

자원을 직접 명시하지 말고 의존 객체  
주입을 사용하라

이펙티브 자바 item5

유현우









웹사이트 번역

문서 번역 <sup>Beta</sup>

사전

한국어 ▾



오늘 저녁에 뭐해?



10 / 3000



번역하기

영어 ▾

What are you doing tonight?

윳 아아 유 두잉 터나이트?

번역 수정 | 번역 평가





웹사이트 번역

문서 번역 <sup>Beta</sup>

사전

한국어 ▾



오늘 저녁에 뭐해?



10 / 3000



번역하기

스페인어 ▾

한국어

영어

일본어

중국어(간체)

중국어(번체)

스페인어 ✓

프랑스어

독일어

러시아어

포르투갈어

이탈리아어

베트남어

태국어

인도네시아어

힌디어

아랍어



# Dictionary 클래스를 사용하는 SpellChecker 클래스


어떠한 단어를 Dictionary 클래스를 통해서 올바른 단어인지 확인하거나  
비슷한 단어를 반환받을 수 있다.



```
1 public class SpellChecker {  
2  
3     private static final Dictionary dictionary = new Dictionary();  
4  
5     private SpellChecker() {}  
6  
7     public static boolean isValid(String word) { ... }  
8  
9     public static List<String> suggestions(String typo) { ...}  
10 }
```

## 단점

1. 자원에 따라 동작이 달라지는 클래스에는 정적 유틸리티 클래스나 싱글턴 방식이 적합하지 않다.
2. 테스트에 비효율적이다.



```
1 public class SpellChecker {  
2  
3     private static final Dictionary dictionary = new Dictionary();  
4  
5     private SpellChecker() {}  
6  
7     public static boolean isValid(String word) { ... }  
8  
9     public static List<String> suggestions(String typo) { ...}  
10 }
```

**자원에 따라 동작이 달라지는 클래스에는 정적 유틸리티  
클래스나 싱글톤 방식이 적합하지 않다.**



웹사이트 번역

문서 번역 <sup>Beta</sup>

사전

한국어 ▾



오늘 저녁에 뭐해?



10 / 3000



번역하기

스페인어 ▾

한국어

영어

일본어

중국어(간체)

중국어(번체)

스페인어 ✓

프랑스어

독일어

러시아어

포르투갈어

이탈리아어

베트남어

태국어

인도네시아어

힌디어


아랍어





## 단점

1. 자원에 따라 동작이 달라지는 클래스에는 정적 유틸리티 클래스나 싱글턴 방식이 적합하지 않다.
2. 테스트에 비효율적이다.



```
1 public class SpellChecker {  
2  
3     private static final Dictionary dictionary = new Dictionary();  
4  
5     private SpellChecker() {}  
6  
7     public static boolean isValid(String word) { ... }  
8  
9     public static List<String> suggestions(String typo) { ...}  
10 }
```

## 1. 자원에 따라 동작이 달라지는 클래스에는 정적 유틸리티 클래스나 싱글톤 방식이 적합하지 않다.

```
1 class SpellChecker {  
2     private static Dictionary dictionary = new Dictionary(); // final 제거  
3  
4     private SpellChecker() {}  
5  
6     public static void setDictionary(Dictionary dictionary) { // setter 추가  
7         SpellChecker.dictionary = dictionary;  
8     }  
9     ...  
10 }
```

**초기화가 되었는지 확인하기 힘들기 때문에 오류가 발생하기 쉽다.  
불변이 아니기 때문에 멀티스레드 환경에서는 쓸 수 없다.**

## 2. 테스트에 비효율적이다.

```
1 class SpellChecker {
2     // 자원을 직접 명시
3     private static final Dictionary DICTIONARY = new Dictionary();
4     private SpellChecker() {}
5
6     public static boolean isValid(String text) {
7         return DICTIONARY.isValid(text);
8     }
9 }
10
11 class Dictionary {
12     // 연산 많이 걸린다고 가정
13     public Dictionary() {
14         System.out.println("테스트마다 인스턴스 생성");
15     }
16     public boolean isValid(String text) {
17         return true;
18     }
19 }
```

```
1 public class SpellCheckerTest {
2
3     @Test
4     public void newInstance1() {
5         Assertions.assertTrue(SpellChecker.isValid("hi"));
6     }
7
8     @Test
9     public void newInstance2() {
10         Assertions.assertTrue(SpellChecker.isValid("hi"));
11     }
12 }
```

