Java
Le stringhe

Le stringhe in Java

- In Java le stringhe non sono semplicemente array di caratteri terminati con zero come avviene in C
- In Java le stringhe sono oggetti appartenenti alla classe String
- Quindi:
  - E' possibile dichiarare variabili di tipo String
  - E' possibile creare istanze di String
  - E' possibile operare su queste istanze invocando i metodi definiti dalla classe String
- Java definisce però alcune estensioni sintattiche per consentire di lavorare in modo più naturale
- Queste estensioni sono molto comode, ma possono creare inizialmente qualche confusione perché creano l'illusione che le stringhe siano tipi primitivi anziché oggetti

Costanti stringa

- La prima estensione è la possibilità di definire costanti stringa con la stessa sintassi del C:

```
"ciao a tutti"
```

- Le costanti stringa possono essere usate in questo modo:

```
String s;
s = "ciao a tutti";
```

- A prima vista questa sintassi sembra del tutto simile a quella utilizzata con un tipo primitivo:

```
int n;
n = 5;
```

- Attenzione: è solo un illusione, la variabile s è un riferimento ad un oggetto di classe String
- L'uso di una costante stringa implica la creazione di un'istanza
- In pratica le istruzioni che abbiamo visto sopra corrispondono a:

```
String s;
s = new String("ciao a tutti");
```

Concatenazione - 1

- La seconda estensione sintattica è costituita dalla possibilità di utilizzare l'operatore + per concatenare le stringhe:

```
"ciao"+" a tutti"
```

- Anche in questo caso abbiamo una sintassi molto comoda ma che può trarre in inganno perchè appare simile a:

```
5.6+4.9
```

- La concatenazione di due stringhe in realtà genera automaticamente una nuova istanza di String il cui contenuto è costituito dai caratteri della prima e della seconda messi insieme

Concatenazione - 2

- Quindi se scriviamo:

```
String s = "ciao"+" a tutti";
```

- Dietro le quinte succede questo:
  - **Viene creata una prima istanza di String con valore "ciao" (uso della prima costante stringa)**
  - **Viene creata una seconda istanza di String con valore " a tutti"(uso della seconda costante stringa**
  - **Viene creata una terza istanza di String con valore "ciao a tutti" (uso dell'operatore di concatenazione)**
  - **L'assegnamento fa sì che `s` punti alla terza istanza**
  - **Alla fine dell'istruzione non esistono più riferimenti alla prima e alla seconda istanza, le cui aree di memoria (non più referenziate) sono quindi recuperate dal garbage collector**

Immutabilità - 1

- Oltre a creare l'illusione che le stringhe in Java siano dei tipi primitivi le estensioni sintattiche creano anche un'altra confusione: fanno pensare che una stringa possa cambiare

- In Java le stringhe sono immutabili!

- Un'istanza della classe String assume un valore quando viene creata e questo valore non può più essere cambiato
- Vediamo con un esempio come nasce la confusione e come vanno in realtà le cose

Immutabilità - 2

- Consideriamo le istruzioni:

```
1. String s = "ciao ";
2. s = s + " a tutti";
```

- Sembra che all'inizio il valore della stringa sia "ciao" e che poi venga cambiata in "ciao a tutti"
- In realtà:
  - Viene creata una prima istanza di String con valore "ciao" (uso della prima costante)
  - Viene creata una seconda istanza con valore " a tutti" (uso della seconda costante)
  - Viene creata una terza istanza con valore "ciao a tutti" (concatenazione)
  - L'assegnamento fa sì che **`s`** punti alla terza istanza
  - La prima e la seconda istanza non hanno più riferimenti, **e le loro aree di memoria sono quindi recuperate dal garbage collector**
- Nessuna stringa ha cambiato valore!

Sostituzioni

- ```
  1. String s = “ciao ”;
  2. s = s + “ a tutti”;
  ```

- ```
  1. String s = new String(“ciao ”);
  2. s = s + new String(“ a tutti”);
  ```

- ```
  1. String s = new String(“ciao ”);
  2. s = new String(s + new String(“ a tutti”));
  ```
  ```

Riassumendo

- Java introduce due estensioni sintattiche per semplificare l'uso delle stringhe: costanti stringa e concatenazione mediante l'operatore +
- Queste due estensioni sono molto comode, ma creano due illusioni:
  - Che le stringhe siano tipi primitivi
  - Che le stringhe possano cambiare valore
- In realtà:
  - Le stringhe sono oggetti, istanze della classe String, e le variabili di tipo String sono riferimenti a queste istanze
  - Le stringhe sono immutabili: nascono con un valore e non possono cambiarlo

Metodi della classe String - 1

- La classe String definisce molti metodi, vediamone alcuni e mettiamo in luce qualche altro aspetto critico
- s1.length(): restituisce la lunghezza della stringa s1
- s1.charAt(index): restituisce un carattere alla posizione prefissata

💣Attenzione: non possiamo utilizzare la notazione s1[index] come in C: le stringhe non sono array!

- s1.indexOf ('c') ritorna l'indice della prima occorrenza di c in s1 (-1 se non c'è)
- s1.equals(s2): dice se s1 ed s2 hanno lo stesso contenuto

💣Attenzione: s1.equals(s2) è diverso da s1 == s2, s1 ed s2 sono riferimenti!

Metodi della classe String - 2

- s2 = s1.substring(10,18): restituisce la sottostringa che va da 10 a 17 (18-1)

- s2 = s1.replace('E','X'): restituisce una stringa con tutte le 'E' sostituite con 'X'

💣Attenzione: né substring né replace modificano la stringa su cui vengono invocati: ne creano una nuova e restituiscono un riferimento alla stringa appena creata.

Immutabilità - 3

- Consideriamo la seguente sequenza

