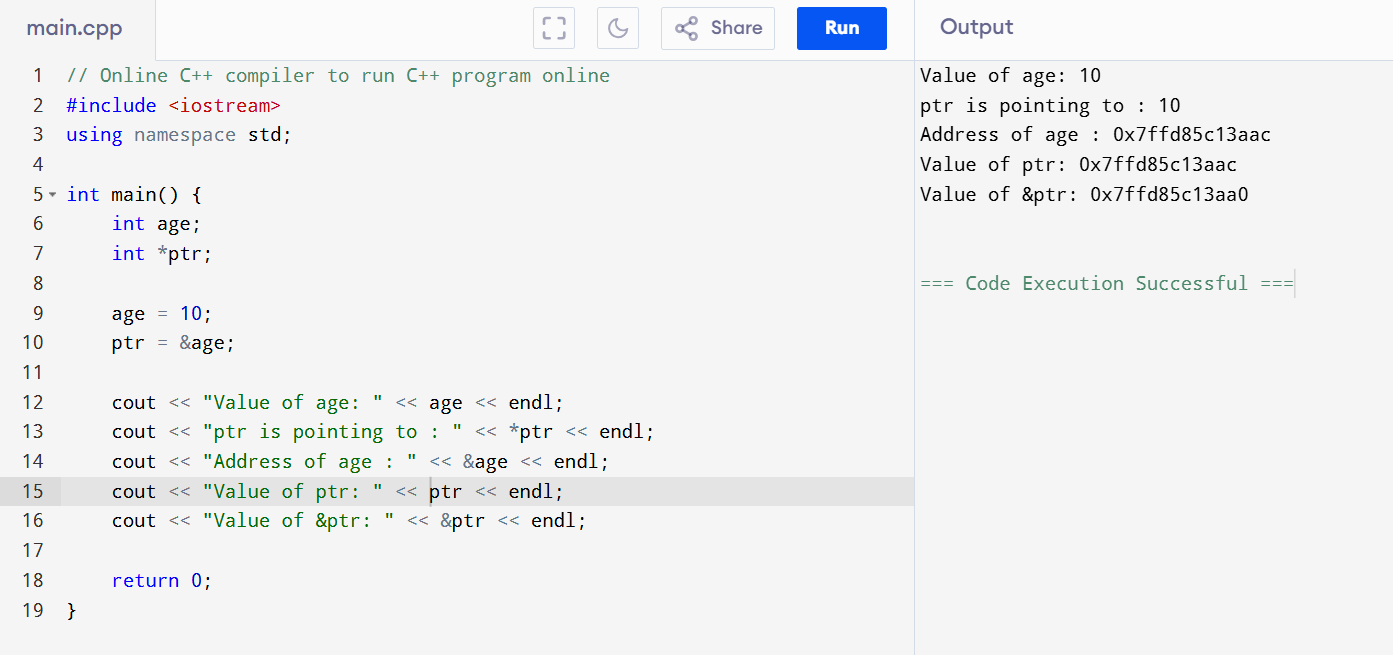
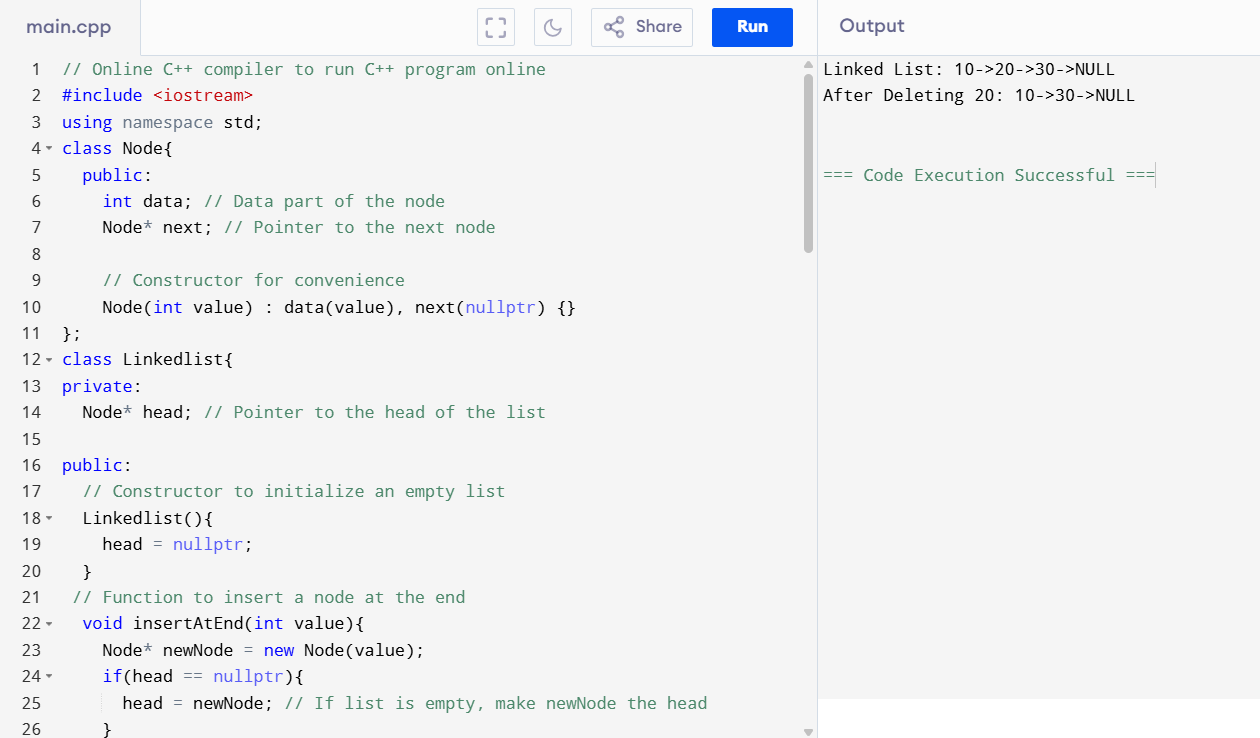
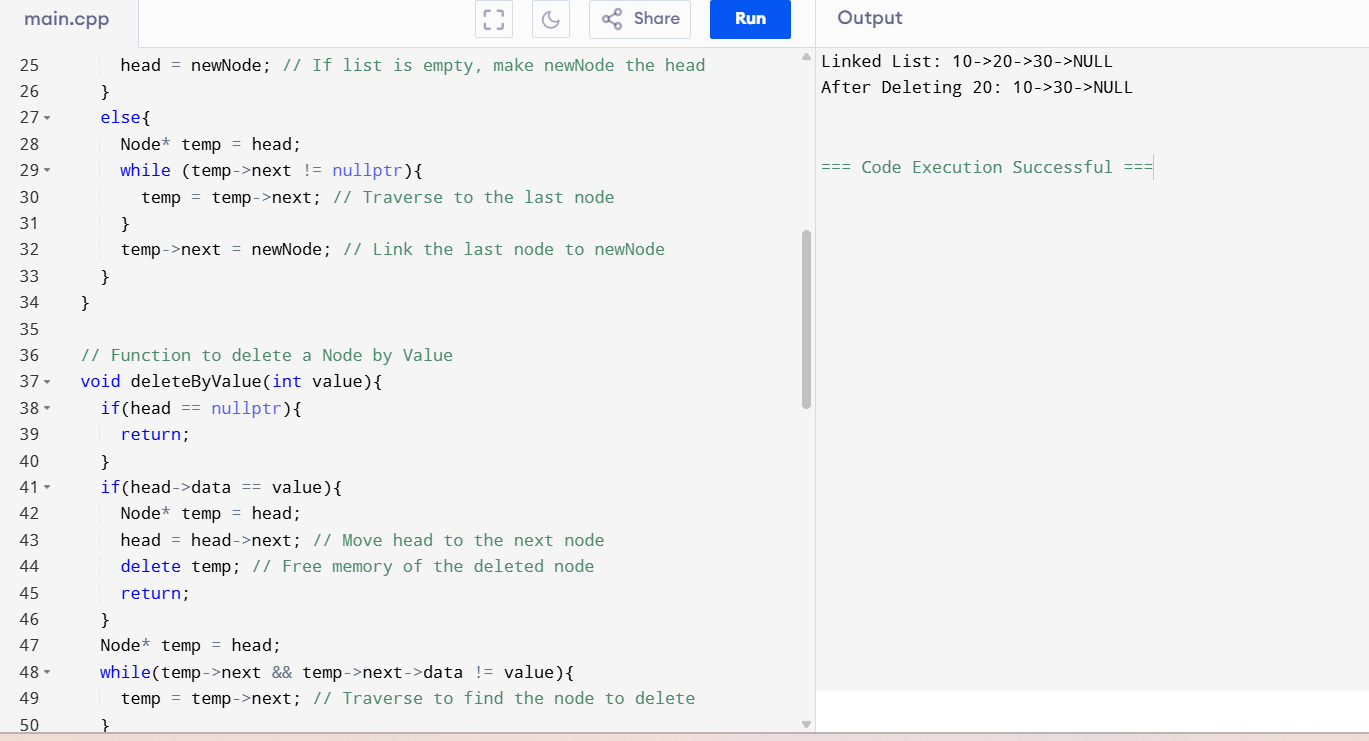
**Day 13 28th June**

