**Analyse de l’exercice #1**

**E.26** : Le tableau suivant démontre les avantages et les inconvénients de l’utilisation des cinq différentes conceptions d’algorithmes pour les conversions des coordonnées polaires et cartésiennes et le sauvegarde de celles-ci.

|  |  |  |
| --- | --- | --- |
| **Conception** | **Avantages** | **Inconvénients** |
| 1 – Cette conception permet de sauvegarder un type de coordonnées en utilisant une paire de variables d’instances | * Création de moins de variables d’instances comme les valeurs de x et rho sont sauvegarde dans la même variable, de la même manière, y et thêta sont aussi stocké dans la même variable, donc moins d’espaces mémoire occupée. | * Le manque de raffinemment puisqu’avec l’utilisation de seulement deux variables d’instances, on doit stocker les coordonnées cartésiennes et polaires dans le même type de variables. |
| 2 – Cette conception restore les coordonnées polaires et retourne simplement les coordonnées cartésiennes | * Les conceptions 2 et 3 sont similaires dans l’implémentation alors ils partagent le même avantage qui est : * La simplification du code avec l’utilisation de deux variables ce qui économise aussi de l’espace mémoire. | * L’inconvénient des deux conceptions sont les calculs qui doivent être effectués à chaque fois pour obtenir les coordonnées en polaires et cartésiennes. |
| 3 – Cette conception restore les coordonnées cartésiennes et elle retourne les coordonnées polaires |
| 4 – Cette conception permet la sauvegarde des deux types de coordonnées soit elle sauvegarde séparément les coordonnées polaires et les coordonnées cartésiennes | * Les calculs seront ainsi plus efficaces à effectuer comme les coordonnées seront entrées dans les variables appropriées. | * Utilisation de plus d’espaces mémoires avec les quatre variables d’instances. |
| 5 – Cette conception utilise une classe abstraite qui a comme sous-classe la conception 2&3. | * L’utilisation de la superclasse permet une bonne organisation et une plus grande flexibilité au changement. * Cette classe abstraite n’aurait pas de méthodes initialisée, ce qui signifie moins d’occupation dans l’espace mémoire. | * Comme cette superclasse utilise les deux conceptions 2 et 3, les calculs vont nécessiter que chaque conception étendant la conception 5 devra convertir ses coordonnées pour obtenir l’autre type de coordonnées non stocké. |

**E.28 :**

À remplir avec comparaison de design 5 et 1.

On implemente la classe abstraite du design 5 et on run un PerformanceRecorder dont les resultats sont

For 21154220 iterations in 10 seconds we have:

maxTime = 4562100 nanoseconds

minTime = 300 nanoseconds

medianTime = 366 nanoseconds

En comparaison, ceux du design 1 sont :

For 9109210 iterations in 10 seconds we have:

maxTime = 3889100 nanoseconds

minTime = 700 nanoseconds

medianTime = 987 nanoseconds

Ainsi, le design 5 et 1 auront une difference efficacité de 621 ns, avec le 5 bien plus rapide que le 1. Cela va dans le sens de l’analyse en **E.26** ou on avait noté que le manque de raffinement dans sa gestion du stockage des cordonnées était un inconvénient de ce design qui pourrait lui poser préjudice.

**E.29 :**

Voir code source dans le répertoire du groupe : [https://github.com/mchib031/SEG/blob/master/Devoir%201/pointcp/pointcp/design1(original)/PerformanceRecorder.java](https://github.com/mchib031/SEG/blob/master/Devoir)

**E.30 : Tableau de comparaison avec temps max, min et moyen**

1. Pour realiser le test on va run la class PerformanceRecorder sur les deux designs mais une fonctions publiques de PointCP a la fois.
2. Tableau échantillonnages relevés

|  |  |  |
| --- | --- | --- |
| Fonctions | Design 1 (en ns) | Design 5 (en ns) |
| toString() | 722 nanoseconds  -  12073479 iterations en 10 secondes | 142 nanoseconds  -  40705749 iterations en 10 secondes |
| rotatePoint() | 50 nanoseconds  -  64639426 iterations en 10 secondes | 90 nanoseconds  -  51587742 iterations en 10 secondes |
| getDistance() | 80 nanoseconds  -  54098094 iterations en 10 secondes | 75 nanoseconds  -  55793734 iterations en 10 secondes |
| convertStorageToCartesian() | 29 nanoseconds  -  76019511 iterations en 10 secondes | 27 nanoseconds  -  76250167 iterations en 10 secondes |
| convertStorageToPolar() | 29 nanoseconds  -  77165596 iterations en 10 secondes | 33 nanoseconds  -  73277097 iterations en 10 secondes |
| getTheta() | 27 nanoseconds  -  77236441 iterations en 10 secondes | 27 nanoseconds  -  76458028 iterations en 10 secondes |
| getRho() | 27 nanoseconds  -  77097466 iterations en 10 secondes | 27 nanoseconds  -  76457183 iterations en 10 secondes |
| getY() | 27 nanoseconds  -  77013511 iterations en 10 secondes | 27 nanoseconds  -  77028916 iterations en 10 secondes |
| getX() | 27 nanoseconds  -  75927901 iterations en 10 secondes | 27 nanoseconds  -  77261117 iterations en 10 secondes |

C)Analyse des résultats: Si le design 5 a quelques méthodes qui sont moins efficaces que le design 1 et permettront moins d’itérations, l’écart est créé par la méthode toString() qui est quasiment 5 fois plus rapides avec le design 5, le rendant le plus efficace des deux concepts.