Lezione 2
Strutture Fondamentali

Vittorio Scarano
Corso di Programmazione Distribuita (2003-2004)
Laurea di I livello in Informatica
Università degli Studi di Salerno

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Anatomia di un semplice programma

HelloWorld.java

```
public class HelloWorld {
   public static void main(String[] args) {
    System.out.println("Ciao!");
    }
}
```

- Compilazione:
  - genera il file HelloWorld.class
- Esecuzione:
  - java HelloWorld

- In generale:
  - maiuscolo/minuscolo
  - parentesi graffe
- Modificatore di accesso
- Dichiarazione di classe
  - per convenzione
    - Ogni parola del nome inizia con carattere maiuscoli
- Attenzione:
  - nome del file uguale al nome della classe!!

Anatomia di un semplice programma (2)

HelloWorld.java

```
public class HelloWorld {
   public static void main(String[ ] args) {
    System.out.println("Ciao!");
    }
}
```

- Metodo main
  - necessario per la classe della applicazione
  - parametri
  - corpo
- Istruzioni
  - terminate con ;
- Chiamata metodo:
  - oggetto.metodo(parametri)
- Parametri chiamata
  - stringa

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

I commenti

```
/* HelloWorld.java
     programma di esempio
     Vittorio Scarano (20/10/2001)
 */
/**
     primo metodo
     @param stringhe su linea di comando
     @return  niente
*/
public class HelloWorld {
       public static void main(String[ ] args) {
      // stampiamo “Ciao!”
     System.out.println(“Ciao!”);
     }
}
```

- Diverse possibilità:
  - commenti alla C
    - su più righe
  - commenti alla Java
    - su una sola riga
- Commenti alla *Javadoc*
  - permettono la creazione automatica di documentazione
  - molto utili per progetti
  - su più linee

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Tipi di dati

- Java è un linguaggio fortemente tipizzato (*strongly typed*)
- Esistono otto tipi primitivi
  - 4 per i numeri interi
  - 2 per i numeri in virgola mobile
  - 1 per i caratteri (Unicode)
  - 1 per i booleani

Tipi interi

- Quattro tipi di dati
  - byte (8), short (16), int (32), long (64)
- Dimensioni indipendenti dalla architettura
- Commenti:
  - int è il tipo di utilizzo più comune
  - long, byte, short per situazioni particolari
- Costanti:
  - interi long: 400000000L
  - esadecimali: 0x4AFF
  - ottali: 0736 (sconsigliato per la confusione)

Tipi per numeri in virgola mobile

- Due tipi di dato:
  - float (32), 6 cifre significative
  - double (64), 15 cifre significative
- Commenti:
  - double è di solito il più utilizzato
- Costanti:
  - float: 3.456F
  - double: 3.456 (senza alcun suffisso)
- Specifica IEEE 754

La specifica IEEE 754

- Standard per la rappresentazione
- Tre valori speciali:
  - infinito positivo (Double.POSITIVE_INFINITY)
  - infinito negativo (Double.NEGATIVE_INFINITY)
  - NaN (*Not a Number*) (Double.NaN)
- Esempi:
  - 1/0 vale Double.POSITIVE_INFINITY
  - 0/0 vale Double.NaN

Il tipo carattere

- Tipo char (16)
- Costanti carattere: 'A'
- Rappresentazione Unicode
  - rappresentazione su 16 bit
  - estende la rappresentazione ASCII (7 bit) e la sua estensione "Latin-1" (8 bit) (ISO 8859-1)
- Costanti carattere attraverso Unicode (0-65535)
  - '\u0000' = 0 ASCII (buon vecchio '\0' in C!)
  - '\u0041' = 65 ASCII ('A')
  - Caratteri particolari:
    - '\b' (Backspace) '\t' (TAB) '\n' (Newline) '\"' '\''

Il tipo booleano

- Contiene valori true o false
- Non è consentito l'utilizzo di interi per rappresentare valori booleani
- Per i programmatori C:
  - niente più preoccupazioni per
    if (x = 0) ….
  - in Java dà errore in compilazione

