Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

12 Ottobre 2022

Struttura di un programma Java

Un *programma* in Java è un insieme di dichiarazioni di *classi*.

Una *classe* non può contenere direttamente delle istruzioni, ma può contenere la dichiarazione di *metodi*, che contengono dichiarazioni ed istruzioni (terminate da ;).

Un *metodo* implementa un'operazione, una funzionalità, un sottoprogramma (in modo simile alle funzioni e procedure in linguaggi come C e Pascal).

Un *oggetto* è un'istanza di una classe.

Un programma in Java può essere visto anche come una collezione di oggetti che soddisfano certi requisiti e che interagiscono tramite delle funzionalità rese pubbliche (*tipo di dato astratto*).

Il primo programma

Occorre una Java Virtual Machine (interprete per Java).

In questo corso faremo riferimento a JDK (Java Development Kit), un ambiente di programmazione in Java molto spartano[1].

Supponiamo di voler scrivere un semplice programma Java che stampa sul video la stringa (o sequenza di caratteri) `Ciao!`.

Possiamo creare una cartella (o directory) in cui mettere i file contenenti i primi esercizi in Java.

Tramite un editore di testo apriamo un file `Ciao.java` in cui scriviamo il seguente programma:

[1] Nelle esercitazioni del corso saranno presentati altri ambienti.

Programma Ciao

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

N.B. Il nome della classe (i.e. il nome che segue la parola `class`) coincide con quello del file.

La *convenzione standard* in Java consiste nell'avere *una sola* classe per file ed il nome del file è il nome della classe seguito dal suffisso `.java`.

Programma Ciao: commenti

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

Il programma è costituito da una sola classe.

La dichiarazione della classe inizia con la parola riservata `class` seguita dal nome (*identificatore*) della classe `Ciao` e da una parentesi graffa aperta (che sarà chiusa alla fine della dichiarazione della classe).

Programma Ciao: il metodo main

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

La classe `Ciao` contiene un metodo, che si chiama `main`.

In Java ogni metodo può essere *chiamato* (*invocato*) solo all'interno di un altro metodo.

Quindi occorre un metodo iniziale per far partire l'esecuzione del programma.

Tale metodo iniziale è il metodo `main`, che deve essere sufficientemente generale.

Il metodo main: intestazione

```
public static void main (String[] args)
```

L'intestazione del metodo `main` è *sempre la stessa* (eccetto che in alcuni testi si usa l'identificatore `argv` al posto di `args` e si scrive `String argv[]` invece di `String[] args`).

Il metodo `main` si caratterizza per le seguenti proprietà:

- è pubblico, denotato tramite la parola riservata `public` (ovvero, visibile da ogni altro punto del programma);
- è statico, denotato tramite la parola riservata `static` (ovvero, dipende dalla classe in cui si trova e non da oggetti, in quanto all'inizio non sono stati creati ancora degli oggetti);

Il metodo main: intestazione (cont.)

```
public static void main (String[] args)
```

- non restituisce niente, denotato tramite la parola riservata `void`, in quanto non vi è alcun metodo che lo chiama a cui restituire un valore;
- i dati in ingresso sono visti come un *array di stringhe* (le stringhe sono oggetti della classe predefinita `String`).

Riparleremo più avanti di come dare i dati in input.

Notare che nel programma `Ciao` non viene utilizzato alcun dato in input.

Il metodo main: corpo

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

Il corpo del metodo `main` contiene una sola istruzione.

È l'istruzione di stampa `System.out.println("Ciao!");`
in cui viene chiamato il metodo `println` della classe (predefinita) `PrintStream` sul *campo* `out` della classe `System`.

Il metodo `println` stampa la stringa (*racchiusa tra i doppi apici*) che compare dentro le parentesi e poi va a capo.

Se si usa il metodo `print` al posto di `println`, viene stampata la stessa stringa `Ciao!`, ma non viene effettuato il ritorno a capo.

