ICT Engineering

SEP Y1 A16

Project Report

Submitted to:

Birgitte von Fyren Balsløv

Steffen Vissing Andersen

Prepared by:

Cristian Cuibaru 253719

Denis Durovic 253794

Krystof Spiller 253812

Ovidiu Muresan 254119

Submitted: 16 December 2016

Abstract

VIA Bus is a company located in Horsens. The manager required a software system for the company, which must meet the requirements of the customer. Program must be able to process and store data about reservations, customers, tours, buses and chauffeurs. The data should be stored in a binary file.

In this report is outlined the process of creating the project from an early stage of analysis, through design, to the implementation and testing. Results of the project are stated in its own chapter, the project is summarized in the conclusion and references are provided in the end of the report.

In the analysis, the requirements of the customer were specified and individual use cases for the program were devised and further elaborated. In the design, the structure of one of the main parts of the program, the model, was designed in form of a UML class diagram and the GUI mockups were drawn. Implementation is about making the program based on the findings from analysi and design, therefore in this section a few examples of the code are shown and their function is described.

Software which the team developed specifically for this customer matches most of the requirements in the model. However, it can’t be used by the customer, because of the missing connection between GUI and the model.

Contents

Table of figures

1. Introduction

VIA Bus is a company located in Horsens, Denmark with Trip Driver as the manager. VIA Bus offers one-day trips and travels for longer period of time to predefined destinations, which include mainly cities and sites in Denmark, but also a few European countries and a service called bus-and-chauffeur, where a customer rents a bus with a chauffeur and chooses destinations of their travel.

VIA Bus is in need of a system to keep track of the tours, chauffeurs, buses, customers and their reservations. Manager from VIA Bus contacted us to create such system. We arranged an interview and talked to the manager about company's requirements.

The project focus is to make a single user program that allows employee to manage company's data. The employee will be responsible for entering data which the program stores and processes, and he will be able to access the data, edit them, delete them and search for them.

The program should habe been developed in Java and include Graphical User Interface through which the program should be controlled.

In next chapters of this report is outlined the process of making this project. The whole process began with analysis, where the requirements of the customer has been specified and diagrams specifying the individual use cases were made. In the design section, the structure of the program is shown in an UML class diagram and in the implementation section, a few examples of the actual code is shown and described.

2. System Analysis

2.1. System Requirements

1. The program should be a standard looking program with a search field, menu and other buttons and tabs.
2. An employee should be able to use the program without a login and a password.
3. An employee should be able to see a list of one of the three types of the tours (one-day trips, travels and bus-and-chauffeur).
4. An employee should be able to make a reservation of one of the tours for the customer.
5. An employee making a reservation for travels or one-day trip should always enter customer name, address, phone number, birthday and optionally company name and its CVR number and email address for a newsletter and names of all the passengers. If the food is available on the tour then optionally an employee should be able to select if a customer doesn't want it or if he wants a vegetarian or vegan meal.
6. After making a reservation for travels or one-day trip, the program should store customer information and passenger names.
7. An employee making a reservation for a bus-and-chauffeur should always enter a name of the person who makes a reservation, organization or company name, address, phone number and optionally an email adress, an interval of reservation including the approximate time for the departure and arrival back to bus terminal, the destination(s) including the approximate time of arrival to each destination, included services
   (breakfast or/and lunch in the bus, reserving a restaurant for the dinner and party bus including or excluding party guide), count of the passengers and optionally email address for a newsletter.
8. After making a reservation for a bus-and-chauffeur, the program should store customer, company or organization information including contact information.
9. An employee should be able to make a new one-day trip or travel.
10. An employee making a new one-day trip or travel should always enter an approximate route and schedule with destinations and times.
11. After making a new one-day trip, travel or reservation for a bus-and-chauffeur, an employee should be able to see a list of the available chauffeurs and assign one of them to a tour. (Swap 11 and 12)
12. After assigning a chaffeur to a tour an employee should be able to select a available bus.
13. An employee should be able to enter a price for the service.
14. An employee should be able to search for a tours, customers, reservations, buses and chauffeurs.
15. An employee should be able to edit, delete and search a reservation for a customer.
16. An employee should be able to edit, delete and search a tour.
17. A discount should be provided to a frequent customer by an employee and the system should suggest an offered discount.
18. An employee should be able to see a list of all the chauffeurs.
19. An employee should be able to see a profile including a timeline of a specific chauffeur.
20. An employee should be able to create a new chauffeur entry to the system.

21. An employee making a new chauffeur entry to the system should always enter chauffeur's name, address, 5-digit employee ID, phone number, email address, whether he or she is a full-time or occasional chauffeur and optionally wishes for trips.
22. An employee should be able to edit, delete and search a chauffeur .
23. An employee should be able to assign a time for a maintenance of a bus and for a personal break for a full-time chauffeur.
24. An employee should be able to find and see a customer information.
25. A system should be able to add and edit and a customer based on reservation change.
26. An employee should be able to see a list of all the buses.
27. An employee should be able to create a new bus entry to the system.
28. An employee making a new bus entry to the system should always enter manufacturing year, manufacturer, model, color, plate number, number of seats and the type of bus (normal or party).
29. An employee should be able to edit, delete and search a bus.

