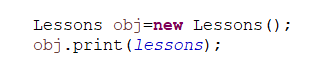
# Assumptions about the System

One single assumption has been taken into consideration in addition to the above scenario. This assumption is as follows:

In order to access the booking system, there is a need for every individual to first sign up using a phone number and password on the system. Once the user has been successfully registered only then they will be able to access the system functionality.

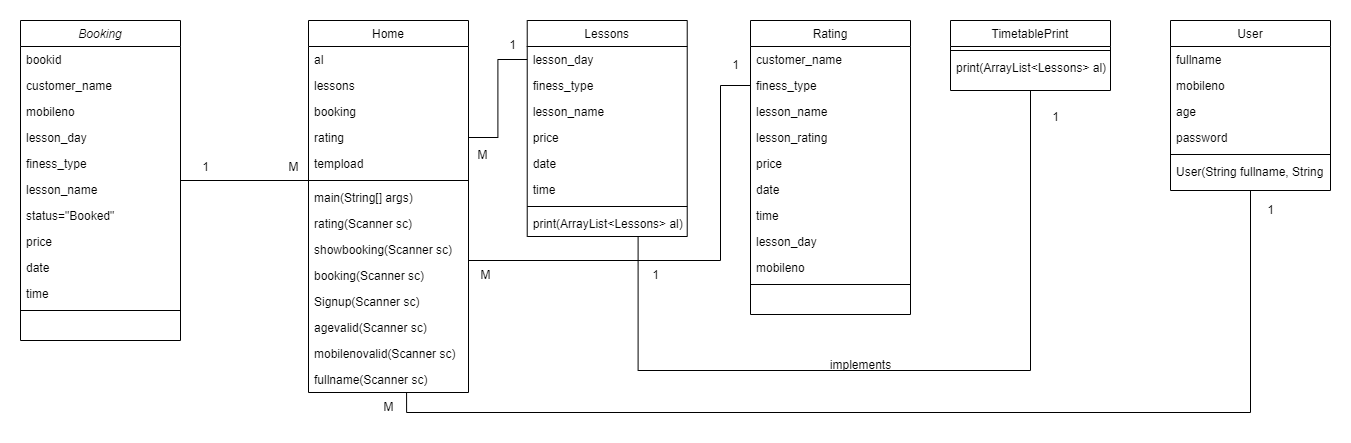
# Overall Structure and Design of the Program

The object-oriented design has been followed which helps to organize the code into logical units called classes. Each class represents an object in the problem domain and contains attributes and methods that define the behavior of that object.



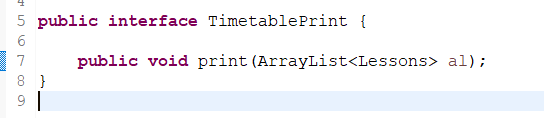
Here, an object of the Lessons class has been created and in that class, a print method has also been created that overrides the Time Table Print interface and this print method can be used to print all Array List objects.

# UML Class Diagram



# Design Patterns

Abstract Factory Method has been used in this program. It is one of the creational design patterns that produce families of related objects without specifying their concrete classes. Below shown is the snapshot of the usage of this Design Pattern.

****

****

An interface named “TimetabelPrint” has been created and within that interface, a method named “print()” has been declared which can be used to show the current available timetable to the user and print method override in Lessons class.

