Stringhe

Il tipo di dati "stringa"

- I tipi di dati più importanti nella maggior parte dei programmi sono i ***numeri*** e le ***stringhe***
- Una ***stringa*** è una ***sequenza di caratteri***, che in Java (come in molti altri linguaggi) vanno ***racchiusi tra virgolette***
  - ***le virgolette non fanno parte della stringa***

```
"Hello"
```

- Possiamo ***dichiarare*** e inizializzare ***variabili di tipo stringa***

```
String name = "John";
```

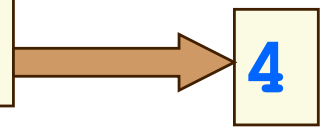
- Possiamo ***assegnare un valore*** ad una variabile di tipo stringa

```
name = "Michael";
```

Il tipo di dati "stringa"

- Diversamente dai numeri, ***le stringhe sono oggetti***
- Una ***variabile*** di tipo stringa può quindi essere utilizzata per ***invocare metodi (non statici)*** della classe **String**
  - ad esempio, il metodo **length** restituisce la ***lunghezza*** di una stringa, cioè ***il numero di caratteri*** presenti in essa (senza contare le virgolette)

```
String name = "John";
int n = name.length();
```



Il tipo di dati "stringa"

- Il metodo **length** della classe **String** ***non è un metodo statico***
  - infatti ***per invocarlo usiamo un oggetto della classe String*** e questo significa che il metodo funziona ***agendo su un oggetto***

```
// NON FUNZIONA!
String s = "John";
int n = String.length(s);
```

```
// FUNZIONA
String s = "John";
int n = s.length();
```

- Una ***stringa di lunghezza zero***, che non contiene caratteri, si chiama ***stringa vuota*** e si indica con due caratteri virgolette ***consecutivi***, senza spazi interposti

```
String empty = "";
System.out.println(empty.length());
```

→ 0

Estrazioni di sottostringhe

- Per estrarre una sottostringa da una stringa si usa il metodo **substring**

Attenzione alla minuscola!

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 4);
// sub contiene "Hell"
```

  - il ***primo*** parametro di **substring** è la ***posizione del primo carattere*** che si vuole estrarre
  - il ***secondo*** parametro è la ***posizione successiva all'ultimo carattere*** che si vuole estrarre

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

Estrazioni di sottostringhe

- La ***posizione*** dei caratteri nelle stringhe viene ***numerata a partire da 0*** anziché da 1
  - Uniformità con altri linguaggi

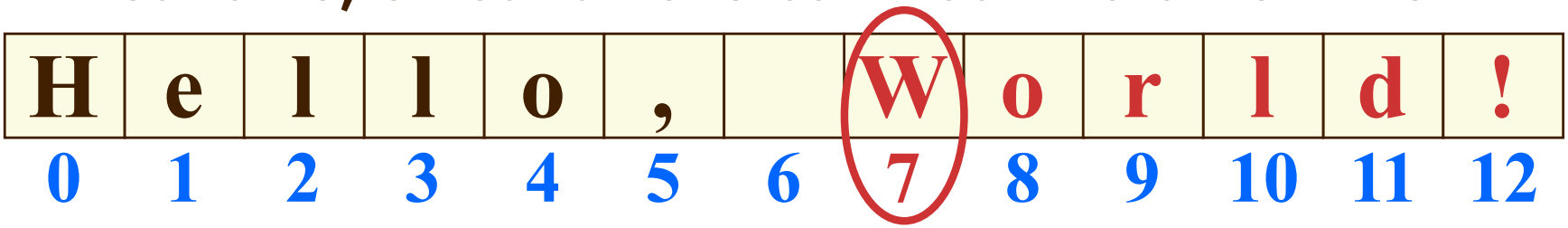
- Alcune cose da ricordare
  - la posizione dell'ultimo carattere corrisponde alla lunghezza della stringa meno 1
  - la differenza tra i due parametri di **substring** corrisponde alla lunghezza della sottostringa estratta

Estrazioni di sottostringhe

- Il metodo **substring** può essere anche invocato con ***un solo*** parametro

```
String greeting = "Hello, World!";
String sub = greeting.substring(7);
// sub contiene "World!"
```

- In questo caso il parametro fornito indica la posizione del primo carattere che si vuole estrarre, e l'estrazione continua fino al termine



Estrazioni di sottostringhe

- Cosa succede se si fornisce un ***parametro errato*** a **substring**?

