Task 1 Create a linked list 4 node and travese it then upload it to github

// Node class

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

// LinkedList class

public class LinkedList {

Node head;

// Method to add a node at the end

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;

}

}

// Method to traverse and display the list

public void display() {

Node current = head;

if (current == null) {

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

return;

}

System.*out*.print("Linked List Elements: ");

while (current != null) {

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

current = current.next;

}

System.*out*.println();

}

// Main method

public static void main(String[] args) {

LinkedList list = new LinkedList();

// Adding 4 elements

list.add(10);

list.add(20);

list.add(30);

list.add(40);

// Displaying elements

list.display();

}

}

Task 2 what is Traverse

* The traversal **starts at the head node**.
* Then it moves to the **next** node using the next pointer.
* This continues until it reaches the **last node**, where next == null.

You typically traverse a list to:

* **Display** the elements.
* **Search** for a value.
* **Modify** or **delete** elements.
* **Count** nodes.

Task 3:

Create a Circular Linked list using Task 1 Singly linked list/ doubly linked list.

// Node class

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

// CircularLinkedList class

public class CircularLinkedList {

Node head = null;

Node tail = null;

// Method to add a node to the circular list

public void add(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

newNode.next = head; // Circular link

} else {

tail.next = newNode; // Add new node after tail

tail = newNode; // Move tail to new node

tail.next = head; // Maintain circular link

}

}

// Method to traverse and display the circular list

public void display() {

Node current = head;

if (head == null) {

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

return;

}

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

do {

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

current = current.next;

} while (current != head); // Stop when we reach the head again

System.*out*.println();

}

// Main method

public static void main(String[] args) {

CircularLinkedList list = new CircularLinkedList();

// Adding 4 elements

list.add(10);

list.add(20);

list.add(30);

list.add(40);

// Displaying the list

list.display();

}

}

Task 4:

Delete a node in the circular linked list

// Node class

class Node {

int data;

Node next;

Node(int data) {

this.data = data;

}

}

// Circular Linked List class

public class Deletenodecircularlinkedlist {

Node head = null;

Node tail = null;

// Add node to circular linked list

public void add(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

tail.next = head;

} else {

tail.next = newNode;

tail = newNode;

tail.next = head;

}

}

// Delete node by value

public void delete(int value) {

if (head == null) {

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

return;

}

Node current = head;

Node prev = tail;

// Special case: deleting head node

do {

if (current.data == value) {

if (current == head) {

if (head == tail) {

head = tail = null; // Only one node

} else {

head = head.next;

tail.next = head;

}

} else {

prev.next = current.next;

if (current == tail) {

tail = prev;

}

}

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

return;

}

prev = current;

current = current.next;

} while (current != head);

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

}

// Display the list

public void display() {

if (head == null) {

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

return;

}

Node current = head;

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

do {

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

current = current.next;

} while (current != head);

System.*out*.println();

}

// Main method

public static void main(String[] args) {

Deletenodecircularlinkedlist list = new Deletenodecircularlinkedlist();

// Adding elements

list.add(10);

list.add(20);

list.add(30);

list.add(40);

System.*out*.println("Original list:");

list.display();

// Deleting a node

list.delete(20); // Delete middle

list.display();

// Try to delete from empty list

}

}

Stacks 👍

Task 5:

Create a stack and pop the element also print the popped element.

Task 6:

Find an element in the stack and display the position

Hint 👍

Int position = names.search(“value”);

Task 7:

Peek the element and print it ..

Task 8:

Check if the stack is empty or not?

Task 5 – 8

import java.util.Stack;

public class StackExample {

public static void main(String[] args) {

// Create a stack of integers

Stack<Integer> stack = new Stack<>();

// Push elements onto the stack

stack.push(10);

stack.push(20);

stack.push(30);

stack.push(40);

// Display the stack

System.*out*.println("Stack: " + stack);

// Pop the top element

int poppedElement = stack.pop();

int peekElement = stack.peek(40); // returns an object from the top of the stack

int position = stack.search(20); //returns the position of the element from the top of the stack

boolean result = stack.empty(); //whether a stack is empty or not

// Print the popped element

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

// Display the stack after pop

System.*out*.println("Stack after pop: " + stack);

System.*out*.println("Stack after peek: " + peekElement);

System.*out*.println("Stack searching : " + position);

System.*out*.println("Stack searching : " + result);

}

}

=========================================================

Task 9

What are the methods of the stack class.. List them down.. With a one liner..

push() - Adds an element to the top of the stack

pop() - Removes and returns the topmost element from the stack

peek() or top() - Returns the top element without removing it from the stack

isEmpty() - Returns true if the stack is empty, false otherwise

size() - Returns the number of elements currently in the stack

clear() - Removes all elements from the stack making it empty

isFull() - Returns true if the stack has reached its maximum capacity (for fixed-size implementations)

Task 10:

Wap to create a queue with custom methods

Is empty ()

Is full()

Enque

Deque

Peek

display()

Home tasks

What is the difference between recursion and iteration

**Recursion** is when a function calls itself to solve a smaller part of the problem, while **iteration** uses loops (for, while) to repeat steps until a condition is met.

