Stringhe

Walter Didimo

La classe String

L’uso di stringhe (sequenze di caratteri alfanumerici) nei programmi è molto frequente

Per tale motivo, l’API di Java offre una classe con molti metodi utili per la gestione delle stringhe, la classe String

Creazione di oggetti String

Un oggetto della classe *String* rappresenta una sequenza di caratteri nell'alfabeto Unicode 2.0

Un oggetto della classe *String* si crea usando l'operatore *new* ed un costruttore che permette di specificare la sequenza di caratteri che l'oggetto deve rappresentare

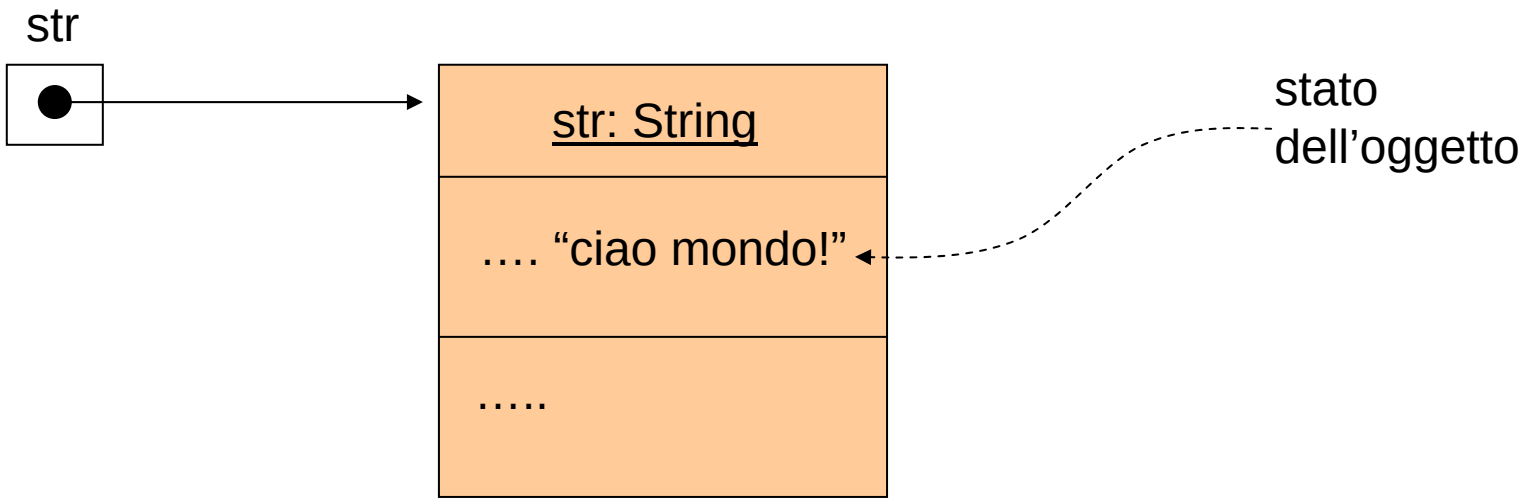
new String ("ciao mondo!")

Creazione di oggetti String

Il riferimento ad un oggetto *String* si può assegnare ad una variabile di tipo *String*

- il suo stato è la sequenza di caratteri rappresentata

String str = new String (“ciao mondo!”)



