Collections

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.

A. package collections;

```
import java.util.ArrayList;
import java.util.Scanner;
public class SearchElement {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number to search: ");
        int num = sc.nextInt();
        if (list.contains(num)) {
            System.out.println("Found");
        } else {
```

```
            System.out.println("Not Found");
        }
    }
}
```

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.

A.

```
package collections;
import java.util.ArrayList;
public class RemoveFruit {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Grapes");
        fruits.add("Orange");
        fruits.remove("Mango");
        System.out.println("Updated List: " + 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.

A. package collections;

```
import java.util.ArrayList;
import java.util.Collections;
public class SortArrayList {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(45);
        numbers.add(12);
        numbers.add(78);
        numbers.add(34);
        numbers.add(9);
        numbers.add(66);
        numbers.add(23);
        Collections.sort(numbers);
        System.out.println("Sorted List: " + 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.

A.
```
package collections;
import java.util.ArrayList;
import java.util.Collections;
public class ReverseArrayList {
    public static void main(String[] args) {
        ArrayList<Character> chars = new ArrayList<>();
        chars.add('A');
        chars.add('B');
        chars.add('C');
        chars.add('D');
        chars.add('E');
        Collections.reverse(chars);
        System.out.println("Reversed List: " + chars);
    }
}
```

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.

A. package collections;

```
import java.util.ArrayList;
public class UpdateSubject {
    public static void main(String[] args) {
        ArrayList<String> subjects = new ArrayList<>();
        subjects.add("Math");
        subjects.add("Science");
        subjects.add("English");
        subjects.add("History");
        System.out.println("Before: " + subjects);
        int index = subjects.indexOf("Math");
        if (index != -1) {
            subjects.set(index, "Statistics");
        }
        System.out.println("After: " + 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.

A. package collections;

```
import java.util.ArrayList;
public class ClearArrayList {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        list.clear();
        System.out.println("Size after clear: " + list.size());
    }
}
```

8. Iterate using Iterator

Write a program to:

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

A.
```
package collections;
import java.util.ArrayList;
import java.util.Iterator;
public class CityIterator {
    public static void main(String[] args) {
        ArrayList<String> cities = new ArrayList<>();
        cities.add("Delhi");
```

```
        cities.add("Mumbai");
        cities.add("Chennai");
        cities.add("Kolkata");
        Iterator<String> itr = cities.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.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.

A.

```
package collections;
import java.util.ArrayList;
class Student {
    int id;
    String name;
    int marks;
    Student(int id, String name, int marks) {
        this.id = id;
        this.name = name;
```

```
        this.marks = marks;
    }
}
public class StudentList {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student(1, "Suma", 90));
        students.add(new Student(2, "Latha", 85));
        students.add(new Student(3, "Bavya", 95));
        for (Student s : students) {
            System.out.println("ID: " + s.id + ", Name: " + s.name + ",
Marks: " + s.marks);
        }
    }
}
```

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.

A.
```
package collections;
import java.util.ArrayList;
```

```
public class CopyArrayList {
    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();
        list1.add("Red");
        list1.add("Green");
        list1.add("Blue");
        ArrayList<String> list2 = new ArrayList<>();
        list2.addAll(list1);
        System.out.println("List1: " + list1);
        System.out.println("List2: " + list2);
    }
}
```

List(LinkedList)

1. Create and Display a LinkedList

Write a program to:

- Create a LinkedList of Strings.
- Add five colors to it.
- Display the list using a for-each loop.

A.
```
package collections;
import java.util.LinkedList;
public class LinkedListCreateDisplay {
    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("Pink");
        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.

A.
```
package collections;
import java.util.LinkedList;
public class AddFirstLast {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        list.add(10);
        list.add(20);
        list.add(30);

        list.addFirst(5);
```

```
        list.addLast(40);
        System.out.println("Updated List: " + list);
    }
}
```

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.

A.
```
package collections;
import java.util.LinkedList;
public class InsertAtPosition {
    public static void main(String[] args) {
        LinkedList<String> names = new LinkedList<>();
        names.add("Suma");
        names.add("Latha");
        names.add("Neha");
        System.out.println("Before: " + names);
        names.add(2, "Bavya");
        System.out.println("After: " + 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.

