

# EBU5304 SOFTWARE ENGINEERING COURSEWORK

## TOURIST TRAIN SYSTEM DEVELOPMENT

GROUP 44

## Content

Summary of Responsibilities and Achievements .....	3
1. INTRODUCTION .....	1
2. PROJECT MANAGEMENT .....	1
2.1 The Scrum Process .....	1
2.2 Extreme programming.....	2
3. REQUIREMENTS .....	2
3.1 Requirement Finding Techniques.....	2
3.2 Development of the Backlog.....	3
3.4 Iteration and Estimation of the Stories .....	4
3.5 Software Prototype.....	5
4. Analysis & Design .....	6
4.1 Class and System Design .....	6
4.1.1 Class Stereotypes.....	6
4.2 Design Principle Applied .....	12
4.3 Re-usability of System Components .....	12
4.4 Further Development.....	13
5. IMPLEMENTATION .....	14
5.1 Identify the Components .....	14
5.2 Integration Build Plan .....	14
5.3 Mapping design to codes .....	16
5.4 Pair Programming.....	16
6. TESTING .....	17
6.1 Introduction and Test Strategy .....	17
6.2 Unit Test .....	17
6.3 System Tests .....	23
7. CONCLUSION .....	24
8. REFERENCE .....	24

## Summary of Responsibilities and Achievements

**Group Number:** 44

**Group Members:**

Name	QM Student No.	BUPT Student No.
Yuqi SHI	130806701	2013212998
Hanxiao MA	130800570	2013213076
Jinfeng REN	130802242	2013213243
Yihan HE	130803412	2013213352
Peng LIN	130801692	2013213188
Xiaokun WANG	130799344	2013212953

**Group Leaders Name:** Yuqi SHI    130806701    2013212998

**Individual Member:** Yuqi SHI    130806701    2013212998

### Individual Members Contribution and Self Appraisal of Work Done

#### Individual Member Contribution

Backlog	Write user stories and specify requirements; Understand all members' user story cards and sift out the useful ones; Translate the user stories into the form of product backlog.
Program	Design the structure of the management system and Identify class relationships Realize the basic operation of operation manager interface; (add/delete/modify a route)
Report	Responsible for the Requirement and Implementation section; Modify the Introduction and Conclusion part; Arrange the appendix of screenshots and references; Integration and type setting of the report.

During these two months, our group has done the work step by step in realizing all the required functions of "Tourist Train Journey Management system". However, at first, our group spent about two weeks on requirement specification part to design a reasonable structure. Through the heated debate, we finally figure out which part is overdesign and which requirements can be implemented in one method. On the other hand, along with the course progress, we try to apply more design principles that we learn from class into our system.

#### Self-appraisal

As the group leader, one of the most crucial task for me is to organize team meeting, such as weekly reports, and thanks to our members, for most time, they can attend meetings on time. What's more, it's really a great challenge to engage everyone in each part of the work. In practice, I find that during pair programming, it's a good idea for the pair to consist a new programmer and an experienced one. The new programmer can learn a lot from the experienced person and in return they can provide new ideas and fresh thinking patterns.

**Individual Member:** Hanxiao MA 130800570 2013213076

### **Individual Members Contribution and Self Appraisal of Work Done**

#### Individual Member Contribution

Backlog	Write user stories to specify operation managers' requirements; Prioritize and estimate user stories to meet the real practice; Specify the iteration of the backlog.
Program	Allow a route runs a number of trips(contain train and driver) Realize of timer synchronization with the system time of PC; Provide the mechanism in the whole program;
Report	Write the content of future development of Analysis and Design section; Modify the implementation part of the report; Specify the iteration and estimation of the stories in the requirement section.

I realize the importance of the Agile method in practice during the teamwork. We set a priority of all the features, and did a part of it each iteration. Agile method makes the coding work methodical and we could modify our program easily whenever the requirements change. Our daily stand-up meeting can solve the problem timely.

#### Self-appraisal

In previous coursework, I used to do all the programming job by myself. During this coursework, by applying the Agile methods, our group leader asks us to do pair programming. It's really a brand-new experience for me and I have to admit that different thinking patterns would definitely improve efficiency and provide a better outcome. When I face the circumstance that the program can't run correctly without reporting errors, I would print the key value in that wrong method to debug.

**Individual Member:** Jinfeng REN 130802242 2013213243

### **Individual Members Contribution and Self Appraisal of Work Done**

#### Individual Member Contribution

Backlog	Add the user stories about the interface design Modify acceptance criteria for each interface Provide advices on the priority of the whole product backlog;
Program	Code the user interfaces using JAVA GUI (operation manager); Designing and drawing the prototype of the GUI; Integrate all the code at the end.

Report	Provide the main screenshots of the Appendix A and test outcomes; Provide advices for the class stereotypes in the analysis & design part; Drawing the prototype of the software.
--------	---

Our team utilize the principle of pair programming perfectly. We are divided into 3 pairs and have a long-time group meeting once a week. During the frequent discussion, we can have a clear structure while quite a lot of new ideas appear. This method decreases our assignments and makes it quite easy when we finally combine all of our codes together.

### Self-appraisal

During this Agile software development, I gain the experience of how to efficiently schedule the program and overcome the difficulties including refractory my code, my design as well as debugging. Also I reviewed all the JAVA knowledge and implemented all the agile programming methods that I learnt in class. Firstly, I write our code in Eclipse however when I finally want to run our codes in cmd, there is something unexpected happens. I find that I have to delete all packets then there won't be that much errors. Anyway, I think this is a wonderful grouping working experience.

**Individual Member:** Yihan HE 130803412 2013213352

### Individual Members Contribution and Self Appraisal of Work Done

#### Individual Member Contribution

Backlog	Write user stories to specify drivers' requirements; Add acceptance criteria to some of the driver stories. Modify the note messages which are specific for drivers.
Program	Designing and drawing the prototype of the GUI; Code the user interfaces using JAVA GUI (tourist, driver);
Report	Draw the stereotypes of classes and the class diagram; Responsible for the implementation section of the report; Assist on the arrangement of Appendix .

During the first meeting we determined each member's responsibility and we then decided to make a communication conference once a week to report the performance of each. During this process we learnt the importance of agile thought and everyone can learn the current schedule of work. But on the other side, I think sometimes we'd better improve efficiency by saving more time on solving questions about the detail of our codes for they could be easily resolved in on-line discuss.

Self-appraisal

One of my task is to analysis the class and system design of the whole program and determine the design principle applied to it and the re-usability of system components. In this process I met some difficulties to explain some lines of code, so I asked my partners and they answered to me with patience so I could finally analyse each class and the relationships among them successfully. In conclusion, I learned the importance of communication and time scheduling during teamwork and I apply knowledge taught in class like agile work, design principle and re-use of class and method, which I believe would help me improve the efficiency when I write a program.

**Individual Member:** Peng LIN 130801692 2013213188

**Individual Members Contribution and Self Appraisal of Work Done**Individual Member Contribution

Backlog	Write user stories to specify tourists' requirements; Add acceptance criteria to some of the tourist stories; Modify the note messages which are specific for tourists.
Program	Analyze IO part of the whole program Assign driver and train to a specific trip in a route Design Junit and write Junit code
Report	Write the content of scrum process in the project management part; Analyze test procedure and test techniques Design test procedure and test cases

In the whole coursework process, not only I pay large attention to complete my mission, but also my teammates show their enthusiasm and wisdom to solve the matter which we face. If I just programming by myself the process may not go forward that quickly. During the whole process our leader offers the latest information to us and makes a large positive effect to the process.

But in the other hand, there also have some mistakes happened in the process. Such as the programs turn up many bugs when we combine the codes together at the end of programming part. It cost us lots of time to debug the code.

Self-appraisal

One of the work I do during the process is programming the in/out put of file in the manage system. I use the knowledge which I have learned during the class and find the useful information as many as possible from the internet. Since the operation of the file are difficult and micromesh, so I tried my best to figure out any cases that would happen and use code to achieve them. Also I write some Junit code to test my java code whether it can run perfectly. From the whole process I have improved my programming skill not only in coding by also in self-thinking, logic function using and communicating with partners. These skills may help me with my studying and working in the future.

**Individual Member:** Xiaokun WANG 130799344 2013212953

### **Individual Members Contribution and Self Appraisal of Work Done**

#### *Individual Member Contribution*

Backlog	Write user stories for the requirements of develop team members and stakeholders; Consider the control from both operation manager and driver and add its acceptance criteria to the backlog.
Program	Design program about read-write. Analyze TDD cycle Design Junit and write Junit code
Report	Analyze TDD cycle for each interface; Write test cases and implement tests; Assist on the arrangement of Appendix B.

During the coursework, I work together with my team members to overcome problems. We usually took a meeting to consult what we should do. Every member played an important role in the project. I realized the importance of agile thought in the teamwork. I think the coursework improved the ability of everyone.

#### self-appraisal

From the coursework, I learned a lot. My partners all performed well. Also I tried to finish the coursework of my part. I joint group meetings for discussing the steps and details. When we had meeting, their ways of thinking gave me inspirations. Thanks to the coursework, I got to apply my knowledge to practice and harvested friendship.

## 1. INTRODUCTION

The report shows the complete agile software development process of a tourist train management system. The whole process using Scrum Process which consists 4 sprint cycles. During these cycles, we analyse, design, implement and test a train management system which has a series of functions that can meet all the requirements stated on the coursework hangout. Besides that, we also realized some of the possible requirements on future.

## 2. PROJECT MANAGEMENT

According to the course acquirement, agile software development process is applied in the train management system development. Agile software development process, which focus on the direct communication among people, cuts off unnecessary documentation and pursues higher effective software developing. We use the method of Scrum and Extreme Programming to achieve Agile.

### 2.1 The Scrum Process

Our group applies the Scrum approach in agile project management. The Scrum approach, which is a general agile technique that focus on the managing iterative development, consists of three main phases including the initial phase, a series of sprint cycles and the project closure phase.

#### 2.1.1 The Initial Phase

The initial phase is the start point of an agile project. On this phase, we spent for about two weeks searching for the functional and non-functional requirements of users, analyse and design the software architecture, and then list a series of user stories to make the product backlog with the involvement of end users.

#### 2.1.2 The Sprint Cycle

Agile software development is an incremental process. The period of time, during which an increment should be accomplished is called the sprint cycle. A sprint cycle contains four steps: assess, select, develop, and review. A lot of agile project management techniques are applied on this phase and all of them are talked about on next paragraphs. There are four sprint cycles on our project and the sprint backlog are selected from the product backlog.

#### 2.1.3 The Project Closure Phase

After finishing all the development of software program, all the developers should complete required documentation, such as user manuals, and assess the experience learned from the project. We write the report of the agile software development and the user manual and readme file for the program on the closure phase.



## 2.2 Extreme programming

### 2.2.1 Iteration planning

Our group define every two weeks as an iteration and plan to deliver increments to the system each sprint. Firstly, we break all of the requirements into user stories, then we sit around a table and determine release priority and specify the detailed requirement that we should meet in each iteration.

### 2.2.2 Continuous Integration

Our team releases small running tested software which meets the requirement of each iteration that planned in the iteration planning. Everyone is engaged in each part of the work. Frequent releases enable us to find errors timely and resolve them.

### 2.2.3 Pair programming

During pair programming, our group enable every pair consist a new programmer and an experienced one. The new programmer can learn a lot from the experienced person and in return they can provide new ideas and fresh thinking patterns. It's quite important to note that pair programming is extremely helpful for decision making.

### 2.2.4 Adapt to changes.

We emphasis on simple design and refactoring which improving the existing code. What's more important. We avoid any duplication in our code and this will inevitably facilitate the incremental development. Reduce Coupling and increase cohesion is also required to our code.

## 3. REQUIREMENTS

### 3.1 Requirement Finding Techniques

#### 3.1.1 Background Reading

Our job is to design a train journey management system containing three interfaces, which is used separately by operation manager, driver and tourist at station. It can automatically synchronize within the system, and thus provide the newest information for users.

As for background reading, we looked into details of related paper and documentation projects referred to similar train management projects. We learned that in a complete train journey system, the most important part is for the operation manager. The driver and tourist system only read the information of the operation manager system. Three subsystems are linked together by time synchronization which enable everyone get the newest information when facing changes.

### 3.1.2 Observation & Interviewing

To obtain some more specific information of the system, we interviewed the tourist and the operation manager of some realistic train management system.

- **Users:** users mainly focus on the type of service they can get from the train journey system. From the interview we find that tourist in the station care about the basic functions of the system such as the arrival and departure time of all the trains passing specific station. When using the system, users want it can be quite efficient searching for the information.
- **Administrator:** Administrators mainly focus on what functions can modify the whole system more coherent and more reasonable for all the users. This role should care about all the details functions in the system, in order to do this the administrator should also know clearly users' activities.

### 3.1.3 Document Sampling

After concluding the documentation of our system, and searching for some other train journey management system, we find some imperfect aspects in existing system that we should take into further consideration:

- We assume that there is no train delayed and broken in our system. Trip is a return round trip of a specific train. The fault-tolerance mechanism is realized that a train shouldn't be allocated to two trips at the same time.
- Some viewpoints from the stakeholders (non-functional requirement) such as the best-practice allocating plan of train and driver to achieve the highest profit with lowest cost.

## 3.2 Development of the Backlog

A product backlog is a prioritized list of work for the development team that is derived from the handout and its requirements. [5] Over our course of the project, the product backlog continued to evolve. As we drilled into the requirements behind some of the earlier user stories, we discovered they were bigger than anticipated. We broke apart user stories into two or more product backlog items. Occasionally, we combined some different stories and change the priority.

As new stories were added, we assigned them story points and a general priority. We rank the user stories. Then we input changes and update the user stories. Eventually, we even got the syncing functionality to work so I could make updates in Excel.

Since the first submission, we changed the following aspects:

- **Control information:** In our final system, the control information that determine the train's state is an information sent by an operation manager. We don't simulate the control button for driver in our product.

- Security improvement: We add the error-avoidance mechanism to avoid there are two train departure from the same station at the same time.
- Work schedule: at the beginning, we have planned to execute five sprint cycles to finish the development. However, due to the lack of experience, we misestimate the working efficiency. Thus, we adjusted the sprint backlog of following cycles and finally we complete the development on four sprint cycles.
- Non-functional requirement: Some non-functional requirements is added to backlog such as the maximum profit determined by the stakeholders.

### 3.4 Iteration and Estimation of the Stories

After identifying requirements, our group gets into the step of Iteration and estimating. They are meant to be high-level estimates that are used to drive iteration planning. Over time, it becomes easier to break down features into units of work that can be delivered within a single iteration.

#### 3.4.1 Iteration

During the planning for iteration, our group creates engineering tasks that allow ourselves to deliver the stories planned for the iteration. Our group iteration includes 4 sprint cycles and each one is planned to be finished on one week. The standard of deciding iterations are the priority of user stories and the cost to finish them.