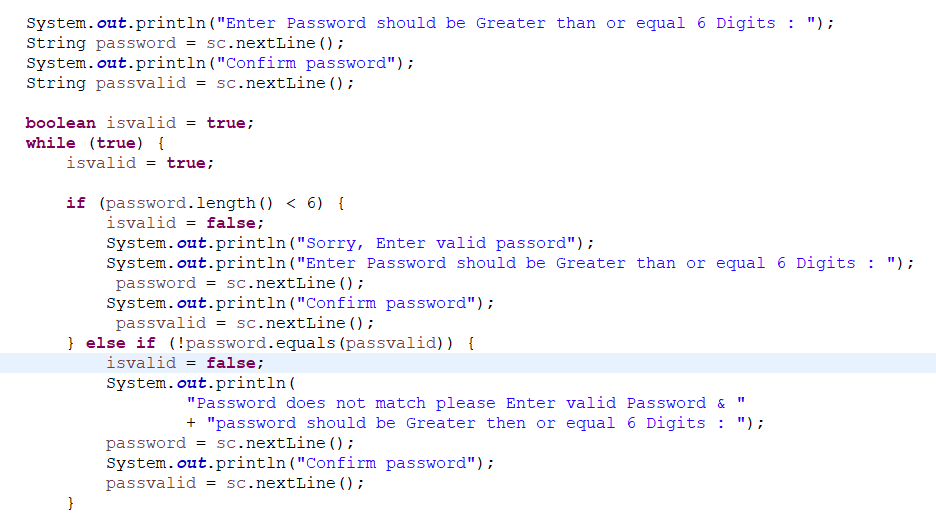
# Testing

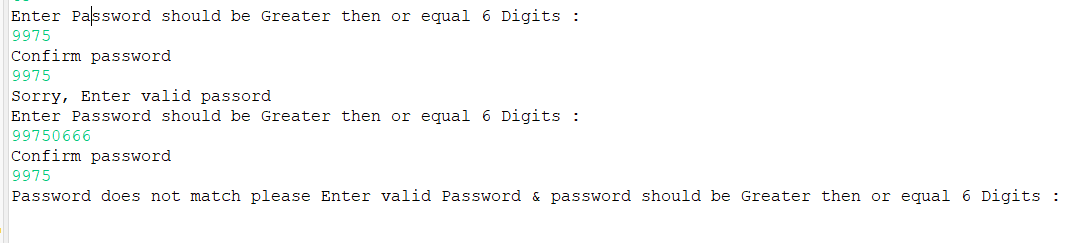
**Test Case 1:** Checking that entered password must equal to or greater than 6 digits

**Test Data:** Entering the Password(9975)

**Expected Output:** The program should show an error message and ask the user to re-enter the password.

**Actual Output:** Error Displayed



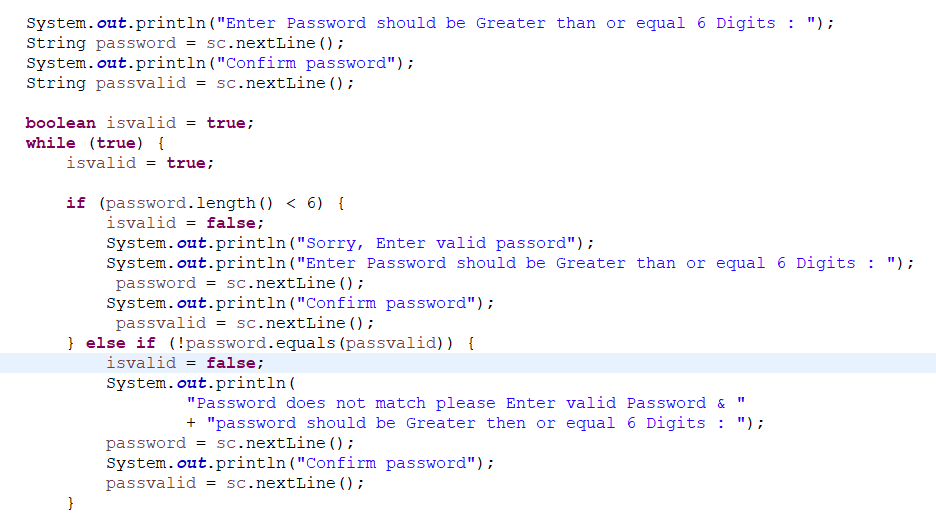


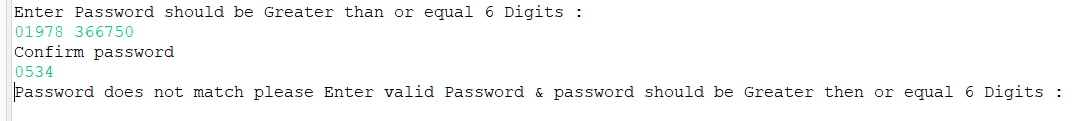
**Test Case 2:** Password and Confirm password must be the same

