Advanced Programming

Strings

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

The String class

- A **string** is a sequence of characters.
- In many languages, strings are treated as an array of characters, but in Java a string is an **object**.
- The **String** class has 11 constructors and more than 40 methods for manipulating strings.
- Not only is it very useful in programming, but also it is a good example for learning classes and objects.

Constructing a String

- You can create a *string object* from a **string literal** or from an **array of characters**.
- To create a string from a string literal:

```
String s = new String(stringLiteral);
```

  - **StringLiteral** is a sequence of characters enclosed inside double quotes.

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

  - Java treats a *string literal* as a String object:

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

- You can also create a string from an array of characters:

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y'};
String message = new String(charArray);
```

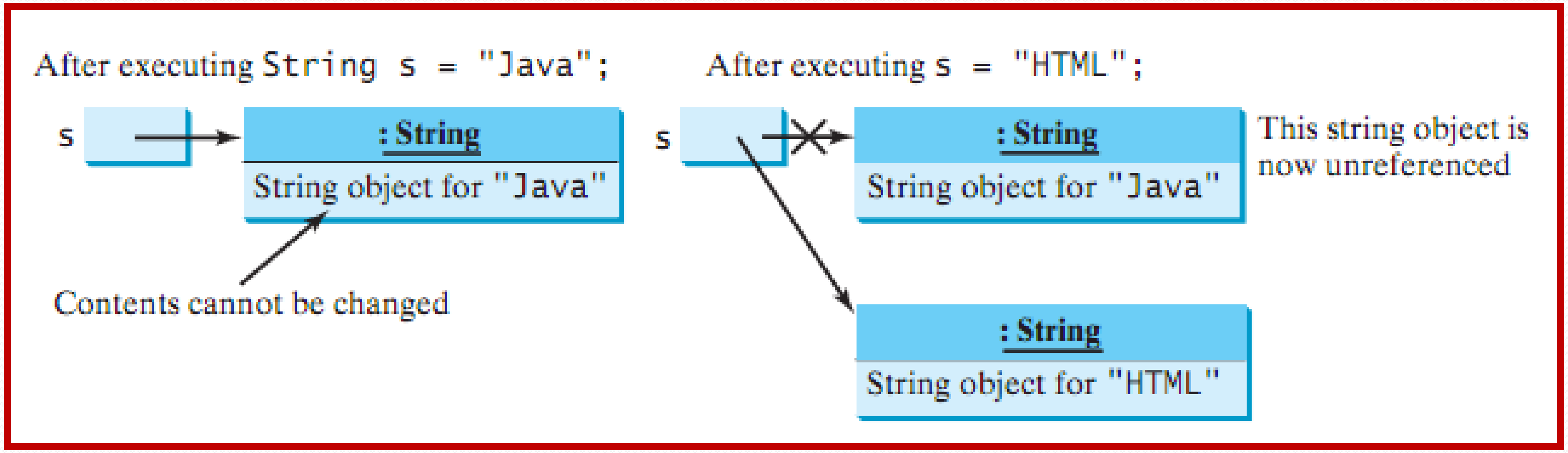
Immutable String and Interned String

- A **String** object is *immutable*; its contents cannot be changed.

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

  - The first: creates a **String** object with the content "**Java**" and assigns its reference to **s**.
  - The second: creates a new **String** object with the content "**HTML**" and assigns its reference to **s**.
  - The first **String** object still exists after the assignment, but it can no longer be accessed, because variable **s** now points to the new object

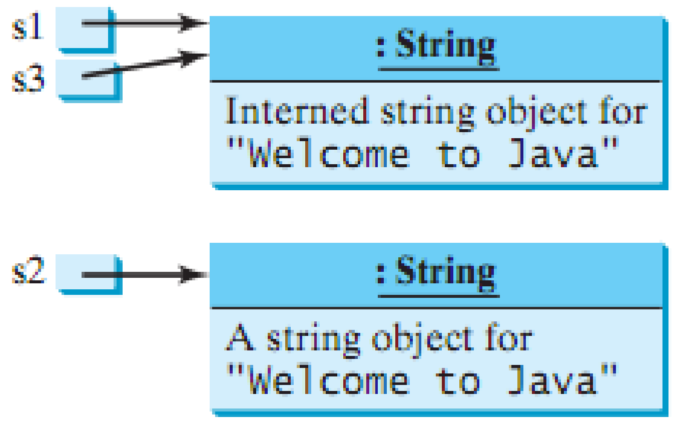
Immutable String and Interned String


After executing String s = "Java";
s
: String
String object for "Java"
Contents cannot be changed
After executing s = "HTML";
s
: String
String object for "Java"
This string object is now unreferenced
: String
String object for "HTML"

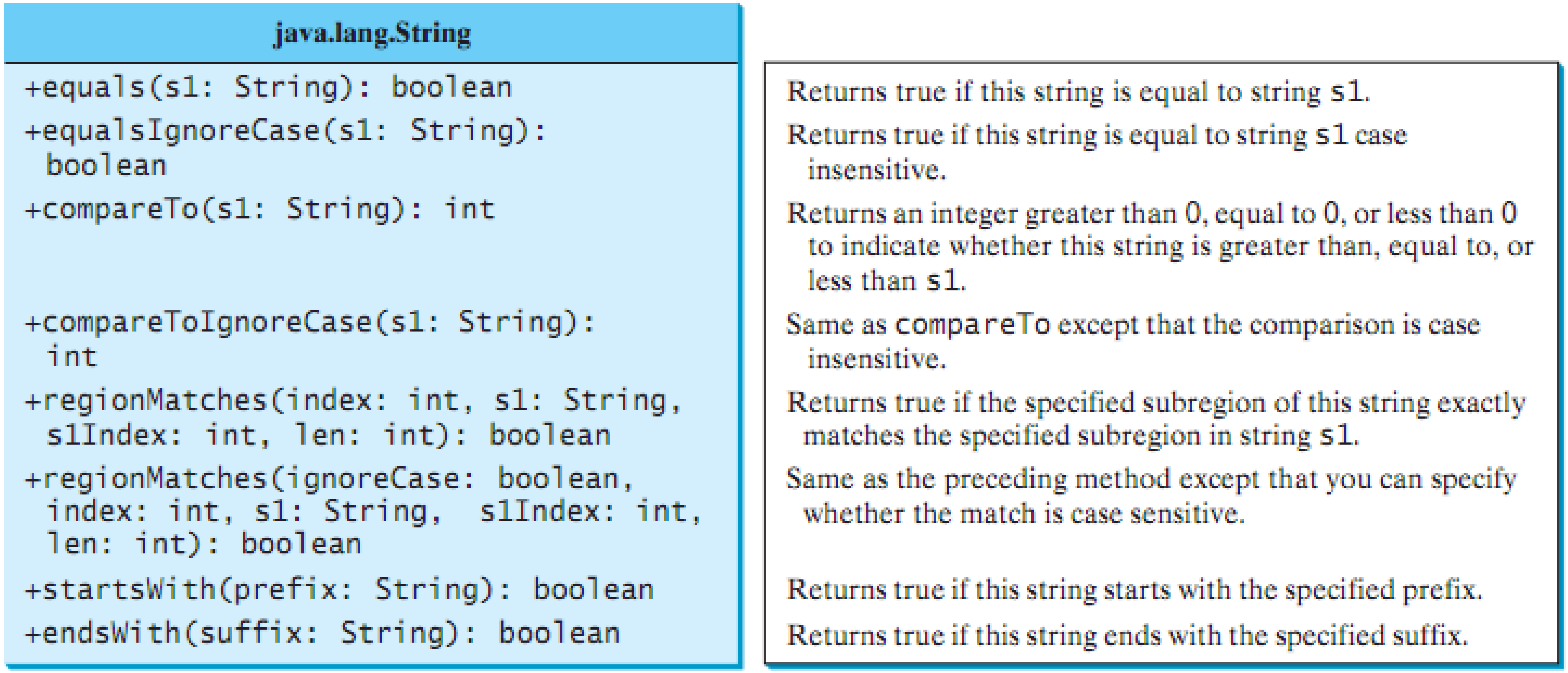
Immutable String and Interned String

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

