Tipi di dati fondamentali

La nozione di tipo di dato

- Il tipo del dato consente di esprimere la natura del dato
- Indica il modo con cui verrà interpretata la sequenza di bit che rappresenta il dato
  - La stessa sequenza può rappresentare un intero o un carattere ad esempio
- Determina il campo dei valori che un dato può assumere
- Specifica le operazioni possibili sui dati

Tipi di dati e Java

- Java è un linguaggio fortemente tipizzato
- Il tipo di ogni variabile o espressione può essere identificato leggendo il programma ed è già noto al momento della compilazione
  - È obbligatorio dichiarare il tipo di una variabile prima di utilizzarla
  - Durante la compilazione sono effettuati tutti i controlli relativi alla compatibilità dei tipi e sono stabilite eventuali conversioni implicite.
- Dopo la dichiarazione non è possibile assegnare alla variabile valori di tipo diverso
  - Salvo casi particolari che vedremo in seguito

Tipi primitivi: interi

- Java fornisce otto tipi primitivi indipendenti dall'implementazione e dalla piattaforma
- **Interi**
  - Tipo **`byte`** **(8 bit)**
    - Interi con segno tra -128 e 127, valore di default 0
  - Tipo **`short`** **(16 bit)**
    - Interi con segno tra -32768 e 32767, valore di default 0
  - Tipo **`int`** **(32 bit)**
    - Interi con segno tra -2^{31} e $2^{31}-1$, valore di default 0
  - Tipo **`long`** **(64 bit)**
    - Interi con segno tra -2^{63} e $2^{63}-1$, valore di default 0

Tipi primitivi: costanti intere

- Una costante intera per default è di tipo **`int`**
- Costanti intere possono essere espresse anche in ottale (prefisso 0) o in esadecimale (prefisso 0x)
- Per costanti intere di tipo **`long`** aggiungere il suffisso **L** oppure **l**
  - Es. 4000L
- Costanti intere di tipo **`byte`** (risp. **`short`**)
  - costanti di tipo **`int`** il cui valore rientra nel range del tipo **`byte`** (risp. **`short`**)

Tipi primitivi: numeri con virgola

- Seguono standard IEEE 754
- Tipo **`float`** **(32 bit)**
  - numeri in virgola mobile con 7 cifre significative
  - compresi tra `1.4E-45` e `3.4028235E+38`
  - valore di default 0.0
  - le costanti vanno terminate con **F** o **f**

    es. `float a=3.456F;`
- Tipo **`double`** **(64 bit)**
  - numeri in virgola mobile in doppia precisione (15 cifre significative)
  - compresi tra `4.9E-324` e `1.79769313486231570E+308`
  - valore di default `0.0`
  - le costanti con virgola sono di tipo **`double`** per default
  - possono essere terminate con **D** o **d** ma non è necessario

Tipi Primitivi: caratteri

- Seguono la codifica Unicode che estende ASCII su 16 bit
- Tipo **`char`** **(16 bit)**
  - Si possono usare i caratteri o i relativi codici numerici preceduti da **`\u`**, sempre tra apici. Es.
    - `char a='A';`
    - `char a='\u0041'; //` (in esadecimale su 4 cifre)
    - `char a=65;`
  - Caratteri con codici tra 0 e 65535
  - valore di default '\u0000' ('\0' del C)
  - Possibilità di usare \ per caratteri particolari ('\n', '\t', '\"', '\b', '\0', …)

Tipi Primitivi: boolean

- Tipo **`boolean` (1 bit)**
- Ammette solo due possibili valori (**`true`**, **`false`**)
- Valore di default **`false`**
- Non si possono assegnare interi alle variabili booleane
  - **`false`** non è 0!!!

