

# Assignment 2: Coding

## Traveling Salesman Problem

Submitted By:

- Sannidhya Pathania (spathan3)
- Karan Malik (kmalik8)
- Thy Do (thydo)
- Aashka Dave (adave8)

**Date: Oct 11, 2022**

### *Table of Contents*

<b>Java Files:</b>	<b>2</b>
<b>Class diagram:</b>	<b>3</b>
<b>Output:</b>	<b>4</b>
Symmetric Data:	4
Asymmetric Data:	7
<b>Conclusion:</b>	<b>9</b>

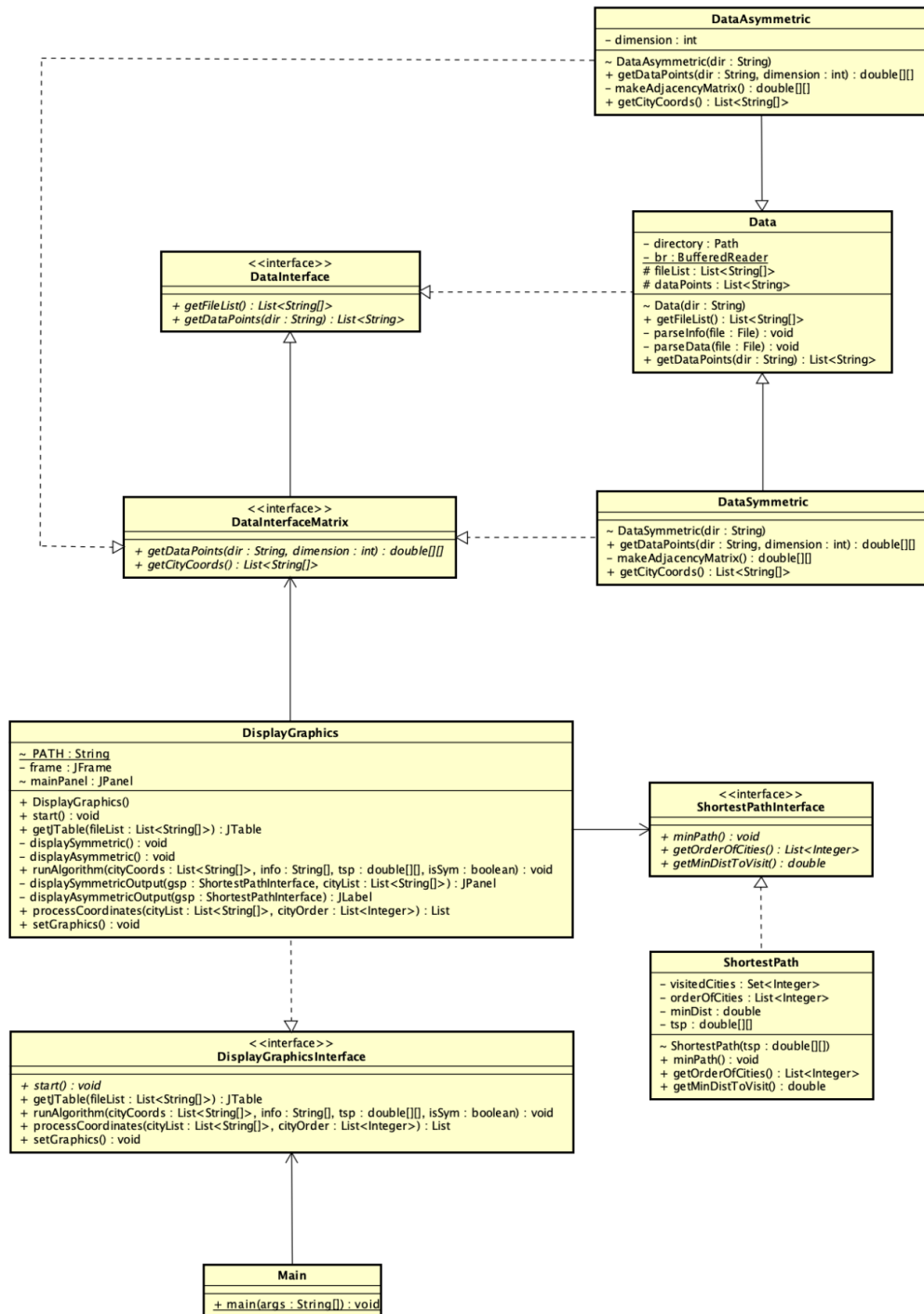
# Java Files:

**GitHub Link:** [https://github.com/thydo/CSE564\\_TSP\\_Project](https://github.com/thydo/CSE564_TSP_Project)

## **Java Files: (Uploaded on Canvas alongside Pdf)**

- Data.java
- DataAsymmetric.java
- DataInterface.java
- DataInterfaceMatrix.java
- DataSymmetric.java
- DisplayGraphics.java
- DisplayGraphicsInterface.java
- Main.java
- ShortestPath.java
- ShortestPathInterface.java

# Class diagram:

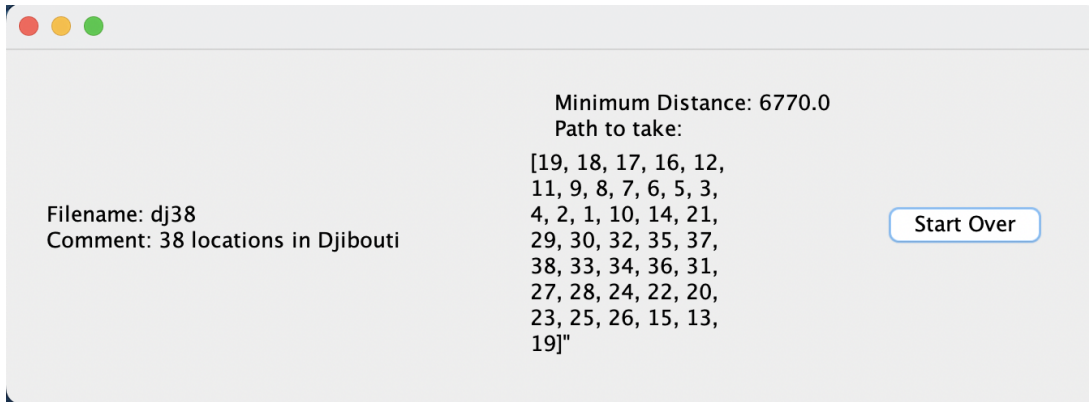


# Output:

## Symmetric Data:

- *Small dataset - Djibouti (38 cities)*

Minimum distance and order of cities

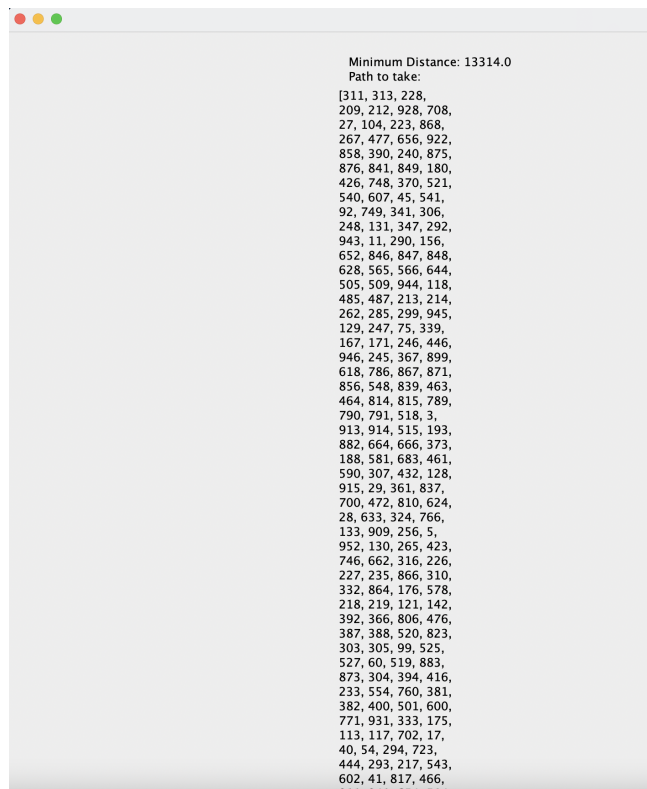


Plotting the cities



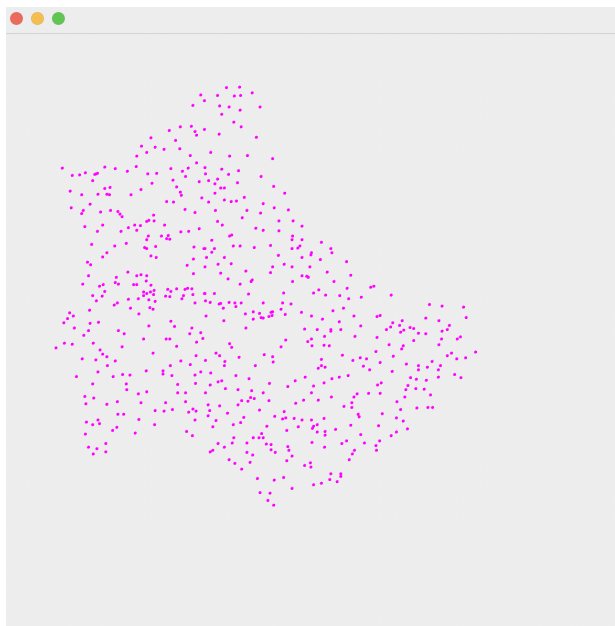
- **Medium dataset - Luxembourg (980 cities)**

Minimum distance and order of cities



```
Minimum Distance: 13314.0
Path to take:
[311, 313, 228,
209, 212, 928, 708,
27, 104, 223, 868,
267, 477, 656, 922,
858, 390, 240, 875,
876, 841, 849, 180,
426, 748, 370, 521,
540, 607, 45, 541,
92, 749, 341, 306,
248, 131, 347, 292,
943, 11, 290, 156,
652, 846, 847, 848,
628, 565, 566, 644,
505, 509, 944, 118,
485, 487, 213, 214,
262, 285, 299, 945,
129, 247, 75, 339,
167, 171, 246, 446,
946, 245, 367, 899,
618, 786, 867, 871,
856, 548, 839, 463,
464, 814, 815, 789,
790, 791, 518, 3,
913, 914, 515, 193,
882, 664, 666, 373,
188, 581, 683, 461,
590, 307, 432, 128,
915, 29, 361, 837,
700, 472, 810, 624,
28, 633, 324, 766,
133, 909, 256, 5,
952, 130, 265, 423,
746, 662, 316, 226,
227, 235, 866, 310,
332, 864, 176, 578,
218, 219, 121, 142,
392, 366, 806, 476,
387, 388, 520, 823,
303, 305, 99, 525,
527, 60, 519, 883,
873, 304, 394, 416,
233, 554, 760, 381,
382, 400, 501, 600,
771, 931, 333, 175,
113, 117, 702, 17,
40, 54, 294, 723,
444, 293, 217, 543,
602, 41, 817, 466,
911, 941, 654, 614]
```

