Devoir surveillé

Durée de l'épreuve : 1 heure 30.

Tout document interdit, de même que l'usage de la calculatrice.

Exercice 1 : Points et chemins

On s'intéresse à la modélisation de points dans le plan et de chemins constitués de points de passage.

Un diagramme de classes du problème est donné en annexe, **à rendre avec votre copie, en y indiquant votre nom**.

Une documentation est également donnée en annexe, et fournit une aide précieuse pour écrire le code demandé.

Q 1 . Annoter le diagramme de classes fourni en annexe, en encadrant ou soulignant et en nommant :

— une classe, une constante, un attribut modifiable, un attribut non modifiable.
— un constructeur, une méthode non statique, une méthode statique.
— un type associé à un paramètre, un type associé à une valeur de retour de méthode.

Q 2 . Que trouve t'on de particulier dans le *main* qui rend `Main` dépendant de `Point` et `Chemin` ?

Q 3 . Donner le code complet de la classe `Main` en considérant que l'application déroule le scénario suivant :

1. on stocke dans une variable `point1` la référence d'un objet `Point` correspondant à la position `(1,1)`.
2. on stocke dans une variable `point2` la référence d'un objet `Point` correspondant à la position `(2,2)`.
3. on stocke dans une variable `point3` la référence d'un objet `Point` correspondant à la position `(3,3)`.
4. on stocke dans une variable `point4` la référence d'un objet `Point` correspondant à la position `(4,4)`.
5. on stocke dans une variable `chemin1` la référence d'un objet `Chemin` correspondant à un chemin vide (existant, mais vide, c'est à dire sans point de passage)
6. on affiche dans la console la représentation texte du chemin référencé par `chemin1`.
7. on ajoute le point référencé par `point1` au chemin référencé par `chemin1`.
8. on affiche dans la console si le chemin référencé par `chemin1` est valide.
9. on ajoute le point référencé par `point1` au chemin référencé par `chemin1`.
10. on affiche dans la console si le chemin référencé par `chemin1` est fermé.
11. on stocke dans une variable `chemin2` la référence d'un objet `Chemin` correspondant à un chemin vide.
12. on ajoute le point référencé par `point1` au chemin référencé par `chemin2`.
13. on ajoute le point référencé par `point2` au chemin référencé par `chemin2`.
14. on ajoute le point référencé par `point3` au chemin référencé par `chemin2`.
15. on ajoute le point référencé par `point4` au chemin référencé par `chemin2`.
16. on affiche dans la console si le chemin référencé par `chemin2` est identique à celui référencé par `chemin1`.
17. on stocke dans une variable `chemin3` la référence du sous-chemin du chemin référencé par `chemin2` commençant à l'indice 2 et de longueur 2.
18. on affiche dans la console si le chemin référencé par `chemin1` est un sous-chemin de celui référencé par `chemin3`.

Q 4 . Donner la sortie standard de l'exécution de l'application précédente (c'est à dire ce qui s'affiche dans la console), en délimitant clairement en fonction des numéros des lignes du scénario précédent.

Q 5 . Donner le code complet de `Point`.

Q 6 . Donner le code de `Chemin`, **en vous limitant aux constantes, attributs, constructeurs et accesseurs en lecture**.

Q 7 . Donner le code de `ajouterUnPoint` de `Chemin`.

Q 8 . Donner le code de `obtenirUnPoint` de `Chemin`.

Q 9 . Donner le code de `exportTexte` de `Chemin`.

Q 10 . Donner le code de `fermer` et `estFerme` de `Chemin`, en utilisant au mieux ce qui est fourni par `Point`.

Q 11 . Donner le code de `passePar` de `Chemin`, en utilisant au mieux ce qui est fourni par `Point`.

Q 12 . Donner le code des deux surcharges de `sousChemin` de `Chemin`.

Q 13 . Donner le code de `estSousChemin` de `Chemin`, en utilisant au mieux ce qui est fourni par `Chemin`.