Tabella riassuntiva dei tipi di dato

Tipo	*contiene..*	*Default*	*Dim.*	*Range*
byte	intero con segno	0	8	-128+127
short	intero con segno	0	16	-32768 +32767
int	intero con segno	0	32	-2147483648 +2147483647
long	intero con segno	0	64	-2^{63} a $+2^{63}-1$
float	IEEE 754	0.0	32	6 cifre significative 10^{-46} a 10^{38}
double	IEEE 754	0.0	64	15 cifre significative 10^{-324} a 10^{308}
char	Carattere Unicode	0 Unicode \u0000	16	
boolean	true o false	false	1	

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Variabili

```
double prezzo;
int giorniDiVacanza;
long popolazioneDelMondo;
char carattereUsatoPerSi;
boolean fatto;
```

- Ogni variabile ha un tipo.
- Si indica il tipo e identificatore
  - Ogni variabile inizia con una lettera e seguita da lettere o cifre.
  - Vietato:
    - usare le parole riservate del linguaggio Java
    - usare i simboli '+', '@' etc. etc
  - Convenzione:
    - ad ogni parola si inserisce una maiuscola, tranne la prima (solo le classi iniziano con maiuscole)

Assegnazioni ed inizializzazioni

- Come in C: lvalue = rvalue
  - dove
    - lvalue è una variabile
    - rvalue è una espressione
- Importante:
  - in Java le dichiarazioni possono essere ovunque

```
double prezzo=4000;
System.out.println (prezzo);
int giorniDiVacanza = 10; //dichiarazione
```

  - visibilità all'interno del blocco (come in C)

Costanti

- Facendo precedere final ad una dichiarazione si dichiara una costante
  - possibile assegnare il valore solo una volta
- Convenzione: nome in maiuscolo

```
final double PI_GRECO = 3.14;
```

- Per le costanti usate da tutti i metodi di una classe:
  - dichiarazione nella classe fuori dai metodi
  - *costanti di classe*
  - precedute da static final

```
public class Pippo {
    static final double N = 32;
…
}
```


Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Operatori

- Consueti operatori aritmetici: + - * /
- Operatore / :
  - divisione intera se entrambi gli operandi sono interi, in virgola mobile altrimenti
- Possibile usarli nella inizializzazione (non come il C)

```
int n = 5;
int a = n *3; // a vale 15
```

- Scorciatoia tipica (C) per operatori in assegnazioni:

```
a + = 6; // a = a + 6
b *= 12; // b = b * 12
```

Operatori

- Come in C per incremento e decremento

```
a ++;   // incrementa a
b --;   // decrementa b
c = ++a +12;
c = a++ +12;
```

- Operatori relazionali e booleani
  - valutazione corto-circuitata

```
if (a == 7)….
if ( (a == 7) && (b >= 10) )….
```

- Operatore ternario

```
a = (x < y) ? x : y;
```


Operatori sui bit

- Operatori con comportamento uguale al C:
  - & | ^ ~

```
int quartoBit = ( n & 8 ) / 8 ;
```

  - >> e <<

```
int nDivisoQuattro =  n >> 2;
```

- Commenti sullo shift a destra:
  - l'operatore >> mantiene il bit di segno (importante per i numeri negativi)
  - esiste l'operatore >>> che fa lo shift anche del bit di segno
  - risolta l'ambiguità del C/C++

Funzioni matematiche

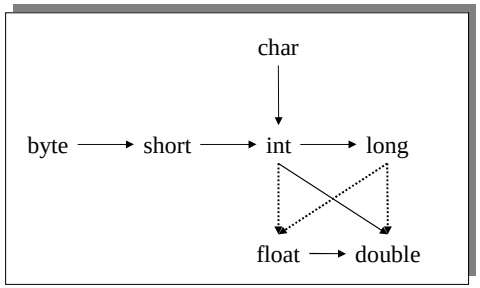
- Raccolte in una classe Math
- Esempio:

```
double x = 4;
double z = Math.sqrt(x);
System.out.println (z);
```

- Alcune funzioni utili:
  - Math.pow(x,a)
  - Math.sin
  - Math.exp
- Alcune costanti utili:
  - Math.PI
  - Math.E

Conversioni tra tipi numerici

- Essendo *strongly typed*, Java permette solo alcune conversioni tra tipi numerici:



- Conversioni
  - con/senza perdita
- Catena di conversione
  - tra interi
  - tra numeri in virgola mobile

Conversione implicite numeriche

- Con operatori binari
  - se uno degli operatori è double allora l'altro è convertito in double
  - altrimenti, se uno degli operatori è float allora l'altro è convertito in float
  - altrimenti, se uno degli operatori è long allora l'altro è convertito in long
  - altrimenti, entrambi gli operatori sono convertiti in int

