Algorithmie & Programmation - Cours 5

Conception orientée objet, Partie 1

Dr. Jérémie Sublime

LISITE Laboratory - RDI Team - ISEP

jeremie.sublime@isep.fr

Plan

Plan

Problèmes traités jusqu'à présent

- Jusqu'à présent nous avons traité des problèmes simples dont le code pouvait se résumer à quelques fonctions simples tenant dans un seul fichier.
- Nous avons utilisé des types simples : int, long, double, booléens.
- Et nous avons vu quelques types de stockage de données simples : tableaux, listes, piles, etc.

Programmation procédurale simple

Ce type utilisant des types simples et structurant le code sous forme d'une succession d'appels de fonctions modulaires est appelée **programmation procédurale**.

Limites de la programmation procédurale

La programmation procédurale avec des types simples a plusieurs limites :

- Chaque fonction ne peut retourner qu'une valeur à la fois.
- L'absence de structures de données complexes rend rapidement le code peu lisible.

Exemple de problème de représentation des données

Comment stocker et manipuler les données de 300 élèves de l'ISEP si chacun d'entre eux est représenté par : Son nom, son prénom, son numéro d'étudiant, ses notes ?

Limites de la programmation procédurale

La programmation procédurale avec des types simples a plusieurs limites :

- Chaque fonction ne peut retourner qu'une valeur à la fois.
- L'absence de structures de données complexes rend rapidement le code peu lisible.

Exemple de problème de représentation des données

Comment stocker et manipuler les données de 300 élèves de l'ISEP si chacun d'entre eux est représenté par : Son nom, son prénom, son numéro d'étudiant, ses notes ?

- Un tableau de 300 strings pour les noms, un autre pour les prénoms, un tableau de 300 entiers pour le numéro d'étudiant et un tableau de 300×10 double pour les notes ?

Limites de la programmation procédurale

La programmation procédurale avec des types simples a plusieurs limites :

- Chaque fonction ne peut retourner qu'une valeur à la fois.
- L'absence de structures de données complexes rend rapidement le code peu lisible.

Exemple de problème de représentation des données

Comment stocker et manipuler les données de 300 élèves de l'ISEP si chacun d'entre eux est représenté par : Son nom, son prénom, son numéro d'étudiant, ses notes ?

- Un tableau de 300 strings pour les noms, un autre pour les prénoms, un tableau de 300 entiers pour le numéro d'étudiant et un tableau de 300×10 double pour les notes ?
- On voit bien que de telles données seront difficilement manipulables ...

Les structures de données

- La solution consiste à créer une **structure de données personnalisée** pour représenter des types complexes tels que les étudiants.
- En **programmation orientée objet**, une telle structure s'appelle une **classe**.

```
CLASSE Etudiant :
    Nom : String
    Prenom : String
    Numero : entier
    Notes : tableau de 10 doubles
FIN CLASSE
```

Les structures de données

- La solution consiste à créer une **structure de données personnalisée** pour représenter des types complexes tels que les étudiants.
- En **programmation orientée objet**, une telle structure s'appelle une **classe**.

```
CLASSE Etudiant :
    Nom : String
    Prenom : String
    Numero : entier
    Notes : tableau de 10 doubles
FIN CLASSE
```

Remarque

En réalité, nous avions déjà vu notre première structure de type classe dans le cours précédent avec les files.

Structures du code en programmation orientée objet

En programmation orientée objet, les classes ne se limitent pas à de simples structures de données. Les classes peuvent en effet contenir des **méthodes** qui s'applique spécifiquement à leur propre structure de données.

- Une classe rectangle aura par exemple des méthodes permettant de calculer son périmètre, une autre pour son aire, etc.
- Pour la classe étudiant, on pourrait imaginer une fonction pour calculer sa moyenne à partir de ses notes.

Remarque

Lors des TP précédents sur les listes, nous aurions pu (et aurions du) coder la majorité des fonctions demandées comme méthodes de la classe liste.

