



Operazioni numeriche - Input

Espressioni, funzioni
matematiche, classi involucro,
Acquisizione di input

Operazioni fra interi e decimali

- In Java gli operatori + - * / si possono combinare come si vuole insieme a costanti numeriche, variabili di frame e variabili istanza per ottenere espressioni aritmetiche
- Gli operandi possono essere sia numeri interi (**byte**, **short**, **int**, **long**) che numeri in virgola mobile (**float**, **double**)
- Il risultato è un intero solo se tutti gli operandi sono interi
- Basta che un solo operando sia in virgola mobile perché il valore di tutta l'espressione sia in virgola mobile

Espressioni aritmetiche

- La grammatica per le espressioni aritmetiche che si possono scrivere in Java è la seguente:
 $E \rightarrow E + T \mid E - T \mid T$
 $T \rightarrow T * F \mid T / F \mid F$
 $F \rightarrow -\langle Op \rangle \mid \langle Op \rangle$
 $\langle Op \rangle \rightarrow (E) \mid \langle Num \rangle \mid \langle Ide \rangle \mid \langle DotExpr \rangle$
- La grammatica implementa le regole tipiche di associatività e precedenza degli operatori
- F è un ulteriore livello di precedenza per l'operatore unario - (che lega più di tutti) usato per cambiare il segno a un operando

Espressioni aritmetiche

- $\langle Num \rangle$ è un simbolo non terminale che genera tutte le possibili costanti numeriche (interi e decimali, con o senza notazione esponenziale)
- $\langle Ide \rangle$ è un simbolo non terminale che genera tutti i possibili identificatori Java
- $\langle DotExpr \rangle$ è un simbolo non terminale che genera tutte le possibili espressioni formate da nomi e/o chiamate di metodi separate da punti (ad esempio **myPurse.getTotal()** rappresenta un **double**, **this.nickels** rappresenta un **int** all'interno di un metodo della classe **Purse**)

Espressioni aritmetiche

- Esempi:
- $7 * 4 \rightarrow$ valore 28 (int)
- $11 + 2.0 / 4 \rightarrow$ valore 11.5 (double)
- $(11 + 2.0) / 4 \rightarrow$ valore 3.25 (double)
- **this.nickels * this.getTotal() - 3 * 7**
 \rightarrow supponendo che il valore della variabile istanza **nickel** dell'oggetto **this** sia 3 (int) e che il totale **this.getTotal()** sia 12.5 (double) si ottiene un valore double a causa del valore double dell'operando **this.getTotal()**. Il valore è 16.5

Divisione

- Bisogna prestare un'attenzione particolare al simbolo /
- In Java esso rappresenta sia la divisione usuale che la divisione intera
- Viene applicata la divisione intera se entrambi gli operandi sono interi
- La divisione intera restituisce solo la parte intera del risultato!
- $7 / 4 \rightarrow$ **valore 1! Non 1.75!**
- $7 / 4.0 \rightarrow$ valore 1.75 (4.0 non intero)

Resto della divisione intera

- Il simbolo % è un operatore binario che si può applicare solo fra due interi
- Calcola il resto della divisione intera fra il primo e il secondo
- $7 \% 4 \rightarrow$ valore 3

11/11/2004

Laboratorio di Programmazione - Luca Tesi

7

Errore comune

- L'overloading del simbolo / porta spesso ad errori logici difficili da individuare
- Ad esempio:

...

```
int p1 = 21; // punteggio prima prova
int p2 = 24; // punteggio seconda prova
int p3 = 22; // punteggio terza prova
double media = (p1 + p2 + p3) / 3;
System.out.println(media); /* Stampa
22.0! Non 22.3333333!! */
```

11/11/2004

Laboratorio di Programmazione - Luca Tesi

8

Errore comune

- Per ottenere il risultato che vogliamo bisogna fare in modo che almeno uno degli operandi sia un double

```
// Se un intero viene assegnato a una variabile
// di tipo double viene convertito a double
```

```
double totale = p1 + p2 + p3;
// totale è un valore double
double media = totale / 3;
```

