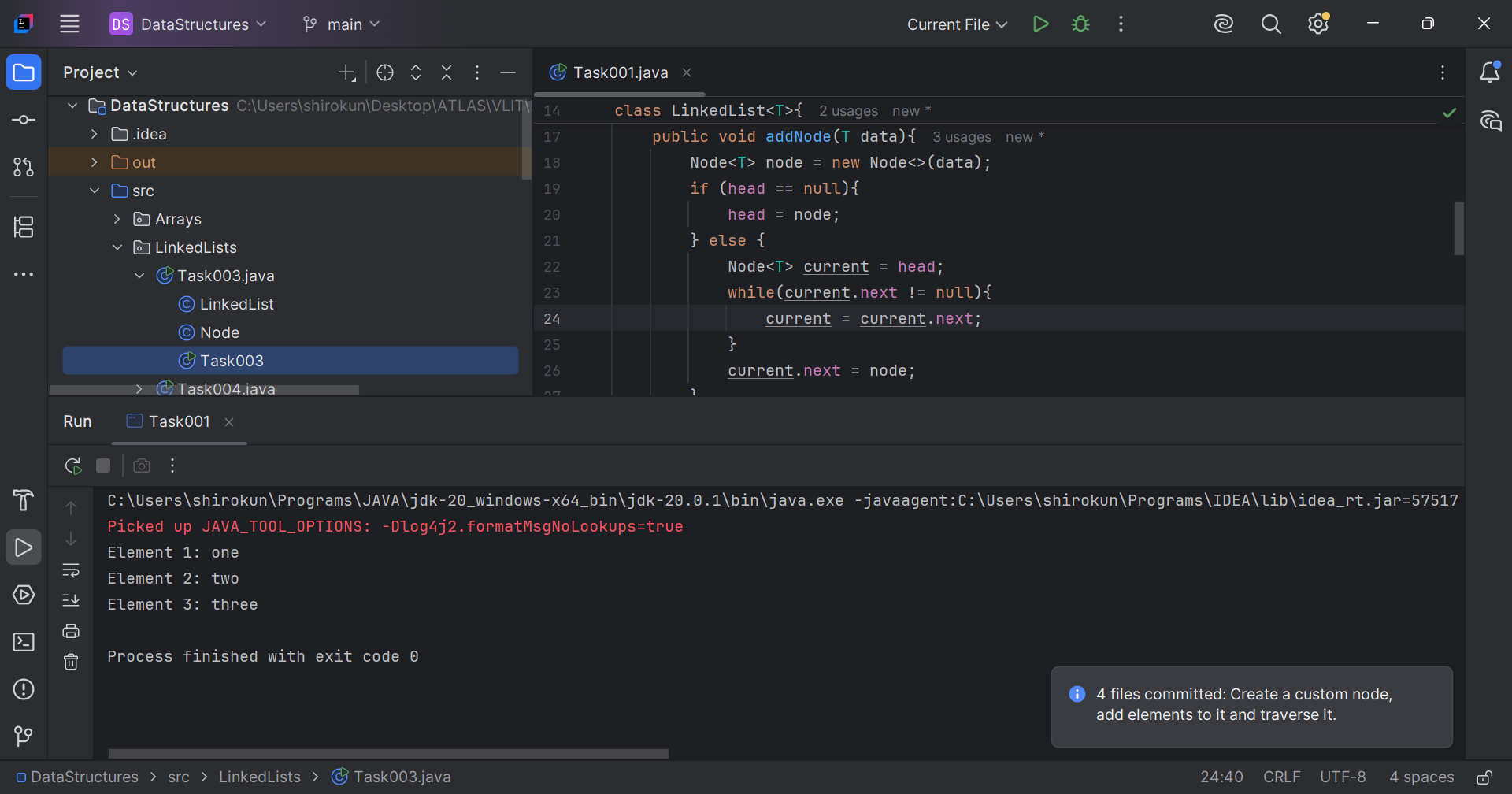
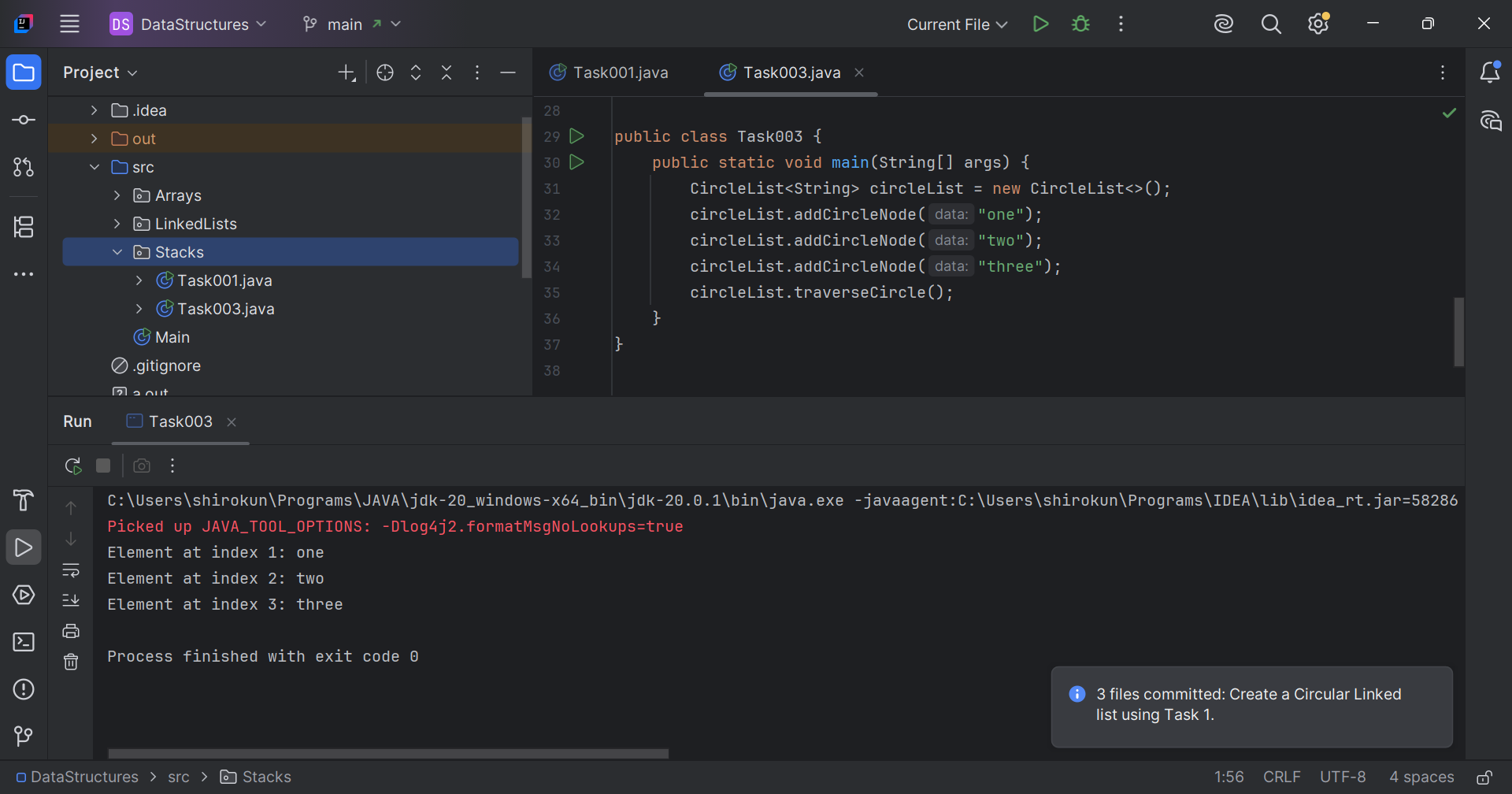
Day 14 – 04/07/2025

