CMPS 356 Enterprise Application Development - Spring 2020
Lab 5 – OOP using JavaScript

Objective

The objective of this lab is to practice the following JavaScript features.

- ✓ **Object Oriented Programming**
- ✓ **Object literals**: comma-separated list of name-value pairs and associated functions wrapped in curly braces.
- ✓ **Classes**: create classes and use them to instantiate objects.
  - Methods , Constructors
  - Method Chaining
  - Inheritance
- ✓ **Module**s: export and require modules.

This Lab has two parts:

- ❖ **PART A**: Banking App (duration: 1h20mins).
- ❖ **PART B:** Develop a Book Donation App (duration: 1h30mins).

PART A – Banking App

In this exercise you will build a simple banking system according the design shown in Figure 1.

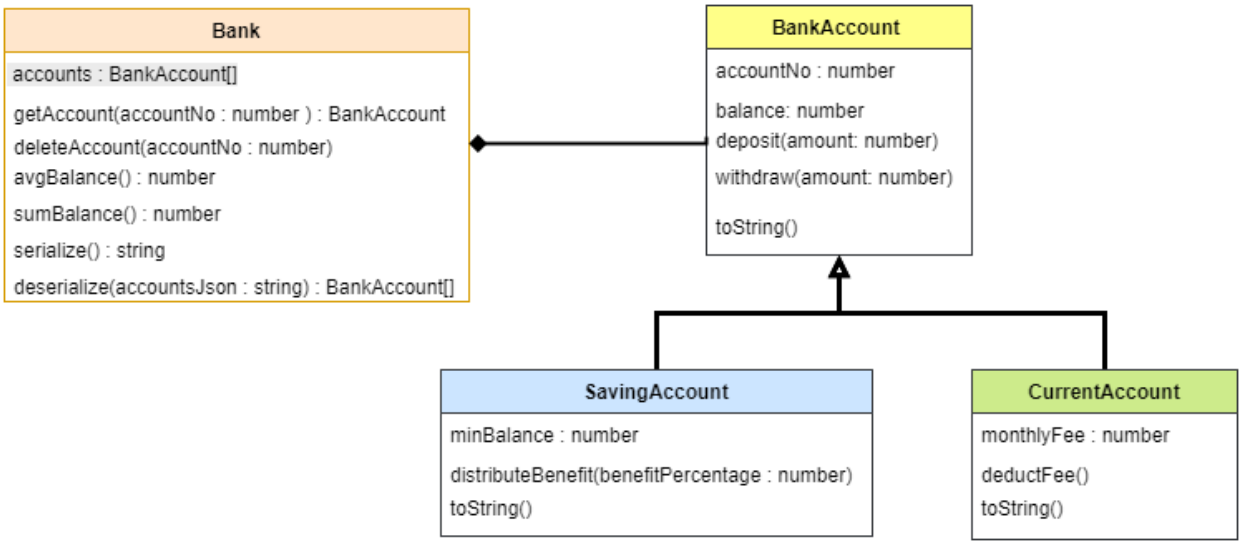

Figure 1. Banking System Class Diagram

1) Create **BankAccount** class with the following properties: accountNo, balance. This class should have a constructor to initialize these 2 properties. This class should have these methods:
   - deposit(amount): this method adds the amount to the balance
   - withdraw(amount): this method subtracts the amount from the balance
   - toString(): this method return Account # **accountNo** has QR **balance**. e.g., Account #123 has QR1000.

Export the **BankAccount** class module as an object.

2) Create app.js program. Declare **accounts** variable array and initialize it with the following accounts:

accountNo	**balance**
123	1000
234	4000
345	3500

Display the content of the **accounts** array.

3) Create **SavingAccount** class that extends BankAccount with an extra property: minBalance and an extra method distributeBenefit(benefitPercentage). This method computes the monthly benefit using the balance += (balance * benefitPercentage). The constructor should extend BankAccount to initialize the minBalance. Also, extend the toString() to indicate that this is a Saving Account. e.g., e.g., **Saving** Account #123 has QR1000.

Test savingAccount in app.js using the same table above and use a minimum balance of 500 for all accounts.

4) Create **CurrentAccount** class that extends BankAccount with an extra property: monthlyFee and an extra method deductFee(). This method subtracts the monthlyFee from the account balance only if the current balance is less than the minimum balance. The constructor should extend BankAccount to initialize the monthlyFee. Also, extend the toString() to indicate that this is a Current Account. e.g., e.g., **Current** Account #123 has QR1000.

   Test currentAccount in app.js using the same table above and use a monthly fee of 15 for all accounts.

5) Create **Bank** class to manage accounts. It should have a property **accounts** to store the accounts. Also, it should have the following methods:

Method	**Functionality**
add(account)	Add account (either Saving or Current) to accounts array.
getAccount(accountNo)	Return an account by Id
deleteAccount(accountNo)	Delete an account by Id
avgBalance()	Get the average balance for all accounts

sumBalance()	Get the sum balance for all accounts
serialize()	Return accounts as a JSON string
deserialize(accountsJson)	Takes JSON string representing accounts and returns an array of accounts.

6) Create app.js program. Declare an instance of Bank class then add the following accounts:

accountNo	**balance**	**type**	**minimumBalance**	**monthlyFee**
123	500	Saving	1000	
234	4000	Current		15
345	35000	Current		25
456	60000	Saving	1000	

- Test all the bank methods described above.
- Display the total balance of all accounts.
- Go through all the **Current** accounts and charge the monthly fee
- Display the total balance of all accounts after charging the monthly fee.
- Go through all the **Saving** accounts and distribute the benefit using a 5% benefit.
- Display the total balance of all accounts after distributing the benefits. After you complete the lab, fill in the ***Lab5-TestingDoc-Grading-Sheet.docx*** and save it inside ***Lab5 – OOP using JavaScript*** folder. Sync your repository to push your work to GitHub.