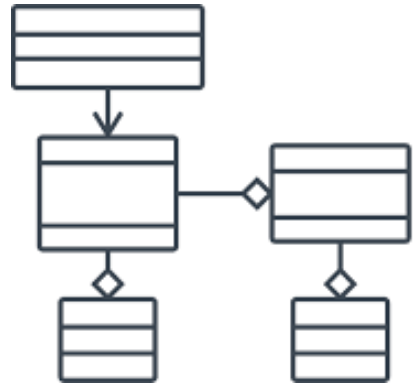
Les principes de base de la programmation objet

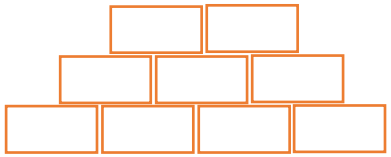


Dessinez moi une maison

5 minutes

Le développement est comme la construction

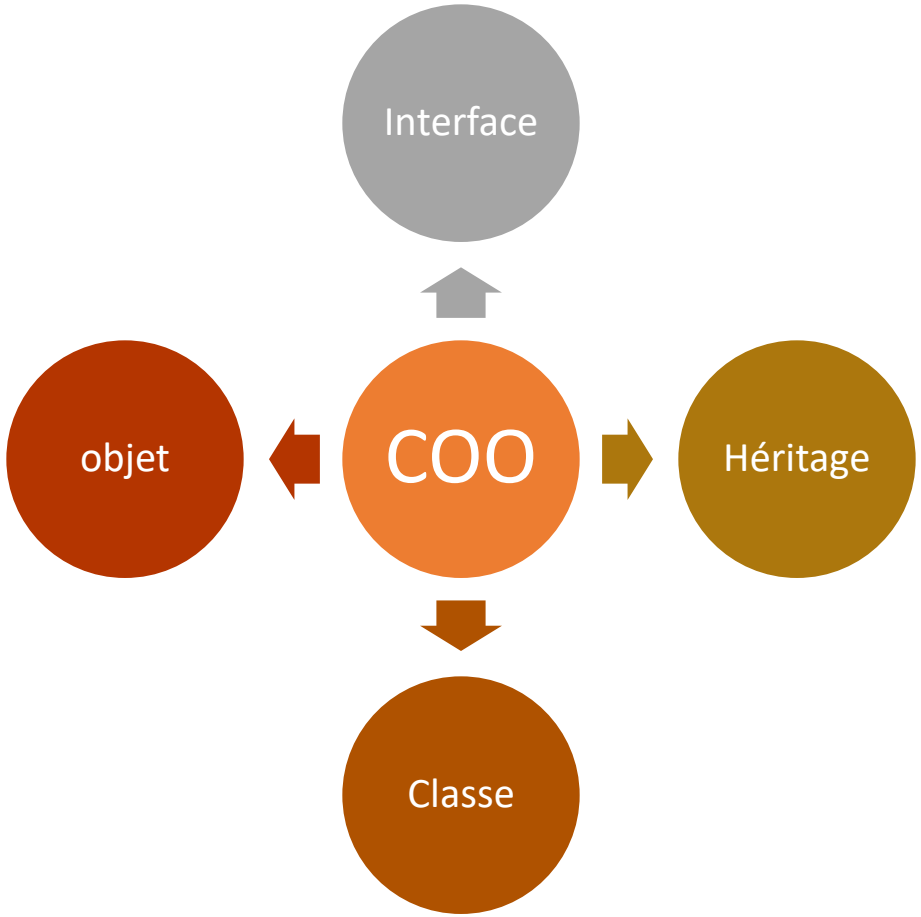
1. Un cahier des charges
2. Un plan
3. Des règles et des bonnes pratiques
4. De bons artisans



briques

objets

Conception orientée objet



Principe S.O.L.I.D

Single responsability

Open close

Liskov substitution

Interface segregation

Dependency inversion

Principe S.O.L.I.D

Responsabilité unique (Single responsibility principle)

« Une classe ne devrait avoir qu'une et une seule raison de changer »

Conséquences du non respect de cette règle :

- Le code est dupliqué de toute part
- Les classes deviennent de plus en plus grandes
- La maintenant est complexe
- La compréhension des traitements est compliquée

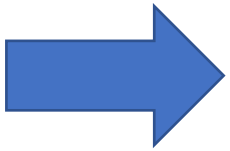
OU

« Une classe ne doit avoir qu'une seule et unique responsabilité »