// Task001: Create a custom node, add elements to it and traverse it.  
  
package Stacks;  
  
class Node<T>{  
 T data;  
 Node<T> next;  
 public Node(T data){  
 this.data = data;  
 this.next = null;  
 }  
}  
  
class LinkedList<T>{  
 Node<T> head;  
  
 public void addNode(T data){  
 Node<T> node = new Node<>(data);  
 if (head == null){  
 head = node;  
 } else {  
 Node<T> current = head;  
 while(current.next != null){  
 current = current.next;  
 }  
 current.next = node;  
 }  
 }  
  
 public void traverse(){  
 int index = 1;  
 Node<T> current = head;  
 while (current != null){  
 System.*out*.println("Element "+index+": "+current.data);  
 current = current.next;  
 index++;  
 }  
 }  
}  
  
public class Task001 {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.addNode("one");  
 list.addNode("two");  
 list.addNode("three");  
 list.traverse();  
 }  
}



Q2. What do you understand by traversing a linked list?  
Ans. Traverse means travel. So, in a linked list traverse means travelling across the linked list element by element. Traversing helps us to search, delete or print elements.

// Task003: Create a Circular Linked list using Task 1.  
  
package Stacks;  
  
class CircleList<T> extends LinkedList<T>{  
 public void addCircleNode(T data) {  
 Node<T> node = new Node<>(data);  
  
 if(head == null){  
 head = node;  
 } else {  
 tail.next = node;  
 }  
 tail = node;  
 tail.next = head;  
 }  
  
 public void traverseCircle(){  
 Node<T> current = head;  
 int index=1;  
 do {  
 System.*out*.println("Element at index "+index+": "+current.data);  
 current = current.next;  
 index++;  
 }while (current != head);  
 }  
}  
  
public class Task003 {  
 public static void main(String[] args) {  
 CircleList<String> circleList = new CircleList<>();  
 circleList.addCircleNode("one");  
 circleList.addCircleNode("two");  
 circleList.addCircleNode("three");  
 circleList.traverseCircle();  
 }  
}



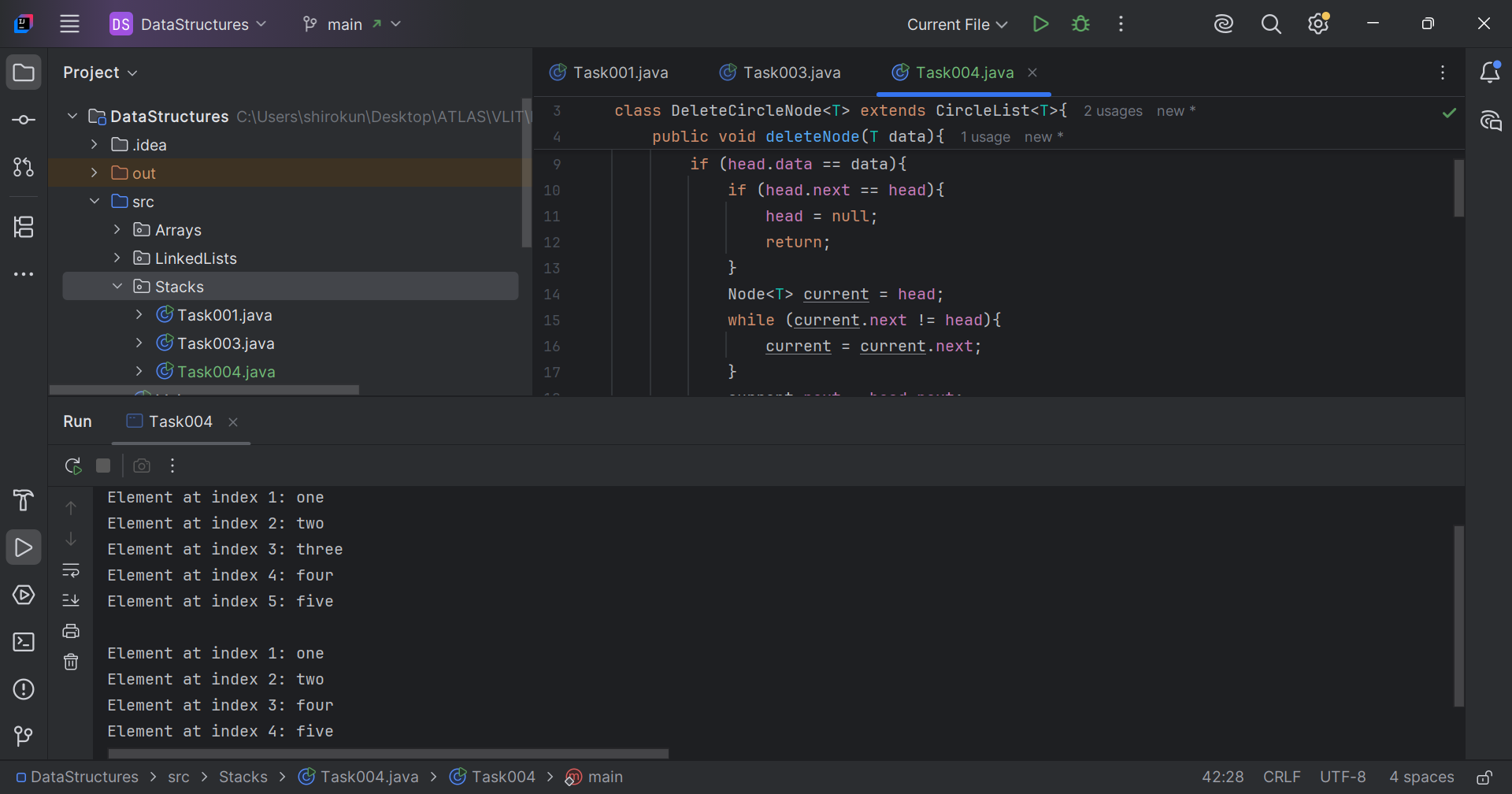
// Task004: Delete a node in the circular linked list.  
  
