

1

## CONCEPTION LOGICIELLE



Présentée par: Mme HASSAM-OUARI Kahina  
 Email: [kahina.hassam@yncrea.fr](mailto:kahina.hassam@yncrea.fr)  
 Bureau: T336  
 Département: Organisation, Management et Informatique



2

## La programmation orientée Objets Introduction

- Principes de base de la POO
- Qu'est ce qu'une application Objet
- Les propriétés de l'approche OO
- Exemple d'une application orientée Objets

3

## Un Objet, Kézako?

- Un objet peut être définie comme une boîte aux lettres qui reçoit et envoie des messages



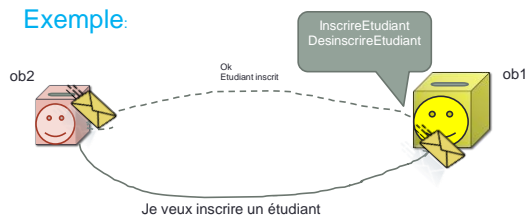
- Un Objet est caractérisé par:
  1. Son identité ou identifiant qui est unique
  2. Ses données qui lui sont propre
  3. Les méthodes et traitements qu'il peut réaliser et qu'il met à la disposition d'autres objets.

4

## Communications entre objets

- Les objets communiquent entre eux par échanges de messages.
- Les messages les plus échangés sont des demandes de traitements

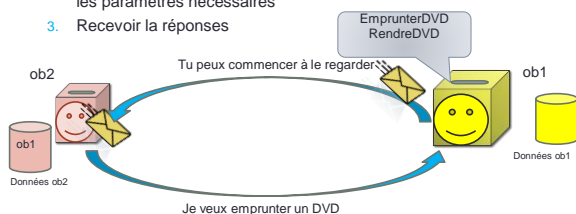
### Exemple:



5

## Demande de réalisation de traitements

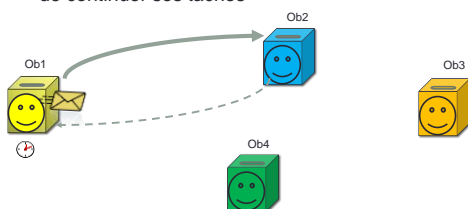
- Pour envoyer une demande de réalisation d'une tâche, un objet doit:
  1. Connaître l'id de l'objet qui va réaliser la tâche
  2. Lui envoyer un message avec le **nom du traitement (méthode)** et les paramètres nécessaires
  3. Recevoir la réponses



6

## Type de communication entre objets

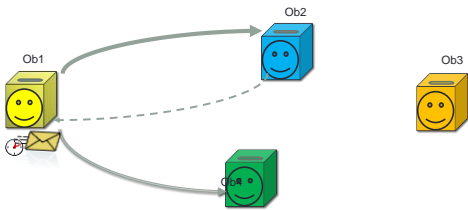
- Les objets communiquent entre eux de manière asynchrone
- L'objet qui demande le traitement attend la réponse avant de continuer ses tâches



7

Type de communication entre objets

- Les objets communiquent entre eux de manière asynchrone
- L'objet qui demande le traitement attend la réponse avant de continuer ses tâches



---

---

---

---

---

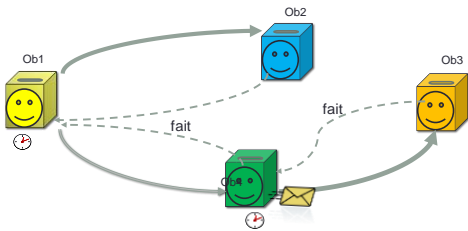
---

---

8

Type de communication entre objets

- Les objets communiquent entre eux de manière asynchrone
- L'objet qui demande le traitement attend la réponse avant de continuer ses tâches



---

---

---

---

---

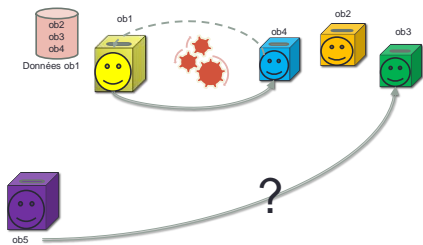
---

---

9

Création d'objets et Références

- Tout objet peut créer d'autres objets
- Le créateur possède la référence vers les objets qu'il a créé



