1 Convenzioni del codice da utilizzare

Stabilito pro nobis et inter nos, e soprattutto che sia funzionale, perché "Avere un buono standard di programmazione è meglio che non averne nessuno. Tuttavia, [...] uno standard di programmazione scadente è peggio che non averne affatto" - Bjarne Stroustrup paragrafo "Can you recommend a coding standard?".

1. 4 spazi.
2. CamelCase per i nomi delle classi, dromedaryCase per i nomi delle variabili e snake_case per i nomi delle funzioni.
3. A costo di essere verbosi i nomi di variabili e funzioni devono essere autoesplicativi.
4. Le costanti in SNAKE_CASE ma con solo maiuscole.
5. Dopo un `if`, `while`, dichiarazione di funzione, et cetera, si va a capo con una graffa e poi si va a capo di nuovo. Idem per le parentesi tonde.
6. Non innesare più di tre livelli, inverti ed estrai.
7. Se nella dichiarazione di una funzione gli argomenti superano gli 80 caratteri andare a capo dopo la parentesi aperta e per ogni parametro dopo di essa.
8. Qualora si passasse "by reference" un valore ad una funzione tramite un puntatore, il puntatore nella funzione avrà in coda `_ptr`. -¿ non negoziabile, non so java non farmi ammattire
9. Non compilare automaticamente, ogni package deve avere un `MAKEFILE` che permetta la compilazione a prescindere dalle tecnologie usate, sia l'editor di testo o sistema operativo.
10. Tutti i nomi saranno in inglese.

Esempio punto 2 per camel case e compagnia, 5 e 7 per le regole di ritorno a capo e 8 per i pass by reference.

```
public class Test
{
    private ArrayClass somethingArray = new ArrayClass;
    Integer get_first_value_over_threshold(
        Array array_ptr,
        int threshold
    ) {
        lorem_ipsum();
        return dolor_sit_amet();
    }
}
```

Esempio del punto 3 per i ritorno a capo prima delle graffe;

```
public class Test
{
    int foo()
    {
        something_something();
        return x;
    }
}
```

Esempio del punto 5 di pratiche da NON seguire:

```
void register_user([...]) {
    UserLoginFields userLoginFields = [...]; // ha 2 campi, id e password
    Server server = [...];
    User user;
    if(userLoginFields.getFieldsCount() == 2) {
        if(userLoginFields.getId() >= 0) {
            if(server.login(userLoginFields)) {
                user = server.getLoginData();
            } else {
                throw new InvalidCredentialsError(
                    UserLoginFields.INVALID_CREDENTIALS_ERROR_MSG);
            }
        } else {
            throw new InvalidIDError(
                UserLoginFields.INVALID_ID_ERROR_MSG);
        }
    } else {
        throw new InvalidFieldsCountError(
            UserLoginFields.INVALID_FIELDS_COUNT_ERROR_MSG);
    }
}
```

Come fare:

```
void register_user([...]) {
    UserLoginFields userLoginFields = [...]; // ha 2 campi, id e password
    Server server = [...];
    User user;
    if(!userLoginFields.getFieldsCount() == 2) {
        throw new InvalidFieldsCountError(
            UserLoginFields.INVALID_FIELDS_COUNT_ERROR_MSG);
    }
    if(!userLoginFields.getId() >= 0) {
        throw new InvalidIDError(
            UserLoginFields.INVALID_ID_ERROR_MSG);
    }
    if(!server.login(userLoginFields)) {
```

```
            throw new InvalidCredentialsError(
                UserLoginFields.INVALID_CREDENTIALS_ERROR_MSG);
        }
        user = server.getLoginData();
    }
```

2 Convenzioni sullo stile di codice

Per rendere più facile l'analisi - preso da The Power of 10: Rules for Developing Safety-Critical Code.

1. Evitare `goto`, ricorsione ed altre pratiche che rendono difficile seguire il flusso di esecuzione.
2. Tutti i cicli for devono avere un upper bound hard coded per impedire che il codice diverga con un'esecuzione incontrollata.
3. Evita di allocare elementi sullo heap dopo l'inizializzazione // abrogata per come è strutturato java.
4. Restringi le funzioni ad una singola pagina stampata - non più di 60 righe commenti inclusi.
5. Usa in media almeno due asserzioni a runtime per funzione // questo va bene nei test, ma una volta testato rimuoverli.
    - Per abilitare gli assert: `java -ea MyClass`
    - Per disabilitare gli assert: `java -da MyClass`
6. Riduci l'ambito dei dati al minimo possibile (va a braccetto con il pattern `Information Expert`).
7. Controllare il valore di ritorno di tutte le funzioni non void, oppure eseguire un cast a void per indicare che il valore di ritorno è inutile.
8. Utilizzare il preprocessore solo per file header e macro semplici. // sebbene l'uso del preprocessore in java sia limitato
9. Limitare l'uso dei puntatori a una sola dereferenziazione, e non utilizzare puntatori a funzione. // non usare proprio i puntatori di funzioni, rendere espliciti quelli dedgli oggetti
10. Compilare con tutti i possibili warning attivi; tutti i warning devono poi essere risolti prima del rilascio del software.