CSc 4350: Software Engineering

Fall 2016

Team Phoenix

Fergus Kelley

Andrew Truong

Ryan Ocampo

Deividas Rutkauskas

Viraj Shah

Document #4 — Object Design

October 3rd , 2016

Class Interfaces

**public** **class** Employee {

/\* Employee object attributes

\* to be filled in by the

\* manager or administrator

\*/

**private** String FirstName;

**private** String LastName;

**private** String Username;

**private** String Password;

**private** String JobTitle;

/\* to be generated by the system \*/

**private** **int** UserID;

/\* Creates an Employee object to be

\* sent to the database.

\*

\* @precondition:

\* FillInCheck(p)

\* all relevant Employee

\* attributes like FirstName, LastName,

\* etc. have to be provided as parameters.

\*

\* @postcondition:

\* Employee object is created and sent to

\* the database.

\*/

**private** **void** Create() {...}

/\* Deletes an existing Employee

\* from the database.

\*

\* @precondition:

\* CheckIfExists()

\* The relevant Employee has to

\* exist within the database.

\*

\* @postcondition:

\* Employee object is created and sent to

\* the database

\*/

**private** **void** Delete(Employee e) {...}

/\* Edits an existing Employee's

\* information in the database.

\*

\* @precondition:

\* CheckIfExists()

\* The relevant Employee has to

\* exist within the database.

\*

\* @postcondition:

\* Employee object is updated

\* within the database.

\*/

**private** **void** Edit(String[] params) {...}

/\* Other methods omitted \*/

}

/\*Hotel rooms are reserved by guests through actions of the

\* Employees that create such rooms

\*

\*

\*/

**public** **class** Hroom {

/\*There are initially no room objects created

\*/

/\*The rooms all have distinct room numbers, with rnum comprising of a

\* literal combination of numbers and letters

\*/

**private** String rnum;

/\*The hroom iS returns an initial boolean value False unless modified by

\* Employee, indicating if the room is a designated smoking area

\*

\*/

**private** **boolean** iS;

/\*The hroom hP returns an initial boolean value False unless modified by

\* Employee, indicating if the room allows for pets

\*/

**private** **boolean** hP;

/\*The hroom dA returns an initial boolean value False unless modified by

\* Employee, indicating if the room is accessible to those with disabilities

\*/

**private** **boolean** dA;

/\*The hroom price returns a value of initially 0 until modified by

\* Employee

\*/

**private** **int** price;

/\*the hroom clean date uses the reference of the reservation start date object,

\* and is merely scheduled 1 day after the reservation start date

\*

\*/

**private** Date cleanDate;

/\*isiS() returns the boolean value of iS

\*/

**public** **boolean** isiS() {}

/\*ishP() returns the boolean value of hP

\*/

**public** **boolean** ishP() {}

/\*isdA() returns the boolean value of dA

\*/

**public** **boolean** isdA() {}

/\*getPrice() returns the int value of price

\*/

**public** **int** getPrice() {}

/\*setiS() functions as a way to modify whether or not the room allows

\* designated smoking

\* @param: boolean values True or False

\* @pre: isiS() returns false

\* @post: iS is set to value of @param , returns value

\*/

**public** **void** setiS(**boolean** iS) {}

/\* sethP() modifies whether the hotel room can allow pets

\* @param: boolean values True or False

\* @pre: ishP() returns false

\* @post: iS is set to value of @param, returns value

\*/

**public** **void** sethP(**boolean** hP) {}

/\*setdA() modifies whether the hotel room can be accessible to those with disabilities

\* @param: boolean values True or False

\* @pre: isdA() returns false

\* @post: iS is set to value of @param, returns value

\*/

**public** **void** setdA(**boolean** dA) {}

/\*setPrice() modifies the price rate of reservation per night

\* @param: int value greater than 0

\* @pre: getPrice() returns an int value

\* @post: method returns value of @param input by employee

\*/

**public** **void** setPrice(**int** price) {}

}

**public** **class** InventoryItem {

**private** String Name;

**private** **int** Quantity;

**private** **int** ExpectedQuantity;

**private** **boolean** IsConsumable;

/\* Creates an InventoryItem object

\* to be sent to the database.

\*

\* @precondition:

\* FillInCheck(p)

\* all relevant InventoryItem

\* attributes like Name, Quantity,

\* etc. have to be provided as parameters.

