**ID:hrajranj**

**Day 14 – 2nd July 2025**

**Task 1:**

**Create  a custom node , add elements to it and traverse it**.

class Node<T> {

    T data;

    Node<T> next;

    public Node(T data) {

        this.data = data;

        this.next = null;

    }

}

class MyLinkedList<T> {

    private Node<T> head;

    // Add element at the end of the list

    public void add(T data) {

        Node<T> newNode = new Node<>(data);

        if (head == null) {

            head = newNode;

        } else {

            Node<T> current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

    }

    // Traverse and display the list

    public void traverse() {

        Node<T> current = head;

        System.out.println("Linked List elements:");

        while (current != null) {

            System.out.println(current.data);

            current = current.next;

        }

    }

}

public class Task1 {

    public static void main(String[] args) {

        MyLinkedList<String> list = new MyLinkedList<>();

        // Add elements using custom add method

        list.add("Apple");

        list.add("Banana");

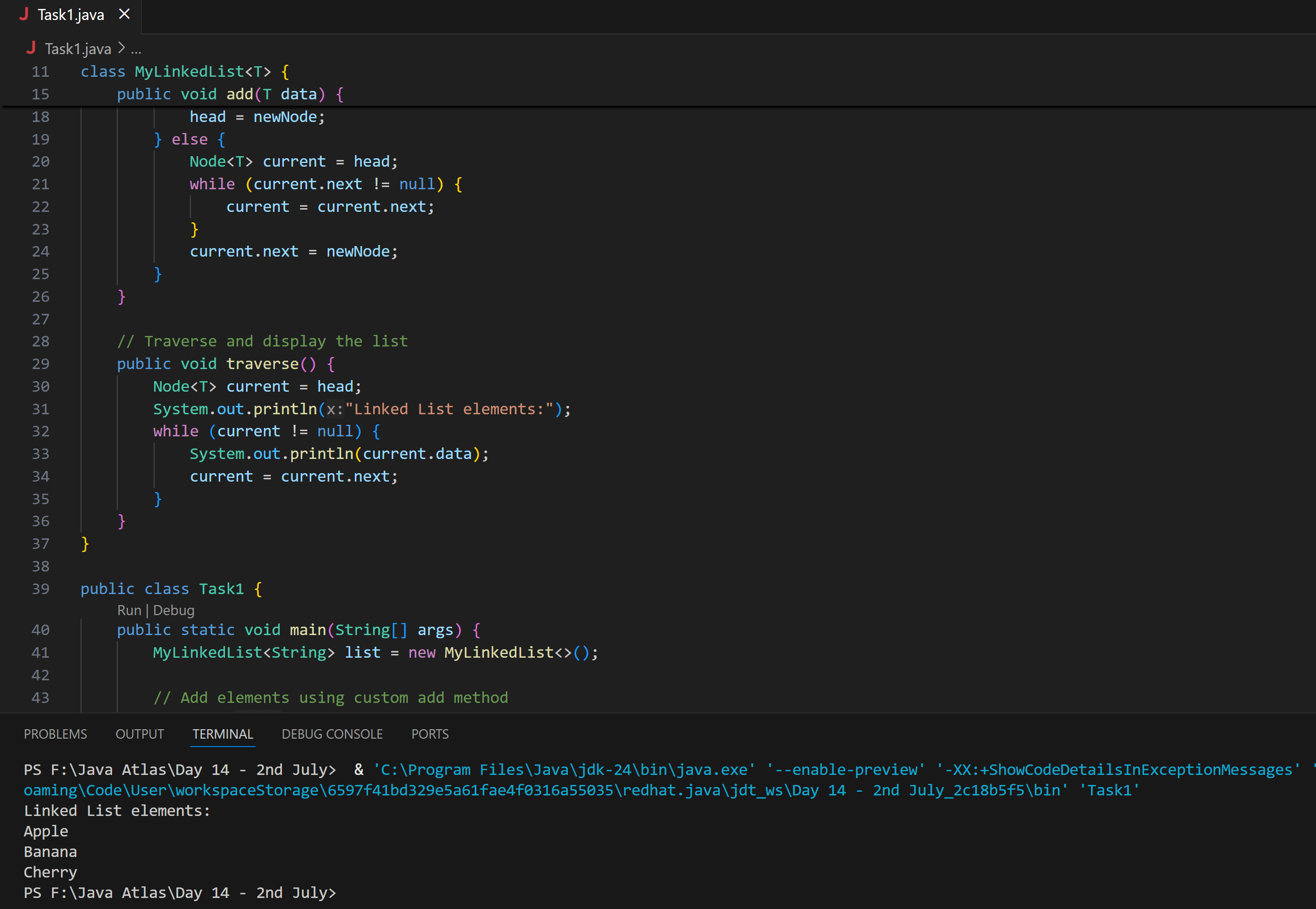
        list.add("Cherry");

