PUSAT PENGAJIAN SAINS KOMPUTER

(School of Computer Sciences)

SEMESTER II 2022/2023

HELAIAN KULIT TUGASAN

(ASSIGNMENT COVER SHEET)

Nama *(Name)* 1) MUHAMMAD IRFAN BIN ZAINAL 2) 3) 4)	Nombor Matrik *(Matric Number)* 1) 159147 2) 3) 4)

Masa/Kumpulan Tutorial (Jika Berkenaan) *(Tutorial Time/Group (If Applicable))* -	Tutor (Jika Berkenaan) *(Tutor (If Applicable))* -	Tarikh Akhir Penyerahan Tugasan *(Date Assignment is Due)* 10 July 23

Kod Kursus *(Course Code)* CPT212	Tajuk Kursus *(Course Title)* REKA BENTUK & ANALISIS ALGORITMA *(DESIGN & ANALYSIS OF ALGORITHMS)*	No./Tajuk Tugasan *(Assignment No./Title)* TUGASAN II – GRAPHS ALGORITHM AND APPLICATION *(ASSIGNMENT II –ALGORITMA GRAF DN APLIKASI*

Diterima Oleh: Tandatangan Staf *(Received By: Staff Signature)*	Tarikh/Masa Diterima *(Date/Time Lodged)*

Pengisytiharan

Saya/~~Kami*~~ mengesahkan bahawa tugasan ini adalah kerja saya/~~kami~~* seluruhnya kecuali apabila saya/~~kami~~* memberikan rujukan lengkap kepada kerja yang dilakukan oleh orang lain, dan bahan di dalam tugasan ini tidak pernah diserahkan untuk penilaian dalam mana-mana kursus pengajian formal yang lain.
Declaration

I/~~We~~ certify that this assignment is entirely my~~/our~~* own work, except where I/~~we~~* have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any formal course of study.*
*Potong mana yang tidak berkenaan (bergantung pada sama ada tugasan individu atau kumpulan).

Delete where applicable (depends on whether individual or group assignment).

Tandatangan Pelajar :
(Signature(s) of Student) irfan

Gred *(Grade)*	Pemeriksa *(Marker)* Prof. Dr. Abdullah Zawawi Talib (Honorary Professor))
Ulasan Penanda *(Marker's Comments)*	

Contents

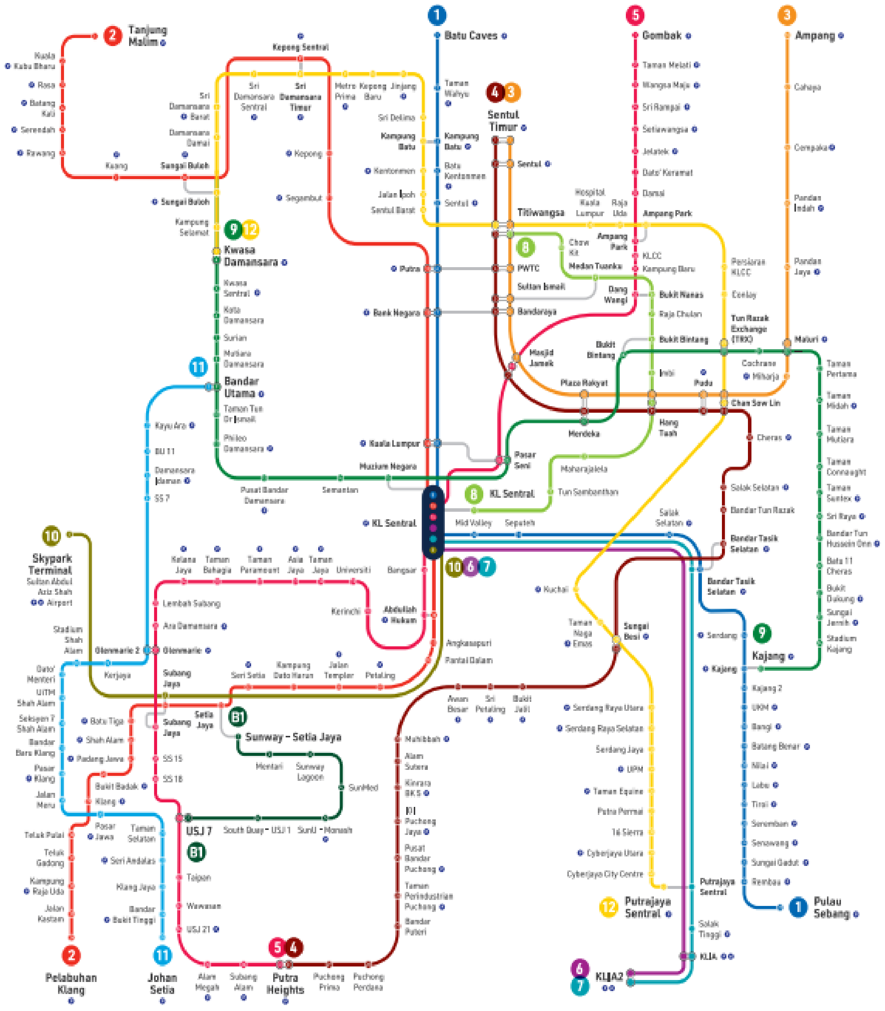
1.0 INTRODUCTION

For this assignment, we were tasked to implement a path finding application program in Java using depth-first search algorithms. To do this, we first must read the graph input from text file. The ADT used to store the graph is Linked List. The program will then show the available node that has been stored. User will be prompted to choose which node to start searching from and the destination node. The program prints out the path found with its weight. If no path was found, the program will display "no path found". User may input other nodes if they wish too.

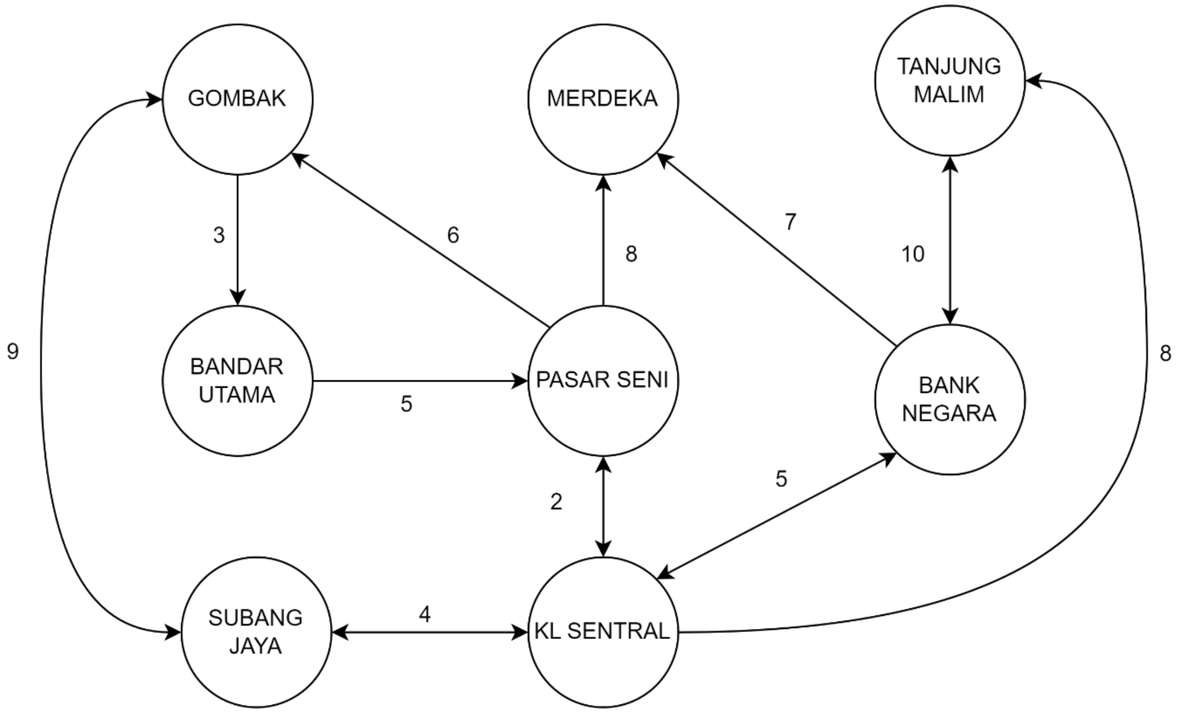
2.0 PROGRAM DETAILS

To make the program more user friendly, we decided to apply the DFS in a real-world scenario. For this implementation, the program is based on KL LRT route. Users can choose which stations they want to start from and their destination. DFS will be done to find the path from start station to the desired station. If the path was found, the program will display the path the user needed to follow arrive to their destination. Do note that, the DFS will only be able to search for only one path even if multiple paths were present to get to the destination. The path found by the algorithm also does not necessarily a shortest path possible.

The graph to be generated is based on the below map:



The above map is then simplified into the following graph:



In this case, the path cost will be represented as the ticket cost. Keep in mind that, the above graph is a modified RapidKL LRT map and not an accurate representation of the real map. The weight of the path is just arbitrarily chosen.

3.0 PROGRAM LIMITATION

- The program would not be able to return the lowest cost path.
- The program will only be able to return one path even if multiple paths is present.
- The filepath may need to be change for it to work.

4.0 DESIGN DECISION

Overall structure

This program includes the following java class: main, userInterface, fileReader, search, graph, vertex, and edges and one input file: LRT_Route.txt

Each subfunction of this program is separated into different java class. This way, we can avoid any confusion during debugging process and have a clean workspace thus improving our work efficiencies.

Building the graph

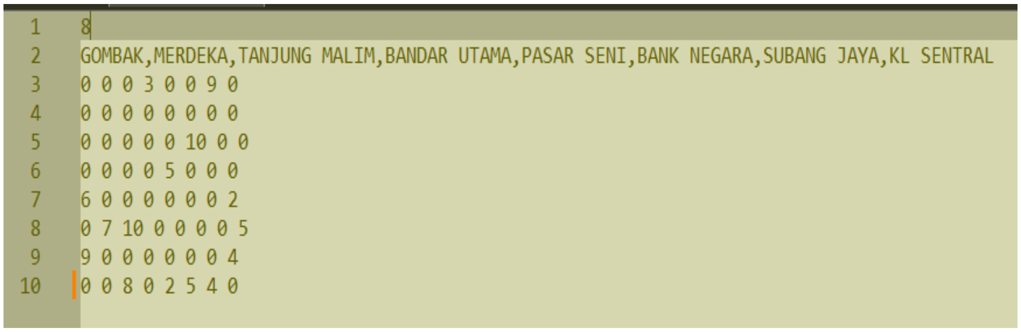
Class involved: GRAPH, EDGE, VERTEX

The graph will be implemented by using linked list to make sure the time complexity for the search is as fast as possible. Each node of linked list is consisted of vertex and the connected edges. The edges will then store the weight of the path and the connected vertex. To build the graph, the function addVertex will first called to initialize the node. After the nodes has been initialized, addEdge function will be called to add connected edge and its weight to the appropriate vertex.

Reading the input file

Class involved: FILEREADER

File input name: LRT_Route.txt



```
8
GOMBAK,MERDEKA,TANJUNG MALIM,BANDAR UTAMA,PASAR SENI,BANK NEGARA,SUBANG JAYA,KL SENTRAL
0 0 0 3 0 0 9 0
0 0 0 0 0 0 0 0
0 0 0 0 0 10 0 0
0 0 0 0 5 0 0 0
6 0 0 0 0 0 0 2
0 7 10 0 0 0 0 5
9 0 0 0 0 0 0 4
0 0 8 0 2 5 4 0
```

Above is the format of the file input in the text file. The first line will include the number of nodes to be read. The second line consist of the name of each node, in this case the name of LRT stations. For the rest of the line, it consists of the connection between each vertex in the form of adjacencies matrix with its weight. Although it is inefficient to use adjacencies matrix to represent the graph due to high space complexity, in this case it makes it easier for the program to read chunk of data at once. The adjacencies matrix can easily be represented in a 2D array. By using this 2D array, it will then make it easy to build the graph. The graph will then be built in a linked list ADT so there will be no problem with the complexity of the search.

