UMUC Inc
Lodging & Reservation Management System (LRMS)
Software Design Document

Name (s): Benjamin Sutter
Lab Section: 9040
Date: (06/19/2022)
Software Design Document

Table of Contents

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the Lodging & Reservation Management System (LRMS). The software allows for the management of trips and reservations to help users or travel agents plan their trip as they go. There are three types of reservations available: Airplane, hotel, and rental car. The user has full control of how they book their reservations and have the ability to edit reservations if needed. The user is also able to see the running total for how expensive their trip will be so they can budget or edit reservations accordingly.

Users will first fill out some personal information to help identify them as the organizer for the trip. Then they will fill in certain info about the trip like the theme of the trip and the start/end date of the trip. Then they will choose a reservation to start the trip. Once the trip has been created, they have full view over every aspect about the trip. They also can save the trip to a file so that it may be loaded later. This means that they can plan multiple different trips at the same time if needed by saving them to different files.

1.2 Scope

Since this is the initial prototype of the project, only the back-end functionality of the software will be implemented for release. All functionality mentioned in the previous section will be implemented and test case files will be provided to show that the back-end functionality indeed functions. This means that a UI is outside the scope of the project (there is an example of what the UI may look like in section 6). One future feature that would be nice but is outside of the scope is the ability to load in a trip file and see that information with graphics and images.

1.3 Overview

The purpose of this Software Design Document is to provide a detailed plan of what the architecture of the software should look like and give insight as to how it should be implemented.

This document is meant to give a developer all of the pieces to the puzzle and show them what the completed puzzle should look like so that all they have to do is complete the puzzle and ensure all the pieces fit together correctly.

It is broken up into various sections for easier navigation and organization. The system overview section will provide insight into the major functionality, context, and design of the software. The system architecture shows what the intended architecture of the software is and detailed aspects about the architecture. The data design section provides an example of what a completed file should look like and includes a data dictionary of all classes, attributes, and methods. The component design section provides further explanation into the classes and what each attribute

and method does. The human interface design section gives an example of what the UI might look like if it were implemented. Lastly, the requirements matrix provides a way to visualize all of the requirements and what classes/attributes are fulfilling them.

1.4 Reference Material
The project referenced skeleton code and UML provided in the class resources.

1.5 Definitions and Acronyms
Lodging & Reservation Management System (LRMS)

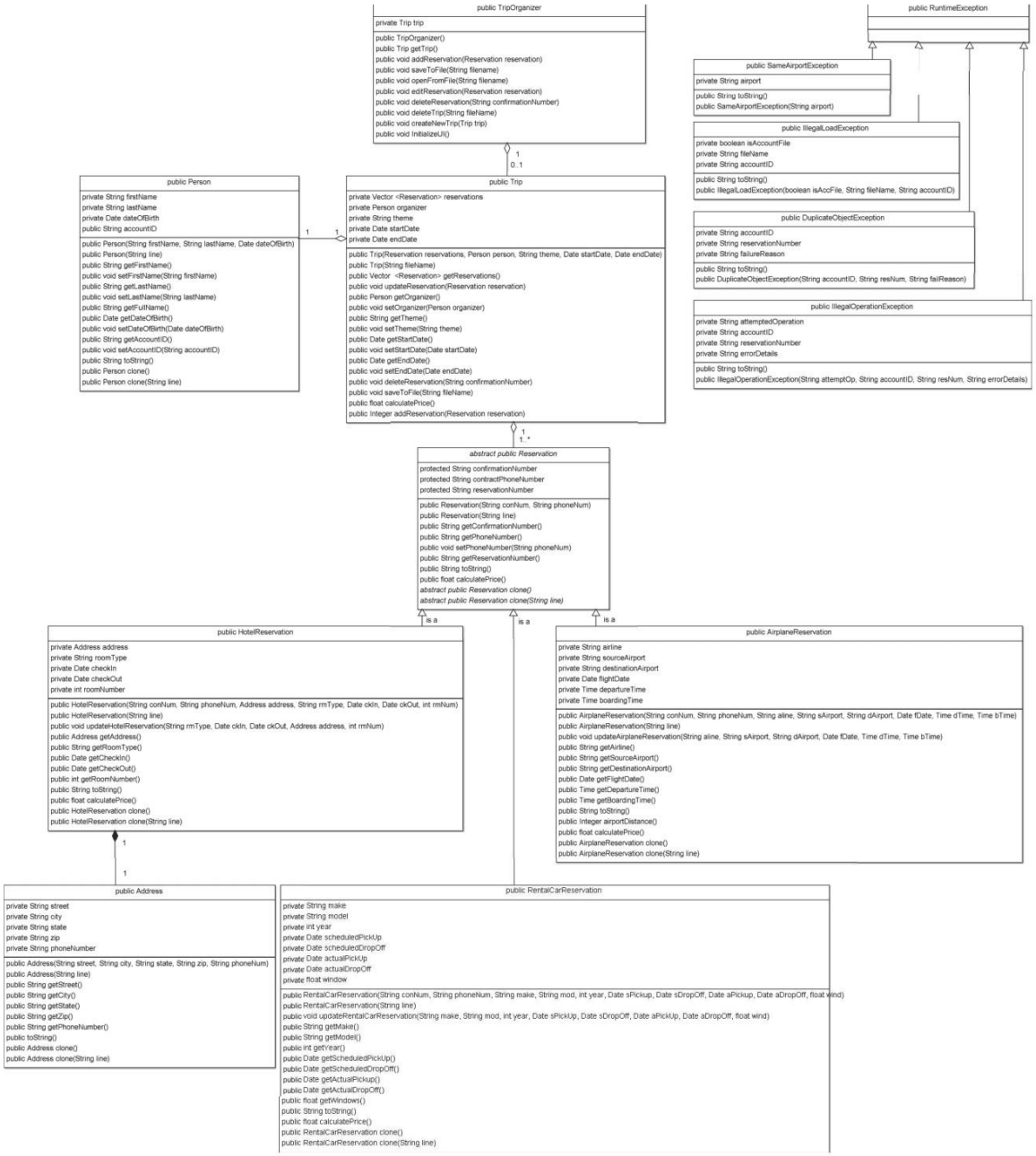
2. SYSTEM OVERVIEW
The LRMS is meant to provide an avenue for users to organize their trips, information about their trips, and have full control over each aspect of planning a trip. Information about their trip includes reservation information and running cost for all reservations within the trip. Users can view these reservations, edit aspects of the reservations if they see fit, and update their trip accordingly. Users can also add or remove any reservations and edit their trip even after it has been saved to a file. These trips are represented by xml files and can be saved and loaded using the TripOrganizer class which will eventually have a UI aspect to accompany it. Users can load in previous trips for viewing or editing. With the ability to save and load trip files comes the ability to share these files with others who are also going on the trip so they can provide feedback or edits.