        // Display the elements

        list.traverse();

    }

}

****

**Task 2**

**What do you understand by traversing elements in a linked list.**

Traversing elements in a linked list means visiting each node in the list, typically starting from the head node and moving sequentially to the next node until the end of the list is reached. This process allows us to access and possibly manipulate the data stored in each node.

**Key Points of Traversing a Linked List:**

1. **Starting Point**: Begin at the head node.
2. **Sequential Access**: Move from one node to the next using the next reference.
3. **End Condition**: Continue until the next reference is None, indicating the end of the list.

**Task 3:**

**Create a Circular Linked list using Task 1 Singly linked list.**

public class Task3 {

}

class Node {

    String data;

    Node next;

    Node(String data) {

        this.data = data;

        this.next = null;

    }

}

class CircularLinkedList {

    Node head = null;

    Node tail = null;

    // Add node at end

    public void add(String data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

            tail = newNode;

            newNode.next = head; // Circular link

        } else {

            tail.next = newNode;

            tail = newNode;

            tail.next = head; // Maintain circular link

        }

    }

    // Traverse and print all elements

    public void traverse() {

        if (head == null) {

            System.out.println("List is empty.");

            return;

        }

        Node current = head;

        System.out.println("Circular Linked List:");

        do {

            System.out.println(current.data);

            current = current.next;

        } while (current != head);

    }

}

public class Task1 {

    public static void main(String[] args) {

        CircularLinkedList cll = new CircularLinkedList();

        // Add elements

        cll.add("Apple");

        cll.add("Banana");

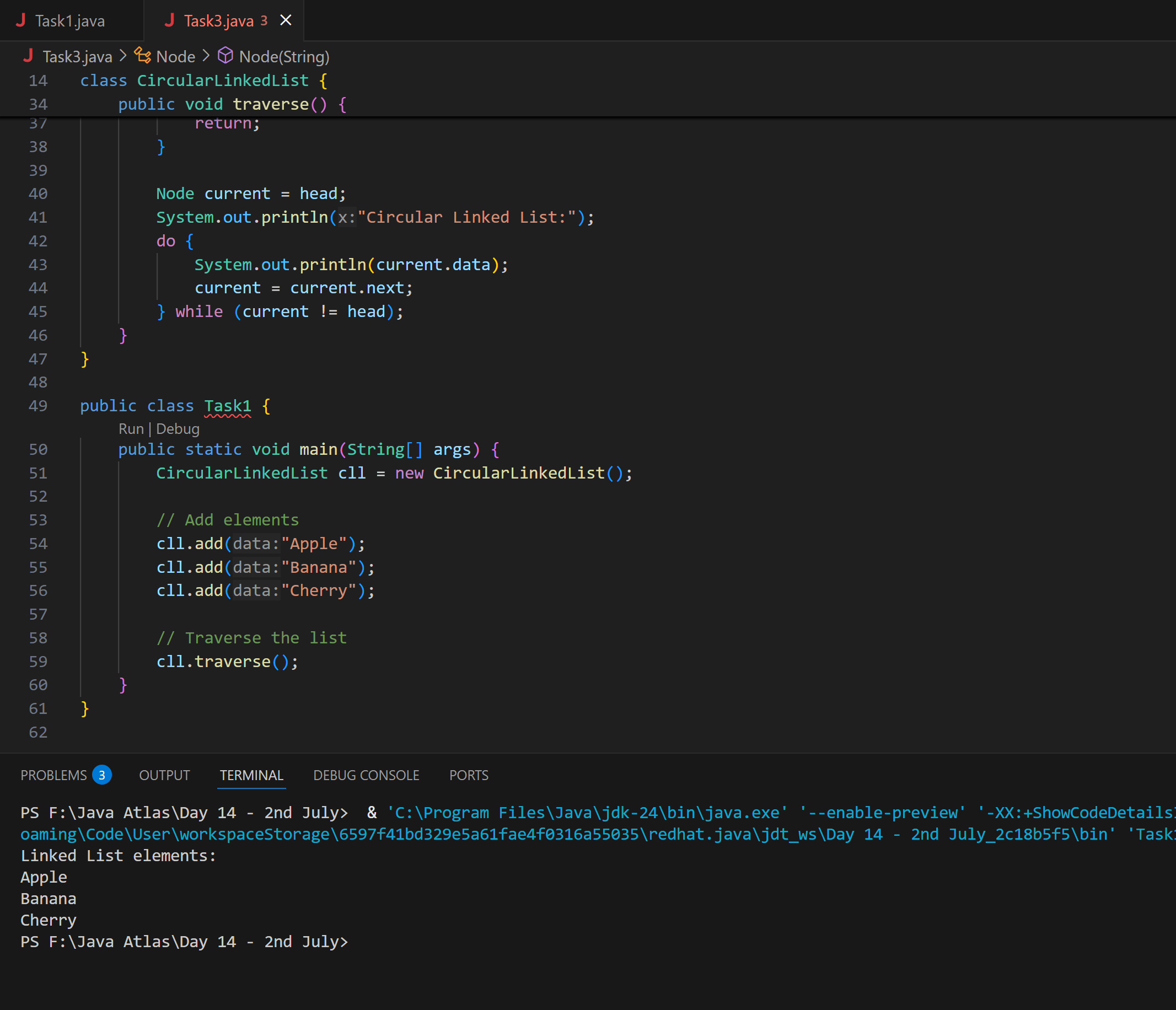
        cll.add("Cherry");

