LAB 6

PRACTICE

Example 6.1:Program demonstrates the working of user defined Generic classes We use < > to specify Parameter type.

CODE:

**package** PRACTICE;

**public** **class** EX6\_1<T> {

//An object of type T is declared

T obj;

//contructor

EX6\_1(T obj){

**this**.obj = obj;

}

**public** T getObject() {

**return** **this**.obj;

}

}

**package** PRACTICE;

**public** **class** Main6\_1 {

**public** **static** **void** main(String[] args) {

// instance of Integer type

//Instance of integer type

EX6\_1 <Integer> obj = **new** EX6\_1<Integer>(27);

System.***out***.println(obj.getObject());

//Instance of string type

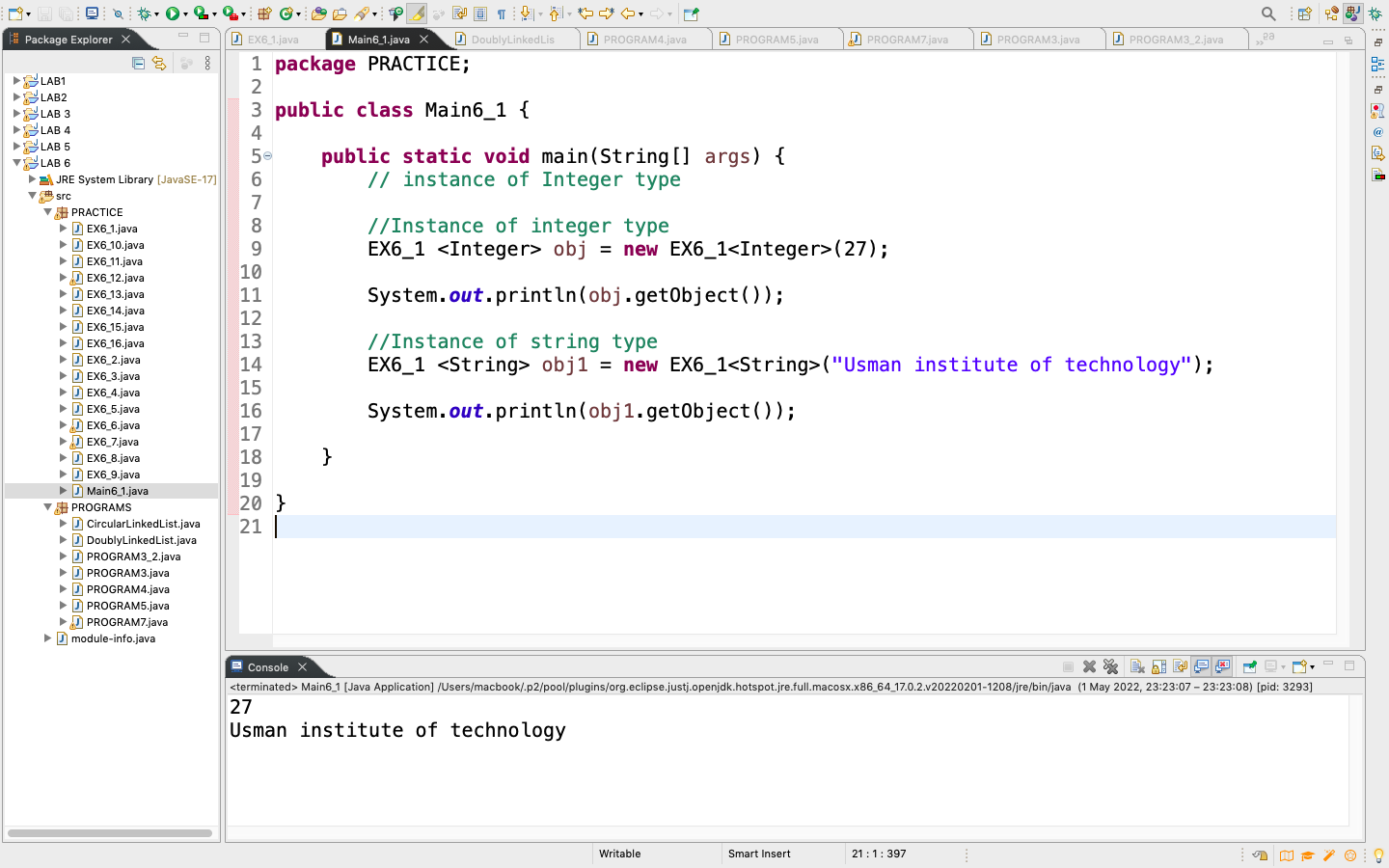
EX6\_1 <String> obj1 = **new** EX6\_1<String>("Usman institute of technology");

System.***out***.println(obj1.getObject());

}

}

OUTPUT:



Example 6.2:Program demonstrates the workingGenerics with Functions

CODE:

**package** PRACTICE;

//To create a generic function

//public static void func(T a, T b){}

//Note: In Parameter type,

//we can not use primitives like

//'int', 'char' or 'double'.

// A Simple Java program

// to show working of user defined

// Generic functions

**public** **class** EX6\_2 {

//A generic method example

**static** <T> **void** displayElement(T element) {

System.***out***.println(element.getClass().getName() + " = " + element);

}

// Driver method

**public** **static** **void** main(String[] args) {

// Calling generic method

// with Integer argument

*displayElement*(27);

// Calling generic method

// with String argument

*displayElement*("Usman institute of technology");

// Calling generic method

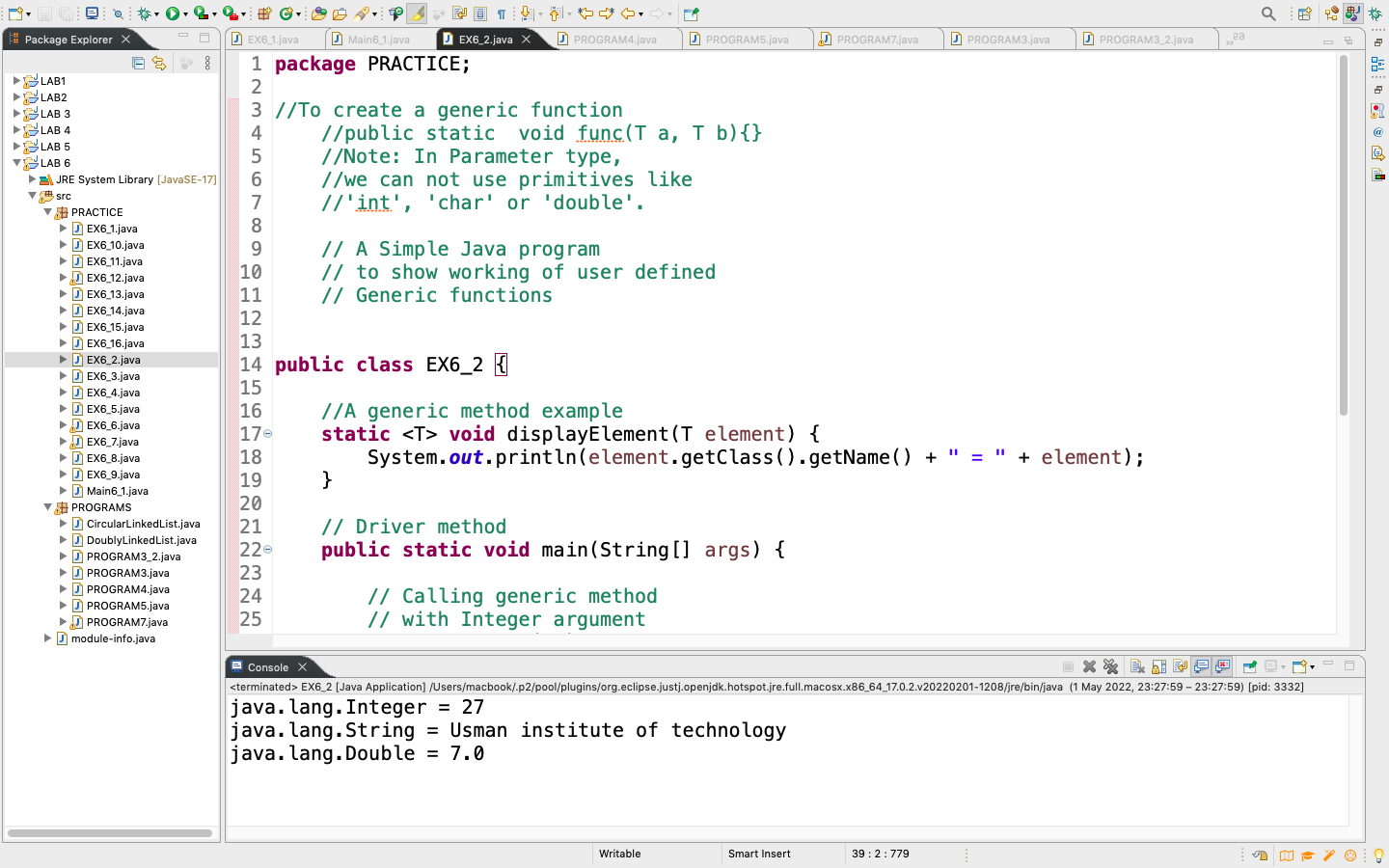
// with double argument

*displayElement*(7.0);

}

}

OUTPUT:



Example 6.3: Write a Java program to create a node of linked list using alternative approach.

CODE:

**package** PRACTICE;

//Linked list implementation in Java

**public** **class** EX6\_3 {

//creating a node

Node head;

**static** **class** Node {

**int** value;

Node next;

Node(**int** d){

value = d;

next = **null**;

}

}

// Driver method

**public** **static** **void** main(String[] args) {

EX6\_3 LinkedList = **new** EX6\_3();

//Assign the values

LinkedList.head = **new** Node(1);

Node Second = **new** Node(2);

Node third = **new** Node(3);

//connect the nodes

LinkedList.head.next = Second;

Second.next = third;

//printing node values

**while**(LinkedList.head != **null**) {

System.***out***.print(LinkedList.head.value + " ");

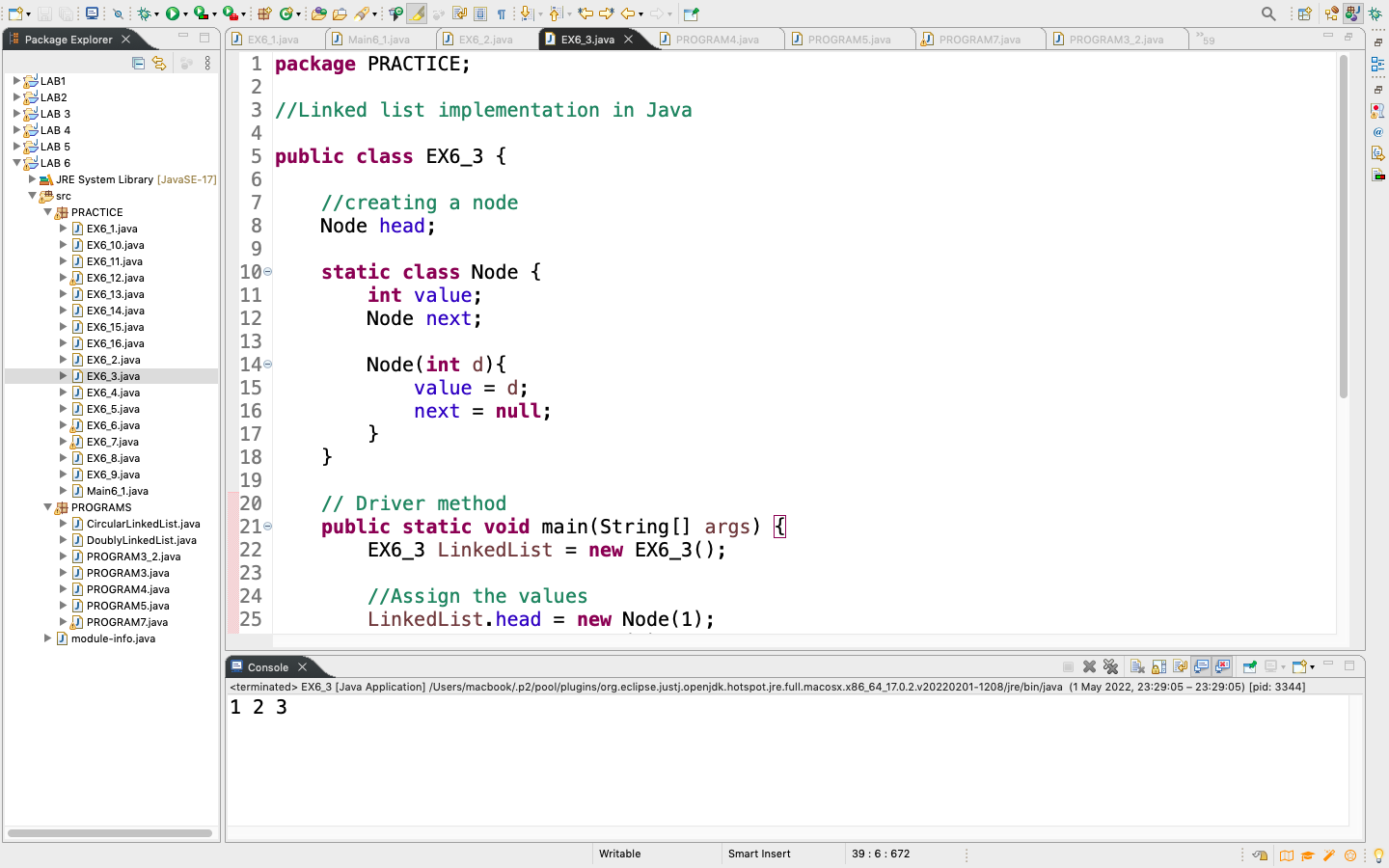
LinkedList.head = LinkedList.head.next;

