Task 01:

Pointers : 10.07 to 10.12

#include<stdio.h>  //-- c lang

#include<iostream.h> // -- c++

int method1(){

// declaration

int age; // variable

int \*ptr;// pointer variable

// assigning

age = 10;

ptr = &age;

printf("value of age is %d", age); // 10

printf("ptr is pointing to %d ", \*ptr); // 10

printf("address of age %d ", &age);// 145245

printf("value of ptr %d ", ptr);//same as above 145245

printf("value of &ptr %d ",&ptr); // ptr address --- 454578

if c++

cout<<"value of age "<<age;

cout<<"ptr is pointing to"<<\*ptr;

cout<<"address of age "<<&age;

cout<<"value of ptr "<<ptr;

cout<<"value of &ptr "<<&ptr;

}

Code:

#include<stdio.h> //-- c lang

int main(){

int age; // variable

int \*ptr;// pointer variable

age = 10;

ptr = &age;

printf("value of age is %d \n", age); // 10

printf("ptr is pointing to %d \n ", \*ptr); // 10

printf("address of age %d \n ", &age);// 145245

printf("value of ptr %d \n ", ptr);//same as above 145245

printf("value of &ptr %d \n",&ptr); // ptr address --- 454578

}

Output:

value of age is 10

ptr is pointing to 10

address of age -535125956

value of ptr -535125956

value of &ptr -535125968

=== Code Execution Successful ===

Task 03:

Use the above code to create  a Java code which creates  a linked list.

// Define Node class

class Node {

int data; // Data part

Node next; // Pointer to next node

// Constructor

Node(int value) {

this.data = value;

this.next = null;

}

}

// LinkedList class

class LinkedList {

private Node head; // Pointer to the head of the list

// Constructor to initialize the list

public LinkedList() {

head = null;

}

// Insert a node at the end

public void insertAtEnd(int value) {

Node newNode = new Node(value);

if (head == null) {

head = newNode; // List is empty

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next; // Traverse to end

}

temp.next = newNode; // Link last node to newNode

}

}

// Delete a node by value

public void deleteByValue(int value) {

if (head == null) return;

if (head.data == value) {

head = head.next; // Move head

return;

}

Node temp = head;

while (temp.next != null && temp.next.data != value) {

temp = temp.next; // Traverse to find node

}

if (temp.next != null) {

temp.next = temp.next.next; // Unlink and let GC clean up

}

}

// Display the list

public void display() {

Node temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println("NULL");

}

}

// Main class with main method

public class Main {

public static void main(String[] args) {

LinkedList list = new LinkedList();

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();

}

}

Output:

Linked List: 10->20->30->NULL

After Deleting 20: 10->30->NULL

=== Code Execution Successful ===

Task 04:

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

Create a nodw

Try to add element at the end of the list

Remove the node

Display all the elements of the node

Find size of the linked list

Index out of bounds

class Node<T> {

    T data;

    Node<T> next;

    public Node(T data) {

        this.data = data;

        this.next = null;

    }

}

Code:

public class Main {

public static void main(String[] args) {

// Try with Integer

LinkedList<Integer> intList = new LinkedList<>();

intList.add(10);

intList.add(20);

intList.add(30);

System.out.println("Integer List:");

intList.display();

System.out.println("Size: " + intList.getSize());

intList.remove(1); // Remove element at index 1 (20)

System.out.println("After removing index 1:");

intList.display();

System.out.println("Size: " + intList.getSize());

intList.remove(5); // Try removing at an invalid index

}

}

// Node class (already provided)

class Node<T> {

T data;

Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

}

// Generic Linked List class

class LinkedList<T> {

private Node<T> head;

private int size;

public LinkedList() {

head = null;

size = 0;

}

// Add element at the end

public void add(T data) {

Node<T> newNode = new Node<>(data);

if (head == null) {

head = newNode;

} else {

Node<T> temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

size++;

}

// Remove element at a specific index

public void remove(int index) {

if (index < 0 || index >= size) {

System.out.println("Index out of bounds.");

return;

}

if (index == 0) {

head = head.next;

} else {

Node<T> temp = head;

for (int i = 0; i < index - 1; i++) {

temp = temp.next;

}

temp.next = temp.next.next;

}

size--;

}

// Display all elements

public void display() {

if (head == null) {

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

return;

}

Node<T> temp = head;

while (temp != null) {

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

temp = temp.next;

}

System.out.println("NULL");

}

// Get size of the list

public int getSize() {

return size;

}

}

Output:

Integer List:

10 -> 20 -> 30 -> NULL

Size: 3

After removing index 1:

10 -> 30 -> NULL

Size: 2

Index out of bounds.