**Test Data:** Password

**Expected Output:** The program should prompt that password and confirm password must be same

**Actual Output:** password and confirm password must be same



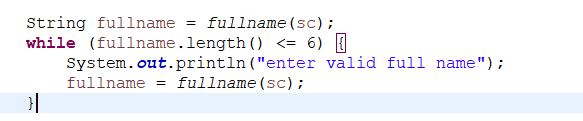


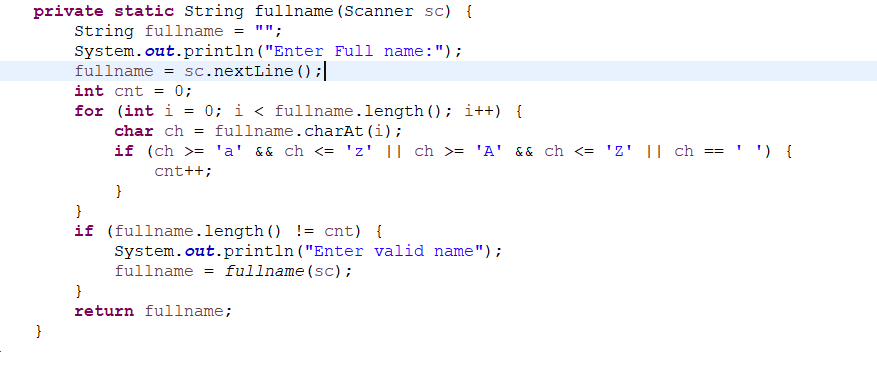
**Test Case 3:** Full name length should be Greater than or equal 6

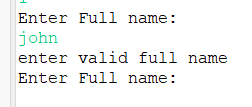
**Test Data:** Entering the Full Name(dadasasas)