\*

\* @postcondition:

\* InventoryItem object is created and sent to

\* the UsedInventory.

\*/

**private** **void** Create() {...}

/\* Getters and setters for

\* the Name attribute.

\*

\* @precondition: the relevant

\* InventoryItem has to exist.

\* @postcondition: Name is

\* set or retrieved

\*/

**private** **boolean** ItemExists()

/\* Getters and setters for

\* the Name attribute.

\*

\* @precondition: the relevant

\* InventoryItem has to exist.

\* @postcondition: Name is

\* set or retrieved

\*/

**private** String getName() {...}

**private** **void** setName() {...}

/\* Getters and setters for

\* the Quantity attribute.

\*

\* @precondition: the relevant

\* InventoryItem has to exist.

\* @postcondition: Quantity

\* is set or retrieved.

\*/

**private** **int** getQuantity() {...}

**private** **void** setQuantity() {...}

/\* Getters and setters for

\* the ExpectedQuantity

\* attribute.

\*

\* @precondition: the relevant

\* InventoryItem has to exist.

\* @postcondition: ExpectedQuantity

\* is set or retrieved.

\*/

**private** **int** getExpectedQuantity() {...}

**private** **void** setExpectedQuantity() {...}

/\* Getters and setters for

\* the isConsumable

\* attribute.

\*

\* @precondition: the relevant

\* InventoryItem has to exist.

\* @postcondition: isConsumable

\* is set or retrieved.

\*/

**private** **boolean** getIsConsumable() {...}

**private** **void** setIsConsumable() {...}

}

/\* This class stores and updates

\* InventoryItems.

\*/

**public** **class** UsedInventory {

/\* Updates quantity for a given

\* InventoryItem

\*

\* @Precondition: the relevant

\* InventoryItem has to exist.

\*

\* @Postcondition: the relevant

\* InventoryItem is updated with the

\* provided quantity.

\*/

UpdateQuantity(InventoryItem e, **int** quantity) {...}

/\* Updates expected quantity

\* for a given InventoryItem

\*

\* @Precondition: the relevant

\* InventoryItem has to exist.

\*

\* @Postcondition: the provided

\* InventoryItem is updated with the

\* provided quantity.

\*/

UpdateExpectedQuantity(InventoryItem e, **int** quantity) {...}

/\* Clears the UsedInventory of all

\* InventoryItem objects

\*

\* @Precondition: none

\* @Postcondition: the UsedInventory

\* array list is empty.

\*/

ResetInventory() {...}

}

/\*The class pertains to the functionality of creating hotel reservations, which is done

\* by the employee

\*

\*/

**public** **class** HReservation {

/\*

\* There are no hotel reservations initially created

\*/

/\*

\* A date object is created for the start of the reservation and the end of the reservation

\*

\*/

Date sDate= **new** Date();

Date eDate= **new** Date();

/\*

\* an int value is initialized for the number of adults and children occupying a room

\*/

**private** **int** ad;

**private** **int** ch;

/\*

\* constructors omitted, an Invoice object is created as a reference

\*/

**public** **static** **void** main(String[] args){

Invoice i= **new** Invoice();

}

/\*

\* getsDate() returns the current month, date, and year

\*/

**public** Date getsDate(){}

/\*

\* geteDate() returns the current month, date, and year

\*/

**public** Date geteDate(){}

/\*

\* getad() returns the value of ad, an initial value of 0

\*/

**public** **int** getad(){}

/\*

\* getch() returns the value of ch, an initial value of 0

\*/

**public** **int** getch(){}

/\*the constructors of object sDate will change

\* @param int month, int date, int year

\* @pre: sDate object is created

\* @post: constructors of sDate are changed

\*/

**public** Date setsDate(**int** month, **int** date, **int** year){}

/\*the constructors of object eDate will change

\* @param int month, int date, int year

\* @pre: eDate object is created

\* @post: constructors of sDate are changed

\*/

**public** Date seteDate(**int** month, **int** date, **int** year){}

/\*employee is able to input number of adults occupying the room

\* @param int num

\* @pre: getad() returns 0

\* @post: setad() will return a number greater than 0, ad is updated

\*/

**public** **int** setad(**int** num){}

/\*employee is able to input number of children occupying the room

\* @param int num

\* @pre: getch() returns 0

\* @post: setch() will return a number greater than 0, ch is updated

\*

\*/

**public** **int** setch(**int** num){}

}

/\*The class Eroom denotes the objects created to hold the data of event rooms that employees

\* make reserve on behalf of guests. Event rooms can be reserved for parties, conferences, receptions, etc.

