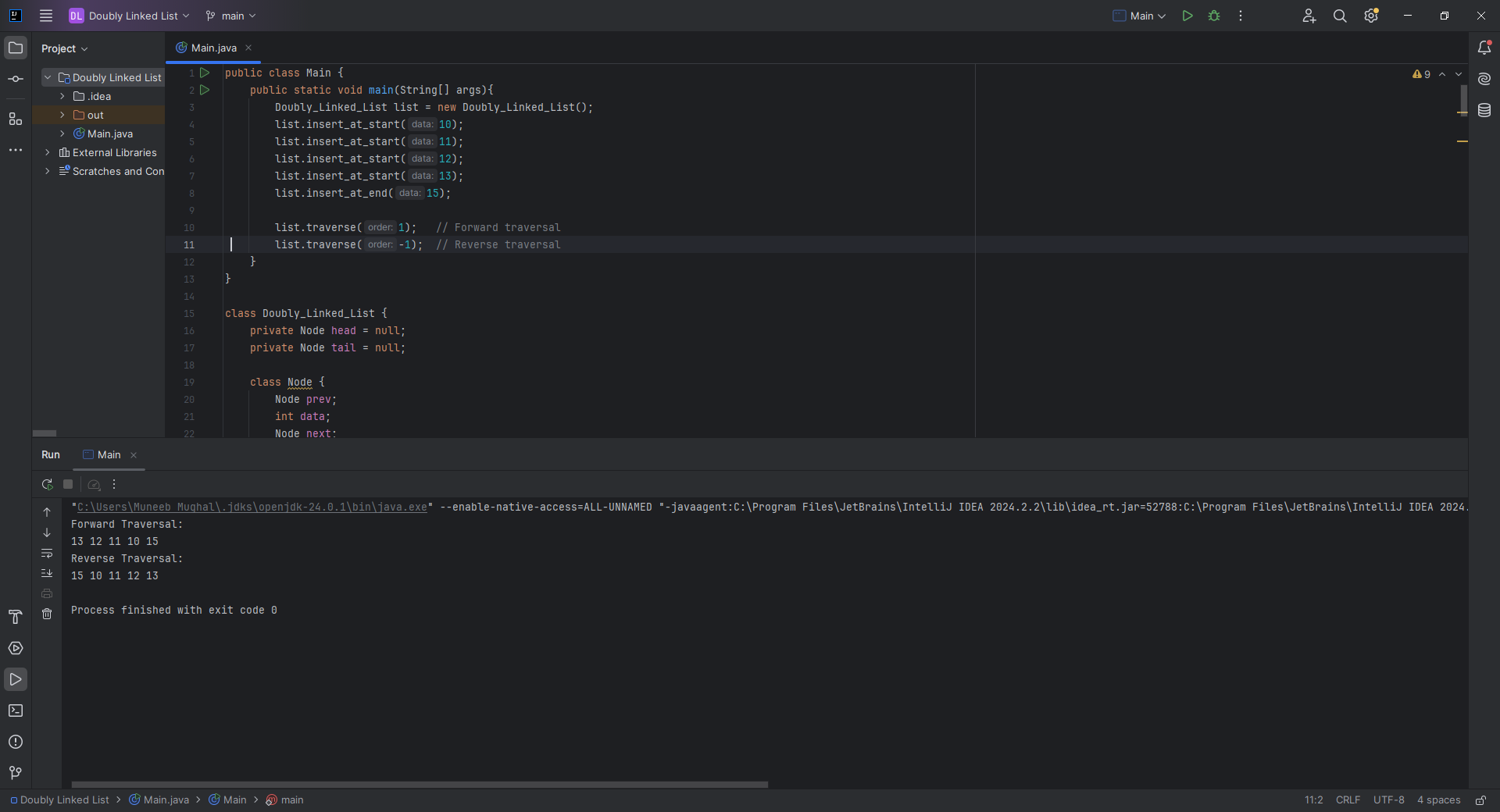
**CLASS TASKS**

**CODE:**

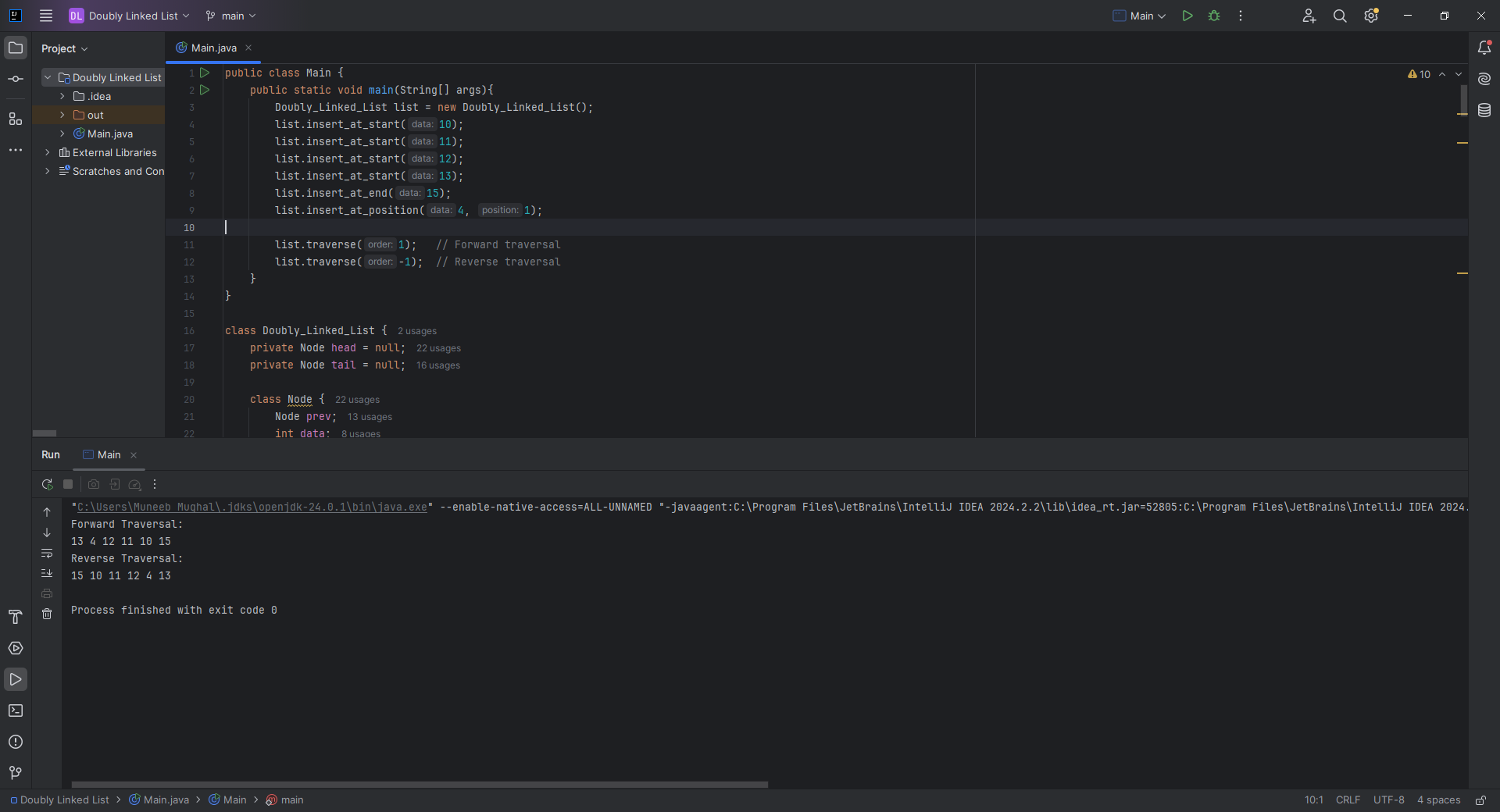
public class Main {  
 public static void main(String[] args){  
 Doubly\_Linked\_List list = new Doubly\_Linked\_List();  
 list.insert\_at\_start(10);  
 list.insert\_at\_start(11);  
 list.insert\_at\_start(12);  
 list.insert\_at\_start(13);  
 list.insert\_at\_end(15);  
 list.insert\_after(10, 15);  
 list.insert\_at\_position(100,0);  
 list.insert\_at\_position(100,3);  
 list.sort();  
  
