Effective Java

Item 50 : 적시에 방어적 복사본을 만들라

Item 51 : 메서드 시그니처를 신중히 설계하라

Item 50

적시에 방어적 복사본을 만들라

얕은 복사, 방어적 복사, 깊은 복사

```java
public class Product {

    private String name;

    public Product(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return name;
    }
}
```

```java
public static void main(String[] args) {
    List<Product> products = new ArrayList<>();
    products.add(new Product("Desktop"));
    products.add(new Product("Keyboard"));

    // .. 추가 코드 ..
}
```

문제1 : 원본 리스트가 변경되면, 멤버 리스트도 변경된다.

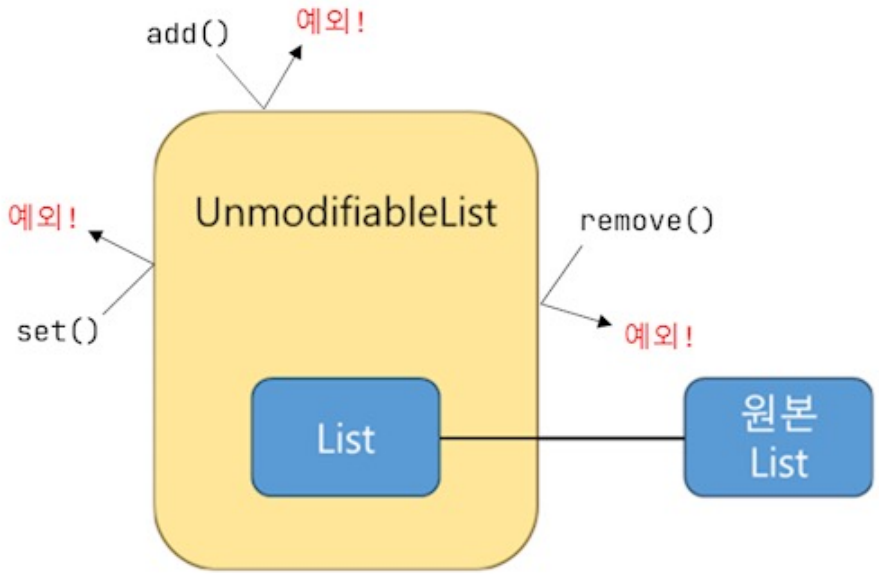
문제2 : 원본 리스트의 원소가 변경되면, 멤버 리스트의 원소도 변경된다.

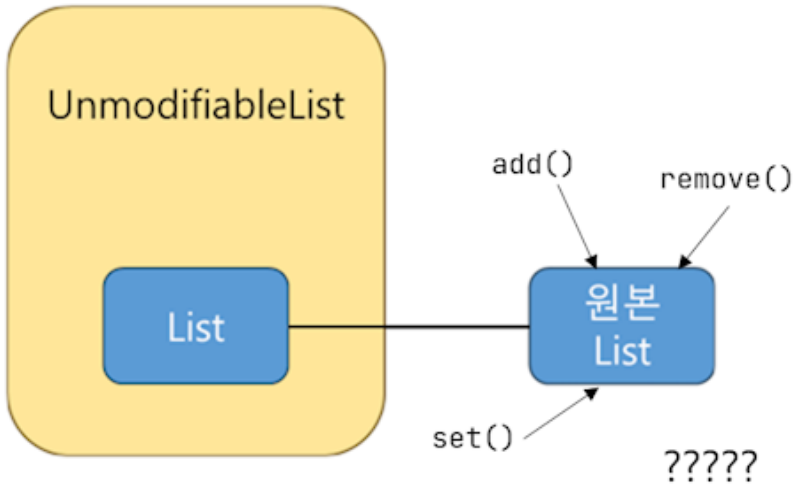
문제3 : getter로 얻은 리스트가 변경되면, 멤버 리스트도 변경된다.

문제4 : getter로 얻은 리스트의 원소가 변경되면, 멤버 리스트의 원소도 변경된다.

얕은 복사, new ArrayList(), 깊은 복사, Collections.unmodifiableList
총 16가지 조합..

Collections.unmodifiableList





얕은 복사

```java
public class Shop {

    private final List<Product> products;

    public Shop(List<Product> products) {
        this.products = products;
    }

    public List<Product> getProducts() {
        return products;
    }
}
```

```java
Shop shop = new Shop(products);
System.out.println(shop.getProducts());
```
[Desktop, Keyboard]

```java
products.remove(0);
System.out.println(shop.getProducts());
```
[Keyboard] **문제1**

```java
products.get(0).setName("ABC");
System.out.println(shop.getProducts());
```
[ABC] **문제2**

```java
List<Product> shopProducts = shop.getProducts();
shopProducts.add(new Product("Monitor"));
System.out.println(shop.getProducts());
```
[ABC, Monitor] **문제3**

```java
shopProducts.get(0).setName("DEF");
System.out.println(shop.getProducts());
```
[DEF, Monitor] **문제4**

방어적 복사 + 얕은 복사

```java
public class Shop {

    private final List<Product> products;

    public Shop(List<Product> products) {
        this.products
          = new ArrayList<>(products);
    }

    public List<Product> getProducts() {
        return products;
    }
}
```

```java
Shop shop = new Shop(products);
System.out.println(shop.getProducts());   [Desktop, Keyboard]
```