        // Traverse the list

        cll.traverse();

    }

}



**Task 4 :**

**Stacks**

**Create  a code to implement a stack**

import java.util.Stack;

public class Task4 {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Apple");

        names.push("Mango");

        names.push("Peach");

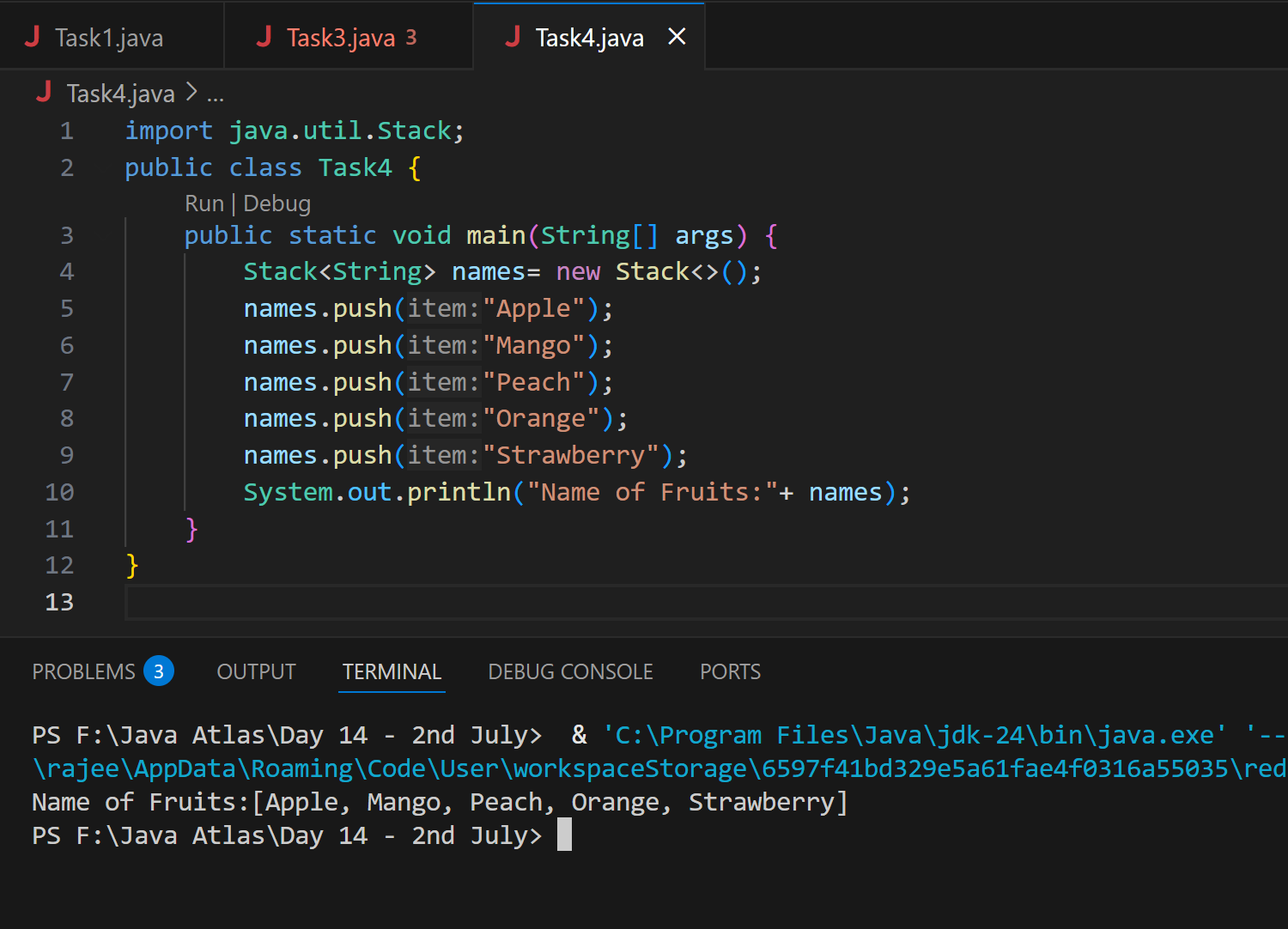
        names.push("Orange");