}

}

}

OUTPUT:



Example 6.4:Write a Java program to append the specified element to the end of a linked list.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_4 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

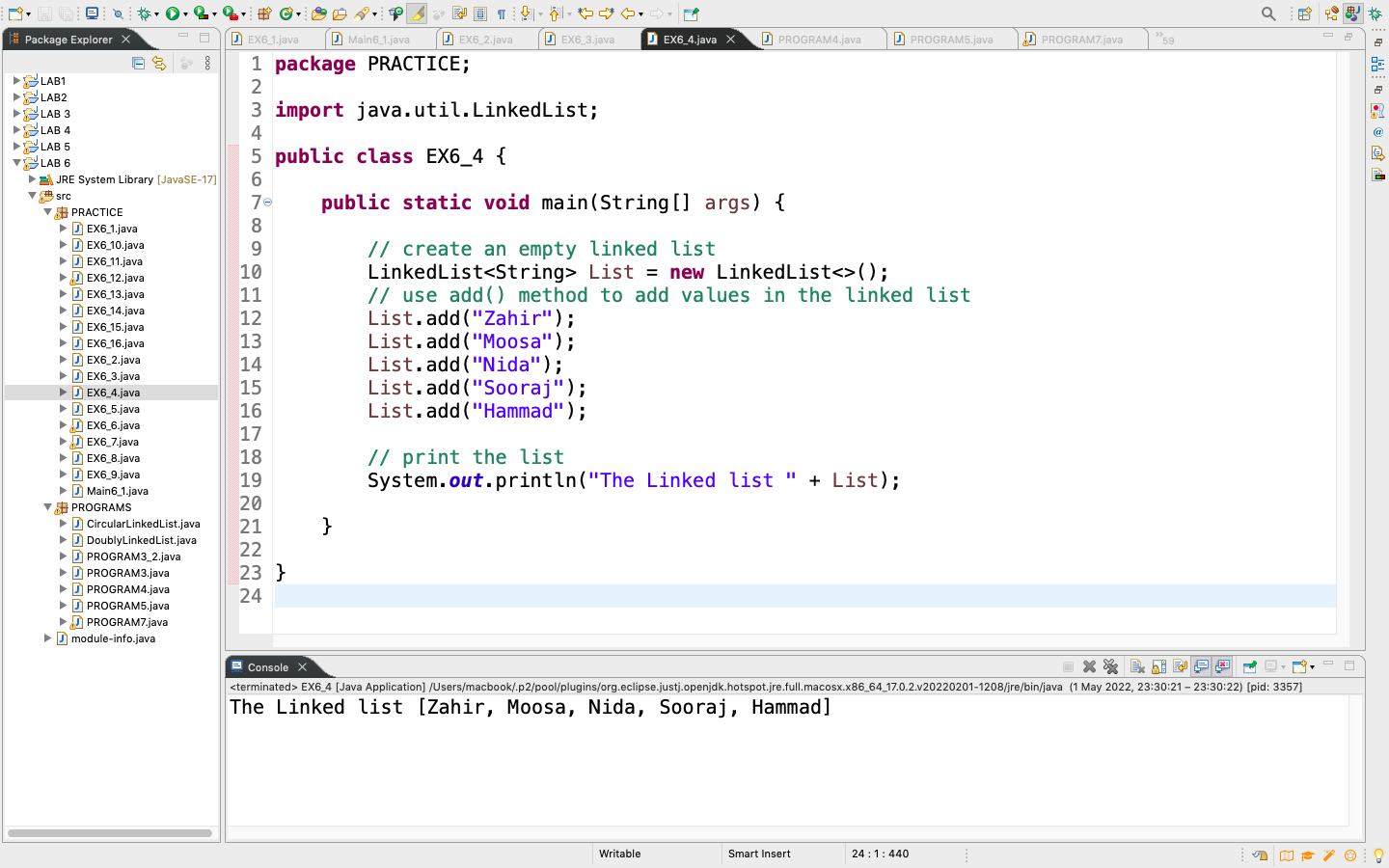
// print the list

System.***out***.println("The Linked list " + List);

}

}

OUTPUT:



Example 6.5:Write a Java program to iterate through all elements in a linked list.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_5 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

// Print the linked list

**for**(String element : List) {

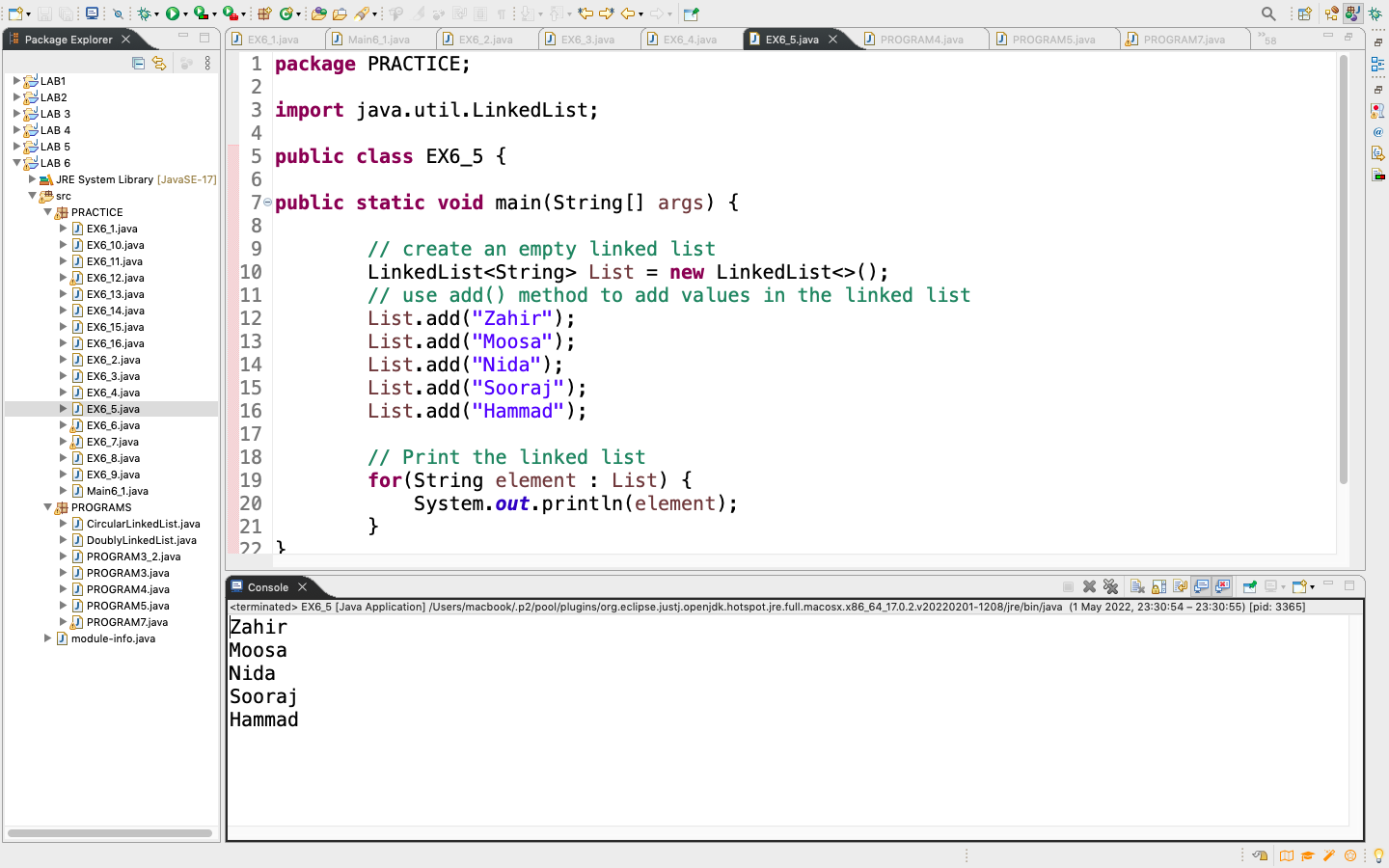
System.***out***.println(element);

}

}

}

OUTPUT:



Example 6.6:Write a Java programto iterate through all elements in a linked list but user will specify the starting position.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**import** java.util.Iterator;

**public** **class** EX6\_6 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

// set Iterator at specified index

Iterator p = List.listIterator(1);

//print list from second position

**while**(p.hasNext()) {

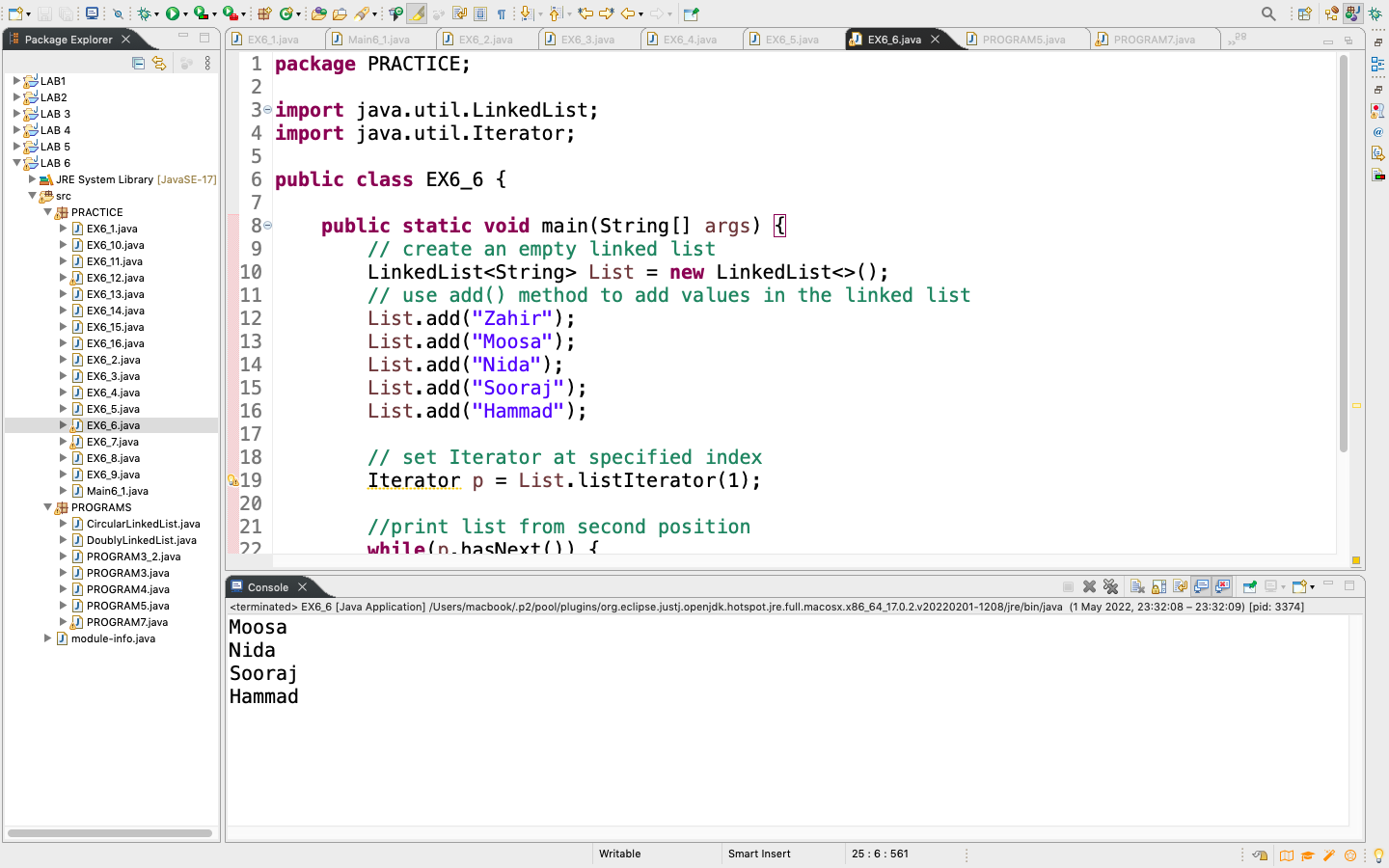
System.***out***.println(p.next());