 System.*out*.println("11 found at index "+ list.find(15));  
 System.*out*.println("Occurance of 15 in the list is : "+ list.countOccurrences(15));  
  
 list.traverse(1); // Forward traversal  
 list.traverse(-1); // Reverse traversal  
 }  
}  
  
class Doubly\_Linked\_List {  
 private Node head = null;  
 private Node tail = null;  
  
 class Node {  
 Node prev;  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 }  
 }  
  
 void insert\_at\_start(int data){  
 Node newNode = new Node(data);  
 if(head == null){  
 head = tail = newNode;  
 } else {  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 }  
  
 void insert\_at\_end(int data) {  
 Node newNode = new Node(data);  
 if(tail == null){  
 head = tail = newNode;  
 } else {  
 tail.next = newNode;  
 newNode.prev = tail;  
 tail = newNode;  
 }  
 }  
  
 void insert\_after(int val, int data){  
 if(head == null){  
 System.*out*.println("List is empty");  
 return;  
 }  
 Node current = head;  
 while(current != null){  
 if(current.data == val){  
 Node newNode = new Node(data);  
 newNode.next = current.next;  
 newNode.prev = current;  
 if(current.next != null){  
 current.next.prev = newNode;  
 } else {  
 // We're inserting at the end  
 tail = newNode;  
 }  
 current.next = newNode;  
 return;  
 }  
 current = current.next;  
 }  
 System.*out*.println(val + " is not in the list");  
 }  
  
 void delete\_from\_start(){  
 if(head == null){  
 return;  
 }  
 head = head.next;  
 if(head != null){  
 head.prev = null;  
 } else {  
 tail = null;  
 }  
 }  
  
 void delete\_from\_end(){  
 if(tail == null){  
 return;  
 }  
 tail = tail.prev;  
 if(tail != null){  
 tail.next = null;  
 } else {  
 head = null;  
 }  
 }  
  
 void traverse(int order){  
 if(head == null){  
 System.*out*.println("Empty List");  
 return;  
 }  
  
 if(order == 1){  
 System.*out*.println("Forward Traversal:");  
 Node current = head;  
 while(current != null){  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 } else if (order == -1){  
 System.*out*.println("Reverse Traversal:");  
 Node current = tail;  
 while(current != null){  
 System.*out*.print(current.data + " ");  
 current = current.prev;  
 }  
 System.*out*.println();  
 } else {  
 System.*out*.println("Invalid traversal order. Use 1 (forward) or -1 (reverse).");  
 }  
 }  
}



