Off-Campus Housing Management Design Document

10.03.2020

Version 1.0

Team Alpaca
Caroline Boozer
Christian Carter
Dalton Craven
Matt Duggan

Table of Contents

Introduction **3**
Purpose 3
System Overview 3
References 3
Definitions, Acronyms, and Abbreviations 3

Design Overview **3**
Introduction 3
Environment Overview 4
Constraints and Assumptions 4
Schedule Constraints 4
Monetary Constraints 4
Technical Constraints 4

Interfaces and Data Stores **4**
System Interfaces 4
Scenario 1 4
Scenario 2 5
Scenario 3 6
Data Stores 7
Example JSON Files 7
Listing JSON 7
Manager JSON 8
Property JSON 8
Tenant JSON 9
Address JSON 10

Structural Design **11**
Overall Class Diagram 11
UML Breakdown 12
The Application Manager 12
The User 13
A Listing 14
Tenant’s Favorites List 15

Class Descriptions **15**

Classes 15
Interfaces 16
Enums 17

Dynamic Model 17
Scenario 1 17
Scenario 2 18
Scenario 3 19

Non-functional Requirements 20
Look and Feel Requirements 20
Usability Requirements 20
Performance Requirements 21
Maintainability and Support Requirements 21
Security Requirements 21
Cultural Requirements 21
Legal Requirements 21

Supplemental Documentation 21

Introduction

Purpose

This is the design document for the housing application. The purpose of this document is to record and display all the classes, attributes, and methods of the student off-campus housing management system. This document will cover the design overview, interfaces, data stores, structural design, class descriptions, a dynamic model with several scenarios, and non-functional requirements.

System Overview

This management system contains an application manager to store and display all of the rentable listings, a function to search and filter through the listings and receive relevant results, and allow students/property managers/real estate agents to create and manage accounts.

References

- Team Alpaca Requirements Document

Definitions, Acronyms, and Abbreviations

- Refer to Team Alpaca Requirements Document

Design Overview

Introduction

This system is designed to display and manage listings for college undergraduate students who are looking for off-campus housing. It has features such as containing an inventory of the listings, and keeping track of when they are rented out or available. It also keeps track of the type of rental. This system will allow users to search and filter to be able to find exactly what they are looking for with ease. This system will only be run on a single machine in an Integrated Development Environment reading input from JSON files and receiving user input via the console.

Environment Overview

This system will reside on a single machine and will be compiled and executed using Java. The program will get its input through the console in an IDE such as Eclipse.

Constraints and Assumptions

- **Schedule Constraints**
  - The system shall be fully implemented and operational within the time allowed by the technical director, Portia Plante. Date TBD.
- **Monetary Constraints**
  - The system must be developed and produced within the allotted budget of $0.
- **Technical Constraints**
  - The system code must be written entirely in Java.

Interfaces and Data Stores

System Interfaces

This is a text-based application, interfaced solely through the terminal. We will be using a visual distinguisher (">") to collect user-inputted data.

Scenario 1

Creating an Account & Logging In

```
> register
Enter your first name: Dalton
Enter your last name: Craven
Enter your email: djcraven@email.sc.edu
Enter your phone number: 8037774177
Enter your birthday (mm/dd/yyyy): 09/07/2000
Enter your permanent address:
300 Main St.
```

```
Columbia, SC 29201

Enter a password: ********
Please choose an account type ([S]tudent or [P]roperty
Manager): S
Enter your student ID: G10472969

> login
Enter your username: djcraven
Enter your password: ********

--- Cola Housing v1.0 ---
Welcome, Dalton!
0 new notifications.
5 roommate requests.
2 favorite listings.
```

Scenario 2

Creating a New Property & Listings

```
> login
Enter your username: propmanager
Enter your password: ***********

--- Cola Housing v1.0 ---
Welcome, Dalton!
17 new lease requests.
50 available listings.
100 total listings.
5 total properties.

> create property
Would you like to create a new property (Y/n)? Y
Enter property name: 650 Lincoln
Enter property address:
650 Lincoln St.
Columbia, SC 29201
Enter property description: A campus partner apartment complex,
close to retail and the Strom Fitness Center with a
resort-style pool, state-of-the-art clubroom and private study
spaces on each floor.
Enter neighborhood: USC Campus
Score the walk to campus (1–10): 2
```

```
What type of property is it ([T]ownhouse, [A]partment Building,
Apartment [C]omplex, [H]ouse)? C
Enter number of bedrooms: 4
Enter number of bathrooms: 2
Enter available transportation options: USC campus shuttles,
on-site parking
Enter available amenities: Gym, pool, movie theater, game room
Enter quiet hours: 12:01–6:00
Are the listings furnished (Y/n)?: Y
Are pets allowed (y/N)? N
Is subleasing allowed (Y/n)? Y
Enter number of listings to generate: 25
-- Saving Property + Generating Listings --
-- Listings Generated! Please add lease details and mark them
as available. --
Property ID: #6
Listing ID: #2

17 new lease requests.
50 available listings.
125 total listings.
6 total properties.
```

Scenario 3

Viewing a Listing

```
> listing view --all
Available:
#1: 650 Lincoln – 2 Bedroom (300/500)
#2: 650 Lincoln – 4 Bedroom (100/500)
#4: Cayce Cove – 2 Bedroom (75/475)

Unavailable:
#3: 650 Lincoln – 1 Bedroom (0/50)

> listing view 2
#2: 650 Lincoln – 4 Bedroom (100/500)
$4,000/sem
A campus partner apartment complex, close to retail and the
Strom Fitness Center with a resort-style pool, state-of-the-art
clubroom and private study spaces on each floor.
```

```
Four Bedroom, Two Bathroom Apartment Complex
Semester or Two-Semester Lease

Would you like to view all details (Y/n)? Y
Property Name: 650 Lincoln
Listing Name: 650 Lincoln – 4 Bedroom
Rating: 4.5/5.0
Cost: $4,000/semester
Payment Options: Cash, Check, Card
Lease Duration: Semester, Two-Semester
Individual Leases: Yes
Description: A campus partner apartment complex, close to
retail and the Strom Fitness Center with a resort-style pool,
state-of-the-art clubroom and private study spaces on each
floor.
Neighborhood: USC Campus
CampusWalk™ Score: 2
Property Type: Apartment Complex
Bedrooms: 4
Bathrooms: 2
Transportation: USC campus shuttles, on-site parking
Amenities: Gym, pool, movie theater, game room
Quiet Hours: 12:01–6:00
Furnished Listings: Yes
Pets Permitted: No
Subleasing Permitted: Yes
```

Data Stores

The system will hold all Listings and User data in an ArrayList during active use, and will use JSON files for archiving. There will be a different JSON file for each type of rental property, and one for users. These will be loaded using the JSONReader class upon the startup of the main method, and will be updated from the same class upon the user exiting the system.

