SOFT252 Reflective Document

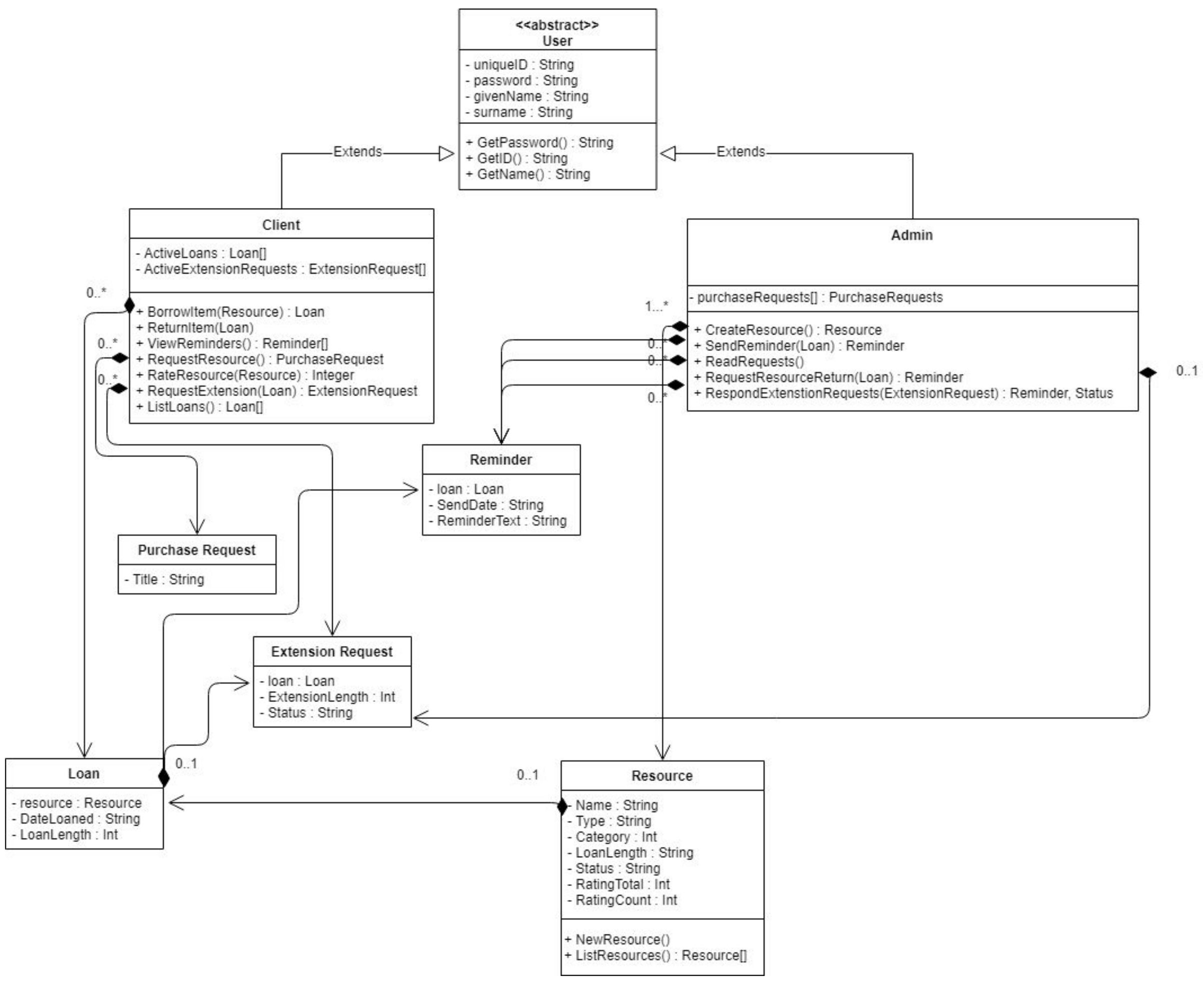
JACK GRIFFITHS - 10547816
GitHub Repo - https://github.com/JackDotGriffiths/SOFT252-Library-Management
Login Details - Available in ‘LOGIN - Details.txt’ supplied in .zip

Design Process

I started off by creating a simple UML diagram for the system. This was the start of my research into the layout the structure in the program. It was clear from the specification that I required two user types, and hence I created a ‘User’ superclass that defined the basic layout of a user, and then ‘Client’ and ‘Admin’ that extended the superclass.

