

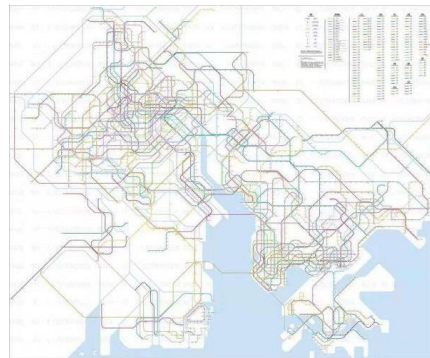
City University of Hong Kong
Department of Computer Science

CS3343 Software Engineering Practice

2021/22 Semester A

**Hong Kong-Shenzhen Metro Route Planning System
in Greater Bay Area**

Test Report



Project Group 6

HUANG Yuqin (56195499)

LIU Wei (56198489)

WANG Zhixuan (56198557, APM)

XU Jiakai (56197616)

XU Rui (56200188, PM)

ZHANG Xun (56198705)

TABLE OF CONTENTS

1 HIERARCHY DIAGRAM	2
2 TESTING STRATEGY	3
3 TEST CASES	3
3.1 Unit Testing	3
3.1.1 1st Layer	3
3.2 Integration Testing	4
3.2.1 2nd Layer	4
3.2.2 3rd Layer	4
3.2.3 4th Layer	5
3.2.4 5th Layer	6
3.2.5 6th Layer	7
3.2.6 7th Layer	9
3.2.7 8th Layer	9
3.3 System Testing	10
3.3.1 9th Layer	10
4 COVERAGE ANALYSIS	11
5 CODE REFACTORING	12
5.1 MetroSystem Class	12
5.2 Language Class	13

