

**CE/CZ2002: Object-Oriented Design & Programming**

**Assignment**

**Group 4**

Building an OO Application

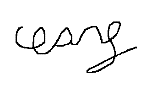
2017/2018 Semester

SCHOOL OF COMPUTER SCIENCE & ENGINEERING NANYANG TECHNOLOGICAL UNIVERSITY

**Declaration of Original Work for CE/CZ2002 Assignment**

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 honoured 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 | Lab Group | Signature/Date |
| Arkar Min | CZ2002 | FSP6 | Min/13.04.2018 |
| Kaw Teck Lin | CZ2002 | FSP6 |  |
| Png Qun Jia | CZ2002 | FSP6 |  |
| Wang Guo Wei | CZ2002 | FSP6 | /13.04.2018 |

1. Design Consideration and Use of OOP concepts

The very first step we take is identifying important classes for the system and group them into Entity Class, Control Class and Boundary Class.

We want to hide our implementation of methods from users and we do not want them to directly interact with the data files. So, we use Layered Architecture for our system (Refer to the class diagram). There will be total four layers, 1. Presentation layer, Business Layer, Persistent Layer and Database Layer. We put the main application interface class and other interface classes in the presentation layer, control classes in business layer, entity classes in persistent layer and the data text.files in the database layer. We have decided to use this architecture because

1. It increases flexibility and reusability. For example, if we want to update from CLI (Command Line Interface) to GUI (Graphical User Interface), we just need to update the main application interface class in the presentation layer. All other classes from other layers remain the same.
2. It also encourages extensibility and maintainability. Different layers can be independently updated, maintained and deployed.
3. It makes the system loosely coupled since the layers must communicate via top-down approach.
4. Use of OOP Design Principle
   1. Single Responsibility Principle

We make sure that every single class has only single responsibility. For example, Room Controller class has methods relating with rooms only. It cannot manipulate or does not have abilities to manipulate other unrelated classes like reservations and so on.

* 1. Open-Closed Principle

We implement this principle for Room class. Room class have “RoomType” attribute. That “RoomType” attribute is an interface. So, it we want to include more room types in the future, we do not need to change code of the Room class. We just need to add a new room type class realizing “RoomType” interface.

* 1. Interface Segregation Principle

All room type classes realizing the “RoomType” interface required all the methods declared in “RoomType” interface. There are no unnecessary methods inside “RoomType” interface.

1. Consideration for Future Extension (Sending reminder via SMS for a few days before reservation date)

For this extension, we can create a SMS controller class and SMS entity class. We can use the Observer design pattern here. In reservation controller, we can add a new method which will compare the date of the reservations with 3 days after local date and it will add reservation’s date and contact number to the list and return the list. Then SMS controller will call this method to get contact number and reservation date. Then there will be a method which creates SMS object with standard message, “Your reservation at Hotel will be expired at dd/mm/yyyy (reservation date)”. Then SMS controller will have another method which will send out this SMS object to users.

In this case, we can clearly see that we are not required to change any existing code inside our system to accommodate this new function.