**Ophelia’s Oasis in the Amlet Desert**

**CSC 470 Software Design and Engineering**

*Javier Hernandezocasio, Nathaniel Spindler, Philip Whiting*

**Index of Contents**

Project Requirements – *page 2*

Business Overview – *page 3*

Domain Modeling – *page 4*

Domain Modeling Review Checklist - *page 5*

Architecture Design – *page 6*

Use Case Analysis and Diagrams – *page 7*

Team Member Contributions and Log – *page 11*

**OOAD - Project Requirements for Hotel Management System (HMS)**

**Functional Requirements**

1. The system shall provide staff with reservation handling for customers.
2. The system shall allow management to set the base rate.
3. The system shall allow management to change the base rate.
4. The system shall allow staff to make reservations.
   1. The system shall allow staff to make *prepaid reservations*.
   2. The system shall allow staff to make *60-days in advance reservations*.
   3. The system shall allow staff to make *conventional reservations*.
   4. The system shall allow staff to make *incentive reservations*.
5. The system shall allow staff to cancel reservations.
6. The system shall allow staff to change reservations.
7. The system shall allow staff to check in guests.
8. The system shall allow staff to check out guests.
   1. The system shall allow staff to print *accommodation bills*.
9. The system shall allow staff to accept payments from guests.
10. The system shall allow staff to classify a guest as a no show.
11. The system shall allow staff to make a backup copy of all files.
12. The system shall allow staff to print reports.
    1. The system shall allow management to print *expected occupancy reports.*
    2. The system shall allow management to print *expected room income reports.*
    3. The system shall allow management to print *incentive reports.*
    4. The system shall allow employees to print *daily arrivals reports.*
    5. The system shall allow management to print *daily occupancy reports.*

**Non-functional Requirements**

1. Reservation information must be stored securely.
2. The system must ensure it does not overbook.
3. The system must ensure secure payment transaction.
4. The system must have some default cost specified if no manually cost is set.
5. Employees must be allowed taco Tuesday.

**Weights and Priorities**

1. Class structure and components
2. Class interaction and framework
3. Accurate reservation type interaction
4. Date IO, i.e. saving reservation information to file
5. Printing out report orders
6. Saving reservation information to a database

**OOAD – Domain Modeling**

**Business Description**

**Overview**

Ophelia’s Oasis in the Amlet desert is a hotel in located within the Amlet desert. The business needs a reservation management system to improve user interactivity and reduce the need for manual reservation management. By using a system like this, the business can increase its efficiency and reservation processing capabilities

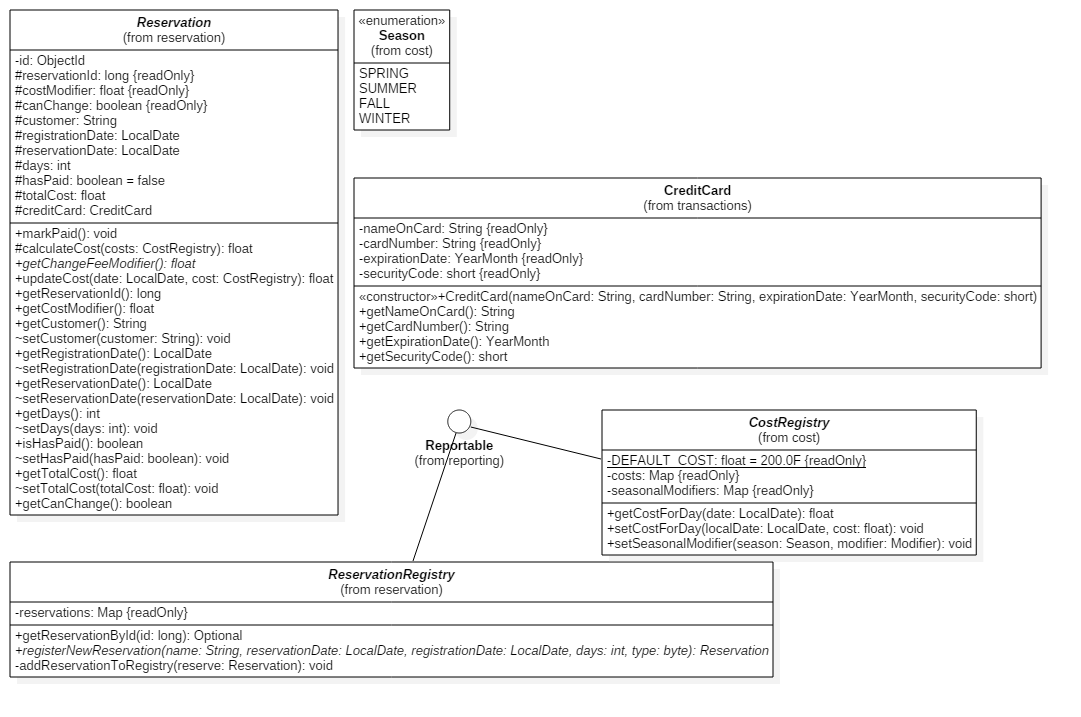
**Business Operations**

This system will, ideally, include a web interface that allows users to access the reservation system at any time. Additionally, there might be a desire to include an automated answering service that can handle these reservations as well.

