Coding

Banking System

Initially, the banking system does not contain any accounts, so implement operations to allow accounts creation as well as making deposits.

Operations:

CREATE_ACCOUNT <accountId>

Should create a new account with the given identifier if it doesn't already exist.
Returns "true" if an account was successfully created,
Returns "false" if an account with accountId already exists.

DEPOSIT <accountId> <amount>

Should deposit the given amount of money to the specified account accountId
Returns the total amount of money in the account after the query has been processed.
If the specified account doesn't exist, it should return -1.

TRANSFER <fromId> <toId> <amount>

This should transfer the given amount of money from account with fromId to account with toId.

Returns the balance of fromId if the transfer was successful, or -1 otherwise.
Returns -1 if fromId or toId doesn't exist.
Returns -1 if fromId and toId are the same.
Returns -1 if funds on the account fromId are insufficient to perform the transfer.

TOP_ACTIVITY <n>

This should return the identifiers of n most active accounts in descending order of financial activity indicator, In case of a tie, sorted alphabetically by accountId in ascending order.

https://interviewdb.vercel.app

Detect potential fraud

Object oriented approach, similar to Design Simple Bank Leetcode Question. I had to perform fraud detection by taking in 1 set of data, calculating the previous list of all means, and comparing the current transaction to see if it's greater than 2 times the previous average means. This question kept building in complexity, so first I had to parse the data into a systematic data structure, and then I had to calculate the list of previous averages. And then I had to use this average value and see if there was a fraud. The data looked like this:
(transaction ID, transaction type, transaction amount)
E.g.
(1323232, "CREDIT", 34.0)
(5465676, "DEBIT", 78.0)
(2186434, "CREDIT, 696.0) ← fraud
(transaction ID, transaction timestamp)
(1323232, 67567567)
(5465676, 67567997)
(2186434, 67568607)

Storage Interface

```
// implement a storage interface add, delete, copy file, capital one // Operation:
// ADD_FILE <fileName>
// DELETE_FILE <fileName>
// Array
// operations = [
//   ["ADD_FILE", "file_1"],
//   ["ADD_FILE", "file_1"],
//   ["ADD_FILE", "file_2"],
//   ["ADD_FILE", "dir_1/file_2"],
//   ["DELETE_FILE", "file_2"],
//   ["DELETE_FILE", "file_2"]
// ]
// COPY <fromFilePath> <toDirectory>
// Should copy the identified file from the fromFilePath to the toDirectory. Note that the original file remains in place
// Returns true if the operation was successful, false otherwise.
// Returns false if a file with same name already exists in toDirectory
// Returns false if toDirectory does not end with a slash (/)
```

https://interviewdb.vercel.app

```
// Example: These inputs are provided as an example of how the input data can be structured, you may select any of these formats or come up with your own one. Note: these operations below are in addition to those in Level 2
// Array
// operations = [
//   ...
//   ["COPY", "/dir_1/file_2", "/"],
//   ["COPY", "/file_2", "/"],
//   ["COPY", "/file_3", "/dir_1/"],
//   [“COPY”, "/file_3", "/dir_3"]
// ]
// true
// false
// false
// false
```

System Design

Credit Card Account management system

Design a credit card account management application. The application should represent a bank with the opportunity to apply for credit cards, manage credit cards, manage payments, and report usages.
Features that need to be supported:
- Process a credit card application
- Create a credit card online account / access online portal
- Provide API to authorize credit card transactions
- Support calculating daily, weekly and monthly statistics
- Pay credit card balance using a third-party payment service

Online banking application

Design an online banking application for use in Capital One Bank.

https://interviewdb.vercel.app

The application should present a Capital One customer with the opportunity to create accounts, deposit/withdraw/transfer money, exchange money, etc.
Features that need to be supported:
- Create an account and handle user authorization for the following types of accounts: 360 Checking, 360 Saving, Mortgage, and Auto Loan
- Deposit/withdraw money to/from the account
- Calculate daily, weekly, monthly statistics
- Transfer money to another Capital One account
- Exchange money (Zelle, money transfer)

Fraud detection system

Design a fraud detection system for large volumes of credit card transactions. The interviewer provides two conditions to detect fraud:
The transaction occurs at a location more than 100 miles away
The transaction amount is triple the average transaction amount from the past week

The interviewer will provide the table schema and ask you to start drawing the system design diagram.
The main challenges focus on:
Database selection
Data pipeline design

https://interviewdb.vercel.app

Case Study

Testing Budget

The interviewer will start with a general discussion of testing and talk about how at capital one, every team has their own testing cadence, e.g. biweekly, monthly, bimonthly, quarterly. Testing can be either simple or rigorous.

