Task1:

#include <iostream>

int main() {

int age;

int \*ptr;

age = 10;

ptr = &age;

std::cout << "value of age is " << age << std::endl;

std::cout << "ptr is pointing to " << \*ptr << std::endl;

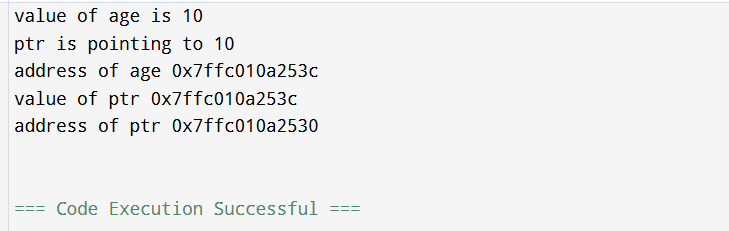
std::cout << "address of age " << &age << std::endl;

std::cout << "value of ptr " << ptr << std::endl;

std::cout << "address of ptr " << &ptr << std::endl;

return 0;

}



Task2:

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

int data;

Node\* next;

Node(int value) : data(value), next(nullptr) {}

};

class Linkedlist{

private:

Node\* head;

public:

Linkedlist(){

head = nullptr;

}

void insertAtEnd(int value){

Node\* newNode = new Node(value);

if(head == nullptr){

head = newNode;

}

else{

Node\* temp = head;

while (temp->next != nullptr){

temp = temp->next;

}

temp->next = newNode;

}

}

void deleteByValue(int value){

if(head == nullptr){

return;

}

if(head->data == value){

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* temp = head;

while(temp->next && temp->next->data != value){

temp = temp->next;

}

if(temp->next){

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

delete nodeToDelete;

}

}

void display(){

Node\* temp = head;

while(temp != nullptr){

cout << temp->data << "->";

temp = temp->next;

}

cout << "NULL" <<endl;

}

~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();

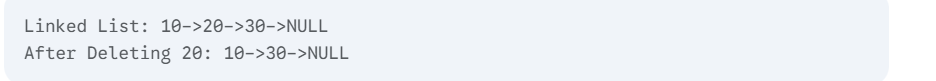
list.deleteByValue(20);

cout << "After Deleting 20: ";

list.display();

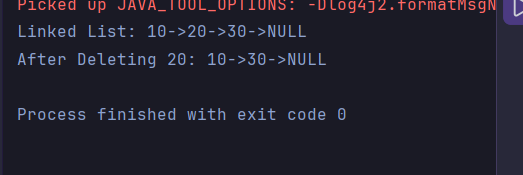
return 0;

}



Task3:

class Node {  
 int data;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
public class Task3 {  
 private Node head;  
  
 public Task3() {  
 this.head = null;  
 }  
  
 public void insertAtEnd(int value) {  
 Node newNode = new Node(value);  
 if (head == null) {  
 head = newNode;  
 } else {  
 Node temp = head;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
 temp.next = newNode;  
 }  
 }  
  
 public void deleteByValue(int value) {  
 if (head == null) {  
 return;  
 }  
 if (head.data == value) {  
 head = head.next;  
 return;  
 }  
 Node temp = head;  
 while (temp.next != null && temp.next.data != value) {  
 temp = temp.next;  
 }  
 if (temp.next != null) {  
 Node nodeToDelete = temp.next;  
 temp.next = temp.next.next;  
 // The nodeToDelete will be garbage collected  
 }  
 }  
  
 public void display() {  
 Node temp = head;  
 while (temp != null) {  
 System.*out*.print(temp.data + "->");  
 temp = temp.next;  
 }  
 System.*out*.println("NULL");  
 }  
  
 public static void main(String[] args) {  
 Task3 list = new Task3();  
  
 list.insertAtEnd(10);  
 list.insertAtEnd(20);  
 list.insertAtEnd(30);  
  
 System.*out*.print("Linked List: ");  
 list.display();  
  
 list.deleteByValue(20);  
  
