Pending Group: Design Document Draft

**Outline** of program structure

1. Rental Types
   1. Vacation (airBnB)
      1. ID
      2. Description
      3. Daily, Weekly, Monthly
      4. Due Upon Arrival
   2. Apartment
      1. ID
      2. Description
      3. Starts as 3/6/12 month lease
      4. Goes to Month to Month after lease end
      5. Due End of Month
   3. Single House
      1. ID
      2. Description
      3. Yearly lease
      4. Goes to Month to Month after lease end
      5. Due End of Month
2. Properties
   1. 3 days for prior notice
   2. 2 days for notice of non-payment
   3. 5 additional days for 2nd notice of non-payment
   4. Fees/extra
      1. Late Fees Apply to Single House and Apartment
      2. Security Deposit applies to Single House and Apartment
      3. Cleaning Fee applies to Vacation
3. Database #1 contains Rentals.
   1. Numerical ID for each rental
   2. Metadata+Description
4. Database #2 contains Tenants
   1. Has ID, can include any number of people.

**Additional Details**

Each rental is one of three types - vacation, apartment, or single family. Each of these type are a subclass of the *Rental*. Because different fees may be applied to different properties, the *Rental* superclass abstractly describes some methods, which are then overridden by the various rental types. These methods are enumerated in the UML diagram attached. The *rental* superclass should declare all methods and variables that are shared by the various rental types. This means street addresses, rent amounts, descriptions, and a *DatePair* array to indicate availability.

Because a Tenant can include multiple people (per project description). The class *tenant* includes an array of people, along with a billing address, which is tied to one member of the array of people. We let *billableMember* be an integer which indexes the member in the array.

Let each reservation have an ID, when inserting new reservation, if two reservations have same id (are just an update to initial reservation) doesn’t matter if their times conflict

*DatePair* consists of a *Tuple*, a class we define as an ordered pair of objects, start and end. *DatePair* specifically uses an ordered pair of *date* objects. Upon instantiation, both values are set to null.

Updating the date pair compares the start and end dates to other *DatePair* objects that are in a sorted list, finds the *DatePair*s that starts before and after the start date, and then attempt to place itself between them, ignoring any datepair with the same ID, in order to avoid a struggle with changing the date on an existing reservation. The datepair will then check to see, in a similar manner, if it’s end date conflicts with the start date of the next reservation. We then can return any conflicting reservations.

The database will have monitor locks to allow for multiple users to update data at the same time without causing a conflict. Attempts to load a Rental or Tenant’s profile simultaneously by multiple clients will result in a waiting message being displayed.