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.

```
import java.util.ArrayList;
import java.util.Scanner;
public class SearchElementDemo {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10);
        numbers.add(25);
        numbers.add(30);
        numbers.add(45);
        numbers.add(50);
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to search: ");
        int input = scanner.nextInt();
        if (numbers.contains(input)) {
            System.out.println(input + " exists in the list.");
        } else {
            System.out.println(input + " does not exist in the list.");
        }

        scanner.close();
    }
}
```

Output;

```
Enter a number to search: 30
30 exists in the list.
```

---

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.

```
import java.util.ArrayList;
import java.util.Scanner;
public class RemoveFruitDemo {
    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");

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

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the name of the fruit to remove: ");
        String fruitToRemove = scanner.nextLine();
        if (fruits.remove(fruitToRemove)) {
            System.out.println(fruitToRemove + " removed successfully.");
        } else {
            System.out.println(fruitToRemove + " not found in the list.");
        }

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

        scanner.close();
    }
}
```

Output:

```
Original list: [Apple, Banana, Mango, Orange, Grapes]
Enter the name of the fruit to remove: Mango
Mango removed successfully.
Updated list: [Apple, Banana, Orange, Grapes]
```

---

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.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
public class SortArrayListDemo {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        Random rand = new Random();

        for (int i = 0; i < 7; i++) {
            numbers.add(rand.nextInt(100) + 1);
        }

        System.out.println("Original list: " + numbers);
        Collections.sort(numbers);

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

Output:

```
Original list: [57, 12, 89, 34, 73, 44, 21]
Sorted list: [12, 21, 34, 44, 57, 73, 89]
```

---

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.

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

Output:

```
Original list: [A, B, C, D, E]
Reversed list: [E, D, C, B, A]
```

---

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.

```
import java.util.ArrayList;
public class UpdateElementDemo {
    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);
        } else {
            System.out.println("\"Math\" not found in the list.");
        }
    }
}
```

Output:

```
Before update: [Physics, Chemistry, Math, Biology, English]
After update: [Physics, Chemistry, Statistics, 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.

```
import java.util.ArrayList;
public class RemoveAllElementsDemo {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);
        numbers.add(50);

        System.out.println("List before clear: " + numbers);

        numbers.clear();
        System.out.println("Size of list after clear: " + numbers.size());
    }
}
```

Output:

```
List before clear: [10, 20, 30, 40, 50]
Size of list after clear: 0
```

---

7. Iterate using Iterator

Write a program to:

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

```
import java.util.ArrayList;
import java.util.Iterator;
public class IteratorDemo {
    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();

        System.out.println("Cities in the list:");
        while (iterator.hasNext()) {
            String city = iterator.next();
            System.out.println(city);
        }
    }
}
```

Output:

```
Cities in the list:
New York
London
Tokyo
Paris
Sydney
```

---

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.

```java
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 String toString() {
        return "Student{id=" + id + ", name='" + name + "', marks=" + marks + "}";
    }
}

public class StudentListDemo {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();

        students.add(new Student(101, "Ajay", 88.5));
        students.add(new Student(102, "vinay", 92.0));
        students.add(new Student(103, "vijay", 79.5));

        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

```
Output:

Student{id=101, name='Ajay', marks=88.5}

Student{id=102, name=’vinay’, marks=92.0}

Student{id=103, name='vijay', marks=79.5}
```

---

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.

```
import java.util.ArrayList;
public class CopyArrayListDemo {
    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<>();
        list1.add("Red");
        list1.add("Green");
        list1.add("Blue");
        list1.add("Yellow");

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

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

        list2.addAll(list1);
        System.out.println("Copied list2: " + list2);
    }
}
```

Output:

```
Original list1: [Red, Green, Blue, Yellow]
Copied list2: [Red, Green, Blue, Yellow]
```

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.

```
import java.util.LinkedList;

public class LinkedListDemo {
    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("Orange");

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

Output:

```
Colors in the LinkedList:
Red
Green
Blue
Yellow
Orange
```

---

2. Add Elements at First and Last Position

Write a program to:

- Create a LinkedList of integers.
- Add elements at the beginning and at the end.
- Display the updated list.

