Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2015-2016**

TD n° 6

Un peu de classe abstraite

1 Interfaces vs Classes abstraites

Exercice 1

1. Peut-on instancier une interface ? une classe abstraite ?
   Correction : *Non pour les deux.*

2. Peut-on y mettre un constructeur ? un constructeur avec un corps ?
   Correction : *Non aux deux pour les interfaces. Oui aux deux pour les classes abstraites, il pourra être appelé avec* `super(...)` *comme d'habitude.*

3. Peut-on écrire le code suivant : `A a = new B();`
   — si `A` est une classe abstraite, dérivée par la classe `B` ?
   — si `A` est une interface, implémentée par une classe `B` ?
   Correction : *Oui pour les deux (si la classe* `B` *n'est pas abstraite).*

4. Une interface/classe abstraite peut-elle contenir des méthodes abstraites ? non-abstraites ? statique et abstraite ?
   Correction : *Java < 8 : une interface ne peut contenir que des méthodes publiques abstraites et non-statiques ; les modificateurs* `public abstract` *n'ont pas besoin d'être déclarés.*
   Java ≥ 8 : on peut aussi définir des méthodes statiques ainsi que des implémentations par défaut des méthodes non-statiques (déclaration précédée du mot-clé `default`*), n'utilisant pas d'attributs non-statiques (car non-existants).*
   Une méthode abstraite a vocation à être implémentée quelque part en aval dans le graphe d'héritage/implémentation, faute de quoi elle ne servirait à rien. Comme les méthodes statiques ne sont pas héritées, les méthodes statiques et abstraites n'existent pas.

5. Une interface/classe abstraite peut-elle contenir des attributs ? avec quels modificateurs ? doivent-il être instanciés ?
   Correction : *Les interfaces ne peuvent avoir que des attributs* `public static final`*, c'est à dire des constantes ; les modificateurs n'ont pas besoin d'être déclarés. Ces attributs doivent être instanciés directement dans le corps de l'interface.*

6. Une interface peut-elle hériter d'une autre interface ? d'une classe abstraite ?
   Correction : *Oui et non.*

7. Une classe abstraite peut-elle hériter d'une autre classe abstraite ? d'une interface ?
   Correction : *Oui et non, elle peut en revanche l'implémenter.*

2 Forme géométrique et factorisation de code

Dans cet exercice, on manipule des formes géométriques que l'on définit par l'interface suivante.

```java
interface FormeGeometrique {
    double perimetre();
    double surface();
}
```

Exercice 2

1. Écrire par exemple les classes `Cercle`, `Triangle`, `Rectangle`, `Carre`.
2. Un polygone est la forme définie par une ligne brisée
   — qui ne se coupe pas elle-même ;
   — qui commence et termine au même point.
   Quel sera le type de `Polygone` ? Quelle sera sa relation avec les classes précédentes ? (On ne demande pas de vérifier si la ligne se coupe.)
   On rappelle que si (x_1, y_1) et (x_2, y_2) sont deux points du plan, alors la distance les séparant est donnée par
   $$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \ .$$
   Correction : *`Polygone` est une classe abstraite qui a un attributs `List<Point> sommets` et dans laquelle on peut implémenter la méthode `perimetre()`. Un `Point` est simplement une classe avec deux attributs `double x,y`.*
3. Un polygone est convexe si, lorsque l'on trace un segment entre deux points quelconques du polygone, alors tous les points du segment appartiennent au polygone.
   Si les sommets d'un polygone convexe, pris dans le sens trigonométrique (au contraire du sens des aiguilles d'une montre), sont $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ sa surface vaut
   $$\frac{1}{2}\left[(x_1\, y_2 + x_2\, y_3 + x_3\, y_4 + \cdots + x_n\, y_1) - (y_1\, x_2 + y_2\, x_3 + y_3\, x_4 + \cdots + y_n\, x_1)\right] \ .$$
   Quel sera le type de `PolygoneConvexe` ? Quelle est sa relation avec les types précédents ? (On ne demande pas de vérifier si un polygone est convexe et on supposera que ses sommets sont donnés dans le sens trigonométrique.)
   Correction : *`PolygoneConvexe` est une classe qui dérive de `Polygone` ; les classes `Triangle` et `Rectangle` en dérive et `Carre` dérive de `Rectangle`.*

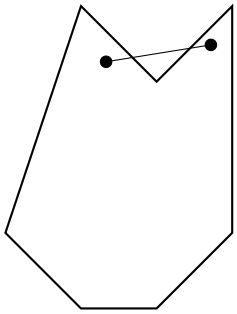
3 Quizz

Le but de cet exercice est de faire la structure des classes pour un programme permettant de poser des quizz notés. Chaque questionnaire sera la donnée d'une liste de questions, chacune valant un certain nombre de points.