Principe S.O.L.I.D

Responsabilité unique (Single responsibility principle)

```java
public class UserService {

    public  User create(User user) {
        // method body
        return user;
    }

    public User update(User user){
        // method body
       return user;
    }

    public void delete(Long id) {
        // method body
    }

    public String formatBirthDate(Date birthDate) {
        // method body
       return "";
    }

    public boolean checkUsers(User firstUser, User secondUser){
        // method body
        return true;
    }

}
```



Class UserService
Create(User user)
Update(User user)
Delete(Long id)

Gère la mise à jour des données d'un client

Class DateHelper
formatDate(Date, String format)

Gère les opérations sur les Date.

Class User
compare(user user)

Permet de comparer deux utilisateurs

Principe S.O.L.I.D
Ouvert/fermé (Open/closed principle)

« Les objets ou entités devraient être ouverts à l'extension mais fermés à la modification.»

On favorise l'utilisation du polymorphisme et de l'abstraction :

- L'héritage
- Les design patterns structurel
- La surcharge

On évite des régressions suite à l'ajout de nouvelles fonctionnalités

OU

« On doit pouvoir changer un comportement sans devoir modifier la définition des méthodes d'une classe. »

Principe S.O.L.I.D
Ouvert/fermé (Open/closed principle)

Comportement habituelle d'une voiture

```java
public class Car {

    public void run() {
        log.info( msg: "La voiture roule");
    }

    public void stop() {
        log.info( msg: "La voiture s'arrête");
    }

}
```

Je veux gérer les voitures amphibies utiliser par l'armée qui navigue sur l'eau.

```java
public class AmphibiousCar extends Car {

    public void navigate() {
        log.info( msg: "La voiture navigue sur l'eau");
    }

}
```

Principe S.O.L.I.D

Substitution de Liskov (Liskov substitution principle)

« Les objets dans un programme doivent être remplaçables par des instances de leur sous-type sans pour autant altérer le bon fonctionnement du programme.»

Les règles a appliquer ne sont pas sujets à interprétation pour une méthode :

- Mêmes paramètres en entrée que la classe mère ou des sous-types de ceux-ci
- Mêmes paramètres en sortie que la classe mère ou des sous-types de ceux-ci
- Mêmes exceptions lancées ou des sous-types de celles-ci

On évite des régressions suite à l'ajout de nouvelles fonctionnalités

OU

« Une classe dérivée doit toujours se comporter comme sa classe parente on doit pouvoir les substituer »

Principe S.O.L.I.D
Substitution de Liskov (Liskov substitution principle)

Contrairement à d'autres langages il n'est pas possible en java de ne pas respecter ce principe, une erreur à la compilation sera lancer.
La redéfinition de méthode d'une classe mère dans une classe fille respecte le principe de Liskov.

Ce qu'il faut retenir c'est que ce principe revient à formaliser un contrat applicable sur les méthodes des classes fille d'une classe parente.

Principe S.O.L.I.D

Ségrégation des interfaces (Interface segregation principle)

« Plusieurs interfaces spécifiques valent mieux qu'une seule interface fourre-tout »

L'objectif est :

- Limiter les responsabilités d'une interface
- Avoir un code modulable, réutilisable
- Respecter le principe de responsabilité unique

OU

« Aucun client ne devrait dépendre de méthodes qu'il n'utilise pas»

Principe S.O.L.I.D

Inversion des dépendances (Dependency inversion principle)

« Une classe doit dépendre d'abstraction (Interfaces), pas d'implémentation (Classes).

Objectifs :

- Permet le découplage, on peut changer une implémentation sans modifier le code
- Facilite les tests unitaires par l'injection de Mock

OU

« On ne passe pas un objet en paramètre lorsqu'une interface est disponible»

Principe S.O.L.I.D

- Ces principes, compris et suivis :
  - Diminue le couplage (la modification d'une classe à peu d'impact sur les autres)
  - Favorise les évolutions
  - Améliore la lisibilité
- Il ne doivent pas être utilisé de manière systématique, il faut :
  - savoir faire des exceptions
  - faire preuve de bon sens.
- Le mettre mot est de garder le code le plus simple possible (principe KISS) et ne développer que ce qui est nécessaire (principe YAGNI)
- SOLID est mise en œuvre dans certains design Patterns.