Benny Iko

9/28/2021

SWEN 646 9040 Software Design and Implementation

Week 6 Software Design Document Part 1

**Table of Contents**

1. INTRODUCTION...................................................................................2

1.1 Purpose...............................................................................................2

1.2 Scope..................................................................................................2

1.3 Overview.............................................................................................2

2. SYSTEM OVERVIEW............................................................................3

3. SYSTEM ARCHITECTURE...................................................................4

3.1 Architectural Design............................................................................4

3.2 Decomposition Description..................................................................5

3.3 Exception Handling.............................................................................5

4 DATA DESIGN……………………………………………………………….

4.1 Data Description…………………………………………………………..

5. COMPONENT DESIGN........................................................................6

6 HUMAN INTERFACE DESIGN…………………………………………….

6.1Overview of the User Interface…………………………………………..

6.2 Screen Images…………………………………………………………….

7.REQUIREMENTS MATRIX………………………………………………………….

**INTRODUCTION**

1.1 Purpose

This software design document describes the scope, architecture and system design of the Account and Reservation Manager System. The purpose of this SDD is to specify the details of this system and to serve as a written design; developers will know what to implement after using this document. Moreover, it may be used for review or submitted for approval. This design document is intended for members of the development team, developers, testers, technical writers, project manager or other persons of interest.

1.2 Scope

The software is to be built as a prototype (stand-alone application) that will be loaded onto the local computer of a business. The software will manage reservations and the accounts that they are associated with. The implementation for the user interface of this application is outside the scope of this document; however, the design of the user interface is within scope. The objective of this project is to provide proof of concept. If proven feasible, then the final product outside the scope of this project, uses a database for accounts and reservations. The benefits of this project include: data storage, information generation, and ease of account and reservation management.

1.3 Overview

This document follows the logical order laid out in the table of contents. It is organized according to the following key sections: introduction, overview, architecture and the interface design. Subsections entail additional detailed information relevant to its title. The introduction section is broken into 3 components: purpose, scope, and this section (overview). The first entails the purpose of the document, then goes on to describe the intended audience. Next, the scope section describes precisely what is considered in scope or out of scope as it pertains to this document. This section provides insight as to how each section is laid out. The system architecture is explained via its respective section using a class diagram. Finally, the component design section is organized using following the format:

Class Name:

Class Description/Purpose:

Class Modifiers:

Class Inheritance:

Class Attributes:

Exceptions Thrown:

Class Constructors:

Class Methods:

This pattern is used for every user defined class that is part of the system, including exceptions that extend the runtime class.

**2. SYSTEM OVERVIEW**

An instance of the Manager class is designed to manage accounts represented by Account objects and reservations represented by Reservation objects. This is done through utilization of ArrayLists, as well as instance methods which allow for completion and cancellation of reservations as well as adding accounts. Reservations are managed indirectly by the manager class through the Account class and its methods. The Manager object allows for addition, removal, completion and cancellation of new or existing reservations. A Manager object calls an Account’s methods to perform operations on it or it’s managed reservations. The UI gathers relevant account and reservation information from the user and knows what to do/not to do in a given situation. Also, there are descendants of runtime exceptions for exceptional events that may occur during the program's execution such as duplicate or invalid data. These events will generate helpful messages. The functionality of some methods results in null return values as opposed to the aforementioned exceptional events. For more information on this, check the Component Design section. There are different types of reservations with varying states, which are used to represent the different available reservation types: houses, apartments and hotels. This is reflected in the design of the classes meant to represent the respective reservation types. There is a base class (Reservation) which defines the common attributes between reservation types and is extended through the ApartmentReservation, HotelReservation and HouseReservation classes.This base class is not meant to be instantiated.

**3. SYSTEM ARCHITECTURE**

Architectural Design (UML Class Diagram)



3.2 Decomposition Description

On startup, the UI creates a Manager object. There is a path that is hardcoded, which holds the directory where data will be stored. All account and reservation data is loaded from this point and saved using the constant value PATH and Account and Reservation constructors which allow objects to be created using files. Account information and reservation information are stored in hierarchical fashion within directories branching from the path. Account’s information should be saved in one file and then each reservation that belongs to that account in separate files. This is accomplished via respective save to file methods. Account loading is accomplished through the Managers methods. The Manager instance manages Accounts, which in turn, manage Reservations. The Manager object allows for addition, removal, completion and cancellation of new or existing reservations. The Manager object calls an Account’s methods to perform operations on it or it’s managed reservations. Also, there are runtime exceptions for exceptional events that may occur during the program's execution. There are different types of reservations with varying states. This is reflected in the design of the classes meant to represent the respective reservation types. The UI handles login and authorization and access control.

