POLYTECH – PeiP2 2015–2016

Contrôle de Algorithmique et Programmation JAVA

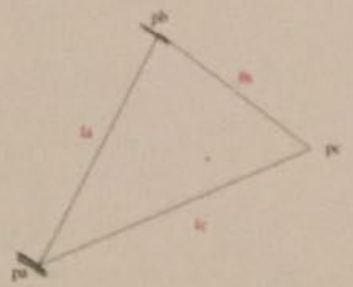
Durée : 1h30 Aucun document autorisé

Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.

► 1. On souhaite représenter les points du plan cartésien. Écrivez la classe `Point` qui déclare :
- deux variables privées `x` et `y`, de type **double**, pour représenter les coordonnées du point courant ;
- deux constructeurs. Le premier a deux **double** en paramètre, le second a un `Point` en paramètre ;
- la méthode `toString` qui renvoie une représentation du point courant sous la forme *"(x,y)"*.
- la méthode booléenne `égal` qui teste l'égalité du `Point` courant **this** et celle du point passé en paramètre ;
- la méthode `distance` qui renvoie la distance entre le `Point` courant **this** et celle du point passé en paramètre. On rappelle que la distance entre deux point $a = (x_a, y_a)$ et $b = (x_b, y_b)$ est égale à $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$.

► 2. On considère que la variable `matP` est un tableau à deux dimensions qui représente une matrice `M`×`N` de `Point`. Donnez la déclaration JAVA de cette variable, avec la création du tableau. On considère que le tableau est initialisé. Écrivez le fragment de code (*pas toute la méthode*) qui recherche dans le tableau `matP` le point `matP[i][j]` dont la distance est minimale avec un point `p` de référence quelconque donné.

► 3. On souhaite représenter des triangles dans le plan cartésien.



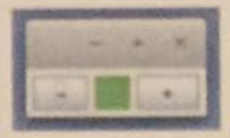
Écrivez la classe `Triangle` qui déclare :
- 3 `Point` `pa`, `pb`, `pc`, pour les 3 sommets ;
- 3 **double** `la`, `lb`, `lc`, pour les 3 longueurs ;
- un constructeur avec 3 paramètres de type `Point` pour initialiser les 6 variables de la classe. Votre constructeur devra vérifier que les 3 points sont distincts, sinon il lèvera une exception `TriangleException` avec comme paramètre le message *"Triangle dégénéré"*. Vous donnerez la déclaration de la classe `TriangleException`.
- la méthode `périmètre` qui renvoie le périmètre du triangle courant **this**.

► 4. On souhaite définir la classe `TriangleEquilatéral` que représente les triangles équilatéraux. Cette classe hérite de la classe `Triangle`. Écrivez cette classe avec un constructeur qui possède 3 `Point` comme paramètres. Le constructeur vérifiera que les 3 points définissent un triangle équilatéral, sinon il lèvera l'exception `TriangleException` avec comme message *"Triangle non équilatéral"*. Rappel : un triangle équilatéral a ses 3 cotés de longueurs égales.

► 5. Le fichier `f` de **double** (pas un fichier de caractères) contient un certain nombre de réels (éventuellement 0). Dans une classe `Test`, écrivez la méthode `sais` qui lit le fichier `f` de réels et initialise le la matrice `matP` de `Point` de l'exercice 2 précédent. Si le nombre de réels contenus dans le fichier est différent de `2`×`M`×`N`, vous signalerez une erreur. Attention, vous ne devez faire qu'un seul parcours du fichier `f`.

Vous pourrez utiliser les classes `DataInputStream` et `FileInputStream` pour ouvrir le fichier de nom indiqué par le paramètre de type `String`. On rappelle que la méthode `readDouble()` d'un objet `DataInputStream` permet de lire un réel **double**.

► 6. On souhaite réaliser en Swing une petite interface graphique qui permet de faire varier un compteur et de visualiser sa valeur sous la forme d'une couleur. Elle possède l'apparence donnée par la figure ci-dessous.



Écrivez la classe `Couleurs` qui possède un `JPanel` encadré par deux `JButton`. À chaque fois qu'on clique sur l'un des boutons, un compteur de type **int** sera incrémenté ou décrémenté et sa nouvelle valeur sera visualisée par une nouvelle couleur de fond du `JPanel`. La valeur initiale du compteur est fixée aléatoirement au départ.

On rappelle que la classe `Color` du paquetage `java.awt` représente des couleurs, et qu'un de ses constructeurs permet la création d'une couleur à partir d'un simple entier de type **int**.

On rappelle aussi que l'interface `ActionListener` de `java.awt.event` possède la méthode abstraite **public void** `actionPerformed(ActionEvent e)` et qu'une sous-classe de `JComponent` hérite de la méthode **void** `addActionListener(ActionListener l)`.