---

---

---

---

---

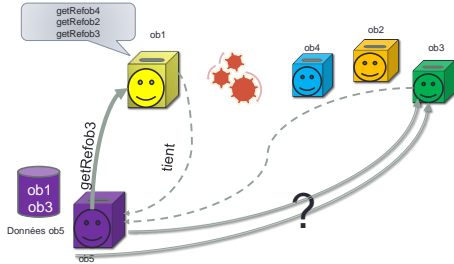
---

---

10

## Echange de Références

- Les références sont **des données**
- Il est possible de les transmettre par échange de message



11

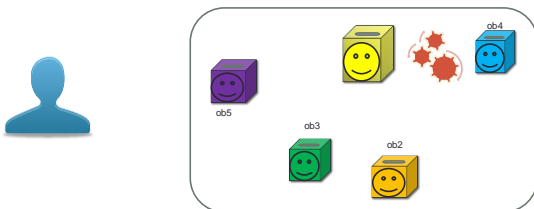
## Suppression d'objets

- A la C++
  - On peut supprimer les objets une fois qu'on a terminé de les utiliser
  - Les objets peuvent se supprimer eux mêmes
 ⇒ Attention aux objets qui les référencent
- A la java:
  - Les objets ne peuvent pas être supprimés
  - Si les objets ne sont plus référencés, ils seront supprimés
 => garbage collector

12

## Une application orientée objet

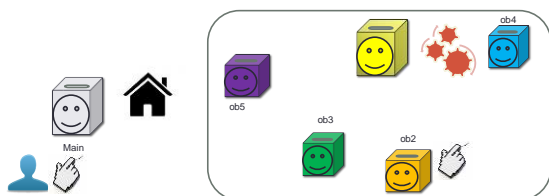
- Une application objets est un ensemble d'objets qui s'échangent des messages (communiquent) pour répondre à une demande utilisateur



13

## Démarrage de l'application objet

- La création des objets nécessaires au démarrage de l'application est réalisée par **UN SEUL** objet nommé *main*
- Chaque application est constituée d'un seul main




---

---

---

---

---

---

---

---

14

## Principes de l'approche objet L'encapsulation

- L'objet protège ses données à fin de garantir leurs intégrité
- SEUL l'objet peut modifier ses propres données
- L'objet seul sait comment sont structurés ses données




---

---

---

---

---

---

---

---

15

## Principes de l'approche objet La responsabilité

- L'objet est responsable des traitements qu'il sait faire et qu'il met à la disposition des autres objets
- Il possède donc TOUTES les données nécessaires pour réaliser ses traitements (méthodes)
- Etre responsable ne veut pas dire que l'objet doit réaliser les traitements seul, il peut faire appel à d'autres objets qu'il connaît

---

---

---

---

---

---

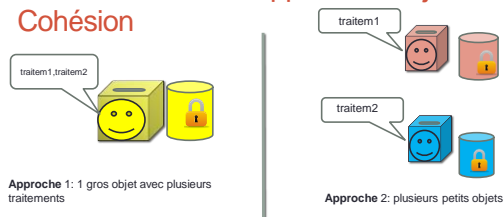
---

---

16

## Structuration d'une application objets

### Cohésion



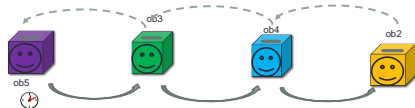
- Dans le paradigme Objet on va préférer l'approche 2, car dans l'approche 1 il sera difficile:
  - De réutiliser tout les traitements, car pas forcément besoin de tout
  - Difficulté à tester tout les traitements

17

## Structuration d'une application objets

### Le couplage

- Les objets communiquent entre eux par échanges de messages, **il faut absolument réduire les chaines de communications entre objets.**
- Les chaines de communications impliquent:
  - Les objets appelants **attendent** la réponse avant de poursuivre leurs exécutions
  - S'il faut faire des **tests**, il faut tester tout les objets avec lequel l'objet initial communique.



18

## Structuration d'une application objets

### Le Défi

- Identifier les objets nécessaires au fonctionnement de l'application
- Préciser le traitement de ces objets dans l'application
- Définir les données dont ils ont besoin
- Etablir la communication entre les objets



En maximisant la cohérence et en minimisant le couplage

19

## Exercice

- Un même référent peut-il désigner plusieurs objets ?
- Plusieurs référents peuvent-ils désigner un même et seul objet ?

---

---

---

---

---

---

---

20

Les objets sans classe n'ont pas de classe!!