\*

\*/

**public** **class** Eroom {

/\*a String literal rnum represents the sequence of letters and numbers

\* that represent the room number of desired event room

\*

\*/

**private** String rnum;

/\*an int value price represents the price per day for booking the room

\*

\*/

**private** **int** price;

/\*an int value party represents the number of people that will attend the event

\* held in the event room

\*

\*/

**private** **int** party;

/\*a boolean value, initially set to false, hS indicates whether or not

\* the event room has a physical stage

\*

\*/

**private** **boolean** hS;

/\*a boolean value, initially set to false, av indicates whether or not

\* the event room has audio-visual equipment installed

\*

\*/

**private** **boolean** av;

/\*the method getrnum() returns the string value of rnum

\*

\*/

**public** String getrnum(){}

/\*the method getprice() returns the value of price

\*

\*/

**public** **int** getprice(){}

/\*the method getparty() returns the value of party

\*

\*/

**public** **int** getparty(){}

/\*the method ishS() returns boolean value of hS, which is false

\*

\*/

**public** **boolean** ishS() {}

/\*the method isav() returns boolean value of av, which is false

\*

\*/

**public** **boolean** isav() {}

/\*setrnum() allows employee to enter a literal denoting the name of the event room

\*@param string rnum

\*@pre: rnum is null

\*@post: rnum is not null, rnum is updated

\*/

**public** String setrnum(String rnum){}

/\*setprice() allows employee to enter a literal denoting the name of the event room

\*@param int price

\*@pre: price is 0

\*@post: price is updated

\*/

**public** **int** setprice(**int** price){}

/\*setparty() allows employee to enter a literal denoting the name of the event room

\*@param int party

\*@pre: party is 0

\*@post: party is updated

\*/

**public** **int** setparty(**int** party){}

/\*setiS() changes the value of boolean iS

\* @param: boolean value, true or false

\* @pre: iS is set to false

\* @post: iS is updated, and set to true

\*/

**public** **int** setiS(**boolean** iS){}

/\*setav() changes the value of boolean av

\* @param: boolean value, true or false

\* @pre: av is set to false

\* @post: av is updated, and set to true

\*/

**public** **int** setav(**boolean** av){}

}

/\*the class allows employee to create bookings for one or more event rooms requested by

\* guests

\*

\*/

**public** **class** Ebooking {

/\*The Date objects sDate and eDate represent the starting and ending dates

\*of which the event room will be reserved

\*

\*/

**private** Date sDate= **new** Date();

**private** Date eDate= **new** Date();

ArrayList<Eroom> events= **new** ArrayList<Eroom>();

/\*

\* constructors omitted, an Invoice object is created as a reference

\*/

**public** **static** **void** main(String[] args){

Invoice i= **new** Invoice();

}

/\*the constructors of object sDate will change

\* @param int month, int date, int year

\* @pre: sDate object is created

\* @post: constructors of sDate are changed

\*/

**public** Date setsDate(**int** month, **int** date, **int** year){}

/\*the constructors of object eDate will change

\* @param int month, int date, int year

\* @pre: eDate object is created

\* @post: constructors of sDate are changed

\*/

**public** Date seteDate(**int** month, **int** date, **int** year){}

/\*add() will allow the employee to add reserved erooms to

\* an arraylist containing all the used event rooms

\* @param Eroom n, arraylist of erooms being reserved

\* @pre: the desired event room objects are created

\* @post: the event room objects are added to arraylist

\*/

**public** **void** add(Eroom n, ArrayList<Eroom> used){}

}

/\* The Employee class creates Employee object

\* that has attributes FirstName, LastName,

\* Username, Password, JobTitle and Unique

\* User ID. The class also provides methods

\* to delete or modify existing Employee objects.

