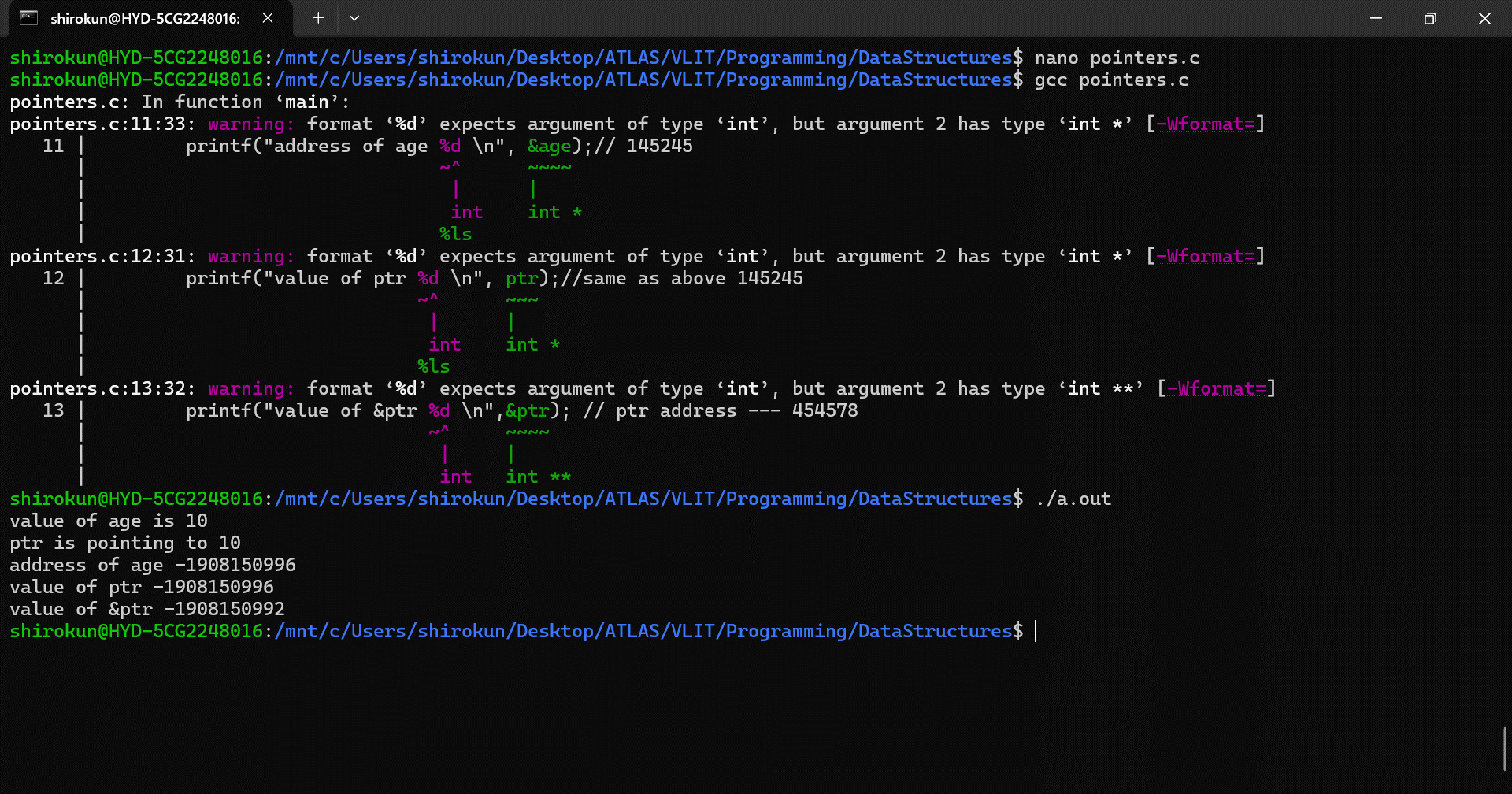
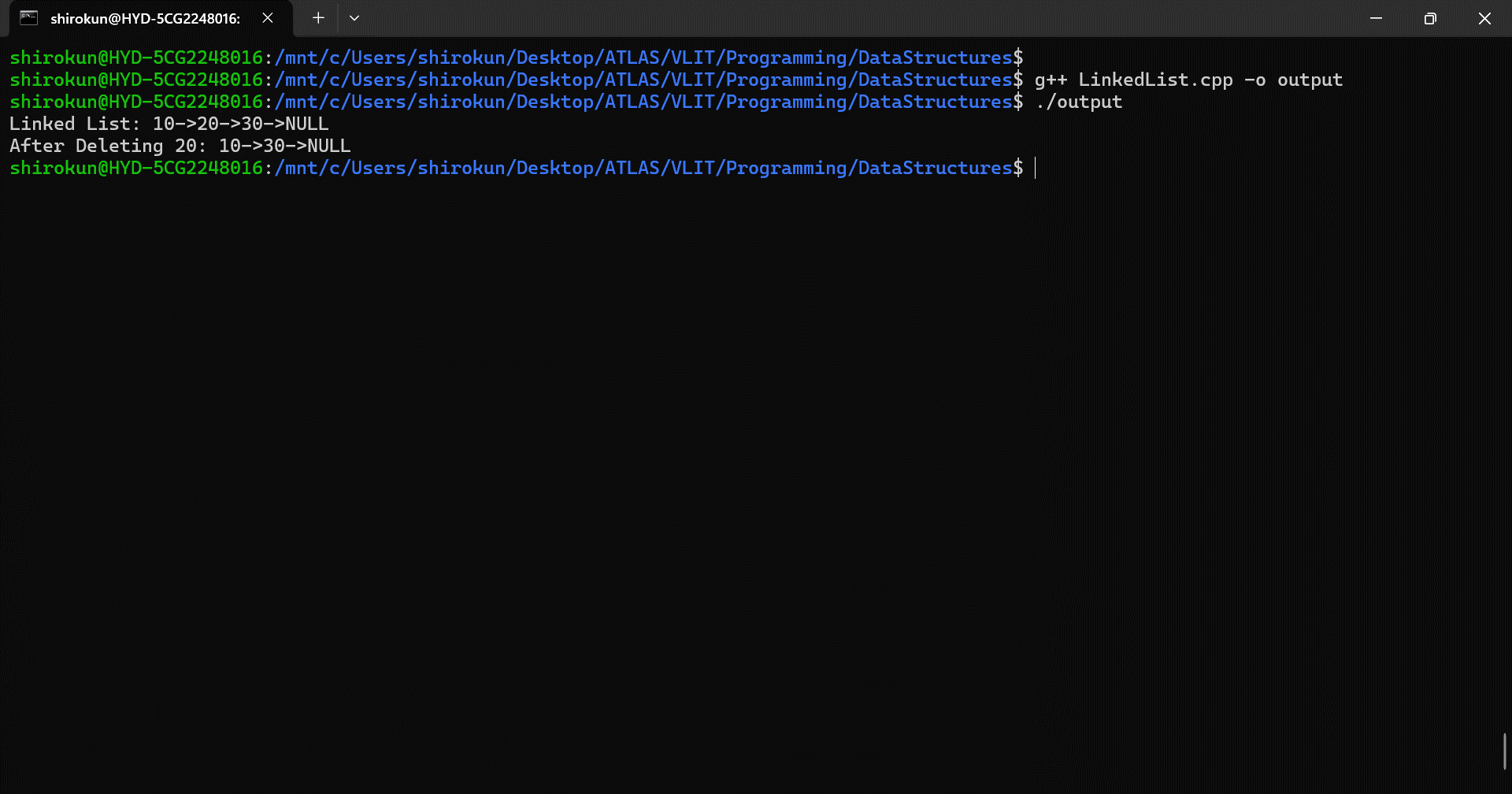
Day 13 – 28/06/2025

// Task001: Pointers.  
  