Iteration	User Story
One	<ol style="list-style-type: none"> <li>1. Operation manager can add/delete/modify one route in the system and assign a train to a specific route.</li> <li>2. Design the basic operation manager GUI of the train system.</li> <li>3. Save the information of route and train to the text file.</li> </ol>
Two	<ol style="list-style-type: none"> <li>1. Operation manager can assign many trips to a specific route at different time. A trip should contain the information of train and driver.</li> <li>2. Further design the operation manager GUI of the train system.</li> <li>3. Error-avoidance mechanism: We can't assign many trains departure from a station at the same time. What's more, a train or driver can't be assigned to many trips at a time</li> </ol>
Three	<ol style="list-style-type: none"> <li>1. We can search a specific trip and display all of the functions</li> <li>2. We can display the location of a specific train through an interface to the main system.</li> </ol>
Four	<ol style="list-style-type: none"> <li>1. Design the GUI for driver and tourist at the station</li> <li>2. The administrator can send control information to driver for controlling the train state</li> </ol>

### 3.4.2 Estimation

To achieve the goal of making non-programming stuff to become clear with our project. There are two main estimating units which agile teams use to concentrate on programming:

---Work Units: These represent the relative amount of work required to implement a feature, compared to other features. Our team settled into a consistent velocity, usually over a few iterations, and we begin to map these work units to units of actual time.

---Ideal Time: Ideal Time excludes non-programming time. We use Ideal Time for estimating.

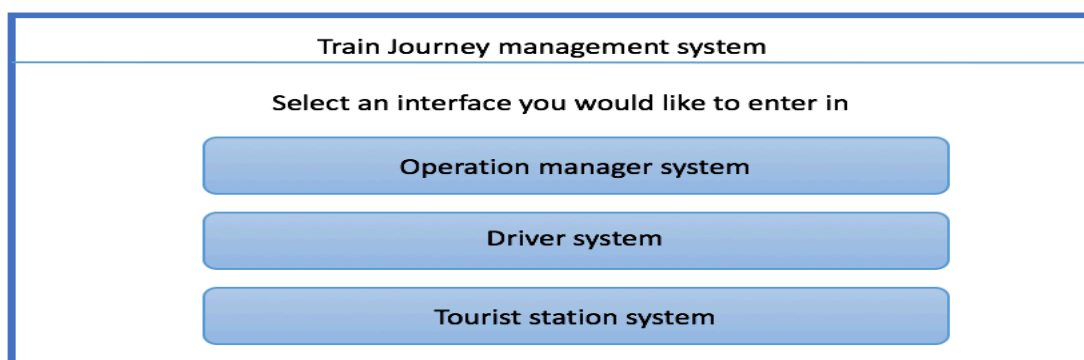
For Accurate Estimates, our group takes the following things into consideration and makes a note on the card.

1. Estimate in consideration of ideal engineering days (story points), not calendar time.
2. Use velocity to determine how many story points the team can finish in an iteration.
3. Use risk management to adjust for risks to the overall release plan.
4. When we find the story is estimated greater than 13 story points, we just break it down into several parts.

## 3.5 Software Prototype

### 3.5.1 Starting interface

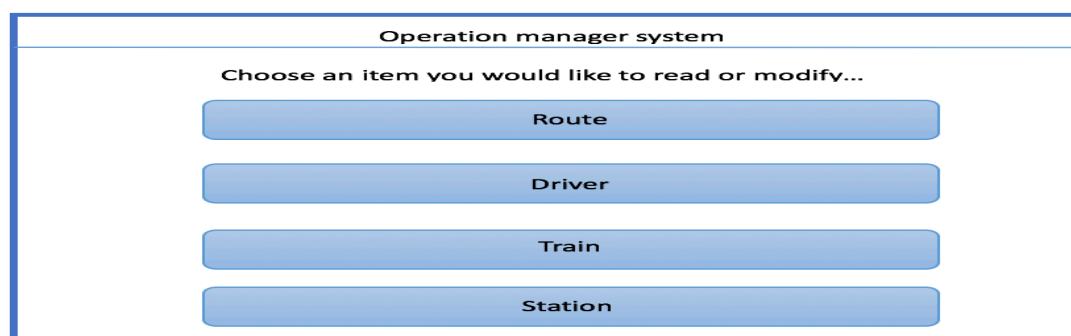
Users may see starting interface firstly when entering the program. The main UI may ask the user to choose their identity and enter the corresponding interface.



The screenshot shows a window titled "Train Journey management system". Inside, the text "Select an interface you would like to enter in" is centered. Below this text are three blue rectangular buttons stacked vertically, each containing white text: "Operation manager system", "Driver system", and "Tourist station system".

### 3.5.2 Operation manager interface

Operation manager can manage route driver and train. They can also read the information of the station.



The screenshot shows a window titled "Operation manager system". Inside, the text "Choose an item you would like to read or modify..." is centered. Below this text are four blue rectangular buttons stacked vertically, each containing white text: "Route", "Driver", "Train", and "Station".

### 3.5.3 Create a new route

**Create a new route**

Both the departure and the terminus is the Center station. You only need to choose the intermediate stations.

Add

Finish adding

## 4. Analysis & Design

### 4.1 Class and System Design

#### 4.1.1 Class Stereotypes

Class	Content
<b>Boundary classes</b>	BgPanel, bkground, CreateRouteInterface, CreateJourneyInterface, EachTrainStopScreen, DriverInterface, InitialInterface, LocomotiveInterface, ManagelInterface, OnBoardScreen RouteInterface, SearchInterface, StationInterface, StopJourney, JourneyInterface, TimeScreen TrainStopScreen
<b>Control classes</b>	AllTrain
<b>Entity classes</b>	Driver, AllStation, Locomotive, Train, simulationTest

Class name	simulationTest	Description
		Simulate the running of the train (convert 1 minute in real world into 1 hour in our system)

Class name	AllTrain	Description
Attribute	ArrayList<Train> allTrain	journeys
Operation (*the information of a journey includes the number of journey, the information of the route, the name of driver and the name of locomotive)	AllTrain()	Constructor method
	initialize()	Read name of journeys from 'allTrain.txt'
	getAllRoute()	Get all the routes
	getAllTrain()	Get all the journeys
	getRoute()	Get all the stations on a specified route
	getRouteTrain()	Get all the journeys of a specified route
	addTrain()	Create a new journey
	nameExist()	Judge whether a specified journey exists
	getTrainByName()	Get the information of a specific journey
	removeTrain()	Delete the information of a specific journey in 'allTrain.txt' and remove relevant files about it
	allPassThroughThisStation()	Get all the journeys of trains pass though a specified station
	allFromFirstToSecond()	Get all the journeys of trains travel from a specified station to another specified station
	allThroughFirstAndSecond()	Get all the journeys of trains pass though two specified stations
	checkTimeConflict()	Avoid two trains pass though a station at the same time
	updateAllTrainLocation()	Update the location of all the trains in corresponding trainlocation file
	toString()	Return all the number of journeys

Class name	AllStation	Description
<b>Attribute</b>	ArrayList<String> allStation	Name of station
<b>Operation</b>	AllStation()	Constructor method
	initialize()	Read information of stations from 'allStation.txt'
	getAllStation()	Get all the stations
	addStation()	Add new stations
	nameExist()	Judge whether a specified station exists
	removeStation()	Delete a specified station

Class name	Driver	Description
<b>Attribute</b>	String name[]	Name of driver
<b>Operation</b>	new_member()	Add new driver
	addToRoute()	Assign a driver to a journey
	deleteTripFromDriver()	Delete a journey of a driver
	deleteDriver()	Remove a driver
	allDriver()	Return the name of all drivers from 'driver.txt'
	search_name()	Get all the journeys of a specified driver
	find_Freetime()	Get the free time of a specified driver
	checkTimeConflict()	Judge whether the driver is free during a specified period

Class name	Train	Description
<b>Attribute</b>	String trainName	Number of a journey
	String fileNameOfTimeTable	Name of file storing time table(information of journeys)
	String fileNameOfTrainLocation	Name of file storing the locations of all the trains
	String routeName	Route name
	String locomotiveName	Train name
	String driverName	Driver name
	ArrayList<String> stationName	Station name
	ArrayList<Integer> arrivingTime	time table(information of journeys)
<b>Operation</b>	Train()	Constructor method
	clearTimeTable()	Clean all the content of time table
	addNewStation()	Add a new station
	getTrainName()	Get number of journey
	getLocomotiveName()	Get locomotive name
	getDriverName()	Get driver name
	getRouteName()	Get route name
	getStationName()	Get station name
	getArrivingTime()	Get the arriving time of a station
	getTimeTable()	Get the time table from file
	updateTrainLocation()	Write the location of train into trainlocation file
	getTrainLocation()	Get the location of train
	passThroughThisStation()	Judge whether this train of journey passes though a specified station
	fromFirstToSecond()	Judge whether this train of journey travels from a specified station to another specified station
	deleteTrain()	Delete all the information of this journey
	toString()	Return the current number of journey

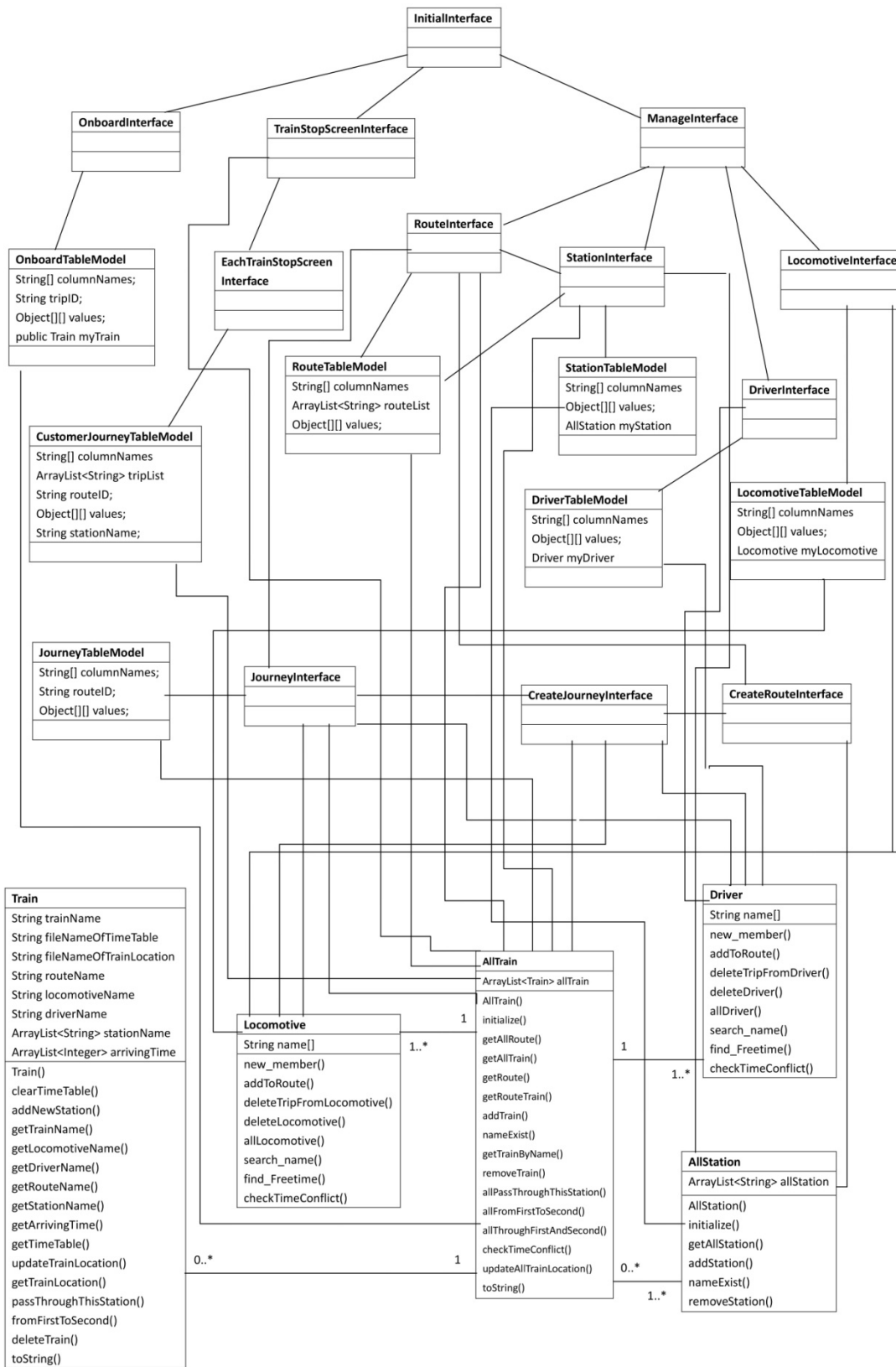


Class name	Locomotive	Description
<b>Attribute</b>	String name[]	Name of train
<b>Operation</b>	new_member()	Add new train
	addToRoute()	Assign a train to a journey
	deleteTripFromLocomotive() ( )	Delete a journey of a train
	deleteLocomotive ( )	Remove a train
	allLocomotive()	Return the name of all trains
	search_name()	Get all the journeys of a specified train
	find_Freetime()	Get the free time of a specified train
	checkTimeConflict()	Judge whether the train is free during a specified period

There should be some other boundary classes on our program (like showing the interfaces and the table into interfaces). However, in order to promote the progress of software development quickly, we didn't design the separated boundary classes. Instead, we combine them together in the GUI classes. It saves the time for us to consider and design the class relationships among them, and the functions are more clear with the assistance of interfaces' notice. All necessary classes are shown on the class diagram.

#### 4.1.2 Class Diagram

The class diagram shows the attributes and operations of every class and the relationships among classes.



## 4.2 Design Principle Applied

### 4.2.1 Single Responsibility Principle (SRP)

SRP requires that every object in the system should have a single responsibility, on which all the objects' services should be focused. Based on the aim to achieve all the functions, we divided independent classes and each of them has unique responsibility. For instance, the class `alltrain` and class `train`, the former operates on all the journeys, such as delete a journey and show all the journeys of which start from a specified station. But the latter operates only on a specified journey.

### 4.2.2 Open-Closed Principle (OCP)

OCP requires the software modules to be “open for extension” and “closed for modification”, which means a module still behaves well in different ways as the requirements change without changing the code. For example, in our progress, the control class, like class `train` is open for extension, and when we store all the data into class like `CustomerTripTableModel`, it cannot be modified. When we want to expand new functions for our system, such as showing the current location of a specific train, we only need to use the stored data and write a new method in class `train` to calculate the location of a train, while the code to store data in table is never changed.

### 4.2.3 Don't Repeat Yourself (DRY)

DRY suggests programmers not to “copy and paste” the similar codes that can be used in similar cases. In our program, we firstly found that there exist some common between driver and train, such as each driver or train can only be assigned to one journey. So we write a method like a template, and the method create a file to record the name of all the objects, and we can add object by adding the name in this file, the same as the method we write to delete objects. Based on the two methods, we rename the object as ‘driver’ or ‘locomotive’, and name the file as ‘driver.txt’ or ‘locomotive.txt’, then the standard methods can be used to add or delete driver or train.

## 4.3 Re-usability of System Components

Instead of writing new code for a module to provide some required service, existing code can be use to improve efficiency. And there are two aspects of code reusing on our program:

### Method reusing

As is taken as an example in design principle, we can apply some methods in driver class into locomotive class. What's more, we can also use some methods in class `train` into `alltrain`. For example, we use a method to judge whether a train passes though two specified stations in class `train`, and in class `alltrain`, when we want to write a method that can find all the train that pass though two specified stations, we can reuse the former method to judge whether each of train passes though two specified stations, if it is, then we add new codes to output it.

