

Assignment

MÔN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

MÃ MÔN: 503005

Đề bài: Module Quản lý dịch vụ thanh toán

Version: 1.0 – Ngày: 20/05/2023

(Sinh viên đọc kỹ tất cả hướng dẫn trước khi làm bài)

I. Giới thiệu bài toán

Một công ty cung cấp dịch vụ thanh toán cần quản lý dịch vụ của họ. Công ty này có 03 loại dịch vụ thanh toán là: Thẻ tiện lợi (Convenient Card), Ví điện tử (EWallet) và Tài khoản ngân hàng (Bank Account). Trong bài này, sinh viên sẽ lập trình một số chức để quản lý quá trình giao dịch của các loại dịch vụ thanh toán.

Các chức năng mà sinh viên phải thực hiện là:

- Cài đặt các lớp liên quan đến các loại dịch vụ thanh toán.
- Đọc thông tin các dịch vụ thanh toán.
- Xử lý các giao dịch thanh toán theo yêu cầu.

II. Tài nguyên cung cấp

Source code được cung cấp sẵn bao gồm các file:

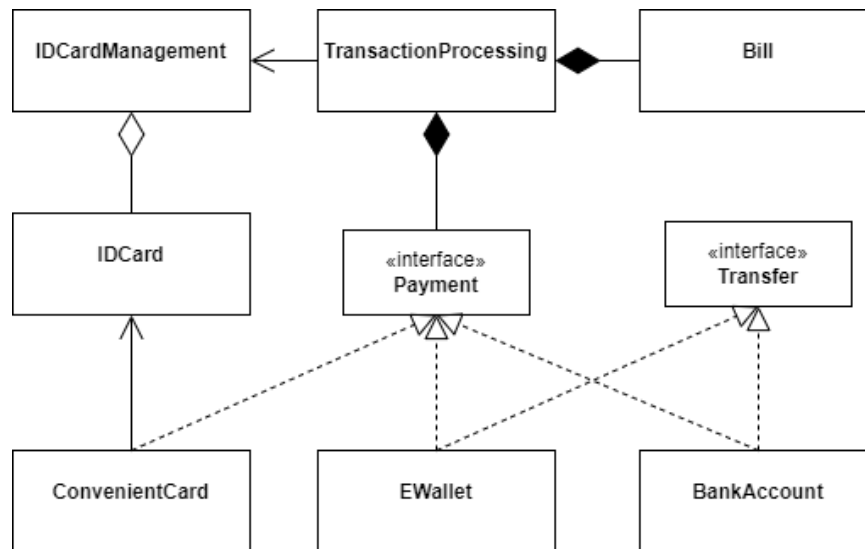
- File đầu vào và kết quả mong muốn:
 - o Folder *input* gồm 4 file:
 - *IDCard.txt*: chứa thông tin của các thẻ định danh.
 - *PaymentInformation.txt*: chứa thông tin các tài khoản của tất cả các loại dịch vụ.
 - *TopUpHistory.txt*: chứa thông tin các giao dịch nạp tiền.
 - *Bill.txt*: chứa dữ liệu các giao dịch thanh toán.
 - o Folder *expected_output* gồm: 7 file *Req3.txt*, *Req4.txt*, *Req5.txt*, *Req6.txt*, *Req7.txt*, *Req8.txt*, *Req9.txt* chứa kết quả mẫu của các Yêu cầu từ 3 đến 9 trong bài.
- File mã nguồn:
 - o *TestReq1.java* và *TestReq2.java*: tạo đối tượng, kiểm thử tương ứng Yêu cầu 1 và Yêu cầu 2.
 - o *Payment.java* và *Transfer.java*: là các interface tương ứng dùng để thanh toán và chuyển tiền.
 - o *Bill.java*: chứa lớp **Bill** được định nghĩa sẵn. File này sinh viên không được chỉnh sửa.
 - o *CannotCreateCard.java*: chứa lớp **CannotCreateCard** là ngoại lệ được dùng cho Yêu cầu 1.

- *IDCard.java*, *IDCardManagement.java*, *ConvenientCard.java*, *EWallet.java*, *BankAccount.java*: chứa các lớp được định nghĩa sẵn để tạo ra đối tượng tương ứng, sinh viên lập trình thêm vào các file này theo yêu cầu đề bài.
- *TransactionProcessing.java*: chứa lớp **TransactionProcessing** được định nghĩa sẵn thuộc tính *paymentObjects* để chứa danh sách các tài khoản thuộc các loại dịch vụ thanh toán khác nhau. Ngoài ra, lớp này còn chứa thêm một đối tượng là **IDCardManagement** để chứa danh sách các **IDCard** dùng cho việc khởi tạo loại dịch vụ thanh toán có liên quan. Phương thức khởi tạo và phương thức get được cung cấp sẵn. Sinh viên sẽ hiện thực thêm vào các phương thức còn trống trong file này và không chỉnh sửa phương thức có sẵn.

