Spécification et implantation

Même s'il est habituellement recommandé de lire tout l'énoncé avant d'y répondre, nous vous demandons exceptionnellement ici de de pas anticiper la lecture des questions.

Exercice 1 : Préambule : qu'est ce qu'un point géométrique ?

1.1 Recenser tout ce qui est défini sur le point géométrique.

1.2 Indiquer les éléments recensés en 1.1 qui peuvent être représentés sous la forme d'attribut.

1.3 Quel formalisme a été utilisé pour répondre à la question 1.1 ?

Exercice 2 : Spécifier en programmation objet : exemple du point géométrique

Quand on veut spécifier un type en programmation objet, on doit prendre le point de vue des utilisateurs de ce type. On ne s'intéresse donc qu'aux méthodes auxquelles ils auront accès. Ces méthodes sont classées en deux catégories :

— les *requêtes* : informations qui peuvent être demandées à un objet instance de la classe (méthodes sans effet de bord, i.e. fonctions) ;
— les *commandes* : services qui peuvent être réalisés par un objet (équivalent de procédures).

Spécifier le point géométrique décrit ci-après, c'est-à-dire identifier les requêtes et commandes, dessiner le diagramme d'analyse [1] UML correspondant, ajouter les informations de type, préciser la sémantique (objectif, conditions d'utilisation et effets) des requêtes et commandes.

On s'intéresse aux points du plan en considérant aussi bien leurs coordonnées cartésiennes (abscisse et ordonnée) que polaires (module et argument). On souhaite pouvoir obtenir la distance qui sépare deux points, translater un point en précisant un déplacement suivant l'axe des X (abscisses) et un déplacement suivant l'axe des Y (ordonnées), afficher les caractéristiques du point qui se limitent à l'abscisse et l'ordonnée sous la forme du couple (abscisse,ordonnée). Enfin, on veut pouvoir modifier l'abscisse, l'ordonnée, le module ou l'argument d'un point.

Exercice 3 : Définir les constructeurs

3.1 Qu'est-ce qu'un constructeur ? Une classe peut-elle avoir plusieurs constructeurs ? Quel est l'intérêt d'un constructeur ?

3.2 Spécifier un constructeur qui initialise un point à partir de ses coordonnées cartésiennes.

3.3 Peut-on spécifier un second constructeur pour initialiser un point à partir de ses coordonnées polaires ? Pourquoi ?

3.4 Après avoir défini les constructeurs précédents, a-t-on accès au constructeur par défaut (celui qui ne prend pas de paramètres) ? A-t-on intérêt à le définir ? Pourquoi ?

1. On appelle diagramme d'analyse UML le diagramme de classes faisant apparaître les rubriques requêtes/commandes et non attributs/méthodes.

Exercice 4 : Implanter en programmation objet

Maintenant que la spécification des points est terminée, on peut s'intéresser aux implantations possibles. Il s'agit de définir pour une classe :

— son état représenté par les *attributs* ;
— son comportement décrit par ses *méthodes*.

Les commandes et les requêtes deviennent des méthodes. Une requête peut correspondre à une information stockée (si un attribut lui correspond) ou calculée à partir de l'état de l'objet.

4.1 *Choix d'implantation.*

4.1.1 Proposer plusieurs implantations de la spécification d'un point géométrique (en utilisant un diagramme de classe attributs/méthodes) et indiquer dans quelles conditions les utiliser.

4.1.2 Écrire la méthode qui change l'abscisse du point pour chaque implantation envisagée.

4.2 On décide d'implanter la classe Point en choisissant l'abscisse et l'ordonnée comme seuls attributs. Écrire la classe correspondante. On se limitera aux attributs, au constructeur et aux méthodes qui permettent de translater un point, l'afficher, changer son abscisse, obtenir son module.

Exercice 5 : Utiliser la classe Point

Dans la suite nous considérerons deux programmes qui utilisent la classe Point. Il s'agit ici de programmes simples mais la classe ainsi créée pourrait tout aussi bien s'intégrer dans une architecture de plus grande échelle utilisant intensivement les points.

5.1 *ExemplePoint1.* Écrire un programme qui crée un point de coordonnées cartésiennes (1,0), puis affiche son module et son abscisse, le translate de (-1, 1) et, enfin, l'affiche. Indiquer ce que doit afficher son exécution et dessiner l'évolution de la mémoire.

5.2 *ExemplePoint2.* Écrire un programme qui crée un point de coordonnées cartésiennes (1,0), met son abscisse à 10 et affiche son module. Indiquer ce que doit afficher son exécution.

Exercice 6 : Comprendre pourquoi les attributs doivent être privés

Regardons les conséquences de déclarer les attributs publics dans la classe Point.

6.1 *Utiliser les attributs.* On suppose que les attributs de la classe Point ont été déclarés publics. L'utilisateur de la classe Point peut donc les utiliser directement. Indiquer les modifications qu'il pourrait faire aux classes ExemplePoint1 et ExemplePoint2 en utilisant les attributs partout où c'est possible.

6.2 *Comprendre le principe de l'accès uniforme.* Le programmeur de la classe Point décide de changer son implantation des points. Il choisit comme attributs les coordonnées polaires, c'est-à-dire le module et l'argument. On dispose donc d'une deuxième version de la classe Point.

