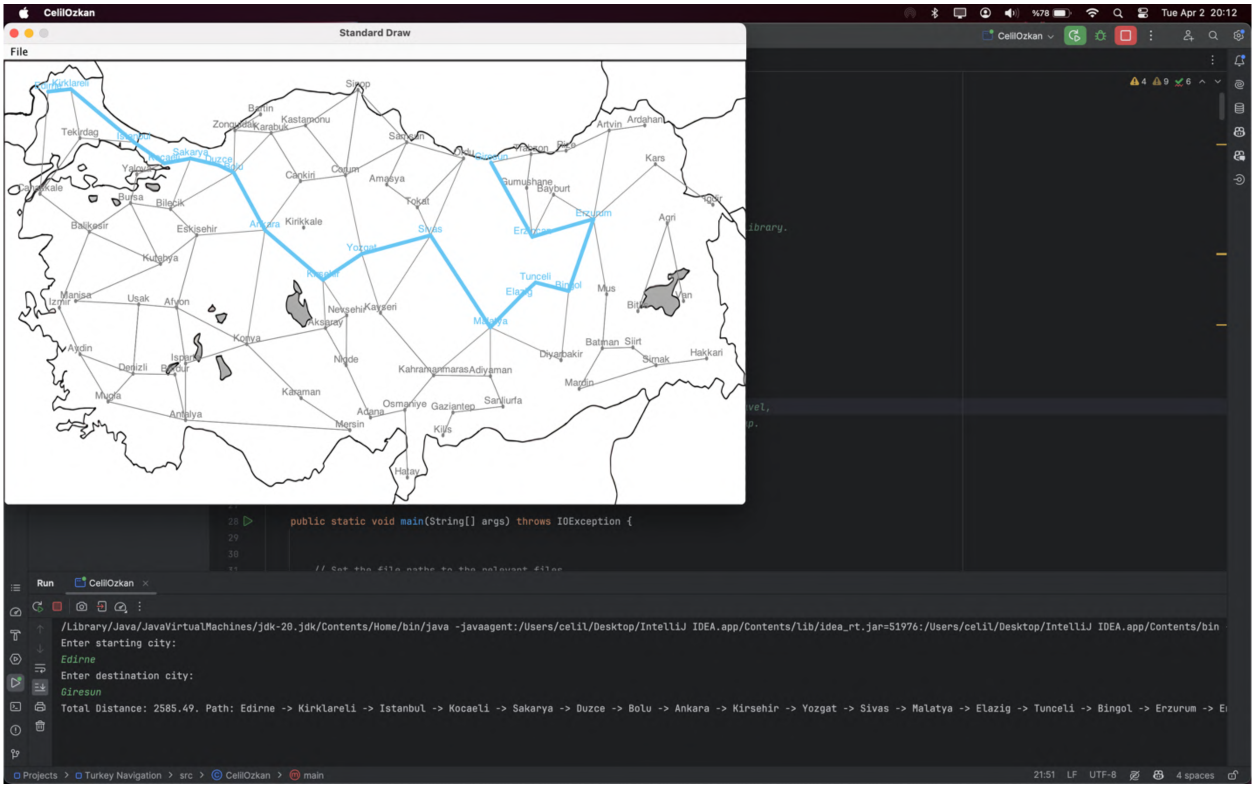
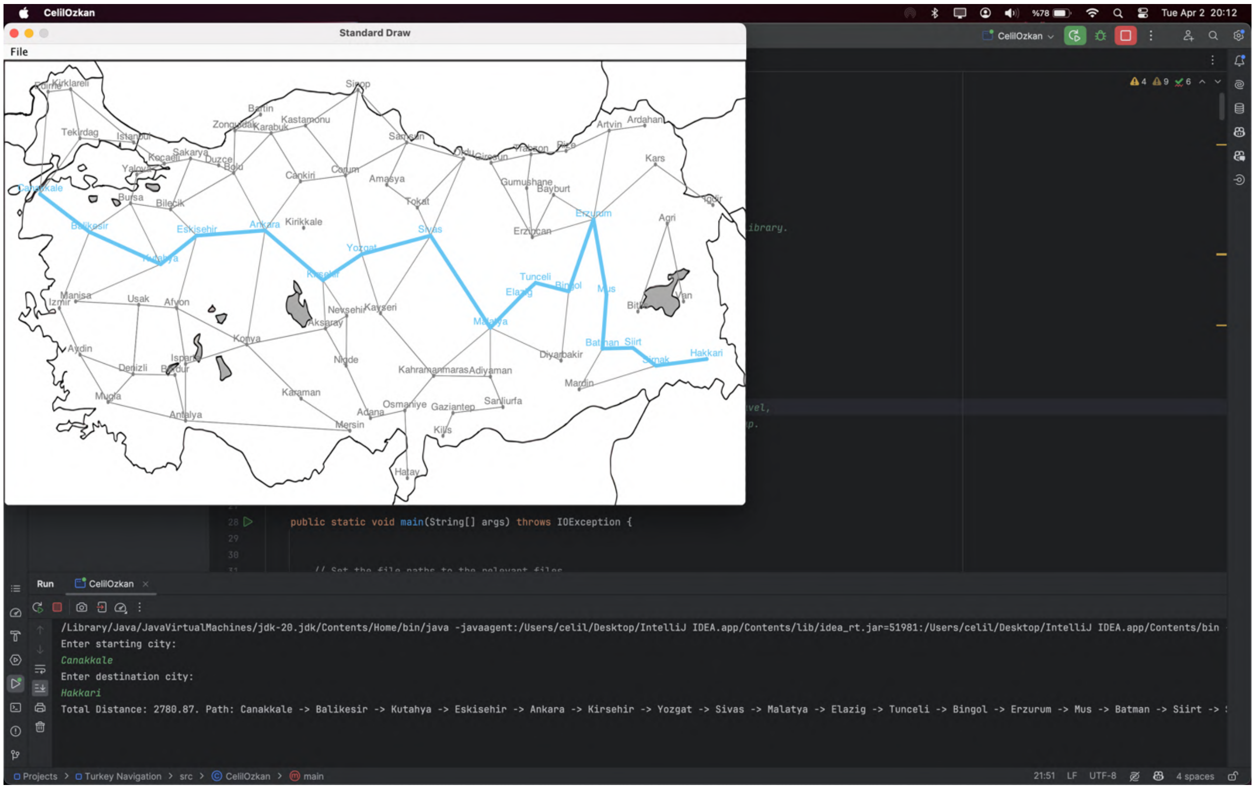


