

ITEM 17

변경 가능성을 최소화 하라

CONTENTS

- | | |
|--------------|---------------|
| 1. 불변 클래스란 | 4. 불변 클래스 단점 |
| 2. 불변 클래스 규칙 | 5. 또 다른 설계 방법 |
| 3. 불변 클래스 장점 | 6. 주의점 |

CONTENTS

1. 불변 클래스란

4. 불변 클래스 단점

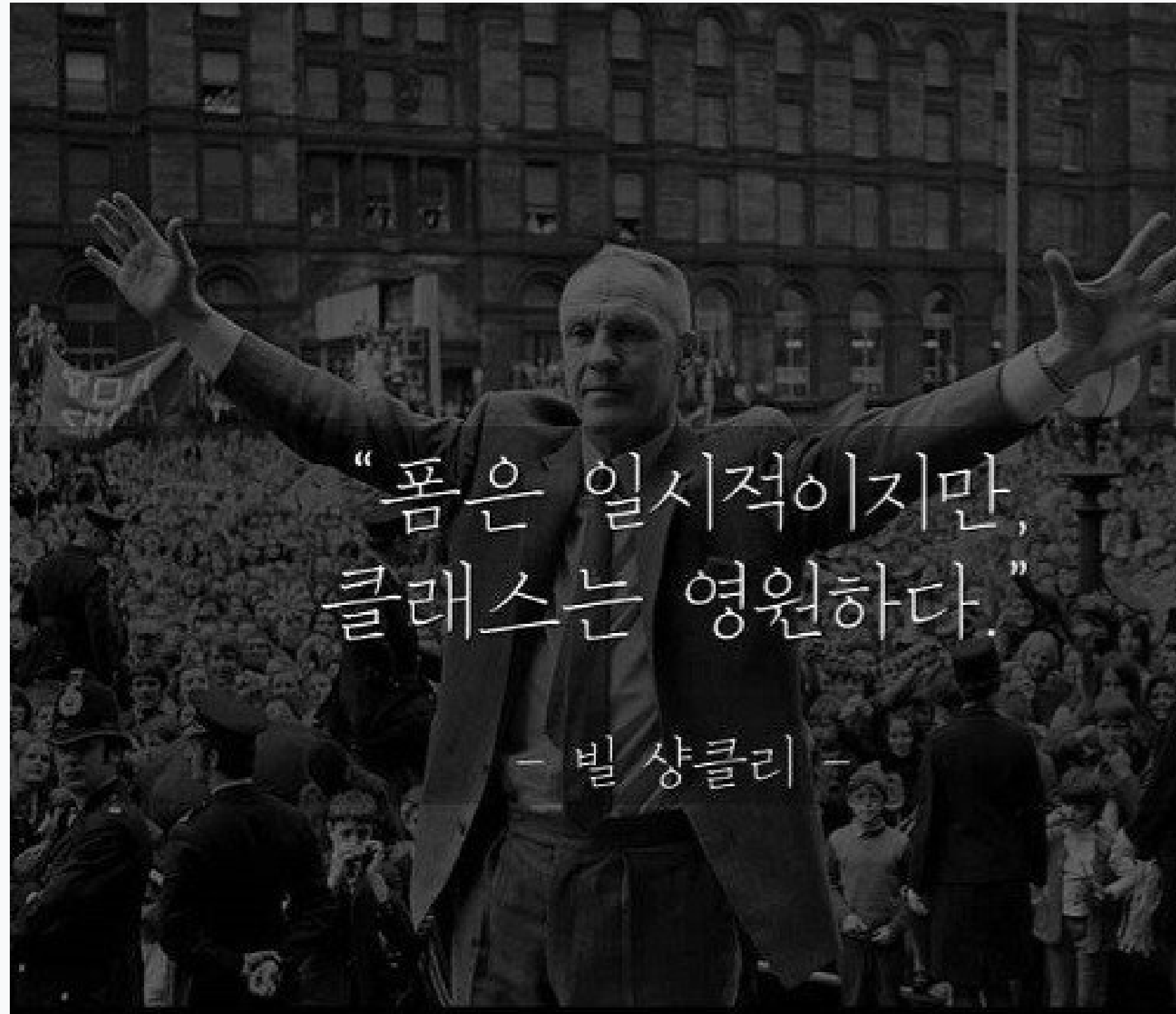
2. 불변 클래스 규칙

5. 또 다른 설계 방법

3. 불변 클래스 장점

6. 주의점

불변 클래스란?



