CZ2002 OBJECT-ORIENTED DESIGN AND PROGRAMMING

AY21/22 Semester 2 Group Assignment

Lab Group: **SS3**

Group Members		
S/N	**Matriculation No.**	**Student Name**
1.	U2023105H	Agarwal Anusha
2.	U2022616A	An Ruyi
3.	U2022486K	Chen Yiting
4.	U2023560G	Peng Wenxyan
5.	U2020430G	Hoang Minh Trung

Date of Submission: 17th April 2022

SCHOOL OF COMPUTER SCIENCE AND 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 honored the principles of academic integrity and have upheld the 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.

Course (CE2002 or CZ2002)	Lab Group	Signature /Date	Name
CZ2002	SS2	Anusha 17/4/22	Agarwal Anusha
		17-Apr-2022	An Ruyi
		陈一婷 17-Apr-2022	Chen Yiting
		彭文琰 17-Apr-2022	Peng Wenxyan
		17/04/2022	Hoang Minh Trung

Important notes:

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

1. Design Considerations

Hotel Reservation and Payment System (HRPS) is an application designed for hotel administrators to manage reservations of hotel rooms and orders of room service. The application covers the key features such as making hotel reservations, recording orders, and viewing records.

1.1 Object-Oriented Principles

1.1.1. Abstraction

Abstraction is the process of denoting distinguished characteristics and behaviours of entities to provide a conceptual boundary to viewer. It is applied throughout all classes. For example, in *Room* class, key attributes like room number, room type and rate, along with dynamic attributes, like the room status are identified and defined. Behaviour of *Room* objects are also defined by its operations like *setRoomStatus()*.

1.1.2. Encapsulation

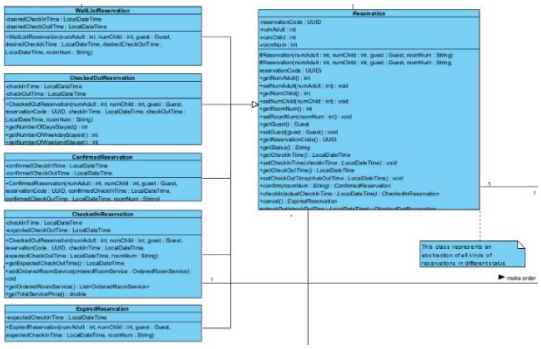
Encapsulation builds a barrier to protect private data of a class and maintain its visibility, ensuring attributes are only accessible to outside by getters and setters.

In our HRPS application, all attributes are made private to achieve encapsulation. To further protect data, attributes are made final whenever possible. Many methods are also made private as they should only be called within the class. This ensures that a class provides necessary interface while not revealing too many details.

1.1.3. Inheritance

Inheritance allows sub-classes to reuse attributes and functions from a super-class.

In our application, *PendingReservation*, *ConfirmedReservation*, *CheckedInReservation*, *CheckedOutReservation*, and *Expired-Reservation* all inherit from the *Reservation* superclass and reuse its code. For instance, the *toString()* method of Reservation is used in all subclasses to generate detailed report(receipt) of an reservation. In the method, we can further distinguish subclasses via *instanceof* operator.



1.1.4. Polymorphism

Polymorphism is the ability of an object to take many forms, while allowing same method to be called.

In our application, *ReservationController* declares and maintains an *ArrayList<Reservation>*, when at run-time, sub-classes of Reservation are instantiated in place of the abstract object. Behaviours of subclasses follow its self-defined methods.

1.1.5. Exception Handling

Exception Handling is used to handle runtime errors such as invalid input, duplication, null pointers, etc.

```java
public class InvalidPriceException extends HRPSException {
    private static final String NEGATIVE_PRICE_MESSAGE = "Price cannot be negative";

    public InvalidPriceException() {
        super(NEGATIVE_PRICE_MESSAGE);
    }
}
```

```java
public class CheckOutUI {
    void run() {
        try {
            ...
        } catch (HRPSException) {
            System.out.println("Warning a detected HRPS exception is caught" + e.getMessage());
        }
    }
}
```

In our application, all self-defined exceptions inherit from *HRPSException*. Therefore, we can throw various specific type of exceptions when creating methods, but only need to catch the general *HRPSException* at the top layer. If an error is encountered, messages printed will follow the ones specified in the corresponding subclasses. For example, if user inputs negative price, exception of *InvalidPriceException* will be raised, and captured in one catch block as HRPSException. This facilitates error capture, display, and new error introduction in future, adding to the maintainability of the program.

1.2. Design Patterns

1.2.1. Singleton Design

Certain classes should have no more than just one instance, like controller class of system and UI of the system. Singleton Design Pattern limits the class so that whoever is using that class can use the one and only class instance in the program scope.