All of the classes, their attributes, and their methods have been designed to facilitate this functionality above and provide a simple yet effective way to organize trips and reservations. Reservations are easily grouped together in the Trip object because they are all subclassed from the main Reservation parent class. Careful consideration is made in terms of what attributes are necessary and what classes should be able to access or edit these attributes.

This is a very high-level summary of the LRMS architecture and inner working of the software. In future sections, much greater detail will be given into the architecture and data designs of the system.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design
The following image is the class diagram of the Trip Organizer software architecture.



3.2 Decomposition Description

On system startup UI will create an instance of TripOrganizer. Then UI will either create a new Trip object and call createNewTrip method or it will call openFromFile method to load an existing trip. The openFromFile method will read file data and call Trip constructor to create the object which in term will call the constructors of the reservation child classes to load their own data. There will be multiple checks to ensure that the Trip is valid. If in invalid trip is recognized the user will be notified accordingly.

Once a Trip is loaded into TripOrganizer, UI can request to add a new Reservation that it

created for that trip through addReservation method, edit an existing reservation through editReservation method, and delete reservation through deleteReservation method. The TripOrganizer will do validation and then call the appropriate Trip methods to perform the correct action.

UI can also request an existing trip is deleted, which will close the loaded trip and delete the associated file.

When Trip object is created, it must be provided with a reservation object, instance of Person, and theme. Therefore, Trip must always have at least one reservation and on deletion, the last reservation cannot be removed. A Trip object can also be loaded from a file using Trip(String filename) constructor and can be saved to a file using saveToFile method.

To safeguard the Trip and Reservation objects, the TripOrganizer will use clone methods to store the Trip and Reservation data and return the object data to UI such that UI cannot change them directly and need to call the organizer to request changes.

3.3 Exception Handling

The system will implement for new exceptions that all children classes of RuntimeException

SameAirportException will be thrown when the same airport is used for the source and destination. It will take a string a parameter and save the name of an airport and it will set the exception message to “The source and destination airports have the same value XXX” where XXX is the name of the airport. The method toString() will return string with the name of the exception and the same message. The exception will be thrown when airport reservation (class Airplane Reservation) is created (in constructor) or updated (method updateAirplaneReservation).

IllegalLoadException will be thrown if a file that does not exist is attempted to be loaded. The exception message will indicate what failed (account file versus reservation file) and the filename that could not be loaded. The message will include the account’s number that was being loaded.

DuplicateObjectException will be thrown if a reservation or account is being added and a duplicate already exists. The generated exception message will indicate the number of the account and/or reservation number and why it failed.

IllegalOperationException will be thrown if a reservation is cancelled or completed by is not finalized. The generated exception message should indicate the operation that was attempted, account id, reservation number, and details why exactly it failed.

Any other issues or errors will use Java builtin exceptions such as IllegalArgumentException for invalid parameters and IllegalStateException when state of

the object cannot be changed. Any code that throws these exceptions will pass a meaningful message in order to help the user better understand what caused the error.

3.4 Design Rationale

The design for the software is minimal, with each class having exactly what is required of it. Each class where attributes should be accessed have getter methods, and some classes that require those attributes to be changed will have setter methods. All local variables are also private/protected to restrict access aside from the getter/setter methods.

The TripOrganizer class is essentially an interface into the trip object and editing/deleting facets about it. It also serves as the manager for grouping all things together and ensuring all pieces of the software interact with each other as they should.

Since there is a lot of commonalities within the reservation objects, it only makes sense to have the parent class of Reservation to isolate that commonality. Because each of the reservation have their differences, they are child classes of the reservation which allows them to add their own additional variables and methods for use.

4. DATA DESIGN

4.1 Data Description

For the Trip system, UI will be the one to manage Trip files (naming and selecting where stored). Each Trip object will be saved in a separate file using XML tag format.

For example: Trip-R12345.txt

```
<trip>Reunion</trip><organizer>Mike</organizer>
<airplane><confirmationNum>R12345</confirmationNum><contactNum>123-456-
7890</contactNum><airline>United</airline><sourceairport>IAD</sourceairport><d
estinationairport>ORL</destinationairport><flightdate>01/05/2014</flightdate><
/airplane>
<hotel><confirmationNum>RH123</confirmationNum><contactNum>123-456-
7890</contactNum><roomtype>Double</roomtype><address><street>123 Main
St</street><city>Rockville</city><state>MD</state><zip>23123</zip></address><c
heckin>01/01/2014</checkin><checkout>01/05/2014</checkout></hotel>
<rentalcar><confirmationNum>RC1245</confirmationNum><contactNum>777-456-
0000</contactNum><make>Ford</make><model>Torus</model><scheduledpickup>01/01/2
014</scheduledpickup><scheduleddropoff>01/05/2014</scheduleddropoff><actualpic
kup>01/01/2014</actualpickup><actualdropoff>01/05/2014</actualdropoff><window>
2.5</window></rentalcar>
```

