COMP2500 실습 10

목차

수업 중에 책임 연쇄 디자인 패턴을 배우면서 체인 안에 있는 각 핸들러가 특정 상황에 대해 책임을 지면 더 이상 다음 핸들러에 책임을 전가하지 않는 모습을 봤습니다. 간단히 말해 이건 `if ... else if ... else if ... else if ... else` 문을 OOP 방식으로 구현하는 방법이죠. 이 실습에서는 이 패턴을 사용하여 좀 더 복잡한 시스템을 만드는 방법을 살짝 엿보겠습니다.

Pocuflix는 간단한 영화 저장소 서비스입니다. 이 서비스를 간단히 설명하면 많은 영화들이 저장소에 저장돼 있고, 사용자 요청에 따라 영화를 반환하는 서비스이죠. 이런 시스템에서 필요한 기능의 몇 가지 예는 다음과 같습니다.

1. 특정 영화를 저장하고 불러올 수 있는 기능
2. 현재 시스템이 점검 중이란 사실 및 점검 종료시간을 사용자에게 알려줄 수 있는 기능
3. 접근권한 관리. 특정 사용자만 이 서비스를 사용할 수 있음
4. 영화 캐싱(caching) 기능. 영화는 용량이 커서 요청이 들어올 때마다 매번 전송하지 않는 게 좋음. 단, 영화 데이터가 바뀐 경우에는 예외

위에서 볼 수 있듯이 각 기능마다 수행하는 로직이 꽤 다르죠? 이런 시스템이 바로 책임 연쇄 디자인 패턴을 사용하기 꽤 적합한 곳이랍니다.

1. 프로젝트를 준비한다

1. `Lab10` 폴더로 이동합니다.
2. 자, 스트레칭 한 번 하시고... 설계와 프로그래밍 시작하겠습니다. :)

2. Pocuflix 시스템 구현하기

전반적인 규칙

- 여러분이 작성하는 클래스들은 반드시 `academy.pocu.comp2500.lab10` 패키지 안에 속해야 합니다.
- `private static final` 멤버 변수 외에 어떤 정적 멤버 변수도 사용할 수 없습니다.
- `academy.pocu.comp2500.lab10.pocuflix` 패키지 안에 있는 파일들을 바꾸지 마세요. 아무리 이 파일들을 변경해도 빌드봇이 무시해버리니 시도조차 하지 마세요. :p
- 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.

2.1 `Request` 클래스 구현하기

이 클래스는 사용자가 시청을 원하는 영화를 반환하라는 요청을 나타내는 클래스입니다. 사용자가 보고 싶은 영화의 제목을 지정하면 시스템이 나머지를 알아서 처리해 주죠. 이때 사용자의 접근권한을 인증해야 하니 이 요청에는 사용자 정보도 들어 있어야 합니다. 일반적으로는 사용자 이름(username)과 비밀번호를 인증에 사용하나, 이건 이 과목의 범위를 넘어서는 것 같네요

2.1.1 생성자를 구현한다

생성자는 다음의 인자를 받습니다.

- 영화의 제목을 나타내는 `String`

```
Request request = new Request("The Lord of the Rings");
```

2.1.2 `setUser()` 메서드를 구현한다

- 이 메서드는 다음의 인자를 받습니다.
  - 현재 요청을 보내는 사용자를 나타내는 `User`
- 인자로 전달된 사용자를 요청의 사용자로 세팅합니다.
- 이 메서드는 아무것도 반환하지 않습니다.

```
Request request = new Request("The Lord of the Rings");
User user = new User("Frodo", "Baggins");

request.setUser(user);
```

2.2 `IRequestHandler` 인터페이스를 선언한다

`IRequestHandler`는 요청을 처리할 수 있는 모든 핸들러(handler)가 구현해야 하는 인터페이스입니다.

- 이 인터페이스에는 하나의 메서드만 존재합니다.
  - `handle()`
- `handle()` 메서드는 다음의 인자를 받습니다.
  - 요청 개체를 나타내는 `Request`
- `handle()` 메서드는 `Request` 개체를 처리하고 `ResultBase` 개체를 반환합니다. 요청을 처리하는 방법은 각 구체(concrete) 핸들러마다 다릅니다.

