**Erreurs de code et les corrections**

**J’ai pris les données sur cette page :**

[**http://shitcode.net/index/language/java**](http://shitcode.net/index/language/java)

**Exemple 1 : if / else inutile et code répétitif**

Dans tous les cas il souhaite retourner true, donc pourquoi mettre un if else statement ?

public static returnTrue(boolean b){

if (b){

return true;

} else {

return true;

}

}

**Exemple 2 : fonction trop longue et mal réflechie**

* Faire du code court. Cette méthode est bien trop longue et surtout mal pensée.

private static int alphabetToNumber(String letter) {

String s = letter.toLowerCase();

if (s.equals("a")) {

return 1;

} else if (s.equals("b")) {

return 2;

} else if (s.equals("c")) {

return 3;

} else if (s.equals("d")) {

return 4;

} else if (s.equals("e")) {

return 5;

} else if (s.equals("f")) {

return 6;

} else if (s.equals("g")) {

return 7;

} else if (s.equals("h")) {

return 8;

} else if (s.equals("i")) {

return 9;

} else if (s.equals("j")) {

return 10;

} else if (s.equals("k")) {

return 11;

} else if (s.equals("l")) {

return 12;

} else if (s.equals("m")) {

return 13;

} else if (s.equals("n")) {

return 14;

} else if (s.equals("o")) {

return 15;

} else if (s.equals("p")) {

return 16;

} else if (s.equals("q")) {

return 17;

} else if (s.equals("r")) {

return 18;

} else if (s.equals("s")) {

return 19;

} else if (s.equals("t")) {

return 20;

} else if (s.equals("u")) {

return 21;

} else if (s.equals("v")) {

return 22;

} else if (s.equals("w")) {

return 23;

} else if (s.equals("x")) {

return 24;

} else if (s.equals("y")) {

return 25;

} else if (s.equals("z")) {

return 26;

} else {

return 0;

}

}

Une façon plus rapide de le faire :

**public** **int** alphabetToNumber(String letter){

letter=letter.toLowerCase();

String alphabet="abcdefghijklmnopqrstuvwxyz";

**return** alphabet.indexOf(letter)+1;

}

**Exemple 3 :**

Problème numéro 1 : RESULT\_OF\_DIVISION\_BY\_0 = 9.9 🡺 la division par 0 est interdite et dans tous les cas, ce n’est jamais égal à 9.99. Donc il faut éviter la désinformation

Ensuite le nom de la fonction n’est pas correct par rapport au calcul. Si on multiplie par 1, on n’obtient pas un pourcentage. Il faut donc multiplier par 100.00 pour obtenir le pourcentage de différence

Le cas ou previousResult=0 et currentResult=0 est mal pensé. Je pense qu’il voulait dire if (previousResult == currentResult), ce qui de toute facon est inutile puisque pris en compte par le calcul du else.

Ensuite par habitude je pense qu’il vaut mieux mettre le previousResult en premier dans les arguments

private static final double RESULT\_OF\_DIVISION\_BY\_0 = 9.99;

public static double getPercentageDifference(long currentResult, long previousResult) {

if (previousResult == 0 && currentResult == 0) {

return 0;

} else if (previousResult == 0) {

return RESULT\_OF\_DIVISION\_BY\_0;

} else {

return (currentResult - previousResult) \* 1.0 / previousResult;

}

}

Pour éviter la division par 0, le mieux est d’interdire en amont de mettre un argument qui entraîne ce souci et de faire une règle de gestion pour le cas où previousResult serait égal à 0. Ensuite il faudrait peut être envisager de prendre en considération les cas négatifs (à voir si le jeu de données le permet ou pas)

If (previousResult !=0){

public double getPercentageDifference(long previousResult, long currentResult) {

return (currentResult - previousResult) \* 100.0 / previousResult;

}

}

**Exemple 4 : compliqué quand simple**

Faire compliqué quand on peut faire simple (qui plus est, en étant faux) quand cela n’est pas nécessaire.

int length = 0;

for(int idx = 0; idx < a.length; i++) {

length++;

}

System.out.println("length is : " + length);

Il suffit juste d’utiliser a.length sachant que de toute façon ce bout de code est faux (notamment le i++ au lieu du idx++).

**Exemple 5 : noms significatifs**

Code source :

dotychczasowyPostep += predkosc \* 1;

Je ne parle pas cette langue mais il semblerait que ce soit du Polonais. A priori dotychczasowy Postęp = Progrès précédent et predkosc = vitesse ?!?!?! Meme traduit j’ai du mal à comprendre ce que cela veut dire …

Pour des polonais cela peut passer mais il faut éviter de mettre des noms de variable dans des langues que peu parle, le mieux étant de nommer ses variables en anglais. Cela peut également rentrer dans la catégorie « choisir des noms prononçables » et « choisir des noms compatibles avec une recherche » par la même occasion.

Par ailleurs en dehors de cette dimension de code aux noms non significatifs, il y a un souci véritable au niveau du code : il n y a absolument aucun intérêt à multiplier par 1 parce que ca redonnera toujours predkosc donc déjà il faudrait écrire : dotychczasowyPostep += predkosc ;Ou bien sinon l’auteur à oublié une valeur derrière comme par exemple 1,1

Après naturellement il faudrait renommer les variables mais là j’ai un peu de mal à savoir ce que c’est censé représenté donc je ne suis pas capable de donner des noms alternatifs

**Exemple 6 : compliqué quand simple**

Autre exemple de comment faire compliqué quand on peut faire simple sans compter que dans ce cas on implémente du code qui va ralentir de façon importante l’exécution totale de notre code. Si par exemple on recherche un numéro du type 07xxxxxxxx, cela prendre pas mal de temps avant d’atteindre cette valeur. Naturellement ce code est tout à fait inutile puisqu’il suffisait de faire return phoneNum. Après l’utilité d’une telle fonction se discute aussi.

int phoneNumberFinder(int phoneNum){

int num = 0;

while (true){

if(phoneNum == num){

return num;

else{

num++;

}

}

}

**Exemple 7 : gestion des erreurs**

try{

//...

getMyDataFromServer();

//...

}catch(Excepion e){

// :D

}

Premièrement exception est mal écrit. Ensuite si on prend la peine de faire un try catch, il faut le faire jusqu’au bout.

On pourrait faire quelque chose du genre :

try{

//...

getMyDataFromServer();

//...

}catch(Exception e){

System.out.println(e.getMessage());

}