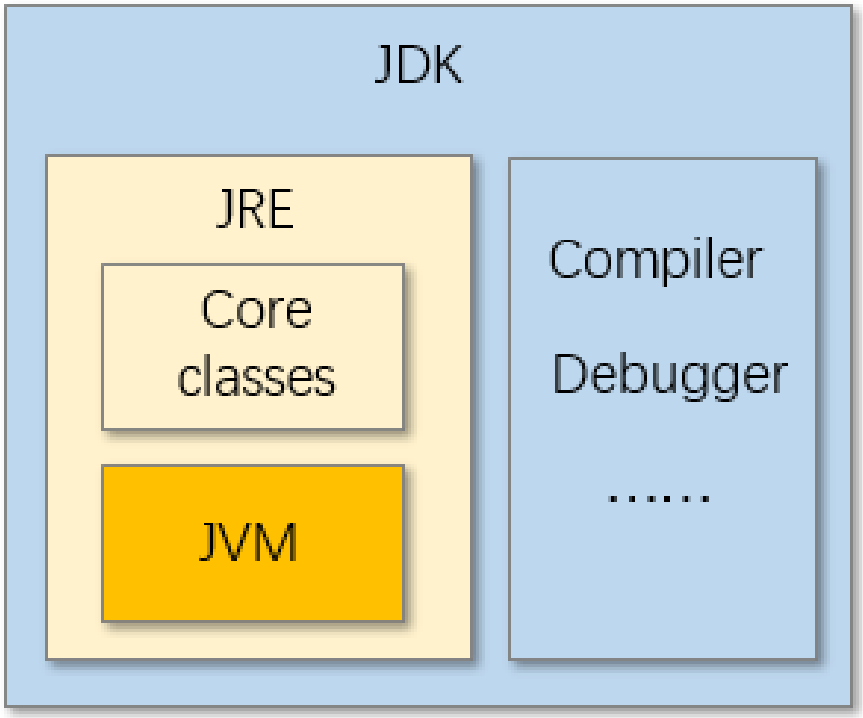
Chapter 7: String

TAO Yida

taoyd@sustech.edu.cn

`String` is a core class frequently used in practice



Objectives

- Immutable character-string objects of class `String`
- Mutable character-string objects of class `StringBuilder`

The `String` Type

- `String` represents a string of characters
- `String` is a predefined class in Java.
- `String` is a *reference type*

String, like any class, has fields, constructors and methods

⇩

```
char[] value;
```

Creating String Objects (Instantiation)

- `String` objects can be created by using the `new` keyword and various `String` constructors
  - `String s1 = new String("hello world");`
  - `String s2 = new String(); // empty string (length is 0)`
  - `String s3 = new String(s1);`
  - `char[] charArray = {'h', 'e', 'l', 'l', 'o'};`
  - `String s4 = new String(charArray);`
  - `String s5 = new String(charArray, 3, 2); // string “lo”`

Offset

Count

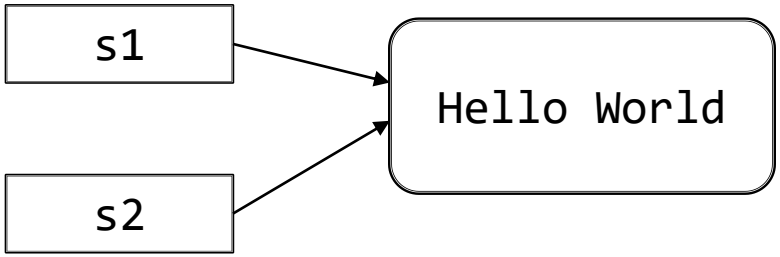
More at: https://docs.oracle.com/javase/10/docs/api/java/lang/String.html

Creating String Objects (Instantiation)

- `String` objects can also be created by string literals (字面常量, a sequence of characters in double quotes)

```
String s1 = “Hello World";

String s2 = s1;
```



