 7 random numbers.**
- **Sort the list in ascending order.**
- **Display the sorted list.**

```
import java.util.*;


public class SortArrayList {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(45);
        numbers.add(12);
        numbers.add(89);
        numbers.add(33);
        numbers.add(7);
        numbers.add(62);
        numbers.add(20);
        System.out.println("Original List: " + numbers);
        Collections.sort(numbers);
        System.out.println("Sorted List: " + numbers);
    }
}
```

---

5. Reverse the ArrayList

Write a program to:

· Create an ArrayList of characters.

· Add 5 characters.

· Reverse the list using Collections.reverse() and display it.

```
import java.util.*;

public class ReverseArrayList {
    public static void main(String[] args) {
        ArrayList<Character> chars = new ArrayList<>();
        chars.add('A');
        chars.add('B');
        chars.add('C');
        chars.add('D');
        chars.add('E');
        System.out.println("Original List: " + chars);
        Collections.reverse(chars);
        System.out.println("Reversed List: " + chars);
    }
}
```

6. Update an Element

Write a program to:

· Create an ArrayList of subjects.

· Replace one of the subjects (e.g., “Math” to “Statistics”).

· Print the list before and after the update.

```
import java.util.*;
```

```
public class UpdateElement {
    public static void main(String[] args) {
        ArrayList<String> subjects = new ArrayList<>();
        subjects.add("Math");
        subjects.add("Science");
        subjects.add("English");
        subjects.add("History");
        subjects.add("Geography");
        System.out.println("Before Update: " + subjects);
        int index = subjects.indexOf("Math");
        if (index != -1) {
            subjects.set(index, "Statistics");
        }
        System.out.println("After Update: " + subjects);
    }
}
```

7. Remove All Elements

Write a program to:

· Create an ArrayList of integers.

· Add multiple elements.

· Remove all elements using clear() method.

· Display the size of the list.

```
import java.util.*;

public class RemoveAllElements {
```

```
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);
        numbers.add(50);
        System.out.println("Before clear: " + numbers);
        numbers.clear();
        System.out.println("After clear: " + numbers);
        System.out.println("Size of list: " + numbers.size());
    }
}
```

8. Iterate using Iterator

Write a program to:

· Create an ArrayList of cities.

· Use Iterator to display each city.

```
import java.util.*;

public class IterateWithIterator {
    public static void main(String[] args) {
        ArrayList<String> cities = new ArrayList<>();
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");
        cities.add("Paris");
```

```
        cities.add("Sydney");
        Iterator<String> itr = cities.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

9. Store Custom Objects

Write a program to:

· Create a class Student with fields: id, name, and marks.

· Create an ArrayList of Student objects.

· Add at least 3 students.

· Display the details using a loop.

```
import java.util.*;

class Student {
    int id;
    String name;
    int marks;
    Student(int id, String name, int marks) {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }
}
```

```
public class StudentList {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student(1, "Alice", 85));
        students.add(new Student(2, "Bob", 90));
        students.add(new Student(3, "Charlie", 75));
        for (Student s : students) {
            System.out.println("ID: " + s.id + ", Name: " + s.name + ", Marks: " + s.marks);
        }
    }
}
```

10. Copy One ArrayList to Another

Write a program to:

· Create an ArrayList with some elements.

· Create a second ArrayList.

· Copy all elements from the first to the second using addAll() method.

```
import java.util.*;

public class CopyArrayList {
    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();
        list1.add("Red");
        list1.add("Green");
        list1.add("Blue");
        ArrayList<String> list2 = new ArrayList<>();
        list2.addAll(list1);
```

```
        System.out.println("List1: " + list1);
        System.out.println("List2: " + list2);
    }
}
```

1. Create and Display a LinkedList

Write a program to:

· Create a LinkedList of Strings.

· Add five colors to it.

· Display the list using a for-each loop.

```
import java.util.*;

public class LinkedListColors {
    public static void main(String[] args) {
        LinkedList<String> colors = new LinkedList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add("Yellow");
        colors.add("Purple");
        for (String color : colors) {
            System.out.println(color);
        }
    }
}
```

2. Add Elements at First and Last Position

Write a program to:

· Create a LinkedList of integers.

· Add elements at the beginning and at the end.

· Display the updated list.

