CMPS 350 Web Development Fundamentals
Lab 6 - Object Oriented Programming and Unit Testing with JavaScript

Objective

This laboratory aims to strengthen your skills in Object Oriented Programming in JavaScript. You will practice working with the following features:

- **Object literals**, which consist of a series of name-value pairs and associated functions enclosed in curly braces.
- **Classes**, which involve creating classes and using them to create objects.
- **Inheritance**, which is the ability of a class to inherit properties and methods from a parent class.
- **Modules**, which allow for the exporting and importing of code between files.
- **Unit Testing**, which involves writing code to test the functionality of your program.

This Lab is divided into three parts:

- **PART A**: Banking App using Object Literals
- **PART B**: Banking App using Classes
- **PART C:** Unit Testing

PART A – Banking App using Object Literals

In this exercise, you will build a simple banking app to manage accounts, including performing various operations, such as depositing/withdrawing an account, adding, deleting, and retrieving accounts.

1. Create a subfolder named **`lab6-js-oop`** on your repository to organize the code produced for this lab.
2. Create a new folder named **`banking-app`** under **`lab6-js-oop`** and open it in VS Code.
3. Add `bank.js` file. Declare an `accounts` array to store the following accounts:

accountNo	balance	type
123	500	Saving
234	4000	Current
345	35000	Current
456	60000	Saving

4. Implement and export the following functions in `bank.js`:

Functions	Functionality
deposit(accountNo, amount)	adds the amount to the account balance.
withdraw(accountNo, amount)	subtracts the amount from the account balance.
add(account)	Add either a Saving or Current account to the accounts array.

getAccount(accountNo)	Return an account by account No
deleteAccount(accountNo)	Delete an account by account No
avgBalance()	Get the average balance for all accounts
sumBalance()	Get the sum balance for all accounts
distributeBenefit(benefitPercentage)	Loops through savings accounts and updates the balance to: balance += (balance * benefitPercentage).
deductFee(monthlyFee).	Loops through current accounts and updates the balance to: balance -= monthlyFee.
toJson()	Return accounts as a JSON string
fromJson(accountJSON)	Return accounts array

5. Create app.js to test bank.js functions. Create a couple of new savings/current accounts. Then make couple of deposit and withdrawal operations. Examples:
   Balances **after depositing** 500 QR to the first two accounts
   - Saving Account #123 has QR1000.
   - Current Account #234 has QR4500.

   Balance after **withdrawing 1000 QR from** the last two accounts
   - Current Account #345 has QR25000.
   - Saving Account #456 has QR59000.
6. Charge a monthly fee of 10QR to current accounts.
7. Display the total balance of all accounts after charging the monthly fee.
8. Distribute 6% benefit to all **Saving** accounts.
9. Display the total balance of all accounts after distributing the benefits.
10. Display the list of accounts in json format

PART B – Banking App Using Classes

In this exercise, you will create a new version of banking app using classes. The app will consist of several classes, each with its own set of properties and methods to represent different types of bank accounts, such as saving accounts and current accounts. You will also create a Bank class that will manage all the accounts and perform various operations, such as adding, deleting, and retrieving accounts, as shown in Figure 1.

