

Dati, variabili e operatori

Lezione VI

Scopo della lezione

- Ripassare i concetti di dato e di variabile
- Introdurre e approfondire i concetti di
 - Tipo di dato
 - Espressione
 - Assegnamento
 - Operatori
- Introdurre i concetti di
 - Promozione e casting

Dati e variabili

- In alcuni casi un programma si riferisce ad alcuni valori che deve poter memorizzare e/o modificare, eventualmente in tempi successivi
- Per potersi riferire a questi valori il programma alloca delle aree di memoria a cui assegna nomi univoci
- Chiamiamo queste aree **variabili**, e chiamiamo **dati** i valori in esse contenuti

Variabili e tipi

- La memorizzazione di una variabile è legata al **tipo** dei dati che essa contiene
- Distinguiamo i seguenti tipi di dati
 - Numerici
 - Interi (`byte`, `int`, `long`)
 - A virgola mobile (`float`, `double`)
 - Alfanumerici
 - Carattere (`char`)
 - Stringa (`String`)
 - Booleani (`boolean`)

Tipi di dati

Tipo	Specificazione di un valore
byte	Un valore intero
int	
long	
float	Un valore decimale
double	
boolean	Un valore di verità
char	Un carattere

Definizione di variabili

- Prima di poter essere utilizzata, una variabile deve essere **definita** (unitamente al tipo corrispondente)
- La definizione di una variabile avviene dichiarando
 - Il tipo della variabile
 - Il nome della variabile**in questo specifico ordine**, terminando la dichiarazione con un punto e virgola

Nomi di variabili

- Il nome di una variabile è composto dai seguenti simboli
 - Lettere (maiuscole e minuscole) dell'alfabeto
 - Numeri
 - Caratteri speciali (_)

indicati in un ordine qualsiasi e iniziando con un simbolo che non sia un numero

In particolare

- Una lettera minuscola è considerata differente dalla corrispondente lettera maiuscola (i nomi delle variabili sono **case sensitive**)
- Non è possibile utilizzare spazi all'interno del nome di una variabile
- Non è possibile utilizzare come nome di variabile una qualunque parola chiave del linguaggio

Esempi

- Sono nomi validi di variabili
 - Nome, Indirizzo1, STATO, eta, stato_civile, statoCivile, StAtOcIvIlE, jkl2df13231, _stato, ...
- Non sono nomi validi di variabili
 - 1Nome, stato civile, ...

Convenzioni

- E' possibile utilizzare più parole per comporre il nome di una variabile
 - Concatenando le singole parole (`statocivile`)
 - Concatenando le singole parole scritte in caratteri minuscoli e scrivendo in maiuscolo tutte le iniziali delle parole a partire dalla seconda (`statoCivile`)
 - Separando le singole parole con il carattere di underscore (`stato_civile`)

Convenzioni (2)

- Pur non essendoci vincoli formali sull'uso di lettere maiuscole e minuscole, è sconsigliato
 - Cominciare il nome di una variabile con una lettera maiuscola
 - Utilizzare esclusivamente lettere maiuscole (ed eventualmente caratteri di underscore) per comporre il nome di una variabile

Esempio

```
/* Definiamo delle variabili */  
  
class Variabili {  
    public static void main(String args[]) {  
        int varIntera;  
        float varAVirgolaMobile;  
        byte varInteraByte;  
        double varADoppiaPrecisione;  
        char varCarattere;  
        boolean varBooleana;  
    }  
}
```

Definizione multipla

- E' possibile definire più variabili di uno stesso tipo simultaneamente, dichiarando
 - Il tipo delle variabili
 - I nomi delle variabili separati da virgole**in questo specifico ordine**, terminando la dichiarazione con un punto e virgola
- Esempio
 - `int eta, anni;`

Assegnamento

- La più semplice operazione riguardante l'uso di variabili è rappresentata dalla memorizzazione di uno specifico valore in una data variabile
- Tale operazione, detta di assegnamento, viene descritta indicando, nell'ordine
 - Il nome della variabile
 - Il simbolo di uguale (=)
 - Il valore