III. Trình tự thực hiện bài

- Sinh viên tải file tài nguyên được cung cấp sẵn và giải nén.
- Sinh viên hiện thực lớp **IDCard** và lập trình thêm vào lớp **IDCardManagement** theo mô tả lớp trong phần tiếp theo.
- Sinh viên hiện thực lớp **ConvenientCard**, **EWallet** và **BankAccount** theo mô tả lớp trong phần tiếp theo.
- Sau khi hoàn thành các lớp trên, sinh viên có thể biên dịch và chạy với phương thức **main** trong file *TestReq1.java* và *TestReq2.java* để kiểm thử Yêu cầu 1 và Yêu cầu 2 tương ứng.
- Sinh viên lập trình thêm vào các phương thức còn trống trong lớp **TransactionProcessing** theo mô tả và yêu cầu trong các phần tiếp theo.
- Sau khi hiện thực các phương thức trong lớp **TransactionProcessing**, sinh viên biên dịch và chạy với phương thức **main** trong file *Test.java* đã gọi sẵn các phương thức. Sinh viên thực hiện các yêu cầu ở mục dưới và so sánh kết quả của mình với kết quả trong folder *expected_output* đã được cung cấp sẵn.
- Đối với các yêu cầu sinh viên không làm được vui lòng không xóa và phải đảm bảo chương trình hoạt động được với phương thức **main** trong *Test.java* được cung cấp sẵn.
- Đường link Google Drive gửi đề này cho sinh viên sẽ chứa kèm một file *version.txt*, sinh viên nên thường xuyên vào xem file này, nếu thấy có nội dung mới thì nên đọc kỹ mô tả và tải lại đề mới nhất. File này được dùng để thông báo nội dung chỉnh sửa và ngày chỉnh sửa trong trường hợp đề bị sai hoặc lỗi.

IV. Mô tả các lớp trong bài



- Lớp **ConvenientCard**, lớp **EWallet** và lớp **BankAccount** hiện thực interface **Payment**, riêng lớp **EWallet** và **BankAccount** hiện thực thêm interface **Transfer**. Lớp **ConvenientCard** sử dụng **IDCard** làm thuộc tính. Lớp **IDCardManagement** dùng để quản lý danh sách các **IDCard**. Lớp **TransactionProcessing** chứa hóa đơn **Bill** và các dịch vụ thanh toán **Payment** để xử lý giao dịch thanh toán.
- Mô tả các lớp như sau:
 - o Tất cả các thuộc tính của lớp đều phải định nghĩa với access modifier là private hoặc protected. Trong quá trình hiện thực các lớp, sinh viên được phép định nghĩa thêm các phương thức get để lấy dữ liệu thuộc tính.
 - o Lớp **IDCard** – Thẻ định danh
 - Gồm 06 thuộc tính:
 - Thuộc tính 1: int – số định danh, gồm 6 ký tự số
 - Thuộc tính 2: String – họ tên
 - Thuộc tính 3: String – giới tính
 - Thuộc tính 4: String – ngày tháng năm sinh, định dạng dd/mm/yyyy
 - Thuộc tính 5: String – địa chỉ
 - Thuộc tính 6: int – số điện thoại, gồm 7 ký tự số
 - Phương thức:
 - Phương thức khởi tạo đầy đủ tham số theo thứ tự của 6 thuộc tính trên.
 - Các phương thức phụ trợ khác như: getter/setter, ... sinh viên có thể tự định nghĩa thêm.
 - Phương thức **toString()**: theo format **số định danh, họ tên, giới tính, ngày tháng năm sinh, địa chỉ, số điện thoại**.

