**String** là một class có sẵn trong thư viện Java.



Kiểu String không phải là kiểu dữ liệu nguyên thủy (Primitive Data Type), mà nó là **Kiểu tham chiếu (Reference Data Type)**.

# Basic String Method

## Độ dài chuỗi length()

String str = "Welcome to Java";

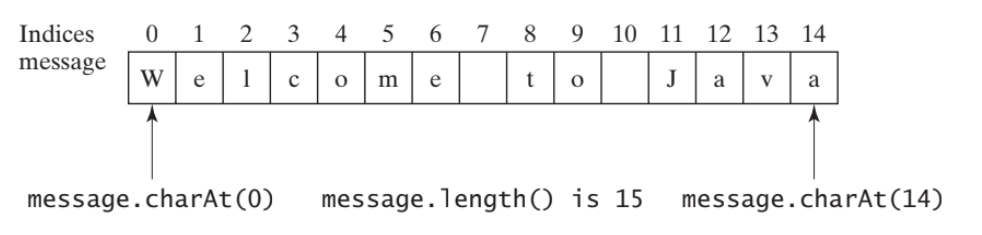
**int** len = str.length(); // len = 15

**int** len1 = "Welcome to Java".length(); // len1 = 15

## **Lấy kí tự trong một chuỗi charAt(index)**

String str;

**char** c = str.charAt(index);



## **Nối chuỗi**

String str = s1.concat(s2);

str = s1 + s2;

* *Note: Toán tử + cũng có thể nối một chuỗi với một số. Cần ít nhất một toán hạng là chuổi để có thể thực hiện nối chuỗi.*

String message = "Welcome " + "to " + "Java"; // kết quả: "Welcome to Java"

String s = "Today is " + 31 + "/" + 5 ; // "Today is 31/5"

String s1 = "Letter " + 'S'; // Letter S

**int** i=1, j=2;

System.***out***.println("i + j is " + i + j); // Kết quả: "i + j is 12"

System.***out***.println("i + j is " + (i + j)); // Kết quả: "i + j is 3"

## **Chuyển đổi chuỗi**

|  |  |
| --- | --- |
| toLowerCase() | Viết hoa |
| toUpperCase() | Viết thường |

String str = "Welcome to Java";

System.***out***.println(str.toLowerCase()); // "welcome to java"

System.***out***.println(str.toUpperCase()); // "WELCOME TO JAVA"

str.toLowerCase();

System.***out***.println(str); // "Welcome to Java"

|  |  |
| --- | --- |
| trim() | Loại bỏ khoảng trắng (whitespace characters) |

System.***out***.println("\t Good Night \n"); //" Good Night /enter"

System.***out***.println("\t Good Night \n".trim());// "Good Night"

## **So sánh chuỗi**

String s1 = "s1", s2 = "s2";

s1.equals(s2); // Trả về true nếu s1 = s2

s1.equalsIgnoreCase(s2);

// Trả về true nếu s1 = s2 không kể chữ hoa hoặc chữ thường

s1.compareTo(s2);

// Trả về số int, (> 0 là s1 > s2), (< 0 là s1 < s2), (= 0 là s1 = s2)

s1.compareToIgnoreCase(s1);

// Tương tự compareTo nhưng không kể chữ hoa hoặc thường

String prefix = "pre", suffix = "suf";

s1.startsWith(prefix); // Trả về true nếu s1 bắt đầu với chuỗi prefix

s1.endsWith(suffix); // Trả về true nếu s1 kết thúc bằng chuỗi prefix

s1.contains(s2); // Trả về true nếu s2 là chuỗi con của s1

* *Note 1: So sánh phương thức .equals() và toán tử ==*

*Toán tử == để so sánh tham chiếu (so sánh địa chỉ) còn Phương thức .equals () để so sánh nội dung*

String str1 = **new** String("string");

String str2 = **new** String("string");

System.***out***.println(str1==str2); // false

System.***out***.println(str1.equals(str2)); //true

* *Note 2:*

String str1 = "string";

String str2 = "string";

System.***out***.println(str1==str2); // true

*Bộ nhớ sẽ kiểm tra xem đã có địa chỉ nào có giá trị là string hay chưa*

* *Nếu chưa, bộ nhớ sẽ lưu giá trị “string ” và một địa chỉ mới trong bộ nhớ và trỏ str1 tới địa chỉ đó.*
* *str2 tiếp tục khởi tạo với giá trị “string” đã có nên bộ nhớ tiếp tục trỏ s2 tới địa chỉ trên.*