A. package collections;

```
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.add("Lion");
        animals.removeFirst();
        animals.removeLast();
        animals.remove("Cat");
        System.out.println("Remaining: " + 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.

A.
```
package collections;
import java.util.LinkedList;
import java.util.Scanner;
public class SearchLinkedList {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        list.add("Delhi");
        list.add("Mumbai");
        list.add("Chennai");
        list.add("Kolkata");
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter city to search: ");
        String city = sc.nextLine();
        if (list.contains(city)) {
            System.out.println("City found");
        } else {
            System.out.println("City not 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.

A.
```
package collections;
import java.util.LinkedList;
import java.util.ListIterator;
public class ListIteratorExample {
    public static void main(String[] args) {
        LinkedList<String> cities = new LinkedList<>();
        cities.add("Delhi");
        cities.add("Hyderabad");
        cities.add("Bangalore");
        cities.add("Mumbai");
        ListIterator<String> itr = cities.listIterator();
        System.out.println("Forward:");
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
        System.out.println("Reverse:");
        while (itr.hasPrevious()) {
            System.out.println(itr.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.

A. package collections;

```
import java.util.Collections;
import java.util.LinkedList;
public class SortLinkedList {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        list.add(45);
        list.add(10);
        list.add(78);
        list.add(32);
        list.add(5);
        Collections.sort(list);
        System.out.println("Sorted List: " + list);
    }
}
```

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.

A. package collections;

```
import java.util.ArrayList;
import java.util.LinkedList;
public class LinkedListToArrayList {
    public static void main(String[] args) {
        LinkedList<String> linkedList = new LinkedList<>();
        linkedList.add("Apple");
        linkedList.add("Banana");
        linkedList.add("Cherry");
        ArrayList<String> arrayList = new ArrayList<>(linkedList);
        System.out.println("LinkedList: " + linkedList);
        System.out.println("ArrayList: " + arrayList);
    }
}
```

9. Store Custom Objects in LinkedList

Write a program to:

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

A. package collections;

```
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 BookList {
    public static void main(String[] args) {
        LinkedList<Book> books = new LinkedList<>();
        books.add(new Book(1, "Java Basics", "Suma"));
        books.add(new Book(2, "Data Structures", "Latha"));
        books.add(new Book(3, "Algorithms", "Bavya"));
        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.

A.
```
package collections;
import java.util.LinkedList;
public class CloneLinkedList {
    public static void main(String[] args) {
        LinkedList<Integer> original = new LinkedList<>();
        original.add(10);
        original.add(20);
        original.add(30);
        LinkedList<Integer> cloned = (LinkedList<Integer>) original.clone();
        System.out.println("Original: " + original);
        System.out.println("Cloned: " + cloned);
    }
}
```

Vector

1.Create a Vector of integers and perform the following operations:

- Add 5 integers to the Vector.
- Insert an element at the 3rd position.
- Remove the 2nd element.
- Display the elements using Enumeration.

A.
```
package day8;
import java.util.Vector;
import java.util.Enumeration;
```

```
public class VectorIntegerOperations {
    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.insertElementAt(25, 2);
        numbers.removeElementAt(1);
        Enumeration<Integer> enumeration = numbers.elements();
        while(enumeration.hasMoreElements()) {
            System.out.print(enumeration.nextElement() + " ");
        }
    }
}
```

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.

A.
```
package day8;
import java.util.Vector;

public class VectorStringOperations {
```

```
    public static void main(String[] args) {
        Vector<String> names = new Vector<>();
        names.add("suma");
        names.add("latha");
        names.add("bavya");
        names.add("shree");
        System.out.println("Original Vector: " + names);
        if(names.contains("bavya")) {
            System.out.println("bavya exists in the Vector");
        } else {
            System.out.println("bavya does not exist in the Vector");
        }
        names.set(names.indexOf("latha"), "lakshmi");
        System.out.println("After replacing latha with lakshmi: " +
names);
        names.clear();
        System.out.println("After clearing all elements: " + names);
    }
}
```

3.**Write a program** to:

- Copy all elements from one Vector to another Vector.
- Compare both vectors for equality.

A.
```
package day8;
import java.util.Vector;
public class VectorCopyAndCompare {
```

```
    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 (copied): " + vector2);
        if (vector1.equals(vector2)) {
            System.out.println("Both vectors are equal.");
        } else {
            System.out.println("Vectors are not equal.");
        }
    }
}
```

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

A.
```
package day8;
import java.util.Vector;
public class VectorSum {
    public static void main(String[] args) {
        Vector<Integer> numbers = new Vector<>();
        numbers.add(10);
```

```
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);
        int total = sumVector(numbers);
        System.out.println("Sum of elements: " + total);
    }
    public static int sumVector(Vector<Integer> vec) {
        int sum = 0;
        for (int num : vec) {
            sum += num;
        }
        return sum;
    }
}
```

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.

