CMPS 350 Web Development Fundamentals
Lab 6 – OOP using JavaScript and Unit Testing

Objective

The objective of this lab is to practice the following JavaScript Object Oriented Programming features including:

- **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.
- **Inheritance**
- **Modules**: export and import modules.
- **Unit Testing**

This Lab has three parts:

- **PART A**: Banking App (duration: 1h20mins).
- **PART B:** Unit Testing

PART A – Banking App

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

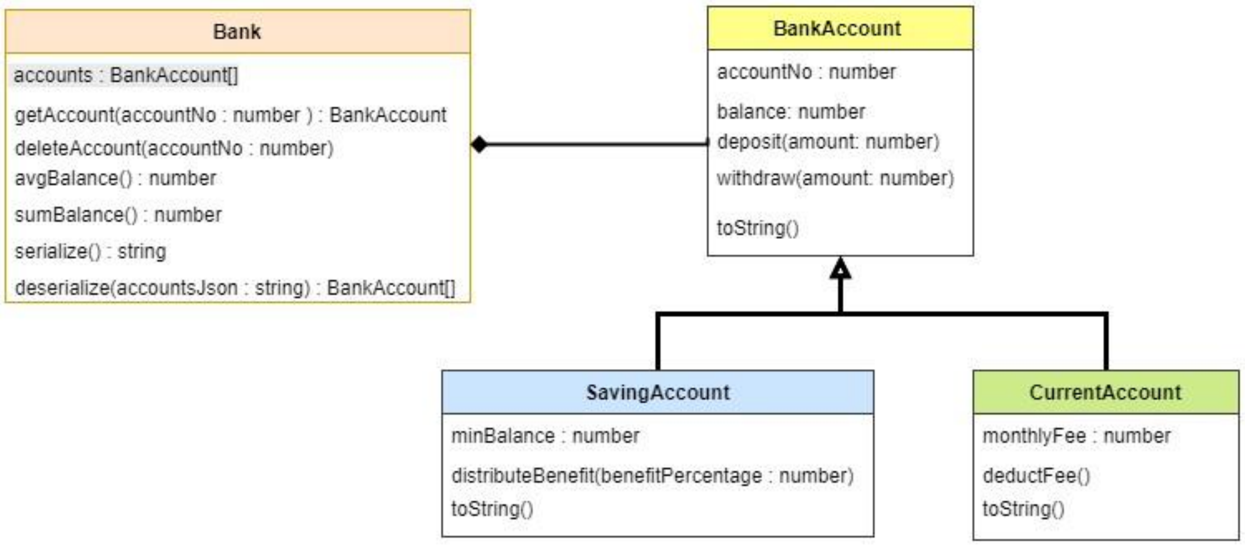


Figure 1. Banking App Class Diagram

1) Create **`BankAccount`** class with the following private properties: `accountNo` and `balance`. The account no, should be randomly generated. But the balance must be initialized through the constructor of the class. In addition, the class should have the following methods:

- getters for both accountNo and balance
- `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 as a module.

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. 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 10 for all accounts.

5) 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.

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		10
345	35000	Current		15
456	60000	Saving	1000	

- 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 5% benefit.
- Display the total balance of all accounts after distributing the benefits.

Part B – Unit Testing Using Mocha and Chai

1. Sync cmps350-lab repo to get the Lab files.
2. Copy ***Lab6-JS OOP*** folder from cmps350-lab repo to your repository.
3. Open ***Lab6- JS OOP \UnitConverter.js*** in Webstorm. You should see a JavaScript file named *UnitConverter.js*. In this exercise, you will create a spec file to unit test the function of the *UnitConverter* class.
4. First, create the package.json file using **npm init**. This file is used to define dependencies by listing the npm packages used by the app.

   Refresh your project to see the **package.json** file.
5. 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.

6. Create a JavaScript file named **UnitConverter.spec.js**
7. Import an instance the *UnitConverter* class to be tested and the *chai expect* package.

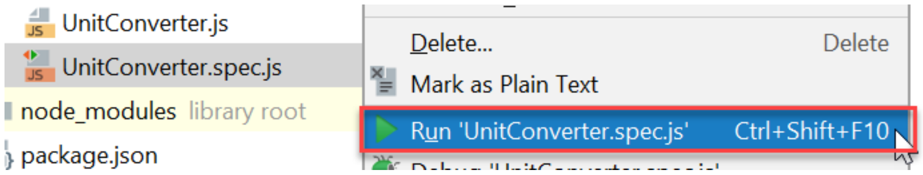
```
import unitConverter from './UnitConverter.js';
import {expect} from 'chai';
```

8. Write 2 test cases for each method of **UnitConverter** class.

You may start with the following inputs and expected results. Then use search for "google unit converter" to compute the expected results for more input values.

Method	Input	Expected Result
kgToOunce	1	35.274
kgToPound	2	4.4092
meterToInch	1	39.3701
meterToFoot	2	6.5617

9. Run the unit tests as you develop them using WebStorm:



Also run the unit tests from the command line using: **npm test**

But first, you should have the following in package.json file.

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

After you complete the lab, push your work to your GitHub repository.