package Stacks;

class DeleteCircleNode<T> extends CircleList<T>{  
 public void deleteNode(T data){  
 if(head == null){  
 System.*out*.println("List is empty.");  
 return;  
 }

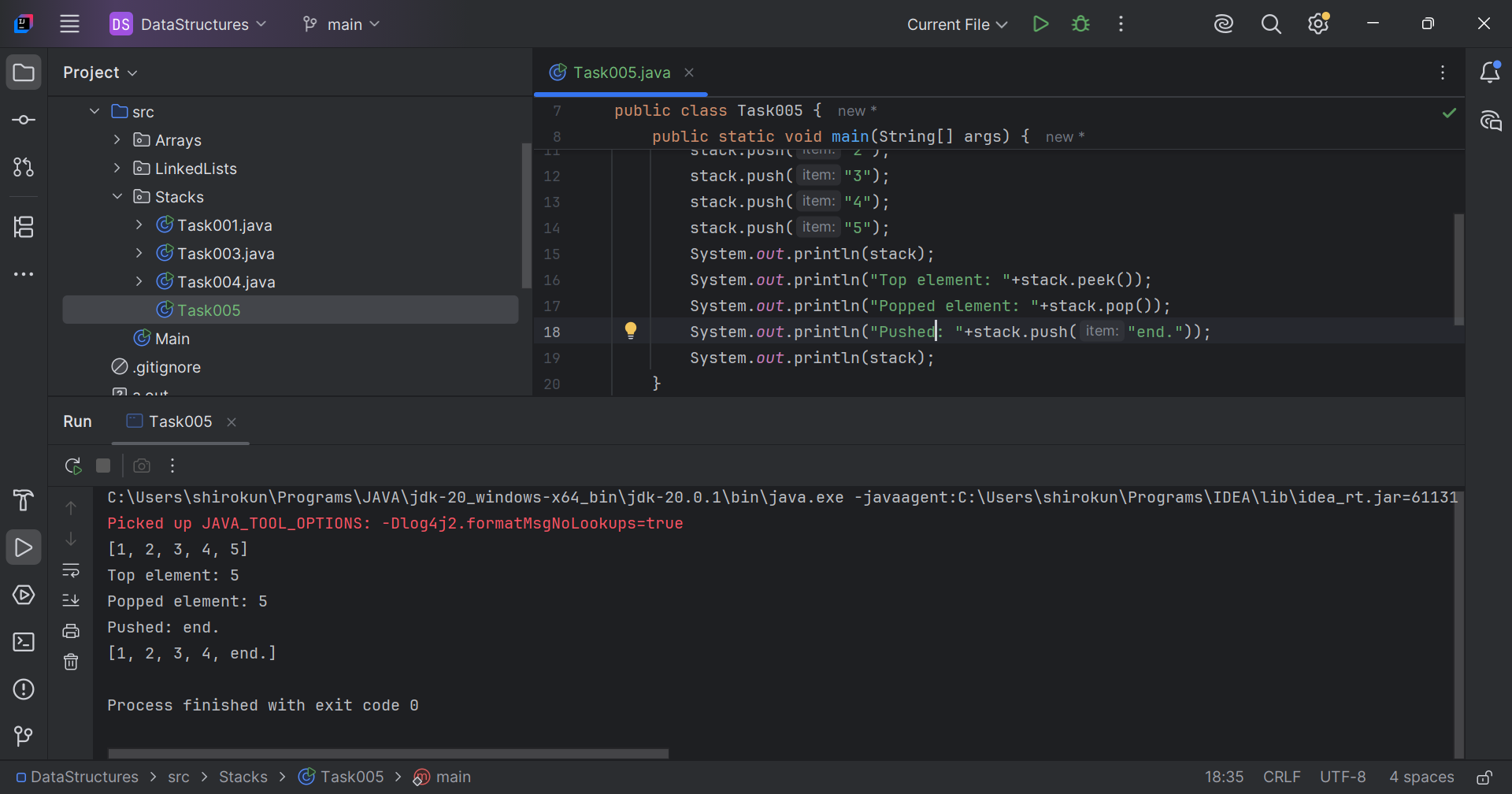
if (head.data == data){

if (head.next == head){  
 head = null;  
 return;  
 }

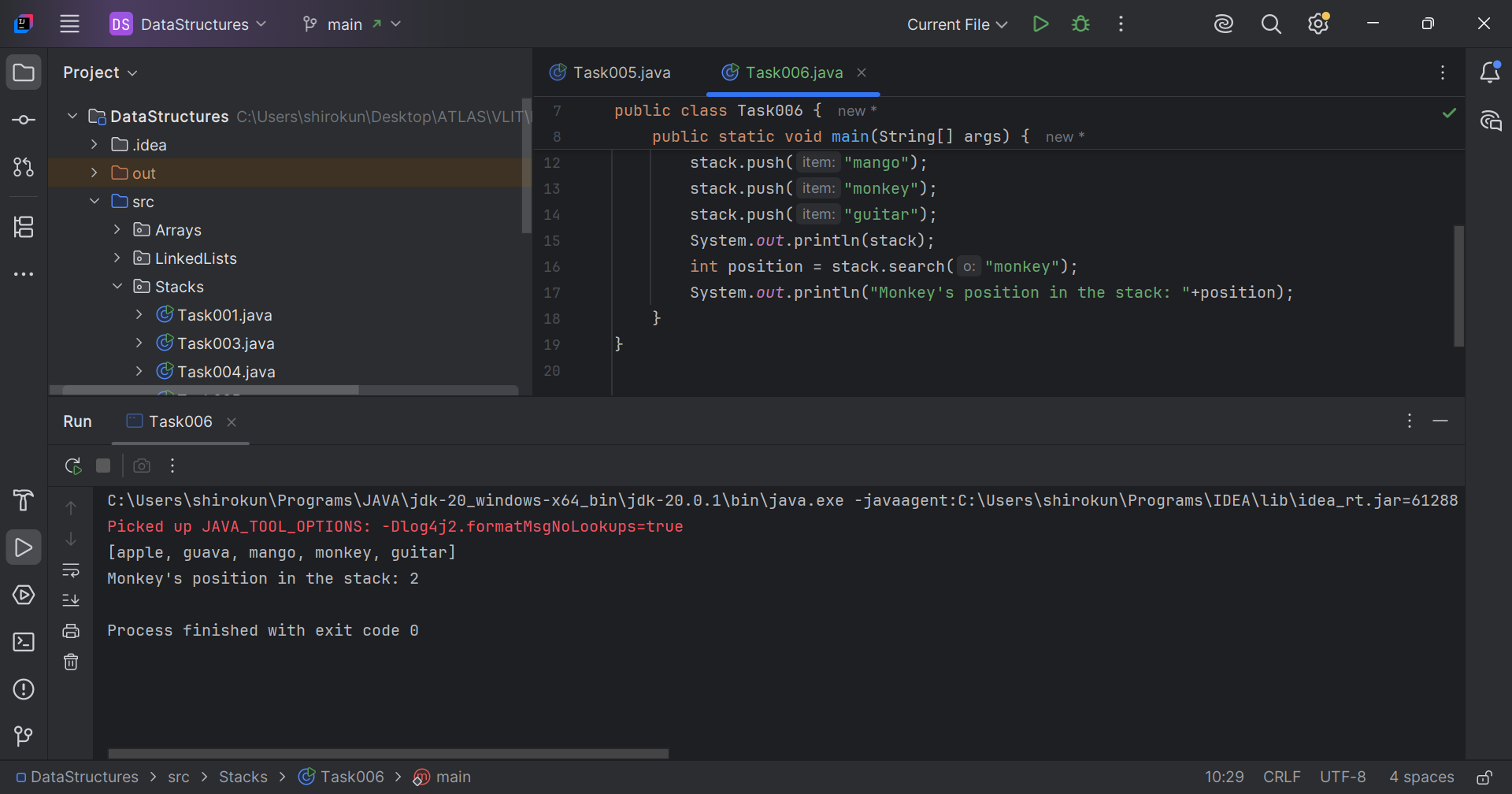
Node<T> current = head;  
 while (current.next != head){  
 current = current.next;  
 }  
 current.next = head.next;  
 head = head.next;  
 return;  
 }  
 Node<T> current = head;  
 while (current.next != head){  
 if(current.next.data == data){  
 current.next = current.next.next;  
 return;  
 }  
 current = current.next;  
 }  
 }  
}  
  
public class Task004 {  
 public static void main(String[] args) {  
 DeleteCircleNode<String> list = new DeleteCircleNode<>();  
 list.addCircleNode("one");  
 list.addCircleNode("two");  
 list.addCircleNode("three");  
 list.addCircleNode("four");  
 list.addCircleNode("five");  
 list.traverseCircle();  
 System.*out*.println();  
 list.deleteNode("three");  
 list.traverseCircle();  
 }  
}



