# Day – 8

## List (ArrayList)

1. Search an Element in ArrayList.Write a program to:

Create an ArrayList of integers.

Ask the user to enter a number.

Check if the number exists in the list.

Code:

import java.util.ArrayList;  
import java.util.Scanner;  
  
public class ArrayListSearch {  
 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);  
   
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter a number to search: ");  
 int num = scanner.nextInt();  
   
 if (numbers.contains(num)) {  
 System.out.println(num + " exists in the list.");  
 } else {  
 System.out.println(num + " does not exist in the list.");  
 }  
 }  
}

2. Remove Specific 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.

Code:

import java.util.ArrayList;  
  
public class ArrayListRemove {  
 public static void main(String[] args) {  
 ArrayList<String> fruits = new ArrayList<>();  
 fruits.add("Apple");  
 fruits.add("Banana");  
 fruits.add("Orange");  
 fruits.add("Mango");  
 fruits.add("Grapes");  
   
 System.out.println("Original list: " + fruits);  
 fruits.remove("Orange");  
 System.out.println("After removal: " + fruits);  
 }  
}

3. Sort Elements in ArrayList.Write a program to:

Create an ArrayList of integers.

Ask the user to enter a number.

Check if the number exists in the list.

Code:

import java.util.ArrayList;  
import java.util.Collections;  
  
public class ArrayListSort {  
 public static void main(String[] args) {  
 ArrayList<Integer> numbers = new ArrayList<>();  
 numbers.add(34);  
 numbers.add(12);  
 numbers.add(78);  
 numbers.add(5);  
 numbers.add(42);  
 numbers.add(90);  
 numbers.add(23);  
   
 System.out.println("Original list: " + numbers);  
 Collections.sort(numbers);  
 System.out.println("Sorted list: " + numbers);  
 }  
}

4. Reverse the ArrayList.Write a program to:

Create an ArrayList of integers.

Ask the user to enter a number.

Check if the number exists in the list.

Code:

import java.util.ArrayList;  
import java.util.Collections;  
  
public class ArrayListReverse {  
 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);  
 }  
}

5. Update an Element in ArrayList.Write a program to:

Create an ArrayList of integers.

Ask the user to enter a number.

Check if the number exists in the list.

Code:

import java.util.ArrayList;  
  
public class ArrayListUpdate {  
public static void main(String[] args) {  
 ArrayList<String> subjects = new ArrayList<>();  
 subjects.add("Math");  
 subjects.add("Science");  
 subjects.add("History");  
 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 from ArrayList

Code:

import java.util.ArrayList;  
public class ArrayListClear {  
 public static void main(String[] args) {  
 ArrayList<Integer> numbers = new ArrayList<>();  
 numbers.add(1);  
 numbers.add(2);  
 numbers.add(3);  
 numbers.add(4);  
   
 System.out.println("Original size: " + numbers.size());  
 numbers.clear();  
 System.out.println("After clear, size: " + numbers.size());  
 }  
}

7. Iterate using Iterator.Write a program to:

Create an ArrayList of cities.

Use Iterator to display each city.

Code:

import java.util.ArrayList;  
import java.util.Iterator;  
  
public class ArrayListIterator {  
public static void main(String[] args) {  
 ArrayList<String> cities = new ArrayList<>();  
 cities.add("New York");  
 cities.add("London");  
 cities.add("Tokyo");  
 cities.add("Paris");  
   
 Iterator<String> iterator = cities.iterator();  
 while (iterator.hasNext()) {  
 System.out.println(iterator.next());  
 }  
 }  
}

8. Store Custom Objects in ArrayList.Write a program to:

Create an ArrayList of cities.

Use Iterator to display each city.

Code:

import java.util.ArrayList;  
  
class Student {  
 int id;  
 String name;  
 double marks;  
   
 public Student(int id, String name, double marks) {  
 this.id = id;  
 this.name = name;  
 this.marks = marks;  
 }  
}  
  
public class ArrayListCustomObjects {  
 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", 76.0));  
 students.add(new Student(3, "Charlie", 92.3));  
   
 for (Student student : students) {  
 System.out.println("ID: " + student.id + ", Name: " + student.name + ", Marks: " + student.marks);  
 }  
 }  
}

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.

