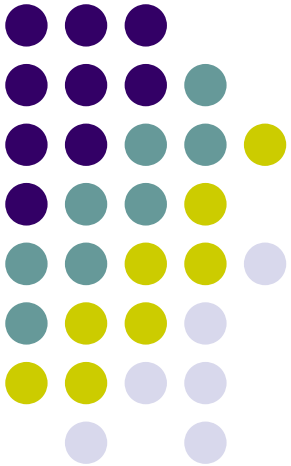
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

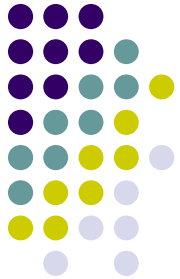
DIETI

Corso di Programmazione

Stringhe

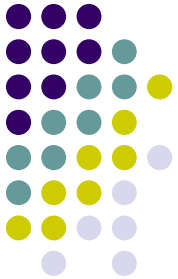


Lavorare con le stringhe



- Le tecniche di manipolazione e gestione delle stringhe sono adatte per
  - validare input dei programmi,
  - visualizzare informazioni per gli utenti
  - eseguire altre manipolazioni basate sul testo, oltre che
    - sviluppare editor di testo, elaboratori di testo, software di impaginazione e in generale software per lavorare con il testo.

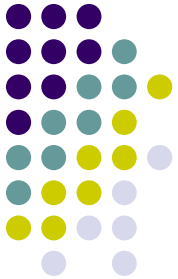
Stringhe di caratteri



- Una stringa è una sequenza di caratteri trattata come un'unica entità.
- Una stringa può includere lettere, cifre e **caratteri speciali**, come +, -, *, / e $.
- Una stringa è un oggetto di classe String.
- Gli oggetti memorizzati String possono essere scritti come sequenze di caratteri tra apici doppi, come per esempio:

"Diego Maradona"	(un nome)
"Via Claudio, 21"	(un indirizzo)
"Napoli, Italia"	(una città e uno stato)
"(+39) 081-121212"	(un numero di telefono)

Ancora su letterali

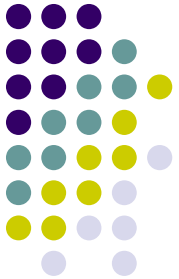


- Un letterale stringa può essere associato a un riferimento String.
- La dichiarazione
  - `String colore = "azzurro";`

inizializza la variabile colore di tipo String per fare riferimento a un oggetto String che contiene la stringa "azzurro".

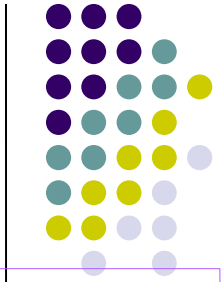
- Per risparmiare spazio di memoria, Java tratta tutti i letterali stringa aventi il medesimo contenuto come un unico oggetto *String con più riferimenti.*

Costruttori di Stringa



- La classe String è utilizzata in Java per rappresentare stringhe.
- La classe String fornisce costruttori per inizializzare gli oggetti String in molti modi diversi.

Esempi di costruttori di String

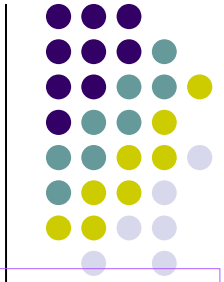


```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
         // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

Esempi di costruttori di String



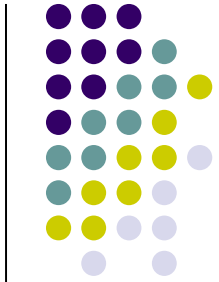
```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
         // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

- Istanzia una nuova String a partire da una sequenza di caratteri racchiusi tra doppi apici.

Esempi di costruttori di String



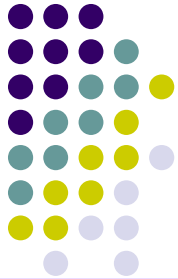
```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
        // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

- Istanzia una nuova String utilizzando un costruttore senza argomento della classe String e ne assegna il riferimento a s1.
- Il nuovo oggetto String non contiene alcun carattere, cioè è una **stringa vuota**, che può anche essere rappresentata come "", e ha una lunghezza 0.

Esempi di costruttori di String



```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
        // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

- Istanzia un nuovo oggetto String utilizzando il costruttore della classe String che prende un riferimento ad una stringa esistente come argomento e copia lo stato di tale stringa nella nuova stringa riferita da s2
- Il nuovo oggetto String riferito da s2 contiene quindi la stessa sequenza di caratteri dell'oggetto riferito da s