In our application, all controller classes such as *OrderController*, *GuestController*, *RoomController* and boundary classes such as *ServiceUI*, *GuestUI*, *RoomUI*, are implemented using the Singleton Pattern. For example, in *OrderController* class, we have *getInstance()*.

```java
public class OrderController {
    private static OrderController instance = null;
    ...

    public static OrderController getInstance() {
        if (Objects.isNull(instance)) {
            OrderController = new OrderController();
        }
        return OrderController;
    }
}
```

And when we need to have an *OrderController* object, we can just call the static method of *OrderController.getInstance()* to get the instance.

```java
public class ServiceUI {
    private final OrderController orderController = OrderController.getInstance();
    ...
}
```

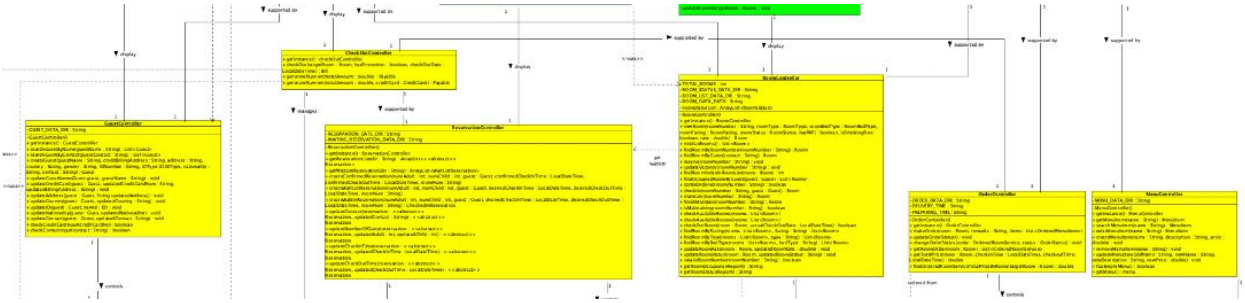
1.2.2. Entity-Control-Boundary (ECB) Design Pattern

The ECB pattern organises the classes according to their role in the use-case realization. The pattern separating classes into three layers (in 3 packages): **entity**, which contains the domain objects; **control**, which encapsulates the business logic and the business rules; **boundary**, which handles user interactions. This adds to maintainability of the programme as whenever there is a bug arises, the user can locate the class file in a certain package from the bug type (a display bug, a logic bug or an entity logic bug).

As shown in UML diagram of boundary below, the boundary classes are split into UI classes to deal with different controllers. For example, *GuestUI* class is designed specifically on guest-related issue, so that in the case of changing UI related to guest operations, we would only need to change this class.

The control classes are split into controllers to deal with specific objects. For example, *ReservationController* deals with Reservation object only. In *ReservationUI*, if we want to change status of a pending reservation, the order of call will be:

1. *ReservationUI*: ask user to "confirm" a pending reservation (choice = 1) or "cancel" it (choice = 2). Call *reservationController.WaitingReservation(reservation,1)* if guest wishes to confirm.
2. *ReservationController*: upon calling, execute *WaitListReservation(reservation,1)*, and call *reservation.confirm(roomNum)* of *WaitListReservation*
3. *WaitListReservation*: execute *confirm(roomNum)*



Finally, domain objects are split into separate entity classes. In this way, we ensure that all classes in our implementation carry their own responsibility, reducing the coupling between classes.

1.3. SOLID Approach

1.3.1. Single-Responsibility Principle (SRP)

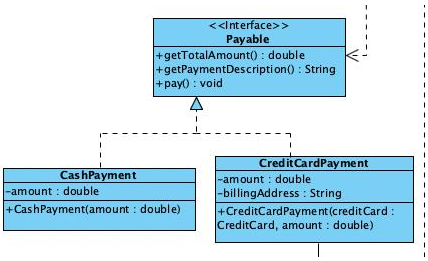
The Single Responsibility Principle states that a class should only carry one responsibility. In our application, SRP is achieved by splitting classes into ECB roles and as mentioned above. For instance, under boundary, each UI class controls only one specific type of interaction. Under control, each class represents a controller to a specific domain object. This helps programmer to easily locate relevant classes if a change needs to be made.

1.3.2. Open-Closed Principle (OCP)

OCP states module should be open for extension but closed for modification. Under the OCP, a module should be able to change what it does, without changing the source code. An example in our application

would be enumeration of room status. If we need to add a new room status, we can simply add new string literal to the enumeration class of room status without having to change other part of code as we iterate over all possible room status *[for (RoomStatus status : RoomStatus.values())]* in generating room occupancy report.

Another example would be payable interface: declare all payment methods that need to be implement in a payment. With the interface, future upgrades of new payment methods like PayLah can be easily supported. We do not have to change the existing code on payment in *CheckOutController* and in *CheckOutUI* that deals with the payment, hence allowing easy extensibility.



1.3.3. Liskov Substitution Principle (LSP)

LSP states that derived classes must be substitutable for their base classes.

