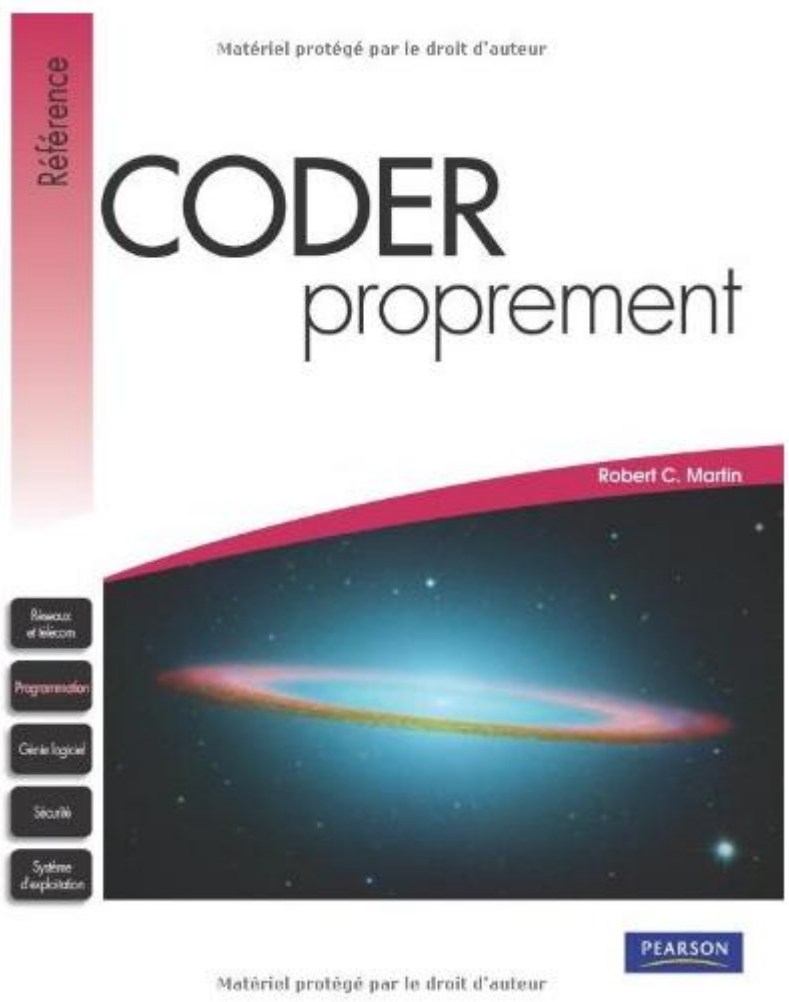
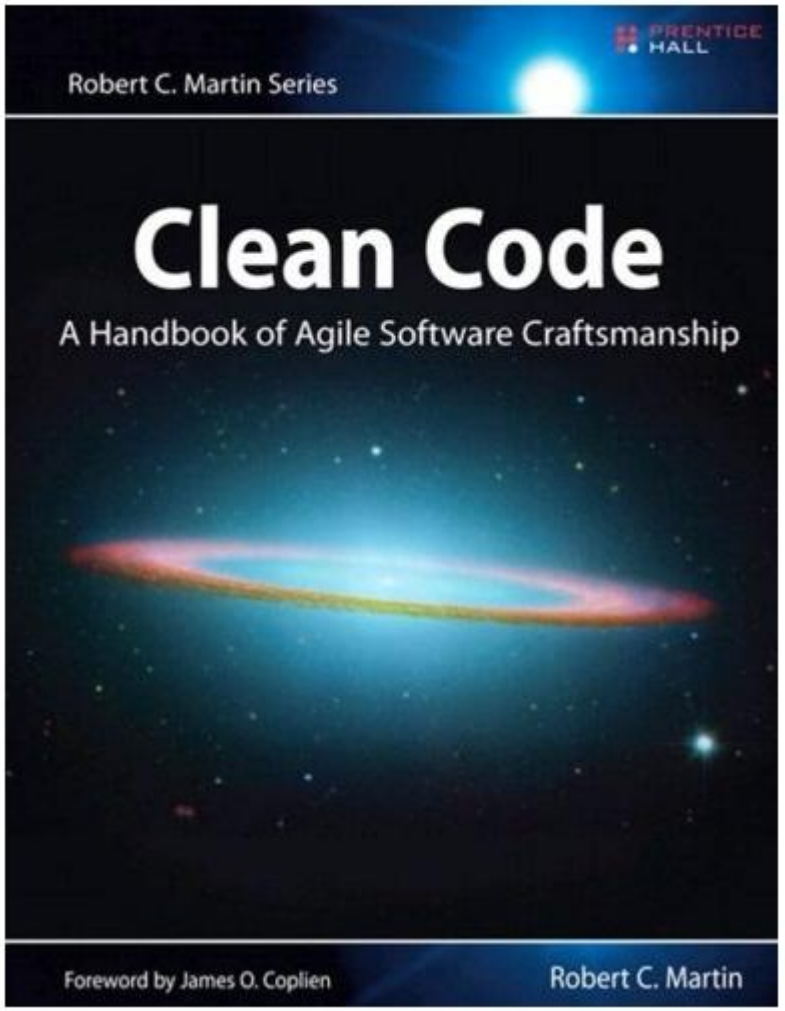
Diginamic
Formation

