Object Oriented Programming
Lab 05 Inheritance

Task 1

Create a class employee which has four methods: getHours (int), getSalary (double), getVacationDays (int), and getVacationForm (String). The general employee usually works for 40 hours, gets $40000 salary and 2 week (10 days) paid vacation per year unless specified otherwise. For vacation form the general employee usually gets the yellow form. (Hint: Return the specified value in getters, no need for attributes and setters)

Task 1:

Create a Lawyer class which is a sub class of Employee. The lawyer gets one more week paid vacation than a general employee per year. The lawyer also gets pink vacation form instead of yellow. This class also has an additional method sue which prints “I will see you in court.”

Task 2:

Create a class for a secretary which is a type of employee. Secretary has an additional method takeDictation which prints the dictation text.

Task 3:

Create a class LegalSecretary which is a type of secretary. The legal secretary gets $5000 more salary in year than all other secretaries. Moreover, it has an additional method named fileLegalBriefs which prints “I could file all day!”

Task 4:

Write an employee class Marketer to accompany the other employees. Marketers make $50,000 ($10,000 more than general employees), and they have an additional method named advertise that prints "Act now, while supplies last!"

Task 5:

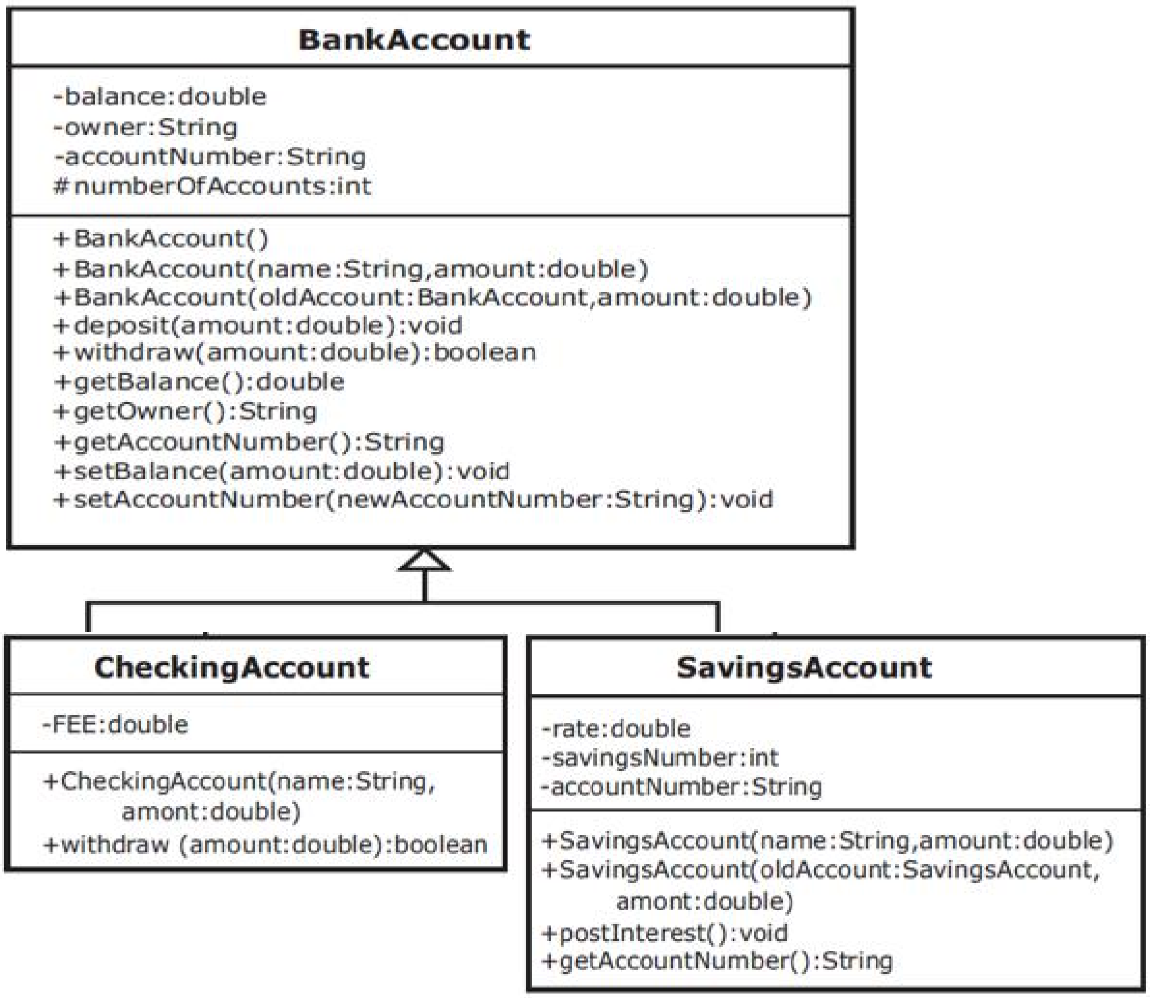
Write an employee class Janitor to accompany the other employees. Janitors work twice as many hours (80 hours/week), they make $30,000 ($10,000 less than others), they get half as much vacation (only 5 days), and they have an additional method named clean that prints "Workin' for the man."