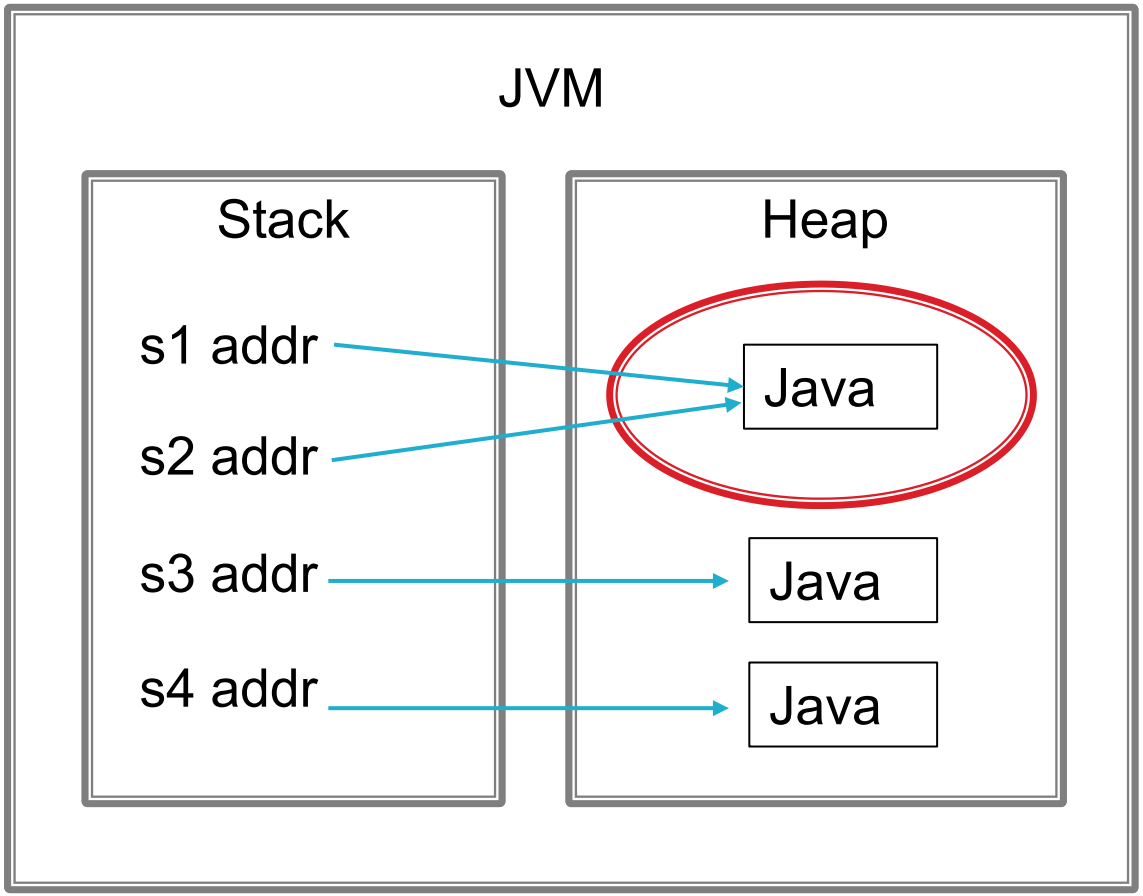
Using String literal vs new keyword

String Constant Pool:
Store string objects created by string literals

```
String s1 = "Java";
String s2 = "Java";

String s3 = new String("Java");
String s4 = new String("Java");
```

