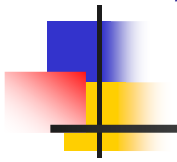
# Linguaggi

*Corso M-Z - Laurea in Ingegneria Informatica*
*A.A. 2008-2009*

Alessandro Longheu
*http://www.diit.unict.it/users/alongheu*
*alessandro.longheu@diit.unict.it*



*- lezione 06 -*

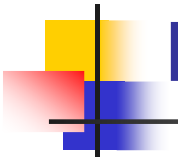
# Stringhe ed Array in Java





## Definizioni

- Java non possiede un tipo primitivo per la stringa; le stringhe non sono "pezzi di memoria" con dentro dei caratteri, come in C, e non sono array di caratteri
- In Java, esiste un'interfaccia java.lang.CharSequence, che definisce i metodi che deve possedere una classe che rappresenti una sequenza di caratteri
- Le stringhe in Java sono oggetti (istanze) della classe String, StringBuilder o StringBuffer, ognuna delle quali è una implementazione dell'interfaccia java.lang.CharSequence, e modellano sequenze di caratteri Unicode a 16 bit
- L'interfaccia java.lang.CharSequence impone quattro metodi:
  - *public char charAt(int index)*, indicizzando la stringa da 0 a length()-1
  - *public int length()*, un metodo (negli array, length è un attributo)
  - *public CharSequence subSequence(int start, int end)*, che restituisce la sottostringa da start a end-1
  - *public String toString()*

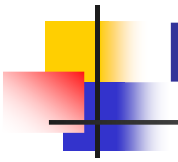


# La classe String

- Gli oggetti della classe String servono per rappresentare sequenze di caratteri immutabili: dopo che una stringa è stata costruita il suo contenuto non può essere modificato, nel senso che una modifica determina di fatto la creazione di una nuova stringa distinta da quella iniziale
- Per lavorare con stringhe modificabili, si dovrebbe utilizzare la classe StringBuilder o StringBuffer
- Fornisce numerosi metodi per lavorare con le stringe
  - Operazioni di base
  - Confronti fra stringhe
  - Costruzioni di stringhe correlate
  - Conversione di stringhe

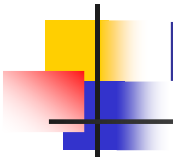






# La classe String

- 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 +.



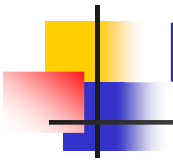
# La classe String

- Java fornisce alcuni supporti extra per gli oggetti della classe Strings, per ragioni di convenienza dato che le stringhe sono frequentemente utilizzate: le stringhe appaiono quasi come dei tipi primitivi ma non lo sono
  - non occorre l'istanziazione esplicita con "new", quindi è possibile scrivere

    *String newString = new String(stringLiteral);*
    *String message = new String("Welcome to Java!");*
    *String message = "Welcome to Java!";*
    *String message = message + "prova"*
  - l'operatore overloaded '+' permette la concatenazione







# La classe String – Costruttori

- public String ()
- public String (String value)
- Public String (StringBuilder value)
- Public String (StringBuffer value)

# La classe String – Metodi di base

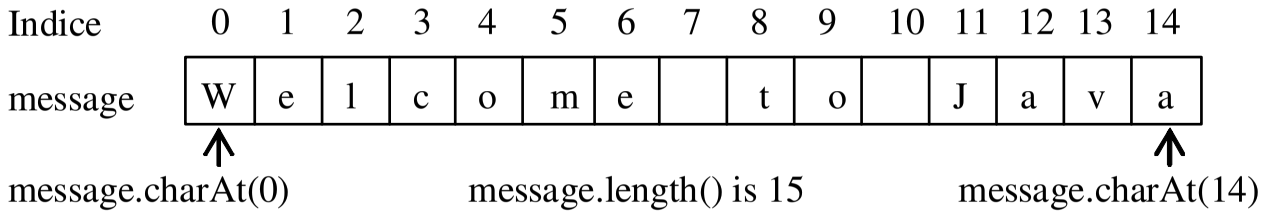
- public charAt(int index)
- public int lenght()
- public CharSequence subSequence(int start, int end)
- public String toString()
- public int indexOf(int ch) con 4 varianti
- public int lastIndexOf(int ch) con 4 varianti





