Day 14

Employee ID: 201933938

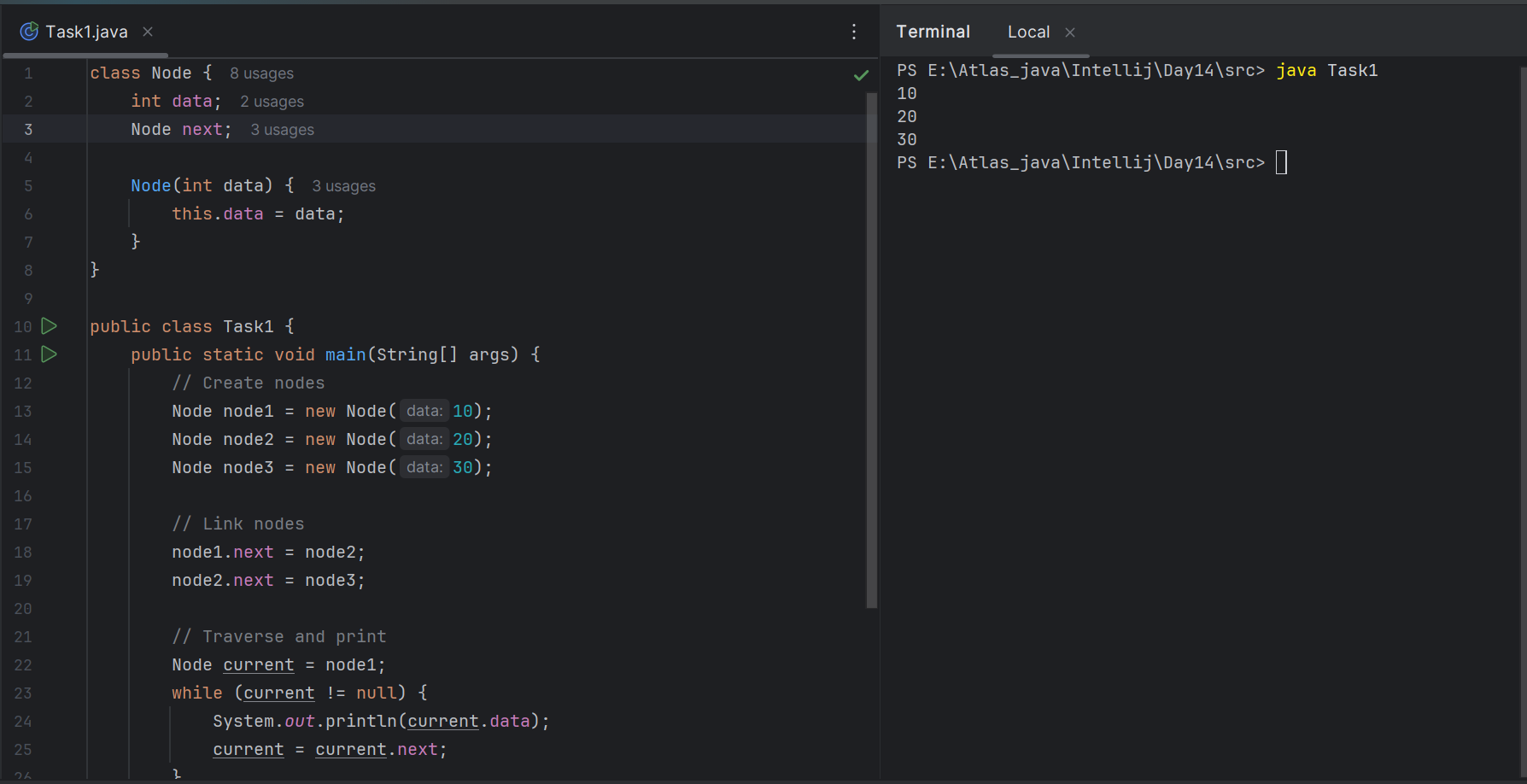
Login ID: iamasif

Name: Shaik Asif

Task 1:

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

09.45 to 9.50



Task 2 👍

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

Traversing a linked list means going through all the elements, one by one, starting from the beginning and following the links (pointers) to the next node, until there are no more nodes left.

Expl:

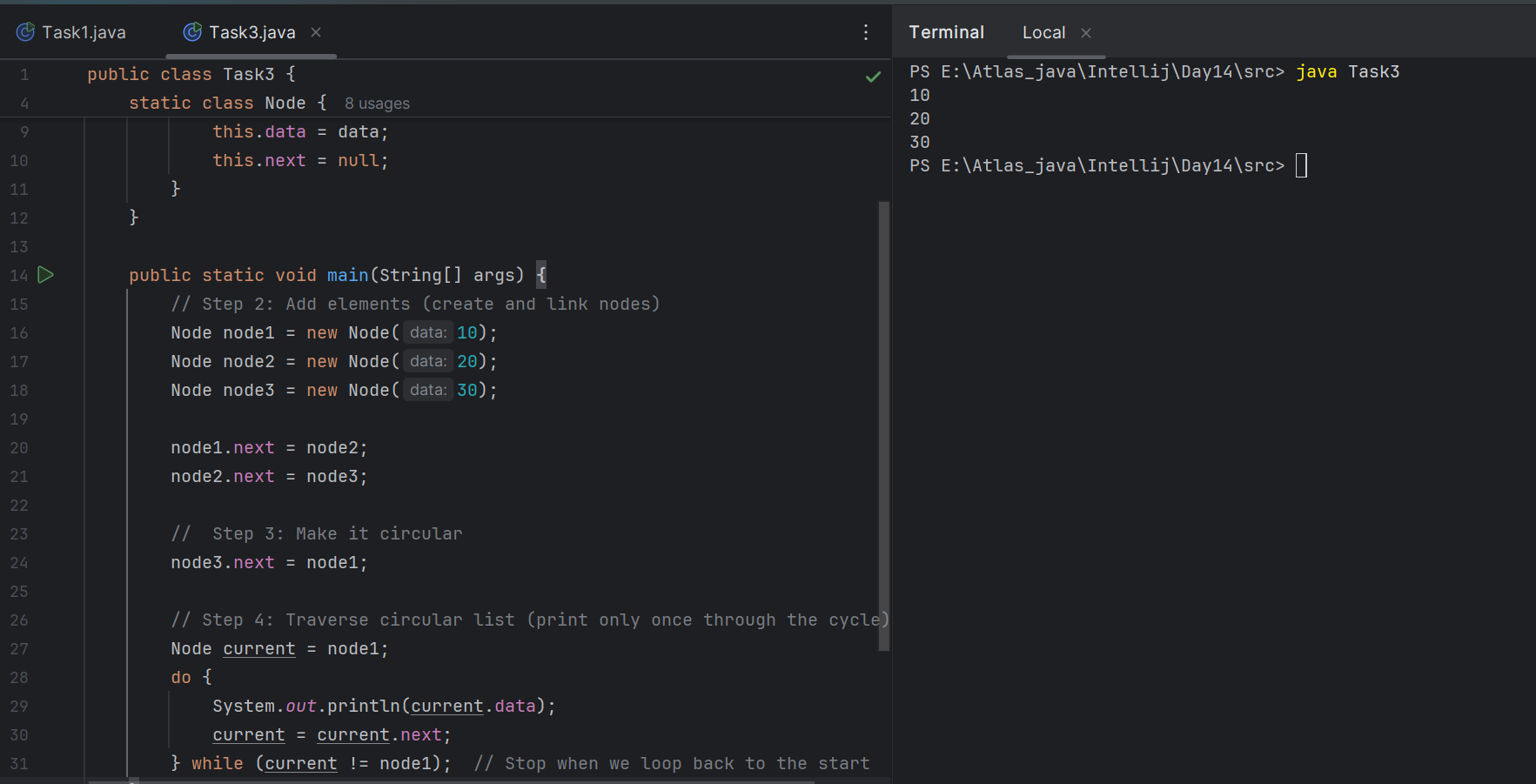
If the list is:

10 → 20 → 30 → null

9.51 to 9.53

Task 3:

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



10.05 to 10.12

10.48

class Node {

int value;

Node nextNode;

public Node(int value) {

this.value = value;

}

}

public class CircularLinkedList {

// if the list is empty

private Node head = null;

private Node tail = null;

// …..