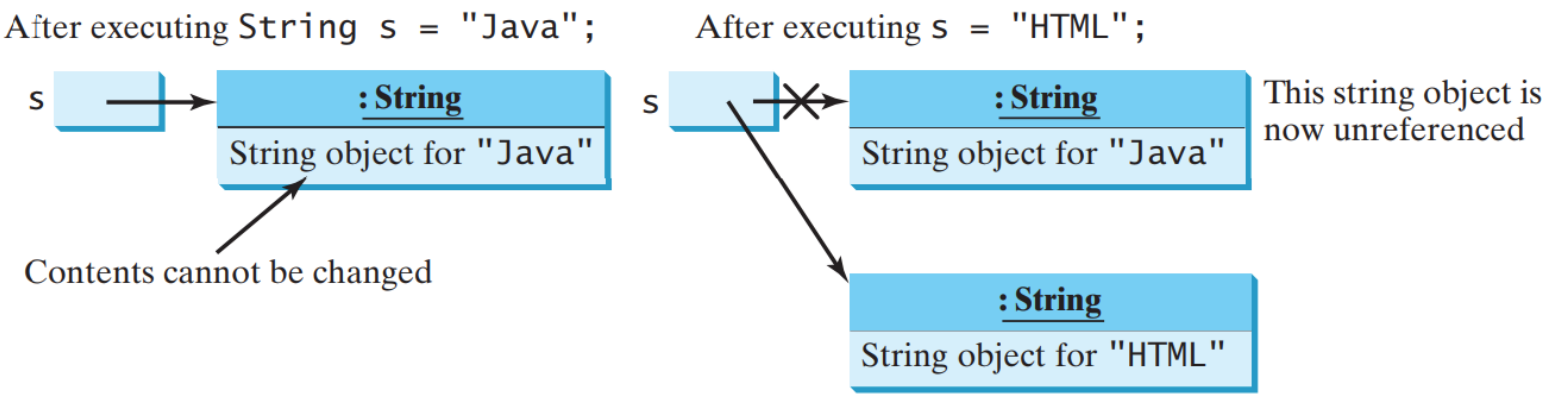
```
System.out.println(s1 == s2); // true
System.out.println(s3 == s4); // false
```



Immutability（不可变性）

- In Java, `String` objects are immutable: their values cannot be changed after they are created.
- Any modification creates a new `String` object

```
String s = "Java";
s = "HTML";
```



String Methods

Instance methods that can be invoked on specific objects. Calling them requires a non-null object reference.

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method
char	charAt(int index)
IntStream	chars()
int	codePointAt(int index)
int	codePointBefore(int index)
int	codePointCount(int beginIndex, int endIndex)
IntStream	codePoints()
int	compareTo(String anotherString)
int	compareToIgnoreCase(String str)
String	concat(String str)
boolean	contains(CharSequence s)
boolean	contentEquals(CharSequence cs)
boolean	contentEquals(StringBuffer sb)
Optional<String>	describeConstable()
boolean	endsWith(String suffix)
boolean	equals(Object anObject)
boolean	equalsIgnoreCase(String anotherString)
String	formatted(Object... args)
byte[]	getBytes()

https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/String.html

The Method `length`

`int length()`	Returns the length of this string.

```
public class StringExamples {
    public static void main(String[] args) {
        String s1 = "hello world";
        System.out.printf("s1: %s", s1);
        System.out.printf("\nLength of s1: %d", s1.length());
    }
}
```

```
s1: hello world
Length of s1: 11
```

The Method charAt

`char charAt(int index)`	Returns the `char` value at the specified index.

```
public class StringExamples {
    public static void main(String[] args) {
        String s1 = "hello world";
        System.out.printf("s1: %s", s1);

        for(int count = s1.length() - 1; count >=0; count--) {
            System.out.printf("%c", s1.charAt(count));
        }
    }
}
```

What's the output?

Comparing Strings

- When primitive-type values are compared with ==, the result is `true` if both values are identical.

```
int a = 2, b =2;
if (a == b) System.out.println("a = b"); // prints a = b
```

- When references (memory addresses) are compared with ==, the result is `true` if both references refer to the same object in memory.

```
String s1 = "Hello World";
String s2 = "Hello World";
if(s1 == s2) System.out.println("s1 = s2"); // prints s1 = s2
```

Comparing Strings

```
String s1 = "Hello World";
String s2 = s1 + "";
if(s1 == s2) System.out.println("s1 = s2"); // prints s1 = s2?
```

- No. The condition will evaluate to `false` because the `String` variables `s1` and `s2` refer to two different `String` objects, although the strings contain the same sequence of characters.
- To compare the actual contents (or state information) of objects (strings are objects) for equality, a method `equals` must be invoked.