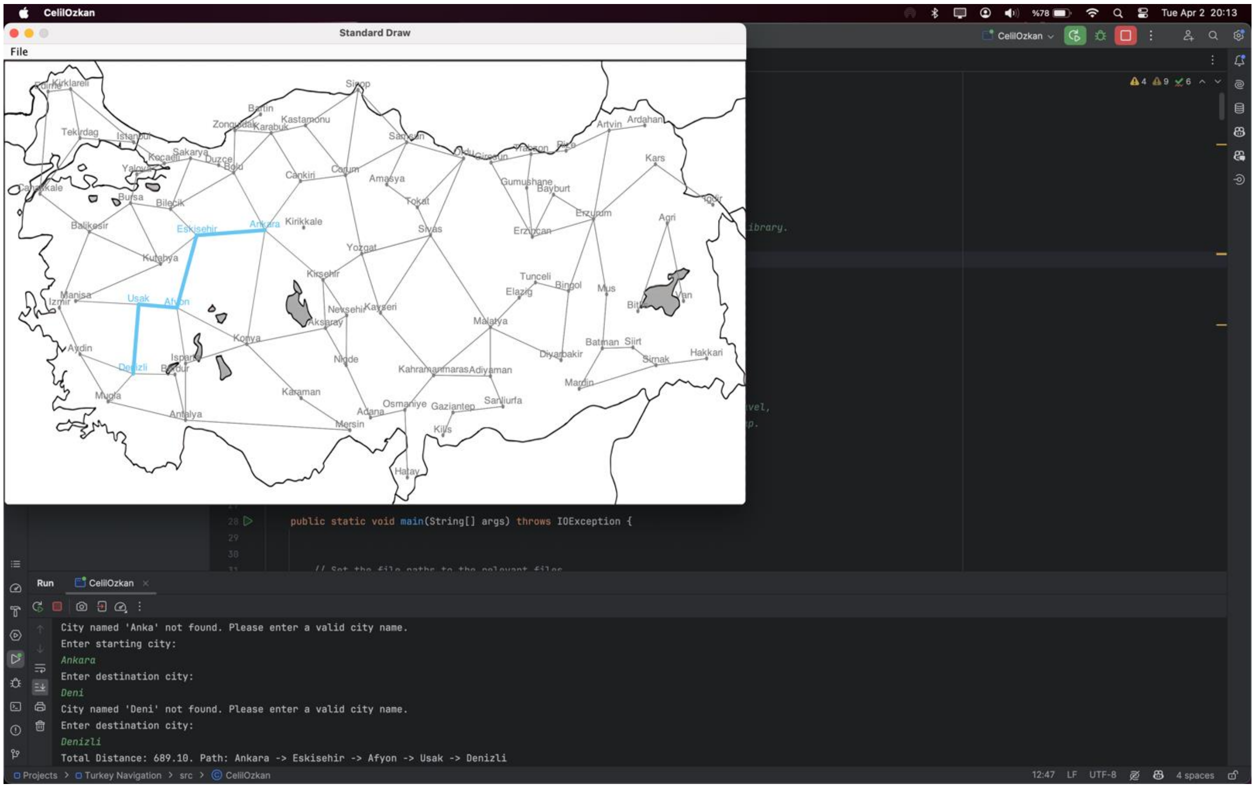
```
    public static void main(String[] args) throws IOException {
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=51976:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Edirne
Enter destination city:
Giresun
Total Distance: 2585.49. Path: Edirne -> Kirklareli -> Istanbul -> Kocaeli -> Sakarya -> Duzce -> Bolu -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya -> Elazig -> Tunceli -> Bingol -> Erzurum -> E
```



