Object Oriented Programming Assignment



Prepared by-

**CHAITANYA KRISHNA CHAUHAN 2020A4PS1869P**

**RISHI GUPTA 2021A7PS0690P**

**RUDRA JEWALIKAR 2021A7PS0450P**

**YASHWARDHAN SINGH 2021A7PS2219P**

An Assignment

On

**Developing a Cab booking System using JAVA Object Oriented Programming**

Prepared for

Dr Amit Dua

(Instructor, CS F213 OOP)

BITS, Pilani

Prepared by

Chaitanya Krishna Chauhan

Rudresh Jewalikar

Rishi Gupta

Yashwardhan Singh

**A Report submitted in partial fulfillment of the requirements of CS F213: Object Oriented Programming Assignment**

Birla Institute of Technology and Science,

Pilani December 2022

**ACKNOWLEDGEMENT**

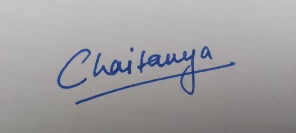
Our team GROUP 2, CS F213, would like to thank Dr. Amit Dua, Assistant professor, Computer Science and Information System, BITS Pilani who gave us this opportunity to make an assignment on object-oriented programming.

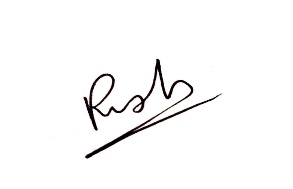
We would also like to thank our Teaching Assistant, Mr Sankha Das who helped us to build this project by providing proper guidance and resources for reference.

**ANTI – PLAGIARISM STATEMENT**

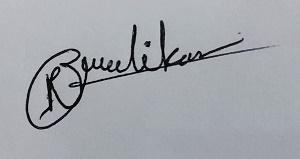
We as a team, declare that the entire project is solely created by us and the references used have been clearly indicated and acknowledged as such. The code is authentic as well as the algorithms and implementations are unique to our project.

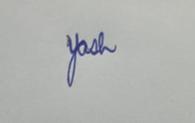
We are well aware that any kind of incorporation of online code or resources from other projects without acknowledgement will be treated as a plagiarism subject to punishment.