```
import java.util.*;

public class AddFirstLast {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(20);
        numbers.add(30);
        numbers.addFirst(10);
        numbers.addLast(40);
        System.out.println(numbers);
    }
}
```

3. Insert Element at Specific Position

Write a program to:

· Create a LinkedList of names.

· Insert a name at index 2.

· Display the list before and after insertion.

```
import java.util.*;

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

```
        LinkedList<String> names = new LinkedList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        System.out.println("Before insertion: " + names);
        names.add(2, "David");
        System.out.println("After insertion: " + names);
    }
}
```

4. Remove Elements

Write a program to:

· Create a LinkedList of animal names.

· Remove the first and last elements.

· Remove a specific element by value.

· Display the list after each removal.

```
import java.util.*;

public class RemoveElements {
    public static void main(String[] args) {
        LinkedList<String> animals = new LinkedList<>();
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Elephant");
        animals.add("Lion");
        animals.add("Tiger");
        System.out.println("Original List: " + animals);
```

```
        animals.removeFirst();
        System.out.println("After removing first: " + animals);
        animals.removeLast();
        System.out.println("After removing last: " + animals);
        animals.remove("Elephant");
        System.out.println("After removing 'Elephant': " + animals);
    }
}
```

5. Search for an Element

Write a program to:

· Create a LinkedList of Strings.

· Ask the user for a string to search.

· Display if the string is found or not.

```
import java.util.*;

public class SearchLinkedList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedList<String> list = new LinkedList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        list.add("Date");
        list.add("Fig");
        System.out.print("Enter a string to search: ");
        String input = sc.nextLine();
```

```
        if (list.contains(input)) {
            System.out.println(input + " is found in the list.");
        } else {
            System.out.println(input + " is not found in the list.");
        }
    }
}
```

6. Iterate using ListIterator

Write a program to:

· Create a LinkedList of cities.

· Use ListIterator to display the list in both forward and reverse directions.

```
import java.util.*;

public class IterateListIterator {
    public static void main(String[] args) {
        LinkedList<String> cities = new LinkedList<>();
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");
        cities.add("Paris");
        cities.add("Sydney");
        ListIterator<String> itr = cities.listIterator();
        System.out.println("Forward Direction:");
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
```

```
        System.out.println("Reverse Direction:");
        while (itr.hasPrevious()) {
            System.out.println(itr.previous());
        }
    }
}
```

7. Sort a LinkedList

Write a program to:

· Create a LinkedList of integers.

· Add unsorted numbers.

· Sort the list using Collections.sort().

· Display the sorted list.

```
import java.util.*;

public class SortLinkedList {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(50);
        numbers.add(20);
        numbers.add(40);
        numbers.add(10);
        numbers.add(30);
        Collections.sort(numbers);
        System.out.println(numbers);
    }
}
```

8. Convert LinkedList to ArrayList

Write a program to:

· Create a LinkedList of Strings.

· Convert it into an ArrayList.

· Display both the LinkedList and ArrayList.

```
import java.util.*;

public class ConvertList {
    public static void main(String[] args) {
        LinkedList<String> linkedList = new LinkedList<>();
        linkedList.add("Red");
        linkedList.add("Green");
        linkedList.add("Blue");
        ArrayList<String> arrayList = new ArrayList<>(linkedList);
        System.out.println("LinkedList: " + linkedList);
        System.out.println("ArrayList: " + arrayList);
    }
}
```

9. Store Custom Objects in LinkedList

Write a program to:

· Create a class Book with fields: id, title, and author.

· Create a LinkedList of Book objects.

· Add 3 books and display their details using a loop.

```
import java.util.*;
```

```
class Book {
    int id;
    String title;
    String author;
    Book(int id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }
}

public class BookList {
    public static void main(String[] args) {
        LinkedList<Book> books = new LinkedList<>();
        books.add(new Book(1, "1984", "George Orwell"));
        books.add(new Book(2, "To Kill a Mockingbird", "Harper Lee"));
        books.add(new Book(3, "The Great Gatsby", "F. Scott Fitzgerald"));
        for (Book b : books) {
            System.out.println("ID: " + b.id + ", Title: " + b.title + ", Author: " + b.author);
        }
    }
}
```

10. Clone a LinkedList

Write a program to:

· Create a LinkedList of numbers.

· Clone it using the clone() method.