3.3 Exception Handling

IllegalLoadExceptions are thrown when there is a problem with loading a file. The message generated will include a brief description of the event with pertinent information regarding account ID, file name, and the object that caused the problem. The file constructors for Account and Reservation will throw the exceptions of this type, and the Manager class method which calls these constructors, getAccounts(), will catch exceptions of the type. A DuplicateObjectExcpetion is thrown when an object already exists, while attempting to add an account or add a reservation to an account. There are different constructors which provide a variety of message possibilities. A message generated by this class may include account Id and information about the origin of the error. Another possibility includes reservation number and origin information. Finally, a message with both account ID and reservation information is a possibility, along with origin information. Both the addAccount() method of Manager and the addReservation() method of Account will throw this exception, but the respective Manager methods will catch them. IllegalOperationException occurs when a reservation is not finalized, that is, while cancelling or on an attempt to complete a reservation; exception objects of this class will be thrown by the Reservation instance methods cancelReservation() and completeReservation() and caught by the same methods in Manager. The message to be displayed by the toString method will include a brief description of the error along with account ID, reservation number and the source of the error. IllegalStateException and IllegalArgumentException are used for invalid parameters and when the state of an object is final; the message accompanying this exceptional event will include the illegal parameter and/or a brief message about the object that cannot be changed. Also,since the Java builtin exceptions such as IllegalArgumentException are for invalid parameters, every class will throw an exception of this type but will not catch it. The exception object will store information for this and will be caught through error propagation by the manager function that is/or calls the culprit.

**4. DATA DESIGN**

For the Account and Reservation Management System, the UI will use and call the software. Each Account object will be stored in a separate directory using the account’s id number. Account’s information will be saved in one file and then each reservation that belongs to that account in a separate file.

For example, acc-123456789.txt

<account><accountid>123456789</accountid><emailaddress>[user@website.com](mailto:user@website.com)

</emailaddress><mailingaddress><street>Main st</street><city>Annapolis</city><state>

Maryland</state><zip>20109</zip></mailingaddress></account>

For example, a hotel reservation H123456789.txt

<hotelreservation><accountid>123456789</accountid><reservationnumber>H123456789</reservationnumber><physicaladdress><street>Main st</street><city>Annapolis</city><state>

Maryland</state><zip>20109</zip></physicaladdress><mailingaddress><street>Main st</street><city>Annapolis</city><state>Maryland</state><zip>20109</zip></mailingaddress><reservationstart>Tue Aug 28 11:10:40 UTC 2018</reservationstart><noofnights>5</noofnights><noofbedrooms>1</noofbedrooms><reservationsize>800</reservationsize><reservationstatus>completed</reservationstatus><noofbeds>2</noofbeds> <haskitchenette>true</haskitchenette></hotelreservation>

For example, a house reservation U123456789.txt

<housereservation><accountid>123456789</accountid><reservationnumber>U123456789</reservationnumber><physicaladdress><street>Main st</street><city>Annapolis</city><state>

Maryland</state><zip>20109</zip></physicaladdress><mailingaddress><street>Main st</street><city>Annapolis</city><state>Maryland</state><zip>20109</zip></mailingaddress><reservationstart>Tue Aug 28 11:10:40 UTC 2018</reservationstart><noofnights>5</noofnights><noofbedrooms>1</noofbedrooms><reservationsize>800</reservationsize><reservationstatus>completed</reservationstatus><noofbeds>2</noofbeds><nooffloors>1</nooffloors><noofbathrooms>2</noofbathrooms> </housereservation>

For example, an apartment reservation A123456789.txt

<apartmentreservation><accountid>123456789</accountid><reservationnumber>A123456789</reservationnumber><physicaladdress><street>Main st</street><city>Annapolis</city><state>