Operazioni di casting numerico

- Possibili per forzare conversioni con perdita di informazione
- Sintassi simile al C/C++:

```
double a=9.98;
int n = (int) a;      //n vale 9
```

- Il valore non viene arrotondato.
  - Per arrotondamenti:

```
double a=9.98;
int n = (int) Math.round(a);      //n vale 10
```

- Non si può convertire booleani a interi:
  - necessario

```
int i = (valoreBooleano) ? 1:0;
```


Priorità ed associatività

Operatori	Associatività
[ ], ( ), chiamata metodo	da sinistra a destra
!, ~, ++, --, +(unario), -(unario), casting, new	da destra a sinistra
*, /, %	da sinistra a destra
+, -	da sinistra a destra
>>, <<, >>>	da sinistra a destra
= =, !=	da sinistra a destra
&	da sinistra a destra
^	da sinistra a destra
|	da sinistra a destra
&&	da sinistra a destra
||	da sinistra a destra
?:	da sinistra a destra
=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=	da destra a sinistra

Le parole riservate del linguaggio

abstract	double	int	super
boolean	else	interface	switch
break	extends	long	synchronized
byte	false	native	this
byvalue	final	null	throw
catch	float	package	transient
char	for	private	true
class	goto	protected	try
const	if	public	void
continue	implements	return	while
default	import	short	
do	istanceof	static	

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Le stringhe in Java

- Soluzione ai problemi del C/C++:
  - la stringa non è un tipo predefinito ma esiste una classe String che offre metodi per la gestione
- Il compilatore le tratta però come tipi predefiniti:
  - crea automaticamente un oggetto String se incontra una stringa tra "…"
  - usa un operatore + di concatenazione
- Le istanze di String sono non modificabili
  - si usa StringBuffer per stringhe modificabili

Alcuni esempi

- Una istanza della classe String

```
String s = “Ciao a tutti!”;
String vuota = “”;
```

- Concatenamento:

```
String inizio = “Ciao ”;
String nome = “Vittorio ”;
String fine = “come stai?”;
String messaggio = inizio + nome + fine;
    // vale “Ciao Vittorio come stai?”
```

- Concatenamento tra valori non stringa:

```
String inizio = “Ci siamo visti ”;
int volte = 15;
String fine = “ volte”;
System.out.println(inizio + volte + fine);
        // stampa a video “Ci siamo visti 15 volte”
```


Sottostringhe

- Tipico approccio di un linguaggio procedurale:
  - uso una *funzione di libreria*
- Tipico approccio di un linguaggio *Object Oriented*
  - si usa *un metodo messo a disposizione dell’oggetto*
- Per estrarre una sottostringa si usa un metodo dell’oggetto String
- Tipico della programmazione ad oggetti
  - non esiste una “funzione” per una certa operazione
  - ma si usa la funzione (il metodo) che l’oggetto mette a disposizione
  - del metodo *indipendente* dalla implementazione interna

Sottostringhe

- Si usa il metodo substring della variabile (oggetto)

```
String s = "Pronto, come stai?";
String p = s.substring(0,5);
                    // p vale "Pronto"
```

- Numerazione dei caratteri alla C/C++
  - si parte da 0

Altre operazioni su stringhe

- La lunghezza

```
String s = "Pronto, come stai?";
int len = s.length();
                    // len vale 18
```

  - notare l'uso del metodo
- Carattere in posizione i

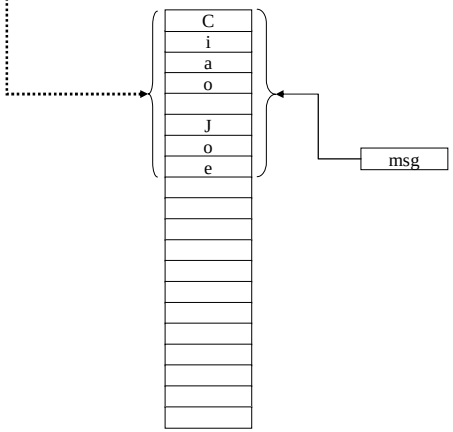
```
String s = "Pronto, come stai?";
char c = s.charAt(3);
                    // c vale 'n'
```

