Jamal Abdi

500770871

Section 3

The Bank interface uses a file called accs to store all the user information. The method startup then reads the file and stores all the information into an array of users. Then the interface will ask for the username and password. If it’s the manager, then they have the manager interface and if it’s a customer they use the customer interface. The manager can make accounts, delete accounts and display all the accounts.

The use case diagram has two actors the manager and the customer. The manager is a secondary actor and the customer is a primary actor. The manager can make accounts, remove accounts and also display all the accounts. The customer can check balance, make a withdrawal, make a deposit and also make an online purchase. Every time the customer makes a purchase, withdrawal or deposit there is a mandatory status update. In the diagram those cases include a status update. When a change is made to the balance the status update updates the status of the customer.

The class diagram in effect demonstrates the classes of the code. There are two classes a manage and a customer. They all extend the main Users class. The difference of customer and manager is that they have a different role and different method. The manager has the create account method, remove account method and the display accounts method. The customer has the display balance method, deposit method, withdrawal method and online purchase method. There is also only one manager so that is also on the UML.

I selected the users class for the analysis. It is the bulk of the code and has a lot of the methods. Including write to file and all the methods used in customer and manager. It contains an object User’s which includes the username, password, role, balance and the status. For the repOK the object had to have a user with an integer for balance and string for the other parameters. The abstraction function is also in the toString method, and it also has the rep invariant.

All the classes are mutable because they can change their parameters. The customer is allowed to change their balance by spending or depositing money. They can also change their status by depositing and spend money. There are safeguards to ensure that the rep is not exposed, like the methods only allow specific changes. The state design ensures that all the changes made to the account is not going to cause an error.

The state design as mentioned earlier also makes it so the rep is not exposed. The customer is offered a lot of freedom in terms of what they can do with their account. The repOK is used to make sure that the balance is an integer while the rest of the parameters are a string. When the manager makes an account they are asked for the username and password, and it is therefore assumed that the balance is 100, the role is customer and the status is Silver.