# La classe String – Metodi di base

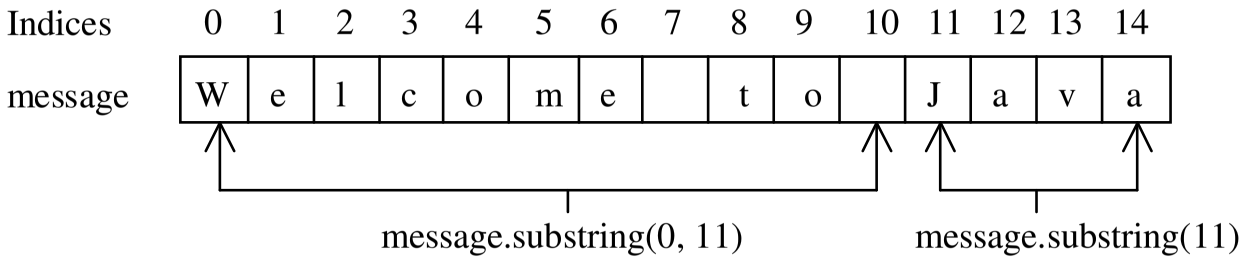
- message[0]  errore!!!
- message.charAt(index)
- L'indice inizia da 0




La lunghezza di una stringa può essere ricavata usando il metodo length():
message = "Welcome";
message.length() (ritorna il valore 7)

# La classe String – Metodi di base

- String s1 = "Welcome to Java";
- String s2 = s1.substring(0, 11) + "HTML";




- Il secondo argomento e' il primo indice non incluso nella sottostringa





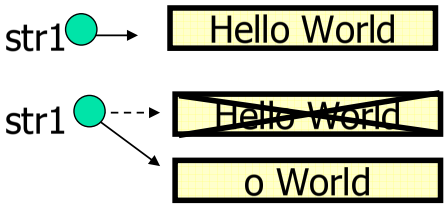
# La classe String – Concatenazione

- *String s3 = s1.concat(s2);  String s3 = s1 + s2;*
- *System.out.println("tre piu cinque "+ 3 + 5);*
  output: tre piu cinque 35
- *System.out.println("tre piu cinque " + (3 + 5));*
  output: tre piu cinque 8
- *System.out.println(3 + 5);*
  output: 8

# La classe String – Modificabilità

- Non si puo modificare il contenuto di un oggetto stringa (Si dice che una stringa e' immutabile)
- Se si vuole modificare una stringa, viene creato un nuovo oggetto String, il riferimento viene aggiornato e la vecchia stringa viene eliminata dal garbage collector

```
String str1 = "Hello World"
str1 = str1.substring(4)
```




**La classe StringBuffer supera questa limitazione**





# La classe String – Confronto

- *String method equals()*
- *String method equalsIgnoreCase()*

```
String str1 = "HeLlO";
String str2 = "hello";
System.out.println(
    str1.equals(str2));
System.out.println(
    str1.equalsIgnoreCase(str2));
```

- Il confronto può anche avvenire solo per una sottostringa (regione), grazie al metodo
  *public boolean regionMatches(int start, String other, int ostart, int count)*
- se si deve confrontare l'inizio o la fine, esistono
  *public boolean startsWith (String prefix, int start)*
  *public boolean endsWith (String suffix)*

# La classe String – Confronto

- String method compareTo(String)

a.compareTo(b) ritorna neg if a<b
ritorna 0 if a equals b
ritorna pos if a>b

```
String str1, str2;
   . . .
   if (str1.compareTo(str2) < 0) {
     // str1 is alphabetically 1st
   } else if (str1.compareTo(str2)==0) {
     // str1 equals str2
   } else { // implies str1 > str2
     // str1 is alphabetically 2nd
   }
```