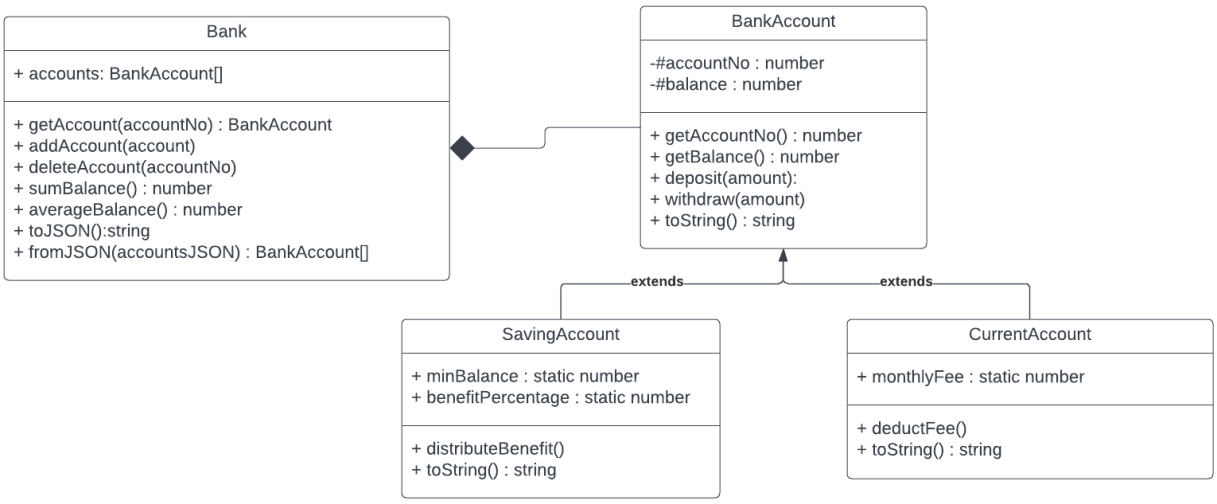


Figure 1. Banking App Class Diagram

1) Create a new folder named **`banking-app-cls`** under **`lab6-js-oop`** and open it in VS Code.
2) Create a package.json file using **npm init inside the BankingApp**. This file is used to define dependencies by listing the npm packages used by the app.
3) Add the following inside the package.json "type": "module," which will allow you to use the import-export feature
4) Create a class called **BankAccount** with two private properties: **accountNo** and **balance**. The account number should be **randomly generated**, and the balance should be initialized through the class constructor. The class should also have the following methods:
   - get properties for both accountNo and balance.
   - deposit(amount): adds the amount to the balance.
   - withdraw(amount): subtracts the amount from the balance.
   - toString(): this method returns the message: Account # **accountNo** has QR **balance**. e.g., Account #123 has QR1000.

Export the **BankAccount** class so that it can be reused outside the BankAccount.js file.

5) Create app.js program to test the **BankAccount** class. Declare an array called accounts and initialize it with the following accounts:

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

Then display the content of the **accounts** array.

6) Create **SavingAccount** class that extends BankAccount with extra static properties: minBalance (set the value to 500) and benefitPercentage (set the value to 0.05). Also, add an extra method distributeBenefit(). This method computes the monthly benefit using the balance += (balance * SavingAccount.benefitPercentage). Additionally, 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 from above.

7) Create **CurrentAccount** class that extends BankAccount with an extra static property: monthlyFee (set the value to 10) and an extra method deductFee(). This method subtracts the monthlyFee from the account balance. 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.

8) Create **Bank** class to manage accounts. It should have **accounts** property 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 account No
deleteAccount(accountNo)	Delete an account by account No
avgBalance()	Get the average balance for all accounts
sumBalance()	Get the sum balance for all accounts
toJson()	Return accounts as a JSON string
fromJson(accountsJson)	Takes JSON string representing accounts and returns an array of accounts.

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

accountNo	**balance**	**type**
123	500	Saving
234	4000	Current
345	35000	Current
456	60000	Saving

- Test all the Bank methods described above.
- Display the total balance of all accounts.
- Increase by 5 the monthly fee of all the **Current** accounts then charge the monthly fee.
- Display the total balance of all accounts after charging the monthly fee.
- For all the **Saving** accounts distribute the benefit using a 6% benefit.
- Display the total balance of all accounts after distributing the benefits.

Part C – Unit Testing Using Mocha and Chai

Unit testing is an essential part of modern software development. It involves testing individual units of code to ensure that they function as expected and meet the application requirements. Unit tests help to catch bugs and errors early in the development process, reducing the likelihood of issues arising in production.

In this section, you will practice writing and running unit tests with **Mocha** and writing assertions using Chai.

1. Install mocha and chai using *node package manager* (npm):
   npm install mocha -d
   npm install chai -d
   This will add 2 dev dependencies to **package.json** file.
2. Add the following inside **package.json** file.

```
"scripts": {
  "test": "mocha **/*.spec.js"
}
```

3. Create a JavaScript file named **bank.spec.js**
4. Import an instance the *Bank* class to be tested and the *expect* function from chis.

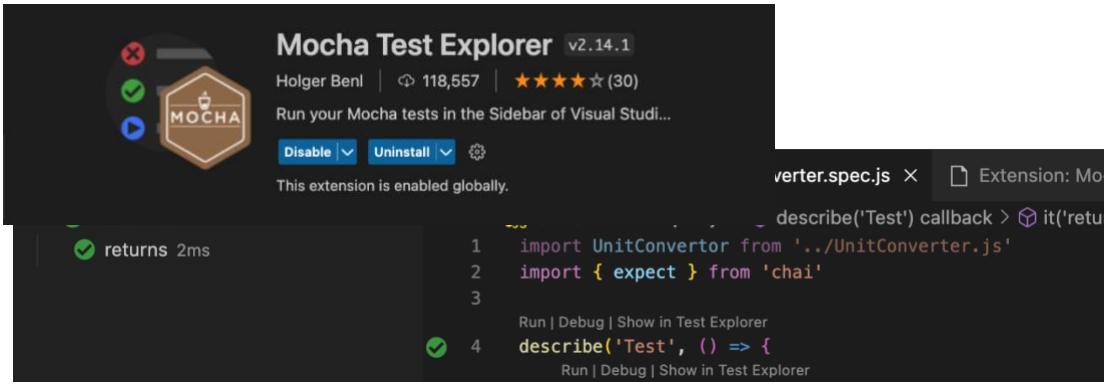
```
import bank from './Bank.js';
import {expect} from 'chai';
```

5. Write 2 unit tests for each method of **Bank.js** class.

Tips

- Use **expect** to make assertions about the output of the functions.
- Use **describe** and **it** functions to structure your tests.
- Use console.log statements to debug your tests if necessary.

6. Run the unit tests from the command line using: **npm test**
7. You can also install and use Mocha Test Explorer extension and run the tests.



Note : After you complete the Lab, push your work to your GitHub repository.