```
// NON FUNZIONA!
String greeting = "Hello, World!";
String sub = greeting.substring(0, 14);
```

- Il programma viene compilato correttamente, ma viene generato un errore in esecuzione

```
Exception in thread "main"
java.lang.StringIndexOutOfBoundsException
String index out of range: 14
```

Concatenzione di stringhe

- Per concatenare due stringhe si usa l'***operatore*** **+**

```
String s1 = "li";
String s2 = "re";
String s3 = s1 + s2; // s3 contiene lire
int lit = 15000;
String s = lit + s3; // s contiene "15000lire"
```

- L'operatore di concatenazione è identico all'operatore di addizione
  - se una delle espressioni a sinistra o a destra dell'operatore **+** è una stringa, l'altra espressione viene ***convertita*** in stringa e si effettua la concatenazione

Concatenzione di stringhe

```
int lit = 15000;
String litName = "lire";
String s = lit + litName;
// s contiene "15000lire"
```

- Osserviamo che la concatenazione prodotta non è proprio quella che avremmo voluto, perché ***manca uno spazio*** tra **15000** e **lire**
  - l'operatore di concatenazione ***non aggiunge spazi!***

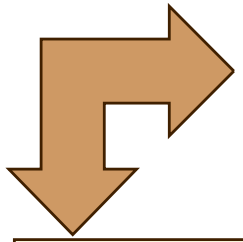
- L'effetto voluto si ottiene così

```
String s = lit + " " + litName;
```

Non è una stringa vuota, ma una stringa con un solo carattere, uno spazio (*blank*)

Concatenzione di stringhe

- La concatenazione è molto utile per ridurre il numero di enunciati usati per stampare i risultati dei programmi



```
int total = 10;
System.out.print("Il totale è ");
System.out.println(total);
```

```
int total = 10;
System.out.println("Il totale è " + total);
```

- Bisogna fare attenzione a come viene gestito il concetto di "***andare a capo***" (cioè alla differenza tra **print** e **println**)

Alcuni metodi utili di String

- Un problema che capita spesso di affrontare è quello della conversione di una stringa per ottenerne un'altra tutta in maiuscolo o tutta in minuscolo

- La classe **String** mette a disposizione due metodi
  - **toUpperCase** converte tutto in maiuscolo
  - **toLowerCase** converte tutto in minuscolo

```
String s = "Hello";
String ss = s.toUpperCase() + s.toLowerCase();
// ss vale "HELLOhello"
```

Alcuni metodi utili di String

```
String s = "Hello";
String ss = s.toUpperCase() + s.toLowerCase();
// s vale ancora "Hello" !
```

- Si noti che l'applicazione di uno di questi metodi alla stringa `s` ***non altera il contenuto*** della stringa `s`, ma ***restituisce una nuova stringa***
- In particolare, ***nessun metodo della classe* String *modifica l'oggetto con cui viene invocato!***
  - si dice perciò che gli oggetti della classe **String** sono ***oggetti immutabili***

Conversione di numeri in stringhe

- Per ***convertire*** un numero in stringa si può ***concatenare il numero con la stringa vuota***

```
int ageNumber = 10;
String ageString = "" + ageNumber;
// ageString contiene "10"
```

- È però più elegante (e più comprensibile) utilizzare il metodo **toString** delle classi **Integer** e **Double**, rispettivamente per numeri interi e numeri in virgola mobile

```
int ageNumber = 10;
String ageString = Integer.toString(ageNumber);
```

Caratteri in una stringa

- Abbiamo visto **substring**
  - A volte è necessario estrarre ed elaborare ***sottostringhe*** di dimensioni minime, cioè ***di lunghezza unitaria***
- Una stringa di lunghezza unitaria contiene ***un solo carattere***
  - ***può essere memorizzato in una variabile di tipo*** **char** anziché in una stringa
- il tipo **char** in Java è ***un tipo di dato fondamentale*** come i tipi di dati numerici e il tipo **boolean**, cioè ***non è una classe***

Caratteri in una stringa

- La presenza del tipo di dati **char** non è strettamente necessaria in Java
  - **ogni** elaborazione che può essere fatta su variabili di tipo **char** potrebbe essere fatta su stringhe di lunghezza unitaria

- L'uso del tipo **char** per memorizzare stringhe di lunghezza unitaria è però importante, perché
  - una variabile di tipo **char** occupa **meno spazio** in memoria di una stringa di lunghezza unitaria
  - le ***elaborazioni*** su variabili di tipo **char** sono ***più veloci***

Caratteri in una stringa

- Il metodo **charAt** della classe **String** restituisce il singolo carattere che si trova nella posizione indicata dal parametro ricevuto