2.2. Use Case Diagram

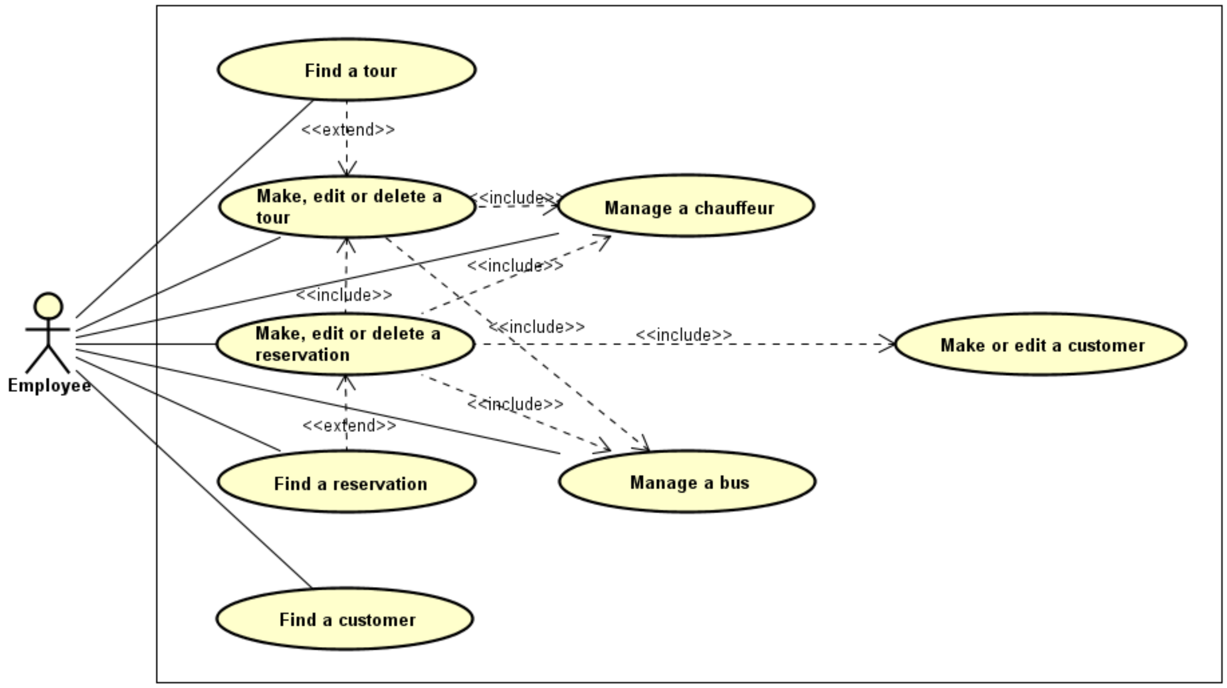


Figure 1 The use case diagram

Find a tour: Search for a tour.

Find a customer: Search for a customer.

Find a reservation: Search for a reservation.

Make, edit or delete a tour: Search, create, modify or delete a tour.

Make, edit or delete a reservation: Create, modify or delete a reservation.

Make or edit a customer: Create, modify or search a customer.

Manage a bus: Search, create, modify or delete a bus.

Manage a chauffeur: Search, create, modify or delete a chauffeur.

2.3. Use Case Description – Make, edit or delete a reservation

Below is an excerpt from this particular use case description regarding only making a reservation.

The full version and rest of the Use Case Descriptions can be viewed in project Astah file, which also contains full version of Use Case Diagram, Activity Diagrams, Class Diagram and Sequence Diagram.

Use Case	Make, edit or delete a reservation
Summary	An employee is able to make, edit or delete a reservation for a customer
Actor	Employee
Precondition	None
Post condition	A reservation with all the data has been created and saved in the system
Base Sequence	1. IF an employee is creating a new travel or one-day trip reservation THEN click the button to create a new reservation in the tour entry. Enter customer name, address, phone number, birthday and optionally company name and its CVR number, email address for a newsletter and names of all the passengers (Use case: Make or edit a customer). Select the seats based on the count of the passengers. IF the food is available on the tour then optionally specify how many customers don't want it and how many want a vegetarian meal. 2. IF an employee is creating a new bus-and-chauffeur reservation THEN click the button for creating a new bus-and-chauffeur reservation and enter a name of the person who makes a reservation, organization or company name, its CVR number, address, phone number and optionally an email address, an interval of reservation including the approximate time for the departure and arrival back to a bus terminal, the destination(s) including the approximate time of arrival to each destination, included services (breakfast or/and lunch in the bus, reserving a restaurant for the dinner, party bus including or excluding party guide) and count of the passengers. (Use case: Make or edit a customer) Depending on the input, system returns a list of available buses. Select a bus. (Use case: Manage a bus) System returns a list of available chauffeurs in the given date interval. Select a chauffeur (Use case: Manage a chauffeur) 3. After creating a new reservation IF CVR number for a company or customer name and birthday for a customer is found in the stored information THEN a system shows the total price and length of previous reservations, which the employee can use to assess how big a discount should be specified. Specify the discount. ELSE it is stored as a new customer (Use case: Make or edit a custome)
Exception Sequence	No available vehicles in the given date interval. 2 as base sequence No bus is chosen and a warning message that bus can not be chosen is shown. Go back to the same base sequence with the data already inputed except the date interval. No available chauffeurs in the given date interval. 2 as base sequence

	The returned list is empty so no chauffeur can not be chosen. Go back to the same base sequence with the data already inputed except the date interval. Not enough free seats. 1 and 2 as base sequence The reservation for that many passengers cannot be made. Back to the same base sequence with the data already inputed except the count of the passengers or the names of the passengers.
Note	A process of creating reservation can be cancelled any time.