Operatori per i Tipi Primitivi (1)

- Java ha gli stessi operatori del C, con qualche leggera differenza
- (**+**, **-**, *****, **/**, **%**, **++**, **--**, **+=**, **-=**, ***=**, **/=**, **%=**)
  - Non sono applicabili a variabili di tipo **`boolean`**
- Relazionali (**<**, **>**, **<=**, **>=**, **==**, **!=**)
  - Producono risultati di tipo **`boolean`** (**`true`**, **`false`**)
  - **<**, **>**, **<=**, **>=** non sono applicabili a variabili di tipo **`boolean`**

Operatori per i Tipi Primitivi (2)

- Logici (**&&, ||, !**)
  - && and || non valutano espressione destra se valore della condizione può essere stabilita dall'espressione sinistra (valutazione abbreviata)

- Bit a bit (solo tipi interi e char)
  - **&** (AND), **|** (OR), **^** (XOR), **~** (complemento bit a bit)
  - Shift **<<** , **>>** (rispetta segno operando), **>>>** (mette 0 come bit più significativo)
    - Es. x << n sposta bit di x di n posizioni a sinistra e riempe i posti lasciati liberi con 0
  - Combinati assegnamento: **&=, |=, ^=, <<=, >>=, >>>=**
  - **~, <<, >>, >>>, <<=, >>=, >>>=** non si applicano a variabili boolean

Priorità e associatività

Operatori	Associatività
[] ( ) . ++(postfisso) --(postfisso)	da sinistra a destra
! ~ ++(prefisso) --(prefisso) +(unario) -(unario)	da destra a sinistra
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 destra a sinistra
= += -= *= /= %= &= |= ^= <<= >>= >>>=	da destra a sinistra

Tipi delle espressioni

- Il tipo delle espressioni con operatori aritmetici su interi (ad eccezione degli shift) è **`int`** a meno che un operatore è **`long`** (in questo caso è **`long`**)
- Per gli operatori di shift non si tiene conto del tipo dell'operando destro
- Se è presente un operando in virgola mobile il tipo è **`float`** a meno che uno degli operandi sia **`double`** (e in questo caso è **`double`**)

Conversione implicita di tipo

- Ampliamento (da più piccolo a più grande):
  - byte → short → int → long → float → double
  - char → int → long → float → double
- Conversione da long a float
  - possibile in quanto il range di **`float`** è più ampio del range di **`long`**
  - perdita di precisione (da 64 a 32 bit)
- Restringimento (da più grande a più piccolo):
  - ammesso negli assegnamenti di **costanti** di tipo **`int`** a tipo **`short`**, **`byte`** o **`char`** a patto che il valore della costante possa essere contenuto nel tipo di destinazione

Esempi conversioni di tipo

```
int a=1000L;
    // Errore: tentativo di assegnare long a int (anche se 1000 in int ci va)
short s=700;
byte b=-70;
int x=s+b;
    // Ok: short e byte sono tipi più piccoli; converte tutto a int come in C
double d=700.23;
float c=-70F;
double x=d+c;
    // Ok: converte tutto a double come in C
float y=d+c;
    // Errore: converte c a double e tenta di assegnare double a float
```

Virgola mobile e Interi

- I tipi decimali accettano qualunque tipo di espressione intera, con eventuale arrotondamento sulle cifre meno significative, ma a nessun tipo di intero si possono assegnare espressioni in virgola.

- **Esempi**

```
float f=1234567L; // OK: diventa 1234567.0
float d=12345678; // OK: diventa 1.2345678E7
float c=-123456782; // OK: arrotondamento sulle
                                    ultime cifre
long x=f; // Errore: tenta di assegnare float a long
```

Char e Interi

- Ai tipi interi **long** e **int** si possono assegnare espressioni **char** che verranno convertite nel relativo codice numerico, a **byte** e **short** non si possono assegnare **char** e a **char** non si può assegnare nessun intero.

- **Esempi**

```
char c='B';
int d=44+c; // OK: c viene convertito a int e d vale 110
char s=d;   // Errore: assegna int a char (anche se 110 è un
            // valore possibile per char)
char x=110; // Ok: 110 è un valore possibile per un char
char t=-7;  // Errore: -7 non è un valore possibile per un char
```

Ancora sulle Conversioni

- Ai tipi **`float`** e **`double`** si possono sempre assegnare espressioni **`char`** mentre il contrario non è mai possibile.