****NAME SIGNATURE

Chaitanya krishna Chauhan

Rishi Gupta

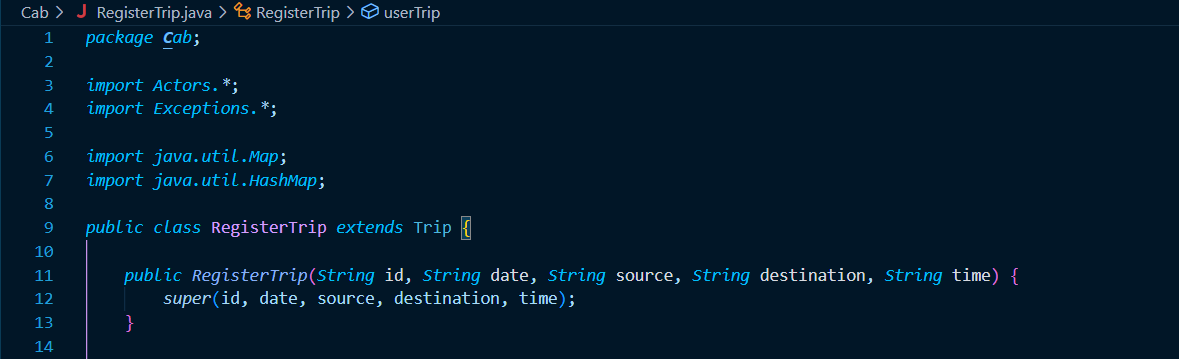
Rudra Jewalikar

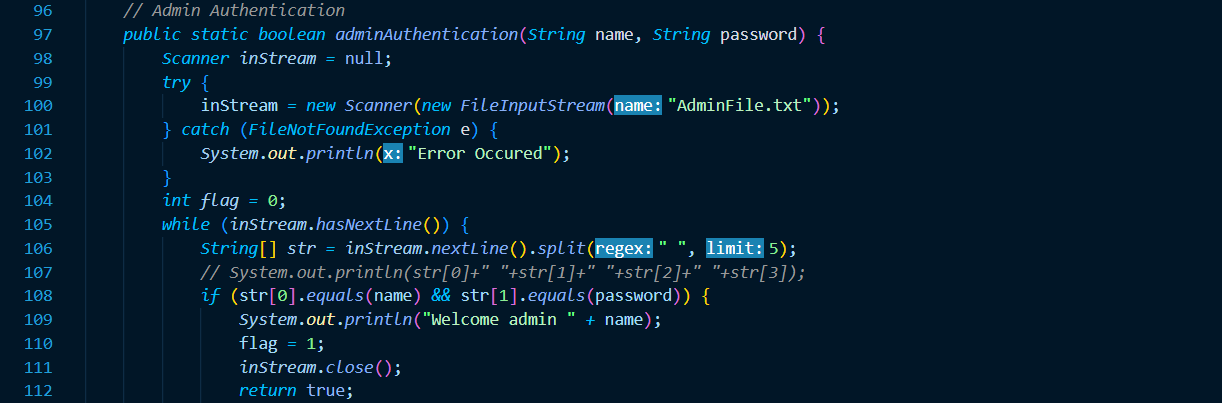
Yashwardhan Singh

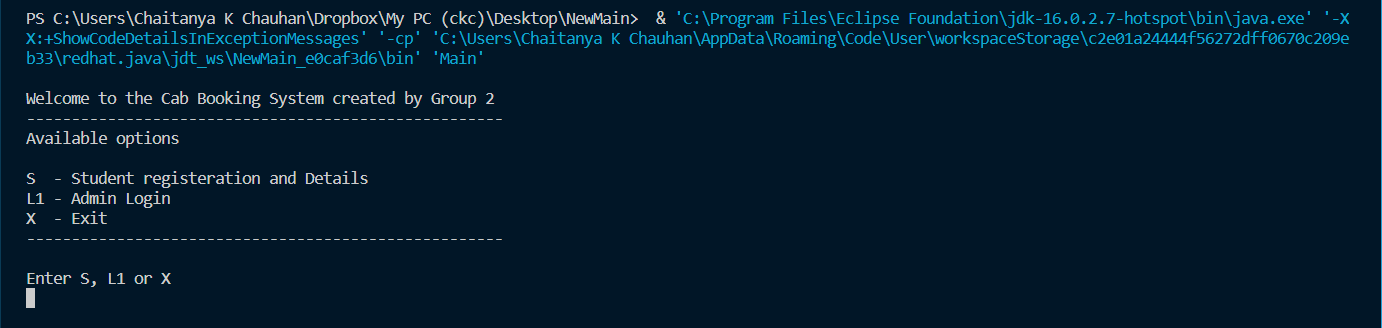
**INDIVIDUAL CONTRIBUTIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | CHAITANYA KRISHNA CHAUHAN | User and Trip class. ‘S’ ‘T’ and ‘C’ operations of the student in main method. Inheritance and Exception Handling. Final File documentation, Desgin Patterns | . |
| 2 | RISHI GUPTA | UML Sequence case diagram, parentUser and Fares class, File Handling, Encapsulation, userAuthentication method, ‘A’, ‘R’,’AC’ and ‘E’ operation in main method. Exception Handling. |  |
| 3 | RUDRA JEWALIKAR | UML Use case diagram, UML Class case diagram, Admin class and Admin login logout ‘L1 and ‘L2’ operations, Switch case implemented, inheritance, adminAuthentication method, AdminException, Design Patterns. |  |
| 4 | YASHWARDHAN SINGH | Study of Solid Principles, Design patterns, File Documentation. | IMG_1030 |

**OVERVIEW OF THE PROJECT**

****Our project is titled Cab Booking System. We have implemented object oriented programming principles namely abstraction, inheritance, encapsulation. We have a parentUser class which is inherited by User and Admin class. We have a registerTrip and registerUser class which are inherited from User and Trip classes respectively.