=== Code Execution Successful ===

Task 05:

List down all the methods of Linked list

**Core Methods of a Singly Linked List:**

| **Category** | **Method Name** | **Description** |
| --- | --- | --- |
| **Insertion** | add(T data) | Add node at the end |
|  | addFirst(T data) | Add node at the beginning |
|  | addLast(T data) | Add node at the end (alias of add) |
|  | addAt(int index, T data) | Insert node at a specific index |
| **Deletion** | remove(int index) | Remove node at a specific index |
|  | removeFirst() | Remove the first node |
|  | removeLast() | Remove the last node |
|  | removeByValue(T value) | Remove node by matching value |
| **Access / Search** | get(int index) | Get data at specific index |
|  | contains(T value) | Check if list contains a value |
|  | indexOf(T value) | Return index of first occurrence |
| **Display** | display() | Print or return all elements |
|  | toString() | Convert list to string representation |
| **Utility** | size() or getSize() | Return number of nodes in list |
|  | isEmpty() | Check if list is empty |
|  | clear() | Remove all nodes from the list |
| **Advanced (Optional)** | reverse() | Reverse the list |
|  | clone() | Create a copy of the list |

Tsk 06

Create linked list using Pre defined class and add elements to it.

Hint:

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

Code:

import java.util.LinkedList;  
  
public class Mainlinkedlist{  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("kolkata");  
 list.add("Tamil nadu");  
  
 // Display the linked list  
 System.*out*.println("Linked List Elements:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
 System.*out*.println("First Element: " + list.getFirst());  
 System.*out*.println("Last Element: " + list.getLast());  
 }  
}

output:

Linked List Elements:

Delhi -> Bombay -> kolkata -> Tamil nadu -> NULL

First Element: Delhi

Last Element: Tamil nadu

Process finished with exit code 0

task 7

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

code:

import java.util.LinkedList;  
  
public class Mainlinkedlist1 {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("Kolkata");  
 list.add("Tamil Nadu");  
  
 // Display the original linked list  
 System.*out*.println("Original Linked List Elements:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
  
 // Remove 2 elements  
 list.removeFirst(); // Removes "Delhi"  
 list.removeLast(); // Removes "Kolkata"  
  
 // Display the updated list  
 System.*out*.println("\nAfter Removing 2 Elements:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
  
 // Print first and last elements  
 if (!list.isEmpty()) {  
 System.*out*.println("First Element: " + list.getFirst());  
 System.*out*.println("Last Element: " + list.getLast());  
 }  
 }  
}

output:

Original Linked List Elements:

Delhi -> Bombay -> Kolkata -> Tamil Nadu -> NULL

After Removing 2 Elements:

Bombay -> Kolkata -> NULL

First Element: Bombay

Last Element: Kolkata

Process finished with exit code 0

Task 8: in the list update the 1st element to a new value Hint: use set(1, "new value");

Code:

import java.util.LinkedList;  
  
public class update {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("Kolkata");  
 list.add("Tamil Nadu");  
  
 // Display the original linked list  
 System.*out*.println("Original Linked List Elements:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
  
 // Remove 2 elements  
 list.removeFirst(); // Removes "Delhi"  
 list.removeLast(); // Removes "Tamil Nadu"  
  
 // Update the first element (index 0) to a new value  
 list.set(0, "Bangalore"); // Changes "Bombay" to "Bangalore"  
  
 // Display the updated list  
 System.*out*.println("\nAfter Removing 2 Elements and Updating First Element:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
  
 // Print first and last elements  
 if (!list.isEmpty()) {  
 System.*out*.println("First Element: " + list.getFirst());  
 System.*out*.println("Last Element: " + list.getLast());  
 }  
 }  
}

output:

Original Linked List Elements:

Delhi -> Bombay -> Kolkata -> Tamil Nadu -> NULL

After Removing 2 Elements and Updating First Element:

Bangalore -> Kolkata -> NULL

First Element: Bangalore

Last Element: Kolkata

Process finished with exit code 0

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

Code:

import java.util.LinkedList;  
  
public class loop {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("Kolkata");  
 list.add("Tamil Nadu");  
  
 // Display using get() in a for loop  
 System.*out*.println("Displaying using get(index) in a for loop:");  
 for (int i = 0; i < list.size(); i++) {  
 System.*out*.print(list.get(i) + " -> ");  
 }  
 System.*out*.println("NULL");  
  
 // Display using for-each loop  
 System.*out*.println("\nDisplaying using for-each loop:");  
 for (String item : list) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
 }  
}

output:

Displaying using get(index) in a for loop:

Delhi -> Bombay -> Kolkata -> Tamil Nadu -> NULL

Displaying using for-each loop:

Delhi -> Bombay -> Kolkata -> Tamil Nadu -> NULL

Process finished with exit code 0

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

Code:

import java.util.LinkedList;  
  
public class NoLoop {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("Kolkata");  
 list.add("Tamil Nadu");  
  
 // Display the linked list without using loops  
 System.*out*.println("Linked List Elements:");  
 System.*out*.println(list); // Implicitly calls list.toString()  
 }  
}

output:

Linked List Elements:

[Delhi, Bombay, Kolkata, Tamil Nadu]

Process finished with exit code 0

task 11: convert the linked list to an array and display Hint : Object[] a = llobj.toArray();

code:

import java.util.LinkedList;  
import java.util.Arrays;  
  
public class linkedlisttoarray {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Add elements to the list  
 list.add("Delhi");  
 list.add("Bombay");  
 list.add("Kolkata");  
 list.add("Tamil Nadu");  
  
 // Convert LinkedList to Array  
 Object[] array = list.toArray();  
  
 // Display the array elements  
 System.*out*.println("Linked List converted to Array:");  
 for (Object item : array) {  
 System.*out*.print(item + " -> ");  
 }  
 System.*out*.println("NULL");  
 }  
}

output:

Linked List converted to Array:

Delhi -> Bombay -> Kolkata -> Tamil Nadu -> NULL

Process finished with exit code 0

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

Code:

import java.util.LinkedList;  
  
public class clone {  
 public static void main(String[] args) {  
 // Original LinkedList  
 LinkedList<String> originalList = new LinkedList<>();  
  
 // Add elements  
 originalList.add("Delhi");  
 originalList.add("Bombay");  
 originalList.add("Kolkata");  
 originalList.add("Tamil Nadu");  
  
 // Clone the linked list  
 LinkedList<String> clonedList = (LinkedList<String>) originalList.clone();  
  
 // Display original list  
 System.*out*.println("Original Linked List:");  
 System.*out*.println(originalList);  
  
 // Display cloned list  
 System.*out*.println("Cloned Linked List:");  
 System.*out*.println(clonedList);  
  
 // Modify original list to check independence  
 originalList.remove("Bombay");  
 clonedList.add("Chennai");  
  
 System.*out*.println("\nAfter modifying both lists:");  
 System.*out*.println("Original List: " + originalList);  
 System.*out*.println("Cloned List: " + clonedList);  
 }  
}

output:

Original Linked List:

[Delhi, Bombay, Kolkata, Tamil Nadu]

Cloned Linked List:

[Delhi, Bombay, Kolkata, Tamil Nadu]

After modifying both lists:

Original List: [Delhi, Kolkata, Tamil Nadu]

Cloned List: [Delhi, Bombay, Kolkata, Tamil Nadu, Chennai]

Process finished with exit code 0

Task 13:

Use pop and push methods on linked list.. LIFO – just follow..

import java.util.LinkedList;  
  
public class pushandpop {  
 public static void main(String[] args) {  
 // Create a LinkedList of Strings  
 LinkedList<String> list = new LinkedList<>();  
  
 // Push elements onto the list (LIFO behavior)  
 list.push("Delhi");  
 list.push("Bombay");  
 list.push("Kolkata");  
 list.push("Tamil Nadu");  
  