```
String s = "John";
char ch = s.charAt(2); // ch contiene 'h'
```

- la convenzione sulla numerazione delle posizioni in una stringa è la stessa usata dal metodo **substring**

Caratteri in una stringa

- Come si può elaborare un variabile di tipo **char**?
  - la si può ***stampare*** passandola a **System.out.print()**
  - la si può ***concatenare a una stringa*** con l'operatore di concatenazione + (verrà convertita in stringa con le stesse regole della conversione dei tipi numerici)
- Una variabile di tipo **char** può anche essere inizializzata con una ***costante di tipo carattere***
  - una costante di tipo carattere è ***un singolo carattere*** racchiuso tra ***singoli*** apici ("apostrofo")
    - char c = 'x';
  - Il singolo carattere può anche essere una "sequenza di escape" : char c = '\n';

Sequenze di "escape"

Sequenze di "escape"

- Proviamo a stampare una stringa che ***contenga*** delle virgolette `Hello, "World"!`

```
// NON FUNZIONA!
System.out.println("Hello, "World"!");
```

- Il compilatore identifica le seconde virgolette come la fine della prima stringa **"Hello, "**, ma poi non capisce il significato della parola **World**
- Basta inserire una barra rovesciata \ (*backslash*) ***prima*** delle virgolette ***all'interno*** della stringa

```
System.out.println("Hello, \"World\"!");
```

Sequenze di "escape"

```
// FUNZIONA!
System.out.println("Hello, \"World\"!");
```

- Il carattere *backslash* all'interno di una stringa non rappresenta se stesso, ma si usa per codificare altri caratteri che sarebbe ***difficile*** inserire in una stringa, per vari motivi (***sequenza di escape***)

- Allora, come si fa ad inserire veramente un carattere *backslash* in una stringa?
  - si usa la sequenza di escape \\

```
System.out.println("File C:\\autoexec.bat");
```

```
File C:\autoexec.bat
```

Sequenze di "escape"

- Un'altra sequenza di escape che si usa è **\n**, che rappresenta il carattere di "nuova riga" o "andare a capo"

```
System.out.println("*\n**\n***");
```

```
System.out.println("*");
System.out.println("**");
System.out.println("***");
```

```
*
**
***
```

- Le sequenze di escape si usano anche per inserire caratteri di lingue straniere o simboli che non si trovano sulla tastiera

- Ad esempio, per scrivere parole italiane con lettere accentate senza avere a disposizione una tastiera italiana

```
System.out.println("Perch\u00E9?");
```

- Queste sequenze di escape utilizzano la codifica standard **Unicode**
  - **http://www.unicode.org**
- per rappresentare i caratteri di tutti gli alfabeti del mondo con ***4 caratteri esadecimali*** (***codifica a 16 bit***, 65536 simboli diversi)

```
Perché?
```

Ricevere dati in ingresso

I dati in ingresso ai programmi

- I programmi visti finora non sono molto utili, visto che eseguono **sempre la stessa elaborazione ad ogni esecuzione.**

- I programmi utili hanno bisogno di **ricevere dati in ingresso** dall'utente

L'input standard dei programmi

- I dati in ingresso possono essere inseriti da tastiera, mouse, microfono, etc.
- Tutti i programmi Java hanno accesso al proprio **output standard** tramite
  - **System.out** di tipo **PrintStream definito nella classe System**
- L'interprete Java mette a disposizione dei programmi in esecuzione il proprio **input standard** (flusso di input), tramite
  - **System.in** di tipo **InputStream definito nella classe System**

La classe Scanner

- La classe **Scanner** appartiene al pacchetto java.util della libreria standard (dalla versione 5.0)
  - La classe **Scanner** appartiene al pacchetto **java.util** che andra' importato
  - questa classe verrà analizzata in maggiore dettaglio in seguito, per ora impareremo soltanto ad **utilizzarla**

- Lo scopo della classe Scanner è quello di fornire una **comoda interfaccia all'oggetto System.in**

Leggere l'input con la classe Scanner

La classe Scanner

- Prima di tutto bisogna creare un ***oggetto*** della classe **Scanner** usando l'istruzione

