CSCI 3060U Course Project

Winter 2022

Phase #2 - Front End Rapid Prototyping

March 6th, 2022

Design Document

Group members:

Ben Tsoumagas 100751395

Thinh Le 100741899

Muaz Rehman 100553376

Cody Malcolm 100753739

Structure of solution:

Overview: The application objective is to manage the front end user interface for the OT-BnB short term rental system. The application should allow users to perform search, post, and rent transactions to manage rentals, as well as create and delete transactions to manage user accounts. Users must be logged in in order to perform any of these transactions, and must be logged out in order to log in as a new user or quit the application.

Solution: The application is run from the 'Application' class. Upon startup, the application initializes main memory with pre-existing data from the Current User Accounts file and the Available Rentals file. Next, the application welcomes the user and prompts for commands. The application will repeatedly prompt for commands and handle them appropriately until the user enters a 'quit' command while logged out, at which point the application will write the application session terminator code to the Daily Transaction file and shut down.

When a command is entered, it is confirmed to be a valid command and then is handled appropriately based on the two possible states of the application - user is logged in, or user is not logged in. Upon startup, no user is logged in and there is no user session. When a user logs in, the application creates and maintains a 'Session' object which contains information about the logged in user, the transactions that have been performed this user session, and the units that have been listed for rent this user session. When a user logs out, the application updates main memory with the newly listed units, prints the transactions to the Daily Transaction file, and resets the user session pointer to null.

When a user is logged in and a transaction is requested (create, delete, search, post, or rent), the main loop will call an appropriate handler for that transaction that will perform all necessary validations that the transaction is acceptable (ie. user has correct privileges) and that the input is valid (eg. user does not enter a string when a number is expected). These handlers will return a 'Transaction' object (or null if the Transaction was not completed) and the main loop will add these to the current session transactions.

Many of these handlers require receiving and validating user input that may be incorrect and require re-prompting an arbitrary number of times in a very similar fashion. In the interest of dramatically reducing code redundancy a 'Utils' class was created to provide some standard functionality for the application. The most important of these functions is the 'getInput' function which is heavily used throughout the application. This function takes a LinkedHashMap of conditions and a String prompt. It will prompt the user for input with the given prompt, then iterate through the conditions that are in the format "regular expression : error message" where the regular expression validates the input in some way (for example, "is this an integer?") and prints the corresponding error message if the validation fails. Once the user enters an input that passes all conditions, the input is returned. A LinkedHashMap was used because it guarantees the order of iteration. Therefore the correct error message will be shown in cases where the input would fail multiple conditions. To make use of this function, we just need to create a LinkedHashMap and put the appropriate key value pairs, then invoke the getInput method.

Intention table:

Class Name	Method Name	Intention
Application		Main class for OT-BnB, which stores users, units, and the current user session (if any).
	Application()	Creates an instance of the Application class, initializes session to null.
	main(String[])	Creates a new application instance, initializes it, and runs it.
	run()	Continually prompts users for recognized commands and adds to transaction file contents as appropriate.
	initialize()	Starts application with the contents of the user accounts and available rentals files.
	populateUnits(String)	Populates rental units in memory from the available rentals file.
	populateUsers(String)	Populates users in memory from the user accounts file.
	writeTransactions(ArrayList<Transaction>)	Writes transactions to the daily transaction file.
	getCommand() -> String	Prompts user for the next command until a valid command is given, and returns the valid command.
	denyAccess(String)	Tells users if they lack permissions and which permission is lacking.

	handleRent() -> Transaction	Handles a user's rent transaction and returns a corresponding Transaction object, or null if the rental was canceled during confirmation.
	handleCreate() -> Transaction	Handles a user's create transaction and returns a corresponding Transaction object.
	handleDelete() -> Transaction	Handles a user's delete transaction and returns a corresponding Transaction object (or null if the input username is invalid).
	handleSearch() -> Transaction	Handles a user's search transaction and returns a corresponding Transaction object.
	handlePost() -> Transaction	Handles a user's post transaction and returns a corresponding Transaction object.
	handleLogin() -> Session	Handles user login and returns a new Session for this user.
Listing		Provides functions related to listing a new unit.
	Listing(String)	Creates an instance of the Listing class, sets username to the given username.
	list() -> Unit	Prompts users for the information needed to create a unit and returns the new Unit.
	queryBedrooms()	Gets the number of bedrooms for the listing from the user.
	queryPrice()	Gets the price for the listing from the user.
	queryCity()	Gets the city for the listing from the user.