```
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));
```


s1
s3
: String
Interned string object for "Welcome to Java"
s2
: String
A string object for "Welcome to Java"

String Comparisons



java.lang.String	
`+equals(s1: String): 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(index: int, s1: String, s1Index: int, len: int): boolean`	Returns true if the specified subregion of this string exactly matches the specified subregion in string `s1`.
`+regionMatches(ignoreCase: boolean, index: int, s1: String, s1Index: 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.

String Comparisons

- How do you compare the contents of two strings?

```
if (string1 == string2)
   System.out.println("string1 and string2 are the same object");
else
   System.out.println("string1 and string2 are different objects");
```

- The == operator checks only **whether string1 and string2 refer to the same object**; it does not tell you whether they have the same contents.

Equals

- You cannot use the == operator to find out whether two string variables have the same contents ➔ use the **equals** method

```
if (string1.equal(string2))
   System.out.println("string1 and string2 are the same object");
else
   System.out.println("string1 and string2 are different objects");
```

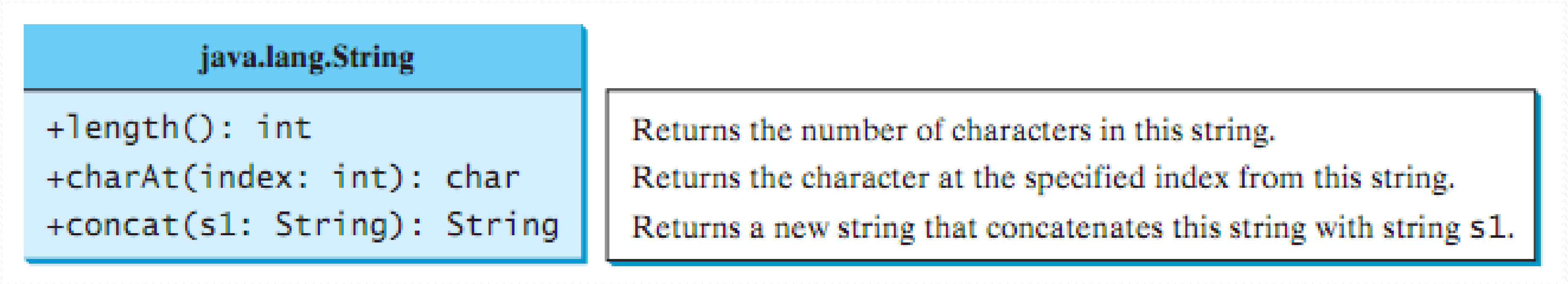
compareTo

- The **compareTo** method can also be used to compare two strings.

  s1.compareTo(s2)

  - 0 if s1 is equal to s2
  - a value less than 0 if s1 is less than s2
  - a value greater than 0 if s1 is greater than s2.
- The **equalsIgnoreCase**, **compareToIgnoreCase** methods for comparing strings, *ignore the case* of the letters when comparing two strings.

String Length, Characters, and Combining Strings



java.lang.String

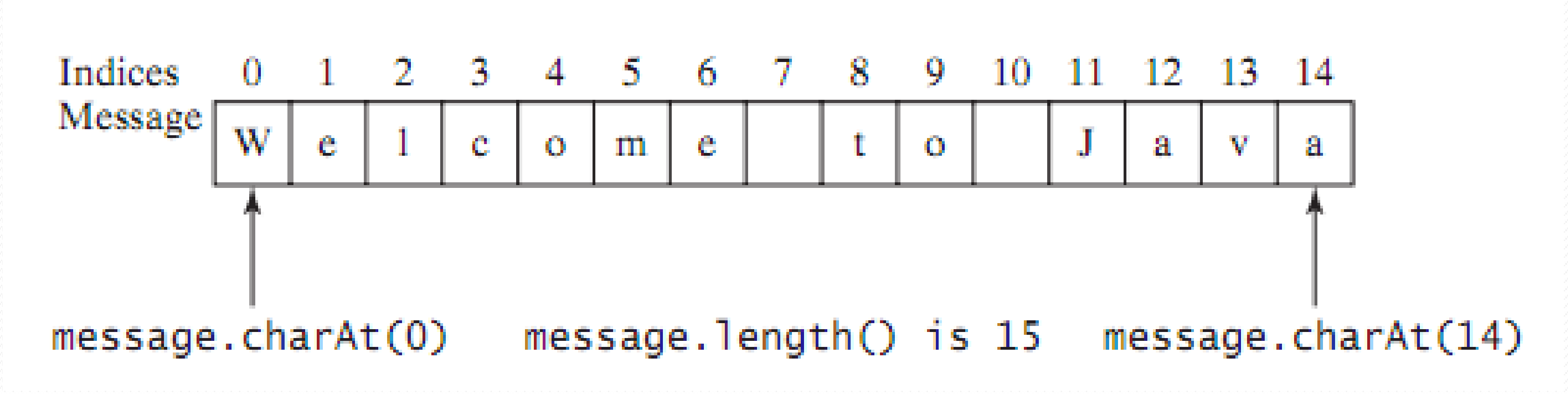
```
+length(): int
+charAt(index: int): char
+concat(s1: String): String
```

Returns the number of characters in this string.

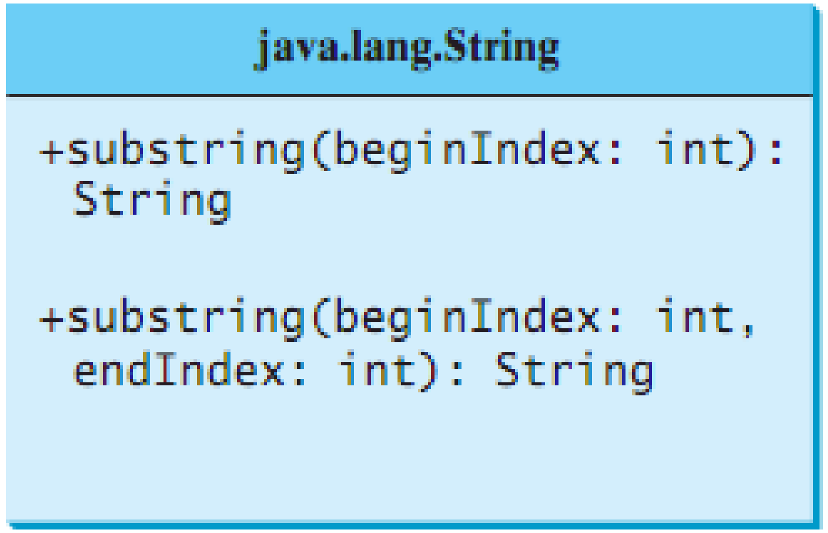
Returns the character at the specified index from this string.

Returns a new string that concatenates this string with string `s1`.

String Length, Characters, and Combining Strings

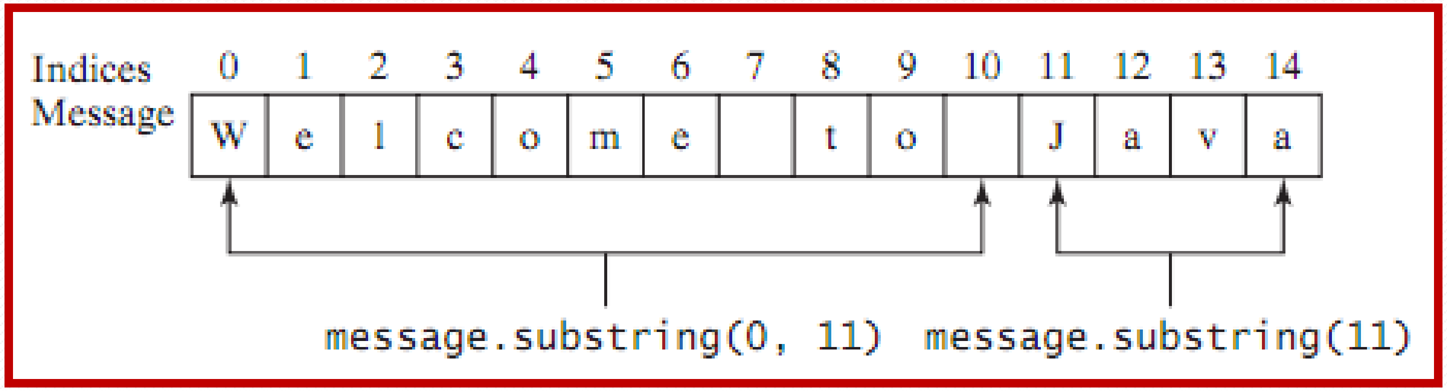


Obtaining Substrings



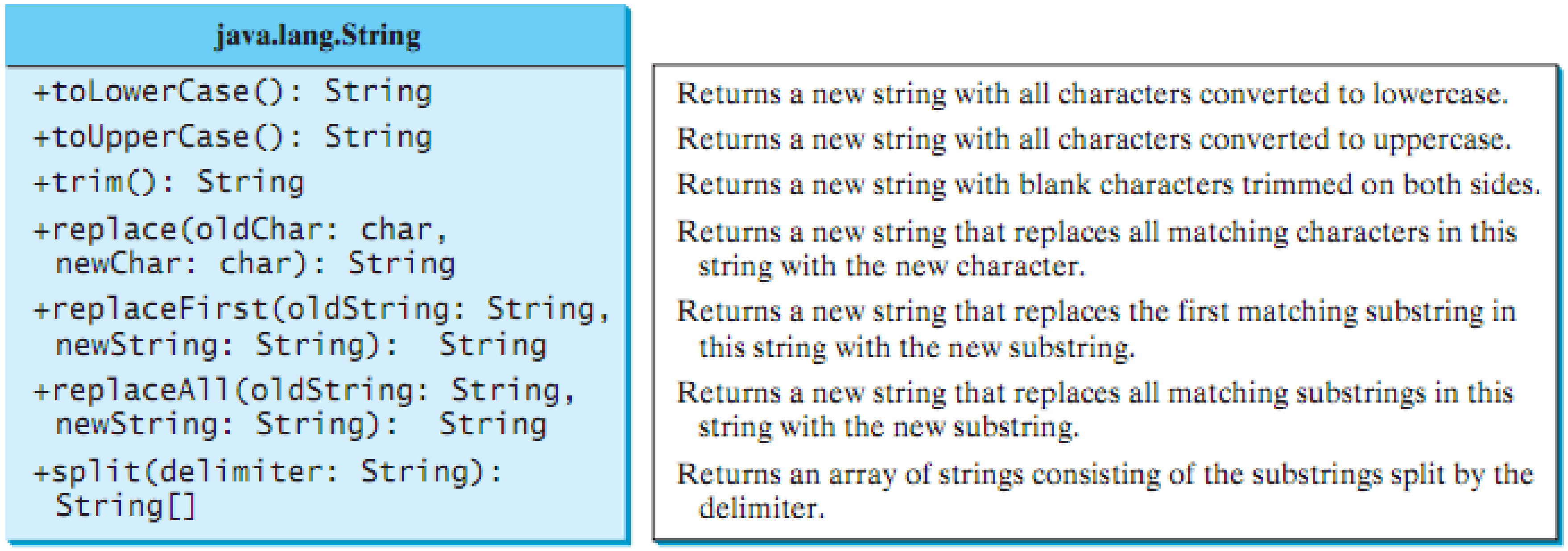
java.lang.String

