Du code propre !

Mode d'emploi

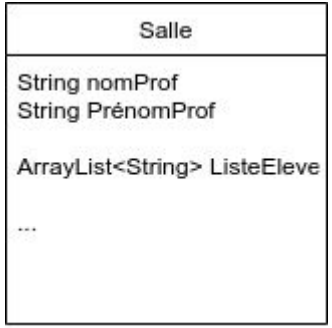
Liste des choses à vérifier

- ☐ Architecture logicielle pertinente/structurée
- ☐ Nom de classes/variables bons
- ☐ Bonne indentation
- ☐ Commentaires présents
- ☐ Conventions respectées
- ☐ Code fragmenté et non dupliqué

Une bonne architecture logicielle ?

Admettons qu'on veuille coder une salle de classe (avec des élèves et un prof). On serait tenté de tout mettre dans une seule classe… Tout serait mélangé, bon courage pour débug ça !

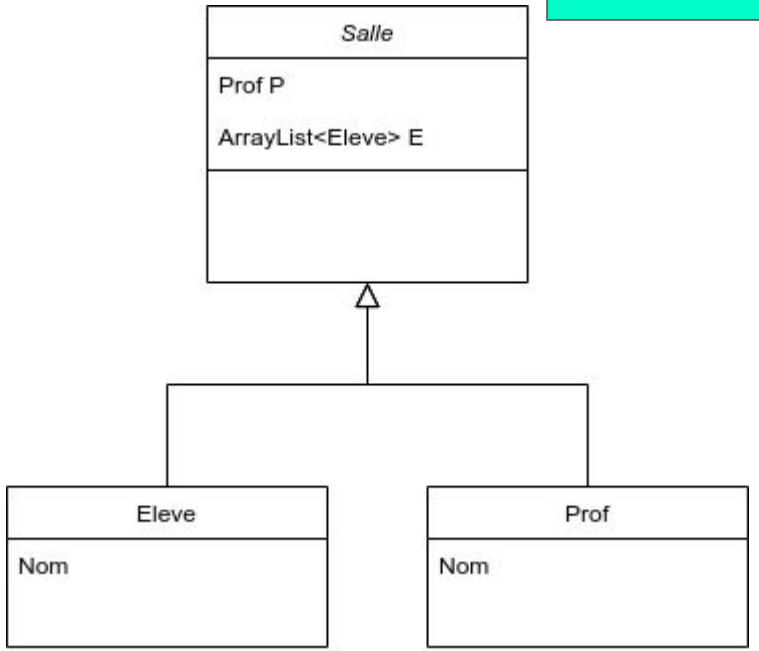
Ouai bof ça…



Une bonne architecture logicielle ?

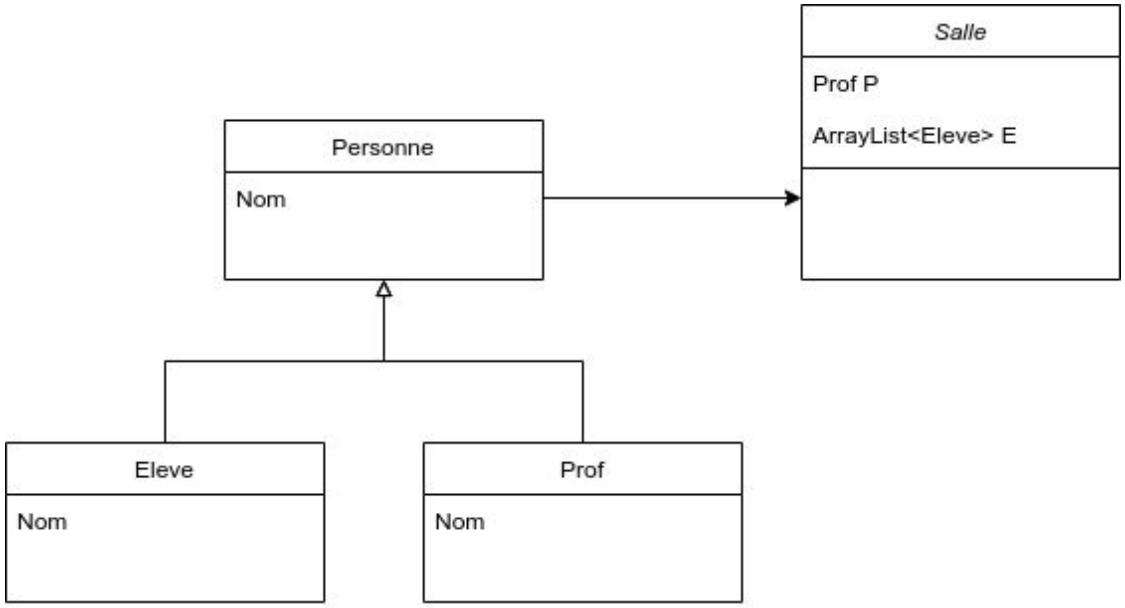
C’est quand même mieux de segmenter “prof”, “élève” et “salle”, en créant une classe “élève”, une classe “prof”, et une classe “salle” qui contient un prof + X élèves.

