List (ArrayList)

2. Search an Element Write a program to:

•Create an ArrayList of integers.

•Ask the user to enter a number.

•Check if the number exists in the list.

Answer

```
import java.util.ArrayList;
import java.util.Scanner;

public class SearchElement {
   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 sc = new Scanner(System.in);
      System.out.print("Enter a number to search: ");
      int searchNum = sc.nextInt();

      if (numbers.contains(searchNum)) {
         System.out.println(searchNum + " exists in the list.");
      } else {
         System.out.println(searchNum + " does not exist in the list.");
      }
```

```
        sc.close();
    }
}
```

3. Remove Specific Element Write a program to:

- Create an ArrayList of Strings.
- Add 5 fruits.
- Remove a specific fruit by name. Display the updated list.

Answer

```
import java.util.ArrayList;

public class RemoveFruit {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();

        // Adding 5 fruits
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Orange");
        fruits.add("Grapes");
        System.out.println("Fruits before removal: " + fruits);
        String fruitToRemove = "Mango";
        if (fruits.remove(fruitToRemove)) {
            System.out.println(fruitToRemove + " removed successfully.");
        } else {
            System.out.println(fruitToRemove + " not found in the list.");
        }
```

```
System.out.println("Fruits after removal: " + fruits);
   }
}
```

4. Sort Elements Write a program to:

- Create an ArrayList of integers.
- Add at least 7 random numbers.
- Sort the list in ascending order.
- Display the sorted list.

Answer

```
import java.util.ArrayList;
import java.util.Collections;

public class SortArrayList {
   public static void main(String[] args) {
      ArrayList<Integer> numbers = new ArrayList<>();

      // Adding 7 random numbers
      numbers.add(45);
      numbers.add(12);
      numbers.add(89);
      numbers.add(32);
      numbers.add(7);
      numbers.add(66);
      numbers.add(23);

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

```
        Collections.sort(numbers);

        System.out.println("After Sorting: " + numbers);
    }
}
```

5. Reverse the ArrayList Write a program to:

•Create an ArrayList of characters.

•Add 5 characters.

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

Answer

```
import java.util.ArrayList;
import java.util.Collections;

public class ReverseArrayList {
    public static void main(String[] args) {
        ArrayList<Character> chars = new ArrayList<>();

        // Adding 5 characters
        chars.add('A');
        chars.add('B');
        chars.add('C');
        chars.add('D');
        chars.add('E');

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

```
        // Reversing the list
        Collections.reverse(chars);

        System.out.println("Reversed List: " + chars);
    }
}
```

6. Update an Element Write a program to:

•Create an ArrayList of subjects.

•Replace one of the subjects (e.g., “Math” to “Statistics”).

•Print the list before and after the update.

Answer

```
import java.util.ArrayList;

public class UpdateElement {
    public static void main(String[] args) {
        ArrayList<String> subjects = new ArrayList<>();

        // Adding subjects
        subjects.add("Math");
        subjects.add("Science");
        subjects.add("English");
        subjects.add("History");

        System.out.println("Before Update: " + subjects);
        int index = subjects.indexOf("Math");
        if (index != -1) {
            subjects.set(index, "Statistics");
```

```
        }

        System.out.println("After Update: " + subjects);
    }
}
```

7. Remove All Elements Write a program to:

- Create an ArrayList of integers.
- Add multiple elements.
- Remove all elements using clear() method.
- Display the size of the list.

Answer

```
import java.util.ArrayList;

public class ClearArrayList {
    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("Before clear: " + numbers);

        numbers.clear();

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

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

8. Iterate using Iterator Write a program to:

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

Answer

```
import java.util.ArrayList;
import java.util.Iterator;

public class IterateArrayList {
    public static void main(String[] args) {
        ArrayList<String> cities = new ArrayList<>();

        // Adding cities
        cities.add("Delhi");
        cities.add("Mumbai");
        cities.add("Bangalore");
        cities.add("Hyderabad");
        cities.add("Pune");

        Iterator<String> it = cities.iterator();
        System.out.println("List of Cities:");
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

9. Store Custom Objects Write a program to:

•Create a class Student with fields: id, name, and marks.