Plotting the cities



- **Large dataset - Ireland (8246 cities)**

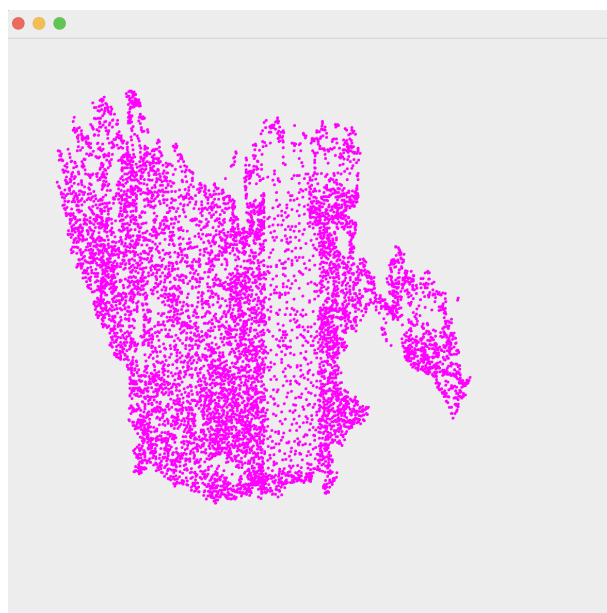
Minimum distance and order of cities

```

Minimum Distance: 252169.0
Path to take:
[3598, 3416, 3414,
3326, 3331, 3288,
3292, 3359, 3435,
3532, 3458, 3567,
3516, 3484, 3393,
3344, 3287, 3257,
3213, 3262, 3294,
3338, 3383, 3423,
3558, 3609, 3620,
3647, 3693, 3723,
3772, 3812, 3946,
3939, 3866, 3885,
3996, 4033, 4057,
4071, 3967, 4058,
4053, 4142, 4151,
4144, 4182, 4200,
4257, 4316, 4365,
4450, 4495, 4602,
4693, 4710, 4794,
4919, 4979, 5124,
5120, 5039, 4879,
4846, 4737, 4588,
4520, 4452, 4400,
4328, 4264, 4248,
4186, 4101, 4121,
4133, 4108, 4031,
4000, 3950, 3886,
3867, 3872, 3888,
3806, 3743, 3686,
3611, 3589, 3562,
3544, 3420, 3350,
3335, 3526, 3590,
3632, 3699, 3754,
3762, 3844, 3914,
4005, 4028, 4088,
4098, 4158, 4208,
4249, 4299, 4374,
4372, 4397, 4456,
4460, 4429, 4378,
4325, 4319, 4252,
4315, 4446, 4567,
4607, 4642, 4681,
4712, 4795, 4859,
4921, 4873, 4892,
4934, 4978, 5066,
5123, 5200, 5269,
5196, 5129, 5156,
5221, 5373, 5431,
-----