**Home Task 1:** Inserting at a Specific Position

**CODE:**

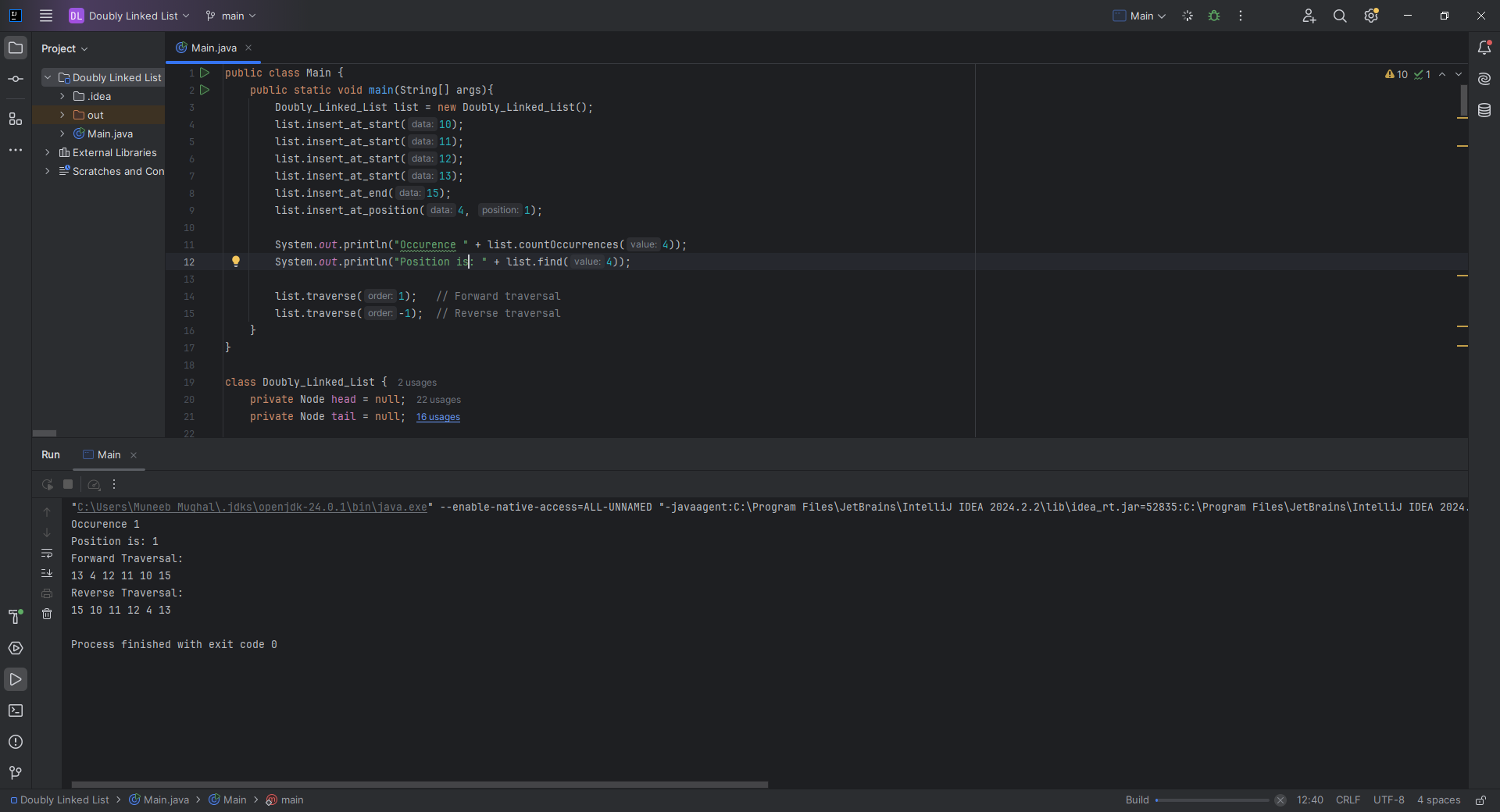
void insert\_at\_position(int data, int position) {  
 if (position < 0) {  
 System.*out*.println("Invalid position");  
 return;  
 }  
 if (position == 0) {  
 insert\_at\_start(data);  
 return;  
 }  
 Node newNode = new Node(data);  
 Node current = head;  
 int index = 0;  
  
 while (current != null && index < position - 1) {  
 current = current.next;  
 index++;  
 }  
  
 if (current == null) {  
 System.*out*.println("Position out of bounds");  
 return;  
 }  
  
 newNode.next = current.next;  
 newNode.prev = current;  
  
 if (current.next != null) {  
 current.next.prev = newNode;  
 } else {  
 tail = newNode; // If inserted at the end  
 }  
 current.next = newNode;  
}

****

**Home Task 2:** Finding and Counting Elements

**CODE:**

int find(int value) {  
 Node current = head;  
 int position = 0;  
 while (current != null) {  
 if (current.data == value) {  
 return position;  
 }  
 current = current.next;  
 position++;  
 }  
 return -1; // Not found  
}  
  
int countOccurrences(int value) {  
 Node current = head;  
 int count = 0;  
 while (current != null) {  
 if (current.data == value) {  
 count++;  
 }  
 current = current.next;  
 }  
 return count;  
}



**Home Task 3:** Sorting the Doubly Linked List

**CODE:**

void sort() {  
 if (head == null) return;  
  
 boolean swapped;  
 do {  
 swapped = false;  
 Node current = head;  
  
 while (current.next != null) {  
 if (current.data > current.next.data) {  
 // Swap nodes by rearranging pointers (not just data)  
 Node nextNode = current.next;  
 Node prevNode = current.prev;  
 Node nextNext = nextNode.next;  
  
 // Re-link previous node to nextNode  
 if (prevNode != null) {  
 prevNode.next = nextNode;  
 } else {  
 head = nextNode;  
 }  
 nextNode.prev = prevNode;  
  
 // Link nextNode and current node  
 nextNode.next = current;  
 current.prev = nextNode;  
  
 // Link current node to nextNext node  
 current.next = nextNext;  
 if (nextNext != null) {  
 nextNext.prev = current;  
 } else {  
 tail = current;  
 }  
  
 swapped = true;  
 } else {  
 current = current.next;  
 }  
 }  
 } while (swapped);  
}

