|  |  |
| --- | --- |
| Student Name | Asadullah |
| Roll Number | 21SW036 |
| Section # | 03 |
| Lab # | 07 |

**Task#01**

Question statement

The Music app name is FireAir(class) music app which can perform following functions

•print all the linked list of songs/music names one by one.

•Add new songs/music (insertion at start, end, middle)

•Deletes a song/music using the number (Deletion at start, end, middle)

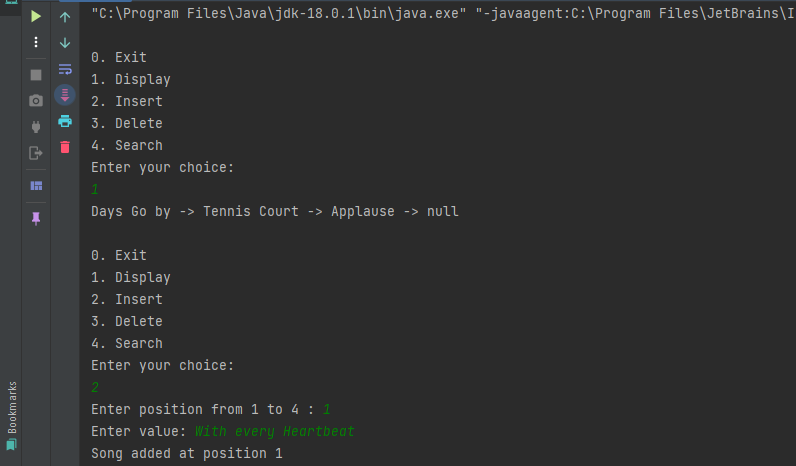
•Searches song/music.

# Q1.Java

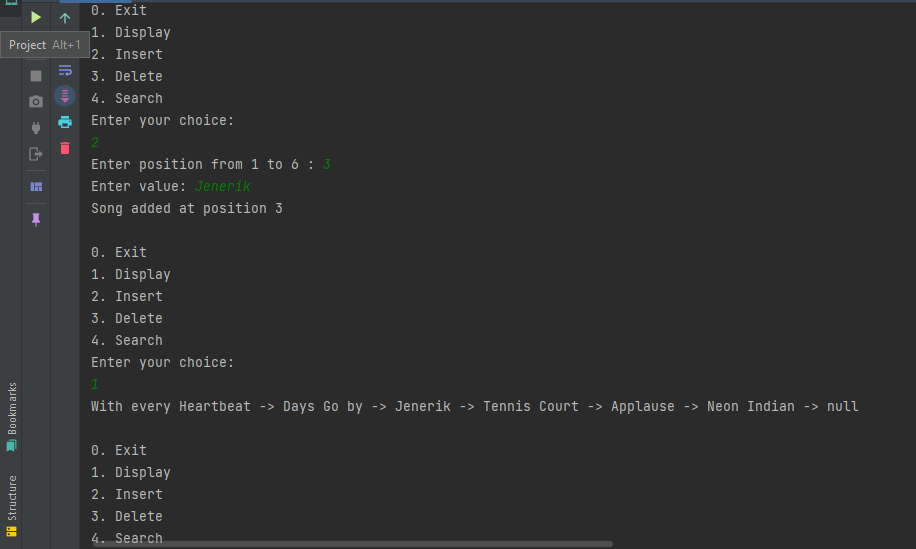
**Code:**

import java.util.Scanner;  
  
  
class Node{  
 Node next;  
 String str;  
 public Node(String str){  
 this.str = str;  
 } // end of constructor  
} // end of class Node  
  
class FireAir{  
  
 public void print(Node head){  
 for (Node p = head; p!=null; p=p.next){  
 if (p.next==null)  
 System.*out*.print(p.str+" -> null");  
 else  
 System.*out*.print(p.str+" -> ");  
 } // end of for loop  
 System.*out*.println();  
 } // end of print()  
  
 public int length(Node head){  
 int count = 0;  
 for (Node p = head; p!=null; p=p.next){  
 count++;  
 }  
 return count;  
 } // end of length()  
  
  
 public Node add(Node head){  
  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter position from 1 to "+(length(head)+1)+" : ");  
 int pos = sc.nextInt();  
 String s = sc.nextLine();  
 System.*out*.print("Enter value: ");  
 String string = sc.nextLine();  
 Node node = new Node(string);  
  