These requests are passed the central reservation system, which will validate the request, and added it to the central registry is it is approved. Exact process may vary with the type of reservation. For certain reservations, guests will receive emails notifying them of the coming reservation date, as well as the need to provide a payment to ensure their reservation won’t be canceled. Room numbers will not be assigned until the morning of the arrival. Obviously, the system must prevent overbooking, while striving to achieve full capacity every day. The guests, if they have not prepaid, must provide a credit card before they are allowed to occupy a room. On checkout, they will pay for the room, if not already done, and receive a receipt for their stay.

Daily, the system will generate different types of management reports, such as vacancy reports, income, as well as expected income and occupancy. All reservations records must be kept, whether canceled or not. These will be stored locally after each transaction, and will be backed up elsewhere at the end of the day.

**UML Class Diagram**



1

1

**Domain Model Review Checklist**

1. This design model contains the bulk of all significant classes that will be required for this system to work.
2. This model design does show important relationships where they exist.
3. These models are designed similar to real-world concepts, and accurately reflect relationships these real world items would have.
4. This model shows multiplicity constraints, such as the registries of which we only need one instance.
5. The classes shown are all enumerated/abstract classes intended to show relationships, without showing the particular implementations thereof.
6. All classes show the important and relevant attributes that are necessary for that class to function properly within the application context.
7. The class naming and attributes are simple and clear to understand.
8. The classes aren’t used as attributes types. Instead, a composite part relationship is indicated between them.
9. Classes do show operations, since the designs were ported straight from the Java classes. However, due the accelerated schedule of this project, this can be accepted.