Esempi di costruttori di String

```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

ATTENZIONE!
Gli oggetti *String son*o immutabili, perché **la classe *String n*on fornisce metodi che permettono di modificare il contenuto di un oggetto *String d*opo la sua creazione**;

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

- Istanzia un nuovo oggetto String utilizzando il costruttore della classe String che prende un oggetto String come argomento e ne assegna il riferimento a s2
- Il nuovo oggetto String contiene la stessa sequenza di caratteri dell'oggetto s di tipo String che viene passato come argomento al costruttore.

Esempi di costruttori di String

```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
        // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.printf(
       "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3,
s4);
    }
```

Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

- Istanzia un nuovo oggetto String e ne assegna il riferimento a s3 utilizzando il costruttore della classe String che prende un riferimento ad un array di caratteri come argomento.
- Il nuovo oggetto String contiene una copia dei caratteri dell'array.

Esempi di costruttori di String

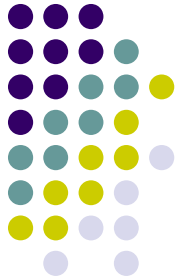
```
public static void main(String[] args) {
        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd',
'a', 'y'};
         // usa costruttori String

        String s = new String("hello");
        String s1 = new String();
        String s2 = new String(s);
        String s3 = new String(charArray);
        String s4 = new String(charArray, 6, 3);
        System.out.pr
       "s1 = %s%ns2 =
s4);
    }
```

- Istanzia un nuovo oggetto String e ne assegna il riferimento a s4 utilizzando il costruttore della classe String che prende un riferimento ad un array di caratteri e due interi come argomenti.
- Il secondo argomento specifica la posizione di partenza (*offset*) da cui si accede ai caratteri dell'array. Il terzo argomento specifica il numero di caratteri (*count*) a cui accedere nell'array;
- Il nuovo oggetto String è formato da tali caratteri.
- Se gli argomenti specificati portano all'accesso a un elemento

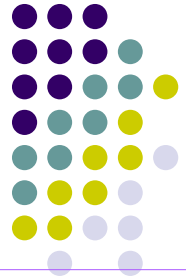
Output

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

I metodi **length** , **charAt** e **getChars**



- I metodi di **length**, **charAt** e **getChars** di String, rispettivamente,
  - **lenght** determina la lunghezza di una stringa,
  - **charAt** prende un argomento di tipo intero e restituisce il carattere che si trova in quella specifica posizione nella stringa,
  - **getChars** copia un insieme di caratteri da una stringa in un array di caratteri passato come argomento.
    - Questo metodo prende quattro argomenti: l'indice nella stringa a partire dal quale devono essere copiati i caratteri, l'indice del carattere che **segue** l'ultimo carattere da copiare, l'array di caratteri nel quale devono essere copiati i caratteri e la posizione nell'array a partire dal quale i caratteri devono essere



Esempio

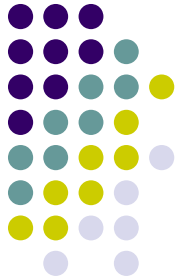
```
public static void main(String[] args) {
        char[] pivot = new char[5];
        String s = new String ("Diego");
        s.getChars(1,  3, pivot, 0);
        System.out.printf("La lunghezza della stringa è: %d%n",
s.length());
        System.out.printf("Il primo carattere della stringa è: %c%n",
s.charAt(0));
        System.out.printf("Il primo carattere estratto è %c
%n",pivot[0]);
    }
```

Output