- Lớp **IDCardManagement** – Quản lý danh sách thẻ định danh – **SV không chỉnh sửa lớp này**
 - Gồm 01 thuộc tính:
 - *idCards: ArrayList<IDCard>* – danh sách các thẻ định danh
 - Phương thức:
 - **public IDCardManagement(String path)**: phương thức khởi tạo truyền vào đường dẫn đến file IDCard.txt để đọc danh sách các thẻ định danh.
 - **public void readIDCard(String path)**: dùng để đọc file và tạo đối tượng thẻ định danh để thêm vào danh sách.
- Lớp **ConvenientCard** – Thẻ tiện lợi
 - Gồm 03 thuộc tính:
 - *type: String* – loại thẻ, có 02 loại thẻ chính là “Student” và “Adult”.
 - Thuộc tính 2: IDCard – thẻ định danh
 - Thuộc tính 3: double – số dư tài khoản
 - Phương thức:
 - Phương thức khởi tạo truyền vào 01 tham số với kiểu dữ liệu là *IDCard* tương ứng với đối tượng thẻ định danh. Mặc định số dư tài khoản là 100.
 - Các phương thức override theo interface mà lớp này hiện thực.
 - Phương thức nạp tiền với tham số kiểu double truyền vào số tiền nạp, phương thức này sẽ cộng trực tiếp số tiền nạp vào số dư tài khoản.
 - Phương thức **toString()**: theo format **thẻ định danh,loại thẻ,số dư tài khoản**
- Lớp **CannotCreateCard** – Ngoại lệ không thể tạo thẻ
 - Chứa phương thức khởi tạo truyền vào tham số kiểu *String* là thông điệp.
- Lớp **EWallet** – Ví điện tử
 - Gồm 02 thuộc tính:
 - Thuộc tính 1: int – số điện thoại, gồm 7 ký tự số
 - Thuộc tính 2: double – số dư tài khoản
 - Phương thức:
 - Phương thức khởi tạo truyền vào 01 tham số với kiểu dữ liệu là int tương ứng với số điện thoại. Mặc định số dư tài khoản là 0.
 - Các phương thức override theo các interface mà lớp này hiện thực.
 - Phương thức nạp tiền với tham số kiểu double truyền vào số tiền nạp, phương thức này sẽ cộng trực tiếp số tiền nạp vào số dư tài khoản.
 - Phương thức **toString()**: theo format **số điện thoại,số dư tài khoản**

- Lớp **BankAccount** – Tài khoản ngân hàng
 - Gồm 03 thuộc tính:
 - Thuộc tính 1: *int* – số tài khoản, gồm 6 ký tự số
 - Thuộc tính 2: *double* – tỉ lệ lãi suất
 - Thuộc tính 3: *double* – số dư tài khoản
 - Phương thức:
 - Phương thức khởi tạo truyền vào 02 tham số theo thứ tự kiểu dữ liệu tham số 1 là *int* tương ứng với số tài khoản và tham số 02 là *double* tương ứng với tỉ lệ lãi suất. Mặc định số dư tài khoản là 50.
 - Các phương thức override theo các interface mà lớp này hiện thực.
 - Phương thức nạp tiền với tham số kiểu *double* truyền vào số tiền nạp, phương thức này sẽ cộng trực tiếp số tiền nạp vào số dư tài khoản.
 - Phương thức **toString()**: theo format **số tài khoản,tỉ lệ lãi suất,số dư tài khoản**
- Interface **Transfer** – dùng cho chuyển khoản – **SV không chỉnh sửa interface này**
 - Có 02 lớp sẽ hiện thực interface này là lớp **EWallet** và lớp **BankAccount**.
 - 01 thuộc tính:
 - *transferFee*: *double* – thuộc tính hằng với tỉ lệ phí giao dịch là 0.05
 - Phương thức:
 - **public boolean transfer (double amount, Transfer to)**: dùng để chuyển tiền giữa các tài khoản với tham số *amount* là số tiền chuyển và *to* là tài khoản muốn chuyển đến.
 - **public double checkBalance()**: trả về số dư tài khoản.
- Interface **Payment** – dùng cho thanh toán – **SV không chỉnh sửa interface này**
 - Có 03 lớp sẽ hiện thực interface này là lớp **ConvenientCard**, **EWallet** và **BankAccount**.
 - Phương thức:
 - **public boolean pay(double amount)**: dùng để thanh toán số tiền *amount* truyền vào.
 - **public double checkBalance()**: trả về số dư tài khoản.
- Lớp **Bill** – hóa đơn – **Sinh viên không chỉnh sửa lớp này**
 - Thuộc tính:
 - *billID*: *int* – mã hóa đơn
 - *total*: *double* – tổng tiền
 - *payFor*: *String* – mục đích chi trả
 - Phương thức:
 - Các phương thức phụ trợ.
 - Phương thức **toString()**: theo format **mã hóa đơn,tổng tiền,mục đích chi trả**

- Lớp **TransactionProcessing** – Xử lý giao dịch
 - Thuộc tính:
 - *paymentObjects*: *ArrayList<Payment>* – danh sách chứa các tài khoản thanh toán
 - *idcm*: *IDCardManagement* – danh sách chứa các thẻ định danh
 - Phương thức:
 - Phương thức khởi tạo truyền vào 02 tham số là đường dẫn của file *IDCard.txt* để đọc danh sách các thẻ định danh và đường dẫn của file *PaymentInformation.txt* để đọc danh sách các tài khoản thanh toán.
 - ***ArrayList<Payment> getPaymentObject()***: trả về danh sách các tài khoản thanh toán.
 - Các phương thức còn lại sẽ tương ứng với Yêu cầu 3 đến Yêu cầu 9 và sẽ được mô tả bên dưới phần Yêu cầu đề bài.
- Lớp **TestReq1, TestReq2** – Chứa phương thức main kiểm thử Yêu cầu 1 và 2
- Lớp **Test** – Chứa phương thức main kiểm thử Yêu cầu 3 đến Yêu cầu 9
 - Phương thức:
 - ***boolean writeFile(String path, ArrayList<E> list)***: phương thức dùng để ghi danh sách thành file theo đường dẫn từ tham số *path* truyền vào.