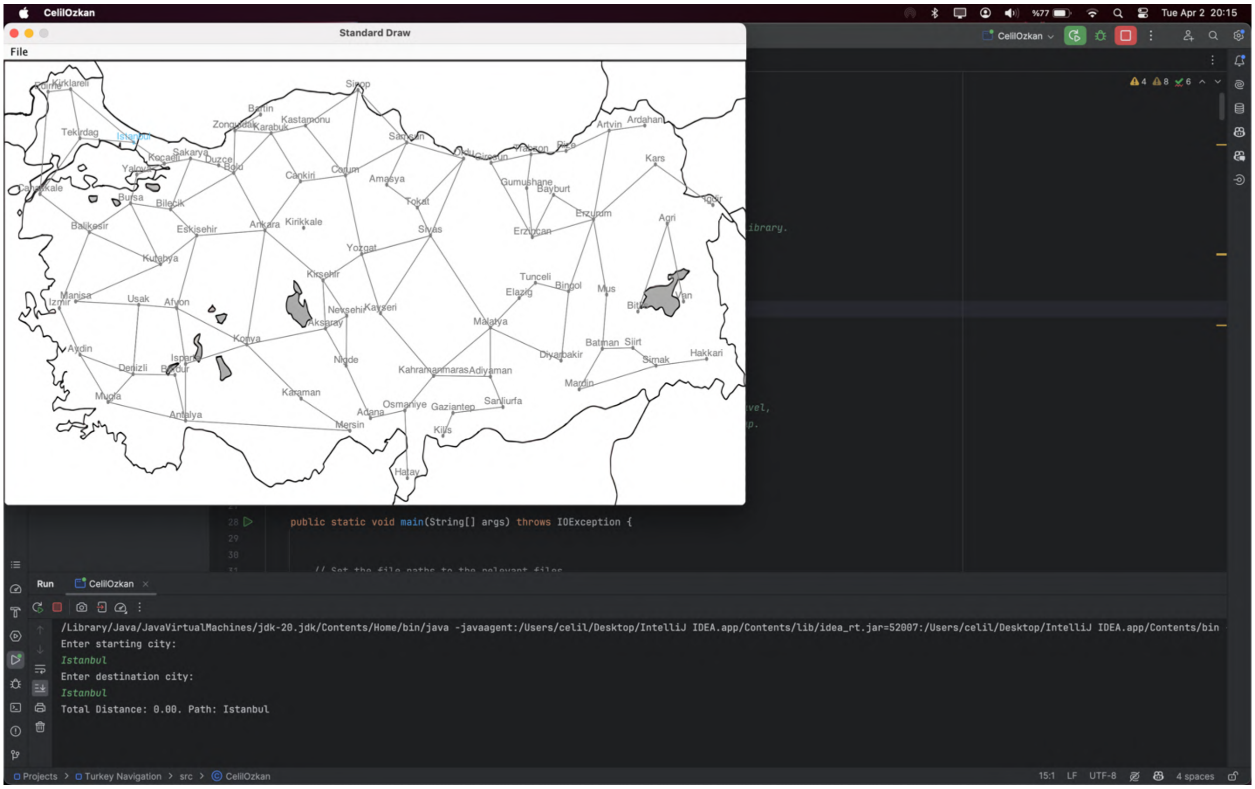
```
    public static void main(String[] args) throws IOException {
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=51981:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Canakkale
Enter destination city:
Hakkari
Total Distance: 2780.87. Path: Canakkale -> Balikesir -> Kutahya -> Eskisehir -> Ankara -> Kirsehir -> Yozgat -> Sivas -> Malatya -> Elazig -> Tunceli -> Bingol -> Erzurum -> Mus -> Batman -> Siirt -> 
```



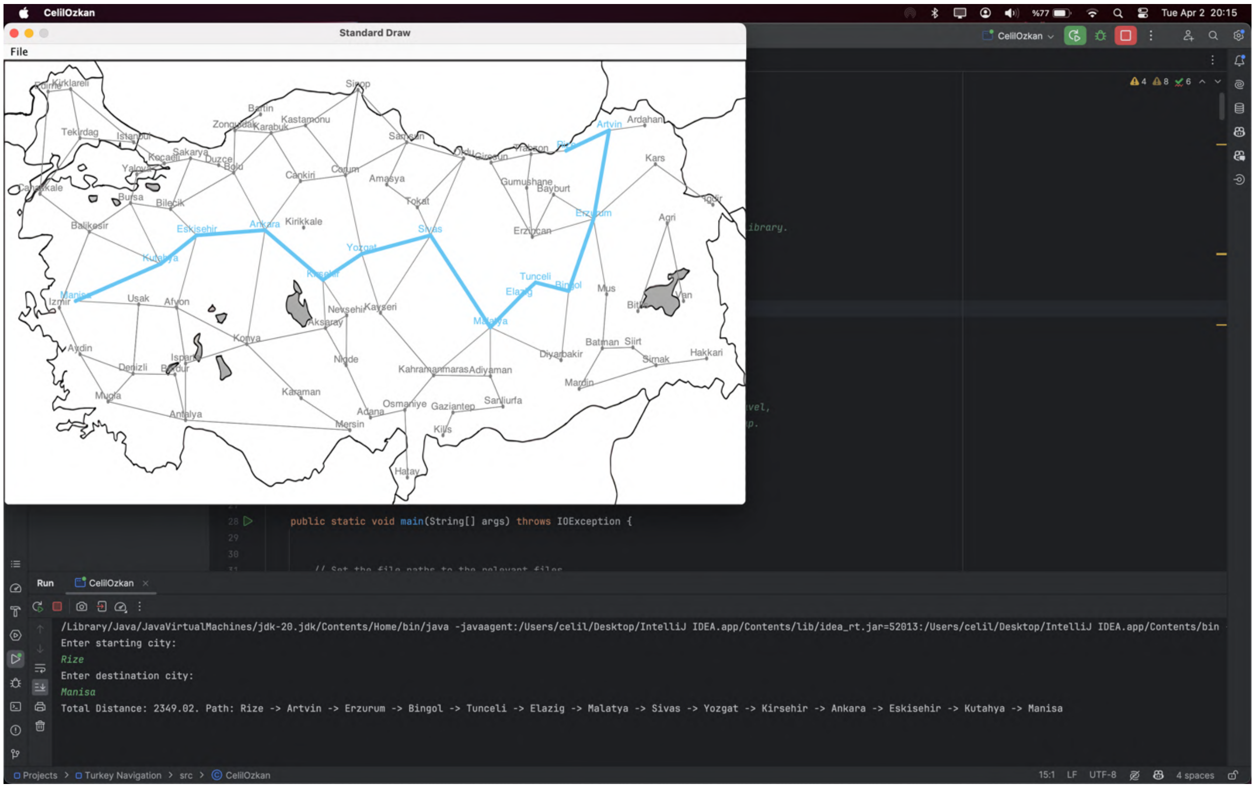
```
public static void main(String[] args) throws IOException {


    // Set the file paths to the relevant files
```

```
City named 'Anka' not found. Please enter a valid city name.
Enter starting city:
Ankara
Enter destination city:
Deni
City named 'Deni' not found. Please enter a valid city name.
Enter destination city:
Denizli
Total Distance: 689.10. Path: Ankara -> Eskisehir -> Afyon -> Usak -> Denizli
```



