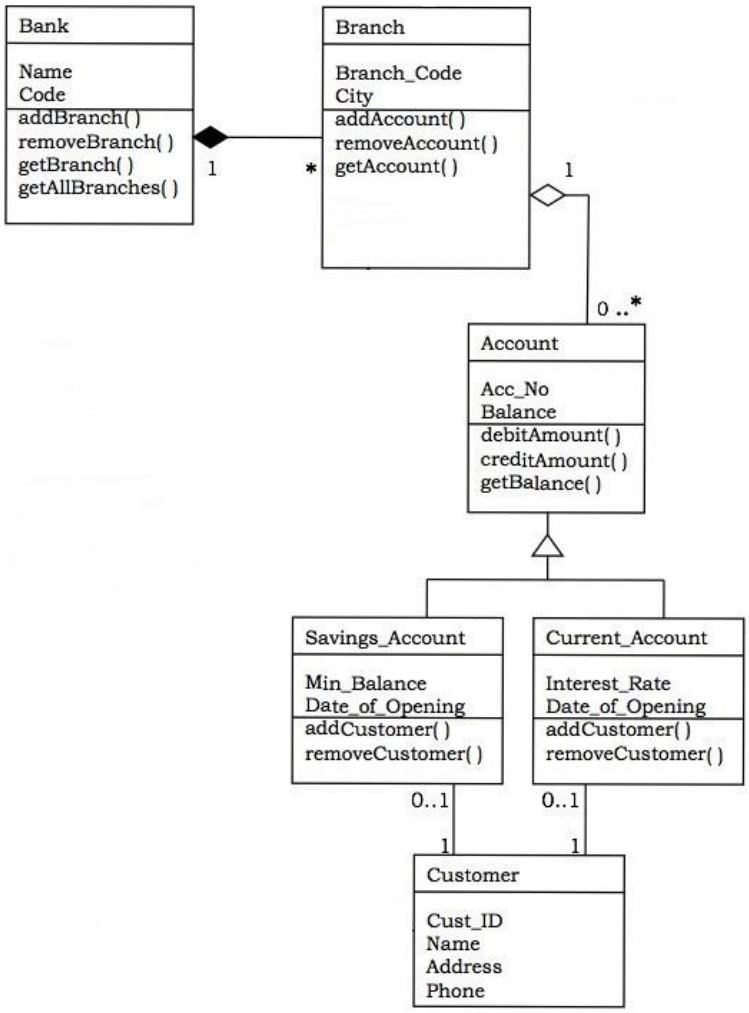
## Offline 2 on Python, NumPy and Matplotlib

1. **UML Class Diagram** – Unified Modeling Language (UML) is a graphical language for modeling the structure and behavior of object-oriented systems. UML is widely used in industry to design, develop and document complex software.
   UML Class diagram describes the internal structure of classes and relationships between the classes. In this assignment, you will implement the following UML class diagram using Python OOP.




**Bank**

| Attributes | Methods |
|---|---|
| **name** – a string that represents the name of the bank | **addBranch(self, branch_obj)** – will include the **branch_obj** Branch object to the **branch_list** |
| **code** – an integer representing the code number of the bank | **removeBranch(self, branch_code)** – will search for the branch having code number **branch_code** and remove that branch object from the **branch_list** |
| **branch_list** – a list that will contain all the Branch objects opened by the bank | **getBranch(self, branch_code)** – will search for the branch having code number **branch_code** and return that branch object from the **branch_list.** Return **None** if not found. |
| | **getAllBranches(self)** – will print all the branch codes and branch city names from the **branch_list** |
| | **updateInfo(self, newname, newcode)** – will update the name and code number of the bank |
| | **__init__(self, name, code)** – constructor for this class. Initialize **branch_list** to an empty list. |

**Branch**

| Attributes | Methods |
|---|---|
| **branch_code** – an integer representing the code number of the branch | **addAccount(self, account_obj)** – will include the **account_obj** Account object to the **account_list** |
| **city** – a string representing the city name of the branch | **removeAccount(self, account_number)** – will remove the account from the **account_list** |
| **bank** – a Bank object representing the owner bank of the branch | **getAccount(self, account_number)** – will search for the account associated with **account_number** from the **account_list** and return the Account object otherwise return **None**. |
| **account_list** - a list that will contain all the Account (both Savings and Current) objects created from the branch | **updateInfo(self, branch_code, city)** – will update the branch_code and city of the branch |
| | **__init__(self, branch_code, city, bank)** – constructor for this class. Initialize the **account_list** to an empty list. |

**Account**

| Attributes | Methods |
|---|---|
| **acc_no** – a string representing the account number | **debitAmount(self, amt)** – will deduct the **amt** amount from the **balance** value. Override this method within the **Savings_Account** and **Current_Account** classes. For savings account always maintain the **min_balance** value and for current account always ensure the balance is minimum **0** |
| **balance** – a floating point number that represents the balance of the account | **creditAmount(self, amt)** – will increase the **balance** amount by the amount **amt** |
| **branch** – a Branch object representing the owner branch of the account | **getBalance(self)** – will return the current balance of the account |
| | **__init__(self, acc_no, branch_obj)** – constructor for the class. Initialize the **balance** to zero |

