SdD-Part 2

Table of Contents

[1. Introduction 2](#_Toc107783628)

[1.1 Purpose 2](#_Toc107783629)

[1.2 Scope 2](#_Toc107783630)

[1.3 Overview 2](#_Toc107783631)

[1.4 Reference Material 3](#_Toc107783632)

[1.5 Definitions and Acronyms 3](#_Toc107783633)

[2. System Overview 3](#_Toc107783634)

[3. System Architecture 4](#_Toc107783635)

[3.1 Architectural Design 4](#_Toc107783636)

[3.2 Decomposition Description 5](#_Toc107783637)

[3.3 Exception Handling 5](#_Toc107783638)

[3.4 Design Rationale 6](#_Toc107783639)

[4. Data Design 6](#_Toc107783640)

[4.1 Data Description 6](#_Toc107783641)

[4.2 Data Dictionary 7](#_Toc107783642)

[5. Component Design 8](#_Toc107783643)

[6. Human Interface Design 15](#_Toc107783644)

[6.1 Overview of User Interface 15](#_Toc107783645)

[6.2 Screen Images 15](#_Toc107783646)

[6.3 Screen Objects and Actions 18](#_Toc107783647)

[7. Requirements Matrix 18](#_Toc107783648)

[8. Appendices 19](#_Toc107783649)

# 1. Introduction

## 1.1 Purpose

The Software Engineering is the process of analyzing user requirements, designing the software, developing the software and testing the application to meet the goals of an applications. The Software design and architecture document encompasses the architectural design of the application under development. This document contains the details of architectural design and implementation of a software to manage the accounts and their associated accommodation reservations.

## 1.2 Scope

They underlying application is a standalone application that will provide a contract for console or GUI interface developed by other team of software developers. The current scope of application is to develop a backend API to manage and facilitate the customer accounts along with their accommodation reservations persisted to a file. The persistence is implemented in the form of a JSON file for a given account and each reservation will have its own file to store the reservation information. The application will implement a manager class that exposes all the application logic through the methods of Manager class that can be called from outer Graphical User Interface or Command Line Interface. In this way the application’s scope is limited to the backend programming for the standalone customer accounts and accommodation reservation management.

## 1.3 Overview

This software development document covers the design aspects of the stand-alone account and accommodation management application. This document is divided into 8 sections and various sub-sections under the main sections. Each section defines and explains the various aspects of the software development process. Normally large-scale application development process follows some high level of software development model such as waterfall or the modern Scrum Agility framework but being a small-scale application going to be developed as a monolithic application that will be used by other client apps and user interfaces.

## 1.4 Reference Material

[1] What is Scrum? <https://www.scrum.org/resources/what-is-scrum>

[2] What is Software Engineering? <https://www.castsoftware.com/glossary/what-is-software-engineering-definition-types-of-basics-introduction>

## 1.5 Definitions and Acronyms

|  |  |
| --- | --- |
| Term | Definition |
| SDD | Software Design Document |
| Monolithic | Huge, Giant or massive |
| Scrum | Scrum is a lightweight project management framework. |
| Waterfall | The waterfall is a Software development life cycle model. |
| SOLID | First five principles of object-oriented design. |
| SoC | Separation of Concern SOLID design Principle. |

# 2. System Overview

The underlying project is going to be developed using Object-Oriented programming paradigm. The application implementation is limited to entities and control classes as the User Interface development is outside the scope of this application. The application has been designed by utilizing one of the SOLID “Separation of Concern” - SoC. The applications help facilitate the User Interfaces by exposing the API methods from the Manager Class. The class provides all the signatures and contracts to help integrate the application into outer user interfaces. The application also provides a way to manage the accounts and accommodation data by saving it to the files in a directory. The internal logic and control of application is transparent to the User Interface which only concerns with the Manager class methods and call them to interact with the application. The Manager class in turn interact with the logic of entire application from creating and managing instances of the customer accounts and their accommodation reservation requests. The application has implemented accommodation entities in a hierarchy where Reservation class encapsulates all the major information about the reservation including its accommodation details and addresses. The three concrete classes have been visualized to facilitate creation of reservation objects for different accommodation types such as House Reservation, Cabin Reservation, and Hotel Reservation.