Letterali String

Un oggetto *String* si può anche creare attraverso l'uso di letterali String

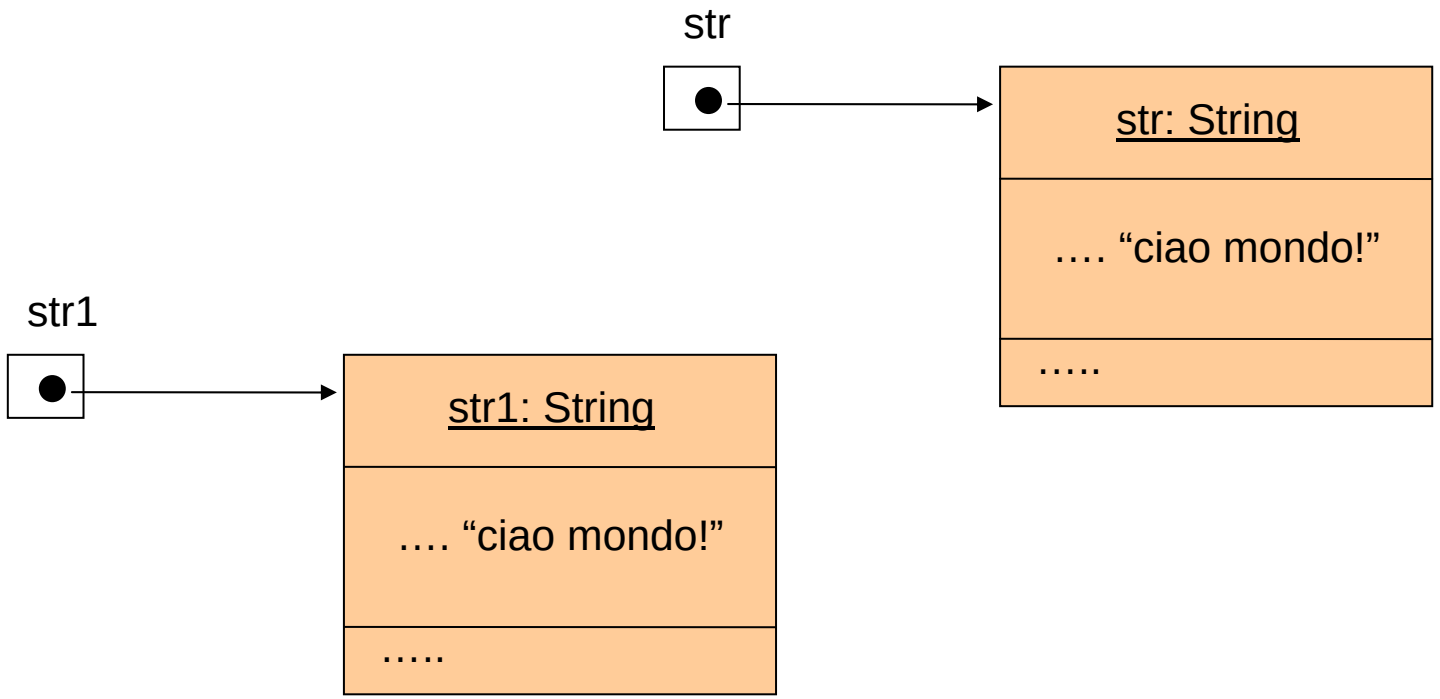
Un letterale *String* è una sequenza di caratteri tra apici doppi e definisce automaticamente un oggetto *String*; si può cioè scrivere direttamente

String str = "ciao mondo!"

Analogie e differenze tra oggetti

Considera il seguente frammento di codice

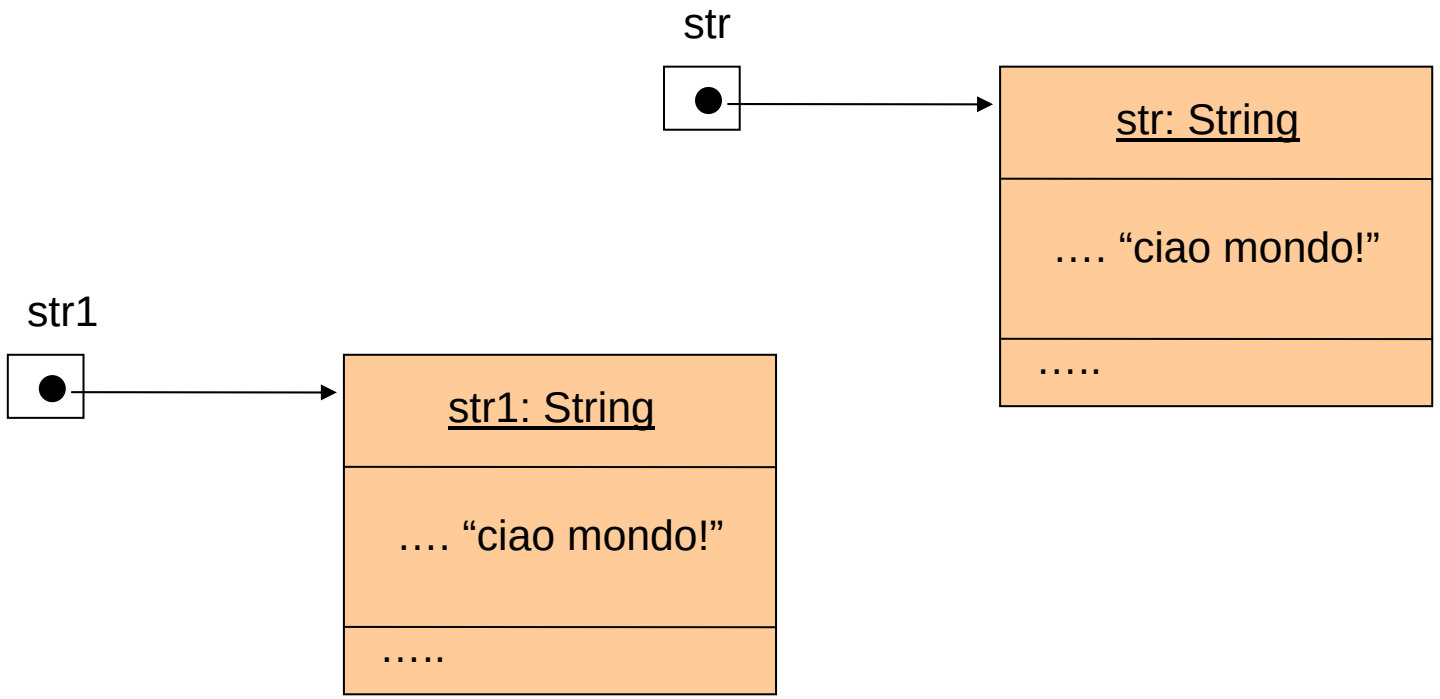
```
String str = new String (“ciao mondo!”);
String str1 = new String (“ciao mondo!”);
```