  - attenzione: sempre contando da 0 !
- Non è possibile cambiare il valore:
  - per cambiare si deve usare una riassegnazione

```
String msg = "Ciao Joe";
msg = msg.substring(0,3) + " Bill";
          // msg vale "Ciao Bill"
```


Anticipazione: allocazione oggetti

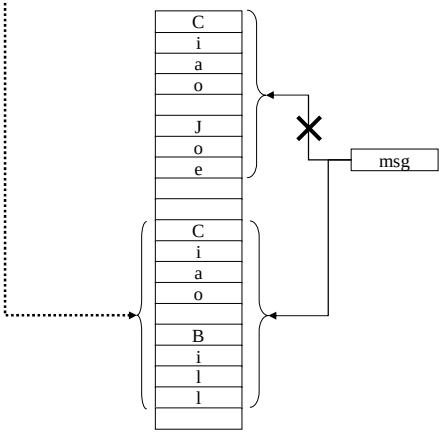
```
String msg = "Ciao Joe";
```



- Per comprendere le stringhe:
  - dichiarare un oggetto (variabile) String comporta la allocazione di un oggetto
  - a cui la variabile msg "*fa riferimento*"

Anticipazione: allocazione oggetti (2)

```
String msg = "Ciao Joe";
msg = msg.substring(0,3) + " Bill";
```



- La successiva istruzione:
  - estrae (tramite il metodo substring) parte della string a cui fa riferimento msg
  - la concatena a " Bill"
  - e *crea un nuovo oggetto*
  - a cui fa riferimento adesso la variabile msg
  - l'oggetto stringa "Ciao Joe" se non è riferita da altre variabili verrà deallocata dal garbage collector

Altre operazioni sulle stringhe (2)

- Per verificare la uguaglianza tra stringhe:

```
String s = “Pronto”;
String p = “Pronti”;
if (s.equals(p)) ….
```

- Attenzione:

```
String s = “Pronto”;
String p = “Pronti”;
if (s == p) ….
```

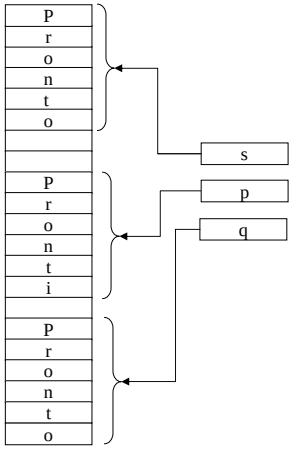
  – è differente (vedremo in seguito)
- Possibile anche:

```
if (“Hello”.equals(p)) ….
```

- Classe String contiene più di 50 metodi:
  – consultare le API

Anticipazione: l’uguaglianza oggetti

```
String s = “Pronto”;
String p = “Pronti”;
String q = “Pronto”;
```



- La uguaglianza tra stringhe:
  – il metodo `if (s.equals(p)) ….`
  – permette di valutare la uguaglianza tra gli oggetti stringa
  – se usiamo `if (s == p ) ….`
  – allora valutiamo la uguaglianza tra le variabili di riferimento **non** la uguaglianza tra stringhe

```
if (s.equals(p))    //falso
if (s == q)         // falso
if (s.equals(q))    // vero
```


Alcuni commenti sulle stringhe

- Immutabilità di una stringa:
  – permette la condivisione
    - una copia di s e la stringa *condividono* la stessa memoria
    - sarà chiarito quando parleremo di oggetti, copie e *cloni*
- Confronto con il C
  – la stringa ***non*** è un array di caratteri
  – piuttosto, è *una sorta* di puntatore a caratteri la cui allocazione viene dinamicamente gestita a run-time
  – notevole risparmio e facilità di gestione

Input e output

- Si è visto come è facile stampare su video (descrittore di standard output):

```
System.out.println (“Il risultato è “ + ris);
```

- Per lo standard input (tastiera) non è così immediato
  – usiamo una finestra di dialogo messa a disposizione da Swing (interfaccia grafica per Java)
- Si usa la chiamata di metodo:

```
String risposta;
risposta = JOptionPane.showInputDialog(“Come ti chiami?”);
```

  – nella stringa risposta c’è quello che digita l’utente

Input di tipi di dato numerici

- showInputDialog permette input di stringhe
- Per ottenere tipi di dato numerici si ottiene dalla stringa ottenuta il valore nel tipo desiderato
  - si usa un metodo messo a disposizione dalla classe Integer

```
String risposta;
risposta = JOptionPane.showInputDialog(“Quanti anni hai?”);
int eta = Integer.parseInt (risposta);
```