V. Mô tả file input và file output

- File input có tên *IDCard.txt* chứa danh sách thẻ định danh với mỗi dòng ứng với thuộc tính của 01 thẻ định danh được cách nhau bằng dấu “,” theo định dạng:
số định danh,họ tên,giới tính,ngày tháng năm sinh,địa chỉ,số điện thoại
100001,John Doe,Male,01/05/2005,123 Main St,1234567
100003,Jane Smith,Female,15/08/2007,456 Oak Ave,2345678
Với số định danh gồm 6 ký tự số, ngày tháng năm sinh được đặt theo định dạng dd/mm/yyyy, số điện thoại gồm 7 ký tự số.
- File input có tên *PaymentInformation.txt* chứa thông tin của các tài khoản thanh toán tương ứng, với mỗi dòng ứng với thông tin của một đối theo định dạng:

- **ConvenientCard**

số định danh

100009

100010

Số định danh này sẽ tương ứng thẻ định danh trong ConvenientCard.

- **EWallet**

số điện thoại

6789012

8901234

○ **BankAccount**

số định danh, tỉ lệ lãi suất

100003, 0.01

100005, 0.01

- File input có tên *TopUpHistory.txt* chứa thông tin nạp tiền vào các tài khoản thanh toán, với mỗi dòng là một lệnh nạp tiền tương ứng với tài khoản theo định dạng:

mã loại tài khoản, thông tin tài khoản, số tiền nạp

CC, 100001, 50

BA, 100002, 1000

EW, 2345678, 600

CC, 100009, 400

CC, 100010, 200

Với mã tài khoản là “CC” ứng với tài khoản ConvenientCard, “EW” ứng với tài khoản EWallet, “BA” ứng với tài khoản BankAccount. Thông tin tài khoản là số định danh hoặc số điện thoại tùy theo loại tài khoản. Ví dụ dòng đầu tiên là nạp vào tài khoản **ConvenientCard** có thể định danh mang mã định danh là 10001 và số tiền nạp là 50.

- File input có tên *Bill.txt* chứa thông tin của các hóa đơn với mỗi dòng là một hóa đơn theo định dạng:

mã hóa đơn, tổng tiền, mục đích chi trả, mã loại tài khoản, thông tin tài khoản

1, 222, Clothing, BA, 100002

2, 100, Food, CC, 100001

3, 550, Clothing, BA, 100008

Ví dụ dòng đầu tiên là hóa đơn mã hóa đơn là 1, tổng tiền chi trả là 222, chi trả cho *Clothing* (quần áo), trả bằng tài khoản ngân hàng có số tài khoản là 100002.

- Thư mục *expected_output* gồm có 7 file ứng với kết quả mẫu cho các yêu cầu từ 3 đến 9. **Mỗi dòng trong từng file *.txt ứng với một đối tượng, có định dạng ghi ra theo phương thức *toString()* của lớp tương ứng.**

Lưu ý:

- Sinh viên có thể tự thêm dữ liệu vào file input để thử nhiều trường hợp khác nhau nhưng lưu ý thêm dữ liệu phải đúng định dạng đã được nêu bên trên.
- Sinh viên nên đọc kỹ nội dung trong các phương thức **main** để xác định hướng hiện thực.
- Sinh viên có thể thêm một số thao tác vào phương thức **main** để kiểm các phương thức đã định nghĩa tuy nhiên đảm bảo bài làm của sinh viên **phải chạy được trên phương thức main ban đầu đã cung cấp sẵn**.
- Sinh viên có thể thêm phương thức mới để phục vụ bài làm tuy nhiên bài làm của sinh viên phải chạy được với các file *TestReq1.java*, *TestReq2.java* và *Test.java* đã được cung cấp sẵn.
- Sinh viên tuyệt đối không đặt các đường dẫn tuyệt đối khi định nghĩa phương thức liên quan đến đọc file. Nếu sinh viên tự ý đặt đường dẫn tuyệt đối dẫn đến quá trình chấm không đọc được file để chấm thì tương đương với bài làm biên dịch lỗi.
- **Tuyệt đối tuân thủ theo tên của các phương thức được yêu cầu cụ thể trong đề bài.**

VI. Yêu cầu

Sinh viên không tự ý thêm thư viện ngoài, chỉ dùng những thư viện có sẵn trong các file nhận được.

Sinh viên thực hiện bài tập lớn trên Java 11 hoặc Java 8. Sinh viên không được dùng kiểu dữ liệu *var*. Bài làm của sinh viên sẽ được chấm trên Java 11, sinh viên tự chịu trách nhiệm tất cả các lỗi phát sinh nếu dùng các phiên bản Java khác.

