Parte quinta: la ricorsività

Per comprendere cos'è la ricorsività, riferiamoci subito ad un esempio. Proviamo a prendere due specchi e a porli uno di fronte all'altro. Se guardiamo all'interno di uno dei due specchi, notiamo che vengono riflessi all'infinito. Oppure, immaginiamo un computer: nello schermo viene rappresentata l'immagine di se stesso (quindi se stesso) con un'ulteriore rappresentazione di se stesso sullo schermo e così via (fig. 1).
In entrambi i casi ci troviamo di fronte ad un oggetto ricorsivo. Si parla di ricorsività diretta quando cerchiamo di definire qualcosa riferendoci alla sua stessa definizione. Più generalmente, un oggetto si può definire ricorsivo quando viene definito attraverso la descrizione di se stesso. In definitiva, *una funzione viene detta ricorsiva quando richiama se stessa dal suo interno.*

Figura 1

Quindi, affermiamo che per attivare un procedimento ricorsivo dobbiamo avere tre elementi caratteristici:

- *Condizione di verifica del caso*
- *Soluzione del caso particolare*
- *Soluzione generale*

La condizione di verifica del caso serve appunto per verificare se si è di fronte ad un caso particolare.

Vedremo adesso due esempi: il fattoriale e la serie di Fibonacci.

1. Il fattoriale

Il fattoriale è una funzione avente per dominio l'insieme dei numeri naturali, i cui valori sono anch'essi numeri naturali. Come si può osservare, il fattoriale di un generico numero *n* cresce molto rapidamente al crescere di *n*: si ha infatti:

0! = 1	4! = 24	8! = 40.320
1! = 1	5! = 120	9! = 362.880
2! = 2	6! = 720	10! = 3.628.800
3! = 6	7! = 5040	11! = 39.916.800

Se applichiamo la definizione di $n!$, ossia

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \qquad (1)$$

al numero (n - 1), otteniamo il prodotto dei primi (n - 1) numeri naturali, cioè

$$(n-1)! = (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \qquad (2)$$

moltiplicando per n entrambi i membri della (2) otteniamo

$$n \cdot (n-1)! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$$

ossia, osservando che il secondo membro di quest'ultima uguaglianza è $n!$, si può scrivere

$$n! = n \cdot (n-1)!$$

Quindi, *per calcolare il fattoriale di un numero n basta moltiplicare per n il fattoriale del numero precedente*.

L'ultima relazione permette di formulare una **definizione ricorsiva** del fattoriale:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases} \qquad (3)$$

Mediante le relazioni (3) è possibile ricondurre il calcolo del fattoriale di un numero naturale qualsiasi a quello di 0!, che è noto.

Proviamo quindi a calcolare 5!. Applicando la formula espressa nella (3), otteniamo che 5! = 120, infatti $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 120$.

Abbiamo visto cos'è il fattoriale. Adesso scriviamo il metodo ricorsivo che calcoli il fattoriale di un numero n:

```
public long fattoriale(long n)
{
   if(n == 0)
      return 1;
   else
      return n * fattoriale(n - 1);
}
```

Notiamo all'interno del metodo fattoriale la chiamata a se stesso.

2. La serie di Fibonacci

I seguenti numeri rappresentano la serie di Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Notiamo che la serie di Fibonacci comincia con i numeri 0 e 1 e possiede la proprietà che ogni successivo numero di Fibonacci è la somma dei due numeri che lo precedono. La serie di Fibonacci può essere definita in modo ricorsivo come segue:

$$\begin{cases} fibonacci(0) = 0 \\ fibonacci(1) = 1 \\ fibonacci(n) = fibonacci(n-1) + fibonacci(n-2) \end{cases}$$

Come già detto, anche in questo caso possiamo utilizzare un metodo ricorsivo per calcolare la serie di Fibonacci. Di seguito è riportato il codice del metodo fibonacci().

```
public long fibonacci(long n)
{
   if(n == 0 || n == 1)
      return n;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Adesso salviamo i due metodi all'interno della nuova classe NewMath. [NOTA: introduciamo già da ora la classe NewMath che sarà utilizzata come esempio nella prossima parte ed anche in alcune parti successive. L'obiettivo che ci proponiamo è di poter creare una classe che simuli la già esistente classe Math. Questo sarà visto nella parte successiva.]

3. Creiamo la classe NewMath

Creeremo adesso la classe NewMath.

```
public final class NewMath
{

      private NewMath()
      {
```

```
	}

}
```

Abbiamo creato la classe NewMath. Nella parte successiva capiremo perché l'abbiamo dichiarata come final ed abbiamo definito un livello di accesso privato nel metodo costruttore. Tornando ai metodi fattoriale() e fibonacci(), aggiungiamo loro la parola chiave static. La classe NewMath, come abbiamo visto, non permette di creare istanze di se, quindi i suoi metodi devono essere statici. Il codice seguente mostra la classe NewMath con i metodi fibonacci() e fattoriale():

```
public final class NewMath
{

	private NewMath()
	{

	}

	public static long fibonacci(long n)
	{
	   if(n == 0 || n == 1)
	      return n;
	   else
	      return fibonacci(n - 1) + fibonacci(n - 2);
	}

	public static long fattoriale(long n)
	{
	   if(n == 0)
	      return 1;
	   else
	      return n * fattoriale(n - 1);
	}


}
```

4. Conclusioni sulla ricorsività

Abbiamo quindi visto che la ricorsione è una tecnica di programmazione importante che consente a una funzione di richiamare se stessa dal suo interno. È importante sottolineare un possibile problema associato alla ricorsione, ovvero è possibile creare un procedimento ricorsivo che non ottiene mai un risultato definitivo e che non raggiunge mai un punto finale. Questo tipo di ricorsione causa l'esecuzione di un ciclo "infinito". Questo processo ovviamente non raggiunge mai un punto finale. È quindi importante progettare le funzioni ricorsive con estrema attenzione. Se esiste anche la minima possibilità di creare un ciclo infinito, è possibile fare in modo che la funzione conti il numero di chiamate a se stessa.