**Expected Output:** The program should show an error message and ask the user to re-enter the Name.

**Actual Output:** The program shows an error message and asks the user to re-enter a valid name.





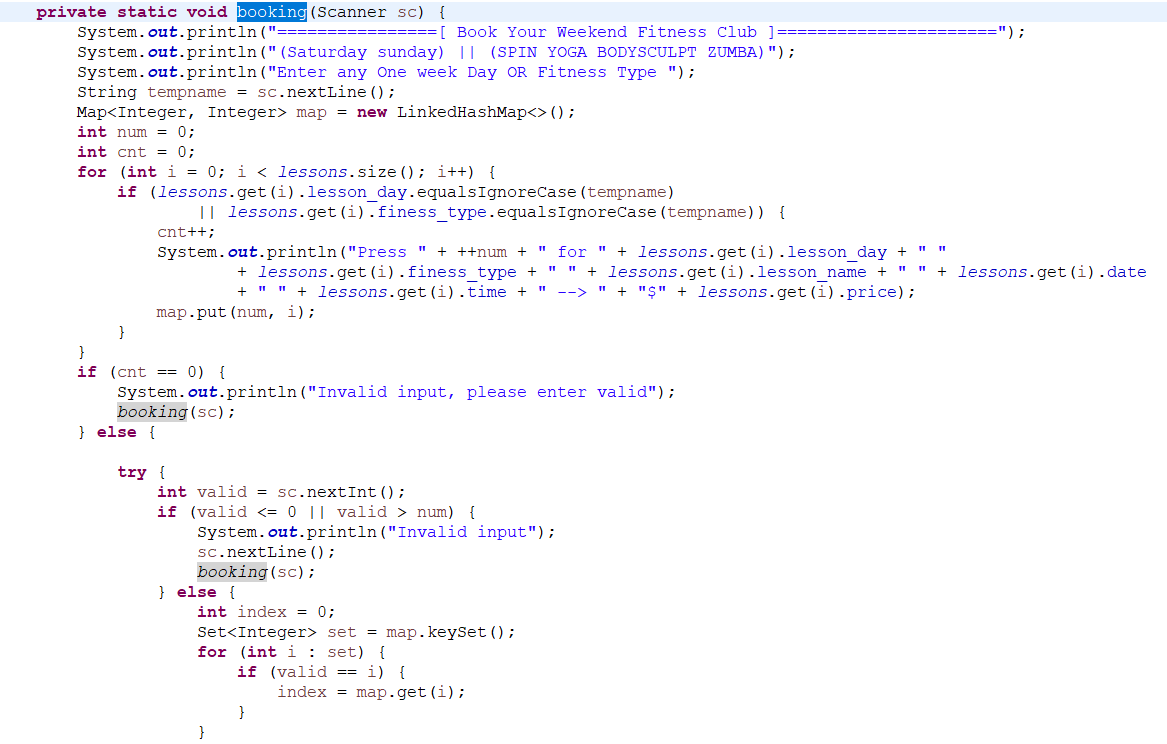


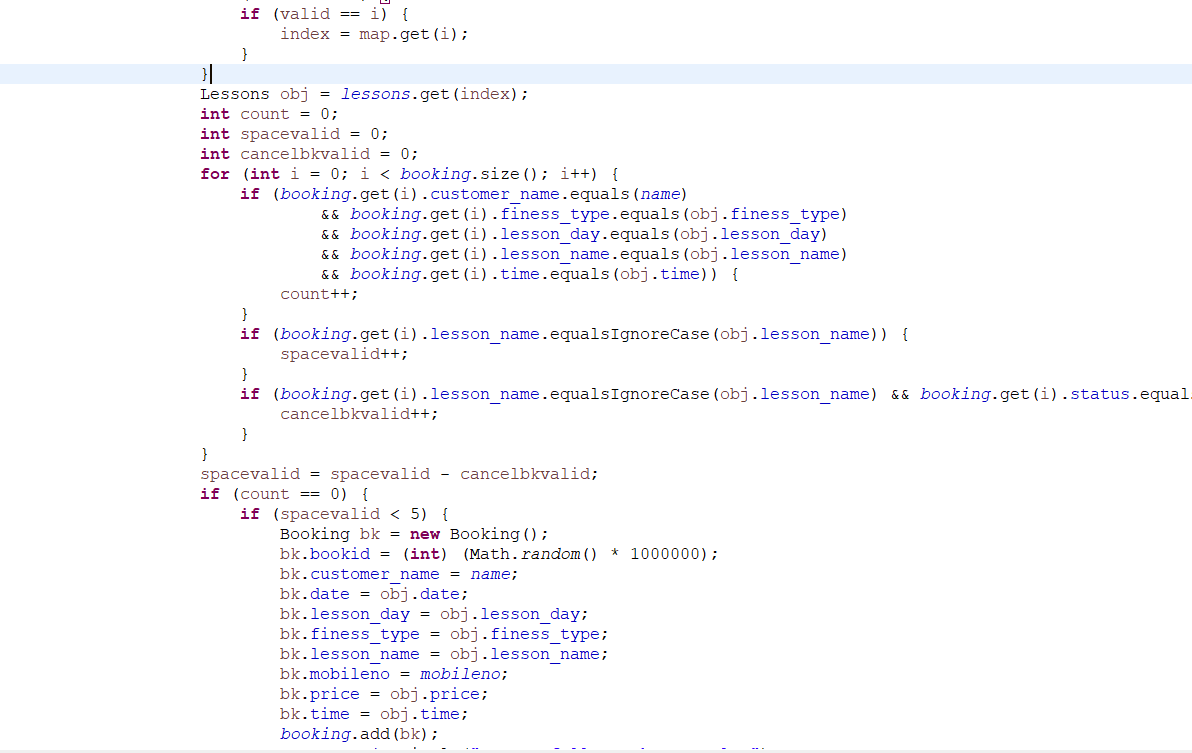
**Test Case 4:** Users can’t Book duplicate lessons