A.

```
package day8;
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 popped = stack.pop();
        System.out.println("Popped element: " + popped);
        int top = stack.peek();
        System.out.println("Current top element: " + top);
        boolean empty = stack.isEmpty();
        System.out.println("Is stack empty? " + empty);
        System.out.println("Final stack: " + stack);
    }
}
```

2.**Reverse a string using Stack**:

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

A.
```
package day8;
import java.util.Scanner;
import java.util.Stack;
```

```
public class ReverseStringUsingStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string to reverse: ");
        String input = sc.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.toString());
        sc.close();
    }
}
```

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

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

A.
```
package day8;
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 == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
                    return false;
                }
                stack.pop();
            }
        }
        return stack.isEmpty();
    }
    public static void main(String[] args) {
        String expression = "(a+b) * (c-d)";
        if (isBalanced(expression)) {
            System.out.println("Valid expression");
        } else {
            System.out.println("Invalid expression");
        }
    }
}
```

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

A.

```
package day8;
import java.util.Scanner;
```

```
import java.util.Stack;
public class DecimalToBinary {
    public static String convertToBinary(int decimal) {
        Stack<Integer> stack = new Stack<>();
        while (decimal > 0) {
            stack.push(decimal % 2);
            decimal /= 2;
        }
        StringBuilder binary = new StringBuilder();
        while (!stack.isEmpty()) {
            binary.append(stack.pop());
        }
        return binary.toString();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a decimal number: ");
        int decimal = sc.nextInt();
        String binary = convertToBinary(decimal);
        System.out.println("Binary equivalent: " + binary);
        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.

A. package day8;

```
import java.util.HashSet;
import java.util.Iterator;
public class HashSetDemo1 {
    public static void main(String[] args) {
        HashSet<String> cities = new HashSet<>();
        cities.add("Mumbai");
        cities.add("Delhi");
        cities.add("Chennai");
        cities.add("Kolkata");
        cities.add("Bangalore");
        boolean added = cities.add("Delhi"); // duplicate
        System.out.println("Added duplicate city? " + added);
        System.out.println("Cities in HashSet:");
        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.

A. package day8;

```
import java.util.HashSet;
public class HashSetOperations {
    public static void main(String[] args) {
        HashSet<String> cities = new HashSet<>();
        cities.add("Mumbai");
        cities.add("Delhi");
        cities.add("Chennai");
        cities.add("Kolkata");
        cities.add("Bangalore");
                System.out.println("Original HashSet: " + cities);
                cities.remove("Chennai");
                System.out.println("After removing Chennai: " + cities);
                boolean exists = cities.contains("Delhi");
                System.out.println("Does Delhi exist? " + exists);
                cities.clear();
                System.out.println("After clearing, size: " + cities.size());
    }
}
```

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

A. package day8;

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

```
import java.util.Collections;
public class HashSetMax {
    public static void main(String[] args) {
        HashSet<Integer> numbers = new HashSet<>();
        numbers.add(10);
        numbers.add(35);
        numbers.add(20);
        numbers.add(5);
        numbers.add(50);
        int max = getMax(numbers);
        System.out.println("Maximum element is: " + max);
    }
    public static int getMax(HashSet<Integer> set) {
        return Collections.max(set);
    }
}
```

LinkedHashSet

1.Create a LinkedHashSet of Integers:

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

A.
```
package day8;
import java.util.LinkedHashSet;
public class LinkedHashSetExample1 {
    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, will not be added
        System.out.println("Elements in LinkedHashSet:");
        for (Integer num : set) {
            System.out.print(num + " ");
        }
    }
}
```

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.