\*/

}

/\* An invoice is a bill that charges customers for provided services or goods, with a statement of sum due for these services and goods.\*/

Public **class** Invoice {

/\*The name of the customers will be stored as a string when it is provided.\*/

Private String Names;

/\*The current credit card number will be stroned in an array.\*/

Private **int**[]CCNum;

/\*The expiration date of the credit card will also be stored in an array.\*/

Private **int**[]CCExp;

/\*The billable items will be stored in an arrayllist and shows the available items customers can order.\*/

Private ArrayList <String> Billable Items

/\*This returns the name of the customer when he or she is ordering.\*/

Public String getNames(){…}

/\*

Public String getBillable(){…}

/\*This returns the credit card number of the customer that was provided.\*/

Public **int** getCCNum(){…}

/\*This returns the credit card expiration date of the customer.\*/

Public **int** getCCExp(){…}

/\*This method sets or updates the name of the customer.

@pre There isn’t a string in Private String Names or needs to be updated

@post Private String Names= name

/\*

Public void setName(String name){…}

/\*This method sets or updates the credit card number of the customers.

@pre There isn’t an array or it needs to be updated

@post private int[]CCNum=a

\*/

Public **void** setCCNum(**int** []a){…}

/\*This method sets or updates the credit card expiration date.

@pre There isn’t an array or it needs to be updated

@post private int[]CCExp=b

\*/

Public **void** setCCExp(**int** []b){…}

}

/\* The Billable Items are items that are resupplied to the customers when needed like towels, soaps, and etc. Most of the billable items are free of charge.\*/

Public **class** BillableItems{

/\*The name of the items that is offered to the customer.\*/

Private string BillableName;

/\*It contains the price of items.\*/

Private **double** price;

/\*It contains the date the item will be expired. \*/

Private **double** date;

/\*It contains the time the item will be expired.\*/

Private **double** time;

/\* This returns the name of the item. \*/

Public string getBName(){…}

/\*This returns the price of the item.\*/

Public **double** getPrice(){…}

/\*This returns the expiration date of the item.\*/

Public **double** getDate(){…}

/\*This returns the time the time will expire.\*/

Public **double** getTime(){…}

/\*This method will add the available billable items to the array list of the items.

@pre There should be an array list that exist which is Private ArrayList <String> Billable Items from Invoice object.

@post The item will be added to the array list.

\*/

Public **void** addItems(string a, **double** b, **double** c, **double** d)[…}

/\*This method will remove the available billable items to the array list of the items.

@pre There should be an array list that exist which is Private ArrayList <String> Billable Items from Invoice object.

@post The item will removed from the array list.

\*/

Public **void** RemoveItems(string a, **double** b, **double** c, **double** d)[…}

}

**public** **class** Order {

// ArrayList of BillableItem objects

**private** List billables;

// Date and time expected represented as a Date object and and integer representing minutes

**private** Date orderDate;

**private** **int** estimatedTime;

// State represented as an integer (either ordered 0, preparing 1, complete 2)

**private** **int** orderstatus;

// Reference to Invoice Object (to which the order is billed represented as a String

**private** String InvoiceNumber;

/\* Getters and setters for

\* the billables attribute.

\* @precondition: the relevant

\* Order has to exist.

\* @postcondition: billables

\* is set or retrieved.

\*/

**public** **void** setBillables(ArrayList<Billables> billablesList) {…}

**public** **int** getNumberofBillables(){…}

/\* Getters and setters for

\* the orderDate attribute.

\*

\* @precondition: the relevant

\* Order has to exist.

\* @postcondition: orderDate

\* is set or retrieved.

\*/

**public** **void** setOrderDate(){…}

**public** String getEstimatedOrderTime(){…}

/\* Getters and setters for

\* the orderStatus attribute.

\*

\*

\* @precondition: the relevant

\* Order has to exist.

\* @postcondition: orderStatus

\* is set or retrieved.

\*/

**public** **void** setOrderStatus(){…}

**public** String getOrderStatus(){…}

/\* Getters and setters for

\* the InvoiceNumber attribute.

\*

\*

\* @precondition: the relevant

\* Order has to exist.

\* @postcondition: InvoiceNumber

\* is set or retrieved.

\*/

**public** **void** setInvoiceReference(String invoice) {…}

**public** String getInvoiceReference(){…}

}

**public** **class** RoomServiceOrder **extends** Order{

// RoomNumber represented as an integer

**private** **int** roomNumber;

/\* Getters and setters for

\* the roomNumber attribute.

