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.**

Program:

```
import java.util.ArrayList;
import java.util.List;
public class ArrayList_Search
{
	public static void main(String[] args)
	{
		List<String> list=new ArrayList<>();
		list.add(0,"Apple");
		list.add(1,"Mango");
		list.add(2,"Kiwi");
		list.add(3,"Banana");
		String search="Mango";
		if(list.contains("Mango"))
			System.out.println(search+" element found in arraylist");
		else
			System.out.println(search+" element not found in arraylist");
	}

}
```

OutPut:

Mango element found in arraylist

---

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.**

Program:

```
import java.util.*;
public class RemoveElement
{
	public static void main(String[] args)
	{
		ArrayList<String> fruits = new ArrayList<>();
		Scanner sc = new Scanner(System.in);
		fruits.add("Apple");
		fruits.add("Banana");
		fruits.add("Mango");
		fruits.add("Grapes");
		fruits.add("Orange");
		System.out.println("Fruits List: " + fruits);
		System.out.print("Enter a fruit to remove: ");
		String fruitToRemove = sc.nextLine();
		if (fruits.remove(fruitToRemove))
		{
			System.out.println(fruitToRemove + " removed.");
		}
		else
		{
			System.out.println(fruitToRemove + " not found.");
```

```
        }
        System.out.println("Updated List: " + fruits);
    sc.close();
  }
}
```

OutPut:

Fruits List: [Apple, Banana, Mango, Grapes, Orange]

Enter a fruit to remove: Mango

Mango removed.

Updated List: [Apple, Banana, Grapes, Orange]

---

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.**

Program:

```
import java.util.*;
public class SortElements
{
        public static void main(String[] args)
        {
                ArrayList<Integer> numbers = new ArrayList<>();
                numbers.add(25);
                numbers.add(10);
                numbers.add(35);
                numbers.add(5);
                numbers.add(60);
```

```
        numbers.add(15);
        numbers.add(45);
        System.out.println("Original List: " + numbers);
        Collections.sort(numbers);
        System.out.println("Sorted List: " + numbers);
    }
}
```

OutPut:

Original List: [25, 10, 35, 5, 60, 15, 45]

Sorted List: [5, 10, 15, 25, 35, 45, 60]

---

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.**

Program:

```
import java.util.ArrayList;
import java.util.Collections;
public class Reverse_ArrayList
{
    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");
```

```
        Collections.reverse(fruits);
        System.out.println(fruits);
    }
}
```

OutPut:

[Grapes, Orange, Mango, Banana, Apple]

---

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.**

Program:

```
import java.util.ArrayList;
public class UpdateArrayList
{
    Public static void main(String[] args)
    {
        ArrayList<String> subjects = new ArrayList<>();
        subjects.add("Math");
        subjects.add("Physics");
        subjects.add("Chemistry");
        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);
```

```
    }
}
```

OutPut:

Before update: [Math, Physics, Chemistry, Biology, English]

After update: [Statistics, Physics, Chemistry, Biology, English]

---

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.**

Program:

```
import java.util.ArrayList;
public class RemoveAll_ArrayList
{
    public static void main(String[] args)
    {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);
        System.out.println("Original List: " + numbers);
        numbers.clear();
        System.out.println("List after clear: " + numbers);
        System.out.println("Size of list: " + numbers.size());
    }
}
```

OutPut:

Original List: [10, 20, 30, 40]

List after clear: []

Size of list: 0

---

7. 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.**

Program:

```
import java.util.ArrayList;
public class CopyOneArrayListTOOther
{
	public static void main(String[] args)
	{
		ArrayList<String> list1 = new ArrayList<>();
		list1.add("Apple");
		list1.add("Banana");
		list1.add("Mango");
		ArrayList<String> list2 = new ArrayList<>();
		list2.addAll(list1);
		System.out.println("First List: " + list1);
		System.out.println("Second List: " + list2);
	}
}
```

OutPut:

First List: [Apple, Banana, Mango]

Second List: [Apple, Banana, Mango]

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 ColorLinkedList
{
	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");
		System.out.println("Colors in the list:");
		for (String color : colors)
		{
			System.out.println(color);
		}
	}
}
```

OutPut:

```
Colors in the list:
Red
Blue
Green
```

Yellow

Purple

---

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.LinkedList;
public class AddElementsFirstAndLast
{
	public static void main(String[] args)
	{
		LinkedList<Integer> list = new LinkedList<>();
		list.add(10);
		list.add(20);
		list.add(30);
		System.out.println("Original List: " + list);
		list.addFirst(5);
		list.addLast(40);
		System.out.println("Updated List: " + list);
	}
}
```

OutPut:

Original List: [10, 20, 30]

Updated List: [5, 10, 20, 30, 40]

---

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

OutPut:

```
Before insertion: [Alice, Bob, David]
After insertion: [Alice, Bob, Charlie, David]
```

---

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.LinkedList;
public class RemoveElements
{
	public static void main(String[] args)
	{
		LinkedList<String> animals = new LinkedList<>();
		animals.add("Dog");
		animals.add("Cat");
		animals.add("Elephant");
		animals.removeFirst();
		animals.removeLast();
		//animals.remove("Cat");
		System.out.println(animals);
	}
}
```

OutPut:

[Cat]

---

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.LinkedList;
public class SearchElement
{
	public static void main(String[] args)
	{
		LinkedList<String> list = new LinkedList<>();
		list.add("Apple");
```

```
        list.add("Banana");
        list.add("Mango");
        String search = "Banana";
        if (list.contains(search))
        {
            System.out.println(search + " is found.");
        }
        else
        {
            System.out.println(search + " is not found.");
        }
    }
}
```

OutPut:

Banana is found.

---

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.LinkedList;
import java.util.ListIterator;
public class ListIterator
{
    public static void main(String[] args)
    {
        LinkedList<String> cities = new LinkedList<>();
        cities.add("Delhi");
        cities.add("Mumbai");
        cities.add("Chennai");
```

```
                cities.add("Kolkata");
                ListIterator<String> it = cities.listIterator();
                System.out.println("Forward:");
                while (it.hasNext())
                {
                        System.out.println(it.next());
                }
                System.out.println("Reverse:");
                while (it.hasPrevious())
                {
                        System.out.println(it.previous());
                }
        }
}
```

OutPut:

Forward:

Delhi

Mumbai

Chennai

Kolkata

Reverse:

Kolkata

Chennai

Mumbai

Delhi

---

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<>();
		linkedList.add("Apple");
		linkedList.add("Banana");
		linkedList.add("Mango");
		linkedList.add("Orange");
		ArrayList<String> arrayList = new ArrayList<>(linkedList);
		System.out.println("LinkedList: " + linkedList);
		System.out.println("ArrayList: " + arrayList);
	}
}
```

OutPut:

```
LinkedList: [Apple, Banana, Mango, Orange]
ArrayList: {Apple, Banana, Mango, Orange}
```

---

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.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 class Main
{
        public static void main(String[] args)
        {
                LinkedList<Book> books = new LinkedList<>();
                books.add(new Book(1, "Java", "James"));
                books.add(new Book(2, "Python", "Guido"));
                books.add(new Book(3, "C++", "Bjarne"));
                for (Book b : books) {
                System.out.println(b.id + " " + b.title + " " + b.author);
        }
   }
}
```

Output:

1 Java James

2 Python Guido

3 C++ Bjarne

---

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 CloneLinkedList
{
	public static void main(String[] args)
	{
		LinkedList<Integer> originalList = new LinkedList<>();
		originalList.add(10);
		originalList.add(20);
		originalList.add(30);
		originalList.add(40);
		originalList.add(50);
		LinkedList<Integer> clonedList = (LinkedList<Integer>) originalList.clone();
		System.out.println("Original LinkedList: " + originalList);
		System.out.println("Cloned LinkedList: " + clonedList);
	}
}
```

OutPut:

```
Original LinkedList: [10, 20, 30, 40, 50]
Cloned LinkedList: [10, 20, 30, 40, 50]
```

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.**

Program:

```
import java.util.*;
public class VectorIntegerOperation
{
    public static void main(String[] args)
    {
            Vector<Integer> numbers = new Vector<>();
            numbers.add(10);
            numbers.add(20);
            numbers.add(30);
            numbers.add(40);
            numbers.add(50);
            numbers.add(2, 25);
            numbers.remove(1);
            System.out.println("Vector Elements:");
            Enumeration<Integer> e = numbers.elements();
            while (e.hasMoreElements())
            {
                    System.out.println(e.nextElement());
            }
    }
}
```

OutPut:

Vector Elements:

10

25

30

40

50

2.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 VectorStringOperation
{
	public static void main(String[] args)
	{
		Vector<String> names = new Vector<>();
		names.add("Alice");
		names.add("Bob");
		names.add("Charlie");
		names.add("David");
		String searchName = "Charlie";
		if (names.contains(searchName))
		{
			System.out.println(searchName + " is present in the vector.");
		}
		else
		{
			System.out.println(searchName + " is not found.");
		}
		int index = names.indexOf("Bob");
		if (index != -1)
		{
			names.set(index, "Brian");
			System.out.println("Replaced 'Bob' with 'Brian'.");
		}
		System.out.println("Updated Vector: " + names);
		names.clear();
		System.out.println("Vector after clearing: " + names);
	}
```

```
}
```

OutPut:

```
Charlie is present in the vector.
Replaced 'Bob' with 'Brian'.
Updated Vector: [Alice, Brian, Charlie, David]
Vector after clearing: []
```

3.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.Vector;

public class SumElements

{

        static int sum(Vector<Integer> v)

        {

                int total = 0;

                for (int n : v) total += n;

                return total;

        }

   public static void main(String[] args)

{

      Vector<Integer> v1 = new Vector<>();

      v1.add(10);

      v1.add(20);

      v1.add(30);


      Vector<Integer> v2 = new Vector<>(v1);

      System.out.println(v1.equals(v2));

      System.out.println(sum(v1));

   }
```

```
}
```

Output:

true

60

Stack

1.Understand how to use the Stack class for LIFO (Last In, First Out) operations.

Program:

```
import java.util.Stack;
public class Stack
{
   public static void main(String[] args)
        {
                Stack<String> stack = new Stack<>();
                stack.push("A");
                stack.push("B");
                stack.push("C");
                System.out.println(stack.pop());
                System.out.println(stack.pop());
                System.out.println(stack);
        }
}
```

OutPut:

C

B

[A]

2.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.Stack;
public class Main
{
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: " + stack.pop());
        System.out.println("Top: " + stack.peek());
        System.out.println("Is empty? " + stack.isEmpty());
    }
}
```

OutPut:

Popped: 50

Top: 40

Is empty? False

---

3. Reverse a string using Stack:

- **Input a string from the user.**
- **Use a stack to reverse and print the string.**

Program:

```
import java.util.Stack;

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

```
        String str = "hello";
        Stack<Character> stack = new Stack<>();

        // Push using for loop
        for (int i = 0; i < str.length(); i++) {
            stack.push(str.charAt(i));
        }

        // Pop and print
        for (int i = 0; i < str.length(); i++) {
            System.out.print(stack.pop());
        }
    }
}
```

OutPut:

olleh

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

- **Input: (a+b) * (c-d)**
- **Output: Valid or Invalid expression**

Program:

```
import java.util.Stack;
public class Main
{
   public static void main(String[] args)
     {
            String exp = "(a+b) * (c-d)";
            Stack<Character> st = new Stack<>();
            for (char c : exp.toCharArray()) {
            if (c == '(') st.push(c);
```

```
            if (c == ')') st.pop();
        }
        System.out.println(st.isEmpty() ? "Valid" : "Invalid");
    }
}
```

OutPut:

Valid

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.**

II. Perform operations:

- **Remove an element.**
- **Check if a city exists.**
- **Clear the entire HashSet.**

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

Program:

```
import java.util.*;
public class CityHashSet
{
    public static void main(String[] args)
    {
        HashSet<String> cities = new HashSet<>();
        cities.add("Mumbai");
        cities.add("Delhi");
        cities.add("Chennai");
        cities.add("Bangalore");
        cities.add("Kolkata");
```

```
        // Try adding a duplicate city
        boolean added = cities.add("Delhi");
        if (!added)
        {
            System.out.println("Duplicate city 'Delhi' was not added.");
        }
        System.out.println("Cities in the HashSet:");
        Iterator<String> it = cities.iterator();
        while (it.hasNext())
        {
            System.out.println(it.next());
        }
        cities.remove("Chennai");
        System.out.println("After removing 'Chennai': " + cities);
        if (cities.contains("Bangalore"))
        {
            System.out.println("Bangalore exists in the HashSet.");
        }
        else
        {
            System.out.println("Bangalore does not exist.");
        }
        // Clear the entire HashSet
        cities.clear();
        System.out.println("HashSet after clearing: " + cities);
    }
}
```

OutPut:

Duplicate city 'Delhi' was not added.

