● String클래스: 문자열을 저장하고 이를 다루는데 필요한 메서드를 함께 제공하는 클래스

▶ 변경 불가능한 클래스(immutable class)와 덧셈연산자: String클래스에는 문자열를 저장하기 위해서 문자형 배열 참조변수(char[]) value를 인스턴스 변수로 정의해두고 있다. 인스턴스 생성 시 생성자의 매개변수로 입력받는 문자열은 이 인스턴스 변수(value)에 문자형 배열(char[])로 저장되는 것이다.

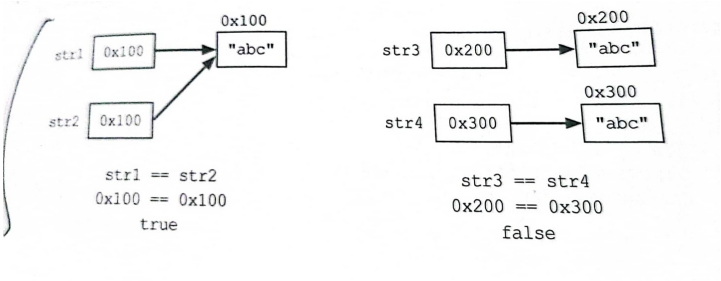
```
public final class String implements java.io.Serializable, Comparable {
    private char[] value;
}

// String클래스는 앞에 final이 붙어 있으므로 다른 클래스의 조상이 될 수 없다.
```

한번 생성된 String인스턴스가 갖고 있는 문자열은 읽어 올 수 만 있고, 변경할 수는 없다. '+'연사자를 이용해서 문자열을 결합하는 경우, 인스턴스 내의 문자열이 바뀌는 것이 아니라 새로운 문자열이 담긴 String인스턴스가 생성되는 것이다. 덧셈연산자'+'를 사용해서 문자열을 결합하는 것은 이처럼 연산 시 마다 새로운 문자열을 가진 String인스턴스가 생성되어 메모리공간을 차지하게 되므로 가능한 한 결합횟수를 줄이는 것이 좋다. 문자열간의 결합이나 추출 등 문자열을 다루는 작업이 많이 필요한 경우에는 StringBuffer클래스를 사용하는 것이 좋다 — StringBuffer인스턴스에 저장된 문자열은 변경이 가능하므로 하나의 StringBuffer인스턴스만으로도 문자열을 다루는 것이 가능하기 때문이다.

▶문자열을 만드는 두 가지 방법과 비교: 문자열을 만드는 방법에는 문자열 리터럴을 지정하는 방법과 String클래스의 생성자를 사용하는 방법이 있다.

```
String str1 = "abc";
String str2 = "abc";
String str3 = new String("abc");
String str4 = new String("abc");
```

String클래스의 생성자를 이용한 경우에는 new연산자에 의해서 메모리할당이 이루어지기 때문에 항상 새로운 String인스턴스가 생성된다. 그러나 문자열 리터럴은 이미 존재하는 것을 재사용하는 것이다.



equals()를 사용했을 때는 두 문자열의 내용("abc")를 비교하기 때문에 두 경우 모두 true를 결과로 얻지만, 등가비교연산자"=="로 각 String인스턴스의 주소를 비교했을 때는 겨로가가 다르다. 문자열 리터럴을 지정할 경우 Heaㅣp의 String constant pool에 "abc"를 담고 있는 String인스턴스가 하나 생성 후 이미 같은 값을 가지고 있는 리터럴이 존재할 경우 새로운 객체를 생성하지 않고 기존에 존재하는 리터럴을 사용한다. 따라서 참조변수 str1와 str2가 모두 같은 String인스턴스를 참조하기 때문에 (이 String인스턴스의 주소를 저장하기 때문에) 결과가 true로 나오지