- **Esempi**

```
char c='B';
float d=44+c;  // OK: c e 44 vengono convertiti a float e d vale 110.0
```

- **IMPORTANTE:** Non sono possibili conversioni di tipo da/verso **`boolean`**

Casting sui Tipi Primitivi

- Un cast esplicito può servire a forzare le conversioni che in Java non sono permesse. La sintassi è uguale a quella del C.

- **Esempi**

```
double d=-1.8345678901234567;
float f=(float)d; // perdita di precisione
int i=(int)d;     // i vale -1 (non c'è arrotondamento)
short s=-700;
char c=(char)s;   // possibile, ma senza senso
boolean b=(boolean)i; // Errore: non sono permessi cast
                      da/verso boolean
```

Tabella delle Conversioni

da \ a	boolean	byte	short	char	int	long	float	double
boolean		n	n	n	n	n	n	n
byte 8bit	n		s	c	s	s	s	s
short 16bit	n	c		c	s	s	s	s
char 16 bit	n	c	c		s	s	s	s
int 32bit	n	c	c	c		s	s*	s
long 64bit	n	c	c	c	c		s*	s
float 32bit	n	c	c	c	c	c		s
double 64bit	n	c	c	c	c	c	c	

n non si applica; s viene fatto automaticamente; c mediante casting esplicito

La divisione intera

- Se entrambi gli operandi sono interi allora il risultato della divisione è un intero
  - 9 / 4 è 2 e non 2.25!
- Se si vuole che il risultato sia un numero decimale allora almeno uno degli operandi deve essere un numero in virgola mobile
  - 9 / 4.0 è 2.25

Riassumendo

- Le conversioni tra tipi di dato possono avvenire in 3 modi:
  a. **Conversioni** durante una operazione di assegnamento
  b. **Promozione** in una espressione aritmetica
  c. **Casting**

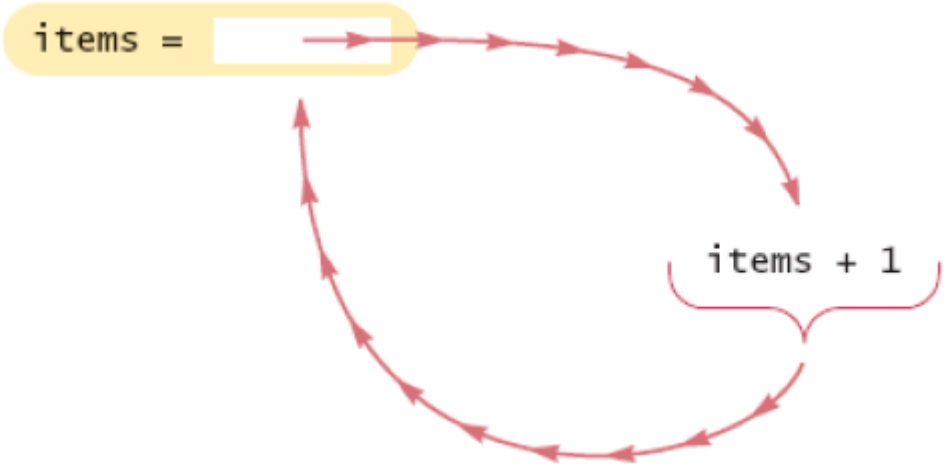
a. quando un valore di un tipo viene assegnato ad una variabile di un altro tipo
   - È consentita solo la conversione larga

b. avviene automaticamente quando operatori aritmetici devono convertire gli operandi

c. tecnica di conversione più pericolosa e potente
   - Tramite un casting esplicito si possono realizzare sia la conversione larga sia quella stretta

Assegnazioni, incrementi decrementi

- `items = items + 1;`
- `items++;`
- `items--;`



Variabili

- In Java le variabili possono essere dichiarate ovunque nel codice
  - `int a=20;`
  - `int n=a*10;`
- Una dichiarazione consiste in una serie modificatori (opzionale), un tipo e un nome
  - La dichiarazione delle variabili di istanza comincia con uno specificatore di accesso (opzionale)
- Variabili **final**
  - Il loro valore non può essere modificato (costante)
  - Possono essere dichiarate in
    - un metodo:

      final nomeTipo nomeVar = espressione;
    - una classe:

      specificatoreDiAccesso **static final** nomeTipo nomeVar = espressione;
  - Si usano in genere nomi con caratteri maiuscoli
- **Nota**: static denota una variabile della classe, quindi non ne viene creata una copia per ogni oggetto istanziato ma tutti gli oggetti fanno riferimento alla stessa variabile