Sinh viên được phép định nghĩa thêm phương thức để hỗ trợ bài làm nhưng chỉ định nghĩa thêm trong các file được yêu cầu nộp. **Sinh viên không tự ý thay đổi tên phương thức và tham số truyền vào của các phương thức có sẵn và các phương thức được yêu cầu trong đề bài.**

Các thuộc tính được đánh số trong phần **Mô tả các lớp trong bài**, khi sinh viên định nghĩa sẽ tự đặt tên cho thuộc tính, chỉ cần đảm bảo đúng kiểu dữ liệu theo mô tả và *access modifier* của tất cả các thuộc tính phải là *private*.

Sinh viên được phép điều chỉnh code các file *TestReq1.java*, *TestReq2.java*, *Test.java* và thêm các dữ liệu mới vào các file trong thư mục *input* để kiểm thử thêm nhiều trường hợp khác sau khi đã kiểm tra kết quả với code có sẵn trong các file này.

Phương thức **boolean pay(double amount)** được xem là thanh toán thành công khi trả về *true* và thanh toán không thành công khi trả về *false*. Cách tính thanh toán của từng loại dịch vụ thanh toán như sau:

- **ConvenientCard**

- Nếu loại thẻ là “Student”: Số tiền cần thanh toán = số tiền *amount*
- Nếu loại thẻ là “Adult”: Số tiền cần thanh toán = số tiền *amount* + phí là 1% số tiền *amount*

Nếu số Số tiền cần thanh toán \geq số dư tài khoản thì cho phép thanh toán, trừ số tiền cần thanh toán vào số dư tài khoản và trả về *true*, ngược lại không đủ số dư thì không trừ và trả về *false*.

- **EWallet**

Số tiền cần thanh toán = số tiền *amount*

Nếu số Số tiền cần thanh toán \geq số dư tài khoản thì cho phép thanh toán, trừ số tiền cần thanh toán vào số dư tài khoản và trả về *true*, ngược lại không đủ số dư thì không trừ và trả về *false*.

- **BankAccount**

Số tiền cần thanh toán = số tiền *amount*

Tài khoản ngân hàng phải luôn phải có số dư tối thiểu là 50. Nếu số Số tiền cần thanh toán \geq số dư tài khoản + 50 thì cho phép thanh toán, trừ số tiền cần thanh toán vào số dư tài khoản và trả về *true*, ngược lại không đủ thì không trừ và trả về *false*.

1. YÊU CẦU 1 (1 điểm)

Sinh viên hiện thực lớp **ConvenientCard**. Loại thẻ của lớp này được quy định như sau:

- Nếu số tuổi theo thẻ định danh < 12 thì sẽ không cho phép tạo thẻ và ném ra ngoại lệ **CannotCreateCard**.
- Nếu số tuổi theo thẻ định danh ≤ 18 thì loại thẻ sẽ là “Student”.
- Các trường hợp còn lại thì loại thẻ sẽ là “Adult”.

Sinh viên hiện thực lớp trên và sử dụng file *TestReq1.java* để kiểm thử lại bài làm, sinh viên có thể sửa đổi nội dung trong phương thức main để kiểm tra nhiều trường hợp. Đây là kết quả in ra màn hình với file *TestReq1.java* mặc định cho sẵn.

```
CannotCreateCard: Not enough age  
Type of the card: Student
```

2. YÊU CẦU 2 (1 điểm)

Định nghĩa phương thức:

public boolean transfer(double amount, Transfer to)

trong lớp **EWallet** và lớp **BankAccount** dùng để chuyển tiền giữa tài khoản hiện tại và tài khoản *to*. Với tham số *amount* là số tiền chuyển và *to* là tài khoản nhận tiền.

Số tiền chuyển = số tiền *amount* + tỉ lệ phí giao dịch *transferFee* * số tiền *amount*

(tỉ lệ phí giao dịch có thể thay đổi tùy theo interface được cung cấp, sinh viên phải dùng thuộc tính *transferFee* của interface)

Số tiền nhận = số tiền *amount*

Với **EWallet** nếu Số tiền chuyển \geq số dư tài khoản thì cho phép chuyển, với **BankAccount** nếu Số tiền chuyển \geq số dư tài khoản + 50 thì cho phép chuyển. Nếu chuyển thành công thì trả về *true*, ngược lại thì không cho phép chuyển thì trả về *false*.

Khi chuyển thành công, Số tiền chuyển sẽ trừ trực tiếp vào số dư tài khoản của tài khoản chuyển và số tiền nhận sẽ cộng trực tiếp vào số dư tài khoản của tài khoản nhận.