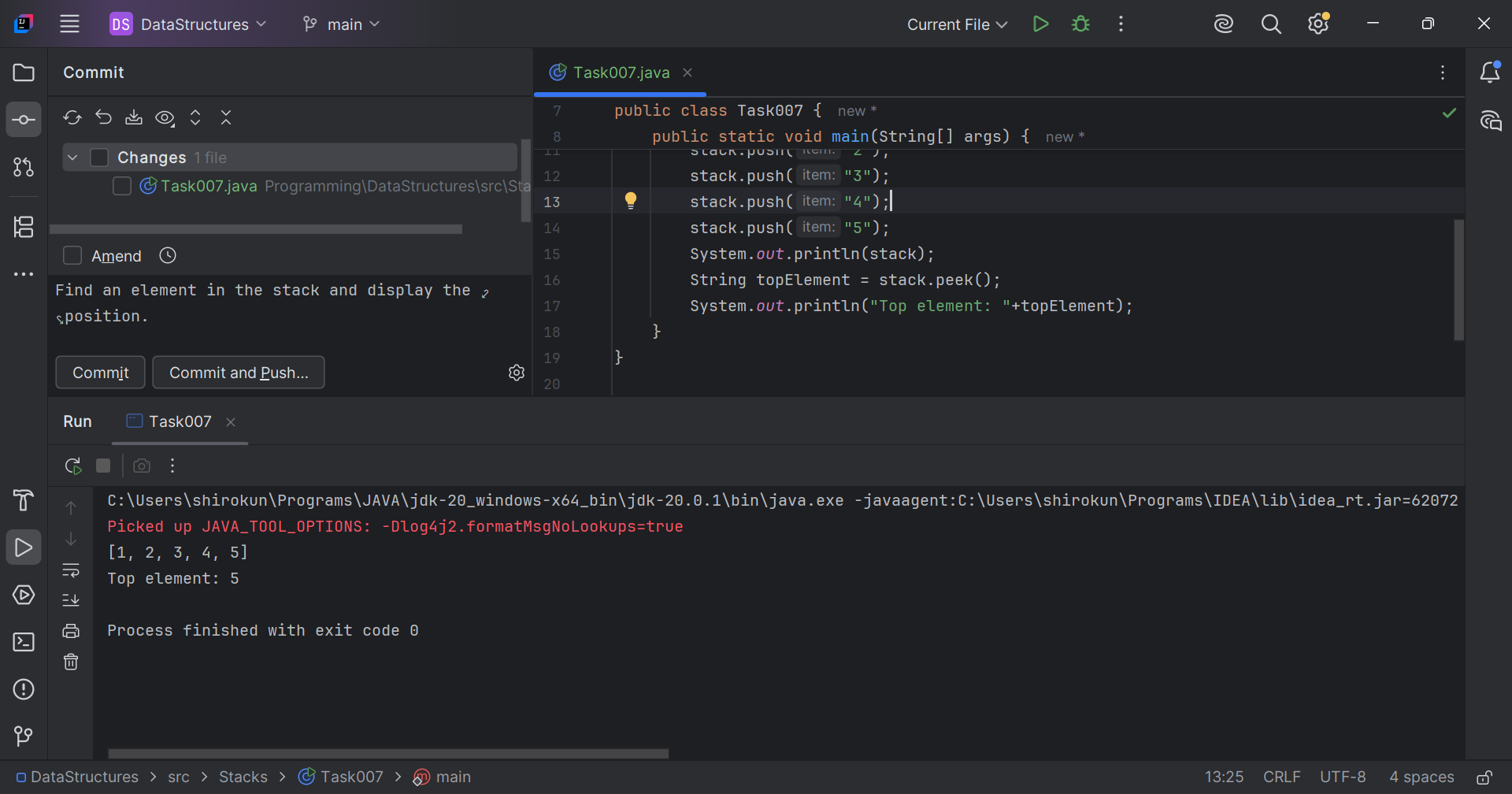
// Task005: Create a stack, pop the element and print it.  
  
package Stacks;  
  
import java.util.Stack;  
  
public class Task005 {  
 public static void main(String[] args) {  
 Stack<String> stack = new Stack<>();  
 stack.push("1");  
 stack.push("2");  
 stack.push("3");  
 stack.push("4");  
 stack.push("5");  
 System.*out*.println(stack);  
 System.*out*.println("Top element: "+stack.peek());  
 System.*out*.println("Popped element: "+stack.pop());  
 System.*out*.println("Pushed: "+stack.push("end."));  
 System.*out*.println(stack);  
 }  
}



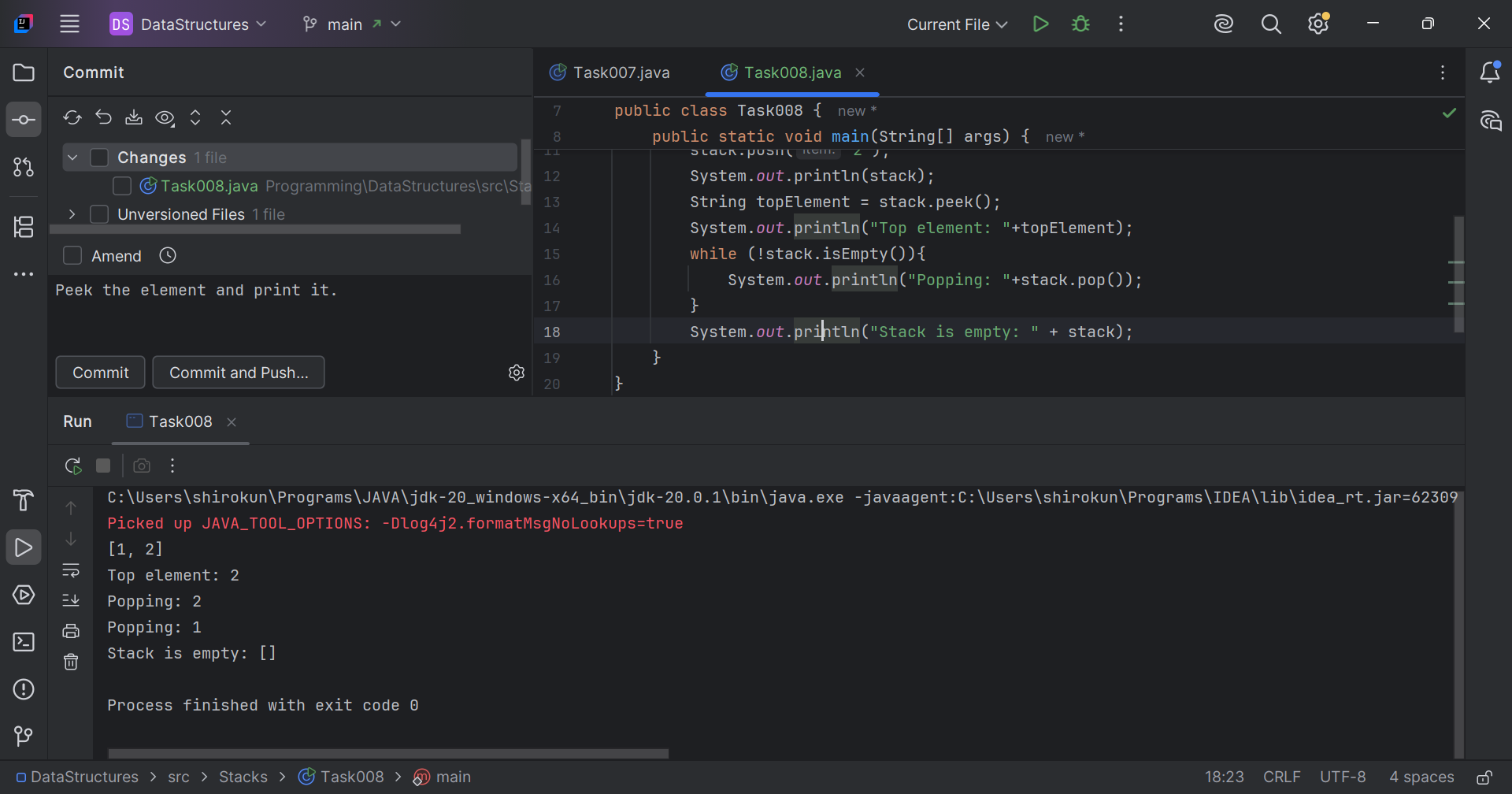
// Task006: Find an element in the stack and display the position.  
  
