Metodologie di Programmazione

Author Valerio Fontana

appunti aggiornati al 24/02/2023

Indice

Chapter 1

Le convenzioni in java

1.1 Motivo per il quale esistono

Come descritto nella documentazione ufficiale di oracle, le convenzioni nello sviluppare un progetto in java sono state create per le seguenti ragioni:

- Circa l'80% del costo totale delle spese per un software durante tutta la sua durata di vita va alla manutenzione di quest'ultimo.
- Difficilmente un software viene mantenuto per tutta la sua vita sempre dallo stesso autore originale.
- Le convenzioni nel codice migliorano la leggibilità del software, consentendo agli ingegneri di comprendere il nuovo codice in modo più rapido e completo.
- Nel caso in cui qualcuno dovesse vendere il proprio codice sorgente come prodotto, deve assicurarsi che sia ben impacchettato e pulito come qualsiasi altro prodotto che creerà/abbia mai creato.

1.2 Nomi dei file

In un progetto realizzato con java, diversi file svolgono diverse funzioni, per questo motivo è importante che siano identificati con dei suffissi che indichino il loro compito all'interno del progetto.

1.2.1 Suffissi

Estensione	Ruolo
.java	Tipi di file contententi codice java puro
.class	Tipi di file contententi codice java compilato dal Java Development Kit per poter essere eseguiti dalla Java Virtual Machine

1.2.2 Prefissi

Nome	Ruolo
GNUmakefile	Il nome preferibile per i file di tipo **makefile**
README	Il nome preferibile per i file di tipo **readme**

1.3 Organizzazione dei File

Un file Java è costituito da sezioni che devono essere separate sia da righe vuote sia da un commento (facoltativo) che identifica ciascuna sezione. I file più lunghi di 2000 righe possono risultare 'ingombranti' e dovrebbero essere evitati. Nella pagina successiva è riportato un esempio di come dovrebbe essser organizzato un file java:

```java
/*
* %W% %E% Firstname Lastname
*
* Copyright (c) 1993-1996 Sun Microsystems, Inc. All Rights Reserved.
*
* This software is the confidential and proprietary information of Sun
* Microsystems, Inc. ("Confidential Information").  You shall not
* disclose such Confidential Information and shall use it only in
* accordance with the terms of the license agreement you entered into
* with Sun.
*
* SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
* THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR
* ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*/

package java.blah;

import java.blah.blahdy.BlahBlah;

/**
 * Class description goes here.
 *
 * @version      1.10 04 Oct 1996
 *  @author      Firstname Lastname
 */

public class Blah extends SomeClass {
    /* A class implementation comment can go here. */

    /** classVar1 documentation comment */
    public static int classVar1;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static Object classVar2;

    /** instanceVar1 documentation comment */
    public Object instanceVar1;

    /** instanceVar2 documentation comment */
    protected int instanceVar2;

    /** instanceVar3 documentation comment */
    private Object[] instanceVar3;

    /**
     * ...method Blah documentation comment...
     */
    public Blah() {
        // ...implementation goes here...
    }

     /**
      * ...method doSomething documentation comment...
      */
    public void doSomething() {
        // ...implementation goes here...
    }

      /**
       * ...method doSomethingElse documentation comment...
       * @param someParam description
       */
    public void doSomethingElse(Object someParam) {
        // ...implementation goes here...
    }
}
```

1.3.1 File .java

Ciascun file sorgente Java dovrebbe contenere una singola classe o interfaccia pubblica. Quando le classi private e le interfacce sono associate a una classe pubblica, possono essere inserite nello stesso file di origine della classe pubblica. La classe pubblica dovrebbe essere la prima classe o interfaccia nel file. Come riportato nell'immagine precedente, i file sorgenti Java dovrebbero essere organizzati nel seguente modo:

- Commenti iniziali, strutturati a loro volta nel seguente modo:

```
/*
 * Classname
 *
 * Version info
 *
 * Copyright notice
 */
```

- Istruzioni Package e Import, come ad esempio:

```
package java.awt;

import java.applet.Applet;
import java.awt.*;
import java.net.*;
```

- Dichiarazioni di classi e interfacce, come riportato nell'immagine precedente.