2.4. Activity Diagram – Make a reservation

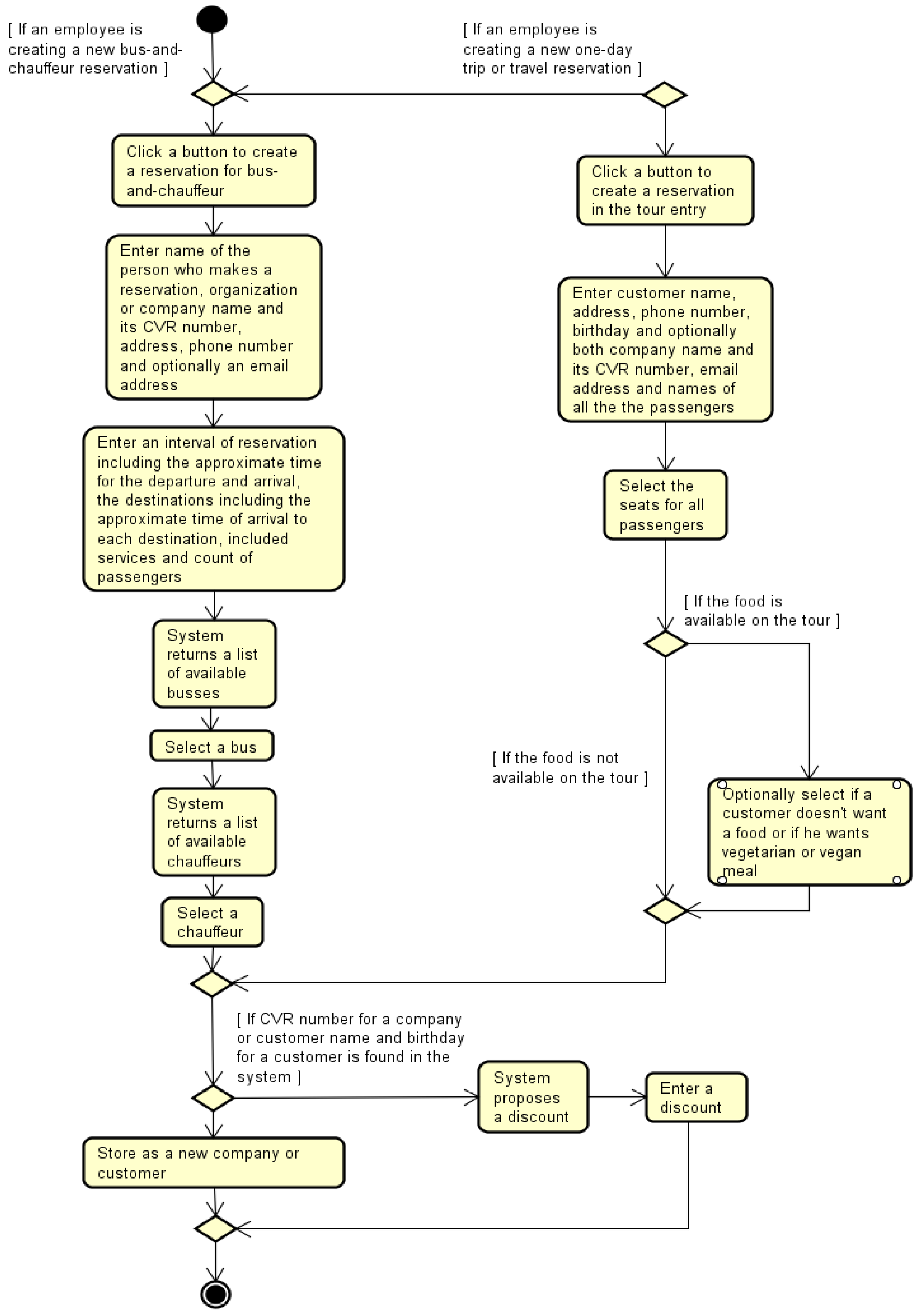

[If an employee is creating a new bus-and-chauffeur reservation]
[If an employee is creating a new one-day trip or travel reservation]
Click a button to create a reservation for bus-and-chauffeur
Enter name of the person who makes a reservation, organization or company name and its CVR number, address, phone number and optionally an email address
Enter an interval of reservation including the approximate time for the departure and arrival, the destinations including the approximate time of arrival to each destination, included services and count of passengers
System returns a list of available busses
Select a bus
System returns a list of available chauffeurs
Select a chauffeur
Click a button to create a reservation in the tour entry
Enter customer name, address, phone number, birthday and optionally both company name and its CVR number, email address and names of all the the passengers
Select the seats for all passengers
[If the food is available on the tour]
[If the food is not available on the tour]
Optionally select if a customer doesn't want a food or if he wants vegetarian or vegan meal
[If CVR number for a company or customer name and birthday for a customer is found in the system]
System proposes a discount
Enter a discount
Store as a new company or customer

Figure 2 The activity diagram

Making a reservation is the most important use case that the system contains, in order to create a reservation you need four objects: Customer, DateTime, Bus and Chauffeu. If the customer information is already present in the system, providing a discount for this customer will be advised with an aid of numbers representing how many hours did he already spend using VIA Bus services and how much money did he spend. This information might help customer to decide if and how big of a discount should be provided. If the customer information is not in the system yet, new customer is created in the system.

3. Design

3.1. UML class diagram

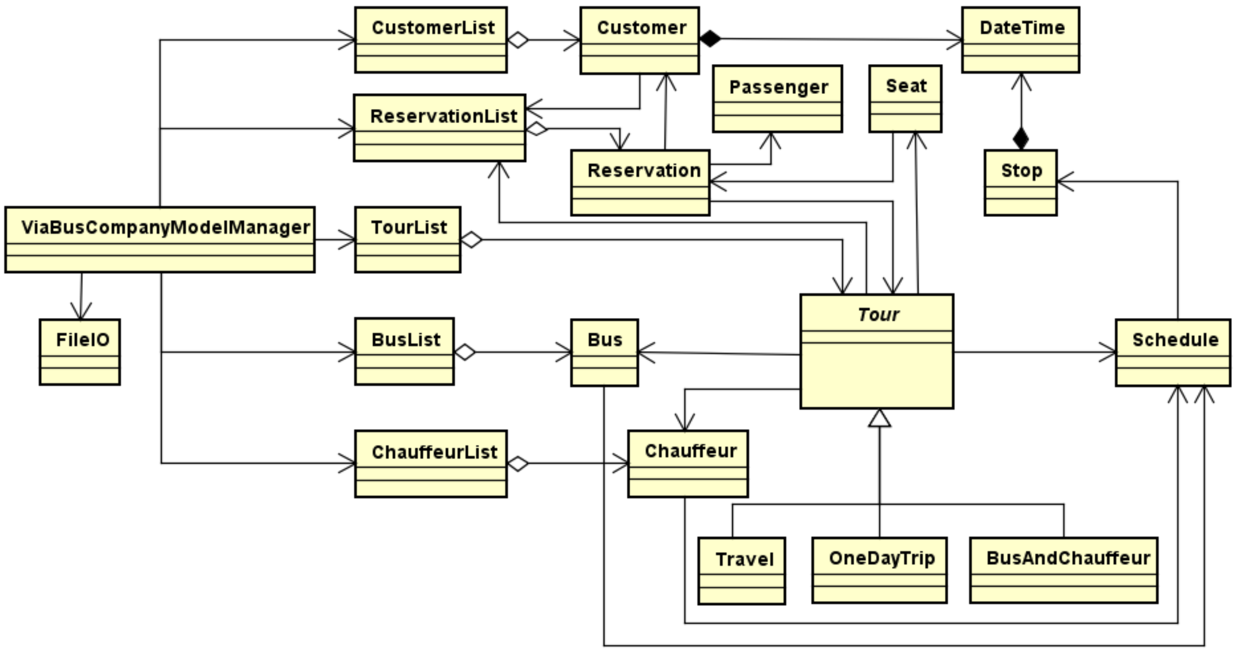


Figure 3 The class diagram

ViaBusCompanyModelManager represents a main model class, that has variables for all list types and also an variable for working with file.