4.2 Data Dictionary

Address

Attributes
private String street
private String city
private String state
private String zip
private String phoneNumber

Methods
public Address(String street, String city, String state, String zip, String phoneNum)
public Address(String line)
public String getStreet()
public String getCity()
public String getState()
public String getZip()
public String getPhoneNumber()
public String toString()
public Address clone()
public Address clone(String line)

AirplaneReservation

Attributes
private String airline
private String sourceAirport
private String destinationAirport
private Date flightDate
private LocalTime departureTime
private LocalTime boardingTime

Methods
public AirplaneReservation(String conNum, String phoneNum, String aline, String sAirport, String dAirport, Date fDate, LocalTime dTime, LocalTime bTime)
public AirplaneReservation(String line)
public void updateAirplaneReservation(String aline, String sAirport, String dAirport, Date fDate, LocalTime dTime, LocalTime bTime)
public String getAirline()
public String getDestinationAirport()
public Date getFlightDate()
public LocalTime getDepartureTime()
public LocalTime getBoardingTime()
public String toString()
public Integer airportDistance()

public float calculatePrice()
public AirplaneReservation clone()
public AirplaneReservation clone(String line)

DuplicateObjectException

Attributes
private String accountID
private String reservationNumber
private String failureReason

Methods
public String toString()
public DuplicateObjectException(String accountID, String resNum, String failReason)

HotelReservation

Attributes
private Address address
private String roomType
private Date checkIn
private Date checkout
private int roomNumber

Methods
public HotelReservation(String conNum, String phoneNum, Address address,
String rmType, Date ckIn, Date ckOut, int rmNum)
public HotelReservation(String line)
public void updateHotelReservation(String rmType, Date ckIn, Date ckOut, Address address)
public Address getAddress()
public String getRoomType()
public Date getCheckIn()
public Date getCheckOut()
public int getRoomNumber()
public String toString()
public float calculatePrice()
public HotelReservation clone()
public HotelReservation clone(String line)

IllegalLoadException

Attributes
private boolean isAccountFile
private String filename
private String accountID

Methods
public String toString()
public IllegalLoadException(boolean isAccFile, String fileName, String accountID)

IllegalOperationException

Attributes
private String attemptedOperation
private String accountID
private String reservationNumber
private String errorDetails

Methods
public String toString()
public IllegalOperationException(String attemptOp, String accountID, String resNum, String errorDetails)

Person

Attributes
private String firstName
private String lastName
private Date dateOfBirth

Methods
public Person(String firstName, String lastName, Date dateOfBirth)
public Person(String line)
public String getFirstName()
public void setFirstName(String firstName)
public String getLastName()
public void setLastName(String lastName)
public String getFullName()
public Date getDateOfBirth()
public void setDateOfBirth(Date dateOfBirth)
public String toString()
public Person clone()
public Person clone(String line)

RentalCarReservation

Attributes
private String make
private String model
private int year
private Date scheduledPickUp
private Date scheduledDropOff
private Date actualPickUp
private Date actualDropOff
private float window

Methods
public RentalCarReservation(String conNum, String phoneNum, String make, String mod, int year, Date sPickup, Date sDropOff, Date aPickup, Date aDropOff, float wind)
public RentalCarReservation(String line)
public void updateRentalCarReservation(String mak, String mod, Date sPickUp, Date sDropOff, Date aPickUp, Date aDropOff, float wind)
public String getMake()
public String getModel()
public int getYear()
public Date getScheduledPickUp()
public Date getScheduledDropOff()
public Date getActualPickup()
public Date getActualDropOff()
public float getWindows()
public String toString()
public float calculatePrice()
public RentalCarReservation clone()
public RentalCarReservation clone(String line)

Reservation

Attributes
protected String confirmationNumber
protected String contractPhoneNumber
protected String reservationNumber

Methods
public Reservation(String conNum, String phoneNum)
public Reservation(String line)
public String getConfirmationNumber()
public String getPhoneNumber()

public void setPhoneNumber(String phoneNum)
public String getReservationNumber()
public float calculatePrice()
public String toString()
public abstract Reservation clone()
public abstract Reservation clone(String line)

SameAirportException

Attributes
private String airport

Methods
public String toString()
public SameAirportException(String airport)

Trip

Attributes
private Vector <Reservation> reservations
private Person organizer
private String theme
private Date startDate
private Date endDate

Methods
public Trip(Vector <Reservation> reservations, Person person, String theme, Date startDate, Date endDate)
public Trip(String fileName)
public Vector <Reservation> getReservations()
public void updateReservation(Reservation reservation)
public Person getOrganizer()
public void setOrganizer(Person organizer)
public String getTheme()
public void setTheme(String theme)
public Date getStartDate()
public void setStartDate(Date startDate)
public Date getEndDate()
public void setEndDate(Date endDate)
public Integer addReservation(Reservation reservation)
public void deleteReservation(String confirmationNumber)
public void saveToFile(String fileName)
public float calculatePrice()

TripOrganizer

Attributes
private Trip trip

Methods
public TripOrganizer()
public Trip getTrip()
public void addReservation(Reservation reservation)
public void saveToFile(String filename)
public void openFromFile(String filename)
public void editReservation(Reservation reservation)
public void deleteReservation(String confirmationNumber)
public void deleteTrip(String fileName)
public void createNewTrip(Trip trip)

5. COMPONENT DESIGN

Trip Class

Information

Name: Trip
Description/Purpose: The trip class is the highest-level class for a single trip. It contains the complete interface for the organizer to interact with.
Modifiers: public
Inheritance: None

Attributes, Exceptions, and Methods

Attributes:
Reservations – Vector – this contains a list of reservations for the trip.
Organizer – Person - This is an object of type person that is the organizer of the trip.
Theme – String – This is the theme of the trip.
StartDate – Date – The date that the trip will begin.
EndDate – Date – The date that the trip will end.