Programmation orientée objet et algorithmie

La programmation orientée objet a de nombreuses implications en algorithmie :

- Elle rend possible l'utilisation de structures de données plus complexes sous la forme de classes.
- Elle permet de sortir de la programmation procédurale classique en organisant son code de manière à avoir des fonctions spécifiques à certaines classes.
- Penser un problème algorithmique en terme de programmation orientée objet nécessite d'avantage de réflexion pour concevoir les structures et organiser son code.
- Un code bien organisé en suivant le principe de la programmation orientée objet est plus modulable, plus lisible, et plus facilement évolutif.

Plan

Qu'est ce qu'une classe ?

Classe : Définition

Une classe est un **type structuré** de donnée définit par : Un ensemble d'**attributs** et un ensemble de **méthodes**.

- En Java, chaque classe doit être écrite dans un fichier à part.
- Chaque fichier doit contenir une seule et unique classe

Exemple de classe

```
public class ListeEntier{
   /*les attributs*/
   int entier;
   ListeEntier suivant=null;

   public ListeEntier(int i){ /*un constructeur*/
      this.entier=i;
   }

   /*une fonction pour ajouter un entier en fin de liste*/
   public void AddElement(int i){
      ListeEntier L=this;
      while(L.suivant!=null){
         L=L.suivant;
      }
      L.suivant = new ListeEntier(i);
   }
}
```

Qu'est ce qu'un objet ? (1/2)

Objet : Définition

Une objet est une **instance de classe** : C'est l'attribution de valeurs au différents attributs qui crée l'objet.

- En Java, l'instanciation d'un objet se fait avec le mot clé **new**.
- La classe étant un modèle, il peut y avoir plusieurs instances de la même classe, et donc plusieurs objets d'une même classe.
- Les méthodes non-statiques de chaque classe ne peuvent être utilisées que si l'objet est instancié (puisqu'elles portent sur les attributs).

```
public class Main {
   public static void main(String [] args) {
      maListe L = new Liste(2); /*creation d'une liste contenant
          un unique element valant 2.*/
   }
}
```

Qu'est ce qu'un objet ? (2/2)

Dans le code d'une classe, l'instance de l'objet courant est désignée par le mot clé **this** qui donne accès à l'objet lui-même et à ses attributs.

```
public class ListeEntier{
   int entier;
   ListeEntier suivant=null;

   public ListeEntier(int i){
      this.entier=i; /*utilisation du mot clef this pour
          affecter l'attribut "entier"*/
   }
```

Principe d'encapsulation

La programmation orientée objet utilise ce qu'on appelle le principe d'**encapsulation** :

- On va chercher à faire en sorte que les **attributs** qui composent les objets ne soient accessibles qu'en suivant certaines routines définies par la classe.
- Certains attributs ne pourront être accédés, manipulés ou modifiés depuis l'extérieur de la classe qu'en passant par des **méthodes**. Ils ne seront pas directement accessibles.
- De même certaines fonctions et méthodes auront aussi un accès restreint.

Attributs d'une classe

Comme nous l'avons vu, les classes sont premièrement des structures de données. Elles peuvent donc contenir des attributs de plusieurs types :

- Des types de base : entiers, flottants, longs, chaines de caractères, etc.
- Des objets complexes : Objets pré-définis en Java ou objets définis dans d'autres classes.
- Des tableaux de types de bases, ou d'objets.

Attributs d'une classe

Comme nous l'avons vu, les classes sont premièrement des structures de données. Elles peuvent donc contenir des attributs de plusieurs types :

- Des types de base : entiers, flottants, longs, chaines de caractères, etc.
- Des objets complexes : Objets pré-définis en Java ou objets définis dans d'autres classes.
- Des tableaux de types de bases, ou d'objets.

Accéder aux attributs

- Dans le code d'une classe, les attributs sont appelés en utilisant le mot clé **this** suivi du nom de l'attribut : *this.attribut*.
- A l'extérieur de la classe, si l'attribut est public, on y accède par le nom de la variable et de l'attribut : *monObjet.attribut*.