}

}

}

OUTPUT:



Example 6.7:Write a Java programto iterate a linked list in reverse order.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**import** java.util.Iterator;

**public** **class** EX6\_7 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

// print the list

System.***out***.println("The Linked list " + List);

Iterator it = List.descendingIterator();

//print list in descending order

System.***out***.println("Elements in Reverse Order:");

**while**(it.hasNext()) {

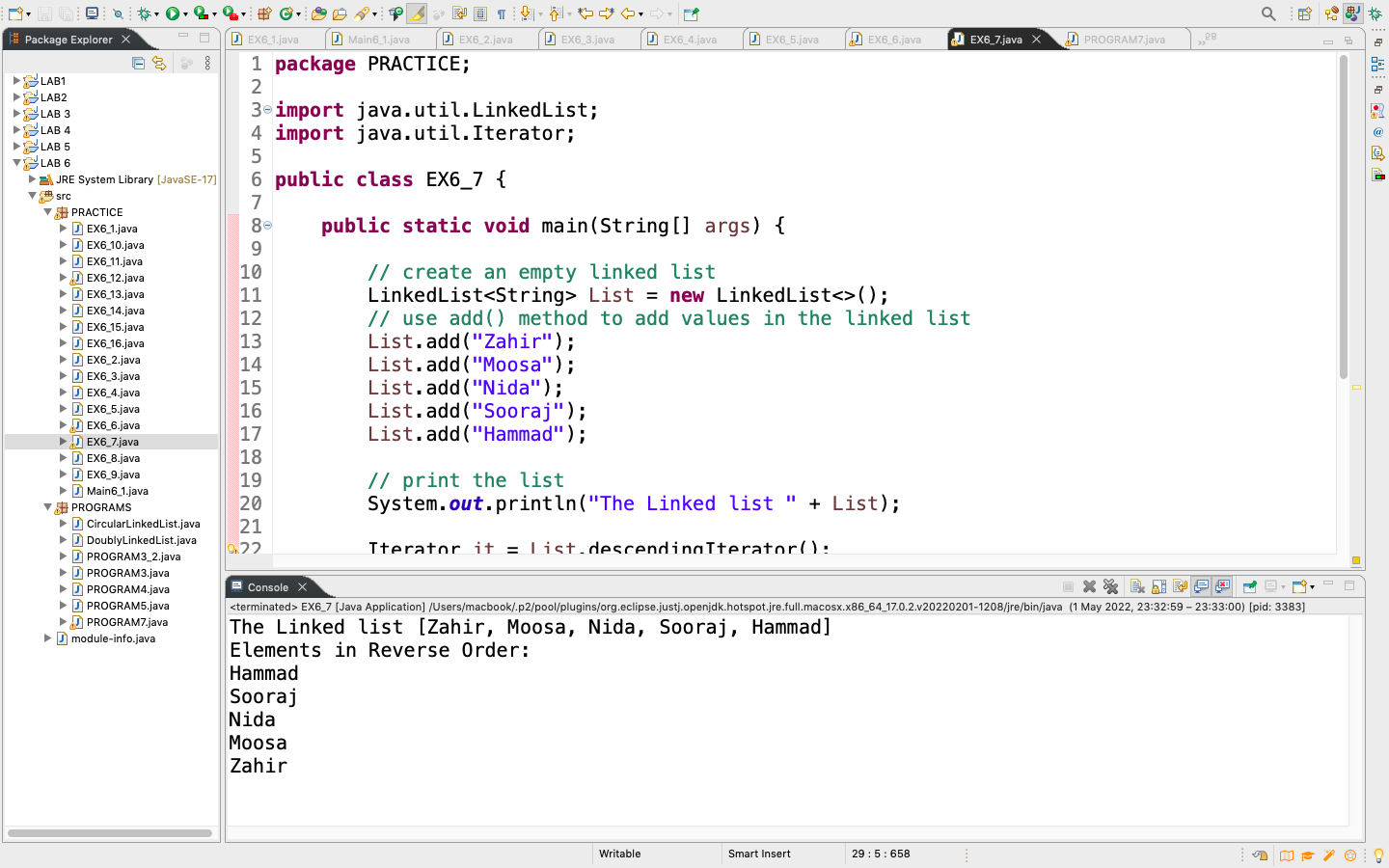
System.***out***.println(it.next());

}

}

}

OUTPUT:



Example 6.8:Write a Java programto add a faculty member after 1stmember.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_8 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

System.***out***.println("Orignal Linked List");

System.***out***.println("Let add the faculty member Hassan after Zahir: " + List);

List.add(1, "Hassan");

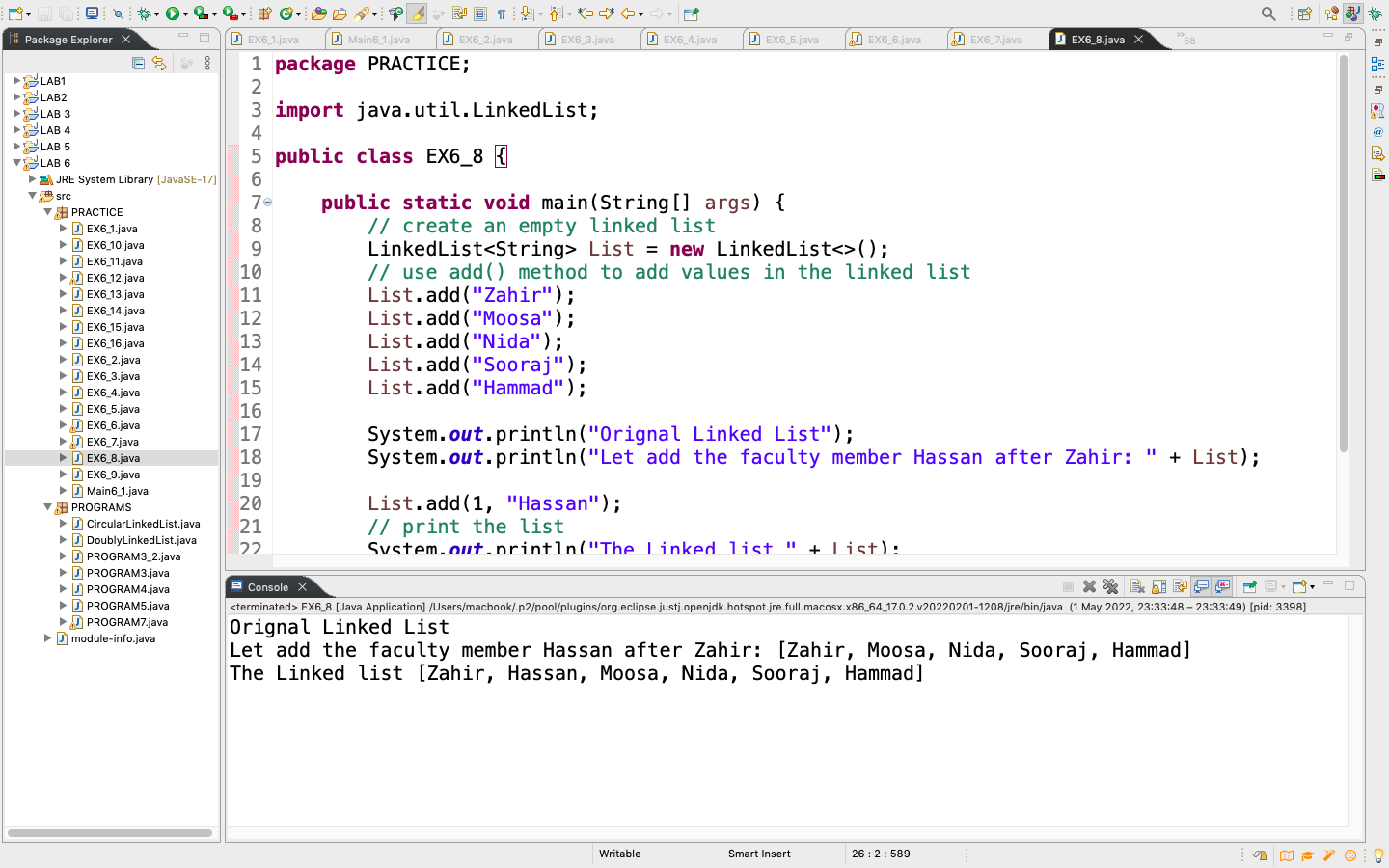
// print the list

System.***out***.println("The Linked list " + List);

}

}

OUTPUT:



Example 6.9:Write a Java programto insert elements into linked list at the first and the last position.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_9 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

System.***out***.println("Orignal Linked List" + List);

//Add element at the beginning

List.addFirst("Ariz");

//Add element at the last

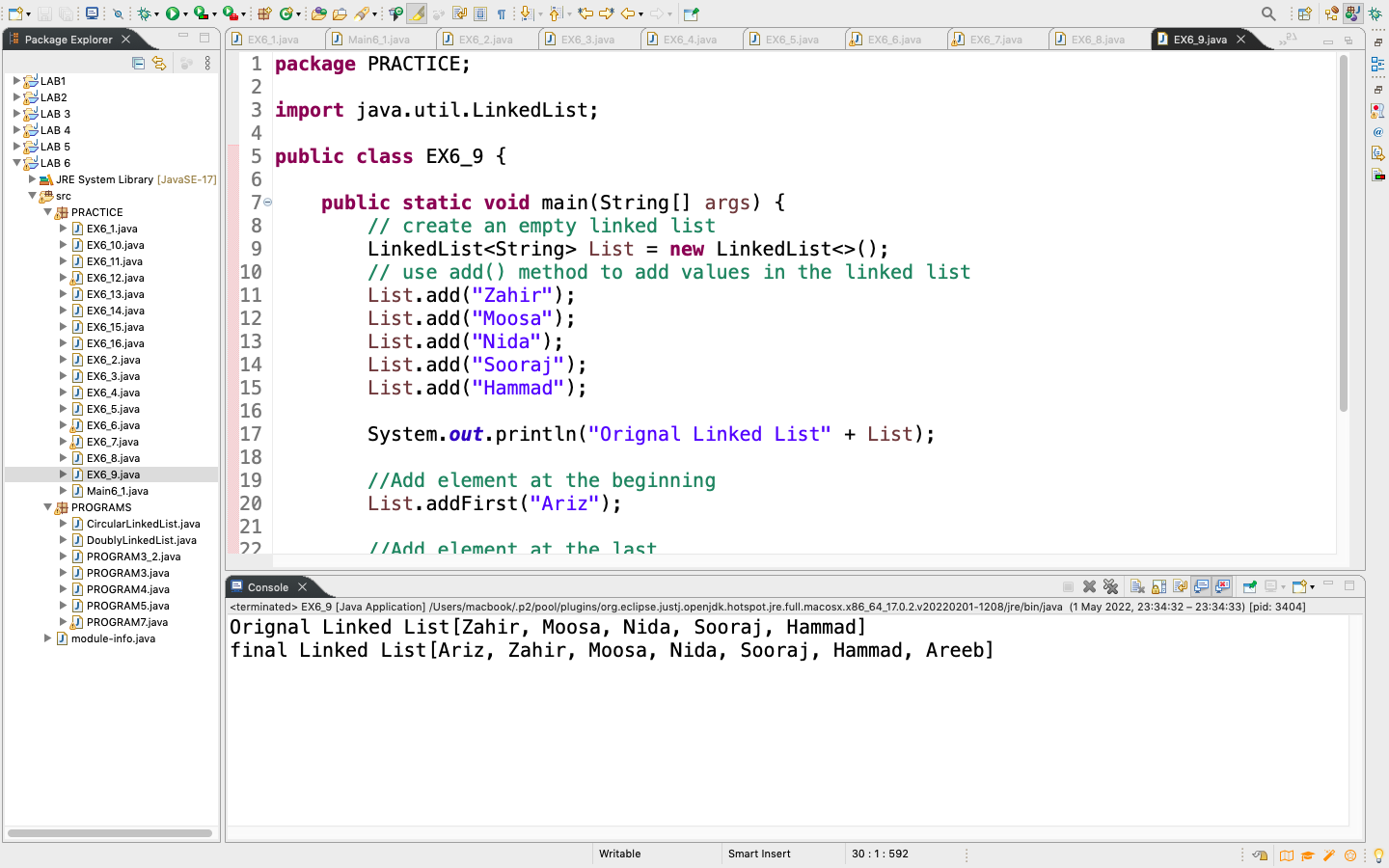
List.addLast("Areeb");

