Vérification automatique de la qualité de code

Bruno Mermet
2011

Introduction

- Plusieurs outils tentent de signaler de potentiels problèmes sur le code
  - Checkstyle
  - Findbugs
  - PMD
- Les intersections entre ces outils sont non-nulles
- Un autre outil STAN (STructure ANalysis for Java) se révèle assez complémentaire
- Dans ce support, seuls Checkstyle et Findbugs sont présentés
- Attention : même si ces outils sont fort utiles, leurs conclusions ne doivent pas systématiquement être appliquées au pied de la lettre

CheckStyle
http://checkstyle.sourceforge.net

Préambule

- Checkstyle est grandement configurable
- On ne présente ici que le fonctionnement de base

Vérification sur les blocs de code

- Pas de bloc vide (EmptyBlock)
- Place de l'accolade ouvrante (LeftCurly)
  - **eol** (end of line)
  - nl (new line)
  - nlow (new line on wrap) : fin de ligne si possible
- Forçage des accolades (NeedBraces)
- Place de l'accolade fermante (RightCurly)
  - **same** – même ligne que l'instruction suivante
  - alone – seule
- Pas de bloc imbriqué sans raison (AvoidNestedBlocks)

Conception des classes (1)

- Visibilité des attributs de classe (VisibilityModifier)
  - Seules les variables de classe public final peuvent être publiques
- Classe finale (FinalClass)
  - Une classe dont tous les constructeurs sont privés doit être « final »
- Une Interface est un type (InterfaceIsType)
  - Interdit les interfaces ne définissant pas de méthodes (interfaces « marqueur » et « constantes »)
- Empêcher l'instanciation d'une classe utilitaire (HideUtilityClassConstructor)
  - Aucun constructeur public (y compris constructeur par défaut) pour une classe ne contenant que des méthodes statiques

Conception des classes (2)

- Conception pour l'extension (DesignForExtension)
  - Les méthodes non privées des classes pouvant être « sous-classées » doivent être soit abstraites, soit finales, soit avec un corps vide
  - Discutable
- Exception mutable (MutableException)

  Une classe décrivant une exception (.*Exception ou .*Error) ne doit comporter que des champs « final »
- Limiter les exceptions renvoyées (ThrowsCount)
  - Limite le nombre d'exceptions qu'une méthode peut renvoyer (à 1 par défaut)
  - Conséquences : évite les « catch » multiples et force la définition d'une hiérarchie d'exceptions
- Déclarations des classes internes à la fin (InnerClassLast)

Syntaxe du code (1)

- ArrayTrailingComma
  - une virgule doit terminer la déclaration d'un tableau sur plusieurs lignes
  - Exemple : int [] a = {1,2,3,};
- AvoidInlineConditionals
  - Éviter l'opérateur cond ? alors : sinon;
- CovariantEquals
  - Vérifie que le paramètre d'un « equals » redéfini dans une classe est bien de type java.lang.Object
- DoubleCheckedLocking

Syntaxe du code (2)

- EmptyStatement

  Détection des instructions vierges (type « ;; »)
- EqualsAvoidNull

  Si on compare une variable de type String à une constante, mettre la constante à gauche :

  "toto".equals(maChaine) > maChaine.equals("toto")
- EqualsHashCode

  La redéfinition de « equals » doit impliquer celle de « hashCode »
- FinalLocalVariable

  Les variables locales non modifiées doivent être déclarées « final »

Syntaxe du code (3)

- HiddenField

  Vérifier qu'aucun paramètre/variable locale ne « cache » un variable d'instance
- IllegalInstanciation

  Si possible, passer par des « fabriques » plutôt que par des constructeurs (Integer.valueOf(3) plutôt que new Integer(3)...)
- InnerAssignement

  Éviter les affectations dans les expressions
- MagicNumber

  Éviter l'utilisation explicite de constantes numériques (autres que -1, 0, 1 et 2) dans le code
- MissingSwitchDefault

Syntaxe du code (4)

- ModifiedControlVariable

  Ne pas modifier les variables de boucle d'un « for » dans le corps de la boucle
- RedundantThrows

  Eviter les redondances dans la déclaration des exceptions renvoyées
- SimplifyBooleanExpression

  Simplifier les expressions booléennes (éviter les « a == true, !false, a || true, etc.)
- SimplifyBooleanReturn

  Éviter « if (valide()) return false; else return true; »
- StringLiteralEquality

  Ne pas utiliser == et != pour comparer des chaînes

Syntaxe du code (5)

- NestedForDepth
  - Limiter l'imbrication de boucles à une profondeur donnée (1 par défaut)
  - Solution : utiliser des méthodes intermédaires
- NestedIfDepth

  Limiter l'imbrication de tests (1 par défaut)
- NestedTryDepth

  Limiter l'imbrication des « try » (1 par défaut)
- NoClone

  Éviter la redéfinition de la méthode clone. À la place, préférer l'utilisation de constructeurs de copie ou d'une méthode « fabrique » statique.
- NoFinalizer

  Éviter les méthodes « finalize »

Syntaxe du code (6)

- SuperClone
  Vérifier qu'une méthode « clone » commence par un appel à super.clone()
- SuperFinalize
  Vérifier qu'une méthode « finalize » commence par un appel à super.finalize()
- IllegalCatch
  Éviter les « catch » de Exception, Error, RuntimeException
- IllegalThrows
  Ne pas déclarer de « throws » pour Error ou RuntimeException
- PackageDeclaration
  Imposer un « package » explicite pour toute classe

Syntaxe du code (7)

