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 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	Lab Group	Signature/Date
Quek Kar Min	SC2002	SC1	 16/4/2022
Melissa Ng Li Yun	SC2002	SC1	 16/4/2022
Vignesh Ezhil	SC2002	SC1	 16/4/2022
Pravind Kummar Kunasilan	SC2002	SC1	 16/4/2022

Important notes:

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

Table of Contents

Link to Github Repository: https://github.com/pravindkk/sc2002

1. System Overview

The **Hotel Reservation and Payment System (HRPS)** computerizes the process of the following:

- Making Hotel Reservations
- Recording of Orders
- Displaying of Records

This system will only be used by the hotel staff.

2. Assumptions

Stated here are the assumptions that were made by the team when developing the system, excluding those that were already mentioned in the assignment brief.

- Room Rate is the base room rate (not including service charge or GST)
- Room Rate is per 24 hours
- Service Charge and GST is 10% and 7% respectively
- Each Guest can only have 1 Reservation
- Check-in time is at 1500 and if by 1600 the Guests do not check-in, the Reservation is considered expired
- Rooms can only have 1 Reservation at any point of time
- Empty Rooms will not be able to order room service
- Guests' credit card numbers are unique
- Room Service Orders will be DELIVERED before the guests check out
- Guests will always check-out at the stated check-out date

3. Design Considerations

Mentioned below are some of the design considerations that we have implemented in the development of the HRPS application.

3.1. SOLID Design Principles

3.1.1. S - Single-responsibility Principle

The Single-Responsibility Principle (SRP) states that each class should only hold one responsibility. In other words, it should be programmed to only do one thing.

Hence, we have split the classes into entities, controllers, UIs and databases. For example, our Guest class is simply just an entity, it will manage guest objects in our system. The GuestController class will

implement the possible use cases that a Guest object can do. While the GuestUI class can only be interacted with by the actor, holding the functions that the actor is able to execute.

3.1.2. O - Open-closed Principle

Bertrand Meyer in his book, Object-Oriented Software Construction, explained that the Open-Closed Principle (OCP) indicated that the entities created, including classes, functions and modules, should be unmodifiable but extendable. The code should be written in a way that allows new functions to be added but in the process, the existing code should not be changed.

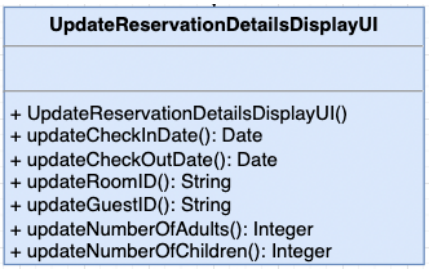
In our system, we have used the database interface to allow for the extension of the existing methods and variables of the class. If there were to be an extension of the system, say now if there was a possibility for guests to book function halls, the system would have to store the function hall information. There would simply be another class entity (DBFunctionHall) that would be connected to the interface.

3.2. Object-Oriented Principles

The HRPS was implemented with various object-oriented (OO) concepts to enhance the reusability and maintainability of the application.

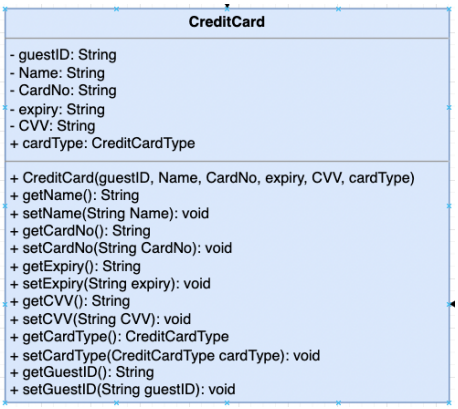
3.2.1. Abstraction

Abstracting all the details that the user does not need to interact with, provides a clean and simple to understand interface. It's not important for the user to know how the program books the reservation and updates the database, what matters is that they can.



3.2.2. Encapsulation

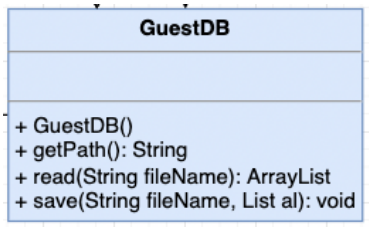
In Java, encapsulation helps in data protection, safeguarding the internal contents of a class. We have restricted direct access to data fields of a class by setting them to public or private. Additionally, for every class



variable, we implement a getter and setter method to aid in obtaining and changing the variable's value.