문제1

```java
products.remove(0);
System.out.println(shop.getProducts());   [Desktop, Keyboard]
```

```java
products.get(0).setName("ABC");
System.out.println(shop.getProducts());   [Desktop, ABC]
```

문제2

```java
List<Product> shopProducts = shop.getProducts();
shopProducts.add(new Product("Monitor"));
System.out.println(shop.getProducts());   [Desktop, ABC, Monitor]
```

문제3

```java
shopProducts.get(0).setName("DEF");
System.out.println(shop.getProducts());   [DEF, ABC, Monitor]
```

문제4

unmodifiableList + 얕은 복사

```java
public class Shop {

    private final List<Product> products;

    public Shop(List<Product> products) {
        this.products =
         Collections.unmodifiableList(products);
    }

    public List<Product> getProducts() {
        return products;
    }
}
```

```java
Shop shop = new Shop(products);
System.out.println(shop.getProducts()); // [Desktop, Keyboard]

products.remove(0);
System.out.println(shop.getProducts()); // [Keyboard]

products.get(0).setName("ABC");
System.out.println(shop.getProducts()); // [ABC]

List<Product> shopProducts = shop.getProducts();
shopProducts.add(new Product("Monitor"));
System.out.println(shop.getProducts()); // UnsupportedOperationException

shopProducts.get(0).setName("DEF");
System.out.println(shop.getProducts()); // [DEF]
```

문제1

문제2

문제3

문제4

방어적 복사 + 깊은 복사

```java
public class Shop {

    private final List<Product> products;

    public Shop(List<Product> products) {
        this.products
          = new ArrayList<>(products);
    }

    public List<Product> getProducts() {
        return products.stream()
                .map(p -> new Product(p.getName()))
                .collect(Collectors.toList());
    }
}
```

```java
Shop shop = new Shop(products);
System.out.println(shop.getProducts()); [Desktop, Keyboard]
```

```java
products.remove(0);
System.out.println(shop.getProducts()); [Desktop, Keyboard]
```

문제1

```java
products.get(0).setName("ABC");
System.out.println(shop.getProducts()); [Desktop, ABC]
```

문제2

```java
List<Product> shopProducts = shop.getProducts();
shopProducts.add(new Product("Monitor"));
System.out.println(shop.getProducts()); [Desktop, ABC]
```

문제3

```java
shopProducts.get(0).setName("DEF");
System.out.println(shop.getProducts()); [Desktop, ABC]
```

문제4

방어적 복사와 유효성 검사

```
public Period(Date start, Date end) {
    if (this.start.compareTo(this.end) > 0) {
        throw new IllegalArgumentException();
    }

    this.start = new Date(start.getTime());
    this.end = new Date(end.getTime());
}
```

```
Date start = new Date(1);
Date end = new Date(2);
Period period = new Period(start, end);
```

```
start = new Date(2);
end = new Date(1);
```

```
public Period(Date start, Date end) {
  this.start = new Date(start.getTime());
  this.end = new Date(end.getTime());

  if (this.start.compareTo(this.end) > 0) {
      throw new IllegalArgumentException();
  }
}
```

유효성 검사를 하기 전에 방어적 복사본을 만들고,
이 복사본으로 유효성 검사 수행
(TOCTOU 공격 방지)

기타

- 가능하면 clone 은 사용하지 말자. (생성자나 정적 팩터리를 사용하자)
- 되도록 불변 객체들을 조합해 객체를 구성해야 안전하다.(+편하다.)
- 문서화를 잘하자.

Item 51

메서드 시그니처를 신중히 설계하라

매개변수 목록은 짧게 유지하자

기본은 4개 이하

긴 매개변수 목록을 짧게 줄여주는 기술

1. 여러 메서드로 쪼갠다.
2. 매개변수 여러 개를 묶어주는 도우미 클래스를 이용

```
Page page = new Page(pageSize, pageNo);
list.find(page);
```

3. Setter

```
Shop shop = new shop();
shop.setPage(pageSize);
shop.setPageNo(pageNo);

shop.execute();
```

매개변수 줄이기 – 1. 여러 메서드로 쪼갠다.

Example

```
list.findElementsInSubList(fromIdx, toIdx, element);
```

VS

```
List<String> subList = list.subList(fromIdx, toIdx);
subList.indexOf(element);
```

직교성 (orthogonality)

기능을 원자적으로 쪼개 제공한다.

기본 기능을 제공하는 메서드 3개로 7개의 기능을 조합해 만들 수 있다.

적절한 추상화, 편의성의 조율

적은 추상화, 높은 직교성

RISC, 마이크로서비스

VS

높은 추상화, 낮은 직교성

CISC, 모놀리식

그 외

- **메서드 이름은 신중히 짓자**
  같은 패키지에 속한 다른 이름들과 일관되도록
- **편의 메서드를 너무 많이 만들지 말자**
  API 익히는 비용, 제작, 유지보수 비용이 높다.
- **매개변수 타입으로는 클래스보다는 인터페이스로**
- **boolean 보다는 원소 2개짜리 열거 타입이 낫다.**
  더 명확하고 나중에 확장하기 쉽다.