From reading through the specification, I could also create a few other elements that would be required for the system to operate. An example of this is a Loan. A loan will have a Client attached (the person who loaned it) , as well as a Resource (The item being loaned). I opted for aggregation as a Loan will always have a Client and Resource attached.

Initial UML



Positives of my Design

My Design makes use of many singletons. These assist with dealing with various users & resources from different places in the program. Firstly, I used a singleton to store a UserManager. This UserManager allowed me to quickly reference users quickly and simply around the program. I also created methods on this UserManger that returned all Users as an ArrayList, which came in helpful when creating the Controller that allows users to Login.

My second example is a ResourceManager, which, much like the UserManager, allowed me to reference and use Resources at any place in the program efficiently. This also allowed for easy addition of resources.

In order to easily control the ‘permissions’ of the program, I created 2 different JFrames which held the various possible functionalities accessible to the logged in user. This allowed for easy functionality control, preventing any user from seeing things they’re not supposed to.

I am quite proud that I was able to meet all of the User Requirements listed in the specification, as all items on the list are accounted for.

Finally, I stuck to the Model-View-Controller architecture when creating this system. Keeping my Model separate from the Controller allowed me to continuously add and edit bits of code without worry that the crossover could cause errors. Debugging my program came very quickly and easily thanks to the creation of the Model, and the View was very simple to create once the functionality was clear.

I also believe that my Commits were regular, however the commit titles could have been more cohesively worded to allow easy tracking of progress.