Maryland</state><zip>20109</zip></physicaladdress><mailingaddress><street>Main st</street><city>Annapolis</city><state>Maryland</state><zip>20109</zip></mailingaddress><reservationstart>Tue Aug 28 11:10:40 UTC 2018</reservationstart><noofnights>5</noofnights><noofbedrooms>1</noofbedrooms><reservationsize>800</reservationsize><reservationstatus>completed</reservationstatus><noofbeds>2</noofbeds><noofbathrooms>2</noofbathrooms> </apartmentreservation>

COMPONENT DESIGN

Class Name: Manager

Class Description/Purpose: The Manager class manages Accounts and Reservations. It contains the interface for the UI to interact with. It interacts with the interface for the Account class.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

Vector accounts - used to store accounts

String PATH - hardcoded path for directory

Exceptions Thrown: IllegalLoadException, IllegalArgumentException,DuplicateJobException, IllegalOperationException

Class Constructors:

public Manager(){

//Create a new Vector object for accounts

//if PATH does not exist create it

//otherwise load data from it

//Loop through every folder in the path

//For every folder in the path

//Create a new account using the constructor with file name parameter

}

Class Methods:

public void addReservation(Integer accountNumber, Reservation reservation){

//Call getAccount(Integer)

//if account object with matching id doesn't exist, throw lllegalArgumentException

//else find the appropriate account that matches the passed in id

//call the accounts add method

}

public Account getAccount(Integer accountNo){

//Loop through accounts

//Get account matched by accountNo

//return null if account does not exist

//Otherwise, return the account

}

public void updateReservation(Integer accountNo, Reservation reservation){

//Make call to getAccount(integer)

//Get account matched by accountNo

//throw IllegalArgumentException if it account does not exist

//Otherwise, call the accounts update method passing the reservation object

}

private void createAccountDirectory(Integer accountNo){

//create directory using accountNo for filename

}

public void addAccount(Account account) {

//Loop through accounts

//Call getAccountNo fo reach object

//if account number exists already throw DuplicateObjectException

//otherwise, call accounts.add(account)

}

public void updateAccount(Account account) {

//call Account update method

//throw IllegalArgumentException if account does not exist

}

public void completeReservation(Integer accountNo, Reservation reservation) {

//find right account,reservation

//if matching account doesn't exist, throw IllegalArgumentException

//call appropriate complete reservation in reservation class

//throw IllegalOperationException if unable to finish

}

public void cancelReservation(Integer accountNo, Reservation reservation) {

//find right account,reservation

//call appropriate cancel reservation in reservation class

//throw error if unable to finish

}

public Float calculatePricePerNight(Integer accountNo, Reservation reservation) {

//find account

//find reservation

//call methods for calculating price

}

public Float calculateTotalReservationPrice(Integer accountNo, Reservation reservation) {

//find account

//find reservation

//call methods for calculating price

}

public void saveToFile(Integer accountID) {

//using accountID find the correct account object

//call object’s saveToFile method

}

public Vector getAccounts() {

//return accounts

}

Class Name: Account

Class Description/Purpose: The Account class stores contact information, as well as information about reservations. It contains the complete interface for the manager to interact with. It directly invokes the Reservation classes’ methods.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

Integer accountID - unique identifier for an account

Vector reservations - list of reservations related to an account

String emailAddress - an email address assigned to an account

Address mailingAddress - the mailing address for an account

Exceptions Thrown:

Class Constructors:

public Account (String directoryName) {

//set path as starting point

//loop through all account directories

//call appropriate Account constructor for each directory

//For each Reservation file, call appropriate Reservation constructor

}

public Account(Integer accountID, Address mailing, String email) {

//validate values

//initialize values

//initialize reservations

}

Class Methods:

public String toString() {

//get formatted data

//return as String

}

public void saveToFile(String directoryName) {

//create directory using this fileName and path

//call updateAccount and updateReservation

//create account and reservation files

}

public void addReservation(Reservation reservation) {

//add reservation parameter to list

//if reservation number already exists, throw DuplicateObjectException

//otherwise, add to account

}

public void completeReservation(String reservationNo) {

//find reservation

//if reservation number doesn't exist, throw IllegalArgumentException

//otherwise, call reservation object's methods

}

public void cancelReservation(String reservationNo) {

//find reservation

//if reservation number doesn't exist, throw IllegalArgumentException

//otherwise, call reservation object's methods

}

public Reservation getReservation(String reservationNo) {

//find the reservation object that matches the parameter’s id

//if reservation number doesn't exist return null

//otherwise

//return that object

}

public void updateReservation(Reservation reservation) {

//if reservation number doesn't exist, throw IllegalArgumentException

//else

//call reservation methods in children's methods

}

public Float calculateReservationPricePerNight(String reservationNo) {

//given reservation No

//validate No

//if reservation number doesn't exist, throw IllegalArgumentException

//if exists,

//call method of appropriate Reservation child

//otherwise, error

}

public Float calculateTotalReservationPrice(String reservationNo) {

//given reservation No

//validate No

//if reservation number doesn't exist, throw IllegalArgumentException and return null

//if exists,

//call method of appropriate Reservation child

//otherwise, error

}

public void updateAccount(Account account) {

//validate that the passed parameter is for the same account as itself

//set this account data equal to parameter data

}

public Integer getAccountid() {

//return account id

}

public void setEmailaddress(String address) {

//validate input

//set email address to input

}

public String getEmailaddress() {

//return this account email address

}

public void setMailingAddress(Address mailing) {

//validate input

//set this account mailing address

}

public Address getMailingAddress() {

//return this accounts mailing address

}

public Vector getReservations() {

//return reservations

}

Class Name: Address

Class Description/Purpose: The Address class contains state and behavior for mailing or physical addresses. It contains the complete interface for the User Interface to interact with.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

String street - the street and house number for an address

String city - the city for an address

String state - the state for an address

String zip - 5 digit zip code

Exceptions Thrown:

Class Constructors:

public Address(String city, String state, String zip, String street) {

//validate parameters

//initialize values

}

Class Methods:

public String toString() {

//get formatted data

//return as String

}

public String getStreet() {

// return this address's street value

}

public String getCity() {

// return this address's city value

}

public String getState() {

// return this address's state value

}

public String getZip() {

// return this address's zip value

}

public void setStreet(String street) {

//validate input, then set value

}

public void setCity(String city) {

//validate input, then set value

}

public void setState(String state) {

//validate input, then set value

}

public void setZip(String zip) {

//validate input, then set value

}

Class Name: Reservation

Class Description/Purpose: The Reservation class is the base class for a reservation. The class contains details of a reservation including location,pricing,as well as length of stay data. It contains the complete interface for an Account to call. The different types of reservations available are hotels, apartments and houses.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

Integer accountID - the unique identifier for the account associated with this reservation

String reservationNumber - the unique identifier for this reservation

Address physicalAddress - the physical address for a reservation

Address mailingAddress - the mailing address for a reservation

Calendar reservationStart - the start date for a reservation

Integer reservationSize - the size in square feet

String reservationStatus - the status of a reservation

Integer noOfBeds - the number of beds for this reservation

Exceptions Thrown:

Class Constructors:

public Reservation(String fileName) {

//loads from filename

//if filename doesn't exist/object cannot be loaded throw IllegalLoadException and fail

//otherwise create the object

}

public Reservation(String reservationNo, Address physical, Address mailing, Calendar startDate, Integer nights, Integer beds, Integer size, Integer bedrooms) {

//constructor for when mailing address is explicitly given

//validate values

//initialize values

//set price, status, account id to default values

}

public Reservation(String reservationNo, Address physical, Integer nights, Integer bedrooms, Integer size, Calendar startDate, Integer beds) {

//constructor for when mailing address is not explicitly given

//validate values

//initialize values

//set price, status, account id to default values

}

Class Methods:

public Float calculatePricePerNight() {

//$100 + $20 if greater than 800 square feet; price assumes a single bedroom/bathroom

}

public void saveToFile(String fileName) {

//saves info on this Reservation to file

}

public Float calculateTotalPrice() {

//return calculatePricePerNight() \* nights

}

public void cancelReservation() {

//validate that this Reservation may be cancelled

//if able, cancel reservation

//otherwise throw IllegalOperationException

}

public void completeReservation() {

//validate that this Reservation may be completed

//if able, complete reservation

//call calculate method

//assign calculate method’s result to reservation price attribute

//otherwise throw IllegalOperationException

}

public String toString() {

//get formatted data

//return as String

}

public void updateReservation(Reservation reservation) {

//validate draft status

// if draft, set value

//otherwise it cannot be changed

}

public String getReservationnumber() {

//return reservation No

}

public Address getPhysicaladdress() {

//return physical address

}

public Address getMailingaddress() {

//return mailing address

}

public void setPhysicalAddress(Address physical) {

//validate draft status, Address

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setMailingAdress(Address mailing) {