```
import java.util.LinkedList;
public class LinkedListAddFirstLastDemo {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);

        System.out.println("Original list: " + numbers);
        numbers.addFirst(10);

        numbers.addLast(50);

        System.out.println("Updated list after adding first and last: " + numbers);
    }
}
```

Output:

```
Original list: [20, 30, 40]
Updated list after adding first and last: [10, 20, 30, 40, 50]
```

---

3. Insert Element at Specific Position

Write a program to:

- Create a LinkedList of names.
- Insert a name at index 2.
- Display the list before and after insertion.

```
import java.util.LinkedList;

public class InsertAtPositionDemo {
    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 at index 2: " + names);
    }
}
```

Output:

```
Before insertion: [Alice, Bob, Charlie, David]
After insertion at index 2: [Alice, Bob, Eve, 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.

```
import java.util.LinkedList;

public class RemoveElementsDemo {
    public static void main(String[] args) {
        LinkedList<String> animals = new LinkedList<>();
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Elephant");
        animals.add("Tiger");
        animals.add("Lion");


        System.out.println("Original list: " + animals);

        String removedFirst = animals.removeFirst();

        System.out.println("After removing first element (" + removedFirst + "): " +
animals);

        String removedLast = animals.removeLast();

        System.out.println("After removing last element (" + removedLast + "): " +
animals);

        boolean removed = animals.remove("Elephant");

        if (removed) {

            System.out.println("After removing 'Elephant': " + animals);

        } else {

            System.out.println("'Elephant' not found in the list.");

        }

    }

}
```

Output:

```
Original list: [Dog, Cat, Elephant, Tiger, Lion]

After removing first element (Dog): [Cat, Elephant, Tiger, Lion]

After removing last element (Lion): [Cat, Elephant, Tiger]

After removing 'Elephant': [Cat, Tiger]
```

---

5. Search for an Element

Write a program to:

- Create a LinkedList of Strings.
- Ask the user for a string to search.
- Display if the string is found or not.

```
import java.util.LinkedList;
import java.util.Scanner;

public class LinkedListSearchDemo {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");
        list.add("Date");
        list.add("Elderberry");

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a 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();
    }
}
```

Output:

```
Enter a string to search: Cherry
Cherry is found in the list.
```

}

6. Iterate using ListIterator

Write a program to:

- Create a LinkedList of cities.
- Use ListIterator to display the list in both forward and reverse directions.

```
import java.util.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 traversal:");
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        System.out.println("\nBackward traversal:");
        while (iterator.hasPrevious()) {
            System.out.println(iterator.previous());
        }
    }
}
```

Output:

```
Forward traversal:
New York
London
Tokyo
Paris
Sydney

Backward traversal:
Sydney
Paris
Tokyo
London
New York
```

---

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.

```java
import java.util.Collections;
import java.util.LinkedList;

public class SortLinkedListDemo {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(42);
        numbers.add(15);

        numbers.add(8);

        numbers.add(23);

        numbers.add(4);


        System.out.println("Before sorting: " + numbers);

        Collections.sort(numbers);


        System.out.println("After sorting: " + numbers);

    }

}
```

```
Output:

Before sorting: [42, 15, 8, 23, 4]

After sorting: [4, 8, 15, 23, 42]
```

---

8. Convert LinkedList to ArrayList

Write a program to:

- Create a LinkedList of Strings.
- Convert it into an ArrayList.
- Display both the LinkedList and ArrayList.

```
import java.util.ArrayList;

import java.util.LinkedList;


public class ConvertLinkedListToArrayList {

    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);
    }
}
```

Output:

```
LinkedList: [Red, Green, Blue, Yellow]
ArrayList: [Red, Green, Blue, Yellow]
```

---

9. Store Custom Objects in LinkedList

Write a program to:

- Create a class Book with fields: id, title, and author.
- Create a LinkedList of Book objects.
- Add 3 books and display their details using a loop.

```
import java.util.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 String toString() {
        return "Book{id=" + id + ", title='" + title + "', author='" + author + "'}";
    }
}

public class BookLinkedListDemo {
    public static void main(String[] args) {
        LinkedList<Book> books = new LinkedList<>();
        books.add(new Book(101, "1984", "George Orwell"));
        books.add(new Book(102, "To Kill a Mockingbird", "Harper Lee"));
        books.add(new Book(103, "The Great Gatsby", "F. Scott Fitzgerald"));
        for (Book book : books) {
            System.out.println(book);
        }
    }
}
```