Specificare un valore

Tipo	Specificazione di un valore
<code>byte</code>	Un valore intero
<code>int</code>	Un valore intero
<code>long</code>	Un valore intero
<code>float</code>	Un valore decimale seguito da <code>f</code>
<code>double</code>	Un valore decimale
<code>boolean</code>	Uno tra i valori <code>true</code> e <code>false</code>
<code>char</code>	Un carattere delimitato da <code>`</code>

Variabili intere

- I tipi `byte`, `short`, `int` e `long` contengono variabili a valori numerici interi codificati in complemento a due
- La loro differenza sta nel numero di bit utilizzati per effettuare la memorizzazione
- Tale numero di bit è fissato nella JVM e quindi è indipendente dal particolare tipo di computer utilizzato

Tipi interi

Tipo	Descrizione	Range
<code>byte</code>	intero con segno a 8 bit	$(-2^7, 2^7-1]$
<code>short</code>	intero con segno a 16 bit	$(-2^{15}, 2^{15}-1]$
<code>int</code>	intero con segno a 24 bit	$(-2^{23}, 2^{23}-1]$
<code>long</code>	intero con segno a 32 bit	$(-2^{63}, 2^{63}-1]$

Variabili decimali

- I tipi `float` e `double` contengono variabili a valori numerici decimali codificati secondo lo standard IEEE 754
- La loro differenza sta nel numero di bit utilizzati per effettuare la memorizzazione
- Tale numero di bit è fissato nella JVM e quindi è indipendente dal particolare tipo di computer utilizzato

Tipi decimali

Tipo	Descrizione
float	decimale a 32 bit (singola precisione)
double	decimale a 64 bit (singola precisione)

Variabili carattere

- Il tipo `char` contiene un carattere codificato secondo lo standard unicode a 16 bit

Variabili booleane

- Il tipo `boolean` contiene uno dei valori di `true` o `false`
- `true` e `false` rappresentano i valori di verità associati a un enunciato logico

Esempio

```
varIntera=2;  
varAVirgolaMobile=3.2f;  
varInteraByte=120;  
varADoppiaPrecisione=5.221;  
varCarattere='2';  
varBooleana=true;
```

Attenzione!

- L'assegnamento di un valore a una variabile può essere fatto solamente dopo che questa è stata definita (pena un errore in fase di compilazione)

```
main() {  
    int a;  
    a=1;  
}
```

Corretto

```
main() {  
    a=1;  
}
```

Errato

Definizione e assegnamento

- E' possibile assegnare un valore a una variabile contestualmente alla loro definizione, dichiarando
 - Il tipo della variabile
 - Il nome della variabile
 - Il simbolo =
 - Il valore per la variabile**in questo specifico ordine**, terminando la dichiarazione con un punto e virgola

```
int a=1;
```

Definizione e assegnamento

- In caso di definizione multipla è possibile assegnare valori alle variabili, dichiarando
 - Il tipo delle variabili
 - Un elenco separato da virgole, contenente per ogni variabile
 - Il nome della variabile
 - Il simbolo =
 - Il valore per la variabile

```
int a=1,b=4;
```

in questo specifico ordine, terminando la dichiarazione con un punto e virgola

Inizializzazione

- Il primo assegnamento fatto a una variabile dopo che questa è stata definita viene chiamato **inizializzazione** della variabile
- Se si tenta di accedere al contenuto di una variabile prima che questa venga inizializzata il compilatore emette un errore

Esempio

```
/* Questo programma non verrà compilato */
import prog.io.*;
public class MancataInizializzazione {
    public static void main(String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int varNumerica;
        video.println(varNumerica);
    }
}
malchiod% javac MancataInizializzazione.java
MancataInizializzazione.java:6: variable varNumerica
    might not have been initialized
        System.out.println(varNumerica);
                           ^
1 error
```