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid
IllegalLoadException – Thrown when the file being loaded does not exist

DuplicateObjectException – Thrown if an identical reservation already exists when adding a new one
IllegalOperationException – Thrown if creating or cancelling a reservation is not finalized

Methods:
Constructors: there are two ways to create a new Trip object. If the user selects to create a new Trip, then constructor Trip(Person organizer, String theme, Reservation reservation) is called. If the user opens a Trip from a file, then constructor Trip(String fileName) is called.

public Trip(Vector <Reservation> reservations, Person person, String theme, Date startDate, Date endDate)
Validate parameters and throw IllegalArgumentException
Create a new Vector object for attribute reservations
Add reservation parameter value to reservations
Assign organizer object to attribute organizer
Assign theme parameter to attribute theme
Assign startDate parameter to the attribute startDate
Assign endDate parameter to the attribute endDate

public Trip (String fileName)
Throw an IllegalLoadException if the file does not exist
Create a new Vector object for attribute reservations
Read the first line of the file filename
    load the theme attribute
    call Person constructor for organizer data passing line and assign to attribute
Loop through all of the other lines of the file
    For each line look for xml tag:
        If the tag is <hotel>
            Create a new HotelReservation object, pass in the current line from the file
            Add the HotelReservation object clone to the reservations Vector
        If the tag is <rentalcar>
            Create a new RentalCarReservation object, pass in the current line from the file
            Add the RentalCarReservation object clone to the reservations Vector
        If the tag is <airplane>
            Create a new AirplaneReservation object, pass in the current line from the file
            Add the AirplaneReservation object clone to the reservations Vector
    Catch any exceptions and print error message to console and cleanup

public int addReservation (Reservation reservation)
Validate parameter is not null and throw IllegalArgumentException
Check if reservation exists and if it does throw exception DuplicateObjectException
Add reservation object clone to list of reservations
Return number of Reservation objects in list
Throw IllegalOperationException if the reservation can't be finalized.

public void updateReservation(Reservation reservation)
Validate the reservation
Match the reservations based on ID
If found, then clone the new reservation and update the reservation in the list
Otherwise fail method and give appropriate message (reservation not found)

public Vector <Reservation> getReservations()

Accessor method for reservations attribute

public Person getOrganizer()
Accessor method for organizer attribute

public void setOrganizer(Person organizer)
Mutator method for organizer attribute

public String getTheme()
Accessor method for theme attribute

public void setTheme(String theme)
Mutator method for theme attribute

public Date getStartDate()
Accessor method for startDate attribute

public Date getStartDate()
Mutator method for startDate attribute

public Date getEndDate()
Accessor method for endDate attribute

public void setEndDate(Date endDate)
Mutator method for endDate attribute

public void deleteReservation(String confirmationNumber)
Find reservation that matches confirmation number and delete it from the vector
If it does not exist, fail method and notify user accordingly
Throw IllegalOperationException if the reservation deletion can't be finalized.

public void saveToFile(String fileName)
Ensure file exists, if not then create it
Create a blank string: tmpString
Iterate through each reservation, call toString() and concatenate it onto tmpString
Open file for writing
Write tmpString to the file
Close file
Catch any errors and notify user accordingly

public float calculatePrice()
Create a blank float: tmpPrice
Iterate through each reservation, call calculatePrice() and add the value to tmpPrice
Return tmpPrice

AirplaneReservation Class

Information

Name: AirplaneReservation
Description/Purpose: This class creates object for reservation of a flight
Modifiers: public

Inheritance: Inherits from the Reservation class

Attributes, Exceptions, and Methods

Attributes:
String airline – name of airline
String sourceAirport – name of airport from where we depart
String destinationAirport – name of destination airport
Date flightDate – date of the flight
LocalTime departureTime – When the plane leaves the airport
LocalTime boardingTime – When the plane begins boarding

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid
SameAirportException – Thrown when source/destination are given or modified and both values are for the same airport

Methods:
Constructors: there are two ways to create a new AirplaneReservation object. If the caller selects to create a new AirplaneReservation, then constructor AirplaneReservation with parameters for attributes is called. If the caller wants to load existing reservation information from a String, then constructor AirplaneReservation(String line) is called.

public AirplaneReservation(String conNum, String phoneNum, String aline, String sAirport, String dAirport, Date fDate, LocalTime dTime, LocalTime bTime)
Call parent's constructor using super(conNum, conNum) - parent will validate common attributes
Validate parameters for unique airplane attributes and throw IllegalArgumentException if they are invalid
Check if sAirport and dAirport are the same value and if so, throw SameAirportException exception
Load method parameters to object's attributes

public AirplaneReservation(String line)
Call parent's constructor to parse and load common attributes: super(line)
Check for substring <sourceairport> and assign value to tmpSource attribute
Check for substring <destinationairport> and assign value to tmpDestination attribute
Check if values are the same and if so, throw SameAirportException
Check for substring <airline> and assign value to tmpAirline attribute
Check for substring <flightdate> and assign value to tmpFlightDate attribute
Check for substring <departuretime> and assign value to tmpDepartureTime attribute
Check for substring < boardingtime > and assign value to tmpBoardingTime attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public void updateAirplaneReservation(String aline, String sAirport, String dAirport, Date fDate, LocalTime dTime, LocalTime bTime)
Check if sAirport and dAirport are the same value and if so, throw SameAirportException
Load method parameters to object attributes

public String getAirline()
Accessor method for airline attribute

public String getSourceAirport()
Accessor method for sourceAirport attribute

public String getDestinationAirport()
Accessor method for destinationAirport attribute

public Date getFlightDate()
Accessor method for flightDate attribute

public LocalTime getDepartureTime()
Accessor method for departureTime attribute

public LocalTime getBoardingTime()
Accessor method for boardingTime attribute

public String toString()
return a string with tags and values for each attribute that has the following values and format:

"<airplane>
call parent's toString method and append the returned String here
<sourceairport>sourceAirport</sourceairport>
<destinationairport>destinationAirport</destinationairport>
<flightdate>flightDate</flightdate>
<airline>airline</airline>
<departuretime>departuretime</departuretime>
<boardingtime>boardingtime</boardingtime>
</airplane>"

private int airportDistance()
if airports are 'IAD' and 'ORL' return 723
if airports are 'IAD' and 'BWI' return 100
if airports are 'IAD' and 'NYC' return 273
if airports are 'ORL' and 'BWI' return 776
if airports are 'ORL' and 'NYC' return 842
if airports are 'BWI' and 'NYC' return 221
else return 0

public float calculatePrice()
return airportDistance() times 2

public AirplaneReservation clone()
Returns a copy of the current AirplaneReservation object

public AirplaneReservation clone(String line)
Given a line from the file, return a copy of the current AirplaneReservation by using the line constructor

HotelReservation Class

Information

Name: HotelReservation
Description/Purpose: This class creates object for reservation of a hotel
Modifiers: public
Inheritance: Inherits from the Reservation class

Attributes, Exceptions, and Methods

Attributes:
Address address – The address of the hotel
String roomType – The type of room that was reserved
Date checkIn – The check-in date for the reservation
Date checkOut – The check-out date for the reservation
int roomNumber – The number of the room that was reserved

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid

Methods:
Constructors: there are two ways to create a new HotelReservation object. If the caller selects to create a new HotelReservation, then constructor HotelReservation with parameters for attributes is called. If the caller wants to load existing reservation information from a String, then constructor HotelReservation (String line) is called.

public HotelReservation(String conNum, String phoneNum, Address address, String rmType, Date ckIn, Date ckOut, int rmNum)
Call parent's constructor using super(conNum, conNum) - parent will validate common attributes
Validate parameters for unique hotel attributes and throw IllegalArgumentException if they are invalid
Load method parameters to object's attributes

public HotelReservation(String line)
Call parent's constructor to parse and load common attributes: super(line)
Check for substring <address> and call the Address line constructor
Check for substring <roomtype> and assign value to tmpRoomType attribute
Check for substring <checkin> and assign value to tmpCheckin attribute
Check for substring <checkout> and assign value to tmpCheckOut attribute
Check for substring <roomnumber> and assign value to tmpRoomNumber attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public void updateHotelReservation(String rmType, Date ckIn, Date ckOut, Address address)
Load method parameters to object attributes

public Address getAddress()
Accessor method for address attribute

public String getRoomType()
Accessor method for roomType attribute

public Date getCheckIn()
Accessor method for checkIn attribute

public Date getCheckOut()
Accessor method for checkOut attribute

public int getRoomNumber()
Accessor method for TING attribute

public String toString()
return a string with tags and values for each attribute that has the following values and format:

"<hotel>
call parent's toString method and append the returned String here
<address>address</address>
<roomtype>roomtype</roomtype>
<checkin>checkin</checkin>
<checkout>checkout</checkout>
<roomnumber>roomnumber</roomnumber>
</hotel>"

public float calculatePrice()
return days (time between check-in and check-out) * 100

public HotelReservation clone()
Returns a copy of the current HotelReservation object

public HotelReservation clone(String line)
Given a line from the file, return a copy of the current HotelReservation by using the line constructor

RentalCarReservation Class

Information

Name: RentalCarReservation
Description/Purpose: This class creates object for reservation of a rental car
Modifiers: public
Inheritance: Inherits from the Reservation class

Attributes, Exceptions, and Methods

Attributes:
String make -The make of the rental car

String model – The model of the rental car
int year – The year of the rental car
Date scheduledPickUp – The scheduled pick-up date
Date scheduledDropOff – The scheduled drop-off date
Date actualPickUp – The date that the rental car was actually picked up
Date actualDropOff – The date that the rental car was actually dropped off
float window – How many windows the car has

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid

Methods:
Constructors: there are two ways to create a new RentalCarReservation object. If the caller selects to create a new RentalCarReservation, then constructor RentalCarReservation with parameters for attributes is called. If the caller wants to load existing reservation information from a String, then constructor RentalCarReservation (String line) is called.

public RentalCarReservation(String conNum, String phoneNum, String make, String mod, int year, Date sPickup, Date sDropOff, Date aPickup, Date aDropOff, float wind)
Call parent's constructor using super(conNum, conNum) - parent will validate common attributes
Validate parameters for unique rental car attributes and throw IllegalArgumentException if they are invalid
Load method parameters to object's attributes

public RentalCarReservation (String line)
Call parent's constructor to parse and load common attributes: super(line)
Check for substring <make> and assign value to tmpMake attribute
Check for substring <model> and assign value to tmpModel attribute
Check for substring <year> and assign value to tmpYear attribute
Check for substring <scheduledpickup> and assign value to tmpScheduledPickUp attribute
Check for substring < scheduleddropoff > and assign value to tmpScheduledDropOff attribute
Check for substring < actualpickup > and assign value to tmpActualPickUp attribute
Check for substring < actualdropoff > and assign value to tmpActualDropOff attribute
Check for substring < window > and assign value to tmpWindow attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public void updateRentalCarReservation(String mak, String mod, Date sPickUp, Date sDropOff, Date aPickUp, Date aDropOff, float wind)
Load method parameters to object attributes

public String getMake()
Accessor method for make attribute

public String getModel()
Accessor method for model attribute

public int getYear()
Accessor method for year attribute

public Date getScheduledPickUp()
Accessor method for scheduledPickUp attribute

public Date getScheduledDropOff()
Accessor method for scheduledDropOff attribute

public Date getActualPickup()
Accessor method for actualPickUp attribute

public Date getActualDropOff()
Accessor method for actualDropOff attribute

public float getWindows()
Accessor method for window attribute

public String toString()
return a string with tags and values for each attribute that has the following values and format:

"<rentalcar>
call parent's toString method and append the returned String here
<make>make</make>
<model>model</model>
<year>year</year>
<scheduledpickup>scheduledpickup</scheduledpickup>
<scheduleddropoff>scheduleddropoff</scheduleddropoff>
<actualpickup>actualpickup</actualpickup>
<actualdropoff>actualdropoff</actualdropoff>
<window>window</window>
</rentalcar>"

public float calculatePrice()
return days (time between pick-up and drop-off) * 23

public RentalCarReservation clone()
Returns a copy of the current HotelReservation object

public RentalCarReservation clone(String line)
Given a line from the file, return a copy of the current HotelReservation by using the line constructor

Person Class

Information

Name: Person
Description/Purpose: This class represents a person for reservation organization purposes.
Modifiers: public
Inheritance: None

Attributes, Exceptions, and Methods

Attributes:
String firstName – The first name of the person
String lastName = The last name of the person
Date dateOfBirth – The date of birth of the person

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid

Methods:
Constructors: there are two ways to create a new Person object. If the caller selects to create a new Person, then constructor Person with parameters for attributes is called. If the caller wants to load existing reservation information from a String, then constructor Person(String line) is called.

public Person(String firstName, String lastName, Date dateOfBirth)
Validate parameters for unique person attributes and throw IllegalArgumentException if they are invalid
Load method parameters to object's attributes

public Person(String line)
Check for substring <firstname> and assign value to tmpFirstName attribute
Check for substring <lastname> and assign value to tmpLastName attribute
Check for substring <dateofbirth> and assign value to tmpDateOfBirth attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public String getFirstName()
Accessor method for firstName attribute

public void setFirstName(String firstName)
Mutator method for firstName attribute

public String getLastName()
Accessor method for lastName attribute

public void setLastName(String lastName)
Mutator method for lastName attribute

public String getFullName()
Creates a string that combines the person's first and last name

public Date getDateOfBirth()
Accessor method for dateOfBirthattribute

public void setDateOfBirth(Date dateOfBirth)
Mutator method for dateOfBirthattribute

public String toString()
return a string with tags and values for each attribute that has the following values and format:

"<person>
	<firstname>firstname</firstname>
	<lastname>lastname</lastname>
	<dateofbirth>dateofbirth</dateofbirth>
</person >"

public Address clone()
Returns a copy of the current person object

public Address clone(String line)
Given a line from the file, return a copy of the current person by using the line constructor

Reservation Class

Information

Name: Reservation
Description/Purpose: This class creates object for reservation that will be subclassed
Modifiers: abstract
Inheritance: None

Attributes, Exceptions, and Methods

Attributes:
String confirmationNumber – The confirmation number of the reservation
String contractPhoneNumber – The phone number to answer questions about the servation
String reservationNumber – The number of the reservation

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid

Methods:
Constructors: there are two ways to create a new Reservation object. If the caller selects to create a new Reservation, then constructor Reservation with parameters for attributes is called. If the caller wants to load existing reservation information from a String, then constructor Reservation (String line) is called.

public Reservation(String conNum, String phoneNum)
Validate parameters for unique reservation attributes and throw IllegalArgumentException if they are invalid
Generate a random UUID for the reservationNumber
Load method parameters to object's attributes

public Reservation (String line)
Check for substring <confirmationnumber> and assign value to tmpConfirmationNumber attribute
Check for substring <contractphonenumber> and assign value to tmpContractPhoneNumber attribute

Check for substring <reservationnumber> and assign value to tmpReservationNumber attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public String getConfirmationNumber()
Accessor method for confirmationNumber attribute

public String getPhoneNumber()
Accessor method for contractPhoneNumberattribute

public void setPhoneNumber(String phoneNum)
Mutator method for contractPhoneNumber attribute

public String getReservationNumber()
Accessor method for reservationNumber attribute

public float calculatePrice()
Calculate the price of the reservation

public String toString()
return a string with tags and values for each attribute that has the following values and format:
"<confirmationnumber>confirmationnumber</confirmationnumber>
<contractphonenumber>contractphonenumber</ contractphonenumber>
<reservationnumber>reservationnumber</reservationnumber>"

public Reservation clone()
Returns a copy of the current address object

public Reservation clone(String line)
Given a line from the file, return a copy of the current address by using the line constructor

Address Class

Information

Name: Address
Description/Purpose: This class is used to represent an address for a hotel reservation
Modifiers: public
Inheritance: None

Attributes, Exceptions, and Methods

Attributes:
String street – The street of the address
String city – The city of the address
String state – The state of the address
String zip – The zip code of the address
String phoneNumber – The phone number to contact the address if needed

Exceptions Thrown:
IllegalArgumentException – Thrown when input is invalid

Methods:
Constructors: there are two ways to create a new Address object. If the user selects to create a new Address, then constructor Address(String street, String city, String state, String zip, String phoneNum) is called. If the caller wants to load existing address information from a String, then constructor Address(String line) is called.

public Address(String street, String city, String state, String zip, String phoneNum)
Validate parameters for unique address attributes and throw IllegalArgumentException if they are invalid
Load method parameters to object's attributes

public Address(String line)
Check for substring <street> and assign value to tmpStreet attribute
Check for substring <city> and assign value to tmpCity attribute
Check for substring <state> and assign value to tmpState attribute
Check for substring <zip> and assign value to tmpZip attribute
Check for substring <phonenumber> and assign value to tmpPhoneNum attribute
If parsing of any variable fails, then return an error and notify user accordingly
Otherwise, call the other constructor using the temporary variables
Catch any exceptions and print error message to console and cleanup

public String getStreet()
Accessor method for street attribute

public String getCity()
Accessor method for city attribute

public String getState()
Accessor method for state attribute

public String getZip()
Accessor method for zip attribute

public String getPhoneNumber()
Accessor method for phoneNumber attribute

public String toString()
return a string with tags and values for each attribute that has the following values and format:
"<address>
< street >street</street>
<city>city</city>
<state>state</ state>
<zip>zip</zip>
<phonenumber>phonenumber</phonenumber>
</address >"

public Address clone()
Returns a copy of the current address object

public Address clone(String line)
Given a line from the file, return a copy of the current address by using the line constructor

TripOrganizer Class

Information

Name: TripOrganizer
Description/Purpose: Creates a class that organizes a trip and its reservations
Modifiers: public
Inheritance: None

Attributes, Exceptions, and Methods

Attributes:
Trip trip – The trip that the organizer is organizing.