« Dynamisez votre carrière digitale ! »

Bonnes pratiques.
Écrivez proprement !!

Ouvrages de référence



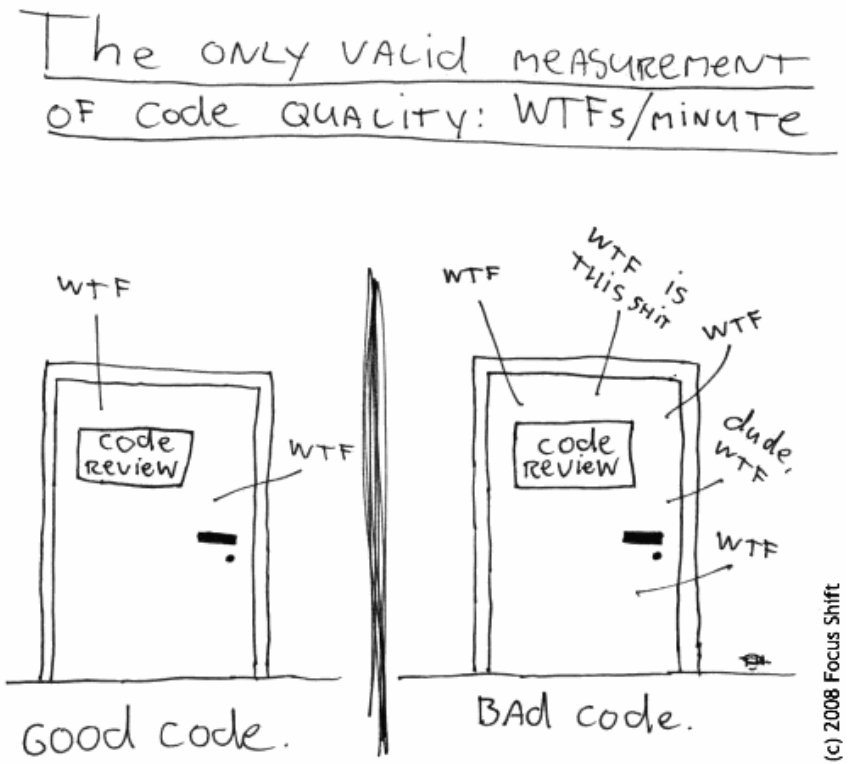


Principes généraux

- Un développeur est un auteur
  - Ecrire est sa profession
  - Un auteur écrit pour être lu