Assegnamento

- E' possibile assegnare a una variabile
 - Un valore predeterminato
 - Il valore di un'altra variabile
 - Il valore di un'espressione, eventualmente contenente una o più variabili

Visualizzazione

- Dando come argomento a `print` o `println` il nome di una variabile, il suo valore viene visualizzato

Operatori

- Un operatore è una funzione tra tipi
- Viene detto unario quando associa un tipo ad un tipo, e binario quando associa un tipo a due tipi
- Ad esempio la somma è un operatore binario, mentre la negativizzazione è un operatore unario

Operatori binari

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Modulo
==	Uguaglianza
!=	Differenza
&&	Congiunzione
	Disgiunzione

Operatori unari

-	Negazione matematica
!	Negazione logica

Autoincremento/decremento

- L'operatore di autoincremento ($++$) aggiunge 1 al valore della variabile cui è applicato, e vi memorizza il risultato
- L'operatore di autodecremento ($--$) sottrae 1 al valore della variabile cui è applicato, e vi memorizza il risultato
- Gli operatori di autoincremento e autodecremento possono essere usati in modo prefisso o postfisso

Esempio

```
import prog.io.*;
public class IncrementoDecremento {
    public static void main(String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int a=2;    video.println(a);
        a++;       video.println(a);
        a--;       video.println(a);
    }
}
```

```
malchiod% java IncrementoDecremento
```

```
2
```

```
3
```

```
2
```

Autoincremento/decremento

- Quando l'operatore è usato in modo **postfisso** (indicandolo dopo il nome della variabile), il valore della variabile viene utilizzato per valutare l'espressione, poi la variabile viene incrementata o decrementata

Esempio

```
import prog.io.*;
public class Postfisso {
    public static void main(String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int a=2;
        video.println(a++ * 2);
        video.println(a);
    }
}
```

```
malchiod% java Postfisso
```

```
4
```

```
3
```

Autoincremento/decremento

- Quando l'operatore è usato in modo **prefisso** (indicandolo prima del nome della variabile), la variabile viene incrementata o decrementata e poi il nuovo valore della variabile viene utilizzato per valutare l'espressione

Esempio

```
import prog.io.*;
public class Prefisso {
    public static void main(String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int a=2;
        video.println(--a * 2);
        video.println(a);
    }
}
malchiod% java Prefisso
2
1
```

Divisione intera e modulo

- Quando l'operatore $/$ è applicato a due valori interi, il risultato dell'operazione è il quoziente (intero) tra i due valori
- L'operatore di modulo ($\%$) ritorna il resto della divisione intera

Divisione intera e modulo

a	b	a/b	a%b
+7	+4	+1	+3
-7	+4	-1	-3
+7	-4	-1	+3
-7	-4	+1	-3

Uguaglianza e differenza

- L'operatore `==` ritorna `true` se le due espressioni usate come operandi hanno lo stesso valore e `false` altrimenti
- L'operatore `!=` ritorna `true` se le due espressioni usate come operandi hanno valori diversi e `false` altrimenti

Assegnamento

- E' possibile assegnare a una variabile il valore di un'espressione in cui compare la stessa variabile
- In questo caso il valore corrente della variabile viene utilizzato per valutare l'espressione, il cui risultato viene poi memorizzato nella variabile

Esempio

```
import prog.io.*;
class Addizione {
    public static void main(String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int a=2, b=1;
        a=a+b;
        video.println(a);
        a=a+b;
        video.println(a);
    }
}
```

Operatori e assegnamento

- E' possibile unire il risultato di una qualunque operazione binaria (ad eccezione di ==) con quello di un assegnamento a una delle variabili coinvolte nell'operazione
- Questa particolare operazione viene realizzata facendo seguire il simbolo dell'operatore dal simbolo =