Ca je valide ! ❤



Une bonne architecture logicielle ?

Et on pourrait aller plus loin et se dire que “Eleve” et “Prof” héritent tous les deux d’une classe “Personne”, ainsi un élève serait considéré comme une “Personne” qui a des attributs d’élèves !



Encore mieux !

Une bonne architecture logicielle ?

Une bonne architecture dépend beaucoup du contexte, l'idée est d'avoir une structure de code tel que :

- Si je rajoute une nouvelle fonctionnalité, ce sera facile.
- Si un bug survient, je sais très vite où est le bug.
- Si je travaille en équipe, je peux répartir le travail facilement.

Dans tous les cas :

Une bonne architecture logicielle, ça se **prépare** ! Ne vous lancez pas tête baissée dans le codage, prenez le temps de dessiner l'architecture sur un bout de papier ! Vaut mieux prendre du temps à réfléchir au plan, plutôt que de prendre du temps à débug un bordel parce-qu'on a pas assez réfléchit ;)

Liste des choses à vérifier

- ☐ Architecture logicielle pertinente/structurée
- ☐ Nom de classes/variables bons
- ☐ Bonne indentation
- ☐ Commentaires présents
- ☐ Conventions respectées
- ☐ Code fragmenté et non dupliqué

Des bons noms de variable/Classe

```
public String A(int [][] M, int r, int x):
this.r = r;
this.ttt = x;

string p =””;
For (for n=0, n<r, n++):
   For (for h=0, h<ttt, h++) :
      if this.H[n][h]==M[n][h] :
        p +=  M[n][h];
      else :
        p += “x”;
       p += “\n”;
return p;
```

Que fait cette méthode ? :)

Des bons noms de variable/Classe

C'est mieux avec des noms clairs, non ? ;)

```
public String A(int [][] M, int r, int x):
this.r = r;
this.ttt = x;

string p =””;
For (for n=0, n<r, n++):
   For (for h=0, h<ttt, h++) :
      if this.H[n][h]==M[n][h] :
        p +=  M[n][h];
      else :
        p += “x”;
       p += “\n”;
return p;
```

pitié faites pas ça.

```
public String DifferenceMatrice(int [][] Matrice,
int longueur, int largeur):
this.longueur = longueur;
this.largeur = largeur;

String str =””;
For (for i=0, i<longueur, i++):
   For (for j=0, j<largeur, j++) :
      if this.Matrice[i][j] == Matrice[i][j] :
        str +=  Matrice[i][j];
      else :
        str += “x”;
       str += “\n”
return str;
```

Liste des choses à vérifier

- ☐ Architecture logicielle pertinente/structurée
- ☐ Nom de classes/variables bons
- ☐ Bonne indentation
- ☐ Commentaires présents
- ☐ Conventions respectées
- ☐ Code fragmenté et non dupliqué

Bonne Indentation

```
public String A(int [][] M, int r, int x):
this.r = r;
      this.ttt = x;

string p =””;
      For (for n=0, n<r, n++):
For (for h=0, h<ttt, h++) :
if this.H[n][h]==M[n][h] :
p +=  M[n][h];
      else :
p += “x”;
            p += “\n”;
return p;
```

Ça marche de faire ça, c’est pas python, mais là aussi… pitié :(

J’ai vraiment besoin d’expliquer pourquoi c’est bien d’indenter correctement ?

Bonne Indentation

A savoir : Netbeans peut vous mettre votre code au propre !

Menu > “Source” > “Format”

(ou le raccourci Alt+Maj+F)

Liste des choses à vérifier

- ☐ Architecture logicielle pertinente/structurée
- ☐ Nom de classes/variables bons
- ☐ Bonne indentation
- ☐ Commentaires présents
- ☐ Conventions respectées
- ☐ Code fragmenté et non dupliqué

Commentaires présents

Archi important, si vous devez garder un truc à faire, ça serait celui-ci !

Rajouter des commentaires (utiles) permet de :

- Permettre à vos camarades de comprendre ce qui a été fait
- Permettre au prof de vous rajouter des points parce-qu'il a compris ce que vous avez fait
- Vous permettre à **vous** de vous rappeler de ce que vous avez fait !

Des commentaires 2.0 : JavaDoc

La Javadoc n’est ni plus ni moins qu’un commentaire… Mais bien formaté, si bien qu’il peut être transformé en manuel !

Il se présente sous forme de commentaire au dessus de chaque méthode, avec des tags comme **@param** (pour expliquer les paramètres de la méthode) ou **@return** (pour expliquer ce que la méthode retourne)

Cf la fabuleuse page wiki :
https://fr.wikipedia.org/wiki/Javadoc

Liste des choses à vérifier

- [x] Architecture logicielle pertinente/structurée
- [x] Nom de classes/variables bons
- [x] Bonne indentation
- [x] Commentaires présents
- [ ] Conventions respectées
- [ ] Code fragmenté et non dupliqué

Les conventions

Les conventions de langage sont plus ou moins les habitudes des informaticiens en ce qui concerne les noms de variables/classes. Si elles ne sont pas respectés, ça ne buggera pas, mais on perd en lisibilité…

Vous les connaissez sans doute déjà :

- “i” et “j” sont des noms de variables plutôt réservés aux boucles for.
- “x” et “y” font plutôt référence à des coordonnées.
- Les noms de variables sont toujours en minuscules, les noms de classes commencent par une majuscule
- etc…

La charte des conventions en Java :
https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html

Les conventions

Dans les conventions, interviennent aussi la position de certaines méthodes dans la classe. En général, on suit l'ordre suivant :

```
package truc;

import xxx; //Tous les imports en haut

public class Machin {

//Les attributs en premier
private int a;
private int b;

//Puis le/les constructeurs
public Machin(){...}

//Puis les getter/setter
public void setA(int a){...}
public int getA(){...}
…

//Puis les méthodes…
public void truc(){...}
…

//En dernier, les méthodes overridés comme equals, toString etc…
@Override
public String toString(){...}
…

}
```

Liste des choses à vérifier

- ☐ Architecture logicielle pertinente/structurée
- ☐ Nom de classes/variables bons
- ☐ Bonne indentation
- ☐ Commentaires présents
- ☐ Conventions respectées
- ☐ Code fragmenté et non dupliqué

Duplication de code

```
if(a == 1){
      filename = “test_numero_”+1+“.txt”;
      //écriture dans un fichier de nom
      “test_numero_1.txt
      try {
      FileWriter fichier = new FileWriter
      (filename);
      fichier . write ( " Hello world ! " );
      fichier . close () ;
      } catch ( IOException e) {
      e. printStackTrace () ;
      }
}
if(a == 2){
      filename = “test_numero_”+2+“.txt”;
      //écriture dans un fichier de nom
      “test_numero_2.txt
      try {
      FileWriter fichier = new FileWriter
      (filename);
      fichier . write ( " Hello world ! " );
      fichier . close () ;
      } catch ( IOException e) {
      e. printStackTrace () ;
      }
}
etc…
```

Dans ce bout de code, on remarque assez vite que le contenu du 1er “if” est quasi le **même** que dans le 2ème “if”. La seule chose qui change, c’est :

```
filename = “test_numero_”+x+“.txt”;
```

On peut éviter cette duplication inutile de code ! Ça permet d’éviter de corriger de partout la même erreur qu’on trouverait sur un des “if” (par exemple, admettons qu’on veuille plus écrire “Hello world !” mais autre chose… On devra le changer autant de fois qu’il y aura de if… Pas cool ça.)

Duplication de code

```
public void ecrireDansFichier(int a){

      filename = “test_numero_”+a+“.txt”;
      //écriture dans un fichier de nom
      “test_numero_a.txt
      try {
      FileWriter fichier = new FileWriter
      (filename);
      fichier . write ( " Hello world ! " );
      fichier . close () ;
      } catch ( IOException e) {
      e. printStackTrace () ;
      }

}

-----------------------------------------------

if(a == 1){
      ecrireDansFichier(1);
}
if(a == 2){
      ecrireDansFichier(2);
}
if(a == 3){
      ecrireDansFichier(3);
}
etc…
```

C’est mieux non ? :)

Si on veut changer quelque chose dans “ecrireDansFichier”, la modification se répercute sur tous les “if” automatiquement. C’est pratique pour corriger quelque chose rapidement !

D’où l’importance **fragmenter une méthode longue en plusieurs sous-méthodes** !

Fragmentation du code

```
public void main(){
	//initialisation
	truc1();
	truc2();
	truc3();

	//sauvegarde
	truc4();
	truc5();
	truc6();

	//Calcul
	truc7();
	truc8();
	truc9();

	//etc…
	…
}
```

Là encore, imaginons qu'on a une méthode **TRES longue** et qui fait **BEAUCOUP de choses**. Plutôt que de tout mettre en brut dans une seule méthode, il peut être intéressant de créer des sous-méthodes pour fragmenter celle-ci.

Fragmentation du code

```
public void initialisation() {
        truc1();
        truc2();
        truc3();
}

public void sauvegarde() {
        truc3();
        truc4();
        truc5();
}
public void calcul() {
        truc6();
        truc7();
        truc8();
}


-------------------------------------

public void main(){
        initialisation();
        sauvegarde();
        calcul();

}
```

L'avantage ici est que les méthodes peuvent être à présent réutilisables dans un autre contexte, et que si un bug survient, il est plus facile de le détecter !

A retenir : Si une méthode est trop longue, il faut la découper en sous-méthodes !

PS : Cette logique s'applique aussi pour les classes ! Si une classe a trop d'attributs, peut-être qu'il est temps de la subdiviser en plusieurs sous-classes !

Pour aller plus loin

Le “Clean code” est un art, et pleins de blog de codage en parlent :)

En voici un qui explique bien les grandes lignes !

https://www.invivoo.com/lart-clean-code-environnement-java/

Bon Nettoyage !