Code:

import java.util.ArrayList;  
  
public class ArrayListCopy {  
public static void main(String[] args) {  
 ArrayList<String> list1 = new ArrayList<>();  
 list1.add("One");  
 list1.add("Two");  
 list1.add("Three");  
   
 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.

Code:

import java.util.LinkedList;  
  
public class LinkedListCreate {  
 public static void main(String[] args) {  
 LinkedList<String> colors = new LinkedList<>();  
 colors.add("Red");  
 colors.add("Green");  
 colors.add("Blue");  
 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 Strings.

Add five colors to it.

Display the list using a for-each loop.

Code:

import java.util.LinkedList;  
  
public class LinkedListAddFirstLast {  
 public static void main(String[] args) {  
 LinkedList<Integer> numbers = new LinkedList<>();  
 numbers.add(2);  
 numbers.add(3);  
 numbers.add(4);  
   
 numbers.addFirst(1);  
 numbers.addLast(5);  
   
 System.out.println(numbers);  
 }  
}

3. Insert Element at Specific Position.Write a program to:

Create a LinkedList of Strings.

Add five colors to it.

Display the list using a for-each loop.

Code:

import java.util.LinkedList;  
  
public class LinkedListInsert {  
 public static void main(String[] args) {  
 LinkedList<String> names = new LinkedList<>();  
 names.add("Alice");  
 names.add("Bob");  
 names.add("Charlie");  
 names.add("Eve");  
   
 System.out.println("Before insertion: " + names);  
 names.add(2, "David");  
 System.out.println("After insertion: " + names);  
 }  
}

4. Remove Elements from LinkedList.Write a program to:

Create a LinkedList of Strings.

Add five colors to it.

Display the list using a for-each loop.

Code:

import java.util.LinkedList;  
  
public class LinkedListRemove {  
 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");  
   
 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("Tiger");  
 System.out.println("After removing Tiger: " + animals);  
 }  
}

5. Search for an Element in LinkedList.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.

Code:

import java.util.LinkedList;  
import java.util.Scanner;  
  
public class LinkedListSearch {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.add("Apple");  
 list.add("Banana");  
 list.add("Cherry");  
 list.add("Date");  
   
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter a string to search: ");  
 String searchStr = scanner.nextLine();  
   
 if (list.contains(searchStr)) {  
 System.out.println(searchStr + " found in the list.");  
 } else {  
 System.out.println(searchStr + " not found in the list.");  
 }  
 }  
}

6. Iterate using ListIterator.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.

Code:

import java.util.LinkedList;  
import java.util.ListIterator;  
  
public class LinkedListListIterator {  
 public static void main(String[] args) {  
 LinkedList<String> cities = new LinkedList<>();  
 cities.add("New York");  
 cities.add("London");  
 cities.add("Tokyo");  
 cities.add("Paris");  
   
 ListIterator<String> iterator = cities.listIterator();  
 System.out.println("Forward direction:");  
 while (iterator.hasNext()) {  
 System.out.println(iterator.next());  
 }  
   
 System.out.println("\nBackward direction:");  
 while (iterator.hasPrevious()) {  
 System.out.println(iterator.previous());  
 }  
 }  
}

7. Sort a LinkedList.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.

Code:

import java.util.LinkedList;  
import java.util.Collections;  
  
public class LinkedListSort {  
 public static void main(String[] args) {  
 LinkedList<Integer> numbers = new LinkedList<>();  
 numbers.add(34);  
 numbers.add(12);  
 numbers.add(78);  
 numbers.add(5);  
   
 System.out.println("Before sorting: " + numbers);  
 Collections.sort(numbers);  
 System.out.println("After sorting: " + 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.

Code:

import java.util.ArrayList;  
import java.util.LinkedList;  
  
public class LinkedListToArrayList {  
 public static void main(String[] args) {  
 LinkedList<String> linkedList = new LinkedList<>();  
 linkedList.add("One");  
 linkedList.add("Two");  
 linkedList.add("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 LinkedList of Strings.

Convert it into an ArrayList.

Display both the LinkedList and ArrayList.

