ApiStringBuffer, ApiStringBuilder

문자열 String이 아닌 StringBuffer, StringBuilder를 사용하는 이유

- String을 수정하게 되면 힙 메모리 영역에 새로운 String 객체가 만들어지고,

가비지 컬랙터가 기존 String 객체를 지우는 등, 메모리적으로 비효율적이다.

- 문자열을 수정해야하는 경우 StringBuffer 나 StringBuildeer를 이용하는 것이 권장된다.

StringBuffer 사용법

1. StringBuffer 객체 생성

```
StringBuffer strBuff = new StringBuffer();
        System.out.println(strBuff); // "" 와 같음 (empty)
```

2. 문자열 추가 .append

```
strBuff.append("안녕하세요");
        System.out.println(strBuff); // "안녕하세요"와 같음
```

3. 문자열 비우기

```
strBuff = new StringBuffer(); // 다시 빈문자열로 생성
        System.out.println(strBuff); // "" 와 같음 (empty)
```

4. 선언할 때 문자열 넣기

```
// 선언할 때 문자열 넣기
        strBuff = new StringBuffer("선언과 동시에 문자열 넣어보기")
        System.out.println(strBuff);
        System.out.println(strBuff.toString()); // toString으
```

5. 수정하여 완성된 문자열을 String타입으로 담기 .toString()

```
String content = strBuff.toString();
        System.out.println(content);
```

String과 StringBuffer의 성능 체크

```
String text = "";
 long before = System.currentTimeMillis();
        for (int i = 0; i < 1000000; i++){
            // 0~9 사이의 랜덤숫자 생성 후 text에 이어붙이기
            int rand = (int)(Math.random()*10);
            text += rand;

            // 5만번 마다 콘솔창에 현재 진행사항 출력
            if (i % 50000 == 0){
                System.out.println(i + "...");
            }
        }
        long after = System.currentTimeMillis();
        long diff = after - before;
        diff /= 1000;
        System.out.println(diff + "초 걸림");
```

```
// String타입인 text에 랜덤한 숫자를 1000000 이어붙이는 경우 62초 가량


StringBuffer sb = new StringBuffer();
before = System.currentTimeMillis();
for (int i = 0; i < 1000000; i++){
      // 0~9 사이의 랜덤숫자 생성 후 text에 이어붙이기
      sb.append(rand);
// StringBuffer 타입인 sb는 순식간에 처리가 완료됨
```

StringBuilder 사용법

1. 객체 생성

```
StringBuilder strBuild = new StringBuilder();
```

2. StringBuilder에 문자열 추가

```
strBuild.append("추가");
```

→ StringBuffer와 StringBuilder는 생성, 추가, 초기화 방법이 동일하다.

StringBuffer와 StringBuilder의 차이점

StringBuffer

StringBuilder

- 동기화 지원

쓰레드 A,B,C에서 각각의 쓰레드들이 하나의 StringBuffer 객체를 사용할 때

쓰레드 A가 StringBuffer 객체를 수정한 경우,

다른 쓰레드 B, C에서 이 사실을 안다.

(=안정적)

- 단일 쓰레드에서는 다소 낮은 성능을 보인다.

- 동기화 미지원

쓰레드 A,B,C에서 각각의 쓰레드들이 하나의 StringBuilder 객체를 사용할 때, 쓰레드 A가 StringBuilder 객체를 수정한 경우, 다른 쓰레드 B,C에서 이 사실을 모를 수도 있다.

(= 문제가 발생할 수 있음)

- 단일 쓰레드에서 좀 더 좋은 성능을 보인다.

Q. 동기화 지원이란?

→ Multi Thread 환경에서 해당 객체가 모든 Thread에서 안정적으로 사용된다는 것을 의미.

퀴즈_UniqueId 메소드

Q.02

가끔 우리가 데이터베이스에 데이터를 추가할때, 유니크한 아이디를 넣어서 추가해야하는 경우가 많다. 그때 사용하는게 UUID.randomUUID().toString() 인데, 우리가 따로 유니크한 아이디를 만드는 메소드를 만들어보자.

메소드 실행시 현재 날짜에 대한 yyMMddHHmmssSSS 포맷의 문자열과
이후 랜덤한 숫자 6자리를 뒤에 덧붙인 21자리의 문자열이 리턴된다. (실제로 이 방법도 쓰임)

실행 예시

```
String uniqueId = makeUniqueId();
System.out.println(uniqueId);
```

실행 결과

```
231027104651805112876
```

231027104651805 는 23년 10월 27일 10시 46분 51초 805밀리초에 대한 숫자이며
뒤에 112876은 랜덤으로 생성된 숫자 6자리이다.

- StringBuffer와 Math.random을 이용할 것.

```
      String uniqueId = makeUniqueId();
      System.out.println(uniqueId);
```

```
Date date = new Date();
        // 받은 날짜~밀리초를 원하는 포맷형식의 문자열로 변경
      String uniqueId = makeUniqueId();
      System.out.println(uniqueId);

    }

   static String makeUniqueId() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyMMddHH
        String inputDate = sdf.format(new Date());
        // inputDate에 6자리 랜덤 숫자 추가
```

```
        StringBuffer rst = new StringBuffer(inputDate);
        for (int i = 0; i < 6; i++) {
           rst.append((int) (Math.random() * 10));
        }

     //   return inputDate;
return rst.toString();
```