Effective Java

[Issue 1 and 2]

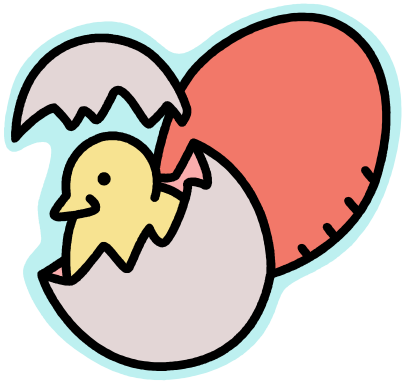
Version 0.1

2009.07.31

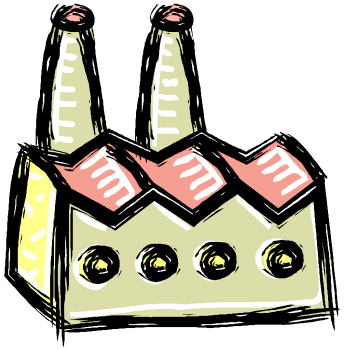
This report is written by Sunny Kwak. (sunnykwak@hanmail.net, sunnykwak.egloos.com)

Issue 1. 생성자 대신 static 팩토리 메소드 사용을 고려하자.

객체는 생성자 함수만이 만들 수 있는 게 아니다!



VS.



```
public Boolean( boolean value )
{
   this.value = value;
}
```

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

java API에 포함된 Boolean 클래스는 생성자(Constructor) 도 제공하지만,

가급적 valueOf() 정적 메소드 사용을 권장하고 있다.

결과는 똑같아 보이지만, valueOf() 메소드를 사용하면 인스턴스 생성이 억제 된다. (자원 절약 효과)

Issue 1. 생성자 대신 static 팩토리 메소드 사용을 고려하자

Static Factory 메소드의 장점

1. 생성자 함수와 달리 팩토리 메소드는 클래스와 다른 명칭을 가질 수 있다.

- BigInteger 클래스의 소수(素數) 인스턴스를 반환하는 메소드의 명칭은 BigInteger.probalePrime() 이며, 생성자의 특징을 명확히 드러낸다.
- 생성자가 여러 개 필요할 경우, 각각의 용도에 맞는 이름을 부여할 수 있다.

2. 호출 될 때마다 새로운 객체를 생성할 수도 있고, 이미 생성된 객체를 반환 할 수 도 있다.

- 인스턴스 생성을 제어할 수 있으므로, 자원을 좀 더 효율적으로 관리할 수 있게 된다.
- 싱글톤(singleton) 또는 인스턴스 생성 불가(noninstantiable) 클래스를 만들 수 있다.

3. 자신이 반환하는 타입의 하위 타입(sub type) 객체도 반환할 수 있다.

- 생성자 함수는 자신이 속한 클래스의 인스턴스만을 만들 수 있지만, Static Factory는 유연하게 하위 클래스의 객체를 생성하고 반환할 수 있다.

4. 매개변수화 타입(parameterized type) 인스턴스를 생성하는 코드를 간결하게 해준다.

```
public static <K, V> HashMap <K, V> newInstance()
{ return new HashMap<K, V>(); }
```

Issue 1. 생성자 대신 static 팩토리 메소드 사용을 고려하자

Static Factory 메소드의 단점

1. **인스턴스 생성을 위한 public/protected 생성자를 선언하지 않고 static factory 메소드만 가지고 있는 클래스는 하위 클래스를 만들 수 없다.**

   - 장점일 수도 있도, 단점이 될 수도 있다.

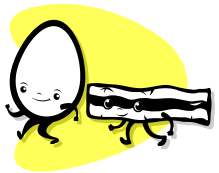
2. **다른 static 메소드와 구분하기 어렵다.**

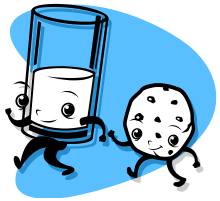
   - 다른 메소드와 섞여 있으면, 용도를 알아차리기 어렵기 때문에, 문서화에 신경 써야 한다.

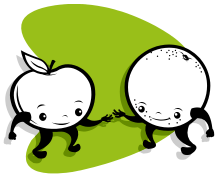
static factory 메소드와 public 생성자는
상호 보완 관계이므로 적절히 골라쓰자!

Issue 2. 생성자의 매개 변수가 많을 때는 빌더(builder)를 고려하자

매개 변수가 많은 생성자의 문제?