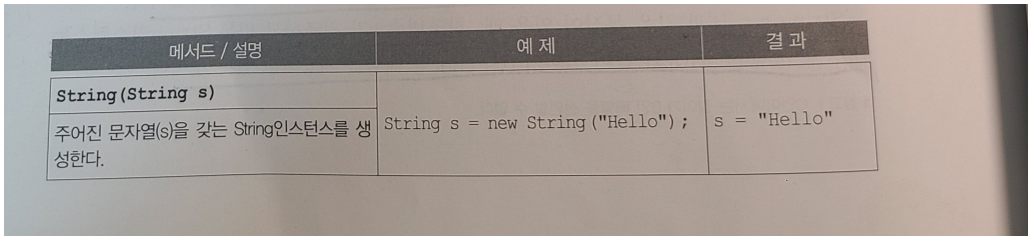
만, String클래스의 생성자를 사용해서 만들 경우 new연산자에 의해서 String인스턴스를 할 때마다 Heap메모리에 새로운 주소를 할당받기 때문에, 참조변수 str3와 str4가 각각 다른 객체의 주소를 참조하여 결과가 false로 나온다.

▶ 문자열 리터럴: 자바 소스파일에 포함된 모든 **문자열 리터럳은 컴파일 시에 클래스 파일에 저장**된다. 이때 같은 내용의 문자열 리터럴은 한 번만 저장된다. 문자열 리터럴도 String인스턴스이고, 한번 생성하면 내용을 변경할 수 없으니 하나의 인스턴스를 공유하면 되기 때문이다. 클래스 파일에는 소스파일에 포함된 모든 리터럴의 목록이 있고, 해당 **클래스 파일이 클래스 로더에 의해 메모리에 올라갇 때**, 이 리터럴 목록에 있는 리터럴들이 JVM의 **Heap 내의 String 상수 저장소(String constant pool)에 저장**된다. 이 때, 이곳에 "abc"와 같은 문자열 리터럴이 자동적으로 생성되어 저장되는 것이다.

▶ 빈 문자열(empty string): char형 배열도 길이가 0인 배열을 생성할 수 있고, 이 배열을 내부적으로 가지고 있는 문자열이 바로 빈 문자열이다. 'String s ="";'와 같은 문장이 있을 때, 참조변수 s가 참조하고 있는 String인스턴스는 내부에 'new char[0]'과 같이 길이가 0인 char형 배열을 저장하고 있는 것이다.

```
class Ex {
    public static void main(String[] args) {
        char[] chArr = new char[0]; // 길이가 0인 배열을 생성해서 char형 배열 참조변수 chAr
        char[] chA = {};
        char c = ' '; // 하지만 char형 변수에는 공백일지라도 반드시 하나의 문자를 지정해야 한
        // '\u0000'은 유니코드의 첫 번째 문자로서 아무런 문자도 지정되지 않은 빈 문자이다
        char cc = '\u0000';

        String s = "";
        // 참조변수 s가 참조하고 있는 String인스턴스는 내부에 'new char[0]'과 같이
        // 길이가 0인 char형 배열을 저장하고 있다
        String chs = new String(chA);
        String chAr = new String(chArr);
        String ss = new String("");
    }
}
```

▶ String클래스의 생성자와 메서드



메서드 / 설명	예 제	결 과
String(String s) 주어진 문자열(s)을 갖는 String인스턴스를 생성한다.	String s = new String("Hello");	s = "Hello"

String(char[] value) 주어진 문자열(value)을 갖는 String인스턴스를 생성한다.	`char[] c={'H','e','l','l','o'};` `String s = new String(c);`	`s = "Hello"`

String(StringBuffer buf)
StringBuffer인스턴스가 갖고 있는 문자열과 같은 내용의 String인스턴스를 생성한다.
	`StringBuffer sb =`
	`  new StringBuffer("Hello");`
	`String s = new String(sb);`
		`s = "Hello"`

char charAt(int index)
지정된 위치(index)에 있는 문자를 알려준다. (index는 0부터 시작)
	`String s = "Hello";`
	`String n = "0123456";`
	`char c  = s.charAt(1);`
	`char c2 = n.charAt(1);`
		`c  = 'e'`
		`c2 = '1'`

int compareTo(String str)
문자열(str)과 사전순서로 비교한다. 같으면, 0을, 사전순으로 이전이면 음수를, 이후면 양수를 반환한다.
	`int i  ="aaa".compareTo("aaa");`
	`int i2="aaa".compareTo("bbb");`
	`int i3="bbb".compareTo("aaa");`
		`i  = 0`
		`i2 = -1`
		`i3 = 1`

