

## STRINGHE IN JAVA

- In Java, le stringhe non sono “pezzi di memoria” con dentro dei caratteri, come in C: sono **oggetti appartenenti alla classe String**
- Una stringa Java rappresenta **uno specifico valore** e come tale è **immodificabile**
- se occorre una stringa modificabile esiste la classe `StringBuffer`
- L'operatore `+` denota la concatenazione:

`"ciao" + " mondo\n"`

NB: la concatenazione di costanti è fatta a tempo di compilazione, quindi non introduce inefficienze

Stringhe - 1

## COSTANTI String

- Le costanti `String` possono essere denotate nel modo usuale:

`"ciao"`      `"mondo\n"`

- Quando si scrive una costante `String` tra virgolette, viene creato **implicitamente un nuovo oggetto di classe `String`**, inizializzato a tale valore
- Una costante `String` **non può eccedere la riga**: quindi, dovendo scrivere stringhe più lunghe, conviene **spezzarle e concatenarle con `+`**

Stringhe - 2

## STRINGHE IN JAVA

- Le stringhe Java **non sono array di caratteri**
- Sono oggetti **concettualmente diversi**  
→ **non si applicano gli operatori degli array!**  
– in particolare, non si applica il classico `[]`
- Per **selezionare il carattere *i*-esimo** si usa il metodo `charAt()`:  

```
String s = "Nel mezzo del cammin";  
char ch = s.charAt(4);
```
- La classe `String` definisce **decine di metodi**: si veda la documentazione (Java 2 API)

Stringhe - 3

## STRINGHE IN JAVA

- Tutte le classi Java definiscono un metodo `toString()` che produce una `String` a partire da un oggetto della classe: **ciò consente di “stampare” facilmente qualunque oggetto di qualunque classe**
- È responsabilità del progettista definire un metodo `toString()` che produca una stringa “significativa”
- Quello di default stampa un identificativo alfanumerico dell'oggetto

Stringhe - 4

## ESEMPIO

```
public class Esempio5 {  
    public static void main(String args[]){  
        String s = "Nel mezzo del cammin";  
        char ch = s.charAt(4);  
        System.out.println(ch);  
        System.out.println("Carattere: " + ch);  
        Counter c = new Counter(10);  
        System.out.println(c);  
    }  
}
```

Converte ch in stringa e lo concatena alla frase.

Usa il metodo `toString()` predefinito di `Counter` → Stampa un identificativo dell'oggetto c

Stringhe - 5

## VARIANTE

- La stampa di poco fa non vi piace?
- Potete ridefinire esplicitamente il metodo `toString()` della classe `Counter`, **facendo-gli stampare ciò che preferite**

Ad esempio:

```
public class Counter {  
    ...  
    public String toString(){  
        return "Counter di valore " + val;  
    }  
}
```

Stringhe - 6

## VARIANTE

Lo stesso identico esempio:

```
public class Esempio5 {
    public static void main(String args[]){
        String s = "Nel mezzo del cammin";
        char ch = s.charAt(4);
        System.out.println(ch);
        System.out.println("Carattere: " + ch);
        Counter c = new Counter(10);
        System.out.println(c);
    }
}
```

ora stamperà:

Counter di valore 10

Stringhe - 7

## ARRAY IN JAVA

- Gli array Java sono **oggetti**, istanze di una **classe speciale** denotata da [ ]
- Quindi, prima si definisce un riferimento...

```
int[] v;          int v[];
```

- Attenzione: è un riferimento, non bisogna specificare la dimensione

La parentesi si può mettere in due modi: dopo il nome delle variabile, come in C, oppure dopo il tipo

```
Counter[] w;      Counter w[];
```

Stringhe - 8

## ARRAY IN JAVA

- ...e poi si crea dinamicamente l'oggetto:

```
v = new int[3];
```

```
w = new Counter[8];
```

- La dimensione si specifica all'atto della creazione

Stringhe - 9

## ARRAY IN JAVA

**Attenzione!!** Ogni elemento dell'array:

- è una variabile, se gli elementi dell'array sono di un **tipo primitivo** (int, float, char, ...)

```
v = new int[3];
```



- è un riferimento a un (futuro) oggetto, se gli elementi dell'array sono (riferimenti a) **oggetti**

```
w = new Counter[3];
```



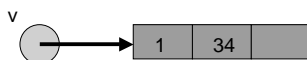
Stringhe - 10

## ARRAY IN JAVA

Quindi nel primo caso, array di valori primitivi in questo caso, ogni elemento dell'array è una normale variabile, "già usabile" così com'è:

```
v = new int[3];
```

```
v[0] = 1; v[1] = 34;
```



