Collections

List(ArrayList)

**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.

Program:

import java.util.\*;

public class CheckNumber {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>(Arrays.asList(10, 20, 30, 40, 50));

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number to check: ");

int num = sc.nextInt();

if (numbers.contains(num)) {

System.out.println(num + " exists in the list.");

} else {

System.out.println(num + " does not exist in the list.");

}

sc.close();

}

}

**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.

Program:

import java.util.\*;

public class RemoveFruit {

public static void main(String[] args) {

ArrayList<String> fruits = new ArrayList<>(Arrays.asList("Apple", "Banana", "Mango", "Orange", "Grapes"));

System.out.println("Fruits list: " + fruits);

fruits.remove("Mango");

System.out.println("After removing Mango: " + 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.

Program:

import java.util.\*;

public class SortArrayList {

public static void main(String[] args) {

ArrayList<Integer> nums = new ArrayList<>(Arrays.asList(45, 12, 78, 34, 56, 23, 90));

Collections.sort(nums);

System.out.println("Sorted list: " + nums);

}

}

**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.

Program:

import java.util.\*;

public class ReverseList {

public static void main(String[] args) {

ArrayList<Character> chars = new ArrayList<>(Arrays.asList('A', 'B', 'C', 'D', 'E'));

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.

Program:

import java.util.\*;

public class UpdateElement {

public static void main(String[] args) {

ArrayList<String> subjects = new ArrayList<>(Arrays.asList("Math", "Physics", "Chemistry"));

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.

Program:

import java.util.\*;

public class ClearList {

public static void main(String[] args) {

ArrayList<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

nums.clear();

System.out.println("List size after clear: " + nums.size());

}

}

**8. Iterate using Iterator**

Write a program to:

* Create an ArrayList of cities.
* Use Iterator to display each city.

Program:

import java.util.\*;

public class IteratorExample {

public static void main(String[] args) {

ArrayList<String> cities = new ArrayList<>(Arrays.asList("Delhi", "Mumbai", "Chennai", "Kolkata"));

Iterator<String> it = cities.iterator();

while (it.hasNext()) {

System.out.println(it.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.

Program:

import java.util.\*;

class Student {

int id;

String name;

double marks;

Student(int id, String name, double 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, "Ashwin", 85.5));

students.add(new Student(2, "Priya", 90.0));

students.add(new Student(3, "Ravi", 78.2));

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 CopyList {

public static void main(String[] args) {

ArrayList<String> list1 = new ArrayList<>(Arrays.asList("A", "B", "C"));

ArrayList<String> list2 = new ArrayList<>();

list2.addAll(list1);

System.out.println("Original List: " + list1);

System.out.println("Copied List: " + list2);

}

}

List(LinkedList)

**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.

Program:

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.

Program:

import java.util.\*;

public class LinkedListAddFirstLast {

public static void main(String[] args) {

LinkedList<Integer> nums = new LinkedList<>();

nums.add(10);

nums.add(20);

nums.add(30);

nums.addFirst(5);

nums.addLast(40);

System.out.println("Updated list: " + nums);

}

}

**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.

Program:

import java.util.\*;

public class LinkedListInsert {

public static void main(String[] args) {

LinkedList<String> names = new LinkedList<>(Arrays.asList("John", "Mike", "Sara", "Emma"));

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.

Program:

import java.util.\*;

public class LinkedListRemove {

public static void main(String[] args) {

LinkedList<String> animals = new LinkedList<>(Arrays.asList("Dog", "Cat", "Elephant", "Tiger", "Lion"));

System.out.println("Original: " + 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.

Program:

import java.util.\*;

public class LinkedListSearch {

public static void main(String[] args) {

LinkedList<String> list = new LinkedList<>(Arrays.asList("Apple", "Banana", "Mango", "Grapes"));

Scanner sc = new Scanner(System.in);

System.out.print("Enter fruit to search: ");

String fruit = sc.nextLine();

if (list.contains(fruit)) {

System.out.println(fruit + " found in the list.");

} else {

System.out.println(fruit + " not found in the list.");

}

sc.close();

}

}

**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.

Program:

import java.util.\*;

public class LinkedListListIterator {

public static void main(String[] args) {

LinkedList<String> cities = new LinkedList<>(Arrays.asList("Delhi", "Mumbai", "Chennai", "Kolkata"));

ListIterator<String> it = cities.listIterator();

System.out.println("Forward:");

while (it.hasNext()) {

System.out.println(it.next());

}

System.out.println("Backward:");

while (it.hasPrevious()) {

System.out.println(it.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.