 System.*out*.print("After Deleting 20: ");  
 list.display();  
 }  
}



Task4:

class Node {  
 Object data;  
 Node next;  
  
 public Node(Object data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
public class Task4 {  
 public Node head;  
 public Node tail; // Added a tail reference  
 public int size;  
  
 public Task4() {  
 this.head = null;  
 this.tail = null; // Initialize tail  
 this.size = 0;  
 }  
  
 public void insertAtEnd(Object value) {  
 Node newNode = new Node(value);  
 if (head == null) {  
 head = newNode;  
 tail = newNode; // If list is empty, new node is both head and tail  
 } else {  
 tail.next = newNode; // Link current tail to new node  
 tail = newNode; // Update tail to the new node  
 }  
 size++;  
 }  
  
 public void deleteByValue(Object value) {  
 if (head == null) {  
 System.*out*.println("List is empty. Cannot delete " + value);  
 return;  
 }  
  
 // Case 1: Deleting the head node  
 if (head.data == null) {  
 if (value == null) {  
 head = head.next;  
 size--;  
 if (head == null) { // If list becomes empty after head deletion  
 tail = null;  
 }  
 return;  
 }  
 } else if (head.data.equals(value)) {  
 head = head.next;  
 size--;  
 if (head == null) { // If list becomes empty after head deletion  
 tail = null;  
 }  
 return;  
 }  
  
 // Case 2: Deleting a node in the middle or at the end  
 Node current = head;  
 Node previous = null;  
  
 while (current != null) {  
 if (current.data == null) {  
 if (value == null) {  
 break;  
 }  
 } else if (current.data.equals(value)) {  
 break;  
 }  
 previous = current;  
 current = current.next;  
 }  
  
 if (current == null) {  
 System.*out*.println(value + " not found in the list.");  
 return;  
 }  
  
 if (previous != null) {  
 previous.next = current.next;  
 size--;  
 if (current == tail) { // If the deleted node was the tail  
 tail = previous; // Update tail to the previous node  
 }  
 }  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public Object get(int index) {  
 if (index < 0 || index >= size) {  
 throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + size);  
 }  
 Node current = head;  
 for (int i = 0; i < index; i++) {  
 current = current.next;  
 }  
 return current.data;  
 }  
  
  
 public void display() {  
 Node temp = head;  
 if (temp == null) {  
 System.*out*.println("List is empty.");  
 return;  
 }  
 while (temp != null) {  
 System.*out*.print(temp.data + "->");  
 temp = temp.next;  
 }  
 System.*out*.println("NULL");  
 }  
  
 public static void main(String[] args) {  
 Task4 list = new Task4();  
  
 System.*out*.println("--- Creating a Node and Adding Elements ---");  
 Node firstIndividualNode = new Node("Individual Node Data");  
 System.*out*.println("Created an individual node with data: " + firstIndividualNode.data);  
  
 list.insertAtEnd(10);  
 list.insertAtEnd("Hello");  
 list.insertAtEnd(4.5);  
 list.insertAtEnd(true);  
 list.insertAtEnd(null);  
  
 System.*out*.print("Linked List after insertions: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 System.*out*.println("**\n**--- Deleting Elements ---");  
 list.deleteByValue("Hello");  
 System.*out*.print("After deleting 'Hello': ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 list.deleteByValue(10);  
 System.*out*.print("After deleting 10: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 list.deleteByValue(null);  
 System.*out*.print("After deleting null: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
 list.deleteByValue(true); // Deleting the last remaining element from this sequence  
 System.*out*.print("After deleting true: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 list.deleteByValue(99);  
 System.*out*.print("After trying to delete 99: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 System.*out*.println("**\n**--- Demonstrating Casting and Access by Index (get method) ---");  
 list.insertAtEnd("First");  
 list.insertAtEnd("Second");  
 list.insertAtEnd("Third");  
 list.insertAtEnd(123.45);  
 System.*out*.print("List for get() demonstration: ");  
 list.display();  
 System.*out*.println("Current List Size: " + list.size());  
 System.*out*.println("Head Data: " + (list.head != null ? list.head.data : "N/A"));  
 System.*out*.println("Tail Data: " + (list.tail != null ? list.tail.data : "N/A"));  
  
  
 try {  
 System.*out*.println("Element at index 0: " + list.get(0));  
 System.*out*.println("Element at index 1: " + list.get(1));  
 System.*out*.println("Element at index " + (list.size() - 1) + ": " + list.get(list.size() - 1));  
  
 if (list.size() > 3) {  
 if (list.get(3) instanceof Double) {  
 Double value = (Double) list.get(3);  
 System.*out*.println("Element at index 3 (casted to Double): " + value);  
 } else {  
 System.*out*.println("Element at index 3 is not a Double: " + list.get(3));  
 }  
 }  
  