```
Scanner console = new Scanner(System.in);
```

- E' possibile creare un oggetto di questo tipo a partire da un qualsiasi **flusso di input**
- Dopo aver creato uno scanner è possibile usarne i metodi di lettura
- Durante ogni esecuzione dei metodi di lettura di **Scanner**, ***il programma si ferma ed attende*** l'introduzione dell'input da tastiera, che viene considerata conclusa quando l'utente batte il tasto **Invio**

```
Scanner console = new Scanner(System.in);
```

I metodi nextInt() e nextDouble()

- Il metodo **nextInt** restituisce un valore numerico di tipo **int**
- E' consigliabile inserire un messaggio di richiesta dati prima di invocare un metodo di Scanner

```
System.out.print("Inserisci il dato: ");
int number = console.nextInt();
```

- Il metodo **nextDouble** restituisce un valore numerico in virgola mobile

```
System.out.print("Inserisci il dato: ");
double fp = console.nextDouble();
```

Attenzione per nextDouble()

- I progettisti della classe **Scanner** hanno usato la ***localizzazione***
  - Significa che il comportamento del programma è legato alla configurazione del sistema su cui viene fatto eseguire
- A seconda della configurazione (**anglosassone** o ***italiana***) un valore di tipo double dovrà essere inserito nella forma
  - **parteIntera.parteDecimale** ad esempio 4.35

    oppure
  - **parteIntera,parteDecimale** ad esempio 4,35

Esempio

```
//… qui manca qualcosa

public class Coins4
{  public static void main(String[] args)
   {  Scanner c = new Scanner(System.in);
      System.out.println("Quante lire?");
      int lit = c.nextInt();
      System.out.println("Quanti euro?");
      double euro = c.nextDouble();
      System.out.print("Valore totale in euro ");
      System.out.println(euro + lit / 1936.27);
   }
}
```

```
Scanner console = new Scanner(System.in);
```

I metodi nextLine() e next() di Scanner

- Il metodo **nextLine** restituisce un oggetto di tipo **String** che contiene **l'intera riga** introdotta dall'utente (fino alla pressione ***del carattere* Invio** = carattere di "fine riga" o "andata a capo")

```
String line = console.nextLine();
```

- Il metodo **next** restituisce un oggetto di tipo **String** che contiene invece **la parola successiva** cioè una sequenza di caratteri terminata da uno "spazio", o da un carattere di tabulazione , o da un carattere di "fine riga"

```
String word = console.next();
```

Esempio

- Scriviamo un programma che genera la password per un utente (da non usare mai!), con la regola seguente
  - si prendono le iniziali dell'utente, le si rendono minuscole e si concatena l'età dell'utente espressa numericamente

```
Utente: Mario Rossi
Età: 19
⇒ Password: mr19
```

Esempio

```
import java.util.Scanner;
public class MakePassword2
{  public static void main(String[] args)
   {  Scanner c = new Scanner(System.in);
      System.out.println("Inserire il nome");
      String firstName = c.nextLine();
      System.out.println("Inserire il cognome");
      String lastName = c.nextLine();
      System.out.println("Inserire l'età");
      int age = c.nextInt();
      String initials = firstName.substring(0, 1)
         + lastName.substring(0, 1);
      String pw = initials.toLowerCase() + age;
      System.out.println("La password è " + pw);
   }
}
```

Take home message

- Per gestire le sequenze di caratteri esiste la classe String
  - E' nel pacchetto java.lang, non serve importarlo
  - Consultare la documentazione java per i metodi
    - Es: length(), substring(i,j), etc
- Per acquisire i dati da standard input c'è la classe Scanner
  - E' nel paccheto java.util, serve importarlo!
    - Scanner console = new Scanner(System.in);
  - Consultare la documentazione java per i metodi
    - Es. next(), nextLine(), nextInt(), nextDouble(), etc