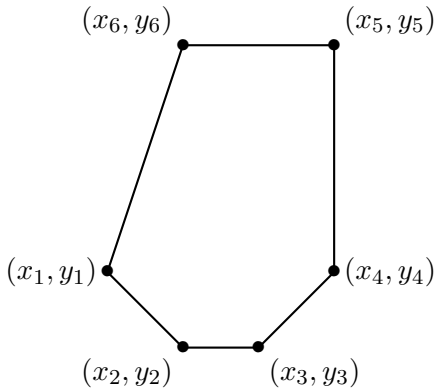
L'exécution du programme dans un terminal donnera quelque chose comme :

```
Bonjour et bienvenu.

>1) Quel est le nom d'un des profs de POO ? (3 reponses) (2 points)
Yunes
>Bonne reponse
>-----------------------------------------------
>2) Combien y a-t-il de lunes autour de Jupiter ? (a 2 pres) (3 points)
```



(a) Un polygone non-convexe



(b) Un polygone convexe

```
60
>Mauvaise reponse
>-----------------------------------------------
>3) Quelle est la densite du plombs? (a 5% pres) (2 points)
10.4
>Bonne reponse
>-----------------------------------------------
>FINI : Vous avez marque 4 points sur 7.
```

On voit dans cet exemple que les questions posées sont de types différents. La première demande une chaîne de caractère, la deuxième demande une valeur entière avec une marge d'erreur entière et la troisième une valeur réelle avec une marge d'erreur en pourcentage. On peut même imaginer d'autres types de questions. Il est donc naturel ici de penser une structure de classes avec de l'abstraction. On vous propose de définir un type `Question` et un type `Quizz`.

Exercice 3 On ne s'intéresse pour l'instant pas au calcul des points.

1. Au vu de l'exemple ci-dessus et du déroulement du jeu (pour chaque question : on pose la question, on récupère une réponse, on vérifie la validité), de quelles méthodes aura-t-on besoin dans `Question` pour pouvoir écrire une méthode **`void`** `poser()` dans une classe `Quizz` sans avoir à se soucier de l'implémentation des questions ? Écrivez la méthode `poser()` pour un déroulement dans un terminal comme dans l'exemple ci dessus.
   Correction : *L'idée générale est d'avoir une interface (ou une classe abstraite)* `Question` *avec une méthode* `String getIntitule()` *et une méthode* **`boolean`** `estCorrecte(Sring reponseFournie)`.

2. Que peut-on faire pour tester `Quizz` ?
   Correction : *Créer une classe* `FausseQuestionTrue` *qui implémente (ou dérive de)* `Question`*, qui renvoie un intitulé* `"FausseQuestionTrue"` *et qui renvoie toujours* **`true`** *lors d'un appel à* `estCorrect`*. On peut également faire une autre classe* `FausseQuestionFalse` *qui renvoie toujours* **`false`** *à la place.*

3. Une question peut avoir une réponse textuelle entière (avec une éventuelle marge d'erreur entière), ou réelle (avec une éventuelle marge d'erreur en pourcentage). Comment faire cela.

Correction : *L'idée ici est d'encourager les élèves à ajouter une sous-classe abstraite* `abstract class QuestionEntiereAbstraite` *qui fournit* `boolean estCorrect(Sring reponseFournie)` *en appelant une nouvelle fonction abstraite* `boolean estCorrect(int i)`*. Même chose pour les questions à réponse réelle.*

Les élèves n'ont a priori pas connaissance des exceptions mais il peut être mentionné que c'est ici qu'on enverrait une exception pour signifier le fait que le candidat n'a pas fournit une réponse "du bon type" (chaîne quelconque au lieu de double, etc).

4. Comment implémenter une question textuelle avec plusieurs bonnes réponses.
   Correction : *Dériver* `Question`

5. Comment implémenter une question à choix multiple.
   Correction : *Dériver* `QuestionEntiereAbstraite`*; la réponse ne sera que l'indice de la bonne réponse.*

Exercice 4 On va maintenant s'occuper du comptage des points.

1. Proposer une manière d'ajouter la gestion des points.
   Correction : *La méthode la plus simple est d'ajouter une seule méthode* `abstract int getPoints()` *dans* `Question`*, éventuellement en ajoutant un attribut correspondant.*

2. Une même question posée à des élèves de L1 ou de L2 ne vaudra pas le même nombre de points. Comment peut-on faire cela sans créer plusieurs fois la même question. (Dans le cas des QCM, il est de plus usuel d'attribuer des points négatifs en cas de mauvaise réponse).
   Correction : *Ajouter une nouvelle classe* `QuestionPonderee` *qui a un attribut* `Question question` *et deux autres* `int pointsSiVrai` *et* `int pointsSiFaux`

Exercice 5 Comment faire pour permettre des questions à laquelle l'utilisateur doit donner plusieurs réponses comme l'illustrent les exemples suivants.

— *Donner le nom de 5 planètes du système solaire.*
— *Donner 10 nombres premiers.*

Correction : *On peut introduire une nouvelle classe abstraite* `QuestionAPlusieursReponses` *(qui sera le type de l'attribut* `question` *dans* `QuestionPonderee`*) qui aura un attribut* `int nbDeQuestions` *et une méthode* `abstract int sontCorrectes(List<String> reponsesFournies)` *qui renvoie le nombre de bonnes réponses.*

La classe abstraite[1] `Question` *étendra maintenant* `QuestionAPlusieursReponses`*, et fixera nbDeQuestion à 1. Les anciennes méthodes abstraites seront inchangées, mais on implémentera la méthode introduite dans le paragraphe précédent en appelant la méthode abstraite* `boolean estCorrect(String reponseFournie)` *avec le premier (et seul) élément de la liste donnée en argument. Ceci permet d'utiliser toutes les anciennes classes telles quelles.*

1. Si c'est une interface, elle sera transformée en classe abstraite.