“폼은 일시적이지만,
클래스는 영원하다.”

— 빌 상클리 —

불변 클래스란?



내부 상태가 변하지 않는 클래스

CONTENTS

1. 불변 클래스란

4. 불변 클래스 단점

2. 불변 클래스 규칙

5. 또 다른 설계 방법

3. 불변 클래스 장점

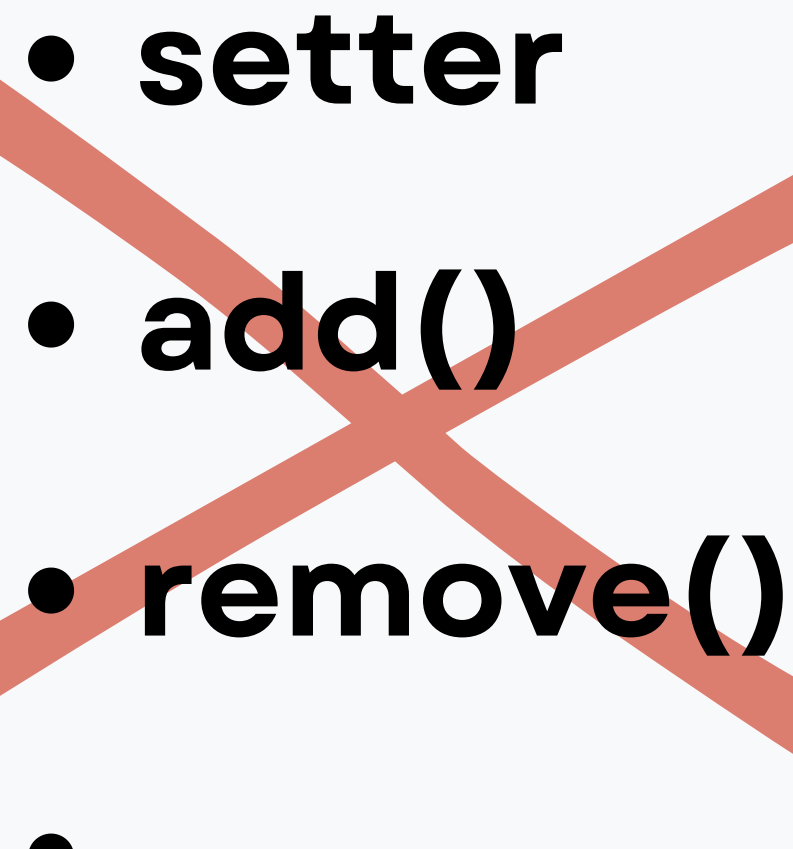
6. 주의점

불변 클래스 규칙

1. 객체의 상태를 변경하는 메서드를 제공하지 않는다.
2. 클래스를 확장할 수 없도록 한다.
3. 모든 필드를 `final`로 선언한다.
4. 모든 필드를 `private`으로 선언한다.
5. 자신 외에는 내부 가변 컴포넌트에 접근할 수 없도록 한다.

불변 클래스 규칙

1. 객체의 상태를 변경하는 메서드를 제공하지 않는다.

- **setter**
 - **add()**
 - **remove()**
 - **...**
- 

불변 클래스 규칙

1. 객체의 상태를 변경하는 메서드를 제공하지 않는다.