}

public void addNode(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

} else {

tail.nextNode = newNode;

}

tail = newNode;

tail.nextNode = head;

}

public void addNode(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode;

} else {

tail.nextNode = newNode;

}

tail = newNode;

tail.nextNode = head;

}

private CircularLinkedList createCircularLinkedList() {

CircularLinkedList cll = new CircularLinkedList();

cll.addNode(13);

cll.addNode(7);

cll.addNode(24);

cll.addNode(1);

cll.addNode(8);

cll.addNode(37);

cll.addNode(46);

return cll;

}

public boolean containsNode(int searchValue) {

Node currentNode = head;

if (head == null) {

return false;

} else {

do {

if (currentNode.value == searchValue) {

return true;

}

currentNode = currentNode.nextNode;

} while (currentNode != head);

return false;

}

}

public void deleteNode(int valueToDelete) {

Node currentNode = head;

if (head == null) { // the list is empty

return;

}

do {

Node nextNode = currentNode.nextNode;

if (nextNode.value == valueToDelete) {

if (tail == head) { // the list has only one single element

head = null;

tail = null;

} else {

currentNode.nextNode = nextNode.nextNode;

if (head == nextNode) { //we're deleting the head

head = head.nextNode;

}

if (tail == nextNode) { //we're deleting the tail

tail = currentNode;

}

}

break;

}

currentNode = nextNode;

} while (currentNode != head);

}

Sacks

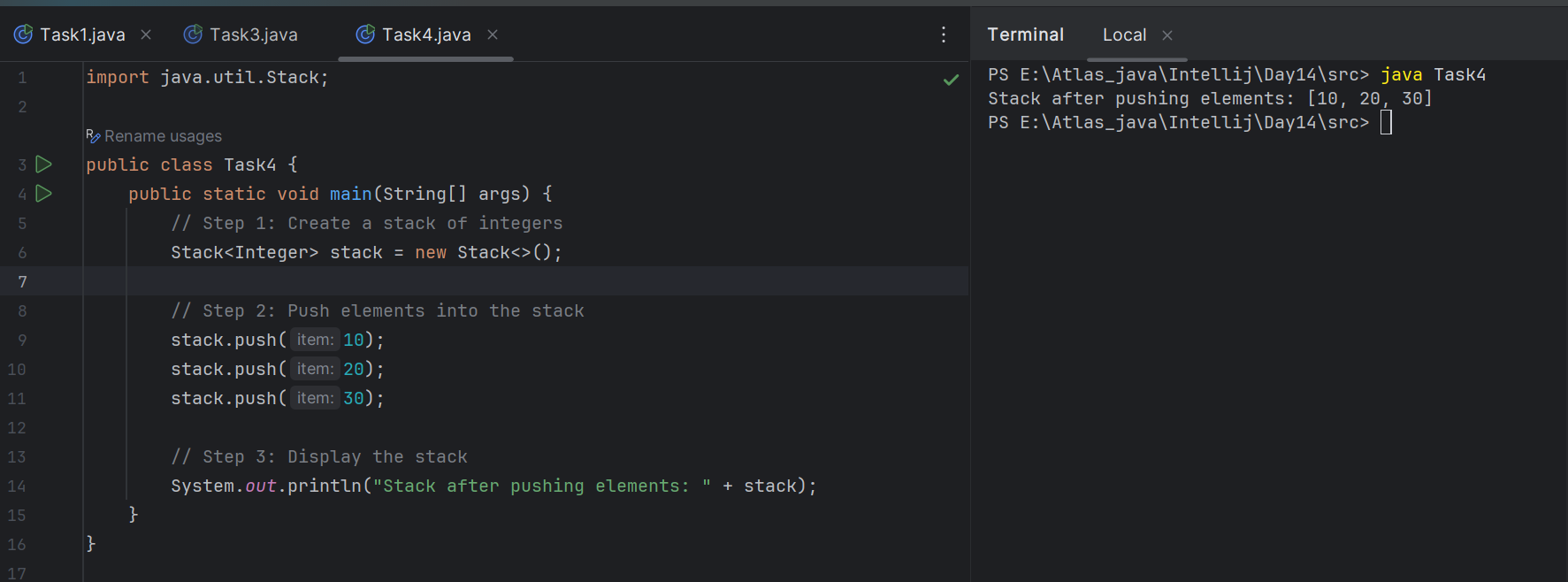
Collection (interface ) ===> List (interface) ⇒ Vector class ===> Stacks (class)

Task 4 :

Stacks

Create a code to implement a stack

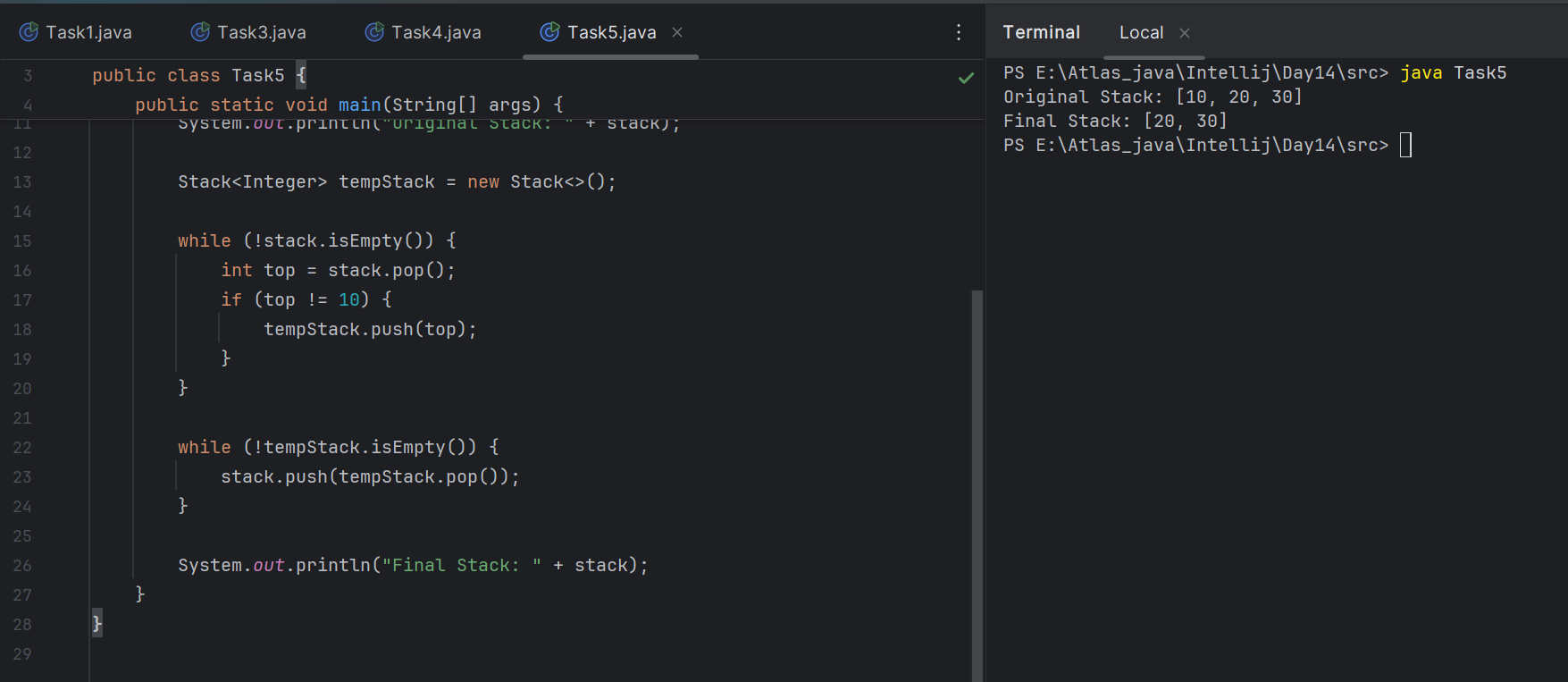
12.42 – plz complete..