```
+substring(beginIndex: int):
  String

+substring(beginIndex: int,
  endIndex: int): String
```



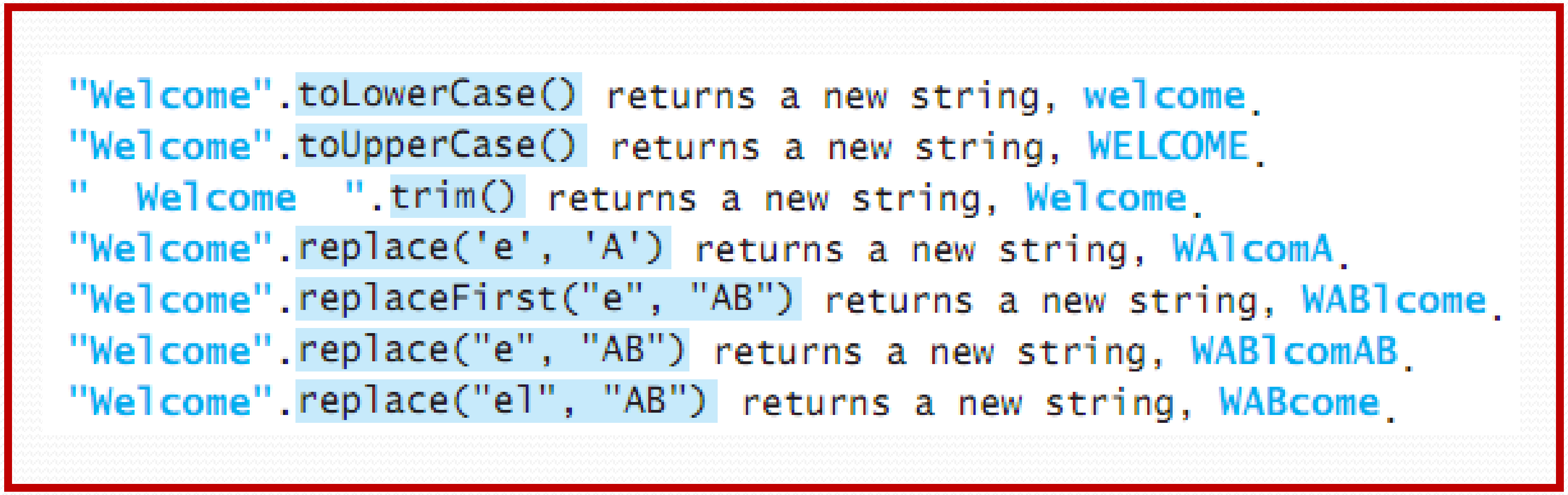
```
message.substring(0, 11)  message.substring(11)
```

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 characters 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 replaces 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.

Converting, Replacing, and Splitting Strings