We look again at the Reservation example. All concrete classes are substitutable by the general abstract *Reservation* class and can be stored in an ArrayList in *ReservationController*, where they are treated as *Reservation* objects and can be used to call *Reservation*'s method like *getCheckInTime()* and *getCheckOutTime()*. This allows easy manipulation in control class without having to overload a method for each subclass. It increases code reusability by allowing repeated use of code fragment that deals with general *Reservation* class only. Moreover, it increases extensibility, since if another subclass of *Reservation* is introduced, we do not need to write additional method to manipulate it.

```
public class ReservationController {
    private ArrayList<Reservation> reservations;
}
```

1.3.4. Interface Segregation Principle (ISP)

Interface Segregation principle states that classes should not depends on interfaces they do not need. We apply ISP by having succinct and interface-specific methods in our interface design.

For instance, in payable interface, pay*()*, *getTotalAmount()* and *getPaymentDescription()* are defined as all payments go under these three steps. all the classes implementing this interface will not suffer redundancy in method implementation. Easy extension of additional payment methods by simply override these succinct interface methods is also allowed by the interface, making the program more extensible.

```
public interface Payable {
    public double getTotalAmount();

    public String getPaymentDescription();

    public void pay();
}
```

1.3.5. Dependency Injection Principle (DIP)

DIP states that both high-level and low-level models should depend on abstractions.

Dependency injection: When we instantiate a *Reservation* object in the

```
public class Reservation {
    private Guest guest;
    ...

    public Reservation(Guest guest, ...) {
        this.guest = guest;
        ...
    }
}
```

constructor as shown in figure, we pass in a reference of guest instead of creating a guest in the constructor. And for all operations in *Reservation* that require guest details, we will make use of the abstraction layer between two modules to perform guest-related operations.

In addition, in the high-level controllers, we use the abstraction layer between controller and lower-level entity to manipulate data on entity they control using mutator methods of entity.

```
public class GuestController {
    public void updateCreditCard(Guest guest, String updatedCreditCardNum, String updatedBillingAddress) {
        guest.setCreditCardDetails(updatedCreditCardNum, updatedBillingAddress);
    }
}
```

And we have created many abstract methods in *Reservation* for its sub-classes to override. This follows DIP as the super-class need not depend on the details of its sub-classes, but both depending on abstractions.

1.4. Proposed Future Features

The current HRPS application has been designed to support easy upgrade of methods and functions. We reap the benefits of adhering to OCP to easily add new features here, without modification to existing code.

1.4.1. Hotel Reservation Reminder Message

This feature is proposed in assignment guide. The feature can be easily realized by adding a method in *ConfirmedReservation* class and call this method at the creation of instance in the *ReservationController*. We can further retrieve guests' contact by the contact they have provided to send a SMS message to them.

```
public class ReservationController {
    public void createConfirmedReservation(...) {
        sendMessage(createConfirmedReservation().generateSMSMessage(), guest.getContact());
    }
}
```

```
public class ConfirmedReservation {
    ...
    public String generateSMSMessage() {
        return String.format("You have successfully booked a reservation, " +
                    "below is your reservation information:" +
                    "\nReservation = %s " +
                    "\nRoom = %s" +
                    "\nSupposedCheckInDate: %s " +
                    "\nSupposedCheckOutDate: %s", this, getRoomNum(), confirmedCheckInTime,
            confirmedCheckOutTime);
    }
}
```

1.4.2. Membership Point Feature

The feature allows hotels to reward guests with points proportional to their spending and promise future redemption (at least a day later) as discount. In this case, we will need to store additional attribute *points_earned* and a method *accumulatePoints*() in the Guest class. This will allow HRPS to record the points earned by guest at each history check-out in accumulation. At time of check-out, *accumulatePoints*() method will be called by passing in the relevant Guest.

```
public class Guest {
    private double points_earned;

    public void accumulatePoints(double points) {
        this.points_earned += points;
    }
}
```

1.4.3. Storing Past Reservation Information

Past reservation information can be stored by specifying a List object in the *ReservationController*. This list will store all the *ChekcedOutReservation* and *ExpiredReservation*. We would need to implement another Serializable reading and writing file to this list in the constructor. In addition, we can implement methods like *getPastReservation*() in *ReservationController* that manipulates the List and support it by *checkPastReservation*() method in *ReservationUI* to retrieve the past reservation information.