Output:

```
Book{id=101, title='1984', author='George Orwell'}
Book{id=102, title='To Kill a Mockingbird', author='Harper Lee'}
Book{id=103, title='The Great Gatsby', author='F. Scott Fitzgerald'}
```

10. Clone a LinkedList

Write a program to:

- Create a LinkedList of numbers.
- Clone it using the clone() method.
- Display both original and cloned lists.

```
import java.util.LinkedList;

public class CloneLinkedListDemo {
  public static void main(String[] args) {
    LinkedList<Integer> originalList = new LinkedList<>();
    originalList.add(10);
    originalList.add(20);
    originalList.add(30);
    originalList.add(40);

    System.out.println("Original LinkedList: " + originalList);



    LinkedList<Integer> clonedList = (LinkedList<Integer>) originalList.clone();

    System.out.println("Cloned LinkedList: " + clonedList);
  }
}
```

Output:

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

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.

```
import java.util.Enumeration;
import java.util.Vector;

public class VectorOperationsDemo {
    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);

        System.out.println("Initial Vector: " + numbers);
        numbers.insertElementAt(25, 2);
        System.out.println("After inserting 25 at index 2: " + numbers);
        numbers.remove(1);
        System.out.println("After removing element at index 1: " + numbers);
        System.out.println("Elements in Vector using Enumeration:");
        Enumeration<Integer> enumeration = numbers.elements();
        while (enumeration.hasMoreElements()) {
            System.out.println(enumeration.nextElement());
        }
    }
}
```

Output:

```
Initial Vector: [10, 20, 30, 40, 50]
After inserting 25 at index 2: [10, 20, 25, 30, 40, 50]
After removing element at index 1: [10, 25, 30, 40, 50]
Elements in Vector using Enumeration:
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.

```
import java.util.Vector;

public class VectorStringDemo {
  public static void main(String[] args) {
    Vector<String> names = new Vector<>();

    // Add at least 4 names
    names.add("Ajay");
    names.add("vijay”);
    names.add("vinay");
    names.add("rahul");

    System.out.println("Original Vector: " + names);
```

```
        // Check if a specific name exists
        String searchName = "vijay";
        if (names.contains(searchName)) {
            System.out.println(searchName + " exists in the vector.");
        } else {
            System.out.println(searchName + " does not exist in the vector.");
        }

        // Replace one name with another (replace "vijay" with "uday")
        int index = names.indexOf("vijay");
        if (index !=-1) {
            names.set(index, "uday");
            System.out.println("After replacing 'vijay' with ''uday " + names);
        } else {
            System.out.println("'vijay' not found in the vector.");
        }

        // Clear all elements from the vector
        names.clear();
        System.out.println("After clearing, vector size: " + names.size());
    }
}
```

Output:

```
Original Vector: [Alice, Bob, Charlie, Diana]
Charlie exists in the vector.
After replacing 'Bob' with 'Brian': [Alice, Brian, Charlie, Diana]
After clearing, vector size: 0
```

Write a program to:

- Copy all elements from one Vector to another Vector.
- Compare both vectors for equality.
- **Write a method** that takes a Vector<Integer> and returns the **sum of all elements**

```
import java.util.Vector;

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

        Vector<Integer> vector1 = new Vector<>();
        vector1.add(10);
        vector1.add(20);
        vector1.add(30);
        vector1.add(40);



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

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

        if (vector1.equals(vector2)) {
            System.out.println("Both vectors are equal.");
        } else {
            System.out.println("Vectors are not equal.");
        }
        int sum = sumOfElements(vector1);
        System.out.println("Sum of elements in vector1: " + sum);
    }

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

Output:

```
Vector1: [10, 20, 30, 40]
Vector2: [10, 20, 30, 40]
Both vectors are equal.
Sum of elements in vector1: 100
```

Stack

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

---

1. **Create a Stack of integers** and:

- Push 5 elements.
- Pop the top element.
- Peek the current top.
- Check if the stack is empty.

```
import java.util.Stack;

public class StackDemo {
    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("Stack after pushing 5 elements: " + stack);

        int poppedElement = stack.pop();
        System.out.println("Popped element: " + poppedElement);
        System.out.println("Stack after pop: " + stack);

        int topElement = stack.peek();
        System.out.println("Current top element: " + topElement);
        boolean isEmpty = stack.isEmpty();
        System.out.println("Is stack empty? " + isEmpty);
    }
}
```