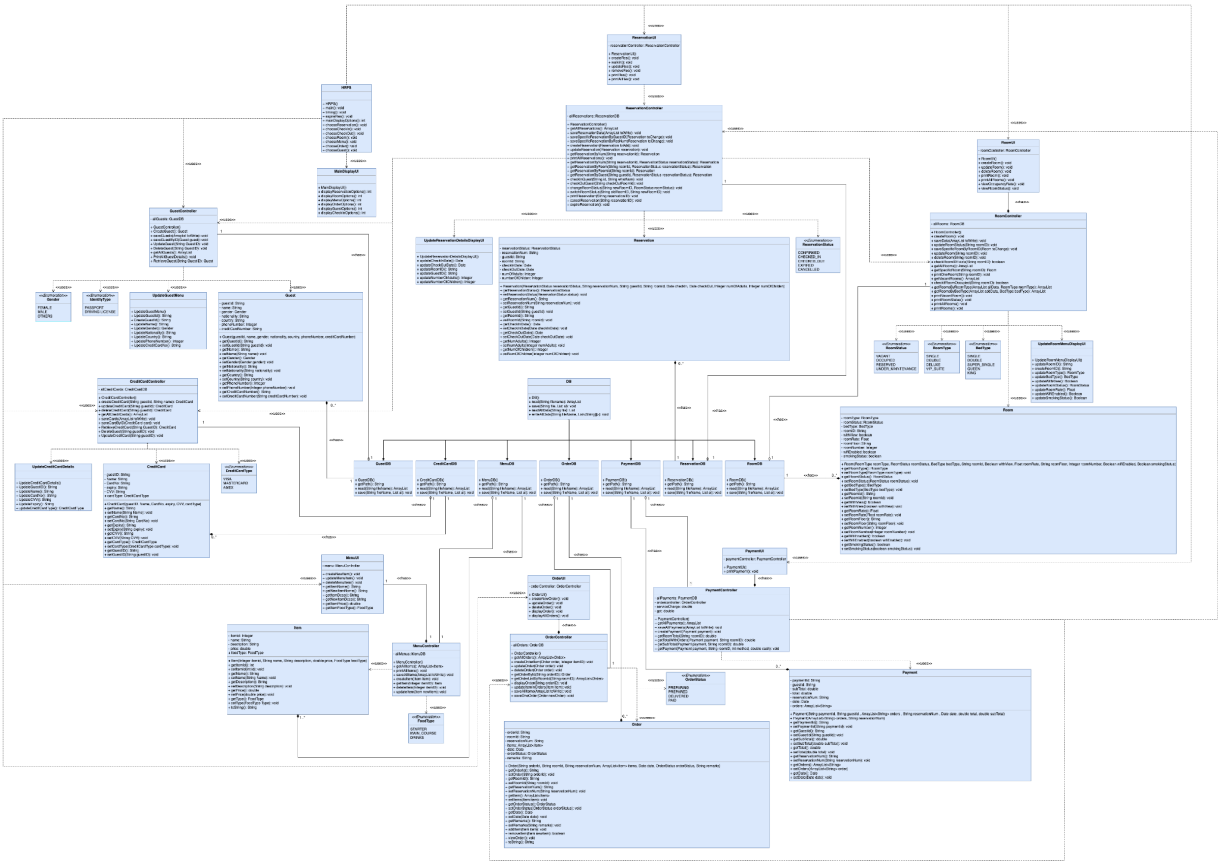
3.2.3. Polymorphism

For our database, every classes' read and save functions are overloaded in order to obtain the accurate fields from the database.



3.3. UML Class Diagram

The separate file is included outside of the report.



4. Feature Enhancement

Our system has been designed with the simplicity of extension for future features to be implemented, such as the following.

4.1. Function Room Booking System

The booking system could be extended to include the booking of the function rooms for large events like weddings and celebrations.