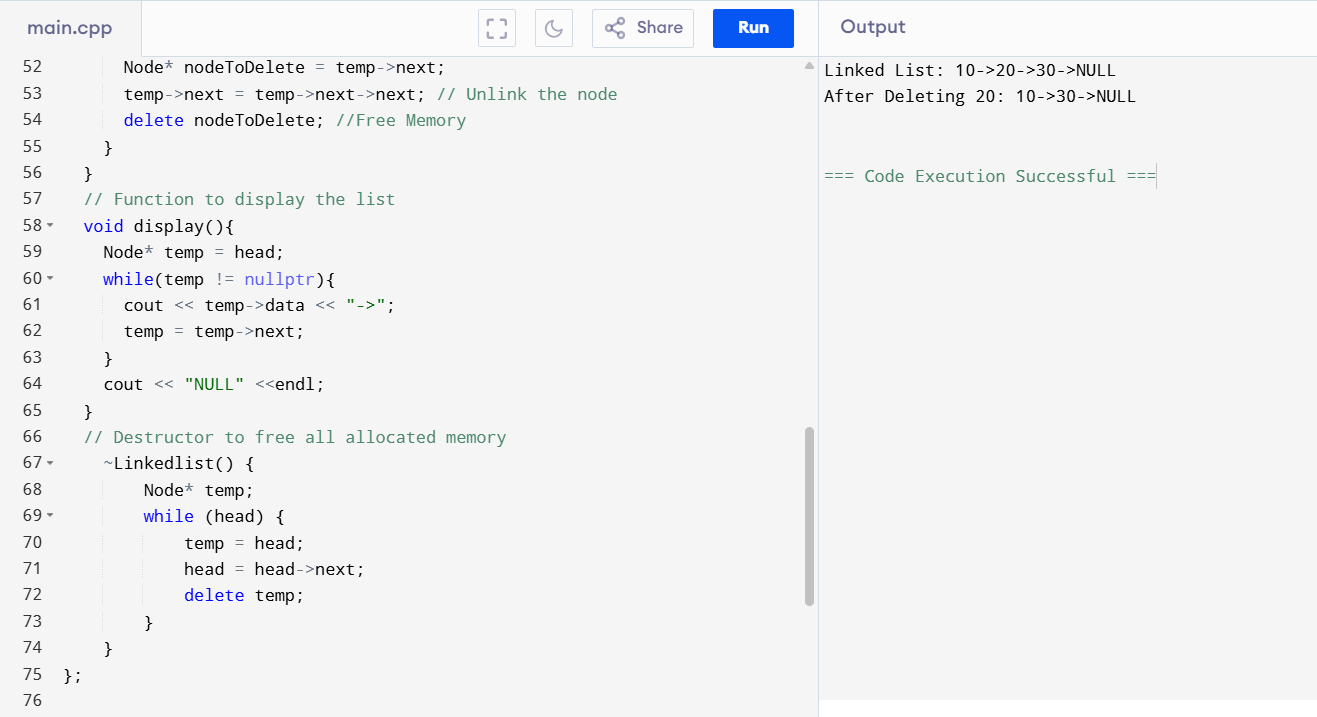
**Task 1:** Pointer in c or c++

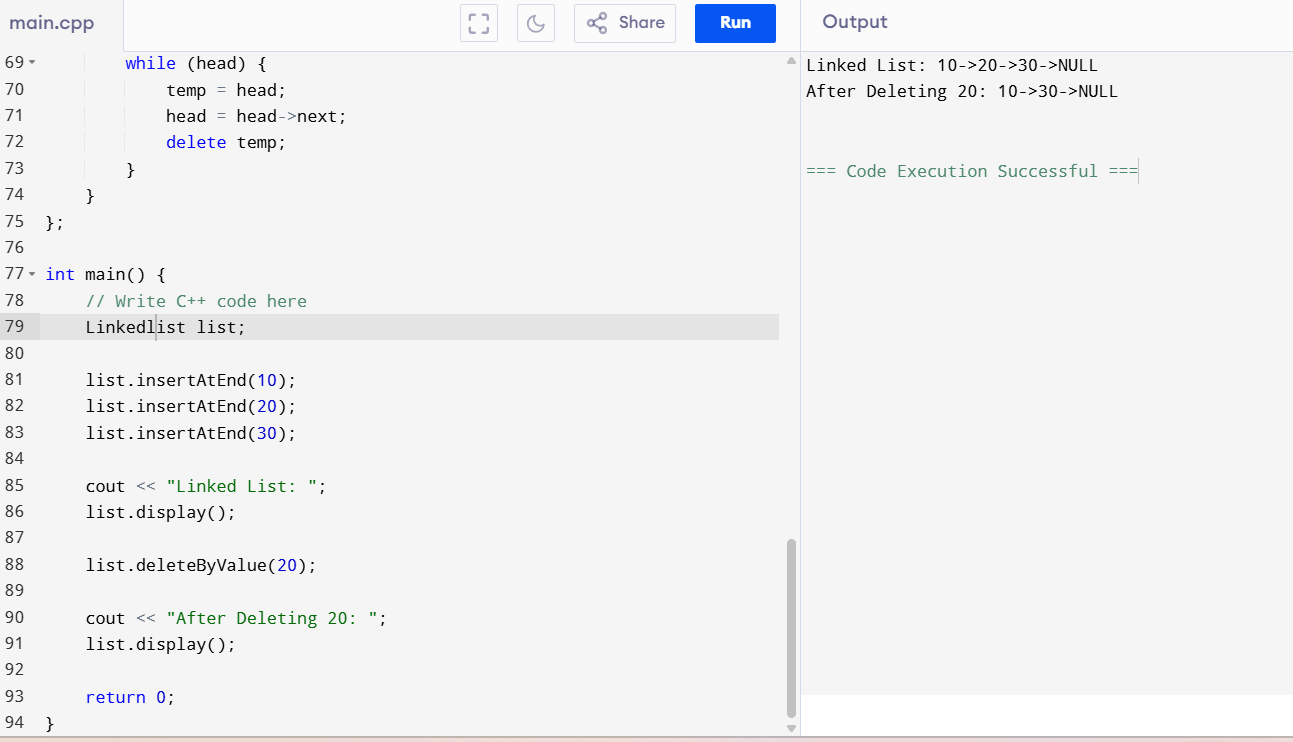


**Task 2**: Linked list in c++

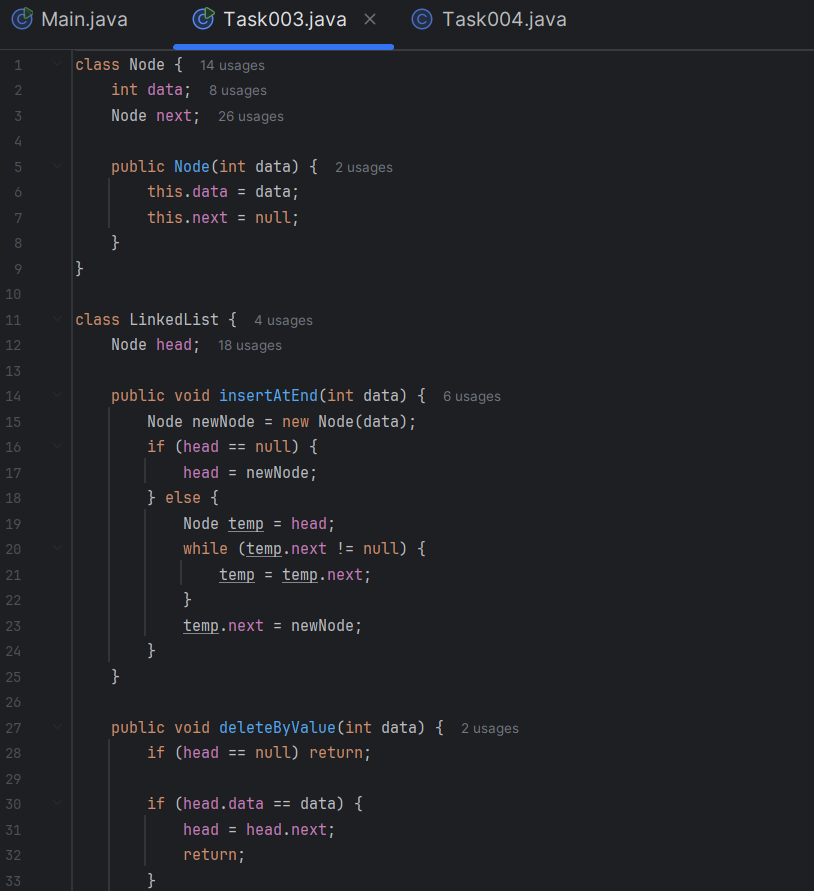


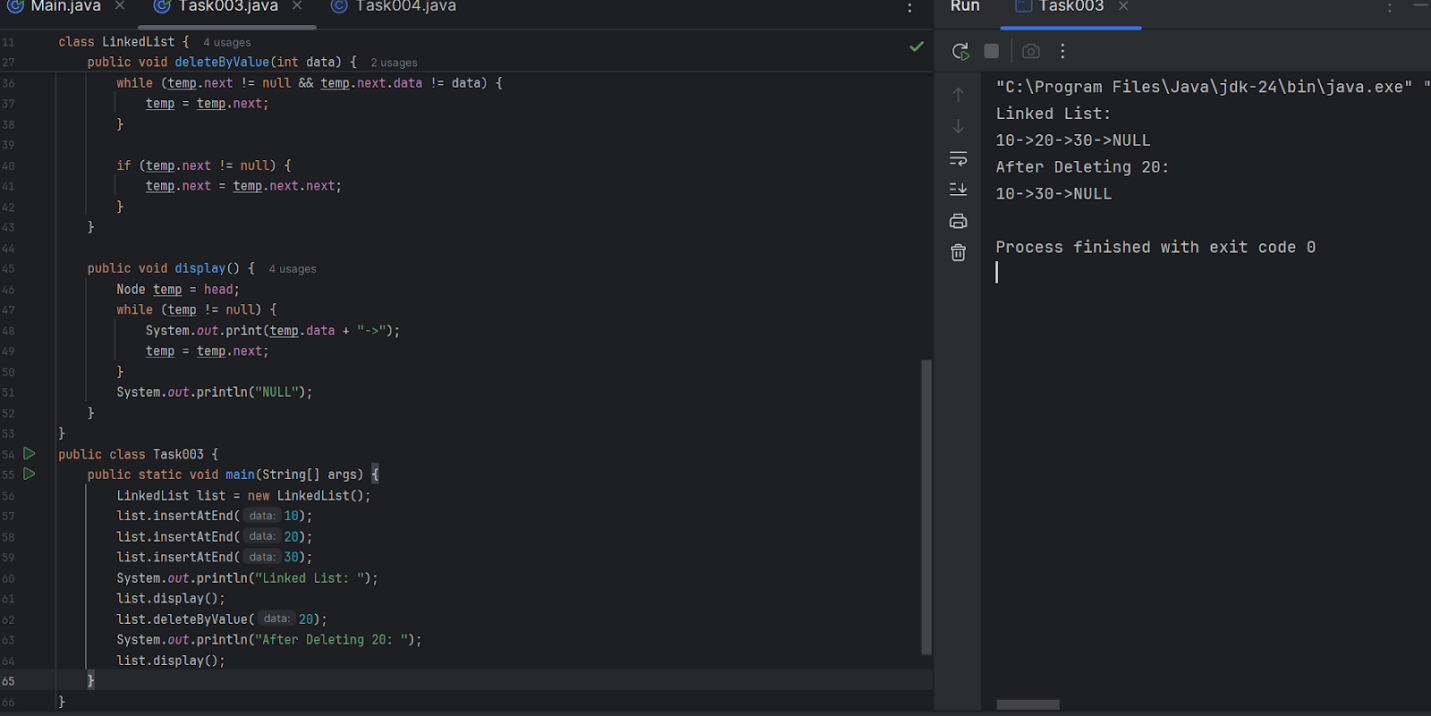






**Task 3:** Use the above code to create  a Java code which creates  a linked list.





**Task 4:** Try to create a node and add a value to it.. Which can take any kind of data in the Node..

class Exception1 extends Exception {

   public Exception1(String message) {

       super(message);

   }

}

class Node3 {

   Object data;

   Node3 next;

   public Node3(Object ch) {

       data = ch;

       next = null;

   }

}

class Linkedlist2 {

   Node3 head;

   int count1 = 0;

   public void insert(Object ch) throws Exception1 {

       if (count1 >= 4) {

           throw new Exception1("Error: Index out of bounds");

       }

       Node3 newNode = new Node3(ch);

       if (head == null)

           head = newNode;

       else {

           Node3 temp1 = head;

           while (temp1.next != null) {

               temp1 = temp1.next;

           }

           temp1.next = newNode;

       }

       count1++;

   }

   public void display() {

       Node3 temp1 = head;

       while (temp1 != null) {

           System.*out*.print(temp1.data + " -> ");

           temp1 = temp1.next;

       }

       System.*out*.print(" null");

   }

   public void delete(String ch){

       Node3 temp=head;

       if(temp.data==ch) {

           head=temp.next;

           temp.next=null;

           return;

       }

       while(temp!=null)

       {

