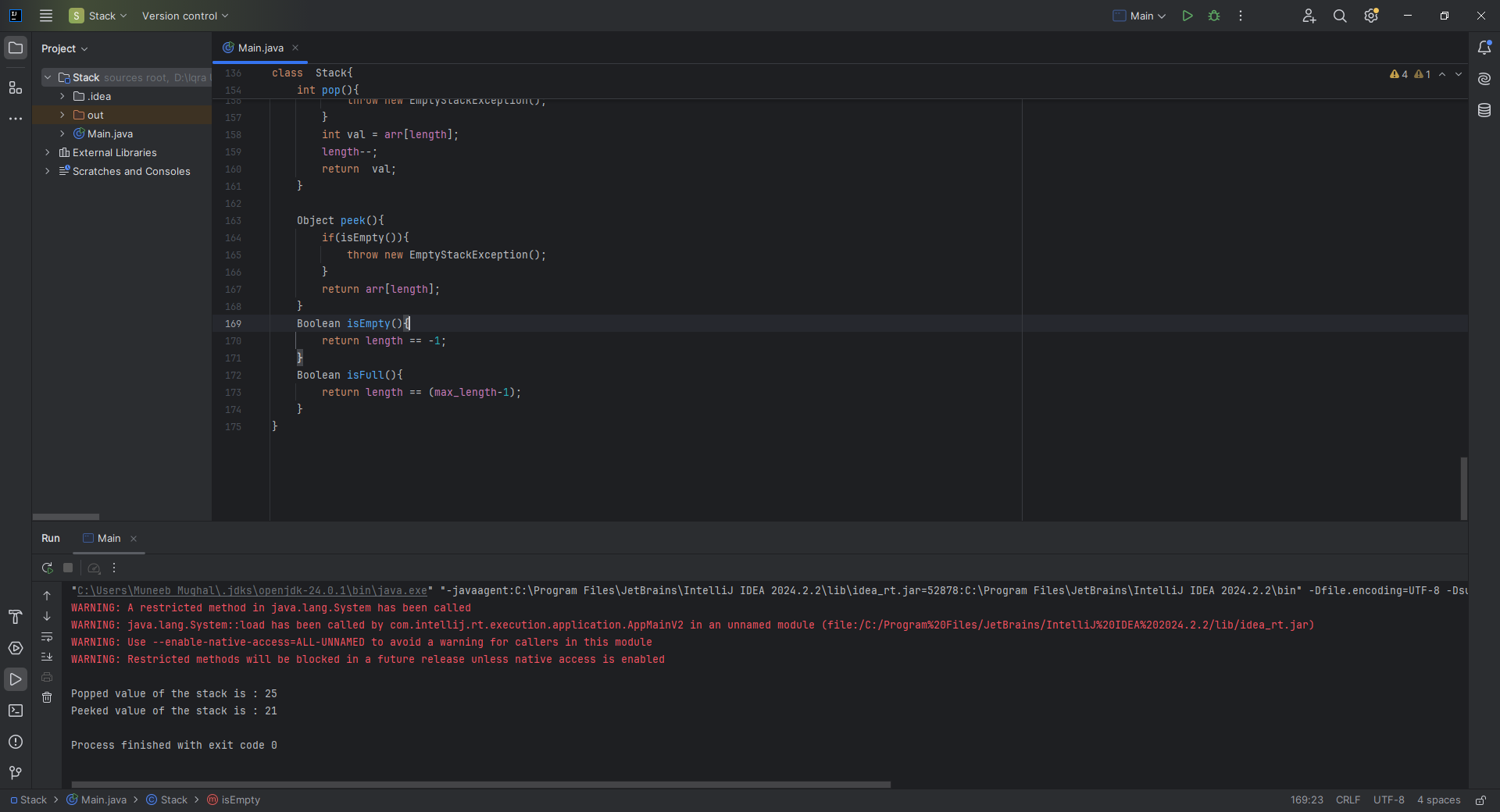
**CLASS TASK 1 & 2:**

**CODE:**

import java.util.EmptyStackException;  
  
class Main{

public static void main(String[] args){  
 Stack stack = new Stack(5);  
  
 stack.push(20);  
 stack.push(21);  
 stack.push(25);  
  
 try{  
 System.*out*.println("Popped value of the stack is : "+stack.pop());  
 }catch (Exception e){  
 System.*out*.println("Stack is "+ e.getMessage());  
 }  
  
 try{  
 System.*out*.println("Peeked value of the stack is : "+stack.peek());  
 }catch (Exception e){  
 System.*out*.println("Stack is "+ e.getMessage());  
 }  
}