 Node p = head;  
  
 int length = length(head);  
  
 // insertion at beginning  
 if (pos == 1) {  
 node.next = head;  
 head = node;  
 // insertion at end  
 } else if (pos==length+1){  
 for (p = head; p.next!=null; p=p.next){  
  
 } // end of for loop  
 p.next = node;  
 // insertion in middle  
 } else {  
 int count = 1;  
 for (p = head; p.next!=null; p=p.next){  
 if (count==pos-1){  
 node.next = p.next;  
 p.next = node;  
 }  
 count++;  
 } // end of for loop  
 } // end of if else  
  
 System.*out*.println("Song added at position "+pos);  
 return head;  
 } // end of add() method  
  
 public Node delete(Node head){  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter position from 1 to "+length(head)+" : ");  
 int pos = sc.nextInt();  
  
  
 int length = length(head);  
 Node p = head;  
  
 if (pos==1){  
 head = p.next;  
 }  
 else if (pos==length){  
 for (p = head; p.next.next!=null; p=p.next){  
  
 }  
 p.next = null;  
 }  
 else {  
 int count = 1;  
 for (p = head; p!=null; p=p.next){  
 if (pos==count+1){  
 p.next = p.next.next;  
 }  
 count++;  
 }  
  
 } // end of if else  
 System.*out*.println("Song deleted from position "+pos);  
 return head;  
 } // end of delete()  
 public void search(Node head){  
 boolean flag = false;  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter song name to search: ");  
 String target = sc.nextLine();  
 int pos = 1;  
 for (Node p = head; p!=null; p=p.next){  
 if (p.str.equalsIgnoreCase(target)){  
 flag = true;  
 break;  
 }  
 pos++;  
 }  
 if (flag)  
 System.*out*.println("Song found at position "+pos);  
 else  
 System.*out*.println("Song not found");  
 } // end of search() method  
  
} // end of class FireAir  
  
public class Q1 {  
  
 public static void choices(){  
 System.*out*.println();  
 System.*out*.println("0. Exit");  
 System.*out*.println("1. Display");  
 System.*out*.println("2. Insert");  
 System.*out*.println("3. Delete");  
 System.*out*.println("4. Search");  
 } // end of choices method  
  
 public static void main(String[] args) {  
  
 FireAir fireAir = new FireAir();  
  
 Node start = new Node("Days Go by");  
 Node p = start;  
 p.next = new Node("Tennis Court");  
 p = p.next;  
 p.next = new Node("Applause");  
 p = p.next;  
  
  
  
 Scanner sc = new Scanner(System.*in*);  
  
 int choice = 0;  
 do{  
 *choices*();  
 System.*out*.println("Enter your choice: ");  
 choice = sc.nextInt();  
 switch (choice){  
 case 1 -> fireAir.print(start);  
 case 2 -> start = fireAir.add(start);  
 case 3 -> start = fireAir.delete(start);  
 case 4 -> fireAir.search(start);  
 }  
 }while (choice!=0);  
  
  
  
  
  
