Stringhe di caratteri in Java: la classe String

Stringhe e classe String

- Una stringa è una sequenza finita di caratteri.

- In Java la rappresentazione di informazioni di tipo testuale (sequenze di caratteri) avviene mediante l'uso di oggetti della classe `String` che rappresentano **stringhe di caratteri**.

- Non esiste il tipo primitivo stringa ma esiste una classe predefinita. → **le stringhe sono oggetti in Java**.

- Le operazioni su stringhe sono realizzate mediante metodi della classe `String`.

- Un oggetto `String` rappresenta una sequenza finita di caratteri dell'alfabeto Unicode.

Stringhe e classe String

- In Java una stringa è una sequenza finita di caratteri racchiusa tra virgolette, come "buongiorno". (le virgolette non fanno parte della stringa)

- Per dichiarare una variabile stringa:

```
String nome;                    // stringa con valore nullo
String nome = "Luigi";
```

- Il secondo caso equivale alla creazione e inizializzazione dell'oggetto **nome**:

```
nome = new String("Luigi");
```

- In Java, la creazione degli oggetti istanza avviene in genere mediante l'uso dell'operatore **new**, l'unica eccezione è data dai letterali **String** che possono essere creati come sopra.

Classe String: metodo length()

- Una stringa vuota è un oggetto `String` che rappresenta una **sequenza vuota di caratteri**, cioè una stringa di lunghezza zero. La stringa vuota è denotata dal letterale "".

- Il metodo `int length()` della classe `String` calcola la lunghezza della stringa, cioè il numero di caratteri che compongono la stringa.

- Ad esempio,
  - `nome.length()` vale 5
  - `"buongiorno".length()` vale 10
  - `"".length()` vale 0

- Gli elementi della stringa vanno dalle posizione 0 a length()-1.

Classe String: metodo charAt()

- Il metodo char `charAt(int pos)` della classe `String` permette di estrarre da una stringa il carattere che occupa una certa posizione.
- Ad esempio
  - `nome.charAt(0)` vale 'L'
  - `"buongiorno".charAt(3)` vale 'n'
- Il metodo `charAt(int pos)` restituisce un carattere e non una stringa (si noti il singolo apice).
- La posizione fornita deve essere compresa tra 0 e length()-1 altrimenti si genera un errore.

Esempio 1

- Stampare i caratteri di una stringa che si trovano in posizione pari e siano diversi dallo spazio (` `).

```
class stampaCarPosPari
{
  public static void main(String args[])
  {
    String saluto="ciao a tutti!!";

    for(int i=0; i<saluto.length(); i++)
    {
      if(saluto.charAt(i) != ' ' && i%2==0)
           System.out.print(saluto.charAt(i));
     }
  }
}
```

- Risultato: **caut!**

Esempio 2

- Calcolare il numero di occorrenze di un carattere c in una stringa s.

```
public static int occorrenze(String s, char c)
{
  int occor;  // numero di occorrenze di c in s
  int i;
  int lung;   // lunghezza di s

 lung = s.length();
 occor = 0;
 for (i=0; i<lung ; i++)
   if (s.charAt(i) == c)
     occor++;
  return occor;
}
```

- Se s="canzone" e c='n', il metodo ritorna 2.

Esempio 3

- Calcolare la stringa inversa di una data stringa s

```
public static String reverse(String s)
{
  String inv;   // la stringa inversa di s
  int i;

  inv = "";
  for (i=s.length()-1; i>=0; i--)
   inv = inv + s.charAt(i);
  return inv;
}
```

- Se s="asor" il metodo ritorna "rosa" .

