

# Stringhe

Dal materiale del prof. Stefano Ferretti

# String

- Introduciamo la *classe* **String**
  - Senza approfondire troppo!
- Una stringa di **lunghezza**  $k$  è una lista di  $k \geq 0$  caratteri
  - La **stringa vuota** ha lunghezza 0
- In Java ogni carattere è memorizzato internamente come un 16-bit **Unicode** character (16-bit integer)
- Ogni carattere della stringa ha un **indice** associato, corrispondente alla sua **posizione** nella stringa
  - Gli indici partono da 0

# String

- **String**: sequenza di 0 o più caratteri delimitati da **doppi apici**

- **Ex: "Hello"**

- **Stringa Vuota: ""**

I doppi apici **non** fanno parte della stringa

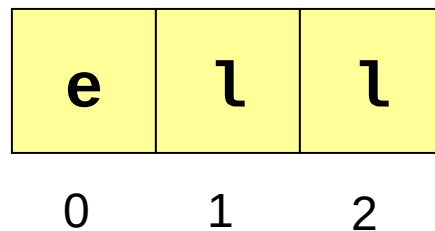
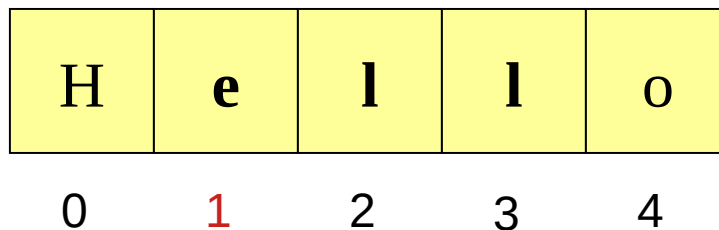
- **Memory model:**

Gli indici vanno da **0** a **lunghezza-1**

H	e	l	l	o
0	1	2	3	4

# Substring

- Una **sottosstringa** (substring) è una sotto-sequenza di caratteri contigui di una stringa



Una substring è una stringa

# String operations

- Le stringhe possono essere **concatenate**
- Valori numerici possono essere convertiti *implicitamente* in stringhe
- Il segno **+** è usato per denotare **addizione** ma anche **concatenazione**
- **REGOLA:** In **a + b** se *almeno* una variabile tra **a** o **b** è una stringa, allora l'altra viene convertita in stringa, se necessario

# Esempio

```
String first = "William";  
String middle = "James";  
String last = "Duncan";  
String fullName =  
    first + " " + middle + " " +  
    last;
```

fullname diventa "William James Duncan"

Quante stringhe vedete in questo programma?

# Concatenazione

**"The sum of " + x + " and " +  
y + " is " + x + y**

Se x è 3 e y è 4 il risultato dell'espressione è  
**"The sum of 3 and 4 is 34"**

*Davvero voglio  
questo?*

# Concatenazione

**"The sum of " + x + " and " +  
y + " is " + (x + y)**

Se x è 3 e y è 4 il risultato dell'espressione è

**"The sum of 3 and 4 is 7"**

perchè *prima* viene effettuata la somma tra interi e  
poi il risultato viene convertito in stringa



# Number to string

- Esempi di **conversione** numero → stringa

```
int age = 34;  
float area = 2.34;  
String s1 = "" + age;  
String s2 = "" + area;
```

Se age è 34 e area è 2.34  
allora s1 e s2 diventeranno  
"34" e "2.34"

# Number to string

- Attenzione: questo è **sbagliato**

```
int age = 34;  
float area = 2.34;  
String s1 = age;  
String s2 = (String) area;
```


error: incompatible types: int  
cannot be converted to String

# Number to string

```
String area = "Area: " + Math.sqrt(2.0);
```

Il valore di area è  
**"Area: 1.4142135623730951"**

Cos'è questo?



