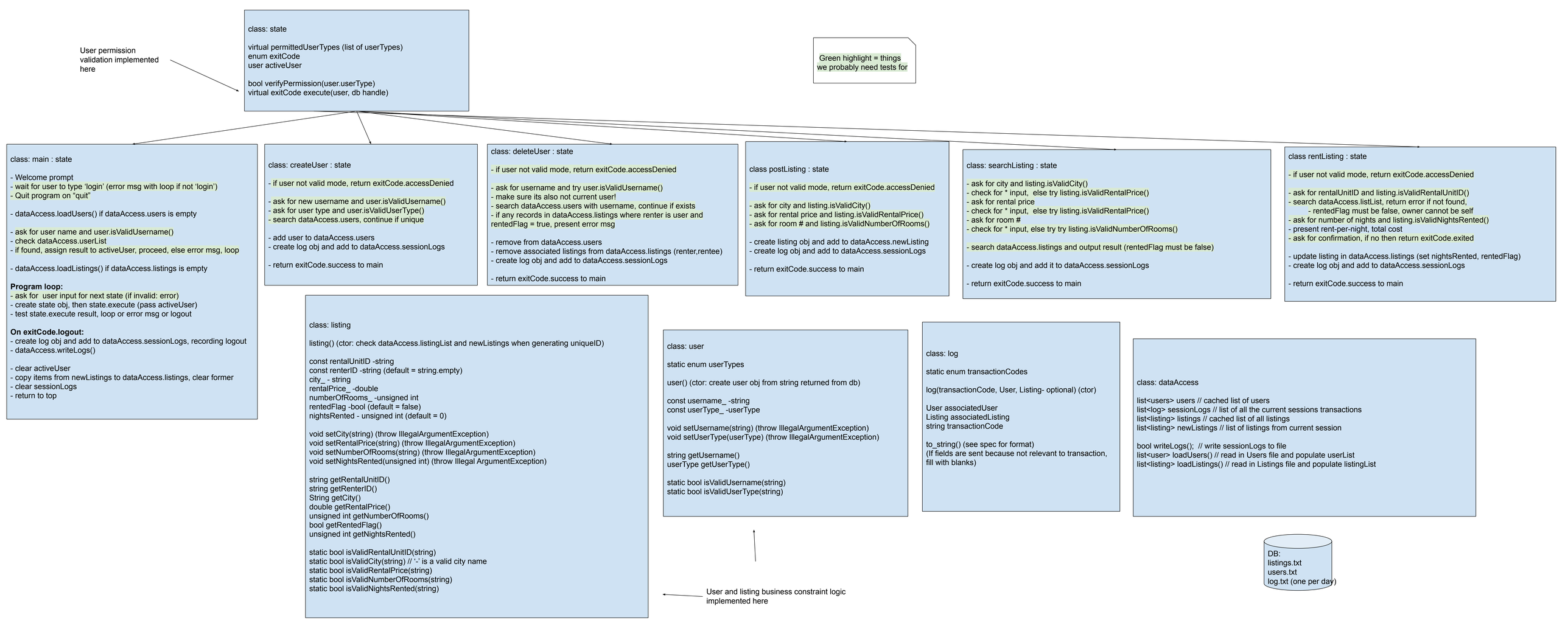
User permission validation implemented here
class: state
virtual permittedUserTypes (list of userTypes)
enum exitCode
user activeUser
bool verifyPermission(user.userType)
virtual exitCode execute(user, db handle)
Green highlight = things we probably need tests for
class: main : state
- Welcome prompt
- wait for user to type 'login' (error msg with loop if not 'login')
- Quit program on "quit"
- dataAccess.loadUsers() if dataAccess.users is empty
- ask for user name and user.isValidUsername()
- check dataAccess.userList
- if found, assign result to activeUser, proceed, else error msg, loop
- dataAccess.loadListings() if dataAccess.listings is empty
Program loop:
- ask for user input for next state (if invalid: error)
- create state obj, then state.execute (pass activeUser)
- test state.execute result, loop or error msg or logout
On exitCode.logout:
- create log obj and add to dataAccess.sessionLogs, recording logout
- dataAccess.writeLogs()
- clear activeUser
- copy items from newListings to dataAccess.listings, clear former
- clear sessionLogs
- return to top
class: createUser : state
- if user not valid mode, return exitCode.accessDenied
- ask for new username and user.isValidUsername()
- ask for user type and user.isValidUserType()
- search dataAccess.users, continue if unique
- add user to dataAccess.users
- create log obj and add to dataAccess.sessionLogs
- return exitCode.success to main
class: deleteUser : state
- if user not valid mode, return exitCode.accessDenied
- ask for username and try user.isValidUsername()
- make sure its also not current user!