Part 1
There are 2 teams. Each team has their own yearly testing budget. Calculate and determine if their budget is enough based on their testing cadence.

Part 2
A team member wrote a script to calculate how to maximize the testing budget. The logic is to convert simple tests to rigorous tests when there's leftover money, but there's an issue in the output where some simple tests have negative numbers. They're asking for help debugging this issue.

```
    For the release cadence determines the total cost to run the minimum required tests.
    If there is surplus budget the surplus is reallocated to determine the new number of
    simple and rigorous tests that can be performed, prints this out, and returns the amount
    of budget remaining after reallocation.
    budget  The budget to allocate surplus The surplus budget after full allocation.  Will be >= 0 and <= min plan cost.
    """
    def compute_test_needs(self, budget):
        simple_needed   = self.simple_tests_needed()
        rigorous_needed = self.rigorous_tests_needed()
        cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
        surplus = budget - cost
        if (surplus >= self.rigorous_cost):
            more = surplus // self.rigorous_cost
            simple_needed   -= more
            rigorous_needed += more
            cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
            surplus = budget - cost
        result = 'Meets' if surplus >= 0 else 'Fails'
        print(f"{self.cadence:10} : Simple= {simple_needed:3.0f}, Rigorous= {rigorous_needed:3.0f}, " +\
            f"Cost ${cost:9.2f}, Surplus {surplus:9.2f}, {result:5}")
        return surplus
```

https://interviewdb.vercel.app

```
    """
    For the release cadence determines the total cost to run the minimum required tests.
    If there is surplus budget the surplus is reallocated to determine the new number of
    simple and rigorous tests that can be performed, prints this out, and returns the amount
    of budget remaining after reallocation.
    budget  The budget to allocate surplus The surplus budget after full allocation.  Will be >= 0 and <= min plan cost.
    """
    def compute_test_needs(self, budget):
        simple_needed   = self.simple_tests_needed()
        rigorous_needed = self.rigorous_tests_needed()
        cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
        surplus = budget - cost
        if (surplus >= self.rigorous_cost):
            more = surplus // self.rigorous_cost
            simple_needed   -= more
            rigorous_needed += more
            cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
            surplus = budget - cost
        result = 'Meets' if surplus >= 0 else 'Fails'
        print(f"{self.cadence:10} : Simple= {simple_needed:3.0f}, Rigorous= {rigorous_needed:3.0f}, " +\
            f"Cost ${cost:9.2f}, Surplus {surplus:9.2f}, {result:5}")
        return surplus


    def compute_test_needs(self, budget):
        simple_needed = self.simple_tests_needed()
        rigorous_needed = self.rigorous_tests_needed()
        cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
        surplus = budget - cost

        if surplus >= self.rigorous_cost:
            additional_rigorous_tests = min(surplus // self.rigorous_cost, (budget - cost) // self.rigorous_cost)
            simple_needed -= additional_rigorous_tests
            rigorous_needed += additional_rigorous_tests
            cost = simple_needed * self.simple_cost + rigorous_needed * self.rigorous_cost
            surplus = budget - cost

        result = 'Meets' if surplus >= 0 else 'Fails'
        print(f"{self.cadence:10} : Simple= {simple_needed:3.0f}, Rigorous= {rigorous_needed:3.0f}, "
            f"Cost ${cost:9.2f}, Surplus {surplus:9.2f}, {result:5}")
    return surplus



bi_weekly = ReleaseTestCadence('BIWEEKLY', 24, 2000.0, 6000.0)
monthly = ReleaseTestCadence('MONTHLY', 11, 2500.0, 8000.0)
bi_monthly = ReleaseTestCadence('BIMONTHLY', 6, 0.0, 12000.0)
quarterly = ReleaseTestCadence('QUARTERLY', 4, 0.0, 22000.0)

bi_weekly.compute_test_needs(240000.00)
monthly.compute_test_needs(240000.00)
bi_monthly.compute_test_needs(240000.00)
quarterly.compute_test_needs(240000.00)