String str1 = **new** String("string");

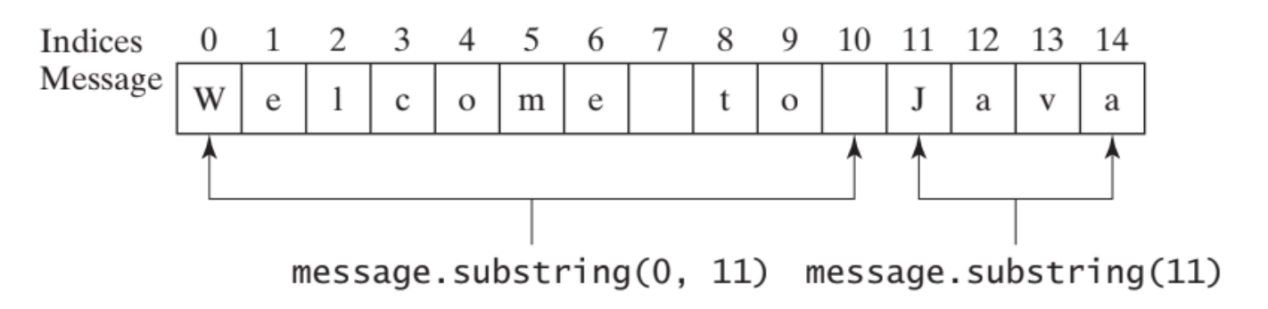
String str2 = **new** String("string");

*Khi thực hiện new String, bộ nhớ sẽ lưu str1 và str2 vào 2 địa chỉ khác nhau trên bộ nhớ.*

## **Chuỗi con**

|  |  |
| --- | --- |
| substring(beginIndex) | Trả về chuỗi con bắt đầu từ beginIndex tới hết chuỗi cha. |
| substring(beginIndex,endIndex) | Trả về chuỗi con bắt đầu từ beginIndex tới endIndex. |

:



String str = "Welcome to Java";

String subStr = str.substring(11); // "Java"

String subStr2 = str.substring(0, 11) + "class";// "Welcome to class"

## **Tìm kiếm một kí tự hoặc chuỗi con**

|  |  |
| --- | --- |
| index(ch), index(substr) | Trả về chỉ số của lần xuất hiện đầu tiên của kí tự hoặc chuỗi con cần tìm. Trả về-1 nếu không tìm thấy. |
| indexOf(ch, fromIndex),  index(substr, fromIndex) | Trả về chỉ số của lần xuất hiện đầu tiên của kí tự hoặc chuỗi con cần tìm sau chỉ số fromIndex của string ban đầu. Trả về-1 nếu không tìm thấy. |
| lastIndexOf(ch),  lastIndexOf(ch, fromIndex),  lastIndexOf(substr), lastIndexOf(substr, fromIndex) | Trả về chỉ số của lần xuất hiện cuối cùng. Trả về-1 nếu không tìm thấy. |

## **Chuyển đổi giữa String và Số**

* Chuyển từ Number sang String:

**int** num = 8;

String numStr = Integer.*toString*(num);

Hoặc

String numStr = num + "";

* Chuyển từ String sang Number:

String str = "8";

**int** intValue = Integer.*parseInt*(str);

**float** floatValue = Float.*parseFloat*(str);

# **Immutability**

* Mỗi khi tạo một chuỗi ký tự, JVM sẽ kiểm tra "**String constant pool**" (một vùng bộ nhớ đặc biệt để lưu trữ các chuỗi) trước tiên. Nếu chuỗi đã tồn tại trong nhóm (pool), một tham chiếu đến cá thể được gộp sẽ được trả về. Nếu chuỗi không tồn tại trong nhóm, một instance chuỗi mới sẽ được tạo và đặt trong pool. (Một **instance** là một trường hợp cụ thể của một class.)

String s1="Hi";

String s2="Hi";//It doesn't create a new instance



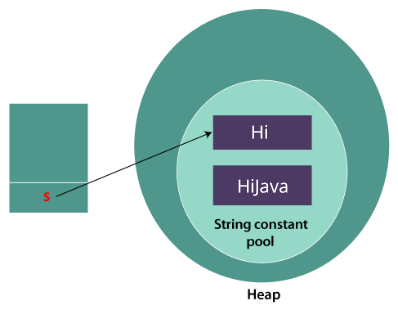
* String trong Java là bất biến (**immutability**) có nghĩa là nó không thể thay đổi được. Bất cứ khi nào có sự thay đổi của chuỗi String bất kỳ, một phiên bản mới sẽ được tạo ra.