System.***out***.println("final Linked List" + List);

}

}

OUTPUT:



Example 6.10:Write a Java programto insert specific element at the front of a linked list.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_10 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

System.***out***.println("Orignal Linked List" + List);

//Add an element to the front of the LinkedList

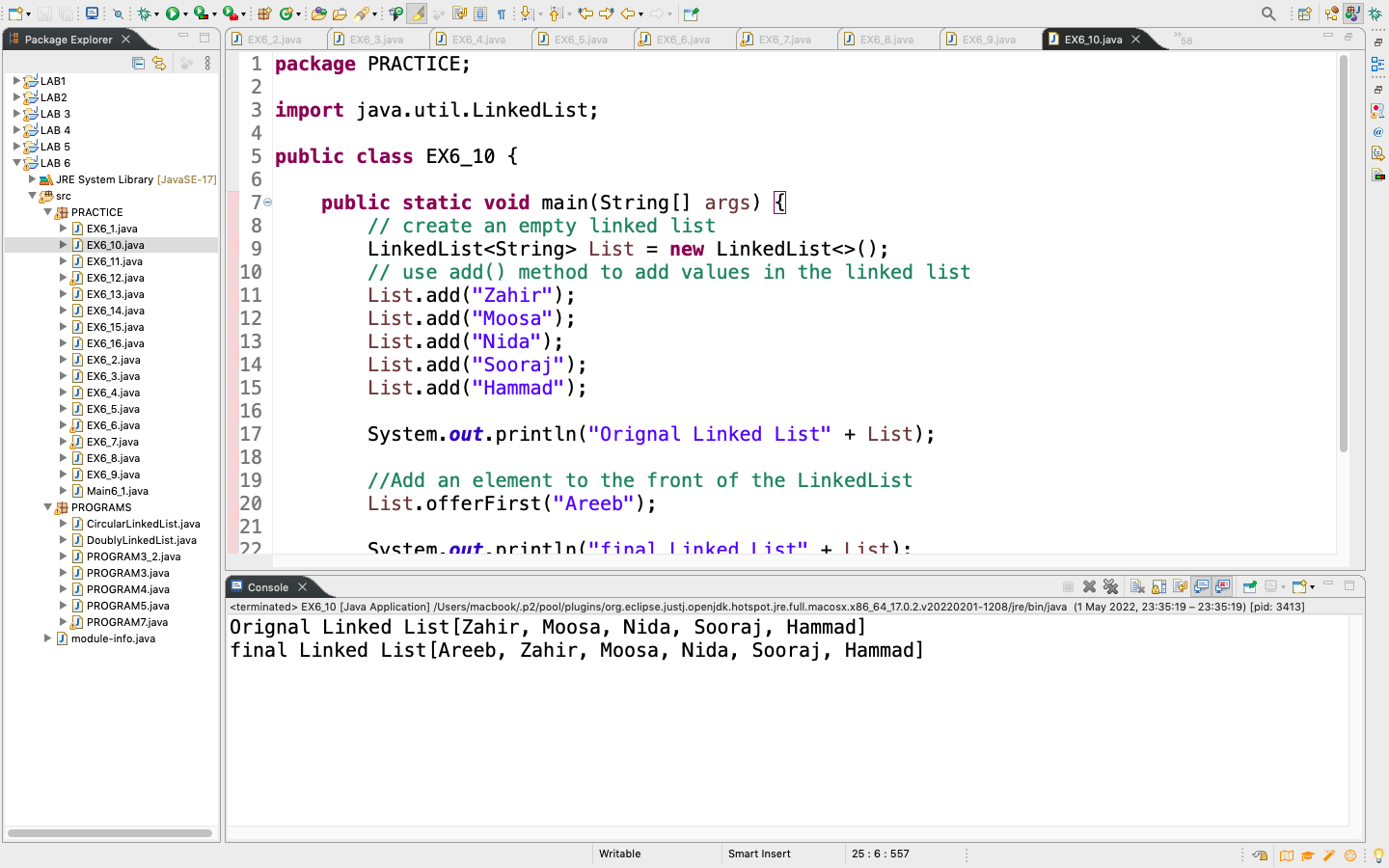
List.offerFirst("Areeb");

System.***out***.println("final Linked List" + List);

}

}

OUTPUT:



Example 6.11:Write a Java programto display the elements and their positions in a linked list.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_11 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

//print the list

System.***out***.println("Orignal Linked List" + List);

**for**(**int** i=0; i<List.size(); i++) {

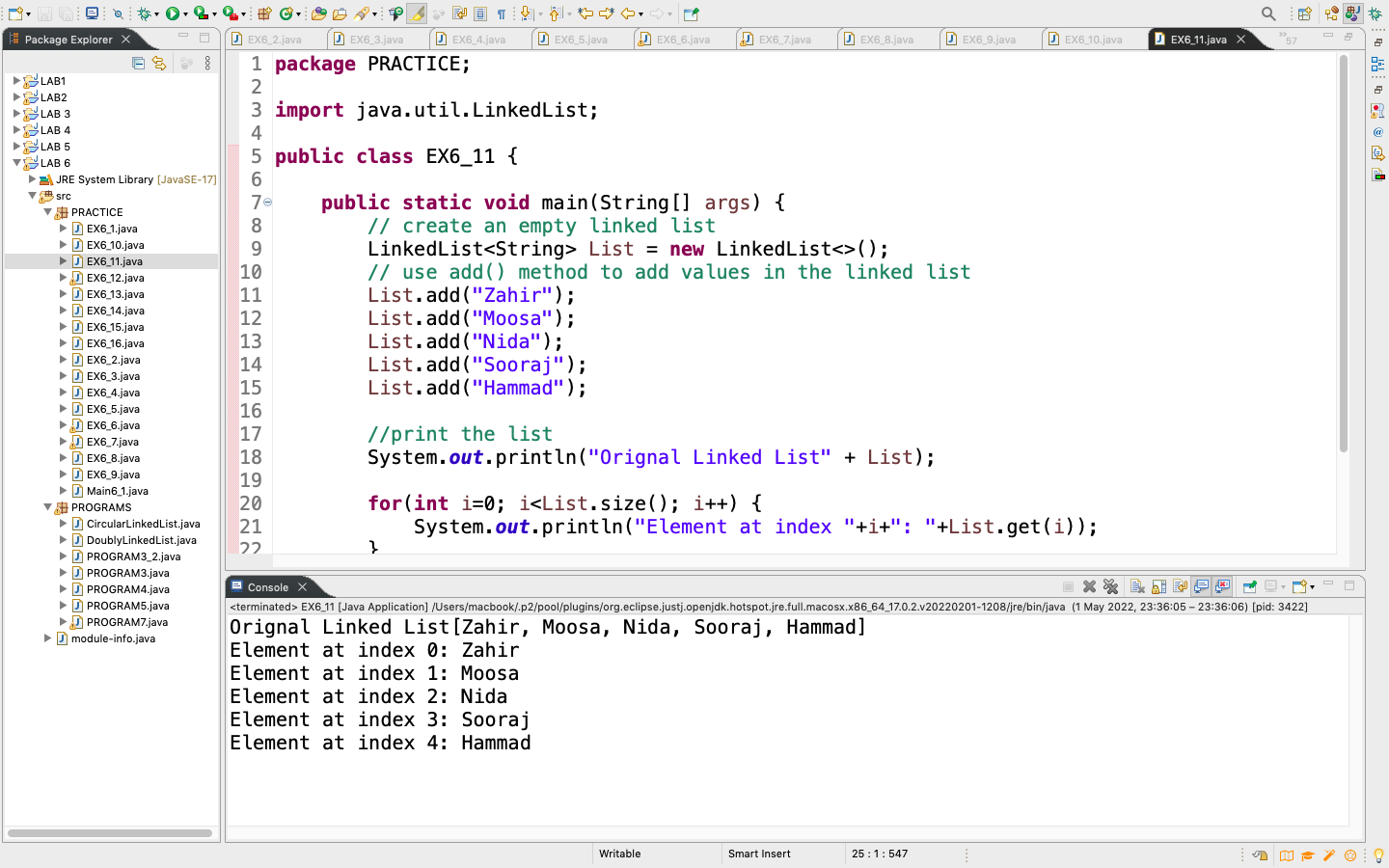
System.***out***.println("Element at index "+i+": "+List.get(i));

}

}

}

OUTPUT:



Example 6.12:Write a Java programto remove first and last elements from the linked list.

CODE:

package PRACTICE;

import java.util.LinkedList;

import java.util.\*;

public class EX6\_12 {

public static void main(String[] args) {

// create an empty linked list

LinkedList<String> List = new LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

//print the list

System.out.println("Orignal Linked List" + List);

// Remove the first element

Object firstElement = List.removeFirst();

System.out.println("Element removed: "+ firstElement);

// Remove the last element

Object lastElement = List.removeLast();

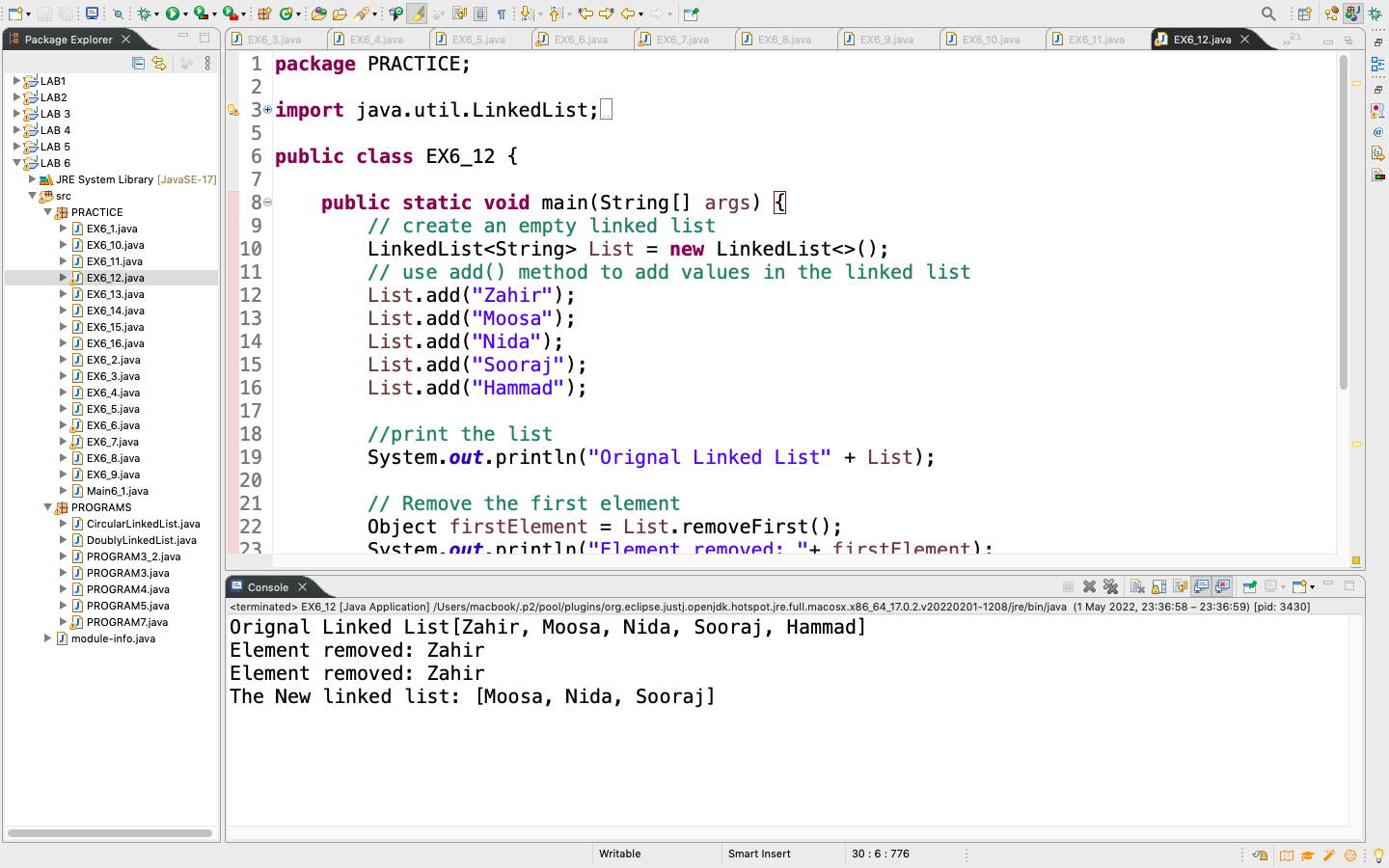
System.out.println("Element removed: "+ firstElement);

System.out.println("The New linked list: " + List);

}

}

OUTPUT:



Example 6.13:Write a Java programto swap two elements in a linked list.