Compilazione del programma

Una volta salvato il file `Ciao.java`, è possibile compilare il codice sorgente contenuto in tale file tramite il comando `javac`:

```
javac Ciao.java
```

La fase di compilazione ha successo (il codice è sintatticamente corretto).

Se invece il compilatore segnala delle situazioni di errori, occorre i) cercare di capire dove sono e quali sono le modifiche da fare; ii) salvare le modifiche; iii) compilare nuovamente il file `Ciao.java`.

Quando la compilazione ha successo, nella cartella compare un file `Ciao.class` generato dal compilatore.

Tale file rappresenta il codice Bytecode che sarà eseguito dalla JVM.

Esecuzione del programma

A questo punto è possibile eseguire il programma chiamando la JVM (interprete di Java) tramite il comando `java` seguito dal nome del file (senza suffisso) nel modo seguente:

```
java Ciao
```

Se non vi sono errori in fase di esecuzione (e questo è il caso), viene stampato `Ciao!` sul video, si va a capo e si ha un nuovo prompt.

Quindi:
il Bytecode contenuto nel file con suffisso `.class` viene eseguito invocando il comando `java` seguito da tale file (omettendo il suffisso).

Polimorfismo e overloading

```
class ProvaStampa {
  public static void main (String[] args) {
    System.out.println("start");
    System.out.print("abc");
    System.out.print(1);
    System.out.println();
  }
}
```

I metodi `print` e `println` sono usati in vari modi diversi:
si ha lo stesso nome, ma parametri in numero e/o tipo diversi, e quindi significati diversi.

In tali casi si dice che il metodo è *sovraccaricato* (dall'inglese *overloaded*). Più in generale si dice che il metodo è *polimorfico*.

Polimorfismo e overloading (cont.)

L'esecuzione del precedente programma dà il seguente output:

```
start
abc1
```

Molti metodi predefiniti in Java sono polimorfici per operare diversamente su interi, numeri reali, stringhe, etc.

Un classico esempio di operatore polimorfico è il + che denota sia la somma su numeri interi (`int`), numeri decimali (`float`, `double`), etc., che la concatenazione di stringhe.

N.B.

- L'operatore + associa a sinistra.

- In presenza dell'operatore + e di una stringa tra i suoi operandi, anche *gli altri operandi sono considerati come stringhe*.

L'operatore polimorfico +

```
class Stampa {
  public static void main (String[] args) {
    System.out.println("abc");
    System.out.println(1+3);
    System.out.print("hello, ");
    System.out.print("world!");
    System.out.println();
    System.out.println("ab"+"cdef");
    System.out.println("34"+"ciao");
    System.out.println(34+"ciao");
    System.out.println(7+3+"hip");
    System.out.println(7+"hip"+3);
    System.out.println(7+(3+"hip"));
  }
}
```

L'operatore polimorfico + (cont.)

Dopo aver compilato il programma tramite `javac Stampa.java`, l'esecuzione con il comando `java Stampa` dà il seguente output:

```
abc
4
hello, world!
abcdef
34ciao
34ciao
10hip
7hip3
73hip
```

N.B. Ogni metodo Java è seguito da una coppia di parentesi ( ), anche se il metodo non ha parametri (espliciti).

Astrazione sui dati in input

Supponiamo di voler scrivere un programma Java che calcola e stampa la somma di due numeri interi.

Per esempio, se vogliamo sommare 7 e 4, possiamo scrivere:

```
class Somma1 {
  public static void main (String[] args) {
    System.out.println(7+4);
  }
}
```

La chiamata del metodo `println` con *parametro attuale* (o argomento) l'espressione 7+4 viene eseguita valutando dapprima l'espressione 7+4 con risultato 11, valore che viene poi stampato sul video.

Dati in input

Però non abbiamo sfruttato la possibilità di dare in ingresso *valori di volta in volta diversi*, di cui si vuole calcolare la somma.