Program:

import java.util.\*;

public class LinkedListSort {

public static void main(String[] args) {

LinkedList<Integer> nums = new LinkedList<>(Arrays.asList(34, 12, 78, 45, 23));

Collections.sort(nums);

System.out.println("Sorted list: " + nums);

}

}

**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.

Program:

import java.util.\*;

public class LinkedListToArrayList {

public static void main(String[] args) {

LinkedList<String> linkedList = new LinkedList<>(Arrays.asList("One", "Two", "Three"));

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.

Program:

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 LinkedListBooks {

public static void main(String[] args) {

LinkedList<Book> books = new LinkedList<>();

books.add(new Book(1, "Java Basics", "James Gosling"));

books.add(new Book(2, "Effective Java", "Joshua Bloch"));

books.add(new Book(3, "Clean Code", "Robert C. Martin"));

for (Book b : books) {

System.out.println(b.id + " - " + b.title + " by " + 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.

Program:

import java.util.\*;

public class LinkedListClone {

public static void main(String[] args) {

LinkedList<Integer> original = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));

@SuppressWarnings("unchecked")

LinkedList<Integer> clone = (LinkedList<Integer>) original.clone();

System.out.println("Original: " + original);

System.out.println("Cloned: " + clone);

}

}

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.

Program:

import java.util.\*;

public class VectorIntegersDemo {

public static void main(String[] args) {

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

// Add 5 integers

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

numbers.add(50);

// Insert at 3rd position (index 2)

numbers.add(2, 25);

// Remove 2nd element (index 1)

numbers.remove(1);

// Display using Enumeration

Enumeration<Integer> en = numbers.elements();

System.out.println("Vector elements:");

while (en.hasMoreElements()) {

System.out.println(en.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.

Program:

import java.util.\*;

public class VectorStringsDemo {

public static void main(String[] args) {

Vector<String> names = new Vector<>();

// Add names

names.add("Alice");

names.add("Bob");

names.add("Charlie");

names.add("David");

// Check if a specific name exists

String searchName = "Charlie";

System.out.println("Contains " + searchName + "? " + names.contains(searchName));

// Replace "Bob" with "Brian"

names.set(1, "Brian");

System.out.println("After replacement: " + names);

// Clear all elements

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**.

Program:

import java.util.\*;

public class VectorCopyCompare {

public static void main(String[] args) {

Vector<String> vector1 = new Vector<>();

vector1.add("Red");

vector1.add("Green");

vector1.add("Blue");

// Copy to another vector

Vector<String> vector2 = new Vector<>(vector1);

// Compare both vectors

boolean areEqual = vector1.equals(vector2);

System.out.println("Vector 1: " + vector1);

System.out.println("Vector 2: " + vector2);

System.out.println("Are they equal? " + areEqual);

}

}

**Stack**

* Understand how to use the Stack class for LIFO (Last In, First Out) operations.
* **Create a Stack of integers** and:
* Push 5 elements.
* Pop the top element.
* Peek the current top.
* Check if the stack is empty.

Program:

import java.util.\*;

public class StackIntegerDemo {

public static void main(String[] args) {

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

// Push 5 elements

for (int i = 1; i <= 5; i++) {

stack.push(i \* 10);

}

System.out.println("Stack after pushes: " + stack);

// Pop the top element

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

System.out.println("Stack after pop: " + stack);

// Peek the top element

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

// Check if stack is empty

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.
* **Use Stack to check for balanced parentheses** in an expression.
* Input: (a+b) \* (c-d)
* Output: Valid or Invalid expression

Program:

import java.util.\*;

public class StackStringReverse {

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<>();

// Push each character

for (char ch : input.toCharArray()) {

stack.push(ch);

}

// Pop to reverse

StringBuilder reversed = new StringBuilder();

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

System.out.println("Reversed string: " + reversed);

sc.close();

}

}

* **Convert a decimal number to binary using Stack**.

