SEG2505

Devoir 1

Soumis par:

Ricardo Saikali

Stefano Stella

Julien Martinet

Elias Maalouf

Le 5 octobre 2020

Université d'Ottawa

E26. Pros and Cons of the designs

	Pros:	Cons:
Design 1	-Compatible avec les 2 types de coordonnées (storage peut être converti) -Constructeur plus simple que les autres car rien est calculé -uses less memory than design 4	-Prend un peu plus de temp pour retourner les valeurs des méthodes getters qu'un point dédié à un type spécifique et qui utilise les méthodes getters de ce même type car on doit vérifier le type et calculer si c'est un type différent qui est demandé -code des getters plus complexe car on doit vérifier les types et possiblement calculer les valeurs de retour
Design 2	-Très rapide pour l'appel de getRho et getTheta car les donnés sont calculés une fois lors de la déclaration -Déclaration d'instance de classe plus rapide si on déclare un point avec des coordonnés polaires car aucune conversion est requise -le code pour getTheta et getRho sont simple -uses less time than design 4	- Plus lent pour getX et getY car les donnés doivent être calculés à chaque fois -Déclaration d'une instance de classe plus lente si on déclare un point avec des coordonnés cartésiens car les valeurs sont convertis et stockés -Plus lent pour le calcul de distance et rotate car on utilise les valeurs de x et y qui ne sont pas stockés et qui doivent donc être calculés à chaque fois. -Code du constructeur plus complexe car il faut vérifier le type donné et calculé les valeurs du type polaire si cartésien est donné -code pour getX et getY plus complexe car il faut calculer les
Design 3	-Très rapide pour l'appel de getX et getY car les donnés sont calculés une fois seulement et ensuite stockés lors de la déclaration -Déclaration d'instance de classe plus rapide si le point est initialisé avec des	-Plus lent pour getRho et getTheta car les valeurs doivent être calculés à chaque fois -Déclaration d'instance de classe plus lente si le point est initialisé avec des valeurs polaires car elles doivent être

	valeurs cartésiennes car aucune valeur est calculée. -Plus rapide pour le calcul de distance car la distance est calculée à l'aide de x et y qui sont déjà stockés -méthode rotate plus rapide car les valeurs x et y sont déjà calculés dans le constructeur -uses less memory than design 4	calculés lors de l'initialisation -code pour get distance plus simple car on n'a pas besoin de calculer x et y car ils sont stockés déjà -Code de get rho et theta plus long car il faut calculer les valeurs polaires --Code du constructeur plus complexe car on doit vérifier le type et convertit a cartésien au besoin
Design 4	-Plus rapide pour tout les fonctions getters car tous les valeurs sont calculés et conservés à l'initialisation -Méthode rotate et distance plus rapide aussi car les valeurs nécessaires sont stockés au lieu d'être calculés quand demandé	-Utilise le double d'espace en mémoire car stocke les 2 formes de points (cartésien et polaire) -L'initialisation prend plus de temp car il faut calculer le type opposé à chaque fois peu importe le type entré -Code du constructeur plus long car le type opposé doit toujours être calculé
Design 5	-La rapidité des getters dépend de la classe concrète utilisé lors de l'initialisation. get x et y plus rapide pour CP3 et get rho et theta plus rapide pour CP2 -Complexité de code simple seulement classe abstraite	-Puisque les points sont soit PointCP2 ou PointCP3 ils peuvent être plus ou moin efficace dépendant de la méthode qu'on appelle. On peut résoudre ce problème en choisissant la class approprié selon les méthodes que nous voulons exécuter.

E28.
D'après nos résultats, c'est clair qu'il y a peu de différence entre les tests que nous avons exécuté. Ceci est dû au fait que nous avons créer des points aléatoires. Les résultats auraient été meilleurs pour design 5 si on pouvait convertir le point avant de faire le test. Par exemple, pour les tests getX(), getY() et getDistance() il serait mieux si on utilisait la classe concrète PointCP3 pour créer les points puisque les valeurs de x et y sont déjà stockées tandis que si on appelait les méthodes getRho() et getTheta() se serait mieux si on utilisait la classe concrète PointCP2 puisque les valeurs rho et theta sont stockées. Donc, à cause de cela les résultats ont été similaires.

E29.

