

Introduction à l'architecture logicielle

Introduction : ingénierie logicielle

Crise du logiciel dans les années 70 :

- Augmentation de la puissance des machines
- Logiciels de plus en plus complexes
- Langages complexes et rudimentaires : Assembleur, Fortran, Cobol

Problématique :

- Baisse de la qualité du code
- Explosion des coûts
- Retards
- Logiciels moins fiables

Émergence de l'ingénierie logicielle

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."—**IEEE Standard Glossary of Software Engineering Terminology**

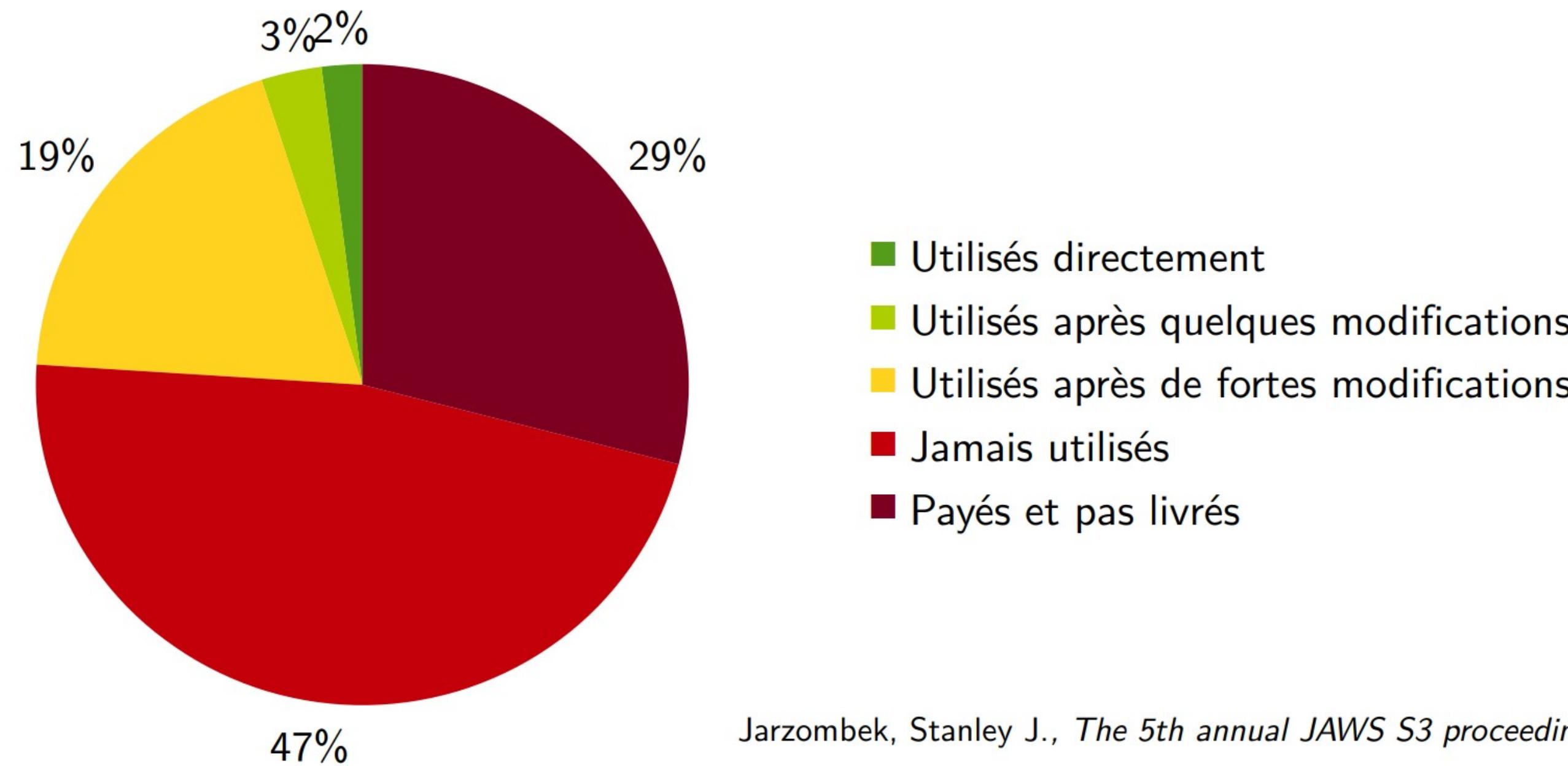
- Gérer la complexité
- Faciliter le travail en équipe
- Maîtriser les budgets
- Améliorer les outils
- Adopter des méthodes et processus de développement pour rationaliser la production d'un logiciel et sa maintenance

Les grands domaines de l'ingénierie logicielle

- Software requirements
- Architecture logicielle
- Gestion de projet
- Qualité et maintenance logicielle
- Processus et méthode de conception

