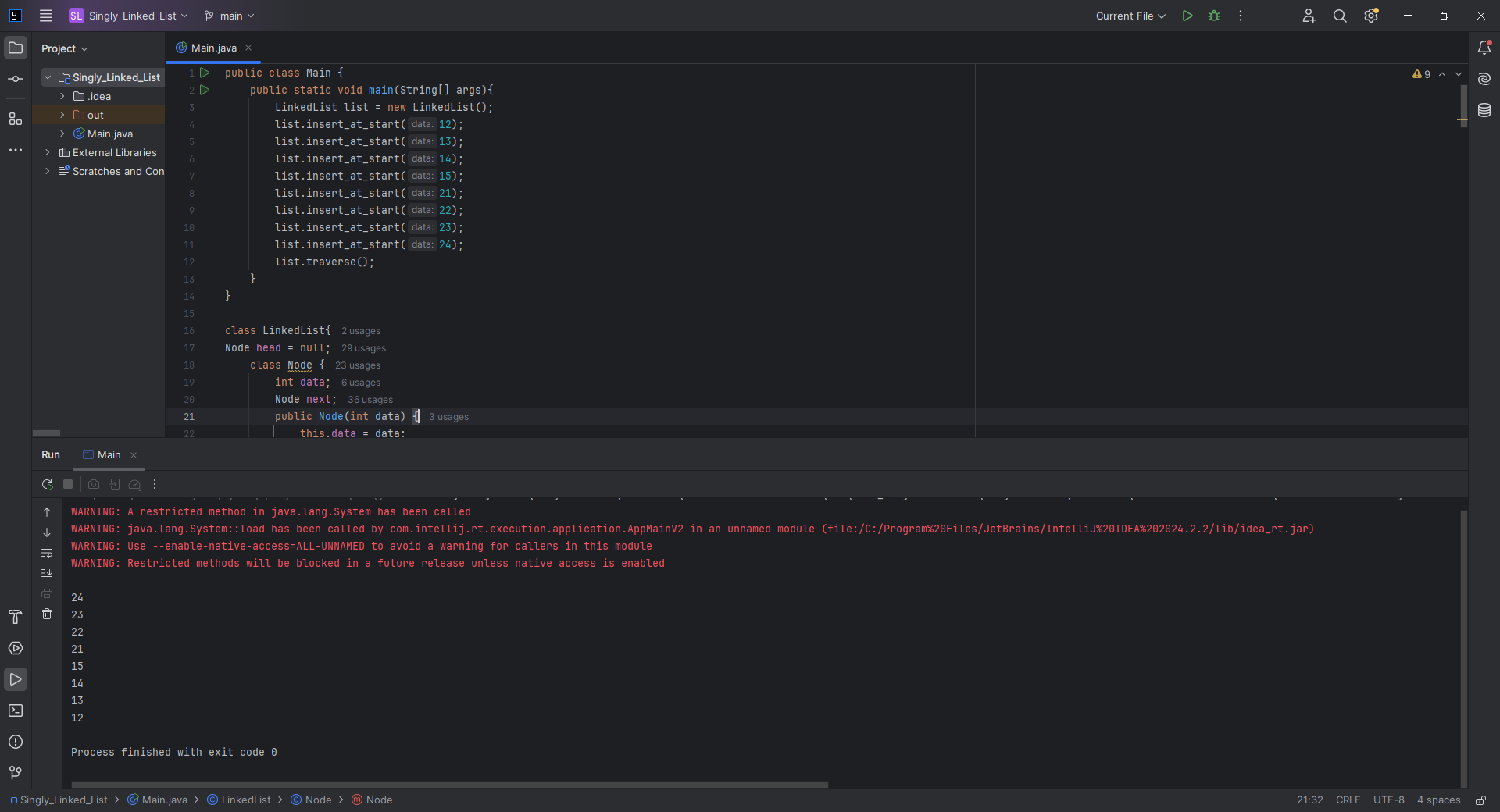
**CLASS TASKS**

**CODE:**

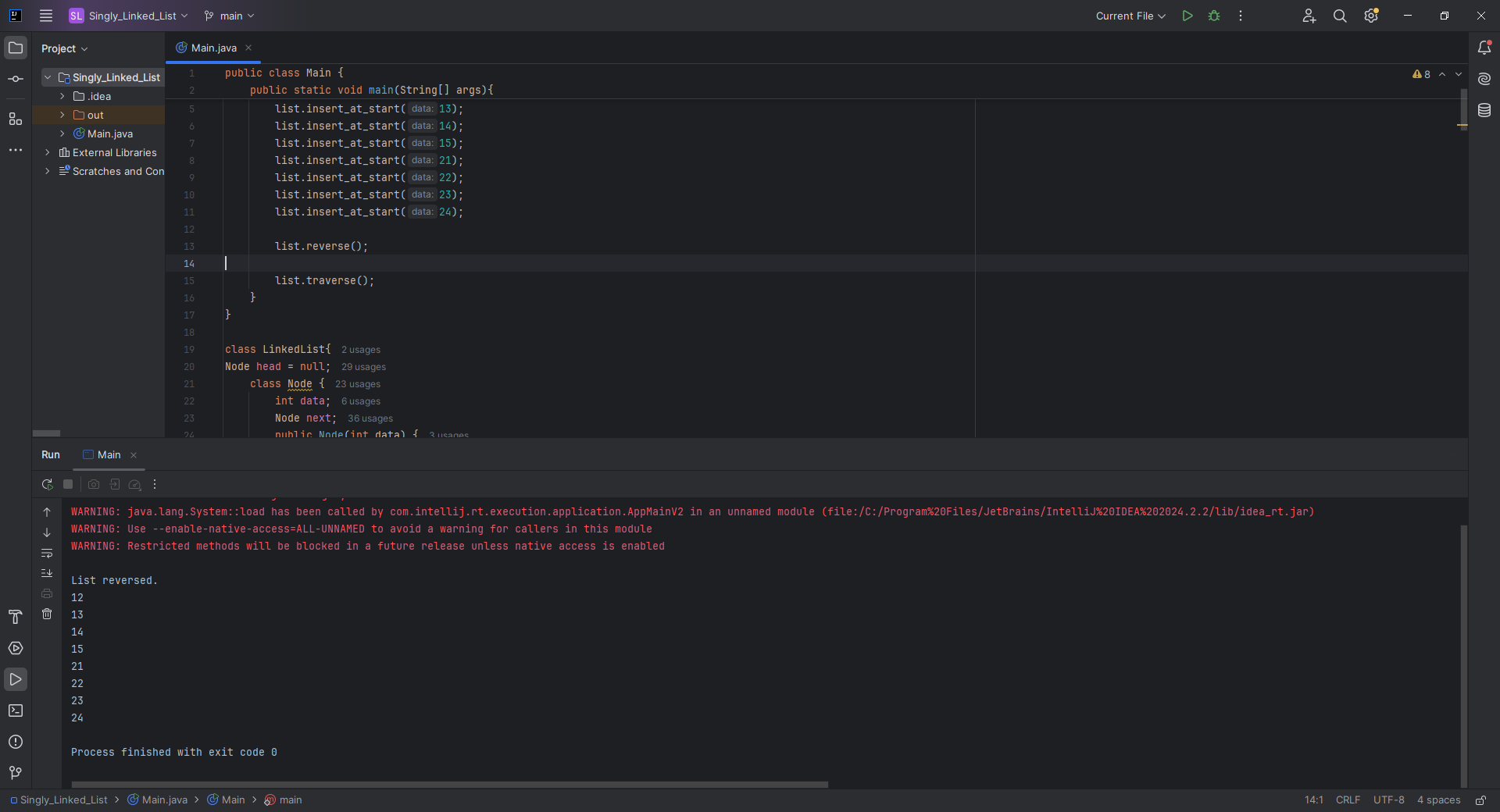
public class Main {  
 public static void main(String[] args){  
 LinkedList list = new LinkedList();  
 list.insert\_at\_start(12);  
 list.insert\_at\_start(13);  
 list.insert\_at\_start(14);  
 list.insert\_at\_start(15);  
 list.insert\_at\_start(21);  
 list.insert\_at\_start(22);  
 list.insert\_at\_start(23);  
 list.insert\_at\_start(24);  
 list.reverse();  
 list.findMiddle();  
  