```
La lunghezza della stringa è: 5
Il primo carattere della stringa è: D
Il primo carattere estratto è i
```

ATTENZIONE: se si sfora la dimensione della stringa (ad esempio tramite i comandi `charAt` e `getChars`) si ottiene una eccezione java.lang.StringIndexOutOfBoundsException

Confronto tra Stringhe

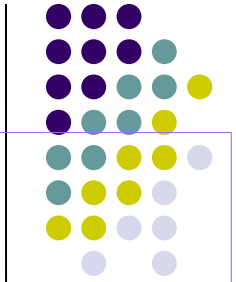


- La classe String fornisce i metodi per *confrontare* le stringhe.
- Per comprendere cosa significa che una stringa è più o meno grande di un'altra, provate a pensare al procedimento che seguireste per mettere in ordine alfabetico una serie di cognomi.
- Il confronto tra stringhe è ***lessicografico***
  - Tutti i caratteri sono rappresentati mediante codici numerici (code points)
  - Il confronto tra stringhe è effettuato confrontando i codici numerici dei caratteri al loro interno.

Metodi per il confronto

- I metodi **equals**, **equalsIgnoreCase**, **compareTo** e **regionMatches** di tipo String
  - **equalsIgnoreCase**, *esegue un confronto senza fare distinzione fra maiuscole e minuscole*
  - **compareTo** *se le stringhe sono uguali restituisce 0; se la stringa che invoca il metodo compareTo è minore di quella passata come argomento restituisce un numero negativo, mentre se è maggiore restituisce un numero positivo*
  - **regionMatches** consente di *confrontare porzioni di due stringhe e verificarne l'uguaglianza*
- Nel prossimo esempio vedremo l'utilizzo di **equals** e dell'operatore di uguaglianza ==

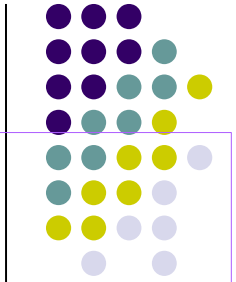
Esempio di confronto



```
public static void main(String[] args) {
        String s1 = new String("Diego"); // s1 è una copia di "hello
             // test di uguaglianza
        if (s1.equals("Diego")) { // vero
            System.out.println("s1 equals \"Diego\"");
        }
        else {
            System.out.println("s1 does not equal "Diego\"");
        }
        // test di uguaglianza con ==
        if (s1 == "Diego") { // falso; non sono lo stesso oggetto
            System.out.println("s1 is the same object as "Diego\"");
        }
        else {
            System.out.println("s1 is not the same object as "Diego\"");
        }
    }
```

- Il metodo equals (un metodo di Object sovrascritto in String) confronta tra loro la stringa s1 e il letterale stringa "hello" e determina se i contenuti delle due stringhe sono *identici*. Se lo sono, restituisce true; altrimenti, restituisce false.
- La condizione precedente è vera perché la stringa s1 era stata inizializzata con il letterale "hello".
- Il metodo equals utilizza un **confronto lessicografico**: confronta i valori interi Unicode, che rappresentano ciascun carattere in ogni stringa.
- Quindi, se la stringa "Diego" viene confrontata con la stringa "DIEGO", il risultato è false, perché la rappresentazione con interi di una lettera minuscola è diversa da quella della maiuscola corrispondente.
- Si può utilizzare equalsIgnoreCase

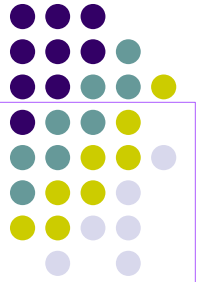
Esempio di confronto



```
public static void main(String[] args) {
        String s1 = new String("Diego"); // s1 è una copia di "hello
             // test di uguaglianza
        if (s1.equals("Diego")) { // vero
            System.out.println("s1 equals \"Diego\"");
        }
        else {
            System.out.println("s1 does not equal \"Diego\"");
        }
        // test di uguaglianza con ==
        if (s1 == "Diego") { // falso; non sono lo stesso oggetto
            System.out.println("s1 is the same object as \"Diego\"");
        }
        else {
            System.out.println("s1 is not the same object as \"Diego\"");
        }
    }
```

- In questo caso l'operatore di uguaglianza == è usato per confrontare la stringa s1 con il letterale stringa "hello".
- Quando si confrontano valori di tipi primitivi utilizzando == , il risultato è true se entrambi i valori sono identici;
- quando si confrontano riferimenti, il risultato è true se entrambi i riferimenti fanno riferimento al medesimo oggetto in memoria.

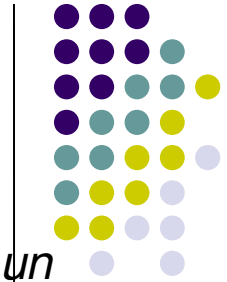
Esempio di confronto