•Create an ArrayList of Student objects.

•Add at least 3 students.

•Display the details using a loop.

Answer import java.util.ArrayList;

```
// Student class
class Student {
    int id;
    String name;
    double marks;

    // Constructor
    Student(int id, String name, double marks) {
        this.id = id;
        this.name = name;
        this.marks = marks;
    }

    // Method to display student details
    void display() {
        System.out.println("ID: " + id + ", Name: " + name + ", Marks: " + marks);
    }
}

public class StudentArrayList {
```

```
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student(1, "Samarth", 85.5));
        students.add(new Student(2, "Vithika", 92.0));
        students.add(new Student(3, "Rohit", 76.8));
        System.out.println("Student Details:");
        for (Student s : students) {
            s.display();
        }
    }
}
```

10. Copy One ArrayList to Another Write a program to:

•Create an ArrayList with some elements.

•Create a second ArrayList.

•Copy all elements from the first to the second using addAll() method.

Answer

```
import java.util.ArrayList;

public class CopyArrayList {
    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 (Copied): " + 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.

Answer

```
import java.util.LinkedList;

public class LinkedListColors {
    public static void main(String[] args) {
        LinkedList<String> colors = new LinkedList<>();

        // Adding 5 colors
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add("Yellow");
        colors.add("Black");
```

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

---

2. Add Elements at First and Last Position

Write a program to:

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

Answer

```
import java.util.LinkedList;

public class LinkedListAddFirstLast {
  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: " + numbers);
    }
}
```

---

3. Insert Element at Specific Position

Write a program to:

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

Answer

```
import java.util.LinkedList;

public class LinkedListInsert {
    public static void main(String[] args) {
        LinkedList<String> names = new LinkedList<>();
        names.add("Amit");
        names.add("Rohit");
        names.add("Priya");
        names.add("Sneha");

        System.out.println("Before Insertion: " + names);
        names.add(2, "Samarth");

        System.out.println("After Insertion at index 2: " + names);
    }
}
```

---

4. Remove Elements

Write a program to:

- Create a LinkedList of animal names.
- Remove the first and last elements.
- Remove a specific element by value.
- Display the list after each removal.

Answer

```
import java.util.LinkedList;


public class LinkedListRemove {
   public static void main(String[] args) {
      LinkedList<String> animals = new LinkedList<>();

      // Adding animals
      animals.add("Dog");
      animals.add("Cat");
      animals.add("Lion");
      animals.add("Tiger");
      animals.add("Elephant");

      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("Lion");
        System.out.println("After removing 'Lion': " + animals);
    }
}
```

---

5. Search for an Element

Write a program to:

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

Answer

```
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("Mango");
        list.add("Orange");
        list.add("Grapes");

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a fruit to search: ");
        String search = sc.nextLine();
```

```
        if (list.contains(search)) {
            System.out.println(search + " is found in the list.");
        } else {
            System.out.println(search + " is not found in the list.");
        }

        sc.close();
    }
}
```

---

6. Iterate using ListIterator

Write a program to:

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

Answer

```
import java.util.LinkedList;
import java.util.ListIterator;

public class LinkedListListIterator {
    public static void main(String[] args) {
        LinkedList<String> cities = new LinkedList<>();
        cities.add("Delhi");
        cities.add("Mumbai");
        cities.add("Chennai");
        cities.add("Kolkata");

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

```
        ListIterator<String> it = cities.listIterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }

        System.out.println("\nReverse Direction:");
        while (it.hasPrevious()) {
            System.out.println(it.previous());
        }
    }
}
```

---

7. Sort a LinkedList

Write a program to:

- Create a LinkedList of integers.
- Add unsorted numbers.
- Sort the list using Collections.sort().
- Display the sorted list.

Answer

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

public class LinkedListSort {
    public static void main(String[] args) {
        LinkedList<Integer> numbers = new LinkedList<>();
        numbers.add(45);
        numbers.add(10);
```

```
        numbers.add(78);
        numbers.add(23);
        numbers.add(56);

        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.

