CSC365 Spring 2019

Lab1: Java POJO Application

Due: demo your work by April 11

The purpose of this lab is for you to understand what managing a collection of data entails without a help from DBMS.

Please read and understand all instructions before starting your implementation.

Physical data design depends on the workload. For example, *range queries* (e.g., find everyone who is older than 30 years) are best served by a balanced tree structure, while *match queries* (e.g., find everyone who is named John Smith) can be supported by both hash tables and tree structures (hash tables may be preferred because the amortized search time is O(1) compare to O(log(n)). Arrays and linked lists may also be suitable depending on the workload (i.e., the type of queries and updates that are expected and their frequencies).

In this lab, you will create a program that keeps track of credit cards, customers, and vendors, where all the information will be stored in main memory. It is your job to make the program as efficient as possible by choosing the data structures that give the best performance. Please use the data structures that are available in Java and do not implement your own data structures. I am assuming you will choose from HashSet, HashMap,TreeSet,TreeMap, and ArrayList in most cases.

Your program will store information about the following entities. Create two classes for each entity. For example, the **Customer** class will store information about a single customer, while the **CustomerCollection** class will store information about a collection of customers. Store the collection of customers using the data structure that is most appropriate and provide appropriate access methods.

customer: SSN (9 digit number), id (assign a unique sequential number), name, address, phone number. Example: John Smith with SSN 234234234 (9 digit number) and id 45 lives on 123 Main Street and has phone number 2342414230 (10 digit number). SSN is of course unique. The value of id is also unique and is automatically assigned by the system (e.g., use a static variable that increments by one to assign it). For the SSN of the customer, ask the user to provide it. If another customer with the same SSN already exists in the system, rejects the user request with an error message.

credit card: number (assign a unique sequential number), type (enum value, can be Visa, MC, American Express, or Discover), credit limit, current balance (must be less than the credit limit). For example, credit card with number 100 is Visa, credit limit of $10,000 and current balance of $3444.23. The credit card number is unique. You cannot have two credit cards with the same number. Add an additional bit that represents if the credit card is active. Credit cards are initially inactive.

ownership: which customer owns which credit card. For example, customer with id 7 owns credit card with credit card number 123. A customer can own multiple credit cards and a credit card can be owned by multiple customers (e.g., husband and wife may carry credit cards with the same credit card number). Add an additional bit that represents whether the ownership information is current. For example, if a user cancels a credit card, than this bit needs to be set to false.

vendor: id (assign a unique sequential number), name, address of main office. Example: vendor with id 23 has name “Best Buy” and main office address “123 Main St., Austin, TX”.

transaction: Contains information that a customer purchased something with a credit card. Example: On January 5th, 2015, customer with id 255 used CC with credit card number 234 at vendor id 233 for $654.23 worth of purchases. Store all information: date, customer, credit card, and vendor.

payment: Contains information that a payment was made on the balance of a credit card. For example, on June 2, 2015, $400.34 was paid on credit card 234.

Your program should support the following commands for updates. Identify object by their IDs.

1. u1: Create a new customer.
2. u2: Create a new credit card for an existing customer (will affect both the credit card and ownership data). Initially, the credit card will not be active.
3. u3: Issue a credit card duplicate for additional customer (will affect only the ownership data)
4. u4: Cancel a credit card.
5. u5: Activate a credit card.
6. u6: Add a new vendor.
7. u7: Create a new transaction. This will affect the balance of the credit card.
8. u8: Allow a customer to pay off credit card. This will affect both the payment data and credit card balance.

Your program should support the following commands for queries.

1. q1: Locate customer by ID or SSN.
2. q2: For a given customer (specified ID or SSN), print credit card information (i.e., credit card number, credit limit, and balance for each credit card).
3. q3: For a given credit card (specified by CC number), print credit card information (e.g., balance, credit limit, card holders).
4. q4: For a given credit card, print all transactions that are in a specified date range.

1. Your program should display a menu for supported commands and show a prompt and wait until the user types a command.
2. Show the result of the command the user typed.
3. For each command, you can either show prompts and ask the user to provide necessary information or demand the user to type the command with necessary arguments: ex. u1 -name=’Jon Snow’ -ssn=123456789 -address=’1 Grand Ave.’ -phone=8051234567.
4. go back to 1 and repeat until the user types z.
5. Your program should end when the user types z.

The updates and queries should be performed mainly by calling methods on objects from the collection classes. Create a **Main** class that contains the **main** method. The code in the Main class should not be very long. **Your program will be tested by executing Main.java**.

Show your program to the TA or the instructor during lab.

Please save and backup all labs after submission. Many labs will depend on previous labs.