String str = "Hi";

s.concat("Java");

System.***out***.println(s); // "Hi"

String “Hi” không bị thay đổi, và một object mới được tạo ra sau lệnh concat là HiJava. Tuy nhiên str vẫn tham chiếu tới “Hi” chứ không tham chiếu tới “HiJava”.



Nhưng nếu gán trực tiếp nó tới giá trị mà nó tham chiếu (reference variable - biến tham chiếu), khi đó str sẽ tham chiếu tới đối tượng “HiJava”. Dù vậy, đối tượng “Hi” vẫn không bị sửa đổi.

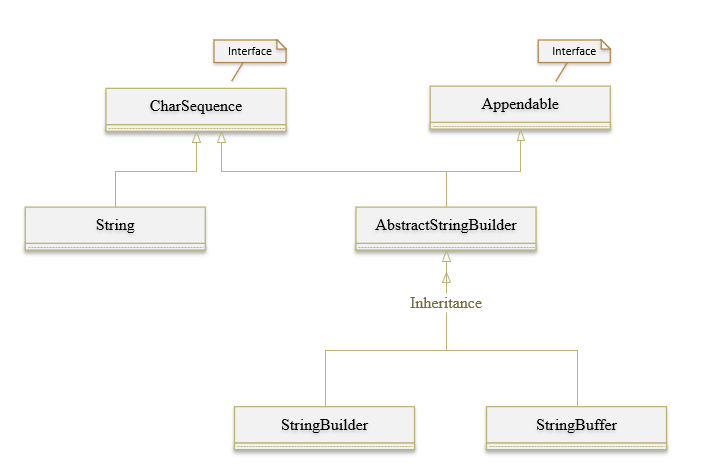
s = s.concat("Java");

System.***out***.println(s); // "HiJava"

* Lý do String object trong Java là Immutability:
* Giả sử, có 5 biến tham chiếu, đều tham chiếu tới một đối tượng là “Hi”. Nếu một trong 5 biến đó thay đổi giá trị của đối tượng, nó sẽ ảnh hưởng tới tất cả các biến tham chiếu khác.
* Tính bất biến của String giúp tiết kiệm bộ nhớ. Khi chúng ta cố gắng khai báo một đối tượng String mới, JVM sẽ kiểm tra xem giá trị đã tồn tại trong String Pool hay chưa. Tính năng này cho phép Java sử dụng không gian đống một cách hiệu quả.

# **StringBuilder, StringBuffer, StringTokenizer**

* Giao diện CharSequence được sử dụng để biểu diễn chuỗi ký tự. Các lớp String, StringBuffer và StringBuilder implements nó vậy nên có thể tạo chuỗi với cả ba lớp này.



String trong Java là bất biến (immutability) có nghĩa là nó không thể thay đổi được. Đối với các chuỗi có thể thay đổi (mutable), có thể sử dụng các lớp StringBuffer và StringBuilder.

* StringBuilder và StringBuffer là giống nhau, nó chỉ khác biệt tình huống sử dụng có liên quan tới đa luồng (Multi Thread). => về tốc độ xử lý StringBuilder là tốt nhất, sau đó StringBuffer và cuối cùng mới là String.

|  |  |
| --- | --- |
| **StringBuffer** | **StringBuilder** |
| Đồng bộ (synchronized) tức là luồng an toàn: không thể có 2 luồng cùng truy cập phương thức của lớp StringBuffer đồng thời. | Không đồng bộ (non-synchronized) tức là luồng không an toàn: có thể có 2 luồng cùng truy cập phương thức của lớp StringBuilder đồng thời. |
| StringBuffer không hiệu quả bằng StringBuilder. | StringBuilder hiệu quả hơn StringBuffer. |

## **StringBuilder (since 1.5)**

* Các Constructor của StringBuilder

|  |  |
| --- | --- |
| **StringBuilder()** | Tạo ra một StringBuilder với dung lượng ban đầu là 16 |
| **StringBuilder(String str)** | Tạo ra một StringBuilder với chuỗi cụ thể |
| **StringBuilder(int capacity)** | Tạo ra một với dung lượng cụ thể |