2.3 `MovieStore` 클래스 구현하기

이 시스템에서 사용자가 원하는 영화를 찾아 반환하려면 우선 그 영화가 저장되어 있어야겠죠? `MovieStore` 클래스가 그런 일을 합니다. :)

또한 `MovieStore`는 요청된 영화를 성공적으로 찾았는지 여부를 알려줄 수 있어야 합니다. 이를 위해 `OkResult`와 `NotFoundResult`를 `academy.pocu.comp2500.lab10.pocuflix` 패키지 안에 정의해 놨으니 확인해보도록 하세요. 이 클래스들은 모두 `ResultBase` 추상 클래스를 상속합니다. 이 외에 다른 핸들러에서 반환할 결과(result)들은 여러분이 직접 구현해야 합니다.

2.3.1 생성자를 구현한다

이 생성자는 아무 인자도 받지 않습니다.

```
MovieStore store = new MovieStore();
```

2.3.2 `add()` 메서드를 구현한다

- 이 메서드는 1개의 인자를 받습니다.
  - 저장소에 추가할 영화를 나타내는 `Movie`
- 인자로 전달된 영화를 저장소에 추가합니다.
- 이 메서드는 아무것도 반환하지 않습니다.

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);
```

2.3.3 remove() 메서드를 구현한다

- 이 메서드는 1개의 인자를 받습니다.
  - 색인(index)을 나타내는 `int`
- 지정된 색인에 위치한 영화를 제거합니다.
- 저장소로부터 영화를 제거하였다면 `true`를, 아니라면 `false`를 반환합니다.

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

store.remove(1); // false
store.remove(0); // true
```

2.3.4 handle() 메서드를 구현한다

- `Request` 개체로 요청한 영화가 존재한다면 `OkResult`를 반환합니다. `OkResult`는 찾은 영화 개체를 포함해야 합니다.
- 만약 영화를 찾지 못했다면 `NotFoundResult` 개체를 반환합니다.

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

Request request = new Request("The Lord of the Rings");

store.handle(request); // OkResult object

request = new Request("Harry Potter");

store.handle(request); // NotFoundResult object
```

2.4 MaintenanceMiddleware 클래스 구현하기

`MaintenanceMiddleware`는 시스템이 점검 중인지 확인합니다. 시스템 점검 시간은 언제나 1시간입니다. 이 클래스는 `ServiceUnavailableResult` 개체를 반환해야 하며, 그 개체 안에는 점검 시작 날짜/시간과 점검 종료 날짜/시간이 들어 있어야 합니다.

2.4.1 생성자를 구현한다

생성자는 다음의 인자를 받습니다.

- 다음 핸들러를 나타내는 `IRequestHandler`
- 점검 날짜/시작 시간(**UTC+0 시간대**)을 나타내는 `OffsetDateTime`

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);

MaintenanceMiddleware middleware = new MaintenanceMiddleware(store, now);
```

2.4.2 ServiceUnavailableResult 클래스 구현하기

- 생성자는 다음의 인자를 받습니다.
  - 점검 시작 날짜/시간(**UTC+0 시간대**)을 나타내는 `OffsetDateTime`
  - 점검 종료 날짜/시간(**UTC+0 시간대**)을 나타내는 `OffsetDateTime`
  - 이 개체의 `ResultCode`는 `SERVICE_UNAVAILABLE`여야 합니다.
- 이 클래스는 생성자의 각 인자를 반환하는 2개의 게터(getter) 메서드를 구현해야 합니다.
  - `getStartDateTime()`
  - `getEndDateTime()`

2.4.3 MaintenanceMiddleware 클래스에 handle() 메서드를 구현한다

- Pocuflix가 점검 중이라면 `ServiceUnavailableResult` 개체를 반환합니다. 아니라면 다음 핸들러에 요청을 넘깁니다.

- 점검 기간은 언제나 1시간이라는 사실을 잊지 마세요!

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);

MaintenanceMiddleware middleware = new MaintenanceMiddleware(store, now);

// sleepSeconds(10)

Request request = new Request("The Lord of the Rings");

middleware.handle(request); // ServiceUnavailableObject
```

2.5 AuthorizationMiddleware 클래스 구현하기

영화 저장소에 권한 없는 접근(unauthorized access)을 허용하면 안 되겠죠? 이런 일을 막기 위해 권한 없는 요청을 처리하는 또 다른 미들웨어를 만들어 보겠습니다. 이 미들웨어는 요청을 보낸 사용자가 허용 목록에 없다면 `UnauthorizedResult` 를 반환합니다.

2.5.1 생성자를 구현한다

생성자는 다음의 인자를 받습니다.

- 다음 핸들러를 나타내는 `IRequestHandler`
- 영화에 접근할 수 있는 사용자 목록을 나타내는 `HashSet`

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

HashSet<User> users = new HashSet<>();
users.add(new User("Frodo", "Baggins"));

AuthorizationMiddleware middleware = new AuthorizationMiddleware(store, users);
```