1 HIERARCHY DIAGRAM

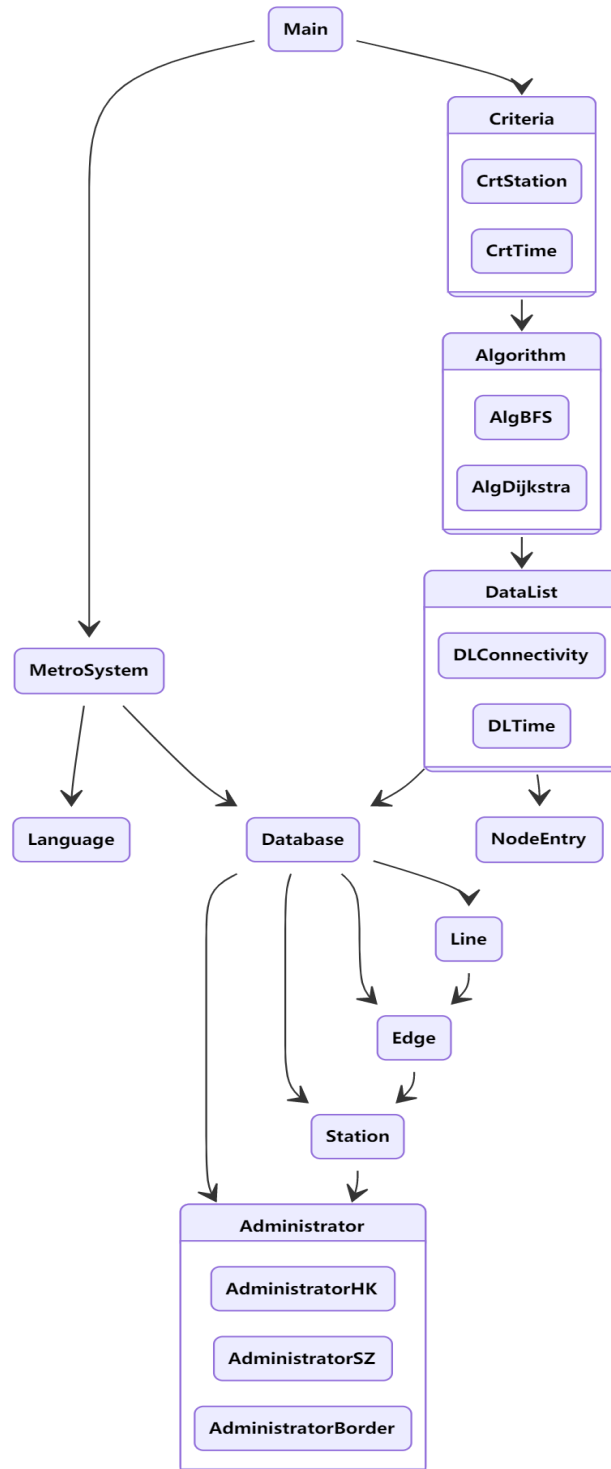


Figure 1.1 Hierarchy Diagram of Program Classes/Modules

2 TESTING STRATEGY

Bottom-up test approach is mainly adopted in our system. The advantages are obvious as our class structure is quite complex overall, it would be a tough task to stub loads of dependent classes or methods. Therefore breaking it down and starting from the bottom can make our test procedure as simple and systematic as possible. Moreover, it is easy to find out problems if any since bottom-up is a gradual approach and test cases are organized in adjacent layers and are closely related.

However, disadvantages also exist. With the Database module and the Station-Edge-Line module being mutually related, it is hard to tell which one should be at the bottom layer. After deliberation, we put Database the top layer as the concept of having a Database class is to gain a high-level control of all the other data.

So, for the modules mentioned above, we used the sandwich approach. Therefore, we create a unique testing method for our system with a local sandwich approach in a global bottom-up strategy.

3 TEST CASES

3.1 Unit Testing

3.1.1 1st Layer

1) testAdministrator.java

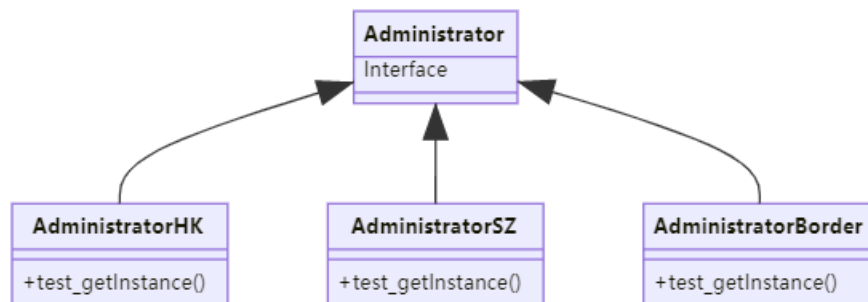


Figure 3.1.1.1 Method Tests of Administrator Class

- Test case analysis

Test Name	Purpose
test_getInstanceHK1	Test <i>getInstance()</i> for its returned object, and check whether this function can successfully return something and it is not a null object.

test_getInstanceSZ1	Test <i>getInstance()</i> for its returned object, and check whether this function can successfully return something and it is not a null object.
test_getInstanceBorder1	Test <i>getInstance()</i> for its returned object, and check whether this function can successfully return something and it is not a null object.
test_getInstanceHK2	Test <i>getInstance()</i> for its returned object, and check whether it is an object of its Administrator super-class, in order to check whether the singleton pattern of AdministratorHK can work normally and prepare for future polymorphism use.
test_getInstanceSZ2	Test <i>getInstance()</i> for its returned object, and check whether it is an object of its Administrator super-class, in order to check whether the singleton pattern of AdministratorSZ can work normally and prepare for future polymorphism use.
test_getInstanceBorder2	Test <i>getInstance()</i> for its returned object, and check whether it is an object of its Administrator super-class, in order to check whether the singleton pattern of AdministratorBorder can work normally and prepare for future polymorphism use.
test_getInstanceHK3	Test <i>getInstance()</i> for its returned object, and check whether it is not a null object, in order to check whether the singleton pattern of AdministratorHK can work normally.
test_getInstanceSZ3	Test <i>getInstance()</i> for its returned object, and check whether it is not a null object, in order to check whether the singleton pattern of AdministratorSZ can work normally.
test_getInstanceBorder3	Test <i>getInstance()</i> for its returned object, and check whether it is not a null object, in order to check whether the singleton pattern of AdministratorBorder can work normally.

