# 이펙티브 자바

## item60 ~ item63

24/06/04

# Item 60

## 정확한 답이 필요하다면 float와 double은 피하라

# 아이템 60 정확한 답이 필요하다면 float와 double은 피하라

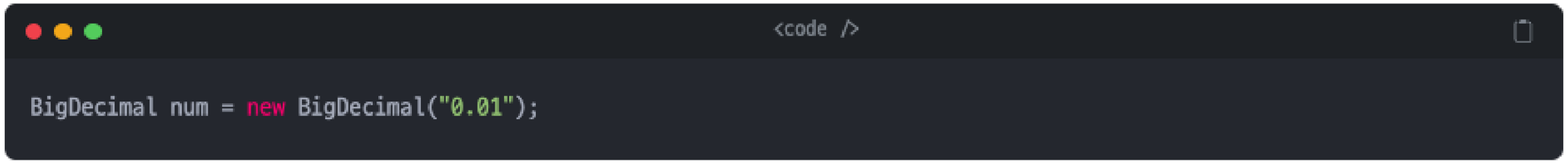
float와 double 타입은 과학 / 공학 계산용으로 설계되었다.
해당 자료형들은 부동소수 타입으로, 정확한 결과를 위해서는 사용을 지양해야 한다.

## 실수를 표현하는 방법 두가지

- 고정 소수점 방식: 소수점에 이용하는 범위를 고정해 수의 표현 범위가 크지 않다.
- 부동 소수점 방식: 소수점을 이용하지 않아 그 만큼 다른 수를 표현할 수 있도록 더 넓은 표현 범위를 가진다.

# 아이템 60 정확한 답이 필요하다면 float와 double은 피하라

정확한 계산을 위해서는 BigDecimal!




but 매우 느리므로 소수점을 정수로 표현 가능하다면 int나 long으로 연산하는 것도 good

# 아이템 60 정확한 답이 필요하다면 float와 double은 피하라

제공되는 메서드

```
// 최대, 최소
num1.min(num2);
num1.max(num2);

// 소수점 맨 끝의 0 까지 비교
num1.equals(num2);

// 소수점 맨 끝의 0을 무시하고 비교
num1.compareTo(num2);

// 사칙연산
num1.add(num2);
num1.subtract(num2);
num1.multiply(num2);
num1.divide(num2);
num1.remainder(num2);
```

# Item 61

## 박싱된 기본 타입보다는 기본 타입을 사용하라

# 아이템 61 박싱된 기본 타입보다는 기본 타입을 사용하라

| 기본타입 | vs | 참조타입 |
| --- | --- | --- |
| • int<br>• double<br>• boolean |  | • String<br>• List<br>• 각 기본타입에 대응하는 참조타입 (Integer,Double...) |

# 아이템 61 박싱된 기본 타입보다는 기본 타입을 사용하라

박싱된 기본 타입의 잘못된 사용 예시


```
static Integer i;

public static void main(String[] args) {
    if (i == 42) {
        System.out.println("unbelievable");
    }
}
```

# 아이템 61 박싱된 기본 타입보다는 기본 타입을 사용하라

박싱된 기본 타입의 잘못된 사용 예시


inetger와 int를 비교할 때 오토 언박싱이 일어나고, 박싱된 기본 타입의 초깃값은 null이기 때문에 null인 참조를 언박싱하며 nullpointerexception이 발생한다.

# 아이템 61 박싱된 기본 타입보다는 기본 타입을 사용하라

박싱된 기본 타입을 사용해야 할 때

- 컬렉션의 원소, 키, 값으로 쓸 때 (컬렉션은 기본 타입 사용 불가)
- 타입 매개변수로 사용할 때
- 리플렉션을 통해 메서드를 호출할 때 등등..

# 아이템 61 박싱된 기본 타입보다는 기본 타입을 사용하라

## 결론

NOTE

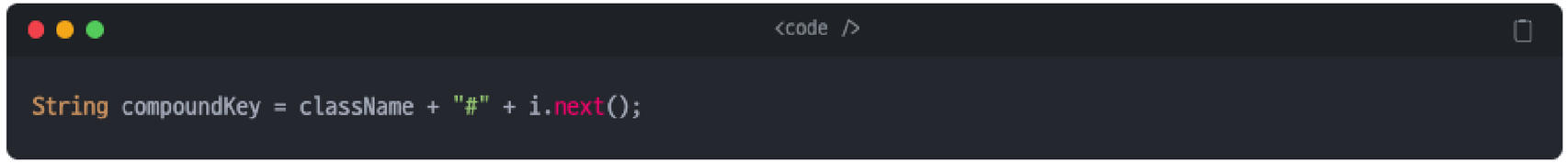
가능하면 기본 타입을 사용하자 .

# Item 62

## 다른 타입이 적절하다면 문자열 사용을 피하라

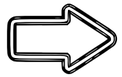
# 아이템 62 다른 타입이 적절하다면 문자열 사용을 피하라

**여러 요소가 혼합된 데이터를 하나의 문자열로**
**표현하는 것은 좋지 않다.**



# 아이템 62 다른 타입이 적절하다면 문자열 사용을 피하라

```
String compoundKey = className + "#" + i.next();
```



<code />

```
public class Compound {
    private CompoundKey compoundKey;

    private static class CompoundKey {
        private String className;
        private String delimiter;
        private int index;

        public CompoundKey(String className, String delimiter, int index{
            this.className = className;
            this.delimiter = delimiter;
            this.index = index;
        }
    }
}
```

# 아이템 62 다른 타입이 적절하다면 문자열 사용을 피하라

## 결론


아무때나 문자열 쓰지마라.