\*

\*

\* @precondition: the relevant

\* RoomServiceOrder has to exist.

\* @postcondition: roomNumber

\* is set or retrieved.

\*/

**public** **void** setRoomNumber(**int** number) {…}

**public** **int** getRoomNumber(){…}

}

**public** **class** TableServiceOrder **extends** Order{

// tableNumber represented as an integer

**private** **int** tableNumber;

/\* Getters and setters for

\* the tableNumber attribute.

\*

\*

\* @precondition: the relevant

\* TableServiceOrder has to exist.

\* @postcondition: tableNumber

\* is set or retrieved.

\*/

**public** **void** setTableNumber(**int** number) {…}

**public** **int** getTableNumber(){…}

**public** **class** CateredMealOrder **extends** Order {

// RoomName represented as an String

**private** String RoomName;

/\* Getters and setters for

\* the roomName attribute.

\*

\* @precondition: the relevant

\* CateredMealOrder has to exist.

\* @postcondition: roomName

\* is set or retrieved.

\*/

**public** **void** setRoomName(String name) {…}

**public** **void** getRoomName(){…}

**public** **class** MaintenanceOrder **extends** Order {

// Description represented as a String

// RoomName represented as a String

**private** String Description;

**private** String RoomName;

/\* Getters and setters for

\* the Description attribute.

\*

\*

\* @precondition: the relevant

\* MaintenanceOrder has to exist.

\* @postcondition: Description

\* is set or retrieved.

\*/

**public** **void** setDescription(String description{…}

**public** String getDescription(){…}

/\* Getters and setters for

\* the roomName attribute.

\*

\*

\* @precondition: the relevant

\* MaintenanceOrder has to exist.

\* @postcondition: roomName

\* is set or retrieved.

\*/

**public** **void** setRoomName(String name) {…}

**public** **void** getRoomName(){…}

}

/\*Restaurant is part of the hotel management software. \*/

Public **class** Restaurant{

/\*The numofTable contains the number of table available in the restaurant of the hotel.\*/

Private **int** numofTable;

/\*The array list is used to contain the list of restaurant items.\*/

Private ArrayList <Object> RestItem;

/\*The array list is used to contain the list of catered meals.\*/

Private ArrayList<Object>CateredMeals;

/\*This returns the number of table of the restaurant.\*/

Public getTable(){…}

/\*This returns the list of available restaurant items on the menu.\*/

Public getRestaurantItem(){…}

/\*This returns the list of available catered meals on the menu.\*/

Public getCateredMeals(){…}

}

/\*The RestaurantMenuItem is an object that allows the adding and removing of items from the menu of the arraylist from the Restaurant class. \*/

Public **class** RestaurantMenuItem{

/\*This contains the name of the restaurant items.\*/

Private string ItemName;

/\*This contains the price of the restaurant items.\*/

Private **double** Price;

/\*This contains the information about the menu items.\*/

Private string Description;

/\*This returns the name of the menu items.\*/

Public getItemName(){…}

/\*This returns the price of the menu items\*/

Public getPrice(){…}

/\*This returns the description of the menu items.\*/

Public getDescription(){…}

/\*This method adds a new item into the menu.

@pre Private ArrayList <Object> RestItem from the Restaurant class must be created.

@post Item is added to the array list.

\*/

Public **void** addItem(String a, **double** b, string c){…}

/\*This method removes item from the menu.

@pre The array list of restaurant menu must be available and created.

@post The item from the array list is removed.

\*/

Public **void** removeItem(String a, **double** b, string c){…}

}

/\*The CateredMealItem is an object that adds and removes catered items from the menu of array list for catered items.\*/

Public **class** catered meal item{

/\*This contains the name of the catered item.\*/

Private string MealName;

/\*This contains the price of the catered item per person.\*/

Private **double** PricePerSeat;

/\*This contains the description of the catered item.\*/

Private string CaterDescription;

/\*This returns the name of the catered item.\*/

Public getCateredName(){…}

/\*This returns the price per person of the catered item.\*/

Public getPrice(){…}

/\*This returns the description of the catered item.\*/

Public getDescrip(){…}

/\*This method will add a catered item to the current menu of the catered items which is in the array list.

@pre There must be an existing array list of the catered item.

@post The catered item will be added to the list of the catered item array list.

