Item 1: Consider static factory methods intead of constructors

개요

이 방법은 객체를 획득할 때 constructors 대신 static factory method를 사용하라는 것입니다. 아래의 Boolean 예처럼 말이죠

```
public static Boolean valueOf(boolean b) {
        return b ? Boolean.TRUE : Boolean.FALSE;
    }
```

> 주의해야할 것은 여기서 설명하는 static factory methods는 Factory Method pattern과 직접적인 연관이 없습니다.

그렇다면 static factory method를 제공하는 것은 어떤 장점이 있을까요?

장점

1. 이름을 가지고 있습니다.

1-1) 잘 지어진 static factory는 읽기 쉽고 client code를 읽기 쉽게 해줍니다.

예를 들어 확률적으로 소수를 내놓는 BigInteger를 object를 return하는 것을 생각해봅시다.

```
BigInteger(int, int, Random)
```

이 보다는

```
BigInteger.probablePrime(int, int, Random)
```

이 것이 어떤 object를 생성하는 것인지 이름에서 설명이 잘 되어있죠.

1-2) 그리고 이름을 가지고 있음으로, 동일한 signature로 여러가지 method를 생성할 수 있습니다.

constructor만으로 동일한 signature를 가지고 있지만 다른 작업을 하고 싶다면 단지 parameter 순서만을 바꿔서 구현해야겠지요. 이건 매우 나쁜 코딩으로 볼 수 있습니다.

static factory를 이용한다면 각각의 작업에 맞는 이름을 정해주면 될 뿐입니다.

2. 호출될 때마다 새로운 object를 생성할 필요가 없습니다.

2-1) 불변 class가 이미 생성된 instance를 사용하도록 하거나 cache instance를 사용할 수 있도록 하거나 그리고 중복된 objects를 생성하는 것을 막을 수 있습니다.

`Boolean.valueOf(boolean)` method가 위 테크닉을 설명해주죠. 이것은 절때 객체를 생성하지 않습니다. Flyweight Pattern과 유사하기도 합니다.

2-2) 반복된 호출로부터 동일한 object를 반환시키는 능력은 instnace-controlled class로 만들어 줍니다.

instance-controlled class란 instance가 존재하는 동안 엄격하게 통제하는 class를 의미합니다. 이 class를 만드는 이유는 몇가지가 있습니다.

1. single-ton이거나 nonininstantiable임을 보장하기 위해서
2. immutable value class가 두 개의 동일한 instance가 없음을 보장하기 위해서 입니다.
    - 이는 Flyweight Pattern의 본질이 되는 요소입니다.
    - Enum types가 위와 같은 보장을 제공해주기도 합니다.

3. static factory return type의 sub type object를 반환할 수 있습니다.

이는 반환될 object를 선택함에 매우 큰 유연성을 제공해줍니다.

Collections frame work에서 인터페이스를 사용하게 하고 해당 인터페이스의 하위 클래스를 반환시키도록 함으로써 익혀야 할 API의 수를 줄일 수 있었습니다. 또한 새로운 알고리즘 추가시 해당 interface를 구현하게 함으로써 추가시킬 수도 있지요.

4. input parameters에 따라 obejct의 클래스를 호출될 때마다 다르게 반환시킬 수 있습니다.

EnumSet class는 public constructor가 없고 오직 static factories만 가지고 있습니다. enum type의 크기에 다라 OpenJDK 구현에서는 두 subclass 중 하나만을 반환하도록 합니다. 좀 더 구체적으로 말하자면 만약 64개 이하라면 RegularEnumSet을 반환하며 그 초과라면 JumboEnumSet instance를 반환하도록 하는 것이죠.

위 두 구현 클래스는 client에게 보이지 않습니다. 따라서 추후 RegularEnumSet을 없애거나, 세번째 네번째 EnumSet의 구현을 제공하더라도 client가 알 필요도 고려할 필요도 없게 되죠.

5. static method를 담고 있는 class가 작성될 때 반환될 object의 class가 존재할 필요가 없습니다.

이렇게 유연한 static factory method는 service provider frameworks의 기초가 됩니다. JDBC처럼 말이죠.

service provider frameworks란?

provider가 service를 구현하고, 시스템은 client가 구현된 코드를 사용할 수 있게 합니다. 이를 통해 client가 구현으로부터 decoupling할 수 있도록 도와주죠.

service provider framework의 세가지 구성요소

1. service interface

   이는 implementation을 나타냅니다.

2. provider registration API

   provider가 register implentations를 사용할 수 있도록 합니다.

3. service access API

   client가 service의 instance를 얻기 위해서 사용합니다.

service access API는 client가 특정 구현을 선택하기 위해 기준을 정하도록 할 수 있습니다. 그런 기준이 없다면 API는 default 구현체를 반환하거나 client가 사용할 수 있는 구현체를 cycle하도록 할 것입니다. 바로 이 Service access API가 유연한 static factory가 됩니다.

부가적인 네번째 요소: service provider interface

이 요소는 service interface의 instance를 생성하는 factory method를 설명합니다. 이것이 없다면, 구현은 reflect로 초기화되어야 합니다.

JDBC에서는 Connection이 service interface 역할을 하며, DriverManager.registerDriver가 provider registration API역할을 하고, DriverManager.getConnection이 service access API가 됩니다. 그리고 Driver가 service provider interface가 됩니다.

service provider framework pattern의 다양한 변형들

service access API가 Client에게 bridge pattern처럼 풍부한 service interface를 제공할 수 있습니다.

Dependency injection frameworks는 매우 강력한 service provider입니다. Java 6부터 범용 목적의 service provider framework로써 `java.util.ServiceLoader`를 제공해줍니다.

https://jurogrammer.tistory.com/25?category=896880

단점

1. public, protected 접근제한자 constructor는 subclass를 지닐 수 없습니다.

예를 들어 Collection Framework의 어떠한 sub class를 지닐 수 없죠.

하지만! 논쟁의 여지가 있지만, programmers가 상속대신 compisition을 사용하도록 강제합니다. 그리고 이것은 불변 타입으로 필요로 하죠.(Item 17) 따라서 장점이라 볼 수 있습니다.

2. 프로그래머가 찾기 어렵습니다.

class를 어떻게 인스턴스화하는지 이해하기 어렵습니다.

따라서 이는 다음과 같은 naming convention을 이용하면 편해집니다.

요약

static factory mehods나 public constructor는 각자 자기 사용 방법이 있습니다. 일반적으로 static factories가 선호되므로 이를 먼저 고려하고 public constructor를 고려하길 바랍니다.