Output:

```
Stack after pushing 5 elements: [10, 20, 30, 40, 50]
Popped element: 50
Stack after pop: [10, 20, 30, 40]
Current top element: 40
Is stack empty? False
```

2. Reverse a string using Stack:

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

```
import java.util.Scanner;
import java.util.Stack;

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

        System.out.print("Enter a string to reverse: ");
        String input = scanner.nextLine();

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


        for (char ch : input.toCharArray()) {
            stack.push(ch);
        }
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }

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

        scanner.close();
```

```
  }
}
```

Output:

```
Enter a string to reverse: hello world
Reversed string: dlrow olleh
```

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

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

```
import java.util.Scanner;
import java.util.Stack;

public class ParenthesesChecker {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter an expression: ");
    String expr = scanner.nextLine();

    if (isBalanced(expr)) {
      System.out.println("Valid expression");
    } else {
      System.out.println("Invalid expression");
    }

    scanner.close();
  }
```

```
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {
            if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
                    return false;
                }
                stack.pop();
            }
        }

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

Output:

```
Enter an expression: (a+b) * (c-d)
Valid 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.

```
import java.util.HashSet;
import java.util.Iterator;

public class HashSetDemo {
    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");

        System.out.println("Cities after adding 5 unique names: " + cities);
        boolean added = cities.add("Tokyo"); // duplicate
        System.out.println("Trying to add duplicate 'Tokyo': " + (added ? "Added" : "Not Added"));

        System.out.println("Cities in the HashSet:");
        Iterator<String> iterator = cities.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

Output:

```
Cities after adding 5 unique names: [Tokyo, Sydney, New York, Paris, London]
Trying to add duplicate 'Tokyo': Not Added
Cities in the HashSet:
Tokyo
Sydney
```

New York

Paris

London

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

```
import java.util.HashSet;


public class HashSetOperationsDemo {
  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");


    System.out.println("Initial HashSet: " + cities);
    boolean removed = cities.remove("Tokyo");
    System.out.println("Removing 'Tokyo': " + (removed ? "Success" : "Not Found"));
    System.out.println("After removal: " + cities);
    String cityToCheck = "Paris";
    boolean exists = cities.contains(cityToCheck);
    System.out.println("Does '" + cityToCheck + "' exist? " + (exists ? "Yes" : "No"));
    cities.clear();
    System.out.println("After clearing, HashSet size: " + cities.size());
    }
}
```

Output:

```
Initial HashSet: [Tokyo, Sydney, New York, Paris, London]
Removing 'Tokyo': Success
After removal: [Sydney, New York, Paris, London]
Does 'Paris' exist? Yes
After clearing, HashSet size: 0
```

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

```
import java.util.HashSet;
import java.util.Collections;

public class HashSetMaxElement {
    public static void main(String[] args) {
        HashSet<Integer> numbers = new HashSet<>();
        numbers.add(15);
        numbers.add(42);
        numbers.add(7);
        numbers.add(29);

        Integer max = getMax(numbers);
        if (max != null) {
            System.out.println("Maximum element: " + max);
        } else {
            System.out.println("HashSet is empty.");
        }
    }

    public static Integer getMax(HashSet<Integer> set) {
        if (set == null || set.isEmpty()) {
            return null;
        }
        return Collections.max(set);
    }
}
```

Output:

```
Maximum element: 42
```

LinkedHashSet

1.Create a LinkedHashSet of Integers:

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

```
import java.util.LinkedHashSet;

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

        System.out.println("Elements in LinkedHashSet: " + numbers);
    }
}
```

Output:

```
Elements in LinkedHashSet: [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.

```
import java.util.LinkedHashSet;
import java.util.Objects;

class Student {
    private int id;
    private String name;

    public 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 student = (Student) o;
        return id == student.id && Objects.equals(name, student.name);
    }


    public int hashCode() {
        return Objects.hash(id, name);
    }
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "'}";
    }
}

public class LinkedHashSetCustomObjects {
    public static void main(String[] args) {
        LinkedHashSet<Student> students = new LinkedHashSet<>();

        Student s1 = new Student(101, "Ajay");
        Student s2 = new Student(102, "vijay");
        Student s3 = new Student(103, "vinay");
        Student duplicate = new Student(101, "Ajay");

        students.add(s1);
        students.add(s2);
        students.add(s3);

        System.out.println("Students after adding 3 unique objects:");
        students.forEach(System.out::println);
        boolean added = students.add(duplicate);

        System.out.println("\nTrying to add duplicate student " + duplicate);

        System.out.println("Was duplicate added? " + (added ? "Yes" : "No"));


        System.out.println("\nStudents in LinkedHashSet after attempting
duplicate add:");

        students.forEach(System.out::println);

    }

}
```