Code:

import java.util.LinkedList;  
  
class Book {  
 int id;  
 String title;  
 String author;  
   
 public Book(int id, String title, String author) {  
 this.id = id;  
 this.title = title;  
 this.author = author;  
 }  
}  
  
public class LinkedListCustomObjects {  
 public static void main(String[] args) {  
 LinkedList<Book> books = new LinkedList<>();  
 books.add(new Book(1, "Java Programming", "John Doe"));  
 books.add(new Book(2, "Data Structures", "Jane Smith"));  
 books.add(new Book(3, "Algorithms", "Bob Johnson"));  
   
 for (Book book : books) {  
 System.out.println("ID: " + book.id + ", Title: " + book.title + ", Author: " + book.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.

Code:

import java.util.LinkedList;  
  
public class LinkedListClone {  
 public static void main(String[] args) {  
 LinkedList<Integer> original = new LinkedList<>();  
 original.add(10);  
 original.add(20);  
 original.add(30);  
   
 @SuppressWarnings("unchecked")  
 LinkedList<Integer> cloned = (LinkedList<Integer>) original.clone();  
   
 System.out.println("Original list: " + original);  
 System.out.println("Cloned list: " + cloned);  
 }  
}

## Vector

1.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.

Code:  
import java.util.Enumeration;  
import java.util.Vector;  
public class VectorOperations {  
 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.add(2, 25); // Insert at 3rd position  
 intVector.remove(1); // Remove 2nd element  
   
 System.out.println("Integer Vector:");  
 Enumeration<Integer> intEnum = intVector.elements();  
 while (intEnum.hasMoreElements()) {  
 System.out.println(intEnum.nextElement());  
 }  
   
 // String Vector  
 Vector<String> strVector = new Vector<>();  
 strVector.add("Alice");  
 strVector.add("Bob");  
 strVector.add("Charlie");  
 strVector.add("David");  
   
 System.out.println("\nContains 'Bob'? " + strVector.contains("Bob"));  
 strVector.set(1, "Robert"); // Replace Bob with Robert  
 System.out.println("After replacement: " + strVector);  
 strVector.clear();  
 System.out.println("After clear, size: " + strVector.size());  
  
 // Copy Vector  
 Vector<Integer> vector1 = new Vector<>();  
 vector1.add(1);  
 vector1.add(2);  
 vector1.add(3);  
   
 Vector<Integer> vector2 = new Vector<>();  
 vector2.addAll(vector1);  
   
 System.out.println("\nVector1: " + vector1);  
 System.out.println("Vector2: " + vector2);  
 System.out.println("Are equal? " + vector1.equals(vector2));  
   
 // Sum of Vector elements  
 System.out.println("Sum of elements: " + sumVectorElements(vector1));  
 }  
   
 public static int sumVectorElements(Vector<Integer> vector) {  
 int sum = 0;  
 for (int num : vector) {  
 sum += num;  
 }  
 return sum;  
 }  
}

## Stack

1.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.

Code:

import java.util.Scanner;  
import java.util.Stack;  
public class StackOperations {  
 public static void main(String[] args) {  
 // Basic Stack operations  
Stack<Integer> stack = new Stack<>();  
 stack.push(10);  
 stack.push(20);  
 stack.push(30);  
 stack.push(40);  
 stack.push(50);  
   
 System.out.println("Stack: " + stack);  
 System.out.println("Popped: " + stack.pop());  
 System.out.println("Current top: " + stack.peek());  
 System.out.println("Is empty? " + stack.isEmpty());  
   
 // Reverse a string using Stack  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("\nEnter a string to reverse: ");  
 String input = scanner.nextLine();  
 String reversed = reverseString(input);  
 System.out.println("Reversed string: " + reversed);  
   
 // Check balanced parentheses  
 System.out.print("\nEnter an expression to check parentheses: ");  
 String expr = scanner.nextLine();  
 System.out.println("Parentheses balanced? " + isBalanced(expr));  
   
 // Decimal to binary using Stack  
 System.out.print("\nEnter a decimal number: ");  
 int decimal = scanner.nextInt();  
 System.out.println("Binary: " + decimalToBinary(decimal));  
 }  
   
 public static String reverseString(String input) {  
 Stack<Character> stack = new Stack<>();  
 for (char c : input.toCharArray()) {  
 stack.push(c);  
 }  
 StringBuilder reversed = new StringBuilder();  
 while (!stack.isEmpty()) {  
 reversed.append(stack.pop());  
 }  
 return reversed.toString();  
 }  
   
 public 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();  
 }  
   
 public static String decimalToBinary(int decimal) {  
 Stack<Integer> stack = new Stack<>();  
 while (decimal > 0) {  
 stack.push(decimal % 2);  
 decimal /= 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.

