Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

26 Ottobre 2022

Stringhe in Java

Le *stringhe* (i.e. sequenze finite di caratteri) in Java sono implementate come *oggetti*.

Esiste la classe predefinita `String` nel pacchetto `java.lang`.

Come ogni altra classe, `String` è una *sottoclasse* della classe `Object`, la classe *madre* da cui derivano tutte le altre classi in Java (*gerarchia di classi*).

La classe `String` è definita `final` (per motivi di ottimizzazione), da ciò segue che non è possibile *estendere* tale classe.

La classe `String` definisce il *tipo* delle stringhe in Java:

```
String s = "abc";
```

Dichiarazione ed inizializzazione per una variabile di tipo `String`, come per una qualsiasi variabile locale.

Stringhe in Java (cont.)

Una *costante* di tipo `String` è una sequenza di caratteri racchiusa tra una coppia di doppi apici `"..."`, e.g. `"ciao!`, `"4+5"`, etc.

N.B. Le stringhe in Java sono oggetti il cui *contenuto non è modificabile*.

Esempio: se, data una stringa *s*, si vuole ottenere una sua sottostringa, *s* non viene modificata per ricavare la sottostringa, ma si genera una *nuova* stringa uguale alla sottostringa cercata.

In Java *le stringhe non sono array di caratteri*, ovvero il tipo `String` è diverso dal tipo `char[]`:

```
char[] c = {'a','b','c'};
String s = c;
```

Si ha un errore in fase di compilazione (tipi non compatibili).

Metodi in Java

Un oggetto è un'*istanza* di una classe.
Le stringhe sono istanze della classe `String`.

Nella classe `String` sono definiti molti *metodi* (operazioni) per lavorare sulle stringhe, ma noi ne introdurremo solo un insieme limitato.

I metodi in Java possono essere:

- *statici*, detti anche *metodi di classe*, in quanto il loro comportamento dipende dalla classe in cui sono definiti (e.g. `main`) e non dagli oggetti su cui sono applicati (ciò è denotato con la parola riservata `static` nell'intestazione del metodo);

- *d'istanza*, il cui comportamento dipende dall'oggetto (i.e. l'istanza della classe) su cui viene applicato.

Chiamata di un metodo

Vedremo più avanti come *chiamare o invocare* un metodo statico.

I metodi che consideriamo per operare sulle stringhe sono *metodi d'istanza*.
Quindi vediamo come chiamare un metodo d'istanza.

Dato un metodo d'istanza `m` della classe `String` ed una stringa `s` su cui invocare il metodo `m`, si scrive:

```
s.m(...)
```

dove le parentesi tonde racchiudono eventuali *parametri espliciti*, ovvero altre informazioni di cui il metodo ha bisogno.

L'oggetto prima del punto `.` (qui denotato da `s`) è il *parametro implicito*.

Alcuni metodi della classe `String`

I parametri espliciti di un metodo d'istanza possono essere assenti, ma le parentesi vanno *sempre* scritte.

N.B. Ogni metodo in Java ha un nome seguito da una coppia di parentesi tonde, *sia nell'intestazione della sua definizione che nella sua chiamata*.

Nel seguito vengono introdotti quei pochi metodi che useremo per lavorare con le stringhe.

Di questi metodi verrà data la sola intestazione (ciò è sufficiente per sapere come invocarli).

La classe `String` contiene *molti* altri metodi.

Lunghezza di una stringa

Metodo `length`

Intestazione

```
public int length()
```

Invocazione: data una stringa `s`

```
s.length()
```

restituisce il numero di caratteri in una stringa.

Esempio:

```
String s = "ciao";
int x = s.length();
```

La variabile `x` vale 4.

N.B. NON confondere il *metodo* `length()` per le stringhe con la *variabile istanza* `length` degli array!

Uguaglianza di stringhe

Metodo `equals`

Intestazione

```
public boolean equals(Object obj)
```

Invocazione: date due stringhe `s1`, `s2`

```
s1.equals(s2)
```

restituisce `true` se il parametro esplicito `s2` è di tipo `String` ed il suo contenuto è uguale a quello del parametro implicito `s1`. Altrimenti restituisce `false`.

Esempio:

```
String s1 = "ciao", s2 = "hello";
boolean p = s1.equals(s2);
```

La variabile `p` vale `false`.

Uguaglianza di stringhe (cont.)

N.B. Le stringhe non si confrontano con l'operatore di confronto ==. L'operatore == NON va usato per confrontare oggetti.

Può essere usato *solo* per confrontare un oggetto con il valore `null`.