Example JSON Files

Listing JSON

```
[{
  "id":"0",
  "name":"listing0",
  "propertyID":"0",
  "neighborhood":"Suburb, nice and friendly",
  "isAvailable":true,
  "cost":1000.00,
  "addressID":"1",
  "paymentOptions":[Cash, Check],
  "leaseDuration":Two Semester,
  "isIndividualLease":false,
  "discount":10.14,
  "rating”:5}
]
```

Manager JSON

```
[{
  "lastName":"Manager",
  "permanentAddressID":"0",
  "dateOfBirth":"Sat Sep 22 00:00:00 EDT 2001",
  "Picture":"",
  "businessName":"Google",
  "firstName":"Matthew",
  "Password":"0000",
  "currentListingID":"",
  "Phone":"3224445555",
  "id":"0",
  "email":"mmanager@gmail.com",
  "listings":[
       {“id”:”3”, “name”:”listing3”}
       {“id”:”5”, “name”:”listing5”}
       ],
  "Username":"mhank"
}]
```

Property JSON

```
[{
  "id":"0",
  "name":"property0",
  "description":"The first listing on the site. A great place right near campus!",
  "neighborhood":"Suburb, nice and friendly",
  "isFurnished":true,
  "propertyType":ApartmentBuilding,
  "Transportation":"campus bus",
  "isPetsAllowed":false,
  "petCost":300.25,
  "amenities":"pool",
  "isSubLeasingAllowed":false,
  "propertySummary":"Small, cute, and quaint!",
  "walkingScore":5,
  "quietHours":"10pm-8am",
  "numberBedrooms": 3,
  "numberBathrooms":5,
  "numberLeases": 2},
 {"id":"1",
  …<property1 info>...},
 {"id":"2",
  …<property2 info>...},
{"id":"3",
  …<property3 info>...}
]
```

Tenant JSON

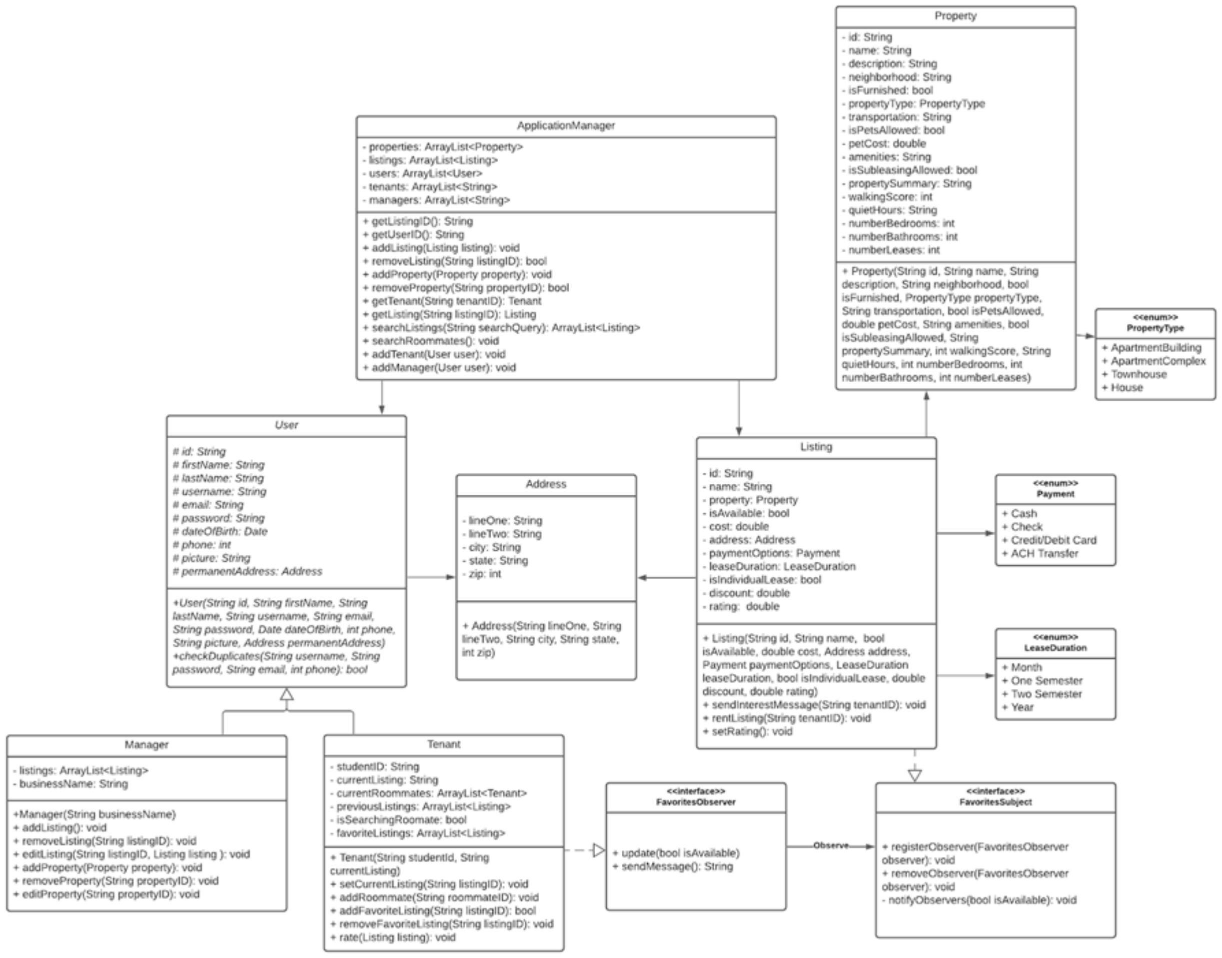
```
[{
  "lastName":"Hank",
  "permanentAddressID":"0",
  "dateOfBirth":"Sat Sep 22 00:00:00 EDT 2001",
  "Picture":"",
  "studentID":"G12345678",
  "firstName":"Matt",
  "Password":"0000",
  "currentListingID":"",
  "Phone":"3224445555",
  "id":"0",
  "email":"mxhank@email.sc.edu",
  "favoriteListings":[
       {"id":"1", "name":"listing1"}
       {"id":"4", "name":"listing4"}
       ],
  "Username":"mhank",
   "currentListingID":"2",
   "previousListings":[],
   "isSearchingForRoommate":true,
   "currentRoommates":[]
}]
```

