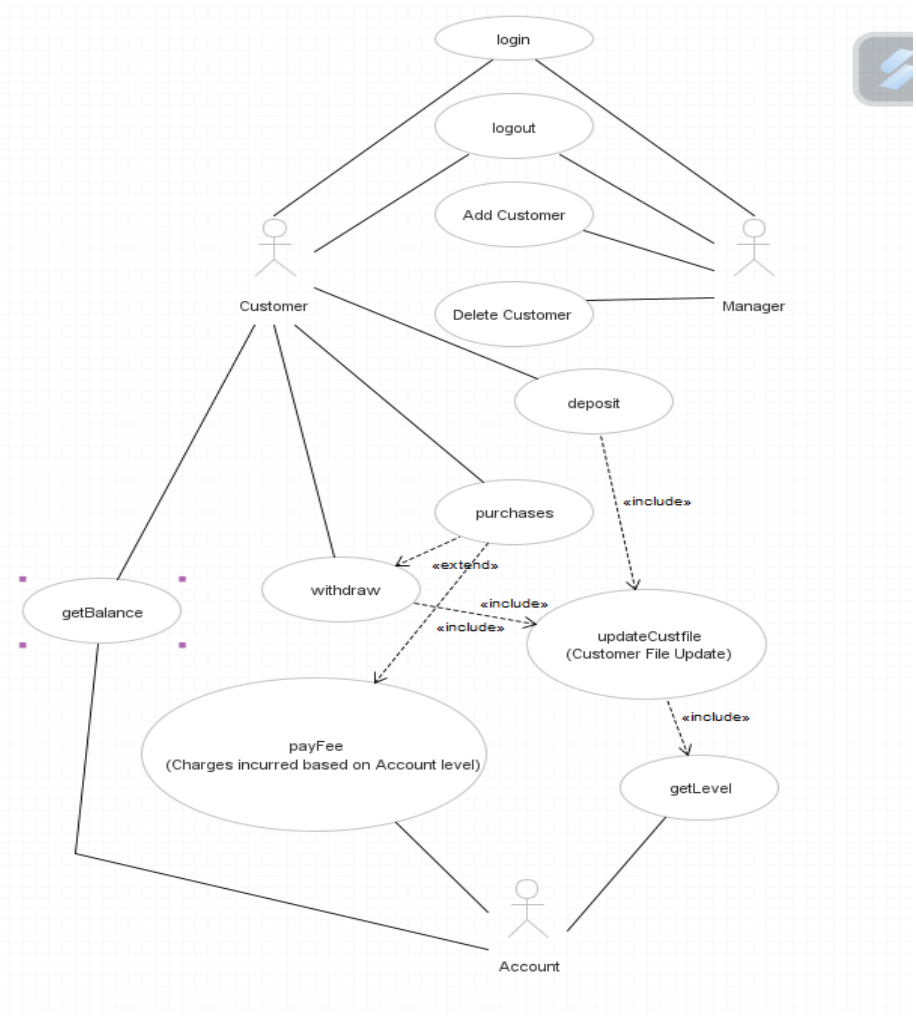
COE528 - Project: Bank Application | Name: Hasib Bhuiyan | ID: 501169402 | Section 02

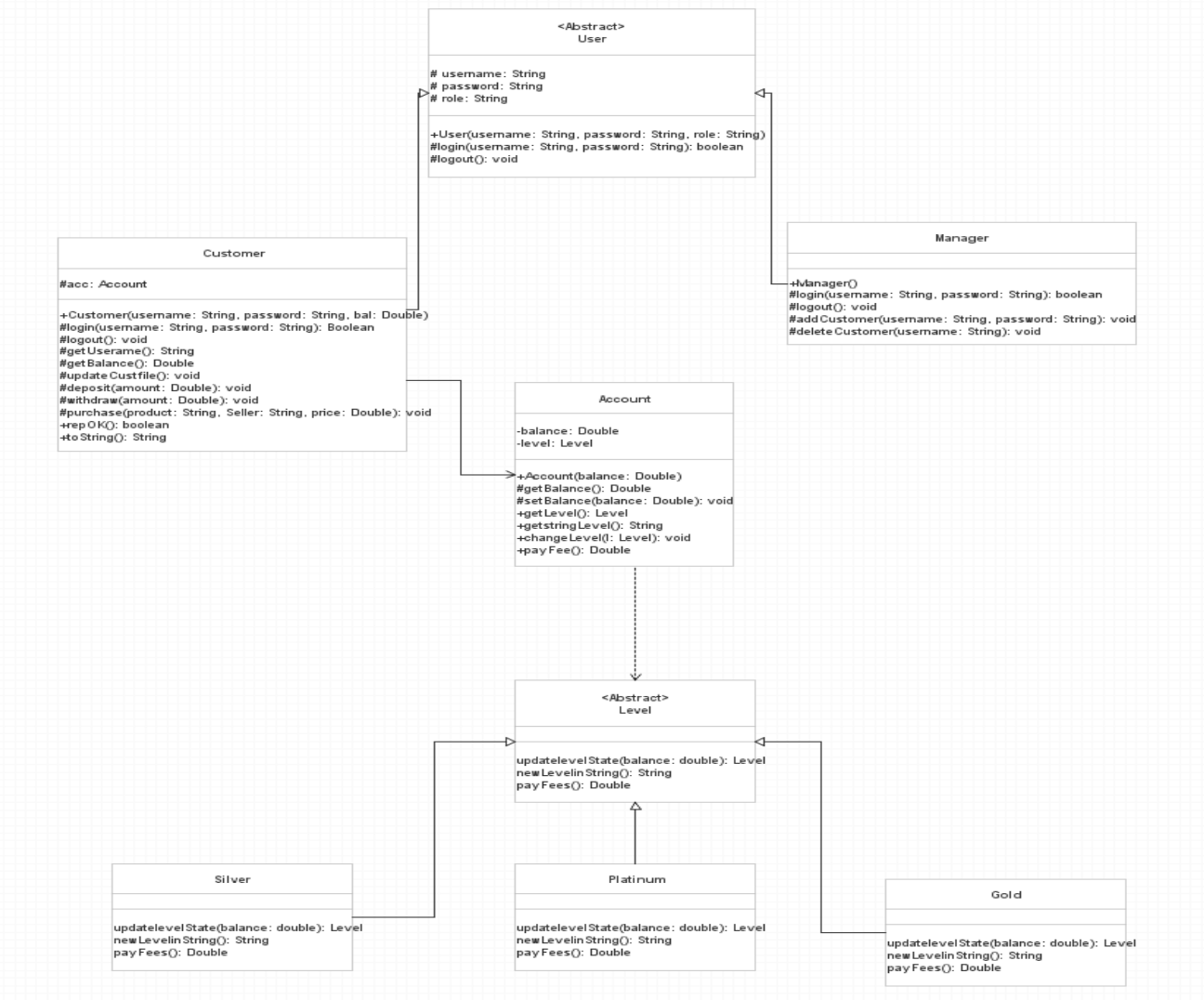


Name: Purchase
Participating Actors: Initiated by Customer, requires communication and information from Accounts
Flow of Events:
1) The customer chooses to make an online purchase, by clicking on the purchase button
2) The customer enters the information, from product name, product producer, and specifically the price of the product.
3) The customer's purchase must be checked and the total charges including the fee from the Account(payFee() function), must be compared with the balance on the Account.
4) If purchase is successful, total charge must be taken from account through the withdrawal function; withdrawal function will implement the changes and deductions from balance and update the customer info file.
5) Customers will get feedback whether the purchase goes through or not.
Entry Condition:

- Customer successfully logged into account, and chose to click the ‘purchase’ button
- Customer provided valid input of details concerning the product and specifically an appropriate price; the prices cannot be under $50, and cannot be more than the account balance.

Exit Condition():

- Customer successfully makes a purchase and receives a message detailing the purchase information
- Customer’s purchase transaction does not go through due to invalid input given; will receive a message detailing the failure of transaction
- Customer clicks on “Return back to Customer Page”, and is redirected back to customer homepage.



Description:

The class “User” is an abstract class and it represents a basic framework of the client actor in the bank application. It consists of general and common client attributes including username,

password and the role, and behavior functions like login and logout. The classes, “Customer” and “Manager” are more specialized classes that extend the User class. The Customer class represents the majority of the clients, that are customers. The customer class, unlike the Manager class, requires the use of an “Account” class that manages and sets principles during transactions. The customer class can be involved in a number of transactions, including deposit, withdrawal and purchase; these changes from transactions are also recorded on the customer’s unique file, by implementing the “updateCustfile” method. Thus many of the transaction functions like deposit and withdrawal include the “updateCustfile”. The transaction function of purchase also extends to the withdraw function as the purchase takes money out of the account, and purchase also includes the payFee function located in the Account class to evaluate the fees incurred from the purchase. The customer class also has its essential functions like the getter methods, and it is the class that uses the repOK function. Now the Manager class really just includes one manager for this project, the username and password being “Admin”. Through the manager class besides the login and logout, there is the addCustomer and deleteCustomer class which of course adds and deletes customers; this includes making the file, and deleting the file, and invoking the customer class to finally add the customer. The Account class as mentioned above controls transaction behaviors including balance, and something referred to as the “Level” which is another class that determines the fee and level of account. The Level class is another abstract class that has three types of levels that extend from it; the Silver, Gold and Platinum classes, which all include, “updatelevelState”, “newLevelinString” and “payFees”. The updatelevelState essentially reevaluates the level and returns the corresponding current level, this is further on used by the Account class to change it’s Level, by invoking through the Level variable itself. The newLevelinString basically gives the String version of the level. Finally ofcourse the fees are determined by the level, so when the Account class class its fee function, it really just retrieves the info by invoking from its level class.

For the one class that requires an Overview clause, abstraction function, and requires the need to implement the repOK and toString() methods, I chose the **Customer class**, especially since this class can be considered one of the main classes, as it holds crucial behavior and properties of the entire project. The customer class has all the “effects”, “modifies”, and “requires” clauses as necessary.

Now the project also required us to use the State design pattern, which gives an object the capability to modify its behavior and essential functionality as internal attributes and states change in specific ways as outlined. This can clearly be seen, when we look at the lower part of the UML class diagram with the Account, Level, and the three inherited classes of Level. The Level class is the exceptional class that can alter its behavior, it is abstract as it's not really considered a class in itself, but a point from where different behaviors branch out from; that is the SIlver, Gold and Platinum classes each holding a different functionality and respond differently depending on the state of Level. This is later used by the Account class since it uses the Level class, this provides assistance to the Account class.