|  |
| --- |
| PROJECT REPORT |



**OBJECT ORIENTED DESIGN AND PROGRAMMING**

**(2017/2018 SEM 2)**

**Members:**



Contents

[APPENDIX B 3](#_Toc510461166)

[UML CLASS DIAGRAM 4](#_Toc510461167)

[abc 4](#_Toc510461168)

[UML SEQUENCE DIAGRAM 5](#_Toc510461169)

[OO DESIGN CONSIDERATION 6](#_Toc510461170)

[SRP – Single Responsibility Principle. 6](#_Toc510461171)

[OCP – Open/Closed Principle. 6](#_Toc510461172)

[LSP – Liskov Substitution Principle. 6](#_Toc510461173)

[ISP – Interface Segregation Principle. 6](#_Toc510461174)

[DIP – Dependency Inversion Principle. 6](#_Toc510461175)

[TESTING 7](#_Toc510461176)

# APPENDIX B

Declaration of Original Work

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Course  (CE2002 or CZ2002) | Lab Group | Signature/Date |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Important notes:

1. Name must EXACTLY MATCH the one printed on your Matriculation Card.

# UML CLASS DIAGRAM

## abc

# UML SEQUENCE DIAGRAM

# Object oriented design concepts

**SDLC**

Software Development Lifecycle

Software Development Life Cycle is a well defined, structured sequence of stages to develop the intended software product. Requirement and planning analysis allows the group members to evaluate various technical approaches that can be followed to implement the Hotel Reservation and Payment System successfully. A key component of a successful software is to have good design. It clearly defines all the architectural communication modules and data flow. It is a fundamental to Evolution stage where easy plug and play to the software allows extendibility, reusability and portability.

## 

## Abstraction

In Object-Oriented(OO) technology, thinking in terms of objects perspective, it is the initial step to identify the essential characteristic of an object , include state, attributes and behaviors related to the object.

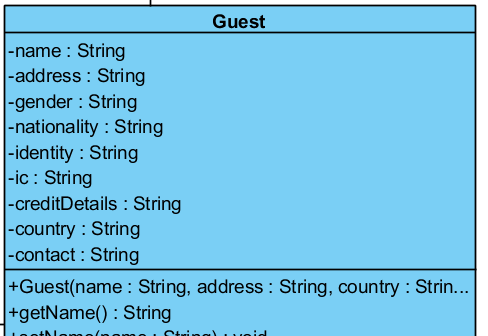
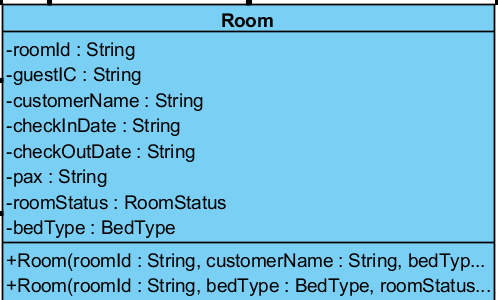


Figure : Room entity class

Figure :Guest entity class

## Encapsulation / Information Hiding

Encapsulation build a barrier to protect an objects private data. In good OO design, an object should only reveal the interface that other object needs to use. Information hiding is major part of encapsulation to avoid unnecessary modification to the data. For example *Guest* class with private attributes such as name and address and modification are only available through getters and mutator methods. Robust classes are design with encapsulation in mind that is within control.

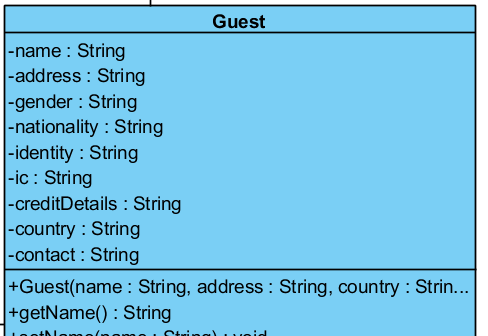
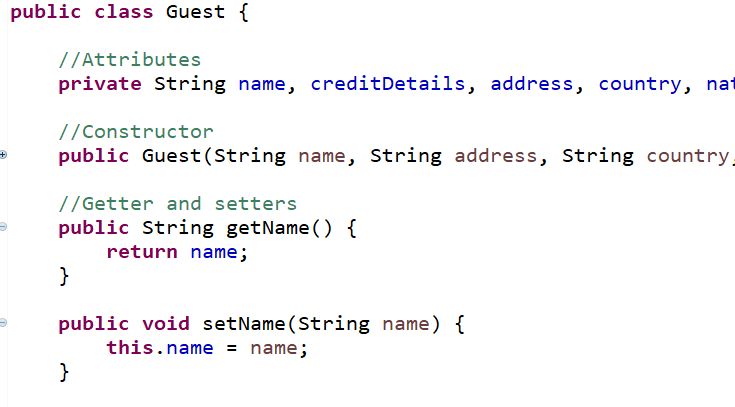


Figure : Guest entity class with code implementation

## Inheritance

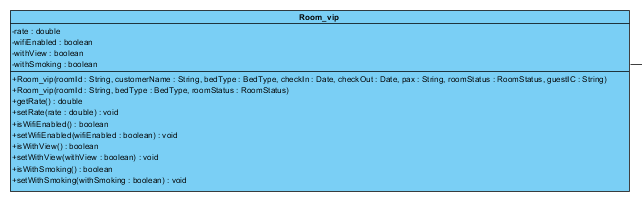
It is a power mechanism allow a class to inherit the attributes and methods of parent class. Some of the advantage is reduce in code duplication, extracting commonalities of various classes. For example abstract parent class of *FileIO,* inherit classes such as *GuestFileIO* and *ReservationFileIO* .

## Polymorphism

Polymorphism means many shapes. During runtime, actual object is being used, Polymorphism enable us to define any number of derived classes such as *GuestFileIO* and *ReservationFileIO* , different base class method such as parseList() and export() to allow polymorphism effect. No matter what kind of FileIO object is, correct method will return accordingly.

# OO DESIGN CONSIDERATION

## SRP – Single Responsibility Principle.



Room\_vip will only be responsible for room that are used by VIP and the only reason that cause an alter will be due to the changes made to the attributes in this class. Therefore, there is only one responsibility and one reason for this class to change.

## OCP – Open/Closed Principle.



As seen in the class diagram, RoomServiceMenu inherit methods from Menu. RoomServiceMenu which is the subclass will override methods in Menu which is the superclass however this will not cause any changes to the methods in the superclass. Therefore, we are able to change what the modules do, without touching the source code of modules.

## LSP – Liskov Substitution Principle.

## ISP – Interface Segregation Principle.

## DIP – Dependency Inversion Principle.

# TESTING