Attributs publics, protégés et privés (1/2)

Les mots clés **public**, **protected** et **private** peuvent être utilisés devant la déclaration d'un attribut dans une classe et on déterminer si cet attribut est accessible ou non depuis d'autres classes :

- public (par défaut) : L'attribut sera accessible depuis n'importe quelle autre classe
- private : L'attribut ne pourra être lu et utilisé que dans des méthodes de sa propre classe, et ne sera pas accessible de l'extérieur.
- protected : Mêmes propriétés que private, mais l'attribut sera aussi accessible depuis les classes filles (voir prochain cours).

Attributs publics, protégés et privés (2/2)

```
public class Etudiant{
   String nom /*public par defaut*/
   public String prenom /*public*/
   private int numeroEtudiant /*privé*/
}
```

Attributs publics, protégés et privés (2/2)

```
public class Etudiant{
   String nom /*public par defaut*/
   public String prenom /*public*/
   private int numeroEtudiant /*privé*/
}
```

```
public class Main {
   public static void main(String [] args) {
      Etudiant E = new Etudiant(1,"Jules","Martin");
      E.prenom="Georges"; /*modifie le prenom de l'etudiant*/
      System.out.println(E.nom); /*affiche "Martin"*/
      E.numeroEtudiant = 17 /*NE COMPILE PAS !!!!!!!*/
   }
}
```

Encapsulation et attributs privés ou protégés

Utiliser le principe d'encapsulation en rendant certains attributs protégés ou privés, et donc inaccessibles de l'extérieur, peut avoir plusieurs avantages :

- Faire passer la modification des attributs par des méthodes propres à la classe (setters/mutateurs) permet de vérifier des contraintes d'intégrité sur l'attribut.
- Faire passer la modification des attributs par des méthodes propres à la classe (setters/mutateurs) permet de vérifier des contraintes d'intégrité sur l'objet.
- Faire passer la modification des attributs par des méthodes propres à la classe (setters/mutateurs) rend plus simples d'éventuelles mutations en cascades sur des sous-objets.

Les attributs statiques

Le mot clé **static** peut être utilisé devant un attribut pour définir une variable qui soit une variable associée à l'ensemble de la classe plutôt qu'une variable liée à un objet instancié : Par exemple un compteur !

- Contraitement aux attributs "classiques", un attribut statique peut être appelé sans instancier l'objet : *maClasse.monAttribut*.
- La valeur d'un attribut classique est partagé entre toutes les instances de la classe : modifier sa valeur sur un objet de cette classe la modifiera pour tous les autres.

Les attributs finaux

Le mot clé **final** utilisé devant un attribut (statique ou non) rendra sa valeur définitive et impossible à changer. C'est utile pour définir des constantes comme π.

```
public class MaClass {
    public final double PI = 3.14159; // Impossible de modifier la valeur
}
```

Les attributs finaux

Le mot clé **final** utilisé devant un attribut (statique ou non) rendra sa valeur définitive et impossible à changer. C'est utile pour définir des constantes comme π.

```
public class MaClass {
    public final double PI = 3.14159; // Impossible de modifier la valeur
}
```

Remarque

Le mot clé **final** utilisé devant une variable locale la rend également immuable.

```
public static int ajouterCinq( final int a){
    final int b = 5;
    return a + b;
}
```

Bien définir ses attributs

- Le choix des attributs d'une classe doit être un choix logique et réfléchi :
  - Les attributs décrivent les objets instanciés : ils doivent être des composants clés et indispensables des objets.
  - Les attributs statiques sont des constantes applicables à tous les objets de la classe concernée. Ils sont en principe peu nombreux.

Détecter les attributs mal placés

- Si votre classe contient des attributs qui décrivent plusieurs objets logiques différents, vous avez un problème : Il faut soit créer plusieurs classes pour séparer les objets logiques, soit remettre les attributs dans les bonnes classes.
- Attention : une classe peut cependant avoir des objets comme attributs !

Les constructeurs (1/2)

En programmation orientée objet, un **constructeur** est une méthode de classe spécifique qui permet de définir comment instancier un objet de cette classe avec le mot clé **new** :

- Les constructeurs sont des méthodes publiques qui doivent obligatoirement porter le même nom que la classe qu'elles permettent d'instancier.
- En Java, une classe peut avoir un ou plusieurs constructeurs : Chaque constructeur peut avoir un nombre de paramètres différents, ou juste des types de paramètres différents.