2) testNodeEntry.java

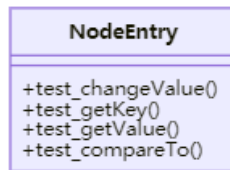


Figure 3.1.1.2 Method Tests of NodeEntry Class

- Test case analysis

Test Name	Purpose
test_getKey1	Test getKey() function of NodeEntry, in order to check whether it can return the desired key value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on integer key values.
test_getKey2	Test getKey() function of NodeEntry, in order to check whether it can return the desired key value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on floating point key values.
test_getKey3	Test getKey() function of NodeEntry, in order to check whether it can return the desired key value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on character key values.
test_getValue1	Test getValue() function of NodeEntry, in order to check whether it can return the correct value field. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on integer values, which we may use in our software.
test_getValue2	Test getValue() function of NodeEntry, in order to check whether it can return the correct value field. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on double values, which we may use in our software.
test_getValue3	Test getValue() function of NodeEntry, in order to check whether it can return the correct value field. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on string values, which we may use in our software.
test_changeValue1	Test changeValue() function of NodeEntry, in order to check whether we can modify its original value field to a new value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on integer value.

test_changeValue2	Test changeValue() function of NodeEntry, in order to check whether we can modify its original value field to a new value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on double values.
test_changeValue3	Test changeValue() function of NodeEntry, in order to check whether we can modify its original value field to a new value. Since the NodeEntry class is a generic class that supports multiple variable types, this testcase is focused on string values.
test_compareTo1	Test compareTo(NodeEntry) function of NodeEntry to check whether this function can verify that the current value is smaller than the target value.
test_compareTo2	Test compareTo(NodeEntry) function of NodeEntry to check whether this function can verify that the current value is equal to the target value.
test_compareTo3	Test compareTo(NodeEntry) function of NodeEntry to check whether this function can verify that the current value is larger than the target value.

3.2 Integration Testing

3.2.1 2nd Layer

1) testStation.java

Station
+test_getName() +test_getId() +test_getAdmin() +test_getEdgeTo() +test_getStationPrice() +test_searchStationByName() +test_searchStationById() +test_stationException()

Figure 3.2.1.1 Method Tests of Station Class

- Test case analysis

Test Name	Purpose
test_getName1	Test the getNameInSpecificLanguage() function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for English station name display mode.

test_getName2	Test the <i>getNameInSpecificLanguage()</i> function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for English station name display mode.
test_getName3	Test the <i>getNameInSpecificLanguage()</i> function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for Traditional Chinese station name display mode.
test_getName4	Test the <i>getNameInSpecificLanguage()</i> function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for Traditional Chinese station name display mode.
test_getName5	Test the <i>getNameInSpecificLanguage()</i> function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for Simplified Chinese station name display mode.
test_getName6	Test the <i>getNameInSpecificLanguage()</i> function of a given station object. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for Simplified Chinese station name display mode.
test_getName7	Test the <i>getNameInSpecificLanguage()</i> function of a wrong station object that cannot be found inside our program's memory.
test_getId1	Test the <i>getId()</i> function of a given station object and check whether it can return us the desired station id number in integer form.
test_getId2	Test the <i>getId()</i> function of a given station object and check whether it can return us the desired station id number in integer form.
test_allEdgeTo	Test the <i>getEdgeTo()</i> function of a given station object and check whether it can return us the correct count of its neighbor edges.
test_searchStationByName1	Test the <i>searchStationByName()</i> function. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great Bay Area, here we test for English station name display mode.
test_searchStationByName2	Test the <i>searchStationByName()</i> function. Noticed that our system is intended to support different system languages to facilitate people's daily lives in the Great