Exceptions Thrown:
None

Methods:
public TripOrganizer()
Insantiate a TripOrganizer object (no parameters needed because trips are loaded in or created later)

public Trip getTrip()
Accessor method for the trip attribute

public void addReservation(Reservation reservation)
Check to make sure there is a trip loaded in, otherwise the method fails and the user is notified
Call addReservation on the local trip attribute

public void saveToFile(String filename)
Check to make sure there is a trip loaded in, otherwise the method fails and the user is notified
Call saveToFile on the local trip attribute

public void openFromFile(String filename)
Set the local trip attribute to be a trip created using the filename constructor

public void editReservation(Reservation reservation)
Check to make sure there is a trip loaded in, otherwise the method fails and the user is notified
Call editReservation on the local trip attribute

public void deleteReservation(String confirmationNumber)
Check to make sure there is a trip loaded in, otherwise the method fails and the user is notified
Call deleteReservation on the local trip attribute

public void deleteTrip(String fileName)
Make sure the file exists
Make sure the local trip matches the one from the full

Delete the file
Set local trip to be null

public void createNewTrip(Trip trip)
Ensure the trip is valid, otherwise throw an IllegalArgumentException
Set the local trip to be equal to the trip passed as a parameter

IllegalLoadException Class

Information

Name: IllegalLoadException
Description/Purpose: Create an exception to be thrown when an illegal load occurs.
Modifiers: public
Inheritance: Inherits from the RuntimeException class

Attributes, Exceptions, and Methods

Attributes:
boolean isAccountFile – If true, the file is an account file, if false it is a reservation file
String filename – The name of the file that failed to load
String accountID – The account that was being loaded

Exceptions Thrown: None

Methods:
public IllegalLoadException(boolean isAccFile, String fileName, String accountID)
Call super constructor
Assign isAccFile object to attribute isAccFile
Assign fileName object to attribute fileName
Assign accountID object to attribute accountID

public String toString()
Creates a meaningful message that gives further detail as to what caused the exception by using member variables to fill in details.

IllegalOperationException Class

Information

Name: IllegalOperationException
Description/Purpose: Create an exception to be thrown when an illegal operation occurs.
Modifiers: public

Inheritance: Inherits from the RuntimeException class

Attributes, Exceptions, and Methods

Attributes:
String attemptedOperation – The operation that was being attempted
String accountID – The account in use when the operation failed
String reservationNumber – The reservation in use when the operation failed
String errorDetails – Further details about the illegal operation

Exceptions Thrown: None

Methods:
public IllegalOperationException(String attemptOp, String accountID, String resNum, String errorDetails)
Call super constructor
Assign attemptOp object to attribute attemptOp
Assign accountID object to attribute accountID
Assign resNum object to attribute resNum
Assign errorDetails object to attribute errorDetails

public String toString()
Creates a meaningful message that gives further detail as to what caused the exception by using member variables to fill in details.

DuplicateObjectException Class

Information

Name: DuplicateObjectException
Description/Purpose: Create an exception to be thrown when an illegal load occurs.
Modifiers: public
Inheritance: Inherits from the RuntimeException class

Attributes, Exceptions, and Methods

Attributes:
String accountID – The account that caused the exception to occur
String reservationNumber – The reservation number that caused the exception to occur
String failureReason – Detailed message about why the error occurred

Exceptions Thrown: None

Methods:
public DuplicateObjectException(String accountID, String resNum, String failReason)
Call super constructor
Assign accountID parameter to attribute accountID
    If blank, then the duplicate object is the reservation and not the account.
Assign reservationNumber parameter to attribute reservationNumber
    If blank, then the duplicate object is the account and not the reservation.
Assign failureReason parameter to attribute failureReason

public String toString()
Creates a meaningful message that gives further detail as to what caused the exception by using member variables to fill in details.
If accountID is blank then the details will not include the account.
If reservationNumber is blank then the details will not include the reservation.

SameAirportException Class

Information

Name: SameAirportException
Description/Purpose: Create an exception to be thrown when the same airport is used for departure and destination.
Modifiers: public
Inheritance: Inherits from the RuntimeException class

Attributes, Exceptions, and Methods

Attributes:
String airport – The airport that resulted in the exception being thrown

Exceptions Thrown: None

Methods:
public SameAirportException(String airport)
Call super constructor
Assign airport object to attribute airport

public String toString()
Creates a meaningful message that gives further detail as to what caused the exception by using member variables to fill in details.

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user is first met with a screen to load a trip from file or create a new trip. (Image #1)

If loading from file, a JFileChooser dialog pops up to select the path.

If they are creating a new trip from scratch, they must fill out some info (Image #2).

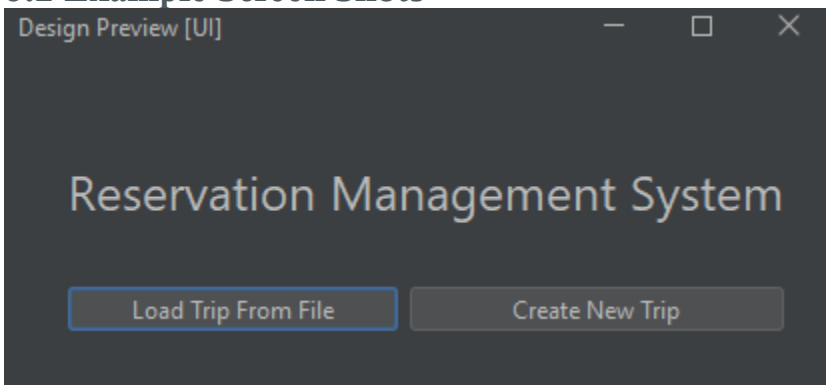
If they are loading a new trip or they create a new trip and fill out all the info (including at least one reservation) they are taken to the trip view (Image #3).