User application

Class involved: USERINTERFACES

This class include all the necessary menu interface to inform the user how to navigate the program. We try our best to make it as intuitive as possible for the user to avoid any complication. The menu will kept repeating itself until user exit the program themselves.

Error checking

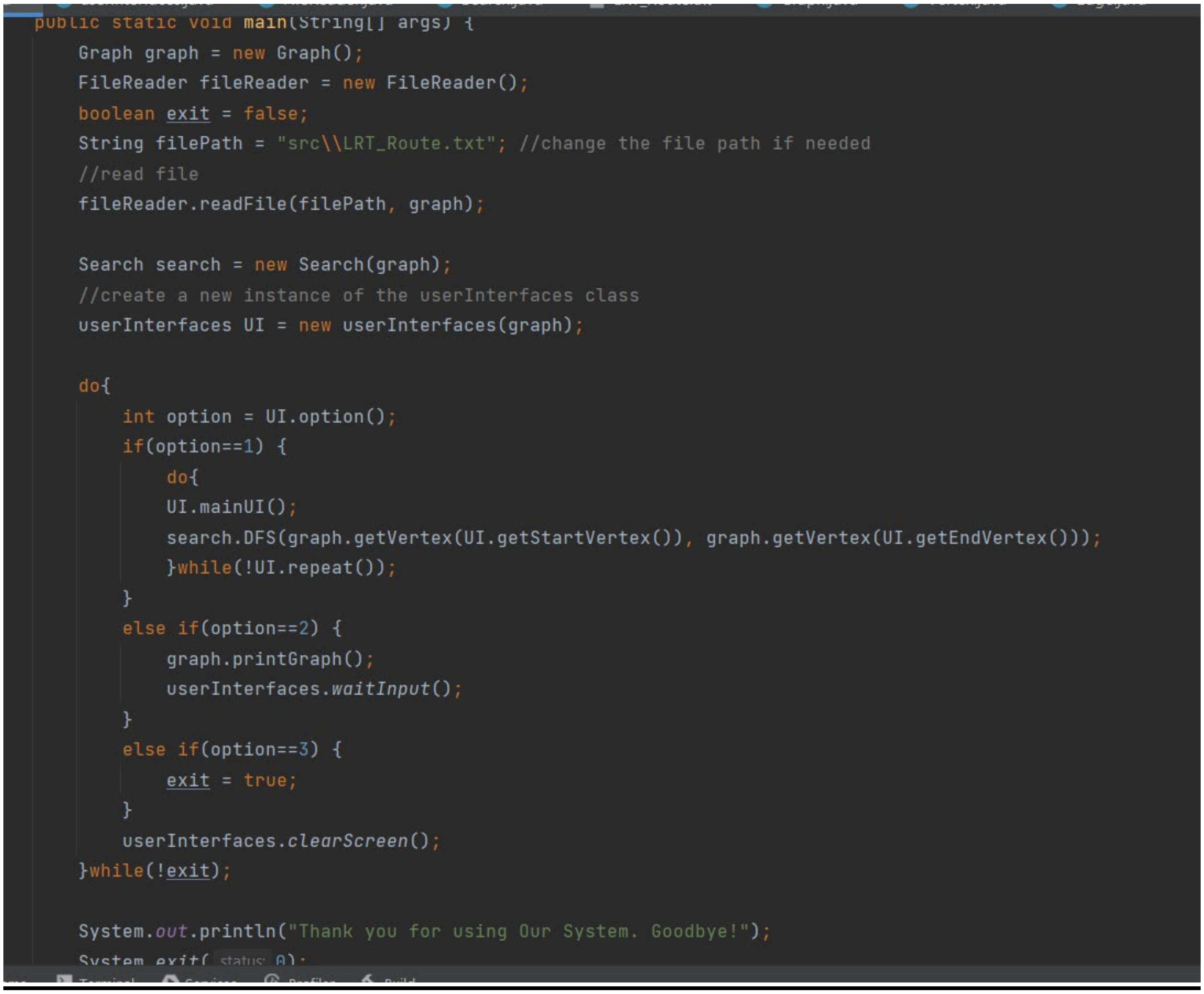
We also include input validation for any user input to avoid our program from breaking. The way the program been coded is that the input is not a case sensitive. As such, if the user input the station name in lowercase, the program will still works as intended. We also did intensive test run and try various combination to try to break the program but as of now, all the test is success, and the program was running without any problem.

Other information

The program was coded on InteliJ Ultimate IDE. In this IDE, the usage of ANSI escape code to clear the screen is not possible as such it was replace by printing 50 empty lines to ensure a clean user interface.

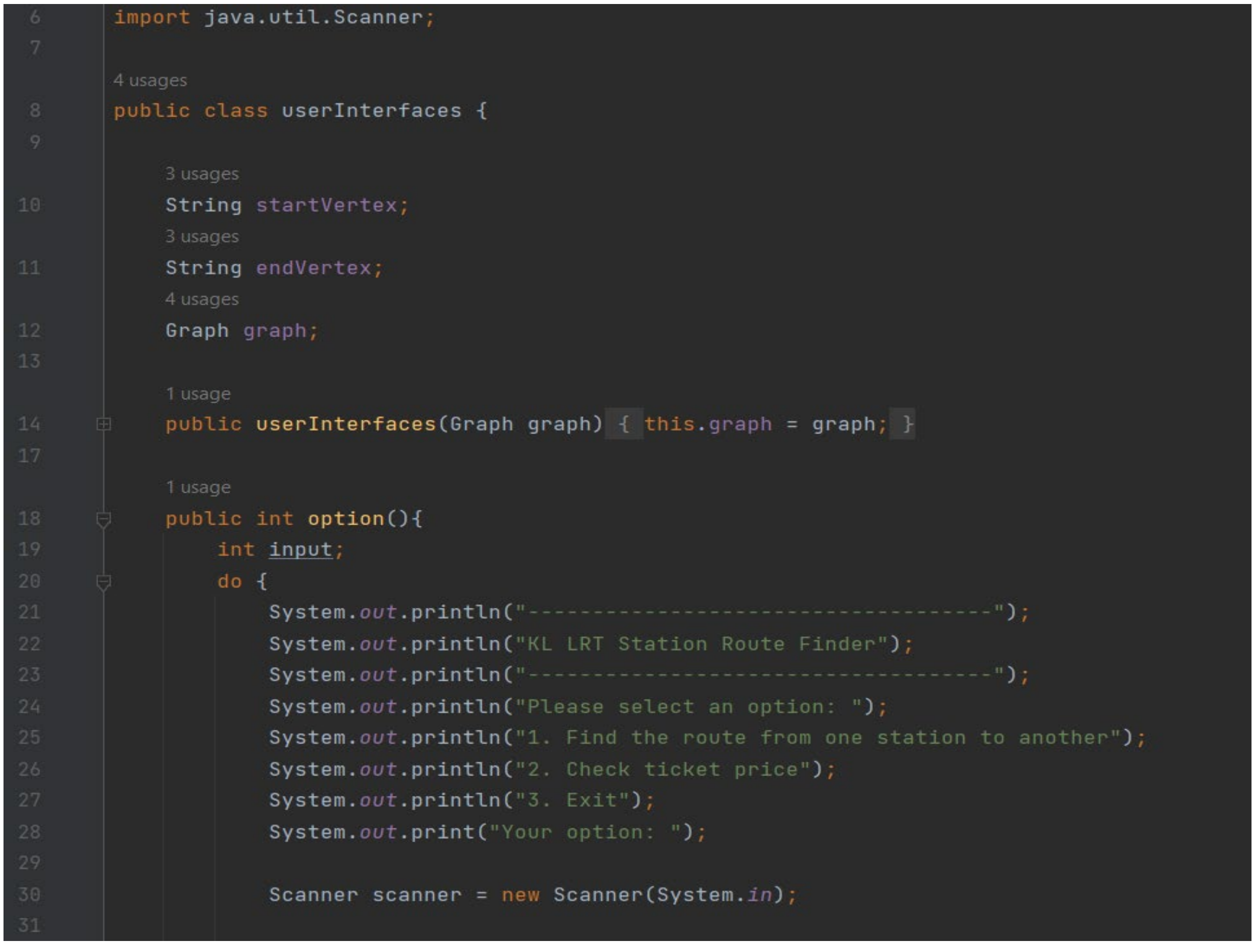
5.0 SOURCE CODE

Main Class

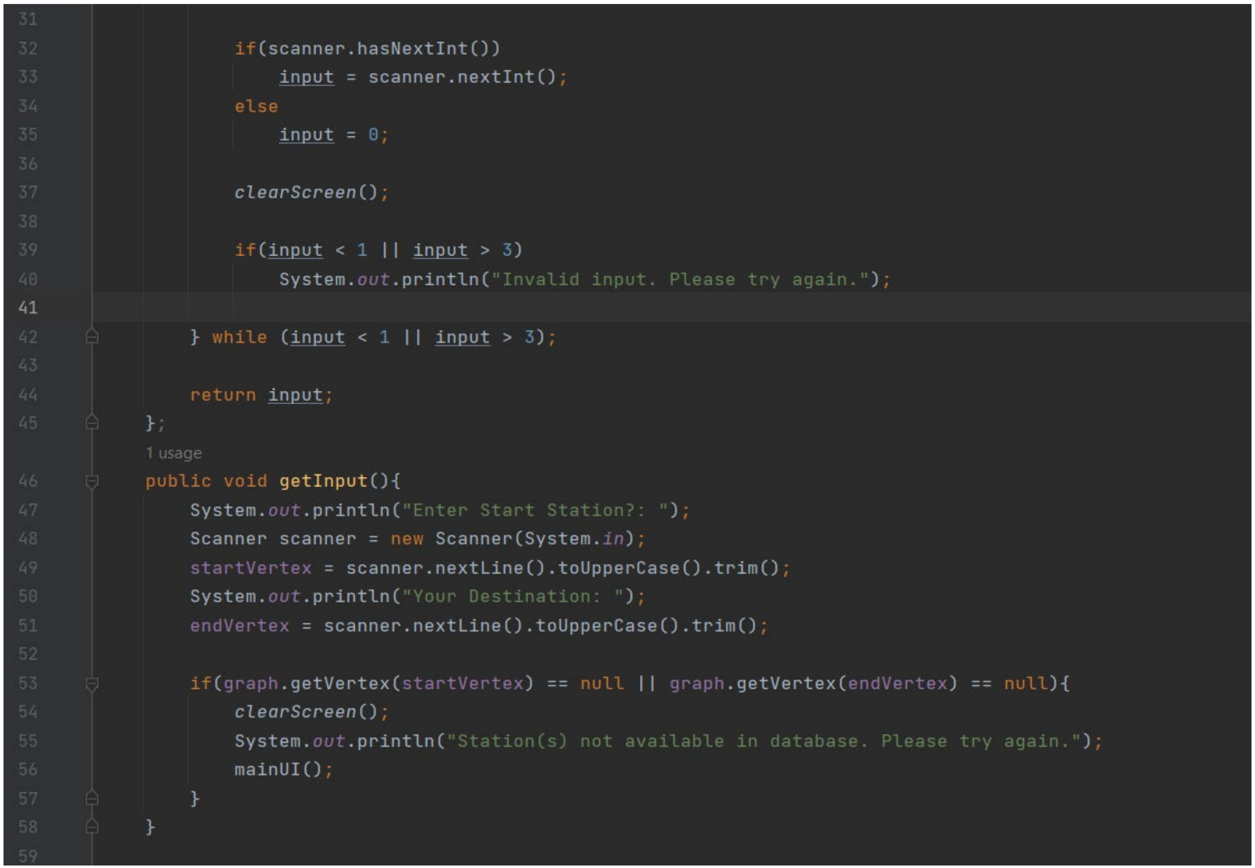


```java
public static void main(String[] args) {
    Graph graph = new Graph();
    FileReader fileReader = new FileReader();
    boolean exit = false;
    String filePath = "src\\LRT_Route.txt"; //change the file path if needed
    //read file
    fileReader.readFile(filePath, graph);

    Search search = new Search(graph);
    //create a new instance of the userInterfaces class
    userInterfaces UI = new userInterfaces(graph);

    do{
        int option = UI.option();
        if(option==1) {
            do{
            UI.mainUI();
            search.DFS(graph.getVertex(UI.getStartVertex()), graph.getVertex(UI.getEndVertex()));
            }while(!UI.repeat());
        }
        else if(option==2) {
            graph.printGraph();
            userInterfaces.waitInput();
        }
        else if(option==3) {
            exit = true;
        }
        userInterfaces.clearScreen();
    }while(!exit);

    System.out.println("Thank you for using Our System. Goodbye!");
    System.exit( status: 0);
```