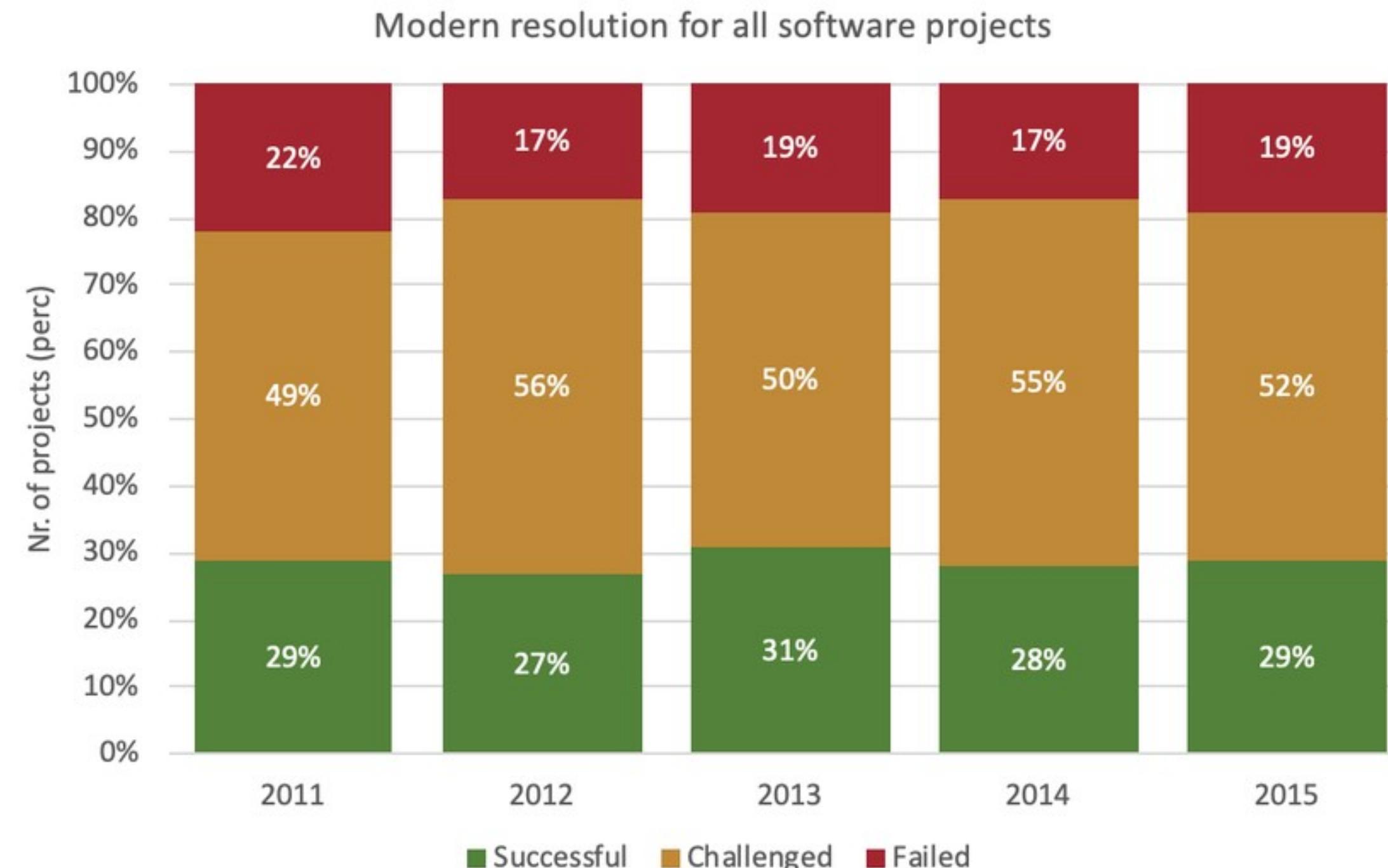
Étude de cas

Department of Defense des États-Unis sur les logiciels de 9 gros projets militaires



Jarzombek, Stanley J., *The 5th annual JAWS S3 proceedings*, 1999

Étude de cas



2015 CHAOS report from Standish Group

Programmation Orientée Objet : Rappel

Objectifs

Extensibilité : capacité d'ajouter de nouvelles fonctionnalités facilement

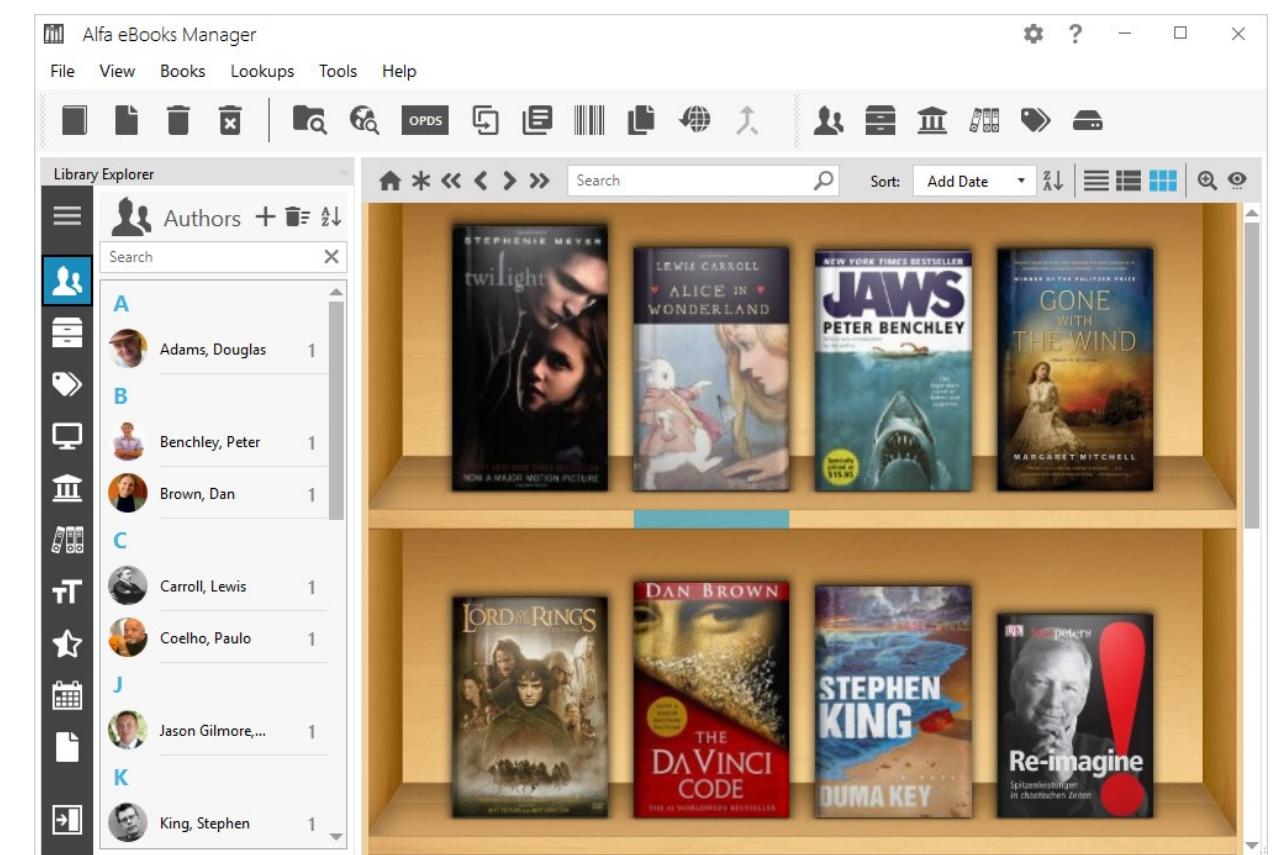
Adaptabilité/Reutilisabilité : capacité à réutiliser un code dans divers contexte

Modularité : créer des codes indépendants, portable et interchangeable

Maintenabilité : Un code maintenable est un code facile à comprendre, modifier et améliorer sans introduire de régressions ou d'erreurs

Adaptabilité

The screenshot shows a restaurant website template. At the top, there's a header with social media icons (Facebook, Twitter, LinkedIn) and a "Make a Reservation" button. Below the header, the main navigation menu includes Home, Theme Features, Menu, Woo Shop, About Us, Blog, Contact Us, and Buy Now. The main content area features a large banner image of a dish being cut with a fork and knife. Below the banner, a "Welcome To Our Restaurant" section includes a short description: "We serve you the best taste in the town at the best price. We are all about taste, healthy and happiness." It also has "View Menu" and "Contact us" buttons. Further down, there are sections for "Today's Special" (a pizza image), "Latest Menu" (a list of desserts like Black Cherry Ice Cream, Creme Brûlée Ice Cream, etc.), and "Words from our Chef!" (an image of a chef). A sidebar on the left contains a "Our Gallery" section with several food images. At the bottom, there's a testimonial from Brain Steens: "Templatic offers world class WordPress theme support and unique, highly innovative and professionally useful WordPress themes. So glad to have found you! All the best and many more years of creativity, productivity and success." A footer bar at the bottom says "When you buy a Templatic theme," followed by the Templatic logo.



Maintenance

Un logiciel ne s'use pas mais se détériore avec le temps:

- Evolution des technologies
- Changements des besoins du client
- Apparition de bugs

La maintenance peut constituer la majeure partie du temps de développement d'un projet. Elle doit être envisagé dès la conception :

- Code bien architecturé
- Code évolutif et robuste
- Utilisation de bonnes méthodologies (Agile, Cycle en V)
- Stratégie de test automatisé
- Utilisation de processus de Devops : CI/CD

Complexité du code

Signe d'un code complexe

- Difficile de modifier le code : un changement a de multiples impacts
- Surcharge cognitive
- « Unknown Unknown » : Difficile de trouver les bonnes informations

Les causes :

- Trop de dépendances entre les classes
- Code obscure
- « inconsistencies »
- Manque de documentation
- Design fragile
- Dette technique

Concepts

Encapsulation

Définition

In software systems, encapsulation refers to the bundling of data with the mechanisms or methods that operate on the data. It may also refer to the limiting of direct access to some of that data, such as an object's components. Essentially, encapsulation prevents external code from being concerned with the internal workings of an object.