Operatori e assegnamento

L'istruzione...	...diventa
<code>a=a+b;</code>	<code>a+=b;</code>
<code>a=a-b;</code>	<code>a-=b;</code>
<code>a=a*b;</code>	<code>a*=b;</code>
<code>a=a/b;</code>	<code>a/=b;</code>
<code>a=a%b;</code>	<code>a%=b;</code>

Precedenza

()	Parentesi
!, ++, --	Negazione, incremento e decremento prefisso
++, --	Incremento e decremento postfisso
-	Negazione unaria
*, /, %	Moltiplicazione, divisione, modulo
+, -	Addizione, sottrazione
==, !=	Uguaglianza, diversità
&&	Congiunzione logica
	Disgiunzione logica
? :	Operatore condizionale
=, +=, -=, *=, /=, %=	Assegnamenti

NaN

- La JVM ritorna NaN ogni volta che l'esecuzione di un'operazione matematica incorre in un risultato indefinibile

Esempio

```
import prog.io.*;
public class NotANumber {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager ();
        double zero=0;
        video.println(zero/zero);
    }
}
```

```
malchiod% javac NotANumber.java
```

```
malchiod% java NotANumber
```

```
NaN
```

```
malchiod%
```

Infinity

- Quando l'esito di un'operazione che riguarda valori a virgola mobile risulta maggiore del più alto valore memorizzabile, la JVM ritorna `Infinity`
- `Infinity` si comporta in modo matematicamente consistente (nel senso che viene interpretato come limite di una successione)

Esempio

```
import prog.io.*;
public class Infinity {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        double numero;
        numero=0;          video.println(numero);
        numero=1/numero;  video.println(numero);
        numero=1/numero;  video.println(numero);
        numero=1/numero;  numero=numero-numero;
        video.println(numero);
    }
}
```

Esempio

```
malchiod% java Infinity  
0.0  
Infinity  
0.0  
NaN
```

Conversione tra tipi

- In alcuni casi è necessario effettuare la conversione di una variabile da un tipo all'altro

Promozione

- Se il tipo a cui si converte ha una capacità di memorizzazione maggiore di quella del tipo da cui si converte, il processo è chiamato **promozione**
- Una promozione è un processo che non comporta mai perdita di dati, e quindi il compilatore Java la effettua in modo automatico

Esempio

```
import prog.io.*;
public class Promozione {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int varPartenza;
        long varArrivo;
        varPartenza = 200;
        video.println(varPartenza);
        varArrivo = varPartenza;
        video.println(varArrivo);
    }
}
```

Casting

- Se il tipo a cui si converte ha una capacità di memorizzazione minore di quella del tipo da cui si converte, il processo è chiamato **casting**
- Il casting è un processo che può comportare perdita di dati, e quindi il compilatore Java richiede che esso venga dichiarato esplicitamente

Casting

- Per indicare al compilatore che si vuole effettuare un casting è necessario far precedere il valore che deve essere convertito dal tipo in cui si desidera effettuare la conversione, racchiuso tra parentesi
- E' possibile utilizzare la stessa tecnica per indicare esplicitamente una promozione

Esempio

```
import prog.io.*;
public class Casting {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        long varPartenza;
        int varArrivo;
        varPartenza = 200;
        video.println(varPartenza);
        varArrivo = (int)varPartenza;
        video.println(varArrivo);
    }
}
```

Conversione

- In alcuni casi non è possibile effettuare una conversione tra tipi, né ricorrendo alla promozione e neppure ricorrendo al casting

Conversione

da \ a	byte	int	long	float	double	char	boolean	String
byte		promoz.	promoz.	promoz.	promoz.	casting	inconv.	inconv.
int	casting		promoz.	promoz.	promoz.	casting	inconv.	inconv.
long	casting	casting		promoz.	promoz.	casting	inconv.	inconv.
float	casting	casting	casting		promoz.	casting	inconv.	inconv.
double	casting	casting	casting	casting		casting	inconv.	inconv.
char	casting	promoz.	promoz.	promoz.	promoz.		inconv.	inconv.
boolean	inconv.	inconv.	inconv.	inconv.	inconv.	inconv.		inconv.
String	inconv.	inconv.	inconv.	inconv.	inconv.	inconv.	inconv.	