Analogie e differenze tra oggetti

I due oggetti creati sono analoghi, perché hanno lo stesso stato, ma sono oggetti differenti!!

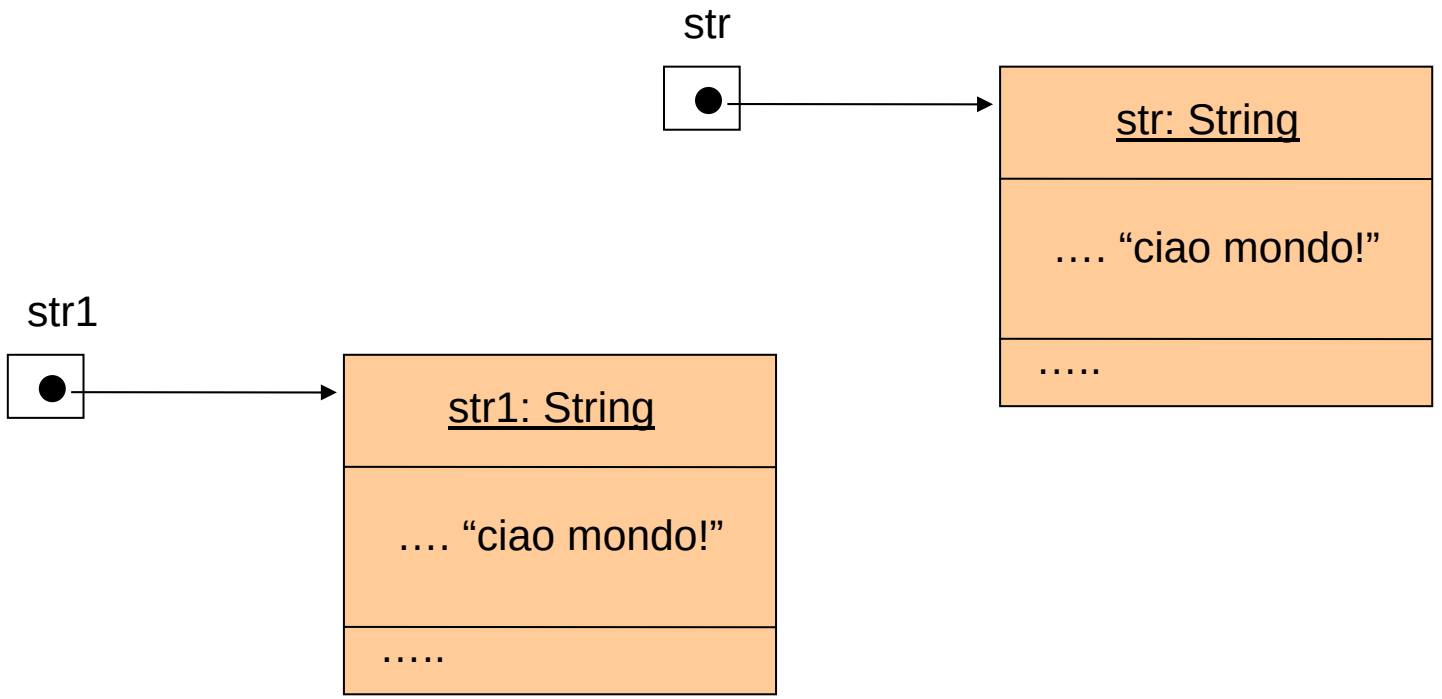
- l'espressione *str == str1* vale *false*



Analogie e differenze tra oggetti

Anche i seguenti oggetti sono differenti!!

