COMP2500 실습 11

목차

다음의 스토리는 엄청난 허구를 담고 있습니다!

POCU 사에는 각각 애플, 마이크로소프트, 삼성 장비를 구비하고 있는 3개의 창고가 있습니다. 각 부서는 매년 장비 구매 예산을 배정받으며, 그 예산을 이용하여 위 창고로부터 필요한 장비를 자유로이 구매할 수 있습니다. 각 직원에게 최고의 장비를 제공하여 생산력을 극대화하자는 취지죠! :D 회사 규모가 커짐에 따라 미래에 다른 브랜드 창고도 열 계획도 있다네요~

각 직원이 쉽게 장비를 구매할 수 있도록 앱을 만들어 봅시다. 이 앱을 통해 직원은 어떤 창고에 구비되어 있는 장비 목록을 확인한 뒤, 원하는 장비를 구매할 수 있습니다. 부서에 예산이 남아있다면 언제라도 이 앱을 열어서 필요한 장비를 구매할 수 있으니 매우 편하겠죠? POCU 사의 툴(tools) 프로그래머가 필요한 라이브러리들을 `academy.pocu.comp2500.lab11.pocu` 패키지 안에 만들어 놓았으니 이걸 이용해서 이 앱을 만들어 보세요. 여러분이 해야 할 일은 장비 구매 앱을 만드는 것뿐이랍니다. :D

1. 프로젝트를 준비한다

1. `Lab11` 폴더로 이동합니다.
2. 자, 스트레칭 한 번 하시고... 설계와 프로그래밍 시작하겠습니다. :)

2. POCU 사내 장비 구매 앱 구현하기

전반적인 규칙

- 여러분이 작성하는 클래스들은 반드시 `academy.pocu.comp2500.lab11` 패키지 안에 속해야 합니다.
- `private static final` 멤버 변수 외에 어떤 정적 멤버 변수도 사용할 수 없습니다.

- `academy.pocu.comp2500.lab11.pocu` 패키지 안에 있는 파일들을 변경해도 빌드봇이 무시합니다.
- `academy.pocu.comp2500.lab11.pocu` 패키지 안에 제공된 파일들은 그냥 여러분의 디버깅과 테스트를 돕기 위해 제공한 것뿐이며, 실제 라이브러리의 구현은 아닙니다. 이 라이브러리에 있는 각 클래스의 자세한 명세는 '부록 A' 섹션을 참고하세요.
- `academy.pocu.comp2500.lab11.pocu` 패키지 안에 있는 클래스에서 이름이 `Mock`으로 끝나거나 바로 위에 `// mock` 주석이 있는 메서드들은 테스트를 위해 제공하는 mock 메서드입니다. 이 메서드들을 변경하면 다양한 상황들을 본인 컴퓨터에서 테스트할 수 있답니다.
- 어떤 클래스 대신 사용할 수 있는 기본 자료형이 존재한다면(예: Boolean/boolean, Integer/int) 그걸 대신 사용하세요. 기본 자료형 대신 클래스 버전이 허용되는 경우는 제네릭(generic) 클래스의 타입(type) 매개변수로 사용할 때 뿐입니다.
- `java.lang.Math`를 사용할 수 없습니다. 사용하면 감점 되니 시도조차 하지 마세요. :)

2.1 App 클래스의 run() 메서드를 구현한다

- 이 메서드는 다음의 인자를 받습니다.
  - 입력 스트림: `BufferedReader in`
  - 출력 스트림: `PrintStream out`
  - 오류 스트림: `PrintStream err`
- `in`으로부터 사용자 입력을 읽어 옵니다.
- `out`으로 텍스트를 출력합니다.
- `err`으로 오류 코드를 출력합니다.

이 메서드는 반드시 다음과 같이 동작해야 합니다. (순서가 틀려서도 안 됨)

1. 사용자에게 창고(warehouse)를 고르라는 메시지를 보여준다.
   - 이 메시지는 반드시 'WAREHOUSE:'로 시작해야 하며, 그 뒤에 오는 메시지는 무엇이든 상관없습니다. (단, 줄 바꿈을 하면 안 됨)
   - 다음 줄부터는 `WarehouseType`의 각 열거형 값을 한 줄에 하나씩 출력해야 합니다. 단, 그 앞에 순서를 나타내는 번호를 붙여야 합니다. 아래 예에서 허용되는 포맷을 확인하세요.