Answer

```
import java.util.ArrayList;
import java.util.LinkedList;

public class LinkedListToArrayList {
    public static void main(String[] args) {
        LinkedList<String> linkedList = new LinkedList<>();
        linkedList.add("Red");
        linkedList.add("Blue");
        linkedList.add("Green");
```

```
        System.out.println("LinkedList: " + linkedList);

        ArrayList<String> arrayList = new ArrayList<>(linkedList);

        System.out.println("ArrayList: " + arrayList);
    }
}
```

---

9. Store Custom Objects in LinkedList

Write a program to:

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

Answer

```
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 LinkedListBooks {
  public static void main(String[] args) {
    LinkedList<Book> books = new LinkedList<>();

    books.add(new Book(1, "The Alchemist", "Paulo Coelho"));
    books.add(new Book(2, "Wings of Fire", "A.P.J. Abdul Kalam"));
    books.add(new Book(3, "Ramayana", "Valmiki"));

    System.out.println("Book Details:");
    for (Book b : books) {
      System.out.println("ID: " + b.id + ", Title: " + b.title + ", Author: " + b.author);
    }
  }
}
```

---

10. Clone a LinkedList

Write a program to:

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

Answer

```
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);
        original.add(40);

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

        @SuppressWarnings("unchecked")
        LinkedList<Integer> cloned = (LinkedList<Integer>) original.clone();

        System.out.println("Cloned List: " + cloned);
    }
}
```

Vector

- **Create a Vector of integers** and perform the following operations:
- Add 5 integers to the Vector.
- Insert an element at the 3rd position.
- Remove the 2nd element.

Display the elements using Enumeration.

Answer

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

public class VectorIntegers {
    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.add(2, 99);
        System.out.println("After inserting 99 at 3rd position: " + numbers);



        numbers.remove(1);
        System.out.println("After removing 2nd element: " + numbers);

        System.out.println("Elements using Enumeration:");
        Enumeration<Integer> e = numbers.elements();
        while (e.hasMoreElements()) {
           System.out.println(e.nextElement());
        }
    }
}
```

- **Create a Vector of Strings** and:
- Add at least 4 names.
- Check if a specific name exists in the vector.
- Replace one name with another.
- Clear all elements from the vector.

Answer

```
import java.util.Vector;

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

    names.add("Samarth");
    names.add("Vithika");
    names.add("Rahul");
    names.add("Priya");

    System.out.println("Initial Names: " + names);

    String searchName = "Rahul";
    if (names.contains(searchName)) {
      System.out.println(searchName + " exists in the Vector.");
    } else {
      System.out.println(searchName + " not found in the Vector.");
    }

    int index = names.indexOf("Priya");
    if (index != -1) {
      names.set(index, "Ananya");
      System.out.println("After replacing Priya with Ananya: " + names);
    }

    names.clear();
    System.out.println("After clearing all elements: " + names);
  }
}
```

- **Write a program** to:
- Copy all elements from one Vector to another Vector.
- Compare both vectors for equality.
- Answer

```
import java.util.Vector;

public class CopyCompareVector {
  public static void main(String[] args) {
    // First Vector
```

```
        Vector<String> v1 = new Vector<>();
        v1.add("A");
        v1.add("B");
        v1.add("C");

        // Second Vector
        Vector<String> v2 = new Vector<>();
        v2.addAll(v1);  // Copy all elements

        System.out.println("Vector 1: " + v1);
        System.out.println("Vector 2 (Copied): " + v2);

        // Compare both vectors
        if (v1.equals(v2)) {
            System.out.println("Both vectors are equal.");
        } else {
            System.out.println("Vectors are not equal.");
        }
    }
}
```

- **Write a method** that takes a Vector<Integer> and returns the **sum of all elements**.

Answer

```
import java.util.Vector;

public class SumVector {
    public static int sumOfVector(Vector<Integer> v) {
        int sum = 0;
        for (int num : v) {
            sum += num;
        }
        return sum;
    }