Address JSON

```
[{
  "zip":29201,"
  lineTwo":"",
  "city":"Columbia",
  "lineOne":"3205 Elmer Street",
  "ID":"0",
  "state":"SC"
}]
```

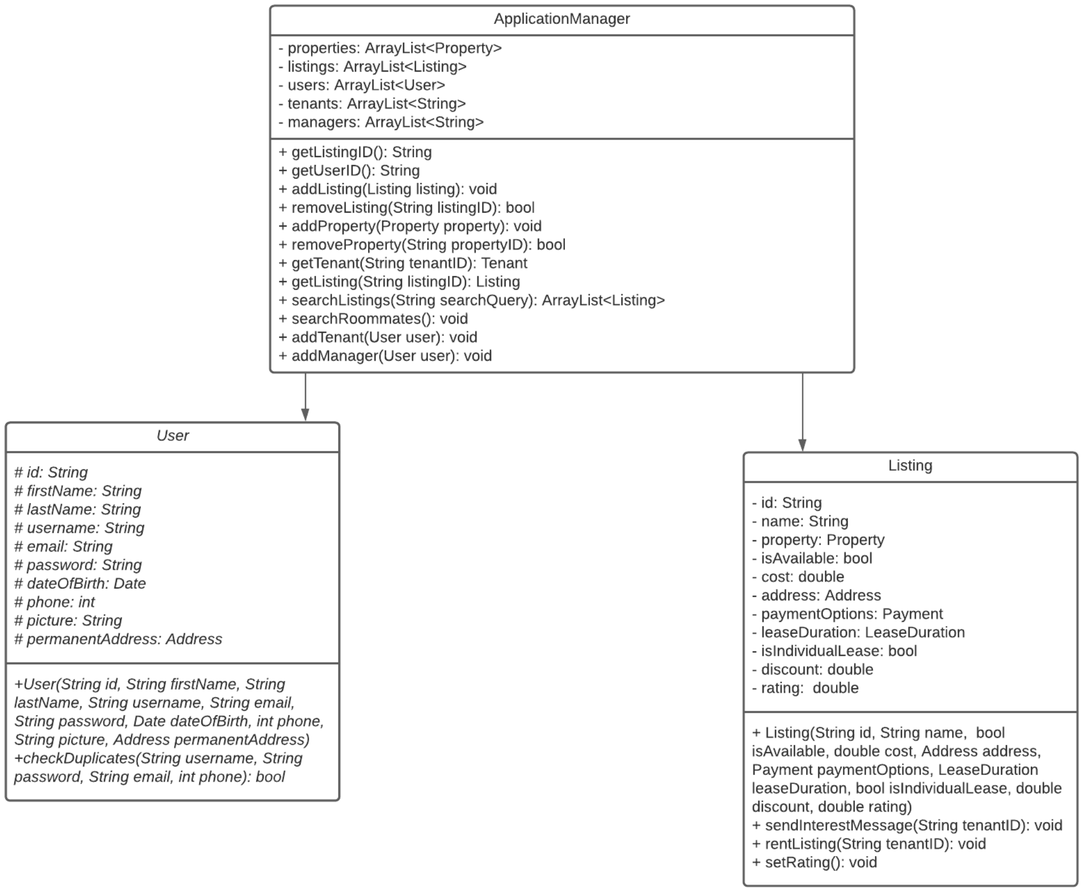
Structural Design

Overall Class Diagram


Property
- id: String
- name: String
- description: String
- neighborhood: String
- isFurnished: bool
- propertyType: PropertyType
- transportation: String
- isPetsAllowed: bool
- petCost: double
- amenities: String
- isSubleasingAllowed: bool
- propertySummary: String
- walkingScore: int
- quietHours: String
- numberBedrooms: int
- numberBathrooms: int
- numberLeases: int
+ Property(String id, String name, String description, String neighborhood, bool isFurnished, PropertyType propertyType, String transportation, bool isPetsAllowed, double petCost, String amenities, bool isSubleasingAllowed, String propertySummary, int walkingScore, String quietHours, int numberBedrooms, int numberBathrooms, int numberLeases)
<<enum>> PropertyType
+ ApartmentBuilding
+ ApartmentComplex
+ Townhouse
+ House
ApplicationManager
- properties: ArrayList<Property>
- listings: ArrayList<Listing>
- users: ArrayList<User>
- tenants: ArrayList<String>
- managers: ArrayList<String>
+ getListingID(): String
+ getUserID(): String
+ addListing(Listing listing): void
+ removeListing(String listingID): bool
+ addProperty(Property property): void
+ removeProperty(String propertyID): bool
+ getTenant(String tenantID): Tenant
+ getListing(String listingID): Listing
+ searchListings(String searchQuery): ArrayList<Listing>
+ searchRoommates(): void
+ addTenant(User user): void
+ addManager(User user): void
User
# id: String
# firstName: String
# lastName: String
# username: String
# email: String
# password: String
# dateOfBirth: Date
# phone: int
# picture: String
# permanentAddress: Address
+User(String id, String firstName, String lastName, String username, String email, String password, Date dateOfBirth, int phone, String picture, Address permanentAddress)
+checkDuplicates(String username, String password, String email, int phone): bool
Address
- lineOne: String
- lineTwo: String
- city: String
- state: String
- zip: int
+ Address(String lineOne, String lineTwo, String city, String state, int zip)
Listing
- id: String
- name: String
- property: Property
- isAvailable: bool
- cost: double
- address: Address
- paymentOptions: Payment
- leaseDuration: LeaseDuration
- isIndividualLease: bool
- discount: double
- rating: double
+ Listing(String id, String name, bool isAvailable, double cost, Address address, Payment paymentOptions, LeaseDuration leaseDuration, bool isIndividualLease, double discount, double rating)
+ sendInterestMessage(String tenantID): void
+ rentListing(String tenantID): void
+ setRating(): void
<<enum>> Payment
+ Cash
+ Check