	Bay Area, here we test for Simplified Chinese station name display mode.
test_searchStationByName3	Test the <i>searchStationByName()</i> function. Special check for an extreme condition that the input station name cannot be found anywhere inside the system.
test_searchStationByName4	Test the <i>searchStationByName()</i> function. Special check for the case that both cities contain a station with the same station name, this is to judge the workability of the identifier administrator parameter.
test_searchStationById1	Test the <i>searchStationById()</i> function, to see whether the right station id number can be retrieved successfully from the system.
test_searchStationById2	Test the <i>searchStationById()</i> function, to see whether the right station id number can be retrieved successfully from the system.
test_searchStationById3	Test the <i>searchStationById()</i> function to make sure the system will simply return a meaningless null value when a given station id input cannot be found in our system.
test_getAdmin1	Test the <i>getAdmin()</i> function. And see if the record is correct thus the returned administrator instance is exactly what we want.
test_getNameS1	Test the <i>getName()</i> function with a predefined system language in English.
test_getNameS2	Test the <i>getName()</i> function with a predefined system language in Simplified Chinese.
test_getNameS3	Test the <i>getName()</i> function with a predefined system language in TraditionalChinese.
test_getNameS4	Test the <i>getName()</i> function in a wrong situation that the system has no predefined or default system language in effect.
test_getStationPrice0	Test the static <i>getStationPrice()</i> function where both of the starting and ending stations that we are looking for are within the Hong Kong administration area.
test_getStationPrice1	Test the static <i>getStationPrice()</i> function where the starting station is in the Shenzhen administration area, while the ending station is under Hong Kong's administration. In this testcase, the warning message was expected to be displayed in the predefined English system language.
test_getStationPrice2	Test the static <i>getStationPrice()</i> function where the starting station is in the Shenzhen administration area, while the ending station is under Hong Kong's administration. In this testcase, the warning message was expected to be displayed in the predefined Traditional Chinese system language.

test_getStationPrice3	Test the static getStationPrice() function where the starting station is in the Shenzhen administration area, while the ending station is under Hong Kong's administration. In this testcase, the warning message was expected to be displayed in the predefined Simplified Chinese system language.
test_stationException	Test the exception thrown out of ExStationNotFound() exceptional class.

3.2.2 3rd Layer

1) testEdge.java

Edge
+test_setIsOpen() +test_toString() +test_getLine() +test_getStartSta() +test_getEndSta() +test_getIsConnect() +test_getTimeSpend()

Figure 3.2.2.1 Method Tests of Edge Class

- Test case analysis

Test Name	Purpose
test_getIsConnect	Test getIsConnect() to check whether it can return the isOpen value correctly.
test_setIsOpen	Test setIsOpen() and use getIsConnect() , whose correctness has been tested through the last test case, to fetch the result in order to check whether the isOpen value can be set correctly.
test_toString	Test @Override toString() to check whether it can return edge information correctly.
test_getLine_1	Test getLine() with inLine value being default - null , to check whether it can return correctly.
test_getLine_2	Test getLine() with assigned inLine value, to check whether it can return correctly.
test_getStartSta	Test getStartSta() to check whether it can return the st_station value correctly.
test_getEndSta	Test getEndSta() to check whether it can return the ed_station value correctly.

test_getTimeSpend	Test getTimeSpend() to check whether it can return the time value correctly.
-------------------	--

3.2.3 4th Layer

1) testLine.java

Line
+test_getName() +test_getNameInSpecificLanguage() +test_getEdges()