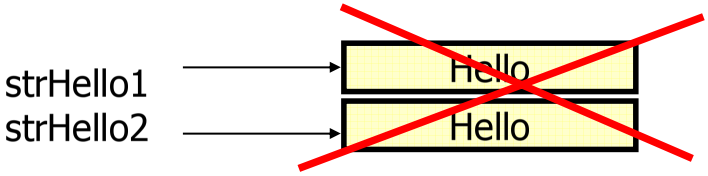
- Il confronto si basa sul valore numerico dei caratteri Unicode, e non tiene conto della nozione localizzata di ordine



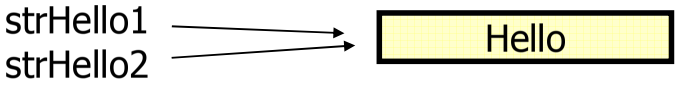


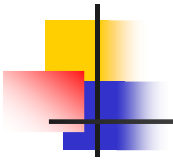
# La classe String – Confronto

Strings ha una caratteristica speciale. In alcuni casi è possibile utilizzare '==' per confrontare due Strings, oltre che equals():
String strHello1 = "Hello";
String strHello2 = "Hello";
Cio' che accade in memoria non e':




ma:





# La classe String – Confronto

Quando il compilatore incontra due linee di codice come le seguenti:

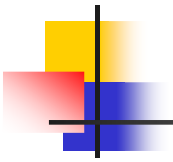
String strHello1 = "Hello";
String strHello2 = "Hello";

Il compilatore è "furbo" abbastanza per riconoscere che le due stringhe sono identiche. Quindi decide di risparmiare memoria ed utilizzare la stessa locazione di memoria. I due riferimenti strHello1 e strHello2 puntano alla stessa locazione di memoria, per cui in tal caso il confronto strHello1==strHello2 da true.
Lo stesso risultato si ottiene scrivendo:

String strHello2 = "Hell" + "o";

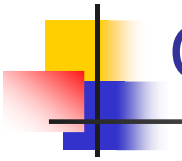






# La classe String – Confronto

il caso speciale per "==" nel confronto fra oggetti String NON SEMPRE FUNZIONA, in particolare se un oggetto String e' creato con l'uso della parola chiave 'new', o se i valori sono dati in input dall'utente, i due oggetti String non occuperanno comunque lo stesso spazio di memoria, anche se i caratteri sono gli stessi.
Pertanto conviene in generale non confrontare String con "==", utilizzarlo solo per confrontare tipi primitivi e utilizzare equals per confrontare oggetti.
"==" però funziona correttamente se si applica il metodo *intern()* ad entrambe le stringhe, quindi:
*s1.equals(s2)* oppure *s1.intern()==s2.intern()*
l'uso di intern() permette l'uso di ==, più veloce di equals()



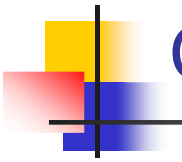
# La classe String
# Gestione maiuscolo e minuscolo

*equalsIgnoreCase()*
*esegue il test di uguaglianza fra due oggetti String ignorando il case*
*toUpperCase()*
*crea un versione della stringa con caratteri maiuscoli (uppercase)*
*toLowerCase()*
*crea un versione della stringa con caratteri minuscoli (lowercase)*

- Nessuno di questi modifica la stringa originale.
- Il concetto di maiuscolo e minuscolo è locale sensitive, esistono infatti toUpperCase(Locale loc) e toLowerCade(Locale loc) per specificare un locale diverso dal default



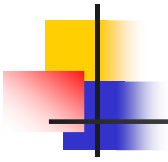




# La classe String
# Gestione maiuscolo e minuscolo

```
String str = "Paul Oakenfold";
String strSmall = str.toLowerCase();
System.out.println(str);
System.out.println(strSmall);
System.out.println(str.toUpperCase());
System.out.println(str);
```

**Output:**
**Paul Oakenfold**
**paul oakenfold**
**PAUL OAKENFOLD**
**Paul Oakenfold**



# La classe String – Ricerca Pattern

```
String str = "catfood";
int location = str.indexOf("food");
System.out.println("pattern food begins at " + location);
System.out.println(
        "pattern dog begins at " +
        str.indexOf("dog"));
```

```
catfood
0123456
```

Output:
pattern food inizia in 3
pattern dog inizia in -1

-1 è ritornato quando NON viene trovato il pattern !