'식품 영양 정보'를 나타내는 데이터 클래스를 예로 들어보자.
다양한 필수, 선택 항목을 포함할 수 있다.

필수 항목 : 식품의 용량, 개수, 칼로리
선택 항목 : 총 지방 함량, 포화 지방, 트랜스 지방, 콜레스테롤,
나트륨, 단백질, 탄수화물, 비타민… 등 수십개 항목

NutritionFacts

+servingSize
+servings
+calories
+fat
+sodium
+carbohydrate
+protein

+NutritionFatcs(int servingSize, int serviings)
+NutritionFacts(int servingSize, int servings, int calories)

1. 인스턴스 생성 시에 초기화 해야 하는 항목 수가 많고…
2. 필수/선택 매개변수들을 초기화할 수 있는 다양한 생성자 함수를 선언해야 한다.
3. 일반적으로, 텔리스코핑(telescoping) 생성자 패턴을 사용한다.

```
NutritionFacts cola
    = new NutritionFacts( 240, 8, 120, 10, 26, 27 );
```

위와 같은 코드는 가독성이 현저히 떨어진다. 즉, 이해하기 어렵다.

각각의 파라미터가 어떤 필드를 설정하는지…?

Issue 2. 생성자의 매개 변수가 많을 때는 빌더(builder)를 고려하자

대안~? 자바빈즈 패턴

매개 변수가 없는 객체를 생성한 후 setter 메소드를 호출해서 각각의 필드 값을 초기화 한다.

```
NutritionFacts cola = new NutritionFacts();
cola.setServingSize(240);
cola.setServings(8);
cola.setCalories(80);
cola.setSodium(35);
```

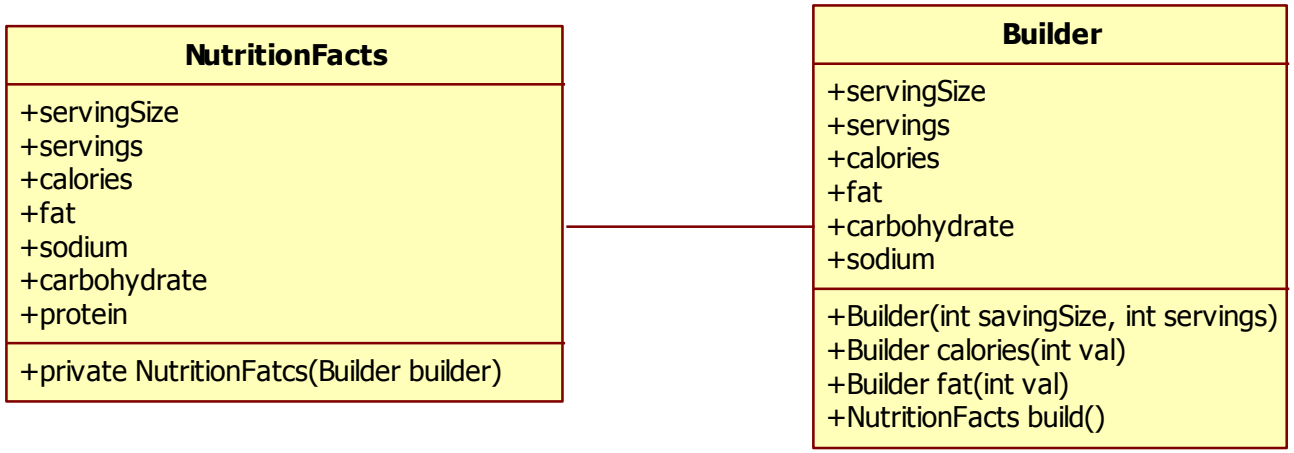
한 시절 유행(?)하던 기법이지만, 생성 과정에서 불완전한 상태에 놓여 있게 된다.

불변(immutable) 클래스를 만들 수도 없다.

Issue 2. 생성자의 매개 변수가 많을 때는 빌더(builder)를 고려하자

확실한 대안~! 빌더 패턴 (Builder pattern)

빌더(builder) 클래스는 자신이 생성하는 객체의 클래스에 포함된 static 멤버 클래스.




```
NutritionFacts cola = new NutritionFacts.Builder(240, 8).
calories(100).sodium(35).carbohydrate(27).build();
```

코드 작성이 쉽고, 이해하기 쉽다. 다만, 코드 분량이 증가하는 단점이 있다.