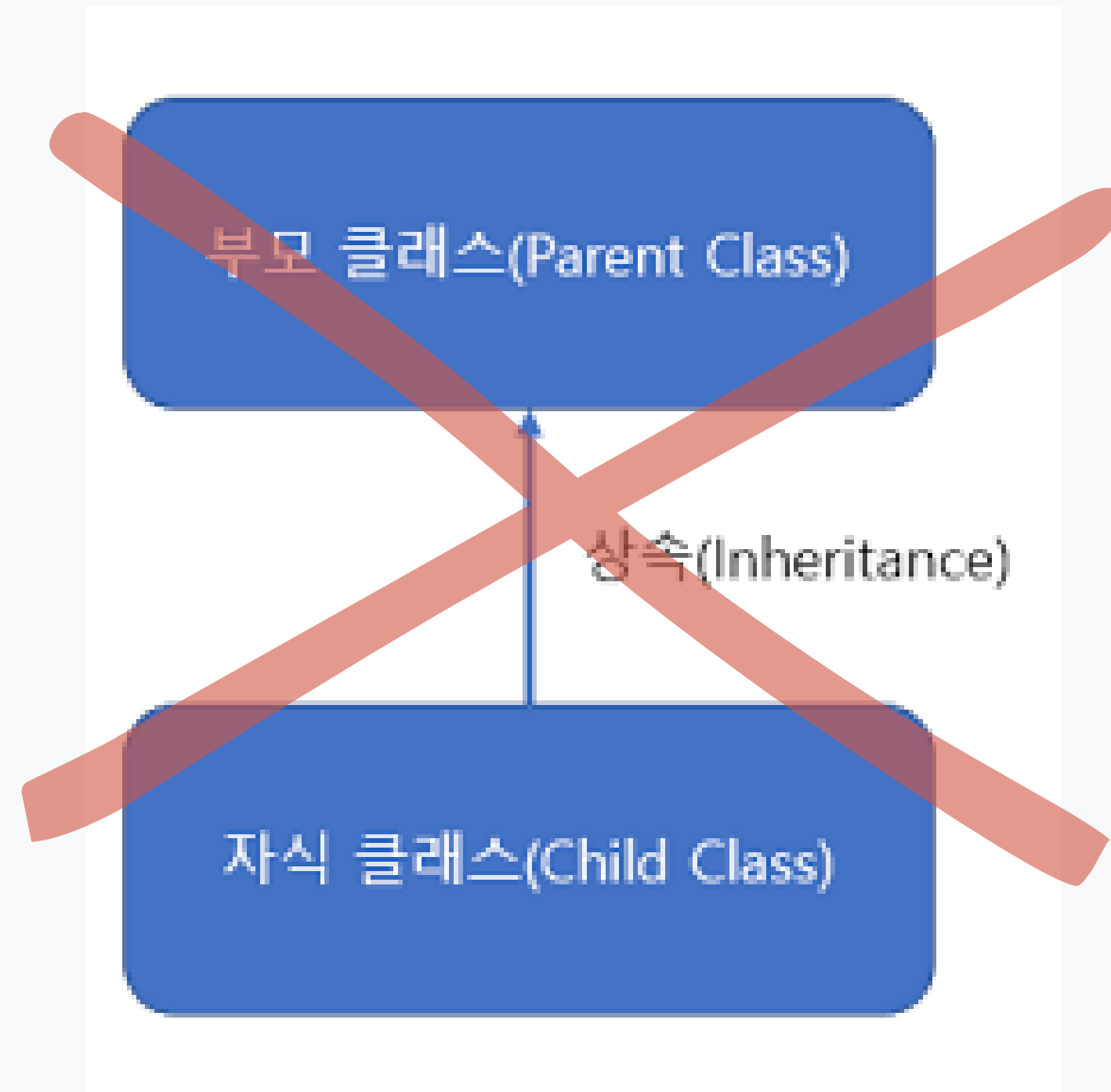
만약에 제공하려면....

방어적 복사 수행 후 변경된 객체 제공

`add(동사)` `->` `plus(전치사)`

불변 클래스 규칙

2. 클래스를 확장할 수 없도록 한다.



불변 클래스 규칙

3. 모든 필드를 final로 선언한다.



불변 클래스 규칙

4. 모든 필드를 **private**으로 선언한다.

~~public final~~ → private → private final

불변 클래스 규칙

5. 자신 외에 내부 가변 컴포넌트 접근 할 수 없게 하자.

- 가변 객체 참조 얻을 수 없도록 해야 함
- 가변 객체 참조 가리키게 하면 안됨
- 가변 객체 필드 그대로 반환하면 안됨

가변 객체 제공 시 방어적 복사 수행

CONTENTS

- | | |
|--------------|---------------|
| 1. 불변 클래스란 | 4. 불변 클래스 단점 |
| 2. 불변 클래스 규칙 | 5. 또 다른 설계 방법 |
| 3. 불변 클래스 장점 | 6. 주의점 |

불변 클래스 장점

- 가변 클래스보다 설계하고 구현하고 사용하기 쉽다.
- 오류가 생길 여지도 적고 훨씬 안전하다.
- 단순하다.
- Thread-Safe 하여 따로 동기화 할 필요가 없다.

불변 클래스 장점

- 불변 객체는 안심하고 공유할 수 있어 재사용 가능하다.
- 불변 객체끼리는 내부 데이터를 공유할 수 있다.
- 객체 만들 때 다른 불변 객체들을 구성 요소로 사용하면 이점 많다.
- 불변 객체는 그 자체로 실패 원자성을 제공한다.

불변 클래스 장점

불변 객체는 안심하고 공유할 수 있어 재활용 가능하다.