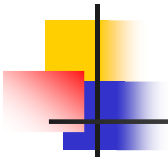


# La classe String – Ricerca Pattern

Si può anche specificare l'indice di partenza della ricerca, utile per trovare tutte le occorenze di un pattern:

```
String str = "abracadabra abracadabra";
int index = str.indexOf("abra");
while (index != -1) {
   System.out.println(
     "found at " + index);
   index = str.indexOf("abra", index + 1);
}  // il -1 finale non viene stampato
```

Output:
found at 0
found at 7
found at 12
found at 19



# La classe String – toString()

- 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.

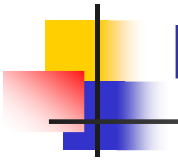




# La classe String – toString()

```
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, quindi stampa un identificativo dell'oggetto c.



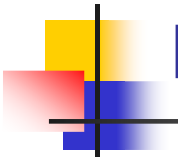
# La classe String – toString()

- È possibile ridefinire esplicitamente il metodo toString() della classe Counter, facendogli stampare ciò che si ritiene opportuno, ad esempio:

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

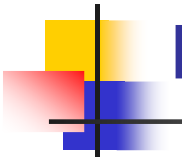






# La classe String – Conversioni

- È possibile convertire da e verso String()
- La conversione verso String utilizza il metodo valueOf della classe String che prevede diverse versioni in overloading, ognuna accettante come parametro un tipo primitivo
- La conversione da String verso un tipo primitivo si realizza con metodi differenti:
  - *Boolean.parseBoolean (String)*
  - *Byte.parseByte(String, int base)*
  - *str.charAt(pos)*
  - *Short.parseShort(String, int base)*
  - *Integer.parseInt(String, int base)*
  - *Long.parseLong(String, int base)*
  - *Float.parseFloat(String)*
  - *Double.parseDouble(String)*

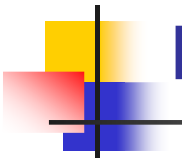


# La classe String – Conversioni

- È possibile convertire da String() ad array di char e viceversa, in particolare:
  - la classe String prevede costruttori che accettano array di char come parametri
  - la stessa classe prevede i metodi toCharArray() e getChars, per ottenere la conversione rispettivamente totale o di una sottostringa di quella di partenza in un array di char
- È possibile convertire da String() ad array di byte, con un approccio simile a quello usato per i char, tuttavia occorre specificare la codifica perché il byte è ad 8 bit, mentre i caratteri della stringa sono codificati a 16; java supporta la gestione delle codifiche, fornendo le predefinite ISO 8859-1, US-ASCII ecc

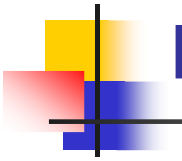






# La classe String
# Espressioni Regolari

- Java fornisce librerie apposite per la gestione delle ER e per la ricerca e/o sostituzione di pattern (espressi tramite ER) all'interno di stringhe
- in generale, il modello fornito prevede:
  - la compilazione dell'ER, effettuata creando un apposito oggetto della classe Pattern al quale si da in ingresso l'ER stessa
  - successivamente, si può ottenere un oggetto Matcher dall'oggetto Pattern per potere ricercare l'ER all'interno di una CharSequence (ad esempio una stringa)
  - il Matcher può quindi effettuare operazioni sulla sequenza, come ad esempio la sostituzione di occorrenze della ER trovate nella CharSequence



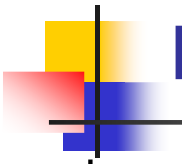
# La classe String
# Espressioni Regolari

- Esempio per sostituire "sun" con "moon"