Output:

```
Students after adding 3 unique objects:

Student{id=101, name='Ajay'}

Student{id=102, name='vijay'}

Student{id=103, name='vinay'}


Trying to add duplicate student Student{id=101, name='Ajay'}

Was duplicate added? No


Students in LinkedHashSet after attempting duplicate add:

Student{id=101, name='Ajay'}

Student{id=102, name='vijay'}

Student{id=103, name='vinay'}
```

3. **Write a program** to:

   - Merge two LinkedHashSets and print the result.

```
import java.util.LinkedHashSet;


public class MergeLinkedHashSets {
    public static void main(String[] args) {
        LinkedHashSet<String> set1 = new LinkedHashSet<>();
        set1.add("Apple");
        set1.add("Banana");
        set1.add("Cherry");

        LinkedHashSet<String> set2 = new LinkedHashSet<>();
        set2.add("Date");
        set2.add("Banana");  // duplicate to test merge behavior
        set2.add("Elderberry");

        System.out.println("Set 1: " + set1);
        System.out.println("Set 2: " + set2);
        set1.addAll(set2);

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

Output:

```
Set 1: [Apple, Banana, Cherry]
Set 2: [Date, Banana, Elderberry]
Merged Set: [Apple, Banana, Cherry, Date, Elderberry]
```

TreeSet

1. Create a TreeSet of Strings:

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

```
import java.util.TreeSet;

public class TreeSetDemo {
  public static void main(String[] args) {
    TreeSet<String> countries = new TreeSet<>();
    countries.add("Brazil");
    countries.add("Canada");
    countries.add("India");
    countries.add("Australia");
    countries.add("Germany");
    System.out.println("Countries in sorted order:");
    for (String country : countries) {
      System.out.println(country);
    }
  }
}
```

Output:

```
Countries in sorted order:
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.

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

```
public class TreeSetIntegerDemo {
    public static void main(String[] args) {
        TreeSet<Integer> numbers = new TreeSet<>();
        numbers.add(10);
        numbers.add(25);
        numbers.add(15);
        numbers.add(30);
        numbers.add(5);

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

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

        int givenNumber = 20;
        Integer lower = numbers.lower(givenNumber);
        Integer higher = numbers.higher(givenNumber);

        System.out.println("Element lower than " + givenNumber + ": " + (lower != null ? lower : "None"));
        System.out.println("Element higher than " + givenNumber + ": " + (higher != null ? higher : "None"));
    }
}
```

Output:

```
Numbers in TreeSet: [5, 10, 15, 25, 30]
First (lowest) element: 5
Last (highest) element: 30
Element lower than 20: 15
Element higher than 20: 25
```

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

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

public class TreeSetCustomComparator {
    public static void main(String[] args) {
        Comparator<String> reverseAlpha = (s1, s2)-> s2.compareTo(s1);

        TreeSet<String> fruits = new TreeSet<>(reverseAlpha);

        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Cherry");
        fruits.add("Date");

        System.out.println("Fruits in reverse alphabetical order:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

Output:

```
Fruits in reverse alphabetical order:
Mango
Date
```

```
Cherry
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.

