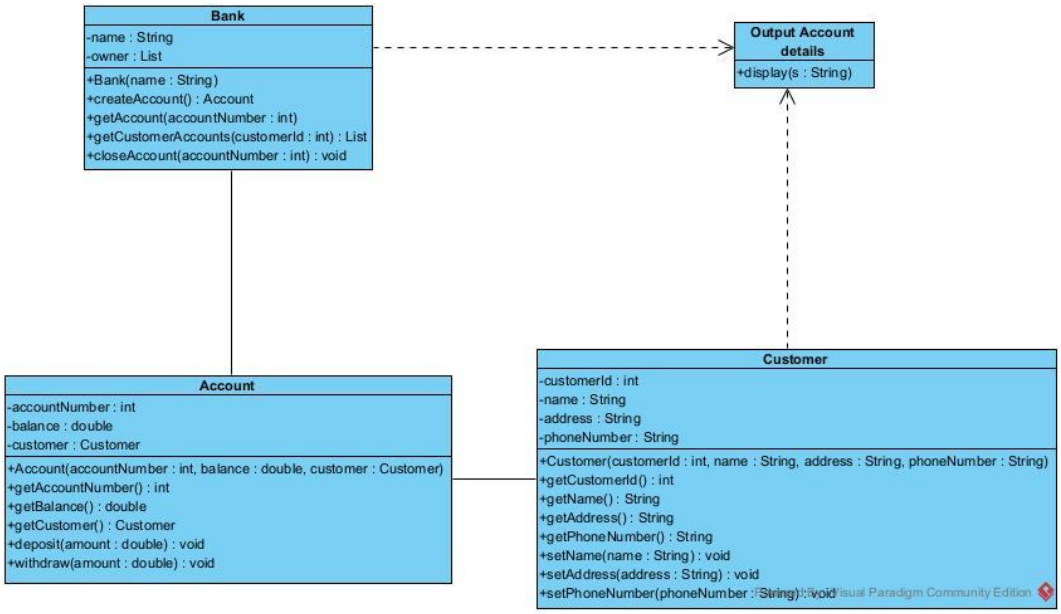
## 1. Experiment with further UML models, including a class diagram.

Booch (1998) defined a diagram as a graphical representation of elements, frequently depicted as a connected network of objects and interactions (vertices and arcs). Diagrams are projections of systems because they are created to let you see a system from many angles. A diagram is a simplified representation of a system's constituent parts. For instance, the nine similar diagrams included in the UML are the Class diagram; Object diagram; Use case diagram; Sequence diagram; Collaboration diagram; State chart diagram; Activity diagram; Component diagram; and Deployment diagram.

A class diagram illustrates classes, interfaces, collaborations, and connections. These diagrams are used to model object-oriented systems the most frequently. Class diagrams address the static design view of a system. Active class diagrams are used to address the static process perspective of a system.

This class diagram represents a straightforward banking system. The three main classes in the system are bank, account, and customer:



The **Bank** class in this class diagram illustrates a bank and features methods like **createAccount()**, **getAccount()**, **getCustomerAccounts()**, and **closeAccount()** in addition to characteristics like **name** and **accounts**. With fields like **accountNumber**, **balance**, and **customer**, as well as methods like **getAccountNumber()**, **getBalance()**, **getCustomer()**, **deposit()**, and **withdraw()**, the **Account** class represents a bank account. The **Customer** class is used to represent a bank customer and has methods like **getCustomerId()**, **getName()**, **getAddress()**, and **getPhoneNumber(),** as well as attributes like **customerId**, **name**, **address**, and **phoneNumber** (Nym, 2022).

## 2. Write a Python program which implements the UML class diagram.

```
class Bank:
    def __init__(self, name):
        self.name = name
        self.accounts = []

    def createAccount(self):
        account_number = len(self.accounts) + 1
        account = Account(account_number)
        self.accounts.append(account)
        return account

    def getAccount(self, account_number):
        for account in self.accounts:
            if account.getAccountNumber() == account_number:
                return account
        return None

    def getCustomerAccounts(self, customer_id):
        customer_accounts = []
        for account in self.accounts:
            if account.getCustomer().getCustomerId() == customer_id:
                customer_accounts.append(account)
        return customer_accounts

    def closeAccount(self, account_number):
        for account in self.accounts:
            if account.getAccountNumber() == account_number:
                self.accounts.remove(account)
                break
```

```
class Account:
    def __init__(self, account_number, balance=0):
        self.account_number = account_number
        self.balance = balance
        self.customer = None

    def getAccountNumber(self):
        return self.account_number

    def getBalance(self):
        return self.balance

    def getCustomer(self):
        return self.customer

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount


class Customer:
    def __init__(self, customer_id, name, address, phone_number):
        self.customer_id = customer_id
        self.name = name
        self.address = address
        self.phone_number = phone_number

    def getCustomerId(self):
        return self.customer_id

    def getName(self):
        return self.name

    def getAddress(self):
        return self.address

    def getPhoneNumber(self):
        return self.phone_number

    def setName(self, name):
        self.name = name

    def setAddress(self, address):
        self.address = address

    def setPhoneNumber(self, phone_number):
        self.phone_number = phone_number
```

(w3resource, 2023)

**References:**

Booch, G., Rumbaugh, J., Jacobson, I. and Wesley, A. (1998). *Unified Modeling Language User Guide, The*. [online] Available at: https://patologia.com.mx/informatica/uug.pdf [Accessed 6 Jun. 2023].

Nym (2022). *Class Diagram for Bank Management System*. [online] Itsourcecode.com. Available at: https://itsourcecode.com/uml/bank-management-system-class-diagram-uml/ [Accessed 6 Jun. 2023].

w3resource. (2023). *Python Class - Bank Account Management System*. [online] Available at: https://www.w3resource.com/python-exercises/class-exercises/python-class-real-life-problem-3.php [Accessed 10 Jun. 2023].