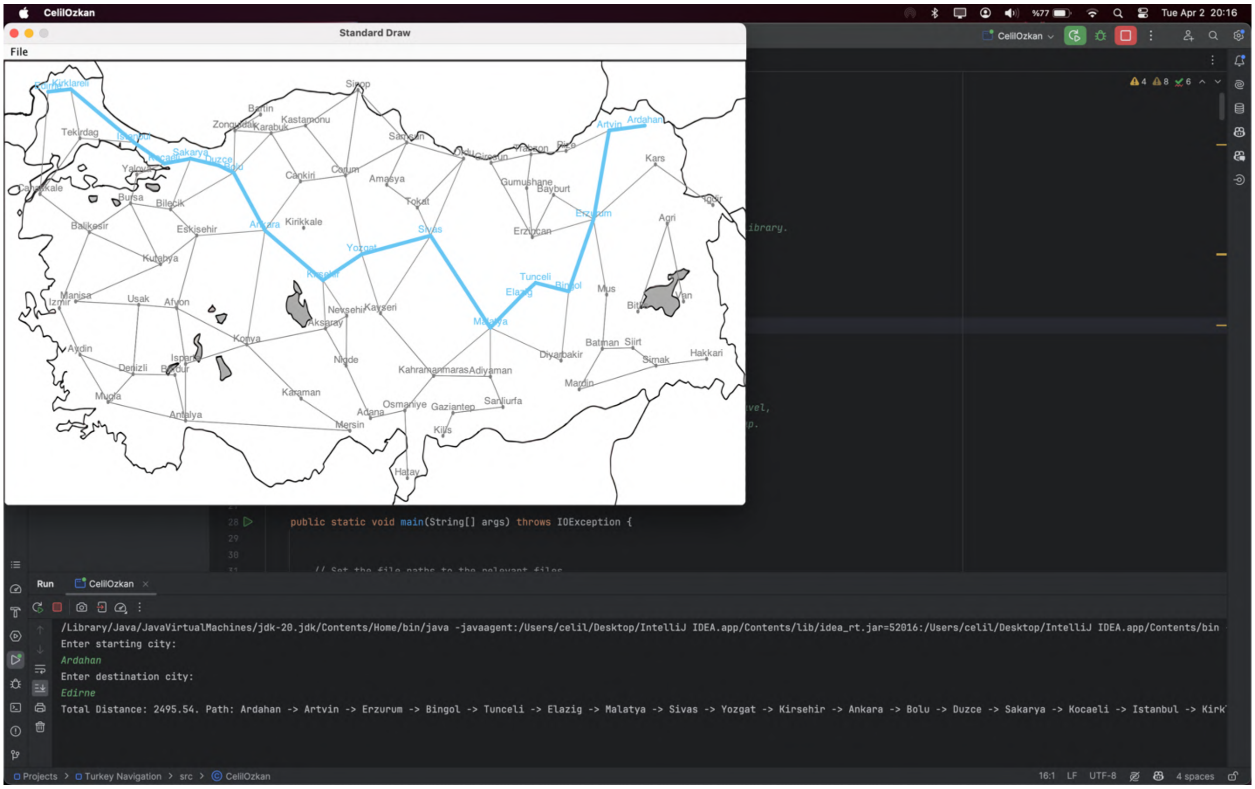
```
public static void main(String[] args) throws IOException {


    // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52007:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Istanbul
Enter destination city:
Istanbul
Total Distance: 0.00. Path: Istanbul
```