Sinh viên hiện thực lớp trên và sử dụng file *TestReq2.java* để kiểm thử lại bài làm, sinh viên có thể sửa đổi nội dung trong phương thức main để kiểm tra nhiều trường hợp. Đây là kết quả in ra màn hình với file *TestReq2.java* mặc định cho sẵn.

```
EWallet 1: 500.0  
EWallet 2: 0.0  
-----  
EWallet 1: 290.0  
EWallet 2: 200.0  
-----  
BankAccount 1: 50.0  
-----  
EWallet 1: 80.0  
BankAccount 1: 250.0  
-----  
BankAccount 1: 250.0  
EWallet 2: 200.0
```

3. YÊU CẦU 3 (2 điểm)

Từ Yêu cầu 3 đến Yêu cầu 9 sẽ liên quan đến dữ liệu đọc từ các file trong thư mục *data* cung cấp. Sinh viên sẽ định nghĩa các phương thức trong lớp **TransactionProcessing** và dùng lớp **Test** để kiểm tra kết quả bài làm.

Hiện thực phương thức:

public boolean readPaymentObject(String path)

Phương thức này sẽ đọc file *PaymentInformation.txt* theo đường dẫn của tham số *path* truyền vào để đọc thông tin các tài khoản thanh toán vào danh sách *paymentObjects*.

Sinh viên hiện thực phương thức trên, biên dịch file *Test.java* và chạy lệnh *java Test 1* để ghi ra kết quả file *Req3.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req3.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

Lưu ý: Đây là phương thức sinh viên phải định nghĩa được mới bắt đầu tính điểm cho các Yêu cầu từ 3 đến 9, nếu không đọc được file thành danh sách các đối tượng **Payment** thì các yêu cầu bên dưới không được tính điểm.

Các yêu cầu bên dưới thao tác trực tiếp trên dữ liệu đã đọc từ file.

4. YÊU CẦU 4 (1 điểm)

Hiện thực phương thức

public ArrayList<ConvenientCard> getAdultConvenientCards()

trả về danh sách các thẻ tiện lợi **ConvenientCard** có loại thẻ là “Adult”.

Sinh viên hiện thực phương thức trên, biên dịch file *Test.java* và chạy lệnh *java Test 2* để ghi ra kết quả file *Req4.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req4.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

5. YÊU CẦU 5 (1 điểm)

Hiện thực phương thức

public ArrayList<IDCard> getCustomersHaveBoth()

trả về danh sách thẻ định danh **IDCard** sở hữu cả 03 loại tài khoản thanh toán.

Sinh viên hiện thực phương thức trên, biên dịch file *Test.java* và chạy lệnh *java Test 3* để ghi ra kết quả file *Req5.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req5.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

6. YÊU CẦU 6 (1 điểm)

Hiện thực phương thức

public void processTopUp(String path)

đọc file và xử lý thông tin đọc vào từ file *TopUpHistory.txt* thông qua đường dẫn từ tham số *path* để nạp tiền cho các tài khoản trong danh sách *paymentObjects* theo các thông tin trong file. (Gợi ý: sau khi đọc thông tin từ file, sinh viên gọi đến phương thức nạp tiền của lớp tương ứng để nạp tiền)

Để kiểm tra kết quả của yêu cầu này, danh sách *paymentObjects* sẽ được ghi ra file. Phương thức **main** đã hiện thực sẵn quá trình này. Sinh viên biên dịch file *Test.java* và chạy lệnh *java Test 4* để ghi ra kết quả file *Req6.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req6.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

Lưu ý: Đây là phương thức sinh viên phải định nghĩa được mới bắt đầu tính điểm cho các Yêu cầu từ 7 đến 9, nếu không đọc được file hoặc không xử lý được đúng yêu cầu nạp tiền thì các yêu cầu bên dưới không được tính điểm.

7. YÊU CẦU 7 (1 điểm)

Hiện thực phương thức

public ArrayList<Bill> getUnsuccessfulTransactions(String path)

đọc file *Bill.txt* thông qua đường dẫn từ tham số *path*, xử lý thông tin các giao dịch đã đọc được và trả về danh sách các hóa đơn thanh toán không thành công (hóa đơn thanh toán không thành công là hóa đơn có tài khoản thanh toán thực hiện lệnh thanh toán thông qua phương thức **pay()** không thành công). Lưu ý thứ tự các giao dịch trong file *Bill.txt* sẽ đi theo thứ tự từ trên xuống.

Để kiểm tra kết quả của yêu cầu này, phần code được cung cấp sẵn trong phương thức **main** đã gọi lệnh thực hiện bước nạp tiền để sinh viên dễ dàng kiểm tra kết quả. Sinh viên biên dịch file *Test.java* và chạy lệnh *java Test 5* để ghi ra kết quả file *Req7.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req7.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

8. YÊU CẦU 8 (1 điểm)

Hiện thực phương thức

public ArrayList<BankAccount> getLargestPaymentByBA(String path)

đọc file *Bill.txt* thông qua đường dẫn từ tham số *path*, xử lý thông tin các giao dịch đã đọc được và trả về danh sách tài khoản ngân hàng có tổng số tiền thanh toán thành công lớn nhất. Nếu chỉ có một tài khoản có số tiền giao dịch thành công lớn nhất thì danh sách kết quả có một phần tử, nếu có nhiều tài khoản lớn bằng nhau thì danh sách kết quả có nhiều phần tử.

