Programmation en Java : superclasses abstraites et interfaces

Cette séance de TP présente de nouvelles notions utilisées dans la conception orientée objets en Java : les superclasses abstraites et les interfaces. Ces notions permettent d'atteindre un plus haut niveau de polymorphisme. Nous montrerons également comment il est possible de générer une documentation du code développé à l'aide de l'utilitaire javadoc.

Ces notes sont adpatées à partir des protocoles de travaux pratiques écrits par Pierre Manneback et Sébastien Frémal en 2014.

Notions théoriques

Superclasse abstraite

Lors de la conception d'une hiérarchie de classes, il peut être judicieux de créer une classe de laquelle hériteront toutes les autres. Cette superclasse ne sera jamais instanciée et permettra aux autres classes d'hériter de son interface et de son implémentation. On parle dès lors de superclasse abstraite. À l'opposé, une classe concrète permet d'instancier des objets.

À titre d'exemple, le TP1 amenait à la création d'un certain nombre de formes géométriques. Supposons que l'on souhaite ajouter une classe générique de laquelle toutes les formes géométriques hériteraient. Cette superclasse générique ne serait jamais instanciée et on pourrait, dès lors, définir celle-ci comme une classe abstraite. Cela se fait à l'aide du motclé `abstract` utilisé, par exemple, de la manière suivante :

```
public abstract class Shape
```

Bien qu'on ne puisse pas instancier un objet à partir d'une classe abstraite, il est possible de déclarer des références vers une telle classe, ce qui permet de tirer profit du polymorphisme.

Mot-clé `final`

Le mot-clé `final` peut-être utilisé dans différentes situations :

- avec une variable : il spécifie que la variable ne pourra plus être modifiée et doit dès lors être initialisée dès sa déclaration ;
- avec une méthode : il empêche toute redéfinition de la méthode déclarée comme telle ;
- avec une classe : il empêche d'hériter de la classe déclarée comme telle ; celle-ci ne pourra donc

jamais être une superclasse.

Dans la hiérarchie des classes, on retrouvera donc les classes abstraites "en haut" et les classes finales "en bas".

Si le mot clé `final` empêche la modification d'une variable, il n'empêche pas de modifier le contenu référencé. Attention donc si vous utiliser `final` pour une variable dont le type est une classe non-immuable ou un tableau.

Interfaces

Une interface est une classe particulière qui présente uniquement :

- des méthodes publiques et abstraites ;
- des données publiques, statiques et finales.

Une telle interface est définie à l'aide du mot-clé `interface` en lieu et place du mot-clé `class`. Une classe voulant implémenter l'interface le fera à l'aide du motclé `implements`.

```
public interface MyInterface { }

public class MyClass implements MyInterface { }
```

Cette classe peut implémenter plusieurs interfaces simultanément en spécifiant, après le mot-clé `implements`, la liste des interfaces séparées par une virgule. C'est ce procédé qui permet en Java de simuler l'héritage multiple.

La nouvelle classe `MyClass` ainsi créée devra définir toutes les méthodes (avec les mêmes arguments et type de retour) présentées dans l'interface (dans le cas contraire, cette classe devra être abstraite). Ceci revient à garantir que toutes les classes implémentant une même interface mettront à disposition un ensemble commun de méthodes qui permettront d'interagir avec elles. Dès lors, d'autres classes "clientes" pourront dialoguer avec les classes réalisant l'interface en étant sure qu'elles définissent bien les méthodes nécessaires à ce dialogue.

Javadoc

L'utilitaire javadoc permet la création de fichiers HTML destinés à documenter un ensemble de classes. La documentation de l'API Java présentée sur le site (https://docs.oracle.com/javase/8/docs/api/) a été générée à partir de cet utilitaire. Celui-ci se base sur l'insertion de commentaires spéciaux et de balises à l'intérieur du code source pour en créer une description claire et précise. De tels commentaires doivent toujours commencer par `/**` et se terminer par `*/` :

```
/** Exemple de commentaires documentés */
```

Des balises permettent d'ajouter des méta-informations qui seront insérées de manière spécifique dans la documentation (voir annexes).

S'il est possible de générer un mini-site HTML à l'aide de la commande `javadoc`, la plupart des IDE utilisent ces informations pour faciliter le travail du développeur. IntelliJ, par exemple, affiche la Javadoc lorsque l'on tape le raccourci `Ctrl+q`.

Exercices

Exercice 1

Cet exercice a pour but d'illustrer l'utilisation des classes abstraites.