Figure 3.2.3.1 Method Tests of Line Class

- Test case analysis

Test Name	Purpose
test_getName_1	Test getName() to check whether it can return the English name properly.
test_getName_2	Test getName() to check whether it can return a simplified Chinese name properly.
test_getName_3	Test getName() to check whether it can return a traditional Chinese name properly.
test_getName_4	Test getName() to check whether it can Error return properly.
test_getNameInSpecificLanguage_1	Test getNameInSpecificLanguage() with English language to check whether it can return properly.
test_getNameInSpecificLanguage_2	Test getNameInSpecificLanguage() with simplified Chinese language to check whether it can return properly.
test_getNameInSpecificLanguage_3	Test getNameInSpecificLanguage() with traditional Chinese language to check whether it can return properly.
test_getNameInSpecificLanguage_4	Test getNameInSpecificLanguage() with language to be null to check whether it can return properly.
test_getEdges	Test getEdges() to check whether it can return all edges of a line properly.

3.2.4 5th Layer

1) testDatabase.java

Database
<pre> +test_getInstance() +test_getLineByName() +test_getStationByName() +test_getStationById() +test_translateId2Name() +test_getPrice() </pre>

Figure 3.2.4.1 Method Tests of Database Class

- Test case analysis

Test Name	Purpose
test_getInstance	Test getInstance() to check whether it can return the database instance correctly.
test_getLineByName_1	Test getLineByName() to check whether it can return correctly with simplified Chinese.
test_getLineByName_2	Test getLineByName() to check whether it can return correctly with English.
test_getLineByName_3	Test getLineByName() to check whether it can return correctly with traditional Chinese.
test_getLineByName_4	Test getLineByName() to check whether it can throw ExLineNotFound exception correctly when no Line can be found.
test_getStationByName_1	Test getStationByName() to check whether it can return correctly with English.
test_getStationByName_2	Test getStationByName() to check whether it can return correctly with simplified Chinese.
test_getStationByName_3	Test getStationByName() to check whether it can return correctly with traditional Chinese.
test_getStationByName_4	Test getStationByName() to check whether it can throw ExStationNotFound exception correctly when no station can be found.
test_getStationById_1	Test getStationById() to check whether it can return correctly.
test_getStationById_2	Test getStationById() to check whether it can throw ExStationNotFound exception correctly with wrong id as no station can be found.
test_translateId2Name	Test translateId2Name() to check whether it can turn ids into names and return correctly.

test_getPrice_1	Test getPrice() to check whether it can return the correct travel price of the HK system.
test_getPrice_2	Test getPrice() to check whether it can return the correct travel price of the Shenzhen system.
test_getPrice_3	Test getPrice() to check whether it can return -1 when null administrator is provided.

3.2.5 6th Layer

1) testMetroSystem.java

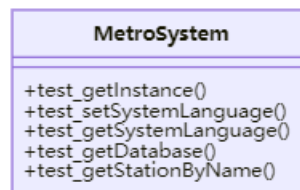


Figure 3.2.5.1 Method Tests of MetroSystem Class

- Test case analysis

Test Name	Purpose
test_getInstance	Test getInstance() to check whether it can return the MetroSystem instance correctly.
test_language_related	Test setSystemLanguage() , getSystemLanguage() to check whether system language can be correctly set and fetch.
test_getDatabase	Test getDatabase() to check whether the database can be correctly returned.
test_getStationByName_1	Test getStationByName() to check whether the specified Station can be returned by correct Name.
test_getStationByName_2	Test getStationByName() to check whether null value can be returned by the wrong Name.
test_getLineByName_1	Test getLineByName() to check whether Line can be returned by correct Name.
test_getLineByName_2	Test getLineByName() to check whether null value can be returned by the wrong Name.