Để kiểm tra kết quả của yêu cầu này, phần code được cung cấp sẵn trong phương thức **main** đã gọi lệnh thực hiện bước nạp tiền để sinh viên dễ dàng kiểm tra kết quả. Sinh viên biên dịch file *Test.java* và chạy lệnh *java Test 6* để ghi ra kết quả file *Req8.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req8.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

9. YÊU CẦU 9 (1 điểm)

Hiện thực phương thức

public void processTransactionWithDiscount(String path)

đọc file *Bill.txt* thông qua đường dẫn từ tham số *path*, xử lý thông tin các giao dịch đã đọc được và thực hiện các thanh toán cho các giao dịch. Tuy nhiên, ở yêu cầu với khách hàng là nữ dưới 18 tuổi hoặc khách hàng là nam dưới 20 tuổi khi thanh toán đơn hàng có *mục đích chi trả* là **Clothing** bằng

tài khoản ví điện tử với *tổng tiền* của hóa đơn trên 500 thì hóa đơn đó *tổng tiền* phải thanh toán thực tế của hóa đơn đó sẽ được giảm 15%.

Để kiểm tra kết quả của yêu cầu này, phần code được cung cấp sẵn trong phương thức **main** đã gọi lệnh thực hiện bước nạp tiền và ghi file danh sách *paymentObjects*. Sinh viên biên dịch file *Test.java* và chạy lệnh *java Test 7* để ghi ra kết quả file *Req9.txt* vào thư mục *output*. Sinh viên xem kết quả file này và file *Req9.txt* trong folder *expected_output* mẫu được cung cấp sẵn để so sánh kết quả.

VII. Lưu ý kiểm tra trước khi nộp bài

- Nếu sinh viên không thực hiện được yêu cầu nào thì để nguyên phương thức của yêu cầu đó, **TUYỆT ĐỐI KHÔNG XÓA PHƯƠNG THỨC CỦA YÊU CẦU** sẽ dẫn đến lỗi khi chạy phương thức **main**. Trước khi nộp phải kiểm tra chạy được với phương thức **main** được cho sẵn.
- Tất cả các file ReqX.txt ($X = \{3,4,5,6,7,8,9\}$) được ghi ra trong thư mục *output*. Đối với sinh viên sử dụng IDE (Eclipse, Netbean, ...) phải đảm bảo file chạy được bằng command prompt, đảm bảo bài làm không nằm trong package. **Trường hợp bài làm đặt trong package thì sẽ bị 0 điểm cả bài.**
- File kết quả ghi đúng thư mục khi chạy chương trình sẽ có dạng như sau:

Name	Date modified	Type	Size
data	15/05/2023 21:19	File folder	
output	17/05/2023 15:26	File folder	
BankAccount.class	15/05/2023 21:20	CLASS File	2 KB
BankAccount.java	15/05/2023 20:52	JAVA File	2 KB
Bill.class	15/05/2023 21:20	CLASS File	2 KB
Bill.java	15/05/2023 20:52	JAVA File	1 KB
CannotCreateCard.class	04/05/2023 14:54	CLASS File	1 KB
CannotCreateCard.java	03/05/2023 11:09	JAVA File	1 KB
ConvenientCard.class	15/05/2023 21:20	CLASS File	2 KB
ConvenientCard.java	15/05/2023 20:52	JAVA File	2 KB
EWallet.class	15/05/2023 21:20	CLASS File	2 KB
EWallet.java	15/05/2023 20:52	JAVA File	2 KB
IDCard.class	15/05/2023 21:20	CLASS File	2 KB
IDCard.java	15/05/2023 20:52	JAVA File	2 KB
IDCardManagement.class	07/05/2023 02:09	CLASS File	2 KB
IDCardManagement.java	07/05/2023 02:07	JAVA File	1 KB
Payment.class	05/05/2023 08:41	CLASS File	1 KB
Payment.java	04/05/2023 15:07	JAVA File	1 KB
Test.class	17/05/2023 15:26	CLASS File	3 KB
Test.java	17/05/2023 15:26	JAVA File	4 KB
TestReq1.class	15/05/2023 21:20	CLASS File	1 KB
TestReq1.java	07/05/2023 02:07	JAVA File	1 KB
TestReq2.class	15/05/2023 21:20	CLASS File	2 KB
TestReq2.java	05/05/2023 08:45	JAVA File	2 KB
TransactionProcessing.class	15/05/2023 21:20	CLASS File	7 KB
TransactionProcessing.java	15/05/2023 20:30	JAVA File	15 KB