Task 5:

Create astack and pop the element also print the popped element.

12.50 plz complete

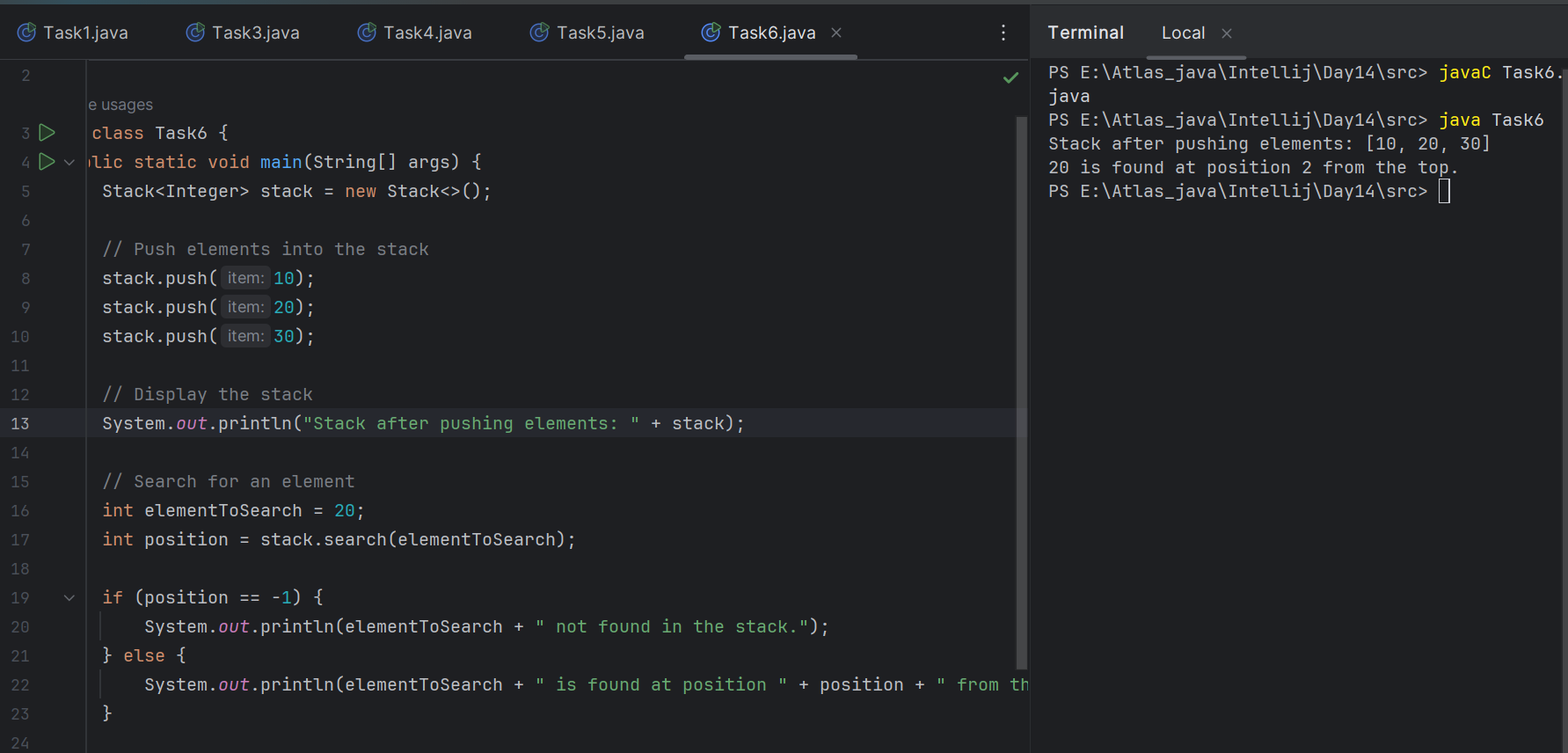


Task 6:

Find an element in the stack and display the position

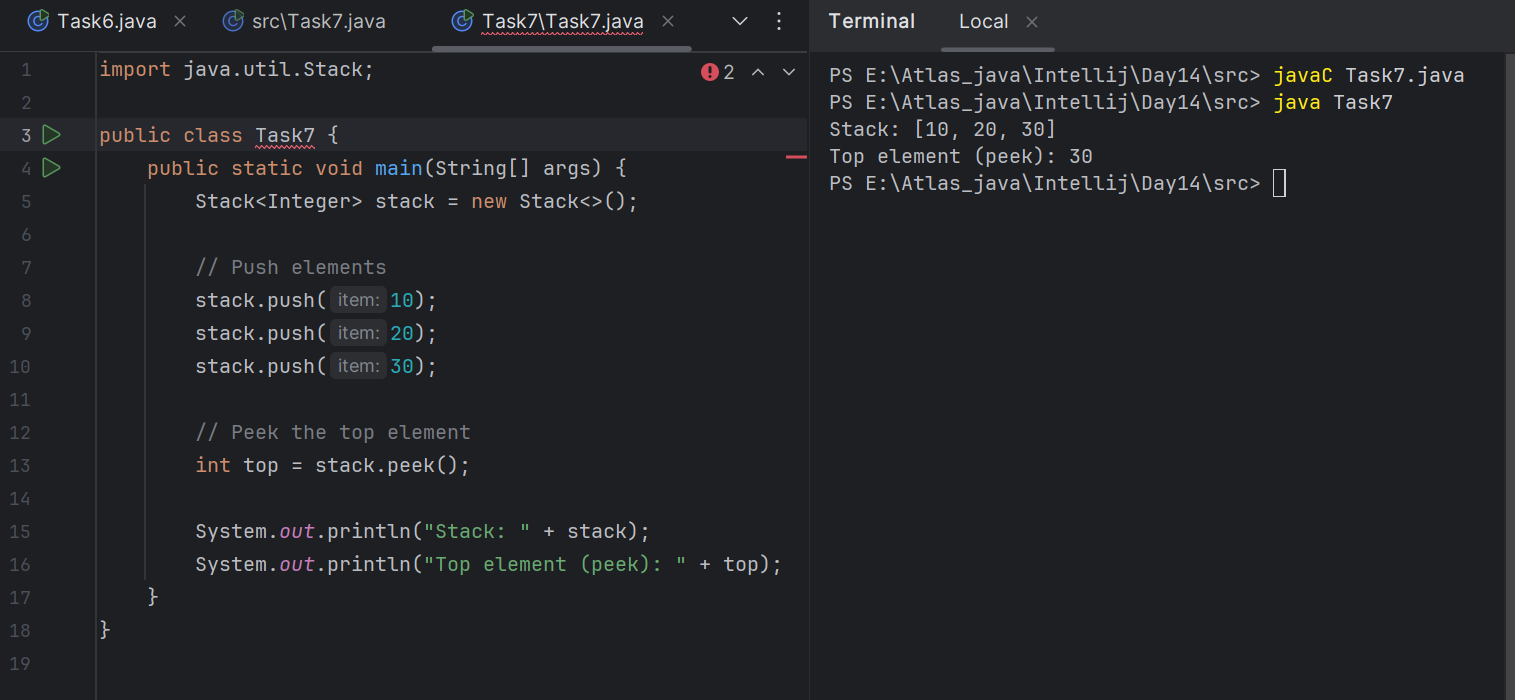
Hint 👍

Int position = names.search(“value”);