\*/

Public **void** addCater(String name, **double** price, string descrip){…}

/\*This method will remove a catered item to the current menu of the catered items which is in the array list.

@pre There must be an existing array list of the catered item.

@post The catered item will be removed from the list of the catered item array list.

\*/

Public **void** removeCater(String name, **double** price, string descrip ){…}

}

Category Interaction Diagram

Interface\_CAT

Processing\_CAT

Objects\_CAT

Database\_CAT

Requirements Traceability Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entry # | Para # | PRMS Requirements Traceability Matrix | Type | Use Case |
| 1 | 2.0 | The PRMS shall provide a graphic user interface (GUI) with tabs. | SW |  |
| 2 | 2.0 | Each tab of the interface shall profile functionality for employees to interact with the PRMS. | SW |  |
| 3 | 2.0 | When the employee clicks on a tab, the PRMS shall display a new interface with the appropriate functionality for that tab. | SW |  |
| 4 | 3.0 | The PRMS shall provide a tab of the interface for creating and managing employee profiles | SW |  |
| 5 | 3.0 | The employee profiles shall be stored in a local database. | SW |  |
| 6 | 3.0 | Each profile shall contain information about the employee. | SW |  |
| 7 | 3.0 | Each profile will contain information about the employee’s access to the PRMS (authentication). | NTH |  |
| 8 | 3.1 | The PRMS shall provide an interface where the new employee profiles are created. | SW |  |
| 9 | 3.1 | Each employee profile shall contain the employee’s name, job title, unique ID number, username, and password. | SW |  |
| 10 | 3.1 | The PRMS shall allow the entry of the employee’s profile information into the new profile window. | SW | UC2 |
| 11 | 3.1 | The PRMS shall create the employee’s unique ID number with cannot be changed. | SW |  |
| 12 | 3.1 | The PRMS will provide functionality for suitably authorized employees to modify the information contained in already existing profiles. | SW | UC2 |
| 13 | 3.2 | The PRMS will limit access by way of a username and password | SW | UC1 |
| 14 | 3.2 | The username and password will be created when the employee profile is created. | SW |  |
| 15 | 3.2 | The employee profile will contain information about which interface tabs are available to each employee based on the employee’s title. | SW |  |
| 16 | 3.2 | The PRMS will log employee interactions with the PRMS in a text file. | NTH |  |
| 17 | 4.0 | The PRMS shall allow the management and reservation of the resort’s hotel rooms. | SW | UC3 |
| 18 | 4.1 | The PRMS shall provide a tab of the user interface for managing the resort’s hotel rooms. | SW | UC3 |
| 19 | 4.1 | The PRMS shall provide functionality for describing the number of floors of the hotel, and the number of rooms on each floor. | SW | UC3 |
| 20 | 4.1 | The PRMS shall assign each room a number based on which floor of the hotel the room is located. | SW |  |
| 21 | 4.1 | The PRMS shall provide functionality to enter and modify default attributes about each room. | SW | UC4 |
| 22 | 4.1 | Each room profile shall have attributes and a list of the rooms inventory. | SW |  |
| 23 | 4.1 | For each item in the room, the room inventory shall keep track of the name, quantity, expected quantity, and whether the item is consumable. | SW |  |
| 24 | 4.1 | The PRMS shall keep track of the number of each item in the rooms inventory that has been replaces. | SW |  |
| 25 | 4.1 | The room profiles shall be stored in a local database. | SWC |  |
| 26 | 4.2 | The PRMS shall provide a tab of the user interface that provides functionality for employees to create and modify reservations for each room. | SW | UC5 |
| 27 | 4.2 | The room reservations shall be stored in a local database. | SWC |  |
| 28 | 4.2 | Each reservation shall contain attributes. | SW |  |
| 29 | 4.2 | The reservation’s billing information shall contain fields. | SW |  |
| 30 | 4.2 | The PRMS shall allow employees to query the list of rooms by the room’s attributes and whether a reservation exists for a given timespan. | SW | UC5 |
| 31 | 4.2 | The PRMS shall allow employees to create reservations for any room that does not have a reservation for a given time span. | SW | UC5 |
| 32 | 4.2 | The PRMS shall require the employee to enter valid information for all the fields in the room reservation information. | SW |  |
| 33 | 4.2 | The PRMS shall allow employees to modify a reservation’s billing information by adding or removing additional charges. | SW | UC6 |
| 34 | 4.2 | The PRMS shall allow employees to modify a reservations timespan to extend or shorten a guest’s stay. | SW | UC5 |
| 35 | 4.2 | The PRMS shall allow employees to output an itemized invoice containing all information about the room, including the calculated cost of the reservation based on the price per night and timespan. | SW | UC6 |