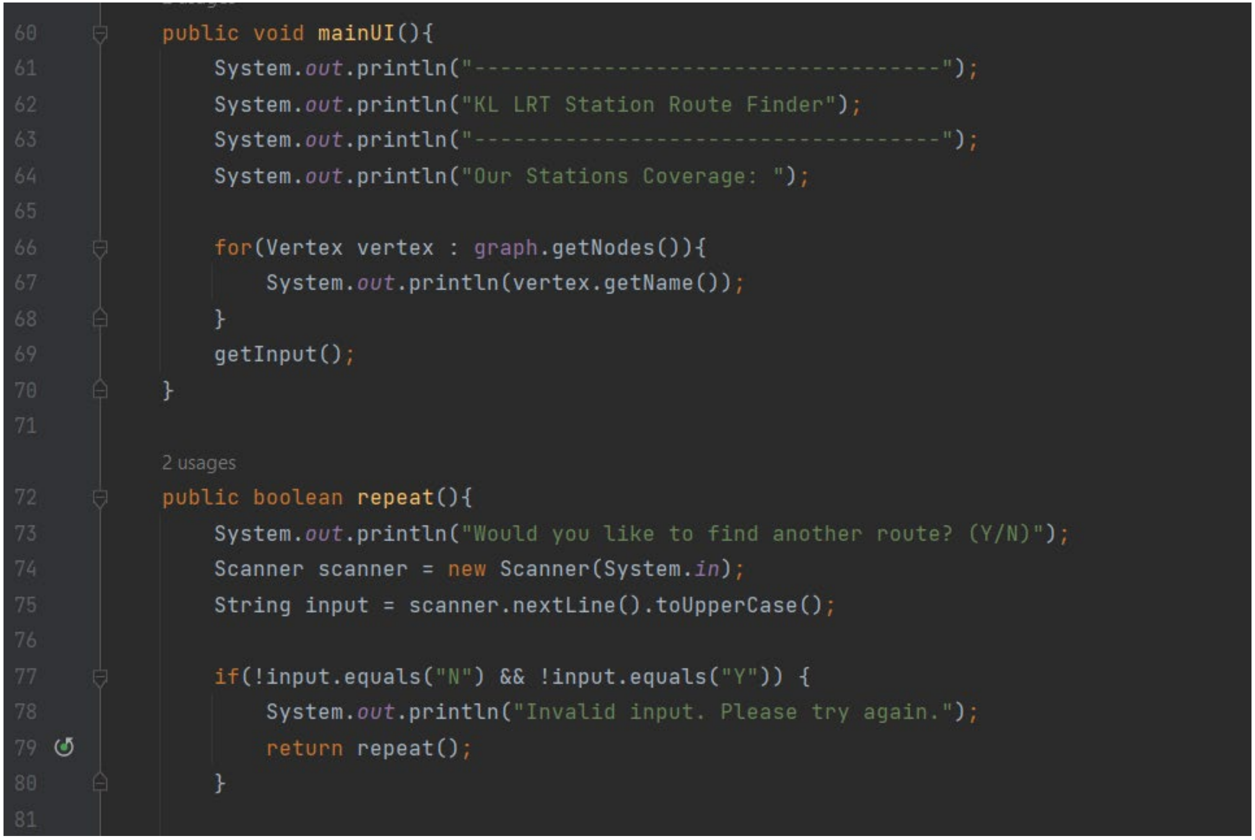
User Interfaces Class



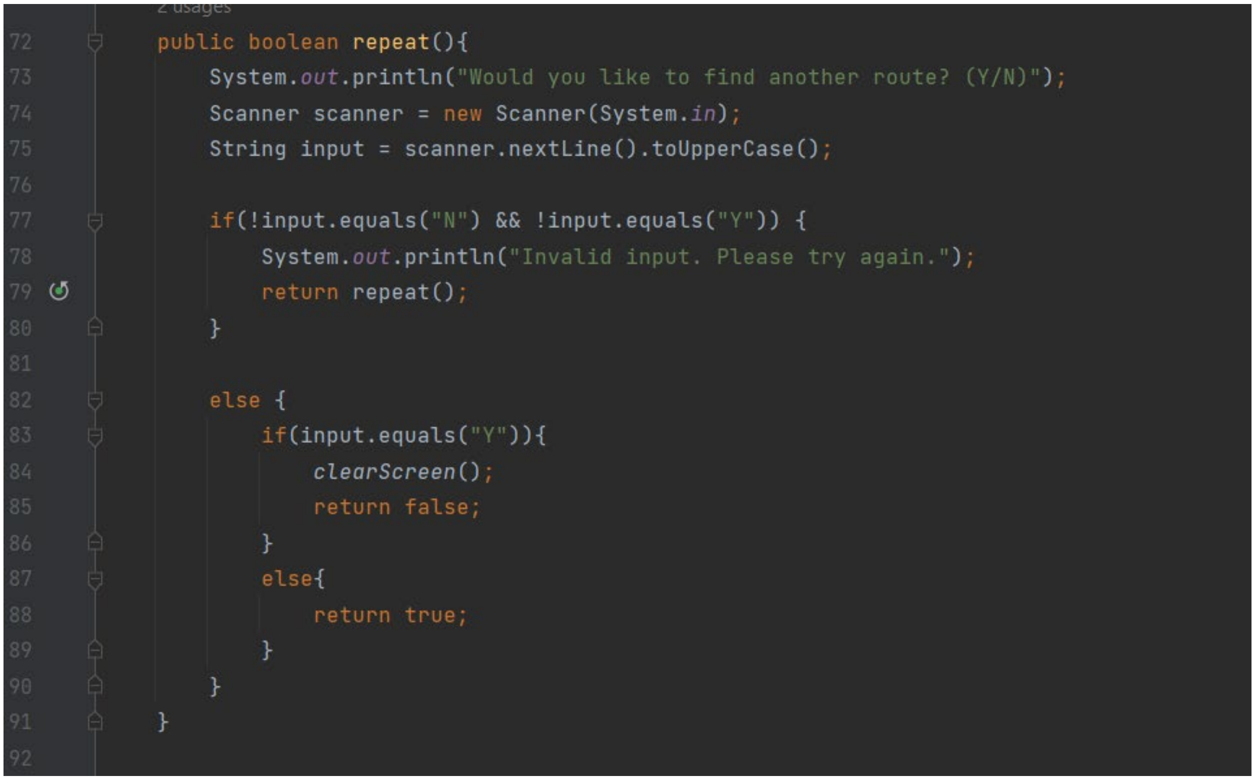
```java
import java.util.Scanner;

4 usages
public class userInterfaces {

    3 usages
    String startVertex;
    3 usages
    String endVertex;
    4 usages
    Graph graph;

    1 usage
    public userInterfaces(Graph graph) { this.graph = graph; }

    1 usage
    public int option(){
        int input;
        do {
            System.out.println("------------------------------------");
            System.out.println("KL LRT Station Route Finder");
            System.out.println("------------------------------------");
            System.out.println("Please select an option: ");
            System.out.println("1. Find the route from one station to another");
            System.out.println("2. Check ticket price");
            System.out.println("3. Exit");
            System.out.print("Your option: ");

            Scanner scanner = new Scanner(System.in);

```



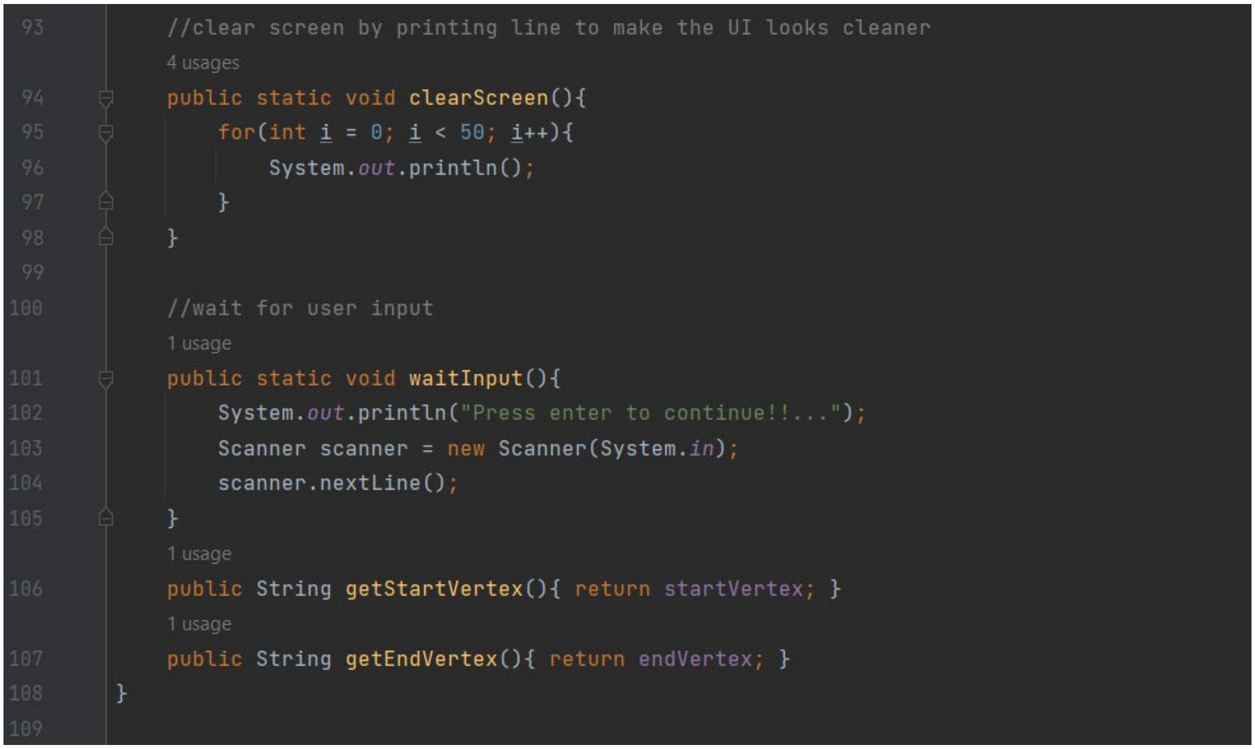
```
            if(scanner.hasNextInt())
                input = scanner.nextInt();
            else
                input = 0;

            clearScreen();

            if(input < 1 || input > 3)
                System.out.println("Invalid input. Please try again.");

        } while (input < 1 || input > 3);

        return input;
    };
    1 usage
    public void getInput(){
        System.out.println("Enter Start Station?: ");
        Scanner scanner = new Scanner(System.in);
        startVertex = scanner.nextLine().toUpperCase().trim();
        System.out.println("Your Destination: ");
        endVertex = scanner.nextLine().toUpperCase().trim();

        if(graph.getVertex(startVertex) == null || graph.getVertex(endVertex) == null){
            clearScreen();
            System.out.println("Station(s) not available in database. Please try again.");
            mainUI();
        }
    }