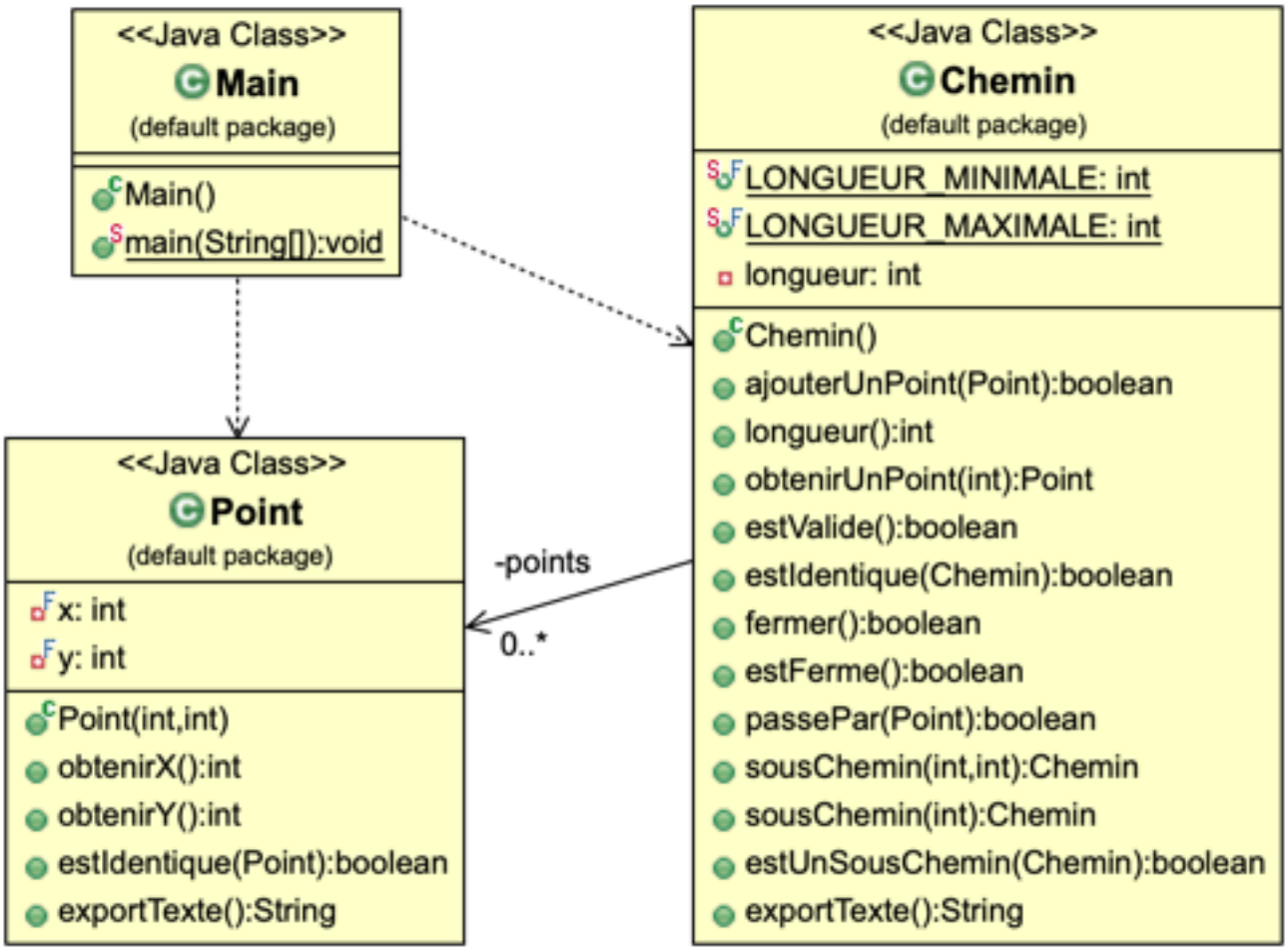
Annexes

Quelques éléments d'aide pour produire le code de `Point`

— le constructeur prend en paramètres les valeurs initiales de `x` et `y`.
— `estIdentique` permet de vérifier l'équivalence du point avec un autre, c'est à dire vérifier que les deux points ont les mêmes valeurs pour `x` et `y`.
— `exportTexte` permet d'obtenir une représentation texte du point sous la forme `(x,y)`.

Quelques éléments d'aide pour produire le code de `Chemin`

— un chemin est formé d'une succession de points de passage.
— la longueur d'un chemin est le nombre de points de passages qu'il contient.
— la longueur minimale d'un chemin est 0, la longueur maximale est 100.
— le constructeur produit un chemin vide (sans aucun point de passage, et donc de longueur minimale). L'espace de stockage, de type tableau, existe et possède le nombre cases permettant d'aller jusqu'à la longueur maximale.
— `ajouterUnPoint` ajoute un point au chemin, en dernière position. Si le chemin est déjà de longueur maximale, le point n'est pas ajouté. La méthode retourne si le point a été ajouté ou non.
— `obtenirUnPoint` permet d'obtenir un point à un indice donné, s'il existe (`null` sinon). Les indices valides sont compris entre 0 inclus et la longueur du chemin exclue.
— `estValide` indique si un chemin est valide, c'est à dire s'il possède au moins 2 points de passage.
— `estIdentique` indique si le chemin est identique à un autre chemin, c'est à dire s'ils ont la même longueur et les mêmes points de passage dans le même ordre. (N.B. : un chemin ne peut pas être identique à `null`).
— `fermer` permet de clôre le chemin, à condition que celui-ci soit valide, en y ajoutant son premier point de passage. La méthode retourne si le chemin a pu être fermé ou non.
— `estFerme` indique si le chemin est fermé, c'est à dire s'il est valide et possède des points identiques aux extrémités.
— `passePar` indique si le chemin contient un point de passage identique à un point donné.
— `sousChemin` permet de construire un nouveau chemin par extraction d'une partie du chemin
  — à partir d'un indice de début et un nombre donné de points à extraire
  — à partir d'un indice de début et jusqu'à la fin du chemin
  si les indices sont invalides, la méthode retourne `null`.
— `estUnSousChemin` indique si le chemin est un sous-chemin d'un autre. On considère que `c1` est un sous-chemin de `c2` s'il existe un sous-chemin de `c2` de même longueur que `c1` identique à `c1`.
— `exportTexte` permet d'obtenir une représentation texte du chemin sous la forme d'un point par ligne, dans l'ordre. (N.B. : `\n` est le caractère spécial qui permet de produire un saut de ligne).


<<Java Class>>
Main
(default package)
Main()
main(String[]):void
<<Java Class>>
Chemin
(default package)
LONGUEUR_MINIMALE: int
LONGUEUR_MAXIMALE: int
longueur: int
Chemin()
ajouterUnPoint(Point):boolean
longueur():int
obtenirUnPoint(int):Point
estValide():boolean
estIdentique(Chemin):boolean
fermer():boolean
estFerme():boolean
passePar(Point):boolean
sousChemin(int,int):Chemin
sousChemin(int):Chemin
estUnSousChemin(Chemin):boolean
exportTexte():String
<<Java Class>>
Point
(default package)
x: int
y: int
Point(int,int)
obtenirX():int
obtenirY():int
estIdentique(Point):boolean
exportTexte():String
-points
0..*