# Documentation

## 

Simulate AirBnb

**Prepared By**

#### ▪ Arpit Sodani 2018B4A30863P

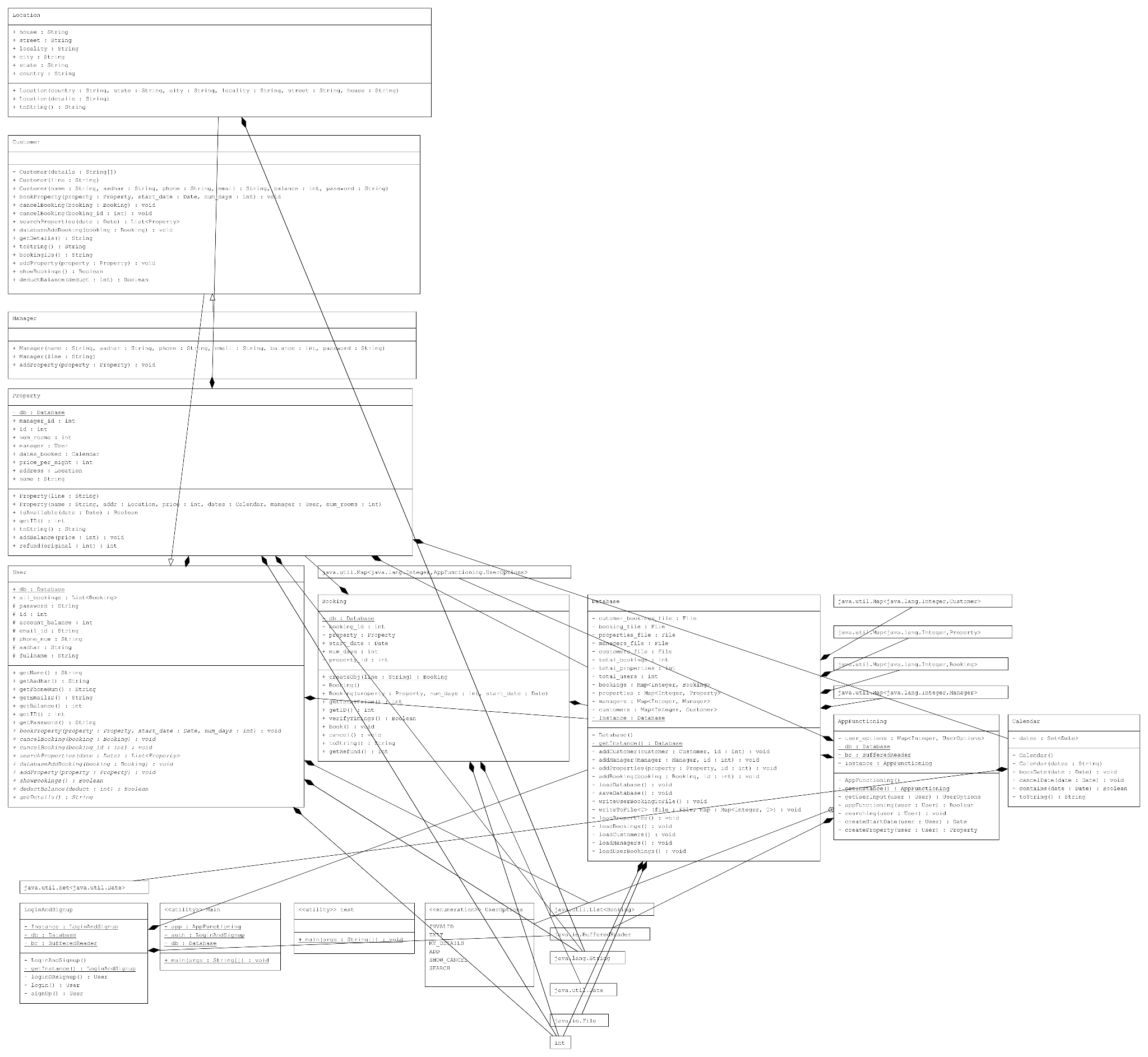
#### ▪ Lakshit Jain 2018B4A30886P

****

**Birla Institute of Technology and Science, Pilani**

# Class Description

1. User: Abstract class extended by Customer. It has all the attributes and methods of customer and manager.
2. Customer: Extended from User. It can search, book and cancel Property.
3. Manager: Extended from Customer. It has all functionality of a Customer and an additional functionality of adding Property.
4. Property: This class represents the physical property which needs to be rented.
5. Booking: Details of each booking is stored in this class
6. Calendar: Used to implement which dates are available or booked of a particular Property.
7. Location: Details of address of Property
8. Database: Implements the logic of saving to the database and loading from the database.
9. AppFunctioning: Implements the logic for functioning of the application.
10. LoginAndSignup: Implements the logic for login and signup for a user.
11. Main: Uses the Database, LoginAndSignup and AppFunctioning classes to run the application.



# OOP DESIGN PRINCIPLES ANALYSIS

## Encapsulate what changes -

Every property in a class should be encapsulated using accessor methods as it allows us to later change the variable inside the class but not reflect the changes to the outer world.

In this project we have declared private variables wherever possible and created getter and setter methods.

## Favor Composition over inheritance -

Allowing a class to contain object instances in other classes so they can be used to perform actions related to the class (an “has a” relationship) instead of allowing a class to use features from another class to extend its functionality.

We have used inheritance in Manager and Customer class where it was required but other than that we have extensively used composition to implement our logic.

## Program to interface -

While implementing multiple classes that will interact with each other, interfaces should be preferred instead of concrete implementation because by interfaces we can replace the implementations with ease.

In this project we have created an abstract user class and extended that to the customer and manager. Not used interfaces in any way.

## Open Closed Principle -

Open for extension, but closed for modification. As a result, when the business requirements change then the entity can be extended, but not modified.

Have indirectly used this principle while displaying multiple options to users to select from. The application is not highly open for extension and closed for modification.

## Loose Coupled Design -

Reducing the dependencies of one class with another class is known as Loose Coupling.