· Display both original and cloned lists.

```
import java.util.*;

public class CloneLinkedList {
    public static void main(String[] args) {
        LinkedList<Integer> originalList = new LinkedList<>();
        originalList.add(1);
        originalList.add(2);
        originalList.add(3);
        originalList.add(4);
        LinkedList<Integer> clonedList = (LinkedList<Integer>) originalList.clone();
        System.out.println("Original List: " + originalList);
        System.out.println("Cloned List: " + clonedList);
    }
}
```

Vector

· Create a Vector of integers and perform the following operations:

· Add 5 integers to the Vector.

· Insert an element at the 3rd position.

· Remove the 2nd element.

· Display the elements using Enumeration.

```
import java.util.*;

public class VectorOperations {
```

```
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<>();
        vector.add(10);
        vector.add(20);
        vector.add(30);
        vector.add(40);
        vector.add(50);
        vector.insertElementAt(25, 2);
        vector.remove(1);
        Enumeration<Integer> enumeration = vector.elements();
        while (enumeration.hasMoreElements()) {
            System.out.println(enumeration.nextElement());
        }
    }
}
```

· **Create a Vector of Strings and:**

· **Add at least 4 names.**

· **Check if a specific name exists in the vector.**

· **Replace one name with another.**

· **Clear all elements from the vector.**

```
import java.util.*;

public class VectorStrings {
    public static void main(String[] args) {
        Vector<String> names = new Vector<>();
        names.add("Alice");
```

```
        names.add("Bob");
        names.add("Charlie");
        names.add("Diana");
        System.out.println("Vector: " + names);
        System.out.println("Contains 'Bob'? " + names.contains("Bob"));
        int index = names.indexOf("Charlie");
        if (index != -1) {
            names.set(index, "Eve");
        }
        System.out.println("After replacement: " + names);
        names.clear();
        System.out.println("After clearing: " + names);
    }
}
```

· Write a program to:

· Copy all elements from one Vector to another Vector.

· Compare both vectors for equality.

· Write a method that takes a Vector<Integer> and returns the sum of all elements.

```
import java.util.*;

public class VectorCopyCompareSum {
    public static void main(String[] args) {
        Vector<Integer> vector1 = new Vector<>();
        vector1.add(10);
        vector1.add(20);
```

```
        vector1.add(30);
        vector1.add(40);

        Vector<Integer> vector2 = new Vector<>();
        vector2.addAll(vector1);

        System.out.println("Vector1: " + vector1);
        System.out.println("Vector2: " + vector2);

        System.out.println("Vectors equal? " + vector1.equals(vector2));

        int sum = sumVector(vector1);
        System.out.println("Sum of elements in Vector1: " + sum);
    }

    public static int sumVector(Vector<Integer> vector) {
        int sum = 0;
        for (int num : vector) {
            sum += num;
        }
        return sum;
    }
}
```

Stack

· **Create a Stack of integers and:**

· Push 5 elements.

· Pop the top element.

· Peek the current top.

· Check if the stack is empty.

```
import java.util.*;

public class StackOperations {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);
        System.out.println("Popped element: " + stack.pop());
        System.out.println("Current top element: " + stack.peek());
        System.out.println("Is stack empty? " + stack.isEmpty());
    }
}
```

· Reverse a string using Stack:

· Input a string from the user.

· Use a stack to reverse and print the string.

```
import java.util.*;

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

```
        System.out.print("Enter a string: ");
        String input = sc.nextLine();
        Stack<Character> stack = new Stack<>();
        for (char ch : input.toCharArray()) {
            stack.push(ch);
        }
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }
        System.out.println("Reversed string: " + reversed.toString());
    }
}
```

· Use Stack to check for balanced parentheses in an expression.

· Input: (a+b) * (c-d)

· Output: Valid or Invalid expression

```
import java.util.*;

public class BalancedParentheses {
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
```

```
                return false;
            }
            stack.pop();
        }
    }
    return stack.isEmpty();
  }

  public static void main(String[] args) {
    String expression = "(a+b) * (c-d)";
    if (isBalanced(expression)) {
      System.out.println("Valid expression");
    } else {
      System.out.println("Invalid expression");
    }
  }
}
```

HashSet

1. Create a HashSet of Strings:

o Add 5 different city names.

o Try adding a duplicate city and observe the output.

o Iterate using an Iterator and print each city.