Code:

import java.util.HashSet;  
import java.util.Iterator;  
  
public class HashSetOperations {  
 public static void main(String[] args) {  
 // Create HashSet of Strings  
HashSet<String> cities = new HashSet<>();  
 cities.add("New York");  
 cities.add("London");  
 cities.add("Tokyo");  
 cities.add("Paris");  
 cities.add("Berlin");  
   
 System.out.println("Original set: " + cities);  
 System.out.println("Adding duplicate 'London': " + cities.add("London"));  
   
 // Iterate using Iterator  
 System.out.println("\nCities:");  
 Iterator<String> iterator = cities.iterator();  
 while (iterator.hasNext()) {  
 System.out.println(iterator.next());  
 }  
   
 // Perform operations  
 cities.remove("Tokyo");  
 System.out.println("\nAfter removing Tokyo: " + cities);  
 System.out.println("Contains Paris? " + cities.contains("Paris"));  
 cities.clear();  
 System.out.println("After clear, size: " + cities.size());  
   
 // Find maximum element in HashSet of Integers  
 HashSet<Integer> numbers = new HashSet<>();  
 numbers.add(10);  
 numbers.add(5);  
 numbers.add(20);  
 numbers.add(15);  
 System.out.println("\nNumbers: " + numbers);  
 System.out.println("Maximum: " + findMax(numbers));  
 }  
   
 public static int findMax(HashSet<Integer> set) {  
 int max = Integer.MIN\_VALUE;  
 for (int num : set) {  
 if (num > max) {  
 max = num;  
 }  
 }  
 return max;  
 }  
}

## LinkedHashSet

1. Create and Display LinkedHashSet

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

Print the elements and observe the order.

Code:

import java.util.LinkedHashSet;  
  
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); // Duplicate  
   
 System.out.println("LinkedHashSet: " + numbers);  
 }  
}

2. 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.

Code:

import java.util.LinkedHashSet;  
import java.util.Objects;  
  
class Student {  
 int id;  
 String name;  
   
 public Student(int id, String name) {  
 this.id = id;  
 this.name = name;  
 }  
   
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Student student = (Student) o;  
 return id == student.id && Objects.equals(name, student.name);  
 }  
   
 @Override  
 public int hashCode() {  
 return Objects.hash(id, 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"));  
   
 // Try adding duplicate  
 Student duplicate = new Student(1, "Alice");  
 System.out.println("Adding duplicate student: " + students.add(duplicate));  
 System.out.println("Students: " + students);  
 }  
}

3. Write a program to:

Merge two LinkedHashSets and print the result.

Code:

import java.util.LinkedHashSet;  
  
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(20);  
 set2.add(40);  
 set2.add(50);  
   
 LinkedHashSet<Integer> merged = new LinkedHashSet<>(set1);  
 merged.addAll(set2);  
   
 System.out.println("Set 1: " + set1);  
 System.out.println("Set 2: " + set2);  
 System.out.println("Merged set: " + merged);  
 }  
}

## TreeSet

1. . Create a TreeSet of Strings:

Add 5 country names in random order.

Print the sorted list of countries using TreeSet.

Code:

import java.util.TreeSet;  
  
public class TreeSetExample {  
 public static void main(String[] args) {  
 TreeSet<String> countries = new TreeSet<>();  
 countries.add("India");  
 countries.add("USA");  
 countries.add("UK");  
 countries.add("Australia");  
 countries.add("Canada");  
   
 System.out.println("Sorted countries: " + countries);  
 }  
}

2. 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.

Code:

import java.util.TreeSet;  
  
public class TreeSetIntegerOperations {  
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("Numbers: " + numbers);  
 System.out.println("First element: " + numbers.first());  
 System.out.println("Last element: " + numbers.last());  
   
 int givenNumber = 15;  
 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:

Sort strings in reverse alphabetical order using Comparator.