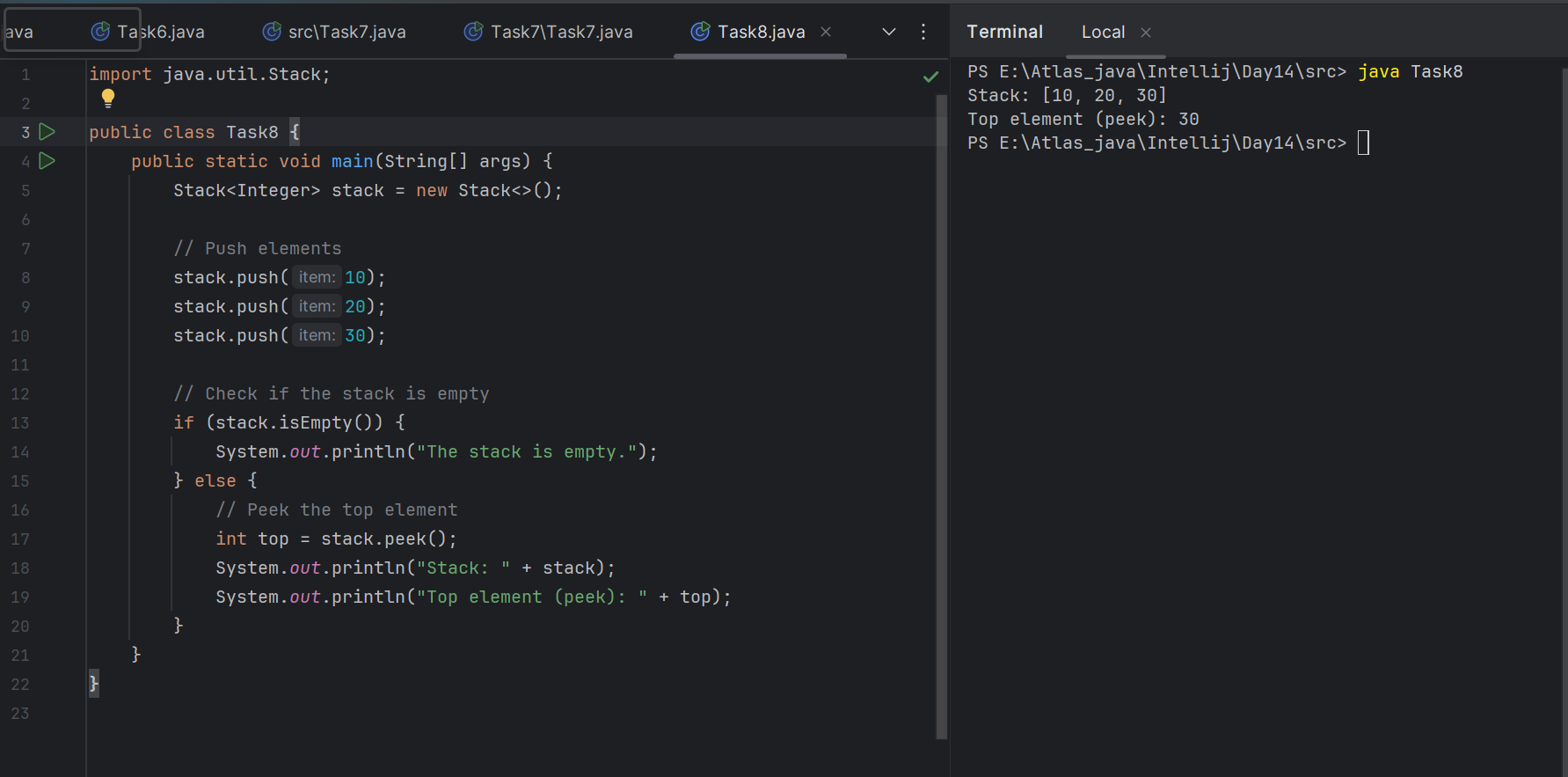
Task 7:

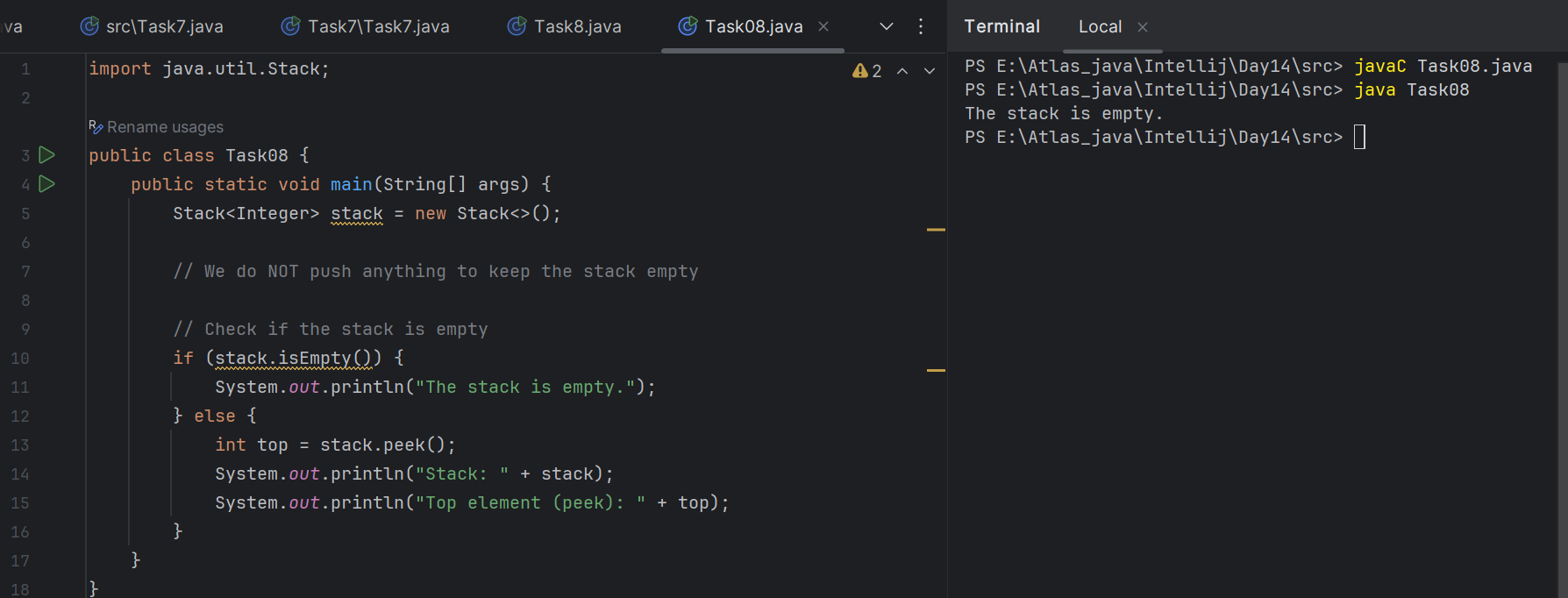
Peek the element and print it ..



Task 8:

Check if the stack is empty or not?





Task 9:

What are the methods of the stack class

| **Method** | **Description** |
| --- | --- |
| push(E item) | Adds (pushes) an item to the top of the stack |
| pop() | Removes and returns the top item from the stack |
| peek() | Returns the top item without removing it |
| empty() | Returns true if the stack is empty |
| search(Object o) | Returns 1-based position of element from the top, or -1 if not found |
| size() | Returns the number of elements in the stack |
| isEmpty() | Same as empty(), returns true if empty |
| clear() | Removes all elements from the stack (inherited from Vector) |
| contains(Object o) | Checks if a particular element exists in the stack |
| get(int index) | Returns the element at the specified index (inherited from Vector) |
| set(int index, E element) | Replaces the element at the specified index |
| iterator() | Returns an iterator to traverse the stack |

Task 10:

What are the common operations in Queues

| **Operation** | **Description** |
| --- | --- |
| **enqueue (add)** | Adds an element to the **rear** of the queue |
| **dequeue (remove)** | Removes and returns the element from thefront of the queue |
| **peek / front / element** | Returns the front element without removing it |
| **isEmpty()** | Checks if the queue has no elements (true if empty) |
| **size()** | Returns the number of elements in the queue |
| **contains(x)** | Checks if a specific element is present in the queue |
| **clear()** | Removes all elements from the queue |

.