```
"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".replace("e", "AB") returns a new string, WABlcomAB.
"Welcome".replace("el", "AB") returns a new string, WABcome.
```

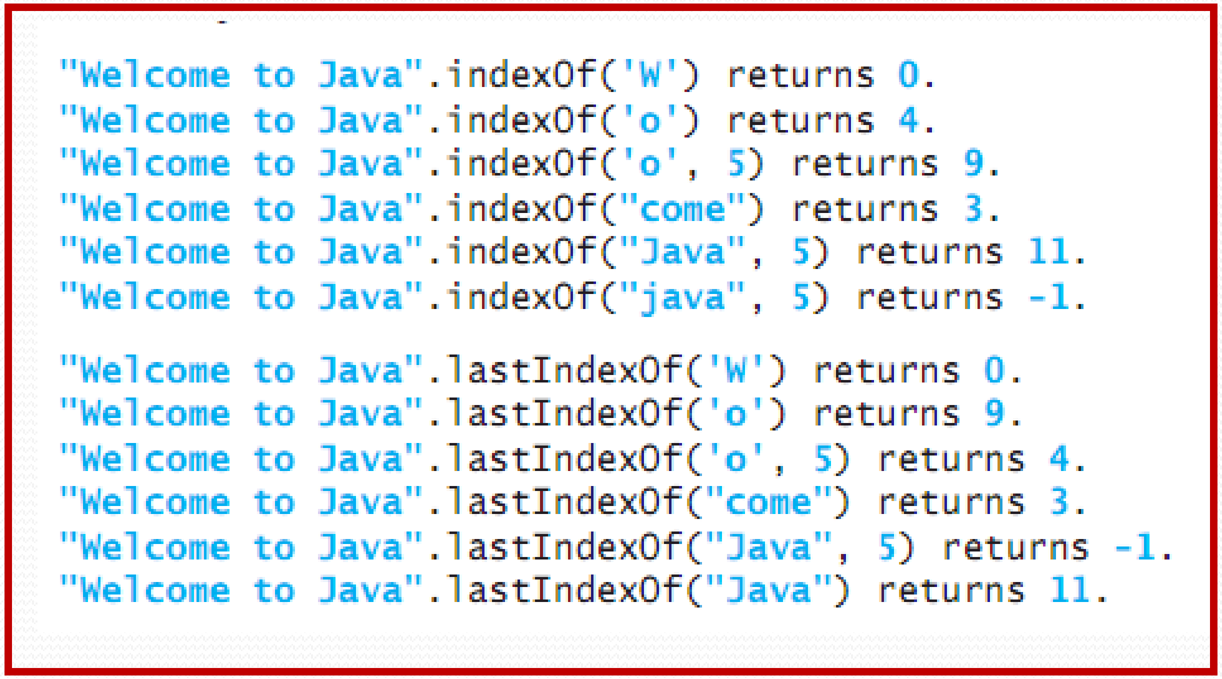
Converting, Replacing, and Splitting Strings

- The **split** method can be used to extract tokens from a string with the specified delimiters.

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

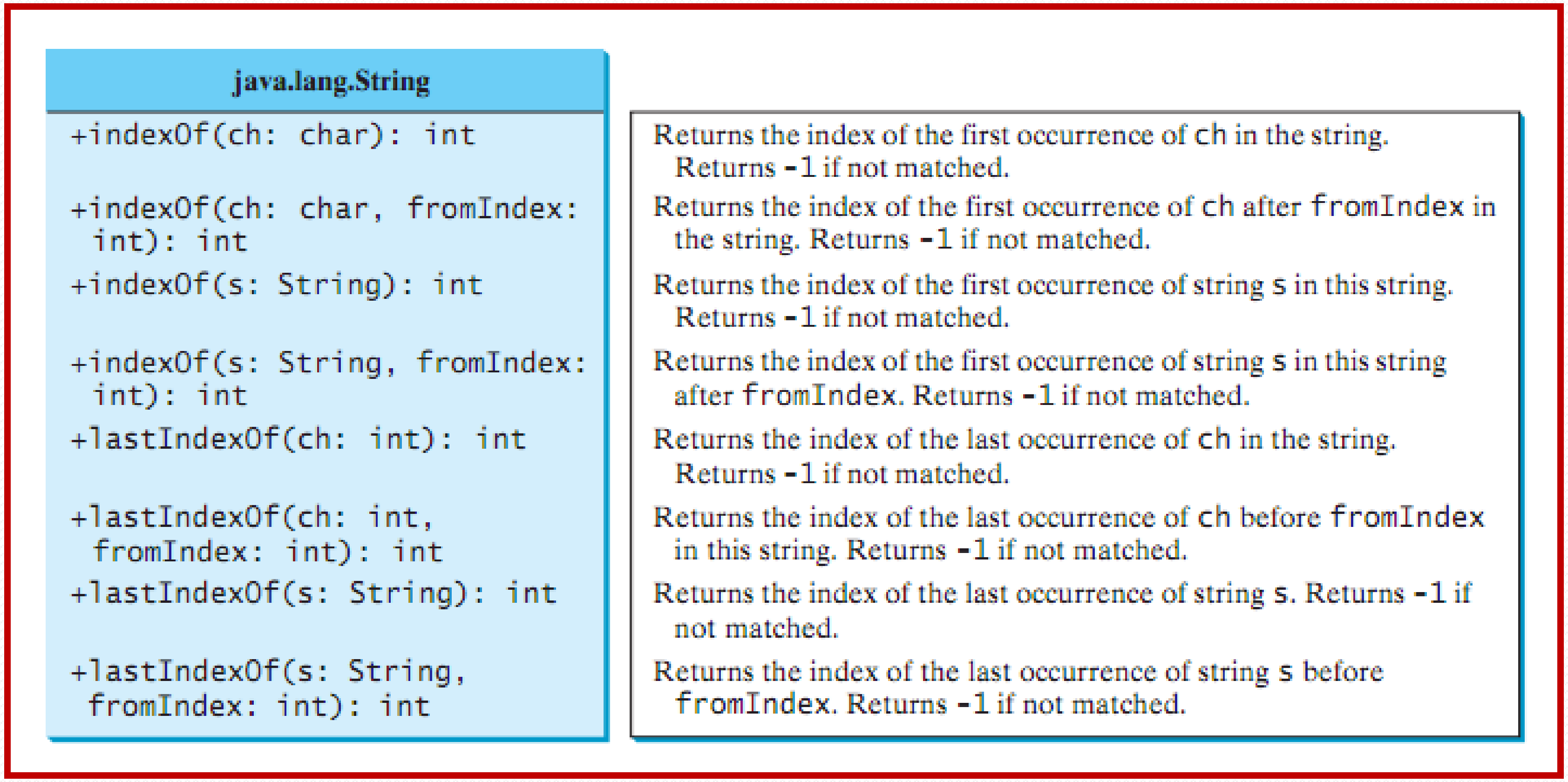
Finding a Character or a Substring in a String

- The **String** class provides several overloaded **indexOf** and **lastIndexOf** methods to find a character or a substring in a string