Code:

import java.util.Comparator;  
import java.util.TreeSet;  
  
public class TreeSetCustomComparator {  
 public static void main(String[] args) {  
 TreeSet<String> reverseSorted = new TreeSet<>(Comparator.reverseOrder());  
 reverseSorted.add("Apple");  
 reverseSorted.add("Banana");  
 reverseSorted.add("Cherry");  
 reverseSorted.add("Date");  
   
 System.out.println("Reverse sorted strings: " + reverseSorted);  
 }  
}

## 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.

Code:

import java.util.LinkedList;  
import java.util.Queue;  
  
public class BankQueueSimulation {  
 public static void main(String[] args) {  
 Queue<String> customerQueue = new LinkedList<>();  
 customerQueue.add("Customer 1");  
 customerQueue.add("Customer 2");  
 customerQueue.add("Customer 3");  
 customerQueue.add("Customer 4");  
 customerQueue.add("Customer 5");  
   
 while (!customerQueue.isEmpty()) {  
 System.out.println("Serving: " + customerQueue.poll());  
 System.out.println("Remaining queue: " + customerQueue);  
 }  
 }  
}

2. Task Manager

Queue of tasks (String values).

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

Code:

import java.util.LinkedList;  
import java.util.Queue;  
  
public class TaskManager {  
 public static void main(String[] args) {  
 Queue<String> tasks = new LinkedList<>();  
 tasks.add("Write report");  
 tasks.add("Review code");  
 tasks.add("Fix bugs");  
   
 System.out.println("Next task: " + tasks.peek());  
 System.out.println("Completed task: " + tasks.poll());  
 System.out.println("Remaining tasks: " + tasks);  
 }  
}

3. Write a method:

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

Code:

import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Queue;  
  
public class EvenNumbersFromQueue {  
 public static void main(String[] args) {  
 Queue<Integer> numbers = new LinkedList<>();  
 numbers.add(1);  
 numbers.add(2);  
 numbers.add(3);  
 numbers.add(4);  
 numbers.add(5);  
 numbers.add(6);  
   
 List<Integer> evens = getEvenNumbers(numbers);  
 System.out.println("Original queue: " + numbers);  
 System.out.println("Even numbers: " + evens);  
 }  
   
 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;  
 }  
}

## 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).

Code:

import java.util.PriorityQueue;  
  
class Patient implements Comparable<Patient> {  
 String name;  
 int severityLevel;  
   
 public Patient(String name, int severityLevel) {  
 this.name = name;  
 this.severityLevel = severityLevel;  
 }  
   
 @Override  
 public int compareTo(Patient other) {  
 // Higher severity comes first  
 return Integer.compare(other.severityLevel, this.severityLevel);  
 }  
   
 @Override  
 public String toString() {  
 return name + " (Severity: " + severityLevel + ")";  
 }  
}  
  
public class HospitalEmergencyQueue {  
 public static void main(String[] args) {  
 PriorityQueue<Patient> emergencyQueue = new PriorityQueue<>();  
 emergencyQueue.add(new Patient("John", 3));  
 emergencyQueue.add(new Patient("Alice", 5));  
 emergencyQueue.add(new Patient("Bob", 1));  
 emergencyQueue.add(new Patient("Eve", 4));  
   
 System.out.println("Patients in order of priority:");  
 while (!emergencyQueue.isEmpty()) {  
 System.out.println("Treating: " + emergencyQueue.poll());  
 }  
 }  
}

2. Print Jobs Priority:

Add different print jobs (String) with priority levels.

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

Code:

import java.util.PriorityQueue;  
  
class PrintJob implements Comparable<PrintJob> {  
 String document;  
 int priority;  
   
 public PrintJob(String document, int priority) {  
 this.document = document;  
 this.priority = priority;  
 }  
   
 @Override  
 public int compareTo(PrintJob other) {  
 // Higher priority comes first  
 return Integer.compare(other.priority, this.priority);  
 }  
   
 @Override  
 public String toString() {  
 return document + " (Priority: " + priority + ")";  
 }  
}  
  
public class PrintJobPriority {  
 public static void main(String[] args) {  
 PriorityQueue<PrintJob> printQueue = new PriorityQueue<>();  
 printQueue.add(new PrintJob("Report.pdf", 2));  
 printQueue.add(new PrintJob("Invoice.doc", 5));  
 printQueue.add(new PrintJob("Presentation.pptx", 3));  
   