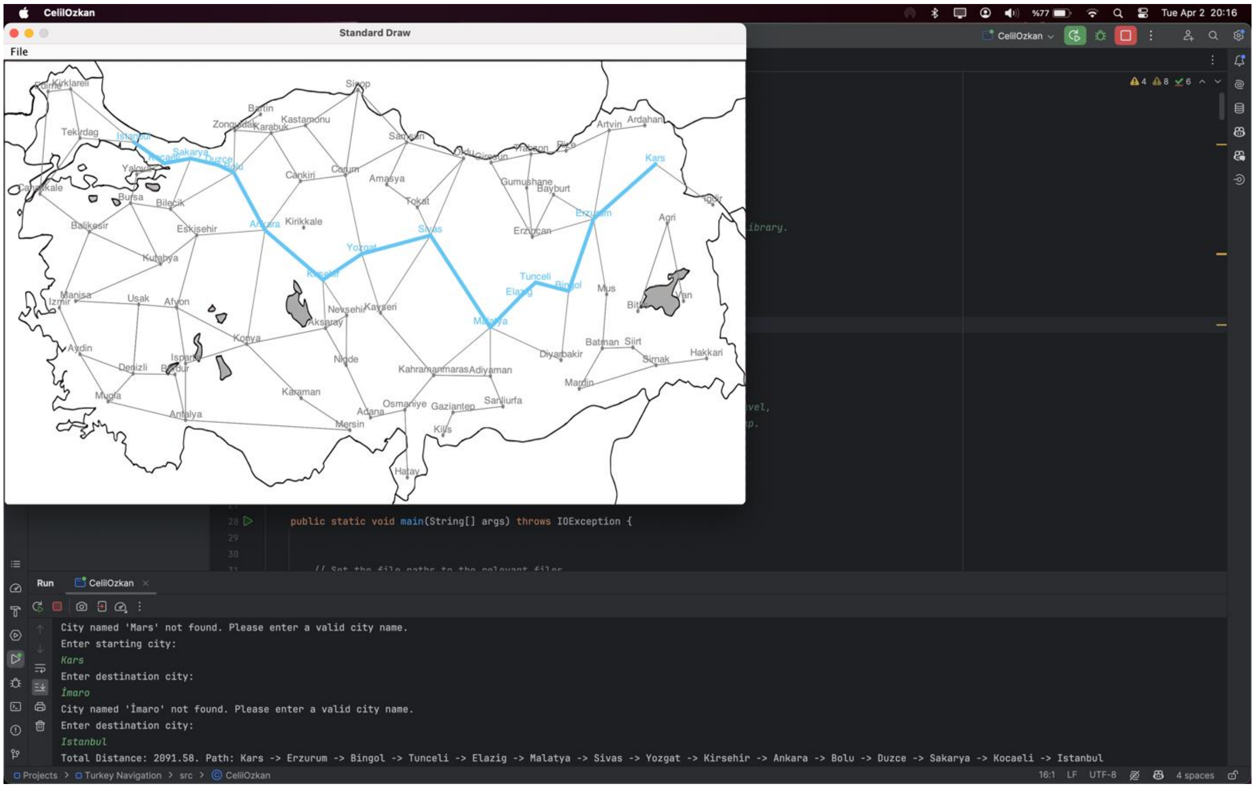
```
public static void main(String[] args) throws IOException {

    // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52013:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Rize
Enter destination city:
Manisa
Total Distance: 2349.02. Path: Rize -> Artvin -> Erzurum -> Bingol -> Tunceli -> Elazig -> Malatya -> Sivas -> Yozgat -> Kirsehir -> Ankara -> Eskisehir -> Kutahya -> Manisa
```



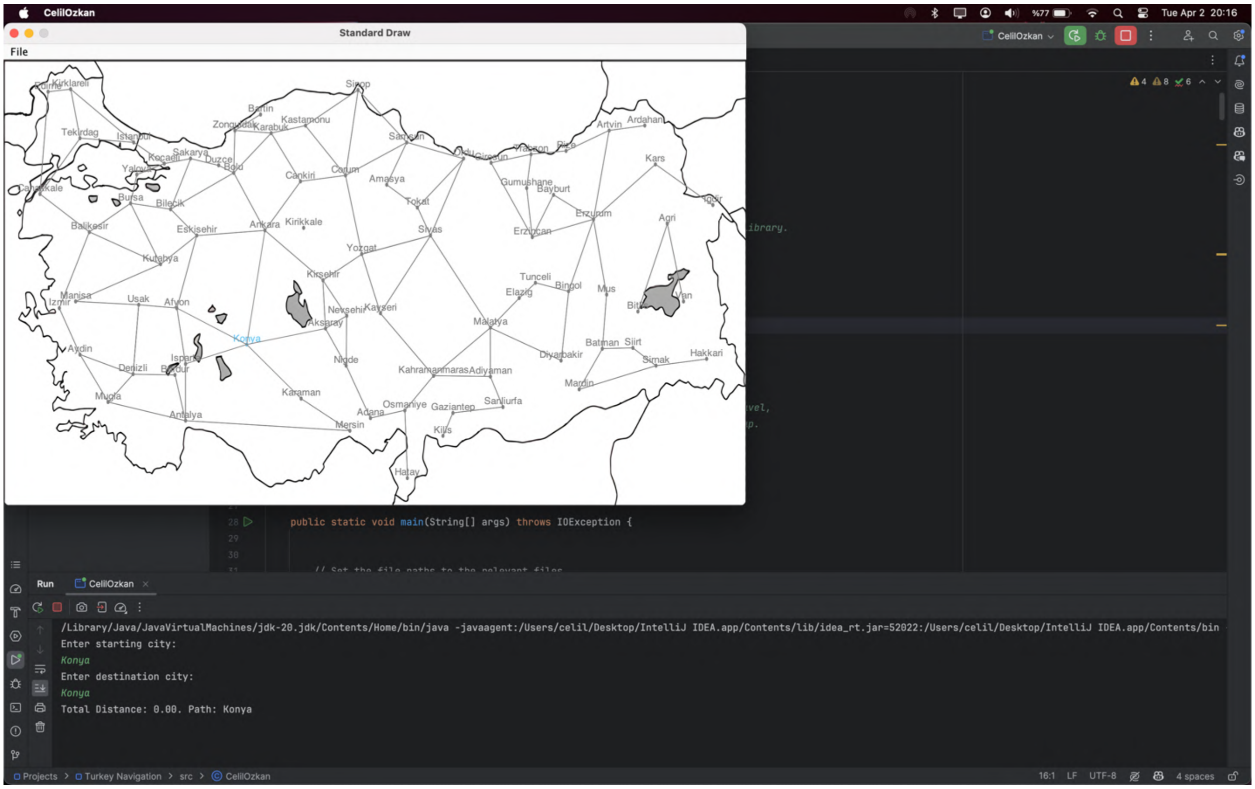
```
public static void main(String[] args) throws IOException {

    // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52016:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Ardahan
Enter destination city:
Edirne
Total Distance: 2495.54. Path: Ardahan -> Artvin -> Erzurum -> Bingol -> Tunceli -> Elazig -> Malatya -> Sivas -> Yozgat -> Kirsehir -> Ankara -> Bolu -> Duzce -> Sakarya -> Kocaeli -> Istanbul -> Kirk
```