**Program**:

**import java.util.\*;**

**public class BalancedParentheses {**

**public static boolean isBalanced(String expr) {**

**Stack<Character> stack = new Stack<>();**

**for (char ch : expr.toCharArray()) {**

**if (ch == '(' || ch == '{' || ch == '[') {**

**stack.push(ch);**

**} else if (ch == ')' || ch == '}' || ch == ']') {**

**if (stack.isEmpty()) return false;**

**char top = stack.pop();**

**if ((ch == ')' && top != '(') ||**

**(ch == '}' && top != '{') ||**

**(ch == ']' && top != '[')) {**

**return false;**

**}**

**}**

**}**

**return stack.isEmpty();**

**}**

**public static void main(String[] args) {**

**String expression = "(a+b) \* (c-d)";**

**System.out.println("Expression: " + expression);**

**System.out.println(isBalanced(expression) ? "Valid expression" : "Invalid expression");**

**}**

**}**

HashSet

1. **Create a HashSet of Strings**:
   * Add 5 different city names.
   * Try adding a duplicate city and observe the output.
   * Iterate using an Iterator and print each city.

Program:

import java.util.\*;

public class HashSetCities {

public static void main(String[] args) {

HashSet<String> cities = new HashSet<>();

// Add cities

cities.add("New York");

cities.add("London");

cities.add("Paris");

cities.add("Tokyo");

cities.add("Sydney");

// Try adding duplicate

boolean added = cities.add("Paris");

System.out.println("Was 'Paris' added again? " + added);

// Iterate using Iterator

Iterator<String> it = cities.iterator();

System.out.println("Cities in the HashSet:");

while (it.hasNext()) {

System.out.println(it.next());

}

}

}

1. **Perform operations**:
   * Remove an element.
   * Check if a city exists.
   * Clear the entire HashSet.

Program:

import java.util.\*;

public class HashSetOperations {

public static void main(String[] args) {

HashSet<String> cities = new HashSet<>();

cities.add("Delhi");

cities.add("Mumbai");

cities.add("Chennai");

cities.add("Kolkata");

System.out.println("Initial HashSet: " + cities);

// Remove element

cities.remove("Mumbai");

System.out.println("After removing 'Mumbai': " + cities);

// Check existence

System.out.println("Contains 'Delhi'? " + cities.contains("Delhi"));

// Clear all

cities.clear();

System.out.println("After clearing: " + cities);

}

}

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

**Program:**

import java.util.\*;

public class HashSetMaxValue {

public static int getMaxValue(HashSet<Integer> set) {

return Collections.max(set);

}

public static void main(String[] args) {

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

numbers.add(10);

numbers.add(25);

numbers.add(5);

numbers.add(40);

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

System.out.println("Maximum value: " + getMaxValue(numbers));

}

}

**LinkedHashSet**

**1.Create a LinkedHashSet of Integers**:

* + Add numbers: 10, 5, 20, 15, 5.
  + Print the elements and observe the order.

Program:

import java.util.\*;

public class LinkedHashSetIntegers {

public static void main(String[] args) {

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

// Add elements

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

numbers.add(5); // Duplicate

// Print elements

System.out.println("LinkedHashSet elements: " + numbers);

}

}

1. **Create a LinkedHashSet of custom objects (e.g., Student with id and name)**:
   * Override hashCode() and equals() properly.
   * Add at least 3 Student objects.
   * Try adding a duplicate student and check if it gets added.

Program:

import java.util.\*;

public class LinkedHashSetIntegers {

public static void main(String[] args) {

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

// Add elements

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

numbers.add(5); // Duplicate

// Print elements

System.out.println("LinkedHashSet elements: " + numbers);

}

}

1. **Write a program** to:
   * Merge two LinkedHashSets and print the result.

Program:

import java.util.\*;

public class LinkedHashSetIntegers {

public static void main(String[] args) {

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

// Add elements

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

numbers.add(5); // Duplicate

// Print elements

System.out.println("LinkedHashSet elements: " + numbers);

}

}

**TreeSet**

**1. Create a TreeSet of Strings**:

* + Add 5 country names in random order.
  + Print the sorted list of countries using TreeSet.

Program:

import java.util.\*;

public class TreeSetCountries {

public static void main(String[] args) {

TreeSet<String> countries = new TreeSet<>();

// Add countries in random order

countries.add("India");

countries.add("Australia");

countries.add("Canada");

countries.add("Brazil");

countries.add("Denmark");

// TreeSet stores in sorted (ascending) order

System.out.println("Sorted Countries: " + countries);

}

}

1. **Create a TreeSet of Integers**:
   * Add some numbers and print the first and last elements.
   * Find the elements lower than and higher than a given number using lower() and higher() methods.

