Eideen Mozaffari

UID: 105988436

Project 7 Report: Employee Account and Admin Account

***Added Features***

I created a new class named EmployeeAccount, which is an extension of the VacationAccount class. This class can record the number of hours clocked in by an employee and the ability to keep track of an employee account’s vacation balance, which can be increased either through donating blood, having an admin give them bonus hours, or through a promotion. This transitions into the next new class I created called AdminAccount. The admin has access to the employee’s account information and can grant the employee extra vacation hours or give them a promotion. The business idea behind these two classes is to allow a user to consolidate information about an employee, whether it be the number of hours they have worked, their title, or even available vacation hours. Furthermore, an admin could be a boss or higher up, that can give an employee a promotion or a bonus for the work they have put in, and they can use the functionalities in this program to update the consolidated information in the employee account.

***Program Design***

The EmployeeAccount class inherits its ID and initial vacation balance from a VacationAccount object, but also includes the number of hours worked and a job title for its object. The EmployeeAccount constructor takes in a VacationAccount object, a double for the number of hours worked, and a string for the job title. The EmployeeAccount stores -1 for its HoursWorked if a negative number of hours is passed as a parameter to the constructor and stores “N/A” as job title when an empty string is passed as a parameter to the constructor. If only a VacationAccount object is passed in, we delegate object creation to our main constructor passing in number of hours worked as 0 and title as the empty string. We also have a default constructor in the case that no parameters are passed in, and in this case, we initialize the member variables of the EmployeeAccount object such that no functions will ever be performed on them by storing -1 for the employee ID.

The EmployeeAccount class has 2 public member methods. The addHoursToAccount method allows an employee to keep track of their hours. The maximum number of hours that can be added for each call to this function is 40 to resemble a standard work-week. For this call to do anything, we also must have a valid employee ID and valid job title. The second method is named syncVacationBalance. The purpose of this function is to transfer the vacation hours stored in an employee’s vacation account to their main employee account. After we have made the transfer, the balance in the vacation account gets set to 0, and the balance in the employee account will be updated. This way, we don’t have to change our BloodDonation and VacationAccount classes, so we can still increase vacation hours as we had done previously. But now, we have a main EmployeeAccount which has access to the VacationAccount (friend class) and will be the primary account that keeps tracking of employee information.

The AdminAccount Class has 3 public member methods. One is solely for setting an AdminID, which is a three-digit number. The second method addBonusVacationHours allows an admin to grant an employee extra vacation. An admin whose ID is the first three letters of the employee ID reflects an administrator over these employees. For example, an admin with ID 123 oversees all employees whose ID begins with 123, such as 123000, 123001, 123002, etc. If the employee has worked at least 80 hours, they are eligible for extra vacation. The third method givePromotion allows an admin to promote an employ, updating their job title and rewarding them with 6 extra vacation hours. Once again, an admin ID corresponding with the first three digits of such employee ID can only give this promotion, and an employee must have worked at lest 160 hours to be promoted. The AdminAccount class is a friend in the EmployeeAccount class, giving it access to its private data.

***Note:*** I have included getters for testing purposes, but their solely just to return private data, and I didn’t feel the need to regard/explain them in my report as new functionalities.

Test Cases

* Creating Class Instances that we will use:

BloodDonation doner1(889543, 65, 101.0000);

BloodDonation doner2(123456, 21, 280.0000);

VacationAccount accountV1(889543);

VacationAccount accountV2(123456);

VacationAccount accountV3(543);

EmployeeAccount accountE1(accountV1, 0, "Intern");

EmployeeAccount accountE2(accountV2, 10, "Manager");

EmployeeAccount accountE3(accountV3);

EmployeeAccount accountE4;

AdminAccount accountA1(889);

AdminAccount accountA2(123);

AdminAccount accountA3(678);

1. Manipulating Employee Account 1
   1. Using the employee account getters to access the object’s data after we’ve created its instance using a VacationAccount.
   2. Using a Blood Donation to add some vacation hours to our VacationAccount, then using our syncVacationAccount function to update the balance in our EmployeeAccount.
   3. Using the addHoursToAccount function to clock the amount of hours our employee worked and making sure the correct value is stored.
   4. Using our admin function addBonusVacationHours to give our Employee extra vacation since they have met the requirements.
   5. Using our admin function givePromotion to our employee, which updates their job title and gives them 6 extra hours of vacation.

