**Institut Informatique Appliquée**

2015-2016

**Cahier des charges fonctionnelles du projet**

**QCM de l'institut informatique appliqué**

**Maxime Millot**

***A-Sommaire***

***A - Sommaire ----------------------------------------------------------------- page 2  
  
B - Base de donnée --------------------------------------------------------- page 3* 1 °) Diagramme de classe --------------------------------------------------------- page 3  
*C - Fonctions détailler -------------------------------------------------------- page 3 - 9* 1 °) Fonction détailler Mobile ------------------------------------------------- page 4 - 5  
*C – Jeux de teste -------------------------------------------------------------- page 3 - 9* 1 °) Détailler --------------------------------------------------------------------- page 6  
*E - Annexe ---------------------------------------------------------------------- page 9* 1 °) Convention de codage pour l'application Windows (C#)----- page 6 - 20 2 °) Convention de codage pour l'application Android (JAVA)---- page 20 - 23 3 °) Convention de codage pour l'application IOS (A venir)------- page \*\* - \*\***

***B - Base de données***

- IdClasse: Int  
- NomClasse: String  
- DateCrea: DateTime  
- DateMAJ: DateTime

- IdUser: Int  
- NomUser: String  
- PrenomUser: String  
- IdUser: String  
- MDPUser: String  
- Email: String  
- IdClasse: Classe  
- IdTypeUser: TypeUser  
- DateCrea: DateTime  
- DateMAJ: DateTime

Sont   
regroupés

Classe

Utilisateur

**1°) Diagramme de classe**

- IdQCM: Int  
- NomQcm: String  
- TempsFinQCM: DateTime  
- TimerQCM : DateTime  
- QCMFerme: bool  
- IdCategorie: Categorie  
- DateCrea: DateTime  
- DateMAJ: DateTime  
- iduser : user

1

\*

Qcm

- IdTypeUser: Int  
- NomTypeUser: String  
- Admin: Bool  
- DateCrea: DateTime  
- DateMAJ: DateTime

1

Reponse

Type d'utilisateur

Possède

\*

\*

Possède

\*

- IdCategorie : Int  
- NomCategorie : String  
- DateCrea: DateTime  
- DateMAJ: DateTime

Categorie

\*

Peuvent   
créer OU  
passer

1

- Identification (Session)  
- CrypteDonnee (2 Objet)  
- DécryptDonnee (2 Objet)

1

Possède

\*

\*

TimeQCM (Bool)  
Acceder Void  
CalculeNoteMax (Float)  
CalculeNote (Float)  
EnvoyeQCM (Objet)  
TrierQCM (ListQCM)

- iduser: User  
- idqcmU: QCM  
- QCMenvoye : Bool

UserQCM

\*

Réponds

\*

Question

Note

- IdQuestion: Int  
- NomQuestion: String  
- IdTest = QCM  
- Video = String  
- Image = string  
- Audio = String  
- DateCrea: DateTime  
- DateMAJ: DateTime

\*

Possède

1

- IdReponse: Int  
- NomReponse: String  
- BonneReponse: Bool  
- idquestion : Question  
- DateCrea: DateTime  
- DateMAJ: DateTime

- IdNote: Int  
- MaxNoteQuestion: Int  
- NoteMauvaiseRepsonse: Int  
- DateCrea: DateTime  
- DateMAJ: DateTime

Possède

1

1

1

Void affichageQuestion

***C - fonction Détailler***

***1°) Fonction Détailler Mobile*Nom :** **Décrypte  
Logique :** Permet d'obtenir une donnée décrypter. **Donné utilisé**: Id, Mdp, Clé Public, Clé Privé **Explication :** Récupérer les données protégés

**Nom :** **Crypte**  
**Logique :** Permet d'obtenir une donnée crypter. **Donné utilisé :** Id, Mdp**,** Clé Public, Clé Privé **Explication :** Protégésles données

**Nom : Identification**  
**Logique :** Ce connecter et pouvoir utiliser l’application sans connexion internet.  
**Explication :** Connections à la base de donnée puis on vérifie l’existence de l’utilisateur.  
**Donné ou fonction utilisé :** Id, Mdp, Décrypte  
**Si** les données (Id, Mdp) sont correcte on récupérer les données liés à l’utilisateur **sinon** on indique que les informations de connections sont incorrectes.