****We have created 4 packages and a main method. The exception package has exception definitions, the actors package have user, trip and admin class, the Cabservices package has the services available and the fares package has the fares class in it along with the text files to store and display the data.

We have used Command Line Interface (CLI) for our program to run. The user gets options for Admin login and Student registration once he runs our program. Other options come up when he has logged in either as student or admin.

**DESIGN PATTERN IMPLEMENTATION**

Analysing our project and the problem given, we would have implemented Command Design Pattern in our program. This is because our cab system shows a behavioural pattern once a request is registered for Student registration, Trip request or Admin login.

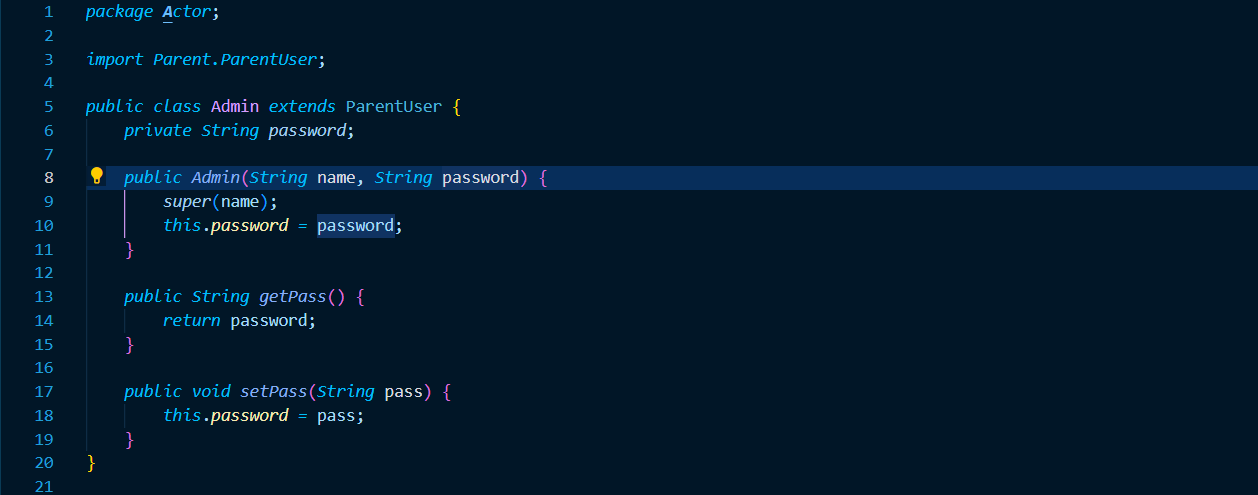
We would have created a generic class with multiple variables to store information. The requests would be accepted as objects of type User or Admin or Trip. We would create it through this class, and store all the members (like name, ID, etc. for student, username, password for admin, and trip ID, the destination for the taxi, etc.). Likewise, we added the required methods for each i.e., User login, Admin login, requesting a trip, assigning trip, etc within the same class. This would enable us to call appropriate methods for each object from the Main method and reduce the number of lines of code in the Main as well as other classes. This would be an ideal use of generics as well.

If we were to modify or redo the project for more options, we would have chosen the Composite Design Pattern. In Composite Design Pattern, the program has a tree like structure and the objects can have multiple objects inside them. We could use these features to make a program which takes asks the user for admin access or student access then, the students if want to book a cab gent an option of 4-seater or 6-seater car, then they get option of A/C or non-A/C car this way it becomes a tree structure.