        names.push("Strawberry");

        System.out.println("Name of Fruits:"+ names);

    }

}

****

**Task 5:**

**Create  a stack and pop the element also print the popped element.**

import java.util.Stack;

public class Task5 {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Apple");

        names.push("Mango");

        names.push("Peach");

        names.push("Orange");

        names.push("Strawberry");

        System.out.println("Before Deletion");

        System.out.println("Name of Fruits:"+ names);

        System.out.println("After Deletion");

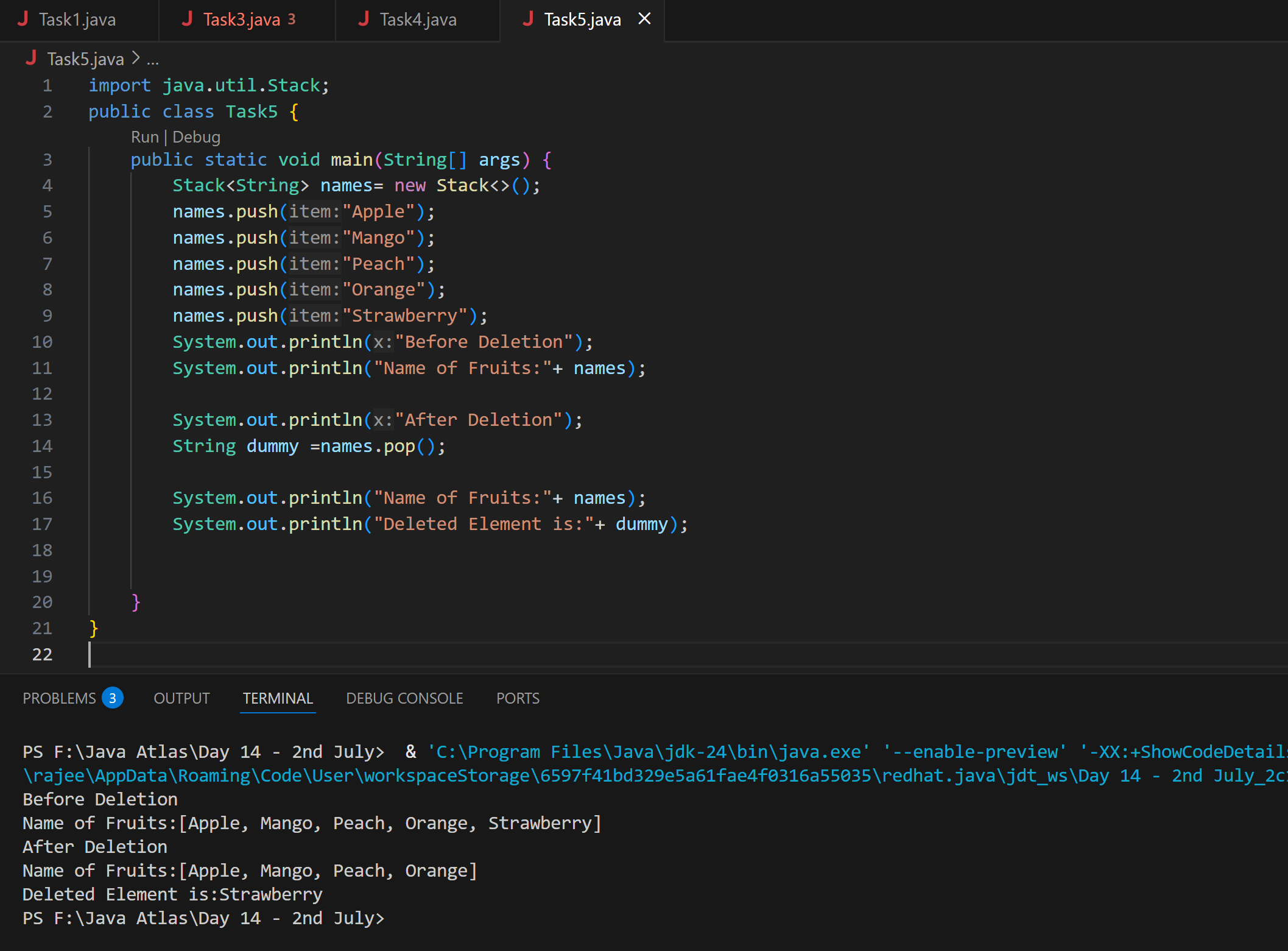
        String dummy =names.pop();