Encapsulation

L'encapsulation est l'un des piliers de la POO.

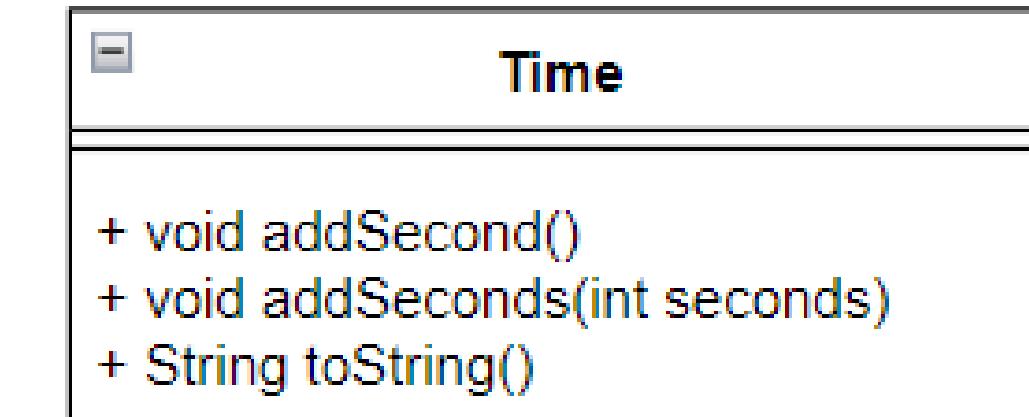
Cela consiste à définir dans une classe ce qui peut être utilisé à l'extérieur de ce qui ne doit pas l'être Le choix des visibilités est primordial.

L'encapsulation est le fait de définir comment doit être utilisé un objet, indépendamment de son fonctionnement interne.

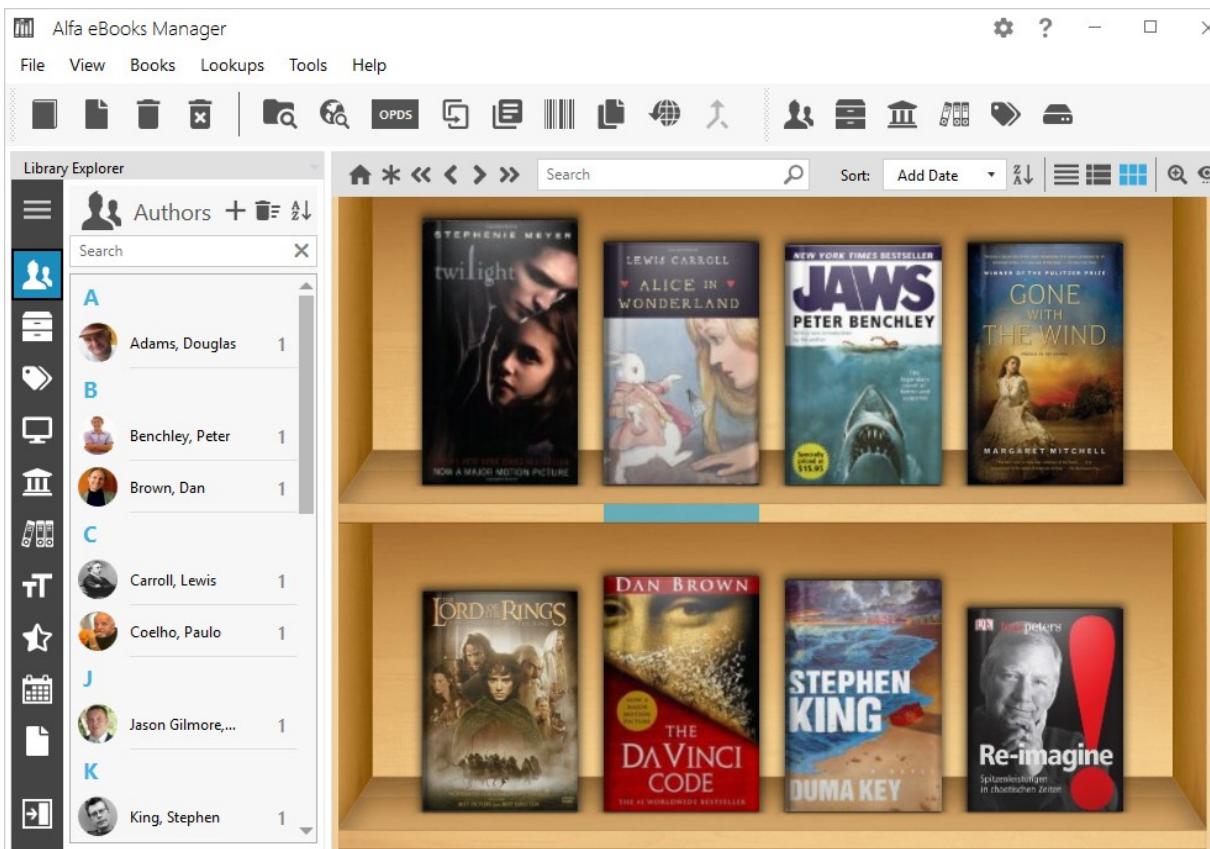
```
public class Time
{
    private int seconds;
    private int minutes;
    private int hours;

    public Time(int hours, int minutes, int seconds)
    {
        this.seconds = seconds;
        this.minutes = minutes;
        this.hours = hours;
    }

    public void addSecond()
    {
        ...
    }
}
```



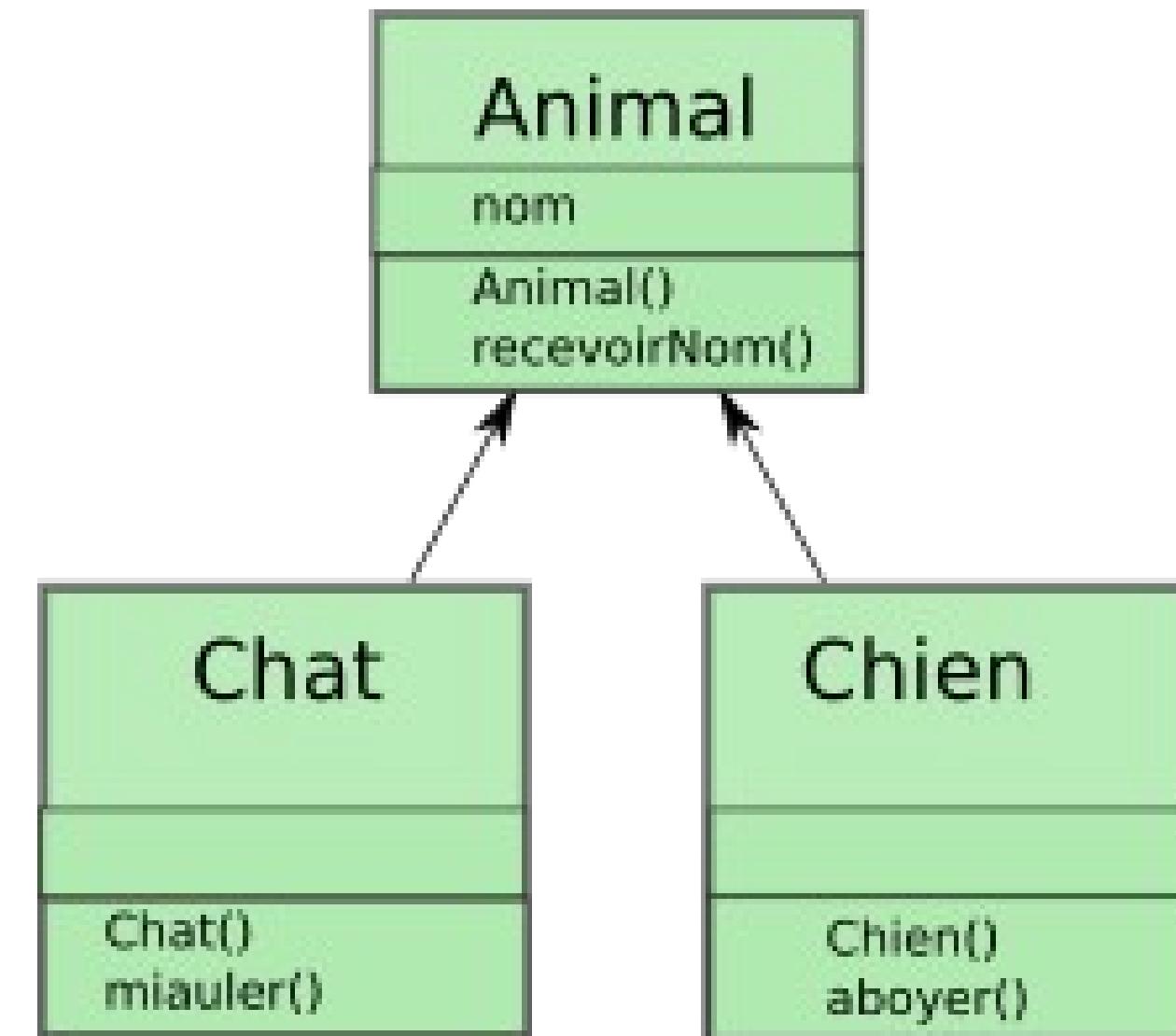
Comment avoir un code réutilisable?