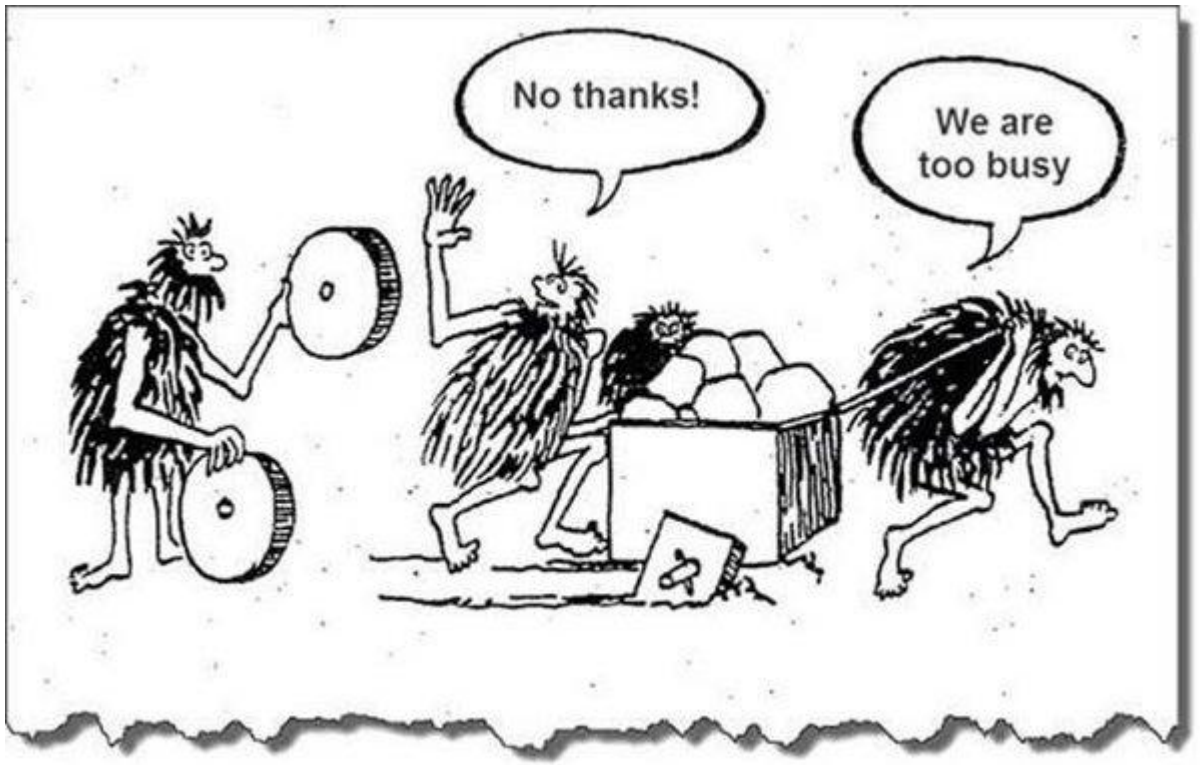
Principes généraux

- Du code propre est le préalable au code robuste
- Produire moins mais mieux.



Principes généraux

- La « dette technique » : le passif à combler, qui se rembourse avec les intérêts, lors des maintenances correctives ou évolutives.



Principes généraux

- Suivre les conventions standards.

```java
/** Classe de stockage de variables avec leurs valeurs
 * @author DIGINAMIC
 */
public class Params {

	/** Map de variables */
	private HashMap<String, ParameterValue> variables;

	/** Nombre maximum de valeurs */
	public static final int MAX_VALUE=50;

	/** Constructeur */
	public Params(){
		variables = new HashMap<>();
	}

	/** Permet d'ajouter un paramètre
	 * @param variable paramètre
	 */
	public void addVariable(ParameterValue variable){
		variables.put(variable.getName(), variable);
	}
}
```