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

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

        System.out.println("Vector: " + numbers);
        System.out.println("Sum of elements: " + sumOfVector(numbers));
    }
}
```

-

Stack

- Understand how to use the Stack class for LIFO (Last In, First Out) operations.
- Answer The Stack class in Java (from java.util) works on **Last In, First Out (LIFO)** principle.
- **push()** → adds element on top
- **pop()** → removes top element
- **peek()** → checks top without removing
- **empty()** → checks if stack is empty
-

- **Create a Stack of integers** and:
- Push 5 elements.
- Pop the top element.
- Peek the current top.
- Check if the stack is empty.

  Answer

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

```
public class StackIntegers {
  public static void main(String[] args) {
    Stack<Integer> stack = new Stack<>();

    // Push 5 elements
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);

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

    // Pop top element
    int popped = stack.pop();
    System.out.println("Popped Element: " + popped);

    // Peek top element
    System.out.println("Top Element after pop: " + stack.peek());

    // Check if stack is empty
    System.out.println("Is stack empty? " + stack.empty());
  }
}
```

- **Reverse a string using Stack**:
- Input a string from the user.
- Use a stack to reverse and print the string
- Answer

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


public class ReverseStringStack {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String input = sc.nextLine();
```

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

        // Push all characters
        for (char c : input.toCharArray()) {
            stack.push(c);
        }

        // Pop to reverse
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }

        System.out.println("Reversed String: " + reversed);
        sc.close();
    }
}
```

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

```
import java.util.Stack;

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

```
        for (char ch : expr.toCharArray()) {
            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            } else if (ch == ')' || ch == '}' || ch == ']') {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((ch == ')' && top != '(') ||
                    (ch == '}' && top != '{') ||
                    (ch == ']' && top != '[')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }

    public static void main(String[] args) {
        String expr = "(a+b) * (c-d)";
        if (isBalanced(expr))
            System.out.println("Valid Expression");
        else
            System.out.println("Invalid Expression");
    }
}
```

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

Answer

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

```
import java.util.Scanner;

public class DecimalToBinary {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter a decimal number: ");
      int num = sc.nextInt();

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

      while (num > 0) {
         stack.push(num % 2);
         num /= 2;
      }

      System.out.print("Binary: ");
      while (!stack.isEmpty()) {
         System.out.print(stack.pop());
      }
      sc.close();
   }
}
```

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.

Answer

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

public class HashSetCities {
  public static void main(String[] args) {
    HashSet<String> cities = new HashSet<>();

    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Pune");
    cities.add("Bangalore");
    cities.add("Chennai");

    // Adding duplicate
    cities.add("Delhi");

    System.out.println("HashSet (no duplicates): " + cities);

    // Iterate using Iterator
    System.out.println("Cities:");
    Iterator<String> it = cities.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
  }
}
```

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

Answer

```
import java.util.HashSet;

public class HashSetOperations {
  public static void main(String[] args) {
    HashSet<String> cities = new HashSet<>();
    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Kolkata");
```

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

        // Remove element
        cities.remove("Mumbai");
        System.out.println("After removing Mumbai: " + cities);

        // Check existence
        System.out.println("Contains Delhi? " + cities.contains("Delhi"));

        // Clear all
        cities.clear();
        System.out.println("After clearing: " + cities);
    }
}
```

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

   Answer

```
import java.util.HashSet;

public class MaxHashSet {
    public static int getMax(HashSet<Integer> set) {
        int max = Integer.MIN_VALUE;
        for (int num : set) {
            if (num > max) max = num;
        }
        return max;
    }

    public static void main(String[] args) {
        HashSet<Integer> numbers = new HashSet<>();
        numbers.add(10);
        numbers.add(50);
        numbers.add(30);
        numbers.add(20);

        System.out.println("Numbers: " + numbers);
        System.out.println("Maximum Element: " + getMax(numbers));
    }
}
```

LinkedHashSet

1.Create a LinkedHashSet of Integers:

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

Answer

```
import java.util.LinkedHashSet;


public class LinkedHashSetExample {
   public static void main(String[] args) {
      LinkedHashSet<Integer> set = new LinkedHashSet<>();


      set.add(10);
      set.add(5);
      set.add(20);
      set.add(15);
      set.add(5); // duplicate


      System.out.println("LinkedHashSet: " + set);
   }
}
```