+ Credit/Debit Card
+ ACH Transfer
<<enum>> LeaseDuration
+ Month
+ One Semester
+ Two Semester
+ Year
Manager
- listings: ArrayList<Listing>
- businessName: String
+Manager(String businessName)
+ addListing(): void
+ removeListing(String listingID): void
+ editListing(String listingID, Listing listing): void
+ addProperty(Property property): void
+ removeProperty(String propertyID): void
+ editProperty(String propertyID): void
Tenant
- studentID: String
- currentListing: String
- currentRoommates: ArrayList<Tenant>
- previousListings: ArrayList<Listing>
- isSearchingRoomate: bool
- favoriteListings: ArrayList<Listing>
+ Tenant(String studentId, String currentListing)
+ setCurrentListing(String listingID): void
+ addRoommate(String roommateID): void
+ addFavoriteListing(String listingID): bool
+ removeFavoriteListing(String listingID): void
+ rate(Listing listing): void
<<interface>> FavoritesObserver
+ update(bool isAvailable)
+ sendMessage(): String
Observe
<<interface>> FavoritesSubject
+ registerObserver(FavoritesObserver observer): void
+ removeObserver(FavoritesObserver observer): void
- notifyObservers(bool isAvailable): void

UML Breakdown

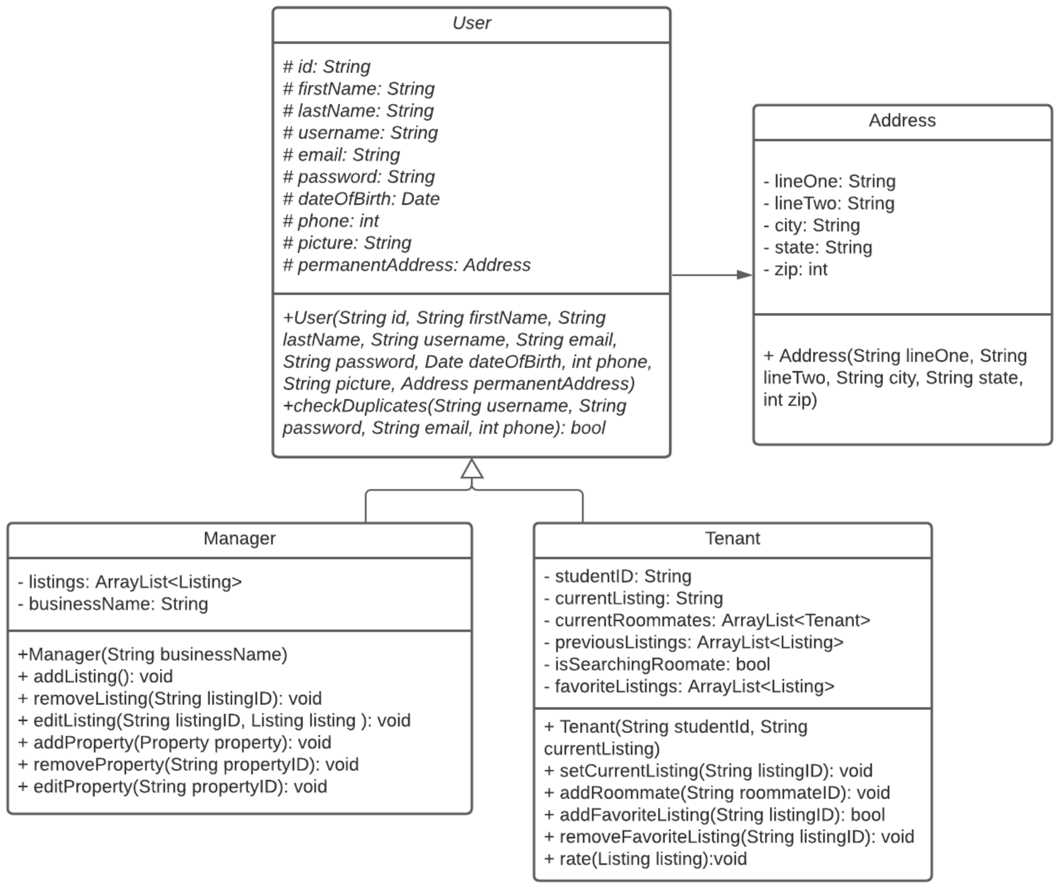
The Application Manager


ApplicationManager
- properties: ArrayList<Property>
- listings: ArrayList<Listing>
- users: ArrayList<User>
- tenants: ArrayList<String>
- managers: ArrayList<String>
+ getListingID(): String
+ getUserID(): String
+ addListing(Listing listing): void
+ removeListing(String listingID): bool
+ addProperty(Property property): void
+ removeProperty(String propertyID): bool
+ getTenant(String tenantID): Tenant
+ getListing(String listingID): Listing
+ searchListings(String searchQuery): ArrayList<Listing>
+ searchRoommates(): void
+ addTenant(User user): void
+ addManager(User user): void
User
# id: String
# firstName: String
# lastName: String
# username: String
# email: String
# password: String
# dateOfBirth: Date
# phone: int
# picture: String
# permanentAddress: Address
+User(String id, String firstName, String lastName, String username, String email, String password, Date dateOfBirth, int phone, String picture, Address permanentAddress)
+checkDuplicates(String username, String password, String email, int phone): bool
Listing
- id: String
- name: String
- property: Property
- isAvailable: bool
- cost: double
- address: Address
- paymentOptions: Payment
- leaseDuration: LeaseDuration
- isIndividualLease: bool
- discount: double
- rating: double
+ Listing(String id, String name, bool isAvailable, double cost, Address address, Payment paymentOptions, LeaseDuration leaseDuration, bool isIndividualLease, double discount, double rating)
+ sendInterestMessage(String tenantID): void
+ rentListing(String tenantID): void
+ setRating(): void