Each list type contains its respective objects. It is worth noting that each customer and tour has their own reservation list, in order to make program functionality more convenient and efficient.

Each bus and chauffeur also has their own schedules, which are assigned to them based on the tours which they have been assigned to. Again, similarly as with reservation lists above, this is not necessary for correct functionality and same functionality could be achieved without this association. This was done for the sake of program efficiency and programming convenience. The problem that might arise if not handled carefully, is inconsistency of the different lists.

Beside its own schedule, chauffeur, bus and as it has been already mentioned, tour also contains objects of individual seats, which store seat number, whether it's occupied or not and for which reservation is this seat reserved. The seats are created for the tour based on which bus was chosen.

3.2. Sequence diagram

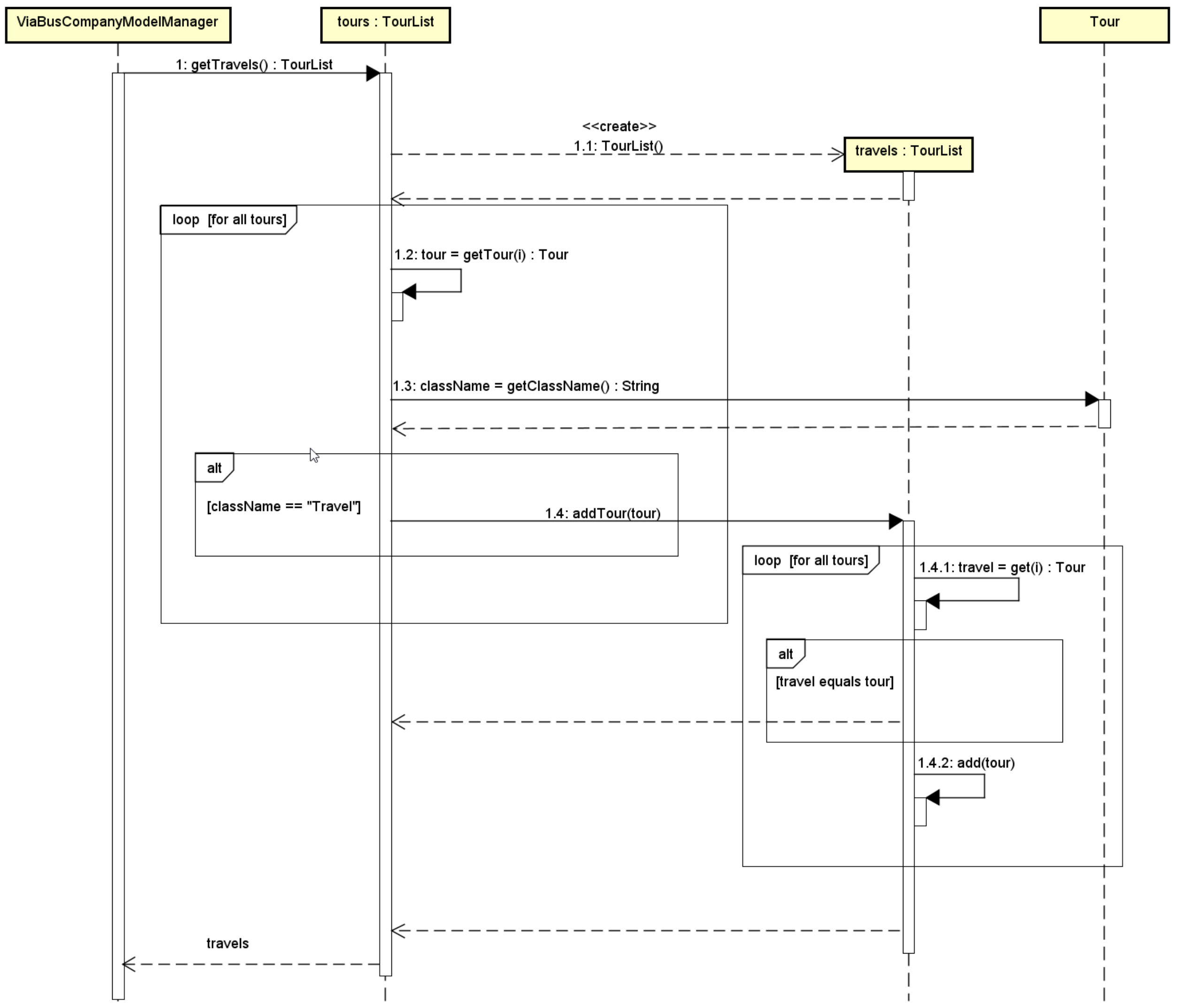
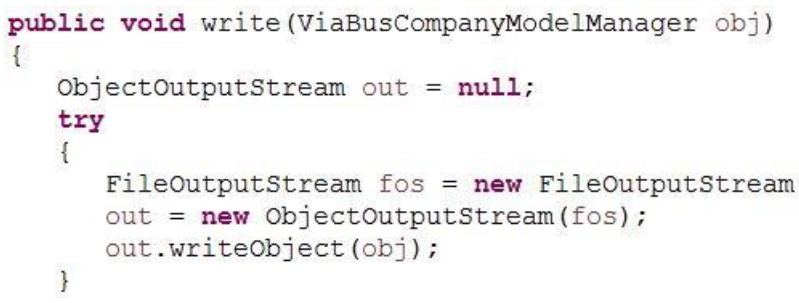


Figure 4 The sequence diagram

The sequence diagram shows step by step process of the method getTravels. First step that the system does is to loop through all the tours that it contains and compare the class names. If tour's class name is equal to “Travel”, then method addTour for the temporary TourList with the tour as an argument is called. This method loops through its own tours, which are all travels in this case, and if the travel is already in the list, it will return and have no effect. Otherwise it adds this particularl travel to the travels TourList. This second loop prevents duplication of the travels in this TourList.