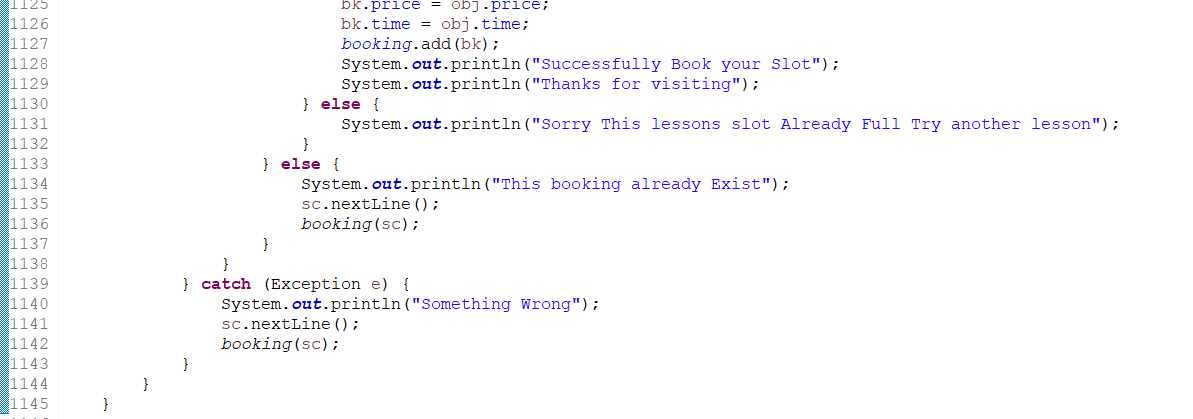
**Test Data:** Selecting the same lesson and same time

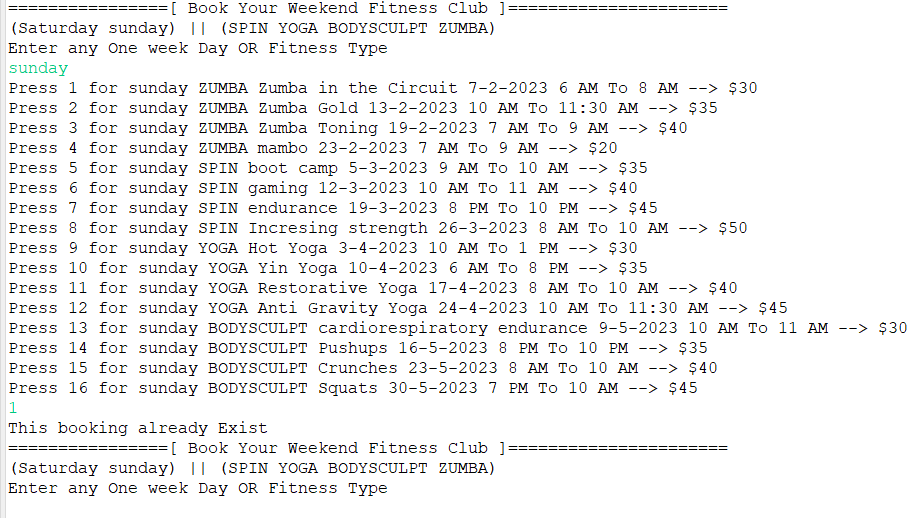
**Expected Output:** This booking already exists