#include<stdio.h>  
void main(){  
 int age;  
 int \*ptr;  
 age = 10;  
 ptr = &age;  
 printf("value of age is %d\n", age);   
 printf("ptr is pointing to %d\n", \*ptr);  
 printf("address of age %d \n", &age);   
 printf("value of ptr %d \n", ptr);  
 printf("value of &ptr %d \n",&ptr);  
  
}

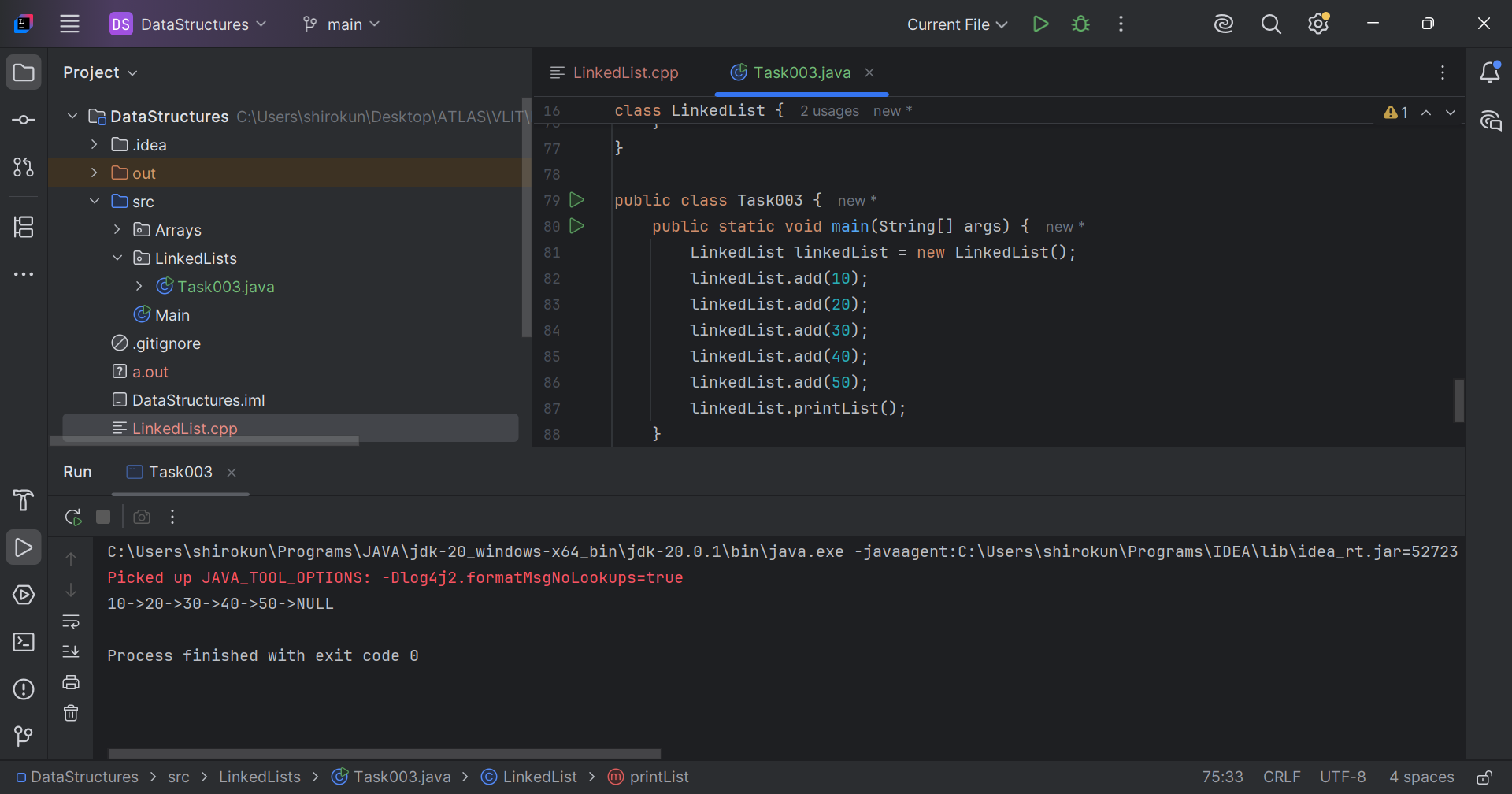


// Task002: LinkedList  
  
#include <bits/stdc++.h>  
using namespace std;  
  
// Define a Node class  
class Node{  
 public:  
 int data; // Data part of the node  
 Node\* next; // Pointer to the next node  
  
 // Constructor for convenience  
 Node(int value) : data(value), next(nullptr) {}  
};  
  
// Class for singly linked list  
class Linkedlist{  
private:  
 Node\* head; // Pointer to the head of the list  
  
public:  
 // Constructor to initialize an empty list  
 Linkedlist(){  
 head = nullptr;  
 }  
 // Function to insert a node at the end  
 void insertAtEnd(int value){  
 Node\* newNode = new Node(value);  
 if(head == nullptr){  
 head = newNode; // If list is empty, make newNode the head  
 }  
 else{  
 Node\* temp = head;  
 while (temp->next != nullptr){  
 temp = temp->next; // Traverse to the last node  
 }  
 temp->next = newNode; // Link the last node to newNode  
 }  
 }  
  
 // Function to delete a Node by Value  
 void deleteByValue(int value){  
 if(head == nullptr){  
 return;  
 }  
 if(head->data == value){  
 Node\* temp = head;  
 head = head->next; // Move head to the next node  
 delete temp; // Free memory of the deleted node  
 return;  
 }  
 Node\* temp = head;  
 while(temp->next && temp->next->data != value){  
 temp = temp->next; // Traverse to find the node to delete  
 }  
 if(temp->next){  
 Node\* nodeToDelete = temp->next;  
 temp->next = temp->next->next; // Unlink the node  
 delete nodeToDelete; //Free Memory  
 }  
 }  
 // Function to display the list  
 void display(){  
 Node\* temp = head;  
 while(temp != nullptr){  
 cout << temp->data << "->";  
 temp = temp->next;  
 }  
 cout << "NULL" <<endl;  
 }

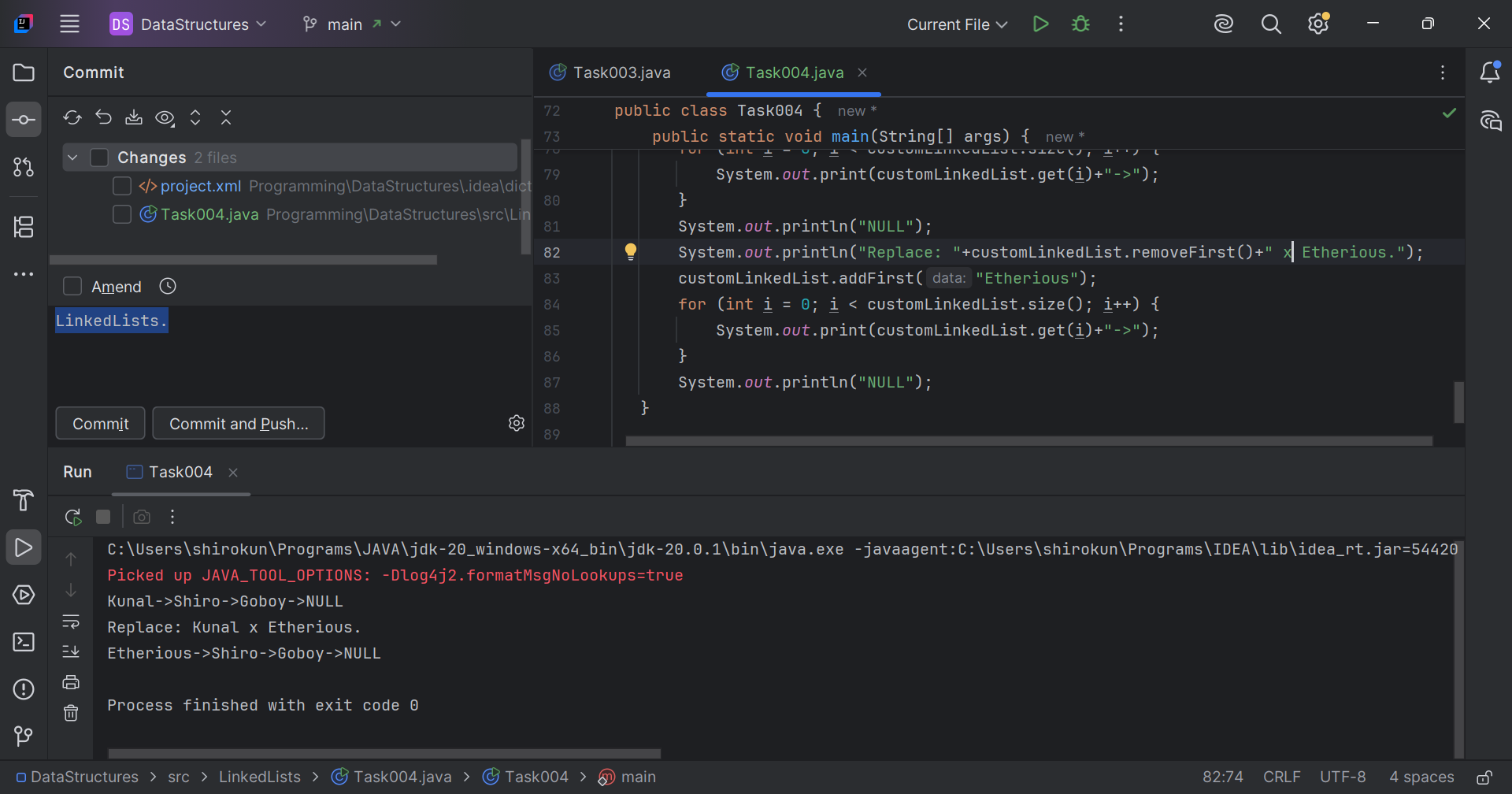
// Destructor to free all allocated memory  
 ~Linkedlist() {  
 Node\* temp;  
 while (head) {  
 temp = head;  
 head = head->next;  
 delete temp;  
 }  
 }  
};  
int main() {  
 Linkedlist list;  
  
 list.insertAtEnd(10);  
 list.insertAtEnd(20);  
 list.insertAtEnd(30);  
  
 cout << "Linked List: ";  
 list.display();  
  
 list.deleteByValue(20);  
  
 cout << "After Deleting 20: ";  
 list.display();  
  
 return 0;  
}



// Task003: Use the above code to create a  
// Java code which creates a linked list.  
  