POLYTECH – PeiP2 2015–2016

Contrôle de Algorithmique et Programmation JAVA

Durée : 1h30 Aucun document autorisé

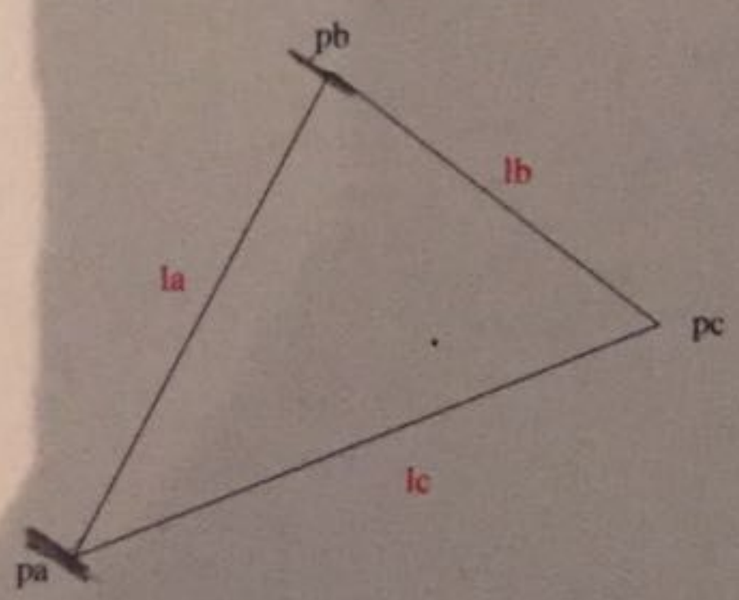
Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.

► 1. On souhaite représenter les points du plan cartésien. Écrivez la classe `Point` qui déclare :

- deux variables privées `x` et `y`, de type **double**, pour représenter les coordonnées du point courant ;
- deux constructeurs. Le premier a deux **double** en paramètre, le second a un `Point` en paramètre ;
- la méthode `toString` qui renvoie une représentation du point courant sous la forme *"(x,y)"*.
- la méthode booléenne `égal` qui teste l'égalité du `Point` courant **this** et celle du point passé en paramètre ;
- la méthode `distance` qui renvoie la distance entre le `Point` courant **this** et celle du point passé en paramètre. On rappelle que la distance entre deux point $a = (x_a, y_a)$ et $b = (x_b, y_b)$ est égale à $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$.

► 2. On considère que la variable `matP` est un tableau à deux dimensions qui représente une matrice `M×N` de `Point`. Donnez la déclaration JAVA de cette variable, avec la création du tableau. On considère que le tableau est initialisé. Écrivez le fragment de code (*pas toute la méthode*) qui recherche dans le tableau `matP` le point `matP[i][j]` dont la distance est minimale avec un point `p` de référence quelconque donné.

► 3. On souhaite représenter des triangles dans le plan cartésien.



Écrivez la classe `Triangle` qui déclare :

- 3 `Point` `pa`, `pb`, `pc`, pour les 3 sommets ;
- 3 **double** `la`, `lb`, `lc`, pour les 3 longueurs ;
- un constructeur avec 3 paramètres de type `Point` pour initialiser les 6 variables de la classe. Votre constructeur devra vérifier que les 3 points sont distincts, sinon il lèvera une exception `TriangleException` avec comme paramètre le message *"Triangle dégénéré"*. Vous donnerez la déclaration de la classe `TriangleException`.
- la méthode `périmètre` qui renvoie le périmètre du triangle courant **this**.

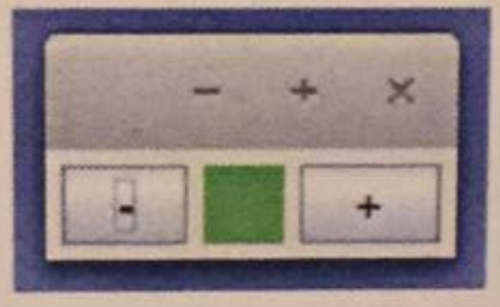
► 4. On souhaite définir la classe `TriangleEquilatéral` que représente les triangles équilatéraux. Cette classe hérite de la classe `Triangle`. Écrivez cette classe avec un constructeur qui possède 3 `Point` comme paramètres. Le constructeur vérifiera que les 3 points définissent un triangle équilatéral.

sinon il lèvera l'exception `TriangleException` avec comme message *"Triangle non équilatéral"*. Rappel : un triangle équilatéral a ses 3 cotés de longueurs égales.

► 5. Le fichier `f` de **double** (pas un fichier de caractères) contient un certain nombre de réels (éventuellement 0). Dans une classe `Test`, écrivez la méthode `main` qui lit le fichier `f` de réels et initialise le la matrice `matP` de `Point` de l'exercice 2 précédent. Si le nombre de réels contenus dans le fichier est différent de $2\times$`M`$\times$`N`, vous signalerez une erreur. Attention, vous ne devez faire qu'un seul parcours du fichier `f`.

Vous pourrez utiliser les classes `DataInputStream` et `FileInputStream` pour ouvrir le fichier de nom indiqué par le paramètre de type `String`. On rappelle que la méthode `readDouble()` d'un objet `DataInputStream` permet de lire un réel **double**.

► 6. On souhaite réaliser en Swing une petite interface graphique qui permet de faire varier un compteur et de visualiser sa valeur sous la forme d'une couleur. Elle possède l'apparence donnée par la figure ci-dessous.



Écrivez la classe `Couleurs` qui possède un `JPanel` encadré par deux `JButton`. À chaque fois qu'on clique sur l'un des boutons, un compteur de type **int** sera incrémenté ou décrémenté et sa nouvelle valeur sera visualisée par une nouvelle couleur de fond du `JPanel`. La valeur initiale du compteur est fixée aléatoirement au départ.

On rappelle que la classe `Color` du paquetage `java.awt` représente des couleurs, et qu'un de ses constructeurs permet la création d'une couleur à partir d'un simple entier de type **int**.

On rappelle aussi que l'interface `ActionListener` de `java.awt.event` possède la méthode abstraite **public void** `actionPerformed(ActionEvent e)` et qu'une sous-classe de `JComponent` hérite de la méthode **void** `addActionListener(ActionListener l)`.