# Projet HPP – Coronavirus

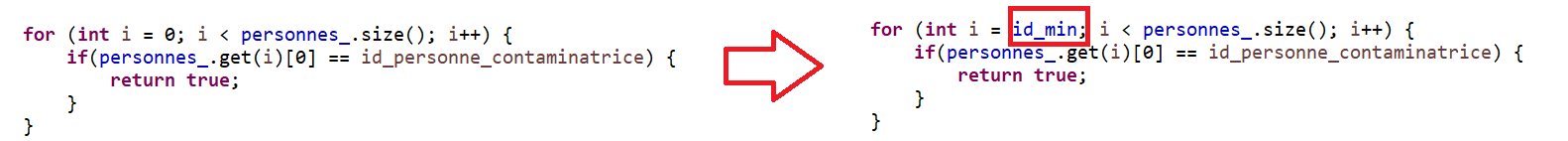
## Historique des optimisations

### Hypothèses

* Nous considérons que le virus reste confiné dans chaque pays. (La gestion du cas international se fait en enlevant simplement une condition)
* Lorsque la personne contaminatrice du cas traité durant l’itération, l’a contaminé il y a plus de 14 jours. Nous créons une nouvelle chaine.

### Optimisation 1

Dans la méthode « presenceIdPersonneContaminatrice » :



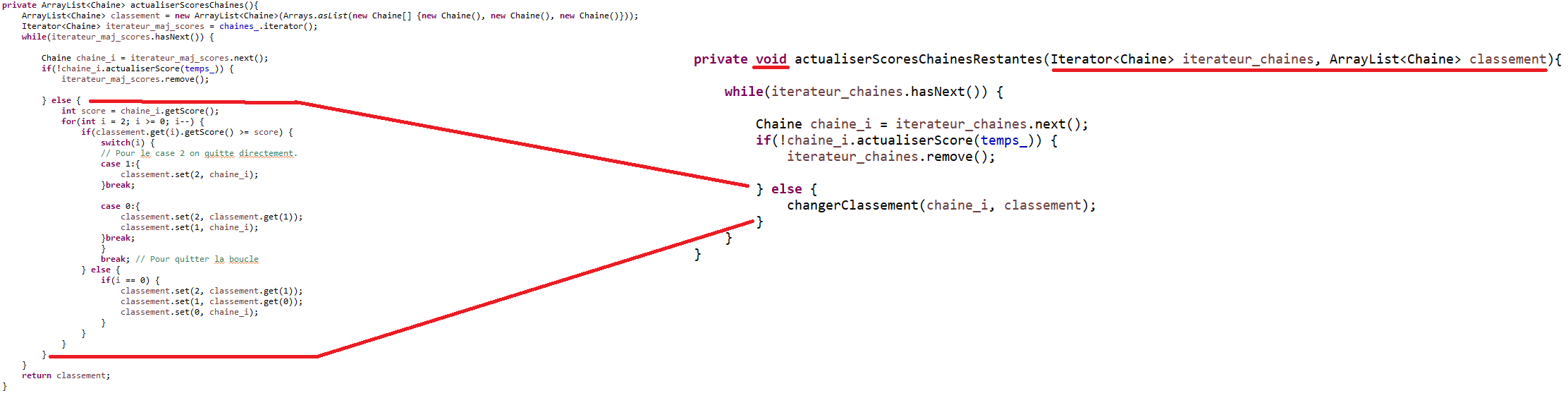
« id\_min » représentant la dernière personne dont le score est = 0.

### Optimisation 2

Nous avons fait en sorte que lorsque nous traitons le cas d’une nouvelle personne, nous devons soit l’ajouter à une chaine existante si nous parvenons à trouver la personne qui l’a contaminée, soit créer une nouvelle chaine avec elle. Comme nous devons parcourir toutes les chaines nous en profitons pour actualiser leur score et le classement. Ensuite la méthode « actionNouveauCas » retourne à présent un itérateur qui permet à la méthode nouvellement renommer « actualiserScoresChainesRestantes » de mettre-à-jour les chaines qui ne l’auraient pas été grâce à l’itérateur retourné.

La grosse optimisation est lorsque nous créons une nouvelle chaine, nous supprimons un parcours entier de la LinkedList<Chaine> qui était fait dans « actualiserScoresChaines ».







### Optimisation 3

Très grosse optimisation qui nous a permis de terminer le traitement du fichier de 1000000 dans un temps raisonnable (2h environ). Le programme est toujours monothread. Nous avons remarqué que La manière dont nous récupérions les données était loin d’être optimale. Selon JvisualVM elle occupe près de 70 % du temps d’exécution. Effectivement à chaque itération, notre programme récupérait les données des lignes suivantes de chaque fichier. (3 accès fichiers). Là où nous avons optimisé c’est qu’à présent, nous n’avons qu’un seul accès fichier. On ne chance à chaque itération, que la donnée du pays qui avait le temps le plus vieux.

De plus, avant nous refermions chaque fichier à chaque fois et nous n’utilisions qu’une seule instance de la classe CSVReader. Maintenant, Chaque fichier à son instance de CSVReader qui effectue une lecture avec des BufferedReader. A chaque itération, le buffer lit la ligne suivante et le buffer n’est fermé que dès qu’il a parcouru tout le fichier.

Si nous devions travailler avec des données qui viendraient s’ajouter au fur et à mesure de l’épidémie à la fin du fichier ceci empêcherait l’accès en écriture. Mais on peut supposer que si nous devions travailler conjointement avec un programme de ce type, avant d’écrire, une « interuption » serait envoyer à un thread à part de notre programme pour :

1. Envoie d’une attente à l’émetteur
2. Fin de l’itération actuelle
3. Fermeture du fichier
4. Envoie d’un signal à l’émetteur pour lui indiquer qu’il peut écrire
5. Attente du signal de fin d’écriture de l’émetteur.
6. Reprise du traitement.

Bien entendu, cela exigerait des fonctions supplémentaires que nous n’avons pas créées car là n’est pas le but du projet.