- JUnitTestCase
  - Vérification de certaines conformités des méthodes de test
- ReturnCount
  - Limiter le nombre de « return » dans une méthode (2 par défaut)
- IllegalType
  - Éviter la mention de certaines classes concrètes et forcer le passage par des interfaces (List au lieu de ArrayList par exemple)
- DeclarationOrder
  - Déclarer dans un ordre donné les constituants d'une classe, notamment : variables statiques, puis variables d'instance, puis constructeurs, puis méthodes

Syntaxe du code (8)

- ParameterAssignement
  - Ne pas affecter de valeur à un paramètre
- ExplicitInitialization
  - Éviter l'initialisation explicite d'une variable d'instance à la valeur par défaut de son type
- DefaultComesLast
  - Vérifier que la clause « default » est la dernière dans un « switch »
- MissingCtor
  - Vérifier qu'une classe concrète définit toujours un constructeur explicite
- FallThrough
  - Éviter les « case » d'un « switch » sans return ou break, à moins d'un commentaire « fall through »

Syntaxe du code (9)

- MultipleStringLiterals
  - Éviter l'utilisation multiple d'une même chaîne constante dans un fichier
- MultipleVariableDeclarations
  - Pas plus d'une déclaration de variable par ligne
- RequireThis
  - Vérifier que tout appel à une variable d'instance ou à une méthode sur l'objet courant utilise « this. »
- UnnecessaryParentheses
  - Vérifier la non utilisation de parenthèses inutiles
- OneStatementPerLine
  - Pas plus d'une instruction par ligne

Duplication de code

- StrictDuplicateCode
  - Principe

    Vérification « sans intelligence » de la duplication de code
  - Conséquences
    - Très rapide
    - Peu couteux en mémoire
    - Pas de fausses alarmes
    - Pas toujours très pertinent

En-têtes

- Header
- RegexpHeader

Imports (1)

- AvoidStarImport

  Éviter l'importation « brutale » avec l'utilisation de l'astérisque (comme « import java.util.* »)
- AvoidStaticImport

  Éviter l'utilisation de l'import statique
- IllegalImport

  Pour interdire l'importation de certains packages
- RedundantImport

  Éviter les imports « redondants » : 2 fois le même import, import depuis java.lang, import depuis le même package

Imports (2)

- UnusedImports
  - Détecter les imports inutilisés
- ImportOrder
  - Faire figurer les imports dans un ordre « logique » (voir http://checkstyle.sourceforge.net/config_imports.html pour plus de détails)
- ImportControl
  - Pour vérifier des contraintes sur les imports propres au projet aidant à la structuration

Javadoc

- JavadocPackage
  Vérifier la présence d'une javadoc pour chaque package
- JavadocType
  Vérifier la présence d'une javadoc pour chaque classe et chaque interface
- JavadocMethod
  Vérifier la présence d'une javadoc pour chaque méthode et chaque constructeur
- JavadocVariable
  Vérifier la présence d'une javadoc pour chaque variable
- JavadocStyle
  Vérifier la conformité de chaque documentation javadoc à un certain standard

Mesures (1)

- BooleanExpressionComplexity
  Limiter le nombre d'opérateurs booléens (&&, ||, !, etc.) dans les expressions booléennes (3 par défaut)
- ClassDataAbstractionCoupling
  Limiter le nombre d'instanciations d'autres classes dans une classe (7 par défaut)
- ClassFanOutComplexity
  Limiter le nombre de classes sur lesquelles une classe repose (20 par défaut)
- CyclomaticComplexity (http://en.wikipedia.org/wiki/Cyclomatic_complexity)
  Vérifier la complexité cyclomatique ; celle-ci repose sur les imbrications de tests (if, switch, ?:), boucles (while, do, for), traitements d'exception, et les opérateurs booléens dans les méthodes et constructeurs. Par défaut, limité à 10. Interprétation
  - 1-4 : bien
  - 5-7 : correct
  - 8-10 : envisager du refactoring
  - 11+ : re-factorer tout de suite !

Mesures (2)

- NPathComplexity

  Autre évaluation de la complexité d'un code. Par défaut, limité à 200

- JavaNCSS

  Limiter la taille du code ; par défaut :

  - Taille max d'une méthode : 50 lignes ;
  - Taille max d'une classe : 1500 lignes ;
  - Taille max d'un fichier : 2000 lignes

Divers (1)

- NewlineAtEndOfFile
- TodoComment

  Permet de se souvenir de tous les « To Do » restant
- Translation

  Vérifier la cohérence entre des fichiers de ressources
- UncommentedMain

  Vérifier que seules les classes/la classe principale a une méthode main non commentée (autres méthodes main = tests)
- UpperEll

  Vérifier que les constantes de type « long » sont définies avec un « L » majuscule
- ArrayTypeStyle

  Vérifier l'unification de la déclaration des tableaux : int [] tab (par défaut) ou int tab [].
- FinalParameters

  Vérifier que les paramètres des méthodes/constructeurs/catch sont « final »

Divers (2)

- DescendantToken
  - Critère très générique et très paramétrable (voir http://checkstyle.sourceforge.net/config_misc.html)
- Indentation
  - Vérification de l'indentation
- TrailingComment
  - Interdire les commentaires non seuls sur une ligne
- Regexp
  - Vérifier la présence d'un texte donné dans un fichier
- OuterTypeFilename
  - Vérifier la cohérence entre le nom d'un fichier et la classe qu'il contient

Modifieurs

- ModifierOrder

  public/protected/private < abstract < static < final < transient < volatile < synchronized < native < strictfp

- RedundantModifier

  Détecter les modifieurs inutiles (public et abstract pour les déclarations de méthodes dans les interfaces par exemple)

