10장. 람다를 이용한 DSL

프로그래밍 언어도 결국 언어이며 언어는 메시지를 명확하고 안정적인 방식으로 전달하는 것에 있다. 저명한 컴퓨터 과학자 하롤드 아벨슨이 프로그램은 사람들이 이해할 수 있ㄷ조록 작성되어야 하는 것이 중요하며 기기가 실행하는 부분은 부차적일 뿐이라 말한 것 처럼 의도가 명확하게 전달되어야 한다.

도메인 전용 언어로 애플리케이션의 비즈니스 로직을 표현함으로써 제대로 구현되었는지를 도메인 전문가가 좀더 쉽게 이해할 수 있다.

메이븐, 앤트, 그래들은 빌드 과정을 표현하는 DSL로 간주할 수 있고, HTML은 웹페이지의 구조를 정의하도록 특화된 언어이다.

DSL의 장점

- 간결함 : API는 비즈니스 로직을 간편하게 캡슐화하므로 반복을 피할 수 있고 코드를 간결하게 만들 수 있다.
- 가독성 : 도메인 영역의 용어를 사용하므로 비 도메인 전문가도 코드를 쉽게 이해할 수 있다. 다양한 조직 구성원 간에 코드와 도메인 영역이 공유될 수 있다.
- 유지보수 : 잘 설계된 DSL로 구현한 코드는 쉽게 유지 보수하고 바꿀 수 있다.
- 높은 수준의 추상화 : DSL은 도메인과 같은 추상화 수준에서 동작하므로 도메인의 문제와 직접적으로 관련되지 않은 세부 사항을 숨긴다.
- 집중 : 비즈니스 도메인의 규칙을 표현할 목적으로 설계된 언어이므로 프로그래머가 특정 코드에 집중할 수 있다.
- 관심사 분리(SoC) : 지정된 언어로 비즈니스 로직을 표현함으로 애플리케이션의 인프라 구조와 관련된 문제와 독립적으로 비즈니스 관련된 코드에서 집중하기가 용이하다.

DSL의 단점

- DSL 설계의 어려움 : 간결하게 제한적인 언어에 도메인 지식을 담는 것이 쉬운 작업은 아니다.
- 개발 비용 : 코드에 DSL을 추가하는 작업은 초기 프로젝트에 많은 비용과 시간이 소모된다. 또한 DSL 유지보수와 변경은 프로젝트에 부담을 주는 요소다.

- 추가 우회 계층 : DSL은 추가적인 계층으로 도메인 모델을 감싸며 이때 계층을 최대한 작게 만들어 성능 문제를 회피한다.
- 새로 배워야 하는 언어 : DSL을 프로젝트에 추가하면서 팀이 배워야 하는 언어가 한 개 더 늘어난다는 부담이 있다.
- 호스팅 언어 한계 : 일부 자바 같은 범용 프로그래밍 언어는 장황하고 엄격한 문법을 가졌다. 이런 언어로는 사용자 친화적 DSL을 만들기가 힘들다.

VM에서 이용할 수 있는 다른 DSL 해결책

내부 DSL

내부 DSL이란 자바로 구현한 DSL을 의미한다. 이전과 달리 람다 표현식이 등장하며 유연성과 표현성이 좋고 장황하지 않은 DSL을 만들 수 있게 되었다. 책에서는 익명 클래스와 람다의 코드를 비교하며 신호대비 잡음이 적다라고 표현했다.

정리하자면 내부 DSL이 가지는 장점은 아래와 같다.

- 기존에 자바를 사용하던 개발자면 별도의 패턴과 기술을 배워 DSL을 구현하는 노력이 현저히 줄어든다.
- DSL 로직과 다른 자바 코드를 함께 컴파일 하고 실행할 수 있어 제3의 도구를 만들지 않아도 된다.
- 새로운 언어를 배우거나 외부 도구를 사용하지 않아도 된다.
- DSL 사용자는 자바 IDE를 이용해 자동 완성과 리팩토링 기능을 그대로 사용할 수 있어 익숙한 도구로 계속해서 개발할 수 있다.

다중 DSL

요즘 JVM에서 실행되는 언어는 100개가 넘는데 JVM 기반 프로그래밍 언어를 사용하면 좀 더 문법적인 잡음이 적은 DSL을 개발할 수 있다. 책에서는 자바보다 스칼라언어가 더욱 간결함을 강조하며 DSL 개발시 더욱 편할 수 있음을 강조했다.

다만, 다른 JVM 기반 프로그래밍 언어를 사용함에 있어 수반되는 불편함도 있다.

- 해당 JVM 기반 프로그래밍 언어에 대해 깊은 이해가 있는 인력이 필요하다.
- 엄밀히 다른 언어가 혼재하므로 여러 컴파일러로 소스를 빌드하도록 빌드 과정의 개선이 필요하다.

- JVM에서 실행하는 언어들이 자바와 백 퍼센트 호환을 주장하고 있지만 그렇지 않을 때가 많다. 그 호환성 때문에 성능이 손실되는 경우도 있다. 스칼라와 자바의 경우에도 호환을 위해 코드 변환이 일부 필요할 수 있다.

외부 DSL

독자적인 문법과 구문을 가지는 해당 도메인을 위한 언어를 개발하는 것을 말한다. 새 언어를 파싱하고, 파서의 결과를 분석하고 외부 DSL을 실행할 코드의 구현이 필요하다. 코드 작업이 아주 클 것이다.

이러한 고생으로 얻게되는 장점은 다음과 같다.

- 무한하고 광대한 자유도
- 니즈에 완벽히 일치하는 언어 개발 가능

최신 자바 API의 작은 DSL

자바의 새로운 기능의 장점을 적용한 첫 API는 네이티브 자바 API 자신이다.

- 스트림 API는 컬렉션을 조작하는 DSL
- 데이터를 수집하는 DSL인 Collectors

DSL 구현 방법

- 메서드 체인: 메소드가 계속해서 객체 자기자신을 반환함으로써 계속해서 체인으로 이어지게끔 함. 유창함을 읨하는 **플루언트 스타일** 이라고도 부른다.

```
Order order = forCustomer( "BigBank" )
        .buy( 80 )
        .stock( "IBM" )
        .on( "NYSE" )
        .at( 125.00 )
        .sell( 50 )
        .stock( "GOOGLE" )
        .on( "NASDAQ" )
        .at( 375.00 )
        .end();
```

- 중첩된 함수: 다른 함수 안에 함수를 이용해 도메인 도델을 만드는 것

```
Order order = order("BigBank",
                    buy(80,
                        stock("IBM", on("NYSE")), at(125.00)),
                    sell(50,
                        stock("GOOGLE", on("NASDAQ")), at(375.00))
                   );
```

- 함수 시퀀싱: 특정 함수가 또 다른 함수를 호출하는 식으로 DSL을 구성함.

```
Order order = order( o -> {
    o.forCustomer( "BigBank" );
    o.buy( t -> {
        t.quantity( 80 );
        t.price( 125.00 );
        t.stock( s -> {
            s.symbol( "IBM" );
            s.market( "NYSE" );
        });
    });
    o.sell( t -> {
        t.quantity( 50 );
        t.price( 375.00 );
        t.stock( s -> {
            s.symbol( "GOOGLE" );
            s.market( "NASDAQ" );
        });
    });
});
```

- 각 구현 방법의 장·단점

패턴 이름	장점	단점
메서드 체인	• 메서드 이름이 키워드 인수 역할을 한다. • 선택형 파라미터와 잘 동작한다. • DSL 사용자가 정해진 순서로 메서드를 호출하도록 강제할 수 있다. • 정적 메서드를 최소화하거나 없앨 수 있다. • 문법적 잡음을 최소화한다.	• 구현이 장황하다. • 빌드를 연결하는 접착 코드가 필요하다. • 들여쓰기 규칙으로만 도메인 객체 계층을 정의한다.
중첩 함수	• 구현의 장황함을 줄일 수 있다. • 함수 중첩으로 도메인 객체 계층을 반영한다.	• 정적 메서드의 사용이 빈번하다. • 이름이 아닌 위치로 인수를 정의한다. • 선택형 파라미터를 처리할 메서드 오버로딩이 필요하다.
람다를 이용한 함수 시퀀싱	• 선택형 파라미터와 잘 동작한다. • 정적 메서드를 최소화하거나 없앨 수 있다. • 람다 중첩으로 도메인 객체 계층을 반영한다. • 빌더의 접착 코드가 없다.	• 구현이 장황하다. • 람다 표현식으로 인한 문법적 잡음이 DSL에 존재한다.