

Programmation orientée objets

Ensimag 2A, examen du mardi 15 décembre 2015

Le chant du SIG...

Durée : 2h - aucune sortie autorisée avant la fin de l'examen

Documents : tous documents papiers autorisés

Le barème est indicatif (total sur 20 points).

Votre code Java devra contenir des **commentaires et des explications pertinentes**.

Les réponses aux questions attendant une réponse textuelle seront données sous forme de commentaires dans le code.

Vous respecterez le *coding style* Java (des points pourront être enlevés dans le cas contraire). Il est également demandé d'adopter autant que possible les autres bonnes pratiques de développement Java telles que : respect du principe encapsulation, usage des exceptions, usage d'annotations...

La documentation Javadoc n'est pas attendue.

Dans la mesure du possible, le code devra compiler et s'exécuter correctement. Si ce n'est pas le cas, en particulier pour les dernières questions, donnez en commentaire des explications et/ou du pseudo-code sur ce que vous cherchez à réaliser.

Vous disposez sur votre machine de deux liens vers :

- la documentation de l'API des classes Java (en html, naviguable)
- une copie locale des ressources Chamillo du cours de POO.

N'oubliez pas :

- de remplir le fichier `QUI_SUIS_JE.txt` avec **votre nom, prénom et groupe** ;
 - d'activer régulièrement le lien du bureau « **Sauvegader l'examen sans quitter** » ;
 - d'activer « **Sauvegader et terminer l'examen** » avant de quitter la salle.
-

Introduction : Système d'Information Géographique (SIG)

Un *Système d'Information Géographique* (SIG), est un système qui permet de gérer des données référencées spatialement, c'est-à-dire attachées à des lieux précis sur la planète, et donc référencées par des coordonnées (GPS par exemple et que, pour simplifiée, on considèrera entières dans le sujet). OpenStreetMap (<http://www.openstreetmap.org>) ou Google Maps (<http://maps.google.com>) sont des exemples de tels systèmes. La figure 1 donne un exemple de ce qu'il est possible de réaliser avec un SIG.

Aujourd'hui, c'est géographie : nous allons dans ce sujet nous intéresser à quelques problèmes techniques très simplifiés liés au développement de tels systèmes, ce qui nous permettra d'en savoir un peu plus sur le dessous des cartes en ligne. Vive donc la géographie qui met la Chine à notre porte et les Russes en fête !

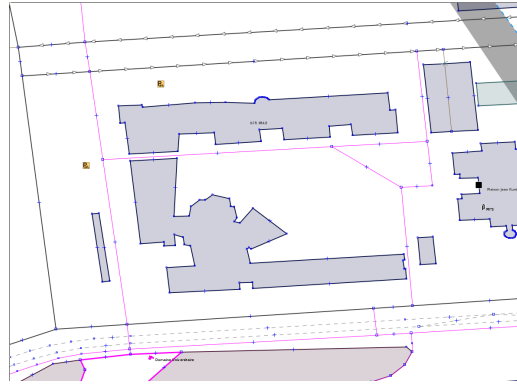


FIGURE 1 – La représentation cartographique d’une certaine école d’ingénieurs grenobloise, avec l’éditeur JOSM (projection WGS84). Données © Contributeurs OpenStreetMap.

Représentation de l’information cartographique vectorielle

Nous allons nous intéresser à des SIG gérant une information vectorielle discrète uniquement. Plus précisément, notre SIG très simplifié devra gérer deux types d’entités géométriques simples, résumés sur le diagramme de la figure 2. La classe SIG apparaissant sur ce diagramme sera traitée dans la section 2.

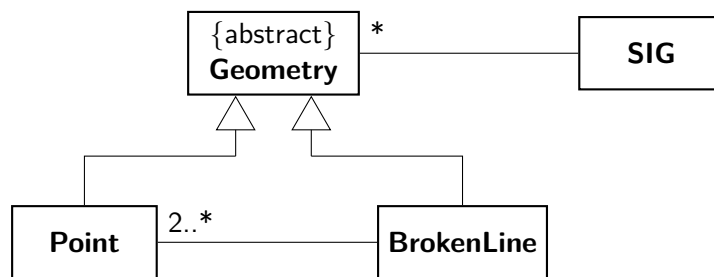


FIGURE 2 – Un fragment très simplifié du standard ISO/TC211 pour la représentation de l’information géographique. Rappel : une flèche creuse représente une relation d’héritage.

