import java.io.\*;

import java.util.\*;

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class LinkedList1 {

Node head;

public LinkedList1() {

this.head = null;

}

// Method to insert a new node at the end of the linked list

public void append(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node last = head;

while (last.next != null) {

last = last.next;

}

last.next = newNode;

}

// Method to display the linked list

public void display() {

Node current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

}

public class LinkedList {

public static void main(String[] args) {

LinkedList1 linkedList = new LinkedList1();

// adding elements to the linked list

linkedList.append(1);

linkedList.append(2);

linkedList.append(3);

linkedList.append(4);

// Displaying the linked list

System.out.println("Linked List Value: ");

linkedList.display();

}

}

2.

// Method to reverse the linked list

public void reverse() {

Node4 prev = null;

Node4 current = head;

Node4 next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}

// Method to display the linked list

public void display() {

Node4 current = head;

while (current != null) {

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

current = current.next;

}

System.out.println();

}

}

public class LinkedList3 {

public static void main(String[] args) {

Linked lst = new Linked();

lst.append(1);

lst.append(2);

lst.append(3);

lst.append(4);

lst.append(5);

// Displaying the linked list

System.out.println("Linked List Values: ");

lst.display();

// Reverse the linked list

System.out.println("Reversed Linked List Values: ");

lst.reverse();

lst.display();

}

}

class Node1{

int data;

Node next;

public Node1(int data) {

this.data=data;

this.next=null;

}

}

class LinkedList1{

Node head;

public LinkedList1() {

this.head=null;

}

//Method to insert a new node at end

public void append(int data) {

Node newNode=new Node(data);

if(head == null) {

head=newNode;

return;

}

Node current=head;

while(current.next !=null) {

current=current.next;

}

current.next=newNode;

}

//method to reverse the linked list

public void reverse() {

Node prev=null;

Node current=head;

Node next=null;

while(current !=null) {

next = current.next;

current.next=prev;

prev=current;

current=next;

}

head=prev;

}

//Method to display the linked list

public void display() {

Node current = head;

while(current !=null) {

System.out.println(current.data + " ");

current=current.next;

}

System.out.println();

}

}

public class Main1{

public static void main(String gdg[]) {

LinkedList1 myLinkedList = new LinkedList1();

//Adding the element into the list

myLinkedList.append(10);

myLinkedList.append(20);

myLinkedList.append(30);

myLinkedList.append(40);

myLinkedList.append(50);

//Displaying the original linked list

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

myLinkedList.display();

//Reversing the list

myLinkedList.reverse();

//Displaying the Reverse value

System.out.println

3) tree:

insertion :

#include <stdio.h>

int main() {

// Declare an array

int array[10] = {1, 2, 3, 4, 5};

// Size of the array

int size = 5;

// Element to be inserted

int element = 10;

// Position at which the element should be inserted

int position;//user

printf("enter position: ");

scanf("%d",&position);

printf("array before insertion:\n");

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

printf("%d ", array[i]);

}

if (position < 0 || position > size) {

printf("Invalid position for insertion.\n");

}

// Shift elements to make space for the new element

for (int i = size - 1; i >= position; --i) {

array[i + 1] = array[i];

}

// Insert the new element

array[position] = element;

// Update the size of the array

size++;

// Print the updated array

printf("\narray after insertion:\n");

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

printf("%d ", array[i]);

}

return 0;

}

#include<stdio.h>

int main(){

int array[10]={1,2,3,4,5};

//size of the array

int size = 5;

//Position to be works as static

int posotion=2;

printf("array before deletion: \n");

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

printf("%d",array[i]);

}

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

printf("Invalid position for deletion");

printf("%d",array[i]);

}

//shift the element to be remove

printf("Update array after the deletion: \n");

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

printf("%d",array[i]);

}

return 0;

}

//

#include <stdio.h>

int main() {

// Declare and initialize an array

int array[10] = {1, 2, 3, 4, 5};

// Size of the array

int size = 5;

// Position from which the element should be deleted

int position = 2;

printf("array before deletion:\n");

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

printf("%d ", array[i]);

}

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

printf("Invalid position for deletion.\n");

}

// Shift elements to

for (int i = position; i < size - 1; ++i) {

array[i] = array[i + 1];

}

// Update the size of the array

size--;

// Print the updated array

printf("\nUpdated array after deletion:\n");

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

printf("%d ", array[i]);

}

return 0;

}

//

4) tree [ ,pre]

|  |  |  |
| --- | --- | --- |
| |  | | --- | | https://mail.google.com/mail/u/0/images/cleardot.gif | |  |

import java.util.Scanner;  
  
class TreeNode {  
    int data;  
    TreeNode left;  
    TreeNode right;  
  
    // Constructor  
    public TreeNode(int data) {  
        this.data = data;  
        this.left = null;  
        this.right = null;  
    }  
}  
  
// Main class  
public class BinaryTree {  
    TreeNode root;  
  
    // Constructor using Main class passing the parameter  
    public BinaryTree(int rootData) {  
        this.root = new TreeNode(rootData);  
    }  
  