# 3. System Architecture

## 3.1 Architectural Design

The following is the class diagram of the Accounts and Accommodation Reservation System.

A picture containing text, screenshot, document

Description automatically generated

## 3.2 Decomposition Description

The application UI will create an instance of the Manager class to interact with the Account and Accommodation Reservation System. The Manager class will load all the existing accounts and their associated accommodation reservations from the files stored on the hard disk. As soon as the Manager class instance is created this persistent data will be made available to the System UI or CLI Interface through the Manager class methods. The data is managed through serialization and deserialization from individual files where each account has its own file and similarly each of the reservation is stored in its individual file in JSON format.

The System UI can add new Account into the Accounts and Accommodation reservation system by calling a method from the Manager class. The System UI can add new Reservation to an Existing Customer Account. The duplication of a Customer Account and a Reservation is not allowed. The System UI can also update an existing account, an existing reservation. The System UI can also cancel an existing reservation. This way the System UI can perform CRUD operations on the Account and the associated Reservations.

The System UI can perform various utility functions. System UI can list all the accounts by calling a function from the Manager class. The System UI can get to calculate Price per night for a given Reservation. The System UI can also calculate total cost of a given reservation by calling a method from the Manager class. The System UI can display complete details of an account or a reservation by calling a corresponding function to get an account, or a reservation based on the account number and the reservation number from Manager class.

## 3.3 Exception Handling

The Accounts and Accommodation Reservation System has been implemented with robust Exception Handling paradigms to counter exceptional situation that may arise during the runtime of the application. There is various situation when an exceptional case may arise such as invalid parameters passed to the application, certain file cannot be found or read while loading entities of an Account or its associated Reservation, the reservation object cannot be updated or marked as complete or cancelled. An exception is implemented to handle the object duplication in case of both an account or a reservation is attempted to be created. To handle this scenario DuplicateObjectException class has been implemented that will raise an exception in case of an attempt is made to create a duplication of an Account or Reservation. In the scenario when an input file is not found or cannot be read while loading an Account or a Reservation, the IllegalLoadException will be thrown to indicate that something went wrong while loading or reading from the file of an account or reservation. In case the System UI cannot update, cancel or complete the reservation an exception of the form IllegalOperationException will be thrown to indicate that an Illegal Operation carried out within the System. An IllegalArgumentException or IllegalStateExceptionin will be thrown in case of invalid parameters or other general exceptions occur.

## 3.4 Design Rationale

The Accounts and Accommodation Reservations application has been developed with object-oriented programming paradigms and following the SOLID principles of object-oriented programming especially the separation of concern. The data has been persisted to files. Various Exception classes has been introduced to handle the exceptional situations during the runtime of the application. The application has been extensible to accommodate future changes and enhancements. By adapting separation of concern in persisting application data, in future the application can adapt XML based file management system or introduce a database to save the information regarding accounts and reservation. No changes will be needed to the program logic as the data persistent layer is transparent to the application’s logic.

# 4. Data Design

## 4.1 Data Description

The Accounts and Accommodation Reservation System will save the Accounts and Reservation data into appropriate JSON files. The convention of saving information is followed:

1. A dedicated hard coded root directory is used to store all the information. For example:   
     
   *C:/tmp-umuc/accounts/*
2. Each account will have its subdirectory named after the Account Number. For Example: *A123456789*
3. The account information is saved in a file named acc-AccountNumber.json inside this account folder as follows:  
     
   *C:/tmp-umuc/accounts/A123456789/acc- A123456789.json*
4. Each of the Reservation associated with this account goes into a separate file inside this accounts folder. The file name comprised of res-ReserationNumber.json as follows:  
     
   *C:/tmp-umuc/accounts/A123456789/res-CAB12345678.json*

## 4.2 Data Dictionary

Account file format - one account per file

|  |
| --- |
| [  {  "accountNumber": accountNumber,  "mailingAddress": {  "street": street,  "state": state,  "zip", zip,  "country": country  },  "emailAddress": emailAddress,  "phoneNumber" : phoneNUmber,  "listOfReservations": [  reservationNumber1,  reservationNumber2,  ……  ReservatinoNumberN  ]  }  ] |