package LinkedLists;  
  
class Node {  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
class LinkedList {  
 Node head;  
 public LinkedList() {  
 head = null;  
 }  
  
 public void add(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 }  
  
 public void deleteAtPosition(int position) {  
 if (head == null) {  
 System.*out*.println("List is empty");  
 return;  
 }  
  
 if (position < 0) {  
 System.*out*.println("Invalid position");  
 return;  
 }  
  
 if (position == 0) {  
 head = head.next;  
 return;  
 }  
  
 Node current = head;  
 int currentIndex = 0;  
  
 while (current != null && currentIndex < position - 1) {  
 current = current.next;  
 currentIndex++;  
 }  
  
 if (current == null || current.next == null) {  
 System.*out*.println("Position out of bounds");  
 return;  
 }  
  
 current.next = current.next.next;  
 }  
  
  
 public void printList() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + "->");  
 current = current.next;  
 }  
 System.*out*.println("NULL");  
 }  
}  
  
public class Task003 {  
 public static void main(String[] args) {  
 LinkedList linkedList = new LinkedList();  
 linkedList.add(10);  
 linkedList.add(20);  
 linkedList.add(30);  
 linkedList.add(40);  
 linkedList.add(50);  
 linkedList.printList();  
 }  
}



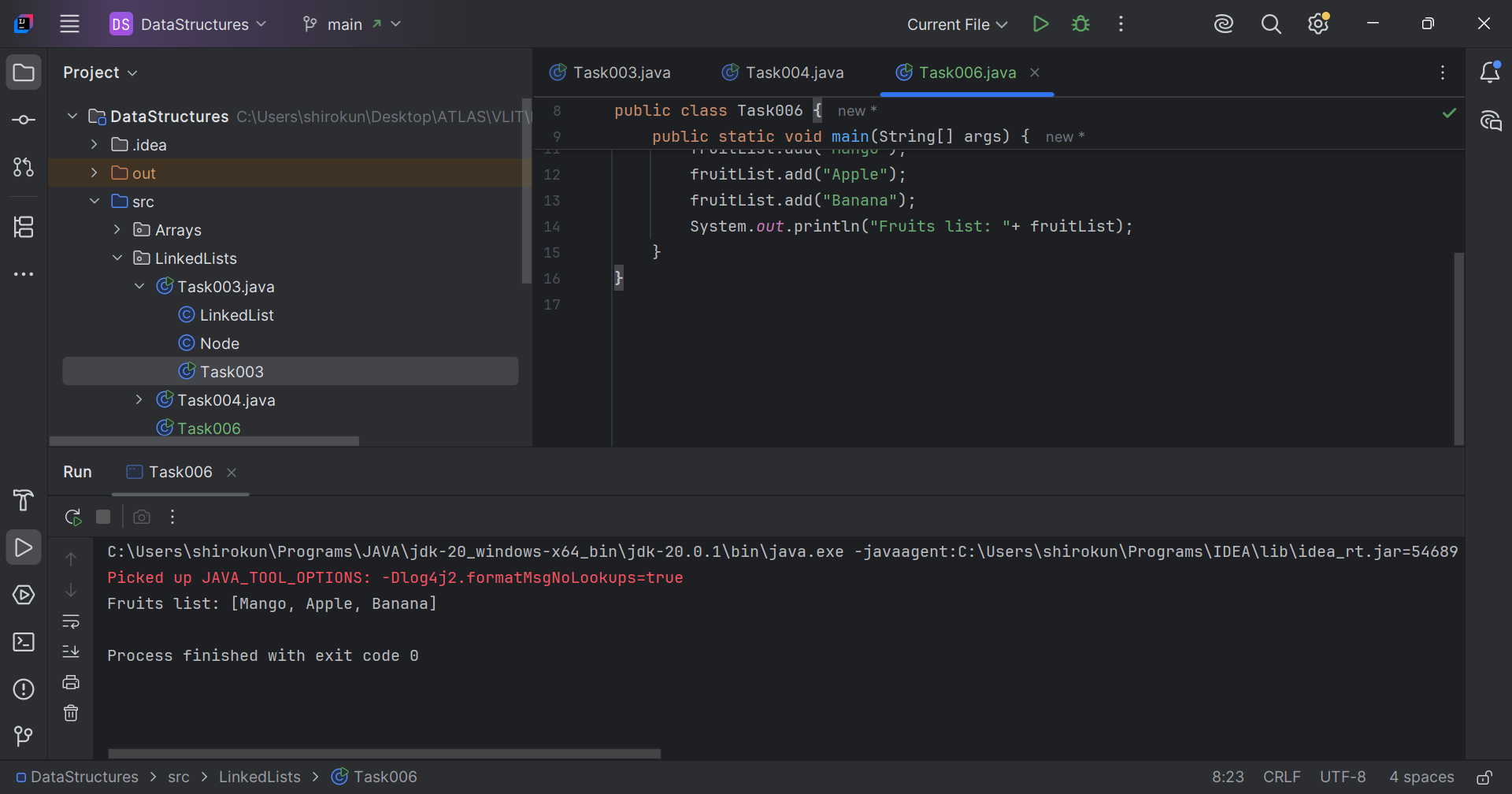
// Task004: Try to create a node and add a value to it.  
// Which can take any kind of data in the Node.  
  
package LinkedLists;  
  
import java.util.NoSuchElementException;  
  
class GNode<T> {  
 T data;  
 GNode<T> next;  
  
 public GNode(T data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
class CustomLinkedList<T> {  
 private GNode<T> head;  
 private int size = 0;  
  
 public void add(T data) {  
 GNode<T> newNode = new GNode<>(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 GNode<T> current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 size++;  
 }  
  
 public void addFirst(T data) {  
 GNode<T> newNode = new GNode<>(data);  
 newNode.next = head;  
 head = newNode;  
 size++;  
 }  
  
 public T removeFirst() {  
 if (head == null) {  
 throw new NoSuchElementException("List is empty");  
 }  
 T removedData = head.data;  
 head = head.next;  
 size--;  
 return removedData;  
 }  
  
 public T get(int index) {  
 checkBounds(index);  
 GNode<T> current = head;  
 for (int i = 0; i < index; i++) {  
 current = current.next;  
 }  
 return current.data;  
 }  
  
 public int size() {  
 return size;  
 }  
  
 private void checkBounds(int index) {  
 if (index < 0 || index >= size) {  
 throw new IndexOutOfBoundsException("Index out of bounds");  
 }  
 }  
}  
  
public class Task004 {  
 public static void main(String[] args) {  
 CustomLinkedList<String> customLinkedList = new CustomLinkedList<>();  
 customLinkedList.add("Kunal");  
 customLinkedList.add("Shiro");  
 customLinkedList.add("Goboy");  
 for (int i = 0; i < customLinkedList.size(); i++) {  
 System.*out*.print(customLinkedList.get(i)+"->");  
 }  
 System.*out*.println("NULL");  
 System.*out*.println("Replace: "+customLinkedList.removeFirst()+" x Etherious.");  
 customLinkedList.addFirst("Etherious");  
 for (int i = 0; i < customLinkedList.size(); i++) {  
 System.*out*.print(customLinkedList.get(i)+"->");  
 }  
 System.*out*.println("NULL");  
 }  
}