Est-ce que la classe ExemplePoint1 compile et s'exécute correctement ?

6.3 *Comprendre le principe de protection en écriture des attributs.* Le programmeur décide de changer l'implantation de sa classe. Cette fois, il choisit pour attributs les quatre coordonnées. On dispose donc d'une troisième version de la classe Point.

Est-ce que la classe ExemplePoint2 compile et s'exécute correctement ?

Exercice 7 : Retour sur la création des points

On veut pouvoir créer un point en fournissant ses coordonnées cartésiennes ou ses coordonnées polaires. Comment faire ?

TOB
TD1.1

Spécification et implantation

Marc Pantel

Exercice 1 : Préambule : qu'est ce qu'un point géométrique

1.1/ Point Géométrique :
1.2/ coordonnées (dimension 2)
- ↳ abscisse } cartésienne
- ↳ ordonnée }

origine
distance à l'origine → module, argument
nom (point nommé)
masse (point matériel)
placer des points → creation par rapport à l'origine
transfater
...
transformation géométrique.

} Attributs / structure des données
} Constructeur.
} Services / Méthodes.

Pour la TOB, il faudrait commencer par les services que le programme doit rendre

1.3/ Langage naturel pas structuré
Il peut être interessant d'utiliser un formalisme pour structurer les données.
↳ Passage à l'UML (Unified Modeling Language) : Diagramme de classe

Point Géométrique
<<attributs>>
abscisse module nom ordonnée argument masse origine ??
<<methodes>>
transfater calculer Distance
<<constructeurs>>
placer.

Diagramme de classe

Exercice 2 : Spécification en programmation objet

Spec : ens. des requêtes et commandes disponibles pour la classe

Point Géométrique
<<requêtes>>
lireAbscisse getAbscisse lireOrdonnée getOrdonnée lireXXX getXXX ... lireDistance(autre : PointGéométrique)
<<commandes>>
transfater(dx, dy) ecrireAbscisse setAbscisse ecrireOrdonnee setOrdonnee : :

Tout ce qui renvoie une valeur est une requête

← Diagramme de classes / d'analyse

dx : translation sur x de dx / valeur ajoutée à l'abscisse
dy : " y de dy / valeur " l'ordonnée

Exercice 3 : Définir les constructeurs

3.1/ Un constructeur doit allouer la mémoire et initialiser les attributs pour rendre l'état de l'objet cohérent. Dans le cas du point géométrique, il pourrait être utile d'avoir plusieurs constructeurs [placer Cartesien (_abs, _ord) ; placer Polaire (mod, arg)]

3.2/ placer_Cartesien (_abscisse, _ordonnée)
    set Abscisse (_abscisse)
    set Ordonnée (_ordonnée)

3.3/ placer_Polaire (_module, _argument)
    assert True (_module >= 0)
    set Module (_module).
    set Argument (_argument.normalise()).

3.4/ placer_Origine ().
    set Abscisse (0).
    set Ordonnée (0).

Interêt : oui pour placer l'origine
Défaut : utilisateur pourra créer des points sans coord.

Exercice 4 : Implanter en programmation objet.

4.1.1/

Point Cartesien
<<attributs>>
abscisse ordonnée
<<méthodes>>
lire Module lire Argument ecrire Module ecrire Argument
<<constructeurs>>
placer Cartesien (...)

Point Polaire
<<attributs>>
module argument
<<méthodes>>
lire Abscisse lire Ordonnée ecrire Abscisse ecrire Ordonnée
<<constructeur>>
placer Polaire (...)

le choix de l'implantation dépend du traitement à faire

Point Mixte
<<attributs>>
abscisse module ordonnée argument
<<méthodes>>
ecrire Abscisse ecrire Module ecrire Ordonnée ecrire Argument
<<constructeur>>
placer Mixte (...)

} maintient de la cohérence

TOB
TD1.2

4.1.2/ Point Cartésien:

```
ecrireAbscisse (_abscisse)
    this.abscisse = _abscisse;
```

Point Polaire.

```
ecrireAbscisse (_abscisse)
    double ordonnée = this.lireOrdonnée();
    this.module = Math.sqrt(abscisse * _abscisse + ordonnée * ordonnée);
    thir.argument = Math.atan(ordonnée / abscisse);
```

Point Mixte

```
ecrireAbscisse (_abscisse)
    this.abscisse = _abscisse;
    majPolaire();
```

Commande ahlikaire privée.

4.2/

```
public class PointCartesien {

    private double abscisse;

    public ecrireAbscisse(double _abscisse){
        this.abscisse = _abscisse;
    }

    public PointCartesien(double _abscisse, double _ordonnee) {
```

⚠ Performance, réahlisabilité...

```
        this.abscisse = _abscisse;          ← performant
        this.ecrireAbscisse(_abscisse);     ← réahlisable
        this.ordonnee = _ordonnee
    }

    public void translater(double _delta_x, double delta_y) {
```

choisir

```
        ( this.abscisse += _delta_x;
        ( this.ecrireAbscisse(this.lireAbscisse + _delta_x);
        ...
    }
```

} même résultat.