String concat(String str) 문자열(str)을 뒤에 덧붙인다.	`String s = "Hello";` `String s2=s.concat(" World");`	`s2="Hello World"`
boolean contains(CharSequence s) 지정된 문자열(s)이 포함되었는지 검사한다.	`String s  = "abcedfg";` `boolean b = s.contains("bc");`	`b = true`
boolean endsWith(String suffix) 지정된 문자열(suffix)로 끝나는지 검사한다.	`String file = "Hello.txt";` `boolean b=file.endsWith("txt");`	`b = true`

boolean equals(Object obj)
매개변수로 받은 문자열(obj)과 String인스턴스의 문자열을 비교한다. obj가 String이 아니거나 문자열이 다르면 false를 반환한다.
	`String s = "Hello";`
	`boolean b  =s.equals("Hello");`
	`boolean b2=s.equals("hello");`
		`b  = true`
		`b2 = false`

boolean equalsIgnoreCase(String str)
문자열과 String인스턴스의 문자열을 대소문자 구분없이 비교한다.
	`String s = "Hello";`
	`boolean b  =`
	`  s.equalsIgnoreCase("HELLO");`
	`boolean b2 =`
	`  s.equalsIgnoreCase("heLLo");`
		`b  = true`
		`b2 = true`

int indexOf(int ch)
주어진 문자(ch)가 문자열에 존재하는지 확인하여 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.(index는 0부터 시작)
	`String s = "Hello";`
	`int idx1 = s.indexOf('o');`
	`int idx2 = s.indexOf('k');`
		`idx1 = 4`
		`idx2 = -1`

int indexOf(int ch, int pos)
주어진 문자(ch)가 문자열에 존재하는지 지정된 위치(pos)부터 확인하여 위치(index) 를 알려준다. 못 찾으면 -1을 반환한다. (index는 0부터 시작)
	`String s = "Hello";`
	`int idx1 = s.indexOf('e', 0);`
	`int idx2 = s.indexOf('e', 2);`
		`idx1 = 1`
		`idx2 = -1`

int indexOf(String str) 주어진 문자열이 존재하는지 확인하여 그 위치(index)를 알려준다. 없으면 -1을 반환한다.(index는 0부터 시작)	`String s = "ABCDEFG";` `int idx = s.indexOf("CD");`	`idx = 2`

String intern()
문자열을 상수풀(constant pool)에 등록한다. 이미 상수풀에 같은 내용의 문자열이 있을 경우 그 문자열의 주소값을 반환한다.
	`String s  = new String("abc");`
	`String s2 = new String("abc");`
	`boolean b = (s==s2);`
	`boolean b2 = s.equals(s2);`
	`boolean b3 =`
	`  (s.intern()==s2.intern());`
		`b  = false`
		`b2 = true`
		`b3 = true`

int lastIndexOf(int ch)
지정된 문자 또는 문자코드를 문자열의 오른쪽 끝에서부터 찾아서 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.
	`String s = "java.lang.Object";`
	`int idx1 = s.lastIndexOf('.');`
	`int idx2 = s.indexOf('.');`
		`idx1 = 9`
		`idx2 = 4`

int lastIndexOf(String str)
지정된 문자열을 인스턴스의 문자열 끝에서부터 찾아서 위치(index)를 알려준다. 못 찾으면 -1을 반환한다.
	`String s = "java.lang.java";`
	`int idx1 =`
	`s.lastIndexOf("java");`
	`int idx2 = s.indexOf("java");`
		`idx1 = 10`
		`idx2 = 0`

int length() 문자열의 길이를 알려준다.	`String s = "Hello";` `int length = s.length();`	`length = 5`
String replace(char old, char nw) 문자열 중의 문자(old)를 새로운 문자(nw)로 바꾼 문자열을 반환한다.	`String s = "Hello";` `String s1 = s.replace('H','C');`	`s1 = "Cello"`

