**Collections**

List(ArrayList)

**1. 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.

ANS:

import java.util.ArrayList;

import java.util.Scanner;

public class SearchElement {

public static void main(String[] args) {

ArrayList<Integer> list = new ArrayList<>();

list.add(10);

list.add(25);

list.add(30);

list.add(45);

list.add(50);

Scanner scanner = new Scanner(System.in);

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

int num = scanner.nextInt();

if (list.contains(num)) {

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

} else {

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

}

scanner.close();

}

}

2. **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.

ANS:  
import java.util.ArrayList;

public class RemoveFruit {

public static void main(String[] args) {

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

fruits.add("Apple");

fruits.add("Banana");

fruits.add("Mango");

fruits.add("Orange");

fruits.add("Grapes");

String fruitToRemove = "Mango";

fruits.remove(fruitToRemove);

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

}

}

3. **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.

ANS:  
import java.util.ArrayList;

import java.util.Collections;

public class SortElements {

public static void main(String[] args) {

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

numbers.add(42);

numbers.add(5);

numbers.add(17);

numbers.add(99);

numbers.add(3);

numbers.add(65);

numbers.add(28);

Collections.sort(numbers);

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

}

}

4. **. Reverse the ArrayList**

Write a program to:

* Create an ArrayList of characters.
* Add 5 characters.

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

ANS:

import java.util.ArrayList;

import java.util.Collections;

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');

Collections.reverse(chars);

System.out.println("Reversed list: " + chars);

}

}

5. **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.

ANS:

import java.util.ArrayList;

public class UpdateElement {

public static void main(String[] args) {

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

subjects.add("Physics");

subjects.add("Chemistry");

subjects.add("Math");

subjects.add("Biology");

subjects.add("English");

System.out.println("Before update: " + subjects);

int index = subjects.indexOf("Math");

if (index != -1) {

subjects.set(index, "Statistics");

}

System.out.println("After update: " + subjects);

}

}

6. **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.

ANS:

import java.util.ArrayList;

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);

numbers.clear();

System.out.println("Size after clear: " + numbers.size());

}

}

7. **Iterate using Iterator**

Write a program to:

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

ANS:

import java.util.ArrayList;

import java.util.Iterator;

public class IterateUsingIterator {

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> iterator = cities.iterator();

while (iterator.hasNext()) {

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

}

}

}

8. **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.

ANS:  
import java.util.ArrayList;

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 String toString() {

return "ID: " + id + ", Name: " + name + ", Marks: " + marks;

}

}

public class StudentList {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

students.add(new Student(1, "Alice", 85.5));

students.add(new Student(2, "Bob", 92.0));

students.add(new Student(3, "Charlie", 78.0));

for (Student s : students) {

System.out.println(s);

}

}

}

9. **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.

ANS:

import java.util.ArrayList;

public class CopyArrayList {

public static void main(String[] args) {

ArrayList<String> list1 = new ArrayList<>();

list1.add("Java");

list1.add("Python");

list1.add("C++");

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

list2.addAll(list1);

System.out.println("List1: " + list1);

System.out.println("List2: " + 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.

ANS:

import java.util.LinkedList;

public class LinkedListExample {

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.

ANS:  
import java.util.LinkedList;

public class AddFirstLast {

public static void main(String[] args) {

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

numbers.add(20);

numbers.add(30);

numbers.add(40);

numbers.addFirst(10);

numbers.addLast(50);

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.

ANS:

import java.util.LinkedList;

public class InsertAtPosition {

public static void main(String[] args) {

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

names.add("Alice");

names.add("Bob");

names.add("Charlie");

names.add("David");

System.out.println("Before insertion: " + names);

names.add(2, "Eve");

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.

ANS;

import java.util.LinkedList;

public class RemoveElements {

public static void main(String[] args) {

LinkedList<String> animals = new LinkedList<>();

animals.add("Lion");

animals.add("Tiger");

animals.add("Elephant");

animals.add("Giraffe");

animals.add("Zebra");

animals.removeFirst();

animals.removeLast();

System.out.println("After removal: " + 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.

