Illustration de l'emploi de this, des modificateurs, surcharge

Considérer le fichier `TestPoint.java` suivant :

```
class Point
{
   private int x = -1 ;
   private int y ;

   public Point(int x, int y)
   {
      x=x;
      y=y;
    }

   public String toString ()
   {
       return "abscisse : " + x + ", ordonnee : " + y ;
   }
}

public class TestPoint
{
   static public void main(String[] args)
   {
      Point p1 = new Point(10,40);
      System.out.println(p1);
   }
}
```

1. Erreur de programmation
   Expliquer pourquoi à l'exécution de ce programme, on obtient :
   `Abscisse : -1, ordonnee : 0`
   et non pas, comme souhaité :
   `Abscisse : 10, ordonnee : 40`
   Modifier le code source pour obtenir le résultat souhaité. Donner les 2 façons de procéder :
   (a) mot clé `this`,
   (b) renommage des arguments.

2. Entrées/Sorties
   Modifier le programme pour que l'abscisse et l'ordonnée du point soient
   (a) transmises sur la ligne de commande lors du lancement de l'exécution du programme.
   Exemple : `java TestPoint 10 40`
   (b) obtenues de façon interactive lors de l'exécution du programme.

3. Surcharge des constructeurs avec un nouveau constructeur sans argument
   On souhaite avoir un constructeur sans argument dans la classe `Point` : `public Point()` et que, par défaut, les attributs `x` et `y` aient comme valeurs respectives 10 et 30.
   Donner les 3 façons de procéder, soit :
   (a) initialisation dans le corps du constructeur ou

(b) appel entre constructeurs : mot clé `this`

(c) initialisation dans la déclaration, corps du constructeur vide.

4. Modificateur `static` pour un attribut
   On désire ajouter dans la classe `Point` deux attributs supplémentaires
   – `totalPoint` de type entier, pour mémoriser le nombre total de points crées jusqu'à présent,
   – `numeroOrdre` de type entier, pour mémoriser le numéro d'ordre de création de ce point.
   Modifiez votre programme pour que lors de l'exécution, vous puissiez lire sur votre terminal :

```
Point numero 1 sur 3 : abscisse : 0, ordonnee : 0
Point numero 2 sur 3 : abscisse : 5, ordonnee : 6
Point numero 3 sur 3 : abscisse : 56, ordonnee : 78
```

5. Modificateur `static` pour une méthode
   On dira que deux points sont égaux s'ils ont la même référence (le même pointeur) ou s'ils ont même abscisse et même ordonnée (mêmes valeurs pour les attributs `x` et `y`).
   Définir dans la classe `Point` le test d'égalité sous les 2 formes : méthode de classe et méthode d'instance c-à-d avec les profils respectifs suivants :
   – `public static boolean egalite (Point p1, Point p2)`
   – `public boolean egalite(Point p)`
   Dans les deux cas, penser au cas particulier ou l'argument serait null.
   Modifier la classe `TestPoint` pour créer un ensemble de points et tester s'ils sont égaux.

6. `this` comme argument de méthode
   Pour utiliser `this` comme argument de méthode, vous pouvez construire l'exemple artificiel suivant : définir dans la classe `Point` une méthode supplémentaire dont le profil est :
   `public boolean egalAsoi()`. Elle renvoie `true` ou `false` selon que ce point est identique à lui-même. La seule réponse possible est évidemment `true`. Cette méthode devra impérativement faire appel à la méthode d'instance `egalite(Point)` qui vient d'être définie.

7. Modificateur `private`
   – pour un attribut :
   Les attributs `x` et `y` de la classe `Point` sont privés.
   (a) Ecrire les accesseurs `getX()` et `getY()`.
   – Les utiliser dans l'affichage du point directement dans le `main`.
   – Dans le test d'égalité de 2 points, expliquer pourquoi il n'est pas indispensable d'utiliser ces accesseurs.
   (b) Ecrire les modificateurs `setX(int)` et `setY(int)` et les appeler dans le `main`.

   – pour une méthode : la méthode est non accessible directement à l'extérieur de la classe où elle est définie. Permet de faciliter l'implémentation des méthodes de l'interface.
   La classe `Point` n'est pas suffisamment complexe pour justifier de méthodes privées.

   – pour un constructeur : Dans la classe `Point`, on désire à présent que les deux attributs `x` et `y` aient des valeurs positives (≥ 0) et si l'appel au constructeur transmet de mauvaises valeurs aucun point ne doit être crée (pointeur `null`). On ne veut pas d'instance avec des valeurs par défaut (définies dans la classe ou attribuées automatiquement à 0 comme c'est le cas pour des attributs déclarés entiers). Comme il n'y a pas d'instruction `return` dans un constructeur, on ne peut renvoyer un pointeur `null`. On définira le constructeur privé et la création d'une nouvelle instance se fait via une méthode `creationPoint`. Ainsi, `creationPoint` teste si les coordonnées vérifient les contraintes données. Si oui, elle appelle le constructeur et renvoie une instance de

`Point`, sinon elle renvoie un pointeur `null`. Son profil sera :
`public static Point creationPoint (int x, int y)`
Ecrire cette méthode et les tests appropriés dans le `main`.
Remarque : il serait plus élégant d'utiliser des exceptions.

8. Modificateur `final` pour un attribut
   La classe `Point` dispose d'un nouvel attribut : un `nom` de type `String`. Ce nom est déclaré `final` : il doit être initialisé au plus tard par le constructeur et ne peut plus être modifié par la suite.
   Ecrire et tester les méthodes de modification de la valeur des différents attributs.