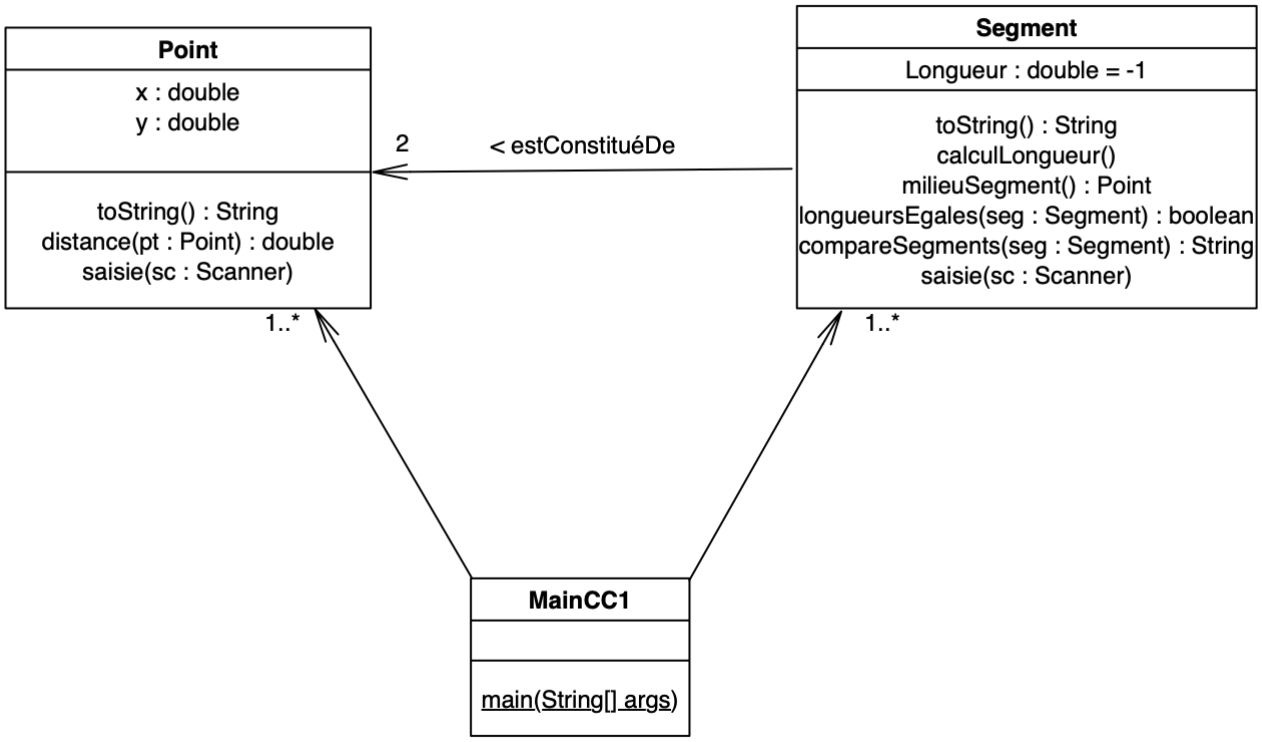
Contrôle 1:
Instructions simples et conditionnelles, classes

Tous documents autorisés. Durée : 1h30
Nous vous conseillons de lire l'intégralité du sujet avant de commencer

1 Structure du programme

Nous mettons le schéma UML qui représente le programme que vous devez mettre en œuvre.



Vous devez donc définir trois classes, toutes appartenant au même paquetage `cc1_2019` :
— Une classe `Point` qui va contenir toutes les informations relatives à un point.
— Une classe `Segment` qui va contenir toutes les informations relatives à un segment.
— Une classe `MainCC1` qui permettra de d'exécuter le programme et de faire des tests (à récupérer sur Moodle).

1.1 Organisation de votre code dans Eclipse

— Dans le projet `HMIN111`, créer un package `cc1_2019`
— Dans ce package, vous devez créer 2 classes : `Point.java, Segment.java`
— Dans ce package, vous devez importer le fichier `MainCC1.java` que vous devez récupérer sur Moodle

1.2 Travail à rendre

Vous devez déposer sur Moodle dans la partie `Dépôt de vos travaux du contrôle CC1` les trois classes java, à savoir `Point.java, Segment.java et MainCC1.java` (pas de zip, les trois classes doivent être déposées séparément)

2 Classe Point et tests dans la classe MainCC1 (*7 points*)

2.1 Structure de la classe `Point`

Dans cette partie, vous devez définir la partie structurelle de la classe, c'est-à-dire ses attributs. Vous choisirez les types des attributs en fonction des informations fournies par le modèle UML. Les attributs devront être privés. Un point est décrit par deux coordonnées :

— x qui définit son abscisse
— y qui définit son ordonnée

2.2 Constructeurs de la classe `Point`

Vous devez écrire les constructeurs suivants :

— Le constructeur vide
— Le constructeur permettant de créer un nouveau point grâce à son abscisse et son ordonnée

2.3 Accesseurs de la classe `Point`

Définir les accesseurs des attributs privés de la classe Point

2.4 Méthode toString() de la classe `Point`

— Écrire une méthode `toString()` qui permet de retourner une chaîne de caractères représentant un point sous la forme suivante : `(x,y)` (x et y étant les valeurs des coordonnées du point)

2.5 Tests dans la classe MainCC1

Dans le fichier `MainCC1.java`, les commentaires vous indiquent où vous devez saisir votre code.