```



```
    public void mainUI(){
        System.out.println("------------------------------------");
        System.out.println("KL LRT Station Route Finder");
        System.out.println("------------------------------------");
        System.out.println("Our Stations Coverage: ");

        for(Vertex vertex : graph.getNodes()){
            System.out.println(vertex.getName());
        }
        getInput();
    }

    2 usages
    public boolean repeat(){
        System.out.println("Would you like to find another route? (Y/N)");
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine().toUpperCase();

        if(!input.equals("N") && !input.equals("Y")) {
            System.out.println("Invalid input. Please try again.");
            return repeat();
        }

```

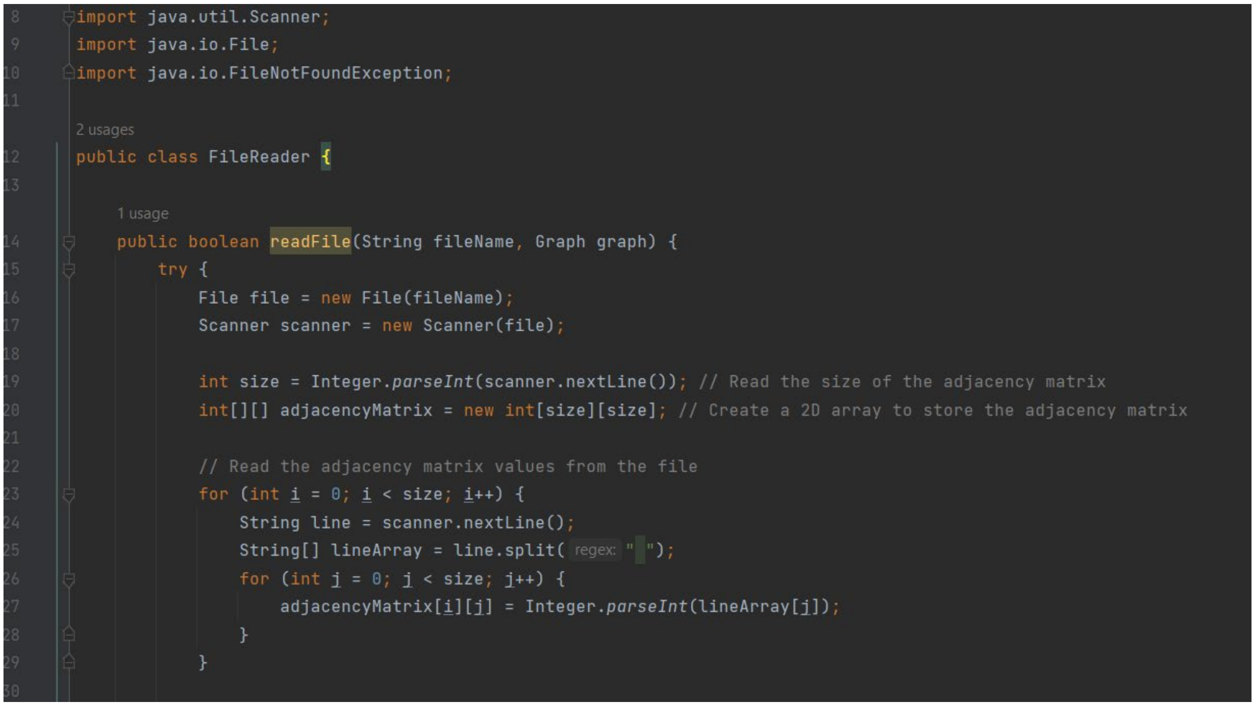


```
    public boolean repeat(){
        System.out.println("Would you like to find another route? (Y/N)");
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine().toUpperCase();

        if(!input.equals("N") && !input.equals("Y")) {
            System.out.println("Invalid input. Please try again.");
            return repeat();
        }

        else {
            if(input.equals("Y")){
                clearScreen();
                return false;
            }
            else{
                return true;
            }
        }
    }

```

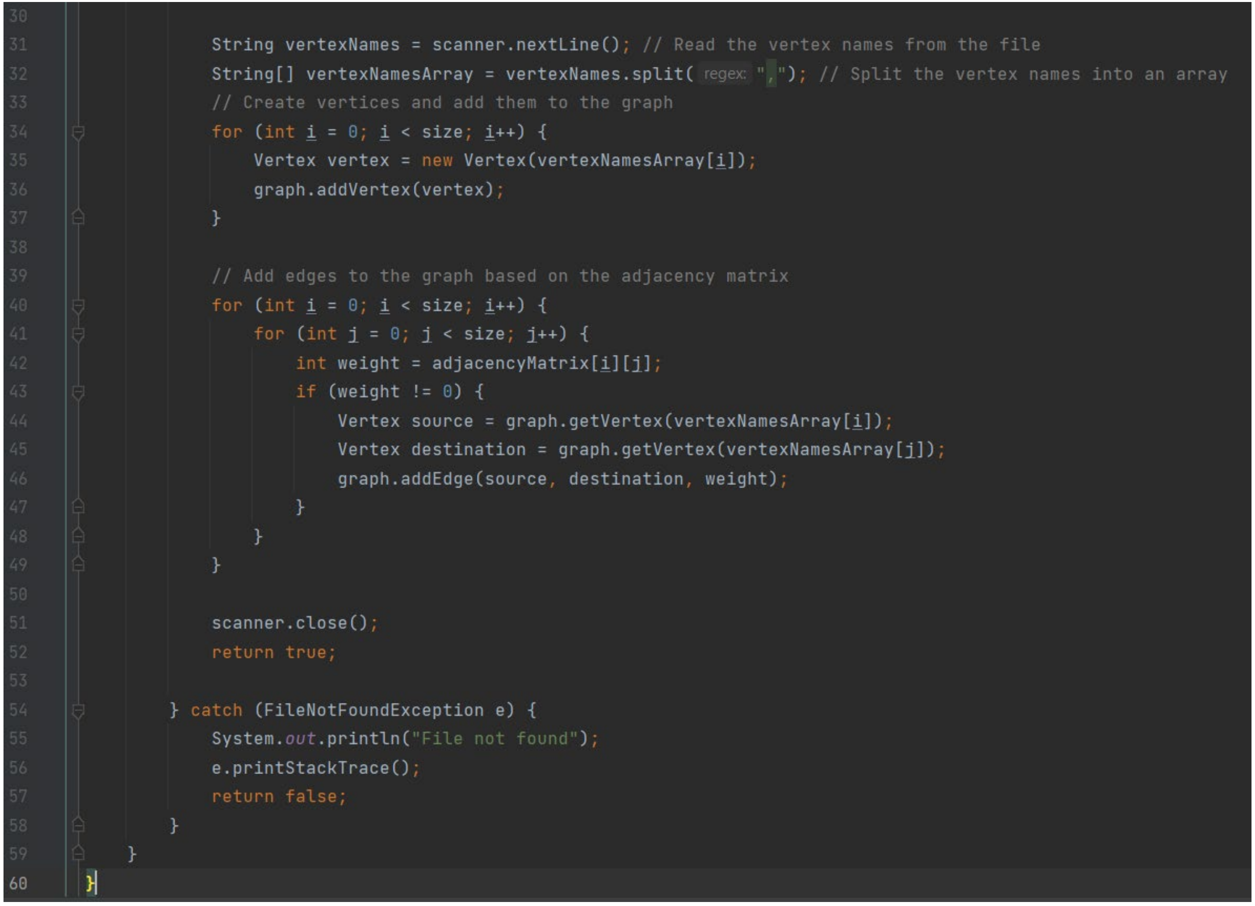


```
    //clear screen by printing line to make the UI looks cleaner
    public static void clearScreen(){
        for(int i = 0; i < 50; i++){
            System.out.println();
        }
    }

    //wait for user input
    public static void waitInput(){
        System.out.println("Press enter to continue!!...");
        Scanner scanner = new Scanner(System.in);
        scanner.nextLine();
    }
    public String getStartVertex(){ return startVertex; }
    public String getEndVertex(){ return endVertex; }
}

```

File Reader Class

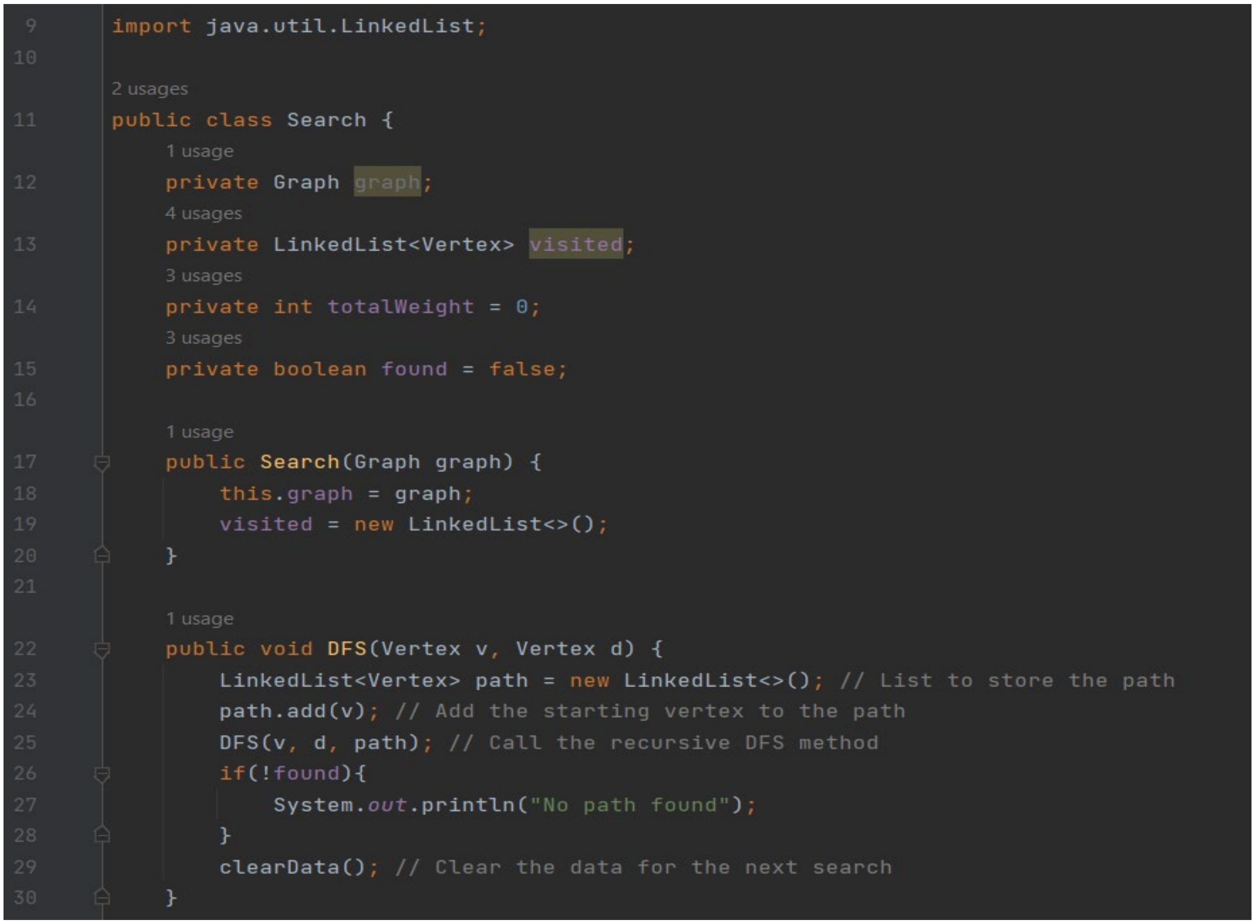


```java
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

2 usages
public class FileReader {

    1 usage
    public boolean readFile(String fileName, Graph graph) {
        try {
            File file = new File(fileName);
            Scanner scanner = new Scanner(file);

            int size = Integer.parseInt(scanner.nextLine()); // Read the size of the adjacency matrix
            int[][] adjacencyMatrix = new int[size][size]; // Create a 2D array to store the adjacency matrix

            // Read the adjacency matrix values from the file
            for (int i = 0; i < size; i++) {
                String line = scanner.nextLine();
                String[] lineArray = line.split( regex: " ");
                for (int j = 0; j < size; j++) {
                    adjacencyMatrix[i][j] = Integer.parseInt(lineArray[j]);
                }
            }

