**bankAccountClass**: Cannot be instantiated. All accounts must be a sub-class, but having one core class that we can point to for anything makes dealing with the accounts much easier.

Public members:

* A function to retrieve the current balance in the account
* Function to retrieve transactions, either by reference number, date range, etc.
* Function to retrieve the interest rate
* Function to retrieve owner’s info
* Function to retrieve account number
* Virtual:
  + Deposit function - makes a transaction report, stores it in the array, and updates the account’s balance.
  + Withdrawal function - verifies the account has the proper funds (error back if it doesn’t), make a transaction report, store it in the array, and update the account’s balance

Private:

* A value to store the account balance. Probably something like double, maybe long double, but that is unlikely.
* Some form of array to store all transactions to the account –likely a separate data type holding dates, verification number, info from the charger, amount charged/credited, etc. –likely pointers, so that it can point at the base class and be able to have derived classes that handle different transactions types, such as checks vs debit card vs direct account transactions
* Function to add/modify transactions
* Value storing the interest rate –by default: 0. Likely to be static per-class, to make it easier to change for everybody, because nobody is special enough to get a custom rate.
* Account number
* Account owner’s info – a pointer to a person data type to eliminate duplicate data & general memory hogging

**CertificateOfDeposit:** public class inherited from **bankAccountClass**. Not much to it: very high interest rate, but withdrawals and deposits are not allowed. When the time is up on the CD, you are able to withdraw your money.

Public:

* Deposit: spits back an error, seeing as the deposit would’ve been made at account creation, and is not modifiable past that time
* Withdrawal: checks if the account has matured, if not: spit back an error, if it has: allow the withdrawal, as long as the withdrawal is <= the account balance.

Private:

* Interest rate – set to a higher value during account creation. However: does not exist inside of this class, simply set the one in **bankAccountClass.**
* Account creation date
* CD maturity time in months. (alternatively: use a single date data type to store the date the CD will mature)

**savingsAccount**: public class inherited from **bankAccountClass**. Almost identical to a standard bank account, but allows a (low) interest rate.

Public:

* No changes. Everything inherited from the base class should be fine. Updating the balance based on the interest rate should be handled outside of the classes

Private:

* No changes, except the interest rate inside **bankAccountClass** should be changed.

**highInterestSavingsAccount**: public class inherited from **savingsAccount**. Requires a minimum balance, but gives a higher interest rate.

Public:

* Withdrawal: check that the balance will stay above minimum balance, if not: follow protocol: either demote their account to a **savingsAccount,** or deny the transaction (most likely the former)

Private:

* The interest rate should be changed to a higher value.
* A value holding the minimum balance for this account

**checkingAccount:** derived from **bankAccountClass,** unable to be instantiated**.** A standard bank account with the ability to write checks. (Of course, who uses checks nowadays?)

Public:

* Virtual:
  + writeCheck: function that processes checks, as they are handled slightly differently than direct transactions. They are still added to the transaction list, just with slightly different info. Basic form would be: verify the account has enough money in it, if it does, make a transaction report and add it to the array, deduct the amount and transfer it to the other account. If it doesn’t: apply a bounce fee and deny the transaction (or let the transaction go through, and apply an overdraft fee, depending on the customer’s preference)

**serviceChargeChecking:** derived from **checkingAccount**, gains no interest, has a limited number of checks, a monthly service charge, and no minimum balance.

Public:

* writeCheck: verifies the customer has not written their limit of checks this month, if they have, deny the transaction, if they have not: *verify the account has enough money in it, if it does, make a transaction report and add it to the array, deduct the amount and transfer it to the other account. If it doesn’t: apply a bounce fee and deny the transaction (or let the transaction go through, and apply an overdraft fee, depending on the customer’s preference),* increment the check counter

Private:

* checkCounter: a value used to hold the amount of checks this customer has written on this account this month
* maxChecks: static value holding the maximum number this person may write in a given month
* serviceCharge: static value holding how much to charge the customer per month. Processed outside of the class level

**noServiceChargeChecking:** derived from **checkingAccount**, gains interest, has an unlimited number of checks, no monthly service charge, and a minimum balance.

Public:

* writeCheck: function exactly the same, but add a check at the end that checks if the account balance is below the minimum amount, and if it is, demote the account to a **serviceChargeChecking**

Private:

* minBalance: a static value holding the amount this type of account must have in it at all times
* set an interest rate for this type of account

**highInterestChecking:** derived from **noServiceChargeChecking.** Exactly the same except the minimum balance is higher, and the interest rate is higher

Private:

* minBalance: higher amount than **noServiceChargeChecking**, still static
* Interest rate is increased.

Final notes: these are just the classes inside of the larger program. The program would deal with calculating the interest gained, service charges, etc. This document also only contains descriptions for the bank accounts; there would be more classes than this in a final product, describing checks, debit cards, customer info, etc… The final program would be able to use a pointer to **bankAccountClass** no matter which type of account it was, allowing ease in handling all the different types. Both **bankAccountClass** and **checkingAccount** should not be directly instantiated, as they are abstract classes, and not valid account types.