```
public static void main(String[] args) {
        String s1 = new String("Diego"); // s1 è una copia di "hello
             // test di uguaglianza
        if (s1.equals("Diego")) { // vero
            System.out.println("s1 equals \"Diego\"");
        }
        else {
            System.out.println("s1 does not equal \"Diego\"");
        }
        // test di uguaglianza con ==
        if (s1 == "Diego") { // falso; non sono lo stesso oggetto
            System.out.println("s1 is the same object as \"Diego\"");
        }
        else {
            System.out.println("s1 is not the same object as \"Diego\"");
        }
    }
```

- Per confrontare il contenuto effettivo (o indicare informazioni) di oggetti e verificarne l'uguaglianza è necessario invocare un metodo. Nel caso delle stringhe, si invocherà il metodo equals.
- La condizione risulta false perché il riferimento s1 era stato inizializzato con l'istruzione s1 = new String("Diego"); che crea un nuovo oggetto String con una copia del letterale stringa "Diego" e assegna il nuovo oggetto alla variabile s1.
- Se s1 fosse stata inizializzata con l'istruzione s1 = "Diego"; che assegna direttamente il letterale stringa "Diego" alla variabile s1, la condizione sarebbe risultata true. Si ricordi che **Java tratta tutti gli oggetti letterali stringa aventi il medesimo contenuto come un unico oggetto String con più riferimenti. Quindi se avessimo avuto più letterali "Diego", tutti**

Concatenazione di stringhe



*Il metodo **concat** concatena due oggetti stringa: restituisce il riferimento ad un nuovo oggetto String contenente i caratteri di entrambe le stringhe originali*

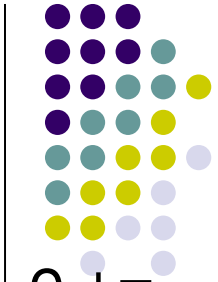
```
public static void main(String[] args) {
        String s1 = "Diego ";
        String s2 = "Maradona";
        System.out.printf("s1 = %s%ns2 = %s%n%n",s1, s2);
        System.out.printf(
        "Result of s1.concat(s2) = %s%n", s1.concat(s2));
        System.out.printf("s1 after concatenation = %s%n", s1);
}
```

Output

```
s1 = Diego
s2 = Maradona

Result of s1.concat(s2) = Diego Maradona
s1 after concatenation = Diego
```

Equivalente a s1+s2

Operatori + e +=

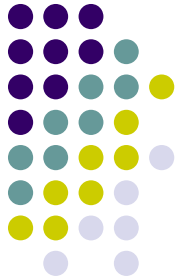


- E' possibile concatenare le stringhe con gli operatori + e +=
- Ogni valore primitivo e ogni oggetto in Java ha una propria rappresentazione di tipo *String*
- Quando uno dei due operandi di + è una *String*, i **valori primitivi** sono convertiti allo stesso tipo, e i due sono quindi concatenati

```
System.out.print("valore corrente del contatore c dopo l'incremento: " +
c.get_value());
```

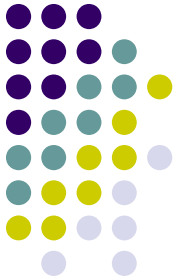
- Un valore booleano è tradotto in una delle due stringhe *"true"* e *"false»*
- *Che succede se l'altro operando è invece un riferimento ad oggetto?*

Il metodo *toString*



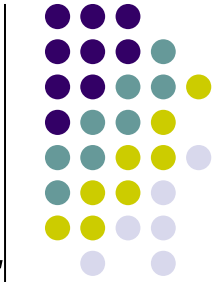
- In Java tutti **gli oggetti** hanno un metodo ***toString*** che restituisce una loro rappresentazione in formato *String*
- Quando un oggetto è concatenato a una stringa, **Java invoca implicitamente il metodo *toString*** di quell'oggetto e ne restituisce la rappresentazione in formato *String*
- Il metodo *toString* può anche essere invocato esplicitamente.

(Alcuni) altri metodi di String



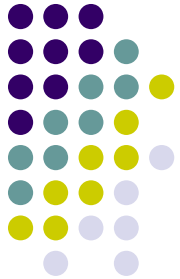
- Localizzazione
  - *indexOf – metodi overloaded* che ricercano uno specifico carattere in una stringa
  - *lastIndexOf - metodi overloaded* che ricercano una specifica sottostringa in una stringa
- Estrazione
  - *substring - metodi overloaded* che permettono di creare un nuovo oggetto stringa copiando una parte di un oggetto stringa già esistente

(Alcuni) altri metodi di String



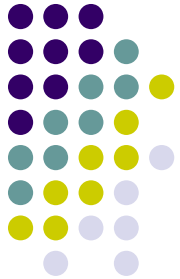
- *Metodi che restituiscono stringhe o array di caratteri che contengono **copie modificate** dei contenuti originali di una stringa:*
  - *replace*
  - *toLowerCase*
  - *toUpperCase*
  - *trim*
  - *toCharArray*
  - *…*
- ***Nessuno di questi metodi modifica la stringa sulla quale vengono invocati***

Il metodo *valueOf*



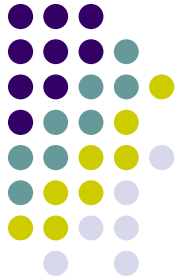
- *La classe String fornisce i metodi static valueOf che prendono un argomento* ***di qualsiasi tipo*** *e lo convertono in un oggetto stringa.*
- valueOf è definito: per i tipi primitivi, per gli array (quindi è possibile usare valueOf per ottenere una stringa da un array di caratteri) e per gli oggetti (attraverso il metodo toString)

La classe StringBuilder



- La classe **StringBuilder** è utile per creare e manipolare informazioni di stringhe *dinamiche*, cioè stringhe *modificabili*.
- Ogni StringBuilder ha una *capacità*, che rappresenta il numero massimo di caratteri che la stringa può contenere.
  - Se la capacità dello StringBuilder viene superata, si espanderà automaticamente per poter contenere i caratteri addizionali.

Costruttori di StringBuilder

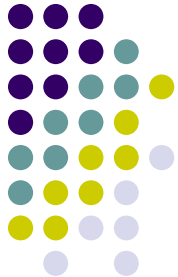