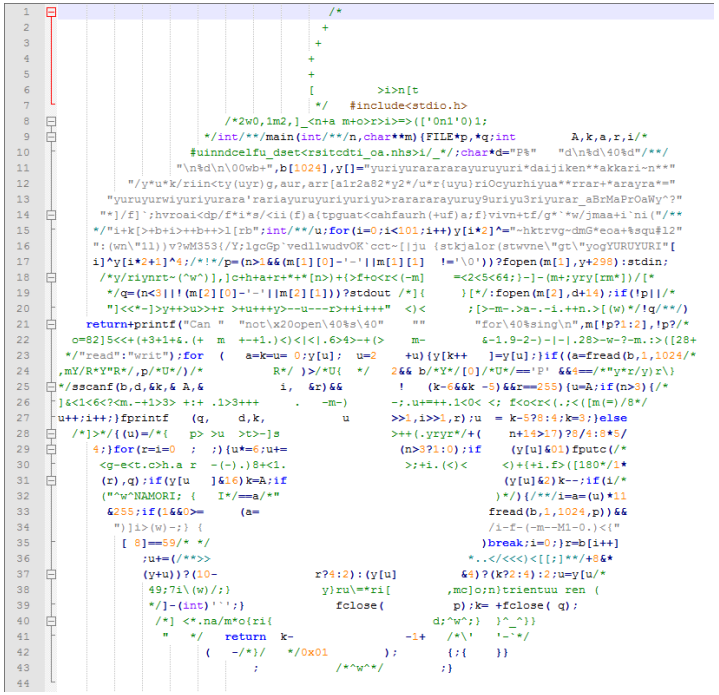
Structure du code

- KISS: **K**eep **I**t **S**imple And **S**tupid

```
 7  <!-- InstanceEndEditable -->
 8  <link rel="stylesheet" type="text/css" href="stylesheet.css"/>
 9  <link href='http://fonts.googl                    obster' rel='styl
10  <!-- InstanceBeginEditable nam
11  <script type="text/javascript"
12  <!--
13  function MM_validateForm() { //v4.0
14    if (document.getElementById){
15      var i,p,q,nm,test,num,min,max,errors='',args=MM_validateForm.argu
16      for (i=0; i<(args.length-2); i+=3) { test=args[i+2]; val=document
17        if (val) { nm=val.name; if ((val=val.value)!="") {
18          if (test.indexOf('isEmail')!=-1) { p=val.indexOf('@');
19            if (p<1 || p==(val.length-1)) errors+='- '+nm+' must contai
20          } else if (test!='R') { num = parseFloat(val);
21            if (isNaN(val)) errors+='- '+nm+' must contain a number.\n'
22            if (test.indexOf('inRange') != -1) { p=test.indexOf(':');
23              min=test.substring(8,p); max=test.substring(p+1);
24              if (num<min || max<num) errors+='- '+nm+' must contain a
25        } } } else if (test.charAt(0) == 'R') errors += '- '+nm+' is re
26      } if (errors) alert('The following error(s) occurred:\n'+errors);
27      document.MM_returnValue = (errors == '');
28  } }
29  //-->
30  </script>
31  <!-- InstanceEndEditable -->
32  </head>
33  <body>
```

The code added just to validate one field!

Structure du code



- Lignes courtes.
- Code indenté.
- Les méthodes similaires sont regroupées.

Règles sur les méthodes

- Une méthode doit-être courte (50 lignes max).
- Une méthode ne doit faire qu'une seule chose.
- Utilisez un nommage descriptif (verbe à l'infinitif).
- Limitez le nombre d'arguments.

Compréhension du code

- Choisir des noms descriptifs et non ambigus.
- Choisir des noms prononçables.
- Choisir des noms que l'on peut chercher facilement.
- Remplacer les nombres "magiques" par des constantes.
- Eviter les préfixes ou les informations de type dans les noms de variables.

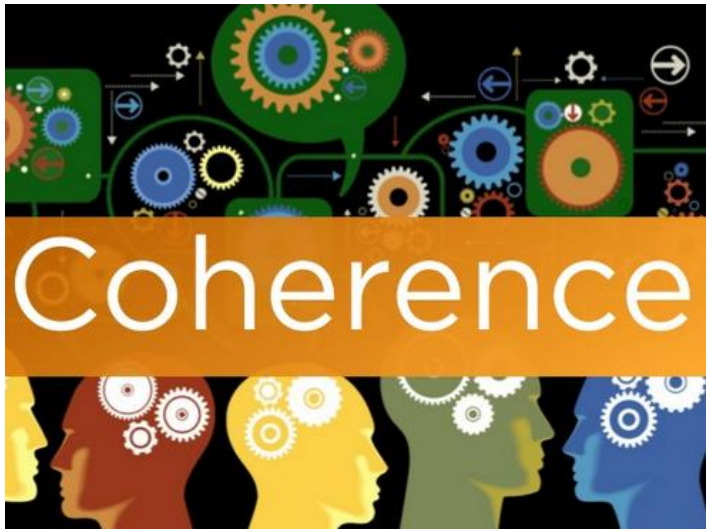
Compréhension du code

- Evitez les conditions négatives.