ANS;

import java.util.LinkedList;

import java.util.Scanner;

public class SearchLinkedList {

public static void main(String[] args) {

LinkedList<String> list = new LinkedList<>();

list.add("Apple");

list.add("Banana");

list.add("Mango");

list.add("Orange");

list.add("Grapes");

Scanner scanner = new Scanner(System.in);

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

String input = scanner.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.");

}

scanner.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.

ANS:

import java.util.LinkedList;

import java.util.ListIterator;

public class ListIteratorDemo {

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> iterator = cities.listIterator();

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

while (iterator.hasNext()) {

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

}

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

while (iterator.hasPrevious()) {

System.out.println(iterator.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.

ANS:

import java.util.Collections;

import java.util.LinkedList;

public class SortLinkedList {

public static void main(String[] args) {

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

numbers.add(25);

numbers.add(5);

numbers.add(18);

numbers.add(42);

numbers.add(12);

Collections.sort(numbers);

System.out.println("Sorted list: " + 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.

ANS:

import java.util.ArrayList;

import java.util.LinkedList;

public class LinkedListToArrayList {

public static void main(String[] args) {

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

linkedList.add("Red");

linkedList.add("Green");

linkedList.add("Blue");

linkedList.add("Yellow");

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.

ANS:

import java.util.LinkedList;

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 String toString() {

return "ID: " + id + ", Title: " + title + ", 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 book : books) {

System.out.println(book);

}

}

}

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.

ANS:

import java.util.LinkedList;

public class CloneLinkedList {

public static void main(String[] args) {

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

originalList.add(10);

originalList.add(20);

originalList.add(30);

originalList.add(40);

@SuppressWarnings("unchecked")

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.
* **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.
* **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**.

**ANS:**

import java.util.Enumeration;

import java.util.Vector;

public class VectorDemo {

public static int sumVector(Vector<Integer> v) {

int sum = 0;

for (int num : v) {

sum += num;

}

return sum;

}

public static void main(String[] args) {

// Integer Vector

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

intVector.add(10);

intVector.add(20);

intVector.add(30);

intVector.add(40);

intVector.add(50);

intVector.insertElementAt(25, 2); // 3rd position (index 2)

intVector.removeElementAt(1); // remove 2nd element (index 1)

System.out.println("Integer Vector elements using Enumeration:");

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

while (en.hasMoreElements()) {

System.out.print(en.nextElement() + " ");

}

System.out.println();

// String Vector

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

strVector.add("Alice");

strVector.add("Bob");

strVector.add("Charlie");

strVector.add("David");

System.out.println("Contains 'Bob'? " + strVector.contains("Bob"));

int index = strVector.indexOf("Charlie");

if (index != -1) {

strVector.set(index, "Eve");

}

System.out.println("String Vector after replacement: " + strVector);

strVector.clear();

System.out.println("String Vector after clear: " + strVector);

// Copy and compare vectors

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

copyVector.addAll(intVector);

System.out.println("intVector equals copyVector? " + intVector.equals(copyVector));

System.out.println("Sum of intVector elements: " + sumVector(intVector));

}

}

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.
* **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

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

ANS:  
import java.util.Scanner;

import java.util.Stack;

public class StackDemo {

public static void main(String[] args) {

// Part 1: Stack of integers

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("Top element: " + stack.peek());

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

// Part 2: Reverse a string using Stack

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a string to reverse: ");

String input = scanner.nextLine();

Stack<Character> charStack = new Stack<>();

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

charStack.push(c);

}

StringBuilder reversed = new StringBuilder();

while (!charStack.isEmpty()) {

reversed.append(charStack.pop());

}

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

// Part 3: Check balanced parentheses

System.out.print("Enter expression to check parentheses: ");

String expr = scanner.nextLine();

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

// Part 4: Decimal to binary conversion using stack

System.out.print("Enter decimal number to convert to binary: ");

int decimal = scanner.nextInt();

System.out.println("Binary: " + decimalToBinary(decimal));

scanner.close();

}

static boolean isBalanced(String expr) {

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

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

if (c == '(') {

stack.push(c);