```

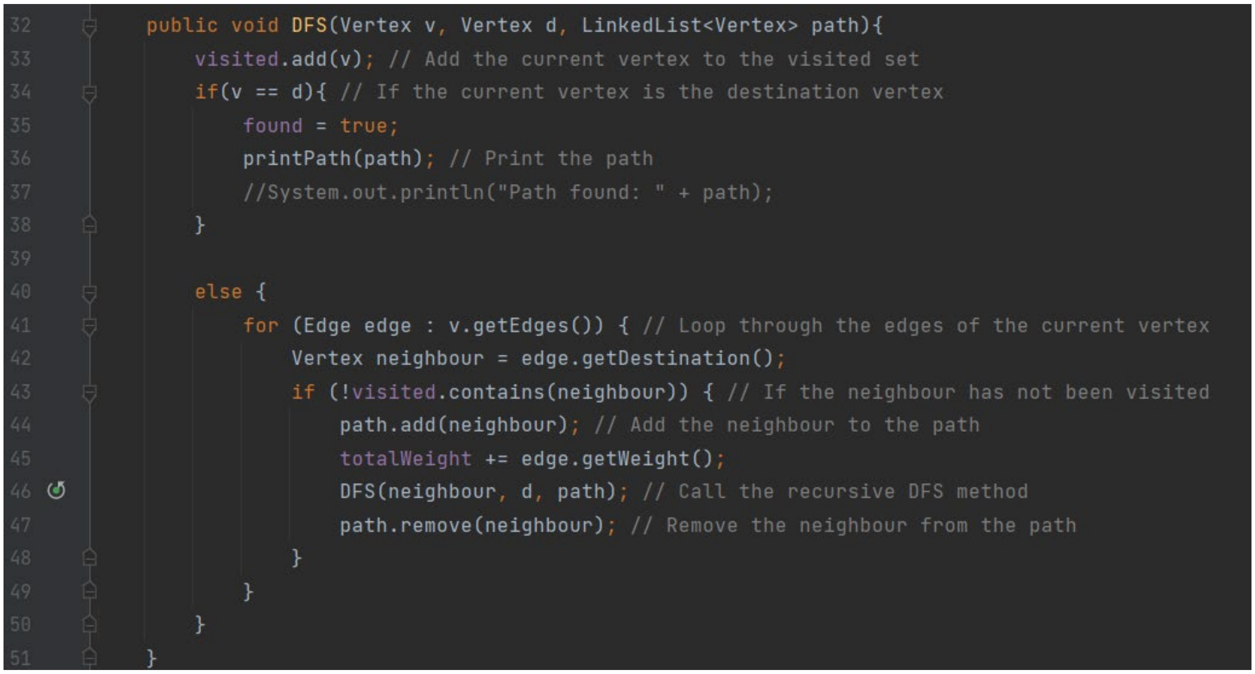


```java

            String vertexNames = scanner.nextLine(); // Read the vertex names from the file
            String[] vertexNamesArray = vertexNames.split( regex: ","); // Split the vertex names into an array
            // Create vertices and add them to the graph
            for (int i = 0; i < size; i++) {
                Vertex vertex = new Vertex(vertexNamesArray[i]);
                graph.addVertex(vertex);
            }

            // Add edges to the graph based on the adjacency matrix
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    int weight = adjacencyMatrix[i][j];
                    if (weight != 0) {
                        Vertex source = graph.getVertex(vertexNamesArray[i]);
                        Vertex destination = graph.getVertex(vertexNamesArray[j]);
                        graph.addEdge(source, destination, weight);
                    }
                }
            }

            scanner.close();
            return true;

        } catch (FileNotFoundException e) {
            System.out.println("File not found");
            e.printStackTrace();
            return false;
        }
    }
}
```

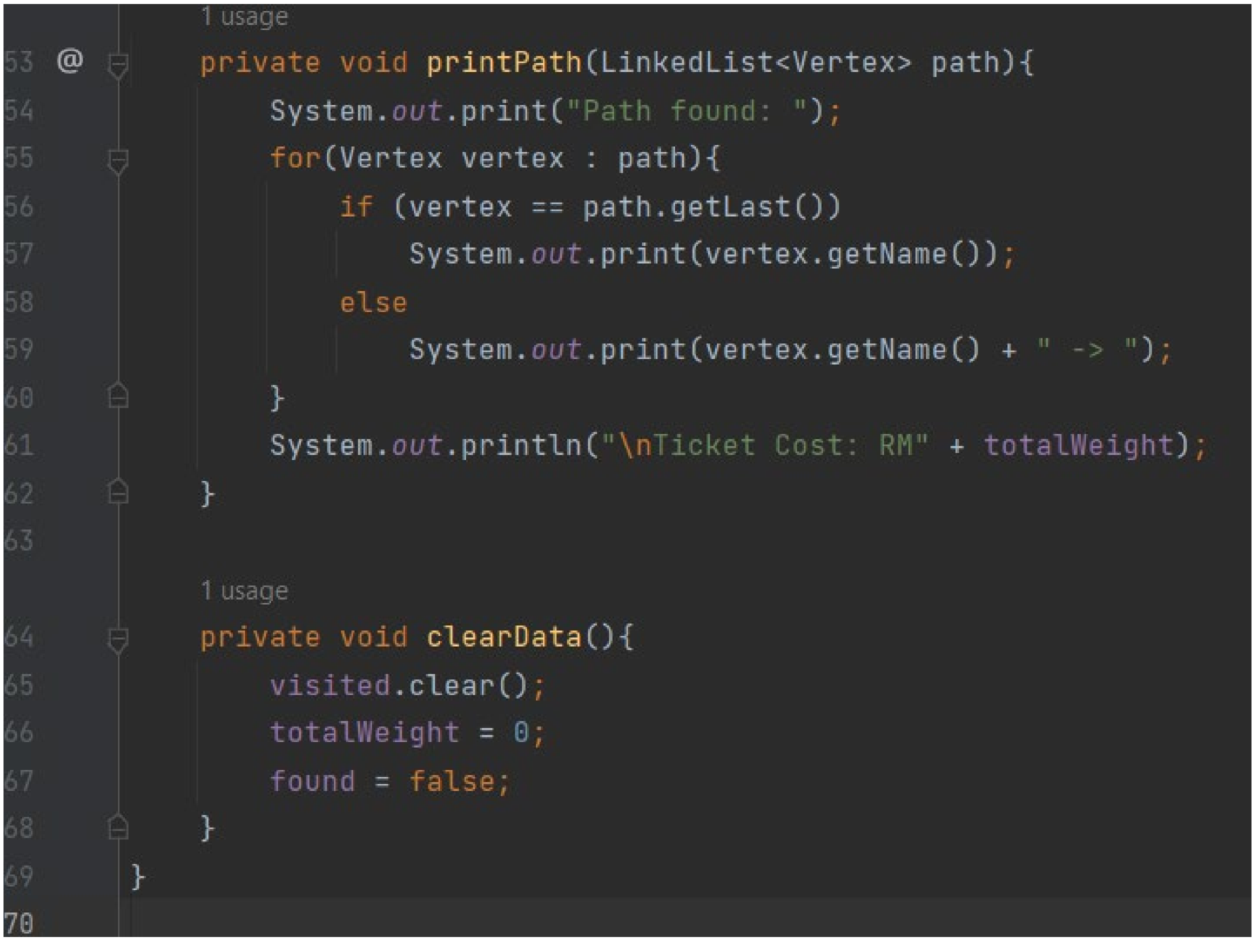
Search Class



```java
import java.util.LinkedList;

2 usages
public class Search {
    1 usage
    private Graph graph;
    4 usages
    private LinkedList<Vertex> visited;
    3 usages
    private int totalWeight = 0;
    3 usages
    private boolean found = false;

    1 usage
    public Search(Graph graph) {
        this.graph = graph;
        visited = new LinkedList<>();
    }

    1 usage
    public void DFS(Vertex v, Vertex d) {
        LinkedList<Vertex> path = new LinkedList<>(); // List to store the path
        path.add(v); // Add the starting vertex to the path
        DFS(v, d, path); // Call the recursive DFS method
        if(!found){
            System.out.println("No path found");
        }
        clearData(); // Clear the data for the next search
    }
```

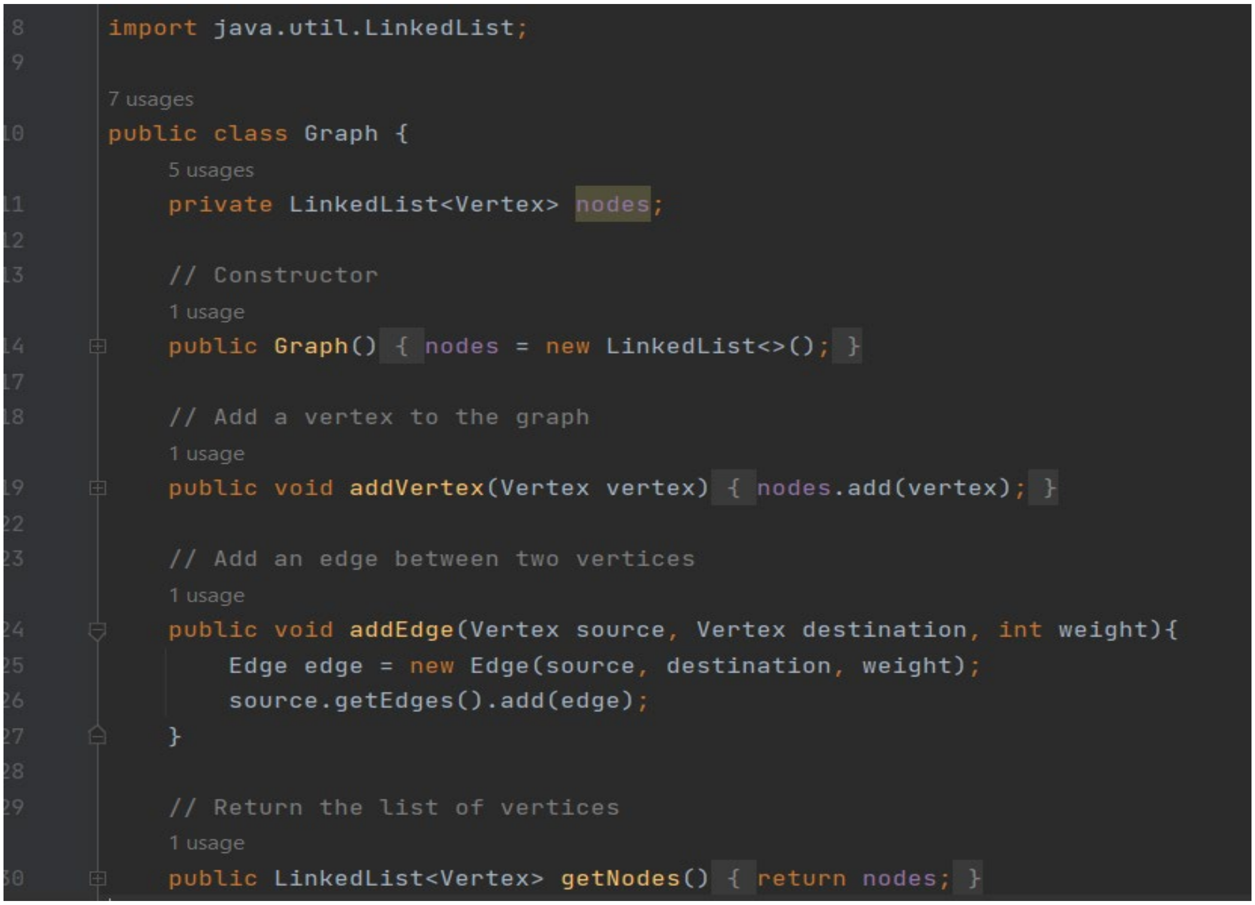


```java
    public void DFS(Vertex v, Vertex d, LinkedList<Vertex> path){
        visited.add(v); // Add the current vertex to the visited set
        if(v == d){ // If the current vertex is the destination vertex
            found = true;
            printPath(path); // Print the path
            //System.out.println("Path found: " + path);
        }

        else {
            for (Edge edge : v.getEdges()) { // Loop through the edges of the current vertex
                Vertex neighbour = edge.getDestination();
                if (!visited.contains(neighbour)) { // If the neighbour has not been visited
                    path.add(neighbour); // Add the neighbour to the path
                    totalWeight += edge.getWeight();
                    DFS(neighbour, d, path); // Call the recursive DFS method
                    path.remove(neighbour); // Remove the neighbour from the path
                }
            }
        }
    }
```

