BE 2 : JAVA CONCURRENT

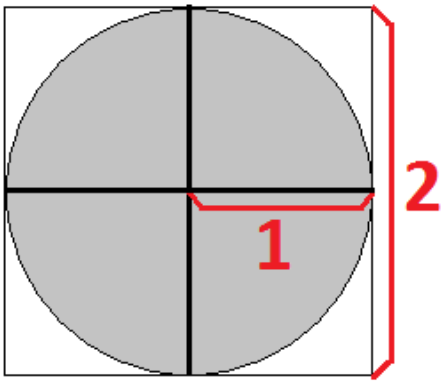
Emeline GOT

Eloy Jesus del Gran Poder INSUZA DIAZ

Sommaire

1 Calcul de Pi par cercle unitaire

Nous avons développé un programme en java pour calculer la valeur de Pi par la méthode de Monte Carlo. Pour calculer une aire, la méthode de Monte Carlo inscrit l'aire à calculer dans un rectangle. Des tirages au hasard sont faits dans le rectangle. Sachant la proportion de tirages tombés dans l'aire à calculer et l'aire du rectangle, l'aire inconnue est le produit de ces deux valeurs. Puisque l'aire d'un cercle de rayon vaut Pi*r^2, nous avons utilisé un cercle de rayon unitaire inscrit dans un rectangle.

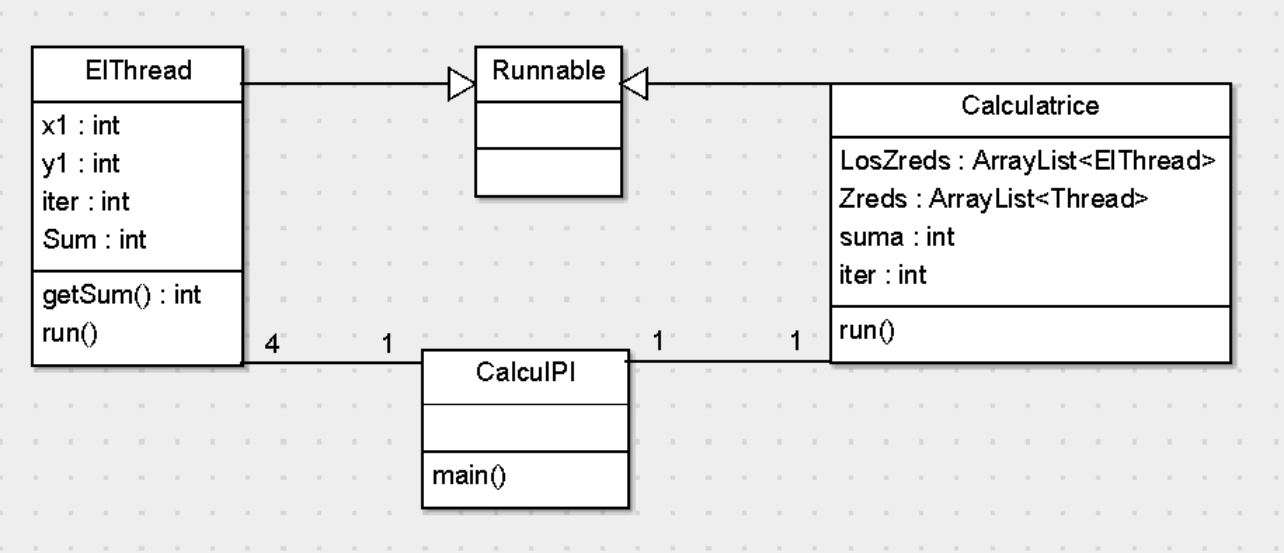


La valeur de Pi est alors déterminée par la formule :

$$\pi = \frac{nombre\ de\ tirages\ dedans}{nombre\ de\ tirages\ total\ x\ aire\ du\ rectangle}$$

Où l'aire du rectangle vaut 4.

Pour rendre ce programme plus rapide nous avons divisé le calcul de Monte Carlo entre quatre threads. Un dernier thread attend qu'ils finissent pour faire le calcul final de Pi. Nous avons utilisé trois classes : la classe **CalculPI** qui contient le main, la classe **ElThread** qui effectue le calcul de la méthode de Monte Carlo dans chaque quadrant et la classe **Calculatrice** qui obtient la valeur de Pi.