2.5.2 `UnauthorizedResult` 클래스를 구현한다

- 생성자는 어떤 인자도 받지 않습니다.
- 이 개체의 `ResultCode` 는 `UNAUTHORIZED` 여야 합니다.
- 이 클래스의 유일한 메서드는 다음과 같습니다.
    - `getErrorMessage()` : `Unauthorized access` 라는 메시지를 반환합니다.

2.5.3 `AuthorizationMiddleware` 클래스에 `handle()` 메서드를 구현한다

만약 요청을 보낸 사용자가 허용된 사용자 목록에 들어있지 않다면 `UnauthorizedResult` 개체를 반환합니다. 그렇지 않다면 다음 핸들러에 요청을 넘깁니다.

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

HashSet<User> users = new HashSet<>();
users.add(new User("Frodo", "Baggins"));

AuthorizationMiddleware middleware = new AuthorizationMiddleware(store, users);

Request request = new Request("The Lord of the Rings");
User user = new User("Bilbo", "Baggins");

request.setUser(user);

middleware.handle(request); // UnauthorizedResult

request = new Request("The Lord of the Rings");
User user = new User("Frodo", "Baggins");

request.setUser(user);

middleware.handle(request); // OkResult
```

2.6 CacheMiddleware 클래스 구현하기

보통 영화의 크기는 기가바이트 단위입니다. 사용자가 같은 영화를 여러 번 요청할 때마다 데이터를 전송한다면 전송할 데이터 용량이 너무 커지죠. 이미 과거에 성공적으로 전송한 요청을 알 수 있는 방법이 있다면 다시 데이터를 전송하지 않아도 되겠죠? 사용자는 예전에 받아놓은 영화를 다시 보면 되니까요. 하지만 영화 데이터가 바뀌었다면(예: 자막 등이 바뀜) 그건 다시 받아야 할 겁니다. 이런 일을 처리해 주는 게 `CacheMiddleware` 입니다.

`CacheMiddleware` 는 요청이 들어올 때 이 요청을 이미 예전에 처리해서 영화를 성공적으로 전송했는지를 판단합니다. 만약 그렇다면 `MovieStore` 에서 영화를 찾는 대신 `CachedResult` 개체를 반환합니다. 캐시(cache)는 만료 횟수(expiry count)를 가집니다. 사용자가 동일한 요청을 이 횟수만큼 보내면 캐시가 만료되는 거죠. 캐시가 만료되면 `MovieStore` 에서 그 영화를 찾아 데이터를 다시 전송합니다.

여기서 한 가지 기억할 점! `CacheMiddleware` 는 결과가 `OkResult` 일 경우에만 그 요청을 캐시에 넣어 저장합니다. 이 시스템에서 반환하는 결과 중 영화 개체를 포함하는 것은 `OkResult` 가 유일하니 다른 종류의 결과는 캐시에 넣을 이유가 없죠.

2.6.1 생성자를 구현한다

생성자는 다음의 인자를 받습니다.

- 다음 핸들러를 나타내는 `IRequestHandler`
- 캐시를 만료시키는 요청 수

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

CacheMiddleware middleware = new CacheMiddleware(store, 3);
```

2.6.2 CachedResult 클래스를 구현한다.

- 이 생성자는 1개의 인자를 받습니다.
  - 캐시가 만료되려면 몇 번의 요청이 더 필요한지를 나타내는 `int`
- 이 개체의 `ResultCode` 는 `NOT_MODIFIED` 여야 합니다.
- 이 클래스는 생성자 인자를 반환하는 게터(getter) 메서드를 구현해야 합니다.
  - `getExpiryCount()`