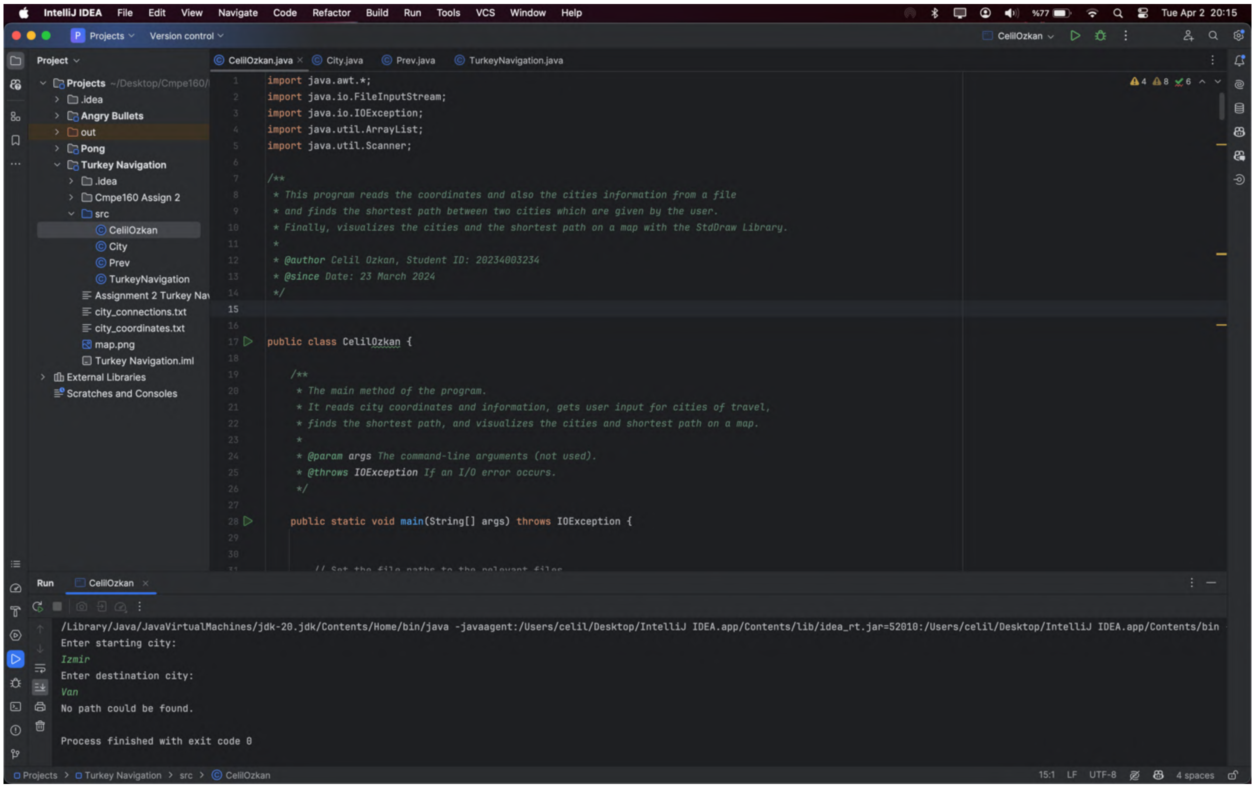
```
public static void main(String[] args) throws IOException {


    // Set the file paths to the relevant files
```

```
City named 'Mars' not found. Please enter a valid city name.
Enter starting city:
Kars
Enter destination city:
İmaro
City named 'İmaro' not found. Please enter a valid city name.
Enter destination city:
Istanbul
Total Distance: 2091.58. Path: Kars -> Erzurum -> Bingol -> Tunceli -> Elazig -> Malatya -> Sivas -> Yozgat -> Kirsehir -> Ankara -> Bolu -> Duzce -> Sakarya -> Kocaeli -> Istanbul
```