 // Display the linked list after pushes  
 System.*out*.println("Linked List Elements after push (Top to Bottom):");  
 System.*out*.println(list); // [Tamil Nadu, Kolkata, Bombay, Delhi]  
  
 // Pop one element (should remove "Tamil Nadu")  
 String poppedElement = list.pop();  
 System.*out*.println("\nPopped Element: " + poppedElement);  
  
 // Display the list after pop  
 System.*out*.println("Linked List after pop:");  
 System.*out*.println(list); // [Kolkata, Bombay, Delhi]  
  
 // Print current first and last elements  
 System.*out*.println("\nFirst Element (Top): " + list.getFirst());  
 System.*out*.println("Last Element (Bottom): " + list.getLast());  
 }  
}

output:

Linked List Elements after push (Top to Bottom):

[Tamil Nadu, Kolkata, Bombay, Delhi]

Popped Element: Tamil Nadu

Linked List after pop:

[Kolkata, Bombay, Delhi]

First Element (Top): Kolkata

Last Element (Bottom): Delhi

Process finished with exit code 0

Task 14:

Splititerator

import java.util.\*;

public class Task0014\_DS\_Linkedlist\_SplitIterator {

    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);

    }

}

CODE:

import java.util.\*;  
  
public class Task0014\_DS\_Linkedlist\_SplitIterator {  
 public static void main(String[] args) {  
 LinkedList<String> lobj = new LinkedList<>();  
 lobj.add("Jaysree");  
 lobj.add("Hariharan");  
 lobj.add(".BV");  
 Spliterator<String> sitobj = lobj.spliterator();  
 //forEachRemaining is a method of Spliterator  
 System.*out*.println("Splitting the list:");  
 sitobj.forEachRemaining(System.*out*::println);  
 }  
}

Output:

Splitting the list:

Jaysree

Hariharan

.BV

Process finished with exit code 0

Task 15:

tryAdvance()

import java.util.LinkedList;

import java.util.Spliterator;

public class Task0015\_DS\_Linkedlist\_SplitItr2Lists {

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); } ) );

  }

}

Output:

spliterator 1

spliterator 2

Jaysree

Hariharan

.BS

Love

Process finished with exit code 0

Task 16:

Create a doubly linked list..

// Node class for Doubly Linked List  
class Node {  
 String data;  
 Node prev;  
 Node next;  
  
 public Node(String data) {  
 this.data = data;  
 this.prev = null;  
 this.next = null;  
 }  
}  
  
// Doubly Linked List class  
class DoublyLinkedList {  
 Node head;  
  
 // Add node at the end  
 public void add(String data) {  
 Node newNode = new Node(data);  
  
 if (head == null) {  
 head = newNode;  
 return;  
 }  
  
 Node temp = head;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
  
 temp.next = newNode;  
 newNode.prev = temp;  
 }  
  
 // Display list forward  
 public void displayForward() {  
 System.*out*.println("Doubly Linked List (Forward):");  
 Node temp = head;  
 while (temp != null) {  
 System.*out*.print(temp.data + " <-> ");  
 temp = temp.next;  
 }  
 System.*out*.println("NULL");  
 }  
  
 // Display list backward  
 public void displayBackward() {  
 System.*out*.println("Doubly Linked List (Backward):");  
 if (head == null) return;  
  
 // Go to the last node  
 Node temp = head;  
 while (temp.next != null) {  
 temp = temp.next;  
 }  
  
 // Traverse backward  
 while (temp != null) {  
 System.*out*.print(temp.data + " <-> ");  
 temp = temp.prev;  
 }  
 System.*out*.println("NULL");  
 }  
}  
  
// Main class  
public class DoublyLinkedListMain {  
 public static void main(String[] args) {  
 DoublyLinkedList list = new DoublyLinkedList();  
  
 list.add("Delhi");  
 list.add("Mumbai");  
 list.add("Kolkata");  
 list.add("Chennai");  
  
 list.displayForward(); // Print from head to tail  
 list.displayBackward(); // Print from tail to head  
 }  
}

Output:

Doubly Linked List (Forward):

Delhi <-> Mumbai <-> Kolkata <-> Chennai <-> NULL

Doubly Linked List (Backward):