Les constructeurs (2/2)

```
public class Etudiant{
    public String nom
    public String prenom
    protected int numeroEtudiant
    /*1er constructeur*/
    public Etudiant(int i, String p, String n){
        this.numeroEtudiant=i;
        this.prenom=p;
        this.nom=n;
    }
    /*un 2eme constructeur*/
    public Etudiant(String p, String n){
        this.prenom=p;
        this.nom=n;
    }
}
```

Les constructeurs (2/2)

```
public class Etudiant{
    public String nom
    public String prenom
    protected int numeroEtudiant
    /*1er constructeur*/
    public Etudiant(int i, String p, String n){
        this.numeroEtudiant=i;
        this.prenom=p;
        this.nom=n;
    }
    /*un 2eme constructeur*/
    public Etudiant(String p, String n){
        this.prenom=p;
        this.nom=n;
    }
}
```

```
public class Main {
    public static void main(String [] args) {
        Etudiant E1 = new Etudiant(1,"Jules","Martin"); /*appel constructeur 1*/
        Etudiant E2 = new Etudiant("Rim","Dachraoui"); /*appel constructeur 2*/
    }
}
```

Méthodes de classe

Les méthodes de classes sont toutes les fonctions qui permettent d'effectuer des modifications ou des opérations à partir des attributs de la classes :

- Les **accesseurs (getters en anglais)** sont un type de méthodes publiques de base renvoyant la valeur d'un attribut donné depuis l'extérieur de la classe.
- Les **mutateurs (setters en anglais)** sont d'autres méthodes publiques de base permettant de modifier la valeur d'un attribut donné depuis l'extérieur de la classe.
- Les autres types de méthodes n'ont pas de nom particulier et permettent toutes sortes de calculs, d'affichages ou de modifications à partir des attributs.

Remarque

Tout comme les constructeurs, les getters et setters peuvent être générés automatiquement par éclipse.

Getters et Setters : exemple

```
public class Etudiant{
    public String nom
    public String prenom
    /*constructeur*/
    public Etudiant(String p, String n){
        this.prenom=p;
        this.nom=n;
    }
    /*getters*/
    public String getNom(){
        return this.nom;
    }
    public String getPrenom(){
        return this.prenom;
    }
    /*setters*/
    public void setNom(String n){
        this.nom=n;
    }
    public void setPrenom(String p){
        this.prenom=p;
    }

}
```

Comment choisir quelles méthodes définir ?

- Au delà des getters et setters, le choix des méthodes à coder dans une classe se fait selon les besoins du programme.
- Chaque classe ne doit contenir que des méthodes qui la concerne et qui utilise ses attributs : Si vous codez dans une classe une méthode ne manipulant aucun des attributs de la classe, votre méthode est probablement dans la mauvaise classe ou au mauvais endroit.
- Le choix de placer une méthode dans une classe doit se faire selon la logique de la programmation orientée objet : Est-il logique que cette classe possède cette méthode ?

Comment choisir quelles méthodes définir ?

- Au delà des getters et setters, le choix des méthodes à coder dans une classe se fait selon les besoins du programme.
- Chaque classe ne doit contenir que des méthodes qui la concerne et qui utilise ses attributs : Si vous codez dans une classe une méthode ne manipulant aucun des attributs de la classe, votre méthode est probablement dans la mauvaise classe ou au mauvais endroit.
- Le choix de placer une méthode dans une classe doit se faire selon la logique de la programmation orientée objet : Est-il logique que cette classe possède cette méthode ?

Savoir reconnaitre un code pourri

Si vous mettez une fonction dans un fichier de classe avec lequel elle n'a rien avoir parce que votre programme ne compile pas lorsque vous mettez ce code ailleurs : Vous avez un problème de conception.

