Règles de nommage

- Ne pas avoir peur de faire des noms longs
  - Un nom long explicite est meilleur que :
    - un nom court énigmatique
    - un commentaire
  - P. ex. l'identificateur suivant est un nom correcte : `initialiseTableauCandidatsAvecNombresPremiersConnus()`

Règles de nommage

- Passer du temps pour le choix des noms
  - Il faut essayer différents noms et vérifier leur pertinence en contexte
  - Les IDE modernes rendent le changement de nom trivial

Conventions de nommage

- Rendre les mots composés lisibles en adoptant une convention
- Chaque langage définit ses propres conventions qu'il est important de respecter
- Rappel en Java :
  - Classes : PascalCase
  - Méthodes et attributs : camelCase
  - Constantes : SCREAMING_SNAKE_CASE
  - Paquets : snake_case

Cas particulier : nommage des attributs

- Bug classique (détectable par les analyseurs de code)

```
public class Bottle {
    private int volume;
    public Bottle( int volume) {
        volume = volume;
    }
}
```

```
public class Bottle {
    private int volume;
    public Bottle( int volume) {
        this.volume = volume;
    }
}
```

Cas particulier : nommage des attributs

- Bug vicieux (indétectable par les analyseurs de code)

```
@Override
private void setFeatures( String pname, int age ) {
    this.name = name.toUpperCase();
    this.age = age;
}
```

- Aucun moyen de s'en prémunir

Cas particulier : nommage des attributs

- Bug vicieux (indétectable par les analyseurs de code)

```
@Override
private void setAttribute( int a ) {
    f(a,x);
}
```

  - puis, on renomme l'argument a en x

```
@Override
private void setAttribute( int x ) {
    f(x,x);
}
```

  - Le compilateur repère l'erreur de masquage d'un attribut, donc on renomme :

```
@Override
private void setAttribute( int y ) {
    f(y,y);
}
```

  - Trop tard, le mal est fait !

Cas particulier : nommage des attributs

- Astuce du préfixe
  - `private String _nom;`
- Avantages
  - Distinguer d'un coup d'œil un attribut d'un paramètre ou d'une variable
  - Profiter de la complétion automatique des IDE sans ambiguïté sur l'attribut
  - Le compilateur nous aide à détecter les bugs précédents

Cas particulier : nommage des attributs

- Bugs maintenant impossibles ou détectables par le compilateur

```
public class Bottle {
    private int _volume;
    public Bottle( int volume) {
        _volume = volume; // Pas d'auto-affectation
    }
}
```

```
@Override
private void setFeatures( String pname, int age ) {
    _name = name.toUpperCase(); // Erreur de compilation : name inconnu !
    _age = age;
}
```

```
@Override
private void setAttribute( int x ) {
    f(x,_x); // Plus de masquage possible
}
```

Nommage des paquets en Java

- Adresse web (URL) de l'équipe de développement à l'envers (+ snake_case):
  - `org.eclipse.swt.graphics`
  - `fr.ensicaen.ecole.mon_projet.model`
  - `fr.ensicaen.ecole.mon_projet.view`

Bannir le code lourd

- N'utilisez pas de `this.attribut` ou `this.methode()`

Pas bien

```
this.discriminant = sqrt(this.b * this.b – 4 * this.a * this.c)
```

Bien

```
_discriminant = sqrt(_b * _b – 4 * _a * _c)
```

- `This` ne devrait être utilisé que comme référence à l'objet, pas pour l'accès à des membres (ce qui est implicite)

Bannir le code lourd

- N'utilisez pas de ***this***.*attribut* ou ***this***.*methode()*.
  - Mauvaise pratique dans les constructeurs :

Pas bien

```
public class Bottle {
    private int volume;
    public Bottle( int volume) {
        this.volume = volume;
}
```

Bien

```
public class Bottle {
    private int _volume;
    public Bottle( int volume) {
        _volume = volume;
}
```

Bannir le code naïf

- Perte de crédibilité :