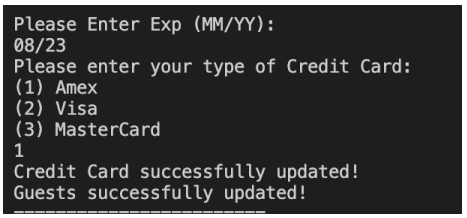
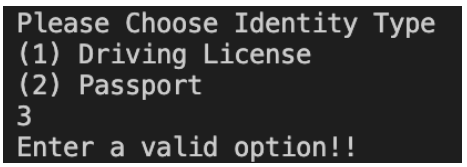
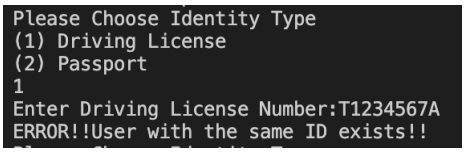
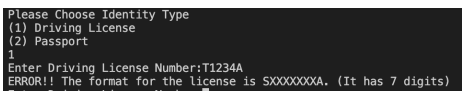
Our system would be able to handle an additional type of booking simply by adding an entirely new set of entities, while not requiring any changes to be done to the existing code. This adheres to the Open-Closed Principle.

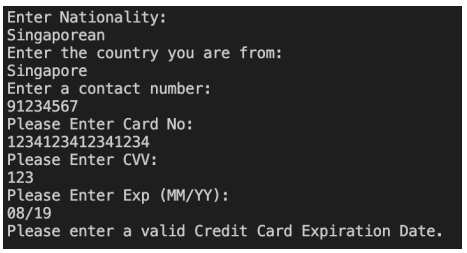
4.2. Staff Authorisation

There could be many roles that staff can hold, Manager, Supervisor, CEO, FrontDesk or just an Employee. Having an additional feature like this would give rise to more reason for the Staff to be split up into further smaller roles. Each role would have its own set of responsibilities that could be allocated respective rights with the help of the system. Hence, this idea follows the Single Responsibility Principle.

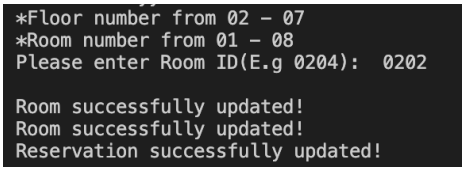
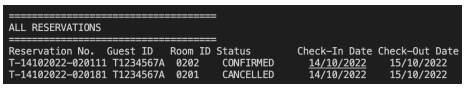
5. Test Cases

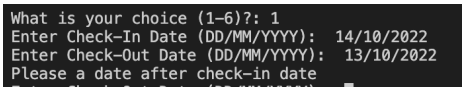
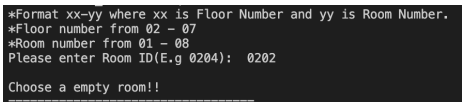
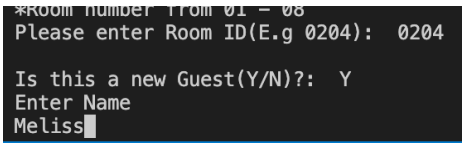
5.1. Guest Creation

	Test Scenario	Expected Results	Actual Outcome
1.	Staff inputs guest details, with all correct values into the fields.	Guest is successfully created Staff will be shown a success message Guest details will be added to database	
2.	Staff inputs guest details, with invalid choices	Staff will be shown an error. Staff will be prompted to re-enter the respective value.	
3.	Staff inputs guest details, with duplicate identity number	Staff will be shown an error. Staff will be prompted to retrieve existing guest records	
4.	Staff inputs guest details, with invalid	Staff will be shown an error Staff will be prompted to re-enter identity number	

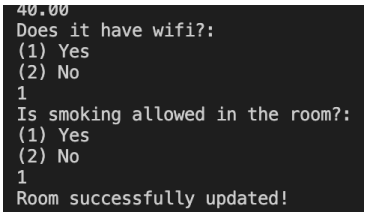
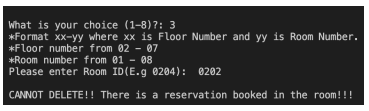
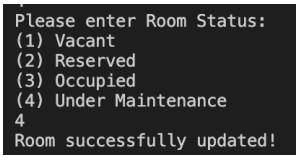
	identity number		
5.	Staff inputs guest details, with an expired credit card	Staff will be shown error Staff will be prompted to re-enter credit card information	

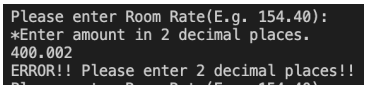
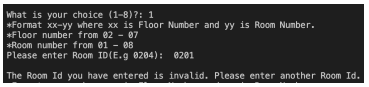
5.2. Reservation System