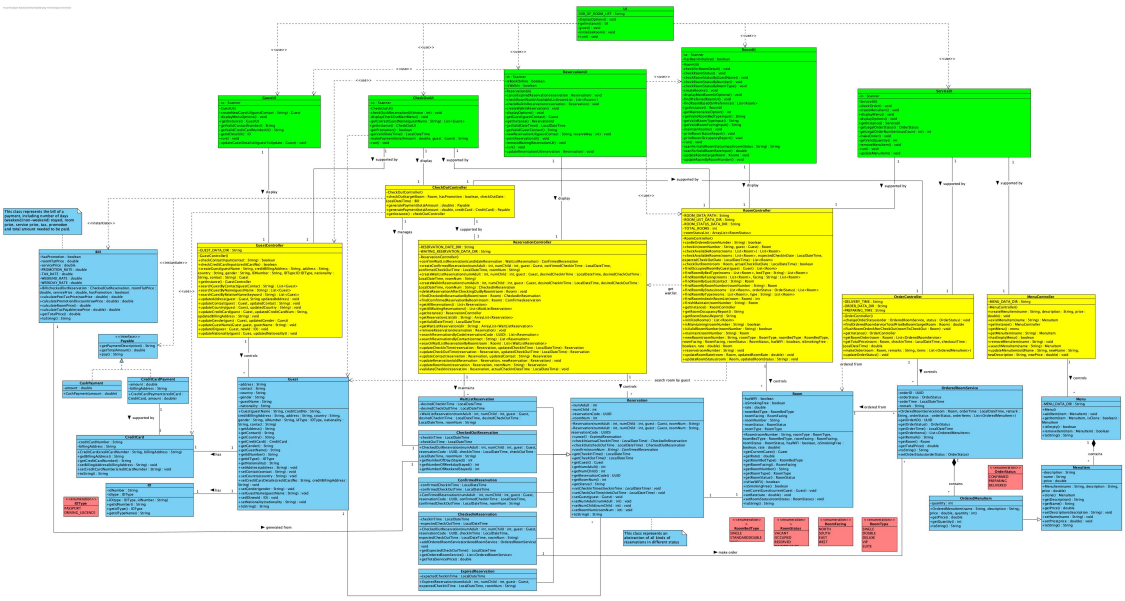
1.5. Assumptions

Below are the assumptions made for HRPS adding on to the ones specified in the manual:

- Guest:
  1. Guests' information is stored in the system, even if they have already checked out.
  2. No two guests have the same contact number.
  3. There can be unlimited number of adults and children in a reservation.
- Room:
  1. Room status change follows VACANT→RESERVED→OCCUPIED→VACANT, except for walk-in reservation, it is allowed to have VACANT→OCCUPIED→VACANT.
  2. Room under maintenance is not allowed to be reserved. To make the room available for reservation again, the user must manually finish the maintenance of that room.
- Reservation:
  1. All checked-out reservation and expired reservation will not be further used. However, we keep them in the system for tracking. (See "Our own proposed feature")
  2. If a Guest wishes to change the room number under a particular reservation, he must cancel the reservation and make a new one with the new room number.
  3. Time is static for reservations. Manual time checking is needed in following cases:
     a) Guest check-in or check-out. User key in actual check-in time to check if a confirmed reservation has expired (≥24 hours after expected check in time). If it has, the guest will not be able to check in under that reservation. User key in actual check-out time to compute room price based on days stayed.
     b) Dropping reservation on wait list, when its desired check-in time exceeds current time.
  4. Confirmed reservation is allowed to be cancelled before expected checked in time however checkedInReservation cannot be cancelled.

  Room Service -
  1. Orders cannot be made unless the Reservation is checked in.
  2. Once an order is made, it cannot be cancelled and are not refundable.
  3. Items in the same order has the same order status as they are processed together.
- Payment:
  1. Credit cards must be provided upon creating guest regardless of payment method guests uses.
  2. A 16-digit number is a valid credit card number.
  3. Payments done are not refundable.
  4. Promotion is 10% off. GST tax is 7% and applicable to the subtotal after promotion deductions.

2. UML Class Diagram for HRPS (clearer .png file is together with the source code)



3. Test Cases

3.1. Input Error Checking

1.	**Entering non-numerical choice** 1.Start HRPS application to enter Main Menu 2.On prompting for choice, type ‘c’	---------- 0. Exit 1. Guest Page 2. Reservation Page 3. Room Page 4. Service Page 5. Check out Page Your choice: c Invalid input. Please enter an integer only. Invalid choice!
2.	**Entering out-of-index choice** 1.Start HRPS application to enter Main Menu 2.On prompting for choice, type ‘100’	---------- 0. Exit 1. Guest Page 2. Reservation Page 3. Room Page 4. Service Page 5. Check out Page Your choice: 100 Invalid choice!
3.	**Entering Invalid Date** 1. Start HRPS application to enter Main Menu 2. On prompting for choice, input ‘2’ to go to Reservation Page 3. On Reservation Page, input ‘3’ to **create a new reservation**	1. On **entering expected check-in date,** type “2030-30-30 25:61” 2. No DateFormatParsing exception Enter expected check in time (yyyy-MM-dd HH:mm): 2030-30-30 25:61 Warning: Invalid date format!
4.	**Entering invalid credit card number** 1. Start HRPS application to enter Main Menu 2. On prompting for choice, input ‘1’ to go to Guest Page 3. On Guest Page, input ‘1’ to **create a new guest** 4. Pass previous input tests	1. On **entering credit card number**, type “1234” (not 16 digit) 2. Credit card number not allowed Please enter Guest Contact: 12345678 Please enter the following details for the new guest. Name: claire Address: nanyang house Living country: singapore Gender: (m/f/o) (o stands for Others)m Nationality: chinese Credit Card Number: 1234 Invalid Credit Card Number. Please re-enter Credit Card Number: 1234123412341234 Credit Card Billing Address: nanyang house