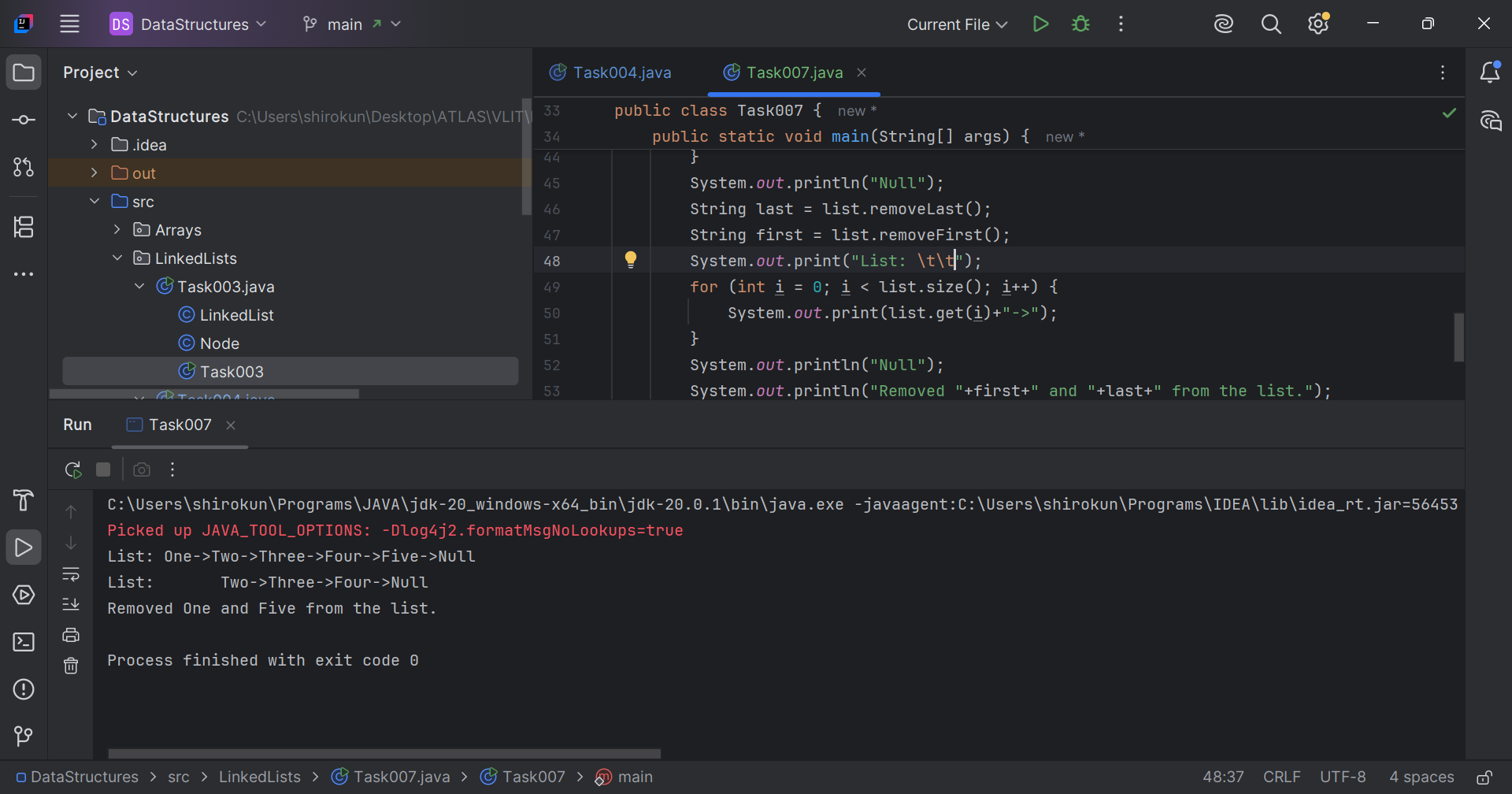
Q5. List down all the methods of Linked list.  
Ans.

* Basic Operations: add(T data), delete(T data), printList()
* Insertion: addAtBeginning(T data), addAtPosition(T data, int position)
* Deletion: deleteAtPosition(int position), deleteHead(), deleteTail()
* Traversal: get(int position)
* Utility: isEmpty(), size(), contains(T data), indexOf(T data)
* Modification: update(T oldData, T newData), updateAtPosition(int position, T newData)

// Task006: Create linked list using Pre  
// defined class and add elements to it.  
  
package LinkedLists;  
  
import java.util.LinkedList;  
  
public class Task006 {  
 public static void main(String[] args) {  
 LinkedList<String> fruitList = new LinkedList<>();  
 fruitList.add("Mango");  
 fruitList.add("Apple");  
 fruitList.add("Banana");  
 System.*out*.println("Fruits list: "+ fruitList);  
 }  
}

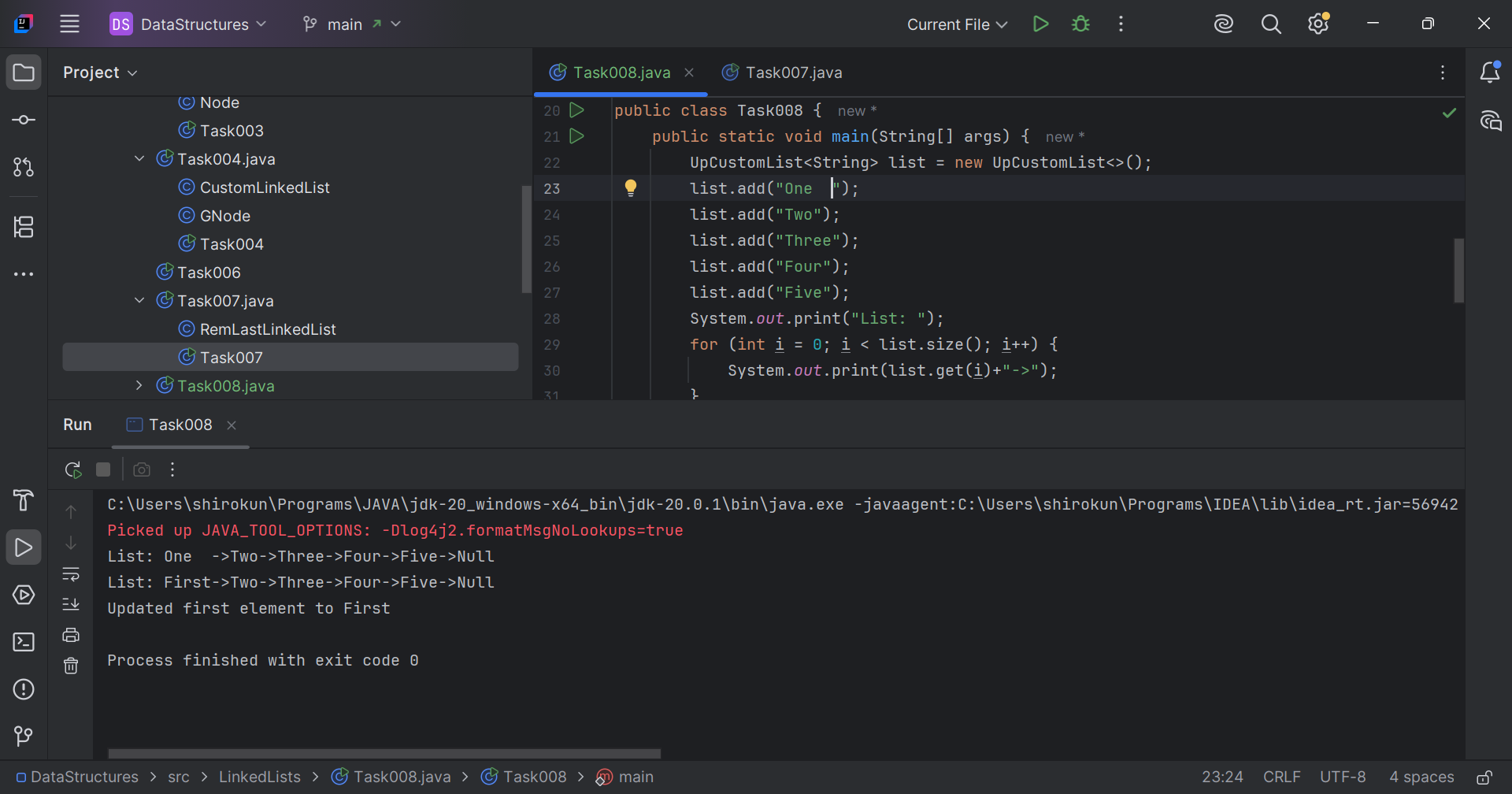


// Task007: remove first and remove last element  
// and display all elements in the linked list.  
  
package LinkedLists;  
  
import java.util.NoSuchElementException;  
  
class RemLastLinkedList<T> extends CustomLinkedList<T>{  
 public T removeLast() {  
 if (head == null) {  
 throw new NoSuchElementException("List is empty");  
 }  
  
 if (head.next == null) {  
 T removedData = head.data;  
 head = null;  
 size--;  
 return removedData;  
 }  
  
 GNode<T> current = head;  
 while (current.next.next != null) {  
 current = current.next;  
 }  
  