| 36 | 4.3 | The PRMS shall provide a tab of the user interface that provides functionality for employees to manage room maintenance and inventory. | SW | UC7 |
| 37 | 4.3 | The PRMS shall allow the employee to note modify the number of items in the room inventory to indicate whether replacements are needed. | SW | UC7 |
| 38 | 4.3 | The PRMS shall provide an interface to note when the room has been cleaned or the inventory has been replaced and update the necessary fields in the room profile. | SW | UC7 |
| 39 | 4.3 | The PRMS shall allow the employee to output a report of all rooms which haven’t been cleaned in a given time period, rooms which have maintenance notes, or rooms with insufficient inventory. | SW | UC7 |
| 40 | 5.0 | The PRMS shall allow the management and reservation of the resort’s events rooms. | SW |  |
| 41 | 5.1 | The PRMS shall provide a tab of the user interface for managing the resort’s events rooms. | SW | UC8 |
| 42 | 5.1 | The PRMS shall provide functionality for describing the number of events rooms in the resort. | SW | UC8 |
| 43 | 5.1 | The PRMS shall provide functionality to enter and modify default attributes about each events room. | SW |  |
| 44 | 5.1 | Each events room shall have attributes. | SW |  |
| 45 | 5.1 | The events room profiles shall be stored in a local database. | SWC |  |
| 46 | 5.2 | The PRMS shall provide a tab of the user interface that provides functionality for employees to create and modify bookings for events rooms. | SW | UC9, UC10 |
| 47 | 5.2 | Each events room reservation shall contain attributes. | SW |  |
| 48 | 5.2 | The booking billing information shall contain fields. | SW |  |
| 49 | 5.2 | The PRMS shall allow employees to query the list of events rooms by the room’s attributes and whether a booking exists for a given timespan. | SW | UC9 |
| 50 | 6.0 | The PRMS shall provide functionality for employees to manage the various services that are necessary to provide to the guests. | SW |  |
| 51 | 6.1 | The PRMS shall provide a tab of the user interface that provides functionality for employees to create and modify service orders. | SW | UC11, UC12 |
| 52 | 6.1 | Each order shall contain the attributes. | SW |  |
| 53 | 6.1 | The PRMS shall allow the employee to create a new order. | SW | UC11 |
| 54 | 6.1 | For a new order, the state of the order shall default to “ordered” and the date and time expected shall default to the current time. | SW |  |
| 55 | 6.1 | The PRMS shall ensure that the employee has selected or created an invoice for the order provided. | SW |  |
| 56 | 6.1 | The PRMS shall display an interface for entering in the various data for the type of order the employee has selected | SW | UC11 |
| 57 | 6.1 | The PRMS shall allow employees to view a list of orders sorted by time expected and filterable by current state | SW | UC13 |
| 58 | 6.1 | The PRMS shall allow the employee to update the state of the order. | SW | UC12 |
| 59 | 6.1 | Once the state of the order is changed to “delivered,” the invoice associated with the order shall be updated with the name and cost of the order. | SW |  |
| 60 | 6.1 | Orders shall be stored in a local database. | SWC |  |
| 61 | 6.2 | The PRMS shall provide an interface that provides functionality for managing the hotel restaurant. | SW |  |
| 62 | 6.2 | The PRMS shall allow employees to create and modify a list of restaurant items available for room service and table service. | SW | UC14 |
| 63 | 6.2 | Each restaurant item shall contain attributes. | SW |  |
| 64 | 6.2 | The PRMS shall allow employees to create and modify a list of tables available for seating at the restaurant | SW | UC14 |
| 65 | 6.2 | Each table shall contain attributes. | SW |  |
| 66 | 6.2 | When creating an order for room service or table service, the PRMS shall require that the employee select one or more items and select or create an invoice for billing. | SW | UC11 |
| 67 | 6.2 | If creating an order for table service, the PRMS shall require that the employee select a table number from the list of restaurant tables. | SW | UC11 |
| 68 | 6.3 | The PRMS shall allow employees to create and modify a list of options available for catered meals. | SW | UC14 |
| 69 | 6.3 | Each catered option shall contain attributes. | SW |  |
| 70 | 6.3 | When creating an order for catered meals, the PRMS shall require that the employee select a conference room booking for delivery and billing, as well as enter the number of servings required. | SW | UC11 |
| 71 | 6.4 | The PRMS shall provide an interface that provides functionality for creating general orders. | SW | UC11 |
| 72 | 6.4 | When creating a general service order, the PRMS shall provide a text box for entering specific information about the order. | SW | UC11 |
| 73 | 7.0 | The PRMS shall provide a tab of the user interface that provides functionality for employees to easily query the various databases used by the software. | SW | UC15 |
| 74 | 7.0 | The PRMS shall provide interfaces for returning information in any given timespan. | SW | UC15 |