1.1 CalculPi

Cette classe contient le main. La variable *iter* détermine combien de tirages chaque thread fera. Deux ArrayLists existent : un pour les **Threads** (*quadrants*) et un autre pour les **ElThread**

(*losthreads*). Quatre Threads de type **ElThread** sont créés et démarrés. Un dernier Thread de type **Calculatrice** est aussi démarré.

1.2 ElThread

Cette classe hérite de **Runnable**. Chaque **ElThread** calcule Monte Carlo dans un quadrant. Ses attributs sont deux entiers *x1* et *y1* pour déterminer le quadrant, un entier *iter* avec les itérations à réaliser et un double *Sum* pour compter les tirages réussis.

La fonction **getSum** permet d'obtenir la valeur de la variable *Sum*.

La fonction **run** commence lorsqu'on démarre le Thread. Elle utilise une boucle for qui se répète *iter* de fois. A chaque fois, elle crée deux nombres aléatoires pour les coordonnées x et y appartenant au quadrant où elle travaille. Ensuite, elle calcule si le point est dans le cercle inscrit (la distance au centre plus petite que 1). Dans le cas où le point est dedans, la fonction augmente *Sum* d'une unité.

1.3 Calculatrice

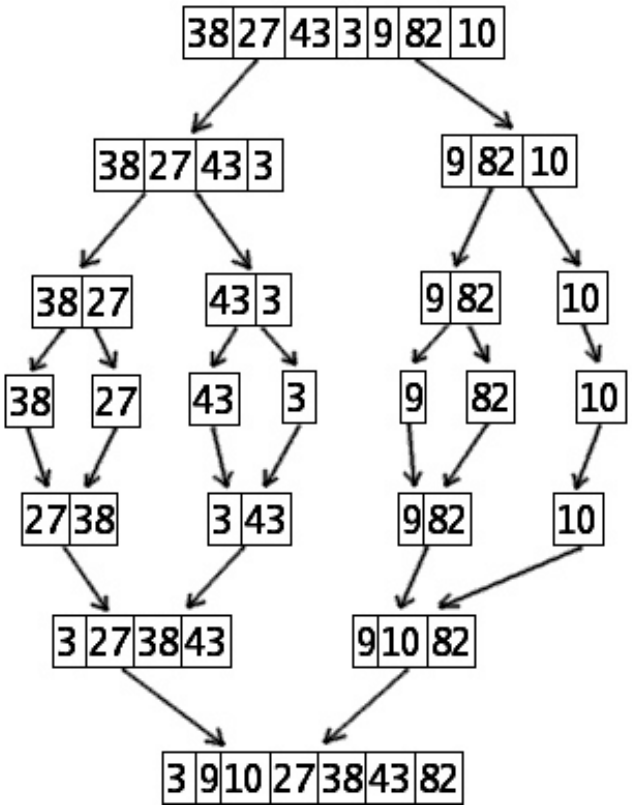
Cette classe hérite de **Runnable**. Elle calcule la valeur de Pi à partir des données des **ElThreads**. Ses attributs sont deux ArrayLists, pour les ElThread (*LosZreds*) et les Threads (*Zreds*), un entier *suma* pour calculer les tirages réussis et un entier *iter* pour le nombre d'itérations.
La fonction **run** commence lorsqu'on démarre le Thread. D'abord elle attend que tous les **ElThreads** finissent en utilisant **join**. Ensuite, une boucle for parcourt la liste de **ElThread** pour obtenir la somme de tirages réussis. Finalement, elle calcule Pi comme le nombre de tirages réussis divisé par le nombre de tirages totaux et elle l'affiche.

2 Tri Fusion

Le principe du tri fusion repose sur la séparation du tableau en deux parties. Les deux parties du tableau sont triées séparément. Il suffit ensuite de fusionner les deux parties pour obtenir le tableau entier trié.