Compréhension du code

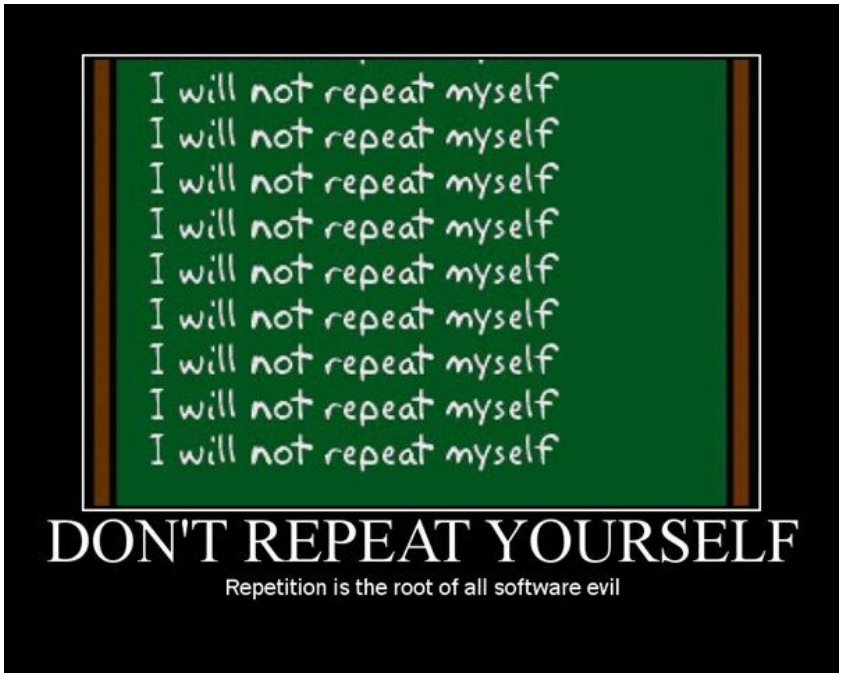
- Soyez **cohérent**. Si vous faites quelque chose d'une certaine manière, effectuez toutes les choses similaires de la même manière.

- Utilisez des variables avec des noms explicites.



Règles sur les méthodes

- Principe **DRY**: Don’t Repeat Yourself
  - Evitez les redondances (duplication de code).
  - Chaque répétition de code représente une abstraction manquée.



Règles sur les méthodes

- ❑ Ne commentez pas du code, supprimer le.
- ❑ Utiliser les commentaires pour :
  - Expliquer.
  - Clarifier.
  - Communiquer sur les points d'attention, des cas aux limites connus.

Ce qui n'est pas documenté n'existe pas !

Principes généraux

- **Règle du Boy Scout** : lorsqu'un fichier source est ouvert, vous devez le laisser plus propre que l'état dans lequel vous l'avez trouvé.

- **Toujours chercher l'origine des problèmes :**
  - Corrigez toujours les bugs, même s'ils sont difficilement reproductibles.
  - Soyez rigoureux. Faites-vous violence sur ce sujet.

The boy scout rule

"Always leave the code you're editing a little better than you found it"

- Robert C. Martin (Uncle Bob)

Règles de conception

- Rendre les données configurables au plus haut niveau
  - Les paramètres de connexion à une base de données doivent être externalisés.

- Le polymorphisme doit remplacer les structures if/else, switch/case.

```
File  Edit  View  Navigate  Code  Analyze  Refactor  Build  Run  Tools  VCS  TeamCity  Window  Help
79                  }
80
81                  if ((i + 1) < pathElements.length) {
82                      if (!fieldIsArrayOperator) {
83                          //catch people trying to search/update into @Reference/@Serialized fields
84                          if (validateNames &&
85                              (mf.get().isReference() || mf.get().hasAnnotation(Serialized.class))) {
86                              throw cannotQueryPastFieldException(propertyPath, fieldName, validatedF
87                          }
88
89                          if (!mf.isPresent() && validatedField.mappedClass.isInterface()) {
90                              break;
91                          } else if (!mf.isPresent()) {
92                              throw fieldNotFoundException(propertyPath, validatedField);
93                          }
94                          //get the next MappedClass for the next field validation
95                          MappedField mappedField = mf.get();
96                          validatedField.mappedClass = mapper
97                                  .getMappedClass((mappedField.isSingleValue()) ? mappedField
98                                          .getType() : mappedField.getSubClass());
99                      }
100                 } else {
101                     break;
102                 }
103             }
104
105             //record new property string
```

Règles de conception

- Respecter les niveaux d'abstraction
  - On ne met pas des **attributs spécialisés** (i.e. qui concernent une classe fille) dans la **classe mère** même si c'est plus commode lors du développement

Exemple dit du « contexte flou »

```
public void afficherAdresse(int numeroRue, String rue, int codePostal, String ville){
                ….
}
```

- La fonction affiche une adresse.
- Le concept "Adresse" n'est pas représenté par une classe:

on parle de contexte flou

- **Et aussi…préférez utiliser des types dédiés plutôt que des types primitifs.**

