CODING RULES

LE REGOLE DI CODIFICA

Le regole di codifica sono un insieme di regole per standardizzare le pratiche di sviluppo software , diffondere lo sviluppo delle migliori pratiche ed evitare errori di sviluppo "classici" in un gruppo di sviluppatori .

Le aziende hanno spesso proprie regole di codifica o si appoggiano a regole di codifica standard.

Tra le più usate:

Le **MISRA** Nate in ambito automotive che si applicano solo ai linguaggi C e C++ poiché sono nate per sviluppi embedded che possono generare problematiche di sicurezza sono molto stringenti e si possono dividere in due categorie:

- Quelle che impongono di usare (o più spesso non usare) alcuni costrutti che potrebbero far generare errori, per esempio il comando goto o la «malloc»
- Quelle che dettano uno stile di programmazione

Le **Google** che si applicano a diversi linguaggi (C, C++, Java, HTML, …) e che hanno regole solo della seconda categoria, infatti si chiamano **«Google Style Guide»**

LE GUIDE DI STILE

Un programmatore produce centinaia di programmi, solitamente lavorando in un gruppo di sviluppo in collaborazione con altri professionisti, e nello sviluppo di un sistema lo stesso segmento di codice deve essere "ripreso in mano" più volte, sia dall'autore che da altri componenti del gruppo di lavoro.
È necessario che tutto il gruppo di lavoro adotti un insieme di tecniche e di regole di scrittura del codice per stilare una corretta e completa documentazione del software per facilitare e supportare molte fasi del ciclo di vita:

- testing: la leggibilità del codice sorgente agevola la sua verifica e aiuta nella ricerca degli errori;
- reverse engineering e re-engineering: permette il riutilizzo del codice in problemi differenti e la cooperazione e scambio di routine tra programmatori;
- integrazione: favorisce la definizione delle interfacce per interagire con i sistemi preesistenti;
- manutenzione: consente la manutenzione rapida ed efficiente del codice anche a distanza di tempo, per aggiungere nuove funzionalità, modificare funzionalità esistenti, correggere gli errori o migliorare le prestazioni.

Chiaramente devono essere definite prima di iniziare lo sviluppo

LE GUIDE DI STILE: MACRO TIPOLOGIE DI REGOLE

- Le regole che riguardano il «naming», cioè i nomi di variabili, classi…
- Le regole che spiegano come fare i commenti, dove farli e cosa scrivere in essi
- Le regole che riguardano la formattazione: dove mettere le parentesi, gli spazi, l'indentazione…
- Le regole che evitano codici troppo complessi: limitazione sul numero di righe per metodo, sull'annidamento (*)…

(*) l'inserimento di una struttura di controllo all'interno di un'altra in un programma. Un caso tipico è quello dell'iterazione su due o più variabili, come nel seguente esempio (scritto in linguaggio C):

```c
/* moltiplicazione tra matrici: A = B*C */

for (i=0; i<n; i++) {        /* ciclo più esterno */
  for (j=0; j<m; j++) {      /* ciclo intermedio */
    a[i][j] = 0.0;
    for (k=0; k<l; k++) {    /* ciclo più interno */
      a[i][j] += b[i][k] * c[k][j];
    }
  }
}
```

Google Java Style Guide

Table of Contents

Di seguito la descrizione di alcune regole di esempio

https://google.github.io/styleguide/javaguide.html

CAPITOLO 2 ELEMENTI BASE DI CODIFICA

Dalla guida

2.3.1 Whitespace characters

Aside from the line terminator sequence, the **ASCII horizontal space character** (**0x20**) is the only whitespace character that appears anywhere in a source file. This implies that:

1. All other whitespace characters in string and character literals are escaped.
2. Tab characters are **not** used for indentation.

L'unico modo per realizzare lo spazio bianco deve essere la barra spaziatrice, quindi vanno evitati altri caratteri che realizzano lo spazio bianco e il tab per l'indentazione

4 FORMATTAZIONE

Dalla guida

4.1.1 Use of optional braces

Braces are used with `if`, `else`, `for`, `do` and `while` statements, even when the body is empty or contains only a single statement.

4.2 Block indentation: +2 spaces

Each time a new block or block-like construct is opened, the indent increases by two spaces. When the block ends, the indent returns to the previous indent level. The indent level applies to both code and comments throughout the block. (See the example in Section 4.1.2, Nonempty blocks: K & R Style.)

4.1.1 - In if..else, for , do e while si devono usare le {} anche se il corpo è vuoto o con una sola istruzione

4.2 – ogni volta che inizia un nuovo blocco o costrutto che necessita indentazione, essa deve essere fatta con 2 spazi

4 FORMATTAZIONE

Dalla guida

4.3 One statement per line

Each statement is followed by a line break.

4.4 Column limit: 100

Java code has a column limit of 100 characters. A "character" means any Unicode code point. Except as noted below, any line that would exceed this limit must be line-wrapped, as explained in Section 4.5, Line-wrapping.

4.3 – deve essere inserito un solo comando per linea
4.4 – la linea deve avere un massimo di 100 caratteri

5 NOMI

Dalla guida

5.2.1 Package names

Package names use only lowercase letters and digits (no underscores). Consecutive words are simply concatenated together. For example, `com.example.deepspace`, not `com.example.deepSpace` or `com.example.deep_space`.

5.2.2 Class names

Class names are written in UpperCamelCase.

Class names are typically nouns or noun phrases. For example, `Character` or `ImmutableList`. Interface names may also be nouns or noun phrases (for example, `List`), but may sometimes be adjectives or adjective phrases instead (for example, `Readable`).

There are no specific rules or even well-established conventions for naming annotation types.

A *test* class has a name that ends with `Test`, for example, `HashIntegrationTest`. If it covers a single class, its name is the name of that class plus `Test`, for example `HashImplTest`.

5.2.1 i package devono essere scritti in lettere minuscole senza altri caratteri per esempio «_»
5.2.2 le classi devono essere scritte con lettere minuscole dove la maiuscola divide parole diverse «HashIntegrationTest»

5 NOMI

Dalla guida

5.2.3 Method names

Method names are written in lowerCamelCase.

Method names are typically verbs or verb phrases. For example, `sendMessage` or `stop`.

Underscores may appear in JUnit *test* method names to separate logical components of the name, with *each* component written in lowerCamelCase, for example `transferMoney_deductsFromSource`. There is no One Correct Way to name test methods.

5.2.3 – Nomi dei metodi come per le classi, ma iniziando con la lettera minuscola

7 JAVADOC

Come descritto nel laboratorio 1 a pag 362-365 del libro