**Actual Output:** This booking already exists







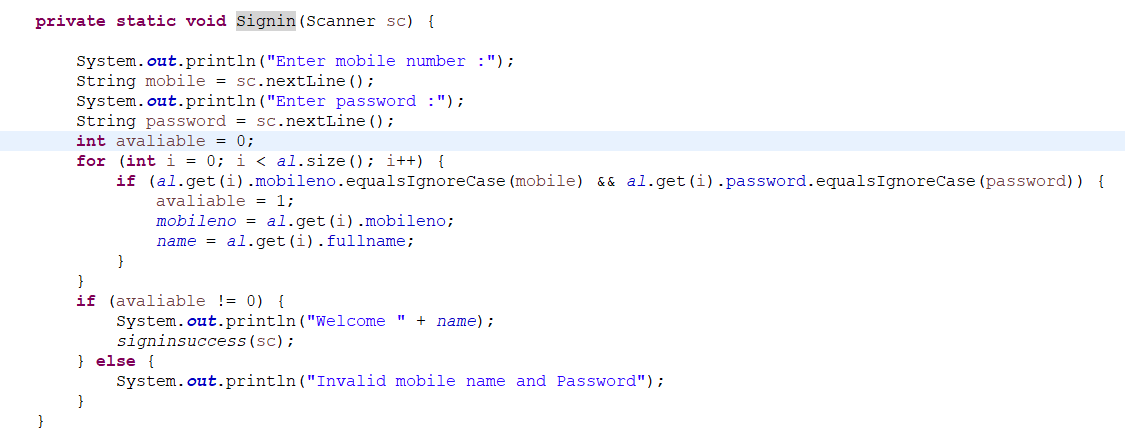


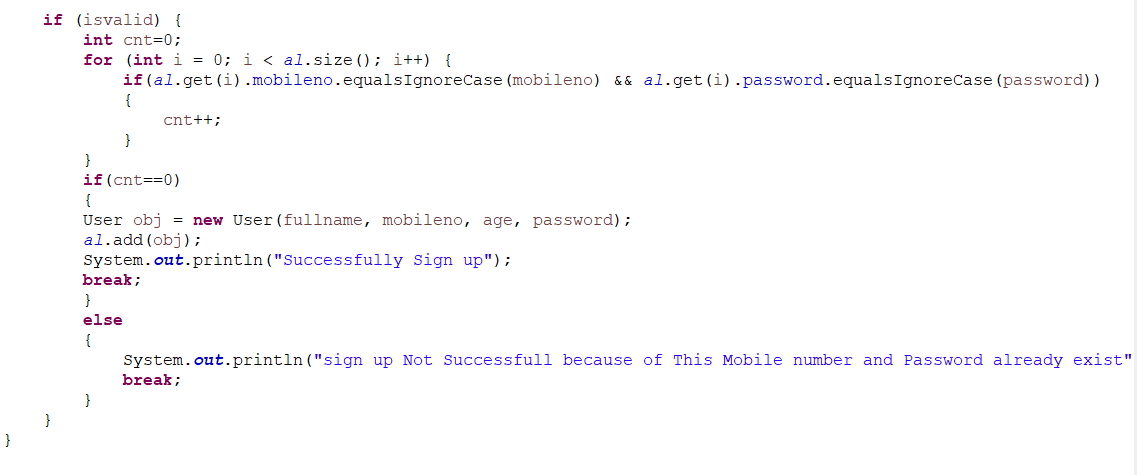
**Test Case 5:** Duplicate Registration Not allowed

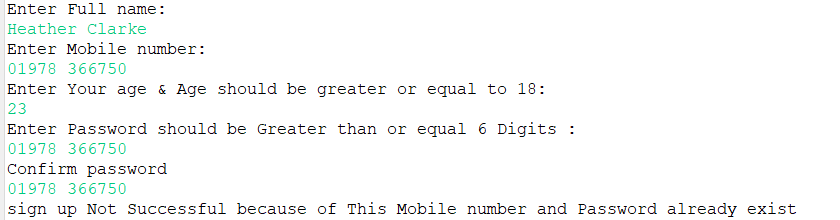
**Test Data:** Already registered user ID and password

**Expected Output:** Singup Not Successful because mobile number already exists

**Actual Output:** Singup Not Successful because mobile number already exists

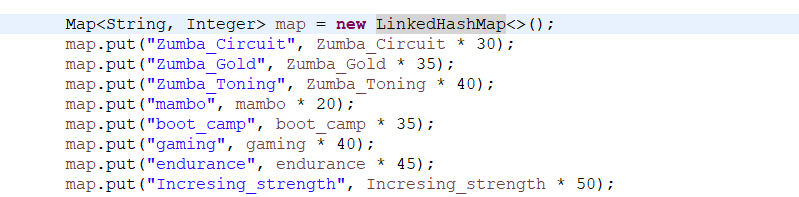






# Refactorization

**import java.util.LinkedHashMap;**



Here, a LinkedHasMap has been used which contains the value on the basis of a key. This means that map is a key and value pair. It maintains insertion order and contains a unique key.

Similar to this, there is a need to maintain a unique lesson name and income for that specific lesson in order to perform calculations. So, a **LinkedHashMap** has been used. The lesson name acts as key and income of that lesson acts as value. These two values are put in that map and this map is used inside of the report function.