

TEAM TELA

SI3 – G3 | 2015-2016

Rapport de projet

Steganographie – Fin de projet

*Version 2 – 10 janvier 2016*

Table des matières

[I. Présentation du sujet 2](#_Toc439896854)

[II. Choix techniques en Java 3](#_Toc439896855)

[a. L’héritage 3](#_Toc439896856)

[b. Le paradigme objet 4](#_Toc439896857)

[c. Bibliothèques externes 5](#_Toc439896858)

[1. Imageio-pnm 5](#_Toc439896859)

[2. Apache Common CLI 6](#_Toc439896860)

[3. Log4j 6](#_Toc439896861)

[4. BoofCV 6](#_Toc439896862)

# Présentation du sujet

Le projet de sténographie se déroule sur un semestre entier. Le but du projet est d’utiliser deux technologies différentes pour chaque partie du projet : un langage **procédural** et un langage **objet**.

Le sujet est de pouvoir dissimuler de l’information dans une image, et être capable de la récupérer sans altération. Pour se faire, la méthode LSB (Less Significant Bit) était imposée. La solution proposée doit être accompagnée d’un banc de tests (beSSnchmarks). Le code doit aussi être testé unitairement afin d’assurer que notre produit est fonctionnel et fiable. Sachant que le projet est divisé en deux partie : **révélation** et **dissimulation**, la technologie utilisée pour une des parties ne devait pas être retrouvée dans l’autre partie.

Voici un schéma général du sujet avec les différentes étapes :

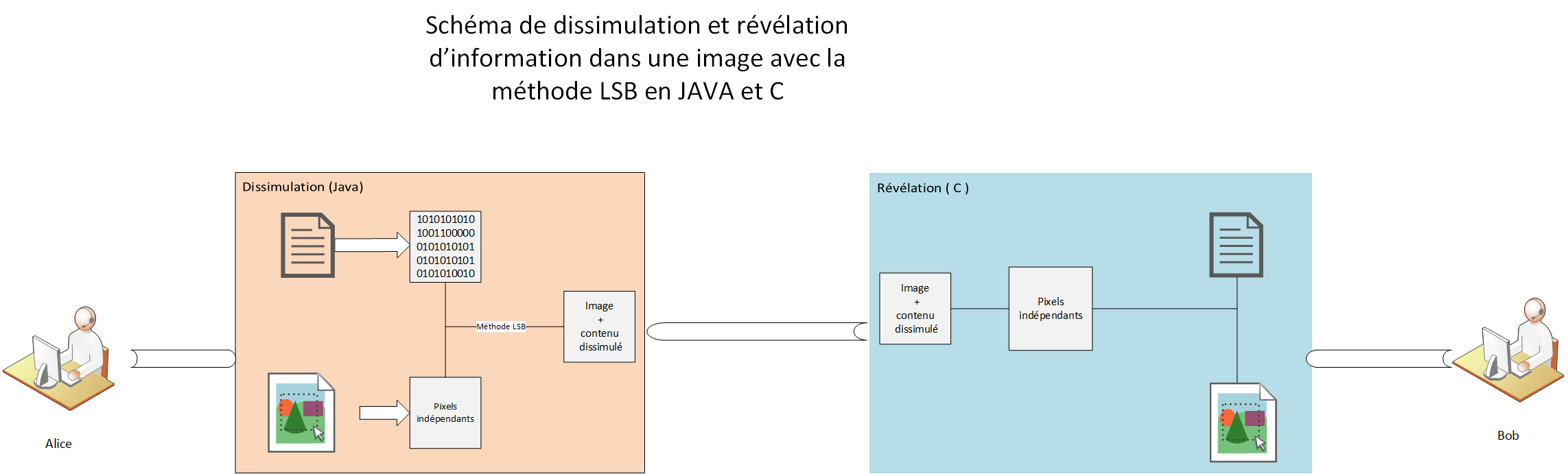


Figure : Schéma général du sujet

Nous nous sommes divisés en deux groupes, avec un responsable pour chaque partie (responsables présents sur JIRA), afin de coordonner le développement et s’assurer de sa réussite. Nos groupes sont composés d’un élève d’IUT et un élève de classe préparatoire, afin de pouvoir, dans chaque partie, profiter des points forts de chaque formation dans les deux sections.

