

적시에 방어적 복사본을 만들라

Effective Java Item 50



자바는 주로 네이티브 메서드를 사용하지 않기 때문에
C, C++ 언어에서 발생하는 메모리 충돌 오류에서 상대적으로 안전합니다.

BUT

아무리 자바라 해도 다른 클래스로부터의 **침범**을 아무런 노력없이 막을 수 없다.

클라이언트가 여러분의 불편식을 깨드리려 혈안이 되어있다고 가정하고 방어적 프로그래밍 하기!



자바에서 제공하는 **Date** 클래스는 가변이기 때문에 쉽게 불변식을 깨뜨릴 수 있습니다.

불변식을 지키지 못한 클래스

```
1 public final class Period {
2     private final Date start;
3     private final Date end;
4
5     /**
6      * @param start 시작 시각
7      * @param end 종료 시각. 시작 시각보다 뒤여야 한다.
8      * @throws IllegalArgumentException 시작 시각이 종료 시각보다 늦을 때 발생한다.
9      * @throws NullPointerException start나 end가 null이면 발생한다.
10    */
11    public Period(Date start, Date end) {
12        if (start.compareTo(end) > 0)
13            throw new IllegalArgumentException(
14                start + "가 " + end + "보다 늦다.");
15        this.start = start;
16        this.end = end;
17    }
18
19    public Date start() {
20        return start;
21    }
22    public Date end() {
23        return end;
24    }
25
26    public String toString() {
27        return start + " - " + end;
28    }
29 }
```

```
1 Date start = new Date();
2 Date end = new Date();
3 Period p = new Period(start, end);
4 end.setYear(99);
```

```
@Deprecated
public void setYear(int year) {
    getCalendarDate().setNormalizedYear(year + 1900);
}
```

Returns a number representing the month that contains or begins with the instant in time represented by this `Date` object. The value returned is between `0` and `11`, with the value `0` representing January.

Deprecated As of JDK version 1.1, replaced by `Calendar.get(Calendar.MONTH)`.

Returns: the month represented by this date.

See Also: `Calendar`

어떻게 하면 좋을까요?



방어적 복사 예시



```
1  public Period(Date start, Date end) {  
2      this.start = new Date(start.getTime());  
3      this.end = new Date(end.getTime());  
4  
5      // 유효성 검사 전에 복사해야 한다.  
6      if(start.compareTo(end) > 0) {  
7          throw new IllegalArgumentException(start + " after " + end);  
8      }  
9  }
```


BUT

period 인스턴스는 아직도 변경 가능 합니다.

```
1  class Period {
2      private final Date start;
3      private final Date end;
4
5      public Period(Date start, Date end) {
6          this.start = new Date(start.getTime());
7          this.end = new Date(end.getTime());
8
9          if(start.compareTo(end) > 0) {
10              throw new IllegalArgumentException(start + " after " + end);
11          }
12
13          public Date start() { return start; }
14          public Date end() { return end; }
15          // ... 생략
16  }
```

```
1  public void someMethod() {
2      Date start = new Date();
3      Date end = new Date();
4      Period period = new Period(start, end);
5
6      // period의 내부를 또 수정했다.
7      period.end().setMonth(2);
8  }
```


방어적 복사 완성

```
1  class Period {
2      private final Date start;
3      private final Date end;
4
5      public Period(Date start, Date end) {
6          this.start = new Date(start.getTime());
7          this.end = new Date(end.getTime());
8
9          if(start.compareTo(end) > 0) {
10             throw new IllegalArgumentException(start + " after " + end);
11         }
12     }
13     public Date start() {
14         return new Date(start.getTime());
15     }
16     public Date end() {
17         return new Date(end.getTime());
18     }
19     // ... 생략
20 }
```


! 매개변수를 방어적으로 복사하는 목적이 불변 객체를 만들기 위해서만은 아니다.

메서드든 생성자든 클라이언트가 제공한 객체의 참조를 내부의 자료구조에 보관해야 할 때면 항상 그 객체가 잠재적으로 변경될 수 있는지를 생각해야 합니다.

만약 변경 될 수 있는 객체라면 그 객체가 클래스에 넘겨진 뒤 임의로 변경되어도 그 클래스가 문제없이 동작할지를 꼭 따져봐야 합니다.

확신할 수 없다면?

-> 복사본을 만들어 저장

확신할 수 있다면?

-> 복사본을 만들기 생략

! 불변 객체들을 조합해 객체를 구성해야 방어적 복사를 할 일이 줄어든다

방어적 복사에는 성능 저하가 따르고, 또 항상 쓸 수 있는 것도 아니다.

호출자가 컴포넌트 내부를 수정하지 않으리라 확신하면 방어적 복사를 생략할 수 있다.

**이러한 상황이라도 호출자에서 해당 매개변수나 반환값을 수정하지 말아야 함을
명확히 문서화하는게 좋다.**

마무리

 클래스의 구성요소가 가변이라면 그 요소는 반드시 방어적으로 복사하기

 아래 경우는 제외

1 복사 비용이 너무 클 때

2 클라이언트가 그 요소를 잘못 수정할 일이 없음을 확신할 수 있을 때