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*: una `String` non è un contenitore!**
  - **se occorre un contenitore esiste `StringBuffer`**
- **L'operatore + denota la concatenazione:**

```
"ciao" + " mondo\n"
```

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

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 +.***

ALCUNE RIFLESSIONI (1/2)

Alla luce di quanto detto sulle costanti stringa:

- **quanti oggetti `String` vengono creati per una *stessa costante*?**

```
String s1 = "ciao";
String s2 = "ciao";
```

  - **Che risultato darà il test `s1 == s2` ?**

- **cosa succede se ci sono concatenazioni?**

```
String s3 = "anna" + "maria";
String s4 = "annamaria";
```

  - **Che risultato darà il test `s3 == s4` ?**

ALCUNE RIFLESSIONI (2/2)

E se utilizziamo variabili?

- **quanti oggetti `String` vengono creati ?**

```
String s5 = "anna";
String s6 = "maria";
String s4 = "annamaria";
```

  - **Che risultato darà il test `s5+s6 == s4` ?**
  - **È uguale al caso precedente?**
    Spiegare...

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:** per i dettagli si veda la documentazione

PRINCIPALI METODI per String (1/2)

```
char charAt(int index)
int  length()
int  compareTo(Object o)
int  compareTo(String s)
int  compareToIgnoreCase(String str)
String concat(String str)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String anotherString)
int indexOf(int ch)
int indexOf(int ch, int from)
int indexOf(String s)
int indexOf(String s, int from)
int lastIndexOf(int ch)
int lastIndexOf(int ch, int from)
int lastIndexOf(String s)
int lastIndexOf(String s, int from)
String replace(char oldChar, char newChar)
```

PRINCIPALI METODI per String (2/2)

```
boolean endsWith(String suffix)

boolean startsWith(String prefix)
boolean startsWith(String prefix, int toffset)

String  substring(int beginIndex)
String  substring(int beginIndex, int endIndex)

String  toLowerCase()
String  toUpperCase()

String  trim()

static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(char[] data)
static String valueOf(char[] data, int offset, int count)
static String valueOf(double d)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(Object obj)
```

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 predefinito stampa un identificativo alfanumerico dell'oggetto.

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.

```
Counter@4abc9
```

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;
 }
}
```

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 stampa:

```
Counter di valore 10
```

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

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

```
v = new int[3];
w = new Counter[6];
```

È un po' come scrivere: `v = new ArrayOfInt(3);`

`v = new ArrayOfCounter(6);`

ARRAY IN JAVA

- **Gli array Java sono *oggetti,* istanze di una**
- **È un riferimento, quindi *non deve specificare alcuna dimensione!***

```
int[] v;          int v[];
Counter[] w;      Counter w[];
```

- La posizione delle `[]` è a scelta: o dopo il nome, come in C, oppure di seguito al tipo *(novità)*

```
w = new Counter[6];
```

La dimensione *si specifica all'atto della creazione*

ARRAY IN JAVA

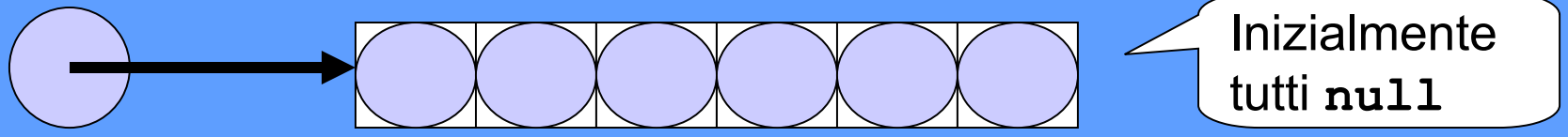
ATTENZIONE: ogni elemento dell'array:

- ***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[6];
```

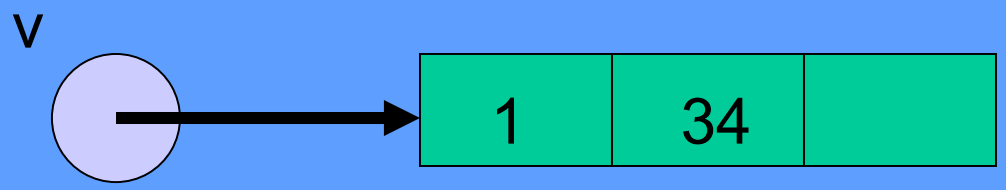


ARRAY IN JAVA

Quindi:

- ***nel primo caso*, ogni elemento dell'array è una normale variabile, usabile "così com'è":**

```
v = new int[3];
v[0] = 1; v[1] = 34;
```

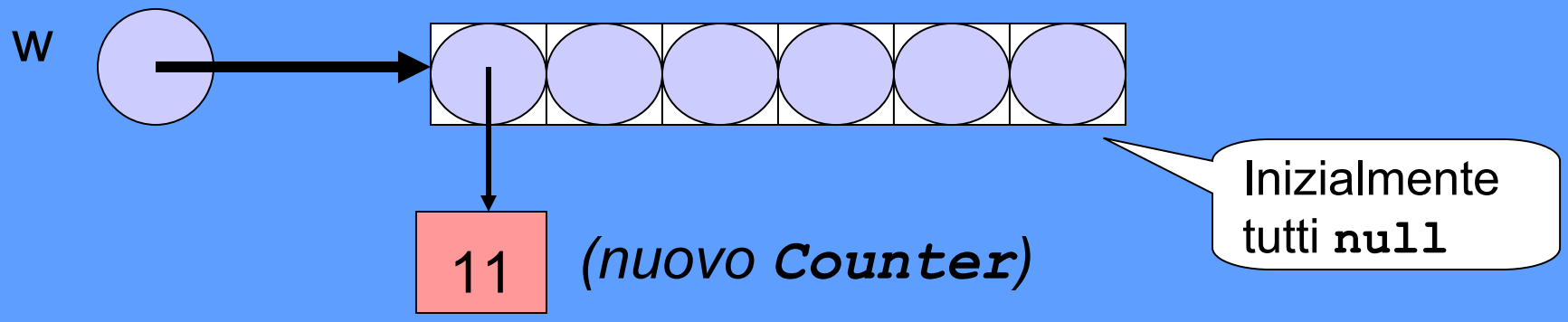


ARRAY IN JAVA

Quindi:

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

```
w = new Counter[6];
w[0] = new Counter(11);
```



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*
`w.length`	*vale 6*

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

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

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`**

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`**)

ESERCIZIO

Quindi:

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

In Java si possono definire variabili *in ogni punto del programma* (non solo all'inizio come in C): in particolare è possibile definire l'indice `i` *dentro* al ciclo `for`

Visibilità *limitata al corpo del ciclo*: fuori da esso, `i` risulta indefinita

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

ESERCIZIO

Esempio d'uso:

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

Output:

```
argomento 0: 34
argomento 1: e
argomento 2: 56
argomento 3: aaa eee
```