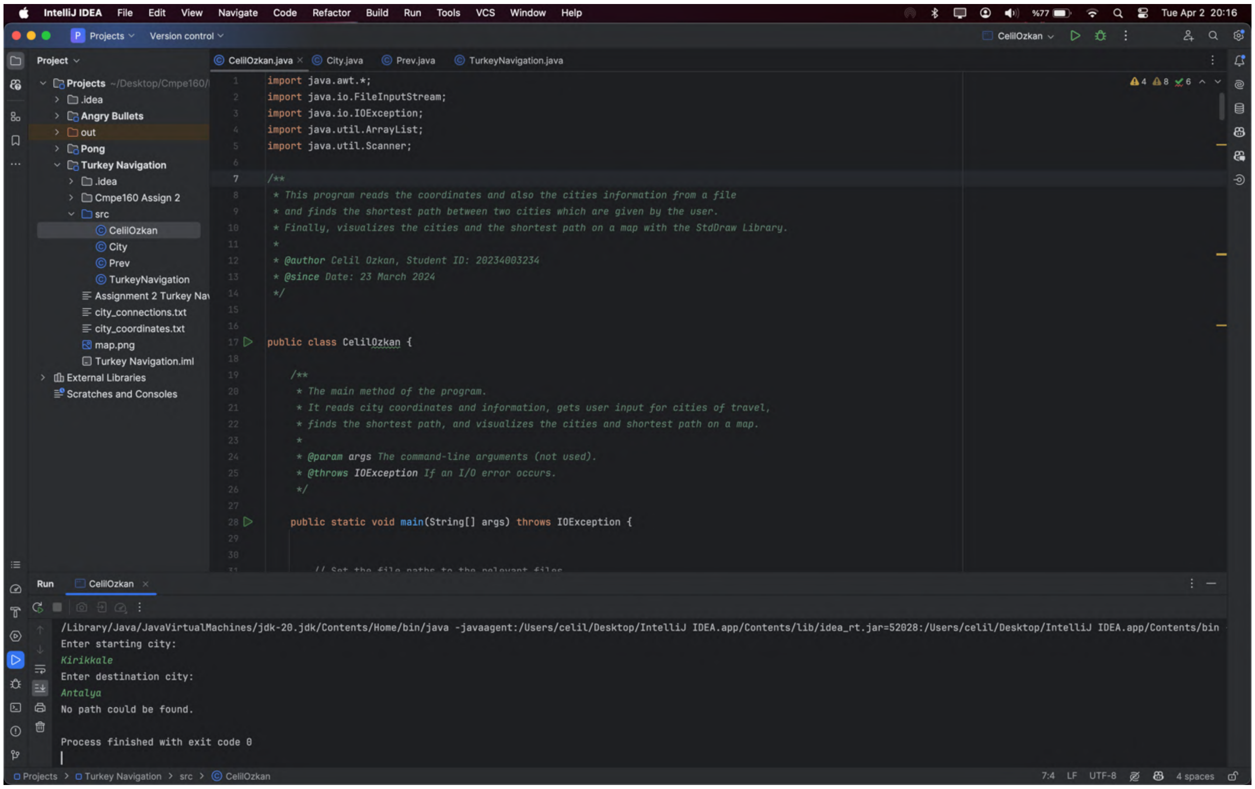
```
public static void main(String[] args) throws IOException {


    // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52022:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Konya
Enter destination city:
Konya
Total Distance: 0.00. Path: Konya
```



```java
import java.awt.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * This program reads the coordinates and also the cities information from a file
 * and finds the shortest path between two cities which are given by the user.
 * Finally, visualizes the cities and the shortest path on a map with the StdDraw Library.
 *
 * @author Celil Ozkan, Student ID: 20234003234
 * @since Date: 23 March 2024
 */


public class CelilOzkan {

    /**
     * The main method of the program.
     * It reads city coordinates and information, gets user input for cities of travel,
     * finds the shortest path, and visualizes the cities and shortest path on a map.
     *
     * @param args The command-line arguments (not used).
     * @throws IOException If an I/O error occurs.
     */

    public static void main(String[] args) throws IOException {


        // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52010:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Izmir
Enter destination city:
Van
No path could be found.

Process finished with exit code 0
```



```java
import java.awt.*;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * This program reads the coordinates and also the cities information from a file
 * and finds the shortest path between two cities which are given by the user.
 * Finally, visualizes the cities and the shortest path on a map with the StdDraw Library.
 *
 * @author Celil Ozkan, Student ID: 20234003234
 * @since Date: 23 March 2024
 */


public class CelilOzkan {

    /**
     * The main method of the program.
     * It reads city coordinates and information, gets user input for cities of travel,
     * finds the shortest path, and visualizes the cities and shortest path on a map.
     *
     * @param args The command-line arguments (not used).
     * @throws IOException If an I/O error occurs.
     */

    public static void main(String[] args) throws IOException {


        // Set the file paths to the relevant files
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52028:/Users/celil/Desktop/IntelliJ IDEA.app/Contents/bin
Enter starting city:
Kirikkale
Enter destination city:
Antalya
No path could be found.

Process finished with exit code 0
```