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

   Answer

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

```
class Student {
  private int id;
  private String name;

  // Constructor
  public Student(int id, String name) {
    this.id = id;
    this.name = name;
  }

  // Override equals()
  @Override
  public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Student student = (Student) obj;
    return id == student.id && Objects.equals(name, student.name);
  }

  // Override hashCode()
  @Override
  public int hashCode() {
    return Objects.hash(id, name);
  }

  // For printing
  @Override
  public String toString() {
    return "Student{id=" + id + ", name='" + name + "'}";
  }
}

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

    // Add students
    students.add(new Student(1, "Samarth"));
    students.add(new Student(2, "Vithika"));
    students.add(new Student(3, "Rahul"));

    // Duplicate student
    students.add(new Student(1, "Samarth"));
```

```
        // Print students
        System.out.println("Students in LinkedHashSet: " + students);
    }
}
```

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

Answer

```
import java.util.LinkedHashSet;

public class MergeLinkedHashSet {
    public static void main(String[] args) {
        LinkedHashSet<String> set1 = new LinkedHashSet<>();
        set1.add("Delhi");
        set1.add("Mumbai");
        set1.add("Pune");

        LinkedHashSet<String> set2 = new LinkedHashSet<>();
        set2.add("Chennai");
        set2.add("Bangalore");
        set2.add("Pune");  // duplicate

        // Merge
        set1.addAll(set2);

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

TreeSet

1. Create a TreeSet of Strings:

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

Answer

```
import java.util.TreeSet;


public class TreeSetStringExample {
```

```
    public static void main(String[] args) {
        TreeSet<String> countries = new TreeSet<>();

        // Add in random order
        countries.add("India");
        countries.add("USA");
        countries.add("Japan");
        countries.add("Australia");
        countries.add("Brazil");

        // Print sorted countries
        System.out.println("Sorted countries: " + countries);
    }
}
```

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

Answer

```
import java.util.TreeSet;

public class TreeSetIntegerExample {
    public static void main(String[] args) {
        TreeSet<Integer> numbers = new TreeSet<>();
        numbers.add(50);
        numbers.add(20);
        numbers.add(10);
        numbers.add(70);
        numbers.add(40);
```

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

        int check = 40;
        System.out.println("Lower than " + check + ": " + numbers.lower(check));
        System.out.println("Higher than " + check + ": " + numbers.higher(check));
    }
}
```

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

Answer

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

public class TreeSetCustomComparator {
    public static void main(String[] args) {
        TreeSet<String> cities = new TreeSet<>(Comparator.reverseOrder());

        cities.add("Delhi");
        cities.add("Mumbai");
        cities.add("Chennai");
        cities.add("Kolkata");

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

Queue

1. **Bank Queue Simulation**:
    - Create a queue of customer names using Queue<String>.

- Add 5 customers to the queue.
- Serve (remove) customers one by one and print the queue after each removal.

Answer

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

public class BankQueueSimulation {
  public static void main(String[] args) {
    Queue<String> bankQueue = new LinkedList<>();

    bankQueue.add("Customer1");
    bankQueue.add("Customer2");
    bankQueue.add("Customer3");
    bankQueue.add("Customer4");
    bankQueue.add("Customer5");

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

    while (!bankQueue.isEmpty()) {
      String served = bankQueue.poll(); // removes head
      System.out.println("Serving: " + served);
      System.out.println("Remaining Queue: " + bankQueue);
    }
  }
}
```

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

Answer

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

public class TaskManager {
  public static void main(String[] args) {
    Queue<String> tasks = new LinkedList<>();

    tasks.add("Task1");
    tasks.add("Task2");
    tasks.add("Task3");
```

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

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

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

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

Answer

```
import java.util.*;

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

    public static void main(String[] args) {
        Queue<Integer> numbers = new LinkedList<>();
        numbers.add(11);
        numbers.add(20);
        numbers.add(35);
        numbers.add(42);
        numbers.add(56);

        System.out.println("Original Queue: " + numbers);
        List<Integer> evens = getEvenNumbers(numbers);
        System.out.println("Even numbers: " + evens);
    }
```

```
}
```