Work Schedule Diagram

Gantt Chart

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Start Date:  August 22, 2016** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Week:** | **1** |  |  |  |  | **2** |  |  |  |  | **3** |  |  |  |  | **4** |  |  |  |  | **5** |  |  |  |  | **6** |  |  |  |  | **7** |  |  |  |  |
|  | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** | **M** | **T** | **W** | **T** | **F** |
| **Document 1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fergus |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Andrew |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ryan |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Devidas |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Viraj |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Document 2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fergus |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Andrew |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ryan |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Devidas |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Viraj |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Document 3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fergus: Gantt, FP, RTM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Andrew: UC, UC diagrams |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ryan: UC, UC diagrams |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Devidas: UC, UC diagrams |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Viraj: UC, UC diagrams |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Document 4** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fergus: Gantt, WSD, RTM |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Andrew: Object design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ryan: Object design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Devidas: Object design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Viraj: Object design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Dictionary

*Graphical User Interface* (abbreviated *GUI*): A program that allows the user to interact with the computer using icons and other visual indicators.

*Local Database*: a collection of digital indexed information that can be searched, referenced, changed, compared or otherwise manipulated. A local database is stored on the same computer that is used to access it.

Rationale

Classes were developed for the following components and objects: employees, hotel rooms, hotel room reservations, inventory, used inventory, invoice, billed items, event rooms, event room booking, event room reservations, restaurant items, in-resort restaurant, and catered meal items. The reason we distinguished the hotel room class and the reservation is because one class considers the attributes of the room, and the other considers the billing and amount of time the room can be reserved. For the sake of organization and ease, we concurred that this organization satisfies those principles. Employee objects are created so as to login with their profiles to access the functionalities of the system. We distinguished used inventory from initial inventory for the sake of documenting reports and statistics behind replenishing the hotel’s resources. The invoice will contain all the costs that the guest has accumulated in a data structure, so in that way we can merely create a billable item object to add to said invoice object. We decided that guests can also book event rooms to hold for special occasions at extra costs. Looking up the information of the room and the process of booking the room are once again separate like the hotel room and reservation object so to keep the distinct information separate to avoid confusion. Initially we thought that an ordering system that can take different kinds of requests would be convenient, but we considered the different kinds of orders that we can get. Thus, we divided orders into maintenance orders, cleaning orders, and food service orders mainly. Finally, classes were considered for keeping track of data concerning an in-resort restaurant. On the assumption that the restaurant offers catering in addition, the menu items offered in restaurant and the catered items that sends out would be handled differently, given they have different attributes. The restaurant itself will be an object that contains data that keeps track of the processes that go on inside the restaurant. The restaurant information management would be a challenging addition to the other hotel on-goings we consider in creating the hotel management system.

Change Log

**Date and Time:** October 3, 2016 — 2:33 PM

**Team member:** Ryan Ocampo

**Description:** Initial creation of document

**Version:** 1 — Current version accumulating changes

**Date and Time:** October 4, 2016 — 11:31 PM

**Team member:** Fergus Kelley

**Description:** Added WSD and Gantt chart

**Version:** 1 — Current version accumulating changes

**Date and Time:** October 5, 2016 — 9:33 PM

**Team member:** Ryan Ocampo

**Description:** Added Category interaction diagrams and Class interfaces done by Viraj, Andrew, and Deividas

**Version:** 1 — Final Version