CODE:

package PRACTICE;

import java.util.LinkedList;

import java.util.\*;

public class EX6\_13 {

public static void main(String[] args) {

// create an empty linked list

LinkedList<String> List = new LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

//print the list

System.out.println("Orignal Linked List" + List);

//Swapping 1st(index 0) element(Zahir) with the 3rd(index 2) element (Nida)

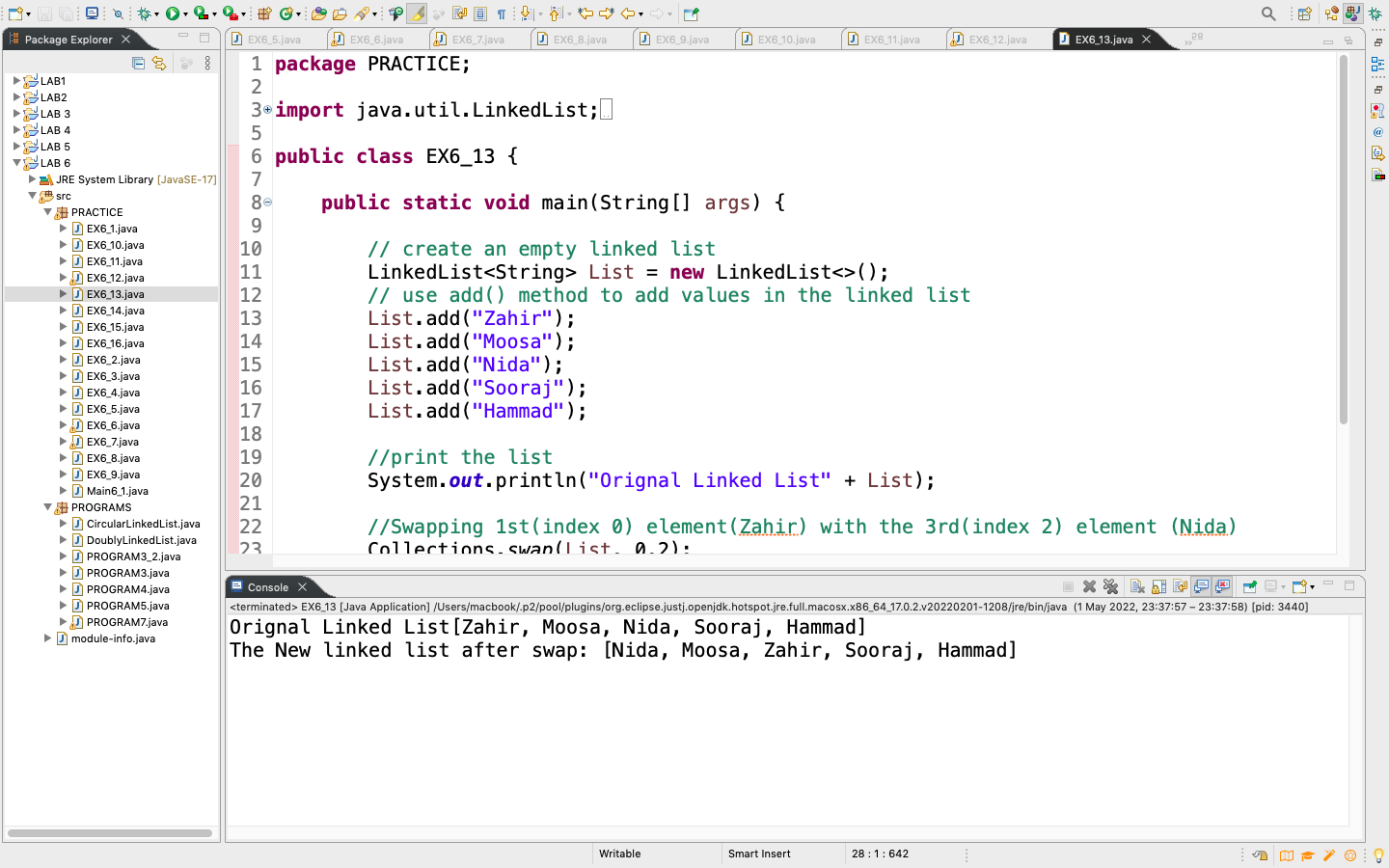
Collections.swap(List, 0,2);

System.out.println("The New linked list after swap: " + List);

}

}

OUTPUT:



Example 6.14:Write a Java programto compare two linked lists.

CODE:

**package** PRACTICE;

**import** java.util.LinkedList;

**public** **class** EX6\_14 {

**public** **static** **void** main(String[] args) {

// create an empty linked list

LinkedList<String> List = **new** LinkedList<>();

// use add() method to add values in the linked list

List.add("Zahir");

List.add("Moosa");

List.add("Nida");

List.add("Sooraj");

List.add("Hammad");

LinkedList<String> List1 = **new** LinkedList<>();

List1.add("Sawera");

List1.add("Shaheer");

List1.add("Areeb");

List1.add("Hassan");

//comparison output in linked list

LinkedList<String> List2 = **new** LinkedList<>();

**for**(String e : List) {

List2.add(List1.contains(e)? "Yes" : "No");

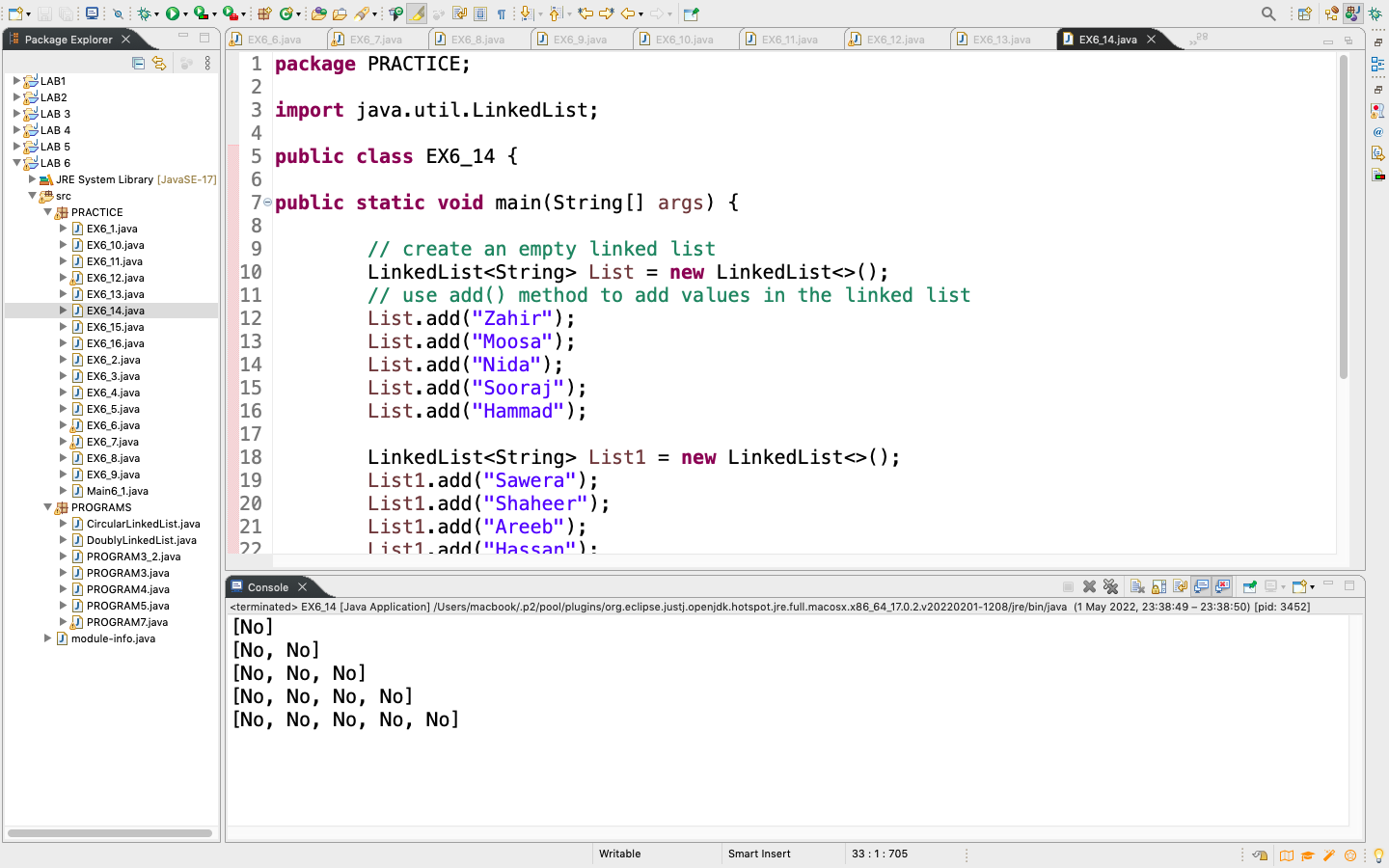
System.***out***.println(List2);

}

}

}

OUTPUT:



Example 6.15:Write a Java programto perform operations onlinked listssuch as insert, insert after, insert at the end, delete and search.

CODE:

**package** PRACTICE;

//Linked list operations in Java

**public** **class** EX6\_15 {

Node head;

//create node

**class** Node{

**int** data;

Node next;

Node(**int** d){

data = d;

next = **null**;

}

}

// Insert at the beginning

**public** **void** insertAtbeginning(**int** new\_data) {

//insert the data

Node new\_node = **new** Node(new\_data);

new\_node.next = head;

head = new\_node;

}

// Insert after a node

**public** **void** insertAfter(Node prev\_node, **int** new\_data) {

**if**(prev\_node==**null**) {

System.***out***.println("The given previous node cannot be null");

**return**;

}

Node new\_node = **new** Node(new\_data);

new\_node.next = prev\_node.next;

prev\_node.next = new\_node;

}

// Insert at the end

**public** **void** insertAtEnd(**int** new\_data) {

Node new\_node = **new** Node(new\_data);

**if**(head==**null**) {

head = **new** Node(new\_data);

**return**;

}

new\_node.next = **null**;

Node last = head;

**while**(last.next!=**null**) {

last = last.next;

last.next = new\_node;

**return**;

}

}

//Delete a node

**void** deleteNode(**int** position) {

**if**(head==**null**) {

**return**;

}

Node temp = head;

**if**(position==0) {

head = temp.next;

**return**;

}

//Find the key to be deleted

**for**(**int** i=0; temp!=**null** && i<position - 1; i++) {

temp = temp.next;

}

//if the key is not present

**if**(temp==**null** || temp.next==**null**) {

**return**;

}

//remove the node

Node next =temp.next.next;

temp.next = next;

}

//Search a node

**boolean** search(Node head, **int** key) {

Node current = head;

**while**(current!=**null**) {

**if**(current.data==key) {

**return** **true**;

}

current = current.next;

}

**return** **false**;

}

//Sort the linked list

**void** sortlinkedlist(Node head) {

Node current = head;

Node index = **null**;

**int** temp;

**if**(head==**null**) {

**return**;

}**else** {

**while**(current!=**null**) {

// index points to the node next to current

index = current.next;

**while**(index!=**null**) {

**if**(current.data>index.data) {

temp = current.data;

current.data = index.data;

index.data = temp;

}

index = index.next;

}

current = current.next;

}

}

}

//print the linked list

**public** **void** printList() {

Node node = head;

**while** (node != **null**) {

System.***out***.print(node.data + " ");

node = node.next;

}

}

**public** **static** **void** main(String[] args) {

EX6\_15 list = **new** EX6\_15();

list.insertAtEnd(1);

list.insertAtbeginning(2);

list.insertAtbeginning(3);

list.insertAtEnd(4);

list.insertAfter(list.head.next, 5);