```
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
```

Finding a Character or a Substring in a String



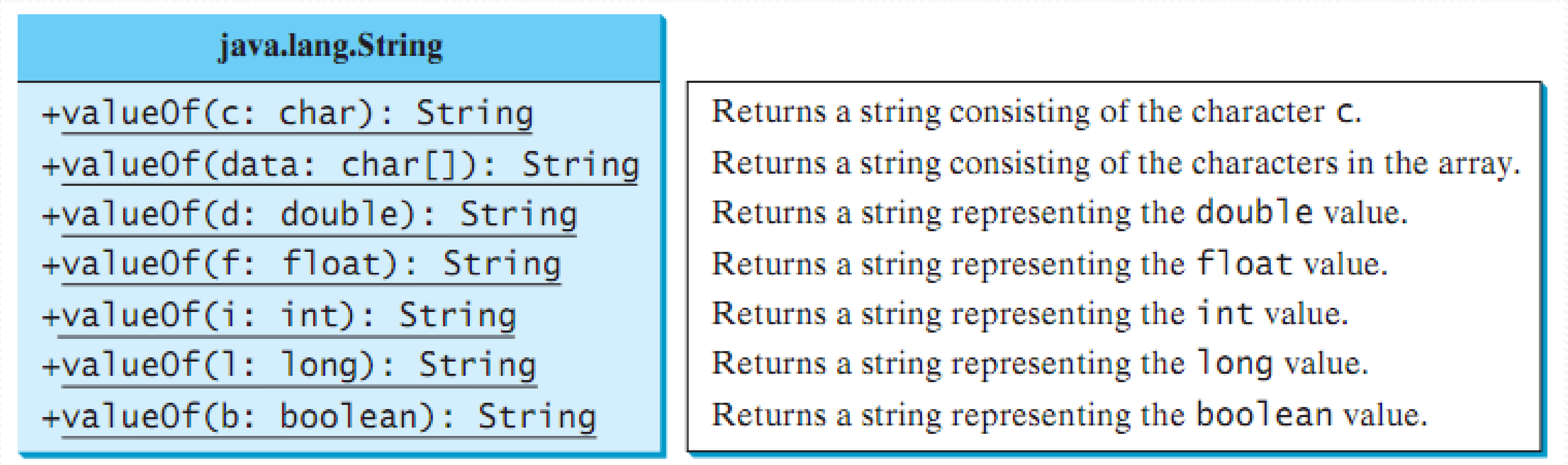
java.lang.String	
`+indexOf(ch: char): int`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`+indexOf(ch: char, fromIndex: int): int`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`+indexOf(s: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

Conversion between Strings and Arrays

- Strings are not arrays, but a string can be converted into an array, and vice versa.
- To convert a *string to an array* of characters, use the toCharArray method.
  - **char**[] chars = "**Java**".toCharArray();
- To convert *an array of characters into a string*, use the **String**(**char**[]) constructor or the **valueOf**(**char**[]) method.
  - **String** str = new **String**(new char[]{'J', 'a', 'v', 'a'});
  - **String** str = **String**.**valueOf**(new char[]{'J', 'a', 'v', 'a'});

Converting Characters and Numeric Values to Strings

- The valueOf method can be used to convert a character and numeric values to strings with different parameter types, char, double, long, int, and float



java.lang.String	
+valueOf(c: char): String	Returns a string consisting of the character c.
+valueOf(data: char[]): String	Returns a string consisting of the characters in the array.
+valueOf(d: double): String	Returns a string representing the double value.
+valueOf(f: float): String	Returns a string representing the float value.
+valueOf(i: int): String	Returns a string representing the int value.
+valueOf(l: long): String	Returns a string representing the long value.
+valueOf(b: boolean): String	Returns a string representing the boolean value.

Formatting Strings

- The **String** class contains the static format method in the **String** class to create a formatted string.

  String.format(format, item1, item2, ..., itemk)

- Example:

  String s = **String.format**("%5.2f", 45.556);

- This method is similar to the **printf** method except that the format method **returns** a formatted string, whereas the printf method **displays** a formatted string.

Finger Exercise

- A string is a palindrome if it reads the same forward and backward. The words “*mom*,” “*dad*,” and “*noon*,” for instance, are all ***palindromes***.
- Write a program that prompts the user to enter a string and reports whether the string is a ***palindrome***.

Palindrome