 } catch (IndexOutOfBoundsException e) {  
 System.*err*.println("Caught an error: " + e.getMessage());  
 }  
  
 System.*out*.println("**\n**--- Demonstrating Index Out Of Bounds ---");  
 try {  
 System.*out*.println("Trying to access element at index " + list.size() + ":");  
 list.get(list.size());  
 } catch (IndexOutOfBoundsException e) {  
 System.*err*.println("Caught Expected Error: " + e.getMessage());  
 }  
  
 try {  
 System.*out*.println("Trying to access element at index -1:");  
 list.get(-1);  
 } catch (IndexOutOfBoundsException e) {  
 System.*err*.println("Caught Expected Error: " + e.getMessage());  
 }  
 }  
}



Task5:

add(element), add(index, element)

get(int index)

set(int index, elemnt);

remove(Object o);

remove(int index)

size()

contains(Object o)

clear()

addFirst(E e)

addLast(E e)

removeFirst()

removeLast()

getFirst()

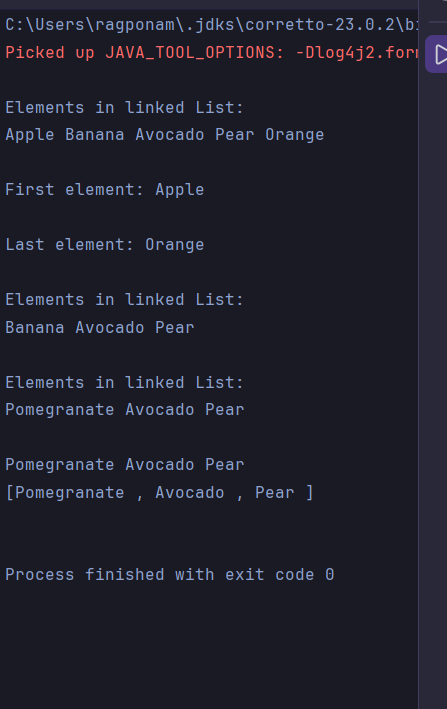
getLast()

peekFirst()

peekLast()

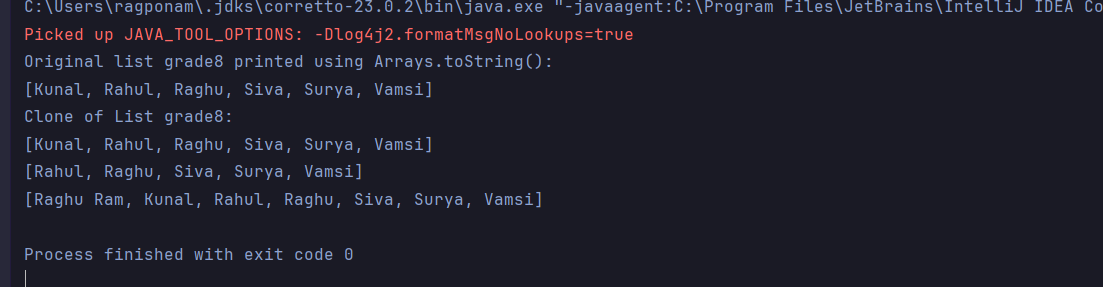
Task 6-10:

import java.util.LinkedList;  
import java.util.\*;  
  
public class Task6to10 {  
 public static void main(String[] args) {  
 LinkedList<String> fruits = new LinkedList<>();  
 fruits.add("Apple ");  
 fruits.add("Banana ");  
 fruits.add("Avocado ");  
 fruits.add("Pear ");  
 fruits.add("Orange ");  
 System.*out*.println("**\n**Elements in linked List: ");  
 for( String fruit:fruits){  
 System.*out*.print(fruit);  
 }  
 System.*out*.println();  
 System.*out*.println("**\n**First element: "+ fruits.getFirst());  
 System.*out*.println("**\n**Last element: "+ fruits.getLast());  
 fruits.removeFirst();  
 fruits.removeLast();  
 System.*out*.println("**\n**Elements in linked List: ");  
 for( String fruit:fruits){  
 System.*out*.print(fruit);  
 }  
 fruits.set(0, "Pomegranate ");  
  