This functionality, which prevents duplication, makes more sense when searching for not only tours. Search method works in a way, that other more specific methods are called with the inputed search phrase as an argument and return lists which are merged. And for example, if the search phrase is “sen” and we have a reservation, which has one of the destinations Horsens and the customer name is Jensen, then this reservation is going to be returned by both of these specific search methods and therefore duplication might occur. This is why it’s important to use this method to prevent duplication when adding tour to a tour list.

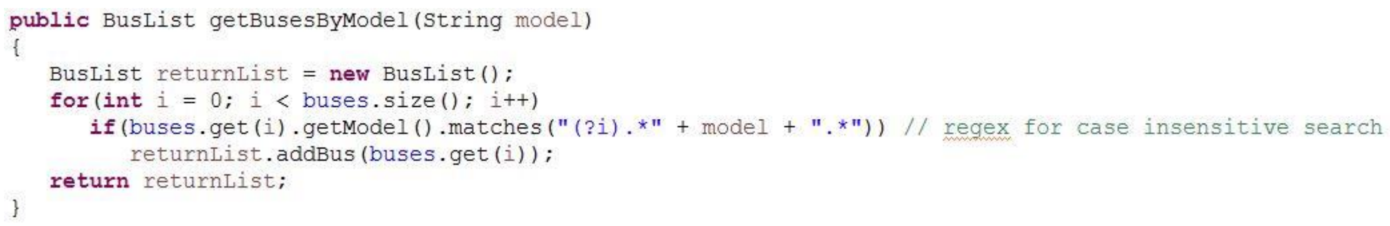
4. Implementation



```java
public void write(ViaBusCompanyModelManager obj)
{
    ObjectOutputStream out = null;
    try
    {
        FileOutputStream fos = new FileOutputStream(file);
        out = new ObjectOutputStream(fos);
        out.writeObject(obj);
    }
```

Figure 5 Method in FileIO class for writing a model to the binary file

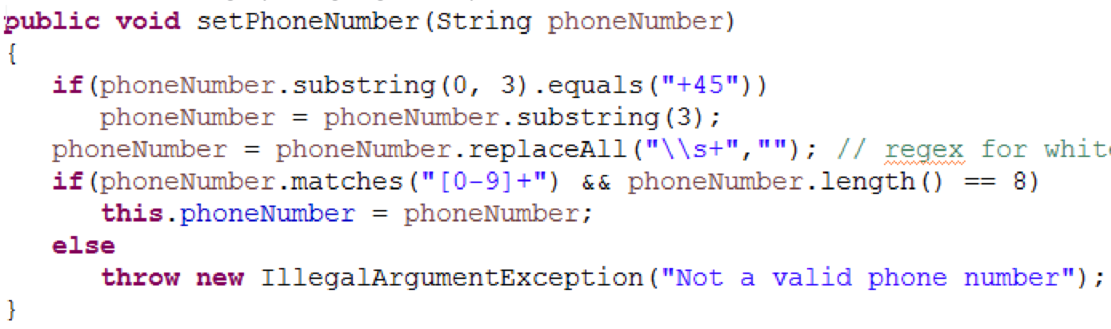
The system stores a single object of type ViaBusCompanyModelManager, which contains whole model. The read method is working in the same way. Just a single object representing our model is read from the file and returned.



```java
public BusList getBusesByModel(String model)
{
    BusList returnList = new BusList();
    for(int i = 0; i < buses.size(); i++)
        if(buses.get(i).getModel().matches("(?i).*" + model + ".*")) // regex for case insensitive search
            returnList.addBus(buses.get(i));
    return returnList;
}
```

Figure 6 Getting all the buses based on a model

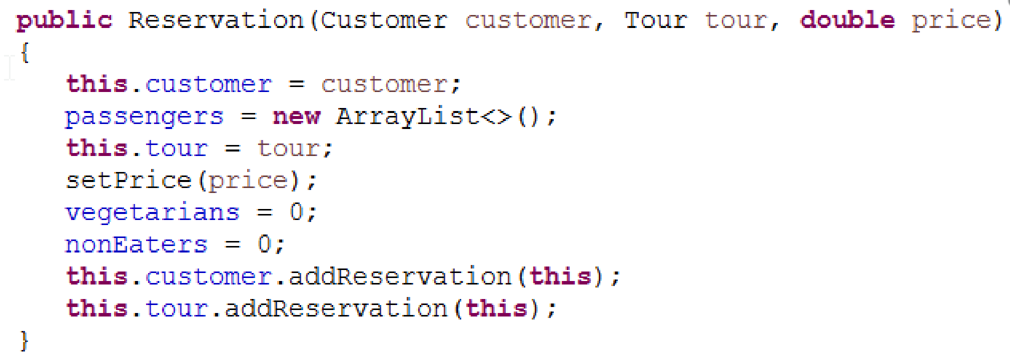
Here the system returns all the buses which contain part of the string specified in the parameter in their stored model string by using regular expression for case insensitive search[1].



```java
public void setPhoneNumber(String phoneNumber)
{
    if(phoneNumber.substring(0, 3).equals("+45"))
        phoneNumber = phoneNumber.substring(3);
    phoneNumber = phoneNumber.replaceAll("\\s+",""); // regex for whitespace
    if(phoneNumber.matches("[0-9]+") && phoneNumber.length() == 8)
        this.phoneNumber = phoneNumber;
    else
        throw new IllegalArgumentException("Not a valid phone number");
}
```