 T removedData = current.next.data;  
 current.next = null;  
 size--;  
 return removedData;  
 }  
}  
  
public class Task007 {  
 public static void main(String[] args) {  
 RemLastLinkedList<String> list = new RemLastLinkedList<>();  
 list.add("One");  
 list.add("Two");  
 list.add("Three");  
 list.add("Four");  
 list.add("Five");  
 System.*out*.print("List: ");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i)+"->");  
 }  
 System.*out*.println("Null");  
 String last = list.removeLast();  
 String first = list.removeFirst();  
 System.*out*.print("List: \t\t");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i)+"->");  
 }  
 System.*out*.println("Null");  
 System.*out*.println("Removed "+first+" and "+last+" from the list.");  
 }  
}

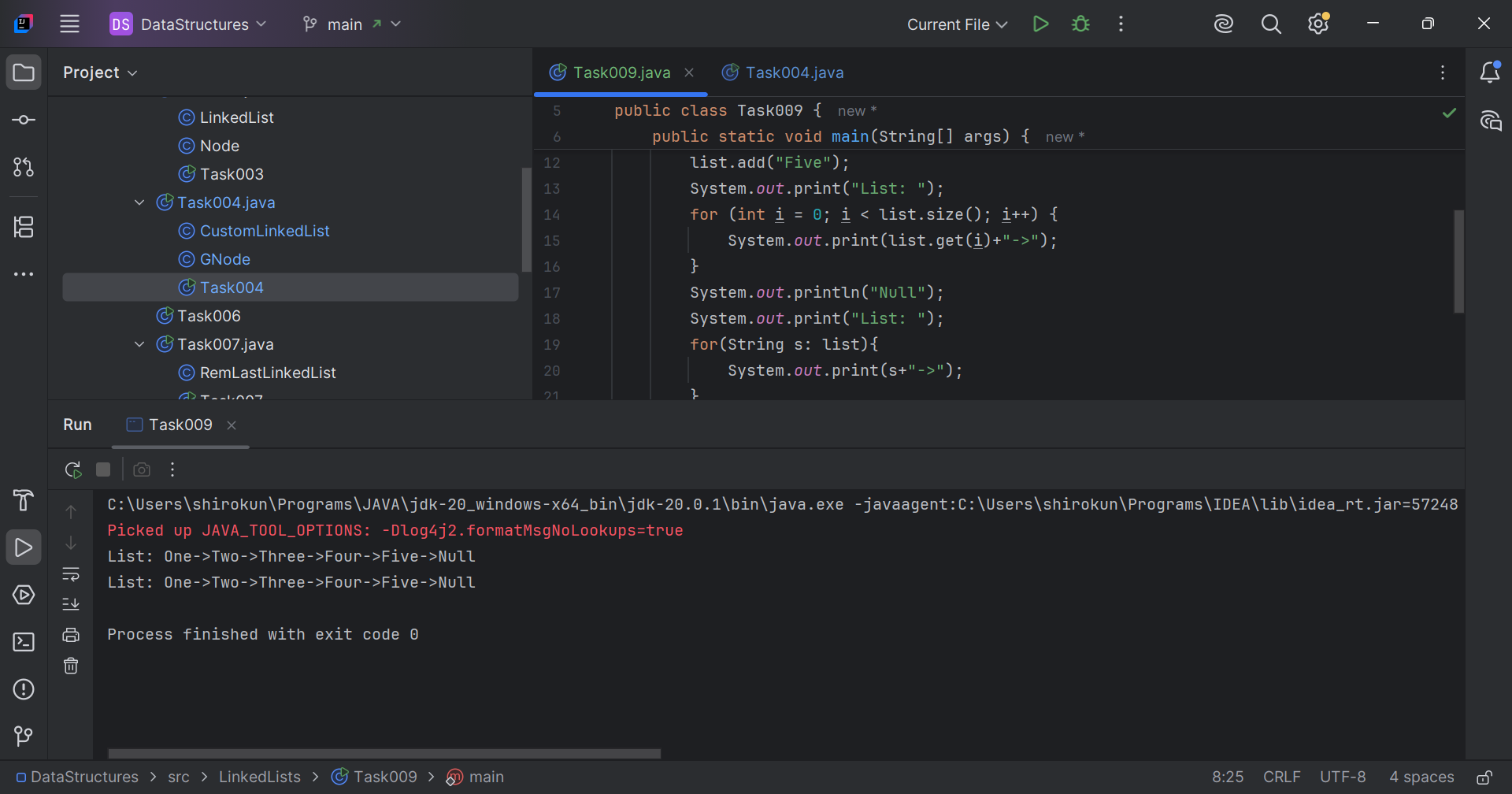


// Task008: update the 1st element to a new value

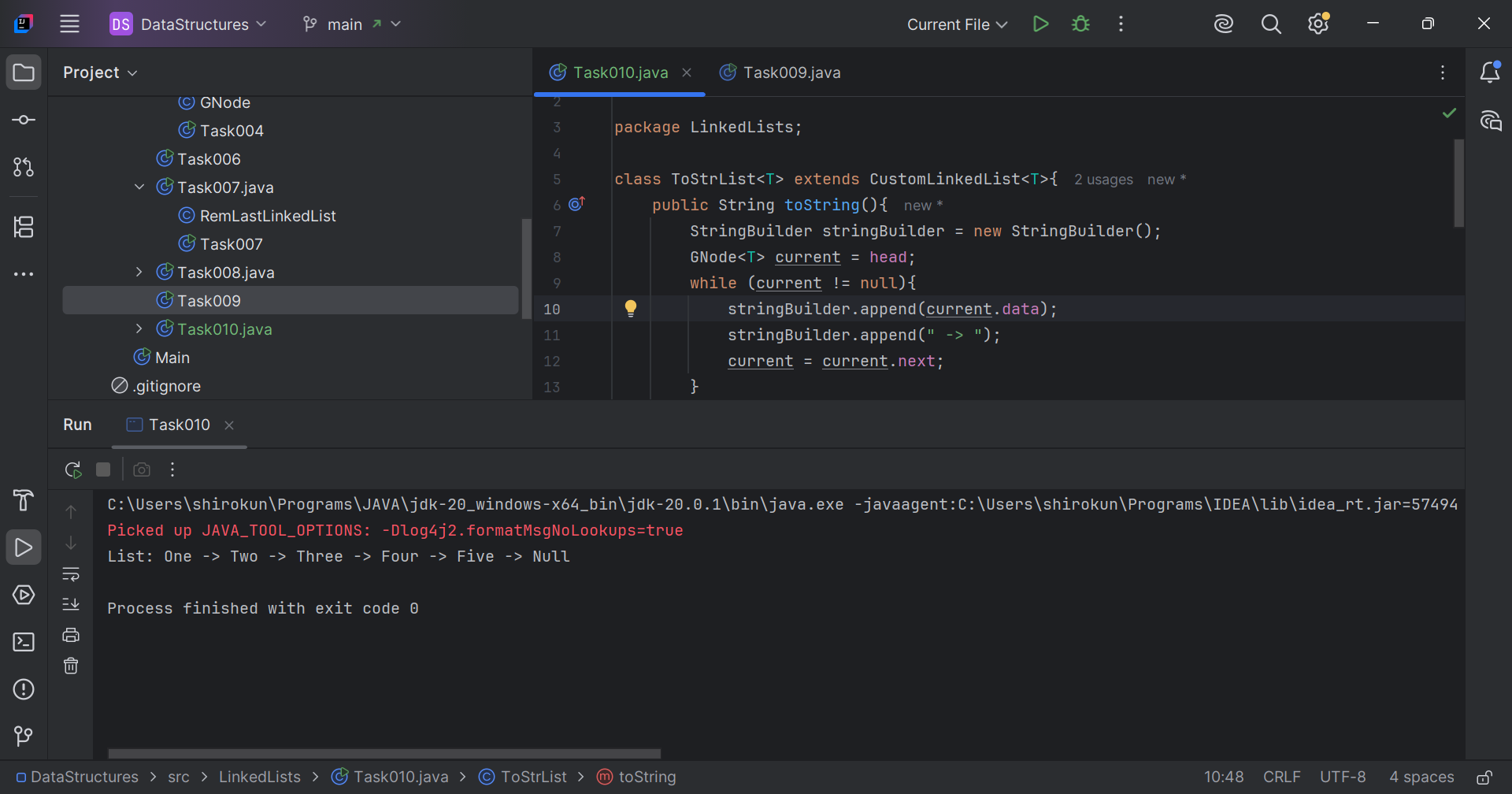
package LinkedLists;  
import java.util.NoSuchElementException;  
  
class UpCustomList<T> extends RemLastLinkedList<T>{  
 public void replace(int position, T value){  
 if (head == null) {  
 throw new NoSuchElementException("List is empty");  
 }  
 GNode<T> current = head;  
 for (int j = 1; j < position; j++) {  
 current = current.next;  
 }  
 current.data = value;  
 }  
}  
public class Task008 {  
 public static void main(String[] args) {  
 UpCustomList<String> list = new UpCustomList<>();  
 list.add("One ");  
 list.add("Two");  
 list.add("Three");  
 list.add("Four");  
 list.add("Five");  
 System.*out*.print("List: ");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i)+"->");  
 }  
 System.*out*.println("Null");  
 list.replace(1, "First");  
 System.*out*.print("List: ");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i)+"->");  
 }  
 System.*out*.println("Null");  
 System.*out*.println("Updated first element to "+list.get(0));  
 }  
}