Path Finding Algorithm PseduoCode

```
// Algorithm to obtain the path with the minimum possible distance from startCity to endCity

algorithm findShortestPath is:

    input: City instance startCity indicating the city that the travel will start

           City instance endCity indicating the city that the travel will end

           Arraylist of City instances cities, which stores all given city objects

    output: Arraylist of cities named shorestPath, storing the city objects from the start city to the endcity with the miniumum possible distance

    // Initialize some neccesary arrays to store various types of data

    init shortestPath, Arraylist of cities sized cities, in order to store cities for the minimul possible distance travel from start city to end city

    init visited, Boolean Arraylist of size cities, in order to store whether a city visited or not

    init distance, Double Arraylist of size cities, to store the minimum distance to the start city for each city

    for each distance in distances do

                distance := Integer.MAX_VALUE // Set all distances to infinity

    set distances[startCity] := 0 // Set the distance of the starting city to itself to 0

    for each city in cities do

        // Pick the closest city with the help of findClosestCity Method (It's explained at the bottom of the pseudocode)

        init city, currentCity := findClosestCity(cities, distances, visited)

        set visited[currentCity.index] := true // Meaning this city is being checked now

        // For each neigbor of a given city, here neighbors are in the datafield of any city objects

        for each neighbor in city.adjacentCities do

            if visited is equal to false // Neigbor isn't visited

            // DistanceBetweenCities is a method of city class returning the distance between the object and it's given neigh dbor (It's explain at the bottom of the code)
```

```
                init double, distance := city.distanceBetweenCities(neighbor)
                if distances[currentCity.index] + distance is smaller than or equal to distances[neighbor.index]
                // If the distance through current city to the neigbor city from the start city is found to be smaller than any other previously found distance update it’s value
                    set distances[neighbor.index] := distances[currentCity.index] + distance
                // Also update what was the last city to reach neighbor
                set neigbor.previousCity := currentCity
    init city, iterated city := endCity // In order to create shortestPath start from the end city
    while iterated city is not equal to null do
        shortestPath.add(0, iteratedCity) // Add iterated city to the first position of the path
        set iteratedCity := iteratedCity.previousCity // Move back to previous city and iterate again
    return shortestPath // Arraylist of cities that is sorted from startCity to endCity in order to achieve minimum distance travel.
```

```
// Algorithm to select next city. In order to find it checks all the distances to startCity and if a city’s distance is minimum and visited attribute is false returns this city
algorithm findClosestCity is:
    input: Arraylist of cities named cities, to hold city objects
            Arraylist of doubles distances, to store distances to the startCity
            Arraylist of booleans visited, to store whether a city visited or not
    output: city closestCity, city with the minimum distance to the startCity also that is not visited before
    init double, minDistance := Double.MAX_VALUE // Set to infinity
    init city, closestCity := null // It will be find in the loop
    for each distance in distances do
        // Here distance and visited are attributes of the city object that is taken from the distances and visited arrays respectively
        if distance is smaller than minDistance and visited is false do
            set minDistance := distance
            set closestCity := city
    return closestCity // The city with the minimum distance to the startCity that is not visited before


// Algorithm to find distance between a city object and it’s neigbor
// Observe that it’s a method of city class. So you need a city to call it and give another city that you want
algorithm distanceBetweenCities is
    input: City otherCity, target city object
    output: Double distance, distance btw the target city and the city object that called the method
   return squareroot of the sum of the x differences square and y differences square of the target city and the city that is called the method
```

References:

- JavaTPoint: Pseudocode in Java (https://www.javatpoint.com/pseudocode-java)

- GeeksforGeeks: What is Pseudocode? A Complete Tutorial (https://www.geeksforgeeks.org/what-is-pseudocode-a-complete-tutorial/)

- OpenAI Chat: Conversational AI Platform (https://chat.openai.com)

- YouTube: Video Tutorial on Path Finding Algorithm (https://www.youtube.com/watch?v=GazC3A4OQTE)

- YouTube: Another Video Tutorial on Path Finding Algorithm (https://www.youtube.com/watch?v=EFg3u_E6eHU&t=439s)

- YouTube: Yet Another Video Tutorial on Path Finding Algorithm (https://www.youtube.com/watch?v=XB4MIexjvY0&t=651s)

Path Finding Algorithm Further Comments:

The document below is written to clarify how my path-finding algorithm works.

1 - Initialize some necessary arrays namely:

Shortest Path: An array of cities that will be filled with the cities that are in the shortest path from the start city to the end city.

Visited: An array of booleans storing whether a city is visited or not.

Distances: An array of doubles that stores the distance from the start city to the city at any index corresponding to the index at the distance array.

2 – Set all the values in the Distances array to a very large number by Integer.MAX_VALUE
(At first, I used Double.MAX_VALUE but it caused some problems I don’t know the exact reason.)

3 – Set the distance of the starting city to 0 in the Distance array, since its distance to itself is 0.

4 – Start an iteration of all the cities, pick the closest city with the help of the findClosestCity method. At the first iteration, it will give the starting city, and for the others the city with the smallest distance to the starting city. Also, observe that for each iteration Distances array will be updated.

5 – Set the picked closest city’s boolean value to true in the Visited array since we are checking it now.

6 – Start another array to iterate the closest city’s neighbors. For each neighbor, if it’s not visited before, check whether if it’s more logical to come to this neighbor city after the closest city or not. In order to do that take the sum of the distance that’s needed to reach the closest city and also the distance between the closest city and its neighbor. If this sum value is smaller than the previous value of the neighbor city in the distance array it means that we’ve found a better way to reach the neighbor city. If it’s the case update the neighbor city’s distance value to the starting city, since we’ve found a better way to reach this city. Also update the neighbor city’s previous city data field with the closest city, since the most logical way to reach the neighbor city is now after reaching the closest city.

7 – After all the iterations ended we will have an array that indicates the minimum possible distances to the starting city which we call the Distances array and all the cities will have the updated previous city data field which indicates the previous city in the most logical way from starting city to a specific city (except the starting city and unreachable cities they will have the previous city data field equals to null).

8 – Start from the end city and define a recursion that iterates over until its value is null. In the first iteration insert the end city to the shortestPath array’s 0’th index. And then insert the end city’s previous city to the shortestPath array’s first position. Iterating in this manner over and over again we will get to the point where the starting city becomes the previous city. After inserting the starting city into the first position, the new previous city will become null, thereby breaking the loop condition.

findClosestCity Method Clarification:

If I need to clarify how the findClosestCity works, at first it sets a variable named closest city to null. Then sets the minDistance to Double.MAX_VALUE. And starts to check all the cities and their values to the starting city using the Distances array. If it’s not visited, the method sets the city with the min value in the distances array to the closest city variable and returns it.