```java
/** Check if a string is a palindrome */
public static boolean isPalindrome(String s) {
      // The index of the first character in the string
      int low = 0;
      // The index of the last character in the string
      int high = s.length() - 1;
      while (low < high) {
            if (s.charAt(low) != s.charAt(high))
                  return false; // Not a palindrome
            low++;
            high--;
      }
      return true; // The string is a palindrome
}
```

Problem: Converting Hexadecimals to Decimals

- Given a hexadecimal number $h_n h_{n-1} \ldots h_1 h_0$, the equivalent decimal value is:

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + \ldots + h_1 \times 16^1 + h_0 \times 16^0$$

- Example: the hex number **AB8C** is:

$$10 \times 16^3 + 11 \times 16^2 + 8 \times 16^1 + 12 \times 16^0 = 43916$$

```java
public static int hexToDecimal(String hex) {
      int decimalValue = 0;
      for (int i = 0; i < hex.length(); i++) {
            char hexChar = hex.charAt(i);
            decimalValue = decimalValue * 16 +
                                    hexCharToDecimal(hexChar);
      }
      return decimalValue;
}

public static int hexCharToDecimal(char ch) {
      if (ch >= 'A' && ch <= 'F')
            return 10 + ch - 'A';
      else
            // ch is '0', '1', ..., or '9'
            return ch - '0';
}
```

HexToDecimalConversion

```
public static void main(String[] args) {
      // Create a Scanner
      Scanner input = new Scanner(System.in);
      // Prompt the user to enter a string
      System.out.print("Enter a hex number: ");
      String hex = input.nextLine();

      System.out.println("The decimal value for hex
            number " + hex + " is " +
            hexToDecimal(hex.toUpperCase()));
}
```

The StringBuilder/StringBuffer Class

- The **StringBuilder/StringBuffer** class is an alternative to the String class.
- **StringBuilder/StringBuffer** is more flexible than **String**. You can add, insert, or append new contents into a **StringBuilder** or a **StringBuffer**
  - Note: the value of a **String** object is fixed, once the string is created.
- The **StringBuilder** class is similar to **StringBuffer** except that the methods for modifying buffer in **StringBuffer** are synchronized.
- Use **StringBuffer** if it may be accessed by multiple tasks concurrently. Using **StringBuilder** is more efficient if it is accessed by a single task.

Modifying Strings in the StringBuilder

- You can append new contents at the end of a string builder, insert new contents at a specified position in a string builder, and delete or replace characters in a string builder.
- The StringBuilder class provides several overloaded methods to append **boolean**, **char**, **char array**, **double**, **float**, **int**, **long**, and **String** into a string builder.

Example

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");
//➔ Welcome to Java
stringBuilder.insert(11, "HTML and ");
//➔ Welcome to HTML and Java
```