 System.*out*.println("**\n\n**Elements in linked List: ");  
 for( String fruit:fruits){  
 System.*out*.print(fruit);  
 }  
 System.*out*.println("**\n**");  
 for (int i = 0;i< fruits.size() ; i++) {  
 System.*out*.print(fruits.get(i));  
 }  
 System.*out*.println();  
 System.*out*.println(fruits);  
 System.*out*.println();  
  
  
 }  
}



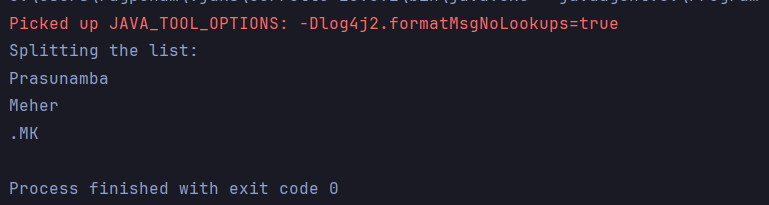
Task 11-13:

import java.util.Arrays;  
import java.util.LinkedList;  
  
public class Task11to13 {  
  
 public static void main(String[] args) {  
 LinkedList<String> grade8 = new LinkedList<>();  
 grade8.add("Kunal");  
 grade8.add("Rahul");  
 grade8.add("Raghu");  
 grade8.add("Siva");  
 grade8.add("Surya");  
 grade8.add("Vamsi");  
 System.*out*.println("Original list grade8 printed using Arrays.toString(): ");  
 Object[] arr = grade8.toArray();System.*out*.println(Arrays.*toString*(arr));  
 LinkedList<String> grade8clone = (LinkedList<String>) grade8.clone();  
 System.*out*.println("Clone of List grade8: **\n**"+grade8clone);  
 grade8clone.pop();  
 System.*out*.println(grade8clone);  
 grade8.push("Raghu Ram");  
 System.*out*.println(grade8);  
 }  
}



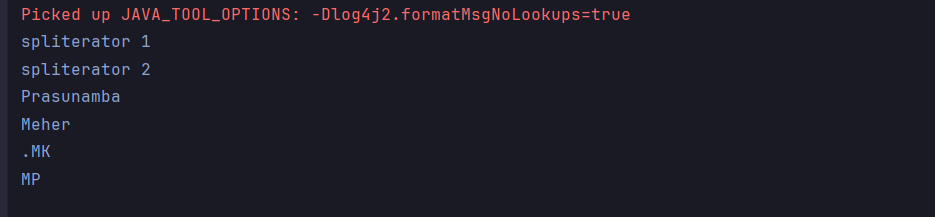
Task14:

import java.util.\*;  
  
public class Task14 {  
 public static void main(String[] args) {  
 LinkedList<String> lobj = new LinkedList<>();  
 lobj.add("Prasunamba");  
 lobj.add("Meher");  
 lobj.add(".MK");  
 Spliterator<String> sitobj = lobj.spliterator();  
 //forEachRemaining is a method of Spliterator  
 System.*out*.println("Splitting the list:");  
 sitobj.forEachRemaining(System.*out*::println);  
 }  
}