        System.out.println("Name of Fruits:"+ names);

        System.out.println("Deleted Element is:"+ dummy);

    }

}

****

**Task 6:**

**Find an element in the stack and display the position**

import java.util.Stack;

public class Task6 {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Apple");

        names.push("Mango");

        names.push("Peach");

        names.push("Orange");

        names.push("Strawberry");

        System.out.println("Name of Fruits:"+ names);

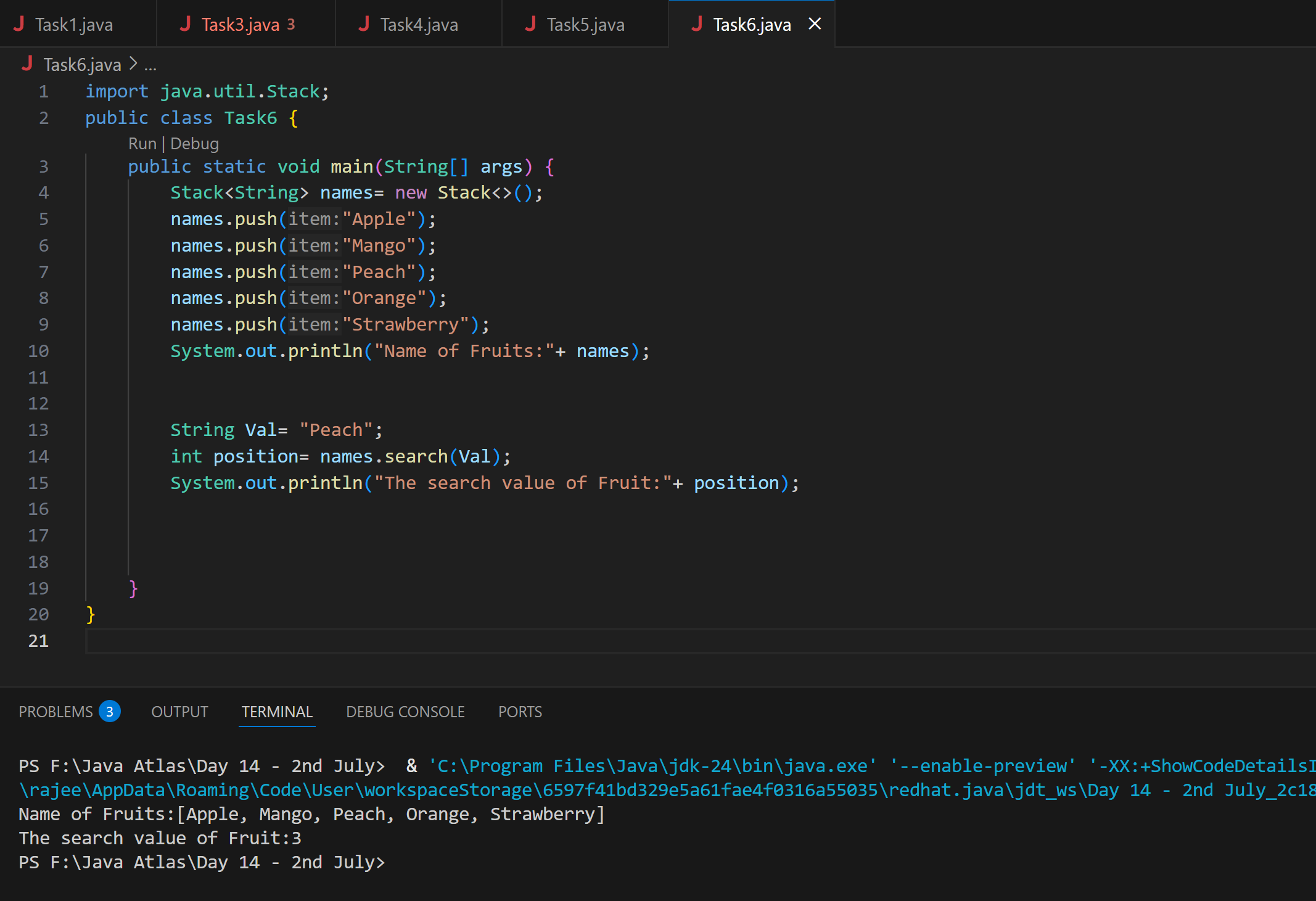
        String Val= "Peach";