Stringhe - 11

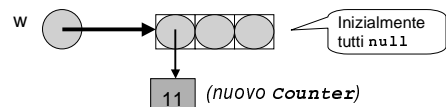
## ARRAY IN JAVA

Se trattiamo riferimenti ad altri oggetti

- nel secondo caso, invece, ogni elemento dell'array è solo un **riferimento**: se si vuole un nuovo oggetto, bisogna crearselo!

```
w = new Counter[3];
```

```
w[0] = new Counter(11);
```



Stringhe - 12

## ARRAY IN JAVA

- In quanto *istanze di una classe "array"*, gli array Java hanno alcune *proprietà*
- Tra queste, il campo-dati pubblico `length` dà la *lunghezza* (dimensione) dell'array:

`v.length` vale 3

Una volta creato, un array ha comunque *dimensione fissa*

- non può essere "allargato" a piacere
- per tali necessità esiste la classe `Vector`

Stringhe - 13

## ESERCIZIO

*Stampare a video gli argomenti passati dalla linea di comando.*

- Il main riceve un *array di String*

```
public static void main(String[] args){  
    ...  
}
```

- Non c'è `argc`, perché la dimensione dell'array è indicata dalla proprietà `length`
- Ogni elemento di `args` è un (riferimento a) `String`

Stringhe - 14

## ESERCIZIO

Quindi:

```
public class EsempioMain{  
    public static void main(String[] args){  
        if (args.length==0)  
            System.out.println("Nessun argomento");  
        else  
            for (int i=0; i<args.length; i++)  
                System.out.println("argomento " + i  
                                   + ": " + args[i]);  
    }  
}
```

L'operatore `+` concatena `String` e anche valori di tipi primitivi (che vengono automaticamente convertiti in `String`)

Stringhe - 15

## ESERCIZIO

Quindi:

```
public class EsempioMain{  
    public static void main(String[] args){  
        if (args.length==0)  
            System.out.println("Nessun argomento");  
        else  
            for (int i=0; i<args.length; i++)  
                System.out.println("argomento " + i  
                                   + ": " + args[i]);  
    }  
}
```

A differenza del C, `args[0]` è il 1° argomento (non il nome del programma!)

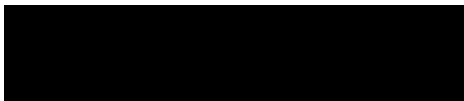
Stringhe - 16

## ESERCIZIO

Esempio d'uso:

`C:> java EsempioMain 34 e 56 "aaa eee"`

Output:



Stringhe - 17

## TIPI PRIMITIVI E CLASSI "WRAPPER"

- In varie situazioni, può essere comodo poter *trattare i tipi primitivi come oggetti*
  - per passarli per riferimento a una funzione
  - per poterli usare quando una funzione richiede obbligatoriamente un "vero oggetto"
- Una *classe "wrapper"* incapsula una *variabile di un tipo primitivo*
  - la classe `Boolean` incapsula un `boolean`
  - la classe `Double` incapsula un `double`
- La classe wrapper ha nome (quasi) identico al tipo primitivo che incapsula, ma con l'iniziale maiuscola

Stringhe - 18

## TIPI PRIMITIVI E CLASSI “WRAPPER”

Tipo primitivo	Classe “wrapper” corrispondente
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

- Ogni classe wrapper definisce metodi per estrarre il valore della variabile incapsulata e viceversa

Stringhe - 19

## CLASSI WRAPPER - ESEMPIO

```
public class Esempio6 {
    public static void main(String args[]){
        int x = 35;
        Integer ix = new Integer(x);
        x = 2*ix.intValue();
        System.out.println("ix =" + ix);
        System.out.println("x =" + x);
    }
}
```

Costruisce un oggetto Integer a partire da un valore int

Estrae da un oggetto Integer il valore int incapsulato al suo interno, mediante intValue()

Stringhe - 20

## ESEMPIO

```
public class Esempio6 {
    public static void main(String args[]){
        int x = 35;
        Integer ix = new Integer(x);
        x = 2 * ix.intValue();
        System.out.println("ix =" + ix);
        System.out.println("x =" + x);
    }
}
```

Conversione implicita da Integer a String (usando il metodo toString() di Integer)

Conversione implicita da int a String (usa la funzione statica toString() di Integer)

Stringhe - 21