CMPS 356 Enterprise Application Development - Spring 2018
Lab 5 – OOP using JavaScript

Objective

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

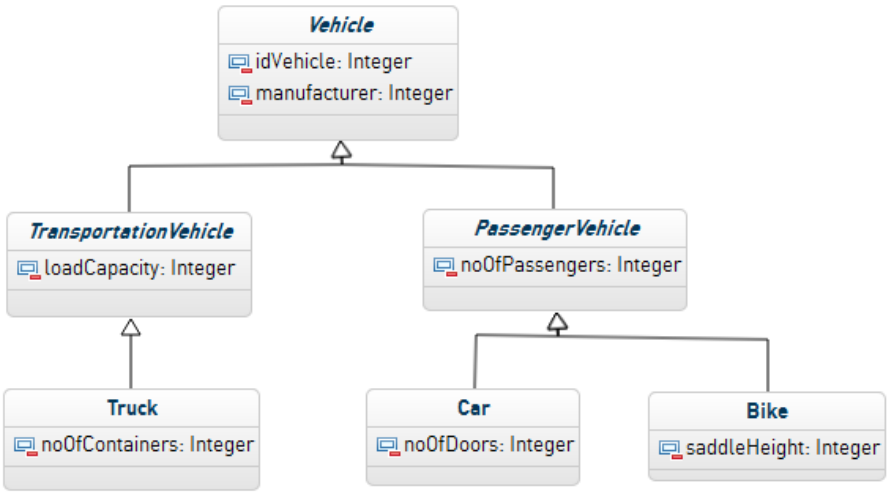
- ✓ **Object Oriented Programming**
- ✓ **Object literals**: comma-separated list of name-value pairs and associated functions wrapped in curly braces.
- ✓ **Classes**: create classes and use them to instantiate objects.
- ✓ **Module**s: export and require modules.

This Lab has two parts:

- ❖ **PART A**: Tutorial
- ❖ **PART B**: Develop a Banking App
- ❖ **PART B:** Develop a Book Donation App

❖ PART A – Tutorial

1. Create a project named **Vehicles** inside your **Lab5-JS OO** folder
2. In the Vehicles project create a folder called **models**
3. Inside the **models** folder create the following classes with their methods and inheritance
4. Export all the classes.



5. Create folder called **repos** that will contain the **VehicleRepo.js file**
6. Create the **VehicleRepo** class inside the **VehicleRepo.js** file .
7. Create the following methods inside the **VehicleRepo** class
    a. addVehicle
    b. deleteVehicle given vehicle ID
    c. displayAllVehicles
8. Create a file called **app.js .**
9. Inside the **app.js** create an array called vehicles of type Vehicle.
10. Add 3 new car objects and 3 truck object into the vehicles array
11. Use the table below to populate the initial data

Vehicle

idvehicle	manufacturer	vehicle_type
9	honda	Bike
10	lamborghini	Car
11	volvo	Truck

Car

idvehicle	noOfPassengers	noOfDoors
10	2	2

Truck

idvehicle	loadCapacity	noOfContainers
11	1000	2

12. Delete the vehicle with id 10
13. Update the name of the vehicle id 11 to Honda
14. Filter the all the vehicles produced by Volvo
15. Filter all the car instances from the vehicles object

Assignment # 4

Deadline 11:59 the day before the Lab

PART B – Banking App

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

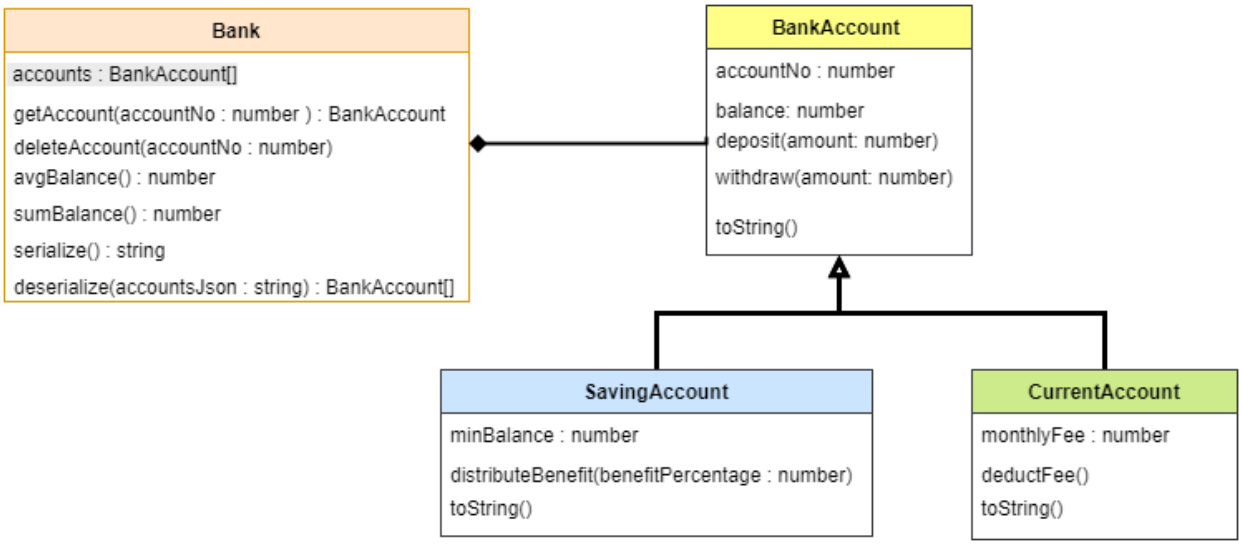


Figure 1. Banking System Class Diagram

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.

PART C – Book Donation App

You are required to develop a book donation App that allows people to donate their unused books to people who needs it.

The book donation system should have the following classes

a) **Book** – bookId, title, author, imageUrl, donor, status

Donor property is an object having of the details of the person who donated the book.

Note that the **status** attribute can have one of the following values:

- **Pending** : As soon as the user adds the book s/he want to donate to the system, the status is set to pending

- **Available** : When the book donor delivered the book to the store then status of the book becomes available

b) **Donor** - donorId, firstname , lastname , phoneNumber , street, city, email, password

c) **BookCatalog** – contains list of books and the following methods

addBook	Input: book object. Adds a book to the list of books. All books are pending when they are first added.
updateBook	Input: book. Updates the book that having matching bookId.
deleteBook	Input: bookId. Deletes the book from the list of books.
getBooks	Returns all the books that are not pending.
getDonorBooks	Input donorId. Returns all the books donated by a particular donor.
getTopXDonors	Input: donorsCount. Returns the top book donors. Eg. If the user passes 3 as donorsCount parameter then this function returns the top 3 donors and the list of books each one donated.

➔ You do not need to create **Donor** and **Book** classes. Just create the objects directly. And test your code by creating **app.js file** to instantiate the **BookCatalog** class and test its methods.

Note that you should make use of the **JavaScript features and capabilities** such as **arrow functions**, array functions **(.map, .reduce, .filter, .splice, .sort…), spread operator**, **object literals**, **classes**. Marks will be deducted for using traditional programming styles such as searching an array using **loops**.

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.