     ```
     WAREHOUSE: Choose your warehouse!
     1. APPLE
     2. MICROSOFT
     3. SAMSUNG
     ```

2. 입력 스트림으로부터 사용자의 선택을 읽어온다.
   - 사용자는 숫자로 본인의 선택을 입력해야 합니다. 예를 들어 위의 예에서 사용자가 '2'를 선택하면 이는 MICROSOFT 창고를 선택했음을 의미합니다.
   - 사용자의 잘못된 입력을 처리하는 것은 여러분의 책임입니다. 잘못된 입력이 들어왔다면 1번 단계로 돌아갑니다.
   - 사용자가 'exit'을 입력하면 이 메서드를 종료해야 합니다.
3. 사용자 소속 부서의 지갑을 구해온다.
   - 사용자가 그 지갑을 사용할 수 있는 권한이 없다면 `err` 스트림에 'AUTH_ERROR'를 출력한 뒤, 이 메서드를 종료해야 합니다.
4. 지갑의 잔고를 출력한다.
   - 잔고는 한 줄에 출력하며, 다음의 포맷을 따릅니다.

     ```
     BALANCE: <balance>
     ```

5. 사용자에게 구매할 장비를 고르라는 메시지를 보여준다.
   - 이 메시지는 반드시 'PRODUCT_LIST:'로 시작해야 하며, 그 뒤에는 어떤 메시지가 와도 상관없습니다. (단, 줄 바꿈을 하면 안 됨)
   - 다음 줄부터는 그 창고로부터 구해온 장비 목록을 열거해야 합니다. 이 목록의 순서를 바꾸면 안 됩니다. 한 줄에 장비 하나씩 번호를 붙여서 출력하되, 이름과 가격을 출력하는 것도 잊지 마세요.

     다음은 허용되는 메시지 포맷의 한 예입니다.

     ```
     PRODUCT_LIST: Choose the product you want to buy!
     1. IPad              10
     2. IPod               2
     3. Macbook Pro       15
     4. Apple TV           9
     ```

6. 입력 스트림으로부터 사용자의 선택을 읽어온다.
   - 사용자는 본인이 구매하고자 하는 장비의 번호를 입력해야 합니다. 예를 들어 위 예에서 사용자가 '2'를 입력하면 이는 IPod를 선택했다는 의미입니다.
   - 사용자의 잘못된 입력을 처리하는 것은 여러분의 책임입니다. 잘못된 입력이 들어왔다면 4번 단계로 돌아갑니다.

    - 사용자가 'exit'을 입력하면 이 메서드를 종료합니다.
7. 6번 단계에서 사용자가 선택한 장비를 구매한 뒤, 4번 단계로 돌아갑니다.

    - 만약에 사용자가 선택한 장비의 구매를 완료하기 전에 창고에서 그 장비가 사라졌다면(다른 직원이 같은 시각에 그 장비를 구매할 수 있으니까요~) 더 이상 그 장비를 구매할 수 없겠죠? 이런 경우는 4번 단계로 돌아갑니다.

2.2 `SafeWallet` 클래스 구현하기

여러분이 잠시 시간을 내서 `Wallet` 클래스에 있는 `deposit()` 메서드를 검토해보니 오버플로우 문제가 발생할 여지가 있다고 판단했습니다. 무턱대고 현재 잔고에 금액을 계속 더하다 보면 그런 일이 발생하고, 그러면 잔고가 마이너스가 되어버리겠죠? 이런 경우에는 이 메서드에서 예외를 던져 프로그램 실행을 멈추는 게 올바르다고 판단했습니다.

2.2.1 `OverflowException` 클래스를 구현한다

- 생성자는 다음의 인자를 받습니다.
    - 메시지를 나타내는 `string`
- 이 클래스는 `RuntimeException`을 상속해야 합니다.