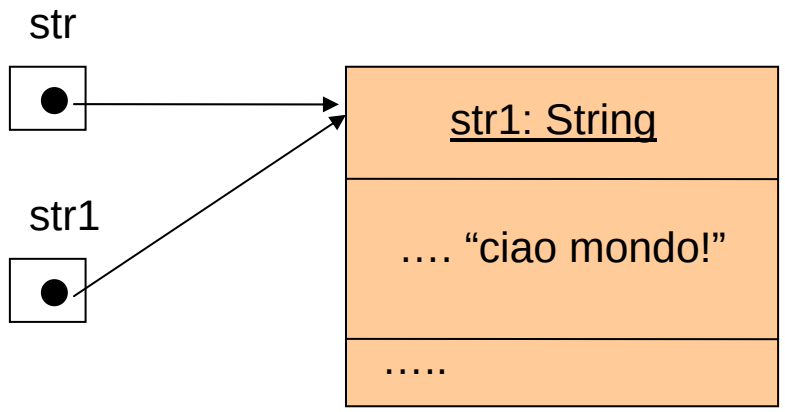
```
String str = new String (“ciao mondo!”);
String str1 = “ciao mondo!”;
```



Analogie e differenze tra oggetti

I seguenti oggetti sono invece identici, perché creati dallo stesso letterale

```
String str = “ciao mondo!”;
String str1 = “ciao mondo!”;
```



La stringa vuota

Anche il letterale *“”* è un letterale *String* – esso rappresenta la stringa vuota

La stringa vuota è la stringa senza caratteri (neanche spazi!!)

Immutabilità degli oggetti String

Un oggetto *String* è immutabile

Questo vuol dire che non c'è alcun modo di cambiare il suo stato una volta creato

Esistono però dei metodi che permettono di ottenere da un oggetto *String* dei nuovi oggetti *String* con un diverso stato

Metodi degli oggetti String

Vogliamo illustrare alcuni metodi degli oggetti *String*, frequentemente utilizzati nei programmi

- *int length()*
- *char charAt (int pos)*
- *boolean equals (String s)*
- *int compareTo (String s)*
- *int indexOf (String s)*
- *String substring (int startPos, int stopPos)*
- *String toUpperCase ()*

Il metodo length

La lunghezza di una stringa è il numero dei suoi caratteri.

- ad esempio, la stringa “ciao mondo!” ha lunghezza 11

Il metodo *length()* può essere invocato su un oggetto *String* per sapere la lunghezza della sequenza di caratteri che l’oggetto rappresenta

```
String str = new String (“ciao mondo!”);
int a = str.length();   // a vale 11
```

Il metodo charAt

In una stringa, ogni carattere occupa una posizione, crescente da sinistra verso destra

- il primo carattere a sinistra occupa la posizione 0, il secondo carattere la posizione 1, e così via (nota che l'ultimo carattere occupa la posizione length() – 1)

Il metodo *charAt* restituisce il carattere (valore *char*) nella posizione specificata

```
String str = new String (“ciao mondo!”);
char c = str.charAt(5);   // c vale ‘m’
```

Il metodo equals

Il metodo *equals* è usato per confrontare due oggetti *String* – se il loro stato è uguale *equals* restituisce *true*, altrimenti restituisce *false*

```
String str = new String (“ciao mondo!”);
String str1 = new String (“ciao mondo!”);
boolean uguali;
uguali = str.equals(str1); // uguali vale true
```

… ricorda che *(str == str1)* vale *false*!!!

- Per capire se due stringhe rappresentano la stessa sequenza di caratteri si deve sempre usare *equals* e non l’operatore relazionale ==

Il metodo compareTo

Il metodo *compareTo* permette di confrontare due stringhe lessicograficamente (ordine alfabetico):

- i caratteri dell'alfabeto Unicode 2.0 sono ordinati – ciò induce automaticamente un ordinamento tra stringhe, che segue le stesse regole del classico ordinamento alfabetico
- Ad esempio:
  - "alba" precede "alfa"
  - "Alfa" precede "alba", perché 'A' precede 'a'
  - "12" precede "A4", perché '1' precede 'A'

Il metodo compareTo

str.compareTo(str1) restituisce:

- un numero negativo se *str* precede *str1*
- 0 se *str* ed *str1* rappresentano la stessa stringa
- un numero positivo se *str* segue *str1*

```
String str = new String (“alfa”);
String str1 = new String (“alba”);
int a = str.compareTo(str1); // a > 0
```

Il metodo indexOf

Il metodo *indexOf* permette di ricercare una sottostringa in una stringa; l'invocazione:

str.indexOf (s)

valuta se la stringa rappresentata da *s* è contenuta in quella rappresentata da *str*; se sì restituisce la posizione di inizio di *s* in *str*, altrimenti restituisce il valore -1