Classe String: metodo concat()

- La **concatenazione** di stringhe, a partire da due stringhe, permette di ottenere una stringa il cui valore è dato dalla sequenza di caratteri della prima stringa seguita dalla sequenza di caratteri della seconda stringa.
- Il metodo `String concat(String x)` della classe `String` implementa la concatenazione di stringhe.
- `concat` crea e restituisce un nuovo oggetto `String` composto dai caratteri della stringa su cui il metodo viene invocato seguiti dai caratteri della stringa argomento `x`.
- Ad esempio

```
String s1, s2, s3;
s1 = "barba";
s2 = "gianni";
s3 = s1.concat(s2);   /* la stringa s3 varrà "barbagianni" */
```

Classe String: operatore +

- La **concatenazione** di stringhe si può effettuare anche tramite l'operatore + (inteso come concatenazione).

- Ad esempio

```
String s1, s2, s3;
s1 = "barba";
s2 = "gianni";
s3 = s1 + s2;   /* la stringa s3 varrà "barbagianni" */
System.out.println (s1 + s2); /* stampa "barbagianni" */
System.out.println ("la "+s1+" di " + s2); /* stampa "la barba di
    gianni" */
```

- L'operatore + è sovraccarico perchè opera su numeri e stringhe. Questo è un esempio di overloading predefinito.

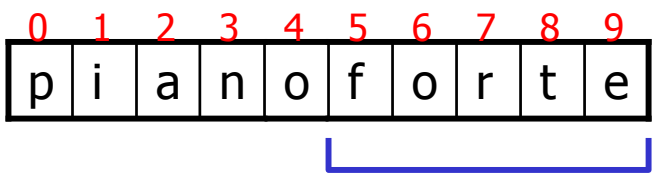
Classe String: sottostringhe

- Il metodo `String substring(int inizio)` della classe `String` crea e restituisce un nuovo oggetto `String` che consiste dei caratteri della stringa su cui il metodo viene invocato.

- I caratteri sono quelli compresi tra quello di posizione inizio e l'ultimo carattere della stringa (incluso). Ad esempio

```
String s1, s2;
s1 = "pianoforte";
s2 = s1.substring(5);   /* s2 vale "forte" */
```

- Si osservi che la posizione 5 è occupata dal carattere 'f' che è il primo carattere della stringa restituita.

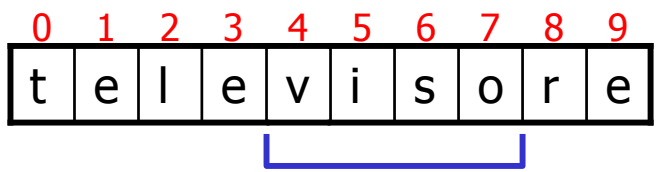


Classe String: sottostringhe

- Il metodo `String substring(int inizio, int fine)` della classe `String` crea e restituisce un nuovo oggetto `String` composto dai caratteri della stringa su cui il metodo è stato invocato che occupano le posizioni tra inizio (incluso) e fine (esclusa).
- La stringa restituita comprende i caratteri tra le posizioni inizio e fine–1. Ad esempio

```
String s1, s2;
s1 = "televisore";
s2 = s1.substring(4,8);    /* s2 vale "viso" */
```

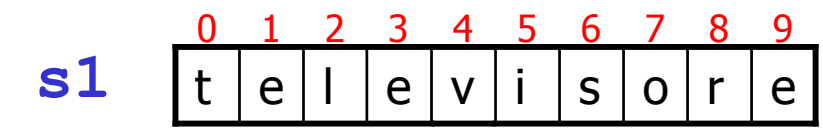
- Si osservi che
  - la posizione 4 è occupata dal carattere 'v' che è il primo carattere della stringa restituita.
  - la posizione 8 è occupata dal carattere 'r' che è il primo carattere escluso dalla stringa restituita.



Classe String: indexOf

- Il metodo `int indexOf(char c)` della classe `String` verifica se la stringa su cui il metodo viene invocato contiene il carattere `c`.