A.

```
package day8;
import java.util.LinkedHashSet;
import java.util.Objects;
class Student {
    int id;
    String name;
    Student(int id, String name) {
        this.id = id;
        this.name = name;
```

```
    }
    @Override
    public boolean equals(Object o) {
        if(this == o) return true;
        if(o == null || getClass() != o.getClass()) return false;
        Student student = (Student) o;
        return id == student.id && Objects.equals(name, student.name);
    }
    @Override
    public int hashCode() {
        return Objects.hash(id, name);
    }
    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "'}";
    }
}
public class LinkedHashSetExample2 {
    public static void main(String[] args) {
        LinkedHashSet<Student> students = new LinkedHashSet<>();
        students.add(new Student(1, "Suma"));
        students.add(new Student(2, "Latha"));
        students.add(new Student(3, "Bavya"));
        students.add(new Student(2, "Latha")); // duplicate, will not be
added
```

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

3.**Write a program** to:

- Merge two LinkedHashSets and print the result.

A.
```
import java.util.LinkedHashSet;
public class MergeLinkedHashSet {
    public static void main(String[] args) {
        LinkedHashSet<Integer> set1 = new LinkedHashSet<>();
        set1.add(10);
        set1.add(20);
        set1.add(30);
        LinkedHashSet<Integer> set2 = new LinkedHashSet<>();
        set2.add(30);
        set2.add(40);
        set2.add(50);
        set1.addAll(set2);
        System.out.println("Merged LinkedHashSet:");
        for (Integer num : set1) {
            System.out.print(num + " ");
        }
    }
```

}

TreeSet

1. Create a TreeSet of Strings:

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

A.
```
import java.util.TreeSet;
public class TreeSetStrings {
    public static void main(String[] args) {
        TreeSet<String> countries = new TreeSet<>();
        countries.add("India");
        countries.add("Brazil");
        countries.add("Australia");
        countries.add("Canada");
        countries.add("Germany");
        System.out.println("Sorted countries:");
        for(String country : countries) {
            System.out.println(country);
        }
    }
}
```

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.

A.
```
package day8;
```

```
import java.util.TreeSet;
public class TreeSetIntegers {
    public static void main(String[] args) {
        TreeSet<Integer> numbers = new TreeSet<>();
        numbers.add(10);
        numbers.add(30);
        numbers.add(20);
        numbers.add(50);
        numbers.add(40);
        System.out.println("First element: " + numbers.first());
        System.out.println("Last element: " + numbers.last());
        int given = 30;
        System.out.println("Lower than " + given + ": " +
numbers.lower(given));
        System.out.println("Higher than " + given + ": " +
numbers.higher(given));
    }
}
```

3.Create a TreeSet with a custom comparator:

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

A.
```
package day8;
import java.util.TreeSet;
import java.util.Comparator;
public class TreeSetCustomComparator {
    public static void main(String[] args) {
```

```
        TreeSet<String> countries = new
TreeSet<>(Comparator.reverseOrder());
        countries.add("India");
        countries.add("Brazil");
        countries.add("Australia");
        countries.add("Canada");
        countries.add("Germany");
        System.out.println("Countries in reverse alphabetical order:");
        for(String country : countries) {
            System.out.println(country);
        }
    }
}
```

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.

A.
```
package day8;
import java.util.LinkedList;
import java.util.Queue;
public class BankQueueSimulation {
    public static void main(String[] args) {
        Queue<String> customers = new LinkedList<>();
```

```
        customers.add("Suma");
        customers.add("Latha");
        customers.add("Bavya");
        customers.add("Shree");
        customers.add("Priya");
        System.out.println("Initial queue: " + customers);
        while (!customers.isEmpty()) {
            String served = customers.poll();
            System.out.println("Served: " + served);
            System.out.println("Queue now: " + customers);
        }
    }
}
```

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

A.
```
package day8;
import java.util.LinkedList;
import java.util.Queue;
public class TaskManager {
    public static void main(String[] args) {
        Queue<String> tasks = new LinkedList<>();
        tasks.add("Wash Dishes");
        tasks.add("Study");
        tasks.add("Go Shopping");
```

```
        tasks.add("Call Mom");
        System.out.println("Next task: " + tasks.peek());
        System.out.println("Completing: " + 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.

A.
```
package day8;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
public class EvenNumberExtractor {
    public static List<Integer> getEven(Queue<Integer> q) {
        List<Integer> evens = new ArrayList<>();
        for (int n : q) {
            if (n % 2 == 0) {
                evens.add(n);
            }
        }
        return evens;
    }
```

```
    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<>();
        q.add(5);
        q.add(12);
        q.add(7);
        q.add(20);
        q.add(3);
        List<Integer> evens = getEven(q);
        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).