```

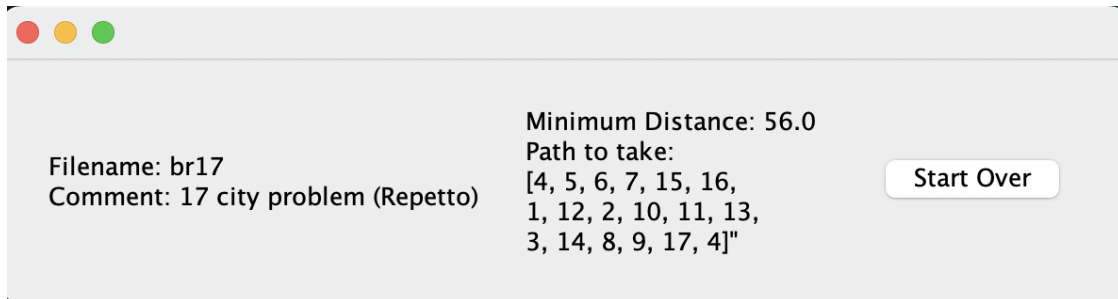
Plotting the cities



## Asymmetric Data:

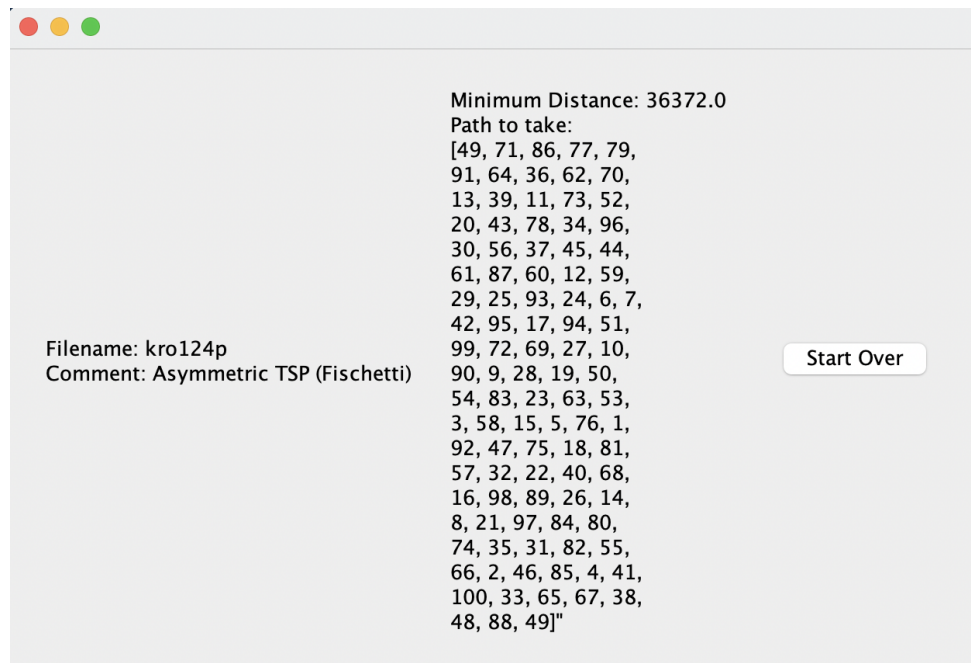
