chapter 5. 진화하기

언어 타입별 코드 재사용을 위한 특징

함수형 언어에서의 코드 재사용은 객체지향 언어와는 접근 방법이 다르다.

- 객체지향언어: 클래스에 종속된 메소드를 만드는 것을 권장하고 반복되는 패턴을 재사용하려 함.
- 함수형 언어: 자료구조에 대해 공통된 변형 연산을 적용하고, 특정 경우에 맞춰 주어진 함수를 사용하여 작업을 구성함으로써 재사용하려 함.

5.1 적은 수의 자료구조, 많은 연산자

- OOP세상에서는 특정 메소드가 장착된 자료구조를 개발자가 만들기를 권장하나, 함수형 프로그래밍에서는 몇몇 주요 자료구조(list, set, map)와 자료구조에 따른 최적화된 연산을 통한 재사용을 선호한다.
- 따라서 주요 자료구조를 다루는 방법이 다양하다.

- 이러한 방식은 커스텀 클래스 구조를 만드는 것보다, 함수 수준의 캡슐화를 통해 좀더 세밀하고 기본적인 수준에서 재사용이 가능해진다.

5.2 문제를 향하여 언어를 구부리기

- 복잡한 비즈니스 문제를 자바와 같은 언어를 통해 프로그래밍 또는 맞추는 것보다 , 유연한 언어를 통해 비즈니스 문제에 어울리게 언어를 조정해라.

5.3 디스패치 다시 생각하기

- 디스패치란 넓은 의미로 언어가 작동방식을 동적으로 선택하는 것을 의미

- 여기서는 함수형 언어별로 디스패치 방식이 어떻게 자바보다 간결함과 유연성을 가능하게 하는지 파악.

5.3.1 그루비로 디스패치 개선하기

- 자바에서 조건부실행의 경우 조건문이 길어질 경우 가독성이 떨어지게 되므로, 개발자들은 팩토리나 추상팩토리 패턴을 통해 이를 해결한다.

- 그루비에서는 자바의 switch와 유사하나 동적자료형을 받을 수 있으므로 보다 강력한 switch문 기능을 제공한다.

- if문의 연속사용과 팩토리 패턴의 중간 정도로 생각하고 간편하게 사용이 가능하고 범위나 복합자료형이 가능하므로 스칼라의 패턴 매칭과 유사함.

5.3.2 클로저 언어 구부리기

- 클로저 같은 리스프 계열의 언어에서는 개발자가 언어를 문제에 맞게 변형이 가능해지므로,

- 결국 언어 설계자와 그 언어를 사용하는 개발자가 만들수 있는 것들의 경계가 불분명해지게 된다.

5.3.3 클로저의 멀티 메서드와 맞춤식 다형성

- 자바상의 팩터리 패턴의 경우 다형성을 이용하여 구현되어, 동적으로 실행이 가능하다.

- 클로저 역시 다형성을 지원한다. 다형성을 상속과 분리하면 강력하고 상황에 맞는 디스패치 방식이 가능해진다.

- 이를 통해 자바의 다형성만큼 상황에 맞으면서도 제약은 훨씬 적은 강력한 디스패치 방식을 구현할 수 있다.

5.4 연산자 오버로딩

- 함수형 언어의 공통적인 기능은 연산자 오버로딩으로, 이것은 +, -, *와 같은 연산자를 새로 정의하여 새로운 자료형에 적용하고 새로운 행동을 하게 하는 기능이다.

5.4.1 그루비

- 그루비의 경우 연산자들을 메소드 이름에 자동으로 매핑하는 연산자 오버로딩을 허용한다.
- 일례로 정수 클래스에서 + 연산자를 오버로딩하려면 plus() 메소드를 오바라이딩하면 된다.

5.4.2 스칼라

- 스칼라는 연산자와 메서드의 차이점을 없애는 방법으로 연산자 오버로딩을 허용한다.

- 즉 연산자는 특별한 이름을 가진 메소드에 불과하다. 따라서 곱셈 연산자를 오버라이드하려면 * 메소드를 오버라이드 하면된다.

- 예제 5-12의 toString()의 경우 선언문대신에 표현을 사용하는 함수형 언어의 전형을 보여준다.

5.5 함수형 자료구조

- 자바에서는 언어 자체의 예외 생성 및 전파 기능을 사용하는 전통적인 방법으로 오류를 처리한다.

- 그러나 함수형 언어들은 예외 패러다임을 지원하지 않기 때문에 다른 방법으로 오류를 처리해야 한다.

예외 패러다임이 없는 이유:

- 1. 부수효과가 발생한다. 예외 발생시 예외적인 프로그램 흐름이 야기되므로, 함수형 언어들은 오류를 나타내는 리턴값에 반응하는 것을 선호한다.

- 2. 호출하는 입장에서는 값을 호출하든, 함수를 호출하든 다를바가 없어야 한다. 그러나 예외가 발생할 경우 호출하는 입장에서는 값을 함수로 대체할 수 없을 것이다.

5.5.1 함수형 오류 처리

- 자바에서 만약 예외를 사용하지 않고 오류를 처리하려면, 메서드가 하나의 값만 리턴할 수 있다는 제약을 풀어야 한다.