Conventions de nommage

- Modules
  - AbstractClassName : commence par Abstract ou finit par Factory
  - ClassTypeParameterName : une seule lettre majuscule
  - MethodTypeParameterName : idem
  - ConstantName : ~ une majuscule puis majuscules/chiffres/underscores
  - LocalFinalVariableName : minuscule puis lettres/chiffres
  - LocalVariableName, MemberName, MethodName : idem
  - ParameterName, StaticVariableName, TypeName idem
  - PackageName : ^[a-z]+(\.[a-zA-Z_][a-zA-Z0-9_]*)*$

    N.B.: convention du langage, mais préférer ^[a-z]+(\.[a-z][a-z0-9]*)*$

Dépassements de taille (1)

- ExecutableStatementCount

  Limiter le nombre d'instructions (par défaut à 30) dans les « blocs » suivants :

  - Constructeur
  - Méthode
  - Bloc d'initialisation statique
  - Bloc d'initialisation d'instance
- FileLength

  Limiter la longueur des fichiers (2000 lignes par défaut)
- LineLength

  Limiter la longueur maximale des lignes (80 caractères par défaut)

Dépassements de taille (2)

- MethodLength
  Limiter le nombre de lignes dans une méthode (150 par défaut)
- AnnonInnerLength
  Limiter la longueur des classes anonymes (20 lignes par défaut)
- ParameterNumber
  Limiter le nombre des paramètres des méthodes (7 par défaut)
- OuterTypeNumber
  Limiter le nombre de types dans un fichier (1 par défaut)
- MethodCount
  Limiter le nombre total de méthodes dans une classe, ainsi que le nombre de méthodes par visibilité dans une classe (100 par défaut)

Espaces (1)

- GenericWhiteSpace

  Pas d'espace de part-et-d'autre des '<' et '>' dans les types génériques
- EmptyForInitializerPad

  Contrôler la présence (ou l'**absence**) d'un espace si la clause d'initialisation est vide.
- EmptyForIteratorPad

  Contrôler la présence (ou l'**absence**) d'un espace si la clause d'itération est vide.
- MethodParamPad

  Contrôler l'absence d'espace et de retour à la ligne entre un nom de méthode et la parenthèse ouvrante qui suit.
- NoWhiteSpaceAfter

  Contrôler l'absence d'espace après une instruction ne finissant pas une ligne

Espaces (2)

- NoWhiteSpaceBefore
  Contrôler l'absence d'espace après une instruction ne commençant pas une ligne
- OperatorWrap
  Vérifier la position de l'opérateur si une ligne est « coupée » : **début de la ligne suivante** ou fin de la ligne précédente
- ParenPad
  Vérifier l'**absence** (ou présence) d'espace à l'intérieur des parenthèses
- TypeCastParenPad
  idem pour les transtypages
- FileTabCharacter
  Vérifier l'absence de caractères de tabulation

Espaces (3)

- WhiteSpaceAfter

  Contrôler la présence d'un espace après virgule, point-virgule et transtypage

- WhiteSpaceAround

  Contrôler pour un certain nombre de « tokens » qu'ils sont bien entourés d'espaces

FindBugs
http://findbugs.sourceforge.net/

Catégories

- Mauvaises pratiques (Bad Practice)
- Correction (Correctness)
- Expérimental (Experimental)
- Internationalisation (Internationalization)
- Code vulnérable (Malicious Code Vulnerability)
- Correction multi-thread (Multithreaded Correctness)
- Performance (Performance)
- Sécurité (Security)
- Epineux (Dodgy)

Mauvaises Pratiques (Bad Practice)

Mauvaises pratiques (1)

- AM: Creates an empty jar file entry
- AM: Creates an empty zip file entry
  - closeEntry() suit directement putNextEntry()
- BC: Equals method should not assume anything about the type of its argument
  - Contrôler le type de l'argument et si ce n'est pas celui de l'objet courant, renvoyer false
- BC: Random object created and used only once
  - Ne pas créer un objet Random à chaque tirage : inefficace et moins aléatoire

Mauvaises pratiques (2)

- BIT: Check for sign of bitwise operation

  Lors d'un test sur un « et » bit-à-bit, utiliser != 0 et non > 0 car en fonction du bit de signe, on peut avoir une erreur
- CN: Class implements Cloneable but does not define of use clone method
- CN: Clone method does not call super.clone()

  L'appel à super.clone() est indispensable pour que le type du clone créé soit le bon
- CN: Class defines clone() but doesn't implements Cloneable

Mauvaises pratiques (3)

- Co: Abstract class defines covariant compareTo() method
- Co: Covariant compareTo() method defined

  Pour implanter correctement compareTo(), il faut que le paramètre soit de type java.lang.Object
- DE: Method might drop exception
- DE: Method might ignore exception

  Traiter ou propager explicitement les exceptions
- DMI: Don't use removeAll() to clear a collection

  Utiliser clear() ; removeAll() est moins clair, moins efficace, et peut lever une ConcurrentModificationException

Mauvaises pratiques (4)

- DP: Classloaders should only be created inside doPrivileged block
- DP: Method invoked that should only be invoked inside a doPrivileged block
- Dm: Method invokes System.exit(...)