The Method `equals`

- Method `equals` tests any two objects for equality—the strings contained in the two `String` objects are identical.

```
String s1 = "Hello World";
String s2 = s1 + "";
if(s1.equals(s2)) System.out.println("s1 = s2"); // true
```

```
String s1 = "hello";
String s2 = "HELLO";
if(s1.equals(s2)) System.out.println("s1 = s2"); // false
```

The Method `compareTo`

```
String s1 = "hello";
String s2 = "HELLO";
int result = s1.compareTo(s2); // value of result?
```

`compareTo` compares two strings (lexicographical comparison):

- Returns 0 if the `String`s are equal (identical contents).
- Returns a negative number if the `String` that invokes `compareTo (s1)` is less than the `String` that is passed as an argument `(s2)`.
- Returns a positive number if the `String` that invokes `compareTo (s1)` is greater than the `String` that is passed as an argument `(s2)`.

Methods `startsWith` & `endsWith`

The methods `startsWith` and `endsWith` determine whether a string starts or ends with the method argument, respectively

```
String s1 = "Hello World";
if(s1.startsWith("He")) System.out.print("true"); // true
```

```
String s1 = "Hello World";
if(s1.startsWith("llo", 2)) System.out.print("true"); // true
```

```
String s1 = "Hello World";
if(s1.endsWith("ld")) System.out.print("true"); // true
```

Locating Characters in Strings

```
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.indexOf('c'));  // 2
System.out.println(s.indexOf('$'));  // -1
System.out.println(s.indexOf('a', 1));  // 13
```

- `indexOf` locates the first occurrence of a character in a `String`.
  - If the method finds the character, it returns the character's index in the `String`; otherwise, it returns `-1`.
- Two-argument version of `indexOf`:
  - Take one more argument: the starting index at which the search should begin.

Locating Characters in Strings

```
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.lastIndexOf('c')); // 15
System.out.println(s.lastIndexOf('$')); // -1
System.out.println(s.lastIndexOf('a', 8)); // 0
```

- `lastIndexOf` locates the last occurrence of a character in a `String`.
  - The method searches from the end of the `String` toward the beginning.
  - If it finds the character, it returns the character's index in the `String`; otherwise, it returns –1.
- Two-argument version of `lastIndexOf`:
  - The character and the index from which to begin searching backward.

Extracting Substrings from Strings

```
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.substring(20));  // hijklm
System.out.println(s.substring(3, 6)); // def
```

- `substring` methods create a new `String` object by copying part of an existing `String` object.
- The one-integer-argument version specifies the starting index (inclusive) in the original `String` from which characters are to be copied.
- Two-integer-argument version specifies the starting index (inclusive) and ending index (exclusive) to copy characters in the original `String`.

String Method replace

```
String s1 = "Hello";
System.out.println(s1.replace('l', 'L')); // HeLLo
System.out.println(s1.replace("ll", "LL")); // HeLLo
```

- `replace` returns a new `String` object in which every occurrence of the first character argument is replaced with the second character argument.
- Another version of method `replace` enables you to replace substrings rather than individual characters (every occurrence of the first substring is replaced).

Concatenating Strings

```
String s1 = "Happy ";
String s2 = "Birthday";
System.out.println(s1.concat(s2)); // Happy Birthday
System.out.println(s1); // Happy
```

- `String` method `concat` concatenates two `String` objects and returns a new `String` object containing the characters from both original `String`s.
- The original `String`s to which `s1` and `s2` refer are not modified

Concatenating Strings

```
public static void main(String[] args) {
    String s = "";
    for (int i = 0; i < 1000; i++) {
        s = s + "." + i:
    }
}
```

- We can use "+" for `String` concatenation
- However, when + is used in a loop, a new string will be created in every iteration (because of immutability), which is inefficient
- Better to use `StringBuilder`, which is mutable