Description:

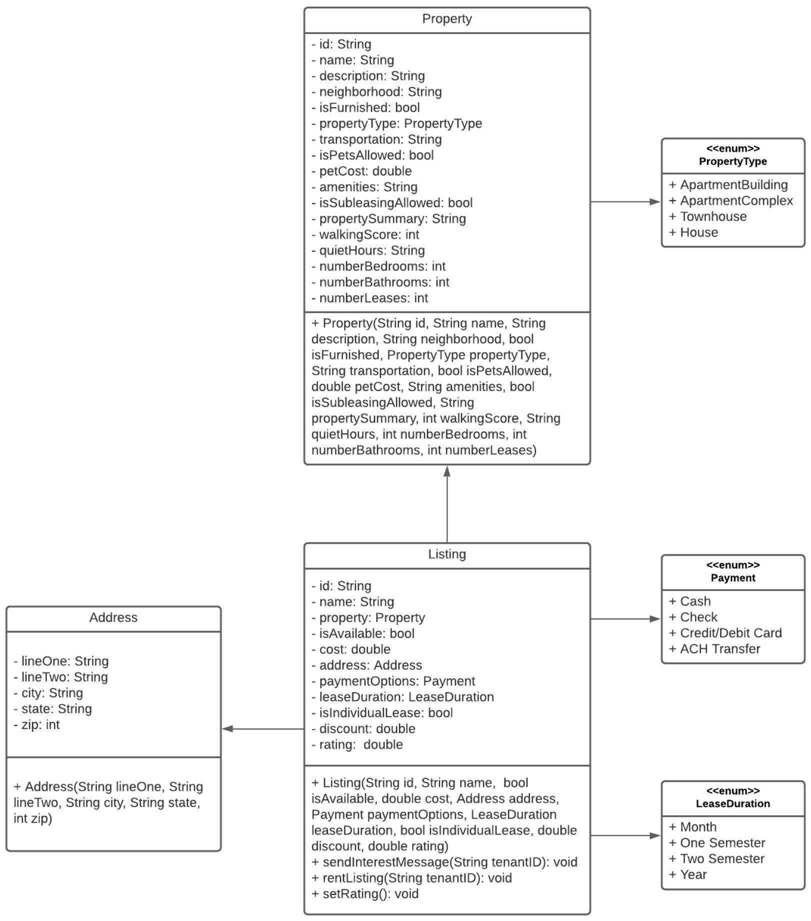
This part of the application is what holds most of the functionality of the program. The application manager is what allows the students using the application to find their perfect off-campus living situation. The user is an abstract base class to act as the base of a user, there will be different types of users. This class just holds the minimum attributes that a user needs to have. Listings is also an important class because this is what holds the methods for creating a listing, which is what the users will be spending most of their time looking at while using this application.

The User


User
# id: String
# firstName: String
# lastName: String
# username: String
# email: String
# password: String
# dateOfBirth: Date
# phone: int
# picture: String
# permanentAddress: Address
+User(String id, String firstName, String lastName, String username, String email, String password, Date dateOfBirth, int phone, String picture, Address permanentAddress)
+checkDuplicates(String username, String password, String email, int phone): bool
Address
- lineOne: String
- lineTwo: String
- city: String
- state: String
- zip: int
+ Address(String lineOne, String lineTwo, String city, String state, int zip)
Manager
- listings: ArrayList<Listing>
- businessName: String
+Manager(String businessName)
+ addListing(): void
+ removeListing(String listingID): void
+ editListing(String listingID, Listing listing): void
+ addProperty(Property property): void
+ removeProperty(String propertyID): void
+ editProperty(String propertyID): void
Tenant
- studentID: String
- currentListing: String
- currentRoommates: ArrayList<Tenant>
- previousListings: ArrayList<Listing>
- isSearchingRoomate: bool
- favoriteListings: ArrayList<Listing>
+ Tenant(String studentId, String currentListing)
+ setCurrentListing(String listingID): void
+ addRoommate(String roommateID): void
+ addFavoriteListing(String listingID): bool
+ removeFavoriteListing(String listingID): void
+ rate(Listing listing):void