  
 } // end of main() method  
} // end of program

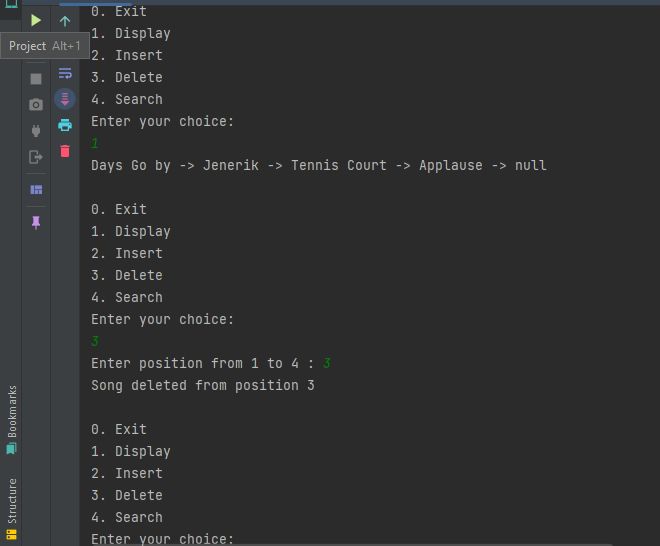
**Output:**

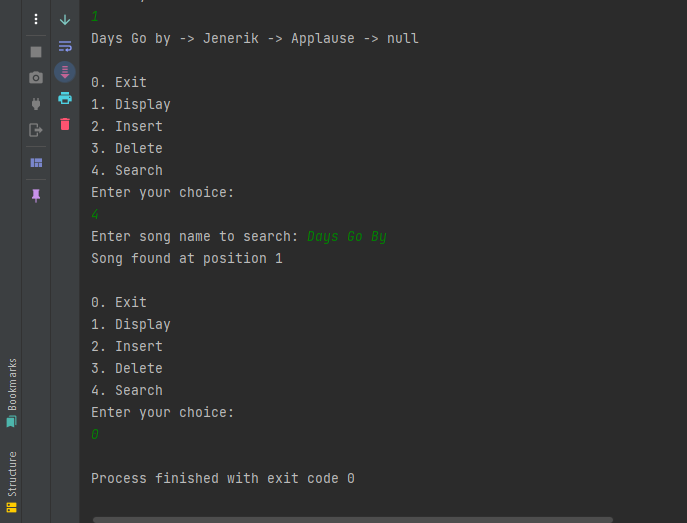
****

****

****

****

****

****

**Task#02**

Question statement

Daily you would very often make the long trip out to Mehran UET. This route required a very specific sequence of buses, trains, and subways to get to the destination which you would follow along with using Google Maps.

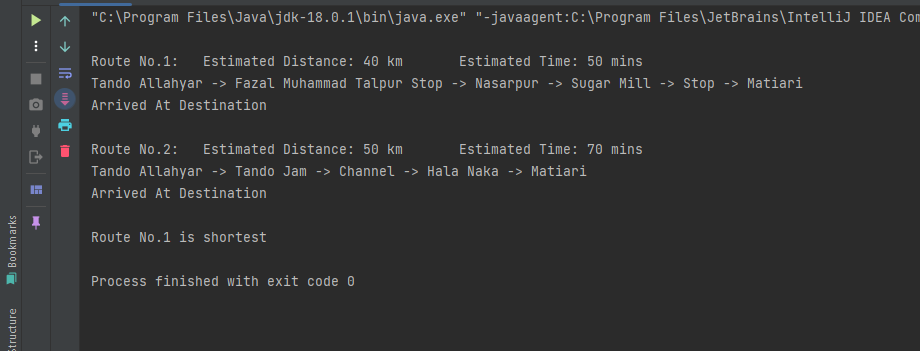
When you type in my origin, and destination, Google uses a complex algorithm to map out all of my possible routes and returns the top options in terms of minimizing the time and effort to get to your direction. At the end of the day, however, all it’s doing is returning a Linked List to your phone. Now create this process.

# Q2.Java

**Code:**

class Node3{ //Node3 Class  
 String location;  
 Node3 next;  
  
 Node3(String location){ //Node3 class Constructor  
 this.location = location;  
 }  
 public void display(Node3 head){  
 for (Node3 p=head; p!=null; p=p.next){  
 if (p.next==null)  
 System.*out*.print(p.location);  
 else  
 System.*out*.print(p.location+" -> ");  
 }  
 System.*out*.println();  
 } // end of display()  
}  
  
  
public class Q2 {  
 public static void main(String[] args) {  
  
  
 System.*out*.println();  
  