2.6.3 CacheMiddleware 클래스에 handle() 메서드를 구현한다

- 요청이 캐시에 저장되어 있었다면 `CachedResult` 개체를 반환합니다. 그게 아니라면 다음 핸들러에 요청을 보냅니다.
- 이 미들웨어는 결과가 `OkResult` 인 요청만 캐시에 저장합니다.

```
MovieStore store = new MovieStore();
Movie movie = new Movie("The Lord of the Rings", Rating.R, 180);

store.add(movie);

CacheMiddleware middleware = new CacheMiddleware(store, 4);

Request request = new Request("The Lord of the Rings");

middleware.handle(request); // OkResult

request = new Request("The Lord of the Rings");

middleware.handle(request); // CachedResult: 3

request = new Request("The Lord of the Rings");

middleware.handle(request); // CachedResult: 2

request = new Request("The Lord of the Rings");

middleware.handle(request); // CachedResult: 1

request = new Request("The Lord of the Rings");

middleware.handle(request); // OkResult
```

2.7 ResultValidator 클래스 구현하기

여태까지 본 바에 따르면 이 시스템이 반환하는 결과는 다음과 같습니다.

1. `OkResult`

2. `NotFoundResult`
3. `ServiceUnavailableResult`
4. `UnauthorizedResult`
5. `CachedResult`

이들은 모두 `ResultBase` 클래스를 상속하지만, 각 구체 클래스에 들어있는 메서드를 호출하려면 캐스팅이 필요하죠. 캐스팅을 하기 전에 결과 개체들의 실제 데이터 형이 캐스팅을 하려는 클래스하고 일치하는지 검증을 하면 좋겠죠? 이를 위해 간단한 클래스를 만들어 봅시다.

2.7.1 생성자를 구현한다

이 생성자는 1개의 인자를 받습니다.

- 검증할 결과 개체를 나타내는 `ResultBase`

```
ResultBase result = new UnauthorizedResult();

ResultValidator validator = new ResultValidator(result);
```

2.7.2 `isValid()` 메서드를 구현한다

- 이 메서드는 1개의 인자를 받습니다.
  - `ResultCode`
- 생성자에 전달한 결과 개체가 `ResultCode` 인자로 특정된 올바른 유효한 결과라면 `true`를 아니라면 `false`를 반환합니다.

```
ResultBase result = new UnauthorizedResult();

ResultValidator validator = new ResultValidator(result);

validator.isValid(ResultCode.UNAUTHORIZED); // true
validator.isValid(ResultCode.OK); // false
```