2) testDataList.java

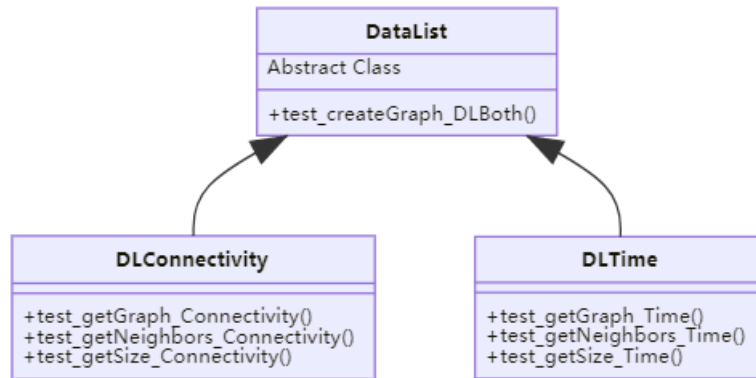


Figure 3.2.5.2 Method Tests of DataList Class

- Test case analysis

Test Name	Purpose
test_getGraph_Connectivity	Test getGraph() of class DLConnectivity to check whether it returns the graph correctly.
test_getNeighbors_Connectivity	Test getNeighbors() of class DLConnectivity to check whether it returns correctly.
test_getSize_Connectivity	Test getSize() of class DLConnectivity to check whether it returns the size correctly.
test_getGraph_Time	Test getGraph() of class DLTime to check whether it returns the graph correctly.
test_getNeighbors_Time	Test getNeighbors() of class DLTime to check whether it returns correctly.
test_getSize_Time	Test getSize() of class DLTime to check whether it returns the size correctly.
test_createGraph_DLBoth	Test getGraph() of both DLConnectivity and DLTime to ensure the values returned are different.

3.2.6 7th Layer

1) testAlgorithm.java

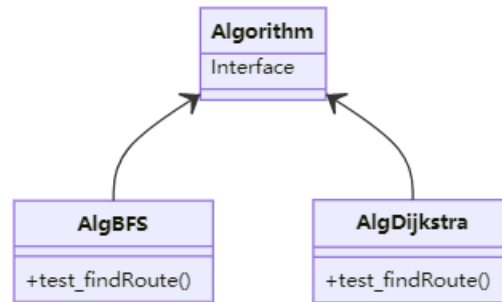


Figure 3.2.6.1 Method Tests of Algorithm Class

- Test case analysis

Test Name	Purpose
test_findRoute_BFS_1	Test <i>findRoute()</i> of class AlgBFS to check whether the BFS algorithm can work and return value correctly.
test_findRoute_Dijkstra_1	Test <i>findRoute()</i> of class algDijkstra to check whether the Dijkstra algorithm can work and return value correctly.

3.2.7 8th Layer

- 1) testCriteria.java

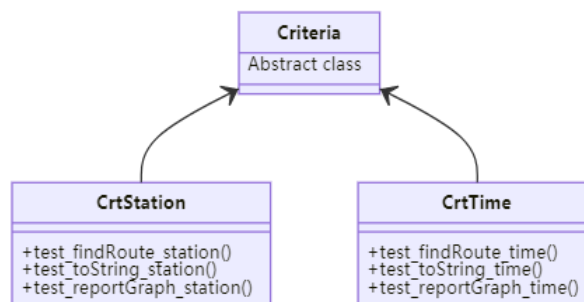


Figure 3.2.7.1 Method Tests of Criteria Class

- Test case analysis

Test Name	Purpose
test_findRoute_station	Test <i>findRoute()</i> of class CrtStation to check whether it can run the correct algorithm(<i>BFS</i>) and can return properly.
test_toString_station	Test <i>@Override toString()</i> of class CrtStation to check whether it can return the criteria name correctly.
test_findRoute_time	Test <i>findRoute()</i> of class CrtTime to check whether it can run the correct algorithm(<i>Dijkstra</i>) and return properly.
test_toString_time	Test <i>@Override toString()</i> of class CrtTime to check whether it can return the criteria name correctly.
test_reportGraph_station	Test <i>reportGraph()</i> of class CrtStation to check whether it can run properly.
test_reportGraph_time	Test <i>reportGraph()</i> of class CrtTime to check whether it can run properly.