} else if (c == ')') {

if (stack.isEmpty()) return false;

stack.pop();

}

}

return stack.isEmpty();

}

static String decimalToBinary(int num) {

if (num == 0) return "0";

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

while (num > 0) {

stack.push(num % 2);

num /= 2;

}

StringBuilder binary = new StringBuilder();

while (!stack.isEmpty()) {

binary.append(stack.pop());

}

return binary.toString();

}

}

**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.
2. **Perform operations**:
   * Remove an element.
   * Check if a city exists.
   * Clear the entire HashSet.
3. **Write a method** that takes a HashSet<Integer> and returns the maximum element.

ANS:

import java.util.HashSet;

import java.util.Iterator;

public class HashSetDemo {

public static int maxElement(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<String> cities = new HashSet<>();

cities.add("New York");

cities.add("London");

cities.add("Tokyo");

cities.add("Paris");

cities.add("Sydney");

boolean added = cities.add("Tokyo"); // duplicate

System.out.println("Trying to add duplicate 'Tokyo': " + added);

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

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

while (iterator.hasNext()) {

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

}

cities.remove("Paris");

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

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

cities.clear();

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

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

numbers.add(10);

numbers.add(25);

numbers.add(7);

numbers.add(50);

System.out.println("Max element in numbers: " + maxElement(numbers));

}

}

**LinkedHashSet**

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

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

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

ANS:

import java.util.LinkedHashSet;

import java.util.Objects;

class Student {

int id;

String name;

Student(int id, String name) {

this.id = id;

this.name = name;

}

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Student)) return false;

Student s = (Student) o;

return id == s.id && Objects.equals(name, s.name);

}

public int hashCode() {

return Objects.hash(id, name);

}

public String toString() {

return "Student{id=" + id + ", name='" + name + "'}";

}

}

public class LinkedHashSetDemo {

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); // duplicate ignored

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

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(1, "Alice")); // duplicate

System.out.println("Students in LinkedHashSet:");

for (Student s : students) {

System.out.println(s);

}

LinkedHashSet<Integer> set1 = new LinkedHashSet<>();

set1.add(1);

set1.add(2);

set1.add(3);

LinkedHashSet<Integer> set2 = new LinkedHashSet<>();

set2.add(3);

set2.add(4);

set2.add(5);

set1.addAll(set2);

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

}

}

**TreeSet**

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

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

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

ANS:

import java.util.Comparator;

import java.util.TreeSet;

public class TreeSetDemo {

public static void main(String[] args) {

// 1. TreeSet of Strings (countries)

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

countries.add("India");

countries.add("USA");

countries.add("Brazil");

countries.add("China");

countries.add("Australia");

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

// 2. TreeSet of Integers

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

numbers.add(50);

numbers.add(20);

numbers.add(70);

numbers.add(10);

numbers.add(40);

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

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

System.out.println("Lower than 40: " + numbers.lower(40));

System.out.println("Higher than 40: " + numbers.higher(40));

// 3. TreeSet with custom comparator (reverse alphabetical)

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

reverseSet.add("Apple");

reverseSet.add("Orange");

reverseSet.add("Banana");

reverseSet.add("Grape");

reverseSet.add("Mango");

System.out.println("Reverse alphabetical order: " + reverseSet);

}

}

**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.
2. **Task Manager**:
   * Queue of tasks (String values).
   * Add tasks, peek at the next task, and poll completed tasks.
3. **Write a method**:
   * That takes a queue of integers and returns a list of even numbers.

ANS:

import java.util.\*;

public class QueueDemo {

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

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

for (int num : queue) {

if (num % 2 == 0) {

evens.add(num);

}

}

return evens;

}

public static void main(String[] args) {

// 1. Bank Queue Simulation

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

bankQueue.add("Alice");

bankQueue.add("Bob");

bankQueue.add("Charlie");

bankQueue.add("David");

bankQueue.add("Eve");

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

while (!bankQueue.isEmpty()) {

String served = bankQueue.poll();

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

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

}

// 2. Task Manager

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

tasks.add("Task1");

tasks.add("Task2");

tasks.add("Task3");

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

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

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

// 3. Method to get even numbers

Queue<Integer> numbers = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5, 6));

List<Integer> evens = getEvenNumbers(numbers);

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

}

}

**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).
2. **Print Jobs Priority**:
   * Add different print jobs (String) with priority levels.
   * Use PriorityQueue to simulate serving high-priority jobs before others.
3. **Write a method**:
   * To merge two PriorityQueue<Integer> and return a sorted merged queue.