Reservation File Format – one reservation per file

|  |
| --- |
| {  "reservationNumber": reservationNumber,  "accountNumber": accountNumber,  "physicalAddress: {physicalAddress},  "mailingAddress: {mailingAddress},  "checkInDate": checkInDate,  "numberOfNights": numberOfNights,  "numberOfBeds": numberOfBeds,  "numberOfBathrooms": numberOfBathRooms,  "lodgingSize": lodgingSize,  "status": status  } |

# 5. Component Design

|  |  |
| --- | --- |
| **Class Name** | Account |
| **Description/Purpose** | This class creates objects of Customer Account for the reservation of the accommodations. |
| **Class Modifiers** | public |
| **Class Inheritance** | None |
| **Class Attributes** | **accountNumber** – The account number to identify this account  **mailingAddress** – instance of Address class. Represents Customer’s mailing address.  **emailAddress** – Email Address of the client.  **phoneNumber** – The phone number of the client.  **listOfReservations** – An array list of reservation number. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors; single and four parameter constructors. The single parameter constructor initializes the object with default values and take account number as parameter. The four parameter constructors take accountNumber, mailingAddress, emailAddress and phoneNumber as parameter and initialize the object with these values. Both constructors initialize empty list of reservations.  **getAccountNumber**: Accessor method for the account number attribute.  **getMailingAddress:** Accessor method for the account’s mailing address  **setMailingAddress:** Mutator method to set or update the mailing address.  **getEmail:** The accessor method for the account’s email address  **setEmail:** The mutator method to set or update the email address.  **getPhone:** The accessor method for the account’s phone number.  **setPhone:** The mutator function to set or update the account’s phone number.  **addReservation:** This function adds a reservation made by this client to the reservation list.  **getReservations:** The accessor method for the account’s reservation list.  **clone:** This function creates a deep copy of the Account object.  **toString:** This function returns the String representation of this function. |

|  |  |
| --- | --- |
| **Class Name** | Address |
| **Description/Purpose** | The class to create address objects such as Mailing address or Physical Address |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | **street:** Represents the street address.  **city:** Represents the address city.  **state:** Represents the address state.  **zipcode:** Represents the address’s zip code.  **country:** Represents the country of address. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors defined for this class. The default constructor initializes the attributes to their default values. The parameter constructor takes values for each of the attribute and initialize them to the parameter values.  **getStreet:** The accessor function to get the street address.  **setStreet:** The mutator function to set or update the street of this object.  **getCity:** The accessor function to get the City of this address.  **setCity:** The mutator function to set or update the city of this object.  **getState:** The accessor function to get the state of this address.  **setState:** The mutator function to set or update the state of this object.  **getZipcode:** The accessor function to get the zip code of this address.  **setZipcode:** The mutator function to set or update the zip code of this object.  **getCountry:** The accessor function to get the country of this address.  **setCountry:** The mutator function to set or update the Country of this object.  **clone:** Function to create a deep copy of this object.  **toString:** Function to get the String representation of this object. |

|  |  |
| --- | --- |
| **Class Name** | Reservation |
| **Description/Purpose** | Abstract class for the Reservation of the Accommodation. |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | **reservationNumber**: The reservation number of this reservation.  **accountNumber**: The account number of the customer.  **physicalAddress**: The physical address of the accommodation.  **mailingAddress**: The mailing address of the accommodation.  **checkInDate**: The check in date of the reservation.  **numberOfNights**: The number of nights reservation is made for.  **numberOfBeds**: The number of beds in the accommodation.  **numberOfBathrooms**: The number of bathrooms in the accommodation.  **lodgingSize**: The area of the accommodation.  **status**: The Status of the accommodation. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default values. A parameter constructor that takes all the attributes values from the parameters and initialize them.  **getReservationNumber:** The accessor method to get the reservation number.  **setReservationNumber:** A mutator function to update or set the Reservation Number  **getAccountNumber:** The accessor method to get the account number.  **setAccountNumber:** A mutator function to update or set the account number  **getPhysicalAddress:** The accessor method to get the Physical Address.  **setPhysicalAddress:** A mutator function to update or set the Physical Address  **getMailingAddress:** The accessor method to get the Mailing Address.  **setMailingAddress:** A mutator function to update or set the Mailing address  **getCheckInDate:** The accessor method to get the Check in Date.  **setCheckInDate:** A mutator function to update or set the Check in date  **getNumberOfNights:** The accessor method to get the number of nights  **setNumberOfNights:** A mutator function to update or set the number of nights  **getNumberOfBeds:** The accessor method to get the number of beds.  **setNumberOfBeds:** A mutator function to update or set the number of beds.  **getNumberOfBathrooms:** The accessor method to get the number of bathrooms  **setNumberOfBathRooms:** A mutator function to update or set the number of rooms  **getLodgingSize:** The accessor method to get the lodging size.  **setLodgingSize:** A mutator function to update or set the lodging size.  **getStatus:** The accessor method to get the status of the reservation.  **setStatus:** A mutator function to update or set the status of reservation.  **clone:** A function to produce the deep copy of reservation object.  **toString:** Function to produce the String representation of the Reservation.  **calculatePrice:** Abstract function to calculate the price of this reservation.  **updateReservation**: Functions updates this reservation. |