**Nom :** Time QCM  
**Logique :** Temps avant la fin du qcm et l’envoie.  
**Donné utilisé :** Qcm, Question  
**Explication :** On regarde :  
**Si** l’utilisateur est entraient de faire le qcm **alors** on fait apparait le temps pour faire le qcm. **Si** le temps pour faire le qcmest égale à faux **alors** il doit répondre à la question en cours et le qcm sera envoyé.  
**Si** l’utilisateur est entraient de faire le qcm **et** que le temps pour faire le qcm est égale à faux **alors** il doit répondre à la question en cours et le qcm sera envoyé.  
**Si** l’utilisateur n’est pas entraient de faire le qcm **et** que le temps pour faire le qcm est égale à faux **alors** il ne peut plus accéder au qcm, il aura une note de 0 et il sera envoyer.

**Nom :** **Acceder**  
**Logique :** Accéder ou afficher les qcms  
**Donné utilisé :** User, Classe, Qcm, Timer Qcm, clique  
**Explication :** On regarde :  
**Si** l’utilisateur ou la classe à laquelle il est liée, peuvent accéder au QCM **alors** il apparait dans la liste.  
**Si** il clique sur le qcm et que la fonction Timer Qcm n’est pas égale faux **alors** il accède au QCM

**Nom :** Trier QCM  
**Logique :** Afficher les qcm  
**Donné utilisé :** Qcm, Catégorie, Clique  
**Explication :** On regarde :  
**Si** l’utilisateur clique sur l’une des colonnes du tableau **alors** les qcm seront triées en fonction de la colonne sélectionné.

**Nom :** **Calcule Note Max**  
**Logique :** calculer la note maximale que peut recevoir un élève  
**Donné utilisé :** Qcm, Note, Question  
**Explication :** On regarde :  
**Si**  on a la façon de noter de chaque question **alors** si il apparait on fait le calcule sinon on informe que des données sont manquante et qu’il faut se reconnecter.

**Nom :** **Calcule Note**  
**Logique :** calculer la note de l’élève  
**Donné utilisé :** Qcm, Note, Question, Réponse  
**Explication :** On regarde :   
**Si**  on a la façon de noter de chaque question **alors** on fait le calcule de chaque question par les choix de l’utilisateur.

**Nom :** **Envoyé QCM**  
**Logique :** Envoyer le résultat  
**Donné utilisé :** Qcm, Note, Question, Réponse, Timer Qcm, crypter  
**Explication :** On regarde :  
**Si**  le qcm est fini (bouton de fin) **OU** si le timer Qcm = faux **alors** on envoie les informations en crypter **et si** timer Qcm = faux **alors** la note de l’élève est égale à 0.

**Nom :** affichage Question  
**Logique :** afficher chaque question  
**Donné utilisé :** Question, Time QCM  
**Explication :** On regarde :   
**Si** l’utilisateur clique sur le bouton (suivant ou revenir) **et** Time QCM n’est pas égale à faux **alors** on fait apparait la question.

***D- Jeu de teste***

***1°) Détailler***

Afin de rendre un logiciel correcte au client plusieurs jeux de teste seront envisager avec une édition d’une fiche de résultat et des actions pour chaque jeu de teste.   
  
Les premiers testes seront effectués lors du développement du programme :

Les testes unitaires : Validez la qualité du code et les performances des différents modules  
Les testes fonctionnels : Validez les spécifications techniques et les exigences fonctionnelles.

Enfin les dernier testes seront effectué à la fin du développement du programme :  
  
Les testes d’installation : Tester les procédure d’installation.  
Les testes intégration : Validez l’intégration des modules entre eux et dans leur environnement d’exploitation défini.  
  
Enfin après chaque résolution d’incident un fiche expliquant l’incident rencontré sera imprimer en plus des deux autres et un teste de non-régression sera effectuer pour être sûr qu’il n’y a pas de régression.  
  
Toute ces fiches seront stocker sur le git ou se trouve les documents.

***E - Annexe***

# *1°) Convention de codage pour l'application Windows* Conventions de codage C# (Guide de programmation C#)

Les conventions de codage répondent aux objectifs suivants :

* Elles confèrent une apparence cohérente au code, afin que les lecteurs puissent se concentrer sur le contenu, pas sur la disposition.
* Elles permettent aux lecteurs de comprendre le code plus rapidement en formulant des hypothèses selon leur expérience antérieure.
* Elles facilitent la copie, la modification et la gestion du code.
* Elles illustrent les meilleures pratiques en C#.