Program:

import java.util.\*;

public class TreeSetIntegers {

public static void main(String[] args) {

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

numbers.add(10);

numbers.add(5);

numbers.add(20);

numbers.add(15);

numbers.add(25);

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

System.out.println("First element: " + numbers.first());

System.out.println("Last element: " + numbers.last());

int checkNum = 15;

System.out.println("Lower than " + checkNum + ": " + numbers.lower(checkNum));

System.out.println("Higher than " + checkNum + ": " + numbers.higher(checkNum));

}

}

1. **Create a TreeSet with a custom comparator**:
   * Sort strings in **reverse alphabetical order** using Comparator.

Program:

import java.util.\*;

public class TreeSetReverseOrder {

public static void main(String[] args) {

// Custom Comparator for reverse alphabetical order

TreeSet<String> cities = new TreeSet<>(Comparator.reverseOrder());

cities.add("Mumbai");

cities.add("Delhi");

cities.add("Kolkata");

cities.add("Chennai");

cities.add("Bangalore");

System.out.println("Cities in reverse order: " + cities);

}

}

Queue

1. **Bank Queue Simulation**:
   * Create a queue of customer names using Queue<String>.
   * Add 5 customers to the queue.
   * Serve (remove) customers one by one and print the queue after each removal.

Program:

import java.util.\*;

public class BankQueueSimulation {

public static void main(String[] args) {

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

// Add customers

bankQueue.add("Alice");

bankQueue.add("Bob");

bankQueue.add("Charlie");

bankQueue.add("David");

bankQueue.add("Eve");

System.out.println("Initial Queue: " + bankQueue);

// Serve customers

while (!bankQueue.isEmpty()) {

String served = bankQueue.poll(); // removes first element

System.out.println("Serving: " + served);

System.out.println("Remaining Queue: " + bankQueue);

}

}

}

1. **Task Manager**:
   * Queue of tasks (String values).
   * Add tasks, peek at the next task, and poll completed tasks.

Program:

import java.util.\*;

public class TaskManager {

public static void main(String[] args) {

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

// Add tasks

tasks.add("Write report");

tasks.add("Send email");

tasks.add("Fix bug #123");

tasks.add("Attend meeting");

// Peek at next task

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

// Poll tasks as they are completed

while (!tasks.isEmpty()) {

System.out.println("Completed: " + tasks.poll());

System.out.println("Remaining Tasks: " + tasks);

}

}

}

1. **Write a method**:
   * That takes a queue of integers and returns a list of even numbers.

Program:

import java.util.\*;

public class EvenNumbersFromQueue {

public static void main(String[] args) {

Queue<Integer> numbers = new LinkedList<>();

numbers.add(10);

numbers.add(15);

numbers.add(22);

numbers.add(33);

numbers.add(44);

List<Integer> evens = getEvenNumbers(numbers);

System.out.println("Even Numbers: " + evens);

}

// Method to return even numbers from queue

public static List<Integer> getEvenNumbers(Queue<Integer> queue) {

List<Integer> evenList = new ArrayList<>();

for (int num : queue) {

if (num % 2 == 0) {

evenList.add(num);

}

}

return evenList;

}

}

**PriorityQueue**

1. **Hospital Emergency Queue**:
   * Create a class Patient with fields: name and severityLevel (int).
   * Use PriorityQueue<Patient> with a comparator to serve the most critical patients first (highest severityLevel).

Program:

import java.util.\*;

class Patient {

String name;

int severityLevel;

public Patient(String name, int severityLevel) {

this.name = name;

this.severityLevel = severityLevel;

}

@Override

public String toString() {

return name + " (Severity: " + severityLevel + ")";

}

}

public class HospitalEmergencyQueue {

public static void main(String[] args) {

PriorityQueue<Patient> hospitalQueue = new PriorityQueue<>(

(p1, p2) -> Integer.compare(p2.severityLevel, p1.severityLevel) // Higher severity first

);

hospitalQueue.add(new Patient("Alice", 3));

hospitalQueue.add(new Patient("Bob", 5));

hospitalQueue.add(new Patient("Charlie", 2));

while (!hospitalQueue.isEmpty()) {

System.out.println("Serving: " + hospitalQueue.poll());

}

}

}

1. **Print Jobs Priority**:
   * Add different print jobs (String) with priority levels.
   * Use PriorityQueue to simulate serving high-priority jobs before others.