## **Class reusing**

When we write the class locomotive, we find that the whole class driver can be a standard and we can make adaption from it without writing any new code because there are too much common between the operations and parameters in these two classes. On the other hand, we also take a reference from the GUI of other program. In order to make the interface pleasing, we download some programs from the Internet; evaluate their interface and select the relevant codes that to be used by us.

## **4.4 Further Development**

### **4.5.1 Adapting to Requirement Changes**

From the group coursework handout, we get a series of possible future changes.

#### **Apply the program into a mobile terminal:**

In the future the requirement of increasing various clients, the program may need to be applied to the mobile terminal in order that clients can make operations on their own mobile phones. This can be achieved by making an APP to accomplish all the functions in the travel system program.

#### **Implement the network architecture:**

If the program can be implement using the network architecture as a basement, different people can log in different interfaces and make different operations, such as the manager can add new journey and delete it, but the client can only search routes, but have no authority of changing routes. What's more, the manager can also send messages to others, such as informing the driver which train he is assigned to, from the manager's computer to the driver's computer.

#### **Add a ticketing system:**

When the manager add a new journey, he can add a new parameter—the number of ticket. As the client selects a journey he can get one ticket and the number of reserving ticket decreases. When there is no reserving ticket, the journey is unavailable for the client. At the same time the number of ticket a client owns can be added into client account.

#### **Enrich the information of stations:**

Now the information of stations only concludes its names, so in order to humanize the system, we can add more information, such as the temperature of where the station located, and some beautiful sights the location has. Besides, this function can combine with the function showing the current location of a train. When the program calculates the location of the train, it can not only embody it on progress bar, but also judge which station the train is to reach, then it will show some tips about the temperature and nearby sights to customers.

### **4.5.2 Using Design Pattern**

Design pattern is a general design mode which can be used in many circumstances to solve a particular problem and has no limitation of program language. The reuse of the same design pattern is a good way to make the design more reasonable, and also make software development more efficient and effective.

## Observer Pattern

We can add a class named JourneyObserver, which implements the observer interface and make the Entrance class extends the observable class. Thus, the JourneyObserver can help to monitor the change of journey in ManagerInterface if the manager add or delete a journey, and thus a driver or a train is assigned or released by the method update (observable obj, object action). Similarly, the Observer Pattern can be used to generate a report recording the operations of manager and print the ticket including information about the journey to the client by adding extra observer classes.

## 5. IMPLEMENTATION

All system codes and documentations are provided in Appendix.

### 5.1 Identify the Components

The standard stereotypes of components are listed below:

#### <Executable>

InitialInterface.java, simulationTest.java

#### <File>

AllStation.java, AllTrain.java, BgPanel.java, CreateRouteInterface.java, CreateJourneyInterface.java, Driver.java, DriverInterface.java, EachTrainStopScreen.java, JourneyInterface.java, Loconotive.java, LoconotiveInterface.java, ManagerInterface.java, OnBoardScreen.java, RouteInterface.java, SearchInterface.java, StaionInterface.java, StopJourney.java, TimeScreen.java, Train.java, simulationTest.java, TrainStopScreen.java, AllTrainTest.java, DriverTest.java

#### <Table>

DriverTableModel.java, RouteTableModel.java, JourneyTableModel.java, OnBoardTableModel.java, EachTrainStopScreenTableModel.java, LocomotiveTableModel.java, StaionTableModel.java

#### <Document>

allStation.txt, allTrain.txt, bg.jpg, Driver.txt, Locomotive.txt

### 5.2 Integration Build Plan

#### 5.2.1 Build Plan Version 1.0

##### Build 1.0

In this build version, we established the whole system's user interfaces by the prototypes and designed the core functions of our system after we listed all the requirements we need to finish.

##### Functionality:

The function we finished in this build version was enabling operation manager to add delete or modify routes and manage journeys of a specific route in our train management system. We also designed the route and journey interface and save the route and journey information as an external file. The preliminary function we finished gave a complete structure of the whole system which proved a clearly way for doing the following works.

**Java File Involved:**

AllTrain.java, RouteInterface.java, CreateRouteInterface.java, JourneyInterface.java, CreateJourneyInterface.java, ManagerInterface.java, RouteTableModel.java, JourneyTableModel.java, BgPanel.java, AllTrainTest.java, Train.java

### 5.2.2 Build Plan Version 2.0

#### Build 2.0

In this build version, we further developed the operation manager interface and implemented the data IO parts and checked the codes by printing the IO results in the command line interface.

**Functionality:**

The function we finished in this build version was allocating driver and locomotive to a specific journey. We improved our main interface and save the driver and locomotive information as external files. For the security of the data, all data were only written into and read from the txt files by using our software.

**Java File Involved:**

Driver.java, DriverInterface.java, DriverTest.java, Loconotive.java, LoconotiveInterface.java, DriverTableModel.java, LocomotiveTableModel.java

### 5.2.3 Build Plan Version 3.0

#### Build 3.0

In this build version, we designed two subsystems called On Board system and Station system which read information from the operation manager system. Operation manager is able to manage the station information and stop train remotely this time. And all the functions can be realized in the GUI rather than the command line interface.

**Functionality:**

The function we finished in this build version was to design the the interfaces for driver on board and tourist in the station separately. It provides integrity for our whole system.

**Java File Involved:**

OnBoardScreen.java, StaionInterface.java, StopJourney.java, TrainStopScreen.java, AllStation.java

### 5.2.4 Build Plan Version 4.0

#### Build 4.0

In this build version, we realized the rest of the functions based on the requirements. And all the functions can be realized in the GUI rather than the command line interface.

**Functionality:**

The function we finished in this build version was the executives among the GUIs by the user's interactions which were realized by the IO part in build version 2.0. We add many error protection mechanism and realize the searching function for all interface. Simulation is added for imitate reality.

**Java File Involved:**

SearchInterface.java, StopJourney.java, TimeScreen.java, simulationTest.java

## 5.3 Mapping design to codes

### 5.3.1 Operation Manager

Device Functional Model	Implementation Model
Operation manager can operate on the all stations	AllStation.java
Manager can modify the train's number and state	AllTrain.java
Manager can allocate driver to specific journey	Driver.java
Locomotive can be added or deleted in the whole database	Locomotive.java
All Operations that you can carried out for a journey	Train.java
Operation can stop the train remotely	StopJourney.java
The interface contains the specific operations that the manager can do.	ManagerInterface.java

### 5.3.2 On Board system

Device Functional Model	Implementation Model
Read all information about a particular on-board journey from the database	AllTrain.java
The interface designed for the driver on board	OnBoardScreen.java
Regulate the form to be saved in the on board system	OnBoardTableModel.java

### 5.3.3 Station system

Device Functional Model	Implementation Model
	TrainStopScreen.java
Read all trains information that passing a specific station from the database	AllTrain.java
Read all station information from the database	AllStation.java

## 5.4 Pair Programming

At first, the pair is made up randomly which leads to a low working efficiency. However, during real practice, the leader finds that it's a good idea for the pair to consist a new programmer and an experienced one. The new programmer can learn a lot from the experienced person and at the same time they can provide brand new ideas and fresh thinking patterns making the work processed in a more efficient way.

What's more important, considering the different programming habits, pair programming provides more convenience when we integrate every part of the codes together at the final stage of our whole work.

## 6. TESTING

### 6.1 Introduction and Test Strategy

Testing is to verify the results from the implementation stage by testing each software build include internal builds, intermediate builds, final customer or external part system software builds. To make sure the software meets all requirements and the internal program logic is correct, both component level and integration level are also involved in the testing part. Different testing techniques are applied in different testing levels. Moreover, Object-oriented testing technique is applied on all testing level.

### 6.2 Unit Test

In unit test, we use Test Driven Development cycle and Write-box techniques to test unite cases. In each selected component test, it includes two conditions: correct situation and incorrect situation in order to ensure its validity.

#### 6.2.1 Test 1 for Add Driver

(1) Test procedure for correct input:

1. From the main screen, choose the management system at first and then click the “Driver” button. The driver interface is displayed.
2. From the driver interface, click the “Add new driver” button. Then, you can input a name.
3. Input “XiaoMing” and select “确定”. The system successfully adds this driver and return “input success”.

(2) Test procedure for incorrect input:

1. From the main screen, choose the management system at first and then click the “Driver ” button. The driver interface is displayed.
- 2.From the driver interface, click the “Add new driver” button.Then, you can input a name.
- 3.Input “rjf” (the system already has this driver) and select “确定”. The system fail to add this driver and return “this name already exist”.



**TEST MATRIX:**

Test Case: File Open	#Test Description	Test Cases	Pass/ Fail	No. of Bugs	Bug #	Comments
N/A	Create driver in system [Driver rjf exists in system]	setup	-	-	-	Added to system for testing purposes
1.1	Test that XiaoMing does not exist in the system, so add successfully	1.1	P	0	0	Correct behavior observed
1.2	Test that rjf does exist in the system, so fail	1.2	P	0	0	Correct behavior observed

**USING TDD:**

Correct situation: success

```

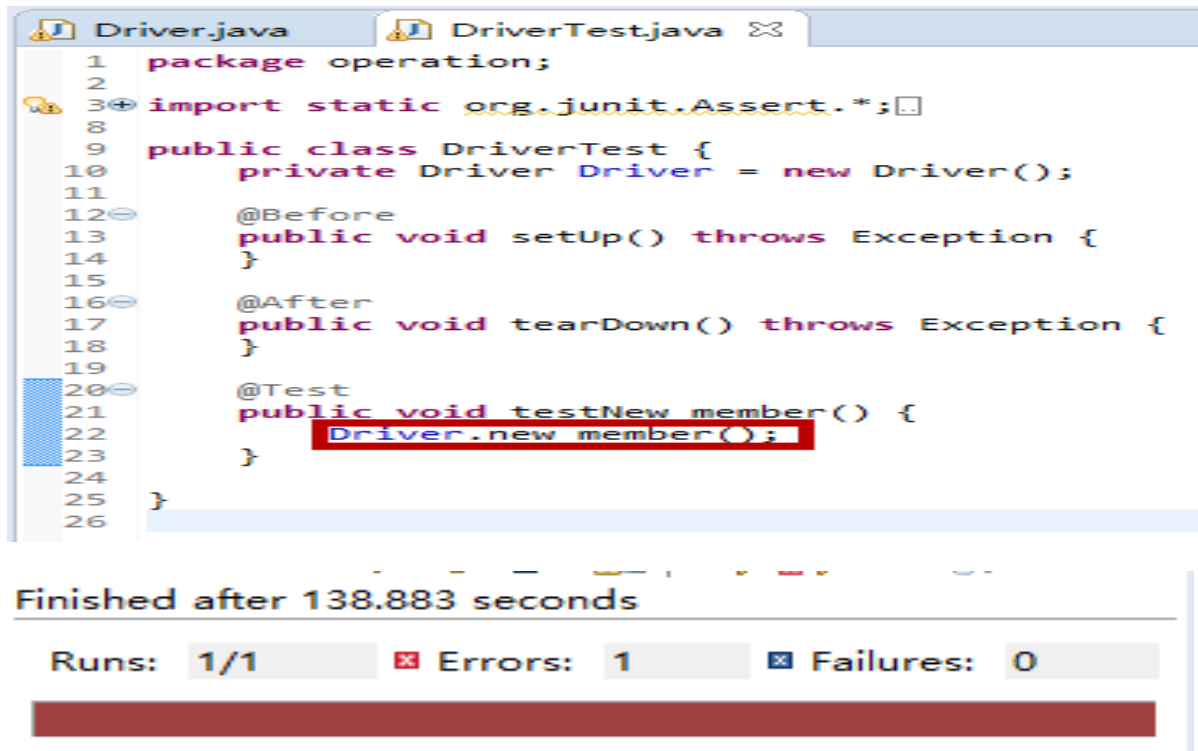
1 package operation;
2
3 import static org.junit.Assert.*;
4
5
6
7
8
9 public class DriverTest {
10     private Driver Driver = new Driver();
11
12     @Before
13     public void setUp() throws Exception {
14     }
15
16     @After
17     public void tearDown() throws Exception {
18     }
19
20     @Test
21     public void testNew member() {
22         Driver.new member();
23     }
24
25 }
26

```

Finished after 7.98 seconds

Runs: 1/1    ✖ Errors: 0    ✖ Failures: 0

Incorrect situation: failure



```

1 package operation;
2
3 import static org.junit.Assert.*;
4
5 public class DriverTest {
6     private Driver Driver = new Driver();
7
8     @Before
9     public void setUp() throws Exception {
10    }
11
12    @After
13    public void tearDown() throws Exception {
14    }
15
16    @Test
17    public void testNew member() {
18        Driver.new member();
19    }
20 }

```

Finished after 138.883 seconds

Runs: 1/1    ✖ Errors: 1    ✖ Failures: 0

### 6.2.2 Test 2 for Add\_Route

(1) Test procedure for correct input:

- 1.From the main screen, choose the management system at first and then click the “Route” button. The route interface is displayed.
- 2.From the route interface, Click the “Add new route” button, then you can select the station your route going through. Next, click “Finish adding” button and input route ID.
- 3.Input “00” and select “确定”. The system successfully add this route and return “input success”.

(2) Test procedure for incorrect input:

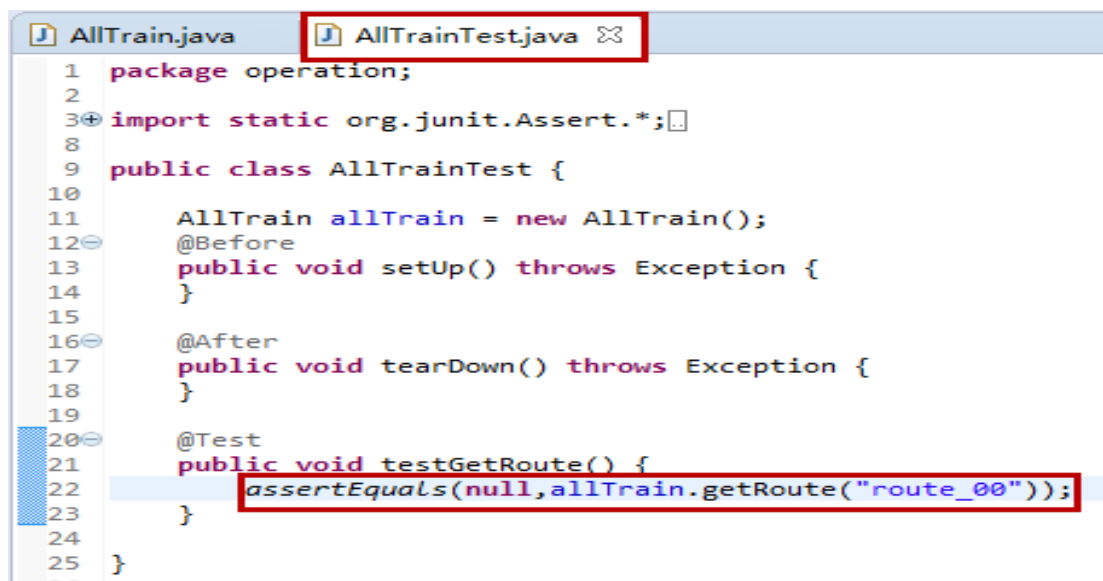
- 1.From the main screen, choose the management system at first and then click the “Route” button. The route interface is displayed.
- 2.From the route interface, Click the “Add new route” button, then you can select the station your route going through. Next, click “Finish adding” button and input route ID.
- 3.Input “01” (the system already has this route) and select “确定”. The system fail to add this route and return “this route already exist”.