String replace(CharSequence old, CharSequence nw)
문자열 중의 문자열(old)을 새로운 문자열(nw)로 모두 바꾼 문자열을 반환한다.
	`String s = "Hellollo";`
	`String s1 =`
	`s.replace("ll","LL");`
		`s1="HeLLoLLo"`

String replaceAll(String regex, String replacement)
문자열 중에서 지정된 문자열(regex)과 일치하는 것을 새로운 문자열(replacement)로 모두 변경한다.
	`String ab = "AABBAABB";`
	`String r =`
	`ab.replaceAll("BB","bb");`
		`r = "AAbbAAbb"`

String replaceFirst(String regex, String replacement)
문자열 중에서 지정된 문자열(regex)과 일치 하는 것 중, 첫 번째 것만 새로운 문자열(replacement)로 변경한다.
	`String ab = "AABBAABB";`
	`String r =`
	`ab.replaceFirst("BB","bb");`
		`r = "AAbbAABB"`

String[] split(String regex)
문자열을 지정된 분리자(regex)로 나누어 문자열 배열에 담아 반환한다.
	`String animals =`
	`"dog,cat,bear";`
	`String[] arr =`
	`animals.split(",");`
		`arr[0] = "dog"`
		`arr[1] = "cat"`
		`arr[2] ="bear"`

String[] split(String regex, int limit)
문자열을 지정된 분리자(regex)로 나누어 문자열배열에 담아 반환한다. 단, 문자열 전체를 지정된 수(limit)로 자른다.
	`String animals =`
	`"dog,cat,bear";`
	`String[] arr =`
	`animals.split(",",2);`
		`arr[0] = "dog"`
		`arr[1]=`
		`"cat,bear"`

boolean startsWith(String prefix)
주어진 문자열(prefix)로 시작하는지 검사한다.
	`String s = "java.lang.Object";`
	`boolean b = s.startsWith("java");`
	`boolean b2=s.startsWith("lang");`
		`b = true`
		`b2 = false`

String substring(int begin)
String substring(int begin, int end)
주어진 시작위치(begin)부터 끝 위치(end) 범위에 포함된 문자열을 얻는다. 이 때, 시작위치의 문자는 범위에 포함되지만, 끝 위치의 문자는 포함되지 않는다. (begin ≤ x < end)
	`String s = "java.lang.Object";`
	`String c = s.substring(10);`
	`String p = s.substring(5,9);`
		`c = "Object"`
		`p = "lang"`

String toLowerCase() String인스턴스에 저장되어있는 모든 문자열을 소문자로 변환하여 반환한다.	`String s = "Hello";` `String s1 = s.toLowerCase();`	`s1 = "hello"`
String toString() String인스턴스에 저장되어 있는 문자열을 반환한다.	`String s = "Hello";` `String s1 = s.toString();`	`s1 = "Hello"`

String toUpperCase() String인스턴스에 저장되어있는 모든 문자열을 대문자로 변환하여 반환한다.	String s = "Hello"; String s1 = s.toUpperCase();	s1 = "HELLO"
String trim() 문자열의 왼쪽 끝과 오른쪽 끝에 있는 공백을 없앤 결과를 반환한다. 이 때 문자열 중간에 있는 공백은 제거되지 않는다.	String s = " Hello World "; String s1 = s.trim();	s1="Hello World"
static String valueOf(boolean b) **static String valueOf(char c)** **static String valueOf(int i)** **static String valueOf(long l)** **static String valueOf(float f)** **static String valueOf(double d)** **static String valueOf(Object o)** 지정된 값을 문자열로 변환하여 반환한다. 참조변수의 경우, toString()을 호출한 결과를 반환한다.	String b=String.valueOf(true); String c = String.valueOf('a'); String i = String.valueOf(100); String l=String.valueOf(100L); String f = String.valueOf(10f); String d=String.valueOf(10.0); java.util.Date dd = new java.util.Date(); String date = String.valueOf(dd);	b = "true" c = "a" i = "100" l = "100" f = "10.0" d = "10.0" date = "Wed Jan 27 21:26:29 KST 2016"

| 참고 | CharSequence는 JDK1.4부터 추가된 인터페이스로 String, StringBuffer 등의 클래스가 구현하였다.
| 참고 | contains(CharSequence s), replace(CharSequence old, CharSequence nw)는 JDK1.5부터 추가되었다.
| 참고 | java.util.Date dd = new java.util.Date();에서 생성된 Date인스턴스는 현재 시간을 갖는다.