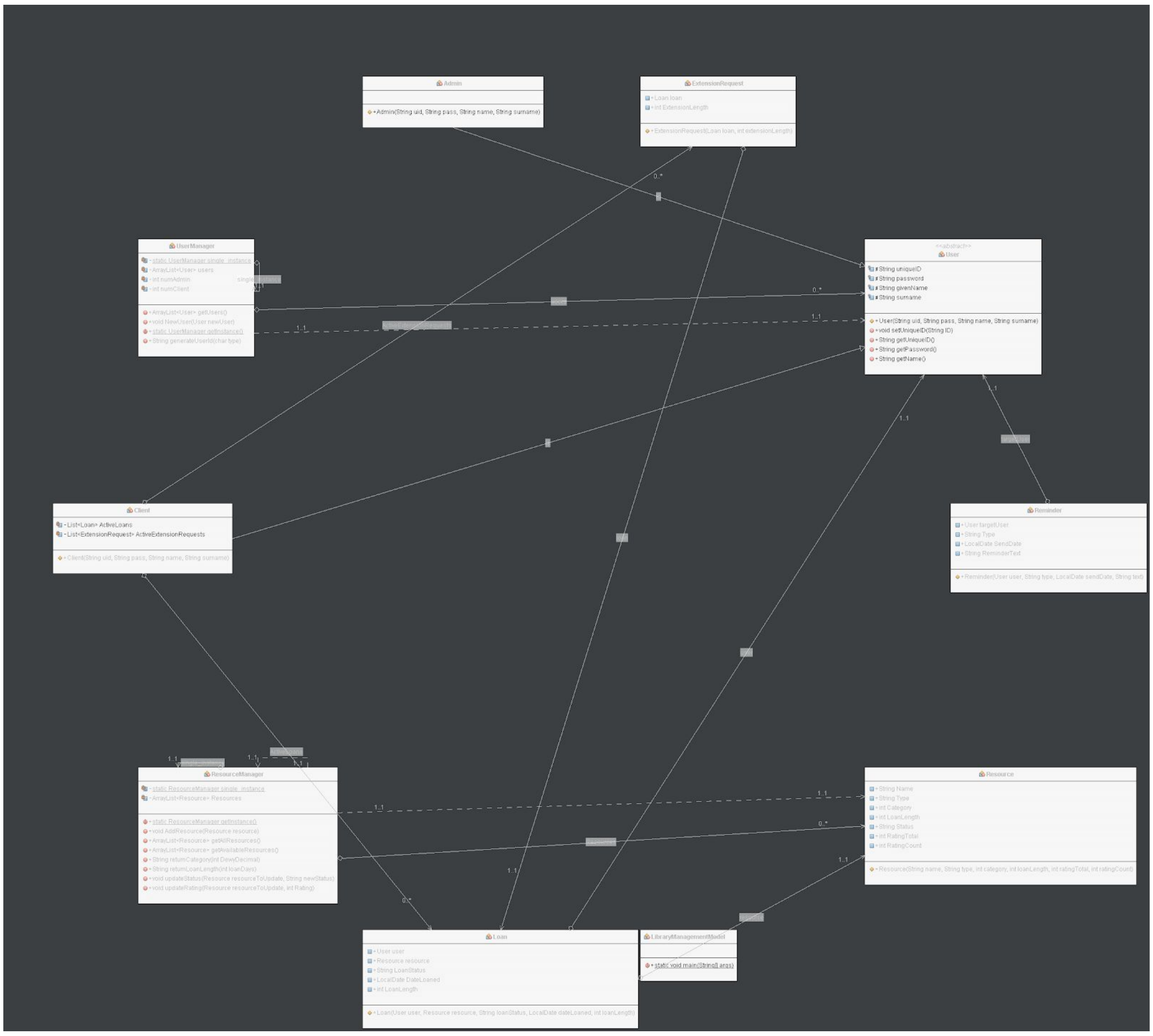
Negatives of the Process

Firstly, I struggled with creating JUnit Testing for the system, so I was unable to test that my Model worked before going into creating the View and Controller. I attempted to get a few tests in but was unable to get them functioning before the deadline. Therefore instead, my testing of this system came from creating the View and then testing the output using the console.

Secondly, while going into creating the Controller and View for the system, I soon came to realise that my Model wasn’t up to scratch with the full functionality required from the system. Thankfully, due to the design of the system, I was able to go in quickly and add/alter bits to suit my new plans. This resulted in a more complicated UML diagram, which is pictured below.

Finally, both a Negative and Positive, The system is a bit lacking in functionality, as the Admin is not able to create users, and data is not saved after closing the program.However , Thanks to the MVC design, additional features would not be too hard to implement in future development.

Updated UML


Admin
+Admin(String uid, String pass, String name, String surname)
ExtensionRequest
+Loan loan
+int ExtensionLength
+ExtensionRequest(Loan loan, int extensionLength)
UserManager
-static UserManager single_instance
-ArrayList<User> users
-int numAdmin
-int numClient
+ArrayList<User> getUsers()
+void NewUser(User newUser)
+static UserManager getInstance()
+String generateUserId(char type)
single_instance
<<abstract>>
User
#String uniqueID
#String password
#String givenName
#String surname
+User(String uid, String pass, String name, String surname)
+void setUniqueID(String ID)
+String getUniqueID()
+String getPassword()
+String getName()
Client
-List<Loan> ActiveLoans
-List<ExtensionRequest> ActiveExtensionRequests
+Client(String uid, String pass, String name, String surname)
Reminder
+User targetUser
+String Type
+LocalDate SendDate
+String ReminderText
+Reminder(User user, String type, LocalDate sendDate, String text)
ResourceManager
-static ResourceManager single_instance
-ArrayList<Resource> Resources
+static ResourceManager getInstance()
+void AddResource(Resource resource)
+ArrayList<Resource> getAllResources()
+ArrayList<Resource> getAvailableResources()
+String returnCategory(int DewyDecimal)
+String returnLoanLength(int loanDays)
+void updateStatus(Resource resourceToUpdate, String newStatus)
+void updateRating(Resource resourceToUpdate, int Rating)
ActiveLoans
Resource
+String Name
+String Type
+int Category
+int LoanLength
+String Status
+int RatingTotal
+int RatingCount
+Resource(String name, String type, int category, int loanLength, int ratingTotal, int ratingCount)
Loan
+User user
+Resource resource
+String LoanStatus
+LocalDate DateLoaned
+int LoanLength
+Loan(User user, Resource resource, String loanStatus, LocalDate dateLoaned, int loanLength)
LibraryManagementModel
+static void main(String[] args)
0..*
1..1