Cities in the HashSet:

Delhi

Chennai

Kolkata

Mumbai

Bangalore

After removing 'Chennai': [Delhi, Kolkata, Mumbai, Bangalore]

Bangalore exists in the HashSet.

HashSet after clearing: []

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<>();
            numbers.add(10);
            numbers.add(5);
            numbers.add(20);
            numbers.add(15);
            numbers.add(5);
            System.out.println("LinkedHashSet elements:");
            for (int num : numbers)
            {
                    System.out.println(num);
            }
      }
}
```

OutPut:

LinkedHashSet elements:

10

5

20

15

_----------

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.**

Program:

```
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 instanceof Student))
                    return false;
              Student other = (Student) obj;
               return id == other.id && name.equals(other.name);
    }
    @Override
     public String toString()
    {
            return id + " - " + name;
    }
}
public class LinkedHashSetStudents
{
    public static void main(String[] args)
```

```
{
    LinkedHashSet<Student> students = new LinkedHashSet<>();
    students.add(new Student(101, "Alice"));
    students.add(new Student(102, "Bob"));
    students.add(new Student(103, "Charlie"));
    boolean added = students.add(new Student(102, "Bob"));
    if (!added)
    {
            System.out.println("Duplicate student not added.");
    }
    System.out.println("Student list:");
    for (Student s : students)
    {
            System.out.println(s);
}
}
```

OutPut:

Duplicate student not added.

Student list:

101 - Alice

102 - Bob

103 - Charlie

3.Write a program to:

- **Merge two LinkedHashSets and print the result.**

Program:

```
import java.util.LinkedHashSet;

public class Merge

{

        public static void main(String[] args)

        {

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

                set1.add("Apple");

                set1.add("Banana");

                LinkedHashSet<String> set2 = new LinkedHashSet<>();
```

```
        set2.add("Mango");
        set2.add("Orange");
        set1.addAll(set2);
        System.out.println("Merged Set: " + set1);
    }
}
```

OutPut:

Merged Set: [Apple, Banana, Mango, Orange]

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 CountryTreeSet
{
    public static void main(String[] args)
    {
        TreeSet<String> countries = new TreeSet<>();
        countries.add("India");
        countries.add("Germany");
        countries.add("Australia");
        countries.add("Brazil");
        countries.add("Canada");
        System.out.println("Sorted Country Names:");
        for (String country : countries)
        {
            System.out.println(country);
        }
    }
}
```

OutPut:

Sorted Country Names:
Australia
Brazil
Canada
Germany
India

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.**

Program:

```
import java.util.TreeSet;
public class TreeSetIntegers
{
    public static void main(String[] args)
    {
            TreeSet<Integer> numbers = new TreeSet<>();
            numbers.add(10);
            numbers.add(20);
            numbers.add(30);
            numbers.add(40);
            System.out.println("First: " + numbers.first());
            System.out.println("Last: " + numbers.last());
            int num = 25;
            System.out.println("Lower than " + num + ": " + numbers.lower(num));
            System.out.println("Higher than " + num + ": " + numbers.higher(num));
    }
}
```

OutPut:

```
First: 10
Last: 40
Lower than 25: 20
Higher than 25: 30
```

3.Create a TreeSet with a custom comparator:

- **Sort strings in reverse alphabetical order using Comparator.**

Program:

```
import java.util.TreeSet;
```

```
import java.util.Comparator;
public class Reverse
{
   public static void main(String[] args)
{
      TreeSet<String> set = new TreeSet<>(Comparator.reverseOrder());
      set.add("Apple");
      set.add("Banana");
      set.add("Mango");
      System.out.println(set);
   }
}
```

Output:

[Mango, Banana, Apple]

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> customerQueue = new LinkedList<>();
        customerQueue.add("Alice");
        customerQueue.add("Bob");
        customerQueue.add("Charlie");
```

```
            customerQueue.add("David");
            customerQueue.add("Eve");
            System.out.println("Initial Queue: " + customerQueue);
            while (!customerQueue.isEmpty())
            {
                String served = customerQueue.remove();
                System.out.println("Served: " + served);
                System.out.println("Queue now: " + customerQueue);
            }
        }
}
```

OutPut:

```
Initial Queue: [Alice, Bob, Charlie, David, Eve]
Served: Alice
Queue now: [Bob, Charlie, David, Eve]
Served: Bob
Queue now: [Charlie, David, Eve]
Served: Charlie
Queue now: [David, Eve]
Served: David
Queue now: [Eve]
Served: Eve
Queue now: []
```

---

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

Program:

```
import java.util.LinkedList;
import java.util.Queue;
public class Main
{
   public static void main(String[] args)
   {
       Queue<String> q = new LinkedList<>();
```

```
        q.add("Task 1");
        q.add("Task 2");
        System.out.println(q.peek());
        System.out.println(q.poll());
        System.out.println(q);
    }
}
```

OutPut:

Task 1

Task 1

[Task 2]

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;
    Patient(String name, int severityLevel)
    {
        this.name = name;
        this.severityLevel = severityLevel;
    }
    @Override
    public String toString()
    {
```

```
        return name + " (Severity: " + severityLevel + ")";
        }
    }
    public class HospitalQueue
    {
        public static void main(String[] args)
        {
            PriorityQueue<Patient> emergencyQueue = new PriorityQueue<>(new
Comparator<Patient>()
              {
                  public int compare(Patient p1, Patient p2)
                  {
                      return Integer.compare(p2.severityLevel,
p1.severityLevel);
                  }
            });
            emergencyQueue.add(new Patient("Alice", 4));
            emergencyQueue.add(new Patient("Bob", 2));
            emergencyQueue.add(new Patient("Charlie", 5));
            emergencyQueue.add(new Patient("David", 3));
            emergencyQueue.add(new Patient("Eve", 1));
            System.out.println("Serving patients in order of severity:");
            while (!emergencyQueue.isEmpty())
              {
                  System.out.println("Attending: " + emergencyQueue.poll());
              }
         }
    }
```

OutPut:

Serving patients in order of severity:

Attending: Charlie (Severity: 5)

Attending: Alice (Severity: 4)

Attending: David (Severity: 3)

Attending: Bob (Severity: 2)

Attending: Eve (Severity: 1)

2, 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.*;


public class Main {
    public static void main(String[] args) {
        PriorityQueue<String> jobs = new PriorityQueue<>();
        jobs.add("High");
        jobs.add("Medium");
        jobs.add("Low");


        while (!jobs.isEmpty()) {
            System.out.println(jobs.poll());
        }
    }
}
```

OutPut:

High

Low

Medium

3.**Write a method:**

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

Program:

```
import java.util.*;


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


   public static void main(String[] args) {
      PriorityQueue<Integer> q1 = new PriorityQueue<>();
      q1.add(3);
      q1.add(1);


      PriorityQueue<Integer> q2 = new PriorityQueue<>();
      q2.add(4);
      q2.add(2);


      PriorityQueue<Integer> result = mergeQueues(q1, q2);
      System.out.println(result);
   }
}
```

OutPut:

[1, 2, 3, 4]

Deque

1, Palindrome Checker:

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

II. Double-ended Order System:

a. **Add items from front and rear.**

b. **Remove items from both ends.**

c. **Display contents of the deque after each operation.**

Program:

```
import java.util.*;
public class DoubleEndedOrderedSystem {

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

        // Add items from front and rear
        orders.addFirst("Order A");
        orders.addLast("Order B");
        orders.addFirst("Order C");
        orders.addLast("Order D");
        System.out.println("After adding orders:");
        System.out.println(orders);
        String frontRemoved = orders.removeFirst();
        System.out.println("Removed from front: " + frontRemoved);
        System.out.println("Current orders: " + orders);
        String rearRemoved = orders.removeLast();
        System.out.println("Removed from rear: " + rearRemoved);
        System.out.println("Current orders: " + orders);
    }
}
```

OutPut:

```
After adding orders:
[Order C, Order A, Order B, Order D]
Removed from front: Order C
Current orders: [Order A, Order B, Order D]
```

Removed from rear: Order D
Current orders: [Order A, Order B]

---

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

Program:

```
import java.util.*;

public class Main {
   public static void main(String[] args) {
      Deque<String> back = new ArrayDeque<>();
      Deque<String> forward = new ArrayDeque<>();
      back.push("Google");
      back.push("YouTube");
      back.push("GitHub");
      System.out.println("Current Page: " + back.peek());
      forward.push(back.pop());
      System.out.println("Back to: " + back.peek());
      back.push(forward.pop());
      System.out.println("Forward to: " + back.peek());
   }
}
```

OutPut:

Current Page: GitHub

Back to: YouTube

Forward to: GitHub