Conventions de nommage JAVA

Packages

Le nom d'un package doit respecter les conventions suivantes :

1. **Tout en minuscule.**
2. **Utiliser seulement [a-z], [0-9] et le point '.'**: Ne pas utiliser de tiret '-', d'underscore '_', d'espace, ou d'autres caractères ($, *, accents, ...).
3. La convention de Sun indique que tout package doit avoir comme root un des packages suivant: com, edu, gov, mil, net, org ou les deux lettres identifiants un pays (ISO Standard 3166, 1981).

Je pense les règles 1 et 2 doivent être suivies sans concessions. De toute manière, la plupart de ces règles sont obligatoires.

```
com.sun.eng
com.apple.quicktime.v2
edu.cmu.cs.bovik.cheese
```

Classes

Les noms des classes doivent respecter les conventions suivantes (d'après SUN):

1. **1ère lettre en majuscule**
2. **Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule**
3. **Donner des noms simples et descriptifs**
4. **Eviter les acronymes** : hormis ceux commum (XML, URL, HTML, ...)

De plus, d'une manière générale :

- **N'utiliser que les lettres [a-z] et [A-Z] et [0-9]** : Ne pas utiliser de tiret '-', d'underscore '_', ou d'autres caractères ($, *, accents, ...).

A débattre :

- Personnellement j'utilise 'Xml' et nom 'XML', par exemple XmlWriter, StatusXml, ... Je trouve que cela permet de mieux séparer le mot Xml du reste du nom de la classe. Cela permet aussi de rester cohérent dans le nommage d'attribut ou de paramètre xmlFile et nom xMLFile.

```
class Raster;
class ImageSprite;
```

Interfaces

Les mêmes conventions que les noms de classes s'appliquent.
D'une manière générale, je suis contre l'utilisation systématique de la lettre ' I ' pour préfixer les interfaces.
En effet, une classe peut devenir une interface sans pour autant changer toutes ses références.

```
interface RasterDelegate;
interface Storing;
interface StringConvertor;
```

Variables

Les noms des variables doivent respecter les conventions suivantes (d'après SUN):

1. **1ère lettre en minuscule**
2. **Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule**
3. **Donner des noms simples et descriptifs**
4. Ne pas commencer les noms avec '$' ou '_' bien que ce soit possible.
5. **Variable d'une seule lettre (pour un usage local)**
    - **int** : i, j, k, m, et n
    - **char** : c, d, et e
    - **boolean** : b

De plus, d'une manière générale :

- **N'utiliser que les lettres [a-z] et [A-Z] et [0-9]** : Ne pas utiliser de tiret '-', d'underscore '_', ou d'autres caractères ($, *, accents, ...).

A débattre :

- **Exceptions pour le caractère '_'** : Personnelement j'utilise toujours le caractère underscore dans 2 cas.
    - **a_xxx** : pour les paramètres (quelques exceptions)
    - **l_xxx** : pour les variables **locales** à une méthode (exceptions : retour, len, variable à 1 lettre)

```
int     i;
char    c;
float   myWidth;


public int getPropertyLength(String a_name) {
    String l_property = getProperty(a_name);
    return l_property.length;
}
```

Constants

Les noms des variables doivent respecter les conventions suivantes (d'après SUN):

- **Tout en majuscule**
- **Séparer les mots par underscore '_'**
- **Donner des noms simples et descriptifs**

De plus, d'une manière générale :

- **N'utiliser que les lettres [A-Z], [0-9] et '_'** : Ne pas utiliser de tiret '-' ou d'autres caractères ($, *, accents, ...).

A débattre :

- **Interface pour les constantes**
    - **En tant qu'inner class**
    - **Indépendante** : Personellement je trouve cela pratique de regrouper des constantes dans une interface. Dans certains cas, le nom de la classe sera toute en majuscule.

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

Fichiers Jars

Personnelement j'utilise désormais une convention simple qui est très populaire dans les projets Open Source (projets Jakarta, repository Maven).
Les noms des fichiers jars doivent respecter les conventions suivantes :

- **Tout en minuscule**
- **Séparer les mots par un tiret '-'**
- **Utiliser seulement les lettres [a-z], [0-9] et '-'** : Ne pas utiliser d'underscore '_' ou d'autres caractères ($, accents, ...).

A débattre :

- **Mettre le no de version dans le nom du fichier jar**
  - **Séparer le no de version et le nom par un tiret** : xxx-1.0.jar
  - **En général utiliser une version numérique** : 1.0 ou 1.05.3
  - **Possibilité d'utiliser des majuscules dans les no de version** : 1.0
  - **Utilisation de beta pour les versions beta**: séparer beta avec le reste par des tirets (1.0-beta, 1.0-beta-5)
- **Utiliser un répertoire 'jars' pour stocker les fichiers jars**

```
gb-fwk-1.0.jar
gb-tools-1.0-beta-7.jar
js-1.5R4-RC3.jar
```

Répertoire

Exemple de structure pour un projet nommé **gb-fwk** avec utilisation de maven.

- `gb-fwk // répertoire racine du projet`
  - `bak // backup de fichiers source`
  - `src // fichier source`
  - `class // fichiers compilés .class`
  - `target // répertoire généré par maven`
    - `docs // Fichiers de doc (format xml utilisation dans maven)`
    - `generated-xdocs // Fichiers source au format html (généré par maven)`
    - `javadoc // Documentation API (généré par maven)`

- `maven-repository // répertoire racine pour le repository maven`
  - `gb-fwk`
    - `jars`
      - `gb-fwk-1.0.jar`
      - `gb-fwk-1.1.jar`

Liens vers un descriptif complet de chaque convention:

JAVA - Introduction aux Normes et Conventions

JAVA - Convention de nommage

JAVA - Normes pour les Noms de packages

JAVA - Normes pour les Noms des classes

JAVA - Normes pour les Noms des méthodes

JAVA - Noms de variables couramment utilisées

JAVA - Normes utilisées pour nommer les constantes

JAVA - Exemples de code

Java - Coder des tests unitaires

Java ANT - Normes et Conventions de nommage des attributs

Représentation numérique de la date et de l'heure

Liens & Ressources sur les Normes et Conventions Java