Le salaire d'un ensemble d'employés doit être calculé. Ces employés sont répartis en 3 catégories et donc en 3 classes distinctes :

- FixedSalaryEmployee : recevant un salaire fixe :

```
(salaire mensuel = salaire)
```

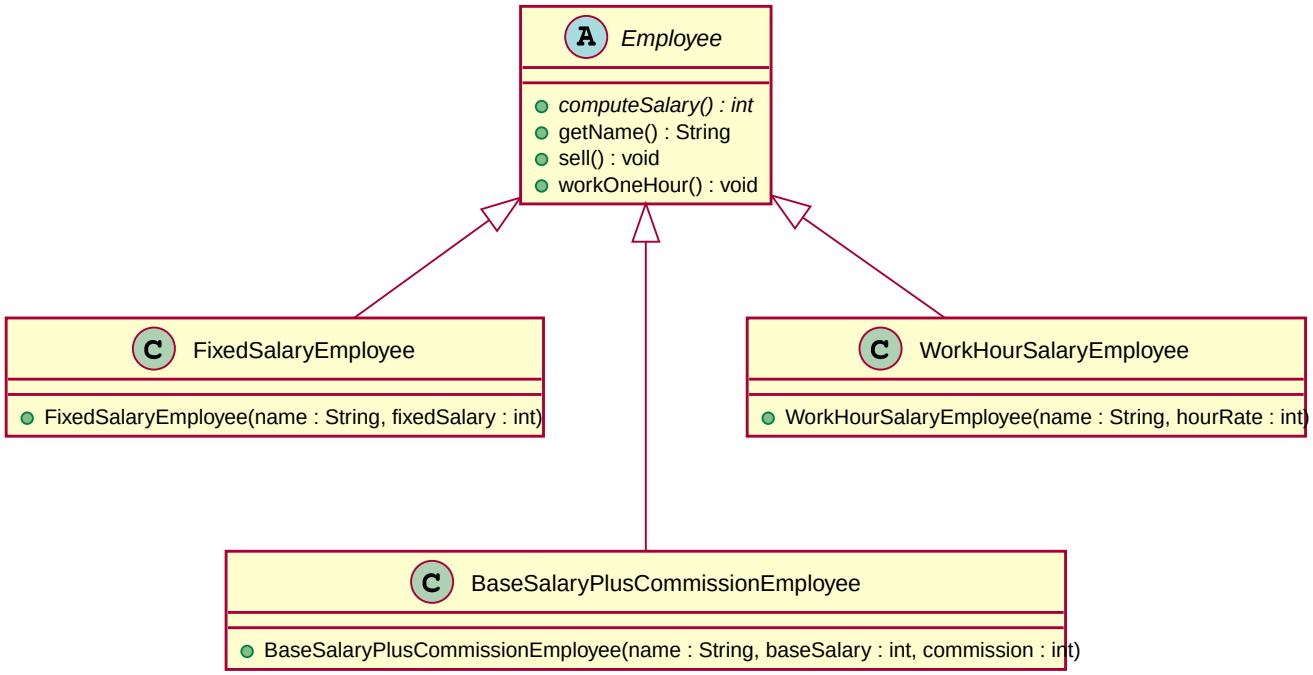
- BaseSalaryPlusCommissionEmployee : recevant un traitement de base plus un pourcentage lié à ses ventes :

```
(salaire mensuel = salaire + ventes * commission)
```

- WorkHourSalaryEmployee : recevant une paie selon le nombre d'heures prestées :

```
(salaire mensuel = taux horaire * heures prestées)
```

Le nom et le salaire de chaque employé doivent pouvoir être obtenus. On suppose connaître les quantités vendues et le nombre d'heures prestées au moment de la création de l'employé (ces données représentées par un type entier, doivent toutefois pouvoir être modifiées ultérieurement). Une superclasse abstraite Employe sera créée en vue de traiter de manière polymorphe les différents employés de l'entreprise.