        int position= names.search(Val);

        System.out.println("The search value of Fruit:"+ position);

    }

}

****

**Task 7:**

**Peek the element and print it .**

import java.util.Stack;

public class Task7 {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Apple");

        names.push("Mango");

        names.push("Peach");

        names.push("Orange");

        names.push("Strawberry");

        System.out.println("Name of Fruits:"+ names);

        String Val= "Peach";

        int position= names.search(Val);

        System.out.println("The search value of Fruit:"+ position);

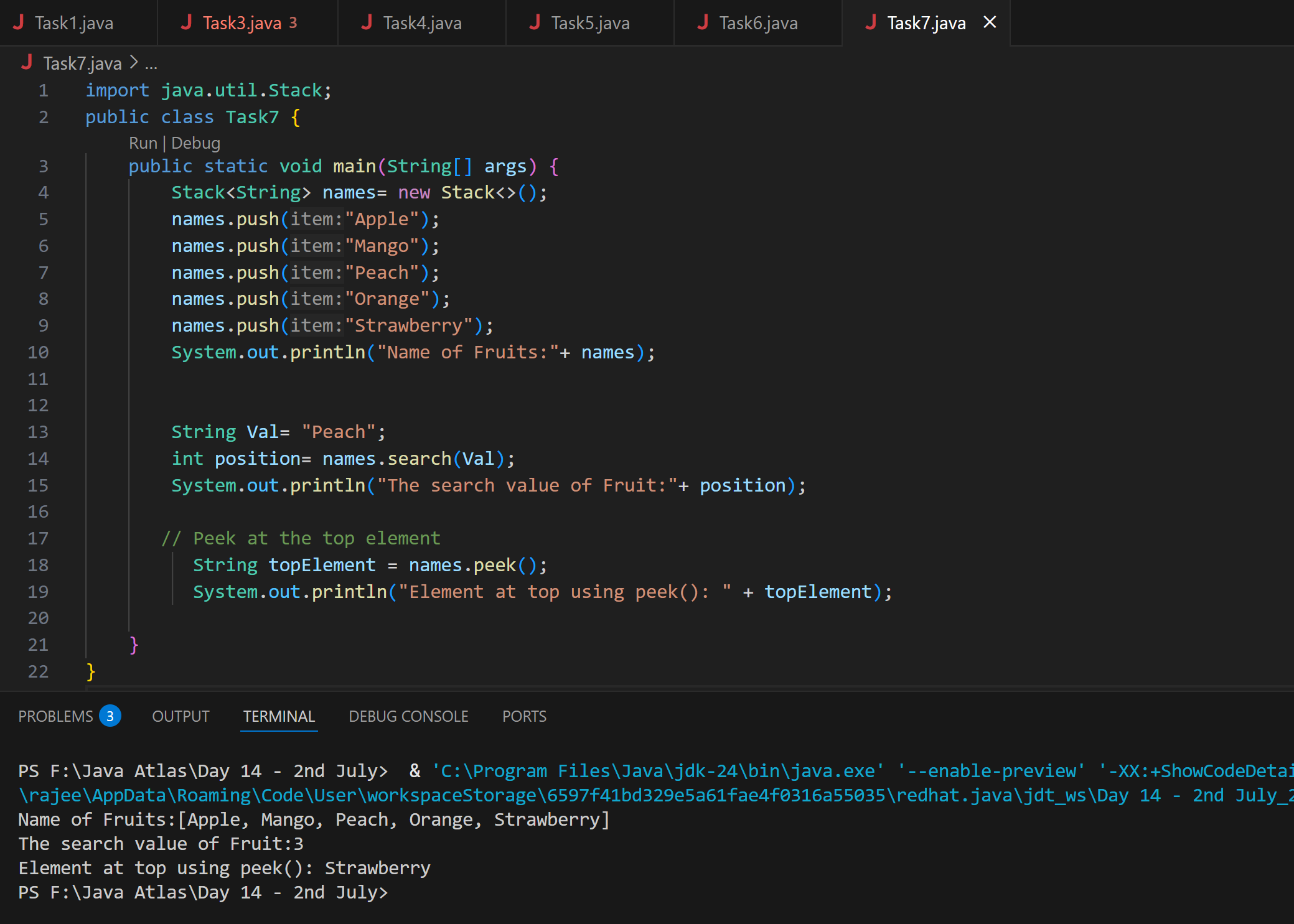
       // Peek at the top element