3.3 System Testing

3.3.1 9th Layer

- 1) testMain.java
- Test case analysis

Test Name	Purpose
test_main	Test the entire <i>main(arg)</i> to check whether it can run properly.

4 COVERAGE ANALYSIS

Element ▲	Class, %	Method, %	Line, %	Branch, %
metroSystem	100% (21/21)	100% (84/84)	98% (456/464)	97% (75/77)

100% classes, 98% lines covered in package 'metroSystem'

Element ▲	Class, %	Method, %	Line, %	Branch, %
AdministratorBor...	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
AdministratorHK	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
AdministratorSZ	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
AlgBFS	100% (1/1)	100% (2/2)	100% (33/33)	100% (7/7)
AlgDijkstra	100% (1/1)	100% (2/2)	100% (43/43)	100% (9/9)
Criteria	100% (1/1)	100% (4/4)	100% (6/6)	100% (0/0)
CrtStation	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
CrtTime	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
Database	100% (1/1)	100% (13/13)	95% (183/191)	96% (28/29)
DataList	100% (1/1)	100% (5/5)	100% (17/17)	100% (3/3)
DLConnectivity	100% (1/1)	100% (2/2)	100% (9/9)	100% (1/1)
DLTime	100% (1/1)	100% (2/2)	100% (16/16)	100% (1/1)
Edge	100% (1/1)	100% (9/9)	100% (19/19)	100% (1/1)
ExLineNotFound	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
ExStationNotFou...	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
Language	100% (1/1)	100% (2/2)	100% (4/4)	100% (0/0)
Line	100% (1/1)	100% (5/5)	100% (29/29)	100% (9/9)
Main	100% (1/1)	100% (1/1)	100% (2/2)	100% (0/0)
MetroSystem	100% (1/1)	100% (7/7)	100% (24/24)	50% (1/2)
NodeEntry	100% (1/1)	100% (6/6)	100% (8/8)	100% (0/0)
Station	100% (1/1)	100% (10/10)	100% (47/47)	100% (15/15)

Figure 4.1 Final Coverage of the Program

We reached 100% coverage for the classes, methods, 98% for lines or statements, 97% for branches. The main coverage strategy we used when designing test cases is branch coverage, and maximized the percentage of decision and condition coverage.

It is noteworthy that since our system is simulated to be open to everyone, we do have some security concerns. Therefore, we used methods such as file readers with fixed input and output paths and a double-checked lock to create singleton objects to avoid bugs in multi-threaded programs. While the multi-threaded programs would be run in reality, they cannot really be active in our current environment. So, some lines and branches are normally impossible to reach, which is the reason why we have few lines and conditions uncovered.

5 CODE REFACTORING

Because we have made comprehensive planning upon the structure of the program before developing, most functions have been considered in advance. We only made some small modifications for refactoring in order to optimize the structure.

5.1 MetroSystem Class

- Before Refactoring

A screenshot of a code editor showing the MetroSystem class. The code is written in C# and includes a private static instance, private fields for Language and Database, a private constructor that initializes these fields and loads data from a database, and several public static methods for getting and setting the instance, language, and database. The line 'public Database getDatabase() { return database; }' is highlighted with a red rectangle.

```
4 public class MetroSystem
5 {
6     private static MetroSystem instance = new MetroSystem();
7     private Language systemLanguage;
8     private Database database;
9
10
11     private MetroSystem() {
12         systemLanguage = Language.English;
13         database = Database.getInstance();
14         database.loadStations();
15         database.loadEdges();
16         database.loadLines();
17         database.loadPrice();
18     }
19
20     public static MetroSystem getInstance() { return instance; }
21
22
23     public void setSystemLanguage(Language l) { systemLanguage = l; }
24
25
26     public Language getSystemLanguage() { return systemLanguage; }
27
28     public Database getDatabase() { return database; }
29
30
31 }
32
33
34
35 }
```

Figure 5.1.1 MetroSystem Class Before Refactoring

Before refactoring, when we wanted to retrieve the Station class and the Edge class to get the needed information, we would type the code like:

```
m.getDatabase().getStationByName()
```

```
m.getDatabase().getLineByName()
```

However this violates the LoD principle, that is: each unit should only have limited knowledge about other units. We need to order the database to return the information, not to control it.

- After Refactoring

```
44 public Database getDatabase() { return database; }
45
46 public Station getStationByName(String name, Language language, Administrator admin) {
47     try {
48         return database.getStationByName(name, language, admin);
49     } catch (ExStationNotFound e) {
50         System.out.println(e.getMessage());
51         return null;
52     }
53 }
54
55 public Line getLineByName(String name, Language language) {
56     try {
57         return database.getLineByName(name, language);
58     } catch (ExLineNotFound e) {
59         System.out.println(e.getMessage());
60         return null;
61     }
62 }
63
```

Figure 5.1.2 MetroSystem Class After Refactoring

The MetroSystem class is regarded as a facade class of our system. So to better reflect its function as a facade, we add new methods **getStationByName()** and **getLineByName()** in the MetroSystem class. When we need to get the station and edge information, we code like:

```
m.getStationByName()
m.getLineByName()
```

This will avoid violating the LoD principle. Also, in case that other information in the Database is needed, we reserve the **getDatabase()** function.

5.2 Language Class

- Before Refactoring

Before refactoring, we had three singleton subclasses under Language class, each representing a language state. When the system runs, the database would have the LanguageEnglish state by default. And when returning the names of the stations, we would use **instanceof** to judge the system language instance, and return the corresponding name under that language. For example:

```
systemLanguage instanceof LanguageEnglish.getInstance()
```

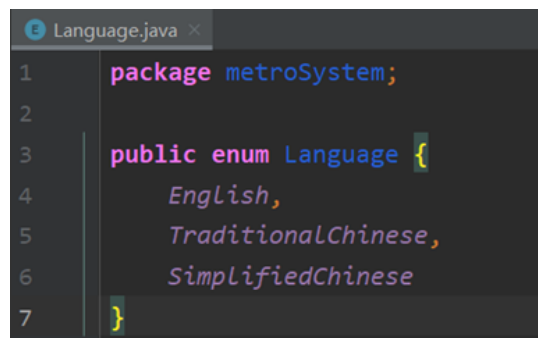
Although the function has no problem returning correct results, judging results by **instanceof** seems not to be a good method. Moreover, representing states by singleton is not a concise way because each singleton subclass needs amounts of code to be built.

Changed the expression of InstanceOf in Language System

- ▶ MetroSystem/src/metroSystem/Language.java
- ▶ MetroSystem/src/metroSystem/LanguageEnglish.java
- ▶ MetroSystem/src/metroSystem/LanguageSimplifiedChinese.java
- ▶ MetroSystem/src/metroSystem/LanguageTraditionalChinese.java
- ▶ MetroSystem/src/metroSystem/Line.java
- ▶ MetroSystem/src/metroSystem/Main.java
- ▶ MetroSystem/src/metroSystem/MetroSystem.java
- ▶ MetroSystem/src/metroSystem/Station.java

Figure 5.2.1 Refactoring of Language Module by GitHub and Fork

- After Refactoring



```
1 package metroSystem;
2
3 public enum Language {
4     English,
5     TraditionalChinese,
6     SimplifiedChinese
7 }
```

Figure 5.2.2 Language Class After Refactoring

After refactoring, we introduce the enumeration representation to simplify the states. Now when judging the system language, we use:

```
systemLanguage == Language.English
```

It reduces the amount of code to some extent, and abandons the *instanceof* judging method.