// Employee Account 1

assert(accountE1.getID() == 889543);

assert(accountE1.getVacationBalance() == 0);

assert(accountE1.getHoursWorked() == 0);

assert(accountE1.getTitle() == "Intern");

assert(accountV1.addVacationToAccount( doner1 ) == true );

assert(accountE1.syncVacationBalance(accountV1) == true);

assert(accountE1.getVacationBalance() == 4);

assert(accountE1.addHoursToAccount(40) == true);

assert(accountE1.addHoursToAccount(40) == true);

assert(accountE1.addHoursToAccount(40) == true);

assert(accountE1.addHoursToAccount(40) == true);

assert(accountE1.getHoursWorked() == 160);

assert(accountA1.addBonusVacationHours(accountE1) == true);

assert(accountE1.getVacationBalance() == 6);

assert(accountA1.givePromotion(accountE1, "General Manager") == true);

assert(accountE1.getVacationBalance() == 12);

assert(accountE1.getTitle() == "General Manager");

1. Manipulating Employee Account 2
   1. Very similar start to manipulating employee account 1, but we initialize our EmployeeAccount object with a different VacationAccount object, and thus a different ID, and this time we say the employee has already worked 10 hours.
   2. We use another admin, who has a corresponding ID, to grant additional vacation.
   3. This time, we try using AdminAccount A2, whose ID does not correspond to the first three letters of EmployeeAccount E1, to give a promotion, but we return false as we should because this admin doesn’t have permission to give a promotion (maybe in real life they are an admin in a different department, hence different admin ID number).
   4. As a result, the EmployeeAccount E2’s title does not change, and they stay a manager.

// Employee Account 2

assert(accountE2.getID() == 123456);

assert(accountE2.getVacationBalance() == 0);

assert(accountE2.getHoursWorked() == 10);

assert(accountE2.getTitle() == "Manager");

assert(accountV2.addVacationToAccount( doner2 ) == true );

assert(accountE2.syncVacationBalance(accountV2) == true);

assert(accountE2.getVacationBalance() == 4);

assert(accountE2.addHoursToAccount(40) == true);

assert(accountE2.addHoursToAccount(40) == true);

assert(accountE2.addHoursToAccount(40) == true);

assert(accountE2.addHoursToAccount(40) == true);

assert(accountE2.getHoursWorked() == 170);

assert(accountA2.addBonusVacationHours(accountE2) == true);

assert(accountE2.getVacationBalance() == 6);

assert(accountA2.givePromotion(accountE1, "CEO") == false);

assert(accountE2.getVacationBalance() == 6);

assert(accountE2.getTitle() == "Manager");

1. Manipulating Employee Account 3
   1. This is our bad data employee account:
      1. We use a VacationAccount object with an invalid ID to create this EmployeeAccount object, and don’t pass in any parameters for hours worked or job title, demonstrating the use of our overloaded constructor and constructor delegation.
   2. Because our EmployeeAccount is filled with bad data (invalid ID and no job title), none of our EmployeeAccount public member methods will evaluate to true or update private data.
   3. For the same reason, none of our AdminAccount public member methods are going to change our EmployeeAccount at all.

// Employee Account 3

assert(accountE3.getID() == -1);

assert(accountE3.getVacationBalance() == 0);

assert(accountE3.getHoursWorked() == 0);

assert(accountE3.getTitle() == "N/A");

assert(accountV3.addVacationToAccount( doner1 ) == false );

assert(accountE3.syncVacationBalance(accountV1) == false);

assert(accountE3.getVacationBalance() == 0);

assert(accountE3.addHoursToAccount(40) == false);

assert(accountE3.getHoursWorked() == 0);

assert(accountA3.addBonusVacationHours(accountE3) == false);

assert(accountE3.getVacationBalance() == 0);

assert(accountA3.givePromotion(accountE3, "CEO") == false);

assert(accountE3.getVacationBalance() == 0);

assert(accountE3.getTitle() == "N/A");