**TEST MATRIX:**

Test Case: File Open	#Test Description	Test Cases	Pass/ Fail	No. of Bugs	Bug #	Comments
N/A	Create route in system [Route_01 exists in system]	setup	-	-	-	Added to system for testing purposes
2.1	Test that 00 does not exist in the system, so add successfully	2.1	P	0	0	Correct behavior observed
2.2	Test that 01 does exist in the system, so fail	2.2	F	0	0	Correct behavior observed

**USING TDD:**

Correct situation: success




```

1 package operation;
2
3 import static org.junit.Assert.*;
4
5 public class AllTrainTest {
6
7     AllTrain allTrain = new AllTrain();
8     @Before
9     public void setUp() throws Exception {
10
11     }
12     @After
13     public void tearDown() throws Exception {
14
15     }
16     @Test
17     public void testGetRoute() {
18         assertEquals(null, allTrain.getRoute("route_00"));
19     }
20 }

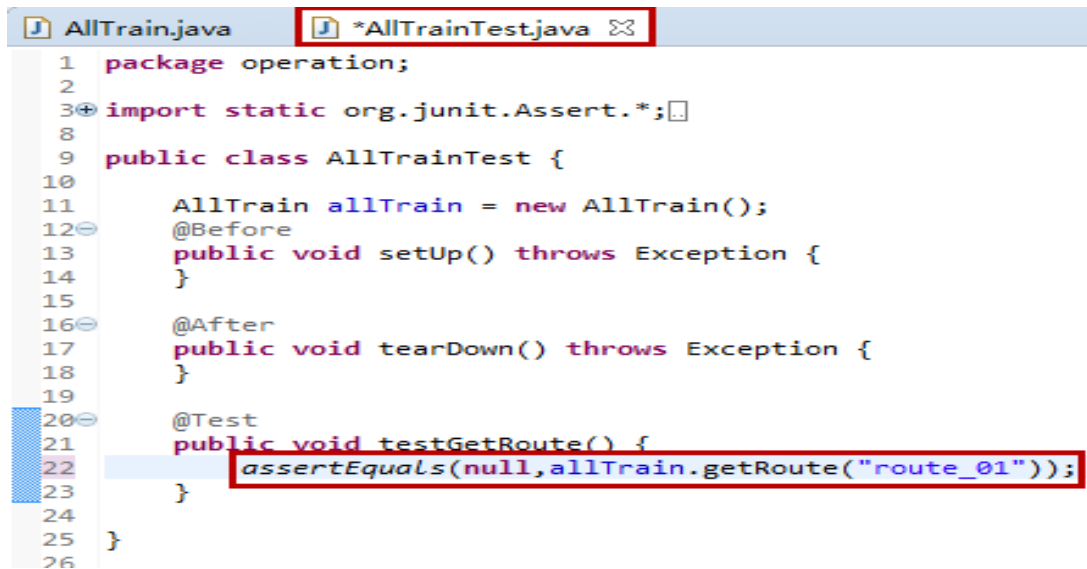
```

Finished after 0.028 seconds

Runs: 1/1    ✖ Errors: 0    ✖ Failures: 0

▶  operation.AllTrainTest [Runner: JUnit 4] (0.001 s)

Incorrect situation: failure



```

1 package operation;
2
3 import static org.junit.Assert.*;
4
5
6
7
8
9 public class AllTrainTest {
10
11     AllTrain allTrain = new AllTrain();
12     @Before
13     public void setUp() throws Exception {
14     }
15
16     @After
17     public void tearDown() throws Exception {
18     }
19
20     @Test
21     public void testGetRoute() {
22         assertEquals(null, allTrain.getRoute("route_01"));
23     }
24 }
25
26

```

Finished after 0.037 seconds

Runs: 1/1    ✖ Errors: 0    ✖ Failures: 1



operation.AllTrainTest [Runner: JUnit 4] (0.000 s)  
 testGetRoute (0.000 s)

### 6.2.3 Test 3 for Add\_Trip

(1) Test procedure for correct input:

- 1.From the main screen, choose the management system at first and then click the “Route” button. The route interface is displayed.
- 2.From the route interface, Click the “Add new Route” button, then you can select the station your route going through. Next, click “Finish adding” button and input route ID. Allocate time for each station. Finally, it display the trip number interface. Input a trip number.
- 3.Input “0001” and select “确定”. The system successfully add this trip and return “input success”.

(2) Test procedure for incorrect input:

- 1.From the main screen, choose the management system at first and then click the “Route” button. The route interface is displayed.
- 2.From the route interface, Click the “Add new route” button, then you can select the station your route going through. Next, click “Finish adding” button and input route ID. Allocate time for each station. Finally, it display the trip number interface. Input a trip number.
- 3.Input “0101” (the system already has this route) and select “确定”. The system fail to add this trip and return “this trip already exist”.

**TEST MATRIX:**

Test Case: File Open	#Test Description	Test Cases	Pass/ Fail	No. of Bugs	Bug #	Comments
<b>N/A</b>	Create route in system [Route_01 exists in system]	setup	-	-	-	Added to system for testing purposes
<b>3.1</b>	Test that 0001 does not exist in the system, so add successfully	3.1	P	0	0	Correct behavior observed
<b>3.2</b>	Test that 0101 does exist in the system, so fail	3.2	F	0	0	Correct behavior observed

**USING TDD:**

Correct situation: success

```

1 package operation;
2
3 import static org.junit.Assert.*;
4
5 public class AllTrainTest {
6
7     AllTrain allTrain = new AllTrain();
8     @Before
9     public void setUp() throws Exception {
10
11     }
12     @After
13     public void tearDown() throws Exception {
14
15     }
16     @Test
17     public void testNameExist() {
18         assertEquals(false, allTrain.nameExist("0001"));
19     }
20 }

```

Finished after 0.026 seconds

Runs: 1/1 Errors: 0 Failures: 0

operation.AllTrainTest [Runner: JUnit 4] (0.001 s)

Incorrect situation: failure

```

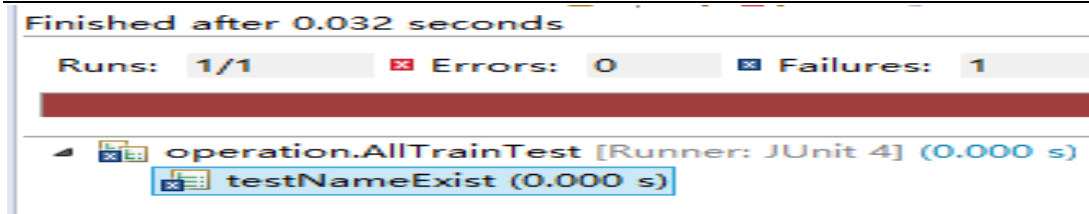
1 package operation;
2
3 import static org.junit.Assert.*;
4
5 public class AllTrainTest {
6
7     AllTrain allTrain = new AllTrain();
8     @Before
9     public void setUp() throws Exception {
10
11     }
12     @After
13     public void tearDown() throws Exception {
14
15     }
16     @Test
17     public void testNameExist() {
18         assertEquals(false, allTrain.nameExist("0101"));
19     }
20 }

```

Finished after 0.026 seconds

Runs: 1/1 Errors: 0 Failures: 0

operation.AllTrainTest [Runner: JUnit 4] (0.001 s)



## 6.3 System Tests

System testing is that functions should be tested in the whole process. Before the whole system test, we should improve the code and the whole program to ensure no defect in the program. In our System test, we use regression testing technique and Object-oriented testing technique. We run our program to check whether there exist any errors.

### 6.3.1 Test for Add\_Route process(System)

#### Test Case:

Input	Enter stations, time, journey, driver and train for the route
Result	The system shows a message that you add successfully
Conditions	No condition exists in test

#### Test Procedure

- 1.From the main screen, choose the management system at first and then click the “Route ” button. The Route interface is displayed.
- 2.Click the “Add new route” button, then you can select the station your route going through. Next, click “Finish adding” button and input route ID that you create. Allocate time for each station. Finally, after setting a trip number, select the driver and train you use in the route.
- 3.Add route successfully.

We carry out the system test and the system meets the integration quality goal in the test plan.

### 6.3.2 Test for Finding\_Route process(System)

#### Test Case:

Input	Enter the station for finding routes through the station
Result	The system shows the routes through the station
Conditions	No condition exists in test

#### Test Procedure

- 1.From the main screen, choose the customer system and then the customer interface is displayed.
- 2.Choose a station showed on the customer interface and click it.
- 3.The interface shows the routes through the station.

We carry out the system test and the system meets the integration quality goal in the test plan.

## 6.4 Test Evaluation

Through the testing process, we have detected all the defects and the bugs in two test cases. Finally all the defects have been finished. Then the whole build's requirements have been achieved. By testing with programming. All the functions are well worked. Actually some of the test cases were failed at the beginning. But after adjusting the code again, all the test cases are successfully passed. And all the functions we wanted achieved at the design process are achieved finally.

## 7. CONCLUSION

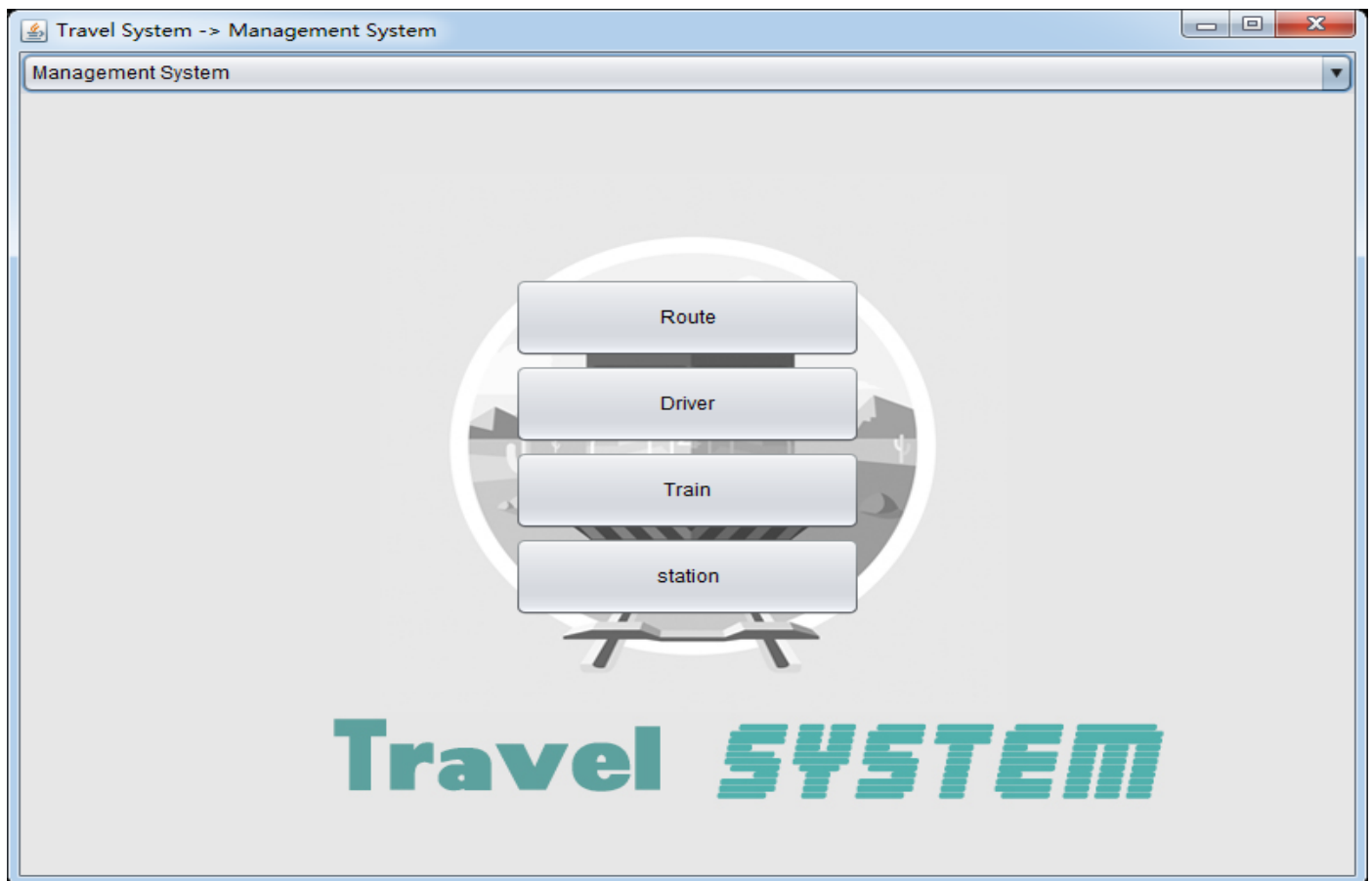
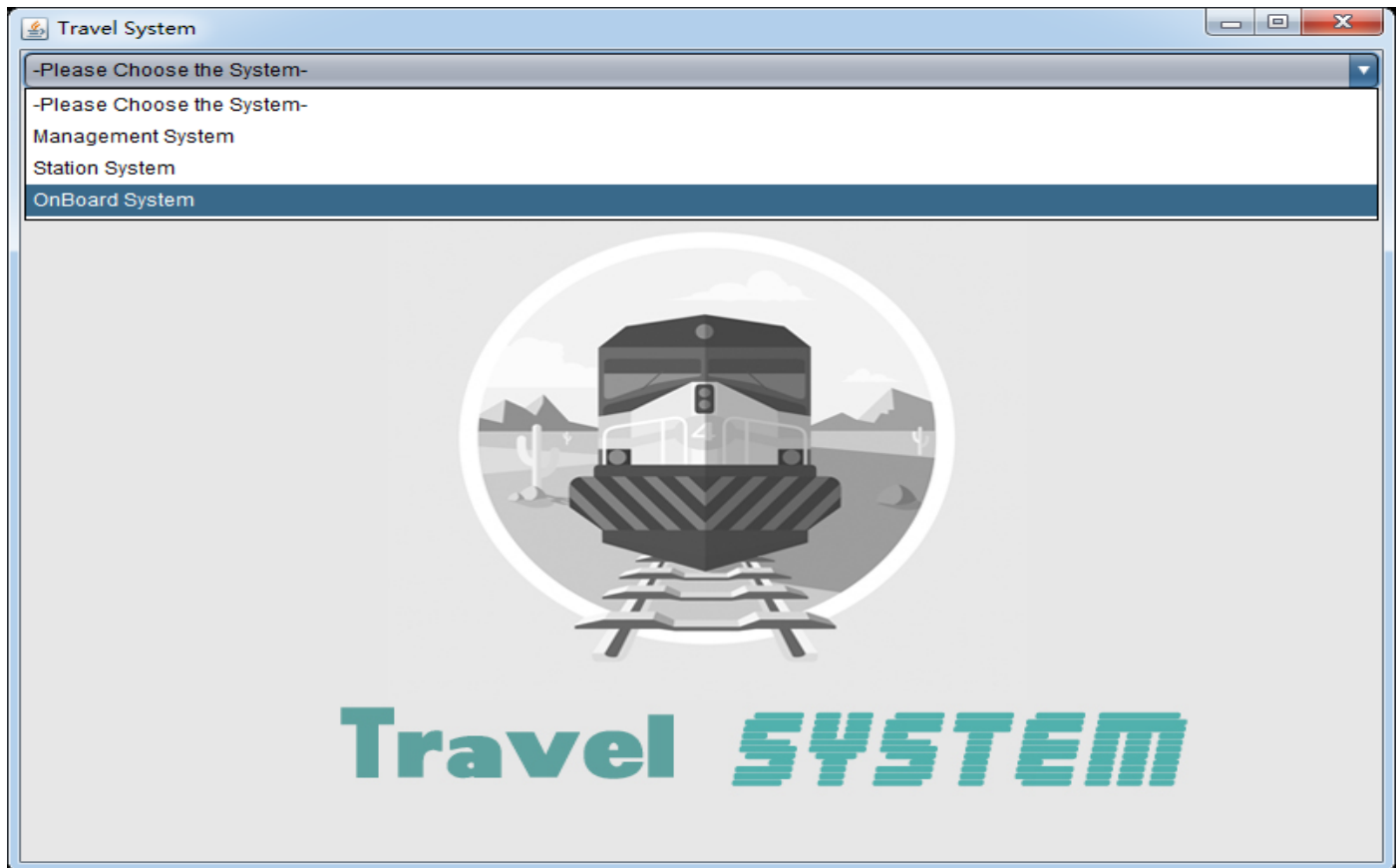
After two-month working, a train journey management system has been developed by our group. This system can satisfy the entire requirements which are stated on coursework handout with reasonable assumptions and has the ability to meet the future changing requirement with appropriate modification. Of course, there are still some defects in our outcome and some details still need to be improved in the future iterations.