  Arrête la machine virtuelle java ; programme pas appelable par un autre ; préférer lever une RuntimeException
- Dm: Method invokes dangerous method runFinalizersOnExit()

  méthode réputée parmi les plus dangereuses (??)
- ES: Comparison of String […] using == or !=

Mauvaises pratiques (5)

- Eq: Abstract class defines covariant equals() method
- Eq: Covariant equals() method defined

  Le paramètre de equals doit être de type Object
- Eq: Equals checks for noncompatible operand

  Ne pas autoriser un equals dont le paramètre n'est pas sous-typ ou super-type de la classe courante
- Eq: Class defines compareTo(...) and uses Objet.equals()

  Comme il est recommandé que compareTo() == 0 soit équivalent à equals() == true, si compareTo est redéfini, logiquement, equals doit l'être

Mauvaises pratiques (6)

- FI: Empty finalizer should be deleteded
- FI: Explicit invocation of finalizer
  - appel réservé à la JVM ; peut avoir été appelé préalablement dans un autre thread
- FI: Finalizer nulls fields
- FI: Finalizer only nulls fields
  - Ne sert à rien
- FI: Finalizer does not call superclass finalizer
- FI: Finalizer nullifies superclass finalizer
  - Si un finalize() à vide est défini
- FI: Finalizer does nothing but call superclass finalizer
  - Inutile : supprimer la méthode

Mauvaises pratiques (7)

- GC: Unchecked type in generic call
  Vérifier le type d'un paramètre en cas de généricité contrainte
- HE: Class defines equals() but not hashcode()
- HE: Class defines equals() but uses Object.hashcode()
- HE: Class defines hashcode() but not equals()
- HE: Class inherits equals and uses Oject.hashcode()
  Tout classe doit vérifier que si a.equals(b), alors a et b doivent avoir même hashcode.
- HE: Class defines hashcode() but uses Object.equals()
  Ne viole *a priori* pas l'invariant, mais certainement problématique

Mauvaises pratiques (8)

- IC: Superclass uses subclass during initialization
  - Exemple

```
public class CircularClassInitialization {
    static class InnerClassSingleton extends CircularClassInitialization {
        static InnerClassSingleton singleton = new InnerClassSingleton();
    }
    static CircularClassInitialization foo = InnerClassSingleton.singleton;
}
```

  - Problème : singleton non initialisé ici
- IMSE: Dubious catching of IllegalMonitorStateException

  Cette exception ne doit normalement pas survenir
- ISC: Needless instantiation of class that only supplies static methods

Mauvaises pratiques (9)

- IT: Iterator next() method can't throw NoSuchElementException

  Quand on implante java.util.Iterator, la méthode next() doit renvoyer une telle exception si la fin de la collection est atteinte

- J2EE: Store of non serializable object into HttpSession
- JCIP: Fields of immutable classes should be final

  Classe implantant jcip.annotations.Immutable ; tous ses champs doivent être « final »

Mauvaises pratiques (10)

- NP: Method with Boolean return type returns explicit null

  Comme, avec l'autoboxing, cette méthode peut être utilisée là où on s'attend à avoir un booléen (sans précaution), cela est source de plantage
- NP: Clone method may return null
- NP: equals() method does not check for null argument

  Par convention, une méthode equals() doit renvoyer false si son argument est « null »
- NP: toString method may return null

  Renvoyer éventuellement une chaîne vide, mais jamais « null »

Mauvaises pratiques (11)

- Nm: Class names should start with an upper case letter
- Nm: Class is not derived from an Exception, even though it is named as such
- Nm: Confusing method names

  Des méthodes différentes ont des noms ne différant que par la casse
- Nm: Very confusing method names (but perhaps intentional)
- Nm: Field names should start with a lower case letter
- Nm: Use of identifier that is a keyword in later versions of Java
- Nm: Method names should start with a lower case letter

Mauvaises pratiques (12)

- Nm: Class names shouldn't shadow simple name of implemented interface

  exemple : package1.Foo() implante package2.Foo()
- Nm: Class names shouldn't shadow simple name of superclass
- Nm: Method doesn't override method in superclass due to wrong package for parameter

```
import alpha.Foo;
public class A {
    public int f(Foo x) { return 17; }
}
----
import beta.Foo;
public class B extends A {
    public int f(Foo x) { return 42; }
    public int f(alpha.Foo x) { return 27; }
}
```

Mauvaises pratiques (13)

- ODR: Method may fail to close database resource
- ODR: Method may fail to close database resource on exception

  S'assurer que quel que soit le chemin suivi dans une méthode, un objet lié à une base de données (connexion, row set) est bien fermé
- OS: Method may fail to close stream
- OS: Method may fail to close stream on exception

  Idem que ODR, mais sur les flux d'E/S
- RC: Suspicious reference comparison to constant

  Utiliser equals() pour comparer des Float, Integer, etc.
- RC: Suspicious reference comparison of Boolean values

Mauvaises pratiques (14)

- RR: Method ignores results of InputStream.read()
- RR: Method ignores results of InputStream.skip()

  Si on lit/saute moins d'octets qu'attendus, des problèmes peuvent survenir si cela n'est pas vérifié
- RV: Method ignores exceptional return value

  Toujours prendre en considération la valeur de retour d'une méthode
- SI: Static initializer creates instance before all static final fields assigned
- SW: Certain swing methods needs to be invoked in Swing thread
- UI: Usage of GetResource may be unsafe if class is extended

  Pb si la classe est étendue par une classe d'un autre package

Mauvaises pratiques (15)

- Se: Non-transient non-serializable instance field in serializable class
- Se: Non-serializable class has a serializable inner class
- Se: Non-serializable value stored into instance field of a serializable class
- Se: Comparator doesn't implement Serializable
- Se: Serializable inner class
- Se: serialVersionUID isn't final
- Se: serialVersionUID isn't long
- Se: serialVersionUID isn't static
- Se: Class is Serializable but its superclass doesn't define a void constructor
- Se: Class is Externalizable but doesn't define a void constructor
- Se: The readResolve method must be declared with a return type of Object
- Se: Transient field that isn't set by deserialization
- SnVI: Class is Serializable, but doesn't define serialVersionUID

Correction (Correctness)

Correction (1 : transtypage)

- BC: Impossible Cast
- BC: Impossible downcast

  transtypage ne pouvant réussir
- BC: Impossible downcast of toArray() result

  Rappel : dans les collections, toArray() renvoie un Object[], dont n'hérite pas T[] quel que soit T. Utiliser à la place toArray(T[]) où T est le type désiré
- BC: instanceof will always return false