# Invocare un metodo (cenno)

- **Math.sqrt(2.0)** corrisponde alla *invocazione* di un metodo della classe **Math**
- **Invocare un metodo** significa “*richiamare una particolare operazione di una certa classe*”
- Anche senza saperlo, l'abbiamo già fatto:  
**System.out.println()**
- La classe **String** offre diversi metodi
  - Cioè, diverse operazioni su stringhe

# Length

- Prototype:
  - **int length()**
    - Ritorna la lunghezza di una stringa
- Example:

```
String name = "Harry";  
int len = name.length();
```

Il valore di len  
sarà 5

# CharAt

- Prototype:

- `char charAt(int i)`

1 parametro

- Ritorna il *carattere* in posizione **i**

- Example:

```
String s = "Hello";  
char c = s.charAt(1);
```

Il valore di c diventa 'e'  
(gli indici partono da 0)

# CharAt

- Prototype:

- `char charAt(int i)`

1 parametro

- Ritorna il *carattere* in posizione **i**

- Example:

```
String s = "Hello";  
char c = s.charAt(1);
```

**Attenzione!!!** charAt **ritorna** un carattere della stringa, ma **non** può essere utilizzato per modificare un carattere della stringa

Il valore di c diventa 'e'  
(gli indici partono da 0)

# Substring

## ■ Prototypes

### □ `String substring(int i)`

- Ritorna la sottostringa dall'indice `i` fino a fine stringa

### □ `String substring(int i, int j)`

- Ritorna la sottostringa tra l'indice `i` e l'indice **`j-1`**

```
int d = 531452;  
String sd = "" + d;  
int len = sd.length();  
sd = "$" + sd.substring(0, len-3)  
      + ", " + sd.substring(len-3);
```

**Method Overloading:** questi metodi hanno lo stesso nome (`substring`) ma diversa *signature* (diversi parametri)

Risultato?



# Substring

## ■ Prototypes

### □ `String substring(int i)`

- Ritorna la sottostringa dall'indice `i` fino a fine stringa

### □ `String substring(int i, int j)`

- Ritorna la sottostringa tra l'indice `i` e l'indice `j-1`

```
int d = 531452;  
String sd = "" + d;  
int len = sd.length();  
sd = "$" + sd.substring(0, len-3)  
      + ", " + sd.substring(len-3);
```

**Method Overloading:** questi metodi hanno lo stesso nome (`substring`) ma diversa *signature* (diversi parametri)

Risultato:  
"\$531,452"

# Trim

- Prototype

- `String trim()`

- Rimuove whitespaces all'inizio e alla fine di una stringa

```
String s = "    Hello world    ";  
s = s.trim();
```

Risultato:  
"Hello world"

# Case conversion

- upper/lower case conversion
- Prototypes
  - `String toLowerCase()` → converte in minuscolo
  - `String toUpperCase()` → converte in maiuscolo

```
String test = "Hello";  
String upper = test.toUpperCase();  
String lower = test.toLowerCase();  
char first = test.toLowerCase().charAt(0);
```

Invocazione  
"in cascata"

Risultato: upper: "HELLO",  
lower: "hello", first: "h"

# Search

- Cercare una stringa all'interno di un'altra

- Prototypes

- `int indexOf(char ch)`

Method Overloading

- Ritorna l'indice della *prima occorrenza* del *carattere ch* a partire *dall'inizio* della stringa (cioè dall'indice 0)

- `int indexOf(char ch, int i)`

- Ritorna l'indice della *prima occorrenza* del carattere *ch* a partire *dall'indice i*

- `int indexOf(String sub)`

- `int indexOf(String sub, int startIndex)`

- Come sopra, ma viene ricercata una *stringa* anzichè un carattere

# String search examples

```
String target = "This is the target string";
```

```
target.indexOf('u')
```

ritorna -1 perché il  
carattere u non occorre  
nella stringa target

```
target.indexOf("the")
```

ritorna 8

```
target.indexOf("target", 13)
```

Ritorna -1...  
perché?

