OOP Project Description

Requirements:

You are to design and build prototype software (stand-alone application loaded on business's local computer) **to manage accounts and associated lodging reservations**. Your software will be used and called by User Interface (UI) that someone else will be implementing. You will get to design the UI in software design document (SDD) but you will not be implementing it for this project.

Your management's software will load all the existing accounts and their associated lodging reservations on startup. For this project, you can hardcode the full path for the directory where the data will be loaded from and saved to using a constant value in Manager. Each account's data should be saved in a separate directory using the account's id number. Account's information should be saved in one file and then each reservation that belongs to that account in separate file.

Account files name convention should be "acc-" followed by the account id number and the appropriate file extension. The reservation files name convention should be "res-" followed by the reservation id and appropriate file extension. On loading if a file has a different name than this convention, it should be ignored.

For example: Given hard-coded path "*C:/tmp-umuc/accounts/*" we may have a subdirectory called "*12345*" and in that subdirectory a file with accounts data: *acc-12345.txt* and two lodging reservations: *res-HAB123.txt* and *res-ABCD124.txt*

So for example, the full path for reservation res-HAB123.txt file would be: "*C:/tmp-umuc/accounts/12345/res-HAB123.txt*"

For the final product outside the scope of this project, server with database will be used for each account and corresponding reservations instead of the text files. UI will not know the directory (and will not be able to obtain the directory) nor will it interact directly with the data storage (only through your system's methods).

When the system starts up, UI will create an instance of your Manager class. Manager class will then load all the account and their corresponding reservations data.

An account includes the following information: unique id number generated by UI (9 digits), mailing address associated with the account, list of lodging reservations, and an email associated with the account.

When a new account is created, it must be provided with an account id, mailing address, and email. New account has no lodging reservations. Account's id cannot be changed. Account's mailing address and email can be changed at any time. A new account can be added to the manager but it cannot be removed. No duplicate accounts, based on id number, are allowed.

There are three types of lodging reservations: hotel room reservation, apartment reservation, and house reservation. Each lodging reservation includes an account id that the reservation belongs to, reservation number generated by UI (10 characters), lodging physical address, mailing address, when reservation starts (Date or Calendar), number of nights, number of beds, lodging size in square feet (integer), price of the reservation, and reservation status (draft, completed, or cancelled).

You can make the reservation number value have a prefix which is unique per the type of reservation. For example, hotel starts with H, apartment with A, and house with O. This way your reservation filename, since it includes the reservation id, also tells you what type of reservation it contains.

In addition, hotel reservation has an indication of whether there is kitchenette or not; apartment reservation has the number of bathrooms; and house reservation has the number of floors and number of bathrooms.

All values must exist on reservation creation except for mailing address. If the mailing address does not have a value, UI will know to use the physical address as mailing address. In addition:

- Account id is updated when reservation is added to a specific account, otherwise it has a negative one value
- New reservation always starts in draft status
- Price of the reservation is not set until the reservation is completed

Lodging price per night is calculated as follows.
(1) Basic price is $100 with an additional fee of $20 if the lodging size is greater than 800 square feet. This price assumes a single bedroom and bathroom
(2) Apartment has an additional fee of $10 per each additional bedroom and $5 for each additional bathroom
(3) House has an additional fee of $20 per bedroom and $5 per each additional floor

Note that tax is already included in the price.

The account and reservation files but must be human readable text files and you must follow **separation of concerns principle**. You must use XML tags (you can model after Trip example) or JSON. There are code examples of both in Course Content->Code Examples.

For this project I require that each class has toString() method (explicit or inherited from parent) that can be used to output the class's data to the screen. Also the output of toString() is the formatted data that will be written to the file and then can be loaded by that class's constructor.

After the account and reservations data is loaded on system startup (on creation of Manager instance), the user (using UI) will be able to:

1. Get the list of loaded accounts
2. Retrieve the loaded account object that matches a specific account id
3. Create a new account object and add to the list managed by Manager (if account object already exists on add action with the same account id, it is considered an error)
4. Request that Manager updates specific account's file with data stored in memory
5. Add draft lodging reservation to an account (if reservation object already exists with the same reservation number, it is considered an error)
6. Complete reservation (if reservation is cancelled, already completed, or for past date, it is considered an error)
7. Cancel reservation (if reservation is already cancelled or for past date, it is considered an error)
8. Edit reservation values that can be changed (if reservation is cancelled, completed, or for past date, it is considered an error)
9. Request for price per night to be calculated and returned for a specific reservation
10. Request for total reservation price to be calculated and returned for a specific reservation

Note that UI will handle the login and authorization and access control but all are outside of the scope of the system you are designing.

Your system will need the following exceptions and validation to be added **for the week 5 assignment**:
1. When loading accounts and/or reservations from file(s) if there is any issue parsing account or reservation files throw unchecked user defined exception called **IllegalLoadException** when violation detected. The exception will extend RuntimeException. The exception message should indicate what failed (account file versus reservation file) and the filename that could not be loaded. The message should include the account's id that was being loaded.
2. On adding account or adding reservation to an account, if duplicate account/reservation exists throw unchecked user defined exception called **DuplicateObjectException** when violation detected. The exception will extend RuntimeException. The generated exception message should indicate the id of the account and/or reservation number and why it failed.
3. On cancelling or completing reservation if it cannot be finalized, throw unchecked user defined exception called **IllegalOperationException** when violation detected. The exception will extend RuntimeException. The generated exception message should indicate the operation that was attempted, account id, reservation number, and details why exactly it failed.
4. On all other issues use Java builtin exceptions such as IllegalArgumentException for invalid parameters and IllegalStateException when state of the object cannot be changed. Code that calls these exceptions must pass a meaningful message.

Note that for this project you must pass the values that are needed for the meaningful exception message to the exception's constructor(s) and have the exception class generate a meaningful message. Each exception's constructor should handle a specific failing scenario.

UI is outside the scope of your design but your system needs to have all the functionality that UI needs to satisfy the project requirements.

As you design the system, you need to make sure it is consistent and it only has the interfaces (constructors, attributes and methods) that are needed by the required functionality. **You cannot arbitrarily add new functionality to the system.**

The system you are designing will have the functionality needed to satisfy the above requirements. You will NOT be implementing the actual user interface (you will still design it though in your Software Design Document in later assignment) but you need the classes, attributes, and methods to receive and provide the information that the user interface will need. You can assume that for a complete solution there would be some other class that implements user interface (UI) with main() method and that it will call your classes/methods. **Your system will not have main method except in the test class.**

When you implement the system, you must have at minimum test cases to verify the major functionality of the system as listed above and a test case for at least one error path per user defined exception. Test class is not part of the class diagram.

There should be a **single test class with main and each test case scenario should be a separate method** called by main method. I should be able to run all methods at once or select a single method to run by commenting out calls to the other methods. Make sure you use comments to document what each test case is verifying and doing. If you hardcode your data files instead of having test cases generate them, make sure to include those files with your code submissions.

Each test case should output a brief description of what it is testing and then some output. For example, a test case testing that exception is generated would output a message that exception was caught and what the exception message was; A test case to test that data was successfully loaded from file would output what was loaded using toString() method.