Les entités géométriques que nous considérerons sont donc :

- *le point* (classe `Point`), décrit par un couple de coordonnées (de type `int`).
- *la ligne brisée* (classe `BrokenLine`), dans notre cas simplement constituée d’une série de points (2 au minimum). La classe `BrokenLine` est utile, par exemple, pour représenter un bâtiment ou une route.

Enfin, la classe abstraite `Geometry` rend compte du concept abstrait d’entité géométrique. Un SIG est alors défini par les entités géométriques (des `Geometry`) qu’il contient (points et lignes brisées).

Notons pour finir que, dans un SIG, une entité géométrique donnée peut être utilisée par *plusieurs* autres éléments cartographiques. Dans notre cas simplifié, un objet `Point` peut donc être partagé par plusieurs `BrokenLine`.

Vous avez maintenant toutes les cartes en main...

1 Des Geometry pour la géographie (9 points)

Nous allons dans cette partie faire une première mise en place des classes qui nous serviront à construire notre SIG, en partant des classes les plus simples : `Geometry`, `Point`, `BrokenLine`.

Question 1 : 1.5 points Classe `Geometry`

Toute entité géométrique de notre SIG, c'est à dire tout objet de type `Geometry`, possède un nom (de type `String`) qui pourra être consulté mais pas modifié. Enfin, une entité géométrique doit pouvoir générer une représentation textuelle (`String`) d'elle-même et calculer son barycentre.

Ecrivez en Java la classe abstraite `Geometry`, dotée d'attribut(s) et accesseur(s) *pertinent(s)*, d'au moins un constructeur et des méthodes suivantes :

- `String getName();`
- `String toString();`
- `Point getBarycenter();`

Prenez garde, bien sûr, à l'encapsulation et à la visibilité des attributs et méthodes (mots clé `private`, `public`, `protected` ou `ø`) et au *coding style Java*¹.

Question 2 : 1.5 point Classe `Point`

Un `Point` est une entité géométrique qui possède deux coordonnées de type `int` (x et y), qui doivent pouvoir être consultées mais ne sont jamais modifiées. La représentation textuelle d'un point est une chaîne de caractères composée de son nom suivi de ses coordonnées — par exemple "un Point <c'est tout> : (12, 14)". Le barycentre d'un `Point` est lui même.

Proposez en Java le code de la classe `Point`.

Question 3 : 0.5 point Classe de test

Ecrivez une classe de test `Test` dotée d'une unique méthode `public static void main(String args[])`; qui crée un objet de type `Point` puis affiche dans le terminal toutes ses propriétés, ainsi que son barycentre.

Question 4 : 1.5 point Qui suis-je ?

On souhaite ajouter la possibilité de créer des entités géométriques sans les nommer explicitement. Lorsqu'on crée des points (ou plus généralement des `Geometry`) sans indiquer de nom, leur nom doit être alors automatiquement généré sous la forme « `NameAuto0` », « `NameAuto1` », etc.

Modifiez les classes précédentes pour gérer cette nouvelle fonctionnalité, puis testez.

Question 5 : 1.5 point Classe `BrokenLine`

Une `BrokenLine` est définie par une séquence de points constituée d'au moins deux points. Un tel objet est construit en précisant les deux premiers points de la ligne brisée. Puis, on peut ajouter et supprimer des points au moyen des méthodes suivantes :

1. Ces remarques ne seront pas répétées mais s'appliquent bien sûr à toutes les questions suivantes.

- `void add(Point p);` qui ajoute le point `p` en fin de séquence (pour ceux qui se poseraient la question, notez qu’une ligne brisée est susceptible de contenir plusieurs fois le même point).
- `void removeFirst();` qui supprime le premier point de la ligne. *Attention* : cette méthode doit garantir qu’il reste toujours au moins 2 points dans la `BrokenLine` : supprimer le premier point d’une `BrokenLine` qui n’a que deux points est donc une erreur.