New Test class created: TestEfficiency.java
See results below

E30.Computation speeds for 500 000 points in ms

Operations	Design 1				Design 5			
	Minimum	Median	Maximum	Moyenne	Minimum	Median	Maximum	Moyenne
Declaration	10	15	30	19.11111	10	17	31	18.66667
getX()	8	8	16	9.777778	8	8	18	10.22222
getY()	8	8	19	10.33333	8	8	21	10.88889
getRho()	0	0	20	4.555556	4	5	22	8.555556
getTheta()	26	26	36	28.22222	28	28	39	30.44444
getDistance()	17	17	38	21.44444	16	16	41	21.33333
rotatePoint()	47	47	59	49.66667	76	77	98	80.33333

a. Comment est-ce que vous avais fait les tests ;
b. Échantillon des sorties lorsque vous avez exécuté les tests ; et
c. Discussion des résultats.

a. Pour tester les implémentations design 2, design 3 et design 4, nous avons décidé de remodeler le fichier PointCPTest.java qui était fourni dans le dossier *design 1(original)* en ajoutant des tests de fonctionnalité pour chacune des méthodes respective, dépendant du design. Nous avons utilisé le design 1 comme modèle pour comparer toutes nos méthodes nouvellement implémentées et pour vérifier l'intégrité des résultats obtenus. Pour tester l'implémentation du design 5, nous avons créé un fichier qui calcule le temps nécessaire pour construire un tableau de 1 000 000 points, nous calculons ensuite également le temps qu'il faut pour exécuter chaque fonction sur chaque élément dudit tableau. Nous effectuons ce processus 9 fois au total, puis nous calculons le temps d'exécution moyen des 9 mesures que nous avions collectées pour chaque méthode. Nous répétons ensuite exactement le même processus avec la conception 1 afin de pouvoir comparer nos résultats et tirer une conclusion logique sur le design le plus efficace.

b. Voir les résultats obtenus ci-dessous dans la section design 5.

c. D'après nos résultats, c'est clair qu'il y a peu de différence entre les tests que nous avons exécuté. Ceci est dû au fait que nous avons créer des points aléatoires. Les résultats auraient été meilleurs pour design 5 si on pouvait convertir le point avant de faire le test. Par exemple, pour les tests getX(), getY() et getDistance() il serait mieux si on utilisait la classe concrète PointCP3 pour créer les points puisque les valeurs de x et y sont déjà stockées tandis que si on appelait les méthodes getRho() et getTheta() se serait mieux si on utilisait la classe concrète PointCP2 puisque les valeurs rho et theta sont stockées. Donc, à cause de cela les résultats ont été similaires.

Design 2

```
You entered: Stored as Cartesian  (5.5,4.5)
--------------------------------
After converting to Polar:
Polar [7.106335201775948,50.71059313749964]

Expected Polar value based off PointCP:
Stored as Polar [7.106335201775948,39.28940686250036]

Conversion from Cartesian to Polar was not successful
--------------------------------
Distance from PointCP3 to (0, 0):
Distance: 7.106335201775948

Distance from PointCP to (0, 0):
Distance: 7.106335201775948

Distance calculation was successful
--------------------------------
After rotating 45 degrees:
Cartesian (0.7071067811865479,7.071067811865475)

Expected Polar value based off PointCP:
Stored as Cartesian  (0.7071067811865479,7.071067811865475)

Rotation was successful
--------------------------------
```

Design 3

```
You entered: Stored as Polar [5.5,4.5]
--------------------------------
After converting to Cartesian:
Cartesian (5.4830453355322035,0.43152502650314717)

Expected Cartesian value based off PointCP:
Stored as Cartesian  (5.4830453355322035,0.43152502650314717)

Conversion from Polar to Cartesian was successful
--------------------------------
Distance from PointCP2 to (0, 0):
Distance: 5.5

Distance from PointCP to (0, 0):
Distance: 5.5

Distance calculation was successful
--------------------------------
After rotating 45 degrees:
Cartesian (3.57196426581601,4.182232810800169)

Expected Cartesian value based off PointCP:
Stored as Cartesian  (3.57196426581601,4.182232810800169)

Rotation was successful
--------------------------------
```