 String [] route1 = {"Tando Allahyar", "Fazal Muhammad Talpur Stop", "Nasarpur", "Sugar Mill", "Stop" ,"Matiari"};  
 String [] route2 = {"Tando Allahyar", "Tando Jam", "Channel", "Hala Naka", "Matiari"};  
  
 int estimatedTime1 = 50;  
 int estimatedTime2 = 70;  
  
 int distance1 = 40;  
 int distance2 = 50;  
  
 Node3 R1 = new Node3(route1[0]);  
 Node3 p = R1;  
 for (int i=1; i<route1.length; i++){  
 p.next = new Node3(route1[i]);  
 p = p.next;  
 }  
  
 Node3 R2 = new Node3(route2[0]);  
 Node3 q = R2;  
 for (int i=1; i<route2.length; i++){  
 q.next = new Node3(route2[i]);  
 q = q.next;  
 }  
  
 System.*out*.println("Route No.1: Estimated Distance: "+distance1+" km Estimated Time: "+estimatedTime1+" mins");  
 R1.display(R1);  
 System.*out*.println("Arrived At Destination");  
  
 System.*out*.println();  
  
 System.*out*.println("Route No.2: Estimated Distance: "+distance2+" km Estimated Time: "+estimatedTime2+" mins");  
 R1.display(R2);  
 System.*out*.println("Arrived At Destination");  
 System.*out*.println();  
  
  
 if (estimatedTime1<estimatedTime2){  
 System.*out*.println("Route No.1 is shortest");  
 } else {  
 System.*out*.println("Route No.2 is shortest");  
 }  
  
 }  
}

**Output:**

****

**Task#03**

Question statement

Create a generic class and implements the following operations:

Task # 1:Implement the isEmpty() method in the Linked List

Task # 2:Implement the getSize() method in the Linked List

Task # 3:Implement the insertAtLast() method in the Linked List

Task # 4:Implement the insertAtLast() method in the Linked List

Task # 5:Implement the insertAtPosition() method in the Linked List

Task # 6:Implement the delete First() method in the Linked List

Task # 7:Implement the deleteLast() method in the Linked List

Task # 8:Implement the deleteAtPosition() method in the Linked List

Task # 9:Implement the search() method in the Linked List to check whether the element exists in the list or not.

Task # 10:delete element by value; implement deleteValue(int value)

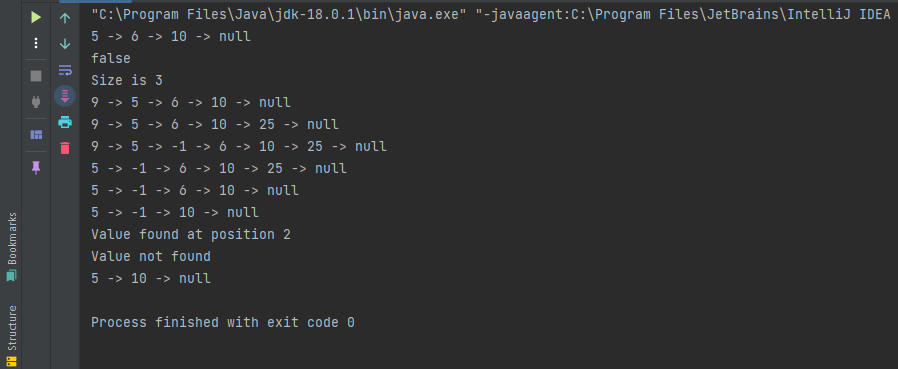
Task # 11: display method

# Q3.Java

**Code:**

class Node2<K> {  
 K val;  
 Node2<K> next;  
  
 public Node2(K val) {  
 this.val = val;  
 }  
}  
class Generic<S>{  
  
  
 public int getSize(Node2<S> head){  
 int count = 0;  
 for (Node2<S> p = head; p!=null; p=p.next){  
 count++;  
 }  
 return count;  
 } // end of class getSize()  
  
 public void display(Node2<S> head){  
 for (Node2<S> p = head; p!=null; p=p.next){  
 if (p.next==null)  
 System.*out*.print(p.val+" -> null");  
 else  
 System.*out*.print(p.val+" -> ");  
 } // end of for loop  
 System.*out*.println();  
 } // end of display() method  
  