- *Small dataset - Repetto (17 cities)*

Minimum distance and order of cities



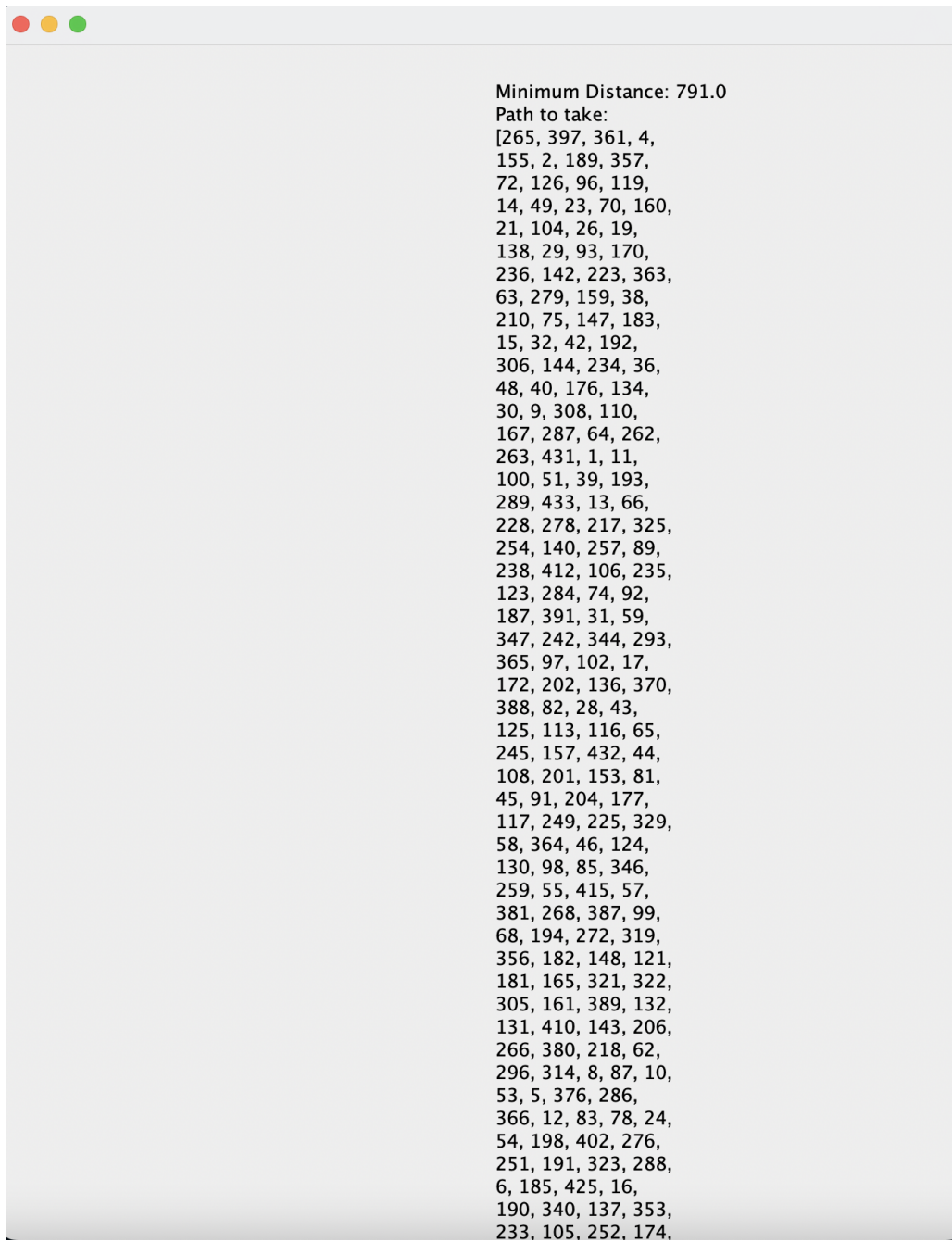
- *Medium dataset - Fischetti (100 cities)*