# Print/println

- Anche i metodi di stampa **print** e **println** hanno diverse “versioni”:

- ❑ `void print(int n)`
- ❑ `void print(double d)`
- ❑ `void print(String s)`

print but *don't*  
move to next line

- ❑ `void println(int n)`
- ❑ `void println(double d)`
- ❑ `void println(String s)`
- ❑ `void println()`

print *and* move  
to next line

# Print/println

Questo codice:

```
double area = 3.14159;  
System.out.println("Area: " + area);
```

E questo:

```
double area = 3.14159;  
System.out.print("Area: ");  
System.out.println(area);
```

Sono equivalenti:

```
Area: 3.14159
```

# Print/println

Anche questo codice:

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

E questo:

```
System.out.print("Hello\n");
```

*\* è detto “*escape character*”  
**\n** rappresenta il “**newline character**”

Sono equivalenti:

Producono lo stesso risultato perchè **\n** è **un** carattere speciale (non due caratteri!) che viene interpretato come “a capo”



# Escape sequences

- Java usa le **escape sequences** per stampare alcuni caratteri speciali, es:
  - `\n`      newline
  - `\t`      tab
  - `\\`      backslash (per stampare `\`)
  - `\"`      double quote (per stampare `"`)
  - `\'`      single quote (per stampare `'`)
  - ...

# Escape sequences

```
System.out.println("Person\tHeight\tShoe size");  
System.out.println("=====");  
System.out.println("Hannah\t5'1\"\t7");  
System.out.println("Jenna\t5'10\"\t9");  
System.out.println("JJ\t6'1\"\t14");
```

**Provate!**

Person	Height	Shoe size
=====		
Hannah	5'1"	7
Jenna	5'10"	9
JJ	6'1"	14

# Metodi

- Se invocate un metodo, dovete essere certi che sia **definito** nella classe corrispondente

```
System.out.length(); // Questa invocazione di  
                     // metodo è errata
```

# Altre classi

- Vediamo altre 2 classi che useremo spesso
  - Senza approfondire troppo!
- Ci servono per:
  - Operazioni matematiche
  - Input

# Math

- La classe **Math** mette a disposizione diverse funzioni matematiche, es.

**Math.sqrt(x)**

radice quadrata di **x**

**Math.round(x)**

arrotonda **x** al **long** + vicino

**Math.pow(x, y)**

calcola **x** elevato ad **y**

**Math.sin(x)**

seno di **x** in *radianti*

# Round

- Per arrotondare un valore **double x** all'intero più vicino utilizziamo **Math.round**
- Questo metodo ritorna **long** (64-bit integer), se vogliamo un **int** serve un cast:

```
int i = (int)Math.round(x);
```

# Round

- Supponiamo che **amount** sia un valore **double** che rappresenta una somma di denaro
  - Es. **3.45** corrisponde a **3** euro e **45** centesimi
- Convertiamo in numero totale di centesimi:

```
int totalCents =  
    (int) Math.round(100 * amount);
```

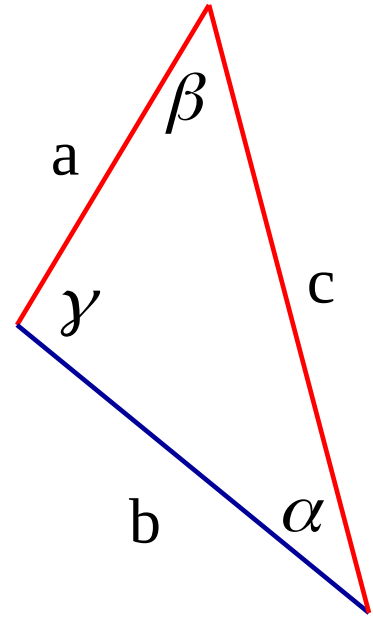
# Square root

$$c = \sqrt{a^2 + b^2 - 2ab \cos \gamma}$$

$$\textit{perimeter} = a + b + c$$

$$s = \textit{perimeter} / 2$$

$$\textit{area} = \sqrt{s(s - a)(s - b)(s - c)}$$



```
double c = Math.sqrt(a*a + b*b - 2.0*a*b*Math.cos(gamma));  
double perimeter = a + b + c;  
double s = perimeter / 2.0;  
double area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
```