Per confrontare due oggetti occorre usare il metodo `equals`, *opportunamente ridefinito* per gli oggetti di una data classe.

Il metodo `equals` è definito nella classe `Object` come segue: dati due riferimenti ad oggetti `x` ed `y` diversi da `null`, si ha che `x.equals(y)` restituisce `true` se e solo se `x` ed `y` riferiscono lo stesso oggetto (i.e. `x==y` ha valore `true`).

Nella classe `String` il metodo `equals` viene ridefinito, ma il nome del metodo non cambia (forma di *polimorfismo*, detta *overriding*).

Java è *case-sensitive* (cfr. il metodo `equalsIgnoreCase`).

Concatenazione di stringhe

Conosciamo già l'operatore + per concatenare elementi, di cui *almeno uno* è una stringa.

Esiste anche il metodo `concat`

Intestazione

```
public String concat(String str)
```

Invocazione: date due stringhe `s1`, `s2`

```
s1.concat(s2)
```

restituisce la stringa risultante dalla concatenazione di `s1` con `s2` (in questo ordine).

Ma è più agevole usare + al posto di `concat`.

Concatenazione di stringhe (cont.)

Esempio:

```
String s1 = "ciao", s2 = "hello";
String t = s1.concat(s2);
```

La variabile `t` vale `"ciaohello"`.

Se poi viene fatto il seguente assegnamento:

```
t = s1 + '!';
```

la variabile `t` vale `"ciao!"`.

Ma se usiamo il metodo `concat` non funziona:

```
t = s1.concat('!');
```

Si ha errore in fase di compilazione (tipi non compatibili).

Estrazione di caratteri

Metodo `charAt`

Intestazione

```
public char charAt(int i)
```

Invocazione: dati una stringa `s` ed un intero `i`

```
s.charAt(i)
```

restituisce il carattere di `s` in posizione `i` (con `i` maggiore o uguale a 0 e minore della lunghezza della stringa, altrimenti viene sollevata un'*eccezione*).
N.B. Anche nelle stringhe le posizioni dei caratteri partono da 0.

Esempio:

```
char c = "esame".charAt(3);
```

La variabile c vale `'m'`.

Estrazione di sottostringhe

Esistono *due* metodi per estrarre sottostringhe e si chiamano entrambi `substring` (*overloading*).

Primo metodo `substring`

Intestazione

```
public String substring(int start)
```

Invocazione: dati una stringa `s` ed un intero `i`

```
s.substring(i)
```

restituisce una *nuova* stringa uguale alla sottostringa di `s` dalla posizione `i` fino alla fine di `s` (stesso vincolo sul valore di `i`).

Esempio:

```
String t = "alternanza".substring(5);
```

La variabile `t` vale `"nanza"`.

Estrazione di sottostringhe (cont.)

Secondo metodo `substring`

Intestazione

```
public String substring(int start, int end)
```

Invocazione: dati una stringa `s` e due interi `i`, `j`

```
s.substring(i,j)
```

restituisce una *nuova* stringa uguale alla sottostringa di `s` dalla posizione `i` fino alla posizione `j-1` *inclusa* (con `i`,`j-1` dentro `s` ed `i` minore di `j`).

Esempio:

```
String t = "alternanza".substring(2,7);
```

La variabile `t` vale `"terna"`.

Valutazione di codice con stringhe

Prova intermedia del 30/11/2012, Esercizio 2

Si consideri il seguente frammento di codice in Java:

```
String s = t; int k = 0; String s1 = "";
while (k < s.length()) {
  if (s.charAt(k) != c)
    s1 = s1 + s.charAt(k);
  else
    s1 = c + s1;
  k++;
}
```

Determinare il valore finale delle variabili `k` ed `s1` nei seguenti casi:

1) `t = "palla"` e `c = 'a'`; (*Sol.* `k = 5`, `s1 = "aapll"`)
2) `t = "palazzo"` e `c = 'z'`. (*Sol.* `k = 7`, `s1 = "zzpalao"`)

Altra valutazione di codice

Prova intermedia del 27/11/2014, Esercizio 2

Si consideri il seguente frammento di codice in Java:

```
String s = t; char[] a = new char[s.length()];
int i;
for (i = a.length-1; i >= 0; i--) {
  if (s.charAt(i) != c)
    a[i] = s.charAt(i);
  else
    a[i] = '+';
}
```

Determinare il valore finale della variabile `i` e degli elementi dell'array `a` nel caso in cui si abbia `t = "qqbfbbz1r"` e `c = 'b'`.

Soluzione

`i = -1`, `a = {'q','q','+','f','+','+','z','1','r'}`.