Oppure

```
double media = (p1 + p2 + p3) / 3.0;
```

11/11/2004

Laboratorio di Programmazione - Luca Tesi

9

Metodi static

- La classe `java.lang.Math` (consultare le API) è una collezione di costanti e metodi **static**
- Abbiamo già visto che una variabile istanza dichiarata come **static** si riferisce alla classe e ne esiste un'unica copia (non viene inserita nello stato degli oggetti della classe che vengono creati)
- Anche un metodo può essere dichiarato come **static** e, analogamente, esso si riferisce alla classe

11/11/2004

Laboratorio di Programmazione - Luca Tesi

10

Metodi static

- Un metodo **static** non può essere invocato su un oggetto della classe
- L'unico modo per mandare in esecuzione il metodo **static** è quello di scrivere `NomeClasse.nomeMetodo(parametri)`;
- All'interno di un metodo **statico** non è disponibile il parametro implicito `this` (poiché non c'è nessun oggetto su cui il metodo è stato invocato)
- Per il resto il meccanismo di esecuzione è analogo a quello dei metodi non statici (in particolare la creazione di una nuova attivazione e il meccanismo di passaggio e gestione dei parametri espliciti)

11/11/2004

Laboratorio di Programmazione - Luca Tesi

11

Funzioni e costanti matematiche

- Definite nella classe `java.lang.Math` come costanti e metodi **static**
- Pi greco: `Math.PI`
- Base dei logaritmi naturali: `Math.E`
- Radice quadrata:

```
double radiceDi2 = Math.sqrt(2);
```
- Coseno:

```
double cosenoDiPiGrecoMezzi =
    Math.cos(Math.PI / 2);
```
- Consultare le API per vedere tutte le altre funzioni disponibili

11/11/2004

Laboratorio di Programmazione - Luca Tesi

12

Conversioni di tipi

- Il compilatore esegue alcune conversioni di tipo implicitamente:
- Quando un valore intero viene assegnato ad una variabile `double` o `float`, il valore viene convertito in `double` o `float` automaticamente

...

```
double pippo = 4; // 4 è una costante intera
System.out.println(pippo); // Stampa 4.0
```

- 4.0 è la rappresentazione di una costante a virgola mobile (c'è il punto decimale)

11/11/2004

Laboratorio di Programmazione - Luca Tesi

13

Conversioni esplicite

- La conversione precedente viene eseguita automaticamente perché non comporta nessuna perdita di informazione (i numeri interi hanno i loro corrispondenti nei numeri a virgola mobile)
- Quando un assegnamento, invece, può provocare una perdita di informazione viene segnalato dal compilatore come errore
- Una errore di questo genere si ha, ad esempio, quando si tenta di assegnare un valore a virgola mobile ad una variabile intera

```
int prova = 3.5; // Errore di
                // compilazione
```

11/11/2004

Laboratorio di Programmazione - Luca Tesi

14

Conversioni esplicite

- Per forzare il compilatore ad accettare l'assegnamento (se si vuole accettare la perdita di informazione che ne deriva) si deve fare una conversione di tipo esplicita
 - Questa operazione si chiama **casting**
- ```
int prova = (int)3.5; // accettato
```
- L'effetto di questo casting è di buttare via la parte decimale del numero a virgola mobile e di assegnare a `prova` solo la parte intera 3

11/11/2004

Laboratorio di Programmazione - Luca Tesi

15

## Casting

- Il casting si può fare fra diversi tipi base, e ogni volta che si fa si accetta la possibilità di perdere informazione
- Ad esempio, facendo un casting da `double` a `float` si potranno perdere alcune cifre significative
- Così come facendo un casting da `int` a `short`

11/11/2004

Laboratorio di Programmazione - Luca Tesi

16

## Casting

- L'operatore di casting (il tipo tra parentesi) lega più degli operatori aritmetici

```
double total = 3.456;
int prova = (int) total * 100;
```

- Esegue il casting su `total` e il valore di `prova` sarà 300

```
int prova = (int)(total * 100);
```

- Esegue il casting sul valore della moltiplicazione e il valore di `prova` sarà 345

11/11/2004

Laboratorio di Programmazione - Luca Tesi

17

## Arrotondamenti

- Il casting da valori a virgola mobile a interi produce come risultato la parte intera del valore, anche se il valore è molto prossimo al valore intero successivo

```
int prova = (int) 4.99; // prova vale 4
```

- Per eseguire gli arrotondamenti secondo la regola usuale (se il primo decimale è da 0 a 4 si prende la parte intera, altrimenti la parte intera + 1) si può usare il metodo statico `Math.round` (consultare le API: il valore restituito da `Math.round` è un `long`! Quindi bisogna fare un ulteriore casting se si vuole assegnare il risultato ad un `int`)