Question 1

- Qu'est-ce qui ne va pas dans cette classe ?

```
package ex1;

public class calcul {

        public int get1(int a, int b){
                return a+b;
        }

        public int get2(int a, int b){
                return a-b;
        }
}
```

Réponse 1

- Le nom de la classe ne commence pas par une MAJ
- Les noms de méthodes ne sont pas explicites
- Il n'y a pas la javadoc

```
package ex1;

public class calcul {

	public int get1(int a, int b){
		return a+b;
	}

	public int get2(int a, int b){
		return a-b;
	}
}
```

Classe corrigée 1

```java
/** Classe qui propose des opérations sur les nombres
 *
 * @author Untel
 */
public class Calcul {

    /** Retourne l'addition de 2 entiers
     * @param a entier 1
     * @param b entier 2
     * @return int
     */
    public int addition(int a, int b){
        return a+b;
    }

    /** Retourne la soustraction de 2 entiers
     * @param a entier 1
     * @param b entier 2
     * @return int
     */
    public int soustraction(int a, int b){
        return a-b;
    }
}
```

Question 2

- Qu'est-ce qui ne va pas dans cette classe ?

```
public class entreprise {

        public int Siret;
        public String Nom;
        public String adresse;
        public Date date_Creation;

        public static final int capitalMax = 3000000;

        public void Afficher_statut(){

        }

}
```

Réponse 2

- Les conventions ne sont pas respectées: le nom de la classe, des variables, constantes et méthodes sont à revoir.
- Les règles de l'encapsulation ne sont pas respectées
- Il doit y avoir de la javadoc

```
public class entreprise {

        public int Siret;
        public String Nom;
        public String adresse;
        public Date date_Creation;

        public static final int capitalMax = 3000000;

        public void Afficher_statut(){

        }

}
```

Classe corrigée 3 (sans javadoc)

```
public class Entreprise {

    private int siret;
    private String nom;
    private String adresse;
    private Date dateCreation;

    public static final int CAPITAL_MAX = 3000000;

    public void afficherStatut(){

    }

    // + getters et setters
}
```

TP n°1: Nettoyage

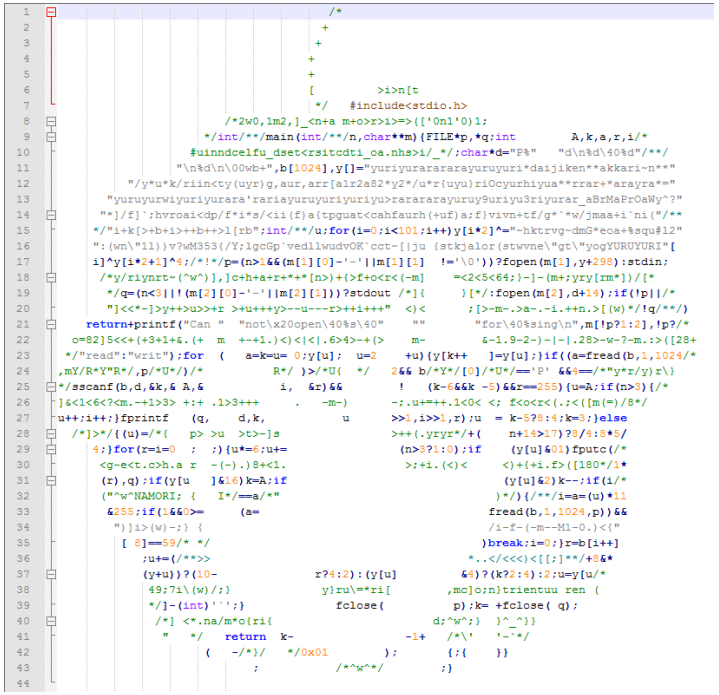
ANNEXES

[Annexes]

Préambule

Les règles qui suivent dans cette annexe sont sujettes à débat et ne sont pas forcément appliquées en entreprise.

Structure du code



- ❑ Déclarez les variables aussi prêt que possible de leurs utilisations.

Règles sur les méthodes

- Evitez les effets de bords.
- Ne pas utiliser un argument "flag", préférer créer 2 méthodes différentes.

La loi de Déméter

- Suivre la loi de Déméter (principe de connaissance minimale) :
  - Une classe ne doit connaître que ses dépendances directes.
  - Faire des façades aux classes de plus bas niveau