System.***out***.println("Linked List");

list.printList();

System.***out***.println("\nAfter deleting an element: ");

list.deleteNode(2);

list.printList();

System.***out***.println();

**int** item\_to\_find = 3;

**if**(list.search(list.head, item\_to\_find)) {

System.***out***.println(item\_to\_find + " is found");

}**else** {

System.***out***.println(item\_to\_find + " is not found");

}

list.sortlinkedlist(list.head);

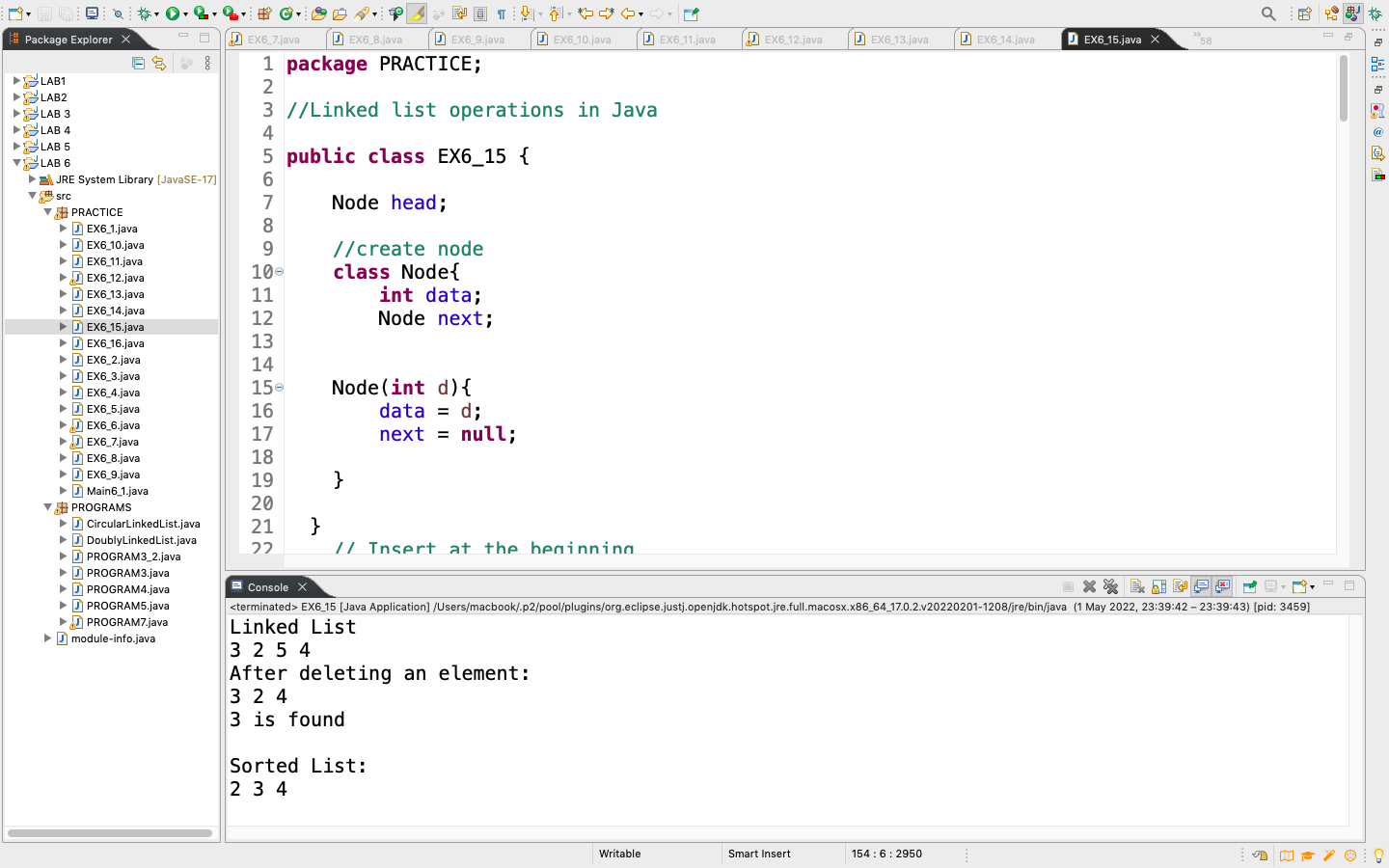
System.***out***.println("\nSorted List: ");

list.printList();

}

}

OUTPUT:



Example 6.16:Write a simple code to show SinglyLinkedList using Generics.

CODE:

**package** PRACTICE;

**public** **class** EX6\_16<T> {

**private** Node<T> head;

**private** Node<T> tail;

**public** **void** add(T element) {

Node<T> nd = **new** Node<T>();

nd.setValue(element);

System.***out***.println("Adding: " + element);

/\*\*

\* check if the list is empty

\*/

**if**(head==**null**) {

//since there is only one element, both head and

//tail points to the same object.

head = nd;

tail = nd;

}**else** {

//set current tail next link to new node

tail.setNextRef(nd);

//set tail as newly created node

tail = nd;

}

}

**public** **void** addAfter(T element, T after) {

Node<T> tmp = head;

Node<T> refNode = **null**;

System.***out***.println("Traversing to all nodes..");

/\*\*

\* Traverse till given element

\*/

**while**(**true**) {

**if**(tmp == **null**) {

**break**;

}

**if**(tmp.compareTo(after) == 0) {

//found the target node, add after this node

refNode = tmp;

**break**;

}

tmp = tmp.getNextRef();

}

**if**(refNode != **null**) {

//add element after the target element

Node<T> nd = **new** Node<T>();

nd.setValue(element);

nd.setNextRef(tmp.getNextRef());

**if**(tmp == tail) {

tail = nd;

}

tmp.setNextRef(nd);

}**else** {

System.***out***.println("Unable to find the given element...");

}

}

**public** **void** deleteFront() {

**if**(head == **null**) {

System.***out***.println("Underflow...");

}

Node<T> tmp = head;

head = tmp.getNextRef();

**if**(head==**null**) {

tail = **null**;

}

System.***out***.println("Deleted: " +tmp.getValue());

}

**public** **void** deleteAfter(T after) {

Node<T> tmp = head;

Node<T> refNode = **null**;

System.***out***.println("Traversing to all nodes..");

/\*\*

\* Traverse till given element

\*/

**while**(**true**) {

**if**(tmp == **null**) {

**break**;

}

**if**(tmp.compareTo(after) == 0) {

//found the target node, add after this node

refNode = tmp;

**break**;

}

tmp = tmp.getNextRef();

}

**if**(refNode != **null**) {

tmp = refNode.getNextRef();

refNode.setNextRef(tmp.getNextRef());

**if**(refNode.getNextRef() == **null**) {

tail = refNode;

}

System.***out***.println("Deleted: "+tmp.getValue());

}**else** {

System.***out***.println("Unable to find the given element...");

}

}

**public** **void** traverse() {

Node<T> tmp = head;

**while**(**true**) {

**if**(tmp == **null**) {

**break**;

}

System.***out***.println(tmp.getValue());

tmp = tmp.getNextRef();

}

}

**public** **static** **void** main(String[] args) {

EX6\_16<Integer> list = **new** EX6\_16<>();

list.add(3);

list.add(32);

list.add(54);

list.add(89);

list.addAfter(76, 54);

list.deleteFront();

list.deleteAfter(76);

list.traverse();

}

}

**class** Node<T> **implements** Comparable<T>{

**private** T value;

**private** Node<T> nextRef;

**public** T getValue() {

**return** value;

}

**public** **void** setValue(T value) {

**this**.value = value;

}

**public** Node<T> getNextRef(){

**return** nextRef;

}

**public** **void** setNextRef(Node<T> ref) {

**this**.nextRef = ref;

}

@Override

**public** **int** compareTo(T arg) {

**if**(arg == **this**.value) {

**return** 0;

}**else** {

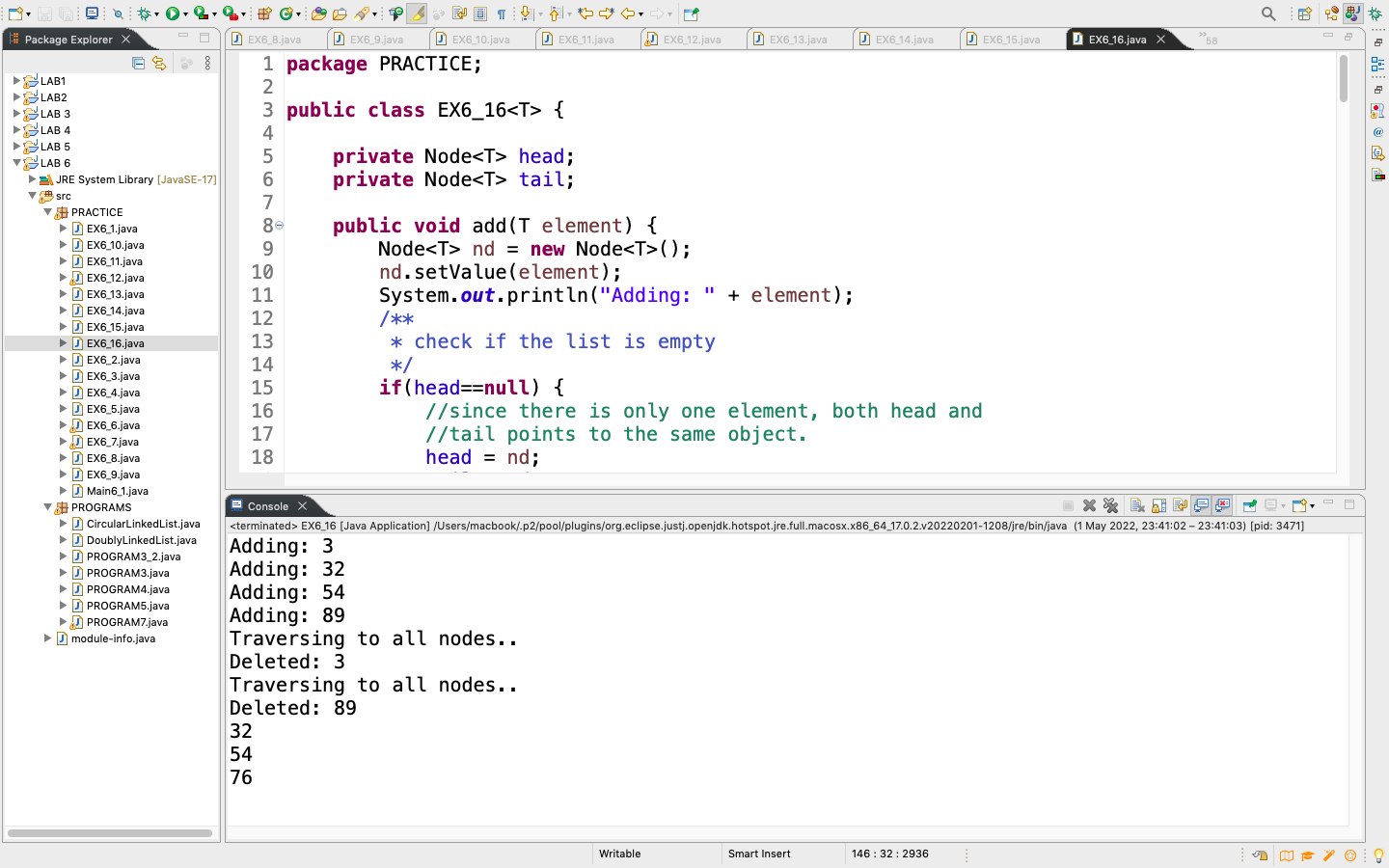
**return** 1;

}

}

}

OUTPUT:



PROGRAM

CIRCULAR LINKED LIST

CODE:

**package** PROGRAMS;

**public** **class** PROGRAM3 {

//creating node class

**class** Node{

**int** item;

Node next;

//constructor

Node(**int** item){

**this**.item =item;

next = **null**;

}

}

//initializing head and tail

Node head, tail = **null**;

**public** **void** addNode(**int** item) {

//creating object of node class

Node new\_node = **new** Node(item);

**if**(head==**null**) {

head = tail = new\_node;

new\_node.next = head;

}**else** {

tail.next = new\_node;

tail = new\_node;

tail.next = head;

}

}

**public** **void** insert(Node prev\_node , **int** data, Node after\_node) {

**if**(prev\_node==**null**) {

System.***out***.println("Previous node cannot be null");

**return**;

}

Node new\_node = **new** Node(data);

prev\_node.next = new\_node;

new\_node.next = after\_node;