 LinkedList.Node temp = list.head;  
 while (temp.next != null){  
 temp = temp.next;  
 }  
 temp.next = list.head.next.next;  
 list.detectAndRemoveCycle();  
 list.traverse();  
 }  
}  
  
class LinkedList{  
Node head = null;  
 class Node {  
 int data;  
 Node next;  
 public Node(int data) {  
 this.data = data;  
 next = null;  
 }  
 }  
 void delete\_from\_start(){  
 if(head == null){  
 return;  
 }  
 head = head.next;  
 }  
 void delete\_from\_end(){  
 if(head == null){  
 return;  
 }  
 if (head.next == null) {  
 head = null;  
 return;  
 }  
 Node current = head;  
 while (current.next.next != null) {  
 current = current.next;  
 }  
 current.next = null;  
 }  
 void insert\_after(int val , int data){  
 Node newNode = new Node(data);  
 if(head == null){  
 System.*out*.println("List is empty");  
 return;  
 }  
 if(head.data == val){  
 newNode.next = head.next;  
 head.next = newNode;  
 return;  
 }  
 Node current = head.next;  
 while(current.next != null){  
 if(current.data == val){  
 newNode.next = current.next;  
 current.next = newNode;  
 return;  
 }  
 }  
 System.*out*.println(val + " is not in the list");  
 return;  
 }  
 void insert\_at\_start(int data){  
 Node newNode = new Node(data);  
 if(head == null){  
 head = newNode;  
 return;  
 }  
 newNode.next = head;  
 head = newNode;  
 }  
 void insert\_at\_end(int data) {  
 Node newNode = new Node(data);  
 if(head == null){  
 head = newNode;  
 return;  
 }  
 Node current = head;  
 while(current.next != null){  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 void traverse(){  
 if(head == null){  
 System.*out*.println("Empty List");  
 return;  
 }  
 Node current = head;  
 System.*out*.println(current.data);  
 while(current.next != null){  
 current = current.next;  
 System.*out*.println(current.data);  
 }  
 }  
}