  - per i double esiste ad esempio il metodo:

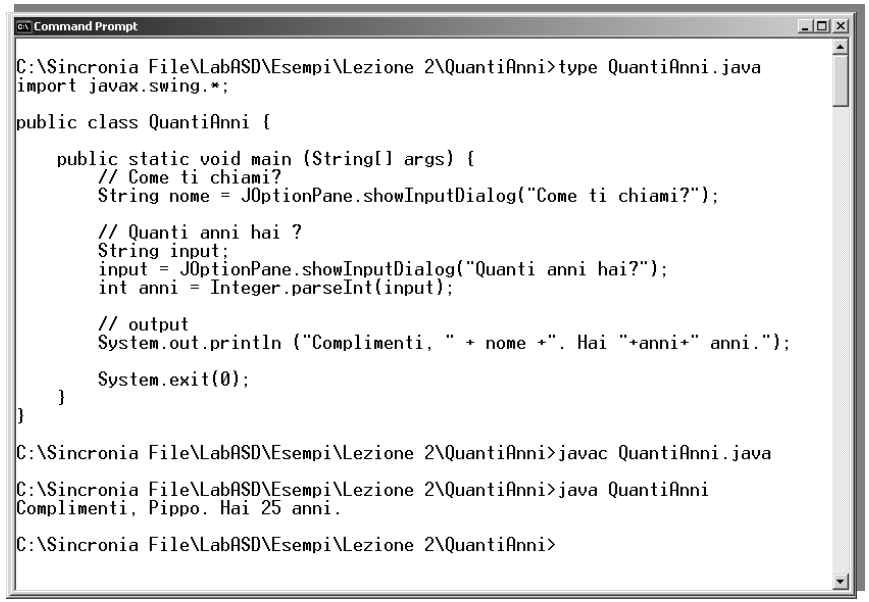
```
double d = Double.parseDouble(risposta);
```


Proviamo con la classe QuantiAnni

```
import javax.swing.*;

public class QuantiAnni {
   public static void main (String[] args) {
     // Come ti chiami?
     String nome =
          JOptionPane.showInputDialog("Come ti chiami?");

     // Quanti anni hai ?
     String input;
     input =
       JOptionPane.showInputDialog("Quanti anni hai?");
     int anni = Integer.parseInt(input);

     System.out.println ("Complimenti, " + nome +
                                  ". Hai "+anni+" anni.");
     System.exit(0);
   }
}
```

- *package* javax.swing:
- Input del nome
- Input della eta
  - intero
  - uso di metodo statico per la conversione stringa/intero
- Output
- Uscita dal programma
  - necessario con uso Swing

Cosa accade…



Formattazione dell’output

- Se usiamo il metodo println() i parametri vengono stampati con la massima rappresentazione possibile
- Questo può non essere quello che vogliamo:

```
x = 1000.0 / 3.0
System.out.println (x);
    // stampa 333.3333333333335
```

- In questo caso vogliamo poter controllare la visualizzazione
- Usiamo la classe NumberFormat
  - una delle utilities del linguaggio per gestire la internazionalizzazione dei programmi

La classe NumberFormat

- Per ottenere il setting dei parametri nazionali:

```
double x = 1000.0 / 3.0
NumberFormat formatter = NumberFormat.getNumberInstance();
String s = formatter.format (x);
System.out.println (s);                    // stampa 333,333
```

  – usa lo standard italiano per rappresentare i numeri con virgola

- Si può anche modificare:

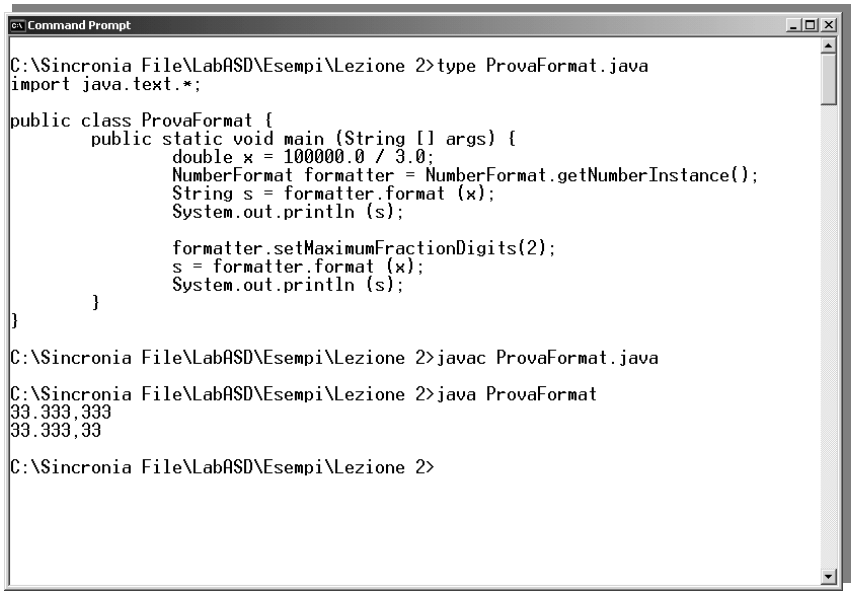
```
double x = 1000.0 / 3.0
NumberFormat formatter = NumberFormat.getNumberInstance();
formatter.setMaximumFractionDigits(2);
String s = formatter.format (x);
System.out.println (s);                    // stampa 333,33
```


Proviamo con la classe ProvaFormat

```
import java.text.*;

public class ProvaFormat {
    public static void main (String [] args) {
        double x = 100000.0 / 3.0;
        NumberFormat formatter =
            NumberFormat.getNumberInstance();
        String s = formatter.format (x);
        System.out.println (s);

        formatter.setMaximumFractionDigits(2);
        s = formatter.format (x);
        System.out.println (s);
    }
}
```

- *package* java.text:
- Stampa del numero
  – con il numero di cifre predefinito
  – con la impostazione nazionale (virgola decimale)
- Cambio del numero di cifre
- Ristampa

Cosa accade…



Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Controllo del flusso

- A disposizione
  - istruzioni condizionali
  - istruzioni iterative
- Essenzialmente basate sul C/C++
- Blocco (istruzione composta):
  - sequenza di istruzioni racchiuse tra parentesi graffe
- Valgono le regole di visibilità usuali (in C)
  - ma non esiste la possibilità di oscurare variabili (tipica del C).

Esempio di visibilità di variabili

```
public static void main(String[] args) {
    int n;
    ….
    {   int k;
        …..
        ….. // k definita solo fin qui
    }
}
```

```
public static void main(String[] args) {
    int n;
    {
        int n;   // errore in compilazione
        …..
    }
}
```

- Blocco:
  - insieme di istruzioni tra parentesi graffe
- Visibilità di k
- Visibilità di n
- Oscuramento variabili non permesso:
  - errore in compilazione

Istruzioni condizionali: if

- Come nel C:

```
if (condizione)
    istruzione1;
else
    istruzione2;
```

  - Ogni istruzione della condizione può essere sostituita da blocchi
  - ramo else non obbligatorio
  - else viene riferita all' if più vicino (regola sintattica per il *dangling else*)

Istruzioni iterative: while e do..while

- Come nel C:

```
while (condizione)
    istruzione;
```

  - controllo ad inizio ciclo
- Controllo a fine ciclo:

```
do
    istruzione;
while (condizione);
```

Istruzioni iterative: for

- Come in C:

```
for (inizializzazione; condizione; passo)
    istruzione;
```

  - possibile (ed utile) dichiarare variabili con visibilità solo all'interno del ciclo:

```
for (int i = 0; i < 10; i++) {
    …. // i qui è definita
}
…. // i qui non è definita
```

  - in confronto a:

```
for (i = 0; i < 10; i++) {
    …. // i qui è definita
}
…. // anche qui i  è definita
```


Istruzioni condizionali: switch

- Simile al C:
  - case con argomenti interi
  - istruzione break per terminare ogni clausola
  - clausola default

```
switch (n) {
    case 1:    …..
               break;
    case 2:    …..
               break;
    …..
    default:   …..
               break;
}
```


Istruzioni per uscire da un flusso

- In C/C++ era presente una istruzione goto
  - permetteva di passare ad eseguire una istruzione in qualsiasi posto nel programma
  - pratica assolutamente sconsigliata
    - *spaghetti programming*
- In Java viene ampliata la usuale istruzione break:

```
for (int i = 0; i < 10; i++) {
    …..
    if (…) break; // esce dal ciclo

}
```