```
Pattern p=Pattern.Compile("sun");
Matcher m=p.matcher(input);
StringBuffer r=new StringBuffer();
boolean found;
while ((found = m.find()))
 m.appendReplacement(r, "moon");
m.appendTail(r);
```

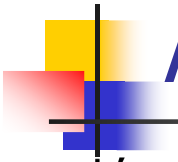






# La classe StringBuilder

- La classe StringBuilder permette l'uso di stringhe modificabili
- Stringbuilder dovrebbe essere usata quindi ogni volta che occorre modificare una stringa; è possibile utilizzare a questo scopo la classe String, ma poiché lavora su stringhe immutabili, ogni modifica di fatto crea un nuovo oggetto di tipo String, allocando memoria in più, con ripercussioni su efficienza e garbage collection
- StringBuilder è simile a String, supporta molti metodi analoghi ed altri specifici per la modifica delle stringhe, tuttavia sono classi "sorelle" entrambe implementazioni indipendenti di CharSequence
- StringBuffer è identica a StringBuilder ma è anche thread safe; è una classe più vecchia che tuttavia viene ancora utilizzata

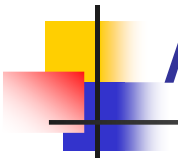


# Array

- L'array in Java fornisce il concetto presente nei più comuni linguaggi di programmazione; un array in Java è però un oggetto che estende implicitamente la classe Object
- Un array e' una struttura statica, una volta creato la sua dimensione (numero di elementi che lo compongono) non puo' essere più modificata; per sequenze di lunghezza modificabile, Java fornisce la classe Vector
- L'array puo' contenere elementi che sono tipi primitivi, o oggetti (in realta' riferimenti). In generale gli array sono omogenei, cioe' ogni elemento è dello stesso tipo. Questo limite puo' essere superato con il polimorfismo.
- una variabile di tipo array ammette gli stessi modificatori degli attributi, però che si applicano alla variabile nel suo complesso e non ai singoli elementi dell'array, per i quali non è possibile specificare alcun modificatore







# Array

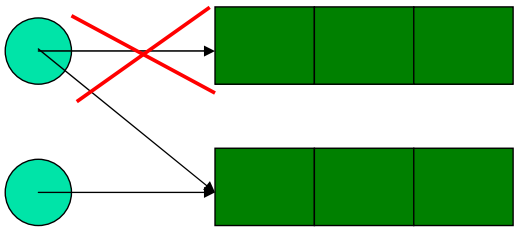
- Gli array Java sono oggetti, istanze di una classe speciale denotata da [ ]; La posizione delle [ ] è a scelta: dopo il nome, come in C, oppure di seguito al tipo:
  *<elemType>[ ] <arrID> oppure*
  *<elemType> <arrID>[ ];*
- Esempi:
  *int[ ] gradi;*
  *int gradi[];*
  *float pressione[];*
  *boolean[] stato;*
- La dimensione si specifica all'atto della creazione:
  *gradi = new int[10];*
  *pressione = new float[100];*
  *stato = new boolean[15]*
- E' possibile la dichiarazione e creazione/inizializzazione implicita:
  *int[] x = {10, 100, 90, 50, 45}*

# Array

- Esempi di assegnazione fra array

int oldcount[] = new int[3];

int counts[] = new int[3];



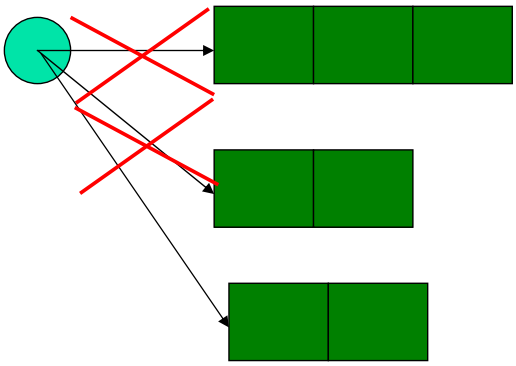
oldCounts = counts;

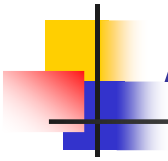




# Array

```
float[] pressione = new float[3];
int i=3;
int j=1;
   while(i>j) {
        pressione = new float[2];
        j++
        }
```





# Array

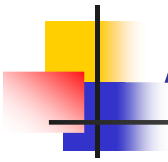
- La dimensione dell'array può essere nota tramite l'attributo length (che nella classe String è invece un metodo length())