Task 11:

Wap to create a queue with custom methods

Is empty ()

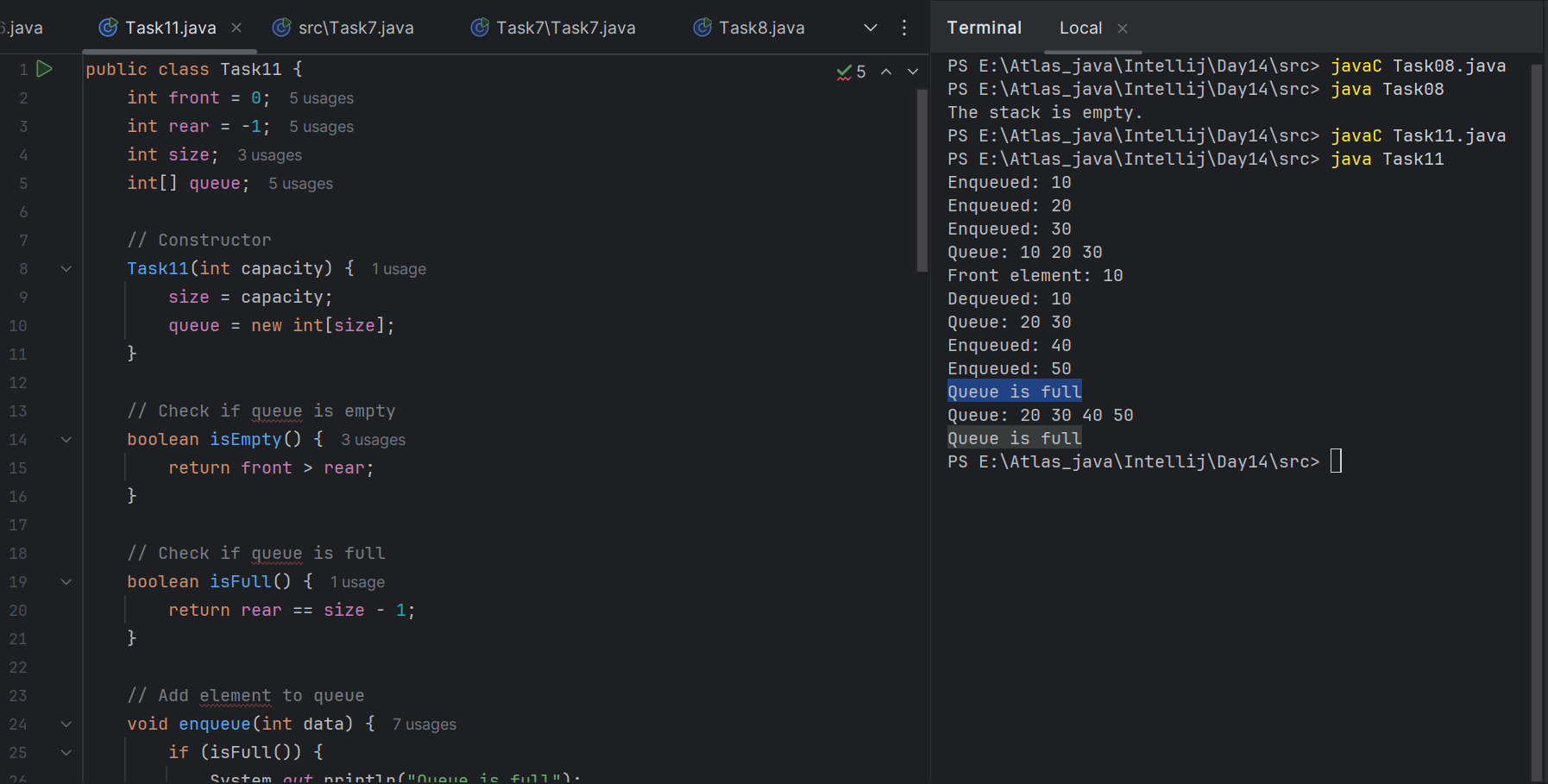
Is full()

Enque

Deque

Peek

display()



public class Queue {

int queueLength = 3;

int items[] = new int[queueLength];

int front = -1;

int back = -1;

void enQueue(int itemValue) {

if(isFull()){

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

} else if(front == -1 && back == -1){

front = back = 0;

items[back] = itemValue;

} else{

back++;

items[back] = itemValue;

}

}

void deQueue(){

if(isEmpty()){

System.out.println("Queue is empty. Nothing to dequeue");

} else if (front == back){

front = back = -1;

} else {

front++;

}

}

void display(){

int i;

if(isEmpty()){

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

} else {

for(i = front; i <= back; i++){

System.out.println(items[i]);

}

}

}

boolean isFull(){

if(back == queueLength - 1){

return true;

} else {

return false;

}

}

boolean isEmpty(){

if(front == -1 && back == -1){

return true;

} else {

return false;

}

}

void peek(){

System.out.println("Front value is: " + items[front]);

}

public static void main(String[] args) {

Queue myQueue = new Queue();

myQueue.enQueue(111);

myQueue.enQueue(222);

myQueue.enQueue(777);

myQueue.display();

myQueue.peek();

}

}

**Add ons:**

//converting stack and deque into a lists and printing their elements in java using streams.

import java.util.\*;

import java.util.stream.Collectors;

class Stack\_Deque\_to\_List.java {

public static void main (String[] args) {

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

Deque<Integer> deque = new ArrayDeque<>();

stack.push(1);

deque.push(1);

stack.push(2);

deque.push(2);

List<Integer> list1 = stack.stream().collect(Collectors.toList());

System.out.println("Using Stack: ");

for(int i = 0; i < list1.size(); i++){

System.out.print(list1.get(i) + " " );

}

System.out.println();

List<Integer> list2 = deque.stream().collect(Collectors.toList());

System.out.println("Using Deque: ");

for(int i = 0; i < list2.size(); i++){

System.out.print(list2.get(i) + " " );

}

System.out.println();

}

}

15.40 to 15.52

================================================================================================================================================

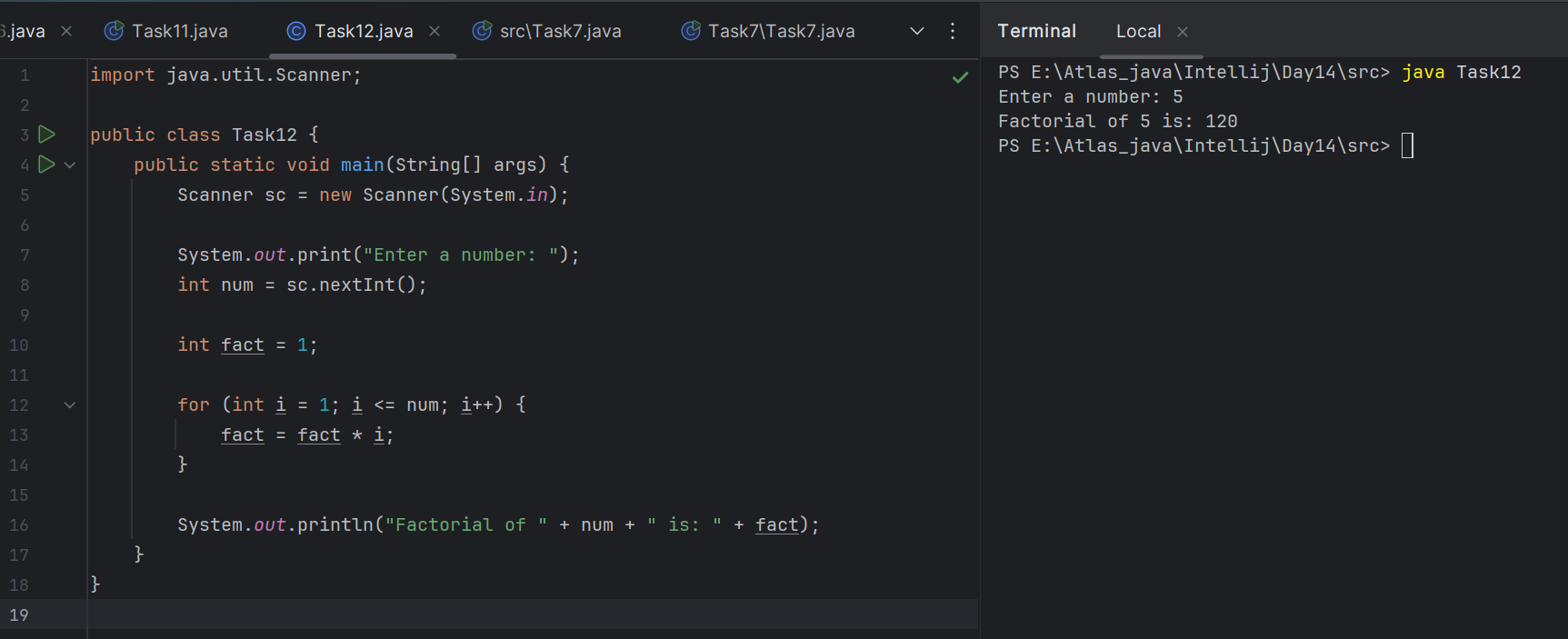
Home Tasks:

Recursion:

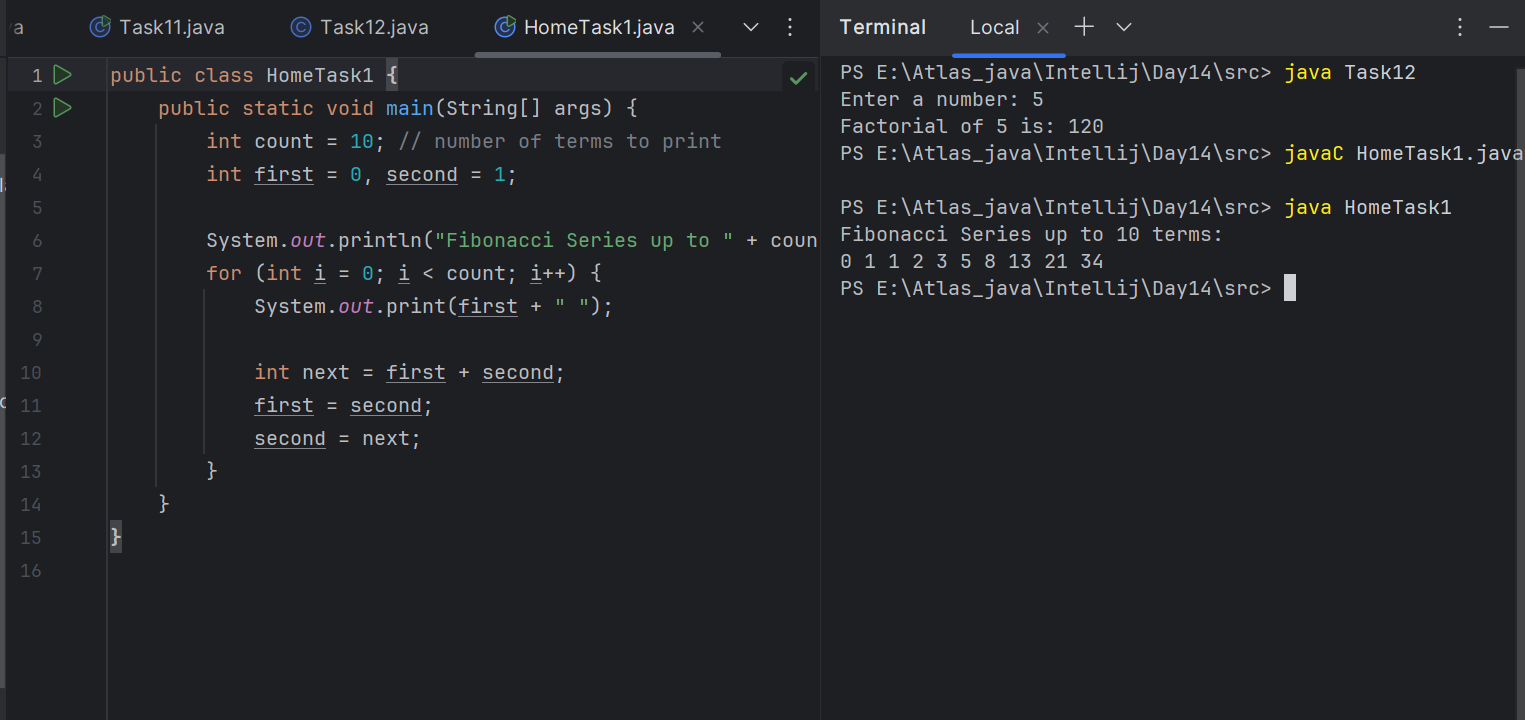
**Recursion** is when a method **calls itself** to solve a problem.

Each call works on a **smaller part** of the problem, and eventually reaches a **base condition** (which stops further calls).

**Wap to find the factorial of a number**

****

**Wap to find the Fibonacci series of a number**

****

What is the difference between recursion and iteration

### **Iteration:**

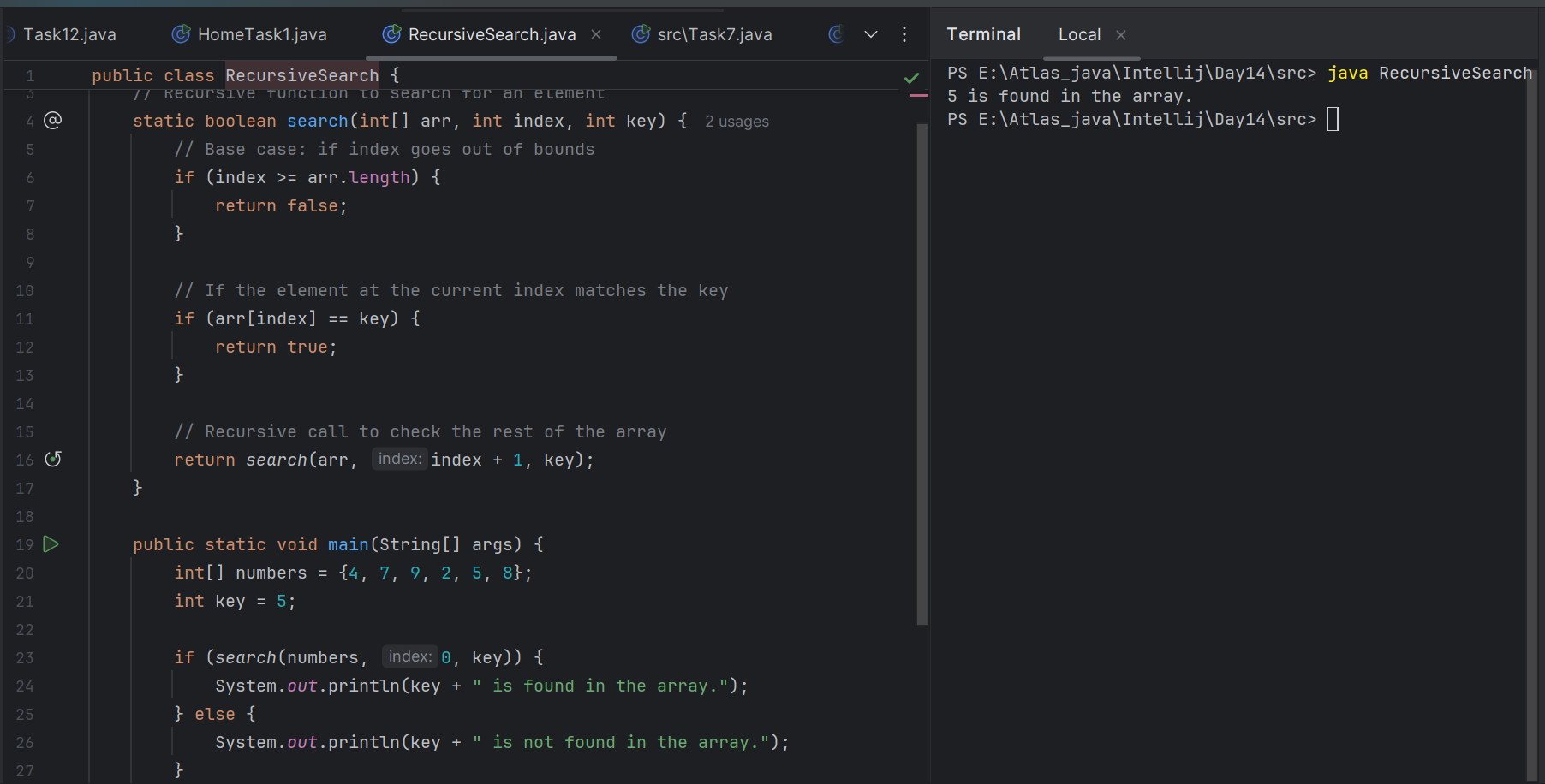
* **Definition**: Repeating a set of instructions using loops (for, while, etc.).
* **Approach**: Uses a loop to repeat code.
* **Memory usage**: More efficient; uses less memory.
* **Performance**: Usually faster than recursion.