```
import java.util.*;

public class HashSetExample {
```

```
    public static void main(String[] args) {
        HashSet<String> cities = new HashSet<>();
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");
        cities.add("Paris");
        cities.add("Sydney");
        cities.add("London"); // duplicate
        Iterator<String> itr = cities.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

2. Perform operations:

o Remove an element.

o Check if a city exists.

o Clear the entire HashSet.

```
import java.util.*;

public class HashSetOperations {
    public static void main(String[] args) {
        HashSet<String> cities = new HashSet<>();
        cities.add("New York");
        cities.add("London");
        cities.add("Tokyo");
```

```
        cities.add("Paris");
        cities.add("Sydney");
        cities.remove("Tokyo");
        System.out.println("Contains Paris? " + cities.contains("Paris"));
        cities.clear();
        System.out.println("HashSet after clear: " + cities);
    }
}
```

3. Write a method that takes a HashSet<Integer> and returns the maximum element.

```
import java.util.*;

public class MaxInHashSet {
    public static int getMax(HashSet<Integer> set) {
        int max = Integer.MIN_VALUE;
        for (int num : set) {
            if (num > max) {
                max = num;
            }
        }
        return max;
    }

    public static void main(String[] args) {
        HashSet<Integer> numbers = new HashSet<>(Arrays.asList(10, 45, 32, 74, 29));
        System.out.println("Maximum element: " + getMax(numbers));
    }
}
```

LinkedHashSet

1.Create a LinkedHashSet of Integers:

o Add numbers: 10, 5, 20, 15, 5.

o Print the elements and observe the order.

```
import java.util.*;

public class LinkedHashSetExample {
    public static void main(String[] args) {
        LinkedHashSet<Integer> numbers = new LinkedHashSet<>();
        numbers.add(10);
        numbers.add(5);
        numbers.add(20);
        numbers.add(15);
        numbers.add(5);
        System.out.println(numbers);
    }
}
```

2. Create a LinkedHashSet of custom objects (e.g., Student with id and name):

o Override hashCode() and equals() properly.

o Add at least 3 Student objects.

o Try adding a duplicate student and check if it gets added.

```
import java.util.*;

class Student {
    int id;
```

```
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Student other = (Student) obj;
        return id == other.id && Objects.equals(name, other.name);
    }

    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "'}";
    }
}
```

```
public class LinkedHashSetCustomObjects {
    public static void main(String[] args) {
        LinkedHashSet<Student> students = new LinkedHashSet<>();
        students.add(new Student(1, "Alice"));
        students.add(new Student(2, "Bob"));
        students.add(new Student(3, "Charlie"));
        students.add(new Student(2, "Bob")); // duplicate
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

3. Write a program to:

o Merge two LinkedHashSets and print the result.

```
import java.util.*;

public class MergeLinkedHashSets {
    public static void main(String[] args) {
        LinkedHashSet<Integer> set1 = new LinkedHashSet<>();
        set1.add(10);
        set1.add(20);
        set1.add(30);

        LinkedHashSet<Integer> set2 = new LinkedHashSet<>();
        set2.add(25);
        set2.add(20);
        set2.add(35);
```

```
        set1.addAll(set2);

        System.out.println("Merged LinkedHashSet: " + set1);
    }
}
```

TreeSet

1. Create a TreeSet of Strings:

o Add 5 country names in random order.

o Print the sorted list of countries using TreeSet.

```
import java.util.*;

public class TreeSetExample {
    public static void main(String[] args) {
        TreeSet<String> countries = new TreeSet<>();
        countries.add("India");
        countries.add("Brazil");
        countries.add("Australia");
        countries.add("Canada");
        countries.add("Japan");
        System.out.println(countries);
    }
}
```

2. Create a TreeSet of Integers:

o Add some numbers and print the first and last elements.

o Find the elements lower than and higher than a given number using lower() and higher() methods.

```
import java.util.*;


public class TreeSetOperations {
    public static void main(String[] args) {
        TreeSet<Integer> numbers = new TreeSet<>();
        numbers.add(50);
        numbers.add(20);
        numbers.add(40);
        numbers.add(10);
        numbers.add(30);


        System.out.println("TreeSet: " + numbers);
        System.out.println("First (lowest) element: " + numbers.first());
        System.out.println("Last (highest) element: " + numbers.last());


        int givenNumber = 25;
        System.out.println("Element lower than " + givenNumber + ": " +
numbers.lower(givenNumber));
        System.out.println("Element higher than " + givenNumber + ": " +
numbers.higher(givenNumber));
    }
}
```

3. Create a TreeSet with a custom comparator:

o Sort strings in reverse alphabetical order using Comparator.