I due valori che vogliamo passare come input al programma sono denotati tramite `args[0]` ed `args[1]`, rispettivamente il primo ed il secondo elemento dell'array `args`.[2]

Possiamo quindi pensare di scrivere il seguente programma:

```
class Conc {
  public static void main (String[] args) {
    System.out.println(args[0]+args[1]);
  }
}
```

[2] Come vedremo più avanti, gli elementi di un array monodimensionale sono individuati tramite un indice che parte da 0.

Dati in input (cont.)

Una volta compilato, il programma viene eseguito passandogli come input una sequenza di valori cosí come richiesto dal programma.

I due valori da sommare vengono semplicemente scritti l'uno dopo l'altro separati da uno spazio bianco:

```
java Conc 7 4
74
```

Il risultato non è `11`, ma `74`!

N.B. L'input al programma (i.e. al metodo `main`) è un array di stringhe. Quindi i *valori in input* 7 e 4 sono *considerati come stringhe*. Ne segue che l'operazione + applicata è la concatenazione di stringhe, quindi il risultato è la stringa 74.

Qualche test...

```
java Conc abc defg
abcdefg

java Conc "abc" "defg"
abcdefg

java Conc abc 11
abc11

java Conc 13 65
1365

java Conc ab cd ef
abcd
```

Nell'ultimo test ci sono valori in input in più, ma nessun problema.

Ancora qualche test...

```
java Conc
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at Conc.main(Conc.java:3)
```

Non viene passato alcun valore in input: fallisce la valutazione di `args[0]`.

```
java Conc ab
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 1
        at Conc.main(Conc.java:3)
```

Viene passato il primo valore, ma non il secondo: fallisce la valutazione di `args[1]`.

Somma di due numeri interi

Affinché i valori in ingresso dati tramite `args` (che sono *sempre* presi come stringhe), siano considerati come numeri interi, occorre *convertire* tali valori in interi tramite il metodo `parseInt` della classe `Integer`, ottenendo il seguente programma:

```
class Somma {
  public static void main (String[] args) {
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
    System.out.println(x+y);
  }
}
```

Finalmente:

```
java Somma 7 4
11
```

Somma di due numeri decimali

Occorre fare una conversione simile anche se si vuole sommare due numeri decimali con il metodo `parseDouble` della classe `Double` :

```
class SommaDouble {
  public static void main (String[] args) {
    double x = Double.parseDouble(args[0]);
    double y = Double.parseDouble(args[1]);
    System.out.println(x+y);
  }
}
```

In tal caso avremo:

```
java SommaDouble 3.5 7.4
10.9
```

mentre utilizzando il programma `Conc` si ha:

```
java Conc 3.5 7.4
3.57.4
```

Esercizio: area di un rettangolo

Scrivere un programma Java che, dati in ingresso la base e l'altezza di un rettangolo, stampa la sua area.

```
class AreaRettangolo {
  public static void main (String[] args) {
    double b = Double.parseDouble(args[0]);
    double a = Double.parseDouble(args[1]);
    System.out.println(b*a);
  }
}
```

Esercizio: area di un triangolo

Scrivere un programma Java che, dati in ingresso la base e l'altezza di un triangolo, stampa la sua area.

```
class AreaTriangolo {
  public static void main (String[] args) {
    double b = Double.parseDouble(args[0]);
    double a = Double.parseDouble(args[1]);
    System.out.println((b*a)/2);
  }
}
```

Cose da ricordare!

Alcune convenzioni standard in Java su classi e loro nome:

- *una sola* classe per file;
- il nome della classe coincide con il nome del file (terminato dal suffisso `.java`);
- il nome di una classe comincia con una lettera maiuscola (se il nome è costituito da più nomi, questi cominciano con una maiuscola e sono concatenati senza _ e −, e.g. Studente, StudenteUniv).

In Java tutto ciò che è racchiuso tra doppi apici `"..."` viene visto come una stringa.

Ogni metodo Java è sempre seguito da una coppia di parentesi ( ), sia nella dichiarazione che nella chiamata, anche se il metodo non ha parametri (espliciti).