# Example

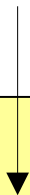
**r** interest rate in percent per year

**m** times/year interest is compounded

**a** initial investment,

**n** years to invest

**v** value of investment

$$v = a \left( 1 + \frac{r}{100m} \right)^{mn}$$


```
double v =  
    a * Math.pow(1.0 + r/(100.0*m), m*n);
```

# Rounding to 2 decimal places

- Assumiamo che **x** sia di tipo double
- La variabile **x2** definita come

```
double x2 =  
    Math.round(x * 100.0) / 100.0;
```

ha il valore di **x** arrotondato a **2** valori decimali

- **Provate!**
  - E se volessi arrotondare a **N** valori decimali?

# Math

## ■ Altre operazioni:

<code>public static double sqrt(double x)</code>	$\sqrt{x}$
<code>public static double sin(double x)</code>	$\sin x$
<code>public static double cos(double x)</code>	$\cos x$
<code>public static double tan(double x)</code>	$\tan x$
<code>public static double exp(double x)</code>	$e^x$
<code>public static double log(double x)</code>	$\ln x$
<code>public static double random()</code>	
<code>public static long round(double x)</code>	
<code>public static double pow(double x, double y)</code>	$x^y$

# Random

- **Math.random()** restituisce un numero double (pseudo-)casuale nell'intervallo **[0.0, 1.0)**
- E.g. **x = Math.random()** assegna ad x un valore casuale tra **0.0** (incluso) e **1.0** (escluso)
  - Ogni volta che rieseguo l'assegnamento il valore di x sarà (*probabilmente...*) differente
  - Provate a scrivere un programma Java il cui main ha una sola istruzione:  
**System.out.println(Math.random())**  
...ed eseguitelo più volte (senza ricompilarlo!)

# Random

- Per generare un numero casuale **intero**, basta fare il **cast**
- Ad esempio:
  - $(\text{int})(\text{Math.random()} * 10)$  genera un intero casuale tra 0 e 9
  - $(\text{int})(\text{Math.random()} * 5 + 2)$  genera un intero casuale tra 2 e 6
- In generale,  $(\text{int})(a + \text{Math.random()} * b)$  genera un intero casuale tra  $a$  e  $a+b-1$ 
  - Quindi, per generare un intero casuale in  $[a,b]$  faccio  $(\text{int})(a + \text{Math.random()} * (b - a + 1))$

# Random

- Esistono anche appositi metodi della classe **Random**, che per ora non vediamo

`nextBoolean()`

Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.

`nextBytes(byte[] bytes)`

Generates random bytes and places them into a user-supplied byte array.

`nextDouble()`

Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.

`nextFloat()`

Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

`nextGaussian()`

Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.

`nextInt()`

Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.

`nextInt(int bound)`

Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

`nextLong()`

Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

# Input

# Input

- Come inserire **interattivamente** dati durante l'esecuzione
- Così come esiste **System.out.print()** per l'output
- Esiste ~~**System.in.read()**~~ per l'input
- **System.in.read()** non è il massimo:
  - Legge un solo carattere (compresi whitespaces)
  - Ne restituisce il codice ASCII/Unicode



# Scanner

- **Scanner** risolve questa limitazione
- È una classe che permette di leggere input da tastiera di vario tipo
- Un oggetto **Scanner** può essere settato per leggere input da varie sorgenti
  - Per ora, consideriamo solo la tastiera (std input)
- Tastiera → “rappresentata” dalla classe **System.in**

# Scanner

- Per poter usare lo scanner, anzitutto va aggiunta questa stringa a inizio programma:

```
import java.util.Scanner;
```

- Quindi creare un oggetto Scanner che legga da tastiera:

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

- L'operatore **new** crea l'oggetto **Scanner**
- Saremo più precisi più avanti!