11/11/2004

Laboratorio di Programmazione - Luca Tesi

18

## Classi involucro

- Ognuno dei tipi base di Java ha una classe corrispondente nel pacchetto `java.lang` (consultare le API)
- Queste classi vengono dette classi involucro
- Gli oggetti di queste classi possono un valore del tipo base corrispondente
- La vera utilità sta nelle costanti e nei metodi statici che forniscono

## Classe Integer

- Prendiamo ad esempio la classe `java.lang.Integer`
- Un oggetto di questa classe può contenere un valore intero
- Tale valore non si trova in una variabile di frame come i soliti int
- Si trova nello heap all'interno dell'oggetto corrispondente
- Costanti statiche utili della classe:
  - `Integer.MIN_VALUE` (minimo numero int rappresentabile)
  - `Integer.MAX_VALUE` (massimo int rappresentabile)

## Classe Integer

- Metodi statici pubblici utili della classe `Integer`  
`public static int parseInt(String s)`
- Cerca di interpretare il contenuto della stringa `s` come la rappresentazione di una costante intera:
- `Integer.parseInt("15")` restituisce l'intero 15

## Classe Integer

- Se la stringa contiene dei caratteri che non possono essere considerati la rappresentazione di un intero allora il metodo `parseInt` lancerà un'eccezione
- Le eccezioni sono il meccanismo di base di Java per gestire gli errori
- Per adesso non gestiamo questa eccezione, vedremo più avanti come si fa
- Se un'eccezione non viene gestita (il termine giusto sarebbe "catturata") il programma termina con errore indicando quale eccezione si è verificata e la pila di attivazioni corrente (ci permette di risalire al punto preciso del programma in cui si è verificata)

## Letture di dati in input

- Vediamo due modi per acquisire dei dati di input dall'utente:
  1. Tramite una finestra grafica di dialogo
  2. Tramite lo standard input (la console, ma in generale può essere un qualunque file)
- I dati in input rendono il programma interattivo e possono fare in modo che il comportamento dello stesso programma sia diverso se vengono dati input diversi

## Finestra di dialogo

- Possiamo chiedere all'utente di inserire una stringa e possiamo trattare l'oggetto corrispondente nel nostro programma
- La classe che ci serve per far apparire la finestra è `javax.swing.JOptionPane` (consultare API)
- Questa classe ha diversi metodi statici `showInputDialog` che restituiscono un oggetto `String`
- Per far apparire la finestra e ottenere l'input:  

```
String input =
 JOptionPane.showInputDialog(
 "Quanti nickel hai?");
```

## Finestra di dialogo

- La stringa che passiamo come parametro sarà visualizzata sulla finestra



## Finestra di dialogo

- L'utente è libero di digitare qualunque cosa nel campo di input
- Poi potrà fare click sui pulsanti OK o Annulla
- Se l'utente fa click su Annulla viene restituito il valore `null` e la variabile `String input` varrà quindi `null`
- Se l'utente fa click su OK viene creato un oggetto `String` che contiene la stringa inserita e il suo riferimento viene assegnato alla variabile `input`

## Esempio di uso

```
import javax.swing.JOptionPane;

public class InputTest {
 public static void main(String argv[]) {
 String input = JOptionPane.showInputDialog(
 "Quanti nickel hai?");
 // Utilizzo il metodo parseInt per ottenere
 // il numero intero digitato
 int nickels = Integer.parseInt(input);
 System.out.println(nickels);
 // Nei programmi che usano la grafica si deve usare
 // questa chiamata per terminare il programma
 System.exit(0);
 }
}
```

## Esempio

- Naturalmente si dovrebbe gestire sia il caso in cui l'utente fa click su Annulla sia il caso in cui quello che digita non è un intero
- Ancora non abbiamo gli strumenti per farlo
- Per adesso ci accontentiamo di un programma che, se qualcosa va storto, termina con errore
- Ricordarsi sempre di inserire `System.exit(0)`; come ultima istruzione del main, altrimenti il programma resta in esecuzione (il gestore della grafica rappresenta un processo della nostra applicazione che rimane attivo fino a quando non lo si chiude esplicitamente con il metodo static `System.exit`)