PriorityQueue

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

Answer

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

class Patient {
  String name;
  int severityLevel;

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

  @Override
  public String toString() {
    return name + " (Severity: " + severityLevel + ")";
  }
}

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

```
    // Highest severity served first
    PriorityQueue<Patient> emergencyQueue = new PriorityQueue<>(
      Comparator.comparingInt((Patient p) -> p.severityLevel).reversed()
    );

    emergencyQueue.add(new Patient("Patient1", 2));
    emergencyQueue.add(new Patient("Patient2", 5));
    emergencyQueue.add(new Patient("Patient3", 3));
    emergencyQueue.add(new Patient("Patient4", 1));

    System.out.println("Serving patients by severity:");
    while (!emergencyQueue.isEmpty()) {
      System.out.println("Serving: " + emergencyQueue.poll());
    }
  }
}
```

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

Answer

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

class PrintJob {
  String jobName;
  int priority; // Higher = more important

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

```
  @Override
  public String toString() {
    return jobName + " (Priority: " + priority + ")";
  }
}

public class PrintJobQueue {
  public static void main(String[] args) {
    PriorityQueue<PrintJob> printQueue = new PriorityQueue<>(
      Comparator.comparingInt((PrintJob j) -> j.priority).reversed()
    );

    printQueue.add(new PrintJob("Document1", 2));
    printQueue.add(new PrintJob("Document2", 5));
    printQueue.add(new PrintJob("Document3", 1));
    printQueue.add(new PrintJob("Document4", 4));

    System.out.println("Processing print jobs by 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.

Answer

```
import java.util.*;

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

  public static void main(String[] args) {
    Queue<Integer> numbers = new LinkedList<>();
```

```
        numbers.add(11);
        numbers.add(20);
        numbers.add(35);
        numbers.add(42);
        numbers.add(56);

        System.out.println("Original Queue: " + numbers);
        List<Integer> evens = getEvenNumbers(numbers);
        System.out.println("Even numbers: " + evens);
    }
}
```

Deque

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

Answer

```
import java.util.*;


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


    // Add all characters to deque
    for (char c : str.toCharArray()) {
      if (Character.isLetterOrDigit(c)) {
        deque.add(Character.toLowerCase(c));
      }
    }


    while (deque.size() > 1) {
      if (deque.removeFirst() != deque.removeLast()) {
```

```
            return false;
          }
        }
        return true;
    }

    public static void main(String[] args) {
        String input = "Madam";
        System.out.println(input + " is Palindrome? " + isPalindrome(input));
    }
}
```

- .

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

Answer

```
import java.util.*;

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

        // Add items from front and rear
        orders.addFirst("Order1");
        orders.addLast("Order2");
        orders.addFirst("Order3");
        orders.addLast("Order4");
```

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

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

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

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

Answer

```
import java.util.*;

public class BrowserHistory {
  Deque<String> backStack = new ArrayDeque<>();
  Deque<String> forwardStack = new ArrayDeque<>();
  String currentPage = "Home";

  public void visit(String page) {
    backStack.push(currentPage);
    currentPage = page;
    forwardStack.clear(); // once a new page is visited, forward history clears
    System.out.println("Visited: " + currentPage);
  }

  public void back() {
    if (!backStack.isEmpty()) {
```

```
        forwardStack.push(currentPage);
        currentPage = backStack.pop();
        System.out.println("Back to: " + currentPage);
    } else {
        System.out.println("No pages in back history.");
    }
}

public void forward() {
    if (!forwardStack.isEmpty()) {
        backStack.push(currentPage);
        currentPage = forwardStack.pop();
        System.out.println("Forward to: " + currentPage);
    } else {
        System.out.println("No pages in forward history.");
    }
}

public static void main(String[] args) {
    BrowserHistory browser = new BrowserHistory();

    browser.visit("Google");
    browser.visit("YouTube");
    browser.visit("GitHub");

    browser.back();
    browser.back();
    browser.forward();
    browser.visit("StackOverflow");
```

```
        browser.forward();
    }
}
```