---

---

---

---


---

---

---

21

## La provenance des objets

- Les objets sont créés à partir des classes 
- Une classe est un « moule » pour fabriquer des objets.
- Une classe définit un ensemble d'objets qui ont:
  - Les mêmes données (nom, type, une valeur de base )
  - Les mêmes types de traitements (nom, paramètres )
- Les objets d'une même classe ont leurs propres données
- L'identité est portée par l'objet

---

---

---

---

---

---

---

22

## Le lien entre les objets et les classes

### Instanciation

- L'opération appelée **instanciation** constitue la création d'un objet à partir d'une classe.
- Un objet créé à partir d'une classe A, cet objet est appelé une instance de la classe A.




---

---

---

---

---

---

---

---

23

## Définition d'une classe en java:

- Une classe java est codée dans un fichier (.java)
- Chaque classe possède un nom (le nom du fichier)
- Une classe possède des données → des propriétés
- Une classe possède des traitements → méthodes

```
public class Voiture {
//Les propriétés
//les méthodes
}
```

Voiture.java

```
public class Videotheque{
//propriétés
//les méthodes
}
```

Videotheque.java

---

---

---

---

---

---

---

---

24

## Propriétés d'une classe en java

Une propriété est définie par:

- Son nom (décrit à l'aide d'une chaîne de caractères)
- Son type:
  - Types de base: String, int, Boolean,...etc
  - Référence vers un autre objet
- Une valeur par défaut (OU PAS !)

### Exemple:

- String nomDvd;
- String marqueVoiture;
- int plafondCarte=200;

---

---

---

---

---

---

---

---



25

## Méthodes en JAVA

Une méthode est défini par:

- Son nom (*décrit à l'aide d'une chaîne de caractères*)
- Son type de retour (*void, basique ou même une référence*)
- Ses paramètres: chaque paramètre possédant un nom et un type
- Son code

- **Exemple:**

- `int` combienDeDVD(){ return nombreDeDVD;}

---

---

---

---

---

---

---

---

26

## Exemple d'une classe

```
public class Etudiant {  
  
    private String nom;  
    private String prenom;  
    private String adresse;  
  
    public String getNom() {  
        return nom;  
    }  
  
    public String getPrenom() {  
        return prenom;  
    }  
  
    public String getAdresse() {  
        return adresse;  
    }  
}
```

---

---

---

---

---

---

---

---

27

## Exercice

- Quels sont les propriétés et les méthodes qu'on pourrait retrouver dans une classe nommée Enseignant?
- Créer cette classe en java?

---

---

---

---

---

---

---

---

28

## Le constructeur

### Une méthode particulière

- Le constructeur est la méthode qui est appelée dès la création de l'objet.
- Le constructeur sert à initialiser les données de l'objet créé
- Il est possible qu'une classe possède plusieurs constructeurs

#### • **Attention:**

- Un constructeur n'a **pas de type de retour**
- Un constructeur porte **TOUJOURS le même nom que la classe**
- Il peut avoir des paramètres ( par défaut il n'a pas de paramètres)

---

---

---

---

---

---

---

---

29

## Le constructeur

### Exemple d'une classe avec plusieurs constructeurs

```
public class Etudiant {

    String nom;
    String prenom;
    String adresse;

    Etudiant(String nomEtu, String prenomEtu, String adrEtu){
        nom=nomEtu;
        prenom=prenomEtu;
        adresse=adrEtu;
    }
    Etudiant(String nomEtu, String prenomEtu){
        nom=nomEtu;
        prenom=prenomEtu;
    }
}
```

Constructeur 1

Constructeur 2

---

---

---

---

---

---

---

---

30

## Instanciation des classes en java

### Exemples

- La création des objets à partir de la classe se fait comme suit en java:

Operator **new** pour créer un nouvel objet      Constructeur de la classe Etudiant

```
Etudiant e1=new Etudiant("Dupont", "Alex", "Rue de Toul");
```

Type de l'objet      Le nom de l'objet

---

---

---

---

---

---

---

---

31

## UML pour la modélisation d'une application objets

---

---

---

---

---

---

---

32

## Pourquoi modéliser

- Un modèle est une simplification de la réalité qui permet de mieux comprendre le système à développer.
- Il permet
  - De visualiser le système comme il est ou comme il devrait l'être.
  - De valider le modèle vis à vis des clients
  - De spécifier les structures de données et le comportement du système.
  - De fournir un guide pour la construction du système.
  - De documenter le système et les décisions prises.