Esempio

```
01: /**
02:    A cash register totals up sales and computes change
due.
03: */
04: public class CashRegister
05: {
06:    /**
07:       Constructs a cash register with no money in it.
08:    */
09:    public CashRegister()
10:    {
11:       purchase = 0;
12:       payment = 0;
13:    }
14:
15:    /**
16:       Records the purchase price of an item.
17:       @param amount the price of the purchased item
18:    */
19:    public void recordPurchase(double amount)
20:    {
21:       purchase = purchase + amount;
22:    }
```

Esempio

```
23:
24:    /**
25:       Enters the payment received from the customer.
26:       @param dollars the number of dollars in the payment
27:       @param quarters the number of quarters in the payment
28:       @param dimes the number of dimes in the payment
29:       @param nickels the number of nickels in the payment
30:       @param pennies the number of pennies in the payment
31:    */
32:    public void enterPayment(int dollars, int quarters,
33:          int dimes, int nickels, int pennies)
34:    {
35:       payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
36:             + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
37:    }
38:
39:    /**
40:       Computes the change due and resets the machine for the next
customer.
41:       @return the change due to the customer
42:    */
43:    public double giveChange()
44:    {
```

Esempio

```
45:         double change = payment - purchase;
46:         purchase = 0;
47:         payment = 0;
48:         return change;
49:     }
50:
51:     public static final double QUARTER_VALUE = 0.25;
52:     public static final double DIME_VALUE = 0.1;
53:     public static final double NICKEL_VALUE = 0.05;
54:     public static final double PENNY_VALUE = 0.01;
55:
56:     private double purchase;
57:     private double payment;
58: }
```

Esempio

```
01: /**
02:    This class tests the CashRegister class.
03: */
04: public class CashRegisterTester
05: {
06:    public static void main(String[] args)
07:    {
08:       CashRegister register = new CashRegister();
09:
10:       register.recordPurchase(0.75);
11:       register.recordPurchase(1.50);
12:       register.enterPayment(2, 0, 5, 0, 0);
13:       System.out.print("Change: ");
14:       System.out.println(register.giveChange());
15:       System.out.println("Expected: 0.25");
16:
17:       register.recordPurchase(2.25);
18:       register.recordPurchase(19.25);
19:       register.enterPayment(23, 2, 0, 0, 0);
20:       System.out.print("Change: ");
21:       System.out.println(register.giveChange());
22:       System.out.println("Expected: 2.0");
23:    }
24: }
```

Tipi primitivi e oggetti

- Valori in Java sono oggetti o tipi primitivi
  - variabili di un tipo primitivo contengono valori
  - variabili oggetto contengono riferimenti a oggetti

- Assegnamenti
  - tra variabili di tipo primitivo viene copiato il valore
    - Es. `x = y;` **// x e y hanno lo stesso valore ma non sono // collegate**
  - tra variabili oggetto viene copiato il riferimento all'oggetto
    - Es. `x = y;` **// x e y si riferiscono allo stesso oggetto**
    - per ottenere una copia di oggetti occorre invocare il metodo `clone()`
      - in alternativa, si può istanziare un nuovo oggetto (con lo stesso valore delle variabili d'istanza)

La Classe Math

- La classe **`Math`** del package **`java.lang`** contiene una serie di metodi *statici* (metodi della classe) da utilizzare per calcolare funzioni matematiche sui tipi primitivi.
- In genere i metodi in **`Math`** lavorano su **`double`** e restituiscono **`double`**, ma questo non è un limite perché un metodo che funziona su **`double`** funziona anche su tutti gli altri tipi (numerici).
- **NOTA** I metodi in **`Math`** non possono essere chiamati su variabili di tipo **`boolean`**