//validate draft status,Address

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public Calendar getStartDate() {

//return this reservationStart

}

public Integer getNumberofnights() {

//return thisnoOfNights

}

public Integer getNumberofbedrooms() {

//return this noOfBedrooms

}

public Integer getReservationsize() {

//return this reservationSize

}

public String getReservationStatus() {

//return this reservationStatus

}

public Integer getNumberofbeds() {

//return this noOfBeds

}

public Integer getAccountID() {

//return this accountID

}

//Set methods must validate that the reservation is in draft otherwise it cannot be changed

public void setAccountID(Integer id) {

//validate draft status,id

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setReservationStart(Calendar start) {

//validate draft status, Calendar

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setNoOfNights(Integer nights) {

//validate draft status, nights

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setNoOfBedrooms(Integer bedrooms) {

//validate draft status, bedrooms

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setReservationSize(Integer size) {

//validate draft status, size

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void setNoOfBeds(Integer beds) {

//validate draft status, beds

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

Class Name: HouseReservation

Class Description/Purpose: A child of Reservation class; provides additional information necessary for reservation of a house.

Class Modifiers: public

Class Inheritance: inherits from Reservation class

Class Attributes:

Integer noOfFloors - the number of floors for the house

Integer noOfBathrooms - the number of bathrooms for the house

Exceptions Thrown: IllegalLoadException, IllegalStateExcpetion

Class Constructors:

public HouseReservation(String fileName) {

//call to super

//loads from filename

//if filename doesn't exist/object cannot be loaded throw IllegalLoadException and fail

//otherwise create the object

}

public HouseReservation(Integer bathroomNo, Integer floorNo, Address physical, Integer nights, Integer bedrooms, Integer size, String status, Calendar startDate) {

//call to super

//constructor for when mailing address is not explicitly given

//validate values

//initialize values

}

public HouseReservation(Integer bathroomNo, Integer floorNo, Address physical, Integer nights, Address mailing, Calendar startDate, Integer bedrooms, Integer size, String status) {

//call to super

//constructor for when mailing address is explicitly given

//validate values

//initialize values

}

Class Methods:

public String toString() {

//get formatted data

//return as String

}

public Float calculatePricePerNight() {

//call super

//determine additional fees

//return value

}

public void setNoOfFloors(Integer floors) {

//validate draft status, floors

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void updateReservation(HouseReservation reservation) {

//validate input/reservation values

//assign this reservation to the parameters values

}

public Integer getFloorsnumber() {

//return noOfFloors

}

public Integer getNumberofbathrooms() {

//return noOfBathrroms

}

public void setNoOfBathrooms(Integer bathrooms) {

//validate draft status, bathrooms

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

Class Name: HotelReservation

Class Description/Purpose: A child of Reservation class; provides additional information necessary for reservation of a hotel.

Class Modifiers: public

Class Inheritance: Inherits from Reservation class

Class Attributes:

Boolean hasKitchenette - if this reservation has a kitchen

Exceptions Thrown: IllegalLoadException, IllegalStateExcpetion

Class Constructors:

public HotelReservation(String fileName) {

//call to super

//loads from filename

//if filename doesn't exist/object cannot be loaded throw IllegalLoadException and fail

//otherwise create the object

}

public HotelReservation(Boolean kitchenette, Address physical, Integer nights, Calendar startDate, Integer bedrooms, Integer size, String status) {

//call to super

//constructor for when mailing address is not explicitly given

//validate values

//initialize values

}

public HotelReservation(Boolean kitchenette, Address physical, Address mailing, Integer nights, Calendar startDate, Integer bedrooms, Integer size, String status) {

//call to super

//constructor for when mailing address is explicitly given

//validate values

//initialize values

}

Class Methods:

public String toString() {

//get formatted data

//return as String

}

public void setHasKitchenette(boolean kitchen) {

//validate draft status, kitchen value

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public void updateReservation(HotelReservation reservation) {

//validate input/reservation values

//assign this reservation to the parameters values

}

public Boolean getKKitchenetteStatus() {

//return isKitchentte

}

Class Name: ApartmentReservation

Class Description/Purpose: A child of Reservation class; provides additional information necessary for reservation of an apartment.