5.	**Entering Duplicated Menu Item**
	1. Start HRPS application to enter Main Menu
	2. On prompting for choice, input '4' to go to Service Page
	3. On Service Page, input '1' to display menu
	4. On Service Page, input '2' to **create a menu item**
		1. On **entering item name**, type an existed name, e.g. "Beef Lasagna"
		2. Throw UpdateMenuDuplicateMenuItemName exception (self-defined)
		`Enter the item name: Beef Lasagna`
		`Item already exists in menu!`

6.	**Entering Non-existed Menu Item**
	1. Start HRPS application to enter Main Menu
	2. On prompting for choice, input '4' to go to Service Page
	3. On Service Page, input '5' to **make new order**
	4. On **entering room number**, input a checked-in room, e.g. "03-01"
		1. On **entering item name**, type a non-existent item, e.g. "non existed"
		2. Throw MenuItemNotExisted exception (No NullPointer exception)
		`Please select the items to order... (enter "STOP" to`
		`Item: non existed`
		`Item named "non existed" does not exist!`

6.	**Entering Unavailable Room**
	1. Start HRPS application to enter Main Menu
	2. On prompting for choice, input '2' to go to Reservation Page
	3. On Reservation Page, input '6' to **make a walk-in reservation**
	4. Pass previous necessary input tests
	5. System generates a list of rooms that fits guest's requirements
		1. On **entering selection of room**, type a room that is not on the list
		2. Room not allowed to be checked in
		`Enter expected check in time (yyyy-MM-dd HH:mm): 2022-03-04 11:11`
		`Enter expected check out time (yyyy-MM-dd HH:mm): 2022-03-05 11:11`
		`These are the available rooms with the specified requirements:`
		`06-04`
		`Select a room:`
		`06-03`
		`Invalid roomNum! please select again:`
		`06-04`
		`Number Of Adults: 1`
		`Number Of Children: 0`
		`06-04 has been set to occupied for walk in guest, yiting!`

7.	**Entering Room Number of Wrong Format**
	1. Start HRPS application to enter Main Menu
	2. On prompting for choice, input '3' to go to Room Page
	3. On Room Page, input '4' to get room details
		1. On entering room number, type "0201"
		2. Room input with wrong format not allowed pass.
		`Enter room number:`
		`0201`
		`Warning: Invalid room number format, please enter xx-xx, eg. 02-01`

8.	**Entering Non-existed room for update room details**
	1. Start HRPS application to enter Main Menu
	2. On prompting for choice, input '3' to go to Room Page
	3. On Room Page, input '2' to update room details
		1. On entering room number, put a non-exist room e.g. "10-01"
		2. No NullPointer Exception
		`Enter room number:`
		`10-01`
		`Warning: The room number you have keyed in does not exists.`

3.2. Functionality Checking

Main functions and assumptions are tested under following scenarios:

Update a guest's credit card From Main UI: 1. Select 1. Guest Page 2. Select 2. Update guest details	Please enter Guest Contact: *1234* Similar records are present in the system: +----------+----------+ | Guest name | Trung | +----------+----------+ | Credit card No | 1234567812345678 | +----------+----------+ | Billing Address | ntu | +----------+----------+ | Address | ntu | +----------+----------+ | Country | sg | +----------+----------+ | Gender | Male | +----------+----------+ | Passport | 1234567 | +----------+----------+ | Nationality | vn | +----------+----------+ | Contact | 1234 | +----------+----------+ Is this the guest whom you are looking for? 1. Yes 2. No Your choice: *1* Please enter the information that you want to update: 1. Name 2. Address 3. Contact 4. Credit Card 5. ID 6. Gender 7. Nationality 0. Cancel *4* Please enter the information to be updated: Guest Credit Card: Credit Card Number: *1234567887654321* Credit card billing address: *nus* Guest details updated: +----------+----------+ | Guest name | Trung | +----------+----------+ | Credit card No | 1234567887654321 | +----------+----------+ | Billing Address | nus | +----------+----------+ | Address | ntu | +----------+----------+ | Country | sg | +----------+----------+ | Gender | Male | +----------+----------+ | Passport | 1234567 | +----------+----------+ | Nationality | vn | +----------+----------+ | Contact | 1234 | +----------+----------+ No similar records found. Exiting
Search a guest by name From Main UI: 1. Select 1. Guest Page 2. Select 4. Find a guest by name	Please enter a keyword to search for guest's name: *Trung* Guest with naming containing Trung: 1. +----------+----------+ | Guest name | Trung | +----------+----------+ | Credit card No | 1234567887654321 | +----------+----------+ | Billing Address | nus | +----------+----------+ | Address | ntu | +----------+----------+ | Country | sg | +----------+----------+ | Gender | Male | +----------+----------+ | Passport | 1234567 | +----------+----------+ | Nationality | vn | +----------+----------+ | Contact | 1234 | +----------+----------+ Please enter a keyword to search for guest's name: *trung* Guest with naming containing trung: 1. +----------+----------+ | Guest name | Trung | +----------+----------+ | Credit card No | 1234567887654321 | +----------+----------+ | Billing Address | nus | +----------+----------+ | Address | ntu | +----------+----------+ | Country | sg | +----------+----------+ | Gender | Male | +----------+----------+ | Passport | 1234567 | +----------+----------+ | Nationality | vn | +----------+----------+ | Contact | 1234 | +----------+----------+
Create a new guest From Main UI: 1. Select 1. Guest Page 2. Select 1. Create a new guest	Please enter Guest Contact: *1234* Please enter the following details for the new guest. Name: *Trung* Address: *ntu* Living country: *sg* Gender: (m/f/o) (o stands for Others)*m* Nationality: *vn* Credit Card Number: *1234567812345678* Credit Card Billing Address: *ntu* ID Type: (1. Passport, 2. Driving License) *1* Passport Number: *1234567* New guest has been added to the system: +----------+----------+ | Guest name | Trung | +----------+----------+ | Credit card No | 1234567812345678 | +----------+----------+ | Billing Address | ntu | +----------+----------+ | Address | ntu | +----------+----------+ | Country | sg | +----------+----------+ | Gender | Male | +----------+----------+ | Passport | 1234567 | +----------+----------+ | Nationality | vn | +----------+----------+ | Contact | 1234 | +----------+----------+

a. Update/Search/Create guests' detail (*Search by name using keyword/s*)

b. Create/Update/Remove/Print reservation

Update a guest's credit card

From Main UI:

3. Select 1. Guest Page
4. Select 2. Update guest details

```
Please enter Guest Contact: 1234
Similar records are present in the system:
---------------------+----------------------
| Guest name         |  Trung               |
---------------------+----------------------
| Credit card No     |  1234567812345678    |
---------------------+----------------------
| Billing Address    |  ntu                 |
---------------------+----------------------
| Address            |  ntu                 |
---------------------+----------------------
| Country            |  sg                  |
---------------------+----------------------
| Gender             |  Male                |
---------------------+----------------------
| Passport           |  1234567             |
---------------------+----------------------
| Nationality        |  vn                  |
---------------------+----------------------
| Contact            |  1234                |
---------------------+----------------------
```

```
this the guest whom you are looking for?
Yes
No
ur choice: 1
ase enter the information that you want to update:
Name
Address
Contact
Credit Card
ID
Gender
Nationality
Cancel

ase enter the information to be updated:
st Credit Card: Credit Card Number: 1234567887654321
edit card billing address: nus
```

```
Guest details updated:
---------------------+----------------------
| Guest name         |  Trung               |
---------------------+----------------------
| Credit card No     |  1234567887654321    |
---------------------+----------------------
| Billing Address    |  nus                 |
---------------------+----------------------
| Address            |  ntu                 |
---------------------+----------------------
| Country            |  sg                  |
---------------------+----------------------
| Gender             |  Male                |
---------------------+----------------------
| Passport           |  1234567             |
---------------------+----------------------
| Nationality        |  vn                  |
---------------------+----------------------
| Contact            |  1234                |
---------------------+----------------------

No similar records found. Exiting
```

Search a guest by name

From Main UI:

3. Select 1. Guest Page
4. Select 4. Find a guest by name

```
Please enter a keyword to search for guest's name: Trung
Guest with naming containing Trung:
1.
---------------------+----------------------
| Guest name         |  Trung               |
---------------------+----------------------
| Credit card No     |  1234567887654321    |
---------------------+----------------------
| Billing Address    |  nus                 |
---------------------+----------------------
| Address            |  ntu                 |
---------------------+----------------------
| Country            |  sg                  |
---------------------+----------------------
| Gender             |  Male                |
---------------------+----------------------
| Passport           |  1234567             |
---------------------+----------------------
| Nationality        |  vn                  |
---------------------+----------------------
| Contact            |  1234                |
---------------------+----------------------
```

```
Please enter a keyword to search for guest's name: trung
Guest with naming containing trung:
1.
---------------------+----------------------
| Guest name         |  Trung               |
---------------------+----------------------
| Credit card No     |  1234567887654321    |
---------------------+----------------------
| Billing Address    |  nus                 |
---------------------+----------------------
| Address            |  ntu                 |
---------------------+----------------------
| Country            |  sg                  |
---------------------+----------------------
| Gender             |  Male                |
---------------------+----------------------
| Passport           |  1234567             |
---------------------+----------------------
| Nationality        |  vn                  |
---------------------+----------------------
| Contact            |  1234                |
---------------------+----------------------
```