Pas bien

```
if (boolean == true) ...
if (boolean != false) ...
if (test) {
    return true;
} else {
    return false;
}
```

Bien

```
if (boolean)
if (boolean)
return test;
```

- **Remarque** : ne jamais utiliser `true` ou `false` dans les tests, uniquement dans les affectations

Bannir le code de geek

- Code offusqué, notation personnelle, idiome non consensuelle

Pas bien

```
int x = y << 1;
```

Bien

```
int x = y * 2;
```

- Yoda conditions

Pas bien

```
if (4 == height) ...
```

Bien

```
if (height == 4)
```

Règles de nommage

- L'humour est à manier avec précaution

Pas bien

```
int _pigeons = 0; // les clients de l'entreprise
```

- Humour douteux

Pas bien

```
#define TRUE FALSE // Happy debugging suckers
```

Règle n°3
→ Écrire des fonctions auto-documentées

Fonctions courtes

- Le corps des fonctions doit être court (typiquement < 20 lignes)
  - Diviser en sous-fonctions

Fonction sans commentaire

- Le corps d'une fonction ne doit pas comporter de commentaires
  - S'il vous semble nécessaire d'ajouter des commentaires, divisez le corps de la fonction en sous-fonctions
    - « *Don't comment bad code, rewrite it.* »
      Brian Kernighan (*auteur du 1er livre sur le C qui reste la référence*)

Fonctions compactes

- Indice de la ligne vide

Pas bien

```
public int doStuff( ) {
    A a = new A();
    int result = a.foo();
                                    // Ligne vide
    B b = new B();
    result  += b.bar();
                                    // Ligne vide
    return result;
}
```

Bien

```
public int doStuff( ) {
    return processA() + processB();
}
```

Fonction à un seul niveau d'abstraction

Pas bien

```
public int[] process( ) {
    int[] array = getArray(); // 1er niveau d'abstraction
    for (int i = 0; i < array.length; i++) {
        if (array[i] < array[i - 1]) {
            swap(array, i - 1, i);
        }
    }

    removeTies(array);        // 2e niveau d'abstraction
    return array;
}
```

Bien

```
public int[] process( ) {
    rearrange(array);
    removeTies(array);
    return array;
}
```

Règle n°4
→ Respecter des standards de formatage de code

Respecter des standards de mise en style de code

- Chaque langage présente ses standards de mise en forme
- Par exemple, en Java
  - à la Unix (avec accolades égyptiennes - *Egyptian brackets*) :

Bien

```
int method( int p ) {
    if (test) {
    } else {
    }
}
```

  - Surtout pas

Pas bien

```
if (test)
{
}
else
{
}
```

Formatage en général

- TRÈS IMPORTANT
  - Toujours encadrer les instructions unilignes par des **accolades**

Pas bien

```
if (test)
    instruction;
```

Bien

```
if (test) {
    instruction;
}
```

  - Remarque : on peut forcer les IDE modernes à les mettre automatiquement

Exemple de conséquence du non respect

- Faille SSL sur iOS/OS X d'Apple (8 janvier 2014)
  - SSL: Secure Socket Layer
    - Protocole de sécurisation des échanges entre clients et serveur
  - Affecte les IPhones & IPads & IPods & Mac
  - Impacte Safari, Mail, ICloud …
  - Permet une attaque de type man-in-the-middle

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer
signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;

(...)

    hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(ctx,
                       ctx->peerPubKey,
                       dataToSign,              /* plaintext */
                       dataToSignLen,           /* plaintext length */
                       signature,
                       signatureLen);
    if(err) {
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
        goto fail;
    }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

Exemple de conséquence du non respect

- Éviter la trop fameuse source de bug :

Pas bien

```
if (condition)
//      statement
other statement
```

Exemple de conséquence du non respect

- Éviter le problème du « else pendant » (“*dangling else*”)