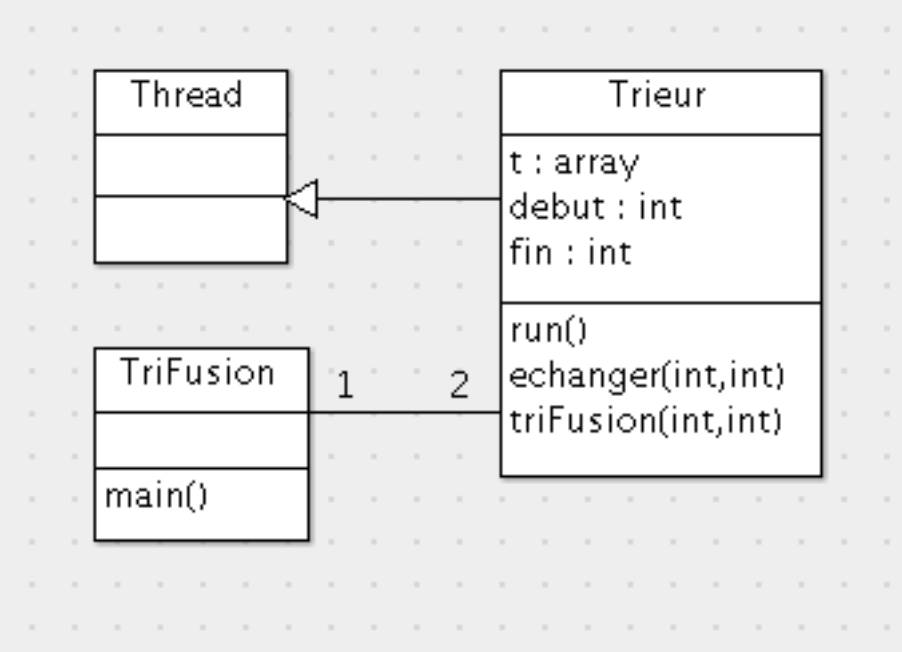
L'algorithme de séparation s'effectue jusqu'à ce que l'on obtienne uniquement deux éléments dans le tableau. On peut alors les trier par simple comparaison et on remonte progressivement au tableau complet à l'aide de la fusion progressive de parties du tableau.

Ce principe est illustré sur l'image suivante :



Dans notre cas, nous avons recours à des threads pour pouvoir traiter les deux parties du tableau en parallèle et donc augmenter la vitesse de tri. Deux threads seront associés chacun à une partie du tableau. Une fois les deux parties triées, on procède à leur fusion.

On utilise deux classes : une classe **TriFusion** qui contient le main et permet l'exécution du tri ; et une classe **Trieur** qui effectue la méthode de tri fusion.



2.1 TriFusion

Cette classe contient le main. La variable *t* est le tableau à trier. Le Trieur *trieur* est le thread qui est créé et qui permet le tri fusion. Une fois le tri effectué, on affiche le tableau trié dans la console.

2.2 Trieur

Cette classe hérite de **Thread**. Elle prend comme argument un tableau *t* à trier, ainsi que des valeurs de début et de fin qui délimitent la zone à trier.

Sa méthode **run** est appelée lorsque l'on lance le thread dans **TriFusion**. La fonction teste dans un premier temps si la zone à trier est de taille inférieure à 2. Auquel cas, on range simplement les deux éléments dans l'ordre avec une comparaison et la fonction **echanger**. Sinon, on repère le milieu du tableau et on crée deux threads de type **Trieur**. Le premier pour trier la première partie du tableau (avant *milieu*), le second pour trier la seconde partie (après *milieu*). On teste si les threads sont terminés avec la méthode **join** et ensuite on procède à la fusion des deux parties en faisant appel à la fonction **triFusion** de la classe **Trieur**.

La fonction **echanger** permet d'échanger deux éléments dans un tableau.

La fonction **triFusion** va permettre la fusion de deux tableaux en conservant l'ordre croissant. Pour cela, on crée un nouveau tableau que l'on va remplir dans l'ordre convenable. On considère deux indices *i1* et *i2* qui vont parcourir chacun une partie et un indice *i* qui va parcourir le tableau à remplir. La fonction compare les éléments des deux parties de tableau et place le plus petit dans le tableau. Une fois que l'une des parties de tableau est vidée, on complète le tableau avec le reste de l'autre partie. Enfin, on recopie le tableau trié dans *t*.