Chennai <-> Kolkata <-> Mumbai <-> Delhi <-> NULL

Process finished with exit code 0

Task 17:

Create a Hash MAp of capacity 10.

import java.util.HashMap;  
  
public class hashmapcapacity10 {  
 public static void main(String[] args) {  
 // Create a HashMap with initial capacity 10  
 HashMap<Integer, String> map = new HashMap<>(10);  
  
 // Add some key-value pairs  
 map.put(1, "Apple");  
 map.put(2, "Banana");  
 map.put(3, "Cherry");  
 map.put(4, "Berry");  
  
 // Display the HashMap  
 System.*out*.println("HashMap contents:");  
 for (Integer key : map.keySet()) {  
 System.*out*.println("Key: " + key + ", Value: " + map.get(key));  
 }  
 }  
}

Output:

HashMap contents:

Key: 1, Value: Apple

Key: 2, Value: Banana

Key: 3, Value: Cherry

Key: 4, Value: Berry

Process finished with exit code 0

Task 18:

Copy data from one map to another map.

import java.util.HashMap;  
  
public class copyfromonetoanother {  
 public static void main(String[] args) {  
 // Original map  
 HashMap<Integer, String> originalMap = new HashMap<>();  
 originalMap.put(1, "Apple");  
 originalMap.put(2, "Banana");  
 originalMap.put(3, "Cherry");  
  
 // New map to copy into  
 HashMap<Integer, String> copiedMap = new HashMap<>();  
  
 // Copy data using putAll  
 copiedMap.putAll(originalMap);  
  
 // Display copied map  
 System.*out*.println("Copied Map:");  
 for (Integer key : copiedMap.keySet()) {  
 System.*out*.println("Key: " + key + ", Value: " + copiedMap.get(key));  
 }  
 }  
}

output:

Copied Map:

Key: 1, Value: Apple

Key: 2, Value: Banana

Key: 3, Value: Cherry

Process finished with exit code 0

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

Code:

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

// 1) Default capacity and load factor (16, 0.75)

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

hm1.put("A", 10);

hm1.put("B", 20);

System.out.println("HashMap 1 (Default): " + hm1);

// 2) Initial capacity = 10, load factor = 0.75 (default)

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

hm2.put("X", 100);

hm2.put("Y", 200);

System.out.println("HashMap 2 (Capacity 10): " + hm2);

// 3) Copy one map to another (copy hm2 into hm3)

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

System.out.println("HashMap 3 (Copied from HashMap 2): " + hm3);

// 4) Initial capacity = 10, Load factor = 0.75

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

hm4.put("P", 111);

hm4.put("Q", 222);

System.out.println("HashMap 4 (Capacity 10, Load Factor 0.75): " + hm4);

}

}

Output:

HashMap 1 (Default): {A=10, B=20}

HashMap 2 (Capacity 10): {X=100, Y=200}

HashMap 3 (Copied from HashMap 2): {X=100, Y=200}

HashMap 4 (Capacity 10, Load Factor 0.75): {P=111, Q=222}

=== Code Execution Successful ===

Task 21:

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

// Node class for Circular Linked List  
class Node1 {  
 int data;  
 Node1 next;  
  
 Node1(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
  
// Custom Circular Linked List class  
class CircularLinkedList {  
 Node1 head = null;  
 Node1 tail = null;  
  
 // Method to add a node to the list  
 public void add(int data) {  
 Node1 newNode = new Node1(data);  
  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 newNode.next = head; // Points to itself (circular)  
 } else {  
 tail.next = newNode; // Link new node after tail  
 tail = newNode; // Update tail  
 tail.next = head; // Make it circular  
 }  
 }  
  
 // Method to display the list  
 public void display() {  
 if (head == null) {  
 System.*out*.println("List is empty.");  
 return;  
 }  
  
 Node1 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)");  
 }  
}  
  
// Main class to run the example  
public class circular {  
 public static void main(String[] args) {  
 CircularLinkedList cll = new CircularLinkedList();  
  
 cll.add(10);  
 cll.add(20);  
 cll.add(30);  
 cll.add(40);  
  
 cll.display();  
 }  
}

Output:

Circular Linked List: 10 -> 20 -> 30 -> 40 -> (back to head)

Process finished with exit code 0