```
int[ ] gradi = new int[10];
for (int i=0; i < gradi.length; i++)  {
      gradi[i] = 0;
}
```

- L'escursione dell'indice dell'array è da 0 a N-1 per N elementi, come in C; length è l'N, quindi una scansione dell'array tramite ciclo può andare da 0 a (array.length)-1







# Array

Esempio che mostra la similitudine con il C:

```
public int maggiore(int[ ]  myArray) {
      int massimo = myArray[0];
    int i;
    for (i = 1; i < myArray.length; i++)
        {
          if (massimo < myArray[i])
              massimo = myArray[i];
        }
     return massimo;
}
```

# Array

- Se un array è di oggetti, allora:
  - l'identificatore dell'array è un riferimento ad un array di oggetti
  - ogni elemento dell'array è un riferimento a un oggetto della classe specificata come tipo base dell'array
- Istanziare l'array di oggetti non assicura l'istanziazione dei vari oggetti che costituiscono gli elementi dell'array, elementi che quindi devono essere esplicitamente istanziati.

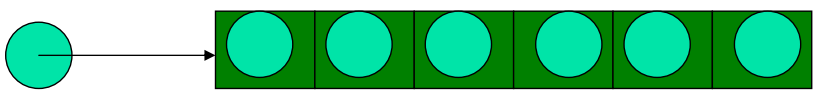




# Array

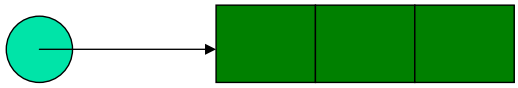
- ogni elemento dell'array è una variabile, se gli elementi dell'array sono di un tipo primitivo (int, float, char, …), ad esempio *v = new int[3];*

- è un riferimento a un (futuro) oggetto, se gli elementi dell'array sono (riferimenti a) oggetti, ad esempio *w = new Counter[6];* presenta 6 oggetti Counter, inizialmente tutti null

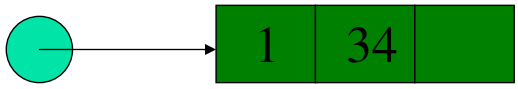


# Array

- Nel primo caso ogni elemento dell'array è una normale variabile usabile cosi com'è:



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

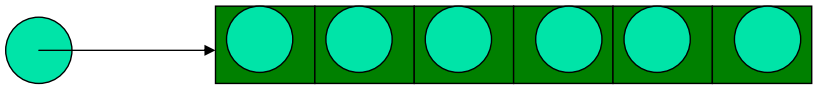




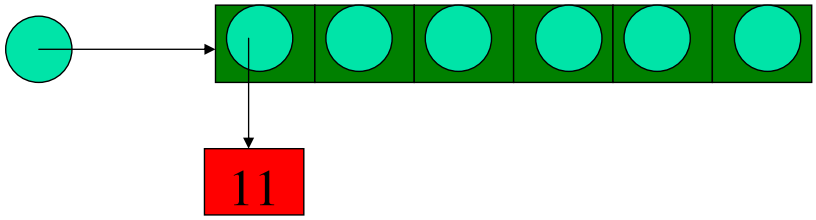


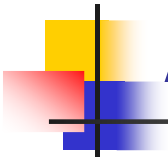

# Array

- Nel secondo caso invece, ogni elemento della array è solo un riferimento: se si vuole un nuovo oggetto bisogna crearlo
- w = new Counter[6];



w[0] = new Counter(11);





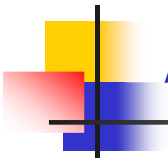
# Array

- Esempio di stampa del vettore di argomenti passati dalla linea di comando

```
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]);
        }
}
```







# Array

- Dichiarazione – crea solo il riferimento – valore null
  - int myInts[];
  - int[] myInts;
- Istanziazione
  - myInts = new int[10];
- dichiarazione e istanziazione
  - int[] myInts = new int[10];
- accesso a ciascun elemento
  - myInts[3] = 9;
  - x = myInts[4];
- inizializzazione statica
  - int[] myInts = {1,2,5,6,7,4};

# Array multidimensionali

- Gli Arrays possono contenere elementi che sono primitivi o oggetti, quindi anche array
- a differenza del C, ogni array interno può avere una propria dimensione, ad esempio:

```
String s[][] =
{
 {"io", "sono", "la riga" "uno"},
 {"io", "sono", "la", "seconda"},
 {"io", "la", "terza"}
};
```

- s.length e' uguale a 3
- s[0].length è uguale a 4
- s[1].length è uguale a 4
- s[2].length è uguale a 3

| | | | |
|---|---|---|---|
| **io** | **sono** | **la riga** | **uno** |
| **io** | **sono** | **la** | **seconda** |
| **io** | **la** | **terza** | |





# Array multidimensionali Esempio

Esempio di accesso agli elementi dell'array multidimensionale:

```
int [ ] [ ] multiplicationTable;
multiplicationTable = new int [11] [21];
for (int i=0; i < multiplicationTable.length; i++) {
   for (int j=0; j < multiplicationTable[i].length;    j++) {
    multiplicationTable[i][j] = i*j;
   }
}
```

# Array multidimensionali Esempio

```
public class ArrayDemo {
   public static void main(String args[])  {
    String s[][] = {
          {"io", "sono", "la riga" "uno"},
          {"io", "sono", "la", "seconda"},
          {"io", "la", "terza"}
     }; for(int row = 0; row < s.length; row++) {
          System.out.println
          ("La riga " + row + " ha " + s[row].length + " colonne");
          for(int col = 0; col < s[row].length; col++) {
              System.out.print(" <" + s[row][col] + "> ");
          }
          System.out.println();
      }
   }
}
```

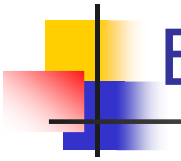
**La riga 0 ha 4 colonne**
**<io> <sono> <la riga> <uno>**
**La riga 1 ha 4 colonne**
**<io> <sono> <la> <seconda>**
**La riga 2 ha 3 colonne**
**<io> <la> <terza>**





# Array multidimensionali Esempio

```
public class Multiple
{ public static void main(String args[])
  {  int[][] numbers;
     numbers = new int[5][];
     for(int row=0; row<numbers.length; row++)
     {  numbers[row] = new int[(int)(Math.random()*8)];
        for(int col=0; col<numbers[row].length; col++)
        {   numbers[row][col] = row*10 + col;
            if (row<1) System.out.print("0");
            System.out.print(numbers[row][col]+ " " );
        }
        System.out.println();
     }
  }
}
```



# Array multidimensionali Esempio

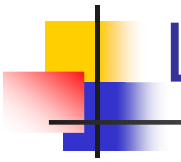
Output del programma:

```
>java Multiple
00 01 02
10 11 12 13 14 15 16
20 21 22 23
30 31 32 33 34 35
40 41 42

>java Multiple
00 01
10 11 12 13
20 21 22 23 24 25 26
30 31 32 33 34 35
40 41 42
```

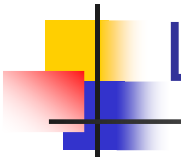






# La classe Vector

- La classe Vector rappresenta una struttura dati simile all'array con alcune caratteristiche addizionali:
  - La dimensione di un vector non deve essere dichiarata, e può crescere quando necessario
  - Esistono metodi per aggiungere, inserire e rimuovere un elemento in una posizione specificata
  - L'elemento di un vector deve essere un oggetto (non può essere un tipo primitivo)



# La classe Vector

- Esempio: use della classe Vector per creare una collezione di Counter

  *import java.util.*;*

  Dichiarazione

  *Vector contatori= new Vector();*

  Aggiungere Elementi

  *contatori.addElement(c1);*

  *contatori.addElement(c2);*

  Accesso all'elemento i

  *c3 = (Counter) contatori.elementAt(i);*

  Modifica di un elemento di posizione i

  *contatori.setElementAt(c4, i);*

  Inserimento di un elemento in posizione i

  *contatatori.insertElementAt(c4, i);*

  Rimuovere un elemento di posizione I

  *contatori.removeElementAt(i);*