```
import java.util.*;
```

```
public class TreeSetCustomComparator {
    public static void main(String[] args) {
        TreeSet<String> countries = new TreeSet<>(new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                return s2.compareTo(s1); // reverse alphabetical order
            }
        });
        countries.add("India");
        countries.add("Brazil");
        countries.add("Australia");
        countries.add("Canada");
        countries.add("Japan");
        System.out.println(countries);
    }
}
```

Queue

1. Bank Queue Simulation:

o Create a queue of customer names using Queue<String>.

o Add 5 customers to the queue.

o Serve (remove) customers one by one and print the queue after each removal.

```
import java.util.*;

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

```
        Queue<String> customers = new LinkedList<>();
        customers.add("Alice");
        customers.add("Bob");
        customers.add("Charlie");
        customers.add("Diana");
        customers.add("Ethan");

        while (!customers.isEmpty()) {
            System.out.println("Serving customer: " + customers.peek());
            customers.remove();
            System.out.println("Queue after serving: " + customers);
        }
    }
}
```

2. Task Manager:

o Queue of tasks (String values).

o Add tasks, peek at the next task, and poll completed tasks.

```
import java.util.*;

public class TaskManager {
    public static void main(String[] args) {
        Queue<String> tasks = new LinkedList<>();
        tasks.add("Write report");
        tasks.add("Email client");
        tasks.add("Prepare presentation");
        tasks.add("Schedule meeting");
```

```
        System.out.println("Next task: " + tasks.peek());

        while (!tasks.isEmpty()) {
            System.out.println("Completing task: " + tasks.poll());
            System.out.println("Remaining tasks: " + tasks);
        }
    }
}
```

3. Write a method:

o That takes a queue of integers and returns a list of even numbers.

```
import java.util.*;

public class EvenNumbersFromQueue {
    public static List<Integer> getEvenNumbers(Queue<Integer> queue) {
        List<Integer> evenNumbers = new ArrayList<>();
        for (int num : queue) {
            if (num % 2 == 0) {
                evenNumbers.add(num);
            }
        }
        return evenNumbers;
    }

    public static void main(String[] args) {
        Queue<Integer> numbers = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8));
        List<Integer> evens = getEvenNumbers(numbers);
```

```
        System.out.println("Even numbers: " + evens);
    }
}
```

PriorityQueue

1. Hospital Emergency Queue:

o Create a class Patient with fields: name and severityLevel (int).

o Use PriorityQueue<Patient> with a comparator to serve the most critical patients first (highest severityLevel).

```
import java.util.*;

class Patient {
    String name;
    int severityLevel;

    Patient(String name, int severityLevel) {
        this.name = name;
        this.severityLevel = severityLevel;
    }

    @Override
    public String toString() {
        return "Patient{name='" + name + "', severityLevel=" + severityLevel + '}';
    }
}

public class HospitalEmergencyQueue {
```

```
    public static void main(String[] args) {
        PriorityQueue<Patient> queue = new PriorityQueue<>(new Comparator<Patient>() {
            @Override
            public int compare(Patient p1, Patient p2) {
                return Integer.compare(p2.severityLevel, p1.severityLevel); // higher severity first
            }
        });

        queue.add(new Patient("Alice", 5));
        queue.add(new Patient("Bob", 3));
        queue.add(new Patient("Charlie", 8));
        queue.add(new Patient("Diana", 1));

        while (!queue.isEmpty()) {
            System.out.println("Serving: " + queue.poll());
        }
    }
}
```

2. Print Jobs Priority:

o Add different print jobs (String) with priority levels.

o Use PriorityQueue to simulate serving high-priority jobs before others.

```
import java.util.*;

class PrintJob {
    String jobName;
    int priority;
```

```
    PrintJob(String jobName, int priority) {
        this.jobName = jobName;
        this.priority = priority;
    }

    @Override
    public String toString() {
        return "PrintJob{name='" + jobName + "', priority=" + priority + '}';
    }
}

public class PrintJobsPriority {
    public static void main(String[] args) {
        PriorityQueue<PrintJob> queue = new PriorityQueue<>(new Comparator<PrintJob>() {
            public int compare(PrintJob p1, PrintJob p2) {
                return Integer.compare(p2.priority, p1.priority); // higher priority first
            }
        });

        queue.add(new PrintJob("Document1", 2));
        queue.add(new PrintJob("Photo", 5));
        queue.add(new PrintJob("Spreadsheet", 3));
        queue.add(new PrintJob("Presentation", 4));

        while (!queue.isEmpty()) {
            System.out.println("Processing: " + queue.poll());
        }
```

```
    }
}
```