Pas bien

```
if (a) then if (b) then s1 else s2
```

Règle n°5
→ Ne pas faire d’optimisation prématurée

L’optimisation obscurcit le code

- Ne pas faire d'optimisation que le compilateur peut faire
- Pas de compromis à la lisibilité

Bien

```
perimeter = 2 * Math.PI * radius;
```

Pas bien

```
perimeter = 6.28 * radius;
```

Optimisation : profilage

- « Premature optimization is the root of all evil »

- L'optimisation se fait sur la base d'un profilage du programme
  - Java
    - Voir les outils de profilage (Profiling Tools) dans IntelliJ IDEA Ultimate
    - Plugin : Java JFR Profiler
  - C
    - Valgrind, GProf
- Quand l'optimisation est nécessaire et rend le code obscur :
  1) Isoler le code optimisé dans une méthode
  2) Commenter ce code à l'aide d'un cartouche

Plan du chapitre

1
Pourquoi faire propre ?

2
Pourquoi la pratique classique est fausse ?

3
Comment faire propre ?

4
Conclusion

Code propre dans l'industrie

- Revue de code
  - Pour garantir le respect des conventions de codage :
  - Exemple chez **Ubisoft** :
    - Relecture par pair avant intégration.
  - Par exemple chez **IBM** :
    - Relecture par un comité avant intégration (inclut la vérification des tests).

Application en projet

- Faire de la relecture par un ou plusieurs membres du groupe avant chaque intégration du travail d'un membre dans le code commun.
  - cf. relecture du rapport.
  - Cela permet aussi un partage de la connaissance sur le code du logiciel

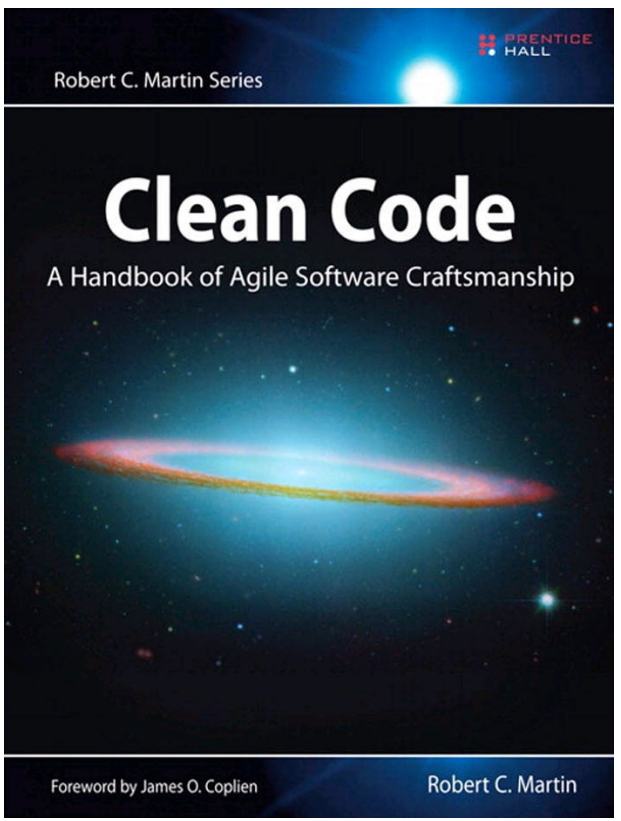
Code Documentation

Que retenir de ce chapitre ?

- Les commentaires sont une source de bruit.
  - Ils doivent être éliminés.
- La contre-partie est la propreté du code :
  - Nommage des identificateurs.
  - Structuration en fonctions
    - courtes
    - auto-documentées
    - avec un seul niveau d'abstraction.
  - Respect des standards de mise en forme.
- En tant qu'artisan du code, vous devez toujours garder votre code propre comme une table d'opération pour un chirurgien.

Lecture

- Robert C. Martin, « *Clean Code - A Handbook of Agile Software Craftsmanship* », Prentice Hall, 2009.



Démo : revue de code

- Code Mendeleïev d'un binôme d'étudiant de l'ENSICAEN