On précise par ailleurs que :

- La représentation textuelle d’une `BrokenLine` est une chaîne de caractères comprenant son nom suivi des propriétés des points qui la constituent.
- Dans cette question, on considère simplement que le barycentre d’une `BrokenLine` est le barycentre des `Point` qui la constituent².

Proposez en Java une implantation de la classe `BrokenLine`.

Justifiez en commentaire les choix de la ou des collection(s) Java que vous êtes amenés à utiliser. Précisez la complexité de toutes les méthodes.

Question 6 : 0.5 point Test de la classe `BrokenLine`

Complétez la classe `Test` pour créer cinq nouveaux `Point` de coordonnées (0,0) (0,100) (100,100) (100,0) (0,0), puis une `BrokenLine` passant par tous ces points.

Affichez dans le terminal cette ligne puis son barycentre, qui devrait être en (40,40).

Question 7 : 2 point Correction du calcul du barycentre d’une `BrokenLine`

Le calcul du barycentre tel qu’il a été implanté à la question 5 n’est pas satisfaisant lorsqu’une `BrokenLine` est constituée de plusieurs objets `Point` qui ont exactement les mêmes coordonnées (par exemple lorsque le premier et le dernier points sont superposés pour former une ligne “fermée” aux extrémités). Ainsi, dans le test précédent, on s’attendrait plutôt à ce que le barycentre du carré formé par les cinq points soit aux coordonnées (50,50)...

En fait, dans le calcul du barycentre, chaque point *du plan* ne doit être compté *qu’une seule fois*. Plusieurs objets `Point` qui ont des coordonnées identiques doivent donc être considérés comme “égaux” et n’être comptabilisés qu’une fois. Notez que, pour gérer l’unicité des points considérés dans le calcul, il est possible d’utiliser une collection adéquate...

Mettez en commentaire la méthode de la classe `Point` `getBarycentre()` écrite à la question 5, puis réécrivez cette méthode en respectant cette nouvelle spécification.

Indiquez dans un commentaire le coût de la nouvelle méthode.

Relancez le programme de test et vérifiez que le barycentre est désormais aux coordonnées (50,50).

2 Le langage des SIG (4 points)

On s’intéresse maintenant à la gestion du stockage des formes géométriques dans une classe SIG représentant un Système d’Information Géométrique.

2. A toute fin (in)utile, rappelons que le barycentre de N points de coordonnées (x_i, y_i) est le point de coordonnées $\left(\frac{\sum_{i=1}^N x_i}{N}, \frac{\sum_{i=1}^N y_i}{N} \right)$

Lors de sa création, un SIG est vide : il ne contient aucune entité géométrique. Celles-ci (points, lignes...) doivent être ensuite ajoutées une à une.

Le SIG doit conserver la trace de l'ordre dans lequel les entités géométriques ont été ajoutées. En effet, lors du dessin d'un SIG (dans les questions suivantes) ses entités seront dessinées selon cet ordre.

Notez que les *noms* des entités géométriques contenues dans le SIG doivent être uniques. Par exemple, si on ajoute deux `BrokenLine` alors elles ne doivent pas avoir le même nom³. Enfin, il faut pouvoir récupérer les entités du SIG à partir de leur nom.

Votre classe SIG devra implanter les méthodes publiques suivantes :

1. `void add(Geometry geom)`, qui ajoute une entité géométrique dans le SIG ; cette méthode échoue (erreur, exception...) si le SIG contient déjà une entité de même nom.
2. `Geometry getGeometry(String name)`, qui retourne l'entité géométrique (unique) de nom `name` (ou `null` si le nom est inconnu).
3. `Iterator<String> nameIterator()`, qui retourne un itérateur permettant de parcourir les noms des entités géométriques du SIG.
4. `Iterator<Geometry> geometryIterator()`, qui retourne un itérateur permettant de parcourir les entités géométriques du SIG selon leur ordre d'insertion.
5. `String toString()`, qui affiche les propriétés de tous les objets géométriques du SIG.