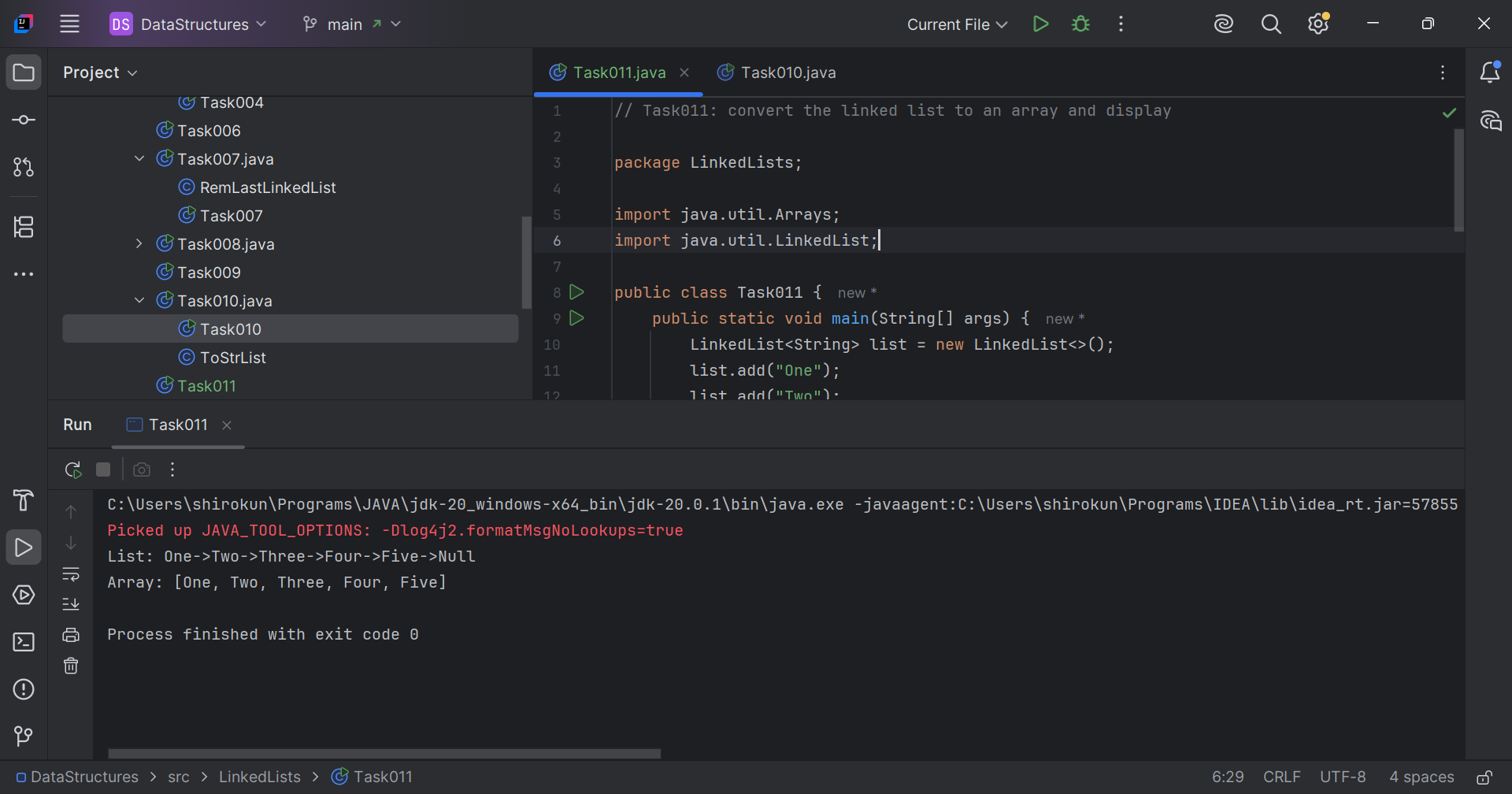
// Task009: Display list with for and foreach loops  
  
package LinkedLists;  
  
public class Task009 {  
 public static void main(String[] args) {  
 CustomLinkedList<String> list = new CustomLinkedList<>();  
 list.add("One");  
 list.add("Two");  
 list.add("Three");  
 list.add("Four");  
 list.add("Five");  
 System.*out*.print("List: ");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i)+"->");  
 }  
 System.*out*.println("Null");  
 System.*out*.print("List: ");  
 for(String s: list){  
 System.*out*.print(s+"->");  
 }  
 System.*out*.println("Null");  
 }  
}



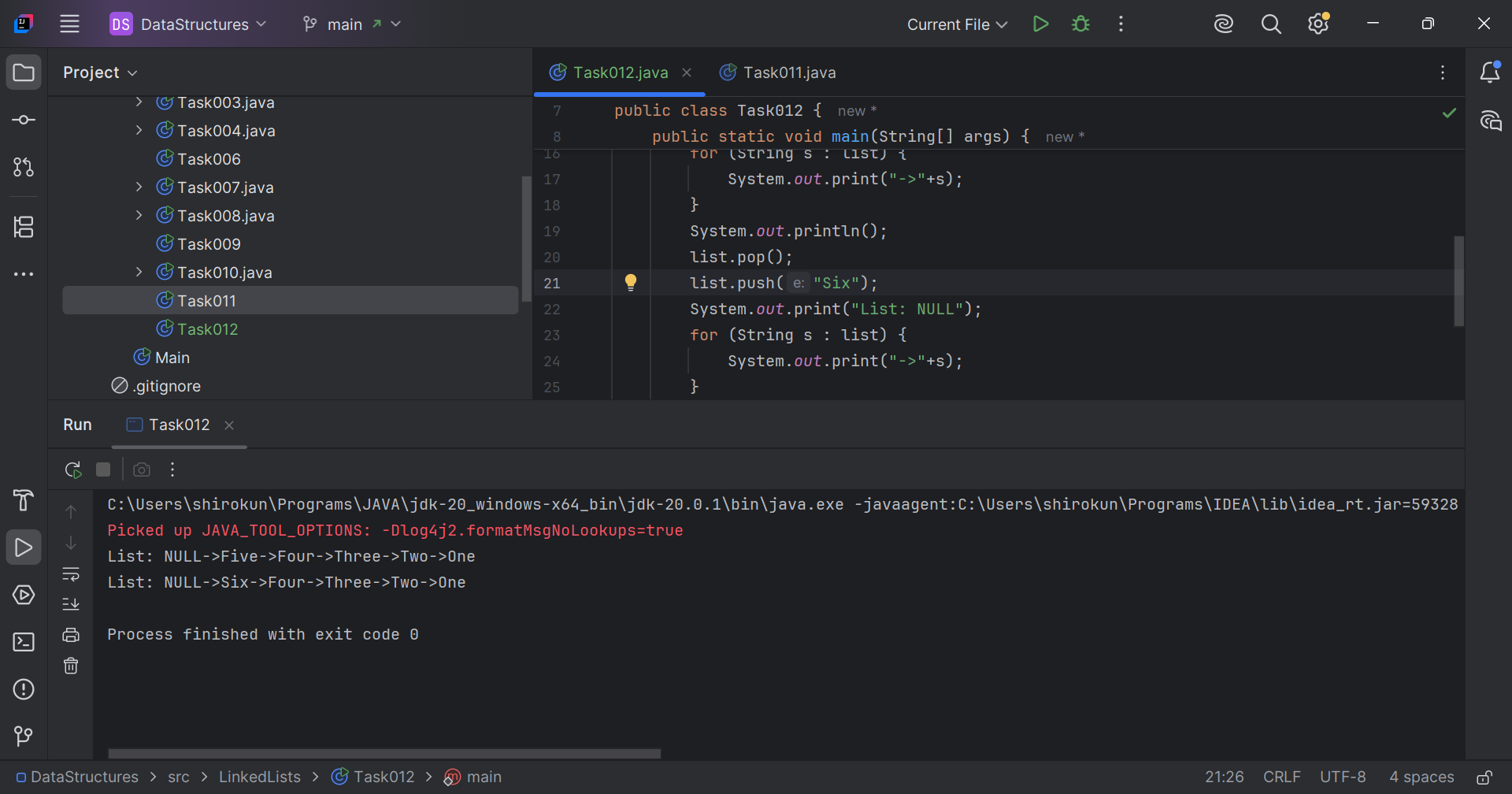
// Task010: display the elements of the linked list without loops  
  
package LinkedLists;  
  
class ToStrList<T> extends CustomLinkedList<T>{  
 public String toString(){  
 StringBuilder stringBuilder = new StringBuilder();  
 GNode<T> current = head;  
 while (current != null){  
 stringBuilder.append(current.data);  
 stringBuilder.append(" -> ");  
 current = current.next;  
 }  
 stringBuilder.append("Null");  
 return stringBuilder.toString();  
 }  
}  
  
public class Task010 {  
 public static void main(String[] args) {  
 ToStrList<String> list = new ToStrList<>();  
 list.add("One");  
 list.add("Two");  
 list.add("Three");  
 list.add("Four");  
 list.add("Five");  
 System.*out*.println("List: "+list);  
 }  
}