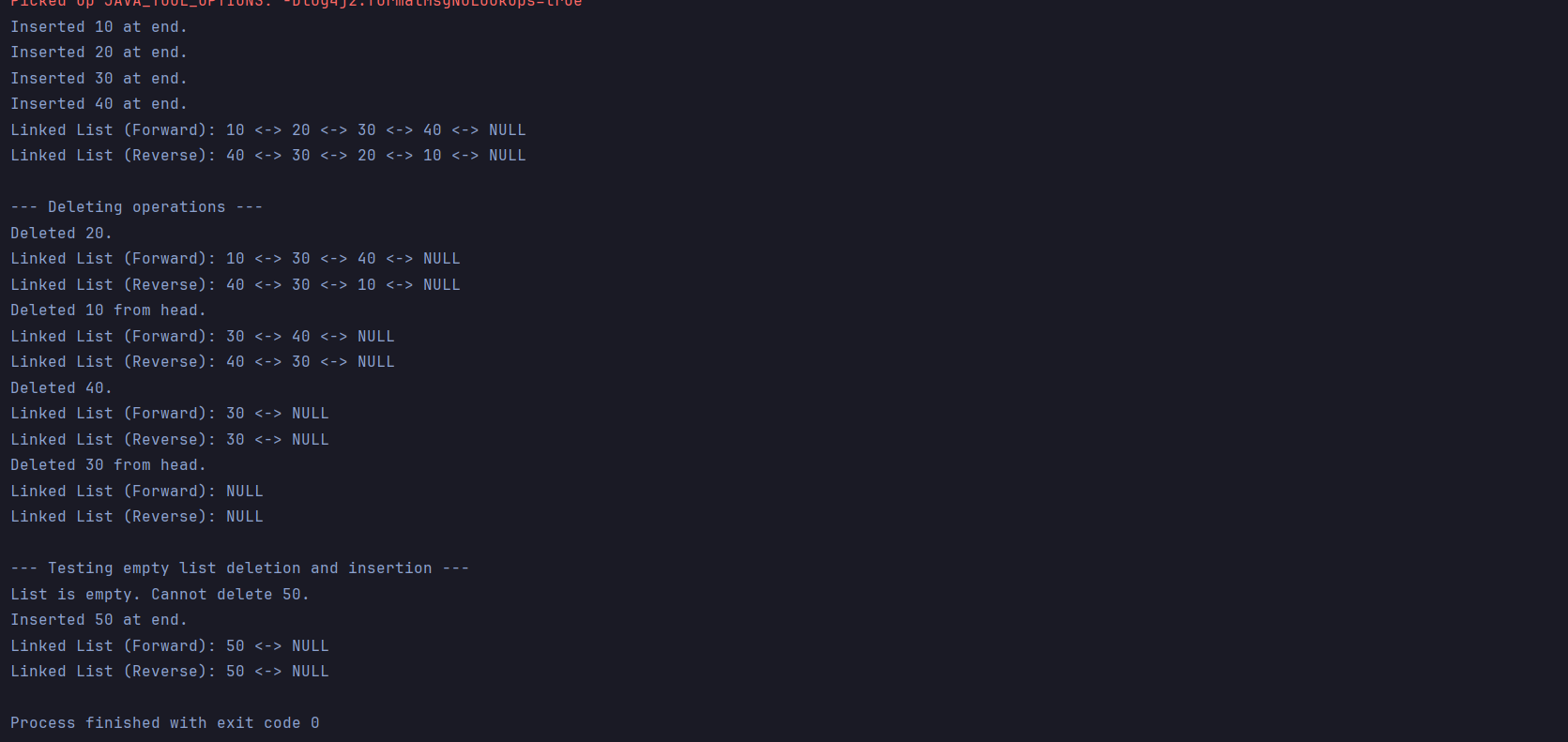
Task15:

import java.util.LinkedList;  
import java.util.Spliterator;  
  
public class Task15 {  
 public static void main(String[] args) {  
  
 LinkedList<String> llobj = new LinkedList<String>();  
 llobj.add("Prasunamba");  
 llobj.add("Meher");  
 llobj.add(".MK");  
 llobj.add("MP");  
  
 Spliterator<String> itobj1 = llobj.spliterator();  
 Spliterator<String> itobj2 = itobj1.trySplit();  
  
 System.*out*.println("spliterator 1");  
 while( itobj1.tryAdvance( (n) -> { System.*out*.println(n); } ) );  
  
 System.*out*.println("spliterator 2");  
 while( itobj2.tryAdvance( (n) -> { System.*out*.println(n); } ) );  
 }  
}