1. Créer deux points p1 et p2 ayant pour coordonnées p1(10,15) et p2(2,3)
2. Écrire le code permettant l'affichage suivant :
   `Les coordonnées du point p1 sont: (10.0,15.0), celles du point p2 sont: (2.0,3.0)`

2.6 Méthode distance() de la classe Point

1. Dans la classe `Point`, créer une méthode qui permet de calculer la distance euclidienne entre deux points.
   — La signature de cette méthode est la suivante : `distance(pt : Point) : double`
   — Rappel : La formule qui vous permet de calculer une distance entre deux points A et B est : **distance** $= \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
   — En java, pour calculer une racine carrée, vous devez utiliser la méthode `Math.sqrt(valeur)`. *Exemple :* Math.sqrt(3) $= \sqrt{3}$
2. Tester cette méthode dans `MainCC1` avec les points p1 et p2 et afficher le résultat sous la forme suivante :
   `La distance entre le point p1 et le point p2 est :` *`valeur de la distance`*

2.7 Méthode saisie() de la classe Point

1. Dans la classe `Point`, définir une méthode permettant d'initialiser un point grâce aux coordonnées x et y saisies par un utilisateur :
   — La signature de cette méthode est la suivante : `saisie(sc : Scanner)`
2. Tester cette méthode dans `MainCC1` de la manière suivante :
   — Créer un nouveau point p3 et utiliser la méthode saisie() pour saisir ses coordonnées
   — Créer un nouveau point p4 et utiliser la méthode saisie() pour saisir ses coordonnées
   — Afficher les points p3 et p4

3 Classe Segment et tests dans Classe MainCC1 (*13 points*)

3.1 Structure de la classe `Segment`

Dans cette partie, vous devez définir la partie structurelle de la classe, c'est-à-dire ses attributs. Vous choisirez les types des attributs en fonction des informations fournies par le modèle UML. Les attributs devront être privés. Un segment est constitué de deux points et possède une longueur qui prend -1 pour valeur par défaut.

3.2 Constructeurs de la classe `Segment`

Vous devez définir trois constructeurs :
— Le constructeur vide
— Le constructeur permettant de créer un segment grâce à ses deux points extrémités.
— Le constructeur permettant de créer un segment dont on connaît toutes les informations

3.3 Accesseurs de la classe `Segment`

Définir les accesseurs des attributs privés de la classe Segment

3.4 Méthode calculLongueur() de la classe `Segment`

— Écrire la méthode `calculLongueur()` qui permet de calculer la longueur d'un segment à partir des coordonnées des deux points du segment. La formule étant la même que le calcul de distance entre deux points, elle utilise donc la méthode `distance()` de la classe `Point`. Le résultat du calcul est affecté à l'attribut `longueur` de la classe `Segment`.

3.5 Méthode toString() de la classe `Segment`

— Écrire une méthode `toString()` qui permet de retourner une chaîne de caractères représentant un segment sous la forme suivante : `le segment constitué des points [`$A(x_A, y_A), B(x_B, y_B)$`], a pour longueur: `*`valeur de la longueur`*.
   — Vous devez réutiliser la méthode `toString()` de la classe `Point` pour afficher les coordonnées des points A et B
   — Vous devez tester la valeur de la longueur. Si celle-ci est égale à -1, alors vous devez recalculer la longueur du segment avant de l'intégrer à la chaîne

3.6 Tests dans la classe MainCC1

1. Créer un nouveau segment s=[p1 p2] (p1 et p2 sont les points que avez définis précédemment)
2. Afficher le segment s

3.7 Méthode milieuSegment() de la classe `Segment`

1. Dans la classe `Segment`, créer une méthode qui permet de retourner le point situé au milieu du segment.
   — La signature de cette méthode est la suivante : `milieuSegment():Point`
   — Rappel : les coordonnées du milieu d'un segment [AB] sont calculées de la manière suivante : $x_M = (x_A + x_B)/2$ et $y_M = (y_A + y_B)/2$
2. Tester cette méthode dans `MainCC1` avec le segment s et afficher le résultat sous la forme suivante :
   `Le milieu du segment s est:` *`coordonnées du point milieu`*

3.8 Méthode saisie() de la classe `Segment`

1. Dans la classe `Segment`, définir une méthode permettant d'initialiser un segment grâce à deux points A et B saisis par un utilisateur :
   — La signature de cette méthode est la suivante : `saisie(sc :Scanner)`
   — Vous devez réutiliser la méthode `saisie()` définie dans la classe `Point` pour permettre à l'utilisateur de saisir les deux points qui constituent le segment
   — Vous devez ensuite recalculer la longueur du segment
2. Tester cette méthode dans `MainCC1` de la manière suivante :
   — Créer un nouveau segment s1
   — Utiliser la méthode `saisie()` pour permettre à l'utilisateur de saisir les points du segment s1
   — Afficher le segment s1

3.9 Méthode longueursEgales() de la classe `Segment`

1. Dans la classe `Segment`, définir une méthode qui compare les longueurs de 2 segments et qui retourne vrai si elles sont égales
   — La signature de cette méthode est la suivante :
   `longueursEgales(seg : Segment) : booleean`
2. Tester cette méthode dans `MainCC1` avec les segments s et s1.

3.10 Méthode compareSegments() de la classe `Segment`

1. Dans la classe `Segment`, définir une méthode qui compare deux segments [AB] et [CD] et qui retourne une chaîne de caractères indiquant si le segment [AB] est plus grand, plus petit ou égal au segment [CD].
   — La signature de cette méthode est la suivante :
   `compareSegments(seg : Segment) : String`
2. Tester cette méthode dans `MainCC1` avec les segments s et s1.

3.11 Modification des coordonnées d'un point du segment

1. Dans `MainCC1`, Modifier le point p1 avec les nouvelles coordonnées (3,5)
2. Recalculer la longueur du segment s dont p1 est une extrémité
3. Comparer de nouveau les segments s et s1