           if(temp.next!=null && temp.next.data==ch)

           {

               temp.next=temp.next.next;

               return;

           }

           temp=temp.next;

       }

   }

   public void size() {

       int count = 0;

       Node3 temp1 = head;

       while (temp1 != null) {

           count++;

           temp1 = temp1.next;

       }

       System.*out*.println(" ");

       System.*out*.println("Size = " + count);

   }

}

public class Task004 {

   public static void main(String[] args) {

       Linkedlist2 tt = new Linkedlist2();

       try {

           tt.insert("alice");

           tt.insert(1);

           tt.insert("sukhwinder");

           tt.insert('c');

           tt.insert("hello");    // This will throw Exception1

           tt.display();

           tt.size();

           System.*out*.println("\nAfter deleting 'sukhwinder':");

           tt.delete("sukhwinder");

           tt.display();

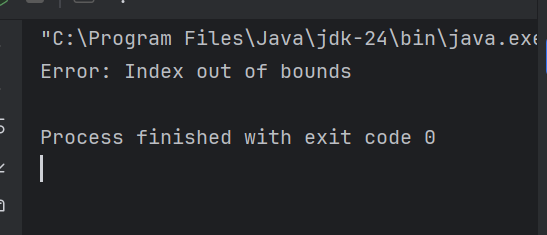
           tt.size();} catch (Exception1 e) {

           System.*out*.println("Error: Index out of bounds");

       }

   }

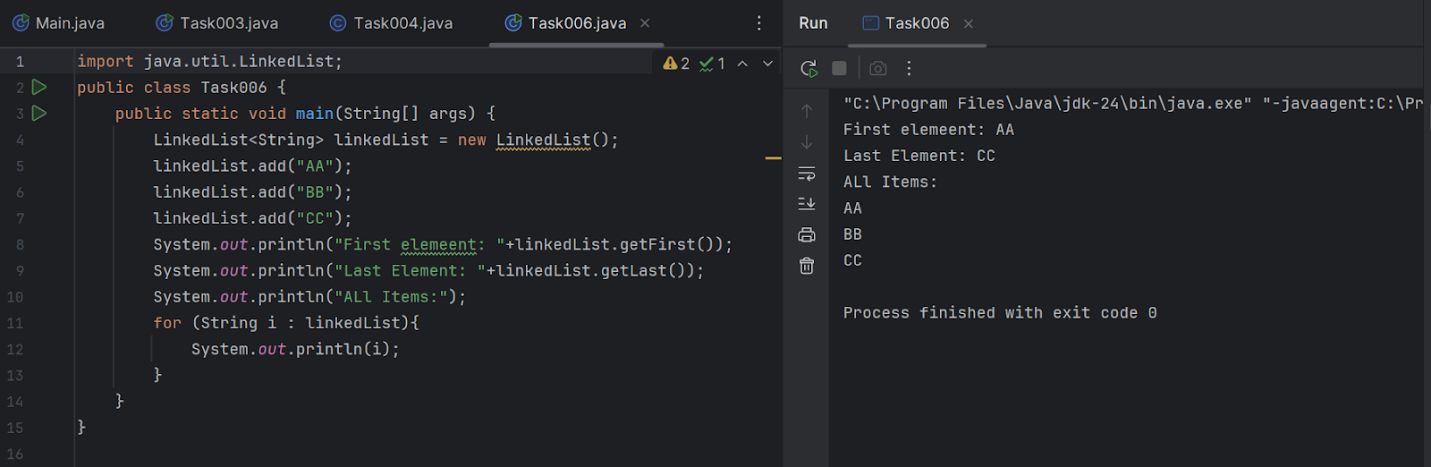
}



**Task 5:** List down all the methods of Linked list

1. Adding elements
   1. add(element)
   2. add(index, element)
   3. addFirst(element)
   4. addLast(element)
   5. addAll(collection)
   6. addAll(index, collection)
2. Removing elements
   1. remove(index)
   2. remove(element)
   3. removeFirst()
   4. removeLast()
   5. clear()
3. Accessing elements
   1. get(index)
   2. getFirst()
   3. getLast()
   4. indexof(element)
4. Other methods
   1. size()
   2. set(index, element)
   3. clone