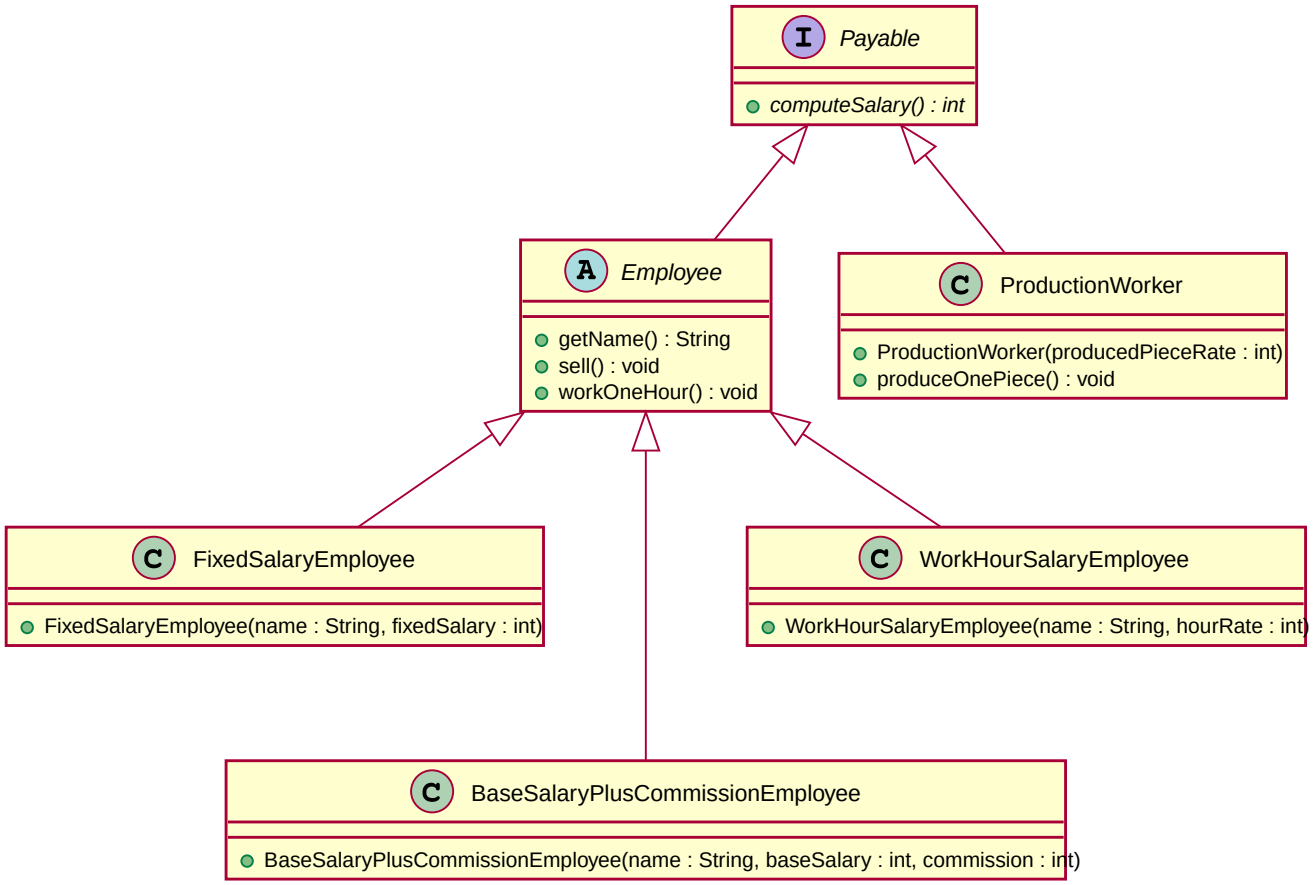
Exercice 2

Cet exercice mettra en évidence l'intérêt de l'utilisation des interfaces. Adapter l'exercice 1 pour qu'il soit capable de gérer de manière polymorphe une classe supplémentaire `ProductionWorker` à celles déjà établies. Cela sera possible en implémentant une interface particulière nommée `Payable`.

Le calcul du salaire pour la nouvelle classe se fera simplement de la manière suivante :

```
salaire mensuel = rétribution par pièce * production
```

La classe `ProductionWorker` ne peut pas hériter d'une quelconque classe de type `Employee` déjà définie.



Exercice 3

L'exercice sur les formes géométriques de la séance précédente sera repris et complété en vue d'afficher les formes géométriques à l'écran. Par souci de facilité, nous ne garderons que la partie concernant les classes PolygoneConvexe, Rectangle et Carre (les classes Ellipse et Cercle peuvent donc être retirées du code).

Une archive JAR est fournie et devra être importée dans le projet; elle contient principalement trois éléments :

- une classe `Panel` gérant l'interface graphique : elle devra être instanciée par la classe principale de la manière suivante :

  ```
  Drawable[] drawables = ...
  Panel pannel = new Panel(drawables);
  ```

  Le paramètre drawables est un tableau de type `Drawable[]` et contient la liste des formes géométriques à dessiner.