**SOLID PRINCIPLES ANALYSIS**

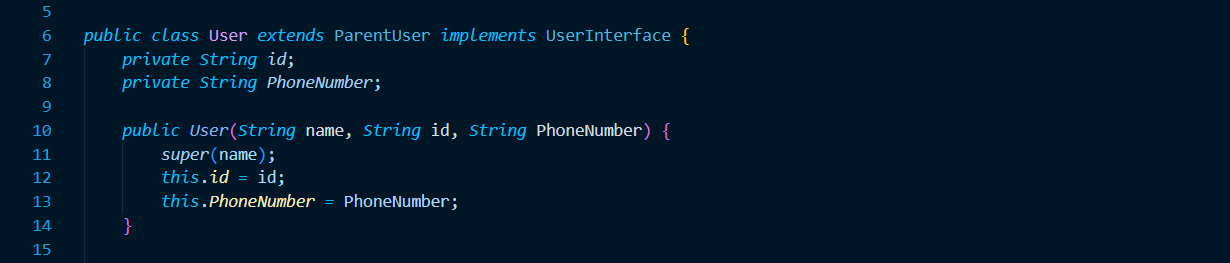
**Single Responsibility:** Each class has its own individual functionality. The User class is for defining the details of the User interface which is extended by registerUser class for registering new students. Similarly, the Admin class has admin details and the method adminAuthentication checks for admin username and password from AdminFile.txt.

The Fares class also follows single responsibility principle as only one responsibility is given to the class which is calculating the fares according to source and destination of travel. The Trip class violates single responsibility principle as it has multiple methods to implement namely listofTravellersContain () and getTripDetails ().

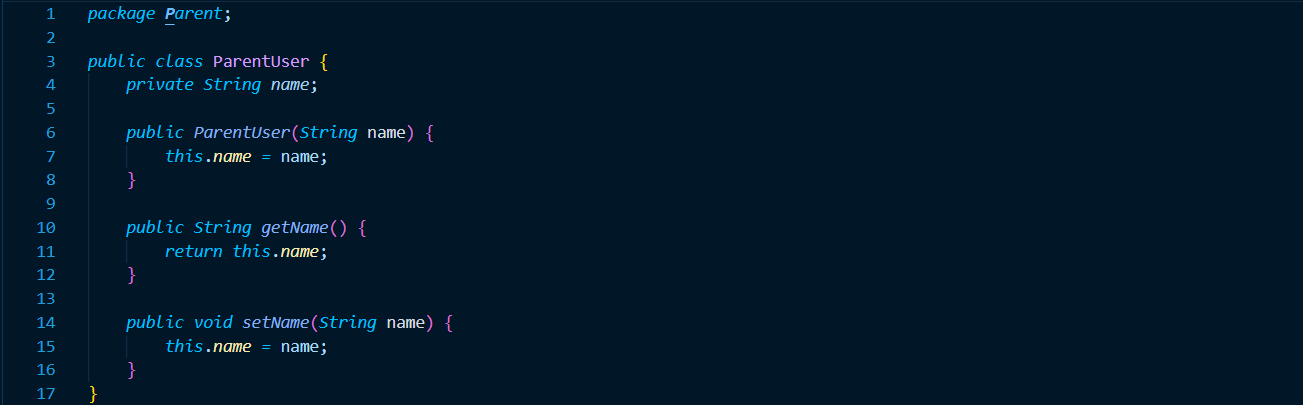


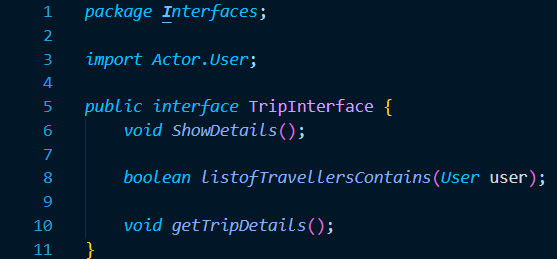
**Open for Extension, Closed for Modification:** The classes created are open for extension and closed for modification and any kind of future changes can be dealt by using inheritance of class. The User class and Admin class are inherited from the ParentUser and thus provide extension but not modification of code. There are several exceptions created like DateFormatException and ValidCityException which are extended from the predefined Exception class and thus, the original definition of the Exception class is not modified and only extended for use.