* *Note: Capacity vs Length*
* *Capacity là dung lượng của mảng*
* *Lenght là kích thước của mảng (bao nhiêu vị trí của mảng đó đã được sử dụng)*

StringBuilder strBuild = **new** StringBuilder(1000);

strBuild.append("testing");

System.***out***.println("Capacity = " + strBuild.capacity()); // Capacity = 1000

System.***out***.println("Length = " + strBuild.length()); // Length = 7

*Khi số lượng ký tự của chuỗi tăng lên thì dung lượng được tính theo công thức (dung lượng cũ\*2)+2*

* Các phương thức của StringBuilder

|  |  |
| --- | --- |
| **append(String s),**  **append(int i),**  **append(float f),…** | Nối chuỗi vào vị trị cuối của chuỗi được chỉ định. |
| **insert(int offset, String s)**  **insert(int, int),**  **insert(int, float),** | Chèn chuỗi s vào vị trí offset của chuỗi được chỉ định. |
| **replace(int startIndex,**  **int endIndex, String str)** | Thay thế chuỗi từ vị trị startIndex đến endIndex bằng chuỗi str. |
| **delete(int startIndex,**  **int endIndex)** | Xóa chuỗi từ vị trí startIndex đến endIndex. |
| **reverse()** | Đảo ngược chuỗi |
| **capacity()** | Trả về dung lượng của chuỗi |
| **length()** | Trả về độ dài của chuỗi |
| **substring(int beginIndex),**  **substring(int beginIndex,**  **int endIndex)** | Trả về chuỗi con |
| **charAt(int index)** | Trả về kí tự tại chỉ số index |
| **indexOf(String s)** | Trả về chỉ số của chuỗi s |

## **StringBuffer (since JDK1.0)**

* Các Constructor của StringBuffer tương tự như StringBuilder.
* Các phương thức trong StringBuffer

Tương tự như StringBuilder, tuy nhiên:

* Các phương thức append, insert, replace, delete, reverse: là phương thức có đồng bộ hóa.
* Còn charAt, length, capacity, substring: tương tự như StringBuilder.
* *Note:* ***Synchronization***

*Đồng bộ trong java (Synchronization in java) là khả năng kiểm soát truy cập của nhiều luồng đến bất kỳ nguồn tài nguyên chia sẻ (shared resource).*

*Giả sử có nhiều luồng muốn truy cập cùng một biến cùng một thời điểm. Ví dụ một luồng muốn đọc, trong khi luồng khác cố gắng thay đổi dữ liệu dẫn dến dữ liệu bị sai lệch. Trong trường hợp này, Java Synchronization là cho phép chỉ một luồng được truy cập vào tài nguyên chia sẻ đó.*

## **StringTokenizer (since 1.0)**

Dùng để phân tách một chuỗi thành các phần tử token.

Delim (delimiters) là một tập hợp các dấu phân cách

* Các Constructor của StringTokenizer

|  |  |
| --- | --- |
| StringTokenizer(String str) | Với delim mặc định là newline, space, tab, carriage return, and form feed. |
| StringTokenizer(String str, String delim) | Với delim được chỉ định |
| StringTokenizer(String str, String delim, boolean flag) |  |

* Các phương thức trong StringTokenizer

|  |  |
| --- | --- |
| **hasMoreTokens()** | Trả về true nếu còn nhiều token trong chuỗi. |
| **nextToken()** | Trả về token tiếp theo (String) khi duyệt đối tượng StringTokenizer. |
| **nextToken(String delim)** | Trả về token tiếp theo dựa trên dấu phân tách. |
| **hasMoreElements()** | Giống như phương thức hasMoreTokens(). |
| **nextElement()** | Giống như nextToken() nhưng nó trả về một đối tượng. |
| **countTokens()** | Trả về tổng số lượng của các token. |

# **RegEx**

RegEx (Regular Expression) là một chuỗi kí tự tạo nên search pattern (một mẫu để tìm kiếm hoặc thao tác với chuỗi).

Import thư viện java.util.regex để thao tác tới RegEx, thư viện này bao gồm các class:

* Pattern Class: Định nghĩa một pattern (được sử dụng trong tìm kiếm)
* Matcher Class: Được sử dụng để tìm kiếm pattern
* PatternSyntaxException Class: Chỉ ra lỗi cú pháp trong một RegEx pattern.