5장. 웹 어댑터 구현하기

웹 어댑터

주도하는 / 인커밍 어댑터

🧩 역할

- 외부로부터 요청을 받아 애플리케이션 코어 호출
- 무슨 일을 해야 할지 알려줌

🕹️ 제어 흐름

웹 어댑터(컨트롤러) → 애플리케이션 계층(서비스)

- 서비스는 포트를 구현하여 웹 어댑터가 통신할 수 있게 한다.
- 의존성 역전 원칙이 적용되었다 !

🧐 어댑터와 유스케이스 사이에 간접 계층을 넣는 이유

- 애플리케이션 코어가 외부 세계와 통신할 수 있는 곳에 대한 명세 = **포트**
- 포트를 적절한 곳에 위치시키면
    - 외부와 어떤 통신이 일어나는지 정확히 알 수 있다.
    - 레거시 코드를 다루기 쉽다.

웹 어댑터의 책임

1. HTTP 요청을 자바 객체로 매핑
2. 권한 검사
3. 입력 유효성 검증
4. 입력을 유스케이스의 입력 모델로 매핑
5. 유스케이스 호출
6. 유스케이스의 출력을 HTTP로 매핑
7. HTTP 응답을 반환

컨트롤러 나누기

웹 어댑터는 한 개 이상의 클래스로 구성할 수 있다 !

☝ AccountController로 계좌 관련 모든 요청 받기

```
package buckpal.adapter.web;

@RestController
@RequiredArgsConstructor
class AccountController {

    private final GetAccountBalanceQuery getAccountBalanceQue
    private final ListAccountsQuery listAccountsQuery;
    private final LoadACcountQuery loadAccountQuery

    private final SendMoneyUseCase sendMoneyUseCase;
    private final CreateAccountUseCase createAccountUseCase;

    @GetMapping("/accounts")
    List<AccountResource> listAccounts() {
        ...
    }
```

```
    @GetMapping("/accounts/{accountId}")
    AccountResource getAccount(@PathVariable("accountId") Lon
        ...
    }

    @GetMapping("/accounts/{accountId}/balance")
    long getAccountBalance(@PathVariable("accountId") Long ac
        ...
    }

    @PostMapping("/accounts")
    AccountResource createAccount(@RequestBody AccountResourc
        ...
    }

    @PostMapping("/accounts/send/{sourceAccountId}/{targetAcc
    void sendMoney(
            @PathVariable("sourceAccountId") Long sourceAccou
            @PathVariable("targetAccountId") Long targetAccou
            @PathVariable("amount") Long amount) {
            ...
    }
}
```

- 클래스마다 코드는 적을수록 좋다.
- 테스트 코드도 마찬가지다.
- 메서드와 클래스명은 유스케이스를 최대한 반영하자.

✌️ SendMoneyController

```
package buckpal.adapter.in.web;

@RestController
@RequiredArgsConstructor
class SendMoneyController {

    private final SendMoneyUseCase sendMoneyUseCase;
```

```
    @PostMapping("/accounts/send/{sourceAccountId}/{targetAcc
    void sendMoney(
            @PathVariable("sourceAccountId") Long sourceAccou
            @PathVariable("targetAccountId") Long targetAccou
            @PathVariable("amount") Long amount) {

            SendMoneyCommand command = new SendMoneyCommand(
                new AccountId(sourceAccountId),
                new AccountId(targetAccountId),
                Money.of(amount));
            sendMoneyUseCase.sendMoney(command);
    }
}
```

- 전용 모델 클래스들은 실수로 다른 곳에서 재사용될 일이 없다.
- 다시 생각해봤을 때, 필드의 절반은 필요없다는 걸 깨닫고 컨트롤러에 맞는 모델을 새로 만들게 될 수 있다.
- 서로 다른 연산에 대한 동시 작업이 쉬워진다.

결론

- 웹 어댑터를 만들 때에는 어떤 도메인 로직도 수행하지 않는 어댑터를 만들고 있다는 점을 염두에 두자 !
- 애플리케이션 계층은 HTTP와 관련된 작업을 해서는 안된다.
- 모델을 공유하지 않는 작은 클래스들을 만드는 것을 두려워하지 마라.
- 작은 클래스는 더 파악하기 쉽고 더 테스트하기 쉬우며 동시 작업을 지원한다.