          String topElement = names.peek();

          System.out.println("Element at top using peek(): " + topElement);

    }

}

****

**Task 8:**

**Check if the stack is empty or not?**

import java.util.Stack;

public class Task8 {

    public static void main(String[] args) {

        Stack<String> names= new Stack<>();

        names.push("Apple");

        names.push("Mango");

        names.push("Peach");

        names.push("Orange");

        names.push("Strawberry");

        System.out.println("Name of Fruits:"+ names);

         // Check if stack is empty

            while (!names.empty()){

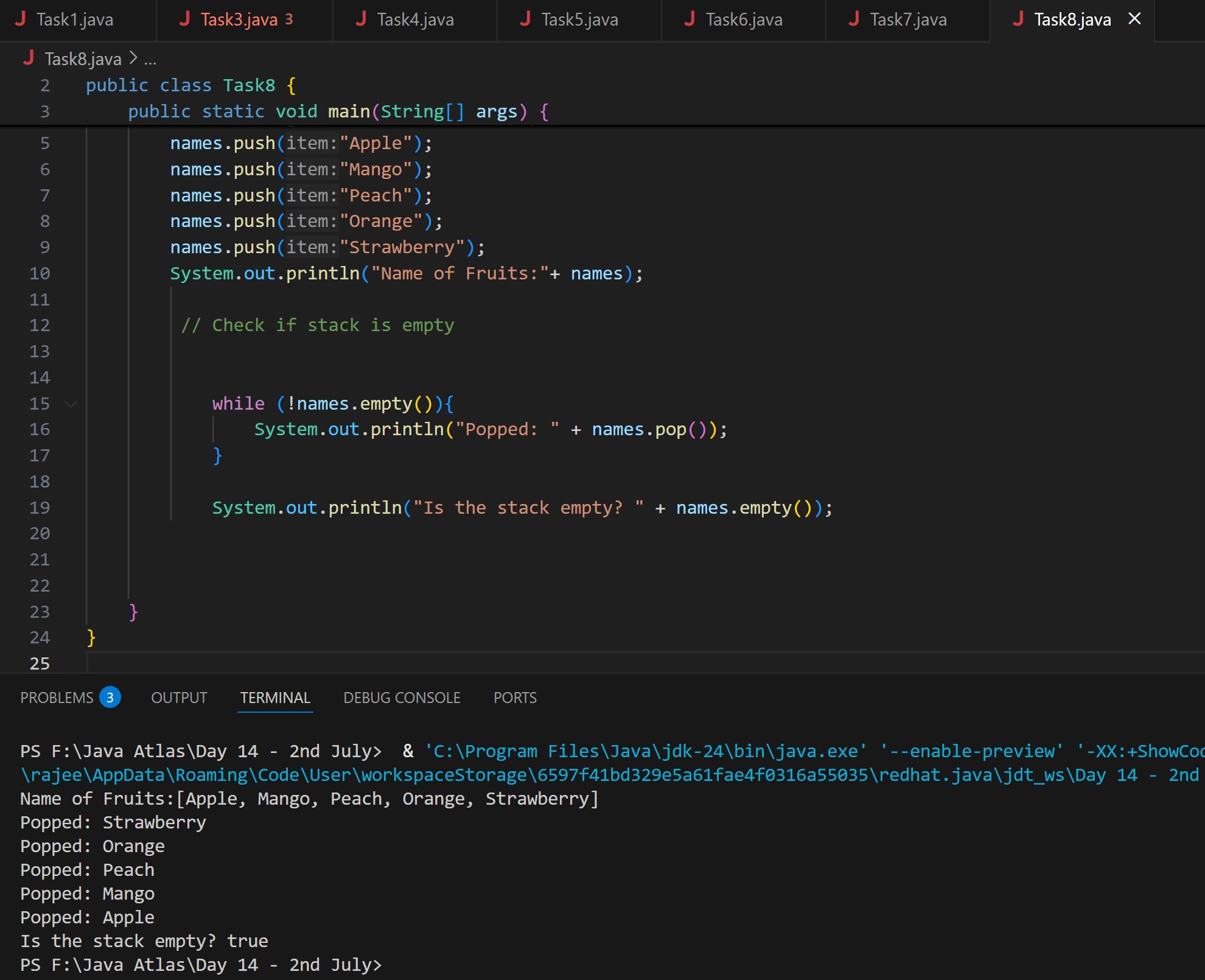
                System.out.println("Popped: " + names.pop());