Thread-Safe -> 공유 O -> 재활용 O

재활용 방법

- 자주 쓰는 값 **public static final 상수**로 제공
- 자주 쓰는 인스턴스 **캐싱 후 정적 팩터리 메서드** 제공

불변 클래스 장점

불변 객체끼리는 내부 데이터를 공유할 수 있다.



```
public class BigInteger extends Number implements Comparable<BigInteger> {  
    final int signum;  
  
    final int[] mag;  
    ...  
  
    public BigInteger negate() {  
        return new BigInteger(this.mag, -this.signum);  
    }  
}
```

불변 클래스 장점

불변 객체들을 구성 요소로 사용하면 이점 많다.

불변 객체는 **Map의 키**와 **Set의 원소**로 쓰기 좋다.

불변 객체는 그 자체로 실패 원자성을 제공한다.

실패 원자성

메서드에서 예외 발생한 후에도 그 객체는 여전히
메서드 호출 전과 똑같은 유효한 상태여야 한다는 성질

- 데이터 무결성 유지
- 예외 발생 시 복원
- 멀티스레딩 환경에서 안전성

CONTENTS

1. 불변 클래스란

4. 불변 클래스 단점

2. 불변 클래스 규칙

5. 또 다른 설계 방법

3. 불변 클래스 장점

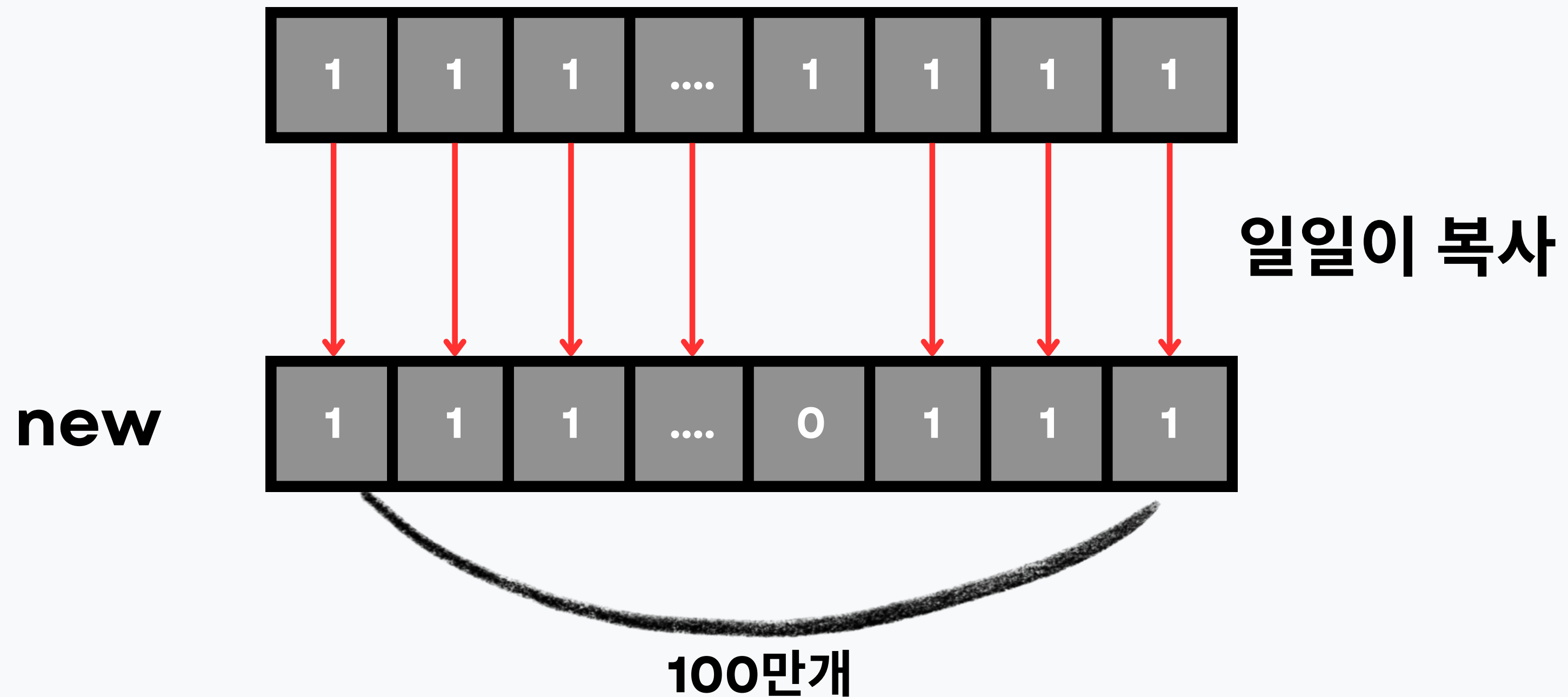
6. 주의점

불변 클래스 단점

- **값이 다르면** 반드시 독립된 객체로 만들어야 한다.
- 불변 객체 생성하는데 **성능이 안 좋을 때가 있다.**

불변 클래스 단점

값이 다르면 반드시 독립된 객체로 만들어야 한다.