Modifying Strings in the StringBuilder

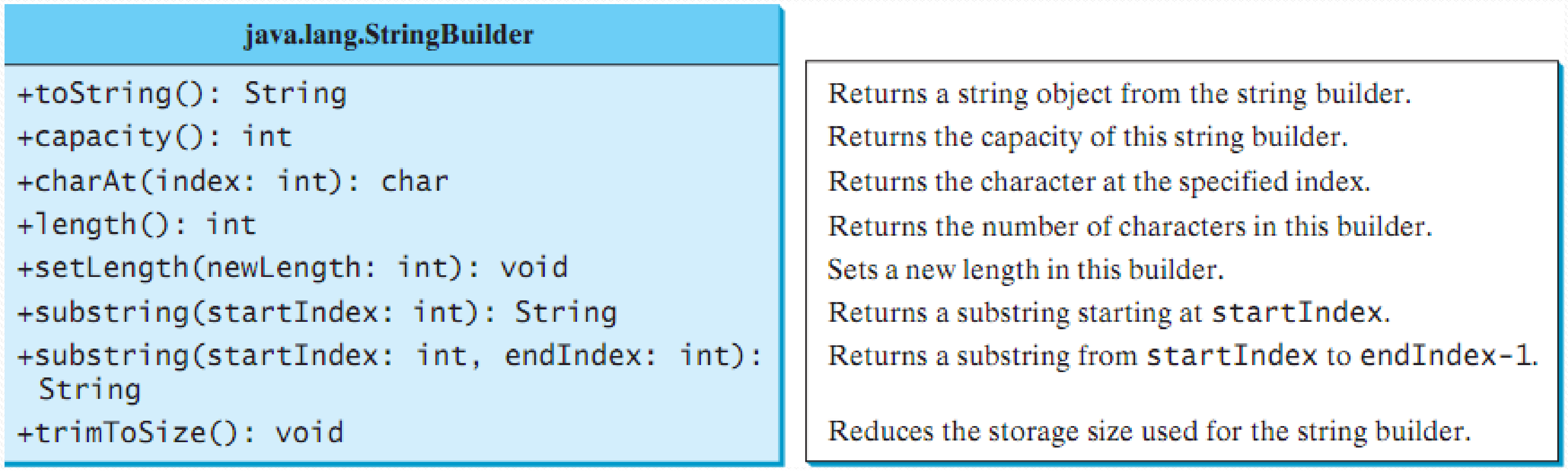
- You can also:
  - *delete* characters from a string in the builder using the two **delete** methods
  - *reverse* the string using the **reverse** method
  - *replace* characters using the **replace** method,
  - or *set a new character* in a string using the **setCharAt** method.

Example

```java
//stringBuilder = "Welcome to Java";
//1. Changes the builder to Welcome Java.
stringBuilder.delete(8, 11);
//2. Changes the builder to Welcome o Java.
stringBuilder.deleteCharAt(8);
//3. Changes the builder to avaJ ot emocleW.
stringBuilder.reverse();
//4. Changes the builder to Welcome to HTML
stringBuilder.replace(11, 15, "HTML");
//5. Sets the builder to welcome to Java.
stringBuilder.setCharAt(0, 'w');
```

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-1`.
`+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-1` 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.

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.

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

- The **capacity()** method returns the current capacity of the string builder.
  - The capacity is the *number of characters* it is able to store without having to increase its size.
- The **length()** method returns the number of characters actually stored in the string builder.
- The **setLength(newLength**) method sets the length of the string builder.
- The **charAt(index**) method returns the character at a specific index in the string builder.

Bài tập

- **Bài 1**: Nhập vào chuỗi s.
  - Đếm số từ trong chuỗi.
  - In các từ trong chuỗi s mỗi từ một dòng.
  - Đảo ngược thứ tự các từ trong chuỗi.
- **Bài 2**: Nhập vào chuỗi s.
  - Đổi chuỗi s ra chữ in.
  - Đổi ký tự đầu của mỗi từ trong chuỗi s thành chữ in, các ký tự còn lại thành chữ thường.
  - Xóa các khoảng trắng thừa trong chuỗi s: các khoảng trắng trước và sau, xóa bớt các khoảng trắng ở giữa các từ chỉ để lại một.

Bài tập

- **Bài 3**: Cho 2 chuỗi s và s1. Tìm vị trí đầu tiên chuỗi s1 xuất hiện trong chuỗi s.
- **Bài 4**: Cho 3 chuỗi s, s1 và s2. Tìm và thay thế tất cả các chuỗi s1 trong chuỗi s bằng chuỗi s2.
- **Bài** 5: Cho chuỗi s. Mã hóa chuỗi s bằng cách dịch chuyển các ký tự trong chuỗi s tiến tới 3 vị trí trong bảng chữ cái. Chỉ mã hóa với các kí tự trong khoảng: a-z; A-Z và 0-9. Các kí tự khác giữ nguyên.

Bài tập

- **Bài 6**: Cho 2 số nguyên k và l và chuỗi thông điệp s. Mã hóa thông điệp theo qui luật mã hóa:
  - Từ thứ nhất trong thông điệp => Các kí tự trong từ này +k
  - Từ thứ hai trong thông điệp => Các kí tự trong từ này –l
  - Từ thứ ba trong thông điệp => Các kí tự trong từ này +k
  - Từ thứ tư trong thông điệp => Các kí tự trong từ này –l
  - Chú ý: Chỉ mã hóa với các kí tự trong khoảng: a-z; A-Z và 0-9. Các kí tự khác giữ nguyên.
  - Các từ cách nhau bằng một khoảng trắng.

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.