1.3.2 Indentazione

L'indentazione è un metodo di formattazione del codice con l'obiettivo di renderlo più leggibile. Quattro spazi dovrebbero essere usati come unità di misura per rappresentare 1 indentazione. L'esatta costruzione dell'indentazione (spazi oppure tab) non è specificata, tuttavia le tabulazioni dovrebbero essere impostate esattamente ogni 8 spazi (e non 4 come di Default).

```
//Indentazione Convenzionale
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ...
}

//Si indenta di 8 spazi per evitare indentazioni troppo 'profonde'
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

```
if ((condition1 && condition2)
        || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

//oppure si può anche scrivere in questo modo
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

//altri modi accettati di scrivere le espressioni booleane
//alpha: nome variabile
//beta: se l'espessione è vera
//gamma: se l'espressione è falsa
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

1.3.3 Lunghezza delle righe

Ogni riga del file non dovrebbe contenere più di 80 caratteri, in quanto non verrebbero gestite bene da molti terminali e strumenti.

1.3.4 Quando andare a capo

Quando un'espressione non entra perfettamente in una singola riga, è meglio spezzala seguendo le seguenti regole:

- Vai a capo subito dopo una virgola o un punto e virgola.
- Vai a capo appena prima di inserire un operatore (aritmetico o booleano che sia).

```
longName1 = longName2 * (longName3 + longName4 - longName5)
          + 4 * longname6; // Esempio da seguire

longName1 = longName2 * (longName3 + longName4
                       - longName5) + 4 * longname6; // Esempio da evitare
```

- È preferibile interrompere la righa ad un livello sintattico superiore della stessa, come ad esempio alla fine di un'istruzione o un'espressione, rispetto a interromperla ad un livello sintattico inferiore, come ad esempio nel mezzo di un'espressione o all'interno di una serie di parentesi.

```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);

var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```

1.4 Commenti

I programmi Java possono avere due tipi di commenti: commenti di **implementazione** e commenti di **documentazione**.
I commenti di implementazione sono quel tipo di elementi che troviamo anche in altri linguaggi, come ad esempio il C++, delimitati da **/*multi-line comment*/** e //.
I commenti alla documentazione, invece, sono un esclusiva del linguaggio Java e sono delimitati da **/**...*/**. Questi ultimi possegono la caratteristica di poter essere successivamente estratti in un file HTML utilizzando lo strumento javadoc.
La discussione delle decisioni di progettazione non banali o non ovvie è appropriata, ma è meglio evitare di duplicare le informazioni che sono presenti (e già chiare) nel codice. È molto facile che i commenti ridondanti diventino obsoleti. In generale, è melgio evitare qualsiasi commento che potrebbe diventare obsoleto man mano che il codice si evolve, la frequenza dei commenti intatti a volte può riflettere la scarsa qualità del codice. Quando ci si sente come 'obbligati' ad aggiungere un commento, si prenda in considerazione la possibilità di riscrivere il codice per renderlo più chiaro. I commenti non devono essere racchiusi in grandi riquadri come riportato sopra e non devono mai includere caratteri speciali come form-feed e backspace.

1.4.1 Commenti di Implementazione

I commenti di implementazione servono per commentare il codice o per commenti sulla sua effettiva implementazione. I commenti di documentazione, invece, hanno lo scopo di descrivere le specifiche del codice, da una prospettiva totalmente priva di implementazione, in modo da poter essere letto da sviluppatori che potrebbero non avere necessariamente il codice sorgente a portata di mano. I programmi possono avere quattro stili di commenti di implementazione:

- **Commenti a blocchi**: utilizzati per fornire descrizioni di file, metodi, strutture dati e algoritmi. I commenti di blocco dovrebbero essere usati all'inizio di ogni file e prima di ogni metodo. Possono essere utilizzati anche in altri luoghi, ad esempio all'interno dei metodi. I commenti a blocchi all'interno di una funzione o di un metodo devono essere rientrati allo stesso livello del codice che descrivono. Un commento di blocco dev essere preceduto da una riga vuota per distinguerlo dal resto del codice.

```
/*
 * Here is a block comment.
 */
```

I commenti di blocco possono iniziare con /*-, riconosciuto da indent(1) come l'inizio di un commento di blocco che non dovrebbe essere riformattato. Esempio: /*-* Here is a block comment with some very special * formatting that I want indent(1) to ignore. * * one * two * three */ https://www.oracle.com/java/technologies/javase/codeconventions-comments.html

1.4.2 Commenti di Documentazione

I commenti di documentazione dovrebbero essere utilizzati per fornire una panoramica del codice e fornire informazioni aggiuntive che non sono prontamente disponibili nel codice stesso. Questo tipo di commenti devono contenere solo informazioni rilevanti per la lettura e la comprensione del programma.