	Test Scenario	Expected Results	Actual Outcome
1.	Staff inputs reservation details, with all correct values into the fields.	Reservation is successfully created Staff will be shown a success message Reservation details will be added to database	
2.	Checking for expired reservations upon **runtime**	Expired reservations will be removed	Expired reservations are removed
3.	Staff updates a reservation room	Staff will be shown a success message Reservation details will be updated to database	
4.	Staff removes a reservation	Staff will be shown a success message Reservation details will be updated from CONFIRMED to CANCELLED	

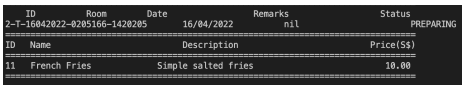
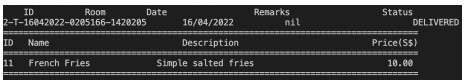
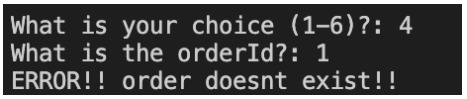
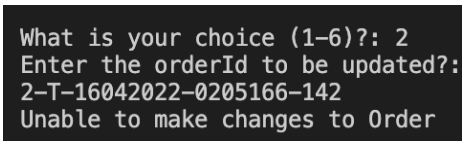
5.	Staff enters in a check-out date before check-in	Staff will be shown an error Staff will be prompted to re-enter check-out date	
6.	Staff tries to reserve a reserved room	Staff will be shown an error Staff will be prompted to re-enter room no.	
7.	Staff tries to create a reservation for a new guest	Staff will be shown that the guest is not found. Then continues with guest creation.	

5.3. Room Management

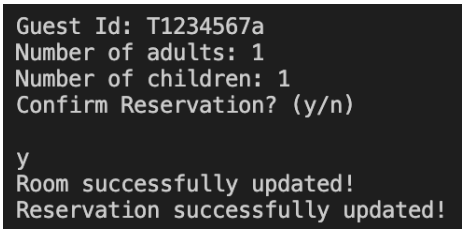
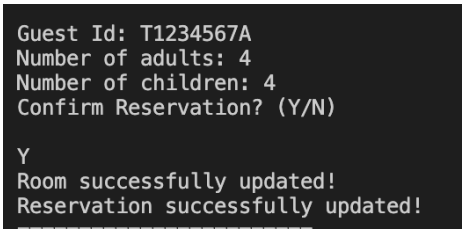
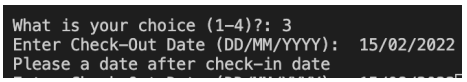
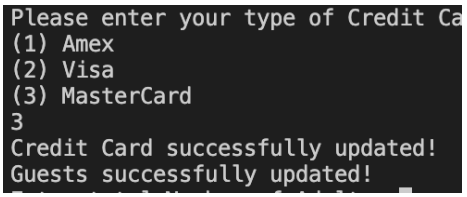
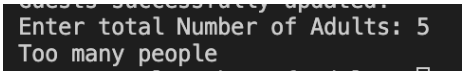
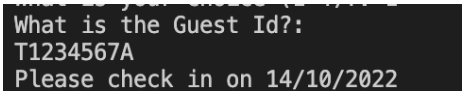
	Test Scenario	Expected Results	Actual Outcome
1.	Staff creates a room	Staff will be shown a success message Room details will be updated to database	
2.	Staff removes a reserved room	Staff will be shown an error Staff will be prompted to re-enter room no.	
3.	Staff updates a room status	Room details will be updated from VACANT to UNDER_MAINTENANCE	
4.	Staff prints all available rooms	Staff will be shown the list	List shown
5.	Staff prints all available rooms by room status	Staff will be shown the list	List shown

6.	Staff enters an invalid room rate	Staff will be shown an error Staff will be prompted to re-enter room rate	
7.	Staff enters an existing room id	Staff will be shown an error Staff will be prompted to re-enter room id	