- 물론 메서드는 여러개의 값을 지닌 하나의 개체 그 서브 클래스의 참조를 통해 리턴할 수 있다.

- Map을 통해 이러한 경우를 가정하면, 발생하는 문제점은 다음과 같다.

  1. Map에 들어가는 값이 타입 세이프 하지 않기 때문에 컴파일러가 오류를 잡아낼 수 없다.

  2. 메서드 호출자는 리턴값을 가능한 결과들과 실제로 비교하기 전에는 성패를 알 수 없다.

- 예외없이 프로그래밍할 경우 필요한 것은 타입 세이프하게 둘 또는 더 많은 값을 리턴해주는 방법이 필요하다.

5.5.2 Either 클래스

- 함수형 언어에서 다른 두 값을 리턴해야 하는 경우가 종종 있어야 하는데, 그런 행동을 모델링하는 자료구조가 Either 클래스이다.

- Either 클래스는 왼쪽 또는 오른쪽 값 중 하나만 가질수 있게 설계되었다. 이런 자료구조를 분리합집합(disjoint union)이라 한다.

- 주로 Either 클래스는 오류 처리에 주로 사용된다. 함수형의 보편적인 관례에 따라 Either 클래스의 왼쪽이 예외, 오른쪽이 결과값으로 구성된다.

로마숫자 파싱

- 제네릭을 통해 함수형 자바에서 Either 클래스를 구현하면 Map 자료구조를 주고받던 것에 비해 타입 세이프티를 지킬수 있고, 제네릭을 통한

- 메서드 선언으로 오류를 분명하게 알 수 있다. 더불어 이런 우회를 통해 게으름(laziness)가 구현이 가능해진다.

게으른 파싱과 함수형 자바

- 함수형 자바 프레임워크에서는 Either 클래스와 함수형 자바의 P1 클래스를 조합해서 사용하면 게으른 오류평가를 구현할 수 있다.

- P1: 매개변수가 없는 _1()란 간단한 메서드의 래퍼로, 코드 블록을 실행하지 않고 여기저기 전달해 원하는 컨텍스트에서 실행하게 해주는 일종의 고계함수

디폴트 값을 제공하기

- Either를 예외 처리에 사용하여 얻는 이점은 게으름만이 아니다. 디폴트 값을 제공한다는 것이 다른 이점이다

예외 조건을 래핑하기

- Either를 사용하면 예외를 래핑하여 구조적인 예외 처리를 함수형으로 변환도 가능하다.

- 물론 자바는 이런 개념을 근본적으로 포함하지 않기 때문에 Either 클래스를 모델링하기가 번거롭다. 스칼라나 그루비는 리턴값을 쉽게

- 생성할수 있는 동적 타이핑 언어이기 때문에 Either 같은 것을 굳이 기본적으로 포함하지 않아도 된다.

- 클로저의 경우 굳이 Either같은 것을 생성하기 보다는, 클로저의 상수 문자열인 키워드를 리턴하는 방법을 주로 사용하여 처리.

5.5.3 옵션 클래스

- Either는 두 부분을 가진 값을 간편하게 리턴할 수 있는 개념이다. 여러 언어의 경우 이와 비슷한 자료구조로 option이란 클래스를 제공한다.

- 이것은 적당한 값이 존재하지 않을 경우 none, 존재할 경우 some을 사용하여 예외 조건을 쉽게 표현한다. 결국 Either의 부분집합

5.5.4 Either 트리와 패턴매칭

- 트리모양의 구조물을 만들면서 Either를 자세히 살펴보자. 먼저 스칼라의 패턴매칭과 유사한 순회메소드를 구현한다.

스칼라 패턴 매칭

- match를 통해 매개변수의 자료형 및 주어진 조건에 매칭되는 값을 얻게 해준다. 보통 스칼라의 case class와 같이 사용되며 이를 통해 순차적인 결정과정을 제거해주는 특성을 가진다.

스칼라 케이스 클래스

- 자료만 가진 간단한 클래스들을 위한 간단한 클래스. 다음과 같은 장점이 있다.

1. 클래스 이름을 사용한 팩토리 클래스를 만들수 있다.

2. 클래스의 모든 변수들이 자동적으로 val(불변형 내부 변수)이 된다.

3. 컴파일러가 적당한 디폴트 equals(), hashcode(), toString() 메서드를 자동으로 생성해 준다.

4. 컴퍼일러가 var(가변객체)를 지원하기 위해, 새로운 복제 객체를 리턴하는 copy() 메서드를 자동으로 생성해 준다.

Either 트리

- 이 Either 트리 구조는 내부적으로 <Either, <Left, Node>>를 바탕으로 패턴 매칭을 순회하여 모든 요소를 순회할 수 있다.

패턴매칭으로 트리 순회하기

- 트리의 내부 구조를 규격화한 덕분에 트리를 따라가면서 각 요소의 자료형에 따른 경우에 대해서만 생각하며 분석이 가능하고

- 스칼라의 패턴 매칭처럼 표현이 풍부하지는 않지만, 놀랍게도 스칼라의 표현과 흡사하게 구현된다.