Figure 7 Method for setting phone number

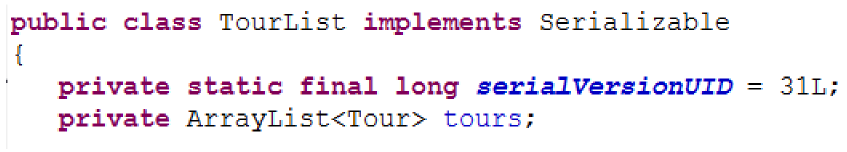
The method for setting phone number checks and processes the inputted string so that only 8 digits are stored. If the prefix “+45” for Denmark is present at the beginning of the string, it’s removed. Spaces between digits are deleted by replaceAll method using a regular expression for whitespace[2].



```java
public Reservation(Customer customer, Tour tour, double price)
{
   this.customer = customer;
   passengers = new ArrayList<>();
   this.tour = tour;
   setPrice(price);
   vegetarians = 0;
   nonEaters = 0;
   this.customer.addReservation(this);
   this.tour.addReservation(this);
}
```

Figure 8 Reservation constructor

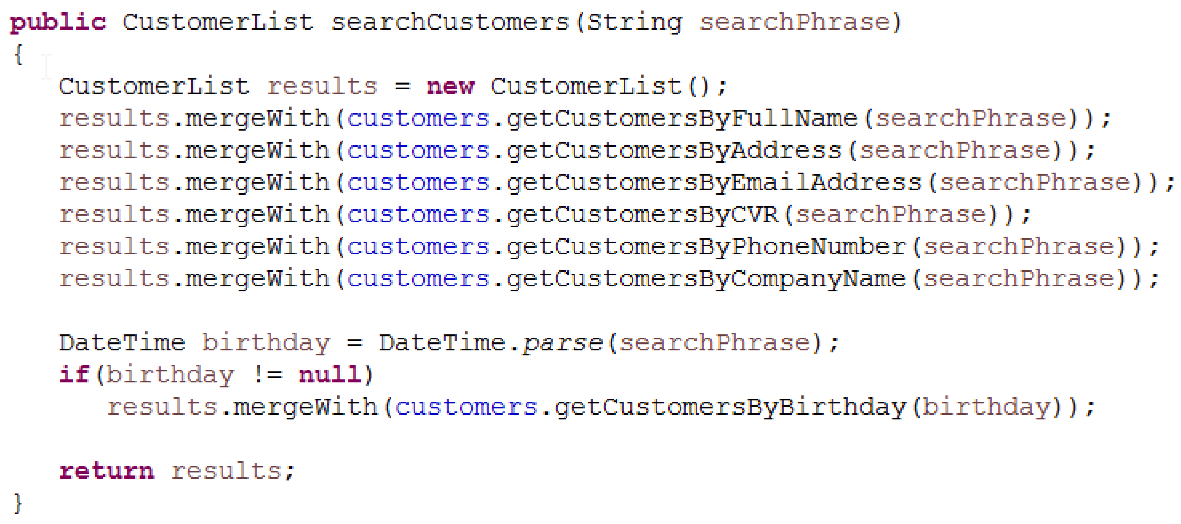
This constructor adds the newly created reservation to the assigned customer's and tour's reservations lists for immediate consistency of different lists.



```java
public class TourList implements Serializable
{
   private static final long serialVersionUID = 31L;
   private ArrayList<Tour> tours;
```

Figure 9 Usual class definition

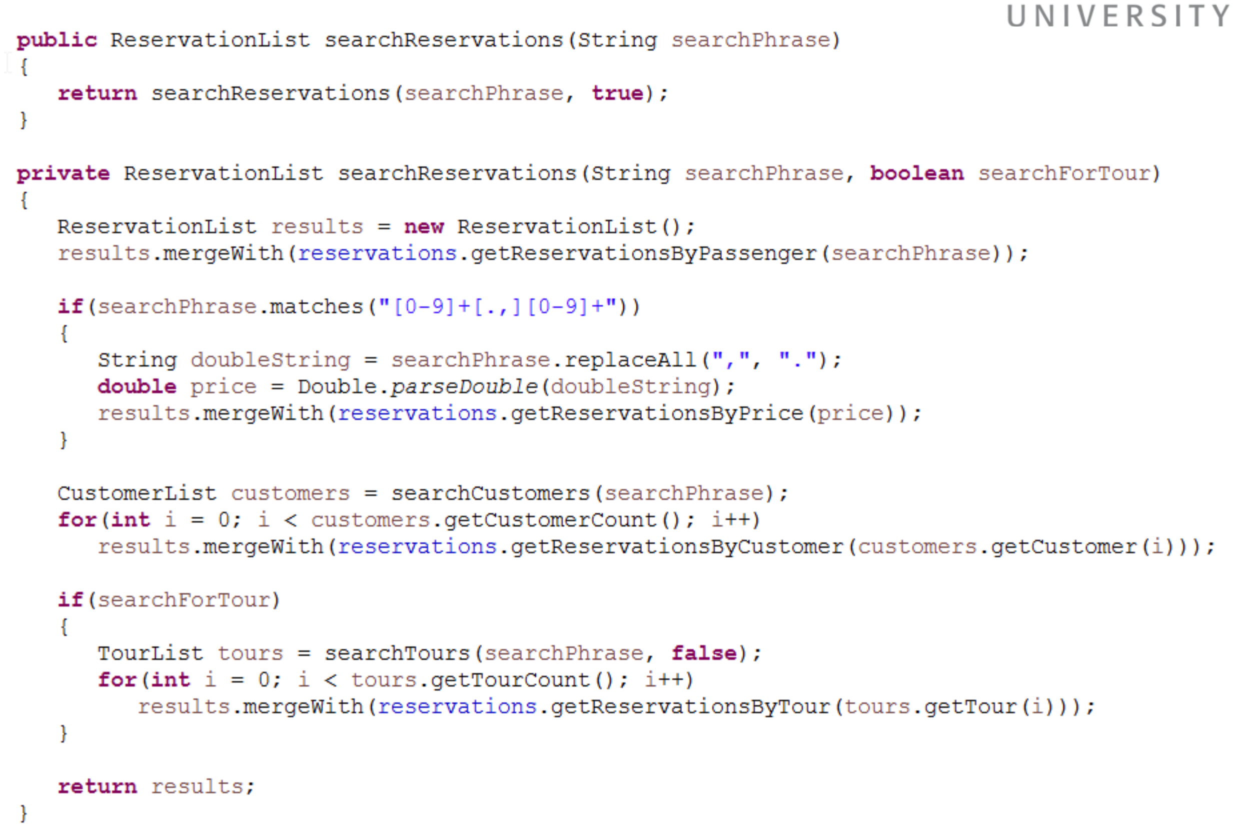
Every class implements Serializable so it can be stored in the file. The serialVersionUID variable is there because of implementing Serializable and it's related to inner functionality of the serialization. All the IDs in all the classes are unique.



```java
public CustomerList searchCustomers(String searchPhrase)
{
   CustomerList results = new CustomerList();
   results.mergeWith(customers.getCustomersByFullName(searchPhrase));
   results.mergeWith(customers.getCustomersByAddress(searchPhrase));
   results.mergeWith(customers.getCustomersByEmailAddress(searchPhrase));
   results.mergeWith(customers.getCustomersByCVR(searchPhrase));
   results.mergeWith(customers.getCustomersByPhoneNumber(searchPhrase));
   results.mergeWith(customers.getCustomersByCompanyName(searchPhrase));

   DateTime birthday = DateTime.parse(searchPhrase);
   if(birthday != null)
      results.mergeWith(customers.getCustomersByBirthday(birthday));

   return results;
}
```