**Home Task 1:** Reverse the Linked List

**CODE:**

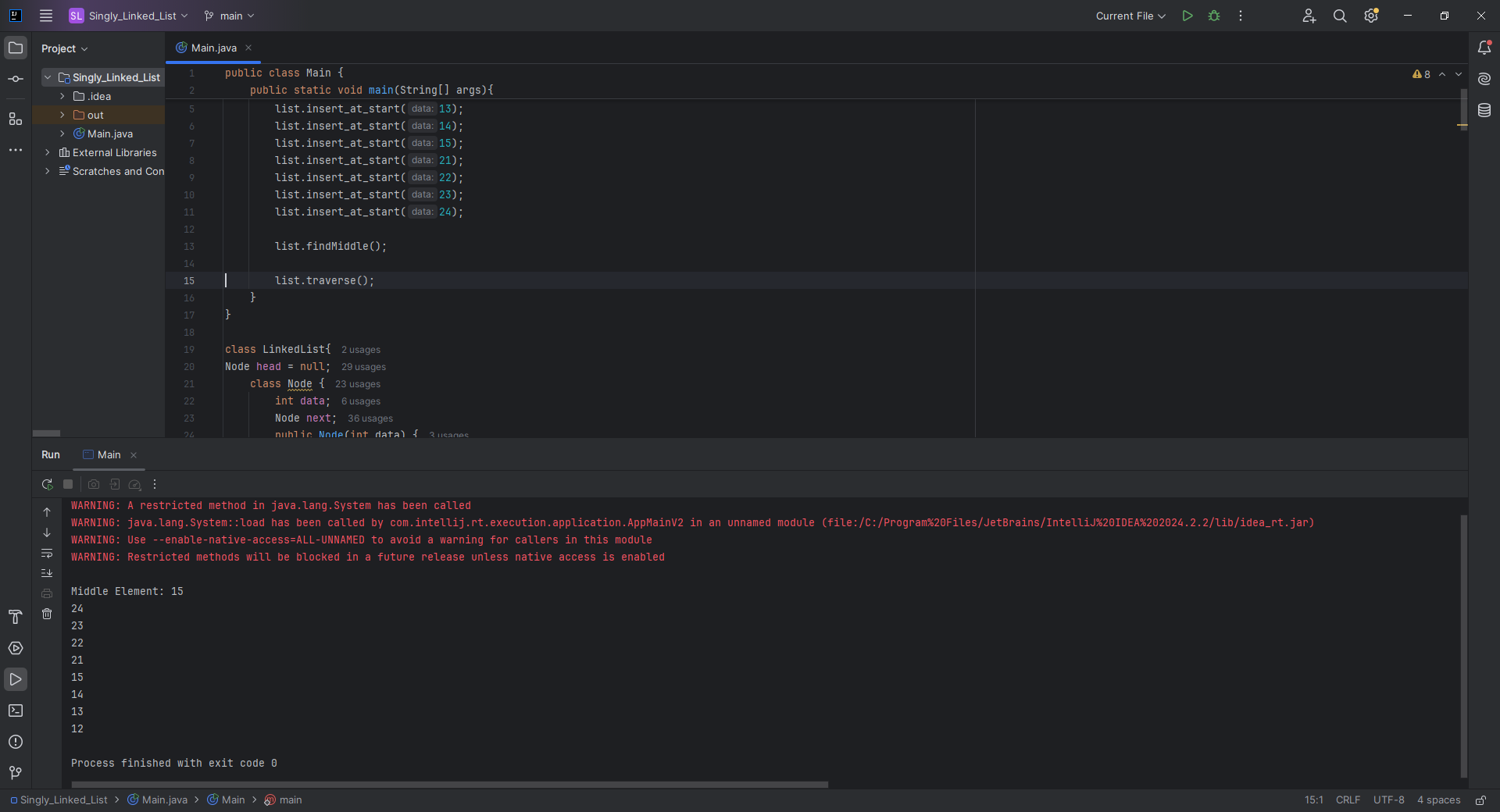
void reverse() {  
 Node prev = null;  
 Node curr = head;  
 while (curr != null) {  
 Node next = curr.next;  
 curr.next = prev;  
 prev = curr;  
 curr = next;  
 }  
 head = prev;  
 System.*out*.println("List reversed.");  
}



**Home Task 2:** Find the Middle Element of the Linked List

**CODE:**

void findMiddle() {  
 if (head == null) {  
 System.*out*.println("List is empty.");  
 return;  
 }  
 Node slow = head;  
 Node fast = head;  
 while (fast != null && fast.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
 }  
 System.*out*.println("Middle Element: " + slow.data);  
}



**Home Task 3:** Detect and Remove Cycles in the Linked List

**CODE:**

void detectAndRemoveCycle() {  
 Node slow = head;  
 Node fast = head;  
  
 while (fast != null && fast.next != null) {  
 slow = slow.next;  
 fast = fast.next.next;  
 if (slow == fast) {  
 removeCycle(slow);  
 System.*out*.println("Cycle detected and removed.");  
 return;  
 }  
 }  
 System.*out*.println("No cycle detected.");  
}  
  
private void removeCycle(Node loopNode) {  
 Node ptr1 = head;  
 Node ptr2 = loopNode;  
  
 while (ptr1 != ptr2) {  
 ptr1 = ptr1.next;  
 ptr2 = ptr2.next;  
 }  
  
 Node prev = ptr2;  
 while (prev.next != ptr2) {  
 prev = prev.next;  
 }  
 prev.next = null; // Remove cycle  
}