A decent amount of loose coupling is there but there are some places where the coupling is very high.

## Depend on Abstraction -

We couldn’t follow this principle as of now as we created concrete classes, consequences of which could be that we will have to implement our whole logic again if we require new implementation of the class.

# Analysis of Design Patterns

Design Patterns are like blueprints to solve some recurring design problems in the code. Design patterns are some typical solutions to common problems in designing. There are three basic design patterns: Creational, Structural and Behavioural.

In this project, the Creational design pattern is used in several classes. The Database class uses the Singleton pattern so as to avoid the use of a very large number of static methods and variables, and to enforce only one instance of database throughout the entire code. The pattern is also used for LoginAndSignup class and AppFunctioning class due to similar reasons. This has helped us avoid the use of static variables and methods as well as give us a global access point to the Database of the file.

The project also uses the Builder design pattern while loading the Database. The function reads the Database line by line and each line is passed to a function to create the object of appropriate type.

# Drawbacks

1. Not implemented error handling and validation. So if anyone does not enter the valid values then the application will stop and throw an error.
2. There are many places where the code is redundant.
3. Some variables are redundant and can be eliminated.
4. Implementation of the Database is text file based. So it is possible that we may encounter some parsing errors.
5. Database management is not done properly, some information is stored multiple times.

# 

# Future work

1. A proper GUI can be created for the application.
2. Currently the project is using a text file based database, but it can be upgraded to use a SQL based database.
3. Serializable Interface can also be implemented to each class, so that we can use ObjectReader and ObjectWriter functions to read and write to the database directly without parsing anything (if we are using a text file based database).
4. Extend this to provide many more functionalities like searching in a specified location, booking a room for more than one person, option for editing of user details, property details or booking details, and many such properties.
5. Improve the Validation and Exception Handling in the application.
6. Also implement the feature of notifications.

# References

Drive Link for project files:

<https://drive.google.com/drive/folders/1e5Uah2LW5OZwwyWUr22CIONNydtknpDL?usp=sharing>

Github link for project codes:

<https://github.com/arpitsodani15/AirBnB_project>