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 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).

Supponiamo di voler scrivere un semplice programma Java che stampa sul video la stringa (o sequenza di caratteri) `Ciao!`. Occorre una Java Virtual Machine (ovvero un interprete per Java). In queste note faremo riferimento a JDK (Java Development Kit), un ambiente di programmazione in Java molto spartano. Dopo aver creato 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 viene scritto il seguente programma:

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

Notare che il nome della classe coincide con quello del file. La convenzione standard in Java consiste nell'avere una sola classe per file ed il nome della classe coincide con il nome del file (terminato dal suffisso `.java`).

Facciamo alcuni commenti sul programma appena scritto. Abbiamo detto che un programma Java è costituito da un insieme di classi. Qui abbiamo una sola classe la cui dichiarazione 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).

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`, il quale deve essere sufficientemente generale. L'intestazione del metodo `main` è

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

ed è sempre la stessa[1].

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`), in quanto all'inizio non sono stati creati ancora degli oggetti;
- 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 array di stringhe (oggetti della classe predefinita `String`). Riparleremo piú avanti di come dare i dati in input. Notiamo che nel programma riportato sopra non viene utilizzato alcun dato in input.

Una volta salvato il file, dalla shell dei comandi è adesso possibile compilare il codice sorgente contenuto nel file `Ciao.java` tramite il comando `javac` nel modo seguente:

[1] In alcuni testi la parte tra parentesi è scritta `String argv[]`. In Java il tipo *array di String* può essere denotato anche con le parentesi quadre messe dopo l'identificatore di array (`args` o `argv` in tal caso), ma si sconsiglia fortemente questa scrittura suggerendo invece di usare la notazione in cui il tipo è scritto in modo compatto, ovvero `String[]`. Per quanto riguarda l'uso di `args` o `argv`, in queste note si scriverà `args`, ma essendo questo il nome (identificatore) del parametro formale del metodo `main`, può essere scelto un qualsiasi identificatore legale.

```
javac Ciao.java
```

La fase di compilazione ha successo (il codice è sintatticamente corretto) e la macchina ci presenta il prompt. Se invece il compilatore segnala delle situazioni di errori, occorre cercare di capire dove sono e quali sono le modifiche da fare. Una volta salvate le modifiche, il file `Ciao.java` può essere compilato nuovamente. Se la compilazione ha successo, si può vedere che la cartella contiene un file `Ciao.class` che viene generato dal compilatore e che rappresenta il codice Bytecode (per visualizzare il Bytecode occorre dare il comando `javap -c Ciao`). A questo punto è possibile eseguire il programma 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. Infatti, nell'istruzione `System.out.println("Ciao!");` si ha la chiamata del metodo `println` della classe (predefinita) `PrintStream` sul campo `out` della classe `System`, che 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.

Polimorfismo e overloading

Dire che cosa viene stampato dal programma seguente:

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

Notare che sia il metodo `print` che il metodo `println` sono usati in vari modi diversi: si ha lo stesso nome, ma parametri in numero e/o tipo diversi. Si dice che il metodo è *sovraccaricato* (dall'inglese *overloaded*). Piú in generale si dice che il metodo è *polimorfico*. Molti metodi predefiniti in Java sono polimorfici per lavorare 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.

Ad esempio, dato il seguente programma Java:

```
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"));
   }
}
```

dopo averlo compilato tramite `javac Stampa.java`, l'esecuzione tramite il comando `java Stampa` dà luogo al seguente output:

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

Da notare che in presenza dell'operatore `+` e di una stringa tra i suoi operandi, anche gli altri operandi sono considerati come stringhe (va ricordato che in Java l'operatore `+` è associativo a sinistra).

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 i numeri 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`, dando come risultato `11`, e quindi tale valore viene stampato sul video. Di nuovo 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` (come vedremo piú avanti, gli elementi di un array monodimensionale sono individuati tramite un indice che parte da 0). Possiamo quindi pensare di scrivere il seguente programma:

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

Una volta compilato, il programma viene eseguito passandogli come input una sequenza di valori cosí come richiesto dal programma. In questo caso il programma si aspetta due valori da sommare, quindi dovremo fornire due interi, che vengono semplicemente scritti l'uno dopo l'altro separati da uno spazio bianco:

```
java Conc 7 4
```

Il risultato che viene stampato non è però `11`, ma `74`. Infatti, occorre ricordare che l'input al programma (ovvero al metodo `main`) è un array di stringhe, quindi i due valori in input `7` e `4` sono considerati come stringhe e di conseguenza l'operazione applicata (denotata tramite l'operatore polimorfico `+`) è la concatenazione di stringhe. Da ciò segue il risultato dato dalla stringa `74`. Facciamo qualche test di verifica:

```
java Conc abc defg
abcdefg

java Conc "abc" "defg"
abcdefg

java Conc abc 11
abc11

java Conc 13 65
1365

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

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

java Conc ab cd ef
abcd
```

Per effettuare la somma sui valori dati in ingresso, che sono *sempre* presi come stringhe, occorre convertire tali valori in interi tramite il metodo `parseInt` della classe `Integer`. Modifichiamo quindi il programma ottenendo il seguente:

```
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);
  }
}
```

Occorre fare una conversione simile anche se si vuole sommare due numeri decimali:

```
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:

```
javac SommaDouble.java
java SommaDouble 3.5 7.4
10.9
```

mentre utilizzando il programma `Conc` si ha:

```
javac Conc.java
java Conc 3.5 7.4
3.57.4
```

Esercizi

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

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

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

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