---

---

---

---

---

---

---

33

## Rôle de L'UML

- UML (Unified Modelling Language) est un langage de modélisation conçu pour construire, visualiser, et spécifier les systèmes d'information (BOOCH et al, 1998), (MORLEY et al, 2000).
- La notation UML repose sur deux concepts essentiels :
  - la modélisation du mode réel au moyen de l'approche orientée objet ;
  - l'élaboration d'une série de diagrammes facilitant l'analyse et la conception du système d'information, et permettant de représenter les aspects statiques et dynamiques du domaine à modéliser et à informatiser.

---

---

---

---

---

---

---

## UML(Unified Modeling Language)

### ❑ UML est un *langage* : il possède

- une syntaxe :
  - symboles : «l'» «vole» «avion»
  - règles de grammaire : «l'avion vole»
- une sémantique : signification des expressions syntaxiques

### ❑ UML est un *langage de modélisation* :

- modélisation : description abstraite d'un système, représentation simplifiée
  - description d'un problème : analyse
  - description d'une solution au problème : conception
- permet de *décrire* clairement un modèle, de l'*expliquer* et de le *manipuler*

---

---

---

---

---

---

---

---

## UML(Unified Modeling Language)

UML définit treize types de diagrammes :

### ❑ diagrammes structurels (6) :

- diagramme de classes : structure statique du système en terme de *classes* et de relations entre ces classes
- diagramme d'objets : structure statique du système en terme d'*objets* et de relations entre ces objets.
- ...

### ❑ diagrammes comportementaux (3) :

- diagramme de cas d'utilisation : comportement du système du point de vue de l'*utilisateur*
- ...

### ❑ diagrammes d'interactions (4)

- diagramme de séquence
- diagramme de communication
- ...

---

---

---

---

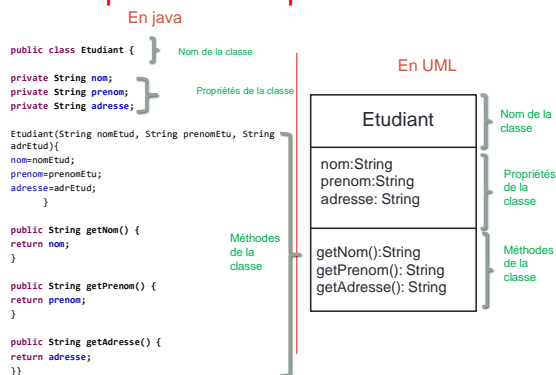
---

---

---

---

## Exemple d'une simple classe




---

---

---

---

---

---

---

---

Exemple de la vidéothèque(V1)

En java

```
public class Videotheque {  
  
    String[] dvds;  
    boolean[] emprunt;  
  
    void ajouterDvd(String nom){  
        //Ajouter le code de la methode  
    }  
  
    boolean emprunterDvd(String nom){  
        //ajouter le code  
        return true;  
    }  
}
```

En UML

Videotheque
dvds: String [] emprunt: Boolean[]
AjouterDvd(String dvd): void emprunterDvd(String nom): Boolean

---

---

---

---

---

---

---

Exemple de la vidéothèque(V2)  
Plus orientée Objets

Videotheque
dvds :DVD[]
AjouterDvd(String dvd): void emprunterDvd(String nom): Boolean

-On Remplace le tableau de chaines par un tableau de DVD.

-On rajoutera dans la classe DVD un booléen qui représentera le fait qui soit emprunté ou pas

---

---

---

---

---

---

---

Lancement de l'application  
Main JAVA

• Tout application java possède **une classe** principale qui définit absolument **une méthode main**, comme suit:

```
public class NomDeLaClasse {  
    public static void main(String[] args){  
        // Le programme commence ici  
        // instantiation des classes pour initialiser les objets  
        //qui vont constituer l'application  
    }  
}
```

---

---

---

---

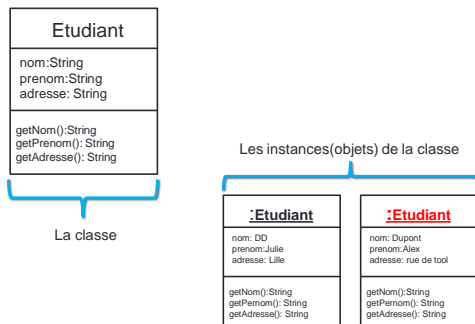
---

---

---

40

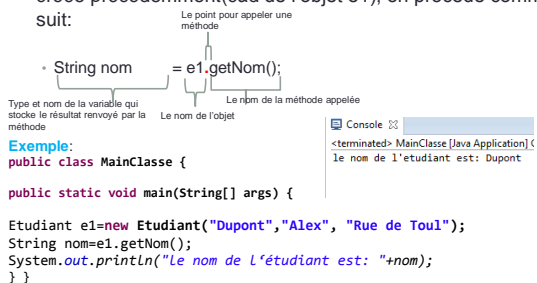
## Représentation d'un objet en UML



41

## Appel de méthode en java

- Pour récupérer la valeur de la variable « nom » de l'étudiant créée précédemment(càd de l'objet e1), on procède comme suit:



42

## Notion de Package

- Ce mécanisme permet de regrouper des éléments. Les éléments à l'intérieur du package auront des accès privilégiés.
- Le nom des classes est unique à l'intérieur d'un package
- L'idée du package est de rapprocher les éléments sémantiquement proches, offrant un ensemble de services homogènes et cohérents.
- Ils séparent le modèle en éléments logiques, et montrent leurs interactions à un plus haut niveau.

- Déclaration d'un package en java:

```
package nomPackage;
```

En résumé:

```
package exemple.ecole;
public class Etudiant {

    private String nom;
    private String prenom;
    private String adresse;

    Etudiant(String nomEtu, String prenomEtu, String
    adresseEtu){
        nom=nomEtu;
        prenom=prenomEtu;
        adresse=adresseEtu;
    }

    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getAdresse() {
        return adresse;
    }
    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }
}

package exemple.ecole;
public class MainClasse {

    public static void main(String[] args) {

        Etudiant e1=new Etudiant("Dupont","Alex",
        "Rue de Toul");
        String nom=e1.getNom();
        System.out.println("Le nom de l'étudiant
        est: "+nom);
    }
}
```

---

---

---

---

---

---

---

---

Exemple de la vidéothèque

- Une classe Dvd:
  - String nom, Boolean emprunte
- Une classe Videotheque:
  - LinkedList<Dvd>
  - AjouterDvd(Dvd cd): void
  - emprunterDvd(Dvd cd ): Boolean

---

---

---

---

---

---

---

---

Exemple de la vidéothèque  
En java ça donne quoi?

```
package hei.ExempleCours;

public class Dvd {
    String nom;
    Boolean emprunte;

    // constructeur de la classe Dvd
    Dvd(String nom){
        this.nom=nom;
        this.emprunte=false;
    }

    // Methode qui met à jour l'etat //du
    Dvd quand il sort de la //vidéothèque
    void emprunter(){
        this.emprunte=true;
    }

    // Methode qui met à jour l'etat //du
    Dvd qund il est retourné
    void rendre(){
        this.emprunte=false;
    }
}

package hei.ExempleCours;
import java.util.HashMap;
import java.util.LinkedList;

public class Videotheque {
    LinkedList<Dvd> listeDesDvd;

    //Constructeur de la classe, il crée la liste des //Dvd
    Videotheque(){
        listeDesDvd= new LinkedList<Dvd>();
    }

    void ajouterDvd(Dvd cd){
        //Ajout du dvd à la liste
        listeDesDvd.add(cd);
    }

    void emprunterDvd(Dvd cd){
        cd.emprunter();
        listeDesDvd.remove(cd);
    }

    void rendreDvd(Dvd cd){
        // Raj du boolean emprunté
        cd.rendre();
        listeDesDvd.add(cd);
    }

    void afficherLaListeDVD() {
        for (int i = 0; i < listeDesDvd.size(); i++) {
            System.out.println(listeDesDvd.get(i).nom);
        }
    }
}
```

---

---

---

---

---

---

---

---

48

## A vous 😊

- Maintenant, que vous avez les deux classes précédentes, créez un Main dans lequel vous:
- Créez une vidéothèque à la quelle on ajoute des DVDs
- emprunter un dvd
- rendre un dvd
- Affiche le contenu de la videothèque

---

---

---

---

---

---

---

---

49

## La solution

```
package hei.ExempleCours;

public class main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Videotheque fnac= new Videotheque();

        Dvd d1= new Dvd("LA PLANETE DES SINGES : L'AFFRONTMENT");
        Dvd d2= new Dvd("Lucy");
        Dvd d3= new Dvd("Spider man");
        Dvd d4= new Dvd("Lucifer");

        fnac.ajouterDvd(d1);
        fnac.ajouterDvd(d2);
        fnac.ajouterDvd(d3);
        fnac.ajouterDvd(d4);

        fnac.afficherLaListeDVD();

        fnac.emprunterDvd(d2);
        fnac.afficherLaListeDVD();}}
```

---

---

---

---

---

---

---

---

50

## Mise en œuvre du mécanisme d'encapsulation en java

- Doit-on accéder directement la propriété *emprunte* pour récupérer sa valeur?