Create Confirmed Reservation

1. Input '2' to go to Reservation Page
2. Create a guest if a guest not created yet
3. Create a reservation in future with preferred room type
4. Print receipt of reservation

```
02-01 has been reserved for Trung!
New reservation added to the system:
----------------------------+----------------------
| Room number               |  02-01               |
----------------------------+----------------------
| Reservation status        |  Confirmed           |
----------------------------+----------------------
| Guest Name                |  Trung               |
----------------------------+----------------------
| Guest Contact             |  1234                |
----------------------------+----------------------
| Number of Adult           |  1                   |
----------------------------+----------------------
| Number of children        |  0                   |
----------------------------+----------------------
| Expected Check-in time    |  2022-04-18T12:00    |
----------------------------+----------------------
| Expected Check-out time   |  2022-04-20T12:00    |
----------------------------+----------------------
```

```
Please enter Guest Contact: 1234
This is your first time to make a reservation!
Guest: Trung
Please choose the preferred room type (Single, Double, Deluxe, VIP, Suite):
single
Please choose the preferred bed type (Single, Double, Master)
single
Please choose the preferred facing (North, South, East, West):
north
Enter expected check in time (yyyy-MM-dd HH:mm): 2022-04-18 12:00
Enter expected check out time (yyyy-MM-dd HH:mm): 2022-04-20 12:00
These are the available rooms with the specified requirements:
| Room number | Flat Rate | Wifi Available | Smoking Free |
|    02-01    |  $ 90.00  |       No       |       No     |
|    03-01    |  $120.00  |       Yes      |       Yes    |
|    04-01    |  $120.00  |       Yes      |       Yes    |
|    05-01    |  $120.00  |       Yes      |       Yes    |
Select a room:
02-01
Number Of Adults: 1
Number Of Children: 0
```

Update a reservation's guest ontact

From Main UI:

1. Select 2 Reservation Page
2. Select 2 Update reservation details

```
Reservation details updated:
----------------------------+----------------------
| Room number               |  02-01               |
----------------------------+----------------------
| Reservation status        |  Confirmed           |
----------------------------+----------------------
| Guest Name                |  Trung               |
----------------------------+----------------------
| Guest Contact             |  4567                |
----------------------------+----------------------
| Number of Adult           |  1                   |
----------------------------+----------------------
| Number of children        |  0                   |
----------------------------+----------------------
| Expected Check-in time    |  2022-04-18T12:00    |
----------------------------+----------------------
| Expected Check-out time   |  2022-04-20T12:00    |
----------------------------+----------------------
```

```
Is this the reservation you are looking for?
1. Yes
2. No
Your choice:
1
-----------------------------
Please enter the information that you want to update:
1. Guest Contact
2. Check In time
3. Check Out time
4. Number Of Guests
0. Cancel
1
Please enter the updated information:
Guest Contact: Please enter Guest Contact: 4567
```

```
Please enter Guest Contact: 1234
Records found!
----------------------------+----------------------
| Room number               |  02-01               |
----------------------------+----------------------
| Reservation status        |  Confirmed           |
----------------------------+----------------------
| Guest Name                |  Trung               |
----------------------------+----------------------
| Guest Contact             |  1234                |
----------------------------+----------------------
| Number of Adult           |  1                   |
----------------------------+----------------------
| Number of children        |  0                   |
----------------------------+----------------------
| Expected Check-in time    |  2022-04-18T12:00    |
----------------------------+----------------------
| Expected Check-out time   |  2022-04-20T12:00    |
----------------------------+----------------------
```

Remove a reservation From Main UI: 1. Select 2 Reservation Page 2. Select 3 Cancel reservation	Please enter Guest Contact: 4567 Records found! | Room number | 02-01 | | Reservation status | Confirmed | | Guest Name | Trung | | Guest Contact | 4567 | | Number of Adult | 1 | | Number of children | 0 | | Expected Check-in time | 2022-04-18T12:00 | | Expected Check-out time | 2022-04-20T12:00 | Is this the reservation you are looking for? 1. Yes 2. No Your choice: 1 Room 02-01 has been set to available.
Print all reservations From Main UI: 1. Select 2 Reservation Page 2. Select 7 Print all reservations	Printing all reservations in the system record... | Room number | 02-01 | | Reservation status | Confirmed | | Guest Name | Trung | | Guest Contact | 4567 | | Number of Adult | 1 | | Number of children | 1 | | Expected Check-in time | 2022-04-18T12:20 | | Expected Check-out time | 2022-04-20T12:00 |