Wap to find the factorial of a number

public class hometask01 {

// Recursive method to find factorial

public static int factorial(int n) {

if (n == 0 || n == 1) {

return 1; // Base case: 0! = 1 and 1! = 1

} else {

return n \* *factorial*(n - 1); // Recursive call

}

}

public static void main(String[] args) {

int num = 5; // Change this to any number you want

int result = *factorial*(num);

System.*out*.println("Factorial of " + num + " is: " + result);

}

}

Wap to find the Fibonacci series of a number

public class hometask02 {

// Recursive method to find the nth Fibonacci number

public static int fibonacci(int n) {

if (n == 0) return 0; // Base case 1

if (n == 1) return 1; // Base case 2

return *fibonacci*(n - 1) + *fibonacci*(n - 2); // Recursive step

}

public static void main(String[] args) {

int terms = 10; // Change this to print more or fewer terms

System.*out*.println("Fibonacci series up to " + terms + " terms:");

for (int i = 0; i < terms; i++) {

System.*out*.print(*fibonacci*(i) + " ");

}

}

}

Wap to reverse a string using recursion..

public class hometask03 {

// Recursive method to reverse a string

public static String reverse(String str) {

if (str.isEmpty()) {

return str; // Base case: empty string

}

return *reverse*(str.substring(1)) + str.charAt(0); // Recursive call

}

public static void main(String[] args) {

String input = "hello";

String reversed = *reverse*(input);

System.*out*.println("Original string: " + input);

System.*out*.println("Reversed string: " + reversed);

}

}

Write a recursive function to search for an element in an array

public class hometask04 {

// Recursive function to search for target in array

public static int recursiveSearch(int[] arr, int target, int index) {

// Base case: if index goes out of bounds

if (index >= arr.length) {

return -1; // not found

}

// If current element matches the target

if (arr[index] == target) {

return index;

}

// Recursive call: search in the rest of the array

return *recursiveSearch*(arr, target, index + 1);

}

public static void main(String[] args) {

int[] array = {4, 7, 2, 9, 1, 5};

int target = 9;

int result = *recursiveSearch*(array, target, 0);

if (result != -1) {

System.*out*.println("Element " + target + " found at index: " + result);

} else {

System.*out*.println("Element " + target + " not found in the array.");

}

}

}

Write a recursive function to count the digits of a positive integer (do also for sum of digits)

public class hometask05 {

// Recursive function to count digits

public static int countDigits(int num) {

if (num == 0) {

return 0;

}

return 1 + *countDigits*(num / 10);

}

// Recursive function to sum digits

public static int sumDigits(int num) {

if (num == 0) {

return 0;

}

return (num % 10) + *sumDigits*(num / 10);

}

public static void main(String[] args) {

int number = 12345;

int digitCount = *countDigits*(number);

int digitSum = *sumDigits*(number);

System.*out*.println("Number: " + number);

System.*out*.println("Total digits: " + digitCount);

System.*out*.println("Sum of digits: " + digitSum);

}

}

Write a recursive function to reverse a null-terminated string

public class hometask06 {

// Recursive function to reverse a string (simulating null-terminated behavior)

public static void reverse(char[] str, int index) {

if (str[index] == '\0') {

return; // Base case: reached null terminator

}

*reverse*(str, index + 1); // Recurse until end

System.*out*.print(str[index]); // Print during stack unwinding

}

public static void main(String[] args) {

// Simulate a null-terminated string

char[] str = {'H', 'e', 'l', 'l', 'o', '\0'};

System.*out*.print("Reversed string: ");

*reverse*(str, 0); // Start recursion from index 0

}

}

Write a recursive function to convert a decimal number to binary

public class hometask07 {

// Recursive function to convert decimal to binary

public static void decimalToBinary(int n) {

if (n == 0) {

return; // Base case

}

*decimalToBinary*(n / 2); // Recursive call

System.*out*.print(n % 2); // Print remainder after recursion

}

public static void main(String[] args) {

int number = 13;

System.*out*.print("Binary of " + number + " is: ");

if (number == 0) {

System.*out*.print("0");

} else {

*decimalToBinary*(number);

}

}

}