Metodi statici

- Non operano su una istanza
  - non sono associati ad un oggetto ma alla classe stessa
- Possono essere invocati senza alcun riferimento ad oggetto
- Sono definiti come tutti gli altri metodi con l'aggiunta del prefisso **`static`**
- Sono invocati utilizzando il nome della classe
  - Il riferimento **this** NON ha senso
  - NON possono accedere variabili di istanza
  - NON possono accedere metodi non statici
  - POSSONO invocare il costruttore

Metodi di Math (1)

- I principali metodi contenuti nella classe **`Math`** sono:
  - Valore assoluto (implementato anche per float, int e long)
    - **`double Math.abs(double x)`**
  - Funzioni trigonometriche
    - **`double Math.sin(x);`**
    - **`double Math.cos(x);`**
    - **`double Math.tan(x);`**
    - **`double Math.asin(x);`**
    - **`double Math.acos(x);`**
    - **`double Math.atan(x);`**

Metodi in Math (2)

- Max e Min (implementati anche per float, int e long)
  - `double Math.max(double x, double y)`
  - `double Math.min(double x, double y)`
- Potenza, esponenziale, logaritmo naturale e radice quadrata
  - `double Math.pow(double x, double y)`
  - `double Math.exp(double x)`
  - `double Math.log(double x)`
  - `double Math.sqrt(double x)`

Metodi in Math (3)

- https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

- Costanti (definite con final e static)
  - **`Math.PI`** (pi greco)
  - **`Math.E`** (base dei logaritmi naturali)

Invocazione di metodi statici

- ClassName.MethodName(parameters)
  - Metodo statico: metodo che non opera su un particolare oggetto della classe (non ha il parametro implicito)

- Esempio:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

Stringhe

- Sequenza di caratteri
- Oggetti della classe String
- Immutabili
  - nessun metodo di String modifica lo stato della stringa
- Stringhe costanti: "**`Carl`**"
- Variabili stringhe:

```
String name = "Carl";
```

- Lunghezza di una stringa:

```
int n = name.length();
```

Sottostringhe

- **`String greeting = "Clown";`**

  Le posizioni dei caratteri di una stringa sono numerate a partire da 0

  - $_0$**`C`** $_1$**`l`** $_2$**`o`** $_3$**`w`** $_4$**`n`**
- **`String sub = greeting.substring(1,4);`**

  Gli argomenti indicano la posizione del primo carattere della sottostringa e quella successiva all'ultimo carattere

  - Es. la stringa sub contiene low
- Se viene omesso secondo parametro si sottintende fino a fine stringa

  - **`String sub = greeting.substring(1);`**

    Ora sub contiene **`lown`**

Concatenazione

- ```
  String fname = "Harry";
  String lname = "Hacker";
  String name = fname + lname;
  ```
- **name** è **"HarryHacker"**

- Se un operando di + è una stringa, l'altro è convertito in una stringa:

  ```
  String a = "Agent";
  String name = a + 7;
  ```
- La stringa **name** è **"Agent7"**
  ```

Conversioni tra stringhe e numeri

- Da stringhe a numeri:
  - stringa contiene un numero (Es. “19” o “19.5”)

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(str);
```

  - La conversione lancia un’eccezione se non viene passata una `String` che non contiene un numero

`NumberFormatException` (di java.lang)

- Da numeri a stringhe:

```
String str = "" + n;
str = Integer.toString(n);
str = Double.toString(d);
```

Programma MakePassword.java

```
public class MakePassword
{
   public static void main(String[] args)
     { String firstName = "Harold";
       String middleName = "Joseph";
       String lastName = "Hacker";
       // estrai l'iniziale
       String initials = firstName.substring(0, 1)
                         + middleName.substring(0, 1)
                         + lastName.substring(0, 1);
       // aggiungi l'età
       int age = 19; // età dell'utente
       String password = initials.toLowerCase() + age;
       System.out.println("Your password is " + password);
     }
}
```