package Stacks;  
  
import java.util.Stack;  
  
public class Task006 {  
 public static void main(String[] args) {  
 Stack<String> stack = new Stack<>();  
 stack.push("apple");  
 stack.push("guava");  
 stack.push("mango");  
 stack.push("monkey");  
 stack.push("guitar");  
 System.*out*.println(stack);  
 int position = stack.search("monkey");  
 System.*out*.println("Monkey's position in the stack: "+position);  
 }  
}



// Task007: Peek the element and print it.  
  
package Stacks;  
  
import java.util.Stack;  
  
public class Task007 {  
 public static void main(String[] args) {  
 Stack<String> stack = new Stack<>();  
 stack.push("1");  
 stack.push("2");  
 stack.push("3");  
 stack.push("4");  
 stack.push("5");  
 System.*out*.println(stack);  
 String topElement = stack.peek();  
 System.*out*.println("Top element: "+topElement);  
 }  
}



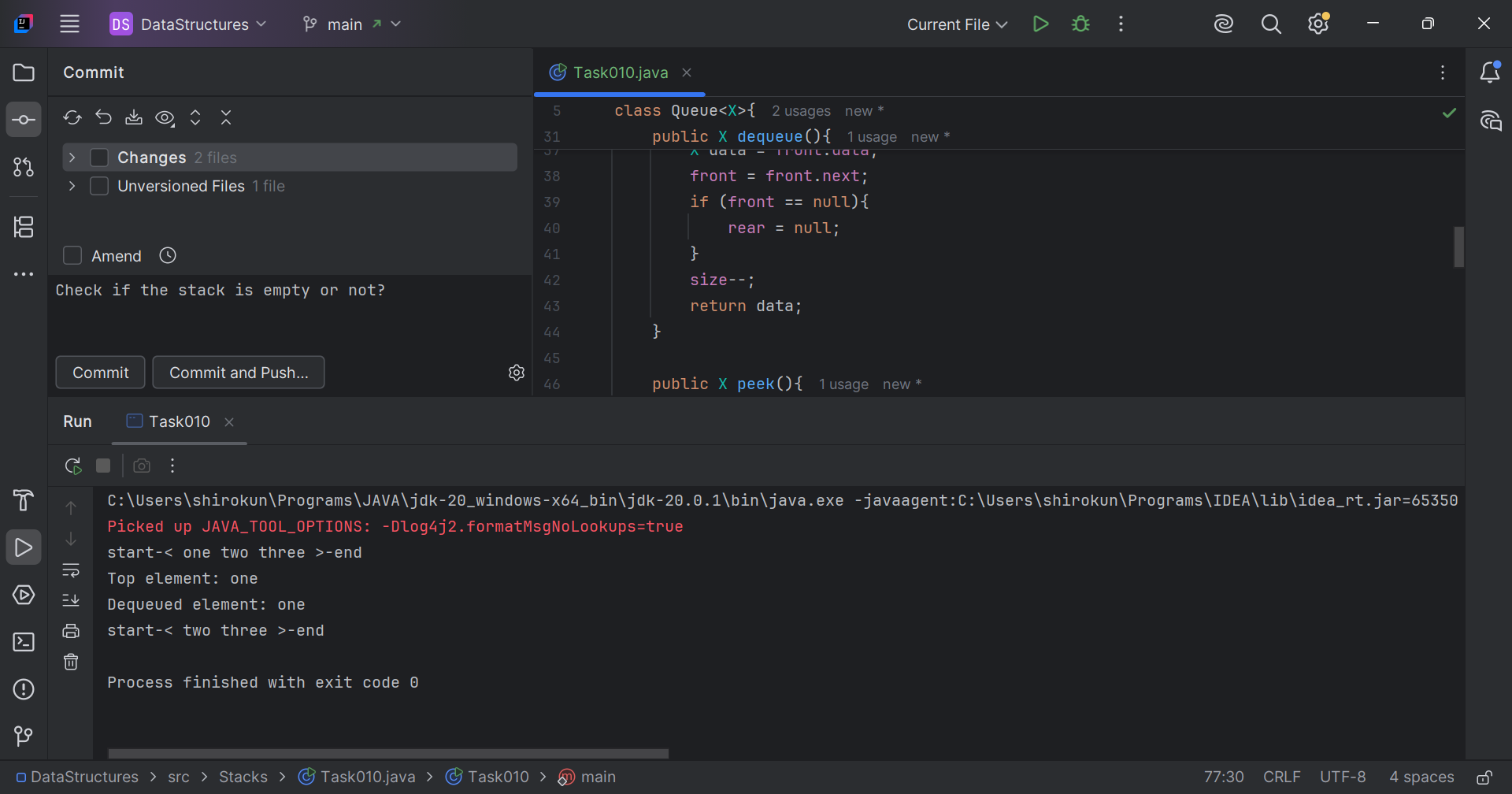
// Task008: Check if the stack is empty or not?  
  