```
s = "ciao a tutti";
s = s.replace("t","p");
```

- Anche stavolta sembra che s cambi valore, ma in realtà:
  - Viene creata una stringa con valore "ciao a tutti" puntata da s
  - Il metodo replace crea una nuova stringa che ha tutte le 't' sostituite con 'p' e s punta a questa nuova stringa il cui valore è "ciao a puppi"
  - La prima stringa non ha più riferimenti e quindi verrà distrutta dal garbage collector
- Nessuna stringa ha cambiato il proprio valore!

Concatenazione - 3

- L'operatore di concatenazione permette anche di concatenare stringhe e altri tipi
- Per esempio:

```
s = "Numero "+5.7;     // s vale "Numero 5.7"
```

- Oppure

```
int n;
n = 10;
s = "Numero "+n;       // s vale "Numero 10"
```

- In entrambi i casi i valori numerici vengono convertiti in stringa e poi concatenati
- Attenzione: per avere concatenazione il primo elemento dell'espressione deve essere una stringa
- `s = n;` `// non è valido`
- Un utile "trucco" per convertire un numero in stringa:

```
s = ""+n;
```

StringBuffer

- Abbiamo visto che gli oggetti di classe String sono immutabili
- Questo in genere non crea problemi, ma ci possono essere situazioni in cui può bisogna cambiare il valore di una stringa
- In questi casi si usa la classe StringBuffer
- StringBuffer definisce parecchi metodi tra cui segnaliamo `setCharAt(int index, char ch)` che consente di modificare il valore di un carattere della stringa
- Nell'uso più classico:
  - si crea un'istanza di StringBuffer copiando una stringa
  - la si modifica
  - si mette il risultato in una stringa

Esempio di uso di StringBuffer (esercitazione 7 – prima eserc. Java)

- Proviamo a invertire una stringa usando StringBuffer

Occorre creare un oggetto copia di tipo StringBuffer:

```
String s;
StringBuffer sb;
char ch;
s = "ciao a tutti";
sb = new StringBuffer(s);
```

Copiare carattere per carattere in ordine inverso:

```
for (int i=0; i<sb.length()/2; i++)
    {
      ch = sb.charAt(i);
      sb.setCharAt(i,sb.charAt(sb.length()-i-1));
      sb.setCharAt(sb.length()-i-1,ch);
     }
```

Convertire l'oggetto StringBuffer in oggetto String:

```
s = sb.toString();
```

Stampa di oggetti

- Tutte le classi Java definiscono un metodo `toString()` che produce un oggetto di tipo String a partire da un oggetto della classe: ciò consente di “stampare” facilmente qualunque oggetto di qualunque classe
- È responsabilità del progettista definire un metodo `toString()` che produca una stringa “significativa”
- Quello di default stampa un identificativo alfanumerico dell’oggetto

Esempio Counter

```
public class EsempioStampa {
 public static void main(String[] args){
  Counter c = new Counter(10);
  System.out.println(c.toString());
 }
}
```

Usa il metodo toString() predefinito di Counter → Stampa un identificativo dell'oggetto c.

```
Counter@4abc9
```

Esempio Counter - variante

- Se questa stampa non ci piace, è possibile ridefinire esplicitamente il metodo `toString()` della classe `Counter`, facendogli stampare ciò che preferiamo

  Ad esempio:

```
public class Counter {
  ...
 public String toString(){
    return "Counter di valore “+this.val;
  }
}
```

Esempio Counter - variante

Lo stesso identico esempio:

```
public class EsempioStampa {
 public static void main(String[] args){
  Counter c = new Counter(10);
  System.out.println(c.toString() );
 }
}
```

ora stamperà:

```
Counter di valore 10
```

Esempio Counter - variante

Lo stesso identico esempio:

```
public class EsempioStampa {
 public static void main(String[] args){
  Counter c = new Counter(10);
  System.out.println(c);
 }
}
```

ora stamperà:

```
Counter di valore 10
```

Java - Le stringhe – classe String

in Java sono oggetti immutabili

Ma ci sono semplificazioni che ne facilitano la creazione,assegnamento, e concatenazione