}  
  
class Stack{  
 private int length = -1;  
 private final int max\_length;  
 private final int[] arr;  
  
 Stack(int max\_length){  
 this.max\_length = max\_length;  
 arr = new int[max\_length];  
 }  
 void push(int val){  
 if(isFull()){  
 System.*out*.println("Stack is Full.");  
 return;  
 }  
 length++;  
 arr[length] = val;  
 }  
  
 int pop(){  
 if(isEmpty()){  
 throw new EmptyStackException();  
 }  
 int val = arr[length];  
 length--;  
 return val;  
 }  
  
 Object peek(){  
 if(isEmpty()){  
 throw new EmptyStackException();  
 }  
 return arr[length];  
 }  
 Boolean isEmpty(){  
 return length == -1;  
 }  
 Boolean isFull(){  
 return length == (max\_length-1);  
 }  
}

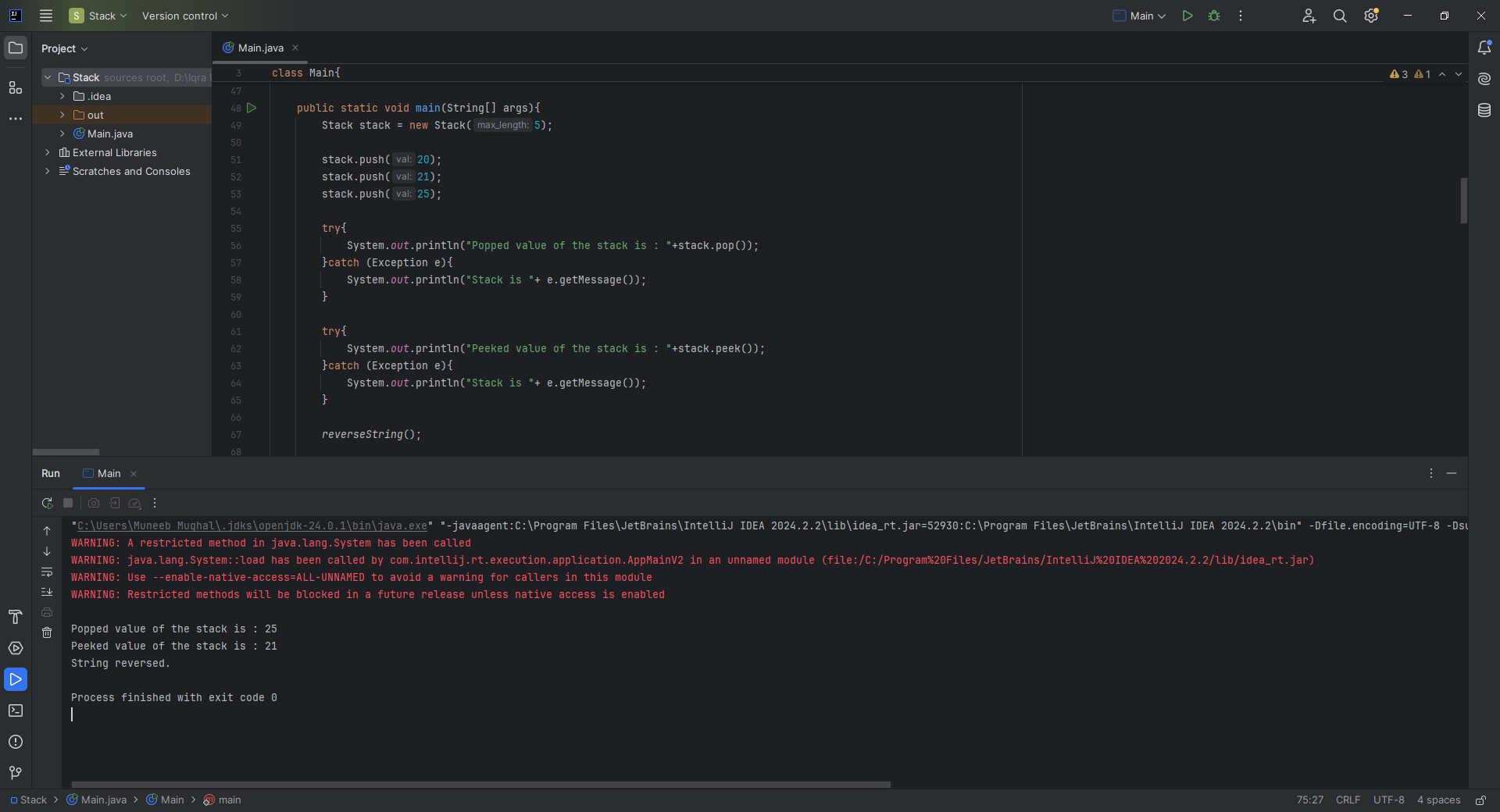


**CLASS TASK 3:** (Reversing a String)

**CODE:**

static void reverseString(){  
 Stack stack = new Stack(5);  
 Stack stack2 = new Stack(5);

String str = "Hello";  
 for (int i = 0; i < str.length(); i++)  
 {  
 stack.push(str.charAt(i));  
 }  
 for (int i = 0; i < str.length(); i++){  
 stack2.push(stack.pop());  
 }  
 System.*out*.println("String reversed.");  
  
}



**CLASS TASK 4:**

**Structure:**

* Base case: Stops recursion (if (n == 0)).
* Recursive case: Function calls itself with a smaller value (n - 1).

**CODE:**

int factorial(int n) {

if (n == 0) return 1; // Base case

return n \* factorial(n - 1); // Recursive call

}

**CLASS TASK 5:**

**STRUCTURE:**

* Starts with: 0, 1, 1, 2, 3, 5, 8, ...
* Each term: fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)
* Base cases:
  + fibonacci(0) = 0
  + fibonacci(1) = 1

**CODE:**

int fibonacci(int n) {

if (n == 0) return 0;

if (n == 1) return 1;