A.
```
package day8;
import java.util.PriorityQueue;
import java.util.Comparator;
class Patient {
    String name;
    int severityLevel;
    Patient(String name, int severityLevel) {
        this.name = name;
```

```
        this.severityLevel = severityLevel;
    }
    @Override
    public String toString() {
        return name + "(" + severityLevel + ")";
    }
}
public class EmergencyQueue {
    public static void main(String[] args) {
        PriorityQueue<Patient> pq = new
PriorityQueue<>(Comparator.comparingInt(p -> -p.severityLevel));
        pq.add(new Patient("Suma", 5));
        pq.add(new Patient("Latha", 9));
        pq.add(new Patient("Priya", 3));
        while (!pq.isEmpty()) {
            System.out.println("Serving: " + pq.poll());
        }
    }
}
```

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

A. package day8;

```
import java.util.PriorityQueue;
import java.util.Comparator;
class PrintJob {
    String job;
    int priority;
    PrintJob(String job, int priority) {
        this.job = job;
        this.priority = priority;
    }
    @Override
    public String toString() {
        return job + "(" + priority + ")";
    }
}
public class PrintJobManager {
    public static void main(String[] args) {
        PriorityQueue<PrintJob> pq = new
PriorityQueue<>(Comparator.comparingInt(j -> j.priority));
        pq.add(new PrintJob("Doc1", 5));
        pq.add(new PrintJob("Photo", 1));
        pq.add(new PrintJob("Report", 3));
        while (!pq.isEmpty()) {
            System.out.println("Printing: " + pq.poll());
        }
    }
```

}

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

A. package day8;

```
import java.util.PriorityQueue;

public class MergePriorityQueues {

   public static PriorityQueue<Integer>
mergeQueues(PriorityQueue<Integer> q1, PriorityQueue<Integer>
q2) {

      PriorityQueue<Integer> merged = new PriorityQueue<>();

      merged.addAll(q1);

      merged.addAll(q2);

      return merged;

   }

   public static void main(String[] args) {

      PriorityQueue<Integer> q1 = new PriorityQueue<>();

      q1.add(10);

      q1.add(30);

      q1.add(20);

      PriorityQueue<Integer> q2 = new PriorityQueue<>();

      q2.add(25);

      q2.add(5);

      q2.add(15);

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

      System.out.println("Merged queue sorted:");
```

```
        while (!merged.isEmpty()) {
            System.out.print(merged.poll() + " ");
        }
    }
}
```

Deque

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

A.
```
package day8;
import java.util.Deque;
import java.util.ArrayDeque;
import java.util.Scanner;
public class PalindromeChecker {
    public static boolean isPalindrome(String s) {
        Deque<Character> dq = new ArrayDeque<>();
        for (char c : s.toCharArray()) {
            dq.addLast(c);
        }
        while (dq.size() > 1) {
            if (dq.removeFirst() != dq.removeLast()) {
                return false;
            }
        }
        return true;
```

```
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine().replaceAll("\\s+",
"").toLowerCase();
        System.out.println(isPalindrome(input) ? "Palindrome" : "Not a
palindrome");
        sc.close();
    }
}
```

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

A.
```
package day8;
import java.util.Deque;
import java.util.ArrayDeque;
public class DoubleEndedOrder {
    public static void main(String[] args) {
        Deque<String> deque = new ArrayDeque<>();
        deque.addFirst("Front1");
        deque.addLast("Rear1");
        deque.addFirst("Front2");
        deque.addLast("Rear2");
```

```
        System.out.println("After adds: " + deque);
        deque.removeFirst();
        deque.removeLast();
        System.out.println("After removals: " + deque);
    }
}
```

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

A.
```
package day8;
import java.util.Deque;
import java.util.ArrayDeque;
public class BrowserHistory {
    public static void main(String[] args) {
        Deque<String> backStack = new ArrayDeque<>();
        Deque<String> forwardStack = new ArrayDeque<>();
        String currentPage = "home";
        backStack.push(currentPage);
        currentPage = "page1";
        backStack.push(currentPage);
        currentPage = "page2";
        System.out.println("Current: " + currentPage);
        // navigate back
        forwardStack.push(currentPage);
        currentPage = backStack.pop();
```

```
        System.out.println("Back to: " + currentPage);
        // navigate forward
        backStack.push(currentPage);
        currentPage = forwardStack.pop();
        System.out.println("Forward to: " + currentPage);
    }
}
```