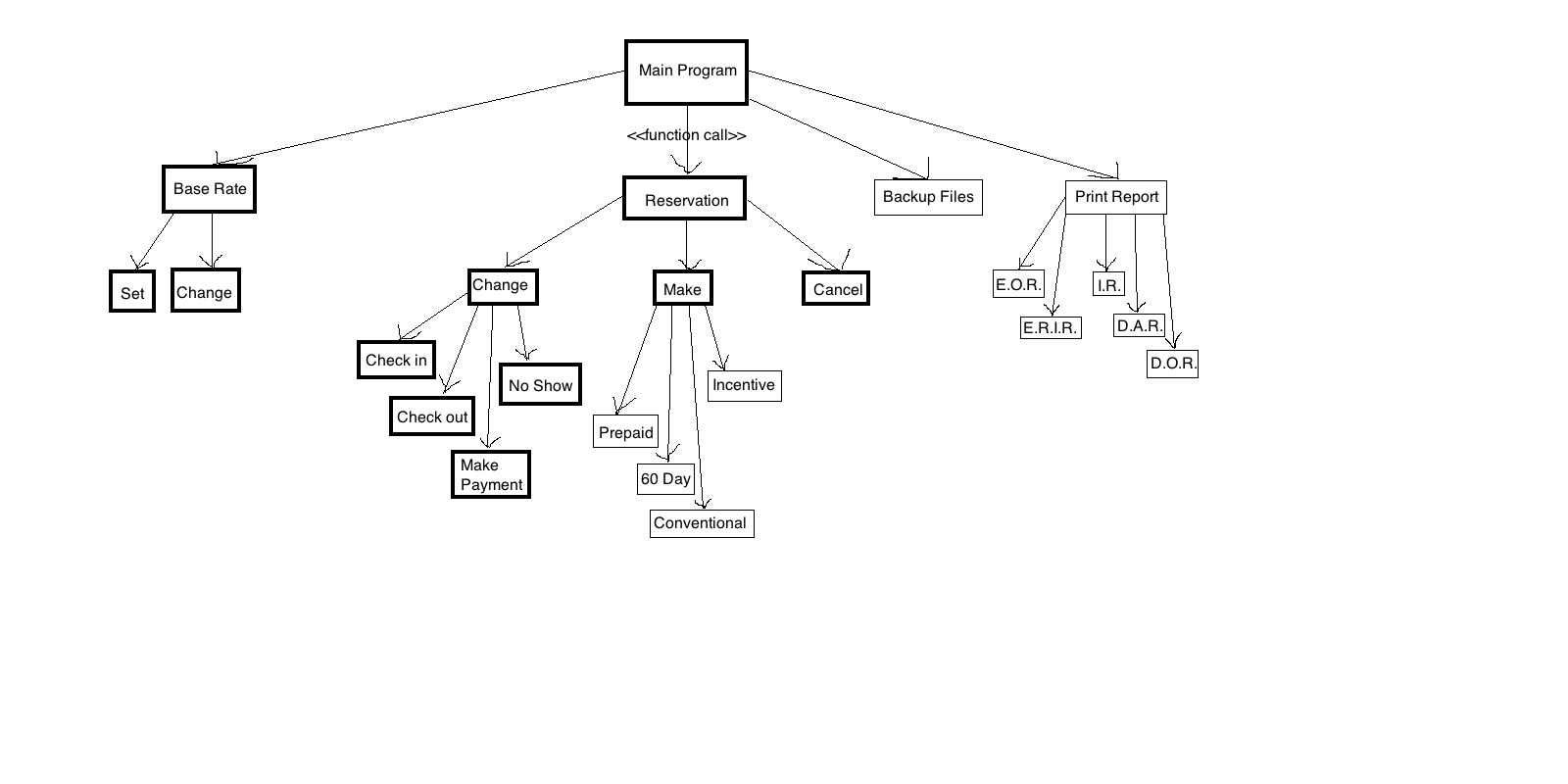
**OOAD – Architectural Design**

**Type of System**

Ophelia’s Oasis in the Amlet desert hotel will use an *interactive* Hotel Management System (HMS) to meet all of the functional requirements outlined by the client. The interactive system is a fitting choice for this application, since it allows the actor to carry out a business process which consists of a relatively fixed sequence of actor requests and system responses. The interactive system is also a fitting choice, because it is relatively simple to design and implement, and is easily applied to client-server relationships.

**Architectural Style**

Ophelia’s Oasis in the Amlet desert hotel will use a *main program and subroutines* architectural style to implement its Hotel Management System which will look much like the one below.



**Functionality and Interface**

As is shown in the above diagram, the engine that drives this system is the main program.  From the main program, once an employee or management interfaces with the system, the system invokes the appropriate subroutine to fulfill the functional requirement which was specified by the client.  The reservation is updated, or the report is printed (as examples), then the function exits and awaits the next user interaction.

      Interface with the system is made by employees and management only, as described in the functional requirements.  Guests contact the employees.  The employees interface with the system to perform the needed task in order to fulfill the appropriate functional requirement.  The system executes the appropriate task.  Then, the system exits once it has finished its performed task and awaits the next employee or management interface.

**Requirements-use case traceability matrix**

R = Requirements

P = Priority

UC = green

UC P = Use case priority

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | P | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 1 | 1 | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 1 | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 1 |  | x | x | x | x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 3 |  |  |  |  |  | x |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 3 |  |  |  |  |  |  | x |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | 1 |  |  |  |  |  |  |  | x |  |  |  |  |  |  |  |  |  |  |  |
| 8 | 1 |  |  |  |  |  |  |  |  | x | x |  |  |  |  |  |  |  |  |  |
| 9 | 1 |  |  |  |  |  |  |  |  |  |  | x |  |  |  |  |  |  |  |  |
| 10 | 3 |  |  |  |  |  |  |  |  |  |  |  | x |  |  |  |  |  |  |  |
| 11 | 4 |  |  |  |  |  |  |  |  |  |  |  |  | x |  |  |  |  |  |  |
| 12 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  | x | x | x | x | x | x |
| UC | P | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 1 | 1 | 1 | 1 | 4 | 1 | 5 | 5 | 5 | 5 | 5 | 5 |

**High-level use cases**

**UC01.** Set the base rate

TUCBW management setting the current base rate for each room depending on the time of year and type of reservation.