Figure 10 Main class method for searching the costumers

In this method all the specific methods for getting customers by different parameters from CustomerList are called and the results are merged to resulting CustomerList which is returned. Also the inputed searchPhrase is parsed by DateTime class and if its format is "day month year", separated by slash, dot, hyphen, underscore or space, then the DateTime object is created and using that object you can search the Customers by birthday.



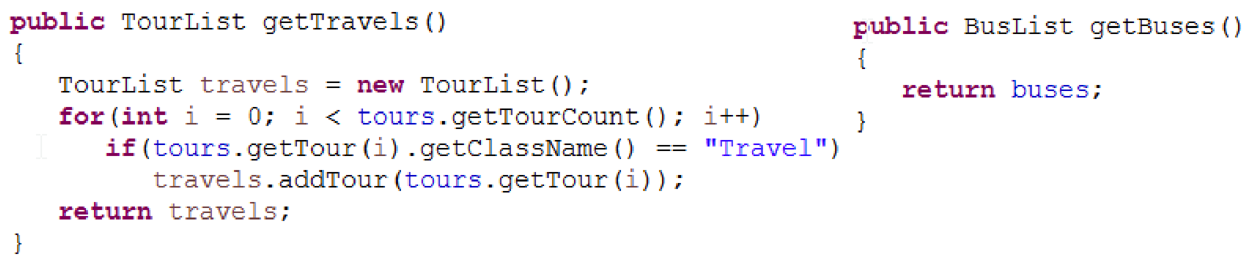
```java
public ReservationList searchReservations(String searchPhrase)
{
   return searchReservations(searchPhrase, true);
}

private ReservationList searchReservations(String searchPhrase, boolean searchForTour)
{
   ReservationList results = new ReservationList();
   results.mergeWith(reservations.getReservationsByPassenger(searchPhrase));

   if(searchPhrase.matches("[0-9]+[.,][0-9]+"))
   {
      String doubleString = searchPhrase.replaceAll(",", ".");
      double price = Double.parseDouble(doubleString);
      results.mergeWith(reservations.getReservationsByPrice(price));
   }

   CustomerList customers = searchCustomers(searchPhrase);
   for(int i = 0; i < customers.getCustomerCount(); i++)
      results.mergeWith(reservations.getReservationsByCustomer(customers.getCustomer(i)));

   if(searchForTour)
   {
      TourList tours = searchTours(searchPhrase, false);
      for(int i = 0; i < tours.getTourCount(); i++)
         results.mergeWith(reservations.getReservationsByTour(tours.getTour(i)));
   }

   return results;
}
```

Figure 11 Main model class method for search a reservaton

This method uses other search functions so it can use all the ReservationList functions which require different objects than a simple String. For example it searches for the customers which are used to search for reservations by this customer. It also calls method to searchTours, which however causes problem, because this method normally calls searchReservations and therefore gets to a loop.

The important part here is that the one-argument method is public and on default it calls the private homonymous two-arguments method with the second argument telling the method to search for a Tour, which is Java method for default argument values[3]. This default argument value is here in order to prevent a loop and a stack overflow. If the default parameter wouldn't be specified to false on a line, where we are calling searchTours method, these two methods would call each other, get to a loop and eventually cause a stack overflow.

This method also uses a regular expression to check if the search phrase matches a format of a double and therefore can be parsed to a double and method for getting reservations by price can be called[4].

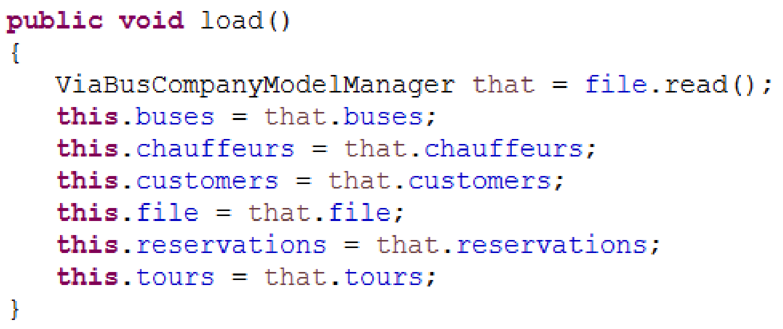


```java
public TourList getTravels()
{
   TourList travels = new TourList();
   for(int i = 0; i < tours.getTourCount(); i++)
      if(tours.getTour(i).getClassName() == "Travel")
         travels.addTour(tours.getTour(i));
   return travels;
}
```

```java
public BusList getBuses()
{
   return buses;
}
```