Program:

import java.util.\*;

class PrintJob {

String jobName;

int priority;

public PrintJob(String jobName, int priority) {

this.jobName = jobName;

this.priority = priority;

}

@Override

public String toString() {

return jobName + " (Priority: " + priority + ")";

}

}

public class PrintJobsPriority {

public static void main(String[] args) {

PriorityQueue<PrintJob> printQueue = new PriorityQueue<>(

(j1, j2) -> Integer.compare(j2.priority, j1.priority)

);

printQueue.add(new PrintJob("Document A", 2));

printQueue.add(new PrintJob("Document B", 5));

printQueue.add(new PrintJob("Document C", 3));

while (!printQueue.isEmpty()) {

System.out.println("Printing: " + printQueue.poll());

}

}

}

1. **Write a method**:
   * To merge two PriorityQueue<Integer> and return a sorted merged queue.

Program:

import java.util.\*;

public class MergePriorityQueues {

public static void main(String[] args) {

PriorityQueue<Integer> q1 = new PriorityQueue<>(Arrays.asList(1, 5, 3));

PriorityQueue<Integer> q2 = new PriorityQueue<>(Arrays.asList(4, 2, 6));

PriorityQueue<Integer> mergedQueue = mergeQueues(q1, q2);

System.out.println("Merged Sorted Queue: " + mergedQueue);

}

public static PriorityQueue<Integer> mergeQueues(PriorityQueue<Integer> q1, PriorityQueue<Integer> q2) {

PriorityQueue<Integer> merged = new PriorityQueue<>(q1);

merged.addAll(q2);

return merged;

}

}

**Deque**

1. **Palindrome Checker**:
   * Input a string and check if it is a palindrome using a Deque<Character>.

Program:

import java.util.\*;

public class PalindromeChecker {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter string: ");

String input = sc.nextLine().toLowerCase().replaceAll("\\s+", "");

Deque<Character> deque = new LinkedList<>();

for (char c : input.toCharArray()) {

deque.add(c);

}

boolean isPalindrome = true;

while (deque.size() > 1) {

if (!deque.pollFirst().equals(deque.pollLast())) {

isPalindrome = false;

break;

}

}

System.out.println("Is Palindrome? " + isPalindrome);

sc.close();

}

}

1. **Double-ended Order System**:
   * Add items from front and rear.
   * Remove items from both ends.
   * Display contents of the deque after each operation.

Program:

import java.util.\*;

public class DoubleEndedOrderSystem {

public static void main(String[] args) {

Deque<String> orders = new LinkedList<>();

orders.addFirst("Order1");

orders.addLast("Order2");

System.out.println("After adding from both ends: " + orders);

orders.addFirst("Order0");

orders.addLast("Order3");

System.out.println("After adding more: " + orders);

orders.pollFirst();

System.out.println("After removing from front: " + orders);

orders.pollLast();

System.out.println("After removing from rear: " + orders);

}

}

1. **Browser History Simulation**:
   * Implement browser back and forward navigation using two deques.

Program:

import java.util.\*;

public class BrowserHistorySimulation {

public static void main(String[] args) {

Deque<String> backStack = new LinkedList<>();

Deque<String> forwardStack = new LinkedList<>();

String currentPage = "Home";

System.out.println("Current Page: " + currentPage);

// Visit new pages

currentPage = visitPage("Page1", currentPage, backStack, forwardStack);

currentPage = visitPage("Page2", currentPage, backStack, forwardStack);

currentPage = visitPage("Page3", currentPage, backStack, forwardStack);

// Go back

currentPage = goBack(backStack, forwardStack, currentPage);

currentPage = goBack(backStack, forwardStack, currentPage);

// Go forward

currentPage = goForward(backStack, forwardStack, currentPage);

}

static String visitPage(String page, String currentPage, Deque<String> back, Deque<String> forward) {

back.push(currentPage);

forward.clear();

System.out.println("Visiting: " + page);

return page;

}

static String goBack(Deque<String> back, Deque<String> forward, String currentPage) {

if (!back.isEmpty()) {

forward.push(currentPage);

currentPage = back.pop();

System.out.println("Back to: " + currentPage);

}

return currentPage;

}

static String goForward(Deque<String> back, Deque<String> forward, String currentPage) {

if (!forward.isEmpty()) {

back.push(currentPage);

currentPage = forward.pop();

System.out.println("Forward to: " + currentPage);

}

return currentPage;

}

}