// Task011: convert the linked list to an array and display  
  
package LinkedLists;  
  
import java.util.Arrays;  
import java.util.LinkedList;  
  
public class Task011 {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.add("One");  
 list.add("Two");  
 list.add("Three");  
 list.add("Four");  
 list.add("Five");  
 System.*out*.print("List: ");  
 for (String s : list) {  
 System.*out*.print(s + "->");  
 }  
 System.*out*.println("Null");  
 Object[] a = list.toArray(new String[0]);  
 System.*out*.println("Array: "+ Arrays.*toString*(Arrays.*stream*(a).toArray()));  
 }  
}

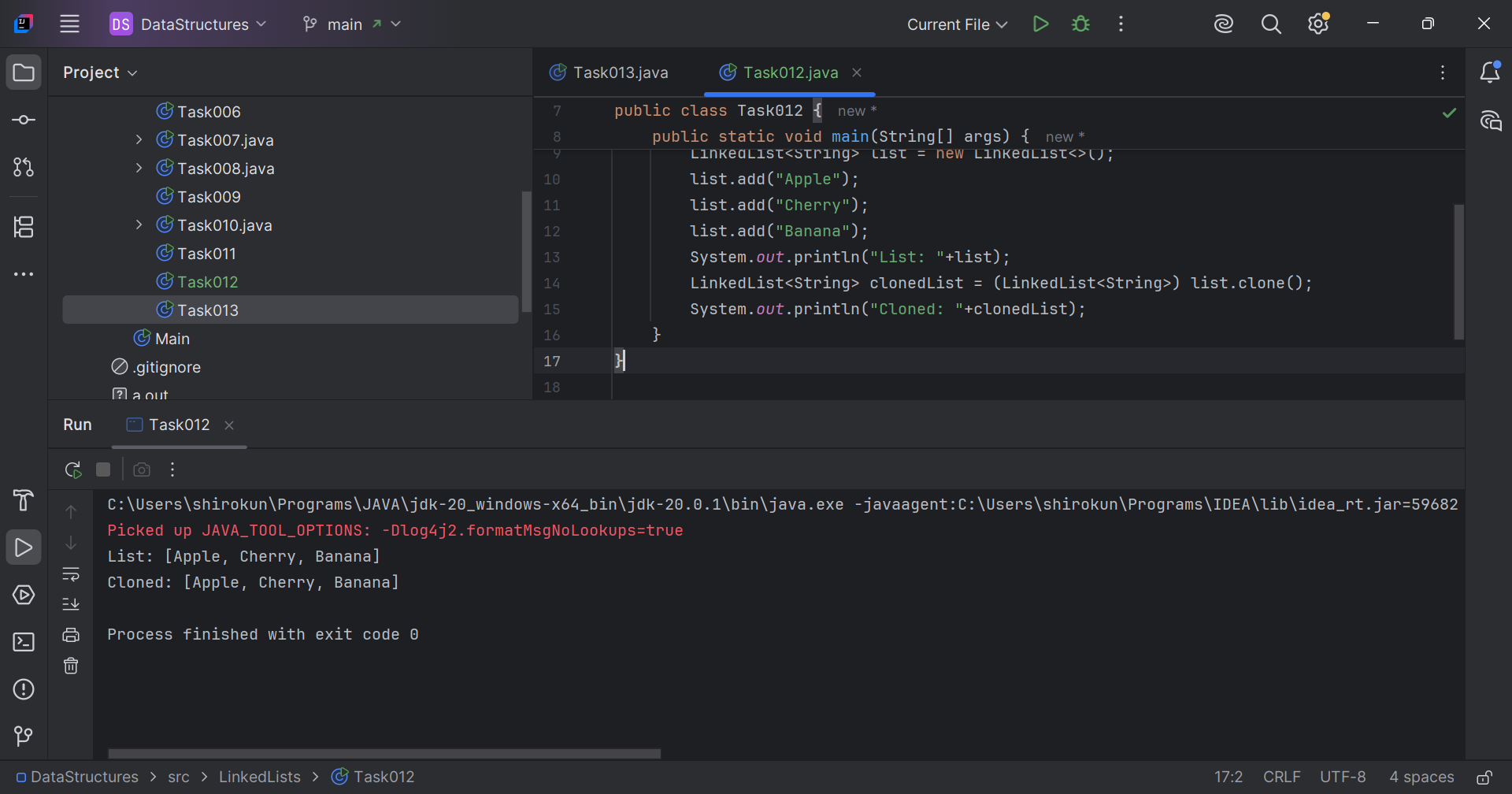


// Task012: Use pop and push methods on linked list.  
  
package LinkedLists;  
  
import java.util.LinkedList;  
  
public class Task012 {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.push("One");  
 list.push("Two");  
 list.push("Three");  
 list.push("Four");  
 list.push("Five");  
 System.*out*.print("List: NULL");  
 for (String s : list) {  
 System.*out*.print("->"+s);  
 }  
 System.*out*.println();  
 list.pop();  
 list.push("Six");  
 System.*out*.print("List: NULL");  
 for (String s : list) {  
 System.*out*.print("->"+s);  
 }  
 System.*out*.println();  
 }  
}



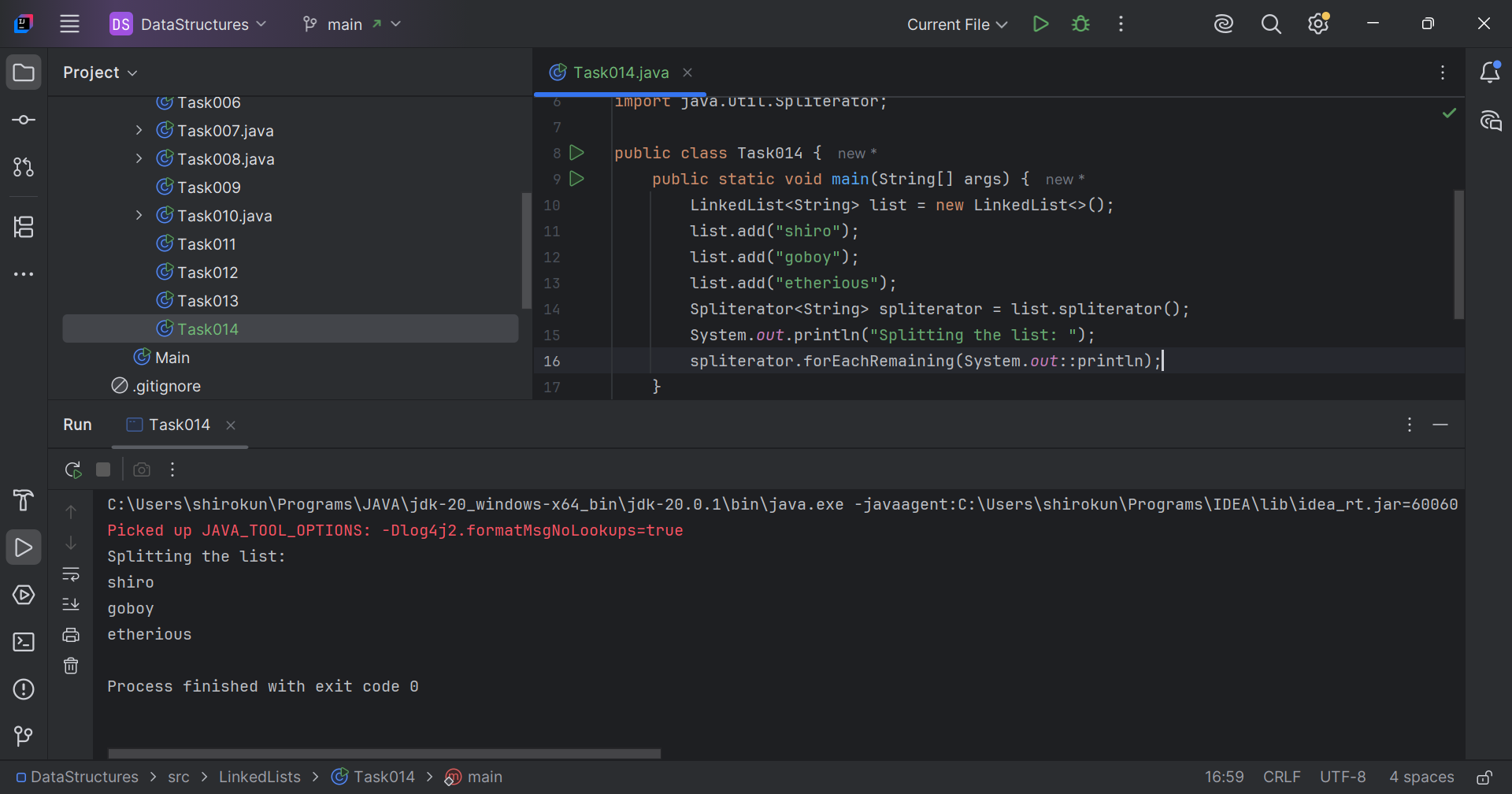
// Task012: Clone a list  
  
package LinkedLists;  
  