Class Modifiers: public

Class Inheritance: Inherits from Reservation class

Class Attributes:

Integer noOfBathrooms - the number of available bathrooms

Exceptions Thrown: IllegalLoadException, IllegalStateExcpetion

Class Constructors:

public ApartmentReservation(Integer bathroomNo, Address physical, Integer bedrooms, Integer beds, Integer nights, String status, Calendar startDate, Integer size) {

//call to super

//constructor for when mailing address is not explicitly given

//validate values

//initialize values

}

public ApartmentReservation(String fileName) {

//call to super

//loads from filename

//if filename does not exist/object cannot be loaded throw IllegalLoadException and fail

//otherwise create the object

}

public ApartmentReservation(Integer bathroomNo, Address physical, Address mailing, Integer beds, Integer bedrooms, Integer nights, String status, Calendar startDate, Integer size) {

//call to super

//constructor for when mailing address is explicitly given

//validate values

//initialize values

}

Class Methods:

public void updateReservation(ApartmentReservation reservation) {

//validate input/reservation values

//assign this reservation to the parameters values

}

public Float calculatePricePerNight() {

//call super

//determine additional fees

//return value

}

public String toString() {

//get formatted data

//return as String

}

public void setNoOfBathrooms(Integer bathrooms) {

//validate draft status, bathrooms

// if draft, set value

//otherwise it fails with IllegalStateException being thrown

}

public Integer getNumberofbathrooms() {

//return no of bathrooms

}

Class Name:IllegalLoadException

Class Description/Purpose: When loading accounts and/or reservations from file(s) if there is any issue parsing account or reservation files, then an object of this class is thrown; the unchecked user defined exception called IllegalLoadException.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

String message

String origin

String fileName

Integer accountID

Exceptions Thrown: N/A

Class Constructors:

public IllegalLoadException(String fileName, Integer accountID, String errorOrigin) {

//Add fileName to message String

//Add accountID to message String

//Add errorOrigin to message String

}

Class Methods:

public String toString(){

//return message String

}

Class Name: DuplicateObjectException

Class Description/Purpose:When adding account or a reservation to an account, if a duplicate

exists, then an object of this class is thrown; the unchecked user defined exception called DuplicateObjectException.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

String message

Integer accountID

String reservationNo

String origin

Exceptions Thrown: N/A

Class Constructors:

public DuplicateObjectException(Integer accountID, String origin){

//Add accountID to String message

//Add origin to String message

}

public DuplicateObjectException(String reservationNo, String origin){

//Add reservationNo to String message

//Add origin to String message

}

public DuplicateObjectException(String reservationNo, Integer accountID, String origin){

//Add reservationNo to String message

//Add accountID to String message

//Add origin to String message

}

Class Methods:

public String toString(){

//return String message

}

Class Name: IllegalOperationException

Class Description/Purpose:The unchecked user exception: IllegalOperationException; when attempting to cancel or complete a reservation, if it cannot be finalized, then an object of this class is thrown.

Class Modifiers: public

Class Inheritance: None

Class Attributes:

String message

String reservationNo

Integer accountID

String origin

Exceptions Thrown: N/A

Class Constructors:

public IllegalOperationException(Integer accountID, String reservationNo, String errorOrigin){

//Concatenate accountID to message

//Concatenate reservationNo to message

//Concatenate errorOrigin to message

}

Class Methods:

public String toString(){

//return message

}

**6. HUMAN INTERFACE DESIGN**

6.1 Overview of the User Interface

Upon starting the program, the accounts and reservations will be loaded. Users will then be presented with a screen [fig 1] that leads to options for functionality related to either accounts or reservations. Upon clicking the “Reservations” button, a user is presented with the interface for functionality related to a reservation [fig 2]. Upon clicking the “Accounts” button, a user is presented with the interface for functionality related to an account [fig 3]. From here users will be prompted to enter relevant information in regards to an account or reservation, such as address information or number of beds. The UI for these screens will use the same look and feel.