Objectives

- Immutable character-string objects of class `String`
- Mutable character-string objects of class `StringBuilder`

Class `StringBuilder`

- `String` objects are immutable. Can we create mutable character-string objects in Java?
- Yes. The class `StringBuilder` helps create and manipulate dynamic string information, i.e., modifiable, mutable strings.
- You can add, insert, or append new contents into StringBuilder

StringBuilder Constructors

- Every `StringBuilder` is capable of storing a number of characters specified by its capacity.
- If a `StringBuilder`'s capacity is exceeded, the capacity automatically expands to accommodate additional characters.

java.lang.StringBuilder	
+StringBuilder()	Constructs an empty string builder with capacity 16.
+StringBuilder(capacity: int)	Constructs a string builder with the specified capacity.
+StringBuilder(s: String)	Constructs a string builder with the specified string.

StringBuilder Constructors

Default initial capacity is 16 chars

```
StringBuilder buffer1 = new StringBuilder();

StringBuilder buffer2 = new StringBuilder(10);

StringBuilder buffer3 = new StringBuilder("hello");

System.out.printf("buffer1 = \"%s\"\n", buffer1);

System.out.printf("buffer2 = \"%s\"\n", buffer2);

System.out.printf("buffer3 = \"%s\"\n", buffer3);
```

```
buffer1 = ""

buffer2 = ""

buffer3 = "hello"
```

StringBuilder Method append

- Class `StringBuilder` provides several `append` methods to allow values of various types to be appended to the end of a `StringBuilder` object.
- Overloaded append() are provided for each of the primitive types, and for character arrays, Strings, Objects, and more.

```
append(boolean b)
append(char c)
append(char[] str)
append(char[] str, int offset, int len)
append(double d)
append(float f)
append(int i)
append(long lng)
append(CharSequence s)
append(CharSequence s, int start, int end)
append(Object obj)
append(String str)
append(StringBuffer sb)
```

```java
String string = "goodbye";
char[] charArray = {'a', 'b', 'c', 'd', 'e', 'f'};
boolean booleanValue = true;
char charValue = 'Z';
int intValue = 7;
long longValue = 10000000000L;
float floatValue = 2.5f;
double doubleValue = 33.3333;
StringBuilder buffer = new StringBuilder();
StringBuilder lastBuffer = new StringBuilder("last buffer");

buffer.append(string); buffer.append("\n");
buffer.append(charArray); buffer.append("\n");
buffer.append(charArray, 0, 3); buffer.append("\n");
buffer.append(booleanValue); buffer.append("\n");
buffer.append(charValue); buffer.append("\n");
buffer.append(intValue); buffer.append("\n");
buffer.append(longValue); buffer.append("\n");
buffer.append(floatValue); buffer.append("\n");
buffer.append(doubleValue); buffer.append("\n");
buffer.append(lastBuffer);

System.out.printf("buffer contains:\n%s", buffer.toString());
```

```
buffer contains:
goodbye
abcdef
abc
true
Z
7
10000000000
2.5
33.3333
last buffer
```

Here we still use the same `StringBuilder` object reference, because `StringBuilder` objects are mutable.

Read the Documentation!

- https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/String.html
- https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/StringBuilder.html

Read the Documentation in IDEA

```
String str = "Java";
String str2 = str;
str.concat(" course");
```

Result of 'String.concat()' is ignored

```
@NotNull
public String concat(
    @NotNull String str
)
```

Concatenates the specified string to the end of this string.
If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a `String` object is returned that represents a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

Params: str – the `String` that is concatenated to the end of this `String`.

Returns: a string that represents the concatenation of this object's characters followed by the string argument's characters.

```
String s1 = "Java";
String s2 = "Java";
```

```
s1.l
```

- m length()
- m lastIndexOf(int ch)
- m lines()
- m lastIndexOf(String str)
- m lastIndexOf(int ch, int fromIndex)
- m lastIndexOf(String str, int fromIndex)
- m toLowerCase(Locale.ROOT)
- m toLowerCase(Locale locale)
- m toLowerCase()