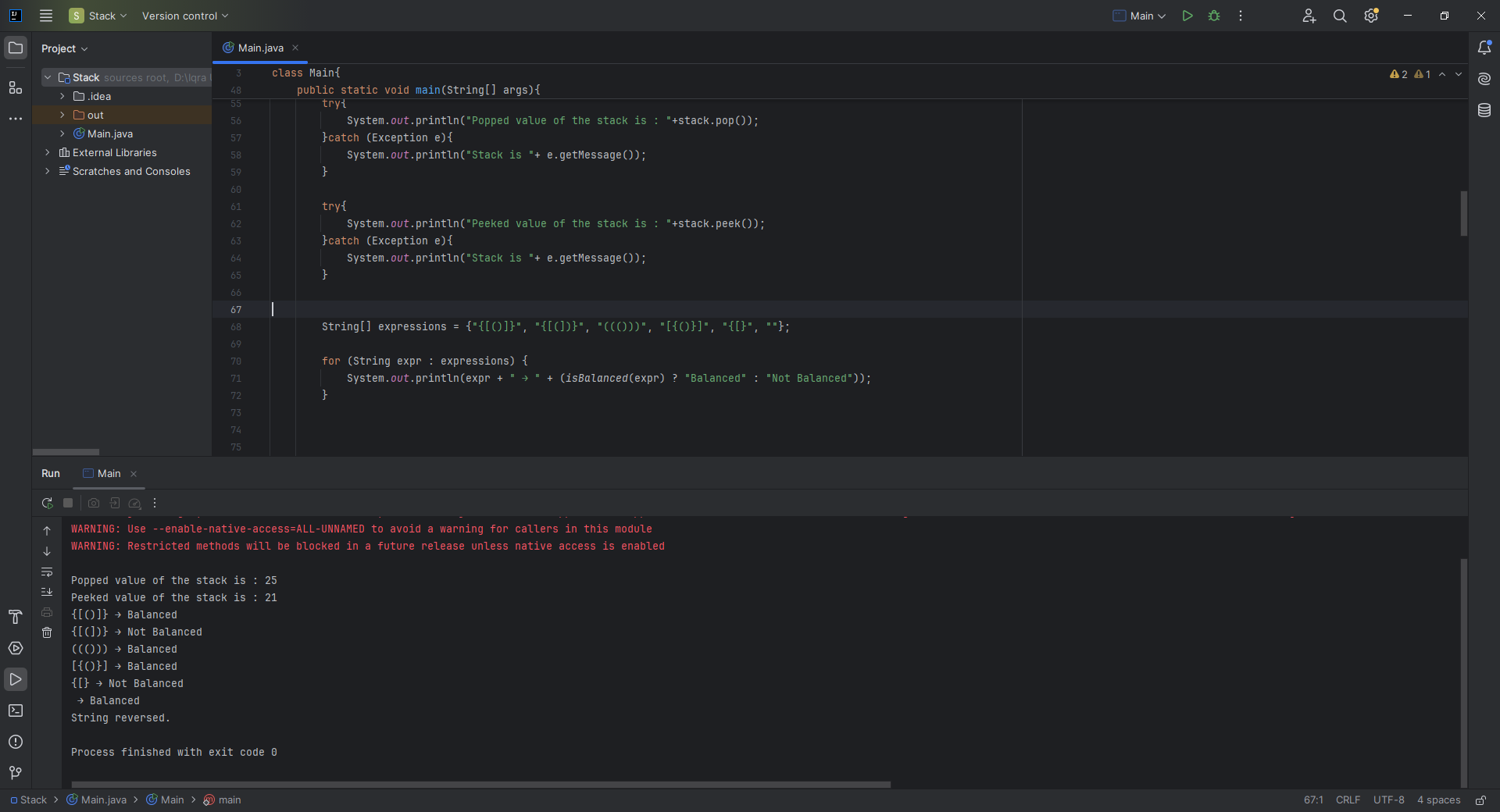
return fibonacci(n - 1) + fibonacci(n - 2);