# Scanner

- Una volta creato, l'oggetto permette di leggere diverse tipologie di dati in input da tastiera
- Ad es. il metodo **nextLine** legge **tutta** una stringa fino a fine linea (anche gli spazi) finchè non si digita INVIO

**answer = scan.nextLine();**

- **Esercizio:** scrivere un programma **Echo.java** che “faccia l'eco”, cioè legga una stringa da tastiera e poi la stampi a video

# Input Tokens

- *White spaces* utilizzati per separare tokens (=”pezzi di stringa”) letti in input
  - White spaces = spazi bianchi, tab, new line
- Il metodo **next** di **Scanner** legge il token successivo e lo ritorna come una stringa
  - Diverso da `nextLine`, che legge fino a new line
- Metodi come **nextInt** o **nextDouble** leggono dati del **tipo specificato** nel loro nome

# Input Tokens

- boolean **nextBoolean()**
  - Scans the next token of the input into a boolean value and returns that value
- byte **nextByte()**
  - Scans the next token of the input as a byte
- double **nextDouble()**
  - Scans the next token of the input as a double
- float **nextFloat()**
  - Scans the next token of the input as a float

# Input Tokens

- int **nextInt()**
  - Scans the next token of the input as an int
- long **nextLong()**
  - Scans the next token of the input as a long
- short **nextShort()**
  - Scans the next token of the input as a short

# NOTA su Scanner

- Per inserire numeri, si può usare **nextDouble()**
- Se avete settato il vostro PC in italiano, è possibile che dobbiate scrivere il numero con la “,” e non con il “.”
  - Es: 1,4 invece di 1.4
- Scanner considera la locazione dell'utente:
  - Se è US → usa i decimali con il punto
    - 1.4
  - Se è Italiano, Francese, ... → usa i decimali con la virgola
    - 1,4

# NOTA su Scanner

- Per usare il “.” invece del “,” ?
  - Cosa che pare ragionevole ...
- Inserite la seguente istruzione dopo aver creato l'oggetto **scan**  
**scan.useLocale(Locale.US);**
- Invece di **import java.util.Scanner;**  
usate **import java.util.\*;**



# Esempio

**Provate!**

```
import java.util.*;  
...  
Scanner scan = new Scanner (System.in);  
scan.useLocale(Locale.US);  
  
double base, altezza, area;  
  
System.out.print("Inserisci la base: ");  
base = scan.nextDouble();  
altezza = 12.0;  
area = base * altezza / 2;  
System.out.println("area = " + area);
```

# Riferimenti

- Lucidi del Libro di Riferimento
- <http://java.sun.com/>
- <http://www.beanshell.org>
- <http://www.di.unipi.it/~tesei/unicam/ProgrLab20042005.html>
- <http://www.dimi.uniud.it/mizzaro/dida/Prog0405>
- <http://www.di.unipi.it/~susanna/IG02/>
- [http://www.sti.uniurb.it/bernardo/teaching/prog\\_elab/dispense\\_prog\\_elab.pdf](http://www.sti.uniurb.it/bernardo/teaching/prog_elab/dispense_prog_elab.pdf)

# Note sulla Licenza

- Parte di questi lucidi sono ispirati da lucidi sotto licenza **Creative Commons**
  - <http://www.creativecommons.org/>
  - I lucidi in questione sono tratti da <http://www.dimi.uniud.it/~mizzaro/dida/Prog0405/>
    - Per il corso di Programmazione e Laboratorio tenuto dal Prof. Stefano Mizzaro
- Di conseguenza, anche questi lucidi vengono distribuiti sotto licenza **Creative Commons**