2.2.2 `SafeWallet` 클래스를 구현한다

- `Wallet` 클래스를 상속받습니다.
- `deposit()` 메서드를 오버라이딩해서 지갑 잔고에 오버플로우 문제가 발생할 때 `OverflowException`을 던지세요.
- 오버라이딩 한 `deposit()` 메서드에서도 부모(`super`)의 메서드를 호출해야 합니다.

2.2.3 `App.run()` 메서드 안에서 `SafeWallet` 사용한다

`App.run()` 메서드 안에서 `Wallet` 클래스 대신 `SafeWallet` 클래스를 사용하세요. `OverflowException` 예외가 발생할 때 프로그램을 정상적인 상태로 복구하는 것은 좀 말이 안 되니 이 경우에는 프로그램이 크래시가 나도록 놔두기로 결정했답니다.

3. 본인 컴퓨터에서 테스트하는 법

- 다음의 코드를 사용해서 사내 장비 구매 앱을 실행할 수 있습니다. `Program.java`의 `main()` 메서드 안에 이 코드를 추가하세요.

```
App app = new App();

app.run(new BufferedReader(new InputStreamReader(System.in)), System.out, System.err);
```

다양한 값을 입력하면서 테스트를 해보세요. 앞에서도 알려드렸듯이 이름이 `Mock`으로 끝나거나 `// mock` 주석이 달린 메서드는 여러분의 컴퓨터에서 쉽게 테스트를 할 수 있도록 제공한 메서드입니다. 이 메서드들을 자유롭게 바꿔가며 다양한 테스트를 해보세요.

4. 커밋, 푸시 그리고 빌드 요청

이건 어떻게 하는지 이제 다 아시죠? :)

A. 부록

다음은 `academy.pocu.comp2500.lab11.pocu` 라이브러리에 있는 클래스들의 자세한 정보입니다.

A.1 `Department` 열거형

- POCU의 각 부서를 나타내는 열거형
- 다음과 같은 열거형 값이 있습니다.
    - `ENGINEERING`
    - `OPERATION`
    - `QUALITY_ASSURANCE`
    - `HUMAN_RESOURCES`

A.2 `Product` 클래스

`Product` 클래스는 `Warehouse` 안에 구비되어 있는 장비를 나타냅니다.

A.2.1 생성자

```
public Product(UUID id, String name, int price)
```

- `id`, `name`, `price`로 지정된 장비를 생성합니다.
- `name`의 최대 길이는 20입니다.

A.2.2 `public` 메서드

- `public UUID getId()` : 장비의 ID를 반환합니다.
- `public String getName()` : 장비의 이름을 반환합니다.
- `public int getPrice()` : 장비의 가격을 반환합니다.

A.3 `User` 클래스

`User` 클래스는 사내 장비 구매 앱을 사용하는 사용자를 나타냅니다. 생성자가 호출될 때 현재 컴퓨터에 로그인된 사용자의 이름과 부서를 알아서 가져옵니다. 실제 이름과 부서를 불러오는 법에 대해서는 고민하실 필요 없습니다. 그냥 알아서 된다고 믿으시고 `getFullname()` 과 `getDepartment()` 메서드를 호출하면 언제나 유효한 값이 반환된다고 생각하세요

A.3.1 생성자

- `public User()` : 이 앱을 실행하는 컴퓨터에 로그인된 사용자의 이름과 부서 정보를 이용해서 새 개체를 생성합니다

A.3.2 `public` 메서드

- `public String getFullName()` : 사용자의 이름을 반환합니다.
- `public Department getDepartment()` : 사용자의 부서를 반환합니다.

A.4 `Wallet` 클래스

`Wallet` 클래스는 각 부서가 배정된 예산을 관리할 때 사용하는 지갑을 나타냅니다. 지갑은 각 부서마다 하나씩 존재하며 같은 부서에 속한 직원들은 동일한 지갑을 공유합니다. 지갑은 사용자 개체에 있는 부서 정보를 토대로 그 부서의 잔고를 구해옵니다.

여기서 한 가지 기억하실 점은 이 클래스에 있는 모든 메서드는 atomic 하게 작동합니다. 즉, 사용자 A가 `Wallet` 의 메서드 중 하나를 호출 중이라면, 사용자 B가 동일한 `Wallet` 에서 메서드를 호출하더라도 A의 메서드 호출이 종료된 후에야 실행됩니다.