```
public static void main(String[] args) {
        StringBuilder buffer1 = new StringBuilder();
        StringBuilder buffer2 = new StringBuilder(10);
        StringBuilder buffer3 = new StringBuilder("Maradona");
        System.out.printf("buffer1 = \"%s\"%n", buffer1);
        System.out.printf("buffer2 = \"%s\"%n", buffer2);
        System.out.printf("buffer3 = \"%s\"%n", buffer3);
}
```

Output

```
buffer1 = ""
buffer2 = ""
buffer3 = "Maradona"
```

Costruttori di StringBuilder



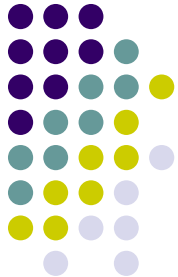
```
public static void main(String[] args) {
        StringBuilder buffer1 = new StringBuilder();
        StringBuilder buffer2 = new StringBuilder(10);
        StringBuilder buffer3 = new StringBuilder("Maradona");
        System.out.printf("buffer1 = \"%s\"%n", buffer1);
        System.out.printf("buffer2 = \"%s\"%n", buffer2);
        System.out.printf("buffer3 = \"%s\"%n", buffer3);
}
```

Output

```
buffer1 = ""
buffer2 = ""
buffer3 = "Maradona"
```

- Costruttore StringBuilder senza argomenti per creare uno StringBuilder che non contiene caratteri e con una capacità iniziale di 16 caratteri (capacità di default per uno StringBuilder).

Costruttori di StringBuilder



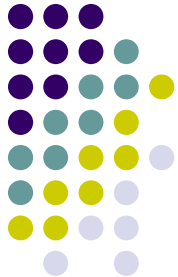
```
public static void main(String[] args) {
        StringBuilder buffer1 = new StringBuilder();
        StringBuilder buffer2 = new StringBuilder(10);
        StringBuilder buffer3 = new StringBuilder("Maradona");
        System.out.printf("buffer1 = \"%s\"%n", buffer1);
        System.out.printf("buffer2 = \"%s\"%n", buffer2);
        System.out.printf("buffer3 = \"%s\"%n", buffer3);
}
```

Output

```
buffer1 = ""
buffer2 = ""
buffer3 = "Maradona"
```

- Costruttore StringBuilder che prende un intero come argomento per creare uno StringBuilder che non contiene caratteri e con una capacità iniziale specificata dall'argomento intero (ovvero 10)

Costruttori di StringBuilder



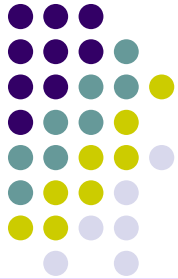
```
public static void main(String[] args) {
        StringBuilder buffer1 = new StringBuilder();
        StringBuilder buffer2 = new StringBuilder(10);
        StringBuilder buffer3 = new StringBuilder("Maradona");
        System.out.printf("buffer1 = \"%s\"%n", buffer1);
        System.out.printf("buffer2 = \"%s\"%n", buffer2);
        System.out.printf("buffer3 = \"%s\"%n", buffer3);
}
```

Output