TUCEW management setting the base rate for the time of year and reservation

**UC02.** Make prepaid reservation

TUCBW a staff member makes a prepaid reservation

TUCEW a staff member receives confirmation

**UC03.** Make 60-days in advance reservation

TUCBW a staff member makes a 60-days in advance reservation

TUCEW a staff member receives confirmation

**UC04.** Make conventional reservations

TUCBW a staff member makes a conventional reservation

TUCEW a staff member receives confirmation

**UC05.** Make incentive reservations

TUCBW a staff member makes an incentive reservation

TUCEW a staff member receives confirmation

**UC08.** Check in

TUCBW a staff member checks in a guest

TUCEW a staff member receive full payment and confirmation

**UC09.** Check out

TUCBW a staff member checks out a guest

TUCEW a staff member receive confirmation

**UC10.** Print accommodation bill

TUCBW a staff member access guest accommodation bill and clicks “print”

TUCEW a staff member receives the print accommodation bill

**UC11.** Accept payment

TUCBW a staff member receives guest credit card information

TUCEW a staff member receives confirmation of payment

**UC13.** Backup copy of all files

TUCBW a staff member clicks on “backup files”

TUCEW a staff member receives confirmation

**Derived use cases**

**UC01.** Set the base rate (Actor: Management, System: HMS)

**UC02.** Make prepaid reservation (Actor: Staff, System: HMS)

**UC03.** Make 60-days in advance reservation (Actor: Staff, System: HMS)

**UC04.** Make conventional reservations (Actor: Staff, System: HMS)

**UC05.** Make incentive reservations (Actor: Staff, System: HMS)

**UC06.** Cancel reservation (Actor: Staff, System: HMS)

**UC07.** Change reservation (Actor: Staff, System: HMS)

**UC08.** Check in (Actor: Staff, System: HMS)

**UC09.** Check out (Actor: Staff, System: HMS)

**UC10.** Print accommodation bill (Actor: Staff, System: HMS)

**UC11.** Accept payment (Actor: Staff, System: HMS)

**UC12.** Classify as no show (Actor: Staff, System: HMS)

**UC13.** Backup copy of all files(Actor: Staff, System: HMS)

**UC14.** Print reports (Actor: Staff, System: HMS)

**UC15.** Print expected occupancy reports (Actor: Management, System: HMS)

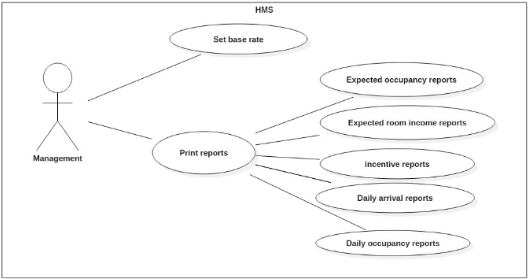
**UC16.** Print expected room income reports (Actor: Management, System: HMS)

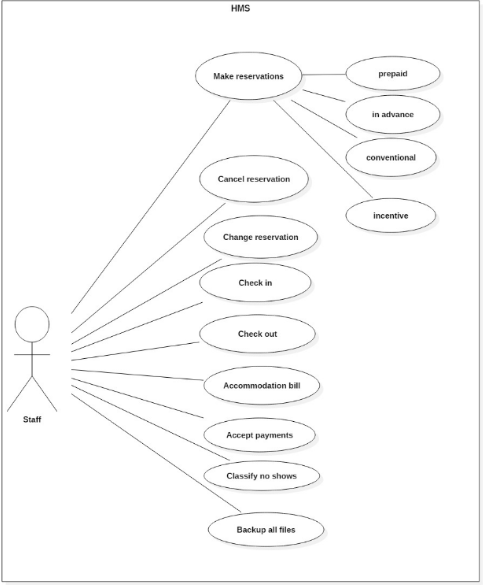
**UC17.** Print incentive reports (Actor: Management, System: HMS)

**UC18.** Print daily arrivals reports (Actor: Management, System: HMS)

**UC19.** Print daily occupancy reports (Actor: Management, System: HMS)

**Use Case Diagram**





**Team Member Contributions**

***Javier Hernandezocasio***

* Requirements Section 3
* Responsible for generating high-level use cases
* Responsible for creating use case diagrams and tables
* User Interaction Consultant

***Nathaniel Spindler***

* Requirements Section 4
* Responsible for Creating Architecture Type
* Responsible for Creating Architecture Style Diagrams
* Architecture Consultant

***Philip Whiting***

* Requirements Sections 1 and 2
* Responsible for Functional Requirements
* Responsible for Business Overview
* Responsible for Domain Modeling and Class Design
* Code Design and Structure Consultant

***Team Meeting Log***

*Tuesday, September 13, 2016 –* Met and discussed general team assignments, and split up tasks.

*Thursday, September 22, 2016 –* Met and discussed specific design approaches and architectural strategies.

***Git Commit Log***

*Javier Hernandezocasio –* 10 commits

*Nathaniel Spindler* – 4 commits

*Philip Whiting –* 33 commits (mostly repository management and code prototyping)