Design 4 (Cartesian)

```
You entered: Stored as Cartesian  (5.5,4.5)
--------------------------------
After converting to Polar:
Polar [7.106335201775948,39.28940686250036]

Expected Polar value based off PointCP:
Stored as Polar [7.106335201775948,39.28940686250036]

Conversion from Cartesian to Polar was successful
--------------------------------
Distance from PointCP3 to (0, 0):
Distance: 7.106335201775948

Distance from PointCP to (0, 0):
Distance: 7.106335201775948

Distance calculation was successful
--------------------------------
After rotating 45 degrees:
Cartesian (0.7071067811865479,7.071067811865475)

Expected Polar value based off PointCP:
Stored as Cartesian  (0.7071067811865479,7.071067811865475)

Rotation was successful
--------------------------------
```

Design 4 (Polar)

```
You entered: Stored as Polar [5.5,4.5]
--------------------------------
After converting to Cartesian:
Cartesian (5.4830453355322035,0.43152502650314717)

Expected Cartesian value based off PointCP:
Stored as Cartesian  (5.4830453355322035,0.43152502650314717)

Conversion from Polar to Cartesian was successful
--------------------------------
Distance from PointCP2 to (0, 0):
Distance: 5.5

Distance from PointCP to (0, 0):
Distance: 5.5

Distance calculation was successful
--------------------------------
After rotating 45 degrees:
Cartesian (3.5719642658160105,4.182232810800169)

Expected Cartesian value based off PointCP:
Stored as Cartesian  (3.57196426581601,4.182232810800169)
```

Design 5

```
C:\Stuff\School\SEG\Project\SEG2505\pointcp\design5>java TestEfficiency
PointCP Declaration time(ms): 26
PointCP Declaration time(ms): 30
PointCP Declaration time(ms): 20
PointCP Declaration time(ms): 10
PointCP Declaration time(ms): 26
PointCP Declaration time(ms): 15
PointCP Declaration time(ms): 15
PointCP Declaration time(ms): 15
PointCP Declaration time(ms): 15

Min time: 10 Median: 15 Max time: 30 Average time: 19.11111111111111
PointCP5 Declaration time(ms): 25
PointCP5 Declaration time(ms): 26
PointCP5 Declaration time(ms): 16
PointCP5 Declaration time(ms): 16
PointCP5 Declaration time(ms): 17
PointCP5 Declaration time(ms): 17
PointCP5 Declaration time(ms): 31
PointCP5 Declaration time(ms): 10
PointCP5 Declaration time(ms): 10

Min time: 10 Median: 17 Max time: 31 Average time: 18.666666666666668
PointCP getX time(ms): 16
PointCP getX time(ms): 16
PointCP getX time(ms): 8
PointCP getX time(ms): 8
PointCP getX time(ms): 8
PointCP getX time(ms): 8
PointCP getX time(ms): 8
PointCP getX time(ms): 8
PointCP getX time(ms): 8

Min time: 8 Median: 8 Max time: 16 Average time: 9.777777777777779
PointCP5 getX time(ms): 18
PointCP5 getX time(ms): 18
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8
PointCP5 getX time(ms): 8

Min time: 8 Median: 8 Max time: 18 Average time: 10.222222222222221
PointCP getY time(ms): 18
PointCP getY time(ms): 19
PointCP getY time(ms): 8
PointCP getY time(ms): 8
PointCP getY time(ms): 8
PointCP getY time(ms): 8
PointCP getY time(ms): 8
PointCP getY time(ms): 8
PointCP getY time(ms): 8

Min time: 8 Median: 8 Max time: 19 Average time: 10.333333333333334
PointCP5 getY time(ms): 20
PointCP5 getY time(ms): 21
PointCP5 getY time(ms): 8
PointCP5 getY time(ms): 8
PointCP5 getY time(ms): 8
PointCP5 getY time(ms): 8
PointCP5 getY time(ms): 8
PointCP5 getY time(ms): 9
PointCP5 getY time(ms): 8

Min time: 8 Median: 8 Max time: 21 Average time: 10.88888888888889
```

```
PointCP getRho time(ms): 17
PointCP getRho time(ms): 20
PointCP getRho time(ms): 4
PointCP getRho time(ms): 0
PointCP getRho time(ms): 0
PointCP getRho time(ms): 0
PointCP getRho time(ms): 0
PointCP getRho time(ms): 0
PointCP getRho time(ms): 0

Min time: 0 Median: 0 Max time: 20 Average time: 4.555555555555555
PointCP5 getRho time(ms): 22
PointCP5 getRho time(ms): 22
PointCP5 getRho time(ms): 5
PointCP5 getRho time(ms): 4
PointCP5 getRho time(ms): 5
PointCP5 getRho time(ms): 4
PointCP5 getRho time(ms): 5
PointCP5 getRho time(ms): 5
PointCP5 getRho time(ms): 5

Min time: 4 Median: 5 Max time: 22 Average time: 8.555555555555555
PointCP getTheta time(ms): 36
PointCP getTheta time(ms): 36
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26
PointCP getTheta time(ms): 26

Min time: 26 Median: 26 Max time: 36 Average time: 28.22222222222222
PointCP5 getTheta time(ms): 39
PointCP5 getTheta time(ms): 39
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28
PointCP5 getTheta time(ms): 28

Min time: 28 Median: 28 Max time: 39 Average time: 30.444444444444443
PointCP getDistance time(ms): 36
PointCP getDistance time(ms): 38
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17
PointCP getDistance time(ms): 17

Min time: 17 Median: 17 Max time: 38 Average time: 21.444444444444443
PointCP5 getDistance time(ms): 41
PointCP5 getDistance time(ms): 39
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16
PointCP5 getDistance time(ms): 16

Min time: 16 Median: 16 Max time: 41 Average time: 21.333333333333332
PointCP rotatePoint time(ms): 59
PointCP rotatePoint time(ms): 59
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47
PointCP rotatePoint time(ms): 47

Min time: 47 Median: 47 Max time: 59 Average time: 49.666666666666664
PointCP5 rotatePoint time(ms): 98
PointCP5 rotatePoint time(ms): 86
PointCP5 rotatePoint time(ms): 77
PointCP5 rotatePoint time(ms): 78
PointCP5 rotatePoint time(ms): 77
PointCP5 rotatePoint time(ms): 77
PointCP5 rotatePoint time(ms): 77
PointCP5 rotatePoint time(ms): 77
PointCP5 rotatePoint time(ms): 76

Min time: 76 Median: 77 Max time: 98 Average time: 80.33333333333333
```