In addition, it's the first time for us to use the theory acknowledge about agile software development on the actual developing process. This amazing developing method breaks the conventional developing methods with more flexible team working and more direct communication among developers and customers, which is becoming a popular software development method nowadays.

## 8. REFERENCE

- [1] Ian Sommerville, Software Engineering, 9th Edition, 2011
- [2] Brett McLaughlin, Gary Pollice, David West, Head First Object-Oriented Analysis and Design; A Brain-Friendly Guide to OOA&D, 2006
- [3] Kathy Sierra, Bert Bates, Head First Java, O'Reilly Media, Inc, 2005
- [4] Agile software development - Wikipedia, the free encyclopedia  
[http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development)
- [5] The product backlog: your ultimate to-do list  
<https://www.atlassian.com/agile/backlogs>

## 9. Appendix(ScreenShot)





Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

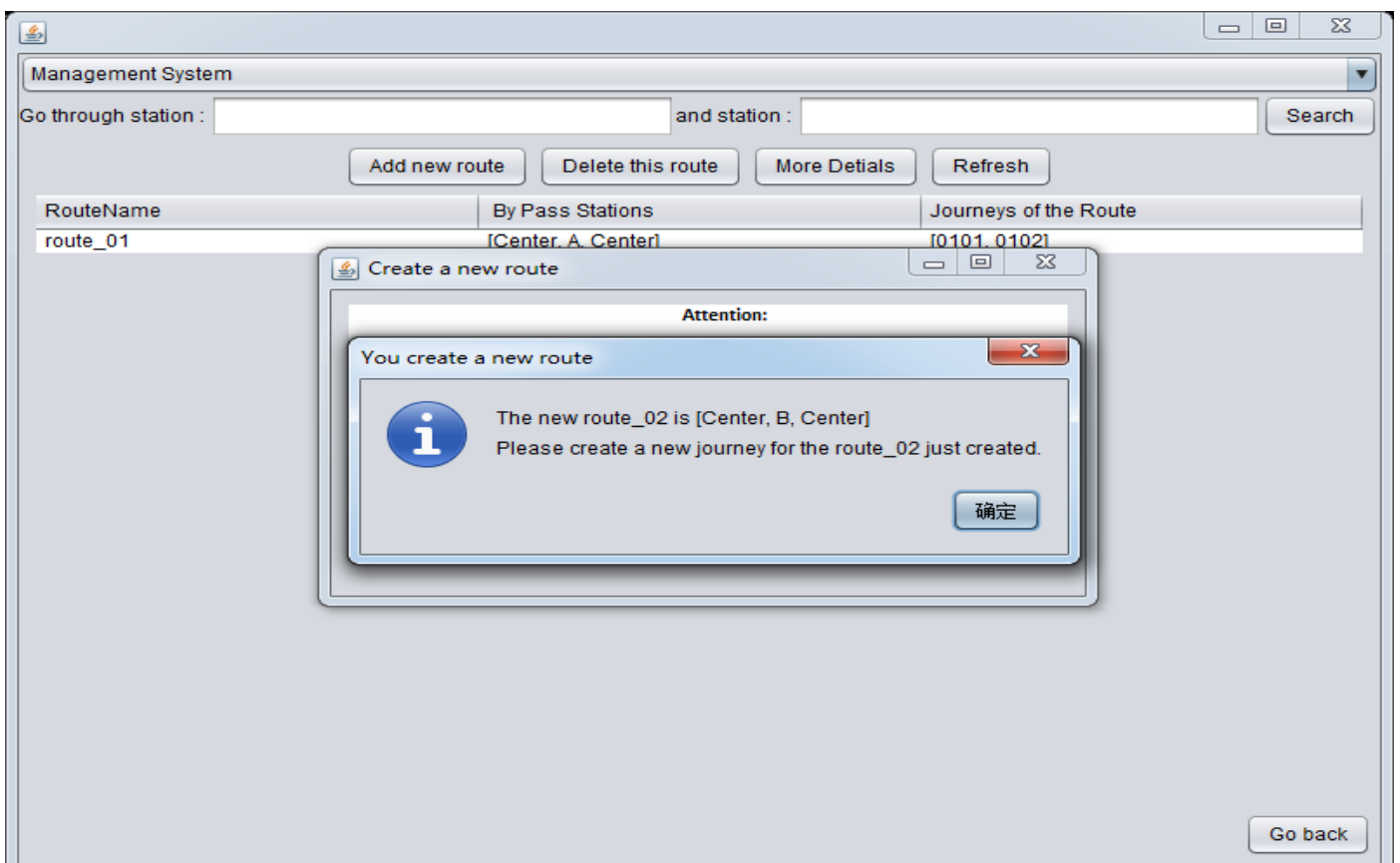
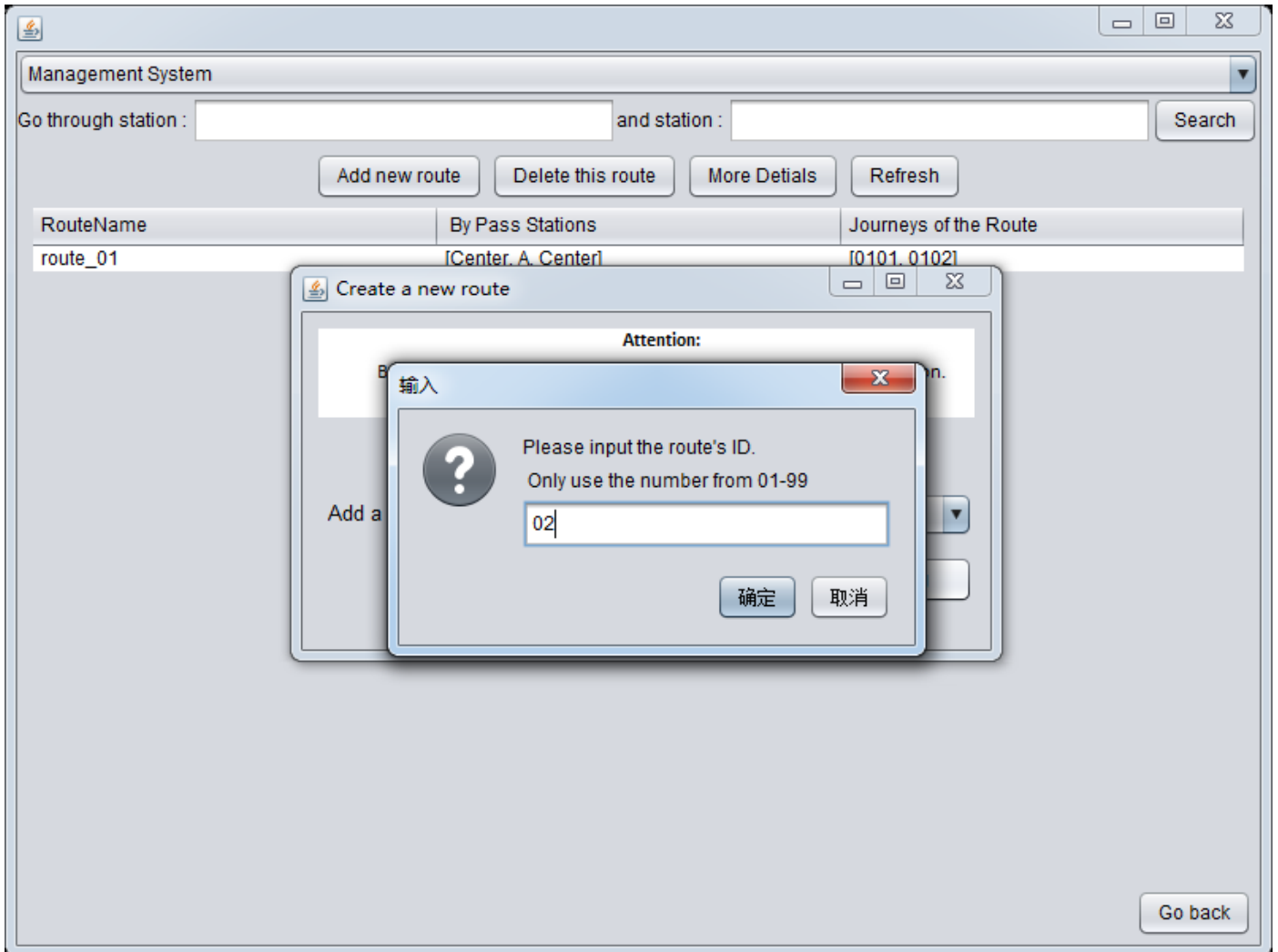
**Create a new route**

**Attention:**

Both the departure station and the terminus is the Center station.  
You only need to choose the intermediate stations.

Current Route: Center -> Center

Add a new station:



Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

Create a new new Journey

Create a Journey For the route\_02

Route:[Center, B, Center]

Input the arrive Time of the 1th Station - Center -

12 : 00

Management System


Go through station :  and station :

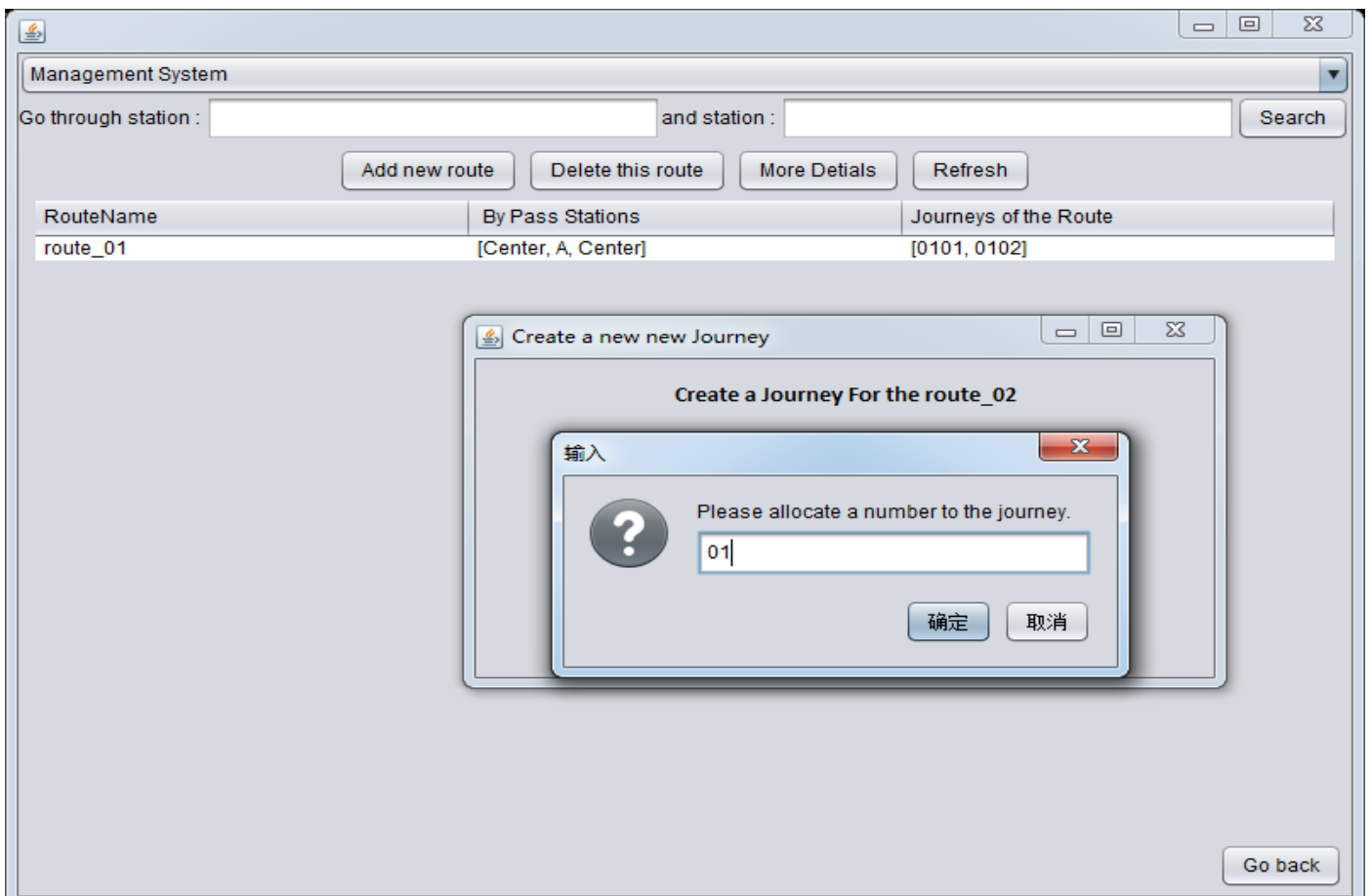
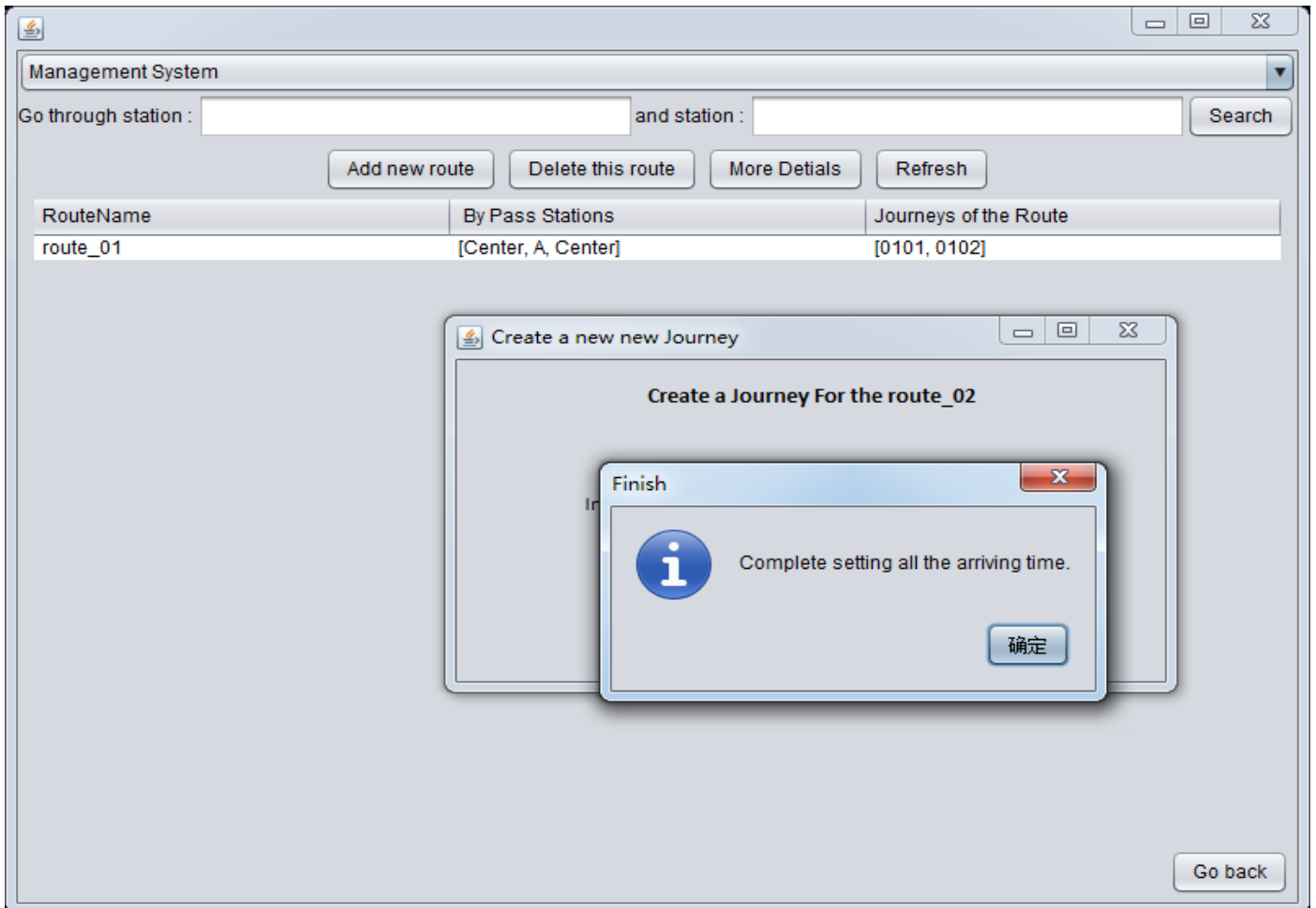
RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