Figure 12 Comparison Between two get methods from main model class

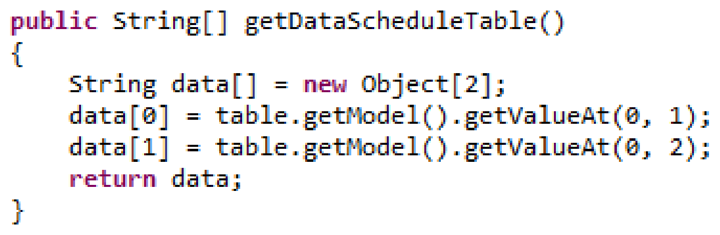
BusList is specified as a variable in the main model class, therefore it can be returned straight away. However, for all three types of the tours there is only one TourList representing all of these tours together. In order to get only tours representing in this case Travels, we have to compare the class name. This method is described in detail in sequence diagram section.



```
public void load()
{
   ViaBusCompanyModelManager that = file.read();
   this.buses = that.buses;
   this.chauffeurs = that.chauffeurs;
   this.customers = that.customers;
   this.file = that.file;
   this.reservations = that.reservations;
   this.tours = that.tours;
}
```

Figure 13 Load method

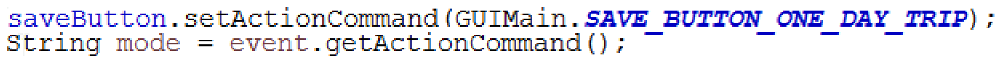
This method loads the data from a binary file. Method read returns the main model object, which has to be assigned to this model variable. Because it's conceptually not allowed to assign to this[5] (it's not a variable but a keyword), it's necessary to use a help variable, in this case called "that", which represents the temporary model, whose individual parts can be assigned to this model.



```
public String[] getDataScheduleTable()
{
    String data[] = new Object[2];
    data[0] = table.getModel().getValueAt(0, 1);
    data[1] = table.getModel().getValueAt(0, 2);
    return data;
}
```

Figure 14 Get data from table

This method gets the value from the table based on column and a row index[6].



```
saveButton.setActionCommand(GUIMain.SAVE_BUTTON_ONE_DAY_TRIP);
String mode = event.getActionCommand();
```

Figure 15 Setting and getting the ActionCommand for a button

In order to recognize which button was clicked, we have to set an action command for each button[7]. In this case a public static String from GUIMain class is used to prevent misspelling. When button is clicked, handler gets an action command from an event, based on which we can recognize which button was clicked and therefore which actions has to be performed.

5. Test

The testing was done only for the model. The whole program is not functional and therefore could not have been tested.

All the problems, that have been discovered by testing the model, have been completely fixed or only provisionally fixed. These problems might require some changes in the model design and therefore have been fixed only to a certain extent.

Model was tested on a few fictional entries that were created in the test class. Then all the methods from the main model class were tested in the same test class.

5.1. Creating individual objects

First the test on individual objects was made. Constructor of the classes representing individual objects in the model is using set methods for most of its arguments. In these methods, the inputed argument is tested and it is evaluated, if the argument is valid. For example, phone number string cannot contain anything else than numbers (except a plus sign in the country calling code). All the other set methods that are not used by the constructor have been tested separately as were all the other methods from the class.

This individual testing of objects was especially valuable when some more complex methods were being implemented.

5.2. Main model class

When the individual objects were created, they were added to the model using repsective add methods. For all the objects, also a delete method was called. This actually revealed some errors in equals method, which is apparently used when deleting an item from an arraylist.

Then the methods for searching objects and finding customer based on multitude of arguments were tested many times with differing search phrase and arguments, results were compared to the expected results and no difference was found. One significant problem which caused a stack overflow error occured and had to be fixed. This error has been caused by the methods calling each other never bypassing the call. Therefore a fix which allowed to bypass the method call using a boolean variable was implemented.

The only remaining methods were get methods for all types of the tours, chauffeurs and buses, and as expected, they were working.

6. Results

The program was supposed to facilitate the work and keeping track of the relevant data for the VIA Bus company. At the time of the deadline, program was still in development.

The model is working as aforementioned in the testing. The GUI layout was created, therefore the missing part is a controller which allows communication between the model and the GUI.

At the time of the deadline, the GUI layout and the model using the console can be shown separately to the customer.

7. Conclusion

Main goal of the project was to create a software for a VIA Bus company providing an easy to use system, allowing an employee to work with reservations, customers, tours, buses and chauffeurs.

Analysis and design of the project was done very carefully. However, implementation has only been partly made and includes the model, GUI layout, but not the controller mediating the communication between them.

Software which was developed in Java specifically for this customer matches most of the requirements in the model. However, it can’t be used by the customer, because of the missing connection between GUI and the model.

8. References

Gaddis, T., 2015. *Starting out with Java, Early Objects.* s.l.: Pearson Education Limited.
ICT Engineering, 2016. *VIA BUS (login required).* [Online] Available at: https://studienet.via.dk/Class/IT-SEP1Y-A16/Session%20Material/VIA%20Bus.pdf

---

[1] Jim Raynor and user108471, In Java, how do I check if a string contains a substring (ignoring case)? [duplicate] In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 6 Jun 2016 [cit. 2016-12-14]. Available on: http://stackoverflow.com/a/5613419

[2] Dave Jarvis and Gursel Koca, Removing whitespace from strings in Java [duplicate] In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 21 Apr 2016 [cit. 2016-12-14]. Available on: http://stackoverflow.com/a/5455809

[3] Jonik and gnavi, Does Java support default parameter values? In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 24 Jun 20009 [cit. 2016-12-14]. Available on: http://stackoverflow.com/q/997482

[4] Adam Liss and unwichtich, Java String - See if a string contains only numbers and not letters [duplicate] In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 8 Jan 2016 [cit. 2016-12-14]. Available on: http://stackoverflow.com/a/10575676

[5] aioobe, Why is assignment to 'this' not allowed in java? In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 2 Nov 2011[cit. 2016-12-14]. Available on: http://stackoverflow.com/a/7979791

[6] Maximin, Getting values from JTable cell In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 14 Dec 2016 [cit. 2016-12-14]. Available on: http://stackoverflow.com/a/16395986

[7] Qwerky, the getSource() and getActionCommand() In: Stack Overflow [online]. New York (NY): Stack Overflow, 2016, edited on 21 Nov 2011[cit. 2016-12-14]. Available on: http://stackoverflow.com/a/8215194