            }

            System.out.println("Is the stack empty? " + names.empty());

    }

}

****

**Task 9:**

**What are the methods of the stack class**

A stack is a data structure that follows the Last In, First Out (LIFO) principle. Here are some common methods of a stack class:

**Common Methods of a Stack Class:**

1. **Push**: Adds an element to the top of the stack.
2. **Pop**: Removes and returns the top element of the stack.
3. **Peek**: Returns the top element of the stack without removing it.
4. **IsEmpty**: Checks if the stack is empty.
5. **Size**: Returns the number of elements in the stack.

**Task 10:**

**What are the common operations in Queues**

Queues are data structures that follow the First In, First Out (FIFO) principle. Here are some common operations associated with queues:

**Common Operations in Queues:**

1. **Enqueue**: Adds an element to the end of the queue.
2. **Dequeue**: Removes and returns the element at the front of the queue.
3. **Peek/Front**: Returns the element at the front of the queue without removing it.
4. **IsEmpty**: Checks if the queue is empty.
5. **Size**: Returns the number of elements in the queue.

**Task 11:**

**Wap to create  a queue with custom methods**

**Is empty ()**

**Is full()**

**Enque**

**Deque**

**Peek**

**display()**

class Node {

    int data;

    Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class CustomQueue {

    private Node front = null;

    private Node rear = null;

    private int size = 0;

    private int capacity;

    public CustomQueue(int capacity) {

        this.capacity = capacity;

    }

    public boolean isEmpty() {

        return front == null;

    }

    public boolean isFull() {

        return size == capacity;

    }

    public void enqueue(int data) {

        if (isFull()) {

            System.out.println("Queue is full. Cannot enqueue " + data);

            return;

        }

        Node newNode = new Node(data);

        if (isEmpty()) {

            front = rear = newNode;

        } else {

            rear.next = newNode;

            rear = newNode;

        }

        size++;

        System.out.println("Enqueued: " + data);

    }

    public void dequeue() {

        if (isEmpty()) {

            System.out.println("Queue is empty. Cannot dequeue.");

            return;

        }

        System.out.println("Dequeued: " + front.data);

        front = front.next;

        size--;

        if (front == null) {

            rear = null;

        }

    }

    public void peek() {

        if (isEmpty()) {

            System.out.println("Queue is empty.");

        } else {

            System.out.println("Front element: " + front.data);

        }

    }

    public void display() {

        if (isEmpty()) {

            System.out.println("Queue is empty.");

            return;

        }

        Node current = front;

        System.out.print("Queue elements: ");

        while (current != null) {

            System.out.print(current.data + " ");

            current = current.next;

        }

        System.out.println();

    }

}

public class Task11 {

    public static void main(String[] args) {

        CustomQueue q = new CustomQueue(5); // Max 5 elements

        q.enqueue(10);

        q.enqueue(20);

        q.enqueue(30);

        q.enqueue(40);

        q.enqueue(50);

        q.display();

        q.dequeue();

        q.dequeue();

        q.display();

        q.peek();

        q.enqueue(60); // Should succeed

        q.enqueue(70); // Should fail (capacity = 5)

        q.display();

        System.out.println("Is Empty? " + q.isEmpty());

        System.out.println("Is Full? " + q.isFull());

    }

}