 public boolean isEmpty(Node2<S> head){  
 int length = getSize(head);  
 if (head==null){  
 return true;  
 } else {  
 return false;  
 }  
 } // end of isEmpty()  
  
 public Node2<S> insertAtFirst(Node2<S> head, S val){  
 Node2<S> node = new Node2<S>(val);  
 node.next = head;  
 head = node;  
 return head;  
 } // end of insertAtFirst()  
  
 public Node2<S> insertAtLast(Node2<S> head, S val){  
 Node2<S> node = new Node2<S>(val);  
 Node2<S> p = head;  
 for (p = head; p.next!=null; p=p.next){  
  
 } // end of for loop  
 p.next = node;  
 return head;  
 } // end of insertAtLast()  
  
 public Node2<S> insertAtPosition(Node2<S> head, int pos, S val){  
 Node2<S> node = new Node2<S>(val);  
 int count = 1;  
 Node2<S> p = head;  
 for (p = head; p.next!=null; p=p.next){  
 if (count==pos-1){  
 node.next = p.next;  
 p.next = node;  
 }  
 count++;  
 } // end of for loop  
 return head;  
 }  
  
 public Node2<S> deleteFirst(Node2<S> head){  
 Node2<S> p = head;  
 head = p.next;  
 return head;  
 } // end of deleteFirst()  
  
 public Node2<S> deleteLast(Node2<S> head){  
 Node2<S> p = head;  
 for (p = head; p.next.next!=null; p=p.next){  
  
 }  
 p.next = null;  
 return head;  
 } // end of deleteLast()  
  
 public Node2<S> deleteAtPosition(Node2<S> head, int pos){  
 int count = 1;  
 Node2<S> p = head;  
 for (p = head; p!=null; p=p.next){  
 if (pos==count+1){  
 p.next = p.next.next;  
 }  
 count++;  
 }  
 return head;  
 } // end of deleteAtPosition()  
  
 public void search(Node2<S> head, S target){  
 int pos = 1;  
 boolean flag = false;  
 Node2<S> p;  
 for (p = head; p!=null; p=p.next){  
 if (p.val==target){  
 flag = true;  
 break;  
 }  
 pos++;  
 }  
  
 if (flag)  
 System.*out*.println("Value found at position "+pos);  
 else  
 System.*out*.println("Value not found");  
  
 } // end of search()  
  
 public Node2<S> deleteValue(Node2<S> head, S val){  
 Node2<S> p;  
 for (p = head; p.next!=null; p=p.next){  
 if (p.next.val==val){  
 p.next = p.next.next;  
 }  
 }  
 return head;  
 }  
} // end of class Generic  
  
public class Q3 {  
  
 public static void main(String[] args) {  
  
 Generic<Integer> generic = new Generic<>();  
  
 Node2<Integer> head = new Node2<>(5);  
 Node2<Integer> p = head;  
 p.next = new Node2<>(6);  
 p = p.next;  
 p.next = new Node2<>(10);  
 p = p.next;  
  
 generic.display(head);  
  
 // isEmpty()  
 System.*out*.println(generic.isEmpty(head));  
 // getSize()  
 System.*out*.println("Size is "+generic.getSize(head));  
  
 Node2<Integer> a = generic.insertAtFirst(head, 9);  
 generic.display(a);  
 head = a;  
  
 Node2<Integer>b = generic.insertAtLast(head, 25);  
 generic.display(b);  
  
 Node2<Integer> c = generic.insertAtPosition(head, 3, -1);  
 generic.display(c);  
  
 Node2<Integer> d = generic.deleteFirst(head);  
 generic.display(d);  
 head = d;  
  
 Node2<Integer> e = generic.deleteLast(head);  
 generic.display(e);  
  
 Node2<Integer> f = generic.deleteAtPosition(head, 3);  
 generic.display(f);  
  
 generic.search(head, -1);  
 generic.search(head, -2);  
  
 Node2<Integer> h = generic.deleteValue(head, -1);  
 generic.display(h);  
  
 } // end of main() method  
} // end of program

**Output:**

****