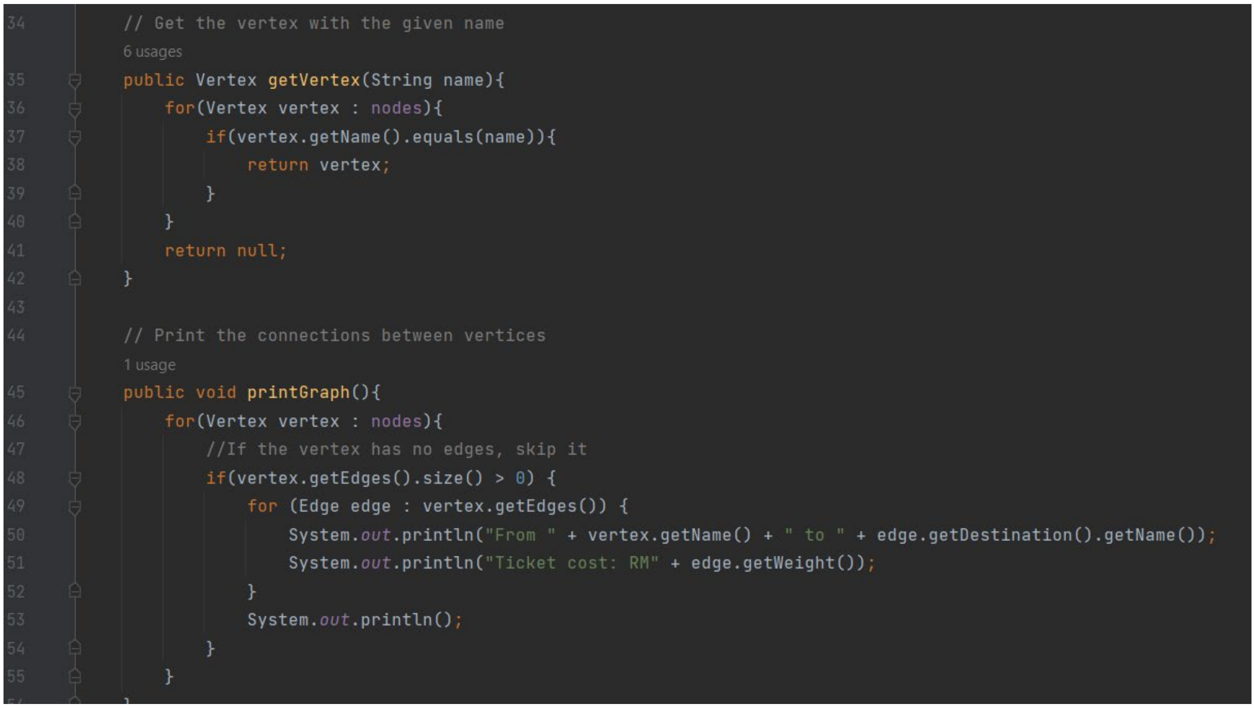


```java
    private void printPath(LinkedList<Vertex> path){
        System.out.print("Path found: ");
        for(Vertex vertex : path){
            if (vertex == path.getLast())
                System.out.print(vertex.getName());
            else
                System.out.print(vertex.getName() + " -> ");
        }
        System.out.println("\nTicket Cost: RM" + totalWeight);
    }

    private void clearData(){
        visited.clear();
        totalWeight = 0;
        found = false;
    }
}
```

Graph Class

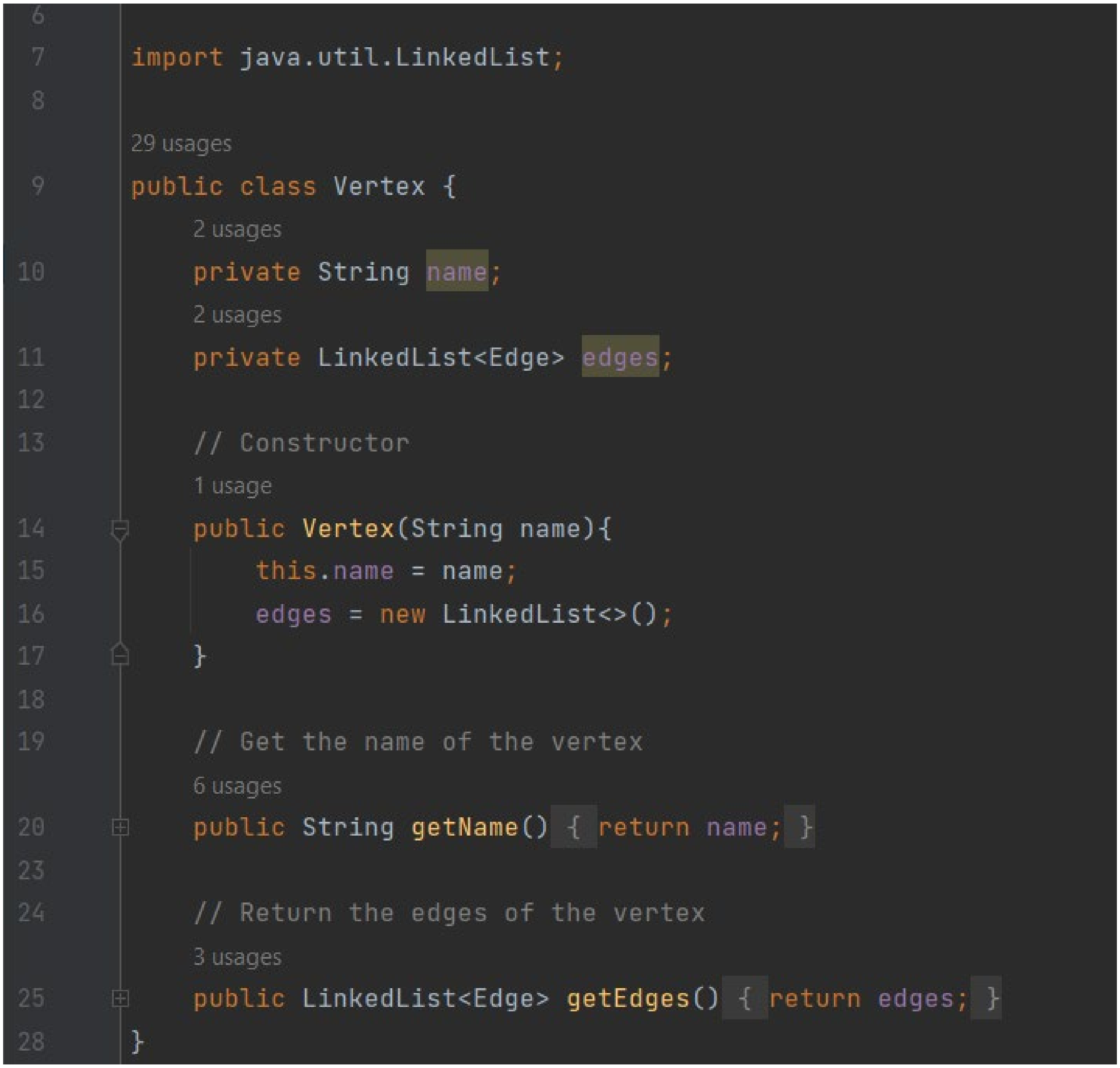


```java
import java.util.LinkedList;

7 usages
public class Graph {
    5 usages
    private LinkedList<Vertex> nodes;

    // Constructor
    1 usage
    public Graph() { nodes = new LinkedList<>(); }

    // Add a vertex to the graph
    1 usage
    public void addVertex(Vertex vertex) { nodes.add(vertex); }

    // Add an edge between two vertices
    1 usage
    public void addEdge(Vertex source, Vertex destination, int weight){
        Edge edge = new Edge(source, destination, weight);
        source.getEdges().add(edge);
    }

    // Return the list of vertices
    1 usage
    public LinkedList<Vertex> getNodes() { return nodes; }
```



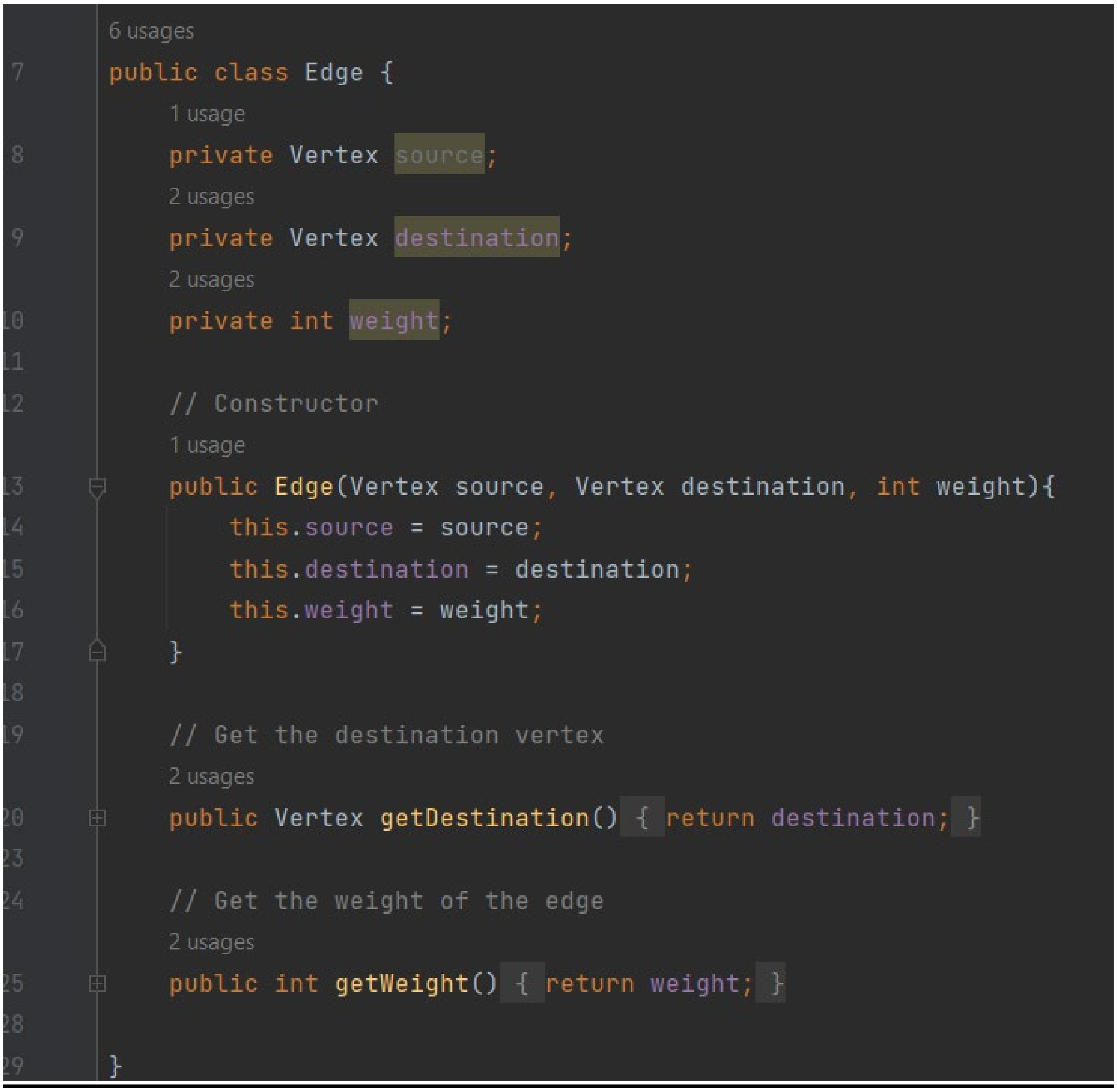
```java
    // Get the vertex with the given name
    6 usages
    public Vertex getVertex(String name){
        for(Vertex vertex : nodes){
            if(vertex.getName().equals(name)){
                return vertex;
            }
        }
        return null;
    }

    // Print the connections between vertices
    1 usage
    public void printGraph(){
        for(Vertex vertex : nodes){
            //If the vertex has no edges, skip it
            if(vertex.getEdges().size() > 0) {
                for (Edge edge : vertex.getEdges()) {
                    System.out.println("From " + vertex.getName() + " to " + edge.getDestination().getName());
                    System.out.println("Ticket cost: RM" + edge.getWeight());
                }
                System.out.println();
            }
        }
```