```
buffer1 = ""
buffer2 = ""
buffer3 = "Maradona"
```

- Costruttore StringBuilder che prende un argomento String per creare uno StringBuilder che contiene i caratteri dell'argomento.
- La capacità iniziale è il numero dei caratteri nell'argomento stringa

Costruttori di StringBuilder



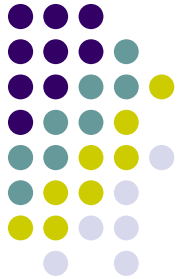
```
public static void main(String[] args) {
        StringBuilder buffer1 = new StringBuilder();
        StringBuilder buffer2 = new StringBuilder(10);
        StringBuilder buffer3 = new StringBuilder("Maradona");
        System.out.printf("buffer1 = \"%s\"%n", buffer1);
        System.out.printf("buffer2 = \"%s\"%n", buffer2);
        System.out.printf("buffer3 = \"%s\"%n", buffer3);
}
```

Output

```
buffer1 = ""
buffer2 = ""
buffer3 = "Maradona"
```

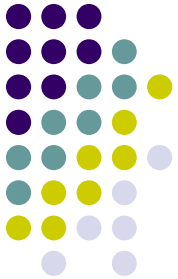
- Usano implicitamente il metodo **toString** della classe **StringBuilder** per stampare gli **StringBuilder** con il metodo **printf**.

I metodi **length** , **capacity** , **setLength** ed **ensureCapacity**



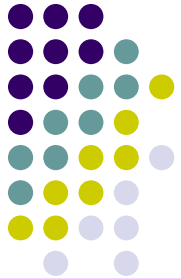
- **length** e **capacity** restituiscono rispettivamente il numero di caratteri effettivamente contenuti in uno StringBuilder e il numero di caratteri che possono essere memorizzati senza allocare ulteriore memoria.
- **ensureCapacity** garantisce che uno StringBuilder abbia almeno la capacità specificata.
- **setLength** aumenta o diminuisce la lunghezza di uno StringBuilder.

I metodi **length** , **capacity** , **setLength** ed **ensureCapacity**



- **length** e **capacity** restituiscono rispettivamente il numero di caratteri effettivamente contenuti in uno StringBuilder e il numero di caratteri che possono essere memorizzati senza allocare ulteriore memoria.
- **ensureCapacity** garantisce che uno StringBuilder abbia almeno la capacità specificata.
- **setLength** aumenta o diminuisce la lunghezza di uno StringBuilder.
- **ATTENZIONE!!!**
  - Aumentare dinamicamente la capacità di uno *StringBuilder* può richiedere un tempo relativamente lungo, ed eseguire molte operazioni di questo tipo può peggiorare le prestazioni di un'applicazione.
  - Se sapete che uno *StringBuilder au*menterà notevolmente di dimensioni, e magari più volte, stabilite sin dall'inizio una capacità elevata per migliorare le prestazioni.

Esempi



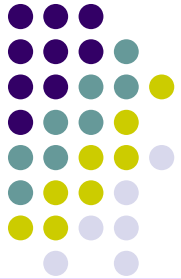
```
public static void main(String[] args) {
        StringBuilder buffer = new StringBuilder("Diego Armando Maradona");
        System.out.printf("buffer = %s%nlength = %d%ncapacity = %d%n%n",
            buffer.toString(), buffer.length(), buffer.capacity());
        buffer.ensureCapacity(75);
        System.out.printf("New capacity = %d%n%n", buffer.capacity());
        buffer.setLength(5);
        System.out.printf("New length = %d%nbuffer = %s%n",
         buffer.length(), buffer.toString());
}
```

```
buffer = Diego Armando Maradona
length = 22
capacity = 38

New capacity = 78

New length = 5
buffer = Diego
```

Output

- Si noti che inizialmente la capacità dello StringBuilder è di 38.
  - Il costruttore StringBuilder che prende un argomento stringa inizializza la capacità alla lunghezza della stringa passata come argomento più 16.

Esempi