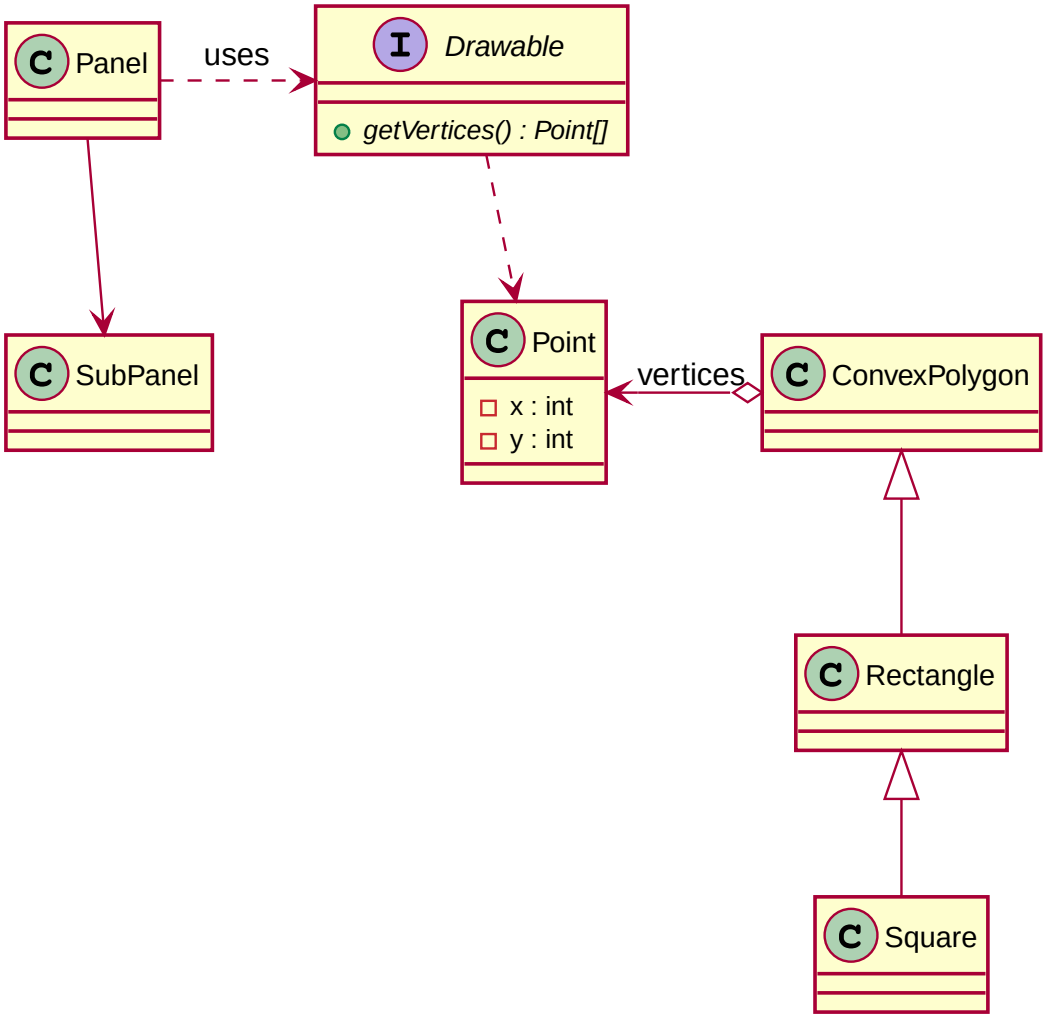
- une interface `Drawable` :

  ```
  public interface Drawable {
      Point[] getVertices();
  }
  ```

L'interface permettra d'établir un langage commun entre la classe `Panel` et les formes géométriques à dessiner.

- une classe `Point` telle qu'attendue par la méthode getListeSommets de l'interface `Drawable`.

- Les classes `ConvexPolygon`, `Rectangle` et `Square` réalisées au TP1 devront être adaptées pour utiliser la classe `Point` fournie dans l'archive.
- Les coordonnées des sommets seront choisis entre 0 et 300.



Exercice 4

Adapter l'exercice 1 pour qu'il fournisse une documentation complète sur les classes développées. Cette adaptation sera réalisée à l'aide de commentaires Javadoc.

Annexe : tags javadoc

Balise	Description	Type
@author	Ajoute une note sur l'auteur (nécessite l'utilisation de author lors de l'appel à javadoc)	Author:
@param	Donne une description d'un paramètre de méthode	Parameters:
@return	Donne une information sur le type de retour	Returns:
@see	Ajoute un renvoi vers une autre classe ou méthode apparentée	See also:
@throws ou @exception	Spécifie les exceptions lancées par la méthode	Throws:
@deprecated	Ajoute une note d'obsolescence	Deprecated
@link	Ajoute un lien hypertexte vers un autre document HTML	
@since	Ajoute une note de validité	Since
@version	Ajoute une note de version	Version

Références

- Deitel, H. M. & Deitel, P. J. (2002), *JAVA Comment programmer – Quatrième édition*. Les éditions Reynald Goulet INC.
- Manneback, P. & Frémal, S. (2014-2015) *Travaux pratiques de Méthodologie et Langage de Programmation*. UMons.
- Manneback, P. (2005-2006) *Méthodologie et Langages de Programmation*. UMons.
- *Java Platform Standard Edition 8 Documentation*. Récupéré de https://docs.oracle.com/javase/8/docs/