Minimum distance and order of cities



- **Large dataset - rbg443 (443 cities)**

Minimum distance and order of cities



```
Minimum Distance: 791.0
Path to take:
[265, 397, 361, 4,
155, 2, 189, 357,
72, 126, 96, 119,
14, 49, 23, 70, 160,
21, 104, 26, 19,
138, 29, 93, 170,
236, 142, 223, 363,
63, 279, 159, 38,
210, 75, 147, 183,
15, 32, 42, 192,
306, 144, 234, 36,
48, 40, 176, 134,
30, 9, 308, 110,
167, 287, 64, 262,
263, 431, 1, 11,
100, 51, 39, 193,
289, 433, 13, 66,
228, 278, 217, 325,
254, 140, 257, 89,
238, 412, 106, 235,
123, 284, 74, 92,
187, 391, 31, 59,
347, 242, 344, 293,
365, 97, 102, 17,
172, 202, 136, 370,
388, 82, 28, 43,
125, 113, 116, 65,
245, 157, 432, 44,
108, 201, 153, 81,
45, 91, 204, 177,
117, 249, 225, 329,
58, 364, 46, 124,
130, 98, 85, 346,
259, 55, 415, 57,
381, 268, 387, 99,
68, 194, 272, 319,
356, 182, 148, 121,
181, 165, 321, 322,
305, 161, 389, 132,
131, 410, 143, 206,
266, 380, 218, 62,
296, 314, 8, 87, 10,
53, 5, 376, 286,
366, 12, 83, 78, 24,
54, 198, 402, 276,
251, 191, 323, 288,
6, 185, 425, 16,
190, 340, 137, 353,
233, 105, 252, 174,
```



# Conclusion:

## **SOLID principles and Wirth and Parnas:**

We have carefully incorporated the SOLID principles to ensure clean coding. While implementing the concepts of Object-oriented programming, we have made sure to keep our code readable and well-documented. We have specified the access as private for the information that is not required for the user to know.

### **1. *Single Responsibility Principle:***

We have divided the responsibility among all the classes. For instance, the Graphics class will manage the GUI for our assignment and doesn't have any other functionality like computing the result. Furthermore, we have designed different classes to handle *Asymmetric* and *Symmetric* data. Also, Our algorithm to figure out the minimum path lies in a separate Class.

### **2. *Open-Closed Principle:***

The Data class earlier was supposed to handle the files and parse the data. The goal was to return the ArrayList of coordinates. The limitation was, our algorithm required the 2-D matrix to work upon. Later, we implemented *AsymmetricData* and *SymmetricData* classes instead of modifying the code in the *Data* class. So we made sure that our classes are open for extension but not for modification.

### **3. *Liskov Substitution Principle:***

The Data class returned the ArrayList of coordinates. The limitation was, our algorithm required the 2-D matrix. *AsymmetricData* and *SymmetricData* classes inherit the *Data* class. Both these classes are better than their parent in terms of returning the more refined data parsing for the algorithm to compute the result.

### **4. *Interface Segregation Principle:***

We have programmed different interfaces depending on the requirement, instead of a general-purpose Interface. Every class is clear about which interface to implement and what functions to include as a part of the requirement.

### **5. *Dependency Inversion Principle:***

We have implemented interfaces for all the classes. We are using the references of interfaces to call the object of classes that implements the given interface. For instance,

the reference of ShortestPathInterface is calling all the methods of the class GetShortestPath. Similarly, for all the classes, we have inverted the dependency on interfaces.

### **Limitations:**

We have identified the following limitations:

1. For 71,000 cities, our Code is giving “***java.lang.OutOfMemoryError***: Java heap space”.
2. The *displaySymmetric* and *displayAsymmetric* methods were supposed to be included in the *DataSymmetric* class and *DataAsymmetric* class respectively. But Since, the working is intertwined with the objects of the Graphics class, we weren't able to define these in their respective classes.
3. Our solution used a greedy approach to find a sub-optimal solution.
4. The GUI isn't able to scale very large inputs to the frame properly.