```
import java.util.LinkedList;
import java.util.Queue;

public class BankQueueSimulation {
    public static void main(String[] args) {
        Queue<String> bankQueue = new LinkedList<>();
        bankQueue.add("Alice");
        bankQueue.add("Bob");
        bankQueue.add("Charlie");
        bankQueue.add("Diana");
        bankQueue.add("Ethan");

        System.out.println("Initial queue: " + bankQueue);
        while (!bankQueue.isEmpty()) {
            String servedCustomer = bankQueue.poll(); // removes head of queue
            System.out.println("Serving customer: " + servedCustomer);
            System.out.println("Queue after serving: " + bankQueue);
        }
    }
}
```

Output:

```
Initial queue: [Alice, Bob, Charlie, Diana, Ethan]
Serving customer: Alice
Queue after serving: [Bob, Charlie, Diana, Ethan]
Serving customer: Bob
Queue after serving: [Charlie, Diana, Ethan]
Serving customer: Charlie
Queue after serving: [Diana, Ethan]
Serving customer: Diana
Queue after serving: [Ethan]
Serving customer: Ethan
Queue after serving: []
```

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

```
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("Email client");
        tasks.add("Prepare presentation");
        tasks.add("Fix bugs");
        tasks.add("Attend meeting");
        System.out.println("All tasks: " + tasks);
        String nextTask = tasks.peek();
        System.out.println("Next task to do: " + nextTask);
        while (!tasks.isEmpty()) {
            String completedTask = tasks.poll();
            System.out.println("Completed task: " + completedTask);
            System.out.println("Remaining tasks: " + tasks);
        }
    }
}
```

Output:

```
All tasks: [Write report, Email client, Prepare presentation, Fix bugs, Attend meeting]
Next task to do: Write report
Completed task: Write report
Remaining tasks: [Email client, Prepare presentation, Fix bugs, Attend meeting]
Completed task: Email client
Remaining tasks: [Prepare presentation, Fix bugs, Attend meeting]
Completed task: Prepare presentation
Remaining tasks: [Fix bugs, Attend meeting]
Completed task: Fix bugs
Remaining tasks: [Attend meeting]
Completed task: Attend meeting
Remaining tasks: []
```

3. Write a method:
    - That takes a queue of integers and returns a list of even numbers.

```
import java.util.*;


public class EvenNumbersFromQueue {
  public static void main(String[] args) {
    Queue<Integer> numbers = new LinkedList<>(Arrays.asList(10, 15, 22, 33, 44, 55, 60));


    List<Integer> evenNumbers = getEvenNumbers(numbers);


    System.out.println("Original queue: " + numbers);
    System.out.println("Even numbers: " + evenNumbers);
  }


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

Output:

```
Original queue: [10, 15, 22, 33, 44, 55, 60]
Even numbers: [10, 22, 44, 60]
```

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).

```
import java.util.PriorityQueue;
import java.util.Comparator;

class Patient {
    private String name;
    private int severityLevel;

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

    public String getName() {
        return name;
    }

    public int getSeverityLevel() {
        return severityLevel;
    }
    public String toString() {
        return "Patient{name='" + name + "', severityLevel=" + severityLevel + '}';
    }
}
```

```
public class HospitalEmergencyQueue {
    public static void main(String[] args) {
        Comparator<Patient> severityComparator = (p1, p2)->
Integer.compare(p2.getSeverityLevel(), p1.getSeverityLevel());


        PriorityQueue<Patient> emergencyQueue = new
PriorityQueue<>(severityComparator);
        emergencyQueue.add(new Patient("Alice", 5));
        emergencyQueue.add(new Patient("Bob", 8));
        emergencyQueue.add(new Patient("Charlie", 3));
        emergencyQueue.add(new Patient("Diana", 9));
        emergencyQueue.add(new Patient("Ethan", 6));


        System.out.println("Serving patients by severity:");


        while (!emergencyQueue.isEmpty()) {
            Patient nextPatient = emergencyQueue.poll();
            System.out.println("Serving " + nextPatient);
        }
    }
}
```

Output:

```
Serving patients by severity:
Serving Patient{name='Diana', severityLevel=9}
Serving Patient{name='Bob', severityLevel=8}
Serving Patient{name='Ethan', severityLevel=6}
Serving Patient{name='Alice', severityLevel=5}
Serving Patient{name='Charlie', severityLevel=3}
```