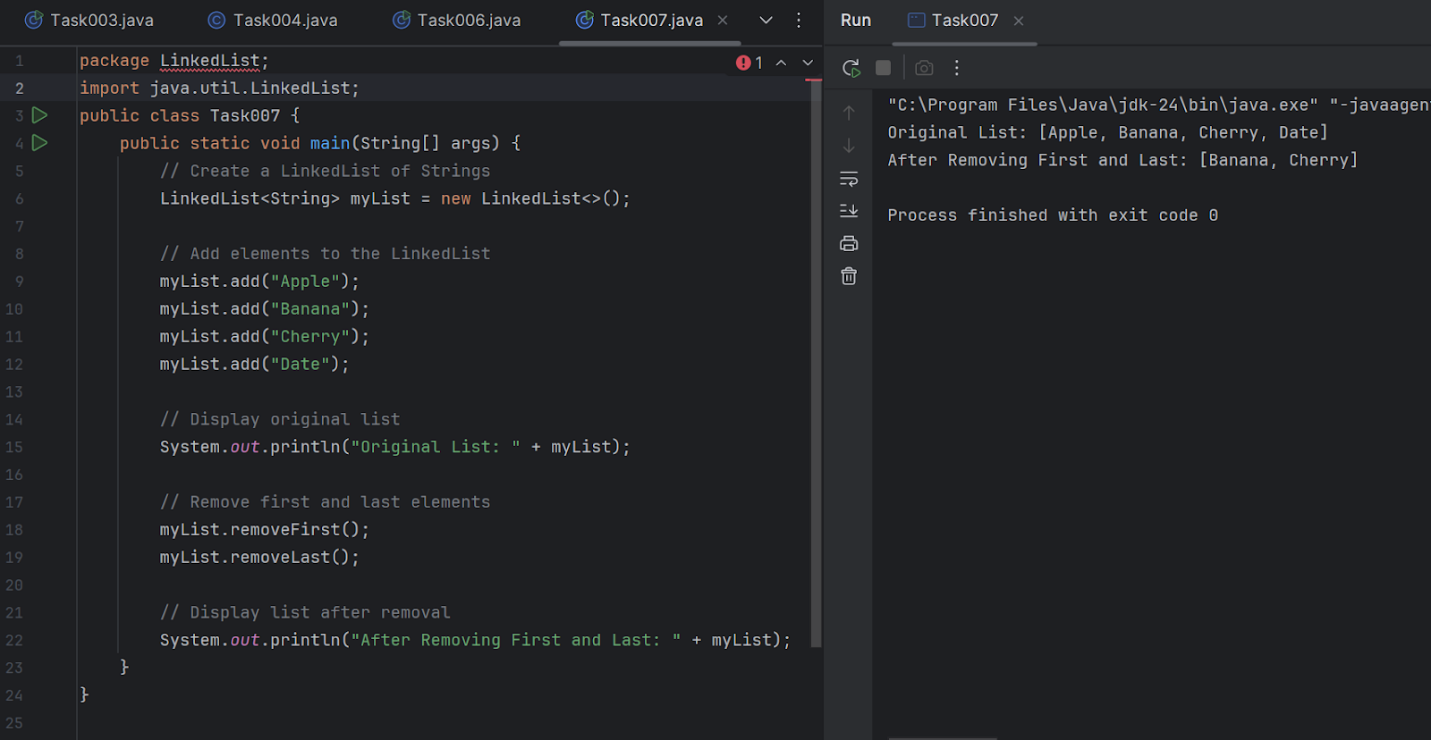
**Task 06:** Create linked list using Pre defined class and add elements to it.



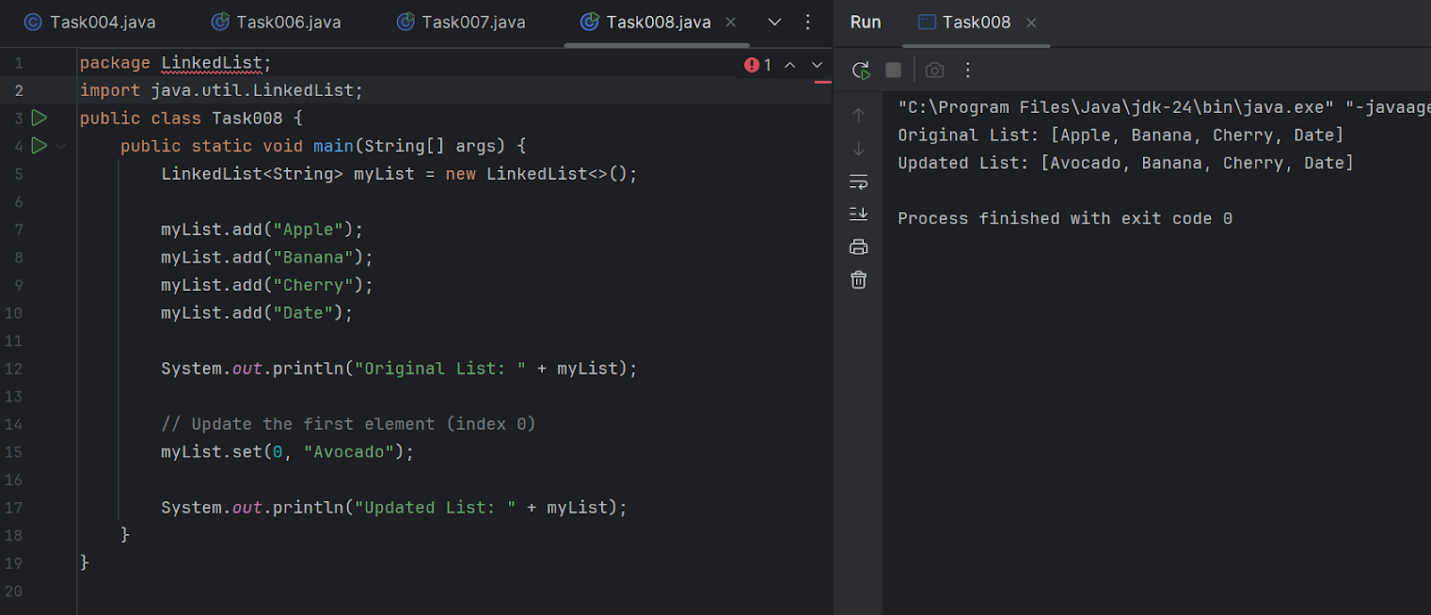
Hint:

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

**Task 7:** remove first and remove last element and display all elements in the linked list

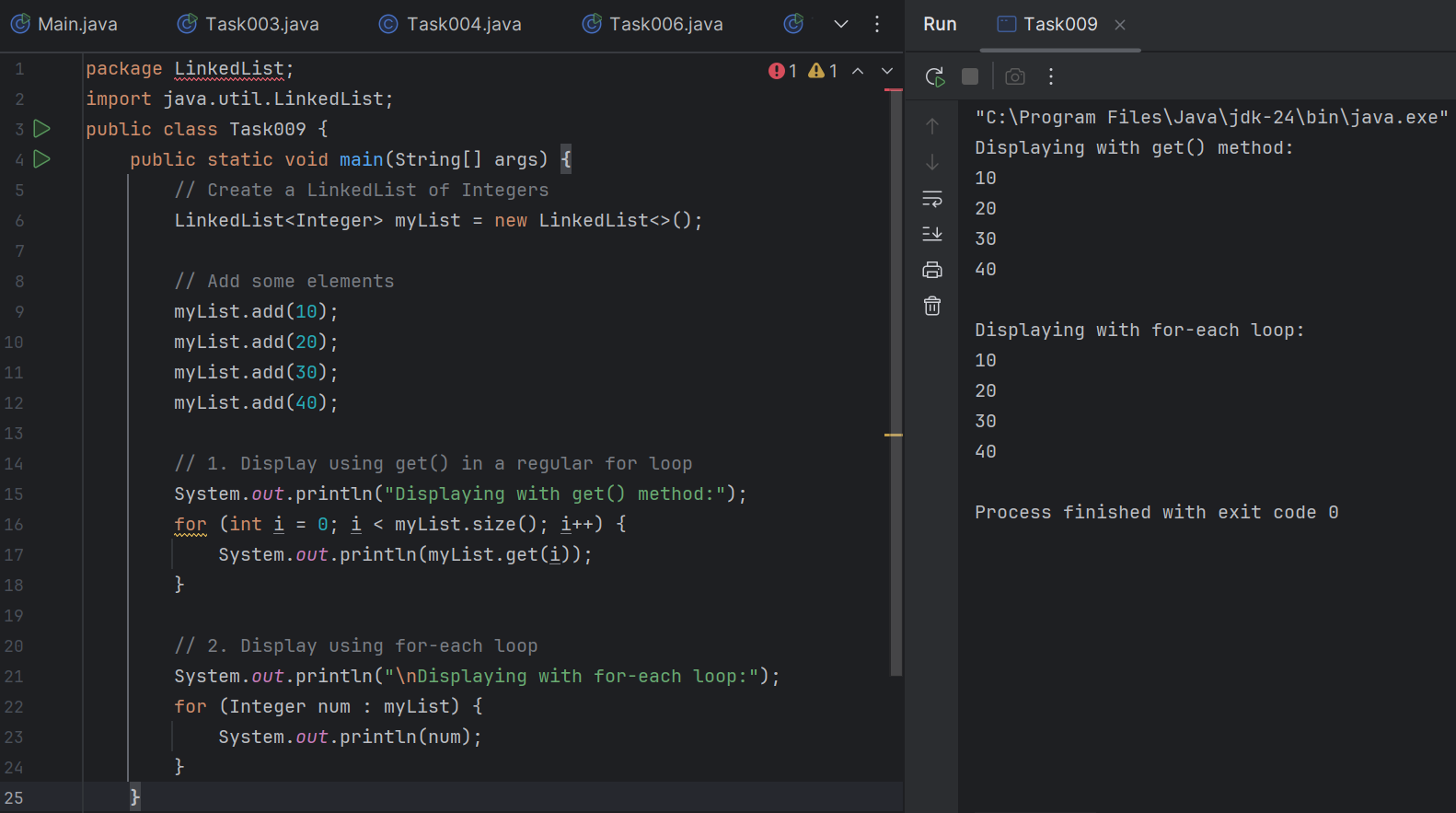


**Task 8:** in the list update the 1st element to a new value

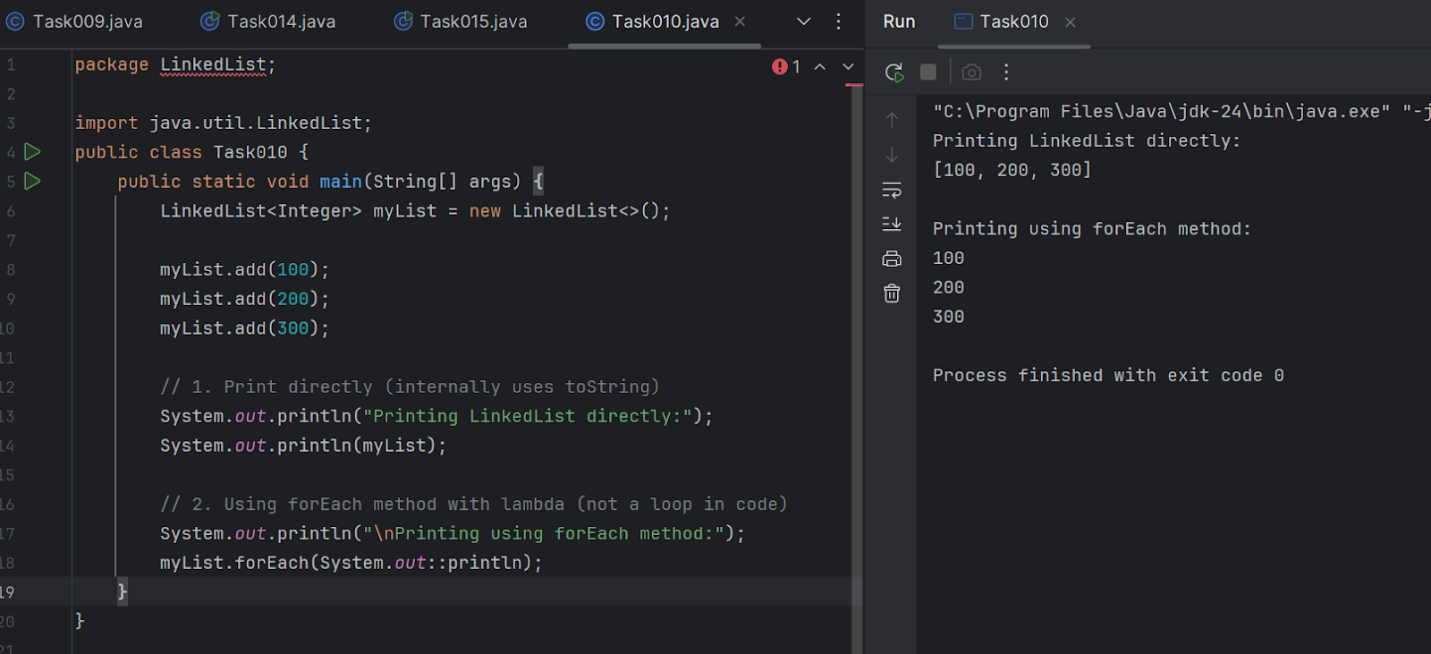


Hint: use set(1, "new value");

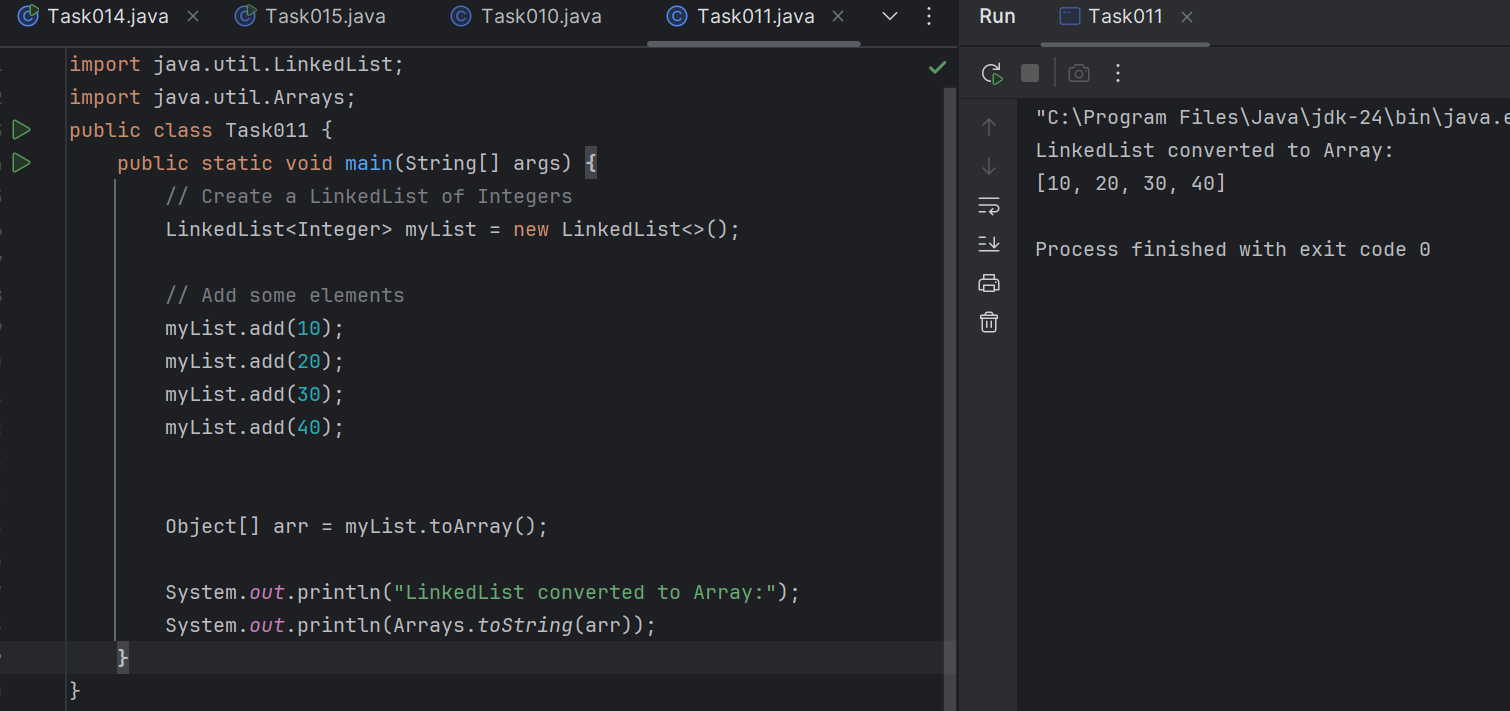
**Task 9:** Task 9: display the list twice 1..... with get method in for loop and 2 ... for each loop