Méthodes publiques, protégées et privées

Comme pour les attributs, les mots clés **public**, **protected** et **private** peuvent être utilisés devant la déclaration d'une fonction dans une classe et on déterminer si cette méthode est accessible ou non depuis d'autres classes :

- public (par défaut) : Sera accessible depuis n'importe quelle autre classe
- private : Ne utilisée que dans d'autres méthodes de sa propre classe, et ne sera pas accessible de l'extérieur.
- protected : Mêmes propriétés que private, mais sera aussi accessible depuis les classes filles (voir prochain cours).

Encapsulation et méthodes privés ou protégés

Utiliser le principe d'encapsulation en rendant certaines méthodes privées ou protégées peut avoir plusieurs avantages :

- Certaines méthodes de classe peuvent ne pas être appelable de l'extérieur pour des raisons de sécurité et n'être exécutables que dans le cadre d'autres méthodes publiques.
- L'encapsulation permet permet d'offrir une interface orientée services et responsabilités : On sait facilement qui a accès à quelles méthodes et fonctions et ainsi quelles sont les responsabilités de telle classe ou tel module.

Méthodes statiques et non statiques

Comme pour les attributs, une méthode peut-être statique :

- Une méthode statique peut être utilisée sans instancier l'objet ou depuis un objet déjà instancié.
- **Une méthode statique ne peut utiliser que des attributs eux-mêmes statiques** ! Sinon le code ne compilera pas.

Remarque

Dans tous les TPs précédent, nous avons utilisé uniquement des méthodes statiques car nous n'avions pas d'objets. Il s'agit en réalité d'un abus, et les méthodes statiques doivent être évitées autant que possible en programmation orientée objet, sauf lorsqu'elles sont réellement nécessaires (utilisations de variables de classe).

Plan

Comment organiser son code

- En Java, chaque classe doit être écrite dans un fichier à part : Ainsi, chaque fichier contiendra le code décrivant les différents attributs et méthodes des objets utiles pour l'algorithme ou le programme créé.
- Chaque fichier doit contenir une seule et unique classe

Comparaison entre la programmation orientée objet et les bases de données

Le modèle utilisé en programmation orienté objet a des similitudes avec les modèles utilisés pour les bases de données :

- Les tables simples (sans tenir compte des tables de jonctions) et les objets simples (sans héritages) sont pensés et conçus de manière similaire.
- Le **mapping objet-relationnel** est d'ailleurs une technique de programmation avancée permettant de mapper, manipuler et stocker directement des objets en base de données.

Java et encapsulation

Il est toujours important de garder à l'esprit le principe d'encapsulation lors de la créations des classes Java :

- Cet attribut a-t-il besoin d'être accessible et modifiable de l'extérieur ? Si non, le mettre en protégé et mettre des getters et setters.
- Y a-t-il des contraintes d'intégrité sur tels attributs ? Si oui les mettre en privé et définir les contraintes dans les setters.
- Par défaut les méthodes sont généralement publiques, cependant certaines méthodes peuvent aussi poser des problèmes d'intégrité si elles sont directement appelées de l'extérieur en dehors du cadre d'une autre méthode ou procédure. Si c'est le cas, elles doivent être privées ou protégées.

Remarque

En programmation orientée objet, on a tendance à favoriser l'encapsulation la plus forte possible.

Java : Un langage orienté objet

Java est un langage qui a été pensé et conçu pour la programmation orientée objet :

- L'intégralité du code d'un programme Java (à l'exception de la classe contenant le main) doit être pensé en terme d'objets.
- Il est nécessaire d'avoir des objets bien définis pour avoir un code Java propre et fonctionnel : Les objets ont des attributs qui leurs sont propres et des méthodes qui manipulent ses attributs.
- L'usage abusif de fonctions statiques ou d'attributs n'appartenant pas aux objets dans lesquels ils sont définis doivent être bannis.
- L'exercice mental de penser et de coder en orienté objet n'est pas simple au début mais vaut le coup : Code plus modulable, plus évolutif, mieux organisé et plus facilement lisible.

Plan

Dans les prochains cours

Programmation orientée objet

- Introduction à UML
- Diagrammes de classes : héritages, classes asbtraites, interfaces, aggrégations, compositions.
- Réflexions sur la programmation orientée objet