|  |  |
| --- | --- |
| **Class Name** | CabinReservation |
| **Description/Purpose** | A concrete class that creates an instance of the Cabin Reservation. |
| **Class Modifiers** | Public |
| **Class Inheritance** | Reservation |
| **Class Attributes** | **fullKitchen:** Represents whether not cabin has a full kitchen.  **loft:** Represents whether this cabin has a loft. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default values. A parameter constructor that takes all the attributes values from the parameters and initialize them.  **toString:** Function to get the String representation of this reservation object.  **clone:** Function to create a deep copy of this reservation object.  **calculatePrice:** Abstract function to calculate the price of this reservation.  **updateReservation**: Functions updates this reservation. **Constructors:** some  **hasLoft:** An accessor method to check cabin has loft or not.  **hasFullKitchen:** An accessor method to check cabin has full kitchen or not.  **setLoft:** Function to set or update whether cabin has loft or not.  **setFullKitchen:** Function to set or update whether cabin has full kitchen or not. |

|  |  |
| --- | --- |
| **Class Name** | HouseReservation |
| **Description/Purpose** | This class creates an instance of HouseReservation Object. |
| **Class Modifiers** | Public |
| **Class Inheritance** | Reservation |
| **Class Attributes** | **numberOfFloors:** Represents how many floors the house has. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default values. A parameter constructor that takes all the attributes values from the parameters and initialize them.  **calculatePrice:** Abstract function to calculate the price of this reservation.  **updateReservation**: Functions updates this reservation.  **toString:** Function to get the String representation of this reservation object.  **clone:** Function to create a deep copy of this reservation object.  **getNumberOfFloors:** An accessor method to get the number of floors of this house.  **clone:** A mutator method to set or update the number of floors of this house. |

|  |  |
| --- | --- |
| **Class Name** | HotelReservation |
| **Description/Purpose** | This class creates an instance of the Hotel Reservation. |
| **Class Modifiers** | Public |
| **Class Inheritance** | Reservation |
| **Class Attributes** | **kitchenette:** Represents whether this hotel reservation has kitchenette or not. |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default values. A parameter constructor that takes all the attributes values from the parameters and initialize them.  **calculatePrice:** Abstract function to calculate the price of this reservation.  **updateReservation**: Functions updates this reservation.  **toString:** Function to get the String representation of this reservation object.  **clone:** Function to create a deep copy of this reservation object.  **hasKitchenette**: A function to test whether this hotel reservation has a kitchenette or not.  **setKitchenette**: Function to set or update whether this hotel has a kitchenette or not. |

|  |  |
| --- | --- |
| **Class Name** | IllegalLoadException |
| **Description/Purpose** | This class will create an Exception object whenever a load operation fails for the account or its associated reservation. |
| **Class Modifiers** | public |
| **Class Inheritance** | RuntimeException |
| **Class Attributes** | None |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default Exception message. A parameter constructor that takes a custom exception message that will be used while throwing this exception.  **toString:** Function to get the String representation of this Exception object. |

|  |  |
| --- | --- |
| **Class Name** | DuplicateObjectException |
| **Description/Purpose** | This class will create an instance of Exception when an attempt is made to create a duplicate object of an account or reservation. |
| **Class Modifiers** | public |
| **Class Inheritance** | RuntimeException |
| **Class Attributes** | None |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default Exception message. A parameter constructor that takes a custom exception message that will be used while throwing this exception.  **toString:** Function to get the String representation of this Exception object. |