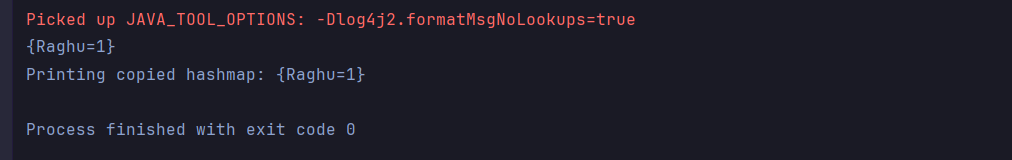
Task16:

public class Task16 {  
  
 class Node {  
 int data;  
 Node next;  
 Node prev;  
  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 this.prev = null;  
 }  
 }  
  
 private Node head;  
 private Node tail;  
  
 public Task16() {  
 this.head = null;  
 this.tail = null;  
 }  
  
 public void insertAtEnd(int value) {  
 Node newNode = new Node(value);  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 } else {  
 tail.next = newNode;  
 newNode.prev = tail;  
 tail = newNode;  
 }  
 System.*out*.println("Inserted " + value + " at end.");  
 }  
  
 public void deleteByValue(int value) {  
 if (head == null) {  
 System.*out*.println("List is empty. Cannot delete " + value + ".");  
 return;  
 }  
  
 Node current = head;  
  
 if (head.data == value) {  
 head = head.next;  
 if (head != null) {  
 head.prev = null;  
 } else {  
 tail = null;  
 }  
 System.*out*.println("Deleted " + value + " from head.");  
 return;  
 }  
  
 while (current != null && current.data != value) {  
 current = current.next;  
 }  
  
 if (current == null) {  
 System.*out*.println(value + " not found in the list.");  
 return;  
 }  
  
 if (current.prev != null) {  
 current.prev.next = current.next;  
 }  
  
 if (current.next != null) {  
 current.next.prev = current.prev;  
 }  
  
 if (current == tail) {  
 tail = current.prev;  
 }  
 System.*out*.println("Deleted " + value + ".");  
 }  
  
 public void display() {  
 Node temp = head;  
 System.*out*.print("Linked List (Forward): ");  
 while (temp != null) {  
 System.*out*.print(temp.data + " <-> ");  
 temp = temp.next;  
 }  
 System.*out*.println("NULL");  
 }  
  
 public void displayReverse() {  
 Node temp = tail;  
 System.*out*.print("Linked List (Reverse): ");  
 while (temp != null) {  
 System.*out*.print(temp.data + " <-> ");  
 temp = temp.prev;  
 }  
 System.*out*.println("NULL");  
 }  
  
  
 public static void main(String[] args) {  
 Task16 list = new Task16();  
  
 list.insertAtEnd(10);  
 list.insertAtEnd(20);  
 list.insertAtEnd(30);  
 list.insertAtEnd(40);  
  
 list.display();  
 list.displayReverse();  
  
 System.*out*.println("**\n**--- Deleting operations ---");  
 list.deleteByValue(20);  
 list.display();  
 list.displayReverse();  
  
 list.deleteByValue(10);  
 list.display();  
 list.displayReverse();  
  
 list.deleteByValue(40);  
 list.display();  
 list.displayReverse();  
  
 list.deleteByValue(30);  
 list.display();  
 list.displayReverse();  
  
 System.*out*.println("**\n**--- Testing empty list deletion and insertion ---");  
 list.deleteByValue(50);  
 list.insertAtEnd(50);  
 list.display();  
 list.displayReverse();  
 }  
}