**Savings_Account** – this class will inherit the **Account** class.

| Attributes | Methods |
|---|---|
| **min_balance** – a floating point number representing the minimum balance of the account | **setCustomer(self, customer_obj)** – will set the **customer** value to the **customer_obj** object |
| **open_date** – a python date that represents the account opening date | **removeCustomer(self)** – will remove the customer from the **customer** variable i.e. set None |
| **customer** – a Customer object that represents the owner user of the account | **__init__(self, acc_no, branch, min_balance)** – constructor for the class. Pass the acc_no, and branch to the parent Account class. Automatically set the open_date to current date and set the customer to None |

**Current_Account** – this class will inherit the **Account** class

| Attributes | Methods |
|---|---|
| **interest_rate** – a floating point number representing the interest rate of the account | **setCustomer(self, customer_obj)** – will set the customer to the customer_obj object |
| **open_date** – a python date that represents the account opening date | **removeCustomer(self)** – will remove the customer from the customer variable i.e. set None |

| | |
|---|---|
| **customer** – a Customer object that represents the owner of the account | **__init__(self, acc_no, branch, interest_rate)** – constructor for the class. Pass the acc_no, and branch to the parent Account class. Automatically set the open_date to current date and set the customer to None. |

**Customer –** maintain a class variable named next_id (initialize to 1 and increment by 1) and set the customer id instance variable value to this class variable value.

| Attributes | Methods |
|---|---|
| **id** – an integer representing the customer id | **setSavingsAcc(self, savings_acc)** – will set the savings_acc value |
| **name** – a string that represents the name of the customer | **getSavingsAcc(self)** – will return the savings_acc object |
| **address** – a string containing the address of the customer | **setCurrentAcc(self, current_acc)** – will set the current_acc value |
| **phone** – a string representing the customer contact number | **getCurrentAcc(self)** – will return the current_acc object |
| **savings_acc** – a Savings Account object this customer owns or None | **__init__(self, name, address, phone)** – constructor for the class. Set the savings_acc and current-acc value to None. Also set the id value to the class variable next_id value. |
| **current_acc** – a Current Account object this customer owns or None | |

**Final Tasks:**

i. Create two instances of Bank class, b1=Bank("DBBL", 1256) and b2=("EBL", 1257). Initialize the **branch_list** as an empty list for both of the banks.

ii. Create 4 Branch objects, br1=Branch(1, 'Dhanmondi', b1), br2=Branch(2, 'Motijheel', b1), br3=Branch(3, 'Mirpur', b2), br4=Branch(4,'Gulshan', b2). Initialize the **account_list** to an empty list.

Also include the branches br1, br2 to bank b1 and also include branches br3, br4 to bank b2.

iii. Call the getAllBranches() method for branch b1 and b2 and check the output.

iv. Create 2 savings account, s1=Savings_Account(1234, br1, 500) and s2=Savings_Account(1235, br4, 1000) and also create 2 current account, c1=Current_Account(5432, br1, 0.10) and c2=Current_Account(5431, br3, 0.12)

Also include these accounts to the corresponding branch account_list.

v. Create 2 customers, cs1=Customer("Afif", "Dhaka", "01911111111"), cs2=Customer("Sohan", "Dhaka", "01511111111").

Set accounts s1 and c1 for customer cs1 and also set accounts s2, c2 for customer cs2.

Also set the s1, c1 accounts customer value to cs1 and set the s2, c2 accounts value to cs2.

vi. Print the cs2 customer current account number, balance, interest rate, branch code and bank name.

2. Declare two numpy arrays A, B both of size (8, 10) with some random values.
    a. Matrix multiply $A^T$ with B and print the result
    b. List all the elements of A whose values are greater than 0.5
    c. Show the $5^{th}$ and $8^{th}$ column values of A
    d. Show all the elements between the $3^{rd}$ to $7^{th}$ columns and between the $2^{nd}$ to $5^{th}$ rows of B
    e. Find out the row wise summation of A and then calculate the mean of all the summed values.

3. Read the attached 'wine.data' file (separated by COMMA and each line ends with the newline) using numpy genfromtxt() function.

   Then scatter plot the column 3(y axis) vs column 2(x axis) and also scatter plot the column 4 (y axis) vs column 2 values using matplotlib scatter library. Use different marker and colors so that we can detect them individually.

4. Generate the following star pattern using python nested for loop.

**Input**

```
Input N: 5
```

**Output**

```
**********
****  ****
***    ***
**      **
*        *
*        *
**      **
***    ***
****  ****
**********
```

5. Implement the merge sort algorithm using python.
   **Input:** an unsorted list of integers
   **Output:** the sorted list of the input integers

[Miss at your own risk]
[Zero tolerance for plagiarism]