## 2. 테스트에 비효율적이다.

```
class SpellChecker {  
    // 자원을 직접 명시  
    private static final Dictionary DICTIONARY = new Dictionary();  
    private SpellChecker() {}  
}
```

```
1 public class SpellCheckerTest {  
2  
3     @Test  
4     public void newInstance1() {  
5         Assertions.assertTrue(SpellChecker.isValid("hi"));  
6     }  
7  
8     @Test  
9     public void newInstance2() {  
10        Assertions.assertTrue(SpellChecker.isValid("hi"));  
11    }  
12 }
```

**테스트를 하나씩 돌리는 경우**

**테스트를 실행할 때 마다 Dictionary 클래스 인스턴스를 생성한다.**

**테스트를 한번에 여러개 돌리는 경우**


**다른 테스트들이 진행될 때 불필요한 Dictionary 클래스 인스턴스가 메모리에 올라가 있다는 문제가 있다.**

**자바를 처음 배우는 사람도 사용할 수 있는  
모든 문제 해결방법**



# 의존 객체 주입

인스턴스를 생성할 때 생성자에 필요한 자원을 넘겨주는 방식




```
1 interface Dictionary {
2     boolean isValid(String word);
3 }
4
5 class SpellChecker {
6     private final Dictionary dictionary;
7
8     // 의존 객체 주입
9     public SpellChecker(Dictionary dictionary) {
10         this.dictionary = dictionary;
11     }
12
13     public boolean isValid(String word) {
14         return dictionary.isValid(word);
15     }
16 }
```

## 장점 1. 다양한 구현체가 생성자로 올 수 있고 불변으로 사용하고 있기 때문에 멀티스레드 환경에서 사용 가능하다.

```
1 interface Dictionary {  
2     boolean isValid(String word);  
3 }  
4  
5 class SpellChecker {  
6     private final Dictionary dictionary;  
7  
8     public SpellChecker(Dictionary dictionary) {  
9         this.dictionary = dictionary;  
10    }  
11  
12    public boolean isValid(String word) {  
13        return dictionary.isValid(word);  
14    }  
15 }
```

**장점 2. 인터페이스를 주입받기 때문에 Dictionary를 익명 클래스로 사용이 가능하고, Mock으로도 쉽게 테스트할 수 있다.**



```
1 public class SpellCheckerTest {
2     @Test
3     public void SpellCheckerImplTest() {
4         // 람다로 처리가 힘들면 테스트용 클래스를 만들 수 있고, Mock을 사용해도 된다.
5         SpellChecker spellChecker = new SpellChecker(new Dictionary() {
6             @Override
7             public boolean isValid(String word) {
8                 return false;
9             }
10        });
11        Assertions.assertFalse(spellChecker.isValid("hi"));
12    }
13 }
```

## 다른 장점들

**의존 객체 주입은 생성자, 정적 팩터리, 빌더를 응용해서 사용할 수 있다.**

- 정적 팩터리(아이템1)
- 빌더(아이템2)

**의존성이 너무 커지는 경우 프레임워크를 활용해서 의존성 주입을 활용할 수 있다.**

### 의존성 주입(Dependency Injection)

Spring 프레임워크는 '3가지 핵심 프로그래밍 모델'을 지원하고 있는데, 그 중 하나가 '의존성 주입'(Dependency Injection, DI)이다. DI란 외부에서 두 객체 간의 관계를 결정해주는 디자인 패턴으로, 인터페이스를 사이에 두서 클래스 레벨에서는 '의존관계가 고정되지 않도록' 하고 '런타임 시에 관계를 동적으로 주입'하여 '유연성을 확보'하고 '결합도를 낮출' 수 있게 해준다.

## Reference

의존성 주입(Dependency Injection, DI)이란? 및 Spring이 의존성 주입을 지원하는 이유: <https://mangkyu.tistory.com/150>

이펙티브 자바 아이템 5 - 자원을 직접 명시하지 말고 의존 객체 주입을 사용하라 - 핵심 정리: <https://pro-dev.tistory.com/105>

이펙티브 자바 아이템 5 - 자원을 직접 명시하지 말고 의존 객체 주입을 사용하라 - 완벽 공략: <https://pro-dev.tistory.com/106>



# Thank you

궁금한 점을 물어보세요