- search dataAccess.users with username, continue if exists
- if any records in dataAccess.listings where renter is user and rentedFlag = true, present error msg
- remove from dataAccess.users
- remove associated listings from dataAccess.listings (renter,rentee)
- create log obj and add to dataAccess.sessionLogs
- return exitCode.success to main
class postListing : state
- if user not valid mode, return exitCode.accessDenied
- ask for city and listing.isValidCity()
- ask for rental price and listing.isValidRentalPrice()
- ask for room # and listing.isValidNumberOfRooms()
- create listing obj and add to dataAccess.newListing
- create log obj and add to dataAccess.sessionLogs
- return exitCode.success to main
class: searchListing : state
- ask for city and listing.isValidCity()
- check for * input, else try listing.isValidRentalPrice()
- ask for rental price
- check for * input, else try listing.isValidRentalPrice()
- ask for room #
- check for * input, else try try listing.isValidNumberOfRooms()
- search dataAccess.listings and output result (rentedFlag must be false)
- create log obj and add it to dataAccess.sessionLogs
- return exitCode.success to main
class rentListing : state
- if user not valid mode, return exitCode.accessDenied
- ask for rentalUnitID and listing.isValidRentalUnitID()
- search dataAccess.listList, return error if not found,
- rentedFlag must be false, owner cannot be self
- ask for number of nights and listing.isValidNightsRented()
- present rent-per-night, total cost
- ask for confirmation, if no then return exitCode.exited
- update listing in dataAccess.listings (set nightsRented, rentedFlag)
- create log obj and add to dataAccess.sessionLogs
- return exitCode.success to main
class: listing
listing() (ctor: check dataAccess.listingList and newListings when generating uniqueID)
const rentalUnitID -string
const renterID -string (default = string.empty)
city_ - string
rentalPrice_ -double
numberOfRooms_ -unsigned int
rentedFlag -bool (default = false)
nightsRented - unsigned int (default = 0)
void setCity(string) (throw IllegalArgumentException)
void setRentalPrice(string) (throw IllegalArgumentException)
void setNumberOfRooms(string) (throw IllegalArgumentException)
void setNightsRented(unsigned int) (throw Illegal ArgumentException)
string getRentalUnitID()
string getRenterID()
String getCity()
double getRentalPrice()
unsigned int getNumberOfRooms()
bool getRentedFlag()
unsigned int getNightsRented()
static bool isValidRentalUnitID(string)
static bool isValidCity(string) // '-' is a valid city name
static bool isValidRentalPrice(string)
static bool isValidNumberOfRooms(string)
static bool isValidNightsRented(string)
class: user
static enum userTypes
user() (ctor: create user obj from string returned from db)
const username_ -string
const userType_ -userType
void setUsername(string) (throw IllegalArgumentException)
void setUserType(userType) (throw IllegalArgumentException)
string getUsername()
userType getUserType()
static bool isValidUsername(string)
static bool isValidUserType(string)
User and listing business constraint logic implemented here
class: log
static enum transactionCodes
log(transactionCode, User, Listing- optional) (ctor)
User associatedUser
Listing associatedListing
string transactionCode
to_string() (see spec for format)
(If fields are sent because not relevant to transaction, fill with blanks)
class: dataAccess
list<users> users // cached list of users
list<log> sessionLogs // list of all the current sessions transactions
list<listing> listings // cached list of all listings
list<listing> newListings // list of listings from current session
bool writeLogs(); // write sessionLogs to file
list<user> loadUsers() // read in Users file and populate userList
list<listing> loadListings() // read in Listings file and populate listingList
DB:
listings.txt
users.txt
log.txt (one per day)