A.4.1 생성자

```
public Wallet(User user)
```

- 현재 사용자가 속한 부서의 지갑을 생성합니다.
- 생성자는 다음의 예외를 던질 수 있습니다.
  - `IllegalArgumentException` : 생성자의 매개변수가 올바르지 못할 때 발생
  - `PermanentlyClosedException` : 사용자가 속한 부서를 시스템이 찾지 못할 때 발생(예: 폐지된 부서 소속 컴퓨터에서 이 앱을 실행한 경우)
  - `IllegalAccessException` : 사용자가 그 지갑에 접근 권한이 없을 때 발생. 보통 신입 직원들은 권한이 없습니다.

A.4.2 `public` 메서드

- `public int getAmount()`
  - 현재 잔고를 반환합니다.
- `public boolean deposit(int amount)`
  - 성공적으로 입금하였다면 `true` 를 반환합니다.
  - `amount` 가 0 이하인 경우 `false` 를 반환합니다.
- `public boolean withdraw(int amount)`
  - 성공적으로 출금하였다면 `true` 를 반환합니다.
  - `amount` 가 올바르지 못한 경우 `false` 를 반환합니다.

A.5 `WarehouseType` 클래스

- 각 창고를 나타내는 열거형
- 다음과 같은 열거형 값이 있습니다.
  - `APPLE`
  - `MICROSOFT`
  - `SAMSUNG`

A.6 `Warehouse` 클래스

`Warehouse` 클래스는 창고를 나타냅니다. `Warehouse` 의 생성자가 호출될 때 현재 창고에 있는 모든 장비들을 메모리에 읽어옵니다. 따라서 `getProducts()` 를 호출하면 그 창고에 있는 장비 목록을 얻어올 수 있습니다. `User` 클래스와 마찬가지로 어떻게 그런 일이 일어나는지는 신경 쓰지 마시고, 그냥 믿고 사용하세요.

이 클래스에 있는 모든 메서드들도 atomic 하게 동작합니다.

A.6.1 생성자

```
public Warehouse(WarehouseType type)
```

- `type` 으로 지정된 창고를 생성합니다. 이 창고에 있는 장비 목록이 이 개체에 로딩됩니다.

- 이 생성자는 다음의 예외를 던질 수 있습니다.
  - `IllegalArgumentException` : 생성자 매개변수가 올바르지 않은 경우에 발생
  - `PermanentlyClosedException` : 시스템이 창고를 찾지 못한 경우에 발생

A.6.2 public 메서드

- `public ArrayList<Product> getProducts()`
  - 창고에 있는 모든 장비를 반환합니다.
- `public void removeProduct(UUID id)`
  - `id` 로 지정된 장비를 제거합니다.
  - 장비를 찾을 수 없다면 `ProductNotFoundException` 예외를 던집니다.

A.7 PermanentlyClosedException 클래스

`PermanentlyClosedException` 는 `Warehouse` 나 `Wallet` 의 생성자가 지정된 창고나 부서를 사내 시스템에서 찾을 수 없을 때 던지는 예외 클래스입니다. 창고나 부서가 문을 닫았는데 아직 라이브러리를 업데이트하지 않았을 경우 이런 일이 일어날 수 있습니다.

이 예외로부터 제대로 프로그램을 복구하는 방법은 없습니다. 따라서 그냥 크래시가 나도록 놔두는 게 최선입니다.

A.7.1 생성자

`public PermanentlyClosedException(String message)` : 상세한 메시지와 함께 `PermanentlyClosedException` 개체를 생성합니다.

A.8 ProductNotFoundException 클래스

`ProductNotFoundException` 은 제거하려는 장비가 목록에서 사라졌을 때 발생합니다. 사용자가 구매를 완료하기 전에 다른 직원이 그 장비를 사버릴 때 이런 일이 발생합니다.

A.8.1 생성자

`public ProductNotFoundException(String message)` : 상세한 메시지와 함께 `ProductNotFoundException` 개체를 생성합니다.