Provate!

```
import prog.io.*;
public class Conversione {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        tipoPartenza varPartenza;
        tipoArrivo varArrivo;
        varPartenza = valore;
        video.println(varPartenza);
        varArrivo = (tipoArrivo)varPartenza;
        video.println(varArrivo);
    }
}
```

- Modificate **tipoPartenza**, **tipoArrivo** e **valore** e guardate che succede

Attenzione!

- Quando durante un'operazione di casting un valore non può essere memorizzato nel tipo di destinazione il risultato dell'operazione è indeterminato (ma sicuramente errato!)
- In particolare l'operazione di casting può causare perdita di informazione

Esempio

```
import prog.io.*;
public class PerditaDati {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int varPartenza;
        byte varArrivo;
        varPartenza = 200;
        video.println(varPartenza);
        varArrivo = (byte)varPartenza;
        video.println(varArrivo);
        varPartenza = varArrivo;
        video.println(varPartenza);
    }
}
```

Esempio

```
malchiod% javac PerditaDati.java
malchiod% java PerditaDati
200
-56
-56
```

Valori memorizzabili

- Ogni tipo fondamentale ha associato un particolare oggetto, detto **type-wrapper**, avente nome simile a quello del tipo e con l'iniziale maiuscola
- Tramite questo oggetto è possibile accedere alle voci `MIN_VALUE` e `MAX_VALUE`, contenenti rispettivamente il minimo e il massimo valore memorizzabile

Overflow

- Quando si tenta di assegnare a una variabile un valore maggiore del più alto valore memorizzabile (o minore del più basso valore memorizzabile) si verifica una condizione chiamata **overflow**

Overflow in Java

- Quando si verifica un overflow in una variabile di tipo intero Java non segnala alcun errore e il risultato dell'assegnamento è indeterminato
- Quando si verifica un overflow in una variabile di tipo decimale Java ritorna il valore speciale `Infinity`

Esempio

```
import prog.io.*;
public class ValoriMassimi {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int variabile;
        variabile = Integer.MAX_VALUE;
        video.println(variabile);
        variabile++;
        video.println(variabile);
        variabile = Integer.MIN_VALUE;
        video.println(variabile);
        variabile--;
        video.println(variabile);
    }
}
```

Esempio

```
malchiod% javac ValoriMassimi.java
malchiod% java ValoriMassimi
2147483647
-2147483648
-2147483648
2147483647
malchiod%
```

Overflow e underflow

- Relativamente a valori di tipo decimale, `MAX_VALUE` e `MIN_VALUE` si riferiscono rispettivamente al massimo e al minimo valore positivo memorizzabile
- In questo caso, quando si tenta di memorizzare un valore positivo minore di `MIN_VALUE` si incorre in una situazione di underflow, e il risultato è approssimato con 0

Overflow e Underflow

```
import prog.io.*;
public class Infinito {
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        float variabile;
        variabile = Float.MAX_VALUE;
        video.println(variabile);
        variabile*=2;
        video.println(variabile);
        variabile = Float.MIN_VALUE;
        video.println(variabile);
        variabile/=10;
        video.println(variabile);
    }
}
```

Overflow e Underflow

```
malchiod% java Infinito  
3.4028235E38  
Infinity  
1.4E-45  
0.0  
malchiod%
```

Costanti

- Una **costante** è un'area di memoria contenente un valore che non viene modificato durante l'esecuzione del programma
- Una costante si dichiara esattamente come una variabile, facendo precedere le parole chiave `final` `static` alla dichiarazione

Costanti

- Una volta assegnato un valore a una costante non è possibile modificarlo, pena un errore di compilazione
- Per convenzione i nomi di variabili sono scritti usando esclusivamente caratteri maiuscoli

