BIRZEIT UNIVERSITY

Strings

By: Mamoun Nawahdah (Ph.D.)
2020

Constructing String object

String newString = **new** **String**(stringLiteral);

String message = new String("Welcome to Java");

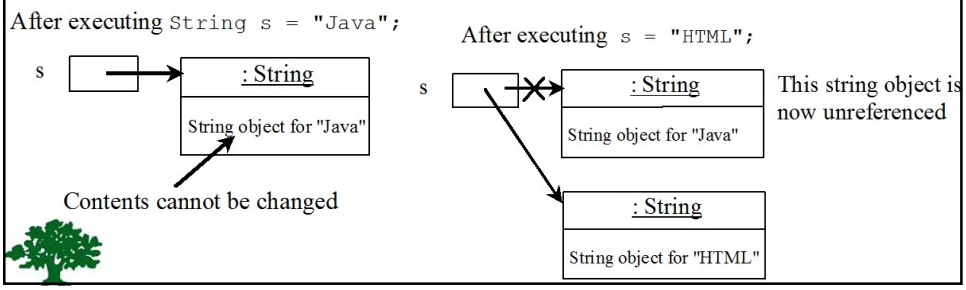
Since strings are used frequently, Java provides a short-hand **initializer** for creating a **string**:

String message = "Welcome to Java";

Strings are Immutable

- A **String** object is immutable.
  - Its contents cannot be changed.
- Does the following code change the contents of the string **s**?

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

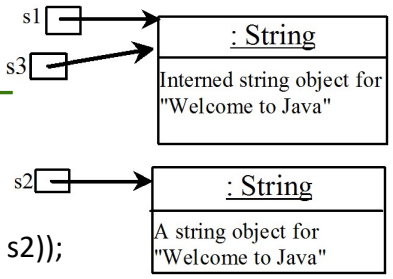


Interned Strings

- Since strings are immutable and are frequently used, **to improve efficiency and save memory**, the **JVM** uses a **unique** instance for string literals with the same character sequence.
- Such an instance is called **interned**.

4

Example




```
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";
System.out.println("s1 == s2 is " + (s1 == s2));

System.out.println("s1 == s3 is " + (s1 == s3));
```

Display:

s1 == s2 is **false**

s1 == s3 is **true**

- A new object is created if you use the **new** operator.
- If you use the string **initializer**, no new object is created **if** the interned object is already created.

5

String Comparisons

java.lang.String	
+equals(s1: Object): boolean	Returns true if this string is equal to string s1.
+equalsIgnoreCase(s1: String): boolean	Returns true if this string is equal to string s1 case-insensitive.
+compareTo(s1: String): int	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
+compareToIgnoreCase(s1: String): int	Same as compareTo except that the comparison is case-insensitive.
+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.
+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean	Same as the preceding method except that you can specify whether the match is case-sensitive.
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix.
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix.

6

String Comparisons

```
String s1 = new  String("Welcome");
String s2 = "Welcome";

if (s1.equals(s2)){
   // s1 and s2 have the same contents
}

if (s1 == s2) {
   // s1 and s2 have the same reference
}
```

7

String Comparisons

compareTo(Object object)

```
String s1 = new String("Welcome");
String s2 = "Welcome";

if (s1.compareTo(s2) > 0) {
     // s1 is greater than s2
}
else if (s1.compareTo(s2) == 0) {
        // s1 and s2 have the same contents
      }
      else {
         // s1 is less than s2
      }
```

8

String Length, Characters, and Combining Strings

java.lang.String	
+length(): int	Returns the number of characters in this string.
+charAt(index: int): char	Returns the character at the specified index from this string.
+concat(s1: String): String	Returns a new string that concatenate this string with string s1.

Finding String Length

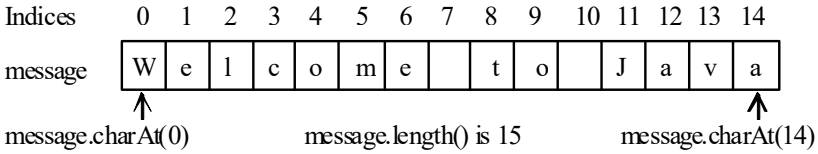
Finding string length using the **length()** method:

```
message = "Welcome to Java";
message.length();  // returns 15
```

9

Retrieving Individual Characters in a String

- ~~Do not use message[0]~~
- Use **message.charAt(index)**
- Index starts from **0**



10

String Concatenation

String s3 = s1.concat(s2);

String s3 = s1 + s2;

s1 + s2 + s3 + s4 + s5

same as

(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

11

Extracting Substrings

java.lang.String	
+substring(beginIndex: int): String	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 8.6.
+substring(beginIndex: int, endIndex: int): String	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1, as shown in Figure 8.6. Note that the character at endIndex is not part of the substring.

