***Soluzione di sottoproblemi***

Spesso i problemi che incontriamo nella realtà sono complessi. A livello dell’analisi e del progetto, è conveniente affrontare la soluzione utilizzando la tecnica di progettazione top-down, che suggerisce di suddividere il problema complesso in un insieme di sottoproblemi autonomi tra loro. All’analisi top-down segue la successiva fase di sviluppo bottom-up in cui ogni sottoproblema è progettato in modo indipendente. Questa impostazione si riflette nella creazione:

1. di un insieme di sottoprogetti, che contengono il modello dei dati e il corrispondente processo risolutivo di ogni singolo sottoproblema (sottoalgoritmo);
2. dall’algoritmo principale (*main*), il cui compito è quello di richiamare i diversi sottoalgoritmi.

In Java, a livello della realizzazione, i sottoprogetti sono codificati mediante metodi richiamati dal metodo *main()* di un’applicazione che implementa l’algoritmo principale. Nel seguito seguiremo questo modello tipico del paradigma procedurale, con l’obiettivo di imparare a costruire metodi che realizzano elaborazioni autonome, progettate con un proprio modello dei dati e sottoalgoritmo. Queste conoscenze saranno però utilizzate nel prossimo modulo per poter definire le elaborazioni sugli oggetti, mediante metodi da inserire (incapsulare) nel corpo della dichiarazione della classe, che definisce il comportamento degli oggetti stessi.

*I metodi*

Il metodo (method) è l’unico costrutto del Java, a disposizione dei programmatori, che permette di codificare una parte dell’elaborazione (modello dei dati + sottoalgoritmo) svolta da una classe.

In una classe, un metodo realizza una particolare elaborazione, definita dal programmatore, in cui possiamo riconoscere:

1. un input, definito da un insieme di variabili di ingresso, denominate parametri;
2. una elaborazione, che traduce il modello dei dati e il sottoalgoritmo;
3. un output, ovvero il risultato dell’elaborazione fornito dal metodo stesso.

Un metodo si comporta come una funzione (*function*) ovvero un particolare sottoprogramma (in altri linguaggi, *procedure* o *subroutine*) e sé stante che restituisce un solo valore. In un sottorpogramma, i dati di ingresso devono essere assegnati, come studieremo successivamente, al momento del suo richiamo. Vedremo che in Java, per realizzare alcuni compiti, i parametri e/o il risultato fornito dal metodo possono mancare.

*Dichiarazione di un metodo*

Un metodo Java è formato da:

1. una intestazione,
2. un corpo,

e si dichiara sulla base del diagramma sintattico riportato di seguito.

**Dichiarazione di un metodo**

Intestazione

{

Corpo

}

**blocco**

L’intestazione si costruisce sulla base del grafo sintattico seguente.

**Intestazione di un metodo**

Modificatore per il controllo dell’accesso

**final**

Tipo di dato risultato

Identificatore metodo

(

**static**

Elenco parametri del metodo

)

Essa deve sempre dichiarare:

* il *tipo di dato del risultato*, che può essere di tipo primitivo o una classe;
* l’*identificatore del metodo*, ovvero il nome che si utilizzerà per richiamare l’elaborazione definita dallo stesso sottoprogramma;
* un elenco, eventuale, dei parametri del metodo (o semplicemente parametri) individuati dal loro tipo di dato (primitivo o classe) e un identificatore, come descritto nel diagramma sintattico seguente. I parametri (*parameter*) definiscono delle variabili o degli oggetti il cui ambito (*scope*) è quello del corpo del metodo stesso.

**Elenco parametri del metodo**

Tipo di dato primitivo

Classe

Identificatore parametro

(

I *modificatori per il controllo dell’accesso* (*public*, *private* e *protected*) del Java sono utilizzati per definire la visibilità di un metodo in classi differenti e saranno affrontati quando impareremo a costruire una nuova classe. Nel seguito useremo solo quello *public*, per indicare che un metodo può essere richiamato anche in altre classi. Per apprendere come realizzare un metodo utilizzeremo sempre la parola chiave *static* che dichiara metodi statici (o di classe). I metodi statici possono essere richiamati direttamente nella classe in cui sono dichiarati (senza creare nuovi oggetti di quella classe), scrivendo il loro nome come un “tradizionale” programma procedurale.

La scrittura dell’intestazione di un metodo è spesso utilizzata dai programmatori per descrivere: l’identificatore del metodo, la lista dei parametri e il tipo di dato restituito. Per questo motivo, nella tecnologia Java, l’intestazione costituisce letteralmente la firma di un metodo (*method signature*).

Il corpo (*body*) di un metodo è formato da un insieme di istruzioni racchiuse in un blocco di elaborazione.

In genere, nel corpo di un metodo è presente:

1. una sezione di dichiarazione delle variabili locali che traducono il modello dei dati “locale”;
2. una parte eseguibile, che contiene la codifica dell’algoritmo risolutivo di un sottoprogetto.

*Definizione del risultato di un metodo*

In Java, il nome di un metodo non individua il risultato fornito, ma l’elaborazione svolta nel corpo dello stesso metodo. Il risultato di un metodo è definito mediante l’istruzione di salto **return**, impiegata con la seguente sintassi:

**return** [*espressione*]optional;

Nel corpo di un metodo, l’istruzione **return**:

1. termina l’esecuzione del metodo, restituendo il controllo dell’elaborazione alla parte di programma chiamante;
2. restituisce come risultato il valore dell’*espressione* scritta alla sua destra.

I programmatori Java tendono a inserire l’elaborazione svolta da un metodo direttamente, quando è possibile, nell’espressione dell’istruzione **return**.

Nel corpo di un metodo, l’istruzione **return** può:

* comparire una o più volte;
* non essere presente, quando la funzione non definisce alcun risultato.