Esempio

```
import prog.io.*;
public class Costante {
    static final float PIGRECO =3.1412f;
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int r=2;
        video.print("L'area di un cerchio ");
        video.print("di raggio 2 e' ");
        video.println(PIGRECO*r*r);
    }
}
malchiod% java Costante
L'area di un cerchio di raggio 2 e' 12.5648
```

Esempio

```
import prog.io.*;
public class CostanteModificata {
    static final float PIGRECO =3.1412f;
    public static void main (String args[]) {
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        int r=2;
        video.print("L'area di un cerchio ");
        video.print("di raggio 2 e' ");
        video.println(PIGRECO*r*r);
        PIGRECO++;
    }
}
```

Esempio

```
malchiod% javac CostanteModificata.java
CostanteModificata.java:8: cannot assign a
    value to final variable PIGRECO
        PIGRECO++;
        ^
1 error
malchiod%
```

Esercizio

- Scrivere un programma che legge in input la descrizione di un'equazione di primo grado espressa nella forma

$$a x + b = 0$$

e stampi il relativo risultato.

Soluzione

```
import prog.io.*;
public class PrimoGrado {
    public static void main(String args[]) {
        double a,b;
        ConsoleOutputManager video
            = new ConsoleOutputManager();
        ConsoleInputManager tastiera
            = new ConsoleInputManager();
        a = tastiera.readDouble("Inserisci il coeff a ");
        b = tastiera.readDouble("Inserisci il coeff b ");
        video.println("Il risultato e' "+(-b/a));
    }
}
```

Esecuzione

```
malchiod% java PrimoGrado
Inserisci il coefficiente a 4
Inserisci il coefficiente b 2
Il risultato e' -0.5
malchiod%
```

Esercizio

- Scrivere un programma che legge in input la descrizione di un'equazione di secondo grado espressa nella forma

$$a x^2 + b x + c = 0$$

e stampi i relativi risultati.

Soluzione

```
import prog.io.*;
public class SecondoGrado {
    public static void main(String args[]) {
        double a,b,c,radDelta,ris1,ris2;
        ConsoleOutputManager video=new ConsoleOutputManager();
        ConsoleInputManager tastiera=new ConsoleInputManager();
        a = tastiera.readDouble("Inserisci il coeff. a ");
        b = tastiera.readDouble("Inserisci il coeff. b ");
        c = tastiera.readDouble("Inserisci il coeff. c ");

        radDelta = Math.sqrt(b*b-4*a*c);
        ris1 = (-b+radDelta)/(2*a);
        ris2 = (-b-radDelta)/(2*a);
        video.println("Il primo risultato e' "+ris1);
        video.println("Il secondo risultato e' "+ris2);
    }
}
```

Esecuzione

```
malchiod% java SecondoGrado
Inserisci il coefficiente a 1
Inserisci il coefficiente b 4
Inserisci il coefficiente c -1
Il primo risultato e' 0.2360679774997898
Il secondo risultato e' -4.23606797749979
malchiod% java SecondoGrado
Inserisci il coefficiente a 1
Inserisci il coefficiente b 0
Inserisci il coefficiente c -4
Il primo risultato e' 2.0
Il secondo risultato e' -2.0
malchiod%
```

Esercizio

- Scrivere un programma che legga un numero decimale e lo tronchi alla seconda cifra dopo il punto.

Soluzione

```
import prog.io.*;
public class Arrotonda {
    public static void main(String args[]) {
        double originale, troncato;
        int appoggio;
        ConsoleOutputManager video=new ConsoleOutputManager();
        ConsoleInputManager tastiera=new ConsoleInputManager();
        originale = tastiera.readDouble("Inserisci un val ");
        appoggio = (int)(originale*100);
        troncato = (double)appoggio/100;
        video.println("Il valore troncato e' "+troncato);
    }
}
```

Esecuzione

```
malchiod% java Arrotonda  
Inserisci un valore 3.14152  
Il valore troncato e' 3.14  
malchiod%
```