12

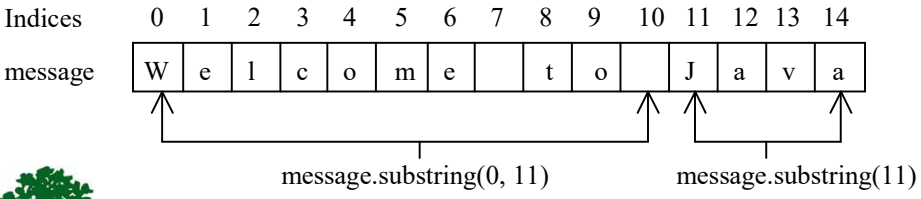
Extracting Substrings

❖ You can extract a single character from a **string** using the **charAt** method.

❖ You can also extract a substring from a **string** using the **substring** method in the **String** class.

```
String s1 = "Welcome to Java";

String s2 = s1.substring(0, 11) + "HTML";
```



13

Converting, Replacing, and Splitting Strings

java.lang.String	
+toLowerCase(): String	Returns a new string with all characters converted to lowercase.
+toUpperCase(): String	Returns a new string with all characters converted to uppercase.
+trim(): String	Returns a new string with blank characters trimmed on both sides.
+replace(oldChar: char, newChar: char): String	Returns a new string that replaces all matching character in this string with the new character.
+replaceFirst(oldString: String, newString: String): String	Returns a new string that replaces the first matching substring in this string with the new substring.
+replaceAll(oldString: String, newString: String): String	Returns a new string that replace all matching substrings in this string with the new substring.
+split(delimiter: String): String[]	Returns an array of strings consisting of the substrings split by the delimiter.

14

Examples

"Welcome".toLowerCase()

returns a new string, **welcome**

"Welcome".toUpperCase()

returns a new string, **WELCOME**

" Welcome ".trim()

returns a new string, **Welcome**

"Welcome".replace('e', 'A')

returns a new string, **WAlcomA**

"Welcome".replaceFirst("e", "AB")

returns a new string, **WABlcome**

"Welcome".replaceAll("e", "AB")

returns a new string, **WABlcomAB**

15

Splitting a String

```
String s1 =  "Java#HTML#Perl";
String[] tokens = s1.split( "#");
for (int i = 0; i < tokens.length; i++)
        System.out.println( tokens[i]  );
```

Displays:

Java
HTML
Perl

16

Matching, Replacing and Splitting by Patterns

❖ You can **match**, **replace**, or **split** a string by specifying a pattern.

❖ This is an extremely useful and powerful feature, commonly known as ***regular expression***.

"Java".matches("Java")

"Java".equals("Java")

"Java is fun".matches("Java.*")

"Java is cool".matches("Java.*")

17

Matching, Replacing and Splitting by Patterns

❖ The **replaceAll**, **replaceFirst**, and **split** methods can be used with a regular expression.

❖ For example, the following statement returns a new string that **replaces** **$**, **+**, or **#** in "**a+b$#c**" by the string **123**.

```
String s = "a+b$#c".replaceAll("[$+#]", "123");
System.out.println(s);
```

Here the regular expression **[$+#]** specifies a pattern that matches **$**, **+**, or **#**.

So, the output is **a123b123123c**

18

Matching, Replacing and Splitting by Patterns

❖ The following statement **splits** the string into an array of strings delimited by some punctuation marks:

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");
for (int i = 0; i < tokens.length; i++)
    System.out.println(tokens[i]);
```

