Class: State	An abstract class that represents a program state. Unifies data and methods used by all states, but requires state execution to be implemented
User execute(User, DataAccess, Scanner)	Abstract method containing unique state logic that must be implemented by child States
void showBanner()	Prints the application banner

Class: Command	Enumeration that represents valid user commands
Command(String)	Constructor: creates a Command with the appropriate string representation
String toString()	Returns the Command's string representation
boolean validateUser(User)	Checks the permissions map to verify whether passed user is authorized to use the current Command
List<Command> getUserPermissions(User)	Builds and returns a list of Commands that the passed user is authorized to use

Class: MainMenu extends State	Extends the State class to present a main menu that provides the user with access to various program functions. Also contains the program entry point.
User execute(User, DataAccess, Scanner)	Implements the execute method from State to present a main menu, then launch other States based on console input.
void main(String[])	Program entry point: initializes the input device and DataAccess object, then launches the main menu

Class: SearchListing extends State	Extends the State class to allow for users that are active to Search Listings and validate user input.
User execute(User, DataAccess, Scanner)	Implements the execute method from State to present the Search listings prompts

Class: PostListing extends State	Extends the State class to allow for users under specific UserTypes to post their rental properties
User execute(User, DataAccess, Scanner)	Implements the execute method from State to post a listing

Class: RentListing extends State	Extends the State class to allow for users under a specific UserTypes to rent existing non-rented listings.
User execute(User, DataAccess, Scanner)	Implements the execute method from State to rent a valid listing

Class: Login extends State	Extends the State class to change the active User based on console input
User execute(User, DataAccess, Scanner)	Implements the execute method from State to set an active User based on console input

Class: Logout extends State	Extends the State class to log out the current user, as well as trigger the write the daily transaction file
User execute(User, DataAccess, Scanner)	Implements the execute method from State to remove the current active user, as well as trigger the write to the daily transaction file

Class: DeleteUser extends State	Extends the State class to allow for the deletion of users along with their posts
User execute(User, DataAccess, Scanner)	Implements the execute method from State to delete a user and their posts

Class: CreateUser extends State	Extends the State class to allow for the creation of new users
User execute(User, DataAccess, Scanner)	Implements the execute method from State to create a user

Class: User	Extends the State class for use with checking validity and setting usernames and UserTypes

String getUsername()	Returns the associated username
UserType getUserType()	Returns the Log's associated usertype
User(String, UserType)	Sets the username and usertype
Boolean isValidUsername(String)	Validates username string

Class: UserType	Enumeration that represents valid User Types
UserType.UserType(String)	Returns UserType based on the string argument
String toString()	Returns a string based on the usertype
UserType fromString(String)	Takes short-hand user type string and returns UserType

Class: Listing	Abstracts Listings and handles all listings functions.
Listing(DataAccess, User, String, double, int)	New Listing Constructor. Takes in required input from post listings and generates the rest
Listing(String, User, String, double, int, bool, int)	Default Listing Constructor. Takes in all input from an already set listing.
getRentalUnitID()	Returns _rentalUnitID variable of listing object
getOwner()	Returns _owner variable of listing object
getCity()	Returns _city variable of listing object
getRentalPrice()	Returns _rentalPrice variable of listing object
getNumberOfRooms()	Returns _numberOfRooms variable of listing object
isRented()	Returns _rentedFlag variable of listing object
getNightsRented()	Returns _nightsRented variable of listing object
setRentedFlag(bool)	sets rented flag to boolean value passed
isValidRentalID(String)	Checks if unitRentalID is 8 characters and alphanumeric using regex. and returns false if statement is true. Or true if the statement is false.

isValidCity(String)	Checks if the city is not between 1 to 25 characters and if it's not alphabetic or contains - and returns false if the statement is true. Or true if the statement is false.
isValidRentalPrice(double)	Checks if the price is less than 1 or greater than 999.99 and returns false if the statement is true. Or true if the statement is false.
isValidNumberofRooms(int)	Checks if rooms are less than 1 or greater than 9 and returns false if the statement is true. Or true if the statement is false.
isValidNightsRented(int)	Checks if nights rented is less than 1 or greater than 14 and returns false if the statement is true. Or true if the statement is false.
generateUnitID()	Creates a randomly Generated uppercase alphanumeric ID of 8 characters
toStringSearch()	Displays Object variables in an user friendly format. Mainly for SearchListings Class.
toString()	Displays Object variables in an user friendly format. Mainly for debugging purposes.

Class: Log	Represents a log to be written to the daily transaction file.
Log(TransactionCode, User)	Constructor: creates a Log without an associated Listing
Log(TransactionCode, User, Listing)	Constructor: creates a Log from passed Transaction Code, User, and Listing
TransactionCode getTransactionCode()	Returns the Log's Transaction Code
String getUsername()	Returns the Log's associated username
User.UserType getUserType()	Returns the Log's associated User Type
String getRentalUnitID()	Returns the Log's associated rental unit ID
String getCity()	Returns the Log's associated city
int getNumberOfRooms()	Returns the Log's associated number of rooms

double getRentalPrice()	Returns the Log's associated rental price
int getNightsRented()	Returns the Log's associated number of nights

Class: TransactionCode	Enumeration that represents a valid Transaction Code
TransactionCode(String)	Constructor: creates the enumeration with an appropriate string representation
String toString()	Returns the Transaction Code's string representation

Class: DataAccess	Handles the program's data by abstracting input and output functionality, providing an interface to access and modify data as needed
DataAccess() {	Default constructor for DataAccess that uses default input and output file locations
DataAccess(String, String, String)	Parameterized constructor for DataAccess that takes custom input and output file locations
User getUser (String)	Searches for a cached User by username. Returns User if found, null otherwise.
Listing getListing (String)	Searches for a cached Listing by rental unit ID. Returns Listing if found, null otherwise.
Listing[] getListings (User)	Searches for all cached Listings by User who owns them. Returns a list of Listings associated with that User.
Listing[] searchListings (String, Double, Integer)	Searches the cached Listings for Listings that match the passed city, rental price, and number of rooms, and returns them in a Listing[] array. Parameters passed as null are equivalent to "any"
void addUser (User)	Adds a User to the cached Users list
void addListing (Listing)	Adds a Listing to the cached Listings list
void addLog (Log)	Adds a Log to the current session's Log list

void removeUser(User)	Removes passed User object from the cached Users list
void removeListings(Listing[])	Removes all Listings in the passed Listing array from the cached Listings
void commitNewListings ()	Adds this session's newly created Listings to the cached Listing list
boolean listingExists(String)	Returns true if a Listing exists in the cached Listings list with the passed rental unit ID, otherwise returns false
boolean userExists(String)	Returns true if a User exists in the cached Users list with the passed username, otherwise returns false
boolean areUsersLoaded()	Returns true if the cached Users list has been previously loaded during the process lifetime, otherwise returns false
boolean areListingsLoaded()	Returns true if the cached Listings list has been previously loaded during the process lifetime, otherwise returns false
void loadUsers()	Reads the users.txt file and creates Users from the contents, loading them into the cached Users list.
void loadListings()	Reads the listings.txt file and creates Listings from the contents, loading them into the cached Listings list
void writeDailyTransactionFile()	Appends the session’s Logs to the daily transaction file, then clears the Log list. If a transaction file does not exist for today, a new file is created with the local date as its name