3. Write a method:

o To merge two PriorityQueue<Integer> and return a sorted merged queue.

```
import java.util.*;


public class PriorityQueueMerge {
    public static PriorityQueue<Integer> mergeQueues(PriorityQueue<Integer> q1,
PriorityQueue<Integer> q2) {
        PriorityQueue<Integer> merged = new PriorityQueue<>();
        merged.addAll(q1);
        merged.addAll(q2);
        return merged;
    }

    public static void main(String[] args) {
        PriorityQueue<Integer> queue1 = new PriorityQueue<>(Arrays.asList(5, 1, 3, 7));
        PriorityQueue<Integer> queue2 = new PriorityQueue<>(Arrays.asList(6, 2, 8, 4));
        PriorityQueue<Integer> mergedQueue = mergeQueues(queue1, queue2);
        System.out.println("Merged and sorted queue:");
        while (!mergedQueue.isEmpty()) {
            System.out.print(mergedQueue.poll() + " ");
        }
    }
}
```

Deque

1. Palindrome Checker:

o Input a string and check if it is a palindrome using a Deque<Character>.

```
import java.util.*;

public class PalindromeChecker {
    public static boolean isPalindrome(String str) {
        Deque<Character> deque = new LinkedList<>();
        for (char ch : str.toCharArray()) {
            deque.addLast(ch);
        }
        while (deque.size() > 1) {
            if (!deque.removeFirst().equals(deque.removeLast())) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine();
        if (isPalindrome(input)) {
            System.out.println(input + " is a palindrome.");
        } else {
            System.out.println(input + " is not a palindrome.");
```

```
        }
    }
}
```

2. Double-ended Order System:

o Add items from front and rear.

o Remove items from both ends.

o Display contents of the deque after each operation.

```
import java.util.*;

public class DoubleEndedOrderSystem {
    public static void main(String[] args) {
        Deque<String> deque = new LinkedList<>();



        deque.addFirst("Order1");
        System.out.println("After addFirst(Order1): " + deque);
        deque.addLast("Order2");
        System.out.println("After addLast(Order2): " + deque);
        deque.addFirst("Order3");
        System.out.println("After addFirst(Order3): " + deque);
        deque.addLast("Order4");
        System.out.println("After addLast(Order4): " + deque);



        String removedFront = deque.removeFirst();
```

```
        System.out.println("After removeFirst() - removed: " + removedFront + ", deque: " +
deque);
        String removedRear = deque.removeLast();
        System.out.println("After removeLast() - removed: " + removedRear + ", deque: " +
deque);
    }
}
```

3. Browser History Simulation:

o Implement browser back and forward navigation using two deques.

```
import java.util.*;

public class BrowserHistorySimulation {
    private Deque<String> backStack = new LinkedList<>();
    private Deque<String> forwardStack = new LinkedList<>();
    private String currentPage;

    public void visit(String url) {
        if (currentPage != null) {
            backStack.push(currentPage);
        }
        currentPage = url;
        forwardStack.clear();
        System.out.println("Visited: " + currentPage);
    }

    public void back() {
        if (!backStack.isEmpty()) {
```

```
        forwardStack.push(currentPage);
        currentPage = backStack.pop();
        System.out.println("Back to: " + currentPage);
    } else {
        System.out.println("No pages in back history.");
    }
}

public void forward() {
    if (!forwardStack.isEmpty()) {
        backStack.push(currentPage);
        currentPage = forwardStack.pop();
        System.out.println("Forward to: " + currentPage);
    } else {
        System.out.println("No pages in forward history.");
    }
}

public static void main(String[] args) {
    BrowserHistorySimulation browser = new BrowserHistorySimulation();
    browser.visit("google.com");
    browser.visit("openai.com");
    browser.visit("github.com");

    browser.back();
    browser.back();
    browser.forward();
    browser.visit("stackoverflow.com");
```

```
        browser.back();
        browser.forward();
        browser.forward();
    }
}
```