A screenshot of a restaurant WordPress theme named 'RESTAURANTE'. The header features social media links and a 'Make a Reservation' button. The main content area has a large banner image of a dish being cut with a fork. Below the banner are sections for 'Welcome To Our Restaurant', 'Today's Special' (a pizza), 'Latest Menu' (a list of desserts like Black Cherry Ice Cream and Bubblegum), and 'Words from our Chef!' (a photo of a chef). There is also a 'Our Gallery' section with thumbnail images of various dishes. At the bottom, there is a quote from Brain Stoem: 'Templatic offers world class WordPress theme support and unique, highly innovative and professionally useful WordPress themes. So glad to have found you! All the best and many more years of creativity, productivity and success.' - Brain Stoem.

Héritage

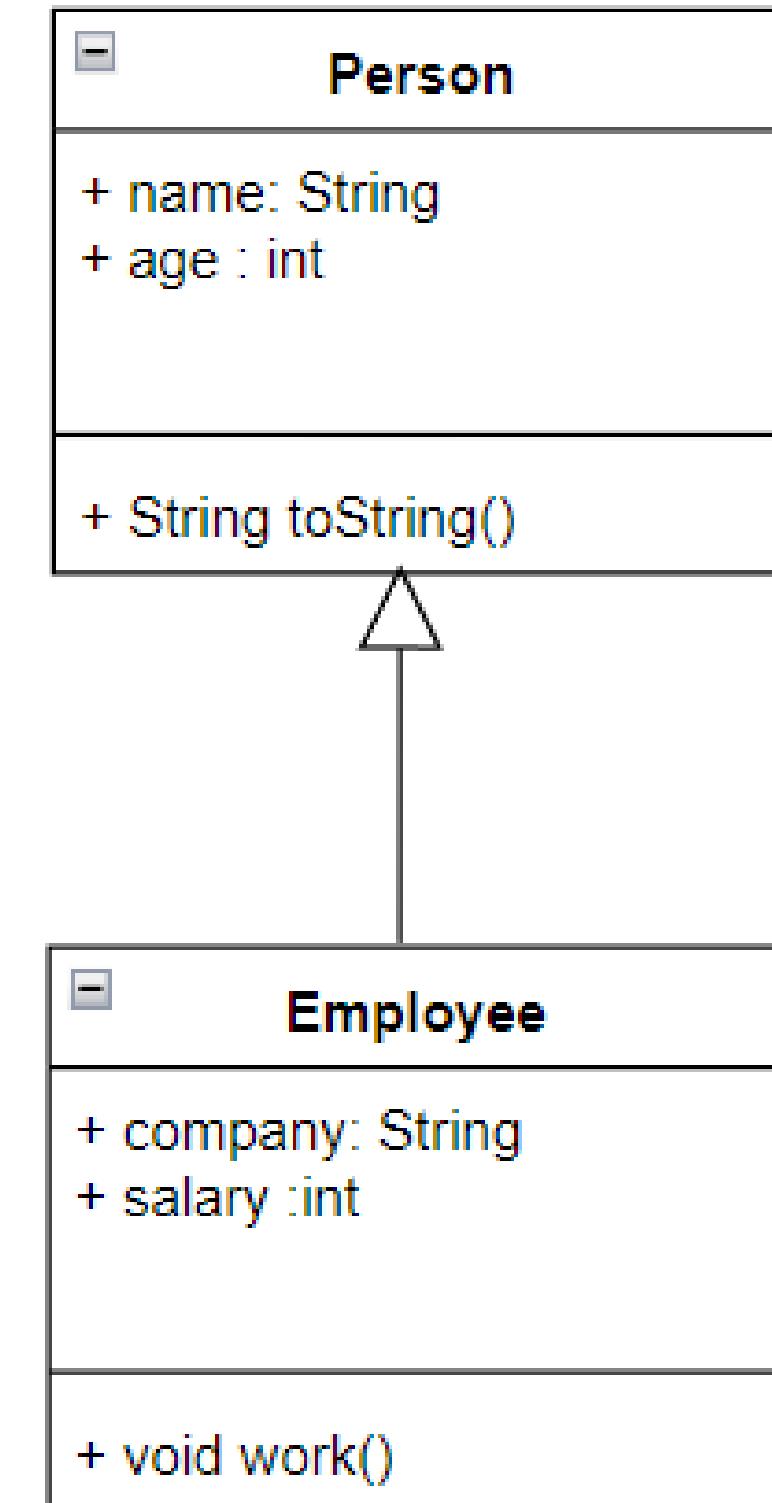
Si certaines classes partagent un ensemble de fonctionnalités, il est possible de les réutiliser du code en servant de l'héritage



Héritage

L'héritage permet de définir une nouvelle classe à partir d'une classe existante.

La classe qui hérite possédera toutes les **methodes** et **propriétés** de la classe parente

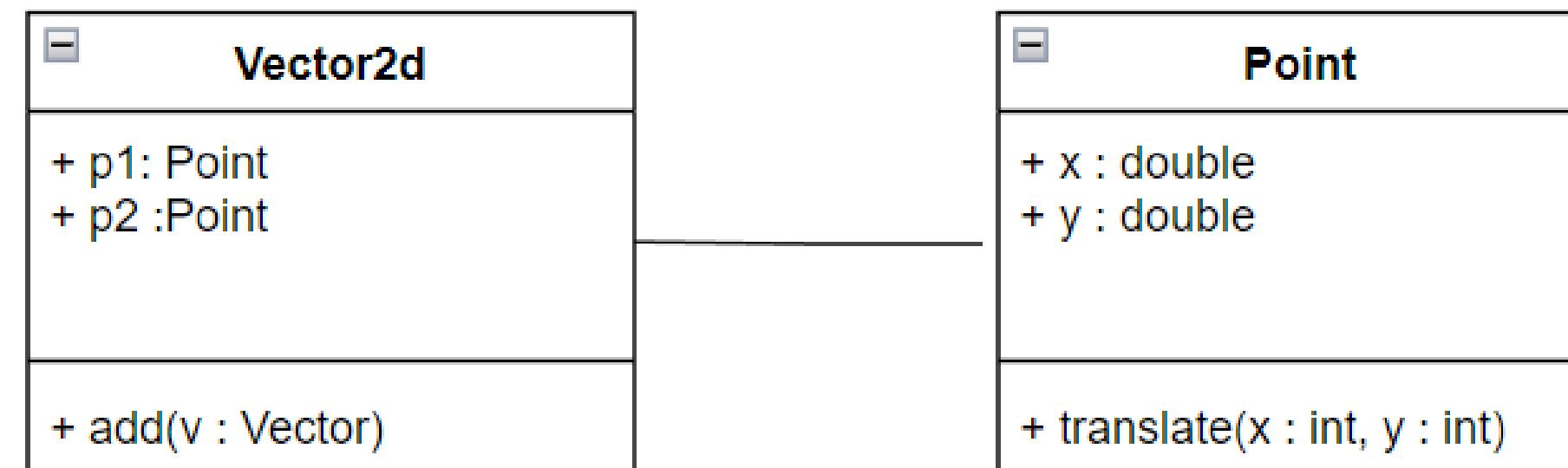


Composition

Définition

On appelle la composition le fait qu'une classe utilise d'autres classes.

Pour cela, une classe doit avoir des **instances** d'une autre classe en **propriété**



Composition

```
public class Vector2D {  
    private Point p1;  
    private Point p2;  
  
    public Vector2D(Point p1, Point p2) {  
        this.p1 = p1;  
        this.p2 = p2;  
    }  
  
    public double length() {  
        double dx = p2.x - p1.x;  
        double dy = p2.y - p1.y;  
        return Math.sqrt(dx * dx + dy * dy);  
    }  
}
```

```
public class Point {  
    public double x;  
    public double y;  
  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Couplage fort/faible

On parle de couplage pour définir l'interconnexion entre 2 classes

Couplage fort : une modification d'une classe nécessite la modification de l'autre

Couplage faible : faible dépendance entre les classes

Un couplage fort entraîne en général un code spaghetti (antipattern) et rend le code difficile à isoler et tester

Comment avoir un code modulaire?

leboncoin

Déposer une annonce

Rechercher sur leboncoin

Mes recherches

Favoris

Messages

Se connecter

Choisir une localisation

Intérim >

Secteur d'acti... >

Fonction >

Filtres 1

Offres d'emploi >

Tri : Pertinence >

Travail en Intérim

15 878 annonces

Conseiller téléphonique /technicien hotline (H/F/X)

Intérim

❖ Candidature simplifiée

Lyon 69007 7e Arrondissement

hier à 21:55 · Nouveau !

 MANPOWER



Géomètre topographe à Valserhône (H/F/X)

Intérim

❖ Candidature simplifiée

Valserhône 01200 Bellegarde-sur-Valserine

hier à 21:55 · Nouveau !

 Sauvegarder la recherche

leboncoin

Déposer une annonce

Rechercher sur leboncoin

Mes recherches

Favoris

Messages

Se connecter

Choisir une localisation

Livraison acc...

Prix

État

Filtres

Ordinateurs

Tri : Pertinence

Annonces Ordinateurs d'occasion (Tours, PC portables ...):
Toute la France

540 483 annonces

Anna ★ 5 (1)

À la une

Urgent

PC Gamer RTX 4070 Ti - Ryzen 7 - 32Go RAM - 2To SSD ✓

1790 € · État neuf

Livraison possible

Bayonne 64100

Laurent ★ 5 (1)

À la une

View order

Ordinateur laptop Azus zenbook 14UX3402ZA-KM020W Oled. SSD 512 Go. RAM 16 Go.

Sauvegarder la recherche

499 € · Très bon état

Comment avoir un code modulaire?

Comment gérer tout les cas ? (sans tout copier coller)

The screenshot shows the Leboncoin homepage with a search bar and various filters. Below the filters, it says 'Annonces Ordinateurs d'occasion (Tours, PC portables ...) : Toute la France' and '540 483 annonces'. A specific listing for a 'PC Gamer RTX 4070 Ti - Ryzen 7 - 32Go RAM - 2To SSD' is highlighted with a red box and a question mark, indicating a modular approach to handling such cases.

Leboncoin

Déposer une annonce

Rechercher sur leboncoin

Choisir une localisation

Livraison acc... >

Prix >

État >

Filtres 1

Ordinateurs > Tri : Pertinence >

Annonces Ordinateurs d'occasion (Tours, PC portables ...) :
Toute la France

540 483 annonces

Anna ★ 5 (1)
A la une Urgent
PC Gamer RTX 4070 Ti - Ryzen 7 - 32Go RAM - 2To SSD ✓
1790 €> · État neuf
Livraison possible
Bayonne 64100

Laurent ★ 5 (1)
A la une
Ordinateur laptop Asus zenbook 14UX3402ZA-KM020W Oled. SSD 512 Go. RAM 16 Go · Sauvegarder la recherche >
499 €> · Très bon état

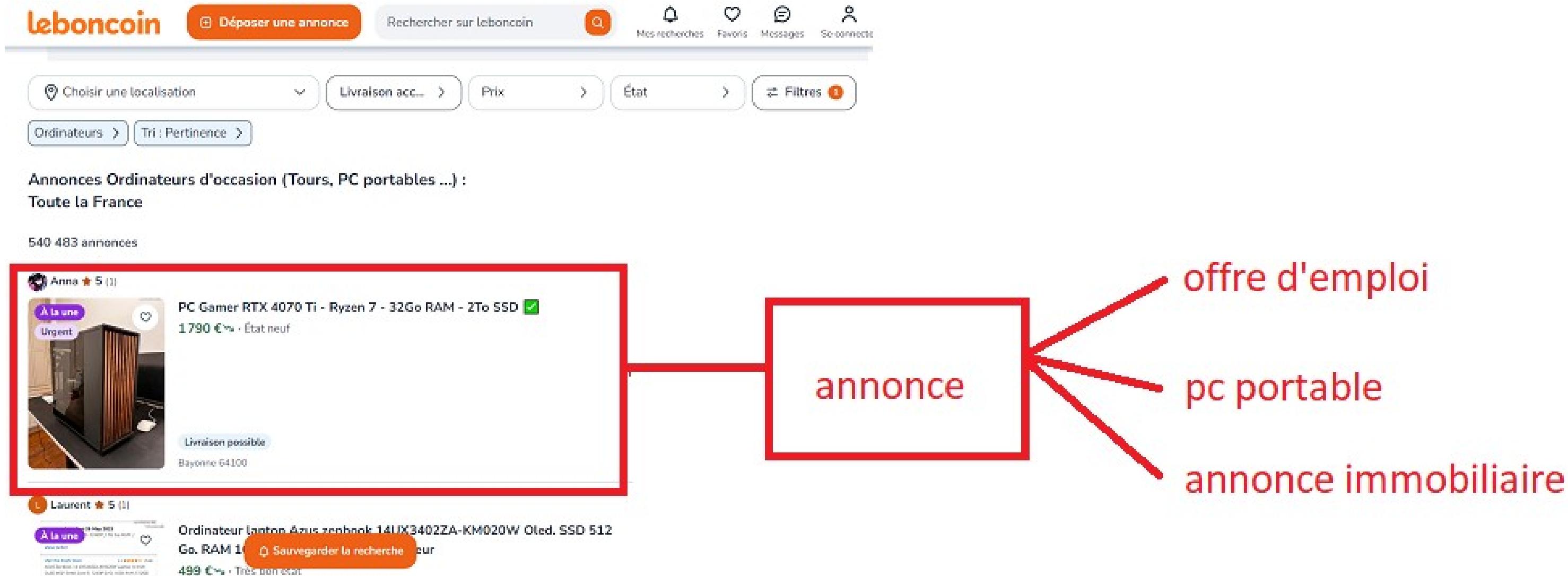
offre d'emploi

pc portable

annonce immobiliare

Abstraction

Faire abstraction des détails et considérer tout comme une annonce

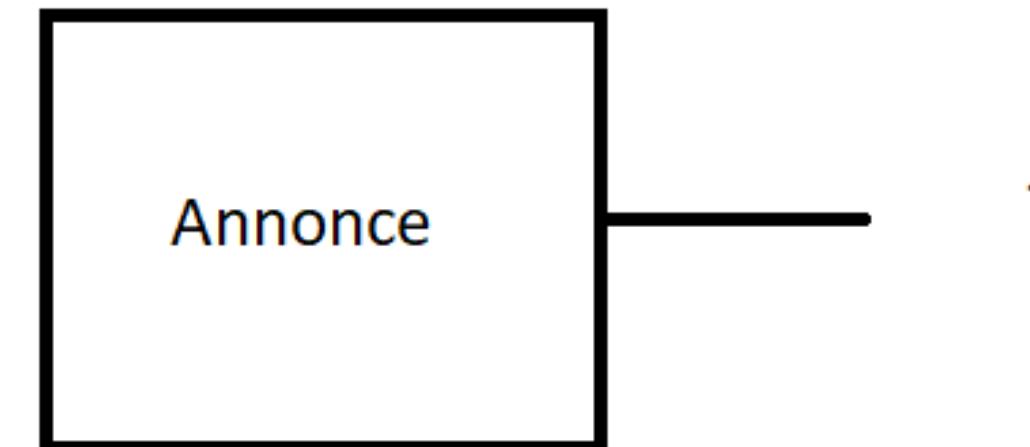


Abstraction

En psychologie, la pensée abstraite, dite aussi capacité d'abstraction, désigne la capacité de l'esprit à créer et utiliser des concepts dans le raisonnement. (Wikipedia)

Conception retirée de son contexte en vue de pratiquer un raisonnement.

Je veux afficher des annonces



Polymorphisme

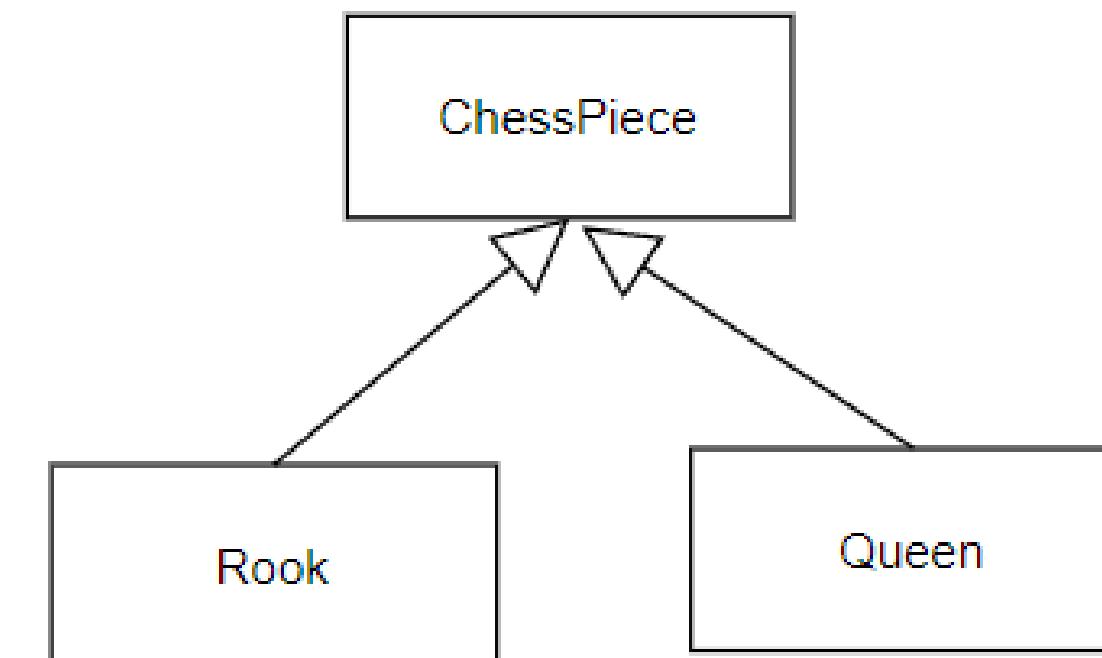
Définition

Polymorphisme vient du grec et signifie qui peut prendre plusieurs formes.

Le polymorphisme est le fait de baser son programme sur des types abstraits et non des classes concrètes. Cela consiste à fournir une interface commune à des classes de différents types

```
ChessPiece p = ?;
```

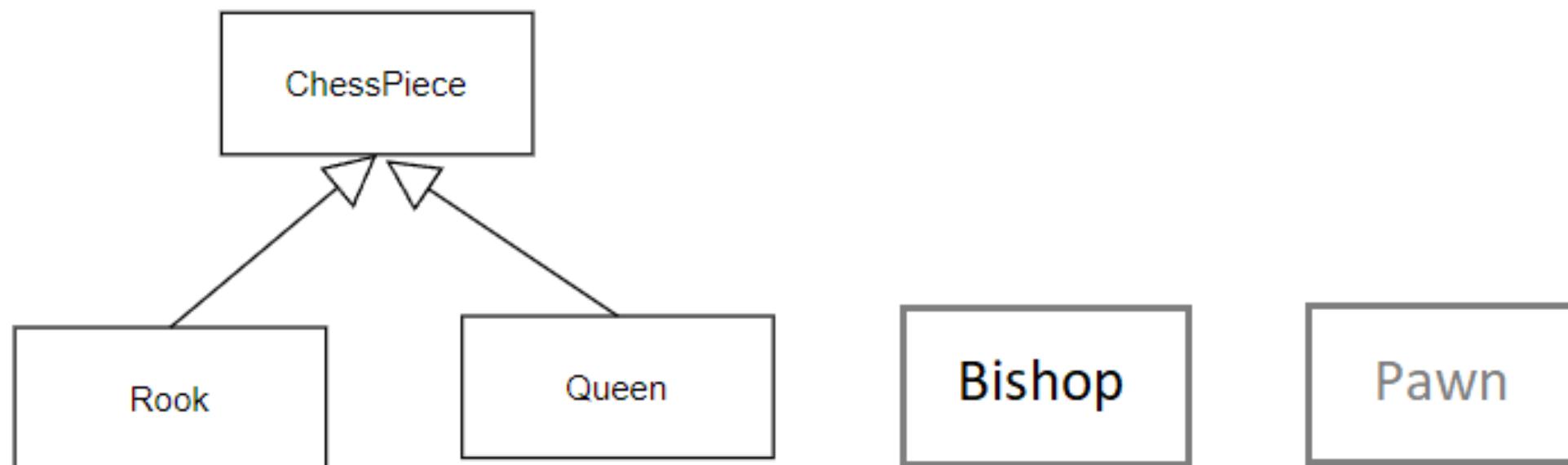
```
p.checkMove(2, 1);
```



Polymorphisme

Ce concept est fondamental en programmation orientée objet. Pour avoir des codes modulaire et flexible, il est recommandé de typer ses variables sur des abstractions comme des interfaces ou des classes abstraites

Cela permet d'ajouter facilement de nouveaux comportement à un programme : Il suffit d'ajouter une nouvelle classe à une famille de classe



Polymorphisme

Se baser sur des abstractions permet de se concentrer sur l'algorithme plutôt que l'implémentation.

Le code devient flexible car il est possible d'interchanger les classes utilisées sans changer l'algorithme

```
while(winner == false) {  
    Scanner in = new Scanner(System.in);  
    int x = in.nextInt();  
    int y = in.nextInt();  
  
    // le vrai type de p ne nous interesse pas et ne change pas l'algo  
    p.checkMove(x,y);  
  
    ...  
}
```

Polymorphisme

program to interfaces not implementations

Design Patterns: Elements of Reusable Object-Oriented Software - Gang of Four (GoF) 1995

Autres bonnes pratiques

- DRY « Don't Repeat Yourself »
- YAGNI « You Aren't Gonna Need It »
- KISS « Keep It Simply Stupid »

Principes SOLID

Principes SOLID

Introduit par Robert C. Martin en 2000 - Design Principles

- **S**ingle Responsability Principe
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependancy Injection Principle

Single Responsibility Principle

The single-responsibility principle (SRP) states that "there should never be more than one reason for a class to change." In other words, every class should have only one responsibility.

Open/Closed Principle

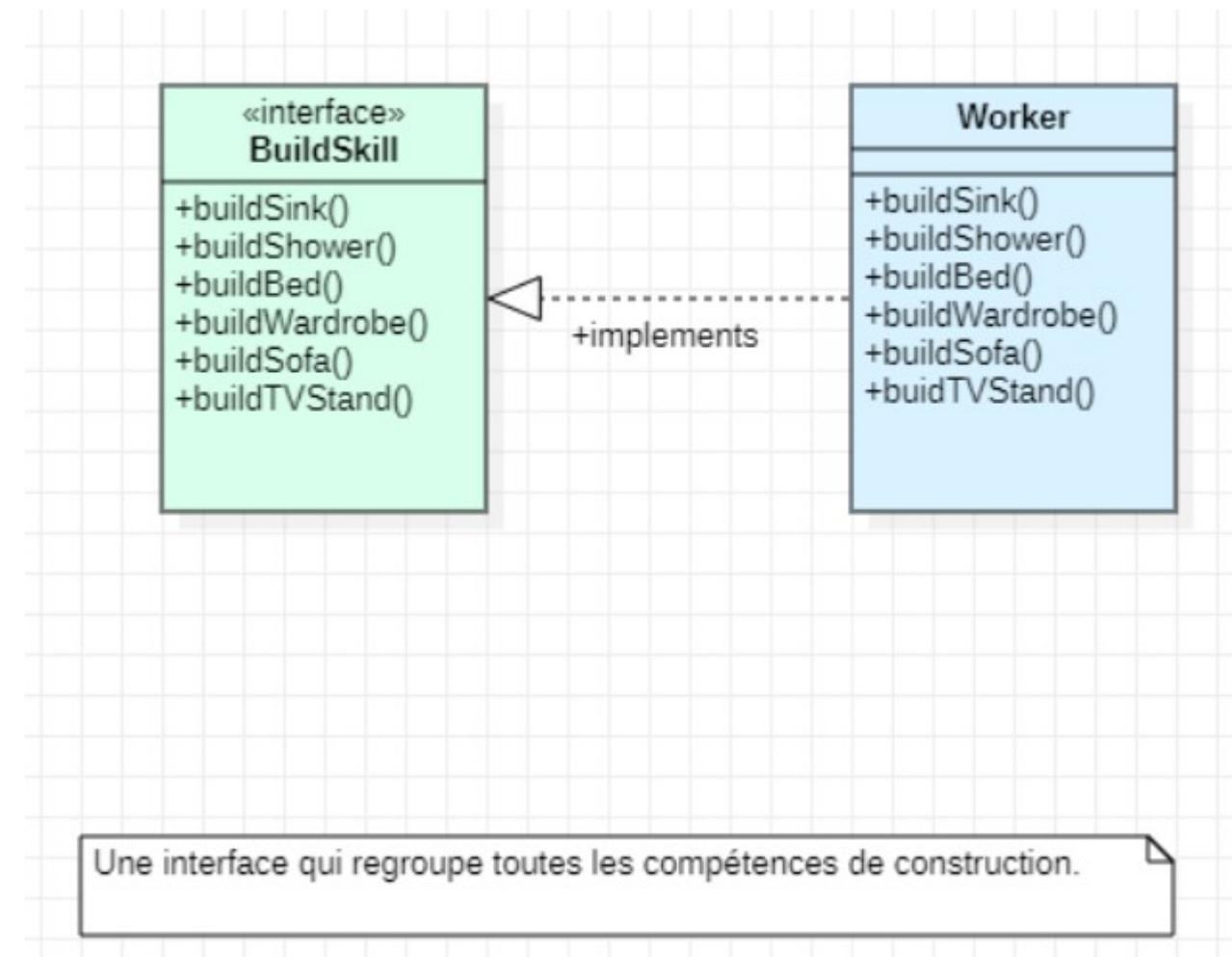
software entities ... should be open for extension, but closed for modification.

Liskov substitution Principle

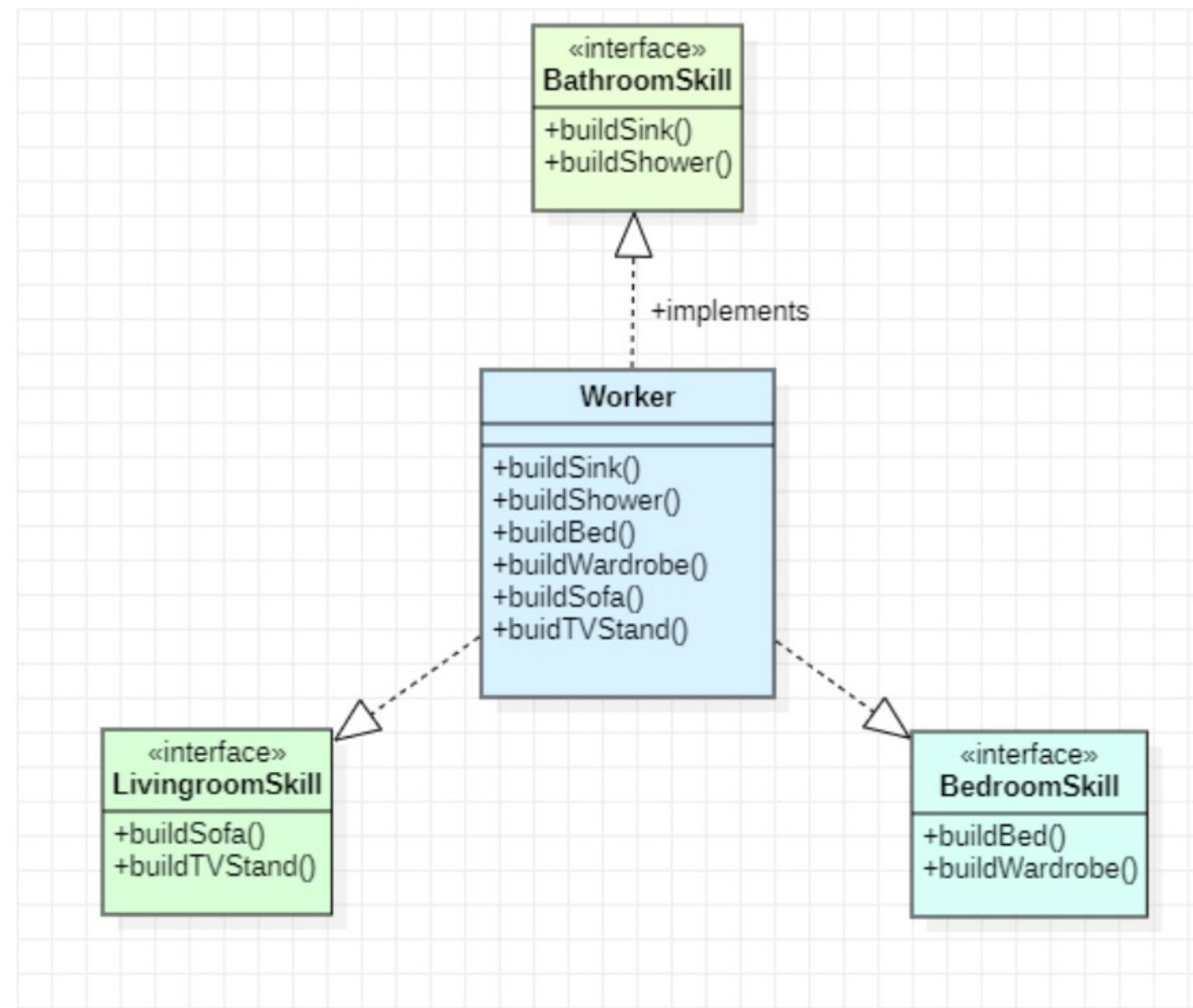
functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

Interface segregation Principle

clients should not be forced to depend upon interfaces that they do not use.



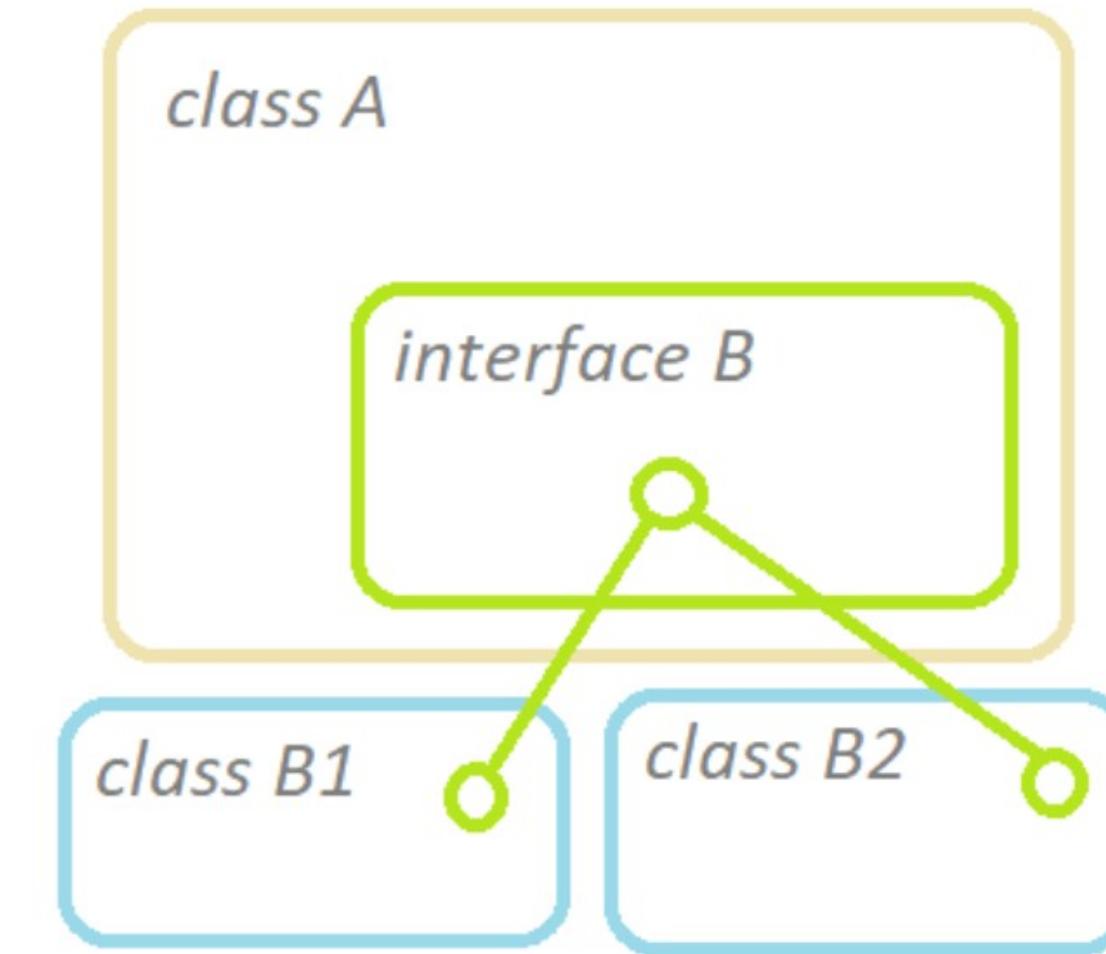
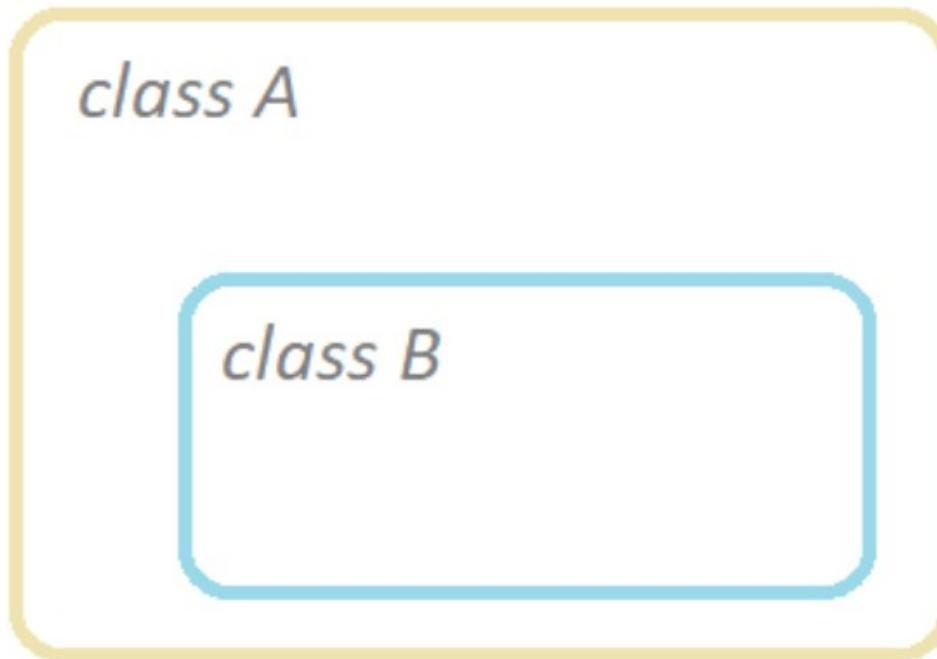
Interface Segregation Principle



Dependency inversion Principle

*High-level modules should not depend on low-level modules.
Both should depend on abstractions*

Inversion de dépendance



Exercices