|  |  |
| --- | --- |
| **Class Name** | IllegalOperationException |
| **Description/Purpose** | This instance of this Exception class will be used to raise an error while an operation of Cancelling or completing a reservation fails. |
| **Class Modifiers** | public |
| **Class Inheritance** | RuntimeException |
| **Class Attributes** | None |
| **Exceptions Thrown** | None |
| **Class Methods** | **Constructors:** There are two constructors. A default constructor to initialize the class objects with default Exception message. A parameter constructor that takes a custom exception message that will be used while throwing this exception.  **toString:** Function to get the String representation of this Exception object. |

|  |  |
| --- | --- |
| **Class Name** | FileIO (Interface) |
| **Description/Purpose** | This is an interface that provides a contract to serialize and deserialize files. This Interfaces implements the concept the Separation of Concern. |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | None |
| **Exceptions Thrown** | **IllegalLoadException**: when an attempt is made to load/read an account or reservation, but its file not found or reading error is raised. |
| **Class Methods** | **serializeAccount**: This function will serialize all the accounts  **deserializeAccounts**: This function will deserialize all the accounts.  **serializeReservations**: This function will serialize all the Reservations.  **deserializeAccounts**: This function will deserialize all the reservations for all the accounts. |

|  |  |
| --- | --- |
| **Class Name** | JsonDao |
| **Description/Purpose** | The instance of this class will implement the FileIO Interface and provide implementation of Custom File Io. In this case the JSON file IO. |
| **Class Modifiers** | Public |
| **Class Inheritance** | FileIO |
| **Class Attributes** | None |
| **Exceptions Thrown** | IllegalLoadException: when an attempt is made to load/read an account or reservation, but its file not found or reading error is raised. |
| **Class Methods** | **serializeAccount**: This function will serialize all the accounts  **deserializeAccounts**: This function will deserialize all the accounts.  **serializeReservations**: This function will serialize all the Reservations.  **deserializeAccounts**: This function will deserialize all the reservations for all the accounts. |

|  |  |
| --- | --- |
| **Class Name** | Status (enum) |
| **Description/Purpose** | Provides data types for the status of the Reservation. |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | **DRAFT**: Indicates that reservation status is DRAFT (default)  **COMPLETED**: Indicates that reservation status is COMPLETED  **CANCELLED** Indicates that reservation status is CANCELLED |
| **Exceptions Thrown** | None |
| **Class Methods** | None |

|  |  |
| --- | --- |
| **Class Name** | ReservationSystem |
| **Description/Purpose** | A Driver class that will provide a main method calling all the methods of the Manager class to test the functionality of the application. |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | None |
| **Exceptions Thrown** | None |
| **Class Methods** | Main: Main method the entry point of the program. |

|  |  |
| --- | --- |
| **Class Name** | Manager |
| **Description/Purpose** | The instance of this class will expose all the functionality of Accounts and Accommodation Reservation Management to the User Interface. |
| **Class Modifiers** | Public |
| **Class Inheritance** | None |
| **Class Attributes** | **DIRECTORY**: hard coded constant for the directory to store accounts and reservation files.  **listOfReservations**: List of all the Reservations for all the accounts.  **listOfAccounts**: List of all the accounts and their associated reservations.  **fileIO**: The object to serialize and deserialize accounts and reservations. |
| **Exceptions Thrown** | **IllegalLoadException**: when an attempt is made to load/read an account or reservation, but its file not found or reading error is raised.  **DuplicateObjectException**: When an attempt is made to create a duplicate customer account or a duplicate reservation. In both cases account number and reservation number must be unique.  **IllegalOperationException**: When an attempt is made to cancel a reservation or complete a reservation and operation fails. This exception will be raised.  **IllegalArgumentException**: When illegal parameter is passed to a function.  **IllegalStateException**: When the state of an object cannot be changed. |
| **Class Methods** | **Constructor**: The default constructor that initialize the Manager class object and loads all the Accounts and their associated reservations from the files.  **addNewAccount**: This function adds new account into the accounts list. If operation is successful, the new account file will be created.  **addReservation**: This function receives a reservation object and add it to the account’s list of reservation. If operation is successful a new file for the reservation will be created.  **serializeAccounts**: This function serializes all the accounts and their associated reservations.  **deserializeAccounts:** This function deserializes the accounts and reservation files and create all the account objects and their associated reservations added to the appropriate lists.  **updateReservation**: This functions updates a given reservation object.  **cancelReservation**: This function cancels a given reservation object.  **getAccounts**: this function returns the list of all the accounts.  **getAccount**: This function returns an account identified by account number.  **updateAccount**: This function updates an account.  **completeReservation**: This function completes a reservation.  **calcPricePerNight**: This function calculates the night cost of a reservation.  **calcTotalPrice:** This function calculates the total price of a reservation.  **getReservation**: This function returns a particular reservation identified by an account number and a reservation number. |