Correction (2 : opérations binaires)

- BIT: Bitwise add of signed byte value
- BIT: Bitwise OR of signed byte value

  Risque de débordement
- BIT: Incompatible bit masks

  Si on compare (b & C) à D et que le résultat ne peut qu'être faut, en raison des valeurs de C et D
- BIT: Incompatible bit masks

  Idem avec un ou (|)
- BIT: Check to see if ((...) & 0) == 0

  Test toujours vrai
- BIT: Check for sign of bitwise operation

Correction (3)

- BOA: Class overrides a method implemented in super class Adapter wrongly
- BSHIFT: 32 bit int shifted by an amount not in the range 0..31
- Bx: Primitive value is unboxed and coerced for ternary operator

  Si on évalue (cond ? v1:v2) et que v1 et v2 sont de types différents (Integer et Float par exemple), ils sont d'abord unifiés avant d'être « dé-boxé »
- DLS: Dead store of class literal

  On stocke Foo.class sans jamais l'utiliser. L'initialistion statique était exécutée avant java 1.5, ne l'est plus depuis.
- DLS: Overwritten increment

  exemple : i = i++

Correction (4)

- DMI: Bad constant value for month
  - les numéros de mois sont entre 0 et 11
- DMI: hasNext method invokes next
  - hasNext n'est pas sensé déplacer l'itérateur
- DMI: Collections should not contain themselves
  - notamment, récursion infinie dans le calcul du hashcode
- DMI: Invocation of hashCode on an array
  - Ne tient pas compte du contenu du tableau. Si nécessaire, utiliser java.util.Arrays.hashcode(tableau)
- DMI: Double.longBitsToDouble invoked on an int
  - Méthode convertissant un long sensé représenté un double en double ; n'a pas de sens avec un int
- DMI: Vacuous call to collections
  - Exemples de code inutile c.containsAll(c) et c.retainAll(c)

Correction (5)

- Dm: Creation of ScheduledThreadPoolExecutor with zero core threads
- Dm: Useless/vacuous call to EasyMock method
- FE: Doomed test for equality to NaN

  renvoie toujours faux, y compris avec NaN
- GC: No relationship between generic parameter and method argument
- HE: Signature declares use of unhashable class in hashed construct

  N.B. : une classe redéfinissant equals et pas hashCode est considérée « unhashable » car elle ne vérifie pas a == b ssi hashcode(a) == hashcode(b)
- HE: Use of class without a hashCode() method in a hashed data structure

Correction (6 : utilisation de equals)

- EC: equals() used to compare array and nonarray
- EC: Invocation of equals() on an array, which is equivalent to ==
  Utiliser à la place java.util.Arrays.equals(...)
- EC: equals(...) used to compare incompatible arrays
- EC: Call to equals() with null argument
- EC: Call to equals() comparing unrelated class and interface
- EC: Call to equals() comparing different interface types
- EC: Call to equals() comparing different types
- EC: Using pointer equality to compare different types

Correction (7 : définition de equals)

- Eq: equals method always returns false
- Eq: equals method always returns true
- Eq: equals method compares class names rather than class objects
- Eq: Covariant equals() method defined for enum
- Eq: equals() method defined that doesn't override equals(Object)
- Eq: equals() method defined that doesn't override Object.equals(Object)
- Eq: equals method overrides equals in superclass and may not be symmetric

  A hérite de B, et A et B utilisent instanceof dans leur « equals », rendant « equals » non symétrique
- Eq: Covariant equals() method defined, Object.equals(Object) inherited

Correction (8 : formatage des chaînes)

- FS: Format string placeholder incompatible with passed argument

  exemple : System.out.printf("%d\n", "coucou")
- FS: The type of a supplied argument doesn't match format specifier

  exemple : String.format("%d","1")
- FS: MessageFormat supplied where printf style format expected

  On utilise des {0} au lieu de %d...
- FS: More arguments are passed than are actually used in the format string
- FS: Format string references missing argument

  Cas inverse du précédent, mais génère une exception
- FS: Illegal format string
- FS: No previous argument for format string

  Exemple : formatter.format("%<s %s","a","b")

Correction (9 : nombres)

- ICAST: integral value cast to double and then passed to Math.ceil
- ICAST: int value cast to float and then passed to Math.round

  Dans ces 2 cas, les 2 opérations conjuguées sont inutiles
- IM: Integer multiply of result of integer remainder

  Attention aux priorités avec le modulo : a % b * c et interprété comme (a % b) * c
- INT: Bad comparison of nonnegative value with negative constant

  On compare un positif « sûr » avec un négatif « sûr »
- INT: Bad comparison of signed byte

  Les octets signés sont dans l'intervalle [-128;127]

Correction (10 : JUnit)

- IJU: JUnit assertion in run method will not be noticed by Junit

  Ne pas faire d'assertions Junit dans un thread spécifiques, leur résultat ne sera pas interprété
- IJU: TestCase declares a bad suite method
- IJU: TestCase has no tests
- IJU: TestCase defines setUp that doesn't call super.setUp()
- IJU: TestCase implements a non-static suite method
- IJU: TestCase defines tearDown that doesn't call super.tearDown()

  Tous ces problèmes concernent Junit 3.x

Correction (11 : boucles infinies)

- IL: A collection is added to itself

  Risque de boucle infinie dans le calcul du hashCode
- IL: An apparent infinite loop
- IL: An apparent infinite recursive loop

Correction (12)

- IO: Doomed attempt to append to an object output stream
  On ne peut pas rajouter des "objets" dans in fichier
- IP: A parameter is dead upon entry to a method but overwritten
  exemple : public void foo(int i) { i = 4;}
  => la valeur initiale du paramètre n'est pas récupérée, mais on modifie la valeur de i
- MF: Class defines field that masks a superclass field
  présence d'un attribut portant le même nom qu'un attribut (visible) d'une superclasse
- MF: Method defines a variable that obscures a field
  variable locale portant le même nom qu'un attribut