```
Java
C
C#
C++
```

19

Finding a Character or a Substring in a String

java.lang.String	
+indexOf(ch: char): int	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
+indexOf(ch: char, fromIndex: int): int	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
+indexOf(s: String): int	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
+indexOf(s: String, fromIndex: int): int	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
+lastIndexOf(ch: int): int	Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.
+lastIndexOf(ch: int, fromIndex: int): int	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
+lastIndexOf(s: String): int	Returns the index of the last occurrence of string s. Returns -1 if not matched.
+lastIndexOf(s: String, fromIndex: int): int	Returns the index of the last occurrence of string s before fromIndex. Returns -1 if not matched.

Finding a Character or a Substring in a String

String s = "Welcome to Java";

s.indexOf('W')	returns **0**
s.indexOf('x')	returns **-1**
s.indexOf('o', 5)	returns **9**
s.indexOf("come")	returns **3**
s.indexOf("Java", 5)	returns **11**
s.indexOf("java", 5)	returns **-1**
s.lastIndexOf('a')	returns **14**

21

Convert Character and Numbers to Strings

❖ The **String** class provides several static **valueOf** methods for converting a character, an array of characters, and numeric values to strings.

❖ These methods have the same name **valueOf** with different argument types **char**, **char[]**, **double**, **long**, **int**, and **float**.

❖ For example, to convert a **double** value to a **string**, use **String.valueOf(5.44)**. The return value is string consists of characters **'5', '.', '4',** and **'4'**.

22

The Character Class

java.lang.Character	
+Character(value: char)	Constructs a character object with char value
+charValue(): char	Returns the char value from this object
+compareTo(anotherCharacter: Character): int	Compares this character with another
+equals(anotherCharacter: Character): boolean	Returns true if this character equals to another
+isDigit(ch: char): boolean	Returns true if the specified character is a digit
+isLetter(ch: char): boolean	Returns true if the specified character is a letter
+isLetterOrDigit(ch: char): boolean	Returns true if the character is a letter or a digit
+isLowerCase(ch: char): boolean	Returns true if the character is a lowercase letter
+isUpperCase(ch: char): boolean	Returns true if the character is an uppercase letter
+toLowerCase(ch: char): char	Returns the lowercase of the specified character
+toUpperCase(ch: char): char	Returns the uppercase of the specified character

23

Examples

Character c = new Character('b');

c.compareTo(new Character('a'))	returns **1**
c.compareTo(new Character('b'))	returns **0**
c.compareTo(new Character('c'))	returns **-1**
c.compareTo(new Character('d')	returns **-2**
c.equals(new Character('b'))	returns **true**
c.equals(new Character('d'))	returns **false**

24

StringBuilder and StringBuffer

- The **StringBuilder/StringBuffer** class is an alternative to the **String** class.
- In general, a **StringBuilder/StringBuffer** can be used wherever a **String** is used.
- **StringBuilder/StringBuffer** is more **flexible** than **String**.
- You can **add**, **insert**, or **append** new contents into a string buffer, whereas the value of a **String** object is fixed once the string is created.

25

StringBuilder Constructors

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.

26

Modifying Strings in the Builder

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	Appends a char array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in data into this string builder.
+append(v: *aPrimitiveType*): StringBuilder	Appends a primitive type value as a string to this builder.
+append(s: String): StringBuilder	Appends a string to this string builder.
+delete(startIndex: int, endIndex: int): StringBuilder	Deletes characters from startIndex to endIndex.
+deleteCharAt(index: int): StringBuilder	Deletes a character at the specified index.
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	Inserts a subarray of the data in the array to the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): StringBuilder	Inserts a value converted to a string into this builder.
+insert(offset: int, s: String): StringBuilder	Inserts a string into this builder at the position offset.
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	Replaces the characters in this builder from startIndex to endIndex with the specified string.
+reverse(): StringBuilder	Reverses the characters in the builder.
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this builder.

27

Examples

```
StringBuilder sb = new StringBuilder("Welcome to ");
```

sb.append("Java");	**Welcome to Java**
sb.insert(11, "HTML and ");	**Welcome to HTML and Java**
sb.delete(8, 11) ;	**Welcome HTML and Java**
sb.deleteCharAt(8) ;	**Welcome TML and Java**
sb.reverse() ;	**avaJ dna LMT emocleW**
sb.replace(11, 15, "HTML") ;	**avaJ dna LMHTMLocleW**
sb.setCharAt(0, 'w') ;	**wvaJ dna LMHTMLocleW**

28

The toString, capacity, length, setLength, and charAt Methods

java.lang.StringBuilder	
+toString(): String	Returns a string object from the string builder.
+capacity(): int	Returns the capacity of this string builder.
+charAt(index: int): char	Returns the character at the specified index.
+length(): int	Returns the number of characters in this builder.
+setLength(newLength: int): void	Sets a new length in this builder.
+substring(startIndex: int): String	Returns a substring starting at startIndex.
+substring(startIndex: int, endIndex: int): String	Returns a substring from startIndex to endIndex-1.
+trimToSize(): void	Reduces the storage size used for the string builder.

29

What are the results of the following expressions?

Suppose that s1, s2, s3, and s4 are four strings, given as follows:

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
String s4 = "Welcome to Java";
```

```
a. s1 == s2
b. s1 == s3
c. s1 == s4
d. s1.equals(s3)
e. s1.equals(s4)
f. "Welcome to Java".replace("Java", "HTML")
g. s1.replace('o', 'T')
h. s1.replaceAll("o", "T")
i. s1.replaceFirst("o", "T")
j. s1.toCharArray()
```

a. true
b. false
c. true
d. true
e. true
f. Welcome to HTML
g. WelcTme tT Java
h. WelcTme tT Java
i. WelcTme to Java
j. []