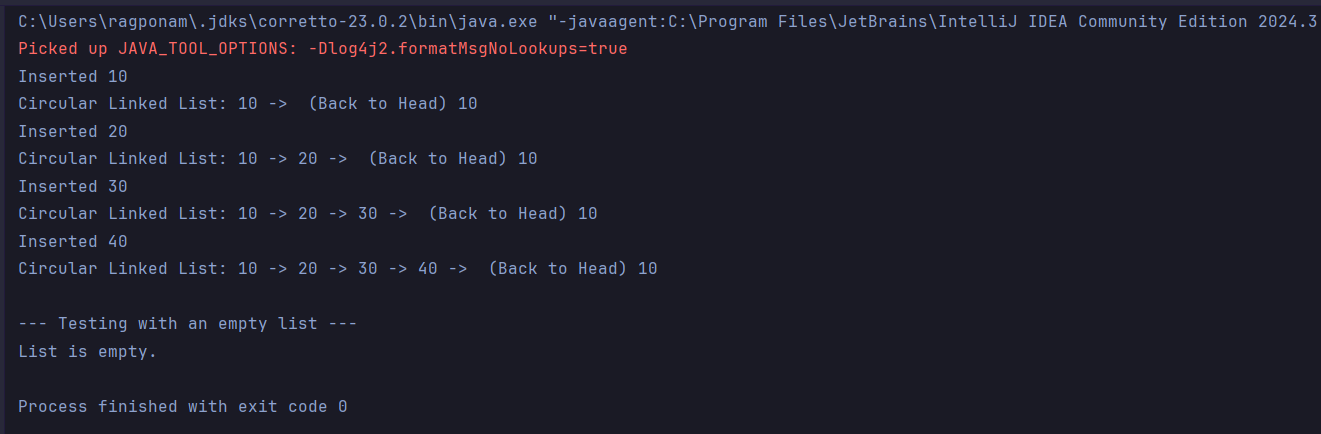
Task 17-19:

import java.util.HashMap;  
  
public class Task17to19 {  
 public static void main(String[] args) {  
  
  
 HashMap<String, Integer> hm1 = new HashMap<>(10, 0.75f);  
 hm1.put("Raghu", 1);  
 System.*out*.println(hm1);  
 HashMap<String, Integer>hm2 = new HashMap<String, Integer>(hm1);  
 System.*out*.println("Printing copied hashmap: "+hm2);  
 }  
}



Task20:

public class Task21 {  
  
 private static class Node {  
 int data;  
 Node next;  
  
 public Node(int value) {  
 this.data = value;  
 this.next = null;  
 }  
 }  
  
 private Node head;  
  
 public Task21() {  
 this.head = null;  
 }  
  
 public void insertAtEnd(int value) {  
 Node newNode = new Node(value);  
  
 if (head == null) {  
 head = newNode;  
 newNode.next = head;  
 } else {  
 Node temp = head;  
 while (temp.next != head) {  
 temp = temp.next;  
 }  
 temp.next = newNode;  
 newNode.next = head;  
 }  
 System.*out*.println("Inserted " + value);  
 }  
  
 public void display() {  
 if (head == null) {  
 System.*out*.println("List is empty.");  
 return;  
 }  
  
 Node current = head;  
 System.*out*.print("Circular Linked List: ");  
 do {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 } while (current != head);  
  
 System.*out*.println(" (Back to Head) " + current.data);  
 }  
  
 public static void main(String[] args) {  
 Task21 myCircularList = new Task21();  
  
 myCircularList.insertAtEnd(10);  
 myCircularList.display();  
  
 myCircularList.insertAtEnd(20);  
 myCircularList.display();  
  
 myCircularList.insertAtEnd(30);  
 myCircularList.display();  
  
 myCircularList.insertAtEnd(40);  
 myCircularList.display();  
  
 System.*out*.println("**\n**--- Testing with an empty list ---");  
 Task21 emptyList = new Task21();  
 emptyList.display();  
 }  
}



Home tasks:

**Advantages of Linked Lists.**

* Linked Lists can have a dynamic size.
* Faster insertions and deletions.
* Efficient memory utilization.
* Can easily be used to implement abstract data types like queues or stacks.

**Disadvantages of Linked Lists.**

* Each node has reference to other node along with the data, which results in more memory consumption.
* Random access is impossible without traversing the whole List.
* Complexity of implementation. (Reference to other nodes.)

**Applications of Linked Lists:**

* Implement stacks and queues.
* Used in Dynamic memory allocation
* Image viewers and music players where next and previous buttons are used.
* Web browser’s back and forward functionality.
* Separate chaining in hash tables.
* Adjacency lists in graphs.
* Process scheduling in Operating systems.