}

**public** **void** delete(**int** data) {

Node del = head,prev = **null**;

// CASE 1:

// If head node itself holds the key to be deleted

**if** (del != **null** && del.item == data) {

head = del.next;

del.next = head;

**return**;

}

// CASE 2:

// If the key is somewhere other than at head

**while**(del!= **null** && del.item!=data) {

prev = del;

del = del.next;

}

**if** (del == **null**) {

System.***out***.println("Item is not present in the linked list");

**return**;

}

prev.next = del.next;

}

**public** **boolean** search(Node head, **int** key) {

Node serc = head;

**while**(serc!=**null**) {

**if**(serc.item==key) {

System.***out***.println("Item is found");

**return** **true**;

}

serc = serc.next;

}

System.***out***.println("Item is not found");

**return** **false**;

}

**public** **void** printNode() {

//initializing current to head

Node current = head;

**if**(head == **null**) {

System.***out***.println("circular Linked list is empty");

**return**;

}**else** {

System.***out***.println("The CIRCULAR Linked List is: ");

**do** {

System.***out***.print(current.item + " ");

current = current.next;

}**while**(current!=head);

System.***out***.println();

}

}

**public** **static** **void** main(String[] args) {

PROGRAM3 list = **new** PROGRAM3();

list.addNode(10);

list.addNode(20);

list.addNode(30);

list.addNode(40);

list.insert(list.head.next, 50, list.head.next.next);

list.delete(30);

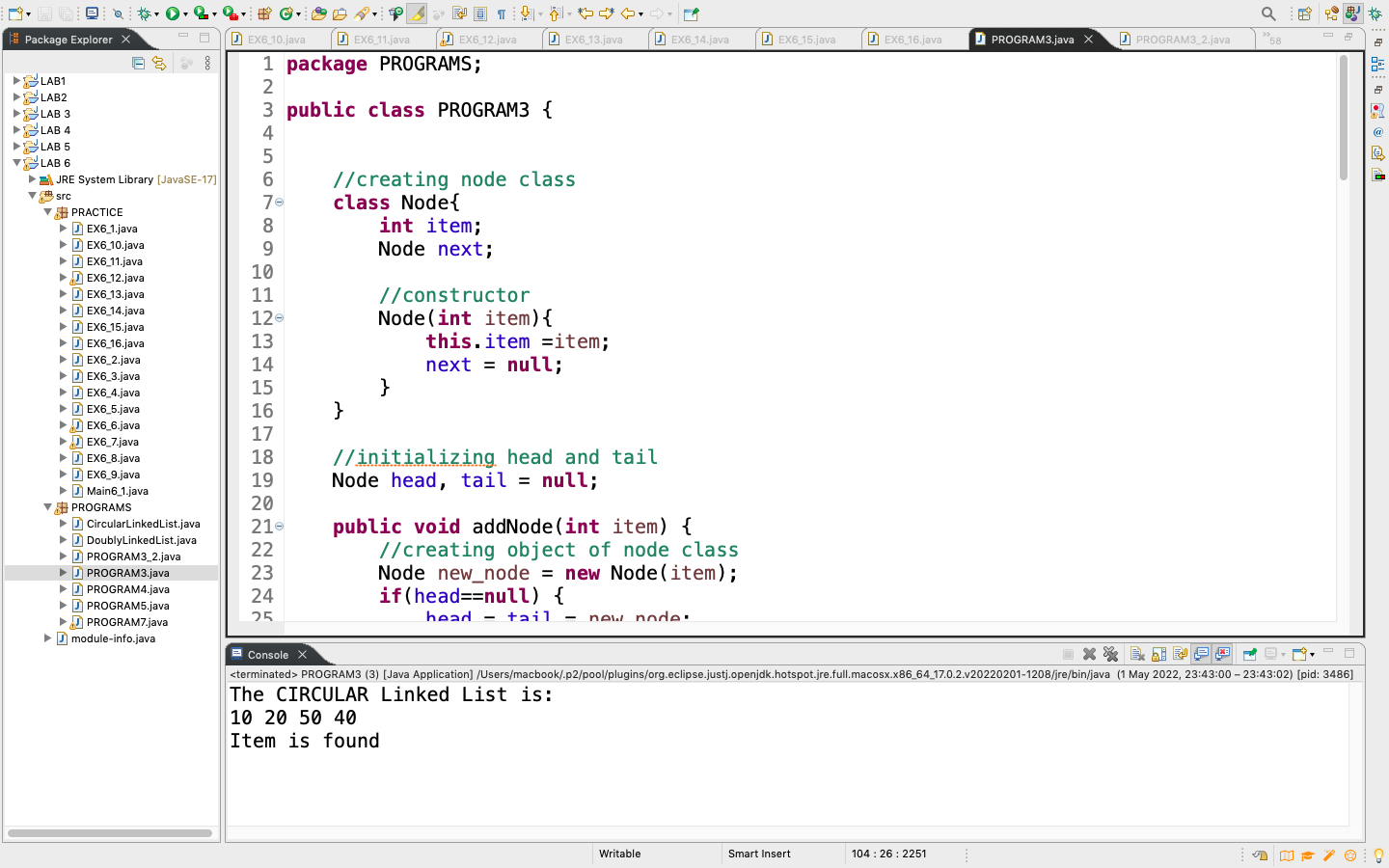
list.printNode();

list.search(list.head, 20);

}

}

OUTPUT:



DOUBLY LINKED LIST

CODE:

**package** PROGRAMS;

**public** **class** PROGRAM3\_2 {

//creating node class

**class** Node{

**int** item;

Node next;

Node previous;

//constructor

Node(**int** item){

**this**.item =item;

next = **null**;

previous = **null**;

}

}

//initializing head and tail

Node head, tail = **null**;

**public** **void** addNode(**int** item) {

//creating object of node class

Node new\_node = **new** Node(item);

**if**(head==**null**) {

head = tail = new\_node;

head.previous = **null**;

tail.next = **null**;

}**else** {

tail.next = new\_node;

new\_node.previous = tail;

tail = new\_node;

tail.next = **null**;

}

}

**public** **void** insert(Node prev\_node, **int** data, Node after\_node) {

**if**(prev\_node==**null**) {

System.***out***.println("Previous node cannot be null");

**return**;

}

Node new\_node = **new** Node(data);

prev\_node.next = new\_node;

new\_node.previous = prev\_node;

new\_node.next = after\_node;

after\_node.previous = new\_node;

}

**public** **void** delete(**int** data) {

Node del = head,prev = **null**;

**if** (del != **null** && del.item == data) {

head = del.next; // Changed head

**return**;

}

**while**(del!= **null** && del.item!=data) {

prev = del;

del = del.next;

}

**if** (del == **null**) {

System.***out***.println("Item is not present in the linked list");

**return**;

}

prev.next = del.next;

del.previous = prev;

}

**public** **boolean** search(Node head, **int** key) {

Node serc = head;

**while**(serc!=**null**) {

**if**(serc.item==key) {

System.***out***.println("Item is found");

**return** **true**;

}

serc = serc.next;

}

System.***out***.println("Item is not found");

**return** **false**;

}

**public** **void** printNode() {

//initializing current to head

Node current = head;

**if**(head == **null**) {

System.***out***.println("Doubly Linked list is empty");

**return**;

}**else** {

System.***out***.println("The Doubly Linked List is: ");

**while**(current!= **null**) {

System.***out***.print(current.item + " ");

current = current.next;

}

}

}

**public** **static** **void** main(String[] args) {

PROGRAM3\_2 list = **new** PROGRAM3\_2();

list.addNode(10);

list.addNode(20);

list.addNode(30);

list.addNode(40);

list.insert(list.head.next, 25, list.head.next.next);

list.delete(40);

list.printNode();

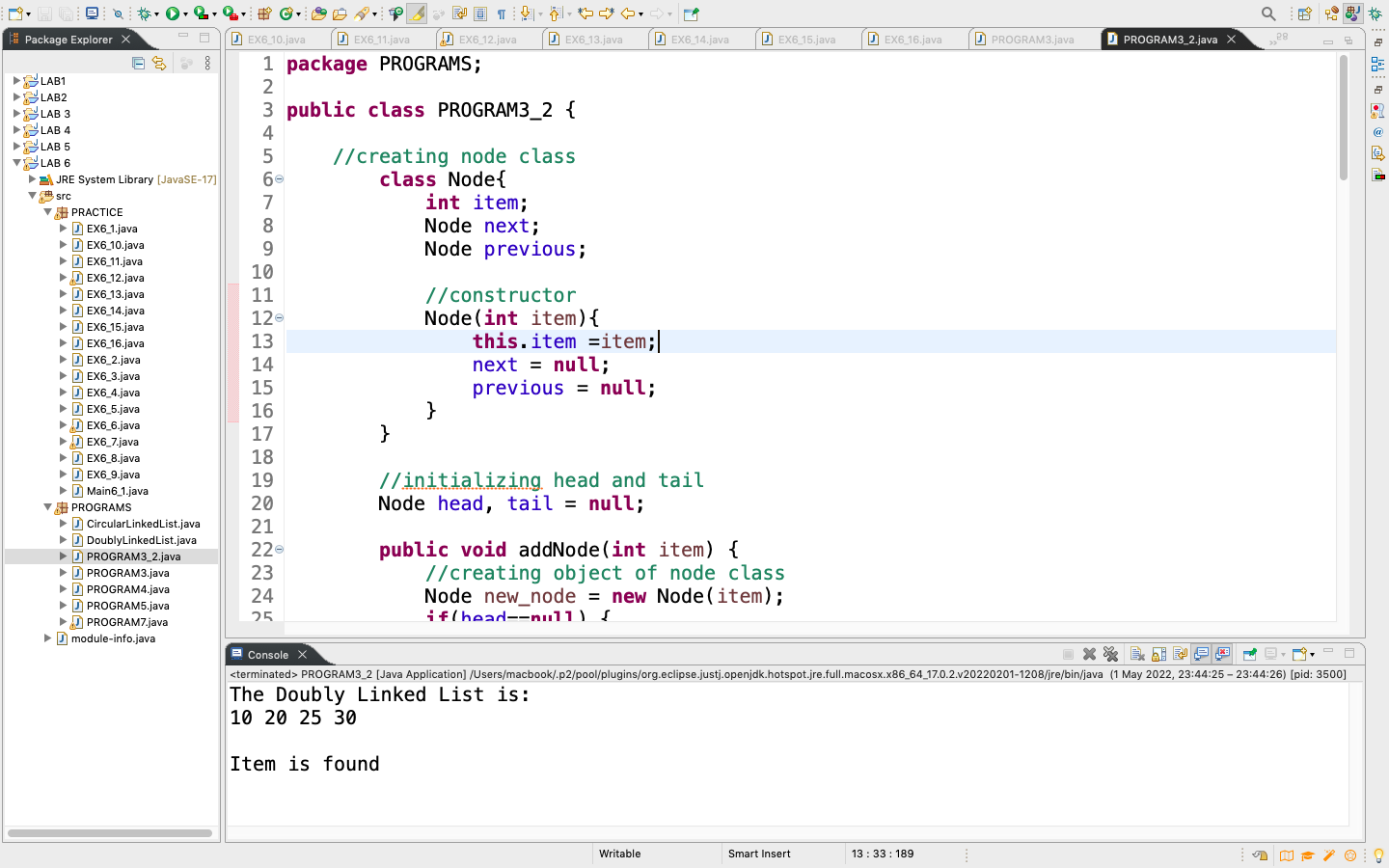
System.***out***.println("\n");

list.search(list.head, 25);

}

}

OUTPUT:



PROGRAM 4

CODE:

**package** PROGRAMS;

**public** **class** PROGRAM4 {

Node head;

**class** Node{

**int** data;

Node next;

Node(**int** data){

**this**.data = data;

}

}

**public** **void** addNode(**int** data) {

Node new\_node = **new** Node(data);

**if**(head==**null**) {

head = new\_node;

}**else** {

Node last = head;

**while** (last.next!=**null**){

last = last.next;

}

last.next = new\_node;

}

**return**;

}

**public** **void** printNode() {

//initializing current to head

Node current = head;

**if**(head == **null**) {

System.***out***.println("Linked list is empty");

}**else** {

System.***out***.println("Linked List is: ");

**while**(current!= **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

}

}

**public** **int**[] ToArray() {

**int**[] Array = **new** **int**[count()];

**int** index = 0;

Node currents = head;

**while**(currents!=**null**) {

Array[index++] = currents.data;

currents = currents.next;

}

**for**(**int** i=0; i<Array.length; i++) {

System.***out***.print(Array[i] + " ");