### **Recursion:**

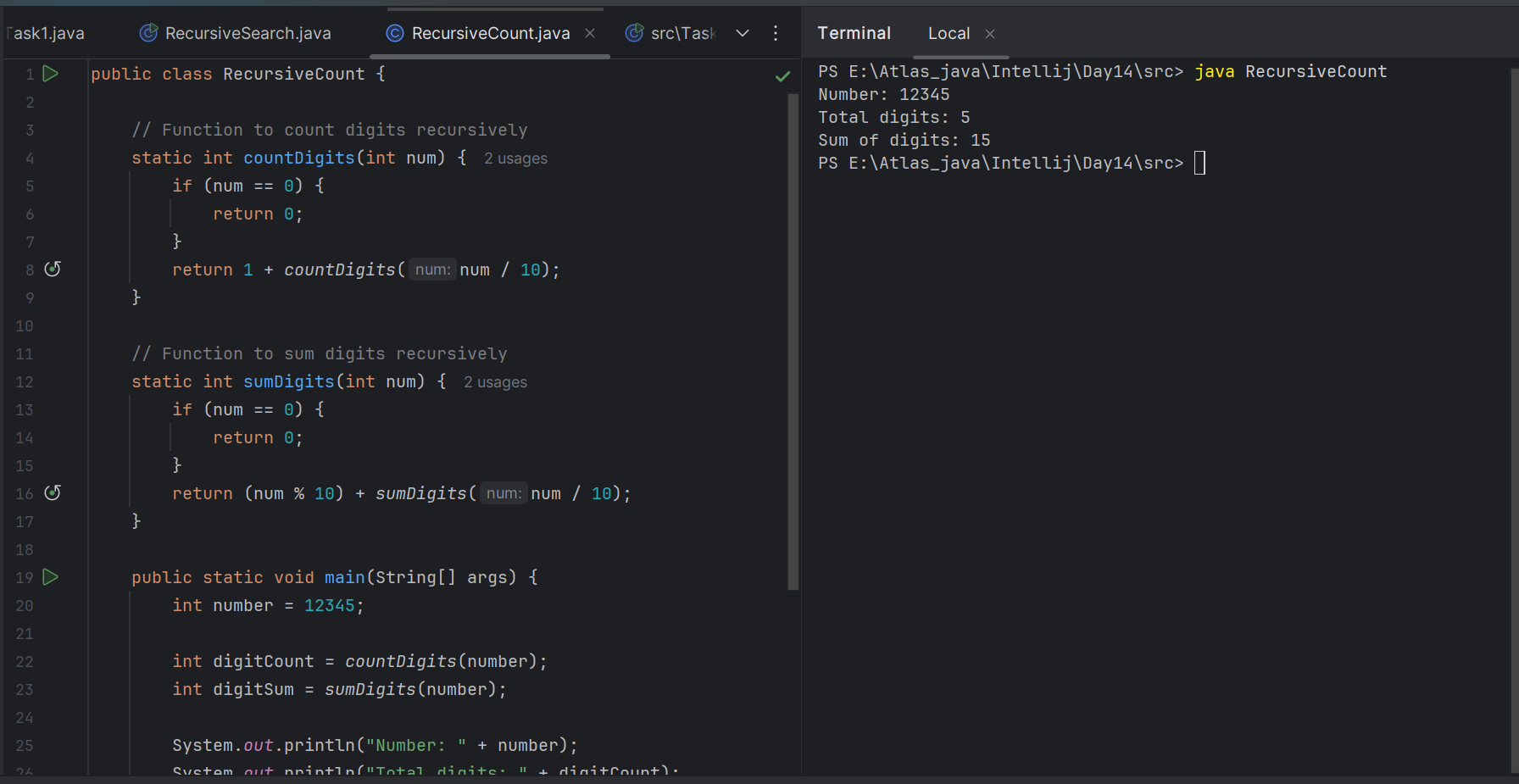
* **Definition**: A function calling itself to solve a smaller part of the problem.
* **Approach**: Uses method calls repeatedly.
* **Memory usage**: Higher; stores function calls in the call stack.
* **Performance**: Can be slower and may cause stack overflow for large inputs.

Carry forward examples till 10 th july

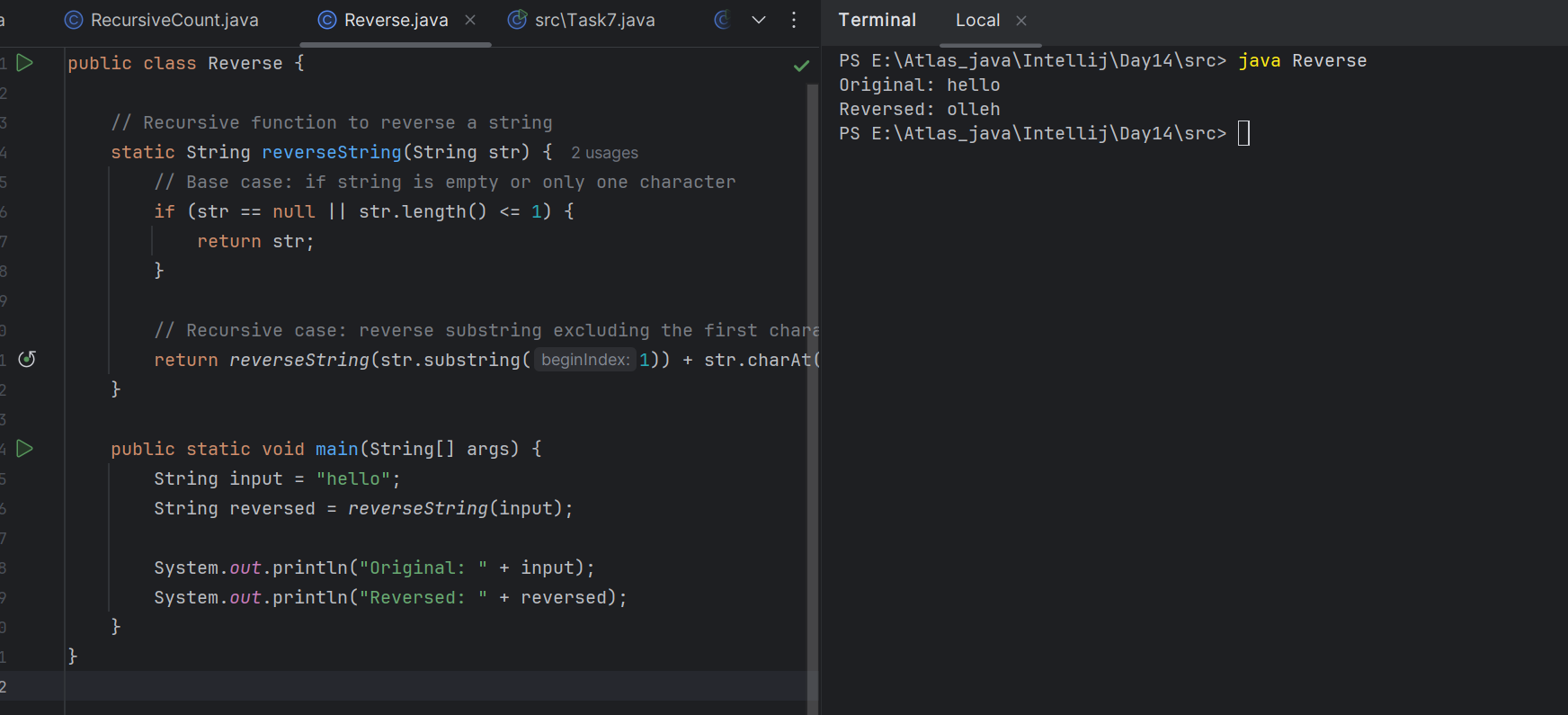
1. Write a recursive function to search for an element in an array



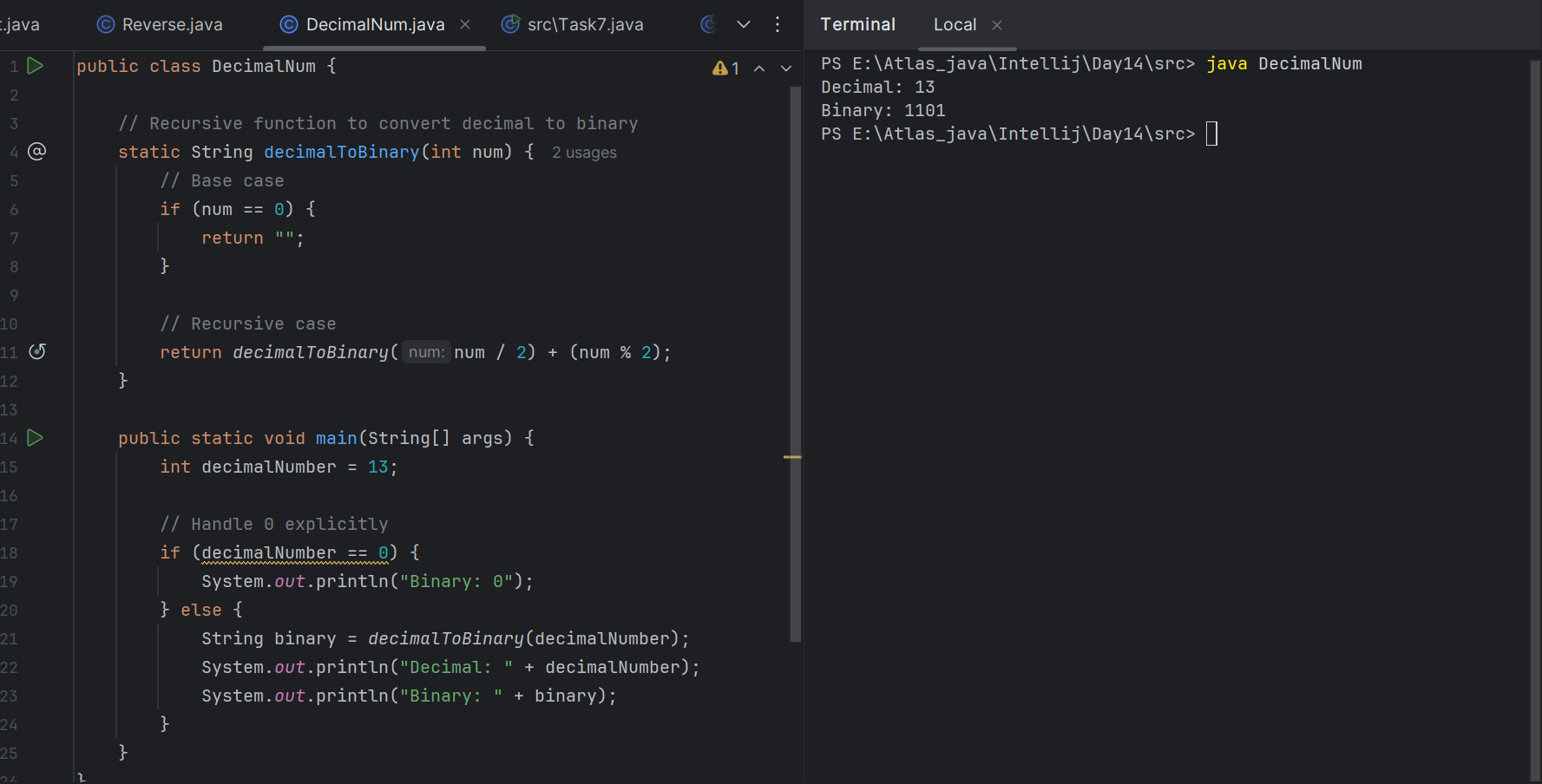
2. Write a recursive function to count the digits of a positive integer (do also for sum of digits)



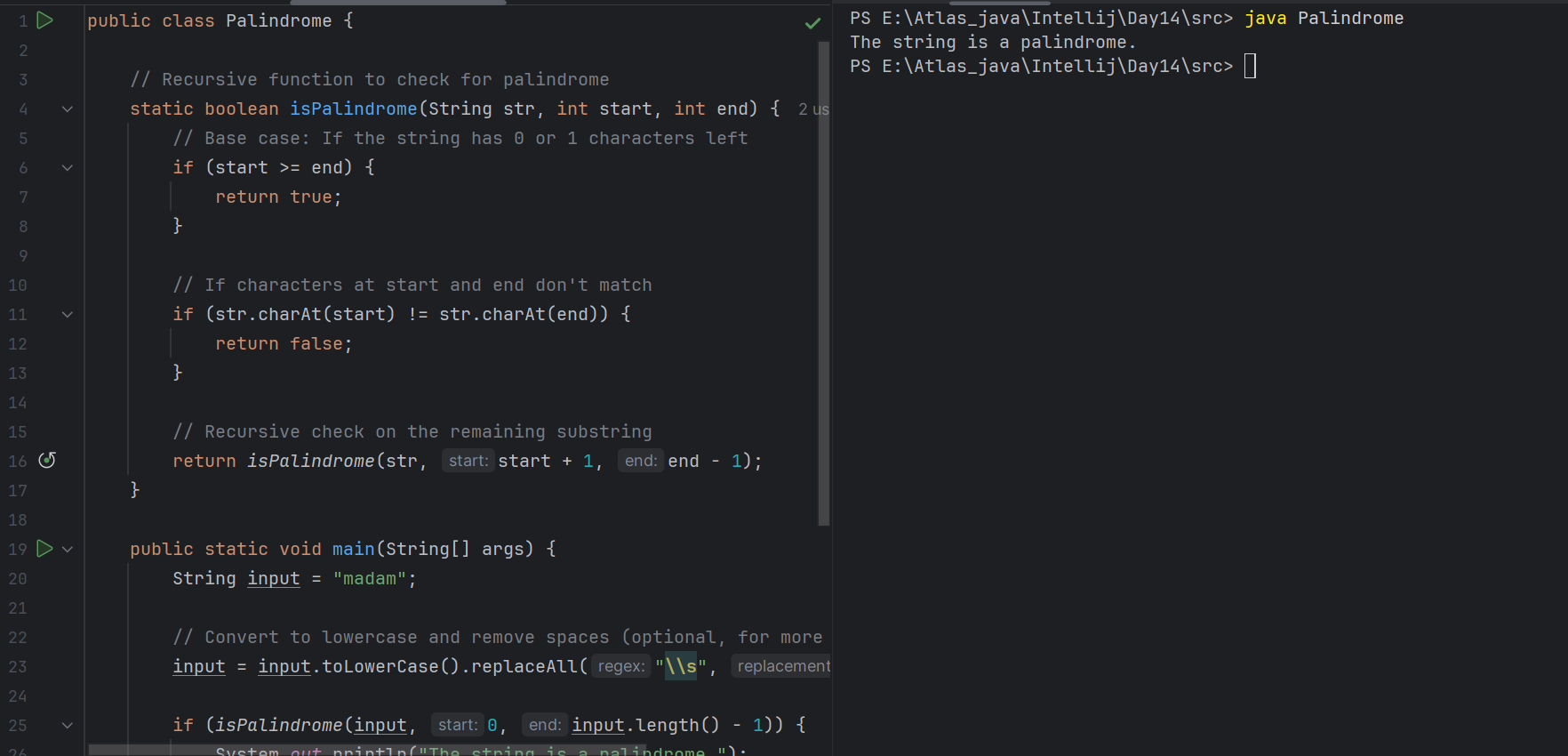
3. Write a recursive function to reverse a null-terminated string



4. Write a recursive function to convert a decimal number to binary



5. Write a recursive function to check if a string is a palindrome or not



6. Write a recursive function to copy one array to another