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

```
import java.util.PriorityQueue;
import java.util.Comparator;

class PrintJob {
    private String jobName;
    private int priority;

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

    public String getJobName() {
        return jobName;
    }

    public int getPriority() {
        return priority;
    }
    public String toString() {
        return "PrintJob{" + "jobName='" + jobName + '\'' + ", priority=" + priority + '}';
    }
}

public class PrintJobsPriority {
    public static void main(String[] args) {
        Comparator<PrintJob> priorityComparator = (p1, p2) ->
Integer.compare(p2.getPriority(), p1.getPriority());

        PriorityQueue<PrintJob> printQueue = new
PriorityQueue<>(priorityComparator);

        printQueue.add(new PrintJob("Document1.pdf", 2));
        printQueue.add(new PrintJob("Photo.png", 5));
        printQueue.add(new PrintJob("Report.docx", 3));
        printQueue.add(new PrintJob("Invoice.xls", 1));
        printQueue.add(new PrintJob("Presentation.ppt", 4));

        System.out.println("Serving print jobs by priority:");

        while (!printQueue.isEmpty()) {
            PrintJob job = printQueue.poll();
            System.out.println("Printing: " + job);
        }
    }
}
```

Output:

```
Serving print jobs by priority:
Printing: PrintJob{jobName='Photo.png', priority=5}
Printing: PrintJob{jobName='Presentation.ppt', priority=4}
Printing: PrintJob{jobName='Report.docx', priority=3}
Printing: PrintJob{jobName='Document1.pdf', priority=2}
Printing: PrintJob{jobName='Invoice.xls', priority=1}
```

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

```
import java.util.PriorityQueue;

public class PriorityQueueMerge {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq1 = new PriorityQueue<>();
        pq1.add(10);
        pq1.add(5);
        pq1.add(20);

        PriorityQueue<Integer> pq2 = new PriorityQueue<>();
        pq2.add(15);
        pq2.add(7);
        pq2.add(25);

        PriorityQueue<Integer> merged = mergePriorityQueues(pq1, pq2);

        System.out.println("Merged PriorityQueue:");
        while (!merged.isEmpty()) {
            System.out.print(merged.poll() + " ");
        }
    }

    public static PriorityQueue<Integer> mergePriorityQueues(PriorityQueue<Integer> pq1, PriorityQueue<Integer> pq2) {
        PriorityQueue<Integer> merged = new PriorityQueue<>();
        merged.addAll(pq1);
        merged.addAll(pq2);
        return merged;
    }
}
```

Output:

```
Merged PriorityQueue:
5 7 10 15 20 25
```

Deque

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

```
import java.util.Deque;
import java.util.ArrayDeque;
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: ");
        String input = scanner.nextLine();

        boolean isPalindrome = checkPalindrome(input);

        if (isPalindrome) {
            System.out.println("\"" + input + "\" is a palindrome.");
        } else {
            System.out.println("\"" + input + "\" is not a palindrome.");
        }

        scanner.close();
    }

    public static boolean checkPalindrome(String str) {
        Deque<Character> deque = new ArrayDeque<>();

        String cleaned = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

        for (char ch : cleaned.toCharArray()) {
            deque.addLast(ch);
        }

        while (deque.size() > 1) {
            if (!deque.removeFirst().equals(deque.removeLast())) {
                return false;
            }
        }

        return true;
    }
}
```

Output:

```
Enter a string: Madam, in Eden, I'm Adam
"Madam, in Eden, I'm Adam" is a palindrome.
```

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

```
import java.util.ArrayDeque;
import java.util.Deque;

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

        orders.addLast("Order1");
        orders.addLast("Order2");
        System.out.println("After adding at rear: " + orders);

        orders.addFirst("Order0");
        System.out.println("After adding at front: " + orders);

        String removedFront = orders.removeFirst();
        System.out.println("Removed from front: " + removedFront);
        System.out.println("After removing from front: " + orders);


        String removedRear = orders.removeLast();
        System.out.println("Removed from rear: " + removedRear);
        System.out.println("After removing from rear: " + orders);

        orders.addFirst("OrderStart");
        orders.addLast("OrderEnd");
        System.out.println("After adding more items: " + orders);


        while (!orders.isEmpty()) {
            System.out.println("Serving order: " + orders.removeFirst());
            System.out.println("Remaining orders: " + orders);
        }
    }
}
```

Output:

```
After adding at rear: [Order1, Order2]
After adding at front: [Order0, Order1, Order2]
Removed from front: Order0
After removing from front: [Order1, Order2]
Removed from rear: Order2
After removing from rear: [Order1]
After adding more items: [OrderStart, Order1, OrderEnd]
Serving order: OrderStart
Remaining orders: [Order1, OrderEnd]
Serving order: Order1
Remaining orders: [OrderEnd]
Serving order: OrderEnd
Remaining orders: []
```