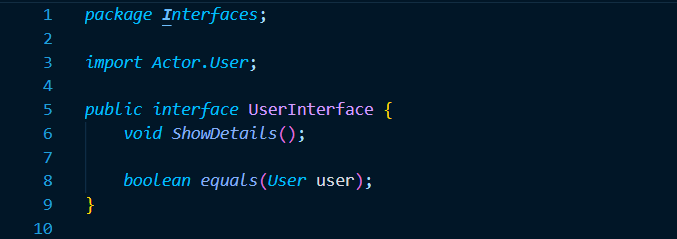
The Fares class violates this principle as it is an individual class which is neither implemented or extended from another class. Hence, any changes in the Fare class will directly have an impact on the whole program and the main method of Main class.

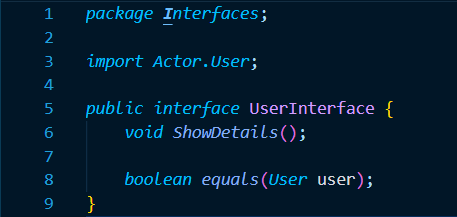


**Liskov Substitution:** Liskov substitution Principle is not followed as the instance of ParentUser can’t be use in place of User or Admin. The ParentUser is a parent class which doesn’t have the methods to show details of User or Admin. Hence, if we use it instead of the inherited classes, we will lose some functionalities. Similarly, the User and Trip classes cannot be used in place of registerUser and registerTrip as they both have different functionalities and inherit the parent class members for calculating and displaying the registered Users and Trips respectively.

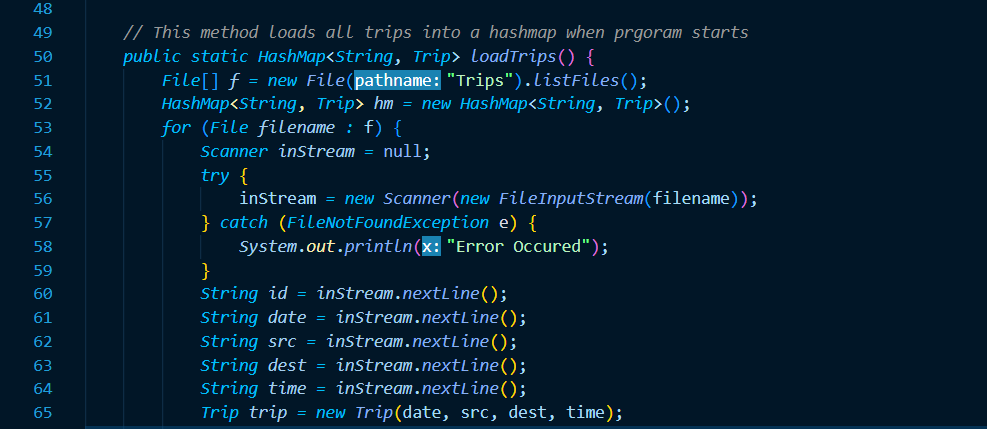


**Interface Segregation:** We have used two interfaces named as TripInterface and UserInterface which are implemented in Trip and User class respectively. We haven’t gone into larger interface segregation and is only restricted to Trip and User interfaces as a whole.