[**Conventions d'affectation de noms**](javascript:void(0))

* Dans les exemples courts qui ne comprennent pas de [directives](https://msdn.microsoft.com/fr-fr/library/sf0df423.aspx), utilisez les qualifications d'espace de noms.Si vous savez qu'un espace de noms est importé par défaut dans un projet, vous n'êtes pas obligé de qualifier entièrement les noms à partir de cet espace de noms. Les noms qualifiés peuvent être interrompus après un point (.) s'ils sont trop longs pour contenir sur une ligne unique, comme indiqué dans l'exemple suivant.

C#

var currentPerformanceCounterCategory = new System.Diagnostics.

PerformanceCounterCategory();

[**Conventions de disposition**](javascript:void(0))

Une bonne disposition utilise la mise en forme pour souligner la structure de votre code et en faciliter la lecture.Les exemples Microsoft et autres se conforment aux conventions suivantes :

* Utilisez les paramètres à vous, de l'éditeur de code (retrait intelligent, retrait de quatre caractères, tabulations enregistrées en tant qu'espaces).
* Écrivez une seule instruction par ligne.
* Écrivez une seule déclaration par ligne.
* Si les lignes qui se suivent ne sont pas mises en retrait automatiquement, mettez-les en retrait d'un taquet de tabulation.
* Ajoutez au moins une ligne blanche entre les définitions des méthodes et les définitions des propriétés.
* Utilisez des parenthèses pour rendre apparentes les clauses d'une expression, comme illustré dans le code suivant.

C#

if ((val1 > val2) && (val1 > val3))

{

// Take appropriate action.

}

[**Conventions de commentaires**](javascript:void(0))

* Placez le commentaire sur une ligne séparée, pas à la fin d'une ligne de code.
* Commencez le commentaire par une lettre majuscule.
* Terminez le texte de commentaire par un point.
* Insérez un espace entre le délimiteur de commentaire (//) et le texte du commentaire, comme illustré dans l'exemple suivant.

C#

// The following declaration creates a query. It does not run.

// the query.

* Ne créez pas de blocs d'astérisques mis en forme autour des commentaires.

[**Directives du langage**](javascript:void(0))

[**String, type de données**](javascript:void(0))

* Utilisez l'opérateur **+** pour concaténer les chaînes courtes, comme illustré dans le code suivant.

C#

string displayName = nameList[n].LastName + ", " + nameList[n].FirstName;

* Pour ajouter des chaînes dans des boucles, en particulier lorsque vous travaillez avec une quantité de texte importante, utilisez un objet [StringBuilder](https://msdn.microsoft.com/fr-fr/library/system.text.stringbuilder.aspx).

C#

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

}

//Console.WriteLine("tra" + manyPhrases);

[**Variables locales implicitement typées**](javascript:void(0))

* Utilisez le [typage implicite](https://msdn.microsoft.com/fr-fr/library/bb384061.aspx) pour les variables locales lorsque le type de la variable est évident à droite de l'affectation ou lorsque le type précis n'importe pas.

C#

// When the type of a variable is clear from the context, use var

// in the declaration.

var var1 = "This is clearly a string.";

var var2 = 27;

var var3 = Convert.ToInt32(Console.ReadLine());

* N'utilisez pas [var](https://msdn.microsoft.com/fr-fr/library/bb383973.aspx) lorsque le type n'est pas apparent à droite de l'affectation.

C#

// When the type of a variable is not clear from the context, use an

// explicit type.

int var4 = ExampleClass.ResultSoFar();

* N'utilisez pas le nom de la variable pour indiquer le type de la variable. Il peut ne pas être correct.

C#

// Naming the following variable inputInt is misleading.

// It is a string.

var inputInt = Console.ReadLine();

Console.WriteLine(inputInt);

* Évitez d'utiliser *var* à la place de dynamic*.*
* Utilisez le typage implicite pour déterminer le type de la variable de boucle dans les boucles [for](https://msdn.microsoft.com/fr-fr/library/ch45axte.aspx) et [foreach](https://msdn.microsoft.com/fr-fr/library/ttw7t8t6.aspx).

L'exemple suivant utilise un typage implicite dans une instruction **for**.

C#

var syllable = "ha";

var laugh = "";

for (var i = 0; i < 10; i++)

{

laugh += syllable;

Console.WriteLine(laugh);

}

L'exemple suivant utilise un typage implicite dans une instruction **foreach**.

C#

foreach (var ch in laugh)

{

if (ch == 'h')

Console.Write("H");

else

Console.Write(ch);

}

Console.WriteLine();

[**Type de données non signé**](javascript:void(0))

* En règle générale, utilisez **int** plutôt que les types non signés.L'utilisation de **int** est commun en C# et il est plus facile d'interagir avec d'autres bibliothèques lorsque vous utilisez **int**.

[**Tableaux**](javascript:void(0))

* Utilisez la syntaxe concise lorsque vous initialisez des tableaux sur la ligne de déclaration.

C#

// Preferred syntax. Note that you cannot use var here instead of string[].

string[] vowels1 = { "a", "e", "i", "o", "u" };

// If you use explicit instantiation, you can use var.

var vowels2 = new string[] { "a", "e", "i", "o", "u" };

// If you specify an array size, you must initialize the elements one at a time.

var vowels3 = new string[5];

vowels3[0] = "a";

vowels3[1] = "e";

// And so on.

[**Délégués**](javascript:void(0))

* Utilisez la syntaxe concise pour créer des instances d'un type délégué.

C#

// First, in class Program, define the delegate type and a method that

// has a matching signature.

// Define the type.

public delegate void Del(string message);

// Define a method that has a matching signature.

public static void DelMethod(string str)

{

Console.WriteLine("DelMethod argument: {0}", str);

}

C#

// In the Main method, create an instance of Del.

// Preferred: Create an instance of Del by using condensed syntax.

Del exampleDel2 = DelMethod;

// The following declaration uses the full syntax.

Del exampleDel1 = new Del(DelMethod);

[**Instructions try-catch et using dans la gestion des exceptions**](javascript:void(0))

* Utilisez une instruction [try-catch](https://msdn.microsoft.com/fr-fr/library/0yd65esw.aspx) pour la plus grande part de la gestion des exceptions.

C#

static string GetValueFromArray(string[] array, int index)

{

try

{

return array[index];

}

catch (System.IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of range: {0}", index);

throw;

}

}

* Simplifiez votre code à l'aide de l'instruction C# [using](https://msdn.microsoft.com/fr-fr/library/yh598w02.aspx).Si vous avez une instruction [try-finally](https://msdn.microsoft.com/fr-fr/library/zwc8s4fz.aspx) dans laquelle le seul code du bloc **finally** est un appel à la méthode [Dispose](https://msdn.microsoft.com/fr-fr/library/system.idisposable.dispose.aspx), utilisez à la place une instruction **using**.

C#

// This try-finally statement only calls Dispose in the finally block.

Font font1 = new Font("Arial", 10.0f);

try

{

byte charset = font1.GdiCharSet;

}

finally

{

if (font1 != null)

{

((IDisposable)font1).Dispose();

}

}

// You can do the same thing with a using statement.

using (Font font2 = new Font("Arial", 10.0f))

{

byte charset = font2.GdiCharSet;

}

[**Opérateurs && et ||**](javascript:void(0))

* Pour éviter les exceptions et accroître les performances en ignorant les comparaisons superflues, utilisez [&&](https://msdn.microsoft.com/fr-fr/library/2a723cdk.aspx) au lieu de [&](https://msdn.microsoft.com/fr-fr/library/sbf85k1c.aspx) et [||](https://msdn.microsoft.com/fr-fr/library/6373h346.aspx) au lieu de [|](https://msdn.microsoft.com/fr-fr/library/kxszd0kx.aspx) lorsque vous effectuez des comparaisons, comme illustré dans l'exemple suivant.

C#

Console.Write("Enter a dividend: ");

var dividend = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter a divisor: ");

var divisor = Convert.ToInt32(Console.ReadLine());

// If the divisor is 0, the second clause in the following condition

// causes a run-time error. The && operator short circuits when the

// first expression is false. That is, it does not evaluate the

// second expression. The & operator evaluates both, and causes

// a run-time error when divisor is 0.

if ((divisor != 0) && (dividend / divisor > 0))

{

Console.WriteLine("Quotient: {0}", dividend / divisor);

}

else

{

Console.WriteLine("Attempted division by 0 ends up here.");

}

[**New, opérateur**](javascript:void(0))

* Utilisez la forme concise d'instanciation d'objets, avec des types implicites, comme indiqué dans la déclaration suivante.

C#

var instance1 = new ExampleClass();

La ligne précédente est équivalente à la déclaration suivante.

C#

ExampleClass instance2 = new ExampleClass();

* Utilisez les initialiseurs d'objets pour simplifier la création d'objet.

C#

// Object initializer.

var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,

Location = "Redmond", Age = 2.3 };

// Default constructor and assignment statements.

var instance4 = new ExampleClass();

instance4.Name = "Desktop";

instance4.ID = 37414;

instance4.Location = "Redmond";

instance4.Age = 2.3;

[**Gestion des événements**](javascript:void(0))

* Si vous définissez un gestionnaire d'événements que vous n'avez pas besoin de supprimer ultérieurement, utilisez une expression lambda.

C#

public Form2()

{

// You can use a lambda expression to define an event handler.

this.Click += (s, e) =>

{

MessageBox.Show(

((MouseEventArgs)e).Location.ToString());

};

}

C#

// Using a lambda expression shortens the following traditional definition.

public Form1()

{

this.Click += new EventHandler(Form1\_Click);

}

void Form1\_Click(object sender, EventArgs e)

{

MessageBox.Show(((MouseEventArgs)e).Location.ToString());

}

[**Membres static**](javascript:void(0))

* Appelez les membres [static](https://msdn.microsoft.com/fr-fr/library/98f28cdx.aspx) en utilisant le nom de classe : *ClassName.StaticMember*.Cette pratique rend le code plus lisible en clarifiant l'accès aux membres static.Ne qualifiez pas un membre static défini dans une classe de base avec le nom d'une classe dérivée.Lorsque ce code est compilé, la lisibilité du code est trompeuse et le code peut s'interrompre à l'avenir si vous ajoutez à la classe dérivée un membre static de même nom.

[**Requêtes LINQ**](javascript:void(0))

* Utilisez des noms explicites pour les variables de requête.L'exemple suivant utilise seattleCustomers pour les clients qui se trouvent à Seattle.

C#

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

* Utilisez des alias pour vous assurer que les noms de propriétés des types anonymes sont correctement écrits en majuscules à l'aide de la casse Pascal.

C#

var localDistributors =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { Customer = customer, Distributor = distributor };

* Renommez les propriétés lorsque les noms de propriétés dans le résultat sont ambigus.Par exemple, si votre requête retourne un nom de client et un ID de distributeur, au lieu de les laisser sous la forme Name et ID dans le résultat, renommez-les pour montrer clairement que Name est le nom d'un client et ID l'ID d'un distributeur.

C#

var localDistributors2 =

from cust in customers

join dist in distributors on cust.City equals dist.City

select new { CustomerName = cust.Name, DistributorID = dist.ID };

* Utilisez le typage implicite dans la déclaration des variables de requête et des variables de portée.

C#

var seattleCustomers = from cust in customers

where cust.City == "Seattle"

select cust.Name;

* Alignez les clauses de requête sous la clause [from](https://msdn.microsoft.com/fr-fr/library/bb383978.aspx), comme illustré dans les exemples précédents.
* Utilisez les clauses [where](https://msdn.microsoft.com/fr-fr/library/bb311043.aspx) avant les autres clauses de requête pour garantir que les clauses de requête ultérieures opèrent sur l'ensemble de données réduit et filtré.

C#

var seattleCustomers2 = from cust in customers

where cust.City == "Seattle"

orderby cust.Name

select cust;

* Utilisez plusieurs clauses **from** au lieu d'une clause [join](https://msdn.microsoft.com/fr-fr/library/bb311040.aspx) pour accéder aux collections internes.Par exemple, une collection d'objets Student peut contenir chacune une collection de scores de test.Lorsque la requête suivante est exécutée, elle retourne chaque score supérieur à 90, avec le nom de l'étudiant correspondant.

C#

// Use a compound from to access the inner sequence within each element.

var scoreQuery = from student in students

from score in student.Scores

where score > 90

select new { Last = student.LastName, score };

#### 2°) Convention de codage pour l'application Android 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