import java.util.LinkedList;  
  
public class Task012 {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.add("Apple");  
 list.add("Cherry");  
 list.add("Banana");

System.*out*.println("List: "+list);  
 LinkedList<String> clonedList = (LinkedList<String>) list.clone();  
 System.*out*.println("Cloned: "+clonedList);  
 }  
}



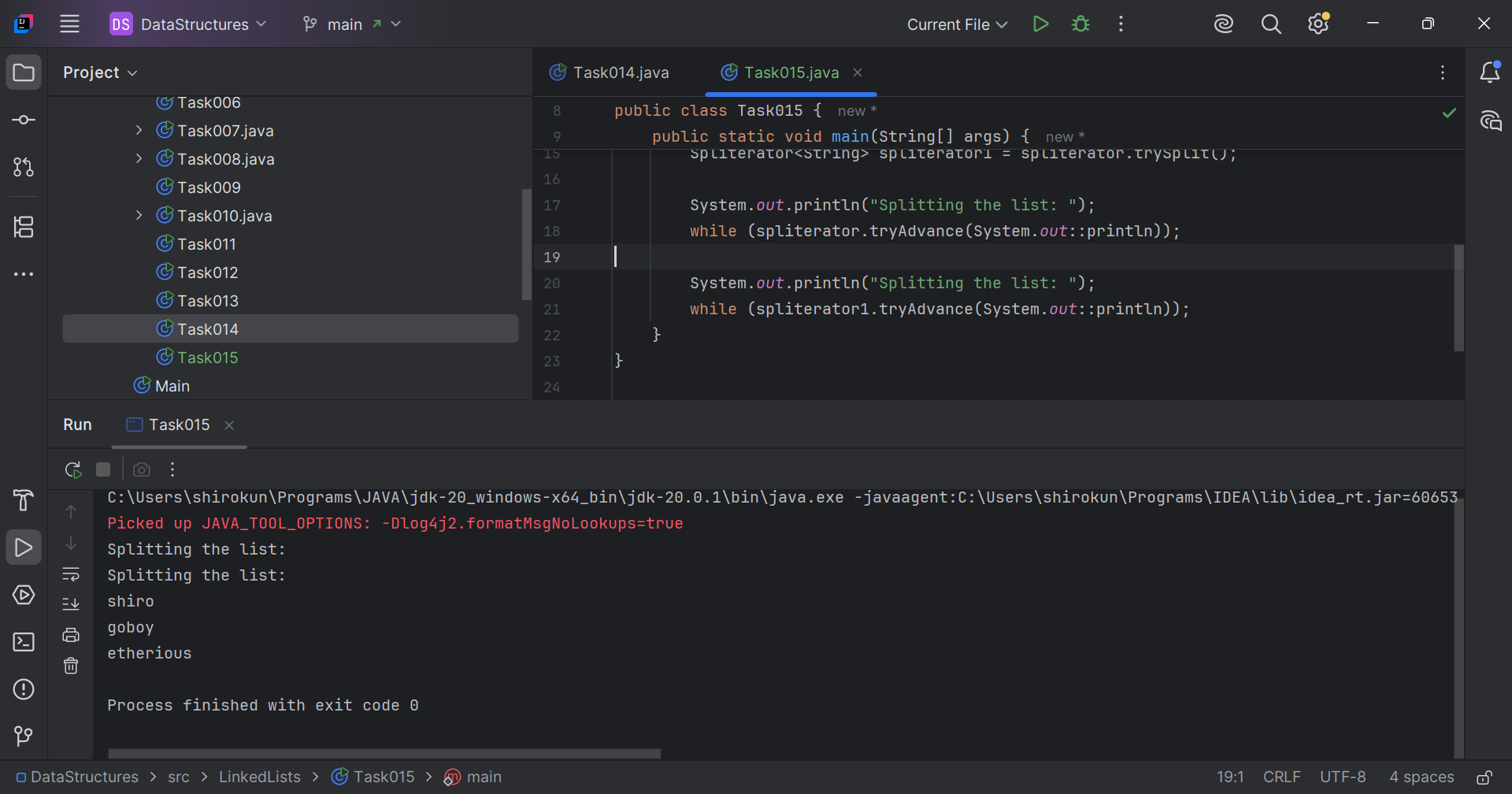
// Task014: Split-iterator.  
  
package LinkedLists;  
  
import java.util.LinkedList;  
import java.util.Spliterator;  
  
public class Task014 {  
 public static void main(String[] args) {  
 LinkedList<String> list = new LinkedList<>();  
 list.add("shiro");  
 list.add("goboy");  
 list.add("etherious");

Spliterator<String> spliterator = list.spliterator();  
 System.*out*.println("Splitting the list: ");  
 spliterator.forEachRemaining(System.*out*::println);  
 }  
}



// Task015: tryAdvance()  
  
package LinkedLists;  
  
import java.util.LinkedList;  
import java.util.Spliterator;  
  
public class Task015 {  
 public static void main(String[] args) {  
 java.util.LinkedList<String> list = new LinkedList<>();  
 list.add("shiro");  
 list.add("goboy");  
 list.add("etherious");

Spliterator<String> spliterator = list.spliterator();  
 Spliterator<String> spliterator1 = spliterator.trySplit();  
  
 System.*out*.println("Splitting the list: ");  
 while (spliterator.tryAdvance(System.*out*::println));  
  
 System.*out*.println("Splitting the list: ");  
 while (spliterator1.tryAdvance(System.*out*::println));  
 }  
}



// Task016: Create a doubly linked list.  
  
package LinkedLists;  
  
class DoublyLinkedList<T>{  
 private Node<T> head;  
 private Node<T> tail;  
 private int size;  
  
 private static class Node<T> {  
 T data;  
 Node<T> prev;  
 Node<T> next;  
  
 public Node(T data) {  
 this.data = data;  
 }  
 }  
  
 public void add(T data) {  
 Node<T> newNode = new Node<>(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 tail.next = newNode;  
 newNode.prev = tail;  
 }  
 tail = newNode;  
 size++;  
 }

public void addFirst(T data) {  
 Node<T> newNode = new Node<>(data);  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 } else {  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 size++;  
 }  
  
 public void removeFirst() {  
 if (head == null) {  
 throw new RuntimeException("List is empty");  
 }  
 if (head == tail) {  
 head = null;  
 tail = null;  
 } else {  
 head = head.next;  
 head.prev = null;  
 }  
 size--;  
 }  
  
 public void removeLast() {  
 if (tail == null) {  
 throw new RuntimeException("List is empty");  
 }  
 if (head == tail) {  
 head = null;  
 tail = null;  
 } else {  
 tail = tail.prev;  
 tail.next = null;  
 }  
 size--;  
 }  
  
 public void printForward() {  
 Node<T> current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
  
 public int getSize(){  
 return size;  
 }

public void printBackward() {  
 Node<T> current = tail;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.prev;  
 }  
 System.*out*.println();  
 }  
}  
  
public class Task016 {  
 public static void main(String[] args) {  
 DoublyLinkedList<String> doublyLinkedList = new DoublyLinkedList<>();  
 int size = doublyLinkedList.getSize();  
 doublyLinkedList.add("one");  
 doublyLinkedList.add("two");  
 doublyLinkedList.add("three");  
 doublyLinkedList.printBackward();  
 doublyLinkedList.removeFirst();  
 doublyLinkedList.addFirst("four");  
 doublyLinkedList.removeLast();  
 doublyLinkedList.add("five");  
 doublyLinkedList.printForward();  
 System.*out*.println("Size of the list: "+size);  
 }  
}