▶ join(): 여러 문자열 사이에 구분자를 넣어서 결합한다.

```
String animals = "dog,cat,bear";
String[] animal_arr = animals.split(","); // 문자열을 ',' 구분자로 나눠서 배열에 저장
String str = String.join("-", animal_arr); // 배열의 문자열을 '-'로 구분해서 결합
System.out.println(str); // dog-cat-bear
```

▶ StringJoiner: java.util.StringJoiner클래스를 이용해서 문자열을 결합한다.

```
StringJoiner sj = new StringJoiner(",", "[", "]");
String[] strArr = {"aaa", "bbb", "ccc"};

for (String s : strArr) {
    sj.add(s);
}

System.out.println(sj.toString()); // [aaa,bbb,ccc]
```

▶ 유니코드의 보충문자: 유니코드는 2 byte(16 bit)문자 체계인데, 20비트로 확장하게 되었다. 확장에 의해 새로 추가된 문자들을 '보충 문자(supplementary characters)'이라고 하고, 이들을 char타입으로 다루지 못해 int타입으로 다룰 수 밖에 없게 되었다. String클래스의 메서드 중에 매개변수가 'int ch'인 것들은 보충문자를 지원하는 것이고, 'char ch'인 것들을 지원하지 않는 것들이다.

▶ 문자 인코딩 변환: getBytes(Stirng charsetName()을 사용하면, 문자열의 문자 인코딩을 다른 인코딩으로 변경할 수 있다. 자바가 UTF-16을 사용하지만, 문자열 리터럴에 포함되는 문자들은 OS의 인코딩을 사용한다. 한글 윈도우즈의 경우 인코딩으로 CP949를 사용하면, UTF-8로 변경하려면 아래와 같이 해야 한다.

```
byte[] utf8_str = "가".getBytes("UTF-8");  // 문자열을 UTF-8로 변환
String str = new String(utf8_str, "UTF-8");  // byte배열을 문자열로 변환
```

UTF-8은 한글 한 글자를 3 byte로 표현하고, CP949는 2byte로 표현한다. 서로 다른 문자 인코딩을 사용하는 컴퓨터 간에 데어터를 주고받을 때는, 적절한 문자 인코딩이 필요하다.

▶ String.format(): 형식화된 문자열을 만들어내는 방법 (printf와 사용방법 동일)

```
String str = String.format("%d 더하기 %d는 %d입니다.", 1, 2, 1+2);
System.out.println(str); // 1 더하기 2는 3입니다.
```

▶ 기본형 값을 String으로 변환: String.valueOf() or 빈문자열 "" 더하기 중에 성능은 valueOf()가 더 좋다.

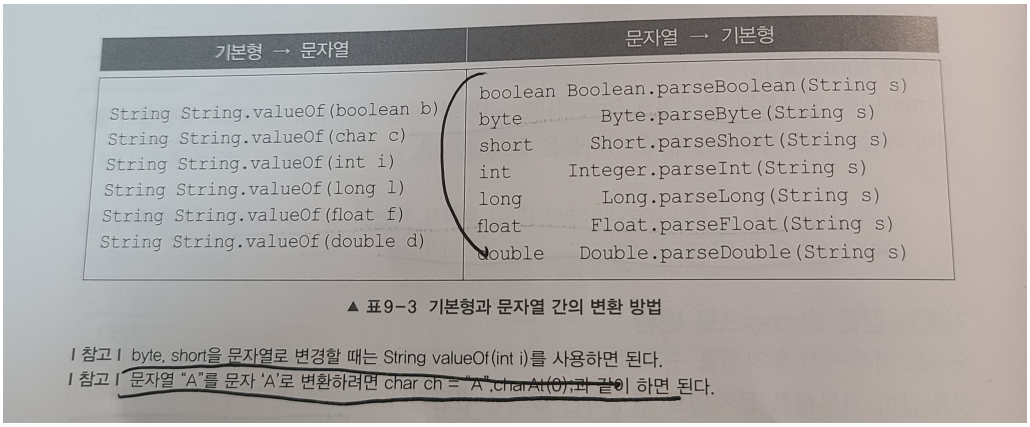
```
int i = 100;
String str1 = i + "";              // 100을 "100"으로 변환하는 방법1
String str2 = String.valueOf(i);   // 100을 "100"으로 변환하는 방법2
```