**Task 10:** display the elements of the linked list with out loops

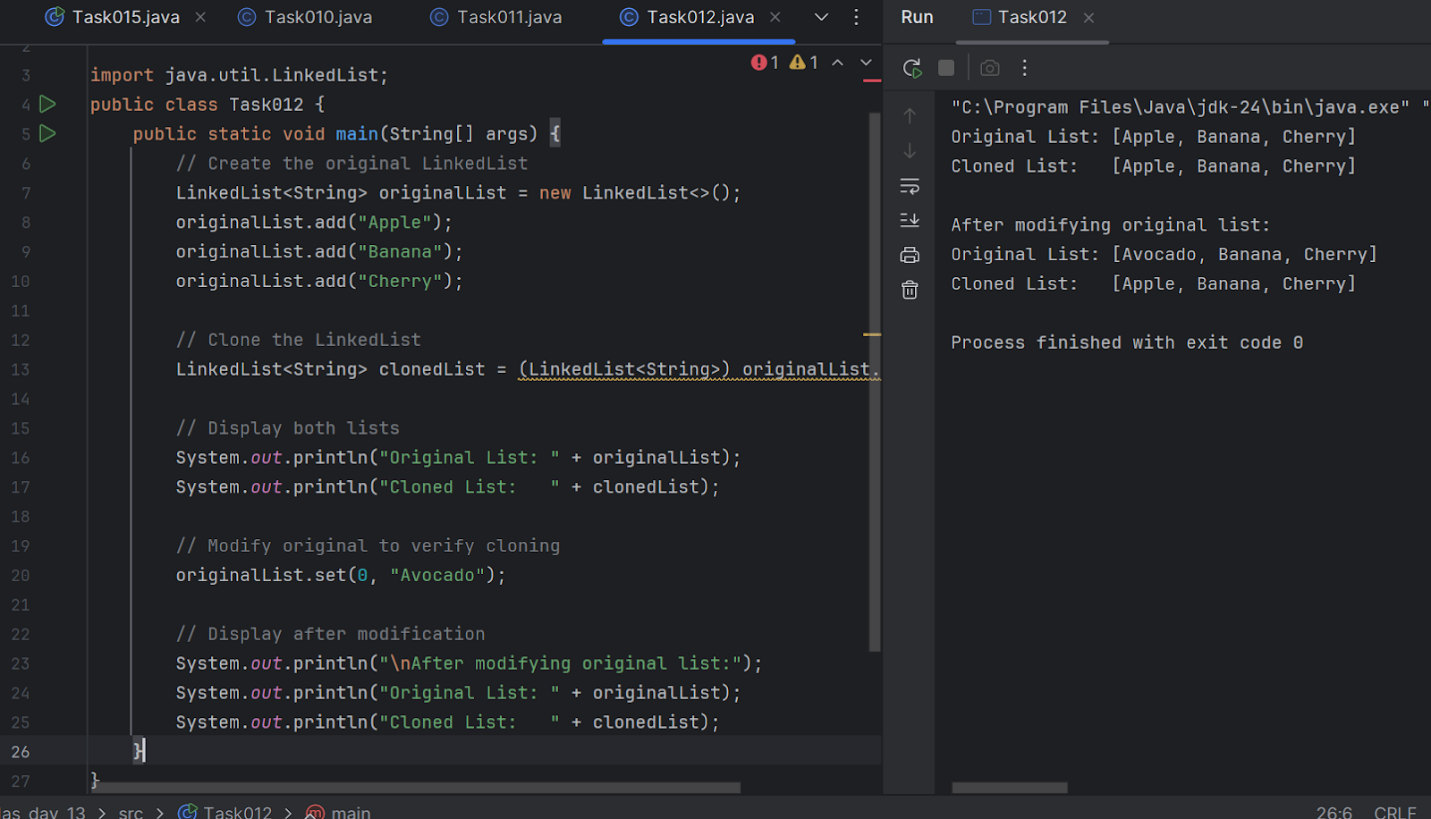


**Task 11**: convert the linked list to an array and display

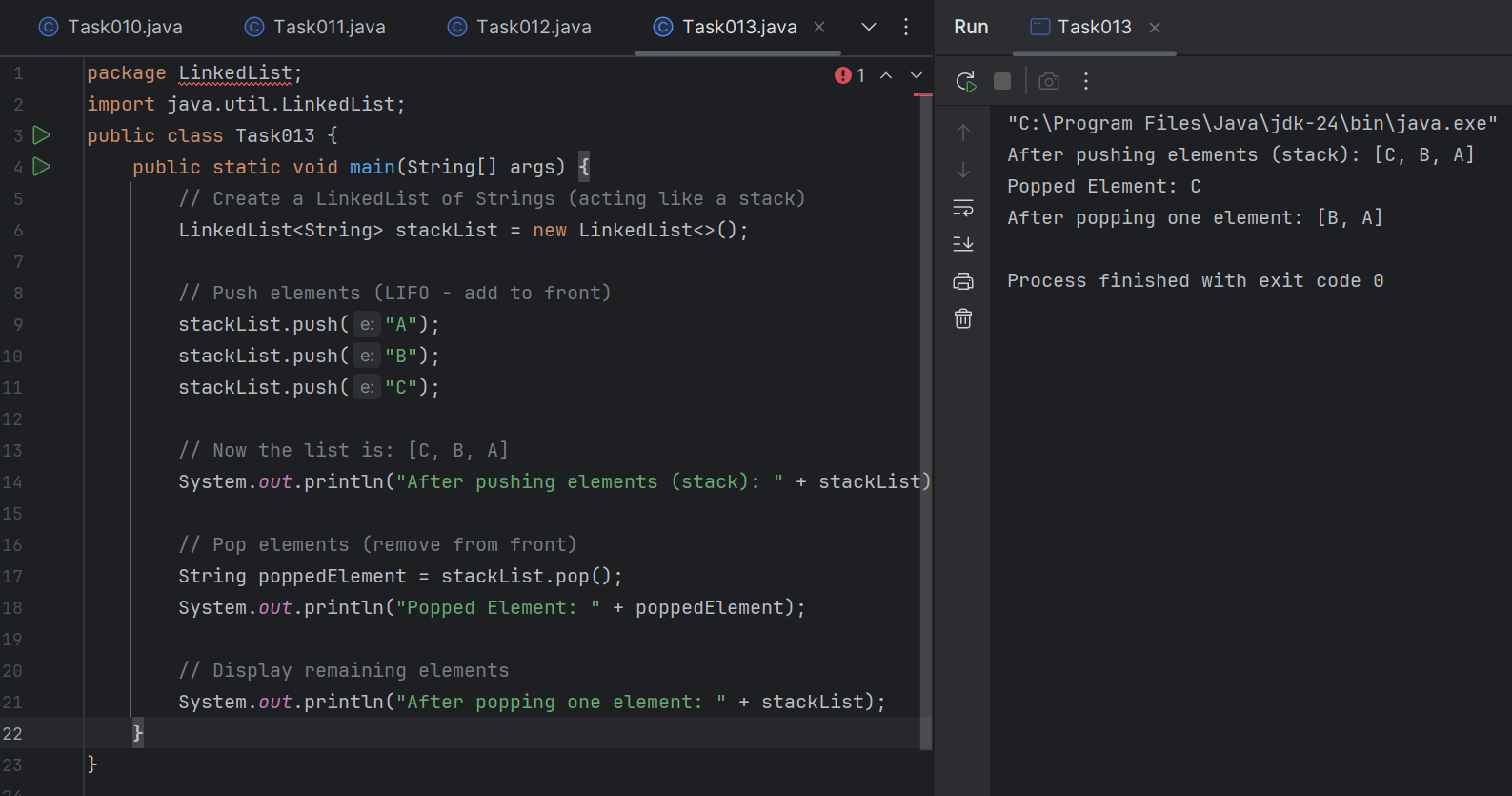


Hint : Object[] a = llobj.toArray();

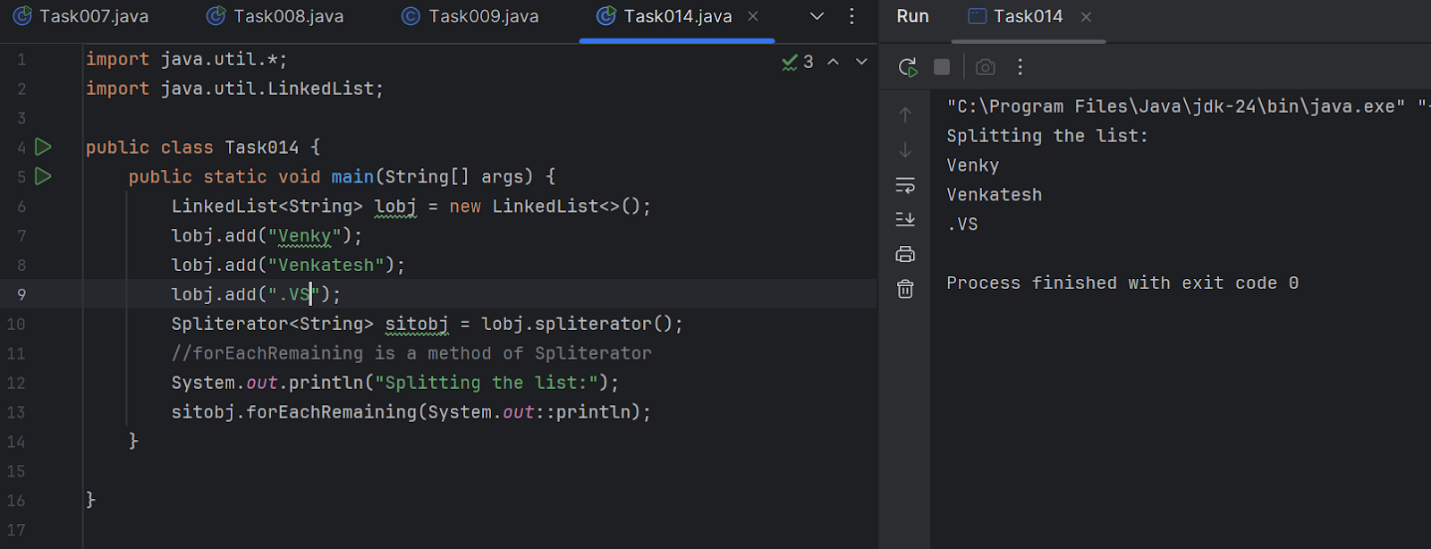
**Task 12**: clone the linked list to check if its getting cloned?