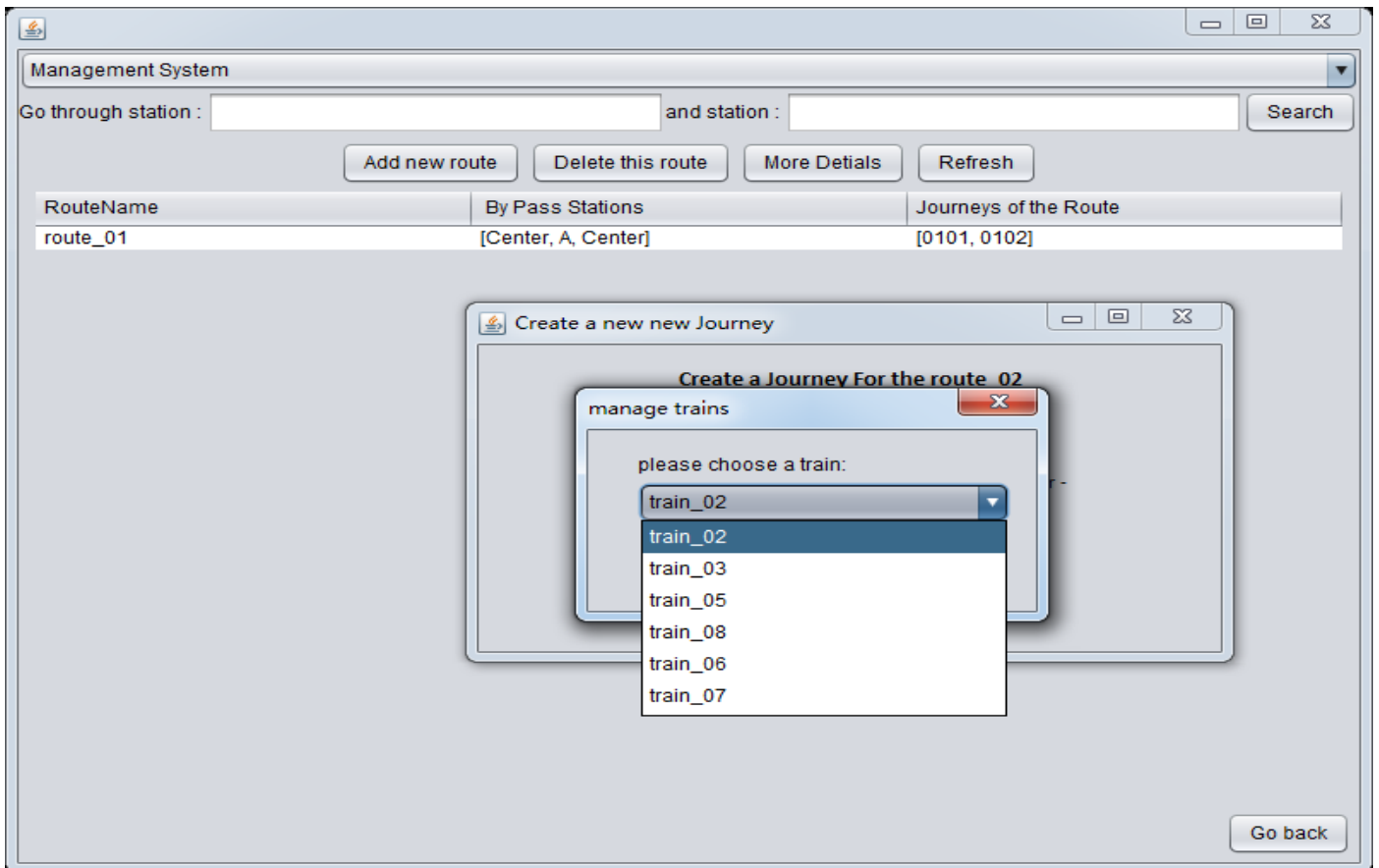
Create a new new Journey

Create a Journey For the route\_02

Successfully added!

 Arrival time 1200 for the 1th station Center is added successfully.





Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

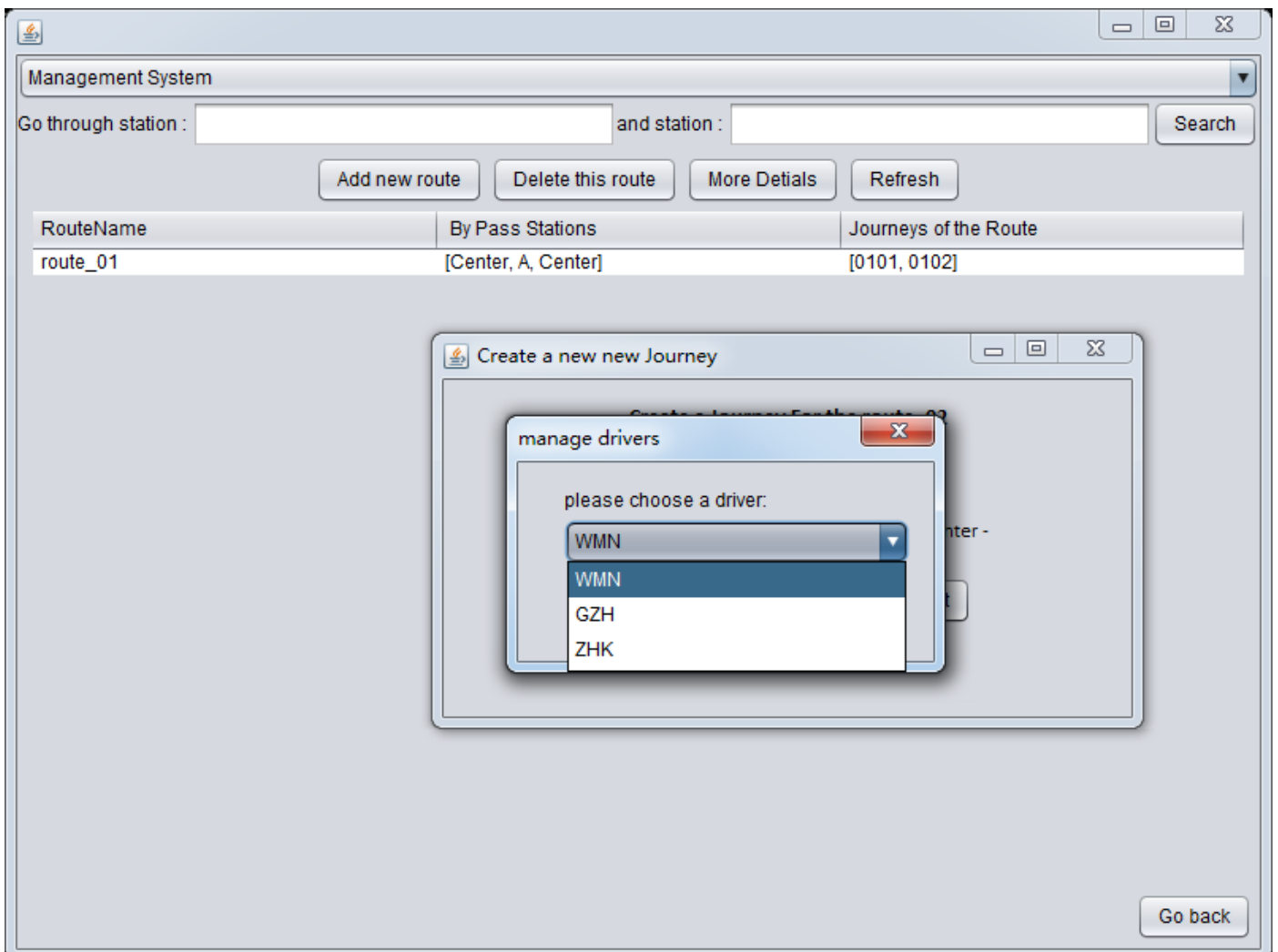
Create a new new Journey

Create a Journey For the route\_02

manage trains

please choose a train:

- train\_02
- train\_02
- train\_03
- train\_05
- train\_08
- train\_06
- train\_07



Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]

Create a new new Journey

Create a Journey For the route\_02

manage drivers

please choose a driver:

- WMN
- WMN
- GZH
- ZHK

Management System

Go through station :  and station :


RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]
route_02	[Center, B, Center]	[0201]

Management System

Go through station :  and station :

RouteName	By Pass Stations	Journeys of the Route
route_01	[Center, A, Center]	[0101, 0102]
route_02	[Center, B, Center]	[0201]

Successfulroute\_02

 Successfully Delete the route\_02

Management System

**The journeys information of the route 01**

JourneyID	Station Center	Station A	Station Center	Train	Driver
0101	01:10	12:10	23:10	train_01	RJF
0102	11:00	13:00	16:00	train_04	BZR

Management System

**The journeys information of the route 01**

JourneyID	Station Center	Station A	Station Center	Train	Driver
0101	01:10	12:10	23:10	train_01	RJF
0102	11:00	13:00	16:00	train_04	BZR