Bien entendu, toutes ces méthodes doivent être de coût minimal.

Question 8 : 1 points Attributs de la classe SIG

En commentaire, justifiez vos choix de conteneur(s) Java pour les attributs de la classe SIG.

Question 9 : 2,5 points La classe SIG

Implantez la classe SIG. Précisez les coûts de chaque méthode.

Question 10 : 0,5 points Test de la classe SIG.

Compléter votre programme de test pour :

- créer un objet SIG ;
- y ajouter un des points et la ligne créés lors des tests précédents ;
- afficher le contenu du SIG dans le terminal.

3 L'envol du SIG (4 points)

Nous nous intéressons maintenant au dessin du SIG dans une fenêtre graphique. Pour cela, plusieurs éléments sont fournis :

- Une interface `Drawable` déclarant une unique méthode de dessin :

```
public interface Drawable {
    public void draw(SimpleDrawer simpleDrawer);
}
```

3. Pour l'unicité des noms, on ne testera pas les noms des `Point` constituant une `BrokenLine`. Par exemple, il est possible d'ajouter au SIG une `BrokenLine` qui contiendrait deux points nommés "pt1" sans que cela ne pose problème. Par contre, ces deux points de même nom ne peuvent pas être ajoutés *directement* dans le SIG.

- Une classe `SimpleDrawer`, utilisée par l'interface, fournissant deux méthodes permettant de dessiner des points et des segments :

```
public void drawPoint(int x, int y) { ... }
public void drawLineSegment(int x0, int y0, int x1, int y1) {...}
```

- Une classe `GeometricPainter` dont le constructeur prend un `Drawable` en paramètre. Cette classe crée une fenêtre à l'écran (800x600, centrée sur l'origine) puis dessine ensuite automatiquement le contenu de l'objet `Drawable` en invoquant sa méthode `draw`.

Important : il n'est pas utile de lire/comprendre/modifier le code fourni. Il suffit de mettre les fichiers `Drawable.java`, `SimpleDrawer.java` et `GeometricPainter.java` dans le même répertoire que vos propres fichiers, rien de plus. Toutes les modifications à faire ne concernent que vos classes.

Question 11 : 4 points Dessin.

Modifiez votre classe `SIG` pour qu'elle réalise l'interface `Drawable` fournie et dessine toutes ses entités géométriques. Notez que vous devrez sans doute modifier également les classes de la hiérarchie des `Geometry`.

Modifiez votre programme de test pour dessiner votre `SIG`. Il suffit pour cela de créer un objet `GeometricPainter` :

```
Drawable toDraw = ...; // un objet adéquat
new GeometricPainter(toDraw); // crée une fenêtre et y dessine toDraw
```

4 Des géométries bien composées (3 points)

On souhaite maintenant représenter des entités plus complexes au sein du `SIG`, elles-mêmes composées d'autres entités. Par exemple, un "domaine universitaire" peut être constitué "d'écoles", elles-mêmes constituées de "bâtiments" (des `BrokenLine`) et de "points d'intérêts" (des `Point`).

Question 12 : 3 points *Avant de coder et/ou si vous manquez de temps, expliquez déjà ce que vous voulez faire à l'aide de commentaires et/ou pseudo-code!*

- Proposez une architecture pour représenter de telles entités au sein de la hiérarchie des géométries.
- Quels sont selon vous les points difficiles pour gérer ces entités correctement ? (objet bien construit, affichage, calcul du barycentre, ajout dans le `SIG`, etc.)
- Implantez ce nouveau type de données dans votre `SIG`.

5 S'il te plait, dessine moi un cygne ! (0 points)

Question 13 : 0 points Maintenant que tout est en place, si (vraiment...) vous voulez perdre du temps, vous pouvez utiliser votre `SIG` pour dessiner un Cygne (ou un sapin (ou un renne (ou ...))). C'est bête, c'est fastidieux, mais, qui sait, ça fera peut-être plaisir à vos enseignants, pour les fêtes ?