5.4. Room Service Ordering System

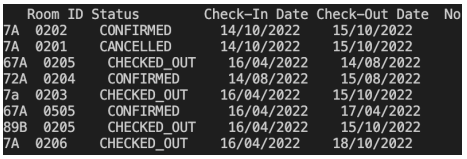
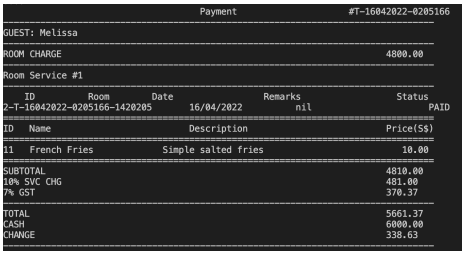
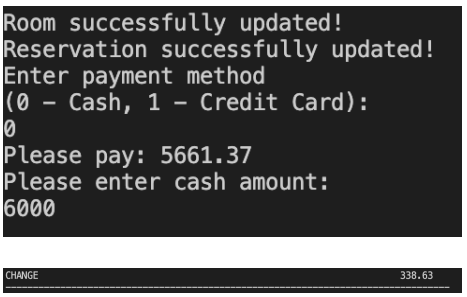
	Test Scenario	Expected Results	Actual Outcome
1.	Staff creates an order	Staff will be shown a success message Order details will be saved to database	
2.	Staff updates an order status	Staff will be shown a success message Order details will be updated to database	
3.	Staff views an invalid order id	Staff will be shown an error Staff will be prompted to re-enter order id	
4.	Staff views all orders	Only those not delivered will show	Only PREPARING / PREPARED orders were shown
5.	Staff tries to make changes to a DELIVERED order	Staff will be shown an error.	

5.5. Check-In Process

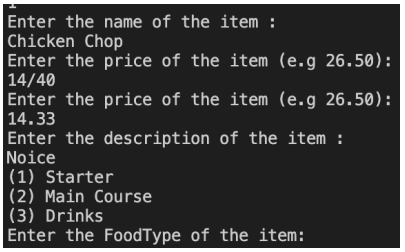
	Test Scenario	Expected Results	Actual Outcome
1.	Staff checks-in a walk-in guest	Staff will be shown a success message Details will be saved to database	
2.	Staff checks-in a reserved guest	Staff will be shown a success message Details will be saved to database	
3.	Staff enters invalid check-out date	Staff will be shown an error Staff will be prompted to re-enter check-out date.	
4.	Staff creates a Reservation & Guest through Check-In function	Staff will be shown a success message Guest & Reservation details will be saved to database	
5.	Staff enters too many adults staying	Staff will be shown an error Staff will be prompted to re-enter number	
6.	Staff checks in Guest before reserved check-in date	Staff will be shown an error	

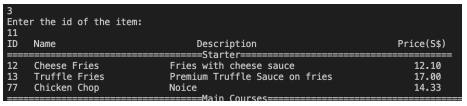
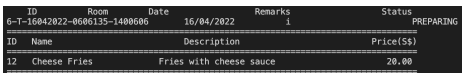
5.6. Check-Out & Payment System

	Test Scenario	Expected Results	Actual Outcome

1.	Staff checks-out a guest	Staff will be shown a success message Room status is VACANT Details will be updated to database	 Bill is also shown
2.	Staff checks-out a guest	Reservation status is changed to CHECKED_OUT	
3.	Staff checks-out guest with room service orders	Staff will be shown a success message Bill is updated with additional order items	
4.	Staff checks-out guest, pays by cash	Staff will be shown a success message	
5.	Staff checks out an unoccupied room	Staff will be shown an error that the room is not occupied	

5.7. Menu Management

	Test Scenario	Expected Results	Actual Outcome
1.	Staff creates a menu item	Staff will be shown a success message New item will be added to menu	

2.	Staff updates menu item price then guest orders it	Guest bill will reflect the updated price	
3.	Staff removes a menu item	Item is removed from the database as well	
4.	Staff views menu list	Menu list shows	Menu list is shown
5.	Staff removes menu item after guest orders it	Guest bill will still reflect the ordered item Menu is deleted from the menu	

6. Reflection

Importance of planning

In order to have a structured way of coding, a lot of time and effort has to be put into designing for the use cases and process flow first. We realised that instead of jumping straight into the class diagram, we had to re-read the assignment brief a few times and break down the entities before we started working on it.

Keeping the big picture in mind

Having to think ahead and see if it made sense in the codes. Otherwise, if there was a change to be made when coding, the same change would have to be reflected in the class diagram. That is not good practice, the process flow should not go backwards like that.

Boundaries need to be set

The assignment brief is not extremely detailed to guide the teams on everything we should think of. Leaving us to experiment and realise for ourselves what makes sense and what doesn't. We realise that because the different entities are so interconnected, there need to be assumptions set to draw the bounds of our system functionalities.