6.2 Screen Images

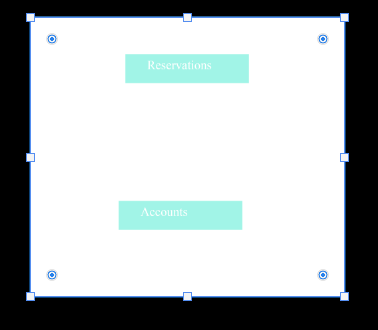
****

Fig 1. - Main Menu which divides major functionality into two entities



Fig 2. - Reservation menu



Fig 3- Account Menu

**7. REQUIREMENTS MATRIX**

| **Requirement** | **Class** | **Constructor/Method** | **Attribute** |
| --- | --- | --- | --- |
| When the system starts up, UI will create an instance of your Manager class | Manager | Manager() |  |
| manage accounts and associated lodging reservations | Manager |  | accounts |
| load all the existing accounts and their associated  lodging reservations on startup | Manager | getAccounts() |  |
| hardcoded the full path for the directory where the data will be loaded from and saved to | Manager |  | PATH |
| you can hardcode the full path for the directory where the data will be loaded from and saved to | Manager | saveToFile(Integer) |  |
| Each account’s data should be saved in a separate directory | Account | saveToFile(String fileName) |  |
| Each reservation that belongs to an account belongs in a separate file. | Reservation | saveToFile(String fileName) |  |
| Account files name convention should be “acc-“ followed by the account id number | Account | saveToFile(String fileName) |  |
| An account includes the following information: unique id number generated by UI | Account |  | accountID |
| An account includes the following information: mailing address | Account |  | mailingAddress |
| An account includes the following information: list of lodging reservations | Account |  | reservations |
| An account includes the following information: email address associated with the account | Account |  | emailAddress |
| When a new account is created, it must be provided with an account id, mailing address, and email. | Account | Account(Integer accountID, Address mailing, String email) |  |
| Account’s mailing address and email can be changed at any time | Account | updateAccount(Account account) |  |
| A new account can be added to the manager | Manager | addAcount(Account account) |  |
| No duplicate accounts | DuplicateObjectExcpetion |  |  |
| There are three types of lodging reservations: hotel room reservation, apartment reservation, and house reservation | ApartmentReservation | ApartmentReservation() |  |
| There are three types of lodging reservations: hotel room reservation, apartment reservation, and house reservation | HouseReservation | HouseReservation() |  |
| There are three types of lodging reservations: hotel room reservation, apartment reservation, and house reservation | HotelReservation | HotelReservation() |  |
| Each lodging reservation includes an account id that the reservation belongs to | Reservation |  | accountNo |
| . Each lodging reservation includes reservation number generated by UI | Reservation |  | reservationNumber |
| Each lodging reservation includes lodging physical address | Reservation |  | physicalAddress |
| Each lodging reservation includes mailing address | Reservation |  | mailingAddress |
| Each lodging reservation includes  Reservation start date | Reservation |  | reservationStart |
| Each lodging reservation includes number of nights | Reservation |  | noOfNights |
| Each lodging reservation includes number of beds | Reservation |  | noOfBeds |
| Each lodging reservation includes lodging size in sq feet | Reservation |  | reservationSize |
| Each lodging reservation includes price of the reservation | Reservation |  | calculateTotalPrice() |
| Each lodging reservation includes reservation status | Reservation |  | reservationStatus |
| hotel reservation has an indication of whether there is kitchenette or not; | HotelReservation |  | hasKitchenette |
| apartment reservation has the number of bathrooms | ApartmentReservation |  | noOfBathrooms |
| house reservation has the number of floors and number of bathrooms. | HouseReservation |  | noOfFloors |
| house reservation has the number of floors and number of bathrooms. | HouseReservation |  | noOfBathrooms |
| All values must exist on reservation creation except for mailing address | Reservation | public Reservation(String reservationNo, Address physical, Integer nights, Integer bedrooms, Integer size, Calendar startDate, Integer beds) |  |
| Account id is updated when reservation is added to a specific account, otherwise it has a negative one value | Account | addReservation(Reservation reservation) |  |
| New reservation always starts in draft status | Reservation |  | reservationStatus |