Vertex Class



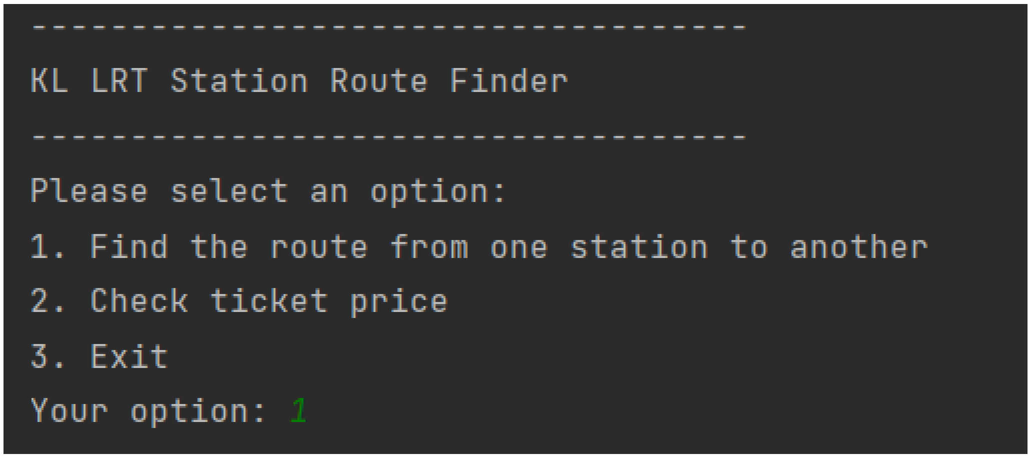
```java
import java.util.LinkedList;

public class Vertex {
    private String name;
    private LinkedList<Edge> edges;

    // Constructor
    public Vertex(String name){
        this.name = name;
        edges = new LinkedList<>();
    }

    // Get the name of the vertex
    public String getName() { return name; }

    // Return the edges of the vertex
    public LinkedList<Edge> getEdges() { return edges; }
}
```

Edge Class

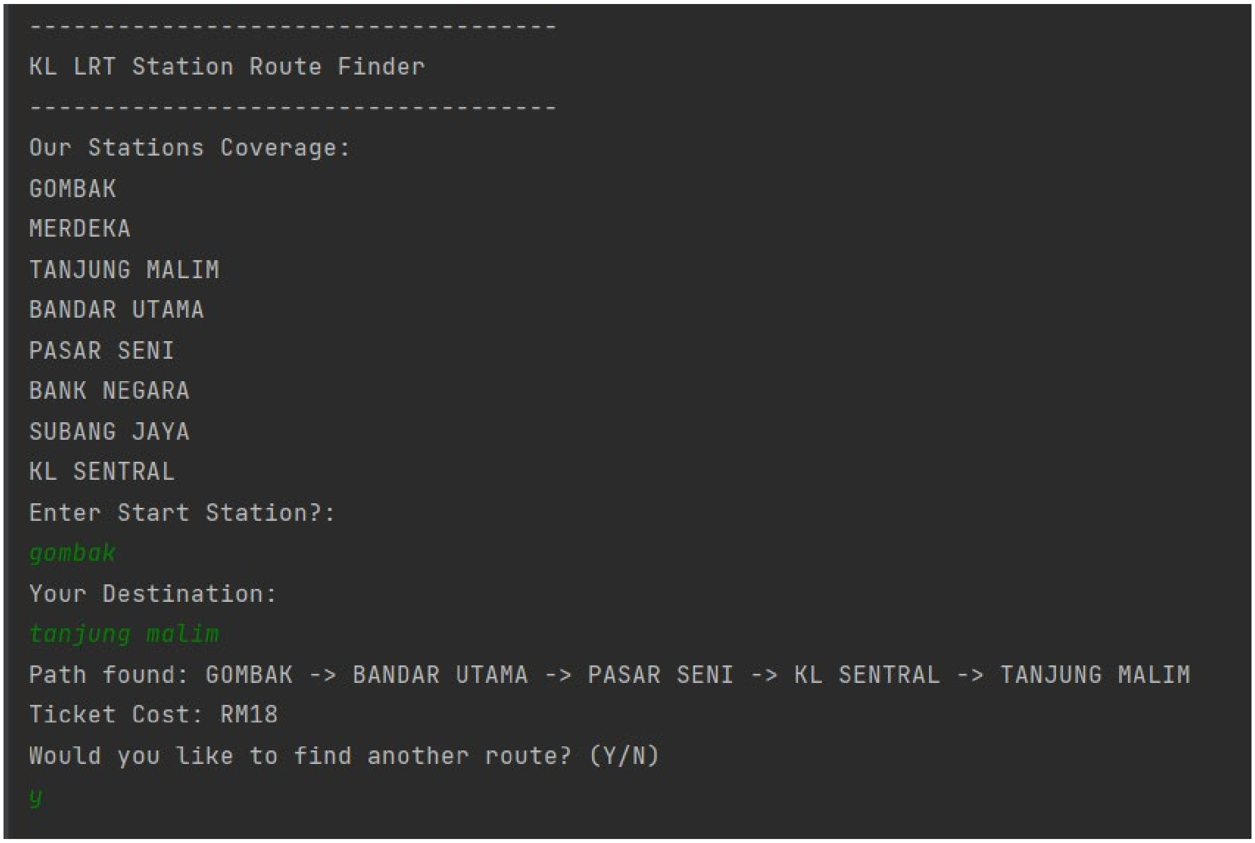


```
6 usages
public class Edge {
    1 usage
    private Vertex source;
    2 usages
    private Vertex destination;
    2 usages
    private int weight;

    // Constructor
    1 usage
    public Edge(Vertex source, Vertex destination, int weight){
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }

    // Get the destination vertex
    2 usages
    public Vertex getDestination() { return destination; }

    // Get the weight of the edge
    2 usages
    public int getWeight() { return weight; }

}
```

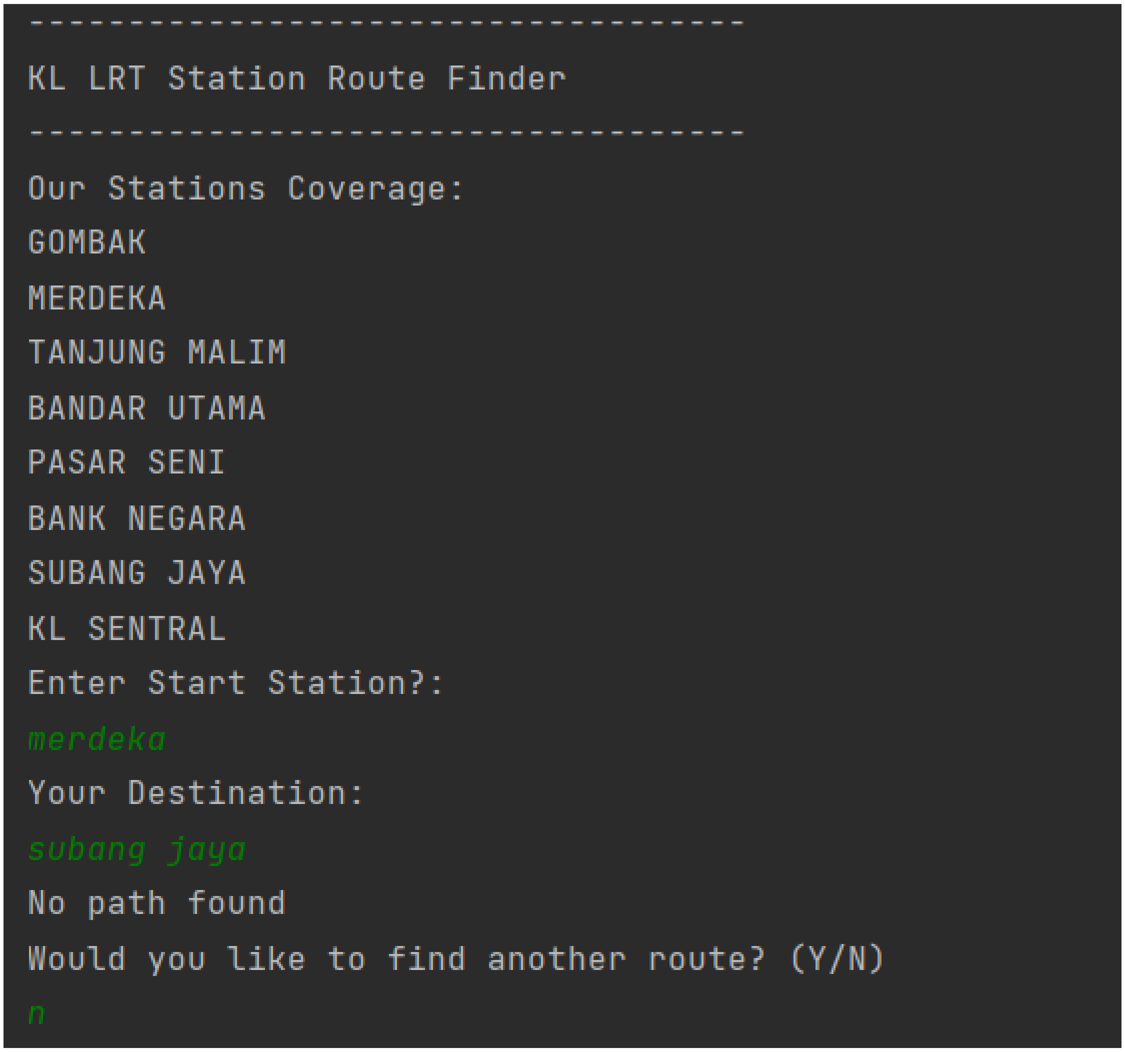
6.0 TEST CASE



```
------------------------------------
KL LRT Station Route Finder
------------------------------------
Please select an option:
1. Find the route from one station to another
2. Check ticket price
3. Exit
Your option: 1
```