# Item 63

## 문자열 연결은 느리니 주의하라

# 아이템 63 문자열 연결은 느리니 주의하라

문자열 연결 연산자 +는 여러 문자열을 하나로 합쳐준다.

그렇다면 시간복잡도는?

# 아이템 63 문자열 연결은 느리니 주의하라

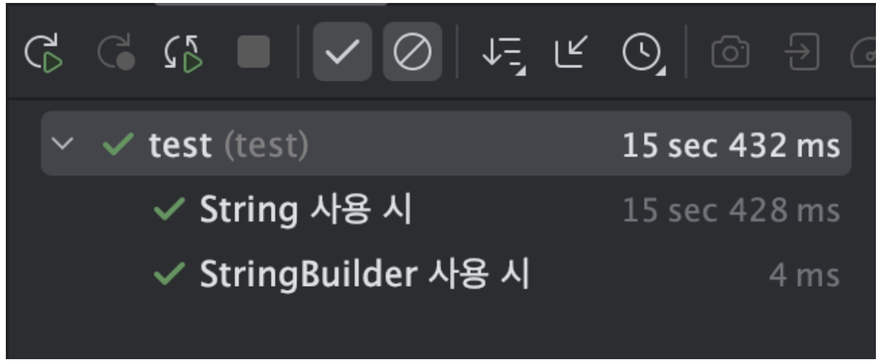
```
public String statement() {  1 usage
    String result = "";
    for (int i = 0; i < numItems(); i++) {
        result += lineForItem();
    }
    return result;
}

public String statement2() {  1 usage
    StringBuilder sb = new StringBuilder( capacity: numItems() * 80);
    for (int i = 0; i < numItems(); i++) {
        sb.append(lineForItem());
    }
    return sb.toString();
}
```

# 아이템 63 문자열 연결은 느리니 주의하라

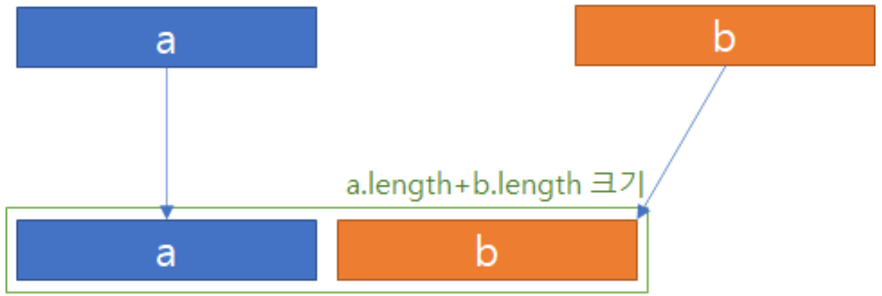
```java
public String statement() {  1 usage
    String result = "";
    for (int i = 0; i < numItems(); i++) {
        result += lineForItem();
    }
    return result;
}

public String statement2() {  1 usage
    StringBuilder sb = new StringBuilder( capacity: numItems() * 80);
    for (int i = 0; i < numItems(); i++) {
        sb.append(lineForItem());
    }
    return sb.toString();
}
```

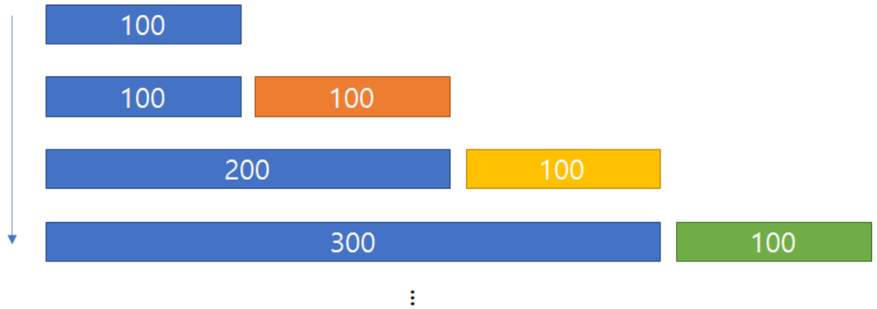


# 아이템 63 문자열 연결은 느리니 주의하라

String의 + 연산은 a의 길이 + b의 길이를 가지는 새로운 배열을 만들어서 a와 b를 모두 복사해넣고 사용한다.




반복문 실행시 다음과 같이 연산



# 아이템 63 문자열 연결은 느리니 주의하라

반면에 StringBuilder는 미리 일정한 크기의 배열을 잡아두고 거기에 String을 붙여나가는 방식이다.

# 아이템 63 문자열 연결은 느리니 주의하라

만약 미리 잡은 배열의 크기가 부족하다면 어떤 에러가 발생?

# 아이템 63 문자열 연결은 느리니 주의하라

만약 미리 잡은 배열의 크기가 부족하다면 어떤 에러가 발생?

배열의 크기가 가변이기 때문에 배열이 가득차게 되면 기존보다 2배 더 크게 새로운 배열을 만든다.

# 아이템 63 문자열 연결은 느리니 주의하라

**만약 멀티쓰레드 방식이라면 synchronized를 가진 StringBuffer를 사용하고,
join 또한 StringBuilder를 내부적으로 사용하므로 빠르다.**