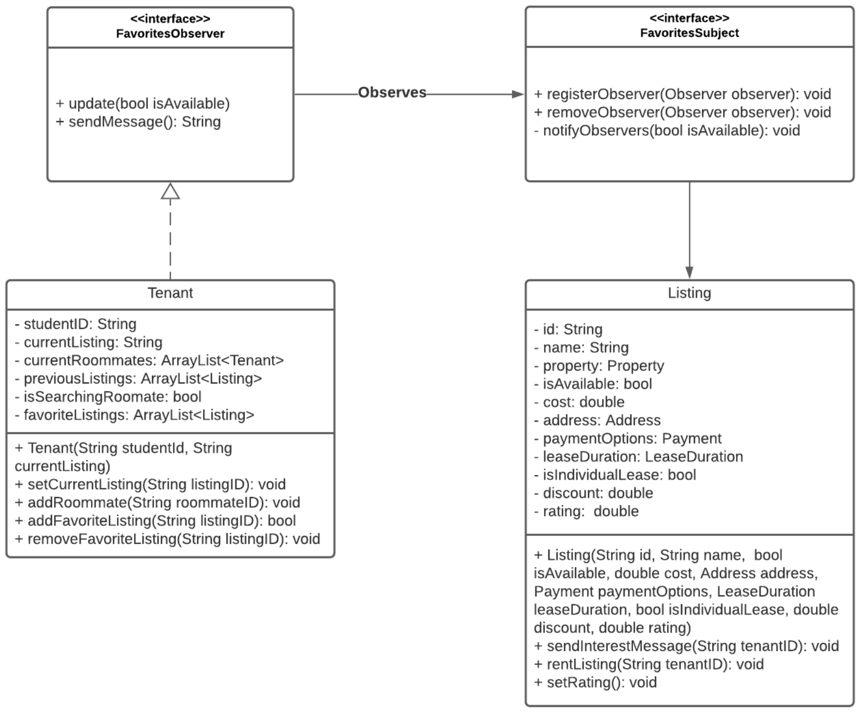
Description:
This part of the application is a breakdown of the user class. We will use *User*, an abstract base class, as a base for each type of user. We will have a tenant, the student, and manager, a real estate agent or property manager. We split these types of users up because the real estate agents and property managers have the same authority in regards to the application whereas the student (or tenant) does not. Therefore, these two types of users have different attributes and methods. We then have a date and address class. These classes were created so that when users input dates and addresses, the format is the same. There are many different ways to enter this information, so to ensure our application will work we had to be sure all users entered this information in the same format.

A Listing


Property
- id: String
- name: String
- description: String
- neighborhood: String
- isFurnished: bool
- propertyType: PropertyType
- transportation: String
- isPetsAllowed: bool
- petCost: double
- amenities: String
- isSubleasingAllowed: bool
- propertySummary: String
- walkingScore: int
- quietHours: String
- numberBedrooms: int
- numberBathrooms: int
- numberLeases: int
+ Property(String id, String name, String description, String neighborhood, bool isFurnished, PropertyType propertyType, String transportation, bool isPetsAllowed, double petCost, String amenities, bool isSubleasingAllowed, String propertySummary, int walkingScore, String quietHours, int numberBedrooms, int numberBathrooms, int numberLeases)
<<enum>> PropertyType
+ ApartmentBuilding
+ ApartmentComplex
+ Townhouse
+ House
Listing
- id: String
- name: String
- property: Property
- isAvailable: bool
- cost: double
- address: Address
- paymentOptions: Payment
- leaseDuration: LeaseDuration
- isIndividualLease: bool
- discount: double
- rating: double
+ Listing(String id, String name, bool isAvailable, double cost, Address address, Payment paymentOptions, LeaseDuration leaseDuration, bool isIndividualLease, double discount, double rating)
+ sendInterestMessage(String tenantID): void
+ rentListing(String tenantID): void
+ setRating(): void
Address
- lineOne: String
- lineTwo: String
- city: String
- state: String
- zip: int
+ Address(String lineOne, String lineTwo, String city, String state, int zip)
<<enum>> Payment
+ Cash
+ Check
+ Credit/Debit Card
+ ACH Transfer
<<enum>> LeaseDuration
+ Month
+ One Semester
+ Two Semester
+ Year

Description:
This part of the application breaks down the listing class. When we have a listing, we also have a property. The property holds the information about the rental that a user is interested in like are pets allowed, what amenities are offered, is it furnished, etc. The listing class holds the more business side of information like the price, address, payment options, whether it is available, etc. We decided to break it up like this because we thought it would break things up and allow for more organization. This part will also use the Address class to make sure the address of each listing is entered the same way to ensure the application will run smoothly. We also made payment and lease duration enums. These hold the options for payment and options for the length of a lease, respectively. Since there are only a relative amount of options (and not an unlimited amount) for each of these we decided to make enums for organization.

Tenant's Favorites List



<<interface>> FavoritesObserver
+ update(bool isAvailable)
+ sendMessage(): String
Observes
<<interface>> FavoritesSubject
+ registerObserver(Observer observer): void
+ removeObserver(Observer observer): void
- notifyObservers(bool isAvailable): void
Tenant
- studentID: String
- currentListing: String
- currentRoommates: ArrayList<Tenant>
- previousListings: ArrayList<Listing>
- isSearchingRoomate: bool
- favoriteListings: ArrayList<Listing>
+ Tenant(String studentId, String currentListing)
+ setCurrentListing(String listingID): void
+ addRoommate(String roommateID): void
+ addFavoriteListing(String listingID): bool
+ removeFavoriteListing(String listingID): void
Listing
- id: String
- name: String
- property: Property
- isAvailable: bool
- cost: double
- address: Address
- paymentOptions: Payment
- leaseDuration: LeaseDuration
- isIndividualLease: bool
- discount: double
- rating: double
+ Listing(String id, String name, bool isAvailable, double cost, Address address, Payment paymentOptions, LeaseDuration leaseDuration, bool isIndividualLease, double discount, double rating)
+ sendInterestMessage(String tenantID): void
+ rentListing(String tenantID): void
+ setRating(): void

Description:
Lastly, we have an option for tenants to add listings to their "favorites list." We decided to use a design pattern we recently learned so that tenants would be observing their favorites list. We implemented the observer idea because it allowed us to keep the idea of organization and kept everything functional.

Class Descriptions

Classes

Application Manager:
Purpose: The main system of the program that keeps track of the listings and their availability, users, and account creation/login.
Property:
Purpose: A class to hold all of the attributes for a property that will be used in the system. This is the actual property and what parameters the manager should include.

User:
Purpose: This is an abstract base class to hold all of the attributes needed for a user.

Address:
Purpose: This will specify the format for holding addresses so that all users will enter them the same way.

Listing:
Purpose: This class holds all of the attributes for a listing that will be added to the system. Tells tenants if the properties in a listing are available. Includes details on a group of properties, such as special deals, payment options. This is also where the rating for a group of properties will be kept.

Manager:
Purpose: This creates a special user account that is called manager. This account can add or edit listings to the application.

Tenant:
Purpose: This creates a user account specified as a tenant. This allows the user to make a favorite listings list, add roommates, and set a current listing. This is different from a manager class where they are able to add listings.