Exercice 2:

1. Le nombre que nous avons trouvé qui donne un temp de 10 secondes pour remplir le ArrayList avec des caractères de «*a*» à «*z*» est 220000000.

2. Pour une taille trouvée qui ne donne pas d'erreur de heep pour ArrayList, LinkedList et pour Array, soit 50000000, nous avons obtenus les runtime suivant. Nous avons dû diminuer le nombre de l'étape précédente car nous étion en train d'obtenir des erreurs de heep.
    a. Array List run time: 4
    b. Linked List run time: 12
    c. Array run time: 1

Ceci dit nous savons déjà que les arrays sont les plus rapides avec un run time de 0, ensuite il y a les ArrayList qui fonctionnent en utilisant un array et finalement il y a les LinkedList qui sont les plus lents. La raison derrière cela est parce que les linked lists utilisent des fonctions plus complexes car ils utilisent un head ou un tail qui doit bouger lorsqu'on ajoute des éléments qui peut prendre du temps . Ensuite il y a les arraylists puisqu'ils ont une classe plus complexe car ils doivent pouvoir grandir (dynamic) et pour cela il doit y avoir des vérification lors des ajout d'éléments pour s'avoir si le array doit grandir ou non. De plus, les arraylists doivent bouger tous les éléments dans le array interne pour les transférer à un array plus grand. Cependant les arrays sont très rapide car utilisent des indices pour accéder aux éléments rapidement.

3. Addition d'éléments

Après avoir changé le nombre d'éléments dans l'étape précédente pour pouvoir obtenir un run time plus raisonnable pour l'addition, voici ce que nous avons obtenus pour l'addition de 100000 éléments:

```
C:\Users\sstella\Desktop\School\SEG2505\SEG2505\pointcp\Exercice2>java Exercice2
Array List run time: 46
Linked List run time: 49
Array run time: 32
Array List addition run time: 32764
LinkedList addition run time: 373839
Array addition run time: 32178
```

Ces résultats font du sense puisque les linked list devraient être les plus lents puisqu'ils ont un time complexity de O(n^2) pour l'addition de tous leurs éléments puisque la fonction get(i) fonctionne en temp O(n). Par après, il y les arraylists et les arrays qui doivent être presque identiques car ils ont les même méthodes pour obtenir des éléments avec des indices.