}

**return** Array;

}

**public** **int** count() {

Node temp = head;

**int** count = 0;

**while**(temp!=**null**) {

count++;

temp = temp.next;

}

**return** count;

}

**public** **static** **void** main(String[] args) {

PROGRAM4 list = **new** PROGRAM4();

list.addNode(10);

list.addNode(30);

list.addNode(70);

list.addNode(20);

list.printNode();

**int** A = list.count();

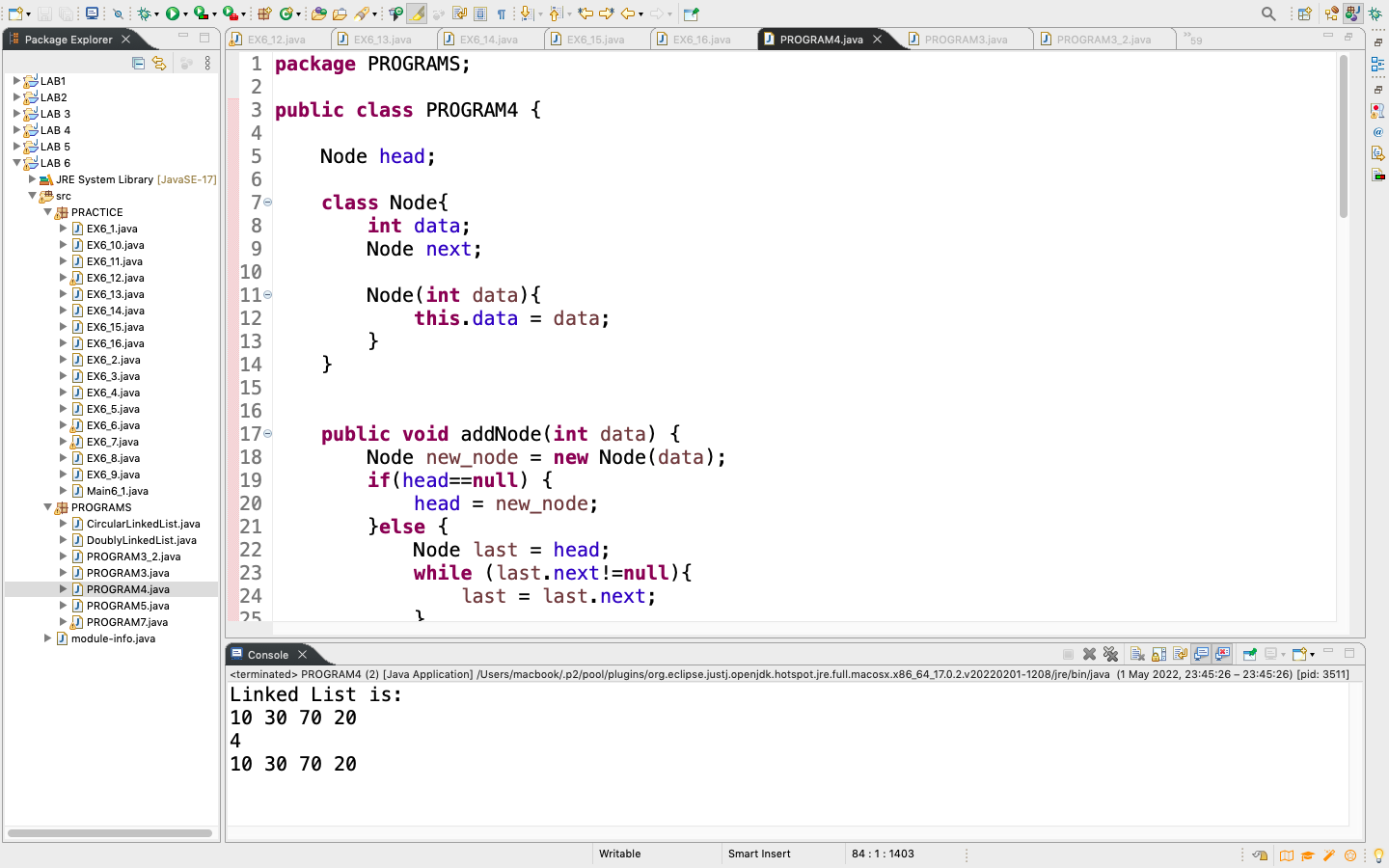
System.***out***.println("\n" + A);

list.ToArray();

}

}

OUTPUT:



PROGRAM 5

CODE:

**package** PROGRAMS;

**public** **class** PROGRAM5 {

Node head;

**class** Node{

**int** data;

Node next;

Node(**int** data){

**this**.data = data;

}

}

**public** **void** addNode(**int** data) {

Node new\_node = **new** Node(data);

**if**(head==**null**) {

head = new\_node;

}**else** {

Node last = head;

**while** (last.next!=**null**){

last = last.next;

}

last.next = new\_node;

}

**return**;

}

**public** **void** printNode() {

//initializing current to head

Node current = head;

**if**(head == **null**) {

System.***out***.println("Linked list is empty");

}**else** {

System.***out***.println("Linked List is: ");

**while**(current!= **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

}

}

**public** **void** Clear() {

head = **null**;

}

**public** **static** **void** main(String[] args) {

PROGRAM5 list = **new** PROGRAM5();

list.addNode(10);

list.addNode(30);

list.addNode(70);

list.addNode(20);

list.printNode();

list.Clear();

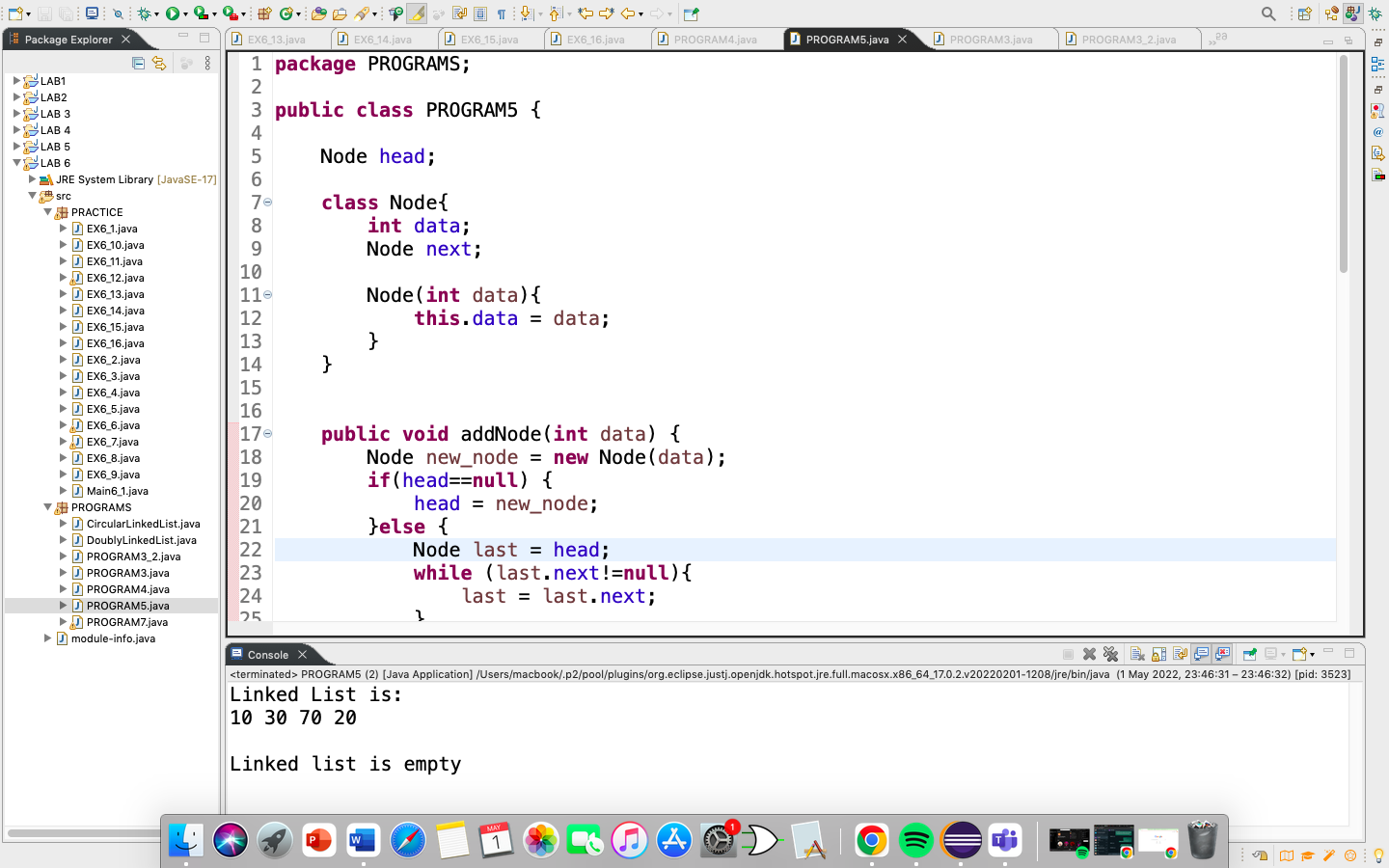
System.***out***.println("\n");

list.printNode();

}

}

OUTPUT:



PROGRAM 7

CODE:

**package** PROGRAMS;

**import** java.util.Scanner;

**public** **class** PROGRAM7 {

Node head;

**class** Node{

**int** data;

Node next;

Node(**int** data){

**this**.data = data;

}

}

**public** **void** addNode(**int** data) {

Node new\_node = **new** Node(data);

**if**(head==**null**) {

head = new\_node;

}**else** {

Node last = head;

**while** (last.next!=**null**){

last = last.next;

}

last.next = new\_node;

}

**return**;

}

**public** **void** printNode() {

//initializing current to head

Node current = head;

**if**(head == **null**) {

System.***out***.println("Linked list is empty");

}**else** {

**while**(current!= **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

}

}

**public** **void** insertAtbeginning(**int** data) {

Node new\_node = **new** Node(data);

new\_node.next = head;

head = new\_node;

}

**public** **void** insert(Node prev\_node,**int** data, Node after\_node) {

**if**(prev\_node==**null**) {

System.***out***.println("Previous node cannot be null");

**return**;

}

Node new\_node = **new** Node(data);

prev\_node.next = new\_node;

new\_node.next = after\_node;

}

**public** **void** delete(**int** data) {

Node del = head,prev = **null**;

**if** (del != **null** && del.data == data) {

head = del.next; // Changed head

**return**;

}

**while**(del!= **null** && del.data!=data) {

prev = del;

del = del.next;

}

**if** (del == **null**) {

System.***out***.println("Item is not present in the linked list");

**return**;

}

prev.next = del.next;

}

**public** **int** count() {

Node temp = head;

**int** count = 0;

**while**(temp!=**null**) {

count++;

temp = temp.next;

}

**return** count;

}

**public** **boolean** search(Node head, **int** key) {

Node serc = head;

**while**(serc!=**null**) {

**if**(serc.data==key) {

System.***out***.println("Item is found");

**return** **true**;

}

serc = serc.next;

}

System.***out***.println("Item is not found");

**return** **false**;

}

**public** **static** **void** main(String[] args) {

PROGRAM7 list = **new** PROGRAM7();

list.addNode(10);

list.addNode(30);

list.addNode(70);

list.addNode(20);

System.***out***.println("The Linked List is: ");

list.printNode();

System.***out***.println("\n");

Scanner input = **new** Scanner(System.***in***);

System.***out***.println("Press 1 to insert element\nPress 2 to insert element at first position\nPress 3 to delete element\nPress 4 to search element\nPress 5 to find length of linked list");

**int** number = input.nextInt();

**if**(number==1) {

System.***out***.println("Enter data to insert");

**int** data = input.nextInt();

System.***out***.println("\nAfter insertion the Linked List is: ");

list.insert(list.head.next, data, list.head.next.next);

list.printNode();

}

**if**(number==2) {

System.***out***.println("Enter data to insert at first");

**int** data = input.nextInt();

System.***out***.println("\nAfter insertion at beginning the Linked List is: ");

list.insertAtbeginning(data);

list.printNode();

}

**if**(number==3) {

System.***out***.println("Enter data to delete");

**int** data = input.nextInt();

list.delete(data);

System.***out***.println("\nAfter deletion the Linked List is: ");

list.printNode();

}

**if**(number==4) {

System.***out***.println("Enter data to search");

**int** data = input.nextInt();

list.search(list.head, data);

}

**if**(number==5) {

System.***out***.println("\nThe length of Linked List is: " + list.count());

}

}

}

OUTPUT:

