ValueObject

2016 년 11 월 14 일

마틴 파울러

도메인 중심 설계
API 설계

프로그래밍 할 때 나는 종종 사물을 화합물로 표현하는 것이 유용하다는 것을 알게 됩니다. 2D 좌표는 x 값과 y 값으로 구성됩니다. 금액은 숫자와 통화로 구성됩니다. 날짜 범위는 시작 날짜와 종료 날짜로 구성되며 그 자체가 연도, 월, 일의 복합 일 수 있습니다.

이렇게하면 두 개의 복합 객체가 동일한 지에 대한 질문에 부딪 힙니다. 둘 다 (2,3)의 데카르트 좌표를 나타내는 두 개의 점 개체가있는 경우이를 동일하게 취급하는 것이 합리적입니다. 속성 값으로 인해 동일한 객체 (이 경우 x 및 y 좌표)를 값 객체라고합니다.

하지만 프로그래밍 할 때주의하지 않으면 내 프로그램에서 그 동작을 얻지 못할 수 있습니다.

자바 스크립트로 포인트를 표현하고 싶다고 해보자.

```
const p1 = {x : 2, y : 3};
const p2 = {x : 2, y : 3};
assert (p1! == p2); // 내가 원하는 것이 아님
```

슬프게도 그 테스트는 통과했습니다. JavaScript는 js 객체에 포함 된 값을 무시하고 참조를 살펴봄으로써 동등성을 테스트하기 때문에 그렇게합니다.

많은 상황에서 값보다는 참조를 사용하는 것이 합리적입니다. 여러 판매 주문을로드하고 조작하는 경우 각 주문을 단일 위치에로드하는 것이 좋습니다. 그런 다음 Alice의 최근 주문이 다음 배송에 있는지 확인해야하는 경우 Alice의 주문에 대한 메모리 참조 또는 ID를 가져와 해당 참조가 배달의 주문 목록에 있는지 확인할 수 있습니다. 이 테스트에서는 순서에 대해 걱정할 필요가 없습니다. 마찬가지로 고유 한 주문 번호를 사용하여 Alice의 주문 번호가 배송 목록에 있는지 테스트합니다.

따라서 두 가지 객체 클래스, 즉 값 객체와 참조 객체를 구분하는 방법에 따라 생각하는 것이 유용하다는 것을 알았습니다 [1] . 각 개체가 평등을 처리하고 내 기대에 따라 작동하도록 프로그래밍하는 방법을 알고 있는지 확인해야합니다. 그 방법은 내가 일하는 프로그래밍 언어에 따라 다릅니다.

일부 언어는 모든 복합 데이터를 값으로 취급합니다. Clojure에서 간단한 화합물을 만들면 이렇게 생겼습니다.

```
> (= {: x 2, : y 3} {: x 2, : y 3})
진실
```

이것이 바로 기능적 스타일입니다. 모든 것을 불변의 값으로 취급합니다.

하지만 기능적 언어가 아니더라도 가치 객체를 자주 생성 할 수 있습니다. 예를 들어 Java에서 기본 포인트 클래스는 내가 원하는 방식으로 작동합니다.

```
assertEquals (new Point (2, 3), new Point (2, 3)); // 자바
```

이것이 작동하는 방식은 포인트 클래스 equals 가 값에 대한 테스트로 기본 메서드를 재정의하는 것 입니다. [2] [3]

JavaScript에서도 비슷한 일을 할 수 있습니다.

```
class Point {
  constructor (x, y) {
    this.x = x;
    this.y = y;
  }
  같음 (기타) {
    return this.x === other.x && this.y === other.y;
  }
}
```

```
const p1 = new Point (2,3);
const p2 = new Point (2,3);
assert (p1.equals (p2));
```

여기서 JavaScript의 문제는 이것이 내가 정의한 메소드와 같음이 다른 JavaScript 라이브러리에 대한 미스터리라는 것입니다.

```
const somePoints = [new Point (2,3)];
const p = new Point (2,3);
assert.isFalse (somePoints.includes (p)); // 내가 원하는 것이 아님

// 그래서 나는 이것을해야한다
assert (somePoints.some (i => i.equals (p)));
```

Object.equals코어 라이브러리에 정의되어 있고 다른 모든 라이브러리가 비교를 위해 사용 하므로 Java에서는 문제가되지 않습니다 (==일반적으로 기본 요소에만 사용됨).

값 객체의 좋은 결과 중 하나는 메모리에 동일한 객체에 대한 참조가 있는지 또는 동일한 값을 가진 다른 참조가 있는지에 대해 신경 쓸 필요가 없다는 것입니다. 그러나 만약 내가 행복한 무지가 문제로 이어질 수 있다는 것을주의하지 않는다면, 나는 약간의 자바로 설명 할 것이다.

```
Date retirementDate = new Date (Date.parse ( "Tue 1 Nov 2016"));

// 이것은 은퇴 파티가 필요하다는 것을 의미합니다
날짜 partyDate = retirementDate;

//하지만 그 날짜는 화요일 이니 주말에 파티하자
partyDate.setDate (5);

assertEquals (new Date (Date.parse ( "Sat 5 Nov 2016")), retirementDate);
// 죄송합니다. 이제 3 일 더 일해야합니다. :-(
```

이것은 Aliasing Bug 의 예입니다. 한 곳에서 날짜를 변경하면 예상했던 것 이상의 결과가 발생합니다 [4] . 앨리어싱 버그를 피하기 위해 간단하지만 중요한 규칙을 따릅니다. 값 객체는 불변이어야합니다 . 파티 날짜를 변경하려면 대신 새 개체를 만듭니다.

```
Date retirementDate = new Date (Date.parse ( "Tue 1 Nov 2016"));
날짜 partyDate = retirementDate;

// 날짜를 불변으로 취급
partyDate = new Date (Date.parse ( "2016 년 11 월 5 일 토요일"));

// 그리고 나는 여전히 화요일에 은퇴합니다.
assertEquals (new Date (Date.parse ( "Tue 1 Nov 2016")), retirementDate);
```

물론 값 객체가 실제로 변경 불가능한 경우 변경 불가능한 것으로 처리하는 것이 훨씬 쉽습니다. 객체를 사용하면 일반적으로 설정 방법을 제공하지 않음으로써이를 수행 할 수 있습니다. 그래서 저의 이전 JavaScript 클래스는 다음과 같습니다. [5]

```
class Point {
  constructor (x, y) {
    this._data = {x : x, y : y};
  }
  get x () {return this._data.x;}
  get y () {return this._data.y;}
  같음 (기타) {
    return this.x === other.x && this.y === other.y;
  }
}
```

불변성은 앨리어싱 버그를 피하기 위해 제가 가장 좋아하는 기술이지만 할당이 항상 복사본을 만들도록하여 버그를 피할 수도 있습니다. C #의 구조체와 같은 일부 언어는이 기능을 제공합니다.

개념을 참조 객체 또는 값 객체로 처리할지 여부는 컨텍스트에 따라 다릅니다. 많은 상황에서 우편 주소는 가치가 같은 단순한 텍스트 구조로 취급 할 가치가 있습니다. 그러나 더 정교한 매핑 시스템은 우편 주소를 참조가 더 의미있는 정교한 계층 모델에 연결할 수 있습니다. 대부분의 모델링 문제와 마찬가지로 상황에 따라 솔루션이 달라집니다. [6]

문자열과 같은 일반적인 프리미티브를 적절한 값 객체로 바꾸는 것이 종종 좋은 생각입니다. 전화 번호를 문자열로 표현할 수 있지만, 전화 번호 객체로 바꾸면 변수와 매개 변수가 더 명확 해지고 (언어가 지원할 때 유형 검사를 통해), 유효성 검사에 자연스럽게 초점을 맞추고, 적용 할 수없는 동작을 피할 수 있습니다 (예 : 정수 ID 번호).

포인트, 돈 또는 범위와 같은 작은 개체는 값 개체의 좋은 예입니다. 그러나 더 큰 구조는 개념적 정체성이 없거나 프로그램에 대한 공유 참조가 필요하지 않은 경우 값 객체로 프로그래밍 할 수 있습니다. 이것은 불변성을 기본으로하는 기능적 언어와 더 자연스럽게 맞습니다. [7]

나는 가치있는 물건들, 특히 작은 물건들은 종종 간과되는 것을 발견했다.-생각할 가치가없는 너무 사소한 것으로 보인다. 그러나 일단 좋은 가치 객체 세트를 발견하면 그에 대한 풍부한 행동을 만들 수 있음을 알게됩니다. 이것을 맛보기 위해 Range 클래스를 사용 해보고 더 풍부한 동작을 사용하여 시작 및 끝 속성으로 모든 종류의 중복 조작을 방지하는 방법을 확인하십시오. 나는 종종 이와 같은 도메인 별 값 객체가 리팩토링의 초점으로 작용하여 시스템을 대폭 단순화 할 수있는 코드베이스를 접하게됩니다. 이러한 단순화는 종종 사람들을 놀라게하는데, 몇 번봤을 때까지는 좋은 친구가됩니다.

감사의 말

James Shore, Beth Andres-Beck 및 Pete Hodgson이 JavaScript에서 값 객체를 사용한 경험을 공유했습니다.

Graham Brooks, James Birnie, Jeroen Soeters, Mariano Giuffrida, Matteo Vaccari, Ricardo Cavalcanti 및 Steven Lowe는 내부 메일 링리스트에 귀중한 의견을 제공했습니다.

추가 읽기

Vaughn Vernon의 설명은 아마도 DDD 관점에서 가치 객체에 대한 가장 심층적 인 논의 일 것입니다 . 그는 값과 엔터티, 구현 팁 및 값 개체를 유지하기위한 기술을 결정하는 방법을 다룹니다.

이 용어는 초기 Noughties에서 관심을 끌기 시작했습니다. 그 당시부터 그들에 대해 이야기하는 두 권의 책은 PoEAA 와 DDD 입니다. Ward 's Wiki 에 대한 흥미로운 토론도있었습니다 .

용어 혼란의 원인 중 하나는 세기가 바뀌면서 일부 J2EE 문헌에서 데이터 전송 객체에 "가치 객체"를 사용했다는 것 입니다. 그 사용법은 지금까지 거의 사라졌지 만, 당

신은 그것에 부딪 칠 수 있습니다.

메모

1 : Domain-Driven Design에서 Evans 분류 는 가치 객체와 엔티티를 대조합니다. 엔티티를 참조 객체의 일반적인 형태로 간주하지만, 참조 / 값 객체 이분법은 모든 코드에 유용하지만 도메인 모델 내에서만 "엔티티"라는 용어를 사용합니다.

2 : 엄격히 awt.geom.Point2D에서 수행되며, awt.Point의 수퍼 클래스입니다.

3 : Java에서 대부분의 객체 비교는 함께 수행됩니다 . `equals`이 연산자는 equals 연산자 대신 사용해야한다는 것을 기억해야하기 때문에 약간 어색 `==`합니다. 이것은 성가신 일이지만 String이 같은 방식으로 작동하기 때문에 Java 프로그래머는 곧 익숙해집니다. 다른 OO 언어는 이것을 피할 수 있습니다. Ruby는 `==`연산자를 사용하지만 재정의 할 수 있습니다.

4 : 자바 -8 이전 날짜 및 시간 시스템의 최악의 기능에 대한 경쟁이 치열합니다.하지만 제 투표는 이것입니다. 고맙게도 우리는 Java 8의 `java.time`패키지 를 사용하여 이 대부분을 피할 수 있습니다.

5 : 클라이언트가 `_data` 속성을 조작 할 수 있기 때문에 이것은 엄격하게 불변이 아닙니다 . 그러나 적절하게 훈련 된 팀은 실제로 불변으로 만들 수 있습니다. 팀이 충분히 훈련받지 못할 까봐 걱정된다면 `freeze`. 실제로 간단한 JavaScript 객체에서 freeze를 사용할 수 있지만 선언 된 접근자가있는 클래스의 명시 성을 선호합니다.

6 : Evans의 DDD 책 에서 이에 대한 더 많은 논의가 있습니다 .

7 : 불변성은 참조 객체에도 중요합니다. 판매 주문이 get 요청 중에 변경되지 않으면 불변으로 만드는 것이 중요합니다. 유용하다면 복사해도 안전합니다. 그러나 고유 주문 번호를 기반으로 동등성을 결정하는 경우 판매 주문이 가치 대상이되지는 않습니다.

번역 : 중국어