불변 클래스 단점

불변 객체 생성하는데 성능이 안 좋을 때가 있다.

문제 상황

1. 객체 완성하기까지 **단계 많음**
2. 중간 단계에서 만들어진 **객체들 버려짐**

불변 클래스 단점

불변 객체 생성하는데 성능이 안 좋을 때가 있다.

해결 방법

- 다단계 연산 예측 가능 시 **가변 동반 클래스 제공**
- 예측 불가능 시 가변 동반 클래스 **public** 제공

해결책 : 다단계 연산 예측 가능 시 가변 동반 클래스 제공

```
public BigInteger modPow(BigInteger exponent, BigInteger m) {  
    ...  
  
    if ( ... ) { // odd modulus  
        result = ... ;  
    } else {  
        ...  
  
        if (m.mag.length < MAX_MAG_LENGTH / 2) {  
            result = ... ;  
        } else {  
            MutableBigInteger t1 = new MutableBigInteger();  
            new MutableBigInteger(a1.multiply(m2)).multiply(new MutableBigInteger(y1), t1);  
  
            MutableBigInteger t2 = new MutableBigInteger();  
            new MutableBigInteger(a2.multiply(m1)).multiply(new MutableBigInteger(y2), t2);  
  
            t1.add(t2);  
  
            MutableBigInteger q = new MutableBigInteger();  
            result = t1.divide(new MutableBigInteger(m), q).toBigInteger();  
        }  
    }  
  
    return ... ;  
}
```

해결책 : 다단계 연산 예측 가능 시 가변 동반 클래스 제공

```
class MutableBigInteger {  
  
    int[] value;  
  
    ...  
  
    void multiply(MutableBigInteger y, MutableBigInteger z) {  
        int xLen = intLen;  
        int yLen = y.intLen;  
        int newLen = xLen + yLen;  
  
        if (z.value.length < newLen)  
            z.value = new int[newLen];  
  
        ...  
    }  
    ...  
}
```

해결책 : 예측 불가능 시 가변 동반 클래스 public 제공

예측 불가능 시 가변 동반 클래스 public으로 제공

개발자가 가변 동반 클래스 직접 사용

불변 클래스 단점

불변 객체 생성하는데 성능이 안 좋을 때가 있다.

해결 방법

- 다단계 연산 예측 가능 시 **가변 동반 클래스 제공**
- 예측 불가능 시 가변 동반 클래스 **public** 제공

CONTENTS

1. 불변 클래스란

4. 불변 클래스 단점

2. 불변 클래스 규칙

5. 또 다른 설계 방법

3. 불변 클래스 장점

6. 주의점

또 다른 설계 방법

1. 모든 **생성자** `private or package-private & public` **정적 팩터리** 제공
2. 어떤 메서드도 객체의 상태 중 **외부에 비치는 값** 변경할 수 없게 설계

또 다른 설계 방법

생성자 숨기기 & public 정적 팩터리 제공

2번 규칙: 상속 못하도록 final class로 생성



1. 모든 생성자를 private or package-private

2. public 정적 팩터리 메서드를 제공

또 다른 설계 방법

어떤 메서드도 외부에 비치는 상태 변경할 수 없게 설계

1번 규칙: 객체 상태 변경 메서드 제공 X

3번 규칙: 모든 필드 `final`로 선언



값 미리 계산 후 `final` 아닌 필드에 캐시

CONTENTS

- | | |
|--------------|---------------|
| 1. 불변 클래스란 | 4. 불변 클래스 단점 |
| 2. 불변 클래스 규칙 | 5. 또 다른 설계 방법 |
| 3. 불변 클래스 장점 | 6. 주의점 |

불변 클래스 관련 주의점

- 불변으로 만들 수 없는 클래스라도 변경할 수 있는 부분을 최소한으로 줄이자.
- 생성자는 불변식 설정이 모두 완료된 상태의 객체 생성해야 한다.