Il metodo indexOf

Ecco un esempio di uso di *indexOf*

```
String str = new String (“ciao mondo!”);
int pos;
pos = str.indexOf (“ao m”); // pos vale 2
pos = str.indexOf (“modo”); // ora pos vale -1
```

Esistono altre varianti del metodo *indexOf* – consultare la documentazione Java per un approfondimento di tale metodo

Il metodo substring

Il metodo *substring* permette di estrarre una sottostringa da una stringa; l'invocazione:

str.substring (inizio, fine)

restituisce un nuovo oggetto *String* che rappresenta la sottostringa di *str* dalla posizione *inizio* (inclusa) alla posizione *fine* (esclusa)

```
String str = new String ("ciao mondo!");
String s = str.substring (3,8);
/* s rappresenta la stringa "o mon" */
```

Il metodo substring

Il metodo *substring* ha una variante, che permette di specificare la sola posizione di inizio:

str.substring (inizio)

restituisce un nuovo oggetto *String* che rappresenta la sottostringa di *str* dalla posizione *inizio* fino alla fine di *str*

```
String str = new String (“ciao mondo!”);
String s = str.substring (3);
/* s rappresenta la stringa “o mondo!” */
```

Il metodo toUpperCase

Il metodo *toUpperCase* serve ad ottenere stringhe di sole lettere maiuscole

str.toUpperCase ()

restituisce un nuovo oggetto *String* ottenuto da *str* convertendo tutte le lettere in maiuscolo

```
String str = new String (“cIaO monDo!”);
String s = str.toUpperCase ();
/* s rappresenta la stringa “CIAO MONDO!” */
```

Esiste anche il metodo *toLowerCase*, che converte le lettere in minuscolo

Concatenazione di stringhe

Abbiamo visto l'uso dell'operatore + per effettuare le somme tra tipi numerici o tra caratteri

L'operatore + è sovraccarico, nel senso che può essere anche usato tra oggetti *String*:

- quando applicato agli oggetti *String*, l'operatore + si chiama operatore di concatenazione
- se *str1* ed *str2* sono due oggetti *String*, l'espressione *str1* + *str2* concatena (cioè giustappone) *str2* ad *str1*

Esempi di concatenazione

```
String str1 = new String (“ciao ”);
String str2 = new String (“mondo!”);
String str = str1 + str2;
/* str referenzia un nuovo oggetto String che
rappresenta la stringa “ciao mondo!” */
```

```
String str1 = new String (“ciao ”);
OutputWindow out = new OutputWindow ();
 /* visualizza “ciao mondo!” su una finestra
grafica */
out.write (str1 + “mondo!”);
```

Concatenazione mista

L’operatore di concatenazione è veramente sovraccarico!!; può essere persino applicato ad operandi di tipo misto

- uno dei due operandi può essere una stringa e l’altro un tipo primitivo; come opera in questi casi?
- il dato di tipo primitivo viene convertito automaticamente ad un valore *String* che lo rappresenta letteralmente; fatto ciò viene effettuata la normale concatenazione tra stringhe

Concatenazione mista

Se ad esempio scrivo:

```
String str = 11 + “ rose”;
```

l’espressione *11 + “ rose”*, viene convertita nell’espressione *“11” + “ rose”* e quindi il risultato sarà *“11 rose”*

Un altro esempio:

```
String str = “”;
str = str + ‘a’;        // str vale ora “a”
str = str + ‘b’;        // str vale ora “ab”
str = str + ‘c’;        // str vale ora “abc”
```

Espressioni con il +

L’operatore di concatenazione + con tipi misti è molto utile, come vedremo più avanti. Occorre però fare attenzione a come si usa:

```
String str = “novembre ” + 2 + 7
// così str vale “novembre 27”


String str = 2 + 7 + “ novembre”
// così str vale “9 novembre”
```

Glossario dei termini principali

Termine	Significato
String	Classe dell'API di Java, i cui oggetti rappresentano sequenze di caratteri nell'alfabeto Unicode 2.0
Letterale String	Sequenza di caratteri tra apici doppi
Stringa vuota	La stringa che non contiene caratteri, cioè ""
Lunghezza di una stringa	Numero dei caratteri che compongono una stringa
Ordine lessicografico	Ordine utilizzato per ordinare le stringhe (analogo all'ordine alfabetico relativamente alle stringhe dei linguaggi naturali)
Operatore di concatenazione	Operatore binario che concatena due stringhe (+). L'operatore di concatenazione può essere anche applicato tra stringhe e tipi primitivi