Héritage comme spécialisation
Corrigé

Exercice 1 : Comprendre la classe PointNommé
À partir du texte source de la classe PointNommé, expliquer les différents éléments de cette classe et dessiner le diagramme de classes UML avec cette classe PointNommé.
Solution : Les points importants sont :

1. la relation d'héritage (`extends`) qui indique que la classe PointNommé est un sous-type de la classe Point et qui, accessoirement, permet de récupérer dans la classe PointNommé, ce qui est déjà défini dans la classe Point.
2. le constructeur qui appelle le constructeur de la superclasse.
3. La redéfinition de la méthode afficher (et dessiner). Notons que `@Override` permet d'expliciter l'intention du programmeur (redéfinir une méthode) et le compilateur pourra ainsi vérifier qu'il s'agit effectivement d'une redéfinition.
4. L'utilisation de `super` pour appeler une méthode de la superclasse sans que la liaison dynamique ne s'applique.
5. Ce qui est spécifique à PointNommé (attributs et méthodes).

Le diagramme de classe est présent dans le cours.

Exercice 2 : Comprendre la relation d'héritage entre Point et PointNommé
Intéressons nous au programme de test TestPolymorphisme.
2.1. Lire le texte du programme TestPolymorphisme et répondre aux questions posées.
Solution : Voir la classe TestPolymorphismeCorrige.
2.2. Exécuter le programme TestPolymorphisme pour vérifier les réponses.
2.3. Ajouter le mot-clé **`final`** devant la déclaration de la méthode `afficher` dans la classe Point. Compiler Point, puis PointNommé et expliquer.
Solution : Le mot-clé `final` placé devant une méthode interdit à une sous-classe de la redéfinir. C'est ce que signale le compilateur quand on essaie de compiler la classe `PointNomme`.
2.4. Ajouter **`final`** devant la déclaration de la classe Point. Compiler Point et PointNommé.
Solution : Placé devant la classe, `final` interdit à une sous-classe d'en hériter. C'est toujours le sens de rendre non modifiable. Hériter d'une classe, c'est pouvoir redéfinir certaines de ses méthodes et être utilisée en lieu et place de cette classe (sous-typage et liaison dynamique).
2.5. Expliquer pourquoi on peut calculer la distance entre un point et un point nommé ?
Solution :
Le polymorphisme (ou le principe de substitution). PointNomme est un sous-type de Point (PointNomme hérite de Point). Là où on attend un Point, on peut mettre toute instance d'une sous-classe de Point. Tout ce qu'on peut demander à Point, PointNommé peut également le faire.

Et entre un point nommé et un point ?

Solution : Car PointNomme hérite de Point et possède donc la méthode distance.

2.6. Pourquoi peut-on attacher à q aussi bien un point qu'un point nommé ?

Solution : Parce que q est déclaré du type Point, on peut donc l'initialiser avec un Point où un sous-type de Point, par exemple PointNomme (puisque PointNomme hérite de Point).

Que se passe-t-il lorsque q est affiché ?

Solution : La liaison statique conduit à sélectionner la signature afficher() de la classe Point. L'objet qui est associé à q est en fait un PointNomme, la méthode qui sera exécutée sera donc la méthode afficher() de la classe PointNomme, qui affiche le nom et les coordonnées.

Quel est ce concept ?

Solution : La liaison dynamique (mais ne pas oublier qu'au préalable il y a la liaison statique).

Exercice 3 : Segments et points nommés

Intéressons nous aux segments et points nommés. Remarquons que la classe Segment définie ne fait référence dans son texte qu'à la classe Point et jamais à PointNommé.

3.1. Peut-on créer un segment à partir d'un point et d'un point nommé ?

Solution : Oui, car un PointNomme est un sous-type de Point (héritage).

Comment serait affiché un tel segment ? Justifier les réponses.

Solution : La méthode afficher() de Segment appelle la méthode afficher() sur ses extrémités de type de Point. La **liaison dynamique** fait que c'est la méthode de la classe de l'objet associé aux extrémités du segment qui sera exécutée. Ainsi, s'il s'agit d'un pont nommé, son nom apparaîtra.

3.2. Créer le programme ExempleSchema2 pour représenter le schéma de la figure 1. On partira du programme ExempleSchema1. Indiquer les modifications à apporter.

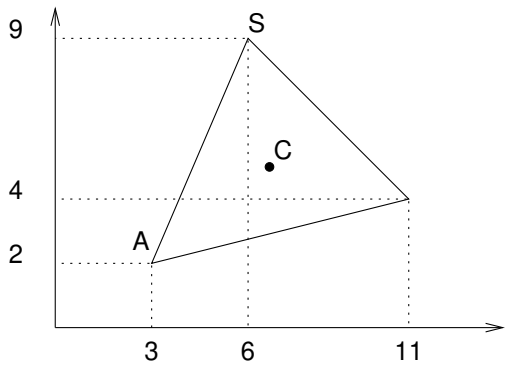

FIGURE 1 – Un exemple de schéma mathématique avec des points nommés

Solution : Il suffit de remplacer quelques new Point par new PointNomme avec le paramètre supplémentaire correspondant au nom.

L'intérêt de la liaison dynamique est que la classe Segment qui a été écrite avant que les point nommés existent n'a pas à être modifiée pour prendre en compte les points nommés. Qaund on affichera un segments, on verra le nom des points nommés si ils sont extrémités du segment.

Notons que l'on n'a pas modifié la classe Point. C'est ce qui permet l'extensibilité : sous-typage et liaison dynamique (polymorphisme).
Remarque : Quand on déclare les points nommés dans le programme principal, il n'est pas nécessaire de changer le type de la poignée dans le cas présent mais ceci signifie que l'on ne pourra pas accéder directement au nom des points nommés créés.