3. 본인 컴퓨터에서 테스트하는 법

- `Program.java`를 아래처럼 바꾼 뒤 실행하세요.

```java
package academy.pocu.comp2500.lab10.app;

import academy.pocu.comp2500.lab10.AuthorizationMiddleware;
import academy.pocu.comp2500.lab10.CacheMiddleware;
import academy.pocu.comp2500.lab10.CachedResult;
import academy.pocu.comp2500.lab10.MaintenanceMiddleware;
import academy.pocu.comp2500.lab10.MovieStore;
import academy.pocu.comp2500.lab10.Request;
import academy.pocu.comp2500.lab10.ResultValidator;
import academy.pocu.comp2500.lab10.ServiceUnavailableResult;
import academy.pocu.comp2500.lab10.UnauthorizedResult;
import academy.pocu.comp2500.lab10.pocuflix.Movie;
import academy.pocu.comp2500.lab10.pocuflix.NotFoundResult;
import academy.pocu.comp2500.lab10.pocuflix.OkResult;
import academy.pocu.comp2500.lab10.pocuflix.Rating;
import academy.pocu.comp2500.lab10.pocuflix.ResultBase;
import academy.pocu.comp2500.lab10.pocuflix.ResultCode;
import academy.pocu.comp2500.lab10.pocuflix.User;

import java.time.OffsetDateTime;
import java.time.ZoneOffset;
import java.util.HashSet;
import java.util.concurrent.TimeUnit;

public class Program {
    public static void main(String[] args) {
        MovieStore store = new MovieStore();

        store.add(new Movie("Harry Potter", Rating.PG13, 180));
        store.add(new Movie("The Lord of the Rings", Rating.R, 180));

        assert (!store.remove(2));
        assert (store.remove(1));

        {
            Request request = new Request("None");

            ResultBase result = store.handle(request);

            assert (result.getCode() == ResultCode.NOT_FOUND);
            assert (result instanceof NotFoundResult);

            request = new Request("Harry Potter");

            result = store.handle(request);

            assert (result.getCode() == ResultCode.OK);
            assert (result instanceof OkResult);

            OkResult okResult = (OkResult) result;

            assert (okResult.getMovie().getTitle().equals("Harry Potter"));
            assert (okResult.getMovie().getRating() == Rating.PG13);
            assert (okResult.getMovie().getPlayTime() == 180);
        }

        {
            OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);
            OffsetDateTime startDateTime = now.plusSeconds(5);
            OffsetDateTime endDateTime = startDateTime.plusHours(1);

            MaintenanceMiddleware middleware = new MaintenanceMiddleware(store, startDateTime);

            Request request = new Request("Harry Potter");

            ResultBase result = middleware.handle(request);

            assert (result.getCode() == ResultCode.OK);
            assert (result instanceof OkResult);

            sleep(10);

            request = new Request("Harry Potter");

            result = middleware.handle(request);

            assert (result.getCode() == ResultCode.SERVICE_UNAVAILABLE);
            assert (result instanceof ServiceUnavailableResult);

            ServiceUnavailableResult serviceUnavailableResult = (ServiceUnavailableResult) result;
```

```
        assert (serviceUnavailableResult.getStartDateTime().compareTo(startDateTime) == 0);
        assert (serviceUnavailableResult.getEndDateTime().compareTo(endDateTime) == 0);
    }

    {
        HashSet<User> users = new HashSet<>();
        users.add(new User("Jane", "Doe"));

        AuthorizationMiddleware middleware = new AuthorizationMiddleware(store, users);

        Request request = new Request("Harry Potter");

        ResultBase result = middleware.handle(request);

        assert (result.getCode() == ResultCode.UNAUTHORIZED);
        assert (result instanceof UnauthorizedResult);

        UnauthorizedResult unauthorizedResult = (UnauthorizedResult) result;

        assert (unauthorizedResult.getErrorMessage().equals("Unauthorized access"));

        request = new Request("Harry Potter");
        request.setUser(new User("James", "Bob"));

        result = middleware.handle(request);
        assert (result.getCode() == ResultCode.UNAUTHORIZED);
        assert (result instanceof UnauthorizedResult);

        unauthorizedResult = (UnauthorizedResult) result;

        assert (unauthorizedResult.getErrorMessage().equals("Unauthorized access"));

        request = new Request("Harry Potter");
        request.setUser(new User("Jane", "Doe"));

        result = middleware.handle(request);

        assert (result.getCode() == ResultCode.OK);
        assert (result instanceof OkResult);
    }

    {
        CacheMiddleware middleware = new CacheMiddleware(store, 3);

        Request request = new Request("Harry Potter");

        ResultBase result = middleware.handle(request);

        assert (result.getCode() == ResultCode.OK);
        assert (result instanceof OkResult);

        request = new Request("Harry Potter");

        result = middleware.handle(request);
        assert (result.getCode() == ResultCode.NOT_MODIFIED);
        assert (result instanceof CachedResult);

        CachedResult cachedResult = (CachedResult) result;

        assert (cachedResult.getExpiryCount() == 2);

        request = new Request("Harry Potter");
        request.setUser(new User("Jane", "Doe"));

        result = middleware.handle(request);

        assert (result.getCode() == ResultCode.OK);
        assert (result instanceof OkResult);

        request = new Request("Harry Potter");

        result = middleware.handle(request);

        assert (result.getCode() == ResultCode.NOT_MODIFIED);
        assert (result instanceof CachedResult);

        cachedResult = (CachedResult) result;

        assert (cachedResult.getExpiryCount() == 1);
```

```
        request = new Request("Harry Potter");
        request.setUser(new User("Jane", "Doe"));

        result = middleware.handle(request);

        assert (result.getCode() == ResultCode.NOT_MODIFIED);
        assert (result instanceof CachedResult);

        cachedResult = (CachedResult) result;

        assert (cachedResult.getExpiryCount() == 2);

        request = new Request("Harry Potter");

        result = middleware.handle(request);

        assert (result.getCode() == ResultCode.OK);
        assert (result instanceof OkResult);
    }
    {
        OffsetDateTime now = OffsetDateTime.now(ZoneOffset.UTC);

        ResultBase result = new ServiceUnavailableResult(now, now);

        ResultValidator validator = new ResultValidator(result);
        assert (validator.isValid(ResultCode.SERVICE_UNAVAILABLE));
        assert (!validator.isValid(ResultCode.OK));
        assert (!validator.isValid(ResultCode.NOT_FOUND));
    }
}

    private static void sleep(int seconds) {
        try {
            TimeUnit.SECONDS.sleep(seconds);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

4. 커밋, 푸시 그리고 빌드 요청

이건 어떻게 하는지 이제 다 아시죠? :)