**Conventions de développement**

**Types de fichiers**

**.java** code source java

**.class** fichier de compilation d’un .java

**Fichiers .java**

Une seule class ou interface par fichier.

Structure d’une class ou interface :

- Commentaires de description de la class

- Package → ne pas utiliser de .\*

- Import

- Déclaration class ou interface

- Variables static, dans l’ordre :

- public → A utiliser le moins possible

- protected

- private

- Variables d’instances, dans l’ordre public, protected, private.

- Constructeurs, en 1er le constructeur par défaut

- Méthodes, regroupées par fonctionnalité

- Variables de bloc

**Mise en page**

indentation de 4 espaces

pas de ligne de plus de 120 caractères

en cas de ligne brisée :

- indentation 8 caractères puis les lignes suivantes alignées avec la 1ère ligne brisée

- chainage des méthodes avant chaque **.**

- coupure après **(** puis après **,** puis après **.** puis avant *opérateur*

une seule déclaration de variable par ligne

une seule instruction par ligne

une ligne blanche précède chaque bloc ou chaque commentaire

un espace blanc de chaque coté des opérateurs

reformater le code avec l’éditeur (intellij : clic-droit sur un fichier → Reformat Code)

**Commentaire de description de Class**

*Aucun caractère accentué dans les commentaires*

*/\*\**  
 *\* Nom : xxxxxx*  
 *\* Description : xxxxxxx*  
 *\** ***@version vnn.nn***  
 *\* Date : dd mmmm yyyy*  
 *\** ***@author xxxxxxxx***  
 *\*/*

**Commentaires de variables**

*Aucun caractère accentué dans les commentaires*

// initialisation de xxxxxxxxxxxxxxxxxxxx

**Commentaires des constructeurs et des méthodes**

*Aucun caractère accentué dans les commentaires*

*/\*\**  
*\* method de creation d'une partie*  
*\** ***@param***

*\** ***@return*** *Object*  
 *\*/*

**Convention de nommage**

Package : que des minuscules

class ou interface : CamelCase avec 1ère lettre en majuscule avec des mots entiers

Variables : CamelCase avec 1ère lettre en minuscule avec des mots entiers

Variables static final (constantes) : intégralement en majuscule

Méthodes et constructeurs : CamelCase avec 1ère lettre en minuscule avec des verbes

**Références :**

Appel d’une méthode static par sa class : Class.method()

Appel d’une méthode non static par un objet de la class : objet.method()

**Spécificités liées aux variables :**

Privilégier l’initialisation des variables dans un fichier dédié, par exemple application.properties, plutôt que dans le code, à part pour les valeurs « zéro » ou « null ».

Le cas échéant, l’utilisation de constantes bien nommées est acceptable, mais « nombre magiques » interdits dans des méthodes.

N’utilisez pas la valeur « null » comme un état « normal » d’une variable.

**Spécificités liées aux opérateurs :**

Utiliser les parenthèses dans les expression avec plusieurs opérateurs pour éviter les problèmes de précédences : **(a == b) && (c == d)** et non **(a == b && c == d)**

**Structure du code :**

Un même bloc de code ne doit pas se répéter plusieurs fois

Utiliser chaque fois que possible l’injection de dépendance, notamment en utilisant les interfaces et de coupler leurs implémentations grâce à des modules.

Encapsuler les conditions limites, par exemple en utuilisant des méthodes pour modifier les attributs d’un objet, ce qui permet de définir la visibilité des méthodes en question et de localiser dans ces méthodes les conditions limites.

Une méthode ne fait qu’une seule chose, mais bien, avec le moins possible de paramètres/arguments.

Les méthodes similaires et dépendants doivent être à proximité les unes des autres.

Éviter l’utilisation des booléens en-dehors des cas où il signifie vrai/faux. Un Enum est une solution possible.

Limiter le nombre de variables d’instance, sinon cela veut probablement dire que la class a trop de fonctions.

**Tests**

Prévoir de compléter