Les spécifications techniques sont diverses et variées ; L’utilisateur doit pouvoir entrer les différents paramètres de dissimulation et de révélation via la ligne de commande. C’est ainsi qu’il pourra gérer le fichier d’entrée (lien vers une image), l’information à dissimuler qui sont les deux paramètres les canaux à utiliser, mais aussi le nombre de bit par pixel affectés par la dissimulation, le pattern à suivre pour dissimuler, le nombre magique qui informe de la fin du message, le format d’entrée, le calcul ou non des métriques, le format ainsi que le chemin de sortie, qui ont, quant à eux, des valeurs par défaut si l’utilisateur n’indique rien.

Les images hôtes pour la dissimulation sont limités en format, c’est ainsi que ceux autorisés sont : JPEG, BMP, PNG, PGM, PPM. Cependant en sortie, les mêmes formats sont disponibles, sauf le JPEG qui est un format qui compresse les données lors de la création, ce qui nous poserait problème car nous perdons donc de l’information.

Les canaux utilisés sont : Rouge, Bleu et Vert pour les images en couleur, Alpha pour celles contenant de la transparence et Gris en cas de photo en nuance de gris (format PGM en l’occurrence).

La présence de trois métriques doit être développée : temps écoulé lors de la dissimulation, corrélation entre l’image d’entrée et de sortie, ainsi que les pixels impactés par la dissimulation. Nous reviendrons dessus plus tard.

# Choix techniques en Java

## L’héritage

Afin de rendre notre projet flexible pour rajouter des fonctionnalités rapidement en impactant au minimum les classes (notamment pour la dernière semaine de projet), nous avons utilisé les design pattern et technologies proposés par le langage. La technique majoritairement présente dans notre projet est l’héritage. En effet, voici des exemples d’application dans notre code :

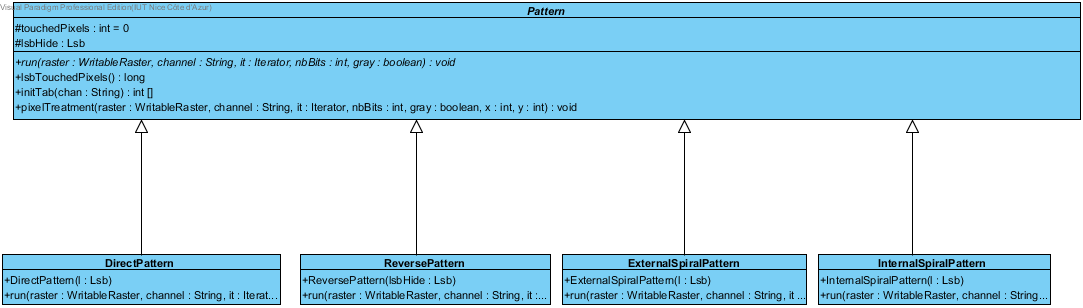


Figure :Hiérarchie des patterns

Grâce à cette organisation, l’ajout d’un pattern est rapide, il suffit de créer une classe fille de Pattern, ainsi que d’implémenter les méthodes adéquates.

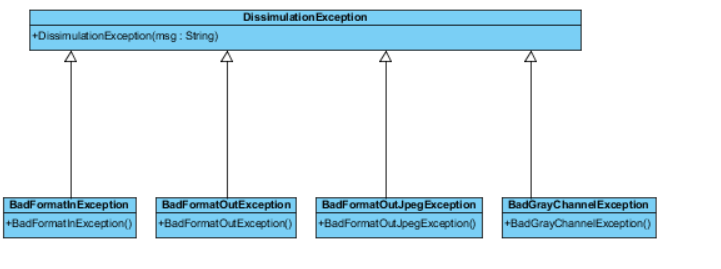


Figure : Hiérarchie de quelques exceptions

Ici, nous voyons quelques exceptions (faute de place) qui sont filles d’une classe centrale, ce qui permet la lisibilité et la remontée de l’exception au plus haut dans le programme.

Nous constatons ainsi que le fait d’utiliser l’héritage permet une meilleure généricité du code, afin de supprimer les doublons, en centralisant les méthodes et les attributs, mais aussi en imposant des méthodes. Une interface aurait fait l’affaire si on n’avait pas eu besoin d’attributs dans la classe mère.