3.3. Negative Case Checking

Prevent Reserved Rooms from Being Set to Maintenance 1. After reservation, go to Room page and input '6' to maintain that room. 2. Room is not allowed to be set under maintenance	Your choice: 6 0. Back 1. Maintain room 2. Finish maintaining room Please enter your choice: 1 Please enter the room number you want to maintain 03-01 Room 03-01 is currently RESERVED and cannot be maintained.
Prevent checking in to a maintaining room (Assumption Room 3) 1. Maintain room 02-01 at Room Page 2. At Reservation Page, make a walk-in reservation with the same preference to 02-01. 3. We can see that 02-01 is not on the list of available room for selections even if it is not occupied nor reserved.	0. Back 1. Maintain room 2. Finish maintaining room Please enter your choice: 1 Please enter the room number you want to maintain 02-01 Room 02-01 has been maintained. 8. Room status Report Your choice: 8 Here is the room status report: Room Report By Current Status: VACANT : 02-02, 02-03, 02-04, 02-05, 02-06, 02-07, 02-08, 03-02, 03-03, 03-04, 03-05, 03-06, 03-07, 03-08, 04-02, 04-03, 04-04, 04-05, 04-06, 04-07, 04-08, 05-01, 05-02, 05-03, 05-04, 05-05, 05-06, 05-07, 05-08, 06-01, 06-02, 06-03, 06-05, 06-06, 06-07, 06-08, 07-01, 07-02, 07-03, 07-04, 07-05, 07-06, 07-07, 07-08 OCCUPIED : 04-01, 06-04 RESERVED : 03-01 MAINTENANCE : 02-01 6. Walk-in reservation 7. Print all reservations Your choice: 6 Please enter Guest Contact: 12345678 Guest: claire Please choose the preferred room type (Single, Double, Deluxe, VIP, Suite): single Please choose the preferred bed type (Single, StandardDouble, King) single Please choose the preferred facing (North, South, East, West): north Enter expected check in time (yyyy-MM-dd HH:mm): 2022-06-01 Invalid date format! Please enter again (in yyyy-MM-dd HH:mm): 2022-06-01 12:00 Enter expected check out time (yyyy-MM-dd HH:mm): 2022-06-02 12:00 These are the available rooms with the specified requirements: 03-01 04-01 05-01
Prevent non-booked rooms from ordering menu items (Assumption: Room Service 1) 1. Start HRPS application to enter Main Menu	1. On **entering the room**, type a non-checked-in room, e.g. "02-01" 2. The room is not allowed to place orders

2. On prompting for choice, input ‘4’ to go to Service Page 3. On Service Page, input ‘5’ to **make an order**	5. Make an order 6. Check or update order status 5 Enter room number to make order: 02-01 This room has no guest and cannot order service now!
Prevent cancelling checked in reservation (Assumption: Reservation 4) 1. On Reservation page, input ‘3’ to cancel a reservation by made by Guest with contact 12341234.	This reservation is found to be “Checked in” and cannot be cancelled. Please enter Guest Contact: 12341234 Records found! ----------+---------- | Room number | 02-01 | ----------+---------- | Reservation status | Checked in | ----------+---------- | Guest Name | Lily | ----------+---------- | Guest Contact | 12341234 | ----------+---------- | Number of Adult | 1 | ----------+---------- | Number of children | 0 | ----------+---------- | Actual Check-in time | 2022-04-28T12:00 | ----------+---------- | Expected Check-out time | 2022-04-30T12:00 | ----------+---------- Is this the reservation you are looking for? 1. Yes 2. No Your choice: 1 Warning: Sorry, this reservation has been checked in , you cannot cancel it!Exiting..

4. Reflection

We had encountered difficulties in designing the program structure as this is our first-time applying OO design in a relatively large application. Much effort was put into visualizing and applying inheritance, polymorphism and abstraction to the class design. However, we finally managed to abide by many OO design principles and patterns as well as the SOLID design principle in this program.

We are amazed at how abstraction helps to deal with complex issues. We apply abstraction to lower-level classes of entities, and build up abstraction progressively to ensure problems are dealt with at the right level, and not brought to higher levels. We also appreciated the magic of OO design in reducing the coupling of program. Low level of coupling can contribute to easy maintenance and integration. We learn that interfaces and abstract method can significantly lower the level of coupling. In addition, we realize adhering to the SOLID principle will not only make a satisfactory program design, but also helps us to gain a clearer scope and better reuse, extend the existing work without much changes. Through this project, we peeked into how a well-defined program that is easy for reuse, extension and maintenance is built, and various techniques are learned. In conclusion, it has been a journey full of insights and gains.