}

**Home Task 1:** Balancing Parentheses Using Stack

**CODE:**

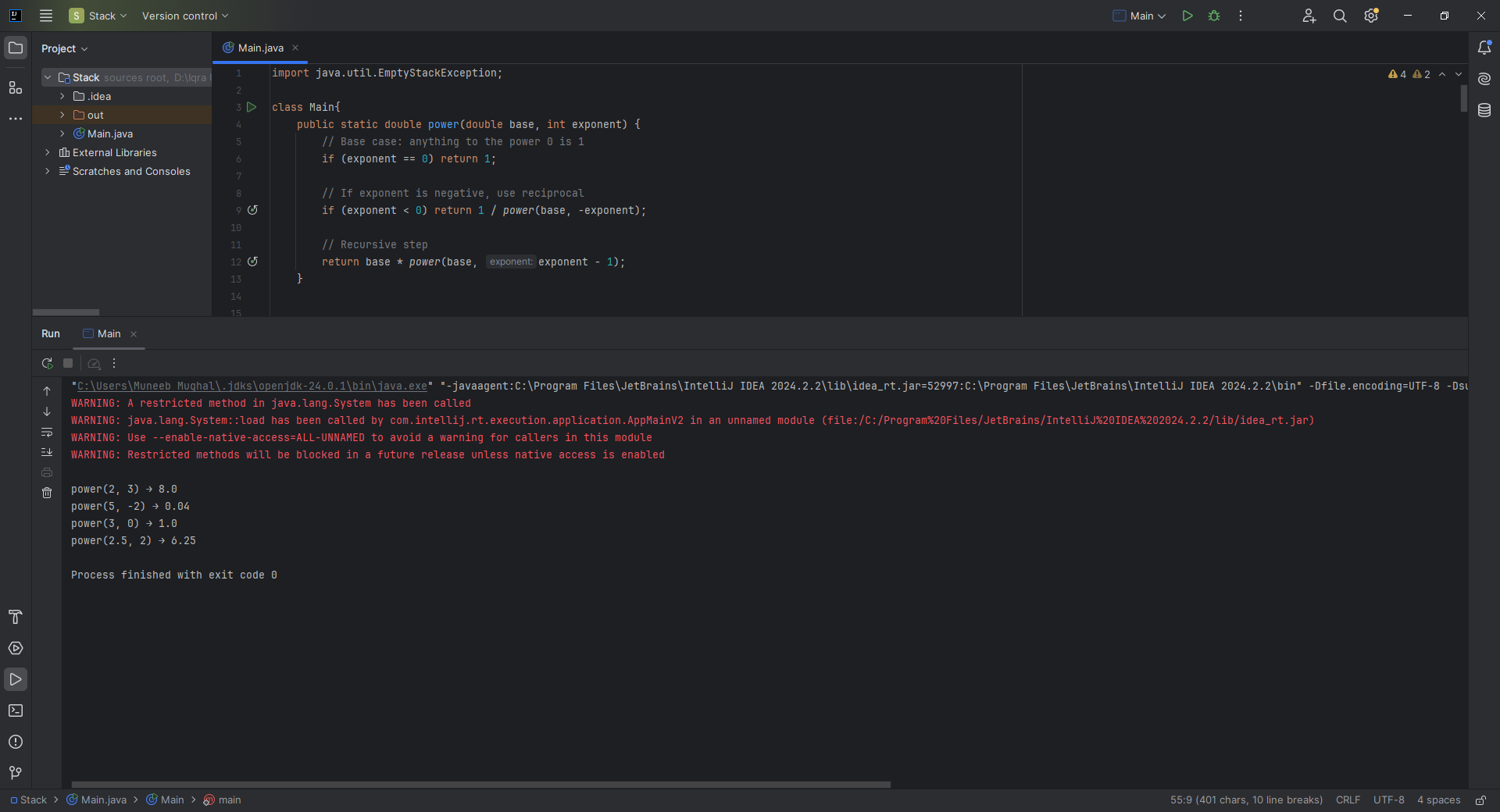
public static boolean isBalanced(String expression) {  
 Stack stack = new Stack(expression.length());  
  
 for (char ch : expression.toCharArray()) {  
 int mapped = *mapBracket*(ch);  
  
 if (mapped > 0) {  
 stack.push(mapped); // Opening bracket  
 } else if (mapped < 0) {  
 if (stack.isEmpty()) return false;  
  
 int top = stack.pop();  
 if (top + mapped != 0) return false; // Mismatched brackets  
 }  
 }  
  
 return stack.isEmpty(); // Must be empty if balanced  
}



**Home Task 2:** Recursive Solution for Power Calculation

**CODE:**

public static double power(double base, int exponent) {  
 // Base case: anything to the power 0 is 1  
 if (exponent == 0) return 1;  
  
 // If exponent is negative, use reciprocal  
 if (exponent < 0) return 1 / *power*(base, -exponent);  
  
 // Recursive step  
 return base \* *power*(base, exponent - 1);  
}



**Home Task 3:** Implementing Tower of Hanoi with Recursion

**CODE:**

static void Hanoi\_Problem() {  
 int numDisks = 3;  
  
 Stack A = new Stack(5); // source  
 Stack B = new Stack(5); // auxiliary  
 Stack C = new Stack(5); // target  
  
 // Push disks in reverse order (largest at bottom)  
 for (int i = numDisks; i >= 1; i--) {  
 A.push(i);  
 }  
  
 *moveDisks*(numDisks, A, C, B, 'A', 'C', 'B');  
  
 System.*out*.println("Hanoi Problem solved.");  
}  
  
static void moveDisks(int n, Stack source, Stack target, Stack auxiliary, char from, char to, char via) {  
 if (n == 1) {  
 int disk = source.pop();  
 target.push(disk);  
 System.*out*.println("Move disk " + disk + " from " + from + " to " + to);  
 return;  
 }  
  
 *moveDisks*(n - 1, source, auxiliary, target, from, via, to);  
 int disk = source.pop();  
 target.push(disk);  
 System.*out*.println("Move disk " + disk + " from " + from + " to " + to);  
 *moveDisks*(n - 1, auxiliary, target, source, via, to, from);  
}

