

Feuille de travaux dirigés n° 3

Chapitre 3 - Spécialisation



Dans cette série d'exercices, nous nous focalisons sur l'héritage de classes, les relations de spécialisation entre classes. Nous faisons toujours la part belle à la programmation Java avec Eclipse. Nous terminons, comme dans les chapitres précédents, par des activités optionnelles autour de la programmation Smalltalk avec VisualWorks. Après la modélisation conceptuelle vue dans le chapitre 1, nous proposons en préambule de nous arrêter sur la "bonne" modélisation de l'héritage.

Voici une suggestion d'exercices et d'activités à faire pour ce chapitre 2 :

1. Modélisation de l'héritage : section 1
2. De la modélisation à la programmation à objets : section 2.
3. Programmation Java avec l'environnement de développement Eclipse : section 3
4. Programmation Smalltalk avec l'environnement de développement VisualWorks : section 4

1 Modélisation de l'héritage

Exercice 3.1 (Héritage or not ?)

Expliciter le type des relations entre :

1. un point et un segment,
2. un carré et un rectangle,
3. un timbre, une adresse et une enveloppe,
4. un château et ses cartes,
5. une classe, des élèves, des professeurs et des cours,
6. un livre, ses pages et son CD d'accompagnement facultatif,
7. un article, sa référence dans le catalogue du magasin et sa référence dans le catalogue des fournisseurs,
8. des poupées russes,
9. une voiture, son modèle et ses options,

2 De la modélisation à la programmation à Objets

Dans cette section, nous nous intéressons à un problème spécifique de conception détaillée de la relation de spécialisation : la représentation des spécialisations UML en relation d'héritage des langages de programmation.

Les problèmes associés sont

1. traitement de l'héritage multiple dans les langages qui n'autorisent que l'héritage simple.
 - En C++ ou Eiffel, l'héritage multiple est autorisé. On gère les conflits par des redéfinitions ou des renommages.
 - En héritage simple, on choisit un axe principal de spécialisation puis

- en Java, on définit des interfaces pour l’héritage secondaires et on les implante : on cumule les avantages du sous-typage sans l’inconvénient de gestion de l’héritage multiple.
 - En Smalltalk, on doit faire du copier coller !
2. Redéfinition éventuelle / renommage de propriétés.

Exercice 3.2 (Association UML → Attribut)

Reprendre les diagrammes de classes des exercices de la section 1 du TD n° 1.

Concevoir les modèles en C++, Eiffel Java et en Smalltalk.

3 Programmation Java avec Eclipse

Nous conseillons d’avoir fait les exercices concernant `Eclipse` dans le TD no 1.

Exercice 3.3 (Java : Programmation)

blabla

4 Programmation Smalltalk avec VisualWorks

Il est impératif d’avoir fait les exercices concernant `VisualWorks` dans les TDs précédents.

Exercice 3.4 (St : Pixel)

Définir des pixels et des points en trois dimensions.

Feuille de travaux dirigés n° 3 Chapitre 3 - Spécialisation

— CORRECTION —



Corrigé exercice 3.1 (Héritage or not ?)

1. Nous sommes passé sur ce problème dans l'exercice de géométrie du TD. Un segment est composé d'au moins deux points, ses extrémités. Deux points suffisent à identifier le segment (cas A). Une relation de spécialisation est possible dans les deux sens entre points et segments (cas B et C).

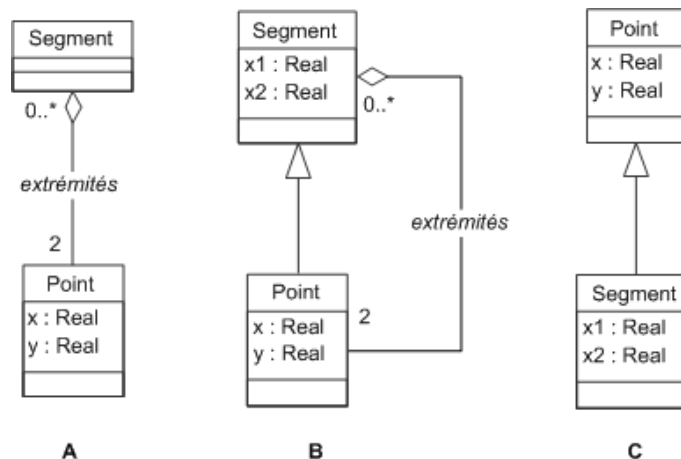


Figure 1 : UML - Spécialisation Point-Segment

Dans le cas B, un point est un segment aux extrémités confondues. En OCL on écrit :

```
context Point
  inv segmentRéduit:
    self.extrémités->forall(p1, p2 : Point | p1.x = p2.x and p1.y = p2.y)
```

Dans le cas C, un segment est un point auquel on ajoute des coordonnées (ou un autre point via une agrégation). Aucune spécialisation ne nous convient car les comportements (les opérations) sont clairement différents entre points et segments. La solution correcte est la solution A (avec ou non séparation des extrémités en deux associations pour faciliter leur navigation).

2. Pour un carré et un rectangle, le raisonnement est identique au cas du segment.

Corrigé exercice 3.2 (Association UML → Attribut)

Non corrigé ici, mais vu à travers d'autres exercices.

Corrigé exercice 3.3 (Java : Programmation)

Non corrigé

Corrigé exercice 3.4 (St : Pixel)

[Pixel]