Interfaces

FavoriteObserver:
Purpose: This designates who can watch (observe) a list created of listings specified by the tenant. It will notify the observers of when one of their favorite listings is no longer available.

FavoriteSubject:
Purpose: This designates what can be watched (subject) by a designated observer. This gives the ability to add, remove and notify the observers.

Enums

LeaseDuration:
Purpose: Sets the parameters that the manager can lease out an apartment on the application.

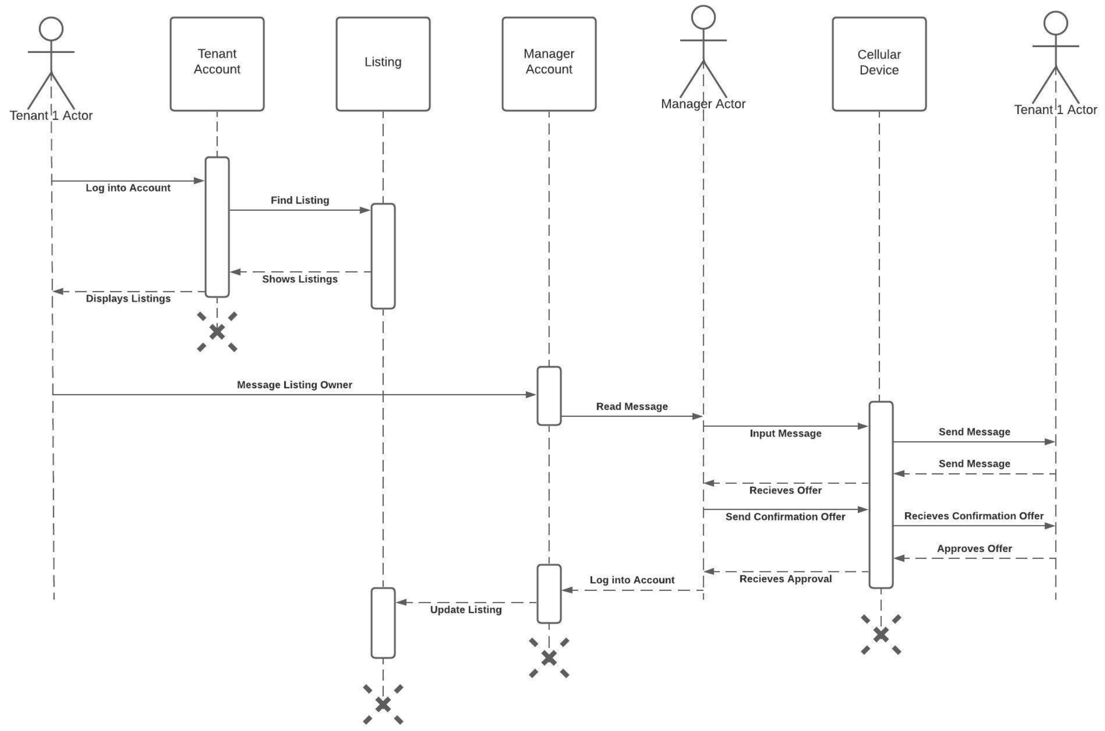
Payment:
Purpose: Designates the type of payment the manager accepts.

PropertyType:
Purpose: Designates what kind of property is able to be added to the site.

Dynamic Model

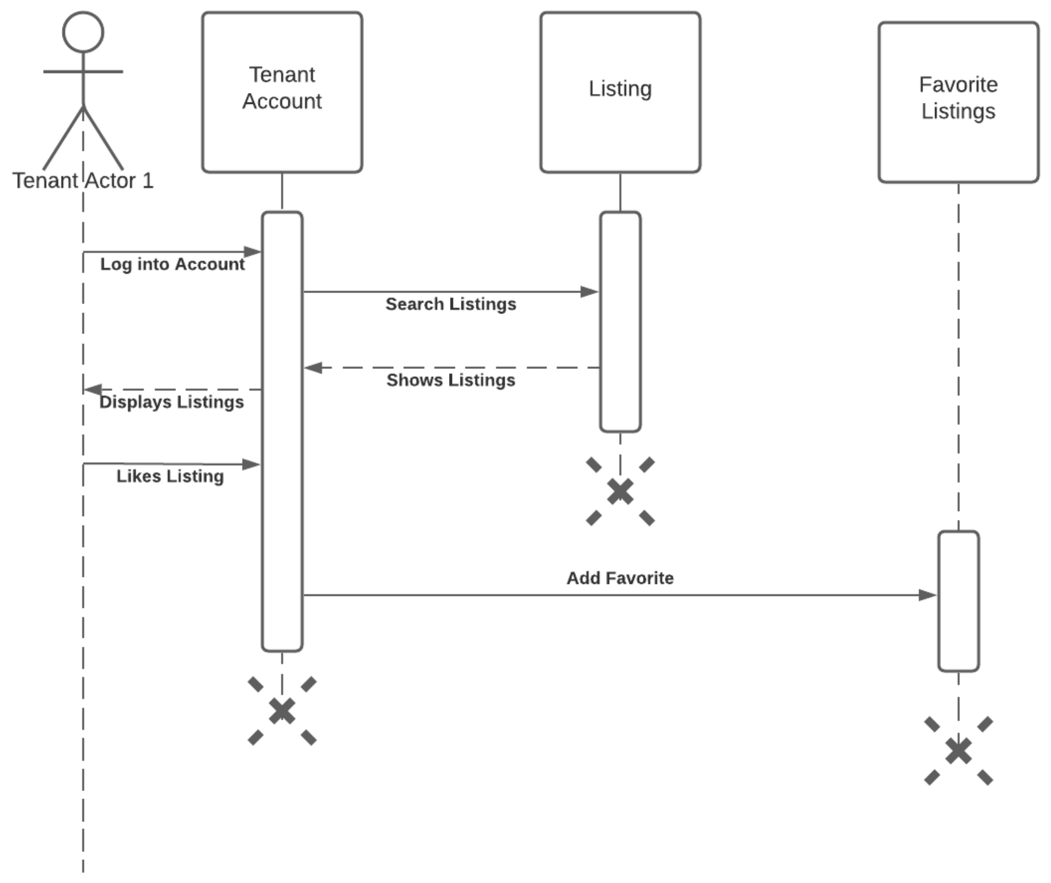
Scenario 1

- **Scenario Name:** Rental Process
- **Scenario Description:** User will log into their account, look for a listing, the system will show listings and display them to the user, the user will find one they are interested in, and message the manager affiliated with the property. The manager will then read the message and send one back. They can then discuss prices and once the lease is finalized, the manager will go into the system and update the availability.
- **Sequence Diagram:**