Task 6:

Create main class and test your code

Note: Use the `super` keyword to interact with the `Employee` superclass as appropriate.

Task 2


BankAccount
-balance:double
-owner:String
-accountNumber:String
#numberOfAccounts:int
+BankAccount()
+BankAccount(name:String,amount:double)
+BankAccount(oldAccount:BankAccount,amount:double)
+deposit(amount:double):void
+withdraw(amount:double):boolean
+getBalance():double
+getOwner():String
+getAccountNumber():String
+setBalance(amount:double):void
+setAccountNumber(newAccountNumber:String):void
CheckingAccount
-FEE:double
+CheckingAccount(name:String, amont:double)
+withdraw (amount:double):boolean
SavingsAccount
-rate:double
-savingsNumber:int
-accountNumber:String
+SavingsAccount(name:String,amount:double)
+SavingsAccount(oldAccount:SavingsAccount, amont:double)
+postInterest():void
+getAccountNumber():String

Task 1: Create a BankAccount Class

- numberOfAccounts is a class variable with initial value of 100001.
- In no argument constructor set the balance to 0 , account number to the current number of Accounts (Hint: accountNumber = numberOfAccounts + "";) and increment the number of accounts.
- In second constructor set the owner and balance from given parameter, account number to the current number of Accounts and increment the number of accounts.
- In third constructor, the owner and the account number will be same as old account and set the balance.
- For withdraw if the user have sufficient balance to withdraw the specified amount, withdraw the amount from balance and return true otherwise return false.
- Implement all the other methods of BankAccount class specified in UML Diagram.

Task 2: Extending BankAccount

- Create a new class called CheckingAccount that extends BankAccount.
- It should contain a static constant FEE that represents the cost of clearing one check. Set it equal to 15 cents.
- Write a constructor that takes a name and an initial amount as parameters. It should call the constructor for the superclass. It should initialize accountNumber to be the current value in accountNumber concatenated with –10 (All checking accounts at this bank are identified by the extension –10). There can be only one checking account for each account number. Remember since accountNumber is a private member in BankAccount, it must be changed through a mutator method (setters).
- Write a new instance method, withdraw, that overrides the withdraw method in the superclass. This method should take the amount to withdraw, add to it the fee for check clearing, and call the withdraw method from the superclass. Remember that to override the method, it must have the same method heading. Notice that the withdraw method from the superclass returns true or false depending if it was able to complete the withdrawal or not. The method that overrides it must also return the same true or false that was returned from the call to the withdraw method from the superclass.

Task 3: Creating a Second Subclass

- Create a new class called SavingsAccount that extends BankAccount.
- It should contain an instance variable called rate that represents the annual interest rate. Set it equal to 2.5%.
- It should also have an instance variable called savingsNumber, initialized to 0. In this bank, you have one account number, but can have several savings accounts with that same number. Each individual savings account is identified by the number following a dash. For example, 100001-0 is the first savings account you open, 100001-1 would be another savings account that is still part of your same account. This is so that you can keep some funds separate from the others, like an Eid account.
- An instance variable called accountNumber that will hide the accountNumber from the superclass, should also be in this class.
- Write a constructor that takes a name and an initial balance as parameters and calls the constructor for the superclass. It should initialize accountNumber to be the current value in the superclass accountNumber (the hidden instance variable) concatenated with a hyphen and then the savingsNumber.
- Write a method called postInterest that has no parameters and returns no value. This method will calculate one month's worth of interest on the balance and deposit it into the account.
- Write a method that overrides the getAccountNumber method in the superclass.

Task 4: Create main class and test your code