```

@一亩三分地

```python
# import requests
# import mysql.connector
# import pandas as pd

"""
    This module represents the different release cadences and the associated data.
    Each enum instance has the number of releases per year and the costs for
    simple and rigorous tests.
    The data is accurate to the case handouts.  The code compiles.
"""
import math

class ReleaseTestCadence():
    """ tests """

    def __init__(self, cadence, releases, simple_cost, rigorous_cost):
        self.cadence = cadence
        self.releases = releases
        self.simple_cost = simple_cost
        self.rigorous_cost = rigorous_cost


    def simple_tests_needed(self):
        """
        Computes the minimum number of simple tests needed for this release cadence
        Returns the minimum number of simple tests needed for this release cadence
        """
        if (self.releases <= 6):
            return 0
        elif (self.releases > 12):
            return (self.releases // 3) * 2
        return self.releases // 2


    def rigorous_tests_needed(self):
        """
        Computes the minimum number of rigorous tests needed for this release cadence
        Returns the minimum number of rigorous tests needed for this release cadence
        """
        if (self.releases <= 6):
            return self.releases
        elif (self.releases > 12):
            return (self.releases // 3)
        return math.ceil(self.releases / 2)

    """
    For the release cadence determines the total cost to run the minimum required tests.
    If there is surplus budget the surplus is reallocated to determine the new number of
    simple and rigorous tests that can be performed, prints this out, and returns the amount
    of budget remaining after reallocation.
    budget  The budget to allocate surplus The surplus budget after full allocation.  Will be >= 0 and <= min plan cost.
    """
```

@一亩三分地

Virtual Card Number

The interviewer asked about VCN (Virtual Card Numbers), asking how much I know about it and my understanding of it (I told him this design is very good, as it can protect cardholders, because you can configure VCN to be valid in certain countries, and if you use this card in countries where the VCN is set to be invalid by default, the card won't be able to make transactions, and only the real cardholder has the permission to log in and configure VCN settings).
Then he gave me a scenario, saying the last two digits of an 8-digit transaction ID represent certain meanings, and the last few digits of the account number represent certain meanings (for example, 0 is auth, 1 is charge, or at certain positions 0 is visa, 1 is mastercard). Then he asked me to analyze whether the code is correct or not - it's actually checking shift operations and validation checks.
Then he asked some extensibility questions that I didn't quite understand clearly - about how to improve the method if the number system expands in the future. I said because each position isn't limited to just 0 and 1, we can set values from 0 to 9 and assign them different meanings based on different results. Then this round ended.

Another variant:
The last part was a case study about virtual credit (card) numbers, which wasn't a great segment - it seemed like the interviewer wasn't very engaged either. They roughly asked:

What are the benefits?
What are the potential benefits for the company?

Then they started reviewing code, which involved several rules for validating whether transactions are compliant. Each transaction has its own ID. There were errors in the code logic that needed to be pointed out - ideally by directly fixing them.
Then there was a follow-up question: with two new rules, write a simple logic (no need to write runnable code). The rules involved looking at the last 3 digits of the VCN and transaction ID, where each digit is either 1 or 0. For example:

If VCN's last digit is 1, it means it's VISA
If transaction ID's last digit is 1, it means it's an online transaction

Then there would be a rule like: for VISA cards, if it's an online transaction, the amount cannot exceed $1,000 (this specific rule is made up, but that's the general idea - the focus is on testing if your logic is clear).

https://interviewdb.vercel.app

Chatbot

Case scenario is if Capital One has a chatbot to answer customers' questions. What are the benefits it can offer from the perspective of customers and capital one? Then ask what kind of data to collect if you want to improve the chatbot? Then you are given some code to review and two table: Clickstream and Transaction

Clickstream

CID	Min	Event
1	1000	Login
1	1001	Enter password
2	1005	Login

Transaction

CID	Min	Transaction
1	1000	Popeye $20
2	1001	Safeway $50
3	1006	Coffee $5

```
class RetrieverQ(object):
        def __init__(self, store):
                self.store = store

        def retrieve_recent(self, customer_id, curr_time):
                result = []
                for index in range(0, 5):
                        val = self.store.retrieve(customer_id, curr_time - 5 + index)
                        if val is not None:
                                result.append(val)
                return result
```

Question 1: what does the function do?
Question 2: how can we improve the code?
Question 3: given the code below, what will the result look like

```
def aggregate(abc):
        return len(abc)
def join_data_left function（currentTime):
        clk = RetrieverQ(ClickStore)
        tsc = RetrieverQ(TransactionStore)
        result = []
        for id in clk.id.items():
```

https://interviewdb.vercel.app

```
        clk_data = clk.retrieve_recent(id,currentTime)
        tsc_data = tsc.retrieve_recent(id,currentTime)
        result.append((id,aggregate(clk_data),aggregate(tsc_data)))
return result
```

https://interviewdb.vercel.app