ANS:

import java.util.\*;

class Patient {

String name;

int severityLevel;

Patient(String name, int severityLevel) {

this.name = name;

this.severityLevel = severityLevel;

}

public String toString() {

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

}

}

class PrintJob {

String jobName;

int priority;

PrintJob(String jobName, int priority) {

this.jobName = jobName;

this.priority = priority;

}

public String toString() {

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

}

}

public class PriorityQueueDemo {

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) {

// 1. Hospital Emergency Queue

PriorityQueue<Patient> patients = new PriorityQueue<>(

(p1, p2) -> Integer.compare(p2.severityLevel, p1.severityLevel)

);

patients.add(new Patient("Alice", 5));

patients.add(new Patient("Bob", 8));

patients.add(new Patient("Charlie", 3));

System.out.println("Hospital Emergency Queue:");

while (!patients.isEmpty()) {

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

}

// 2. Print Jobs Priority

PriorityQueue<PrintJob> printJobs = new PriorityQueue<>(

Comparator.comparingInt(job -> job.priority)

);

printJobs.add(new PrintJob("Doc1", 3));

printJobs.add(new PrintJob("Image1", 1));

printJobs.add(new PrintJob("Doc2", 2));

System.out.println("\nPrint Jobs served in order:");

while (!printJobs.isEmpty()) {

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

}

// 3. Merge two PriorityQueues of integers

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

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

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

System.out.println("\nMerged PriorityQueue:");

while (!mergedQueue.isEmpty()) {

System.out.print(mergedQueue.poll() + " ");

}

}

}

**Deque**

1. Palindrome Checker:
   * Input a string and check if it is a palindrome using a Deque<Character>.
2. Double-ended Order System:
   * Add items from front and rear.
   * Remove items from both ends.
   * Display contents of the deque after each operation.
3. Browser History Simulation:
   * Implement browser back and forward navigation using two deques.

ANS:

import java.util.ArrayDeque;

import java.util.Deque;

import java.util.Scanner;

public class DequeDemo {

// 1. Palindrome checker

public static boolean isPalindrome(String s) {

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

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

deque.addLast(Character.toLowerCase(c));

}

while (deque.size() > 1) {

if (!deque.removeFirst().equals(deque.removeLast())) {

return false;

}

}

return true;

}

// 2. Double-ended Order System

public static void orderSystem() {

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

orders.addFirst("Burger");

orders.addLast("Fries");

System.out.println("Orders after adding front and rear: " + orders);

orders.addFirst("Soda");

orders.addLast("Salad");

System.out.println("Orders after more additions: " + orders);

orders.removeFirst();

System.out.println("Orders after removing first: " + orders);

orders.removeLast();

System.out.println("Orders after removing last: " + orders);

}

// 3. Browser History Simulation

public static class BrowserHistory {

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

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

String currentPage = null;

public void visit(String url) {

if (currentPage != null) {

backStack.push(currentPage);

forwardStack.clear();

}

currentPage = url;

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 page to go back to.");

}

}

public void forward() {

if (!forwardStack.isEmpty()) {

backStack.push(currentPage);

currentPage = forwardStack.pop();

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

} else {

System.out.println("No page to go forward to.");

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Palindrome checker

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

String input = scanner.nextLine();

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

// Order system

System.out.println("\nOrder system demo:");

orderSystem();

// Browser history

System.out.println("\nBrowser history simulation:");

BrowserHistory browser = new BrowserHistory();

browser.visit("google.com");

browser.visit("github.com");

browser.visit("stackoverflow.com");

browser.back();

browser.back();

browser.forward();

browser.forward();

scanner.close();

}

}