package Stacks;  
  
import java.util.Stack;  
  
public class Task008 {  
 public static void main(String[] args) {  
 Stack<String> stack = new Stack<>();  
 stack.push("1");  
 stack.push("2");  
 System.*out*.println(stack);  
 String topElement = stack.peek();  
 System.*out*.println("Top element: "+topElement);  
 while (!stack.isEmpty()){  
 System.*out*.println("Popping: "+stack.pop());  
 }  
 System.*out*.println("Stack is empty: " + stack);  
 }  
}



Q9. What are the methods of the stack class? List them down with a one liner.  
Ans. Stack methods –

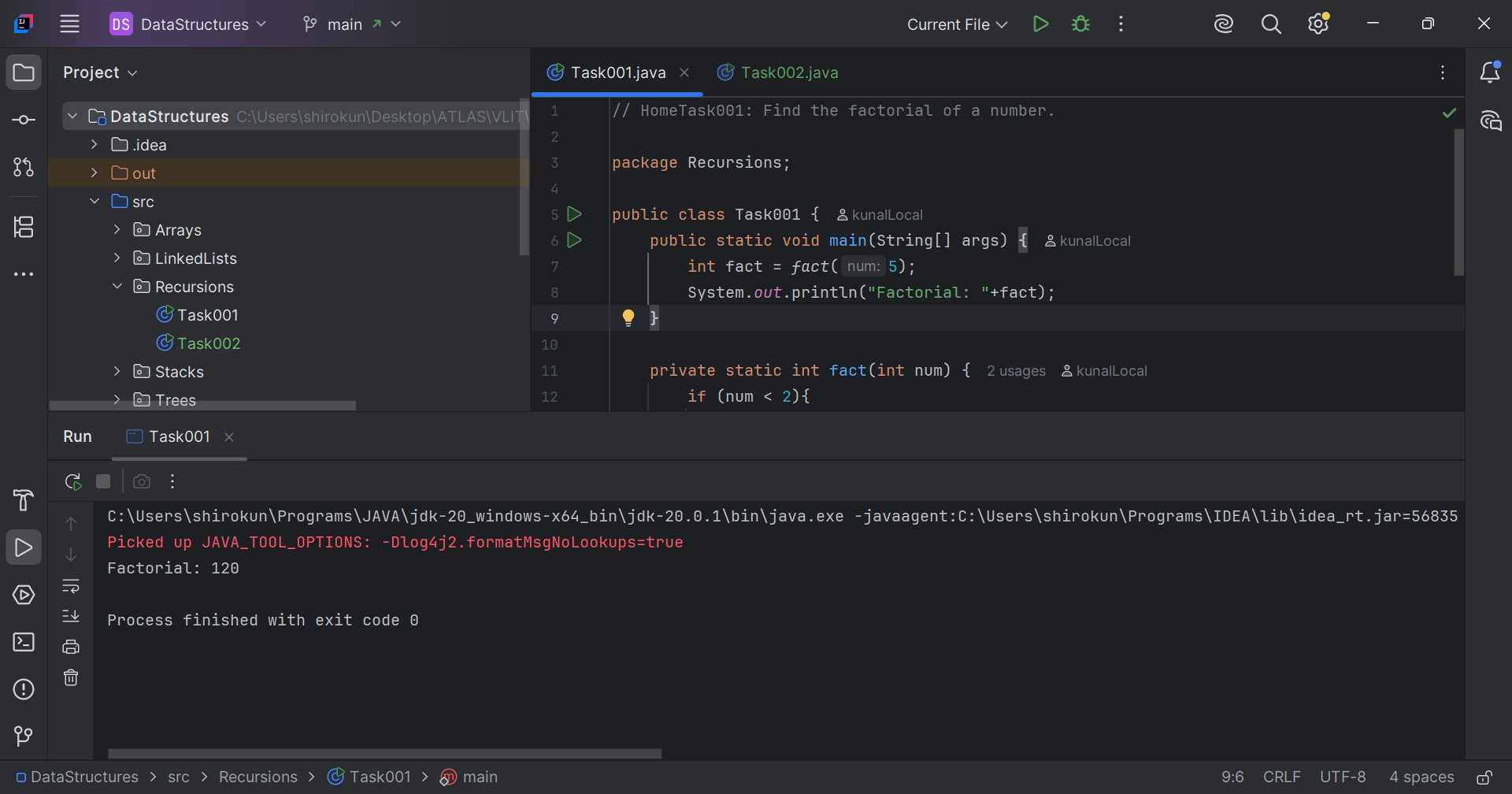
* push(E item): Adds an element to the top of the stack.
* pop(): Removes the top element from the stack and returns it.
* peek(): Returns the top element from the stack without removing it.
* isEmpty(): Checks if the stack is empty.
* search(Object o): Returns the distance from the top of the stack to the specified element.

// Task010: create a queue with custom methods.  
  
package Stacks;  
  
class Queue<X>{  
 Node<X> front, rear;  
 int size, cap;  
  
 public Queue(int cap){  
 this.cap = cap;  
 this.size = 0;  
 this.front = null;  
 this.rear = null;  
 }  
  
 public void enqueue(X data){  
 if (isFull()){  
 System.*out*.println("Reached capacity!!");  
 return;  
 }  
 Node<X> node = new Node<>(data);  
 if (rear == null){  
 front = node;  
 } else {  
 rear.next = node;  
 }  
 rear = node;  
 size++;  
 }  
  
 public X dequeue(){  
 if (isEmpty()){  
 System.*out*.println("Queue is empty.");  
 return null;  
 }  
  
 X data = front.data;  
 front = front.next;  
 if (front == null){  
 rear = null;  
 }  
 size--;  
 return data;  
 }  
  
 public X peek(){  
 if (rear == null){  
 System.*out*.println("Queue is empty.");  
 return null;  
 }  
 return front.data;  
 }  
  
 public void display(){  
 System.*out*.print("start-< ");  
 Node<X> current = front;  
 while (current != null){  
 System.*out*.print(current.data+" ");  
 current = current.next;  
 }  
 System.*out*.println(">-end");  
 }  
  
 public boolean isFull(){  
 return size == cap;  
 }  
  
 public boolean isEmpty(){  
 return size == 0;  
 }  
}  
  
public class Task010 {  
 public static void main(String[] args) {  
 Queue<String> queue = new Queue<>(5);  
 queue.enqueue("one");  
 queue.enqueue("two");  
 queue.enqueue("three");  
 queue.display();  
 System.*out*.println("Top element: "+queue.peek());  
 System.*out*.println(("Dequeued element: "+queue.dequeue()));  
 queue.display();  
  