Leggere l'input da console

- Esiste l'oggetto System.in (della classe java.io.InputStream)
  - Legge solo 1 byte alla volta
- Una stringa però è costituita da caratteri (Unicode usa 2 byte per carattere)
  - In Java 5.0 si usa la classe Scanner (pacchetto java.util) per leggere l'input da tastiera in maniera più semplice

    ```
    Scanner in = new Scanner(System.in);
    ```

  - `int nextInt()` legge il prossimo int da tastiera
  - `double nextDouble()` legge il prossimo double da tastiera
  - `String nextLine()` legge la prossima riga da tastiera (finchè l'utente non pressa Enter)
  - `String next()` legge la prossima parola da tastiera (finchè non viene immesso il prossimo spazio bianco)

Programma Coins.java

```
import java.util.Scanner;
public class Coins{
  public static void main(String[] args){
     final double PENNY_VALUE = 0.01;
     final double NICKEL_VALUE = 0.05;
     final double DIME_VALUE = 0.1;
     final double QUARTER_VALUE = 0.25;

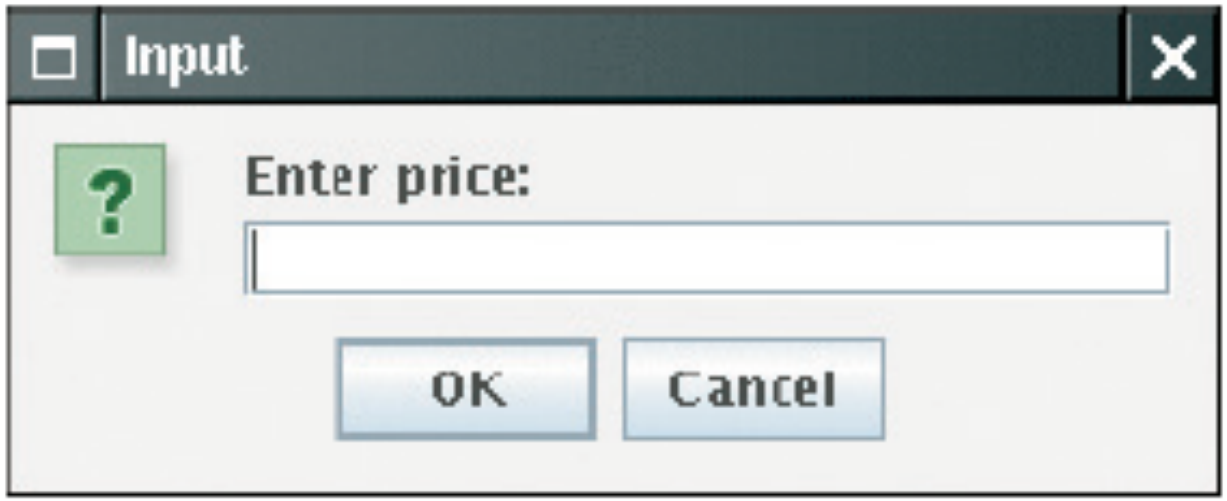
     Scanner in = new Scanner(System.in);
     System.out.println(“Quanti penny hai?");
     int pennies = in.nextInt();
     System.out.println(“Quanti nickel hai?");
     int nickels = in.nextInt();
```

Programma Coins.java

```
            System.out.println(“Quanti dime hai?");
            int dimes = in.nextInt();
            System.out.println(“Quanti quarter hai?");
            int quarters = in.nextInt();
            double total = pennies * PENNY_VALUE
                         + nickels * NICKEL_VALUE
                         + dimes * DIME_VALUE
                         + quarters * QUARTER_VALUE;
            // valore totale delle monete

            System.out.println("Total value = ” + total);
    } //chiude il corpo del main
} //chiude la definizione della classe
```

Leggere l'input da una Dialog Box



An Input Dialog Box

Leggere l'input da una Dialog Box

- `String input = JOptionPane.showInputDialog("Enter price:");`
- Restituisce un oggetto di tipo **`String`**

  `int count = Integer.parseInt(input);`
- Bisogna aggiungere

  `System.exit(0)`

  alla fine del metodo **`main`** di ogni programma che usa **`JOptionPane`**
- **`JOptionPane`** fa parte del package javax.swing