```
public static void main(String[] args) {
        StringBuilder buffer = new StringBuilder("Diego Armando Maradona");
        System.out.printf("buffer = %s%nlength = %d%ncapacity = %d%n%n",
            buffer.toString(), buffer.length(), buffer.capacity());
        buffer.ensureCapacity(75);
        System.out.printf("New capacity = %d%n%n", buffer.capacity());
        buffer.setLength(5);
        System.out.printf("New length = %d%nbuffer = %s%n",
         buffer.length(), buffer.toString());
}
```

```
buffer = Diego Armando Maradona
length = 22
capacity = 38

New capacity = 78

New length = 5
buffer = Diego
```

Output

- Espande la capacità dello StringBuilder ad una capacità che equivale al valore maggiore tra il numero specificato come argomento e il doppio della capacità originale più 2.
- La capacità esistente rimane invariata se quella corrente è maggiore della capacità specificata.

Esempi

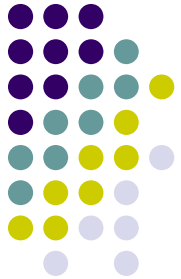
```
public static void main(String[] args) {
        StringBuilder buffer = new StringBuilder("Diego Armando Maradona");
        System.out.printf("buffer = %s%nlength = %d%ncapacity = %d%n%n",
            buffer.toString(), buffer.length(), buffer.capacity());
        buffer.ensureCapacity(75);
        System.out.printf("New capacity = %d%n%n", buffer.capacity());
        buffer.setLength(5);
        System.out.printf("New length = %d%nbuffer = %s%n",
         buffer.length(), buffer.toString());
}
```

Output

```
buffer = Diego Armando Maradona
length = 22
capacity = 38

New capacity = 78

New length = 5
buffer = Diego
```

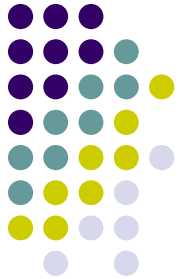
- Imposta la lunghezza dello StringBuilder a 5.
- Se la lunghezza specificata è minore del numero di caratteri corrente nello StringBuilder, il suo contenuto viene ridotto alla lunghezza specificata (ovvero i caratteri nello StringBuilder eccedenti la lunghezza specificata vengono eliminati).
- Se la lunghezza specificata è maggiore del numero di caratteri esistente nello StringBuilder, vengono aggiunti caratteri null (caratteri con rappresentazione numerica 0) finché il numero totale dei caratteri non diventa uguale alla lunghezza

Alcuni metodi per la modifica



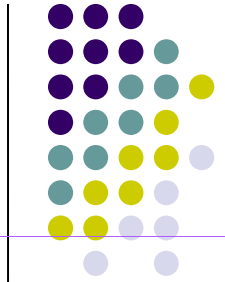
- Inserimento
  - **Append** – metodi overloaded *che consentono di aggiungere valori di diverso tipo alla fine di uno StringBuilder, sono utilizzati per relaizzare + e +=*
  - **Insert** - metodi overloaded *per inserire valori di diverso tipo in qualsiasi posizione in uno StringBuilder*
- Cancellazione
  - **deleteCharAt** - *prende un unico argomento l'indice del carattere da eliminare*
  - **delete** – *prende due argomenti: tutti i caratteri compresi tra l'indice di inizio (incluso) e quello finale (escluso) vengono eliminati*

Altri metodi di StringBuilder



- **setCharAt** prende come argomenti un intero e un carattere e imposta il carattere dello StringBuilder alla posizione specificata al carattere passato come secondo argomento.
- **reverse** inverte il contenuto dello StringBuilder.
- Tentare di accedere a un carattere che si trova al di fuori dei limiti di uno StringBuilder genera un errore (StringIndexOutOfBoundsException).
- **length**, **charAt** e **getChars** sono disponibili anche per StringBuilder

Esempio Replace



```
public static void main(String[] args) {
        StringBuilder buffer = new StringBuilder("Diego Armando
Maradona");
        buffer.replace(0, buffer.length(), "Careca");
        System.out.printf("buffer = %s%n", buffer.toString());
}
```

Output

```
buffer = Careca
```

- Il metodo Java `public` **`StringBuilder replace(int start, int end, String str)`** viene utilizzato per sostituire i caratteri in una sottostringa di un oggetto StringBuilder con i caratteri nella stringa specificata.
- Nell'esempio si è cambiato il valore di tutto il buffer.
  - Usando opportunamente i valori di start ed end è possibile cambiare una sottostringa dello StringBuilder

Riferimenti

- Programmare in Java: Capitolo 14, §14.2, §14.3, §14.4