Correction (13 : pointeur "null" – 1)

- NP: Null pointer dereference
- NP: Null pointer dereference in method on exception path
- NP: Method does not check for null argument
- NP: close() invoked on a value that is always null
- NP: Null value is guaranteed to be dereferenced
- NP: Value is null and guaranteed to be dereferenced on exception path
- NP: Method call passes null to a nonnull parameter

  N.B. : nonnull : marqué @Nonnull ou systématiquement déréférencé
- NP: Method may return null, but is declared @NonNull
- NP: A known null value is checked to see if it is an instance of a type

Correction (13 : pointeur "null" – 2)

- NP: Possible null pointer dereference

  Une branche mène au déréférencement d'un pointeur qui semble pouvoir être nul
- NP: Possible null pointer dereference in method on exception path
- NP: Method call passes null for nonnull parameter
- NP: Method call passes null for nonnull parameter

  Correspond à un risque potentiel de nullité du param.
- NP: Non-virtual method call passes null for nonnull parameter
- NP: Store of null value into field annotated NonNull
- NP: Read of unwritten field

Correction (14 : noms ambigüs)

- Nm: Class defines equal(Object); should it be equals(Object)?
- Nm: Class defines hashcode(); should it be hashCode()?
- Nm: Class defines tostring(); should it be toString()?
- Nm: Apparent method/constructor confusion

  Une méthode (avec type de retour, souvent « void ») a le même nom que la classe ; probablement un constructeur mal défini ?
- Nm: Very confusing method names

  Des noms de méthodes ne diffèrent que par la casse
- Nm: Method doesn't override method in superclass due to wrong package for parameter

  voir exemple dans « Mauvaises Pratiques (12) »

Correction (15)

- QBA: Method assigns boolean literal in boolean expression
  - test avec affectation ; risque de confusion = et ==
  - Exemple : if (a = true)
- RC: Suspicious reference comparison

  Utilisation d'un == ou d'un != sur des références
- RCN: Nullcheck of value previously dereferenced
- RE: Invalid syntax for regular expression
- RE: File.separator used for regular expression

  Problème survenant sous Windows, où le File.separator est un '\', avec un rôle spécifique dans les RE
- RE: "." used for regular expression

  Rappel "." dans une RE signifie n'importe quel caractère, et non "." Passer un "." seul semble donc douteux

Correction (16)

- RV: Random value from 0 to 1 is coerced to the integer 0
- RV: Bad attempt to compute absolute value of signed 32-bit hashcode
    - Problème : un hashCode peut valoir Integer.MIN_VALUE, or abs(Integer.MIN_VALUE) = Integer.MIN_VALUE, négatif également ; en effet, Integer.MAX_VALUE < | Integer.MIN_VALUE|
    - Exemples sur les chaînes : "polygenelubricants", "DESIGNING WORKHOUSES"
- RV: Exception created and dropped rather than thrown

  Exemple : if (x < 0) new IllegalArgumentException("x must be nonnegative");
- RV: Method ignores return value

Correction (17)

- RpC: Repeated conditional tests
  Test style (x == 0 || x == 0)
- SA: Double assignment of field
  Code style x = x = 4;
- SA: Self assignment of field
  Code style x = x avec x variable d'instance
- SA: Self comparison of field with itself
  Test style (x < x) où x est une variable d'instance
- SA: Self comparison of field with itself
  Calcul inutile utilisant une variable d'instance
  Exemples : x&x ; x-x
- SF: Dead store due to switch statement fall through
- SF: Dead store due to switch statement fall through to throw
  Un "break" manque probablement

Correction (18)

- SIC: Deadly embrace of non-static inner class and thread local

  Classe interne qui devrait être statique. Il y a risque d'interblocage entre la classe interne et un thread local à la classe englobante. Comme la classe interne n'est pas statique, elle contient une référence sur une instance de la classe englobante. Si le thread local contient une référence à une instance de la classe interne, les instances internes et englobantes seront toutes les deux accessibles et non éligible pour le ramasse-miettes.

- SIO: Unnecessary type check done using instanceof operator

  Le résultat est en fait déterminable statiquement

- SQL: Method attempts to access a prepared statement parameter with index 0

  Les indices dans un PreparedStatement commencent à 1

- SQL: Method attempts to access a result set field with index 0

Correction (19)

- STI: Unneeded use of currentThread() call, to call interrupted()
  méthode statique ; Thread.interrupted() suffit
- STI: Static Thread.interrupted() method invoked on thread instance
- SQL: Method attempts to access a prepared statement parameter with index 0
  Les indices dans un PreparedStatement commencent à 1
- SQL: Method attempts to access a result set field with index 0

Correction (20)

- Se : Method must be private in order for serialization to work

  Les méthodes définies pour la sérialisation (readObject, writeObject) doivent être privées pour être prises en compte
- Se: The readResolve method must not be declared static
- UMAC: Uncallable method defined in anonymous class

  Méthode définie dans une classe anonyme, ne surchargeant pas une méthode de la classe mère, et non appelée directement dans la classe. Elle ne peut donc pas être appelée du tout
- VA: Primitive array passed to function expecting a variable number of object arguments

  Si on passe un tableau de données de types primitifs à une méthode acceptant un nombre variable de paramètre, un tableau d'objets est créé contenant un objet : le tableau de données initial

Correction (21) – Type Qualifier

- TQ: Value annotated as carrying a type qualifier used where a value that must not carry that qualifier is required
- TQ: Value that might not carry a type qualifier is always used in a way requires that type qualifier
- TQ: Value that might carry a type qualifier is always used in a way prohibits it from having that type qualifier
- TQ: Value annotated as never carrying a type qualifier used where value carrying that qualifier is required

Pour tous ces problèmes, voir la JSR 308

Correction (22) – Uninitialized Read

- UR: Uninitialized read of field in constructor

  Lecture d'un champ non initialisé dans un constructeur ; peut être lié à une confusion entre un paramètre et un champ

- UR: Uninitialized read of field method called from constructor of superclass

  B hérite de A. Le constructeur de A appelle une méthode surchargée dans B lisant une variable d'instance de B.