Istruzione break con etichetta

- Possibile etichettare un ciclo:

```
primociclo:
for (int i = 0; i < 10; i++) {
    ….
}
```

- Possibile selezionare con break il ciclo da cui si vuole uscire:

```
…..
    ….
    if (…) break etichettaciclo;
    ….
```

Un esempio


```
….
int n;
read_data:
while (….)  {
     ….
     for (…)  {
          ….
          if (n < 0) break read_data;
          ….
     }
     ….
}
… // istruzione eseguita se n < 0
```

- Ciclo etichettato
  - etichetta
  - ciclo interno
- Salto con etichetta
  - Istruzione eseguita se n risulta minore di 0
- Se avessimo usato un break *anonimo:*
  - istruzione eseguita

Istruzioni per continuare un ciclo: continue

- Come in C/C++

```
for (int i = 0; i < 10; i++) {
     …..
     if (…) continue; // non esegue sum+=n ma continua il ciclo
     sum += i;
}
```

- Un commento su break e continue
  - non strettamente necessarie
  - possibile ed, in un certo senso, auspicabile il non uso

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Il problema della precisione nei calcoli

- Per applicazioni numeriche serve:
  - poter gestire numeri (interi o decimali) con precisione arbitraria
- Java mette a disposizione nella classe Java.math
  - una classe BigInteger
  - una classe BigDecimal
- Le classi hanno metodi per:
  - convertire un numero in versione *Big*
  - effettuare operazioni (non si può usare + * / etc.!!)

La classe BigInteger

- Metodi:
  - per trasformare un intero in BigInteger

```
BigInteger a = BigInteger.valueOf(100);
```

    - notare il "tipo" della variabile a…. è un oggetto!
  - Operazioni:

```
BigInteger a = BigInteger.valueOf(100);
BigInteger b = BigInteger.valueOf(56);
BigInteger c = BigInteger.valueOf(31);
BigInteger d = BigInteger.valueOf(43);
BigInteger e = a.multiply(b);      // e = a * b
BigInteger f = c.add(d);           //  f = c + d
BigInteger g = a.subtract(c);      // g = a - c
BigInteger h = b.divide(d);        // h = b / d
```


"Sneak preview" di metodi di classi

- C'è una differenza tra:

```
BigInteger a = BigInteger.valueOf(100);
```

```
BigInteger a = BigInteger.valueOf(100);
BigInteger b = BigInteger.valueOf(56);
BigInteger e = a.multiply(b);      // e = a * b
```

  - Il primo è un metodo relativo alla classe:
    - metodo statico
      - quanto di più simile esiste in Java alle funzioni
  - mentre il secondo è un metodo dell'oggetto a
    - tipico esempio di programmazione ad oggetti

Organizzazione della lezione

- Anatomia di un programma Java
- Commenti
- Tipi di dati
- Variabili, assegnazioni e inizializzazioni
- Operatori
- Stringhe
- Controllo del flusso
- Gestione di numeri grandi
- Array

Array

- Struttura dati omogenea ed ordinata
  - accesso ai singoli valori con la specifica di un indice
- Rispetto al C:
  - utilizzo identico, ma dichiarazione diversa.
- Dichiarazione di un array di interi
  - dichiarazione variabile:

```
int [] a;
```

  - creazione array

```
a = new int [100];
```

Array (2)

- Possibile anche:
  - in un unico passo
  - dichiarazione “a la” C:
- Lunghezza di un array:
- La dimensione di un array non si può modificare:
  - i suoi elementi ovviamente si
- Inizializzazione contestualmente alla dichiarazione

```
int [ ] a = new int [100];
```

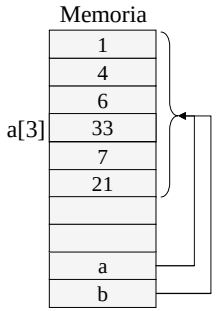
```
int a[ ];
```

```
for (int i=0; i < a.length; i++)
```

```
int [ ] a = { 1, 2, 3, 5, 10, 21 } ;
```


Copia di un array

- Gli Array in Java sono oggetti:
  - stesso trattamento di variabili di riferimento
- Copiando una variabile array in un’altra:
  - entrambe faranno *riferimento* allo *stesso* array