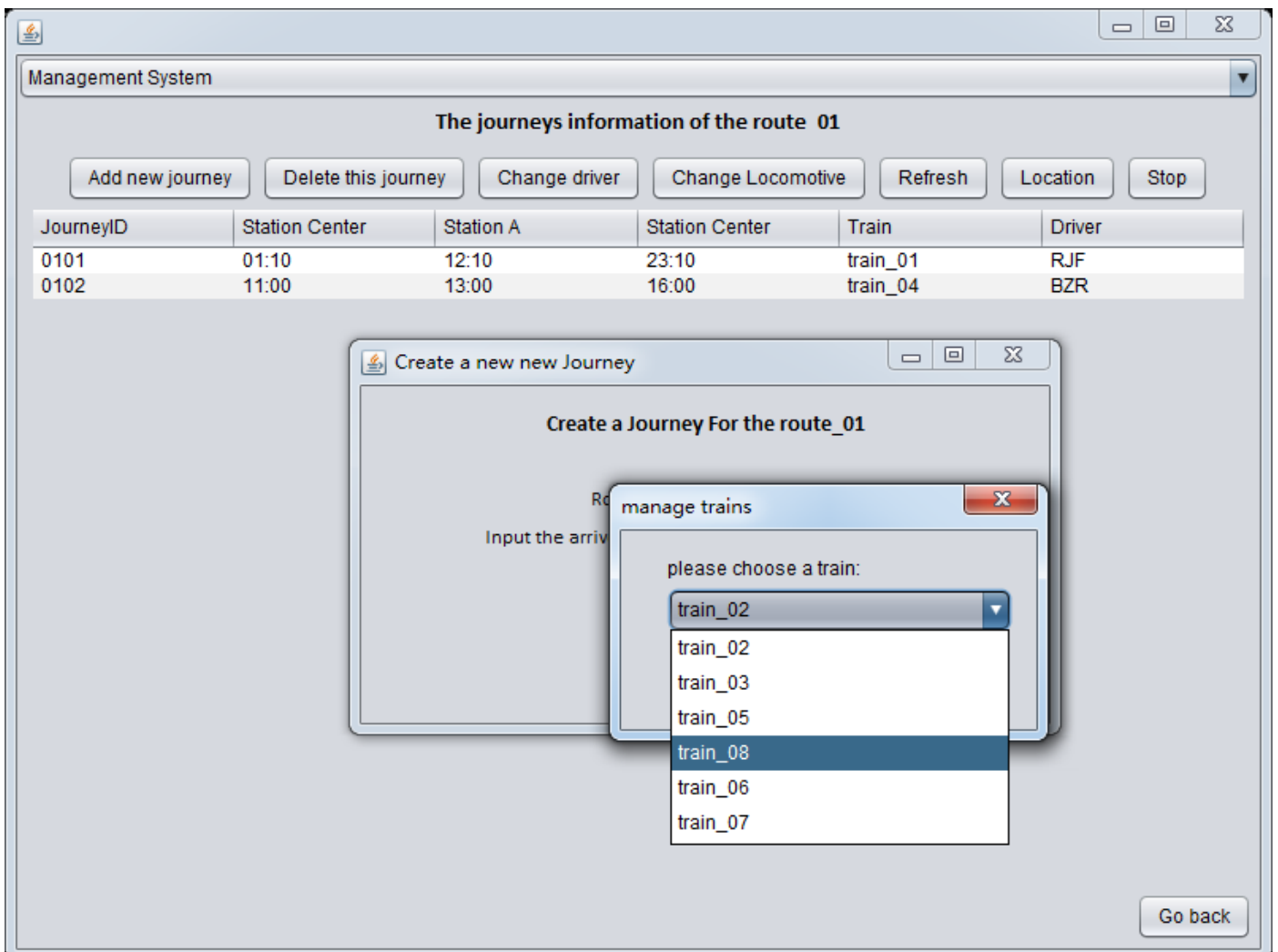
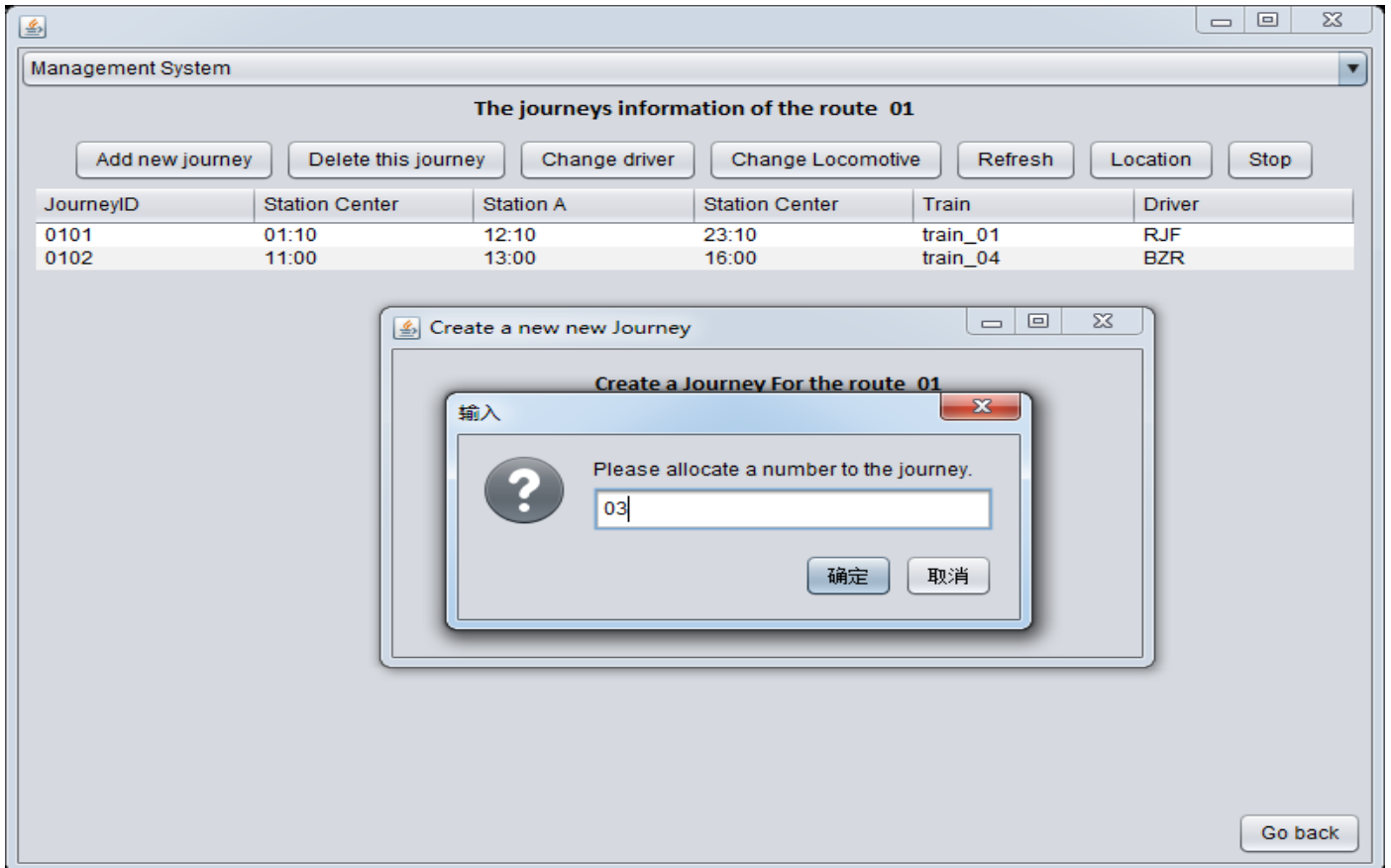
**Create a new new Journey**

**Create a Journey For the route\_01**

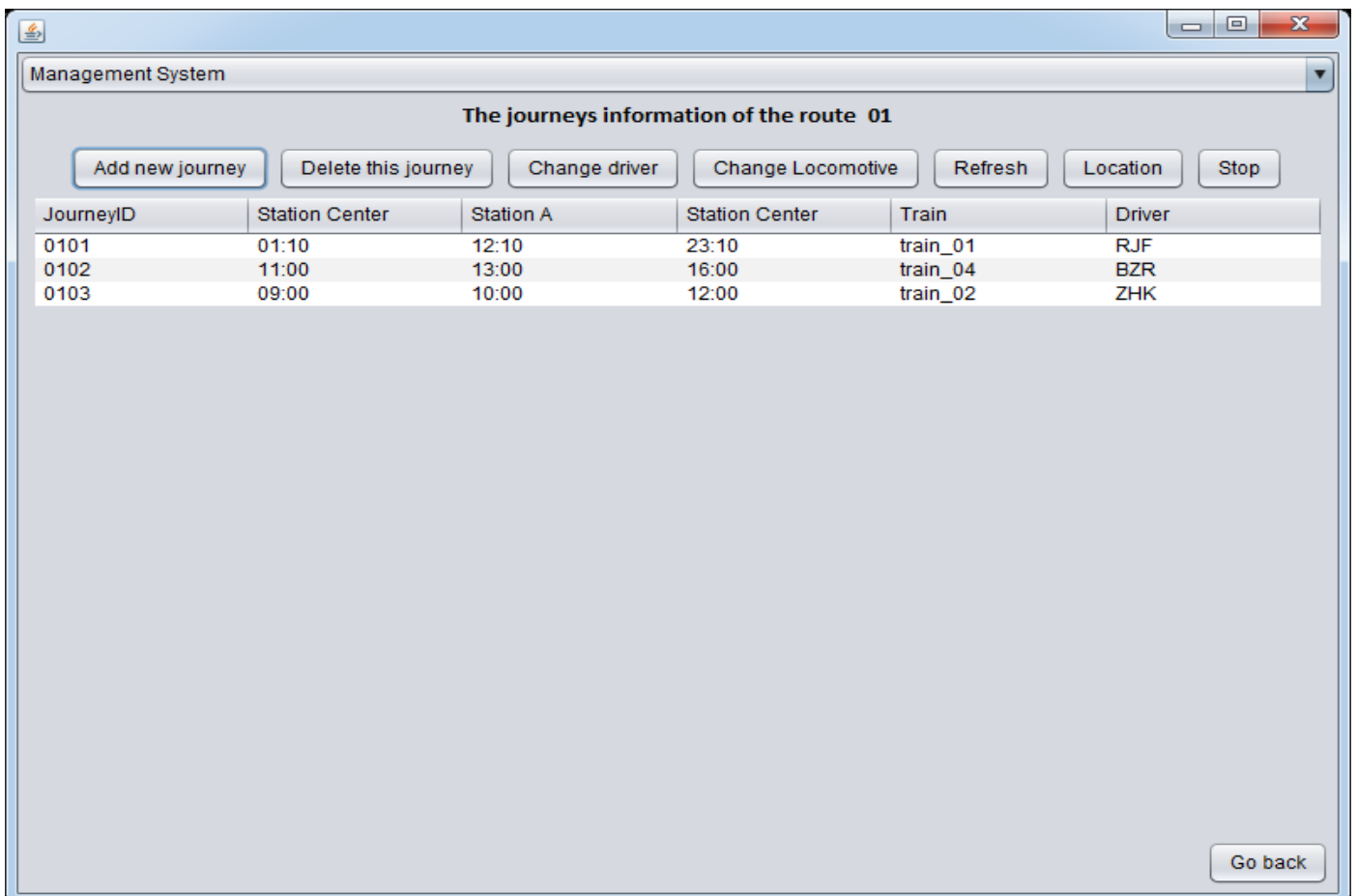
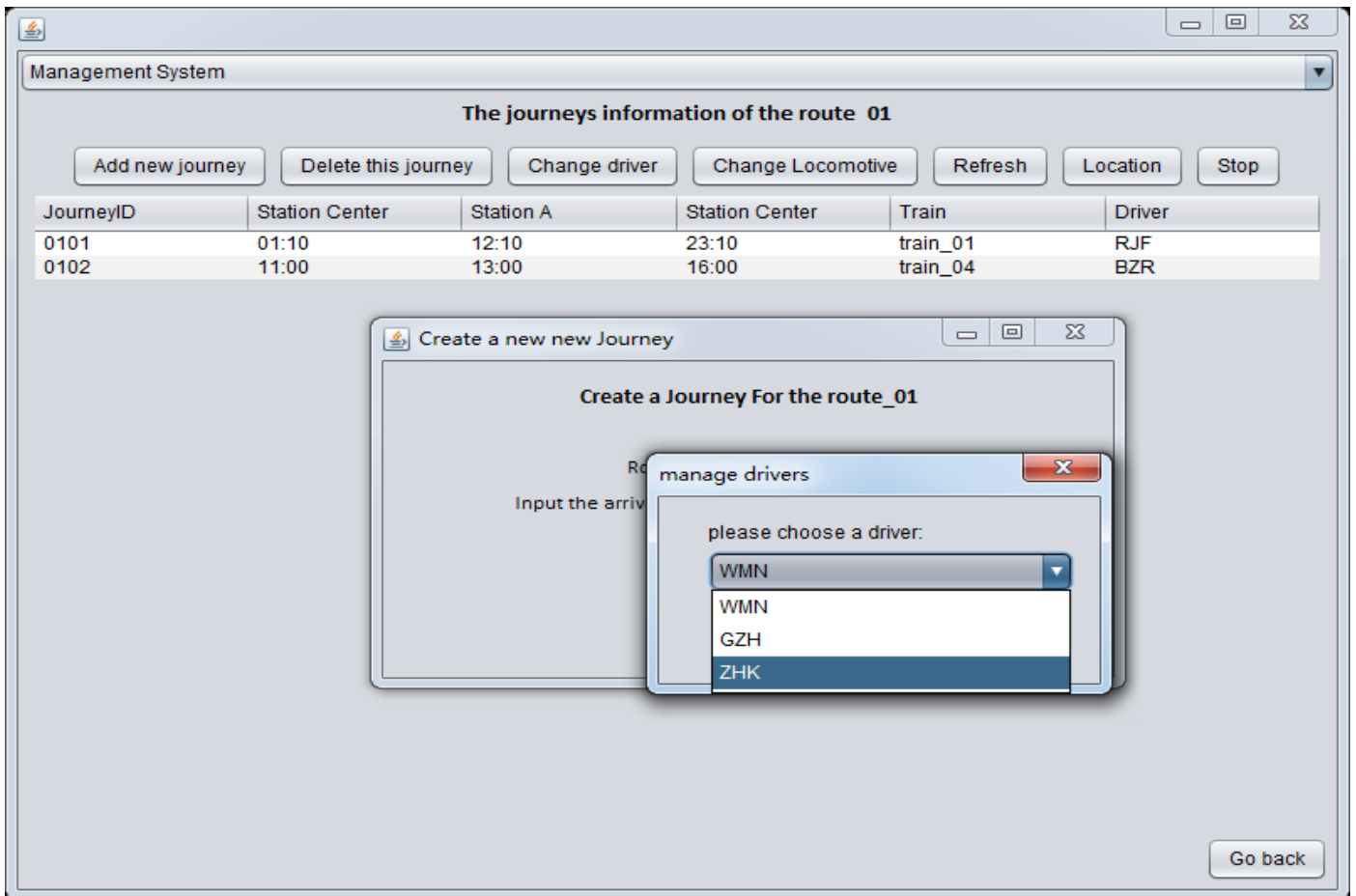
Route:[Center, A, Center]