Mécanisme d'encapsulation



- L'encapsulation en java est mise en œuvre comme suit:
  - Les propriétés sont déclarées « private » → pour que personne n'y accède
- Il existe néanmoins d'autres visibilitées:
  - public → tout le monde peut y accéder
  - package → accessible par les classes du même package

---

---

---

---

---

---

---

---



51

## Récupérer / Modifier une propriété Les getters et les setters

- Pour récupérer / modifier la valeur des propriétés, on définira des méthodes qui s'appellent
  - Getter → pour récupérer la valeur de la propriété (Lecture)
  - Setter → pour modifier la valeur de la propriété (écriture)

### Exemple:

String getNom(); // Méthode pour récupérer la valeur du nom du dvd  
String setNom(String name); // Pour modifier le nom d'un dvd



#### A noter:

Sur eclipse les getters et les setters peuvent être générés automatiquement

52

## Récupérer / Modifier une propriété Les getters et les setters

- Pour récupérer / modifier la valeur des propriétés, on définira des méthodes qui s'appellent
  - Getter → pour récupérer la valeur de la propriété (Lecture)
  - Setter → pour modifier la valeur de la propriété (écriture)

### Exemple:

String getNom(); // Méthode pour récupérer la valeur du nom du dvd  
String setNom(String name); // Pour modifier le nom d'un dvd



#### A noter:

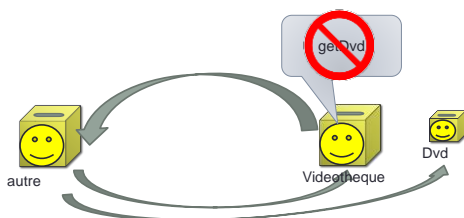
Sur eclipse les getters et les setters peuvent être générés automatiquement

Si tout le monde peut utiliser ces méthodes donc pourquoi déclarer les propriétés « privé » ?

53

## Encapsulation et objet interne

- Si un Objet utilise un autre objet (objet interne), il ne faut jamais donner sa référence car ça nuit à l'**intégrité** des données



54

## Règles à suivre

- Propriétés:
  - Toujours les déclarer private
- Getter
  - private ou public (tant qu'on ne retourne pas d'objet interne)
- Setter:
  - private sauf si l'objet en cours sert de mémoire partagée(besoin de modification par tout le monde)

---

---

---

---

---

---

---

55

## Responsabilité

- Lors du développement d'une application orientée objet, on se pose la question de « ou est ce que je mets mes méthodes »?



Mettez les méthodes dans les classes qui contiennent ou peuvent récupérer toutes les informations nécessaires à sa définition

---

---

---

---

---

---

---

56

## Conclusion

- Une classe mécanisme qui permet de décrire la structure d'un ensemble d'objet (typage)
- Encapsulation → protéger les données
  - Attention à la mise à disposition des références des objets
- La responsabilité → la classe possède tout les éléments pour répondre au traitement

---

---

---

---

---

---

---

Exemple de cours

Book
author : Author
title : String
publicationYear : int
text : String
print()
getTitle() : String
getAuthor() : Author

Author
name : String
firstname : String
birthYear : int
deathYear : int
getName() : String
getDeathYear() : int
setDeathYear(death : int)
toString() : String

---

---

---

---

---

---

---

---

Exemple de cours  
Classes java

```
public class Book {
// les attributs de la classe Book
private Author author;
private String title;
private int publicationYear;
private String text;

// constructeur
public Book(Author someAuthor, String title, int pubYear, String text) {
    this.author = someAuthor;
    this.title = title;
    this.publicationYear = pubYear;
    this.text = text;
}

// les méthodes de la classe Book
public void print() {
    System.out.println(this.text);
}

public Author getAuthor() {
    return this.author;
}

public String getTitle() {
    return this.title;
}
}
```

---

---

---

---

---

---

---

---

Exemple de cours  
Classes java- A vous..

- Ecrivez la classe java « Author » en définissant ses propriété et ses méthodes;
- Ecrivez un programme principale java ou vous déclarez des Auteurs et des Livres

---

---

---

---

---

---

---

---

### Exercice

- Soit A, B et C trois classes. On souhaite réaliser un échange de message en cascade.
- Définissez ces classes avec leurs propriétés et méthodes.
- Écrivez un pg java réalisant un envoi de messages en cascade entre trois objets .

---

---

---

---

---

---

---