Correction (23)

- USELESS_STRING: Invocation of toString on an array
  - Utiliser Arrays.toString() pour afficher un tableau
- USELESS_STRING: Array formatted in useless way using format string
  - Ne pas utiliser directement un tableau en avec une chaîne de formattage
- UwF: Field only ever set to null
  - Toutes les affectations dans un champ donné le sont avec la valeur « null »
- UwF: Unwritten field
  - La valeur d'un champ n'est jamais modifiée

Expérimental (Experimental)

Expérimental

- LG: Potential lost logger changes due to weak reference in OpenJDK

  La spécification de Logger a changé : elle utilise maintenant des « weak references » au lieu de références classiques (« strong references »)

- OBL: Method may fail to clean up stream or ressource
- OBL: Method may fail to clean up stream or resource on checked exception

  Lorsqu'une méthode ouvre un flux, une connexion à un base de données, etc., elle devrait normalement « nettoyer la ressource (close, dispose, etc.)

Internationalisation (Internationalization)

Internationalisation

- Dm: Consider using Locale parameterized version of invoked method

  Ne pas utiliser les méthodes toUpperCase() et toLowerCase() de la classe String, mais plutôt toUpperCase(Locale l) et toLowerCase(Locale l)

- Dm: Reliance on default encoding

  Pour une conversion octet ↔ String, ne pas utiliser l'encodage par défaut, mais utiliser une API correcte en lui passant un objet de type CharSet (ou le nom d'un jeu de caractères)

Code vulnérable (Malicious Code Vulnerability)

Code vulnérable (1)

- DP: Classloaders should only be created inside doPrivileged block
- DP: Method invoked that should be only be invoked inside a doPrivileged block
- EI: May expose internal representation by returning reference to mutable object

  Risque lorsque l'on renvoie directement une référence (sur un objet modifiable) stockée dans un champ. Lorsque cela est nécessaire, renvoyer plutôt une copie de l'objet référencé par le champ
- EI2: May expose internal representation by incorporating reference to mutable object

  Problème inverse : on stocke en interne une référence sur un objet modifiable externe
- FI: Finalizer should be protected, not public

Code vulnérable (2) – champs statiques

- MS: May expose internal static state by storing a mutable object into a static field
- MS: Field isn't final and can't be protected from malicious code
- MS: Public static method may expose internal representation by returning array
- MS: Field should be both final and package protected
- MS: Field is a mutable array
- MS: Field is a mutable Hashtable
- MS: Field should be moved out of an interface and made package protected
- MS: Field should be package protected
- MS: Field isn't final but should be

Correction multi-thread (Multithreaded Correctness)

Performance (Performance)

Performance (1)

- Bx: Primitive value is boxed and then immediately unboxed
- Bx: Primitive value is boxed then unboxed to perform primitive coercion
- Bx: Boxed value is unboxed and then immediately reboxed
- Bx: Method allocates a boxed primitive just to call toString

  new Integer(1).toString() → Integer.toString(1)
- Bx: Method invokes inefficient floating-point Number constructor; use static valueOf instead
- Bx: Method invokes inefficient Number constructor; use static valueOf instead

  Double.valueof(double) → new Double(double)

  le deuxième ne crée par forcément d'objet

Performance (2)

- Dm: The equals and hashCode methods of URL are blocking
- Dm: Maps and sets of URLs can be performance hogs

  Les méthodes `equals` et `hashCode` de la classe URL impliquent une résolution des noms, processus long. Préférer la classe URI
- Dm: Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead

  Il n'existe en effet que 2 instances de cette classe
- Dm: Explicit garbage collection; extremely dubious except in benchmarking code
- Dm: Method allocates an object, only to get the class object

  Préférer la variable de classe `class`

Performance (3)

- Dm: Use the nextInt method of Random rather than nextDouble to generate a random integer
  (int)(r.nextDouble() * n) → r.nextInt(n)
- Dm: Method invokes inefficient new String(String) constructor
  Inutile car les instances de String sont non modifiables
- Dm: Method invokes toString() method on a String
- Dm: Method invokes inefficient new String() constructor
  Utiliser "" à la place
- HSC: Huge string constants is duplicated across multiple class files

Performance (4)

- ITA: Method uses toArray() with zero-length array argument
  Préférer myCollection.toArray(new Foo[myCollection.size()])
- SBSC: Method concatenates strings using + in a loop
  Préférer l'utilisation d'un StringBuilder
- SIC: Should be a static inner class
- SIC: Could be refactored into a named static inner class
  Classe interne ne faisant pas référence à l'instance de la classe englobant à laquelle est est rattachée
  Si la classe est anonyme, la nommer
- SIC: Could be refactored into a static inner class
  La seule référence à la classe englobante est dans un paramètre du constructeur.