- Bên trong thư mục *output*:

Name	Date modified	Type	Size
Req3.txt	15/05/2023 21:20	Text Document	1 KB
Req4.txt	15/05/2023 21:20	Text Document	1 KB
Req5.txt	15/05/2023 21:20	Text Document	1 KB
Req6.txt	15/05/2023 21:15	Text Document	1 KB
Req7.txt	15/05/2023 21:15	Text Document	1 KB
Req8.txt	15/05/2023 21:15	Text Document	1 KB
Req9.txt	15/05/2023 21:22	Text Document	1 KB

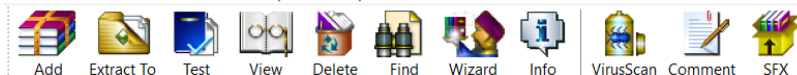
VIII. Hướng dẫn nộp bài

- Khi nộp bài sinh viên nộp lại file *ConvenientCard.java*, *EWallet.java*, *BankAccount.java*, *IDCard.java* và file *TransactionProcessing.java*, **không nộp kèm bất cứ file nào khác và tuyệt đối không được sửa tên 5 file này.**
- **Sinh viên đặt 5 file bài làm vào thư mục *MSSV_HoTen*** (HoTen viết liền, không dấu) và nén lại với định dạng **.zip** và nộp vào mục nộp tương ứng trên hệ thống ELIT (elit.tdtu.edu.vn).
- Trường hợp làm sai yêu cầu nộp bài (đặt tên thư mục sai, không để bài làm vào thư mục khi nộp, nộp dư file, nộp thiếu file, ...) thì bài làm của sinh viên sẽ bị **0 điểm**.
- File nộp đúng sẽ như sau:

- o File nén nộp bài:

 51403039_DungCamQuang.zip	17/05/2023 15:16	WinRAR ZIP archive	3 KB
---	------------------	--------------------	------

- o Bên trong file nén:

51403039_DungCamQuang.zip (evaluation copy)				
File Commands Tools Favorites Options Help				
				
51403039_DungCamQuang.zip - ZIP archive, unpacked size 1,997 bytes				
Name	Size	Packed	Type	Modified
..			Local Disk	
51403039_DungCamQuang			File folder	17/05/2023 15:04

- o Bên trong thư mục:

51403039_DungCamQuang.zip\51403039_DungCamQuang - ZIP archive, unpacked size 1,563 bytes				
Name	Size	Packed	Type	Modified
..			Local Disk	
BankAccount.java	65	63	JAVA File	17/05/2023 14:53
ConvenientCard.java	58	55	JAVA File	17/05/2023 14:53
EWallet.java	58	57	JAVA File	17/05/2023 14:52
IDCard.java	34	32	JAVA File	17/05/2023 14:52
TransactionProcessing.java	1,348	391	JAVA File	17/05/2023 14:48

IX. Đánh giá và quy định

- Bài làm sẽ được chấm tự động thông qua testcase (file input và output có định dạng như mẫu đã gửi kèm) do đó sinh viên tự chịu trách nhiệm nếu không thực hiện đúng theo Hướng dẫn nộp bài hoặc tự ý sửa tên các phương thức đã có sẵn dẫn đến bài làm không biên dịch được khi chấm.
- **Tất cả các trường hợp sinh viên gán cứng đường dẫn trong quá trình đọc file đều sẽ bị 0 điểm cả bài.**
- Testcase sử dụng để chấm bài là file khác với file sinh viên đã nhận, sinh viên chỉ được điểm mỗi YÊU CẦU khi chạy ra đúng hoàn toàn kết quả theo testcase chấm của yêu cầu đó.
- Nếu bài làm của sinh viên biên dịch bị lỗi trên máy chấm thì **0 điểm cả bài.**
- **Tất cả code sẽ được kiểm tra sao chép. Mọi hành vi sao chép code trên mạng, chép bài bạn hoặc cho bạn chép bài nếu bị phát hiện đều sẽ bị điểm 0 vào điểm Quá trình 2 hoặc cấm thi cuối kì.**
- **Đặc biệt, nếu sinh viên sử dụng chat GPT trong bài tập lớn này thì sinh viên sẽ bị xử lý tương tự hành vi sao chép bài.**
- Nếu bài làm của sinh viên có dấu hiệu sao chép trên mạng hoặc sao chép nhau, sinh viên sẽ được gọi lên phòng vấn code để chứng minh bài làm là của mình.
- **Hạn chót nộp bài: 23h59 ngày 04/06/2023.**
- **Sinh viên nộp đúng vào mục bài tập của Bài tập lớn trên hệ thống ELIT.**

-- HẾT --