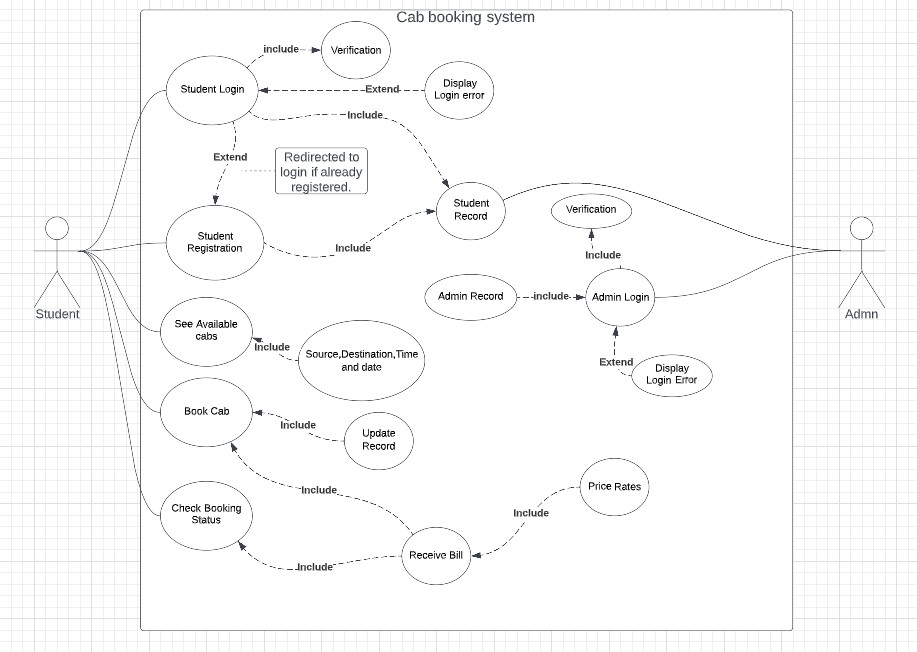




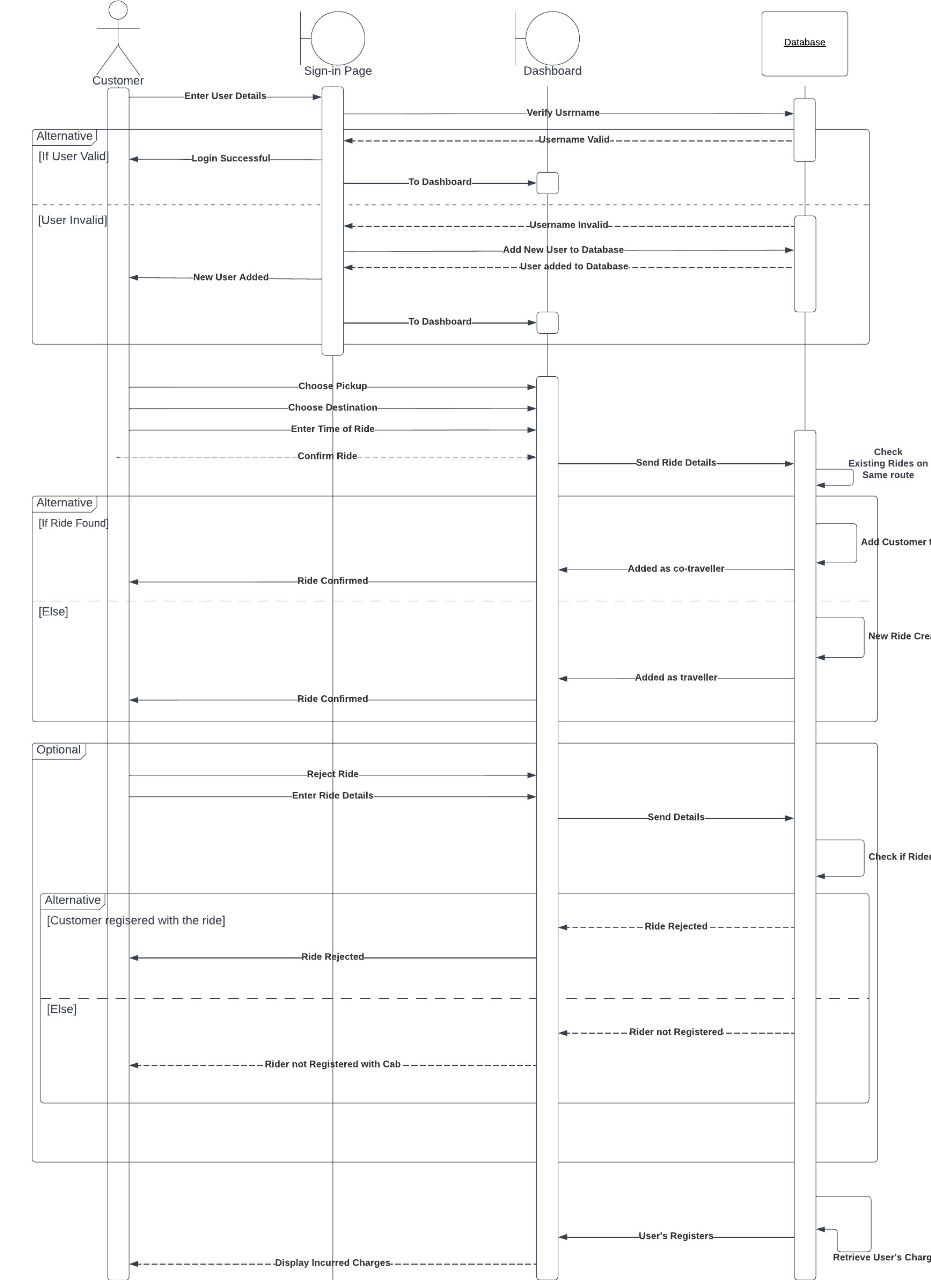
**Dependency inversion**: We had more use of concrete classes as per our project and the solution approach we adopted didn’t require a lot of decoupling. Hence, this principle hasn’t been used a lot by us. We have made use of collection frameworks ArrayList and HashMap. Apart from that, we have used interfaces but no abstract classes have been used and concrete classes are used instead.