▶ String을 기본형 값으로 변환: 기본형.parse기본형() or 기본형.valueOf().

```
int i = Integer.parseInt("100");  // "100"을 100으로 변환하는 방법1
int i2 = Integer.valueOf("100"); // "100"을 100으로 변환하는 방법2
```

원래 valueOf()의 반환타입은 int가 아니라 Integer이지만, 오토박싱(auto-boxing)에 의해 Integer가 int로 자동 변환된다. Integer, Boolean, Byte와 같이 기본형 타입의 이름의 첫 글자가 대문자인 것은 래퍼 클래스(wrapper class)로 기본형 값을 감싸고 있는 클래스라는 뜻에서 붙여진 이름으로 기본형을 클래스로 표현한 것이다. valueOf(String s)는 메서드 내부에서 parseInt(String s)을 호출하므로, 두 메서드는 반환 타입만 다른 같은 메서드다.

```
public static Integer valueOf(String s) throws NumberFormatException {
    return Integer.valueOf(parseInt(s, 10)); // 여기서 10은 10진수를 의미. parseInt(s)와 같
}
```



▲ 표9-3 기본형과 문자열 간의 변환 방법

| 참고 | byte, short을 문자열로 변경할 때는 String valueOf(int i)를 사용하면 된다.
| 참고 | 문자열 "A"를 문자 'A'로 변환하려면 char ch = "A".charAt(0);과 같이 하면 된다.

parseInt()나 parseFloat()같은 메서드는 문자열에 공백 또는 문자가 포함되어 있는 경우, 변환 시 예외(NumberFormatException)가 발생할 수 있으므로 주의해야 한다. 그래서 문자열 양끝의 공백을 제거하는 trim()을 습관적으로 같이 사용하기도 한다.

```
int val = Integer.parseInt(" 123 ".trim()); // 문자열 양끝의 공백을 제거 후 변환
```

그러나 부호를 의미하는 "+"나 소수점을 의미하는 "."와 float형 값을 뜻하는 f와 같은 자료형 접미사는 허용된다. 예를 들면 Integer.parseInt("+" + "100")의 경우 문자 "100"만 숫자 100으로 변환된다.

Integer클래스의 static int parseInt(String s, int radix)를 사용하면 16진수 값으로 표현된 문자열도 변환할 수 있기 때문에, 대소문자 구별 없이 a, b, c, d, e, f도 사용할 수 있다. int result = Integer.parseInt("a", 16)의 경우 result에는 정수 값 10이 저장된다 (16진수 a는 10진수로는 10).

▶ substring(int start, int end): 한 문자열에서 내용의 일부를 추출하는 메서드. 매개변수로 사용되는 문자열에서 각 문자의 위치를 뜻하는 index가 0부터 시작한다는 것과, start부터 end의 범위 중 end위치에 있는 문자는 결과에 포함되지 않는다. end에서 start값을 빼면 substring에 의해 추출된 글자의 수가 된다.

```
class StringEx7 {
    public static void main(String[] args) {
        String fullName = "Hello.java";

        // fullName에서 '.'의 위치를 찾는다.
        int index = fullName.indexOf('.');

        // fullName의 첫번째 글자부터 '.'이 있는 곳까지 문자열을 추출한다.
        String fileName = fullName.substring(0, index);

        // '.'의 다음 문자 부터 시작해서 문자열의 끝까지 추출한다.
        // fullName.substring(index+1, fullName.length());의 결과와 같다. (길이가 10)
        String ext = fullName.substring(index+1);

        System.out.println(fullName + "의 확장자를 제외한 이름은 " + fileName); // Hello
        System.out.println(fullName + "의 확장자는 " + ext); // java
    }
}
```

Resources: 자바의 정석