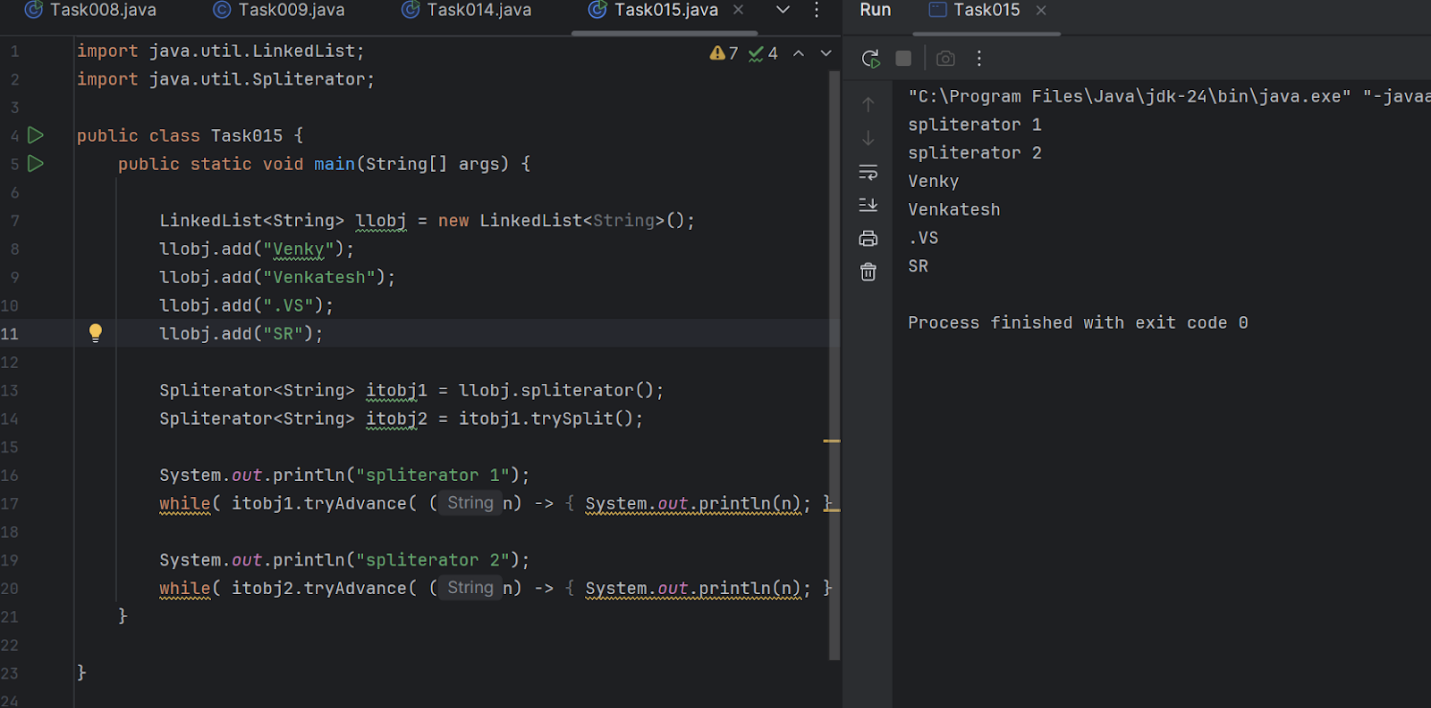
**Task 13:** Use pop and push methods on linked list.. LIFO – just follow..



**Task 14:** Splititerator



**Task 15**: tryAdvance()



**Task 16**: Create a doubly linked list..

public class Task016 {

   // Node class for Doubly Linked List

   static class Node {

       int data;

       Node prev;

       Node next;

       Node(int data) {

           this.data = data;

           this.prev = null;

           this.next = null;

       }

   }

   // Head and tail references

   Node head = null;

   Node tail = null;

   // Method to add elements to the end of the list

   public void add(int data) {

       Node newNode = new Node(data);

       if (head == null) {

           head = tail = newNode;

       } else {

           tail.next = newNode;   // Link last node to new node

           newNode.prev = tail;   // Link new node back to last node

           tail = newNode;        // Move tail to new node

       }

   }

   // Method to display the list forward

   public void displayForward() {

       Node current = head;

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

       while (current != null) {

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

           current = current.next;

       }

       System.*out*.println();

   }

   // Method to display the list backward

   public void displayBackward() {

       Node current = tail;

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

       while (current != null) {

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

           current = current.prev;

       }

       System.*out*.println();

   }

   // Main method to run

   public static void main(String[] args) {

       Task016 dll = new Task016();

       // Add some nodes

       dll.add(10);

       dll.add(20);

       dll.add(30);

       dll.add(40);

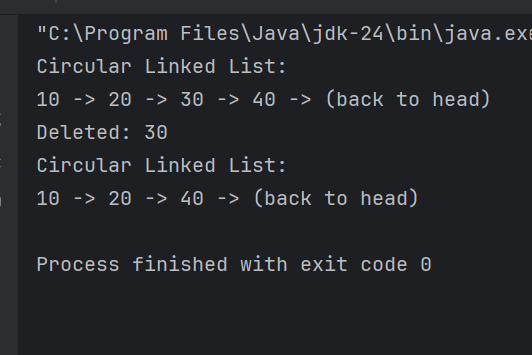
       // Display forward and backward

       dll.displayForward();

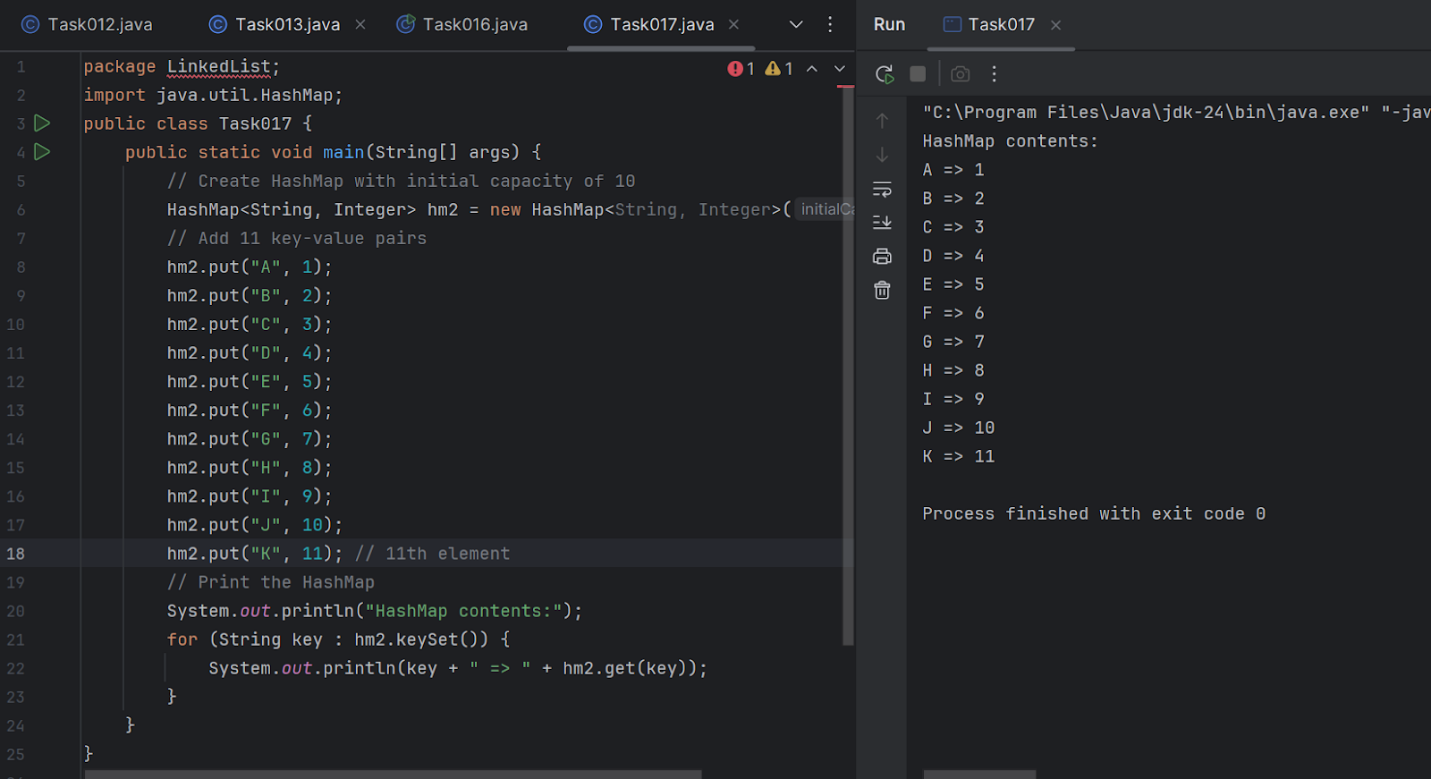
       dll.displayBackward();

