Java: String vs StringBuilder ??

---

1. ??

자바에서 문자열을 다룰 수 있는 대표 클래스는 `String`과 `StringBuilder`이다. 두 클래스 모두 문자열을 표현하는 데 사용되지만, **불변성(immutability)**과 메모리 구조, 성능 면에서 근본적인 차이가 존재한다.

- `String`: 불변(immutable) 객체. 문자열을 수정하는 것처럼 보이지만 실제로는 새로운 객체가 생성됨.
- `StringBuilder`: 가변(mutable) 객체. 동일 인스턴스 내의 버퍼를 직접 수정하여 효율적인 메모리 사용과 빠른 성능을 제공함.

이러한 차이로 인해, 문자열 처리 방식에 따라 두 클래스의 선택은 전혀 달라져야 한다.

---

2. ??? ????? ??

2.1 String – ?? ??? ??? ??

`String` 클래스는 한 번 생성된 이후로 내용을 절대 변경할 수 없는 불변 객체이다. 문자열을 수정할 때마다 새로운 객체가 생성되어 Heap 메모리에 누적된다.

??

```
String s = "hi";
s = s + "!";  // "hi!"라는 새로운 String 객체가 생성됨
```

??? ??

```
[Heap Memory]
"hi"        ← 초기 객체
"hi!"       ← 새로운 객체 (변수 s가 참조)

→ 기존 객체는 GC 대상이 되기 전까지 Heap에 남아 있음
→ 반복적인 수정 시 객체 수 증가 → GC 부하 및 메모리 낭비
```

?? ??

- 내부적으로 `char[]` 배열을 `final`로 선언하여 불변성 보장
- 멀티스레드 환경에서 안전하게 공유 가능
- 문자열 상수 풀 (String Constant Pool)을 활용한 최적화 가능

2.2 StringBuilder – ?? ??? ??? ??

`StringBuilder`는 내부에 동적으로 크기 조절이 가능한 **char[]** 버퍼를 가지고 있으며, 문자열 조작 시 이 버퍼를 직접 수정한다. 불필요한 객체 생성이 없고, 성능과 메모리 측면에서 뛰어나다.

??

```
StringBuilder sb = new StringBuilder("hi");
sb.append("!");
System.out.println(sb);  // 출력: hi!
```

??? ??

```
[Heap Memory]
StringBuilder 객체
 └─ char[] buffer = ['h', 'i', '!'] ← 내부 버퍼가 직접 수정됨

→ 동일 객체 내에서 작업 → 메모리 재사용
→ 객체 수 증가 없음 → GC 부담 없음
```

?? ?? ??

```
public final class StringBuilder {
    char[] value;
    int count;

    public StringBuilder append(String str) {
        ensureCapacityInternal(count + str.length());
        str.getChars(0, str.length(), value, count);
        count += str.length();
        return this;
    }
}
```

---

3. ?? ?? ??

3.1 ?? ?? ???

```
// String (비효율적 방식)
String s = "";
for (int i = 0; i < 1000; i++) {
    s += "a";  // 매번 새로운 객체 생성됨
}
```

```
// StringBuilder (효율적 방식)
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 1000; i++) {
    sb.append("a");  // 내부 버퍼만 수정됨
}
```

3.2 ?? ??

항목	**String**	**StringBuilder**
메모리 구조	불변 객체 , 매번 새 객체 생성	가변 객체 , 하나의 버퍼 사용
연결 방식	`+` 또는 `concat()` 호출마다 새 객체	`append()` 는 동일 객체에 누적
GC 부하	심함	적음
속도	느림	빠름

결론 : 루프나 재귀에서 문자열을 누적해야 할 경우 , `StringBuilder`가 수십 배 빠를 수 있음 .

3.3 ??? ??? ??

- `String`: 불변 → 스레드 안전
- `StringBuilder`: 가변 + 동기화 없음 → 스레드 안전하지 않음
- 다중 스레드에서 문자열 조작이 필요하다면 → `StringBuffer` 사용 고려

```
// 멀티스레드 환경에서 StringBuilder는 위험
StringBuilder sb = new StringBuilder();
```

```
Runnable task = () -> {
    for (int i = 0; i < 1000; i++) {
        sb.append("x");  // 동기화 안 되어 경합 발생 가능
    }
};
```

→ 해결책 : `StringBuffer` 또는 `synchronized` 블록 사용

---

4. ?? ? ?? ?????

4.1 ?? ???? ?????

사용 상황	적절한 클래스	이유
고정된 문자열 / 변경 없음	`String`	불변성으로 안전하고 , 메모리 캐싱 가능
루프 내 문자열 누적 / 대량 조작	`StringBuilder`	메모리 효율 및 성능 우수
멀티스레드 환경에서 동기화가 필요한 경우	`StringBuffer`	메서드 동기화로 스레드 안전성 확보

4.2 ?? ?

- ⚠ `String`으로 반복 연결하지 말 것 (예 : `s = s + "a"` X)
- □ □□ 처리 시에는 반드시 `StringBuilder` 사용
- □ □□□□□ 환경에서는 `StringBuffer` 또는 동기화 처리 필수

Revision #11
Created 20 May 2025 02:35:35 by Dain
Updated 20 May 2025 06:21:38 by Dain