La hiérarchie permet aussi d’ajouter facilement un nouvel élément dans le programme sans tout redéfinir, cela ajoute donc de la robustesse à notre produit ainsi de la flexibilité. Cela nous sera utile, comme dit précédemment, lors de la dernière semaine de projet où l’on devra implémenter une nouvelle fonctionnalité.

## Le paradigme objet

Travaillant avec un langage objet, nous étions obligés d’utiliser la richesse de ce langage afin d’abstraire les informations, les rendre facile d’utilisation et d’échange. Notre projet est séparé en package en fonction de la portée des classes, les services qu’elles rendent …. Le programme a été éclaté au maximum afin, encore une fois de le rendre facile de modification, mais aussi indépendant du sujet. C’est pour cela que nous retrouvons par exemple une classe Lsb, qui est passé en paramètre au constructeur du Pattern. Grâce à ce découpage, si nous voulons utiliser une nouvelle méthode de dissimulation d’information, nous n’aurons qu’à créer une autre classe s’occupant de cela, l’instancier, et la passer en paramètre. Voici des exemples de découpages réalisés dans notre projet :

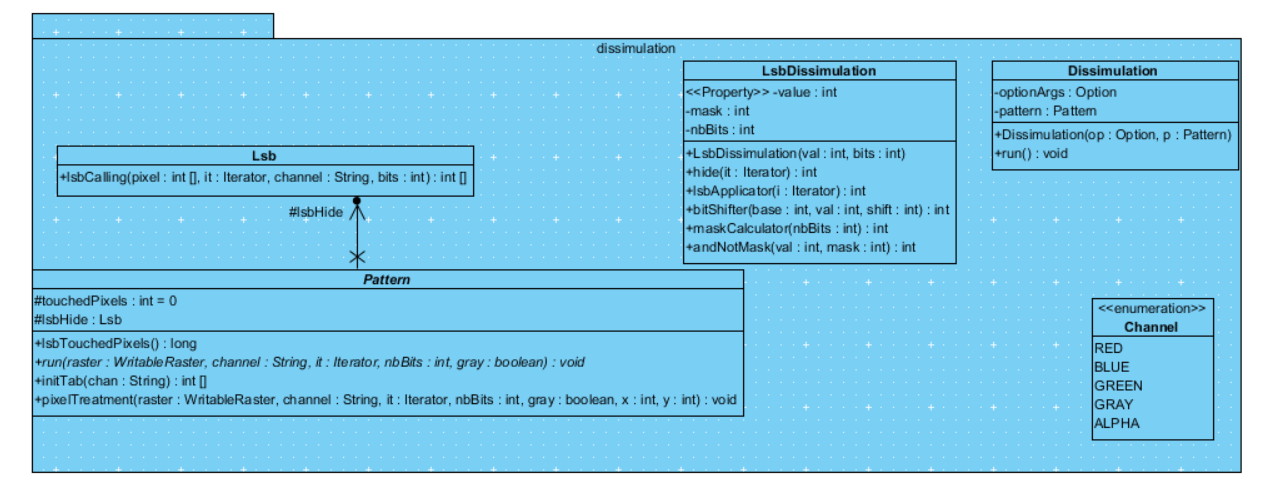


Figure : Diagramme de classe du LSB

C’est donc ce package qui s’occupe de la dissimulation d’une information en LSB, on voit très bien que le parcourt de l’image est indépendamment de la technique utilisée.

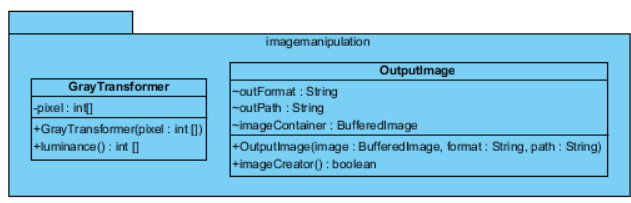


Figure : Package de manipulation de classes

Les manipulations diverses de l’image ont été séparées du reste du code ici car on pourrait très bien ne pas recréer d’image après dissimulation d’information mais afficher le contenu sur la sortie standard par exemple. C’est ainsi que l’on a externalisé ceci afin de les isoler. Cela nous a permis, pendant le projet de se rendre compte de la facilité de modification d’une erreur lors de la mauvaise création d’un fichier pour des formats non supportés.