   }

}

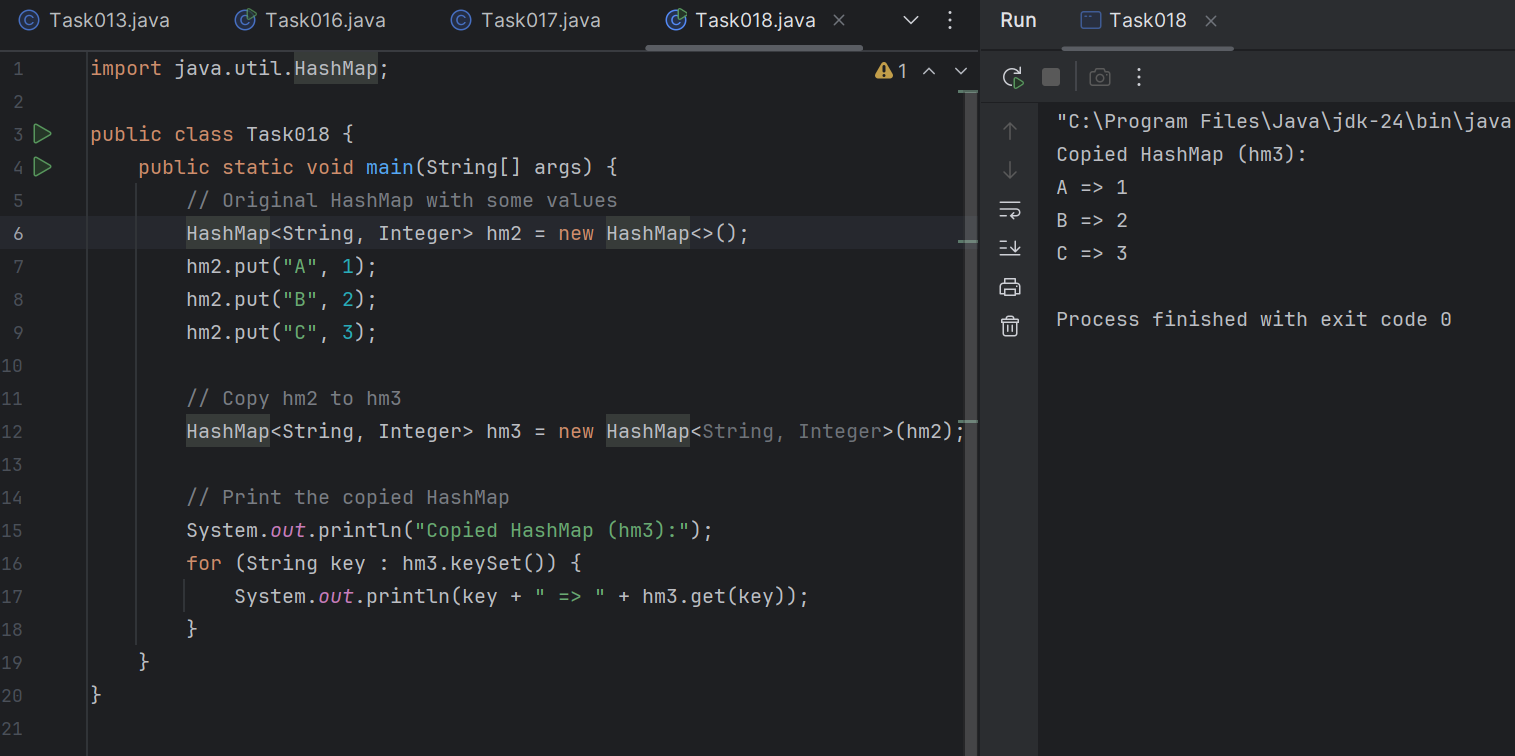


**Task 17:** Create a Hash MAp of capacity 10.



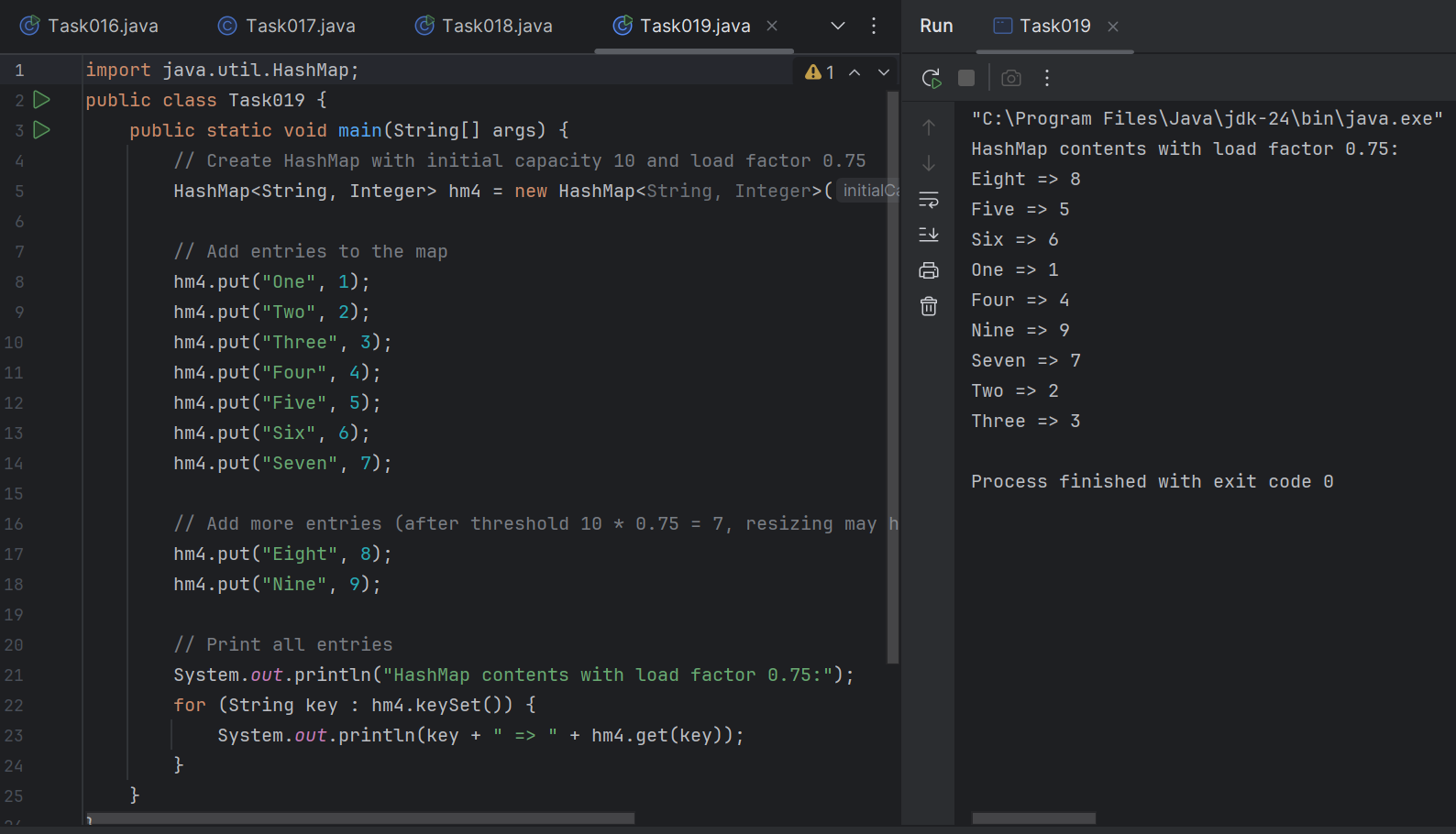
Hint: HashMap<String, Integer> hm2 = new HashMap<String, Integer>(10);

**Task 18:** Copy data from one map to another map.



Hint: HashMap<String, Integer> hm3 = new HashMap<String, Integer>( hm2);

**Task 19:** Create a  hash map using a lod factor



Hint: HashMap<String, Integer> hm4= new HashMap<String, Integer>(10, 0.75f);

Initial capacity  ===10   Load factor  === 0.75f

**Task020:** DS\_HashMapCreateMethods:

Different methods to create a hashmap in java :

1) Constructing a hashmap with default capacity

ex:

 HashMap<String, Integer> hm1 = new HashMap<String, Integer>();

2) Constructing a hashmap with a capacity 10

ex:

HashMap<String, Integer> hm2 = new HashMap<String, Integer>(10);

3)copy one map to another map

ex:

HashMap<String, Integer> hm3 = new HashMap<String, Integer>( hm2);

4)

Specifying load factor along with the capacity

ex:

 HashMap<String, Integer> hm4= new HashMap<String, Integer>(10, 0.75f);

Initial capacity  ===10

Load factor  === 0.75f

**Task 21:** Use custom method of Creating  a circular linked list and traverse the elements (display)

class Node11 {

   int data;

   Node11 next;

   public Node11(int data) {

       this.data = data;

       this.next = null;

   }

}

class CircularLinkedList {

   Node11 head = null;

   Node11 tail = null;

   // Method to add a node

   public void insert(int data) {

       Node11 newNode = new Node11(data);

       if (head == null) {

           head = newNode;

           tail = newNode;

           newNode.next = head; // Circular link

       } else {

           tail.next = newNode;

           tail = newNode;

           tail.next = head; // Point last node to head

       }

   }

   // Method to display the circular linked list

   public void display() {

       if (head == null) {

           System.*out*.println("List is empty.");

           return;

       }

       Node11 current = head;

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

       do {

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

           current = current.next;

       } while (current != head); // Stop when loop returns to head

       System.*out*.println("(back to head)");

   }

   public void delete(int value) {

       if (head == null) {

           System.*out*.println("List is empty.");

           return;

       }

       Node11 current = head;

       Node11 previous = tail;

       boolean found = false;

       do {

           if (current.data == value) {

               found = true;

               // Only one node in the list

               if (head == tail) {

                   head = null;

                   tail = null;

               }

               // Deleting the head

               else if (current == head) {

                   head = head.next;

                   tail.next = head;

               }

               // Deleting other nodes

               else {

                   previous.next = current.next;

                   if (current == tail) {

                       tail = previous;

                   }

               }

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

               return;

           }

           previous = current;

           current = current.next;

       } while (current != head);

       if (!found) {

           System.*out*.println("Value " + value + " not found in the list.");

       }

   }

}

public class Task021 {

   public static void main(String[] args) {

       CircularLinkedList clist = new CircularLinkedList();

       clist.insert(10);

       clist.insert(20);

       clist.insert(30);

       clist.insert(40);

       clist.display(); // Output: 10 -> 20 -> 30 -> 40 -> (back to head)

       clist.delete(30);

       clist.display();

   }

}