    // Method for invoking insert  
    public void insert(int data) {  
        root = insertVal(root, data);  
    }  
  
    private TreeNode insertVal(TreeNode root, int data) {  
        if (root == null) {  
            root = new TreeNode(data);  
            return root;  
        }  
  
        if (data < root.data) {  
            root.left = insertVal(root.left, data);  
        } else {  
            if (data > root.data) {  
                root.right = insertVal(root.right, data);  
            }  
        }  
        return root;  
    }  
  
    // Pre-order traversal  
    public void PreOrder() {  
        PreOrderVal(root);  
    }  
  
    private void PreOrderVal(TreeNode root) {  
        if (root != null) {  
            System.out.print(root.data + " ");  
            PreOrderVal(root.left);  
            PreOrderVal(root.right);  
        }  
    }  
  
    public static void main(String gsh[]) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter a Root value: ");  
        int rootValue = sc.nextInt();  
        BinaryTree tree = new BinaryTree(rootValue);  
        System.out.println("Enter the number of node value: ");  
        int numNode = sc.nextInt();  
        for (int i = 0; i < numNode; i++) {  
            System.out.println("Enter the element " + (i + 1) + ": ");  
            int nodeValue = sc.nextInt();  
            tree.insert(nodeValue);  
        }  
        System.out.println("PreOrder Traversal: ");  
        tree.PreOrder();  
    }  
}

* BSF GRAPH:

import java.util.LinkedList;

import java.util.Queue;

public class GraphBFS {

private int V; //number of vertices

private LinkedList<Integer>[] adjList; //Adjacency list representation

public GraphBFS(int v) {

V = v;

adjList = new LinkedList[v];

for (int i = 0; i < v; ++i)

adjList[i] = new LinkedList<>();

}

// Function to add an edge to the graph

void addEdge(int v, int w) {

adjList[v].add(w);

}

// Function for BFS traversal

void BFS(int s) {

boolean[] visited = new boolean[V];

Queue<Integer> queue = new LinkedList<>(); // Create a queue for BFS

visited[s] = true;

queue.add(s);

while (!queue.isEmpty()) {

s = queue.poll();

System.out.print(s + " ");

// loop for search the possibility value nearby:

for(int neighbor:adjList[s]) {

if(!visited[neighbor]) {

visited[neighbor]=true;

queue.add(neighbor);

}

}

}

// main

public static void main(String arg[]) {

GraphBFS g=new GraphBFS(4);

g.addEdge(0,1);

g.addEdge(0,3);

g.addEdge(1,2);

g.addEdge(2,0);

g.addEdge(2,3);

g.addEdge(3,1);

System.out.println("Starting vertex:");

g.BFS(2)

}

1. Directed

import java.util.ArrayList;  
import java.util.List;  
  
class Graph {  
    private int numVertices;  
    private List<List<Integer>> adjacentList;  
  
    // constructor  
    public Graph(int numVertices) {  
        this.numVertices = numVertices;  
        this.adjacentList = new ArrayList<>(numVertices);  
  
        // loop for number of vertices  
        for (int i = 0; i < numVertices; i++) {  
            this.adjacentList.add(new ArrayList<>());  
        }  
    }  
  
    public void addEdges(int source, int destination) {  
        // undirected Graph  
        this.adjacentList.get(source).add(destination);  
        // this.adjacentList.get(destination).add(source);  
    }  
  
    // method for graph value  
    public void graphPrint() {  
        System.out.println("Graph: ");  
        for (int i = 0; i < numVertices; i++) {  
            System.out.print("Vertex " + i + ": ");  
  
            // foreach loop for encapsulation  
            for (Integer near : adjacentList.get(i)) {  
                System.out.print(near + " ");  
            }  
  
            System.out.println();  
        }  
    }  
}  
  
// main class  
public class ListArray {  
    public static void main(String[] args) {  
        // object for subclass -> Graph  
        Graph grp = new Graph(5);  
  
        // edges  
        grp.addEdges(1, 3);  
        grp.addEdges(0, 2);  
        grp.addEdges(1, 4);  
        grp.addEdges(0, 4);  
        grp.addEdges(2, 3);  
  
        // invoking the graph method  
        grp.graphPrint();  
    }  
}

//collections

import java.util.HashMap;

import java.util.Map;

public class HasMap {

public static void main(String[] args) {

// TODO Auto-generated method stub

//create HashMap using keys and values with <>=Gendrics

//creating object for map

Map<Integer,String> hash=new HashMap();

//Insertion()

//Key-values user Define value

hash.put(1,"AI");

hash.put(2,"ML");

hash.put(3,"CSD");

//Assigning the key for Hashing

System.out.println("key value 1: "+hash.get(2));

System.out.println("key value 1: "+hash.get(3));

//Print all the values of key

System.out.println("HashMap: "+hash);

//Deletion

hash.remove(1);

//Print the value after remove()

System.out.println("After Removal of hash: "+hash);

}

}