Concernant les formats d’image, ils sont stockés dans un Enum car ce sont des constantes. Si on voulait ajouter de nouveaux formats d’image, un ajout dans l’enum du format permettra la reconnaissance de celui-ci et sera opérationnel (si le fonctionnement est similaire aux précédents). Sinon, il faudra ajouter les différentes conditions de traitement dans la classe Config qui gère les paramètres de la ligne de commande par rapport au format entré.

La même chose est possible pour les Canaux, qui sont stockés dans un Enum.

## Bibliothèques externes

Notre projet a été construit avec Maven, nous avons réussi à regrouper toutes nos librairies dans un pom.xml afin de faciliter l’import, avec les futurs bots qui feront tourner nos projets.

### Imageio-pnm

En java, les formats PNM (Portable Anymap) ne sont pas supportés. Les ayant besoin dans notre projet, nous avons testé différentes bibliothèques et choisi Imageio-pnm (<https://github.com/eug/imageio-pnm>). Cependant nous avons rencontré plusieurs problèmes avec cette bibliothèque et c’est grâce à la fonction « issue » de Github et après quelques échanges avec le développeur que nous avons réglé certains problèmes, notamment la non présence du projet sur le repository central de Maven. Nous avons de plus, avec l’aide d’un membre d’un autre groupe (Anasse Ghira – Team GZLS), modifié la librairie afin d’y intégrer un autre constructeur pour l’écriture d’un fichier PNM, qui était jusque-là impossible si nous donnions un BufferedImage issu d’un format non PNM.

### Apache Common CLI

Cette bibliothèque nous a permis de réduire au maximum la compléxité de notre code. En effet, si nous voulions créer notre propre parser de ligne de commande, nous aurions multiplié les boucles et autres itérations, alors que nous avons à disposition une librairie qui permet de le faire, avec des fonctionnalités très intéressantes. Nous l’avons donc mis en place dans notre projet et utilisé des services mis à disposition afin de maximiser la bonne lecture des arguments (ajout de séparateurs, d’arguments par defaut etc). Cette bibliothèque a été d’une grande utilité.

### Log4j

Afin de permettre une écriture propre et ne pas saturer la sortie standard, nous avons utilisé Lo4j qui est un logger, afin d’écrire sur la sortie standard en filtrant le flag du message (ERROR / DEBUG / INFO …), ou en écrivant dans un fichier.

### BoofCV

# Choix techniques en C

## Séparation des fichiers c ou un truc

## Bibliothèques utilisées

### OpenCV

# Divergence entre les paradigmes

Nous avons constaté un certain écart entre les deux paradigmes utilisés. En effet, en JAVA, nous avons eu la possibilité de manipuler des images très rapidement, contrairement au C. Cela est dû au fait que **ImageIO** est présent nativement en Java, alors qu’en C, nous avons installé la librairie **OpenCV** qui nous a posé aussi bien des problèmes de paramétrages en fonction des machines, des systèmes d’exploitation etc, mais aussi dans la compréhension et l’utilisation avec des problèmes rencontrés.

L’autre gros écart est la gestion des erreurs. En Java, nous avons eu une grande facilité à résoudre des problèmes tels que le **NullPointerException** ou alors un **ArrayOutOFBoundException**, alors qu’en C, lorsque l’on rencontre un **Core** **Dumped**, ou **Segmentation** **Fault**, il est plus difficile de trouver la cause, d’autant plus si le projet a déjà pris de l’ampleur. Une erreur en C est nettement plus bloquante que celle en JAVA car ce dernier nous donne la ligne exacte du problème. L’utilisation du debugger en JAVA a été très utile et a permis de résoudre de nombreux problèmes, aussi bien d’exécution, que de cohérences (mauvais résultats …).

Pour terminer, en terme de test, JAVA nous propose JUnit afin de réaliser des tests unitaires efficaces, nous permet d’exécuter nos tests à la volée afin de vérifier que tout fonctionne parfaitement. Alors qu’en C, les bibliothèques de test ne sont pas abondantes, ou alors la doc n’est pas approfondie, et ces bibliothèques ne fournissent pas toujours les services dont on a besoin. SonarQube a