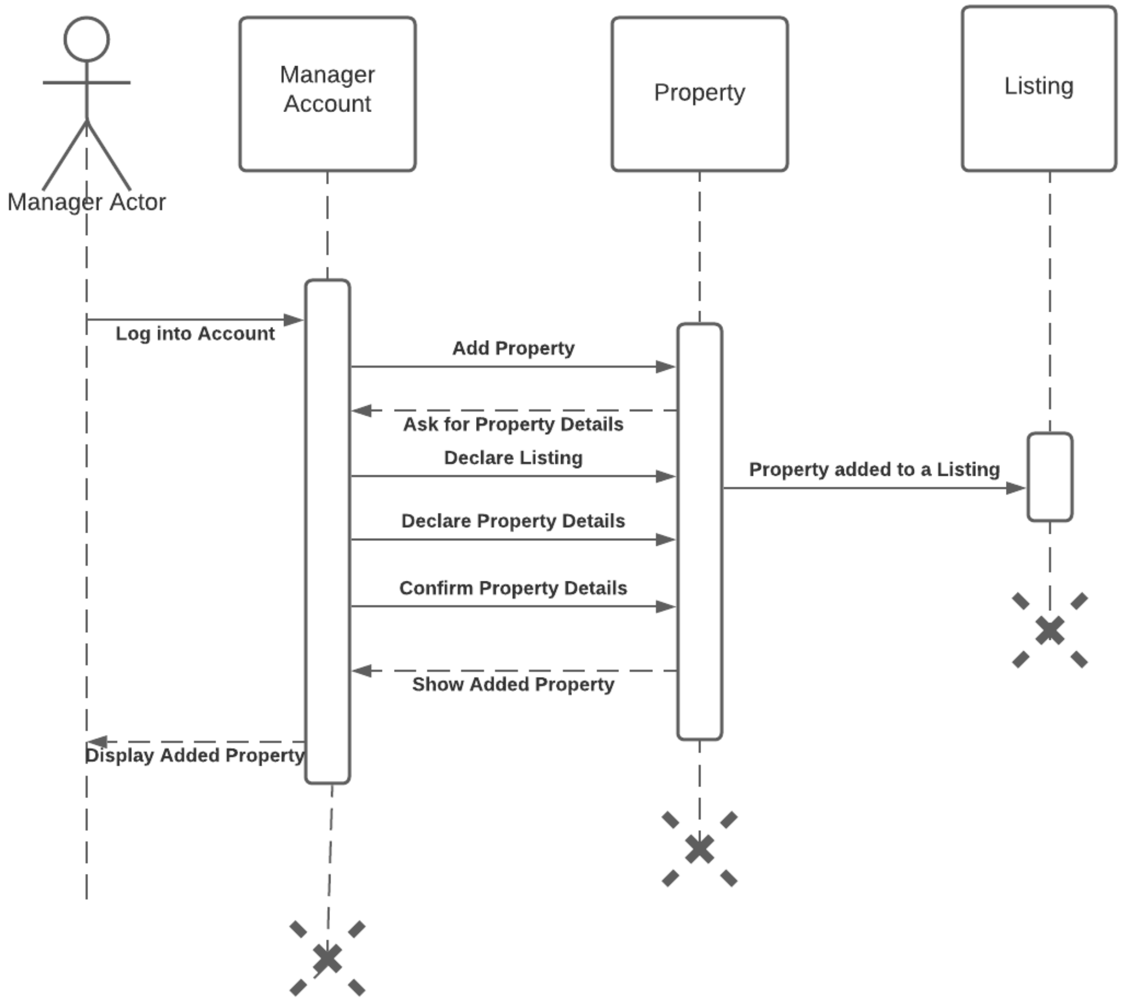
Scenario 2

- **Scenario Name:** Favoriting Process
- **Scenario Description:** User will log into their account, look for a listing, the system will show listings and display them to the user, the user will find one they like. The user can then notify the system and it will be added to their "favorites" list.
- **Sequence Diagram:**



Scenario 3

- **Scenario Name:** Adding Property
- **Scenario Description:** The user (who is a property manager or real estate agent) will log into their account so they can then start the adding process. They will be prompted, from the system, for the property details. They will then add them, confirm everything and then "publish" the listing.
- **Sequence Diagram:**



Non-functional Requirements

- **Look and Feel Requirements**
    - The system shall be organized so that potential tenants can efficiently find housing near the University of South Carolina.
    - The system shall be organized so that property managers can easily manage their listings.
- **Usability Requirements**
    - The system shall be able to be used by potential tenants associated with the University of South Carolina.

    - The system shall be able to be used by property managers in the Columbia area.
- **Performance Requirements**
    - The system shall be able to operate on any user device capable of running Java applications (Windows, macOS, Linux).
- **Maintainability and Support Requirements**
    - The system shall be able to be supported on any device capable of running Java applications.
    - The system shall allow users to manage listings without developer support/assistance.
    - The system shall allow users to search for listings without developer support/assistance.
- **Security Requirements**
    - The system shall check potential tenants' school IDs in order to make an account.
    - The system shall require a business or property address to be listed for property managers to create an account.
- **Cultural Requirements**
    - The system shall be nondiscriminatory of any cultural linkage in any format on the application.
- **Legal Requirements**
    - The system shall comply with federal, state, and local laws, as well as university policy, regarding the storage of any user's personal information.

Supplemental Documentation

Software Requirements Specification