# 6. Human Interface Design

## 6.1 Overview of User Interface

The System UI will use the Manager class object to interact the system. The manager class provides access to all the underlying functionality of the Accounts and Accommodation Reservation System. The System UI can create new Account by entering the account details and pass the object to a function in the Manager class that will take care of adding the new Account object and save it to the appropriate JSON file. Similarly, the System UI can create, update, cancel and complete a Reservation by inputting appropriate details of the Reservation for a given Account and pass the information to the Manager class methods. The Manager class methods act accordingly and update the Reservation information and save it to the appropriate JSON file. The System UI can get information on an Account, Reservation, all the reservations of a given account and can display them in some tabular UI component or individual object information can be displayed on the UI. In short, the System UI can have access to all the underlying logic, information and details of the system through the usage of the Manager class Object.

## 6.2 Screen Images

Account Management Screen

Graphical user interface

Description automatically generated

Display All accounts

Graphical user interface, text, application

Description automatically generated

Reservation Management Screen

Graphical user interface

Description automatically generated

## 6.3 Screen Objects and Actions

The System UI will create a Manage class Object that will expose the underlying functionality of the Accounts and Accommodation Reservation System to the System UI. The User can enter Account information to create new account and pass the account object created from the input details to the Manager class’s addAccount () method. This function will persist the account information in JSON file accordingly to the File format and naming convention of the account file. The user can create new Reservation for the selected account by enter reservation information and pass the reservation object to the Manager class method addReservation (). The function stores the reservation information into appropriate JSON file named according to the naming convention of the reservation files. The application user can update, cancel and complete the selected reservation by passing the information to the Manager class. The Manage class will update the memory object for the corresponding reservation and update the Reservation File accordingly. The System UI mainly interact with three objects the Manager class objects, The Account Object and Reservation Object. The System UI will not manipulate any of the Reservation and Account object directly. It has to call the Manager class methods to update and manage these objects.

# 7. Requirements Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Class | Constructor/Method | Attribute |
| Reservation Management | Manager | Manager() |  |
| Encapsulates the account details | Account |  | Customer’s Account |
| Encapsulates the Reservation details | Reservation |  | Reservation made for the Customer’s Account |
| Account can have multiple Reservations | Reservation |  | ArrayList of Reservations |
| Account can have House reservation | HouseReservation |  |  |
| Account can have Hotel Reservation | HotelReservation |  |  |
| Account can have Cabin reservation | CabinReservation |  |  |
| Add account into the system. | Manager/Account | addNewAccount(Account account) |  |
| Add reservation for an account | Manager/Account/Reservation | addReservation (Reservation reservation) |  |
| Update a given Reservation | Manager/Reservation | updateReservation(Reservation reservation) |  |
| Update a given account. | Manager/Account | updateAccount(Account account) |  |
| Cancel a Reservation | Manager/Reservation | cancelReservation(Reservation reservation) |  |
| Complete a Reservation | Manager/Reservation | completeReservation(Reservation reservation) |  |
| Collection information on a Reservation | Manager/Account/Reservation | getReservation(String accountNumber, String reservationNumber) |  |
| Calculate one night cost of reservation | Manager/Reservation  CabinResrevatrion  HouseResrvation  HotelReservation | calcPricePerNight(String reservationNumber) |  |
| Calculate total cost of a given Reservation | Manager/Reservation  CabinResrevatrion  HouseResrvation  HotelReservation | calcTotalPrice(String reservationNumber) |  |
| Get information and details of an account | Manager/Account | getAccount(String accountNumber) |  |
| Get all the information on accounts | Manager/Account | getAccounts() | ArrayList of accounts |

# 8. Appendices