RegexManager		Utility class that contains regex expressions the program uses.
Rental		Provides functions related to renting an available unit.
	book(ArrayList<Unit>, ArrayList<Unit>, User) -> Transaction	Manages the flow of the rental prompt and returns the transaction.
	queryNumNights() -> int	Prompts users for the number of nights to rent for and returns it.
	queryUnit(ArrayList<Unit>, ArrayList<Unit>, String) -> Unit	Prompts user for the unit ID and returns it.
	printDetails() -> String	Prints per night and total cost of the rental; return the daily price.
	confirmBooking() -> boolean	Prompts user confirmation of the rental and returns it.
Search		Provides functions related to searching available units.
	search(ArrayList<Unit>, User) -> Transaction	Manages the user's search transaction and returns it.
	displayResult(List<Unit>)	Displays search results.
	queryCity() -> String	Gets the city to search for from the user and returns it.
	queryMaxPrice() -> int	Gets the maximum price to search with from the user and returns it.

	queryMinBedrooms() -> int	Gets the minimum bedrooms to search with from the user and returns it.
Session		Maintains list of transactions performed during the session.
	Session(User)	Creates an instance of the Session class, sets user to current user.
	addTransaction(Transaction)	Adds the transaction to the transaction list.
	addUnit(Unit)	Adds the unit to the unit list.
	getUser() -> User	Gets the current user and returns it.
	getTransactions() -> ArrayList<Transaction>	Gets the list of transactions and returns them.
	getUnits() -> ArrayList<Unit>	Gets the current units and returns them.
Transaction		Formats and holds all information to be printed.
	Transaction(String)	Creates an instance of the Transaction class, sets code to current code. Used by quit transaction.
	Transaction(String, String, String)	Creates an instance of the Transaction class, sets code, username, user type to current values. Used by create, delete transactions.
	Transaction(String, String, String, String, int, String)	Creates an instance of the Transaction class, sets code, username, user type, city, bedrooms, and price to current values. Used by search transactions.

	Transaction(String, String, String, String, String, int, String)	Creates an instance of the Transaction class, sets code, username, user type, city, bedrooms, price, and id to current values. Used by post transactions.
	Transaction(String, String, String, String, String, int, String, int)	Creates an instance of the Transaction class, sets code, username, user type, city, bedrooms, price, id, and numNights to current values. Used by rent transactions
	getOutput() -> String	Gets the string representation of the transaction and returns it.
Unit		Represents all posted rental units
	Unit(String, String, int, int, String, String)	Creates an instance of the Unit class, sets host, city, price, bedrooms, rentalFlag, rentalID to current values.
	Unit(String, String, int, int)	Creates an instance of the Unit class, sets host, city, price, bedrooms to current values.
	getUnit(ArrayList<Unit>, String) -> Unit	Returns the unit with the given unit ID.
	anyAvailableUnits(ArrayList<Unit>, String) -> boolean	Returns true if the current user can rent any of the provided units, otherwise false.
	meetsCriteria(String, int, int) -> boolean	Returns true if the unit satisfies all conditions and can be rented (ie. should appear in search results), otherwise false.
	rent()	Sets the rented flag to true for a unit.
	getHost() -> String	Returns the unit's host.
	getCity() -> String	Returns the unit's city.

	getPrice() -> int	Returns the unit’s daily price.
	getBedrooms() -> int	Returns the unit’s number of bedrooms.
	getID() -> String	Returns the unit ID.
	getRented() -> boolean	Returns the unit’s rented status.
User		Holds user information and user functionality.
	User(String, String)	Creates an instance of the User class, sets username and user type to current values.
	getNewUsername(ArrayList<User>) -> String	Prompts user for username, guards against duplicates and invalid entries, and then returns the validated username.
	getNewUserType() -> String	Prompts user for user type until it is valid and returns it.
	getUser(ArrayList<User>, String) -> User	Returns the requested user or null if they do not exist.
	login(ArrayList<User>) -> User	Prompts user for usernames, until an existing username is given, then returns the associated user.
	deleteUser(ArrayList<Unit>)	Removes all units belonging to the user and updates the user’s deleted flag.
	isAdmin() -> boolean	Returns true if usertype is admin, otherwise false.
	getUsername() -> String	Returns username.
	getUsertype() -> String	Returns user type.

	getDeleted() -> boolean	Returns the deleted flag.
Utils		Miscellaneous utilities for other classes to use.
	getInput(LinkedHashMap<String, String>, String) -> String	Prompts users for input that satisfies conditions.
	getInput(LinkedHashMap<String, String>, String, boolean) -> String	Prompts users for input that satisfies conditions, allowing wildcards.
	pad(String, int) -> String	Pads string with spaces to the right to the desired length. If the string is longer than given length, crops the string instead.
	padZeroes(String, int) -> String	Pads string with zeros on the left side to the desired length.
	priceToInt(float) -> int	Converts a price from dollars to cents.
	parsePrice(int) -> String	Converts a price from cents to dollars.
	readFile(String) -> ArrayList<ArrayList<String>>	Reads a file to return it as a nested list.
	discard(String, String, int)	Prints the reason for an ignored file input.