There they have the option to edit, delete, view, or add a reservation. Confirmation is required for deletion. The view button shows all information about the specific reservation

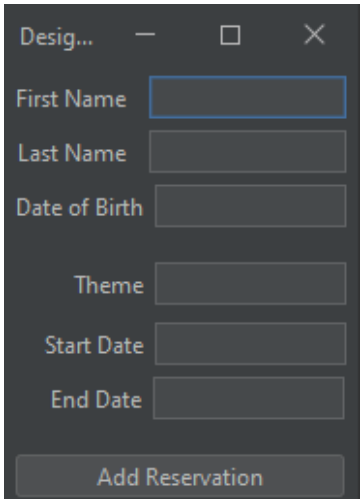
If they click edit or view they are taken to the reservation creation screen (Image #4).

Any errors or exceptions will be shown in a JOptionPane along with the detailed information from the exception.

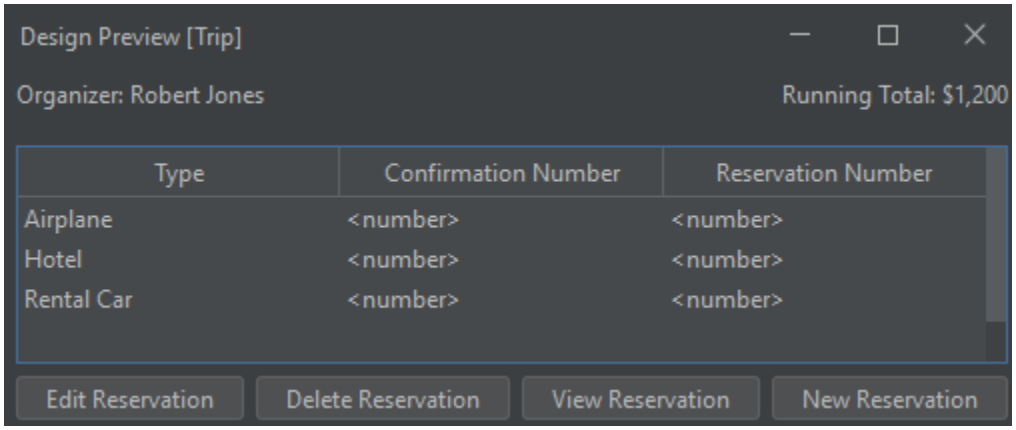
6.2 Example Screen Shots



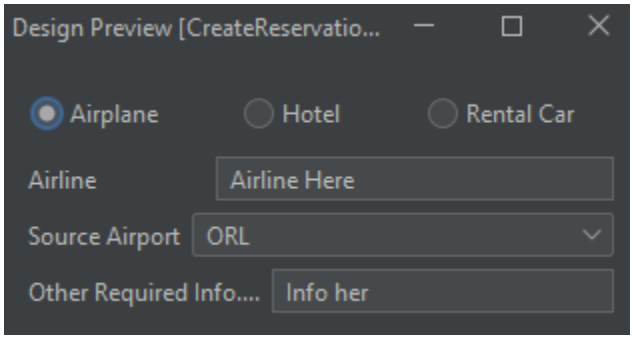
UI Image #1: Main Screen



UI Image #2: Dialog that appears when "Create New Trip" is pressed



UI Image #3: The view screen of the trip



UI Image #4: Screen for new or edit reservation. If edit reservation, info is pre-populated

7. REQUIREMENTS MATRIX

Requirement	Class	Constructor/Method	Attribute
Trip organizer	TripOrganizer	TripOrganizer()	
Trip should record	Trip		Person

that name of the person that is organizing the trip			organizer
Trip can have multiple reservations	Trip		Vector <Reservation>
Trip can have airplane reservation	AirplaneReservation		
Every trip has theme	Trip		String theme
Add reservation to trip	TripOrganizer/ Trip	addReservation(Reservation reservation)	
We only work with one trip at a time	TripOrganizer	openFromFile() and createNewTrip() both set the trip local variable	Trip trip
We need the ability to save a trip to a file and reopen it.	TripOrganizer	saveToFile() and openFromFile()	
Trip can have a hotel reservation	HotelReservation		
An address contains a street, city, state, ZIP, country and phone number.	Address	All of these attributes are populated in constructor	street, city, state, zip, phoneNumber
Trip can have a rental car reservation	RentalCarReservation		
Rental confirmation cannot be changed once a new reservation is created.	Rseservation	No setter for confirmationNumber, only a getter	confirmationNumber
Each type of reservation will have a different price	AirplaneReservation, HotelReservation, RentalCarReservation	calculatePrice()	Airplane price is miles * $1 Hotel price is nights * $100 Car is days * $23
The software should support four airports: IAD, ORL, BWI, and NYC	AirplaneReservation	airportDistance()	
The software will validate that the source Airport and destination Airport are different and throw a	SameAirportException	SameAirportException()	

new exception SameAirportExceptio n with an appropriate message otherwise.			
The software should allow you to create, save and open a trip.	TripOrganizer/ Trip	createNewTrip(), saveToFile() and openFromFile()	Trip trip
For each trip the software should allow the user to manage the hotels, airplanes trips and rental cars.	Trip/All Reservation Subclasses	editReservation() updateHotelReservatio n() updateAirplaneReserv ation() updateRentalCarReser vation()	Vector <Reservation>
The software should also keep a running total of the price of each part of the trip and the grand total price for the trip.	TripOrganizer/ Trip	For each reservation in the trip, the total is calculated by calling calculatePrice() on each reservation	Vector <Reservation>