

La classe **String** (inclusa nel package **java.lang**)

La classe consente di creare e manipolare stringhe di caratteri. La classe è *final* e quindi le sue funzionalità non possono essere estese dall'utente. Il valore di una stringa è pure *final* per cui una stringa è un *oggetto immutabile*, questo non costituisce in pratica una limitazione dato che i metodi di cui è fornita la classe consentono di elaborare nuovi valori da un oggetto `String`.

Gli oggetti di tipo `String` possono essere inizializzati in più modi, ma normalmente degli 11, i **costruttori** più utilizzati sono due:

- **public String ()**: che costruisce una stringa vuota (perché sia allocata è necessario utilizzare l'istruzione `nomeStringa = new String ()`);
- **public String (String value)**: che costruisce una stringa contenente la stringa `value`.

I principali metodi della classe sono:

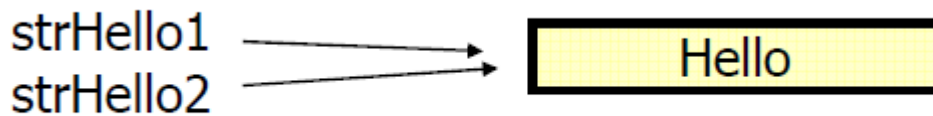
- **int length()**: lunghezza della stringa
- **String substring(int inizio)**: sottostringa dalla posizione `inizio` sino alla fine della stringa
- **String substring(int inizio, int dopolafine)**: sottostringa dalla posizione `inizio` fino al carattere che precede `dopolafine`
- **char charAt(int indice)**: restituisce il carattere la cui posizione è indicata dal valore di `indice`
- **string.concat (String str)**: restituisce una nuova stringa ottenuta dalla concatenazione delle stringa `string` con la stringa `str`;
- **string.equals (String str)**: confronta la stringa `string` con la stringa `str`, restituendo `true` se risultano uguali;
- **string.equalsIgnoreCase (String str)**: confronta la stringa `string` con la stringa `str` ignorando la differenza tra lettere minuscole e maiuscole;
- **string.compareTo (String str)**: confronta la stringa `string` con la stringa `str`, restituendo `0` se risultano uguali, un numero positivo se `str` precede in ordine alfabetico `string` e un numero negativo se `str` anticipa segue in ordine alfabetico `string`; la classe realizza l'interfaccia `Comparable`
- per ricerche in una stringa:
 - **int indexOf(char c[,int Start])**: restituisce la posizione del carattere `c` nella stringa oppure `-1` se il carattere non è presente
 - **int indexOf(String s[,int Start])**: restituisce la posizione della sottostringa `s` nella stringa oppure `-1` se la sottostringa non è presente.
 - **lastIndexOf(...)**: ricerca l'ultima posizione del carattere o della stringa
 - **startsWith(String prefisso [,int offset])** : verifica se combacia il prefisso, eventualmente a partire da una posizione data
 - **sendsWith(String suffisso)**: verifica se combacia il suffisso
- [altri](#) metodi:
 - **replace(char old, char new)** : sostituisce il carattere
 - **toLowerCase()** : restituisce stringa con tutti i caratteri minuscoli
 - **toUpperCase()** : restituisce stringa con tutti i caratteri maiuscoli
 - **toCharArray()** : converte la stringa in un array di `char`

Confronto tra String

http://www.diit.unict.it/users/alongheu/linguaggi/aa0910/lezione06_stringhe_e_array.pdf

In alcuni casi è possibile utilizzare ‘==’ per confrontare due Strings, oltre che `equals()`:

```
String strHello1 = "Hello";  
String strHello2 = "Hello";
```



Il compilatore è “furbo” abbastanza per riconoscere che le due stringhe sono identiche. Quindi decide di risparmiare memoria ed utilizzare la stessa locazione di memoria. I due riferimenti `strHello1` e `strHello2` puntano alla stessa locazione di memoria, per cui in tal caso il confronto `strHello1==strHello2` da *true*.

Lo stesso risultato si ottiene scrivendo: `String strHello2 = “Hell” + “o”;`

Il caso speciale per “==” nel confronto fra oggetti String **NON SEMPRE FUNZIONA**, in particolare:

- se un oggetto String è creato con l’uso della parola chiave ‘**new**’,
- o se i valori sono dati **in input dall’utente**,

i due oggetti String **non occuperanno** comunque **lo stesso spazio di memoria**, anche se i caratteri sono gli stessi.

Pertanto conviene in generale non confrontare String con “==”, utilizzarlo solo per confrontare tipi primitivi e utilizzare `equals()` per confrontare oggetti.

“==” però funziona correttamente se si applica il **metodo intern()** ad entrambe le stringhe, quindi:

```
s1.equals(s2)           oppure           s1.intern()==s2.intern()
```

l’uso di `intern()` permette l’uso di “==”, più veloce di `equals()`