 }  
}

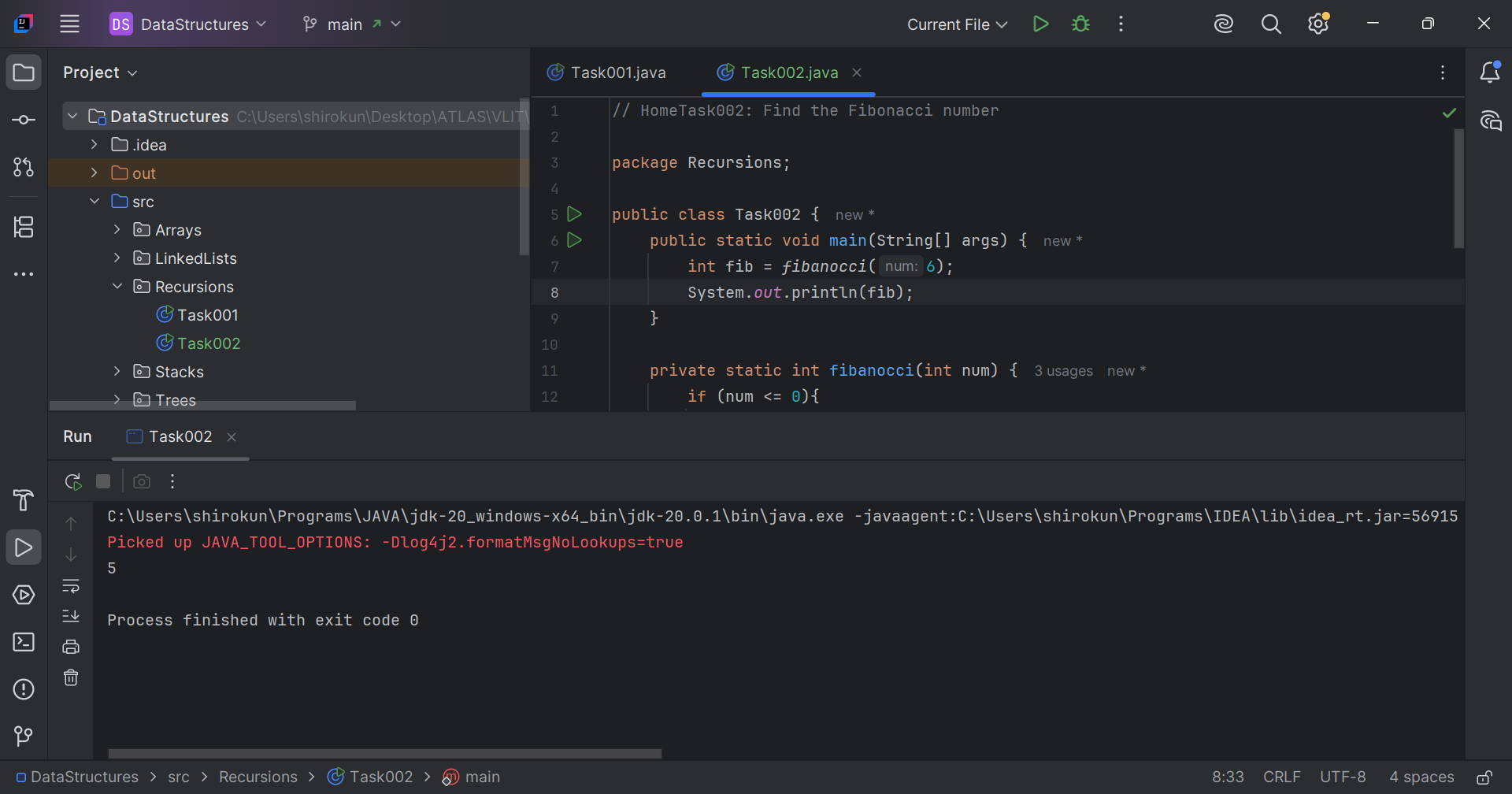


Home Tasks.

// HomeTask001: Find the factorial of a number.  
  
package Recursions;  
  
public class Task001 {  
 public static void main(String[] args) {  
 int fact = *fact*(5);  
 System.*out*.println("Factorial: "+fact);  
 }  
  
 private static int fact(int num) {  
 if (num < 2){  
 return num;  
 }  
 return num\**fact*(num-1);  
 }  
}



// HomeTask002: Find the Fibonacci number  
  
package Recursions;  
  
public class Task002 {  
 public static void main(String[] args) {  
 int fib = *fibanocci*(6);  
 System.*out*.println(fib);  
 }  
  
 private static int fibanocci(int num) {  
 if (num <= 0){  
 throw new IllegalArgumentException("Incorrect number");  
 } else if (num == 1) {  
 return 0;  
 } else if(num == 2){  
 return 1;  
 } else{  
 return *fibanocci*(num-1)+*fibanocci*(num-2);  
 }  
 }  
}



Q3. What is the difference between recursion and loops?  
Ans.

Iteration:

* Uses loops (e.g., for, while) to repeat a process.
* Each iteration builds on the previous one.
* Typically uses a fixed amount of memory.

Recursion:

* A function calls itself repeatedly.
* Each call creates a new stack frame.
* Can use more memory due to the recursive call stack.

// HomeTask004: Reverse a string using recursion.  
  
package Recursions;  
  
public class Task004 {  
 public static void main(String[] args) {  
 String rev, str = "Hello";  
 rev = *reverseStr*(str);  
 System.*out*.println(rev);  
 }  
  
 private static String reverseStr(String str) {  
 if (str.length() == 1){  
 return str;  
 } else {  
 return *reverseStr*(str.substring(1)) + str.charAt(0);  
 }  
 }  
}