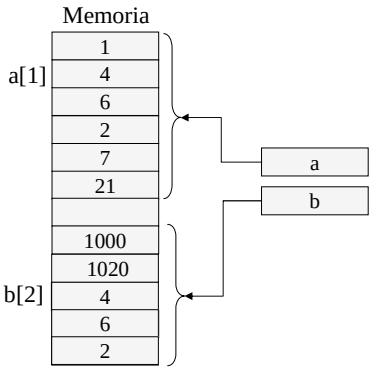



```
int [ ] a = { 1, 4, 6, 2, 7, 21 } ;
int [ ] b = a;

b[3] = 33;
```


Copia di un array (2)

- Utilizzo di un metodo della classe System
  - copia da from a to, partendo dalla posizione fromIndex per count posizioni

```
System.arraycopy ( from, fromIndex, to, toIndex, count)
```




```
int [ ] a = { 1, 4, 6, 2, 7, 21 } ;
int [ ] b = {1000, 1020, 1024, 1050, 1088};

System.arraycopy (a, 1, b, 2, 3);
```


Parametri su linea di comando

- Ogni programma Java accetta parametri sulla linea di comando (simile al C)
  - parametri di main: array di stringhe
- Una differenza:
  - args [0] è il primo parametro non il nome del programma

```
public class ReverseEcho {
    public static void main (String[ ] args) {
        for (int i = args.length - 1; i >= 0; i --)
            System.out.println (args[i]);
    }
}
```

Array multidimensionali

- Dichiarazione simile al caso monodimensionale:

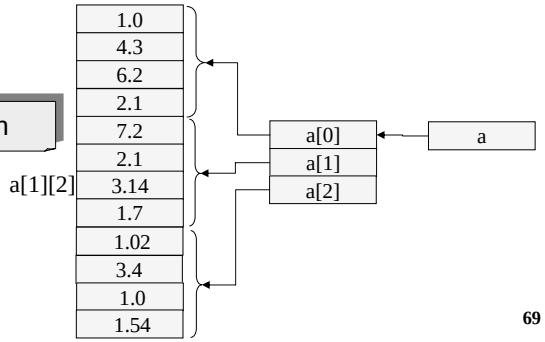
```
double [ ] [ ] a ;
a = new double [3][4];
```

- Memorizzazione e gestione simile al C:
  - array di puntatori
  - uso:
    - lunghezza:

```
… a[0].length
```

    - swap:

```
double[] temp;
temp = a[ i ];
a[ i ] = a[ j ];
a[ j ] = temp;
```



Array frastagliati

```
public class ProvaFrastagliati {
    public static void main (String [ ] args) {
        int [ ] [ ]  a = new int [10] [ ];
        int contatore = 0;
        for (int i = 0; i <  a.length; i++) {
            a[i] = new int [i+1];
            for (int j = 0; j < a[i].length; j++)
                a[ i ][ j ] = contatore++;
        }
        for (int i = 0; i <  a.length; i++) {
            for (int j = 0; j < a[i].length; j++)
                System.out.print (a[ i ][ j ]);
            System.out.println();
        }
    }
}
```

- “righe” di diversa dimensione
  - possibile per il tipo di memorizzazione
- Allocazione
  - 10 array di interi
  - ognuno di dimensione crescente
  - output

L’Array frastagliato creato dal programma

a[0]	0									
a[1]	1	2								
a[2]	3	4	5							
a[3]	6	7	8	9						
a[4]	10	11	12	13	14					
a[5]	15	16	17	18	19	20				
a[6]	21	22	23	24	25	26	27			
a[7]	28	29	30	31	32	33	34	35		
a[8]	36	37	38	39	40	41	42	43	44	
a[9]	45	46	47	48	49	50	51	52	53	54

Array frastagliati: la memorizzazione

```
public class ProvaFrastagliati {
    public static void main (String [ ] args) {
        int [ ] [ ]  a = new int [10] [ ];
        int contatore = 0;
        for (int i = 0; i <  a.length; i++) {
            a[i] = new int [i+1];
            for (int j = 0; j < a[i].length; j++)
                a[ i ][ j ] = contatore++;
        }
        for (int i = 0; i <  a.length; i++) {
            for (int j = 0; j < a[i].length; j++)
                System.out.print (a[ i ][ j ]);
            System.out.println();
        }
    }
}
```