 System.out.println("Print jobs in order of priority:");  
 while (!printQueue.isEmpty()) {  
 System.out.println("Printing: " + printQueue.poll());  
 }  
 }  
}

3. Write a method:

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

Code:

import java.util.PriorityQueue;  
  
public class MergePriorityQueues {  
public static void main(String[] args) {  
 PriorityQueue<Integer> pq1 = new PriorityQueue<>();  
 pq1.add(10);  
 pq1.add(20);  
 pq1.add(30);  
   
 PriorityQueue<Integer> pq2 = new PriorityQueue<>();  
 pq2.add(15);  
 pq2.add(25);  
 pq2.add(35);  
   
 PriorityQueue<Integer> merged = mergeQueues(pq1, pq2);  
   
 System.out.println("Merged queue (sorted):");  
 while (!merged.isEmpty()) {  
 System.out.println(merged.poll());  
 }  
 }  
   
 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>.

Code:

import java.util.ArrayDeque;  
import java.util.Deque;  
import java.util.Scanner;  
  
public class PalindromeChecker {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter a string to check for palindrome: ");  
 String input = scanner.nextLine();  
   
 if (isPalindrome(input)) {  
 System.out.println(input + " is a palindrome.");  
 } else {  
 System.out.println(input + " is not a palindrome.");  
 }  
 }  
   
 public static boolean isPalindrome(String str) {  
 Deque<Character> deque = new ArrayDeque<>();  
 String cleanStr = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();  
   
 for (char c : cleanStr.toCharArray()) {  
 deque.addLast(c);  
 }  
   
 while (deque.size() > 1) {  
 if (!deque.removeFirst().equals(deque.removeLast())) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

2. Double-ended Order System:

Add items from front and rear.

Remove items from both ends.

Display contents of the deque after each operation.

Code:

import java.util.ArrayDeque;  
import java.util.Deque;  
  
public class DoubleEndedOrderSystem {  
 public static void main(String[] args) {  
 Deque<String> orders = new ArrayDeque<>();  
   
 // Add items from front and rear  
 orders.addFirst("Order 1 (VIP)");  
 orders.addLast("Order 2");  
 orders.addFirst("Order 3 (VIP)");  
 orders.addLast("Order 4");  
   
 System.out.println("Current orders: " + orders);  
   
 // Remove items from both ends  
 System.out.println("Processing: " + orders.removeFirst());  
 System.out.println("Processing: " + orders.removeLast());  
   
 System.out.println("Remaining orders: " + orders);  
 }  
}

3. Browser History Simulation:

Implement browser back and forward navigation using two deques.

Code:

import java.util.ArrayDeque;  
import java.util.Deque;  
import java.util.Scanner;  
  
public class BrowserHistory {  
 private static Deque<String> backHistory = new ArrayDeque<>();  
 private static Deque<String> forwardHistory = new ArrayDeque<>();  
 private static String currentPage = null;  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
   
 while (true) {  
 System.out.println("\nCurrent page: " + (currentPage != null ? currentPage : "None"));  
 System.out.println("1. Visit new page");  
 System.out.println("2. Go back");  
 System.out.println("3. Go forward");  
 System.out.println("4. Exit");  
 System.out.print("Choose option: ");  
 int choice = scanner.nextInt();  
scanner.nextLine(); // consume newline  
   
 switch (choice) {  
 case 1:  
 System.out.print("Enter URL: ");  
 String url = scanner.nextLine();  
 visitPage(url);  
 break;  
 case 2:  
 goBack();  
 break;  
 case 3:  
 goForward();  
 break;  
 case 4:  
 return;  
 default:  
 System.out.println("Invalid choice");  
 }  
 }  
 }  
   
 private static void visitPage(String url) {  
 if (currentPage != null) {  
 backHistory.push(currentPage);  
 }  
 currentPage = url;  
 forwardHistory.clear();  
 }  
   
 private static void goBack() {  
 if (backHistory.isEmpty()) {  
 System.out.println("Cannot go back - no history");  
 return;  
 }  
 forwardHistory.push(currentPage);  
 currentPage = backHistory.pop();  
 }  
   
 private static void goForward() {  
 if (forwardHistory.isEmpty()) {  
 System.out.println("Cannot go forward - no history");  
 return;  
 }  
 backHistory.push(currentPage);  
 currentPage = forwardHistory.pop();  
 }  
}