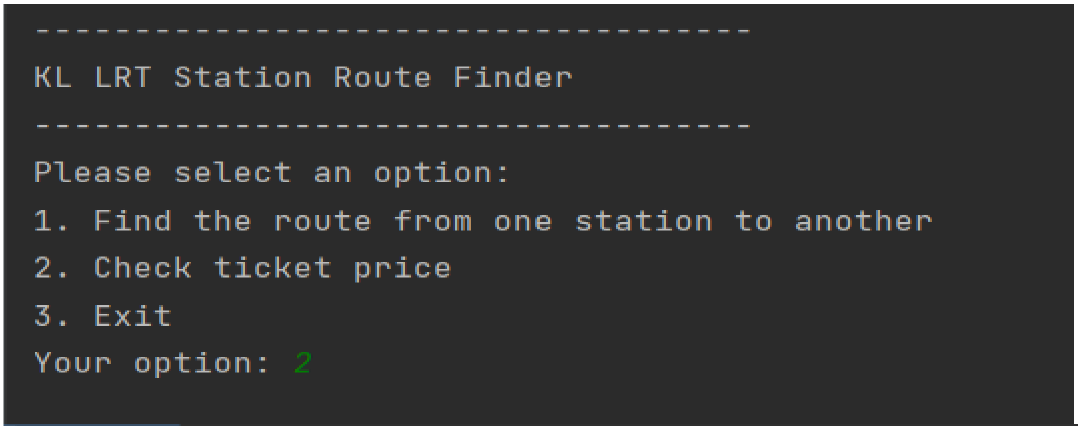


```
------------------------------------
KL LRT Station Route Finder
------------------------------------
Our Stations Coverage:
GOMBAK
MERDEKA
TANJUNG MALIM
BANDAR UTAMA
PASAR SENI
BANK NEGARA
SUBANG JAYA
KL SENTRAL
Enter Start Station?:
gombak
Your Destination:
tanjung malim
Path found: GOMBAK -> BANDAR UTAMA -> PASAR SENI -> KL SENTRAL -> TANJUNG MALIM
Ticket Cost: RM18
Would you like to find another route? (Y/N)
y
```

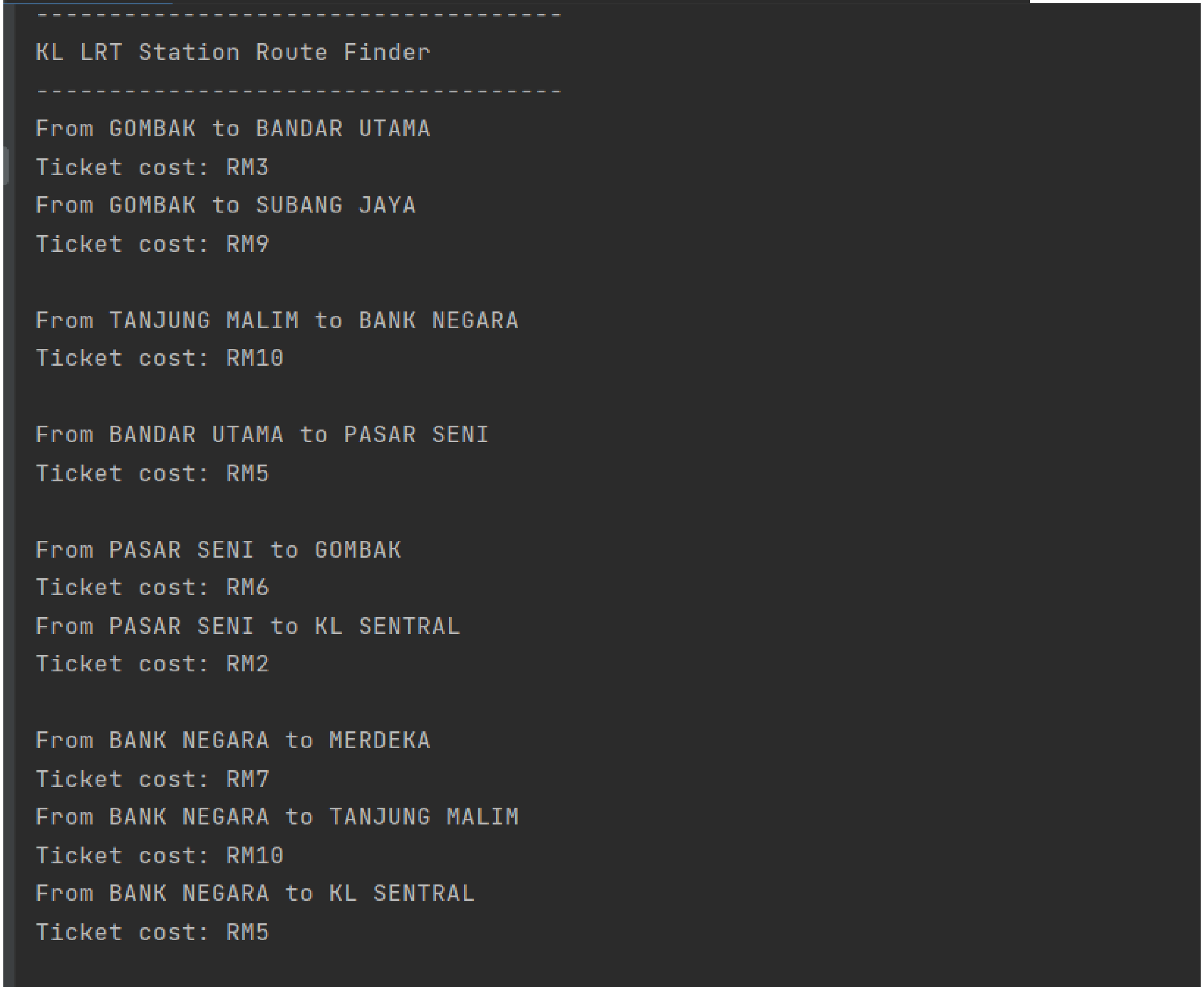


```
------------------------------------
KL LRT Station Route Finder
------------------------------------
Our Stations Coverage:
GOMBAK
MERDEKA
TANJUNG MALIM
BANDAR UTAMA
PASAR SENI
BANK NEGARA
SUBANG JAYA
KL SENTRAL
Enter Start Station?:
merdeka
Your Destination:
subang jaya
No path found
Would you like to find another route? (Y/N)
n
```

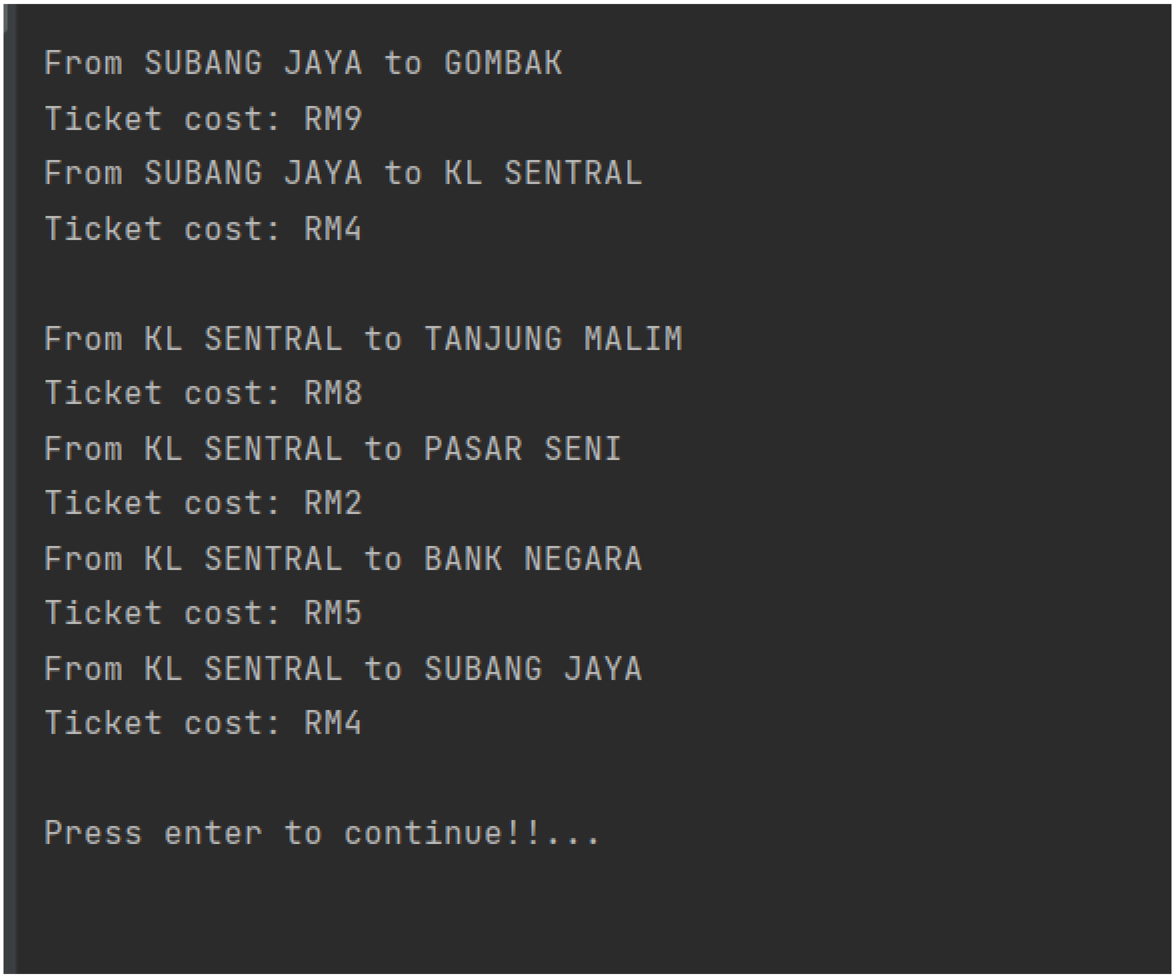
Second option



```
------------------------------------
KL LRT Station Route Finder
------------------------------------
Please select an option:
1. Find the route from one station to another
2. Check ticket price
3. Exit
Your option: 2
```



```
------------------------------------
KL LRT Station Route Finder
------------------------------------
From GOMBAK to BANDAR UTAMA
Ticket cost: RM3
From GOMBAK to SUBANG JAYA
Ticket cost: RM9

From TANJUNG MALIM to BANK NEGARA
Ticket cost: RM10

From BANDAR UTAMA to PASAR SENI
Ticket cost: RM5

From PASAR SENI to GOMBAK
Ticket cost: RM6
From PASAR SENI to KL SENTRAL
Ticket cost: RM2

From BANK NEGARA to MERDEKA
Ticket cost: RM7
From BANK NEGARA to TANJUNG MALIM
Ticket cost: RM10
From BANK NEGARA to KL SENTRAL
Ticket cost: RM5
```



```
From SUBANG JAYA to GOMBAK
Ticket cost: RM9
From SUBANG JAYA to KL SENTRAL
Ticket cost: RM4

From KL SENTRAL to TANJUNG MALIM
Ticket cost: RM8
From KL SENTRAL to PASAR SENI
Ticket cost: RM2
From KL SENTRAL to BANK NEGARA
Ticket cost: RM5
From KL SENTRAL to SUBANG JAYA
Ticket cost: RM4

Press enter to continue!!...
```

7.0 REFERENCES

Implementing Generic Graph in Java. (2019, August 7). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/implementing-generic-graph-in-java/

Graph and its representations. (2012, November 13). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/graph-and-its-representations/

Kumar Chandrakant. (2018, November 28). *Graphs in Java | Baeldung*. Baeldung. https://www.baeldung.com/java-graphs

DevGlan. (2020). *Graph Implementation in Java | DevGlan*. Devglan. https://www.devglan.com/datastructure/graph-implementation-java