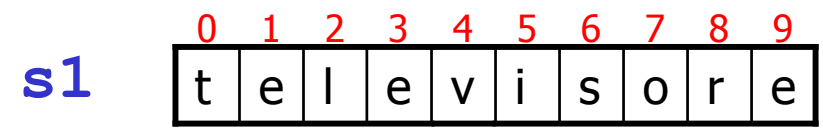
- Se il carattere c'è restituisce la prima posizione in cui occorre il carattere oppure restituisce il valore –1 che indica una posizionenon ammessa. Ad esempio
  - `"JAVA".indexOf('V')` vale 2
  - `s1.indexOf('e')` vale 1
  - `s1.indexOf('u')` vale –1



Classe String: indexOf

- Il metodo `int indexOf(String s)` della classe `String` verifica se la stringa su cui il metodo viene invocato contiene il carattere la stringa `s`.

- Se la stringa c'è restituisce la prima posizione da cui occorre oppure restituisce il valore –1 che indica una posizione non ammessa. Ad esempio
  - `s1.indexOf("tele")` vale 0
  - `s1.indexOf("levis")` vale 2
  - `s1.indexOf("lava")` vale –1
  - `s1.indexOf("ore")` vale 7

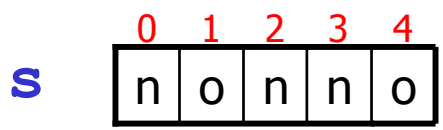


Classe String: lastIndexOf

- Il metodo `int lastIndexOf(String s)` della classe `String` verifica se la stringa su cui il metodo viene invocato contiene la stringa `s`.

- Se la stringa c'è restituisce l'ultima posizione da cui occorre oppure restituisce il valore –1.
- Ad esempio
  - `s.lastIndexOf("no")` vale 3

- Esiste anche il metodo `lastIndexOf(char c)`. Ad esempio
  - `s.lastIndexOf('o')` vale 4



Classe String: esempio di uso di indexOf

- Estrarre da una stringa una sottostringa delimitata da due caratteri.

```
public static String estraiStringa(String da, char i, char f)
{
  int posin  = da.indexOf(i);
  int posfin = da.lastIndexOf(f);

  if(posin == -1)
     return null;
  else
    if(posfin == -1)              // se non si trova la fine
     return da.substring(posin);// restituisce quello che c'è
   else
     return da.substring(posin, posfin+1);
  }
```

Classe String: valueOf

- La conversione da tipi primitivi a stringhe è realizzata mediante un certo numero di metodi di classe della classe **`String`**.
- Ciascuno di questi metodi **`valueOf`** accetta come argomento il valore di un certo tipo primitivo e restituisce un valore **`String`** che è la rappresentazione dell'argomento sotto forma di stringa.
- Questi metodi si usano invocandoli così: **`String.valueOf(x)`**. Ad esempio: **`s=String.valueOf(23)`**. x avrà valore "23".
- Esistono anche dei metodi per effettuare l'operazione inversa: da stringhe a tipi primitivi (int, char, float, ...).

Classe String: equals

- Per confrontare due stringhe non è corretto usare l'operatore di uguaglianza (==) ma occorre usare il metodo della classe `String`

```
boolean equals(String s)
```

- Esempi:

```
String a, b, c;
a = "internet";
b = "inter" + "net";
c = "inter".concat("net");

System.out.println(a.equals(b)); // stampa true
if(a.equals(c))
    System.out.println("Le due stringhe sono uguali");
System.out.println(b.equals(c)); // stampa true
```

Classe String: esercizi

- Possibili metodi da implementare usando i metodi della classe String:
  - Estrazione del carattere o dei due caratteri al centro di una stringa.
  - Ricerca di un carattere in una stringa con e senza l'uso del metodo substring.
  - Realizzare un metodo che ci dice se una stringa è palindroma.
  - Realizzare un metodo che restituisce tutte le sottostringhe di lunghezza n da una data stringa.
  - Realizzare un metodo che restituisce tutti gli anagrammi di una data stringa.