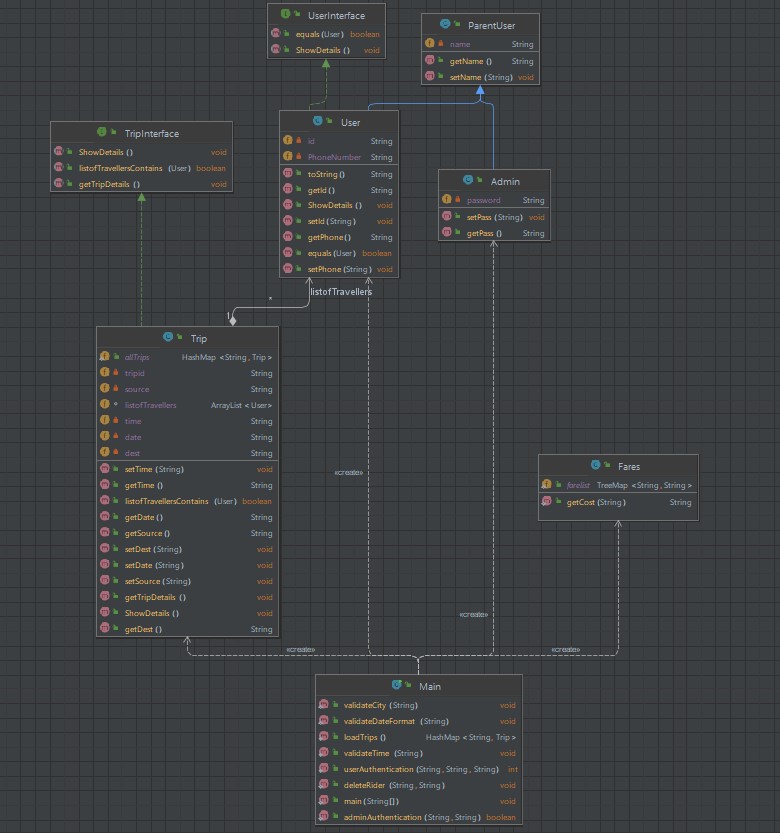
**UML DIAGRAMS**

****

**UML Use Case Diagram**

****

**UML Sequence Case Diagram**

****

**UML Class Case Diagram**

**THANK YOU**