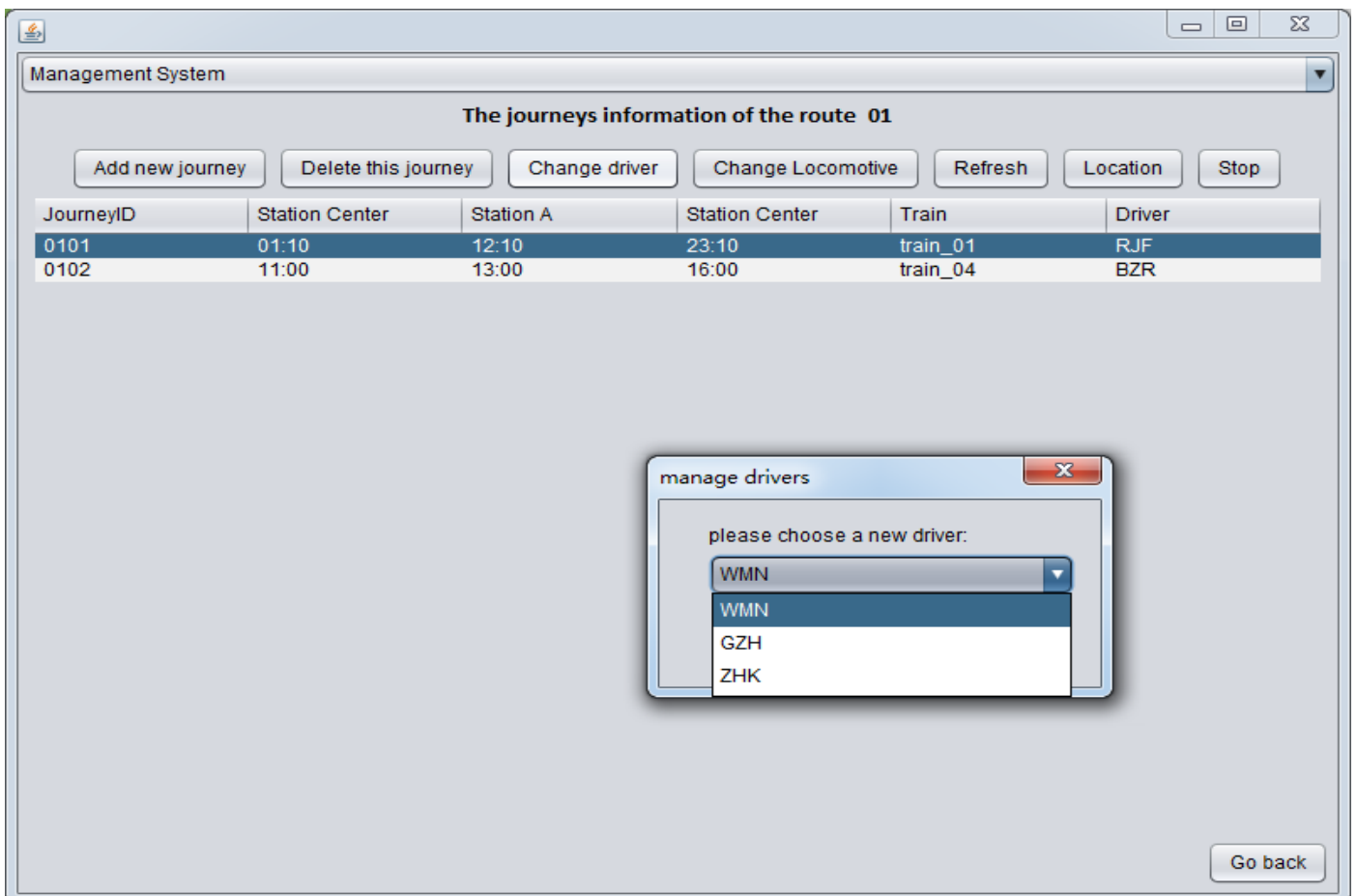
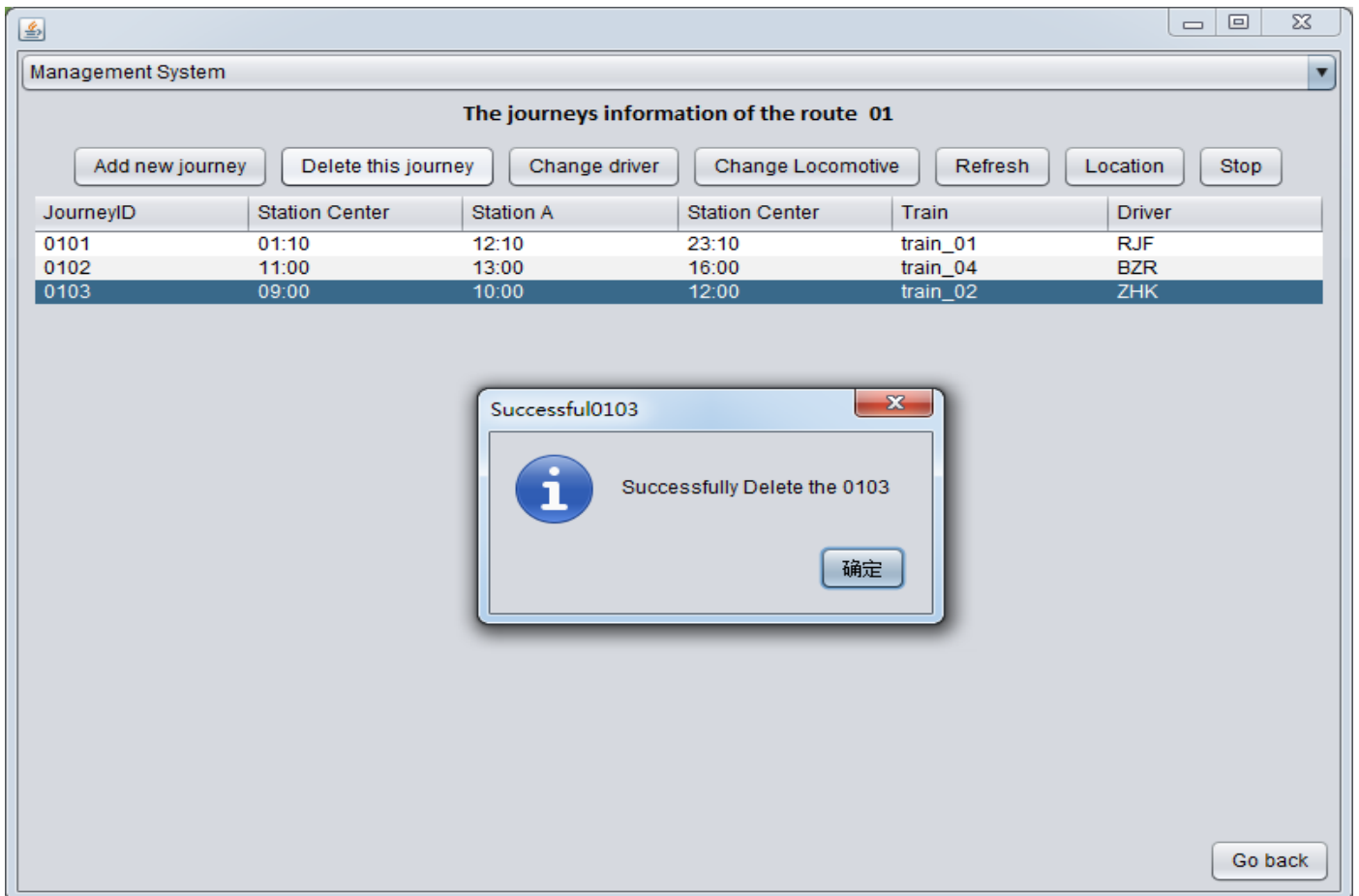
Input the arrive Time of the 1th Station - Center -

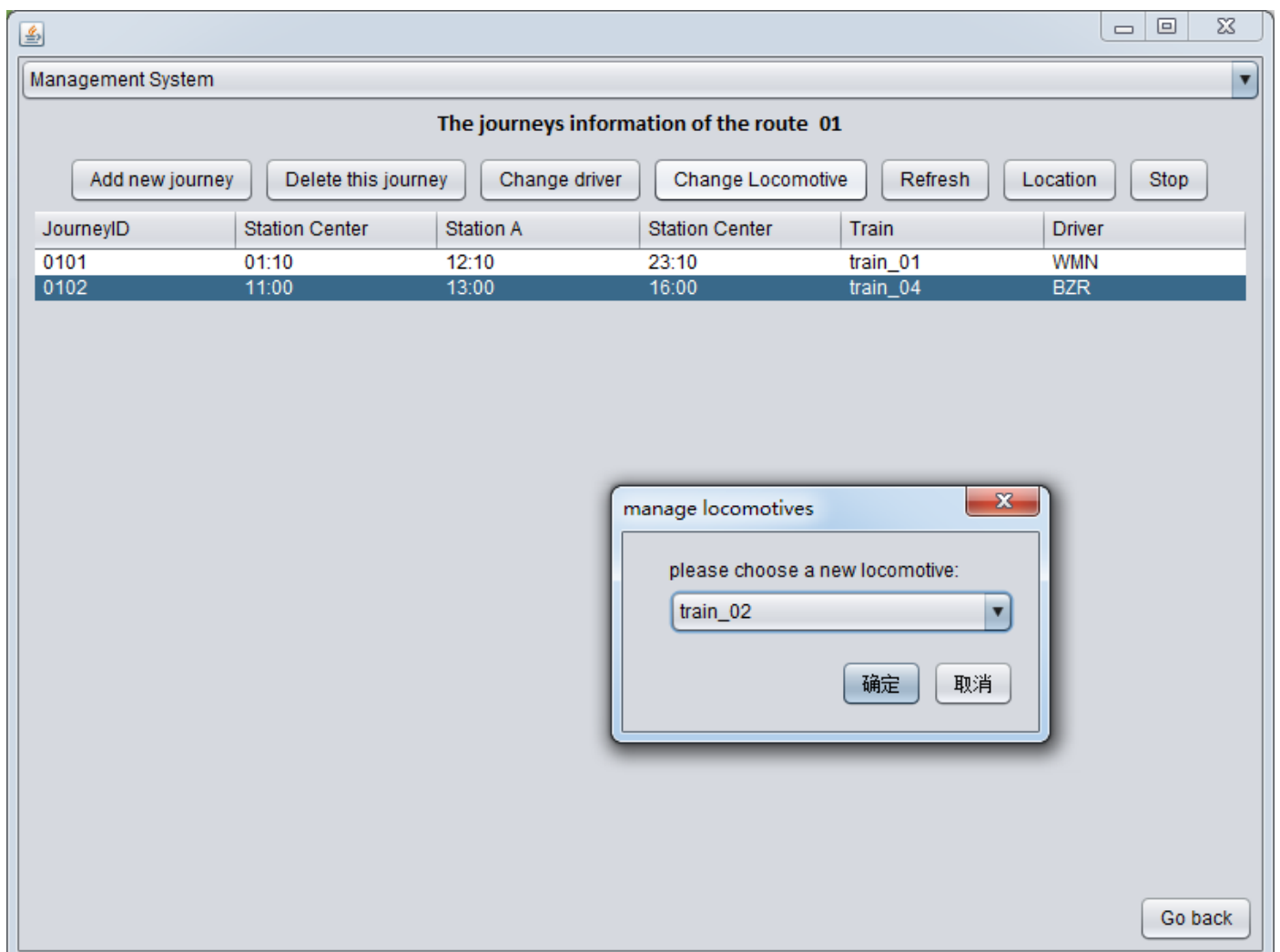
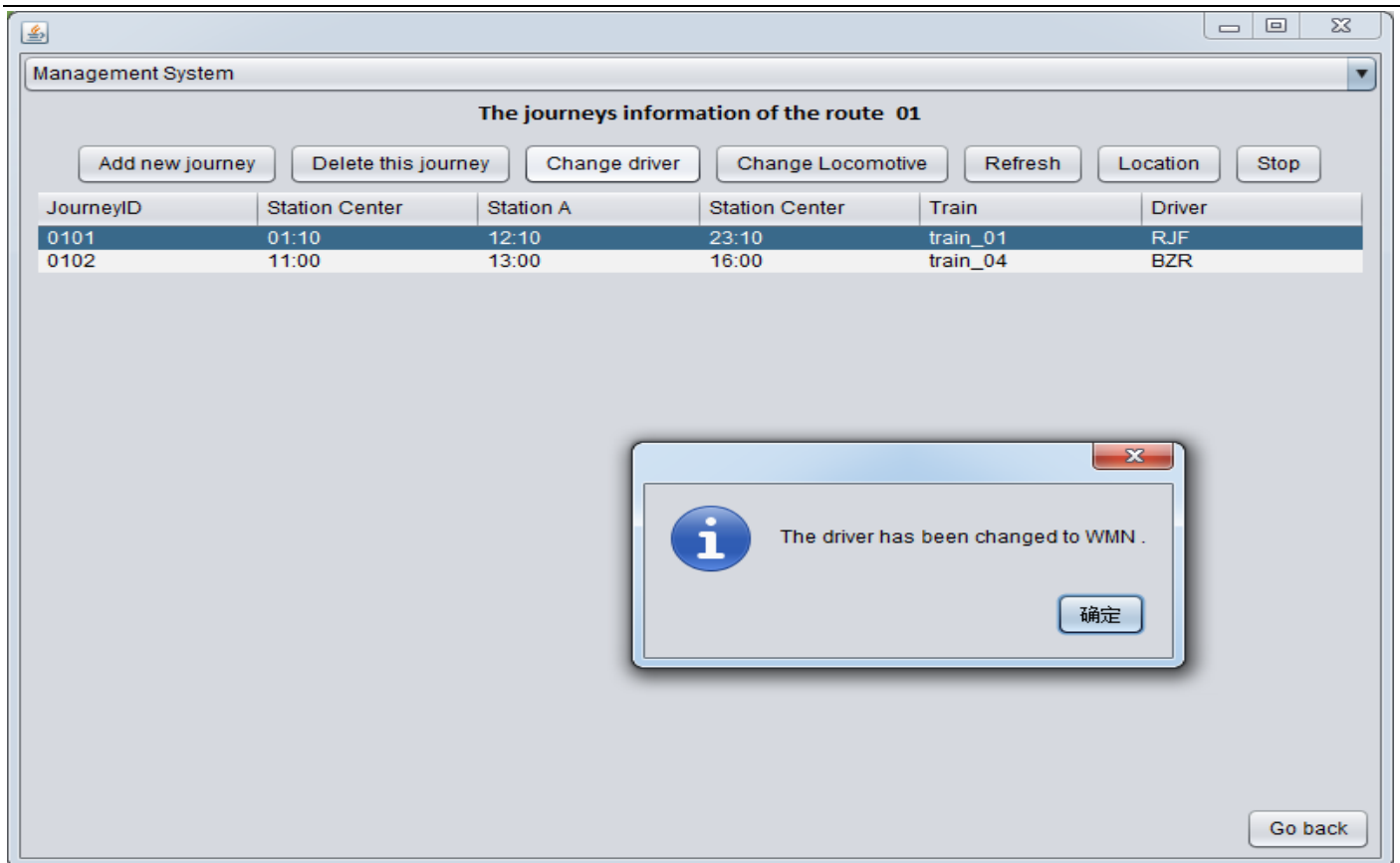
:











Management System

**The journeys information of the route 01**

JourneyID	Station Center	Station A	Station Center	Train	Driver
0101	01:10	12:10	23:10	train_01	WMN
0102	11:00	13:00	16:00	train_02	BZR

Location of 0101:  
Between Center and A. At a distance of 77%

77%

Management System

**The driver information**

Driver Name	Journeys of the Driver	Free Time
RJF	[]	From 0 to 2400 //
WMN	[0101]	From 0 to 110 // From 2310 to 2400 //
GZH	[]	From 0 to 2400 //
BZR	[0102]	From 0 to 1100 // From 1600 to 2400 //
ZHK	[]	From 0 to 2400 //

Management System

**The driver information**

Driver Name	Journeys of the Driver	Free Time
RJF	□	From 0 to 2400 //
WMN	[0101]	From 0 to 110 // From 2310 to 2400 //
GZH	□	From 0 to 2400 //
BZR	[0102]	From 0 to 1100 // From 1600 to 2400 //
ZHK	□	From 0 to 2400 //

**Add new driver**

Please input a new driver name

Management System

**The driver information**

Driver Name	Journeys of the Driver	Free Time
RJF	□	From 0 to 2400 //
WMN	[0101]	From 0 to 110 // From 2310 to 2400 //
GZH	□	From 0 to 2400 //
BZR	[0102]	From 0 to 1100 // From 1600 to 2400 //
ZHK	□	From 0 to 2400 //
JDK	□	From 0 to 2400 //

Management System

### The driver information

Driver Name	Journeys of the Driver	Free Time
RJF	□	From 0 to 2400 //
WMN	[0101]	From 0 to 110 // From 2310 to 2400 //
GZH	□	From 0 to 2400 //
BZR	[0102]	From 0 to 1100 // From 1600 to 2400 //
ZHK	□	From 0 to 2400 //
JDK	□	From 0 to 2400 //

confirm

?
Do you want delete this driver

是(Y)
否(N)

Management System

### The locomotive information

Locomotive Name	Journeys of the Locomotive	Free Time
train_01	[0101]	From 0 to 110 // From 2310 to 2400 //
train_02	[0102]	From 0 to 1100 // From 1600 to 2400 //
train_04	□	From 0 to 2400 //
train_03	□	From 0 to 2400 //
train_05	□	From 0 to 2400 //
train_08	□	From 0 to 2400 //
train_06	□	From 0 to 2400 //
train_07	□	From 0 to 2400 //

Management System

**The station information**

Station Name	Journeys go through the station	Other Information
A	[0101, 0102]	other information
B	[]	other information
C	[]	other information
D	[]	other information
E	[]	other information
F	[]	other information
G	[]	other information
H	[]	other information
I	[]	other information
J	[]	other information
K	[]	other information
M	[]	other information

Travel System -> Station System

Station System

**Choose a station**

A

B

C

D

E

F

G

H

I

J

K

M