Performance (5)

- SS: Unread field: should this field be static?
- UM: Method calls static Math class method on a constant value
- UPM: Private method is never called
- UrF: Unread field
- UuF: Unused field
- WMI: Inefficient use of keySet iterator instead of entrySet iterator

Sécurité (Security)

Sécurité

- Dm: Hardcoded constant database password
- Dm: Empty database password
- HRS: HTTP cookie formed from untrusted input
- HRS: HTTP Response splitting vulnerability
- SQL: Nonconstant string passed to execute method on an SQL statement

  Préférer un PreparedStatement pour éviter les injections de code
- SQL: A prepared statement is generated from a nonconstant String
- XSS: JSP reflected cross site scripting vulnerability
- XSS: Servlet reflected cross site scripting vulnerability in error page
- XSS: Servlet reflected cross site scripting vulnerability

Epineux (Dodgy)

Code épineux (1)

- BC: Questionable cast to abstract collection
- BC: Questionable cast to concrete collection
- BC: Unchecked/unconfirmed cast
- BC: instanceof will always return true
- BSHIFT: Unsigned right shift cast to short/byte
- CI: Class is final but declares protected field
- DB: Method uses the same code for two branches
- DB: Method uses the same code for two switch clauses
- DLS: Dead store to local variable

  Une valeur est stockée dans une variable locale qui n'est plus utilisée par la suite. Conséquence possible du comportement du compilateur
- DLS: Useless assignment in return statement

  Instruction type return *a=2+1* avec *a* variable locale

Code épineux (2)

- DLS: Dead store of null to local variable
  - Depuis java 1.6, n'influence plus le GC
- DLS: Dead store to local variable that shadows field
- DMI: Code contains a hard coded reference to an absolute pathname
- DMI: Non serializable object written to ObjectOutput
- DMI: Invocation of substring(0), which returns the original value
- Dm: Thread passed where Runnable expected
- Eq: Class doesn't override equals in superclass
  - la superclasse redéfinit equals, et la sous-classe définit de nouveaux champs sans redéfinir equals
- Eq: Unusual equals method
  - méthode equals semblant ne pas tester la compatibilité du type du paramètre avec le type courant

Code épineux (3)

- FE: Test for floating point equality
  - Comparer toujours à un epsilon prêt. Sinon, utiliser la classe BigDecimal
- FS: Non-Boolean argument formatted using %b format specifier
- IA: Ambiguous invocation of either an inherited or outer method

  Une classe interne appelle une méthode toto() qui existe dans sa classe mère et dans la classe englobante. La méthode appelée est celle de la classe mère, mais est-ce le comportement voulu ? Si oui, écrire plutôt super.toto()

Code épineux (4)

- IC: Initialization circularity
- ICAST: integral division result cast to double or float

  int x = 2 ; int y = 5 ;

  double value1 = x / y → double value2 = x / (double) y ?
- ICAST: Result of integer multiplication cast to long

  convertir en « Long » avant la multiplication

  exemple : 1000*3600*24*j → 1000L*3600*24*j
- IM: Computation of average could overflow

  (a+b)/2, en cas de débordement, peut donner un résultat négatif

Code épineux (5)

- IM: Check for oddness that won't work for negative numbers
  a%2 vaut -1 si a est impair négatif ; Donc pour tester une imparité, ne pas faire a*%2 ==1 mais plutôt a & 1 == 1 ou a%2 != 0
- INT: Integer remainder modulo 1
  x%1 vaut toujours 0 !
- INT: Vacuous bit mask operation on integer value
  - Opération bit-à-bit inutile (ex : v & 0xffffffff)
- INT: Vacuous comparison of integer value
  - Exemple : x <= Integer.MAX_VALUE

Code épineux (6)

- MTIA: Class extends Servlet class and uses instance variables
- MTIA: Class extends Struts Action class and uses instance variables
- NP: Dereference of the result of readLine() without nullcheck
- NP: Immediate dereference of the result of readLine()
- NP: Load of known null value
- NP: Possible null pointer dereference due to return value of called method
- NP: Possible null pointer dereference on branch that might be infeasible

Code épineux (7)

- NP: Parameter must be nonnull but is marked as nullable
- NP: Read of unwritten public or protected field
- NS: Potentially dangerous use of non-short-circuit logic
- NS: Questionable use of non-short-circuit logic

  Utilisation de & et | vraisemblablement à la place de && et ||
- PZLA: Consider returning a zero length array rather than null
- QF: Complicated, subtle or wrong increment in for-loop

  La variable incrémentée dans la boucle n'est pas utilisée dans le test de sortie de boucle

Code épineux (8)

- RCN: Redundant comparison of non-null value to null
- RCN: Redundant comparison of two null values
- RCN: Redundant nullcheck of value known to be non-null
- RCN: Redundant nullcheck of value known to be null
- REC: Exception is caught when Exception is not thrown
- RI: Class implements same interface as superclass
- RV: Method checks to see if result of String.indexOf is positive
- RV: Remainder of hashCode could be negative
- RV: Remainder of 32-bit signed random integer
  - Y penser dans le cas d'un modulo

Code épineux (9)

- RV: Method ignores return value, is this OK?
- SA: Double assignment of field
- SA: Double assignment of local variable

  Instruction type x = x = 7
- SA: Self assignment of local variable

  Instruction type x = x
- SF: Switch statement found where one case falls through to the next case
- SF: Switch statement found where default case is missing
- ST: Write to static field from instance method

Code épineux (10)

- Se: private readResolve method not inherited by subclasses
- Se: Transient field of class that isn't Serializable
- TQ: Value required to have type qualifier, but marked as unknown
- TQ: Value required to not have type qualifier, but marked as unknown
- UCF: Useless control flow
- UCF: Useless control flow to next line

Exemple :

```
if (argv.length == 1);
        System.out.println("Hello, " + argv[0]);
```

Code épineux (11)

- UrF: Unread public/protected field
- UuF: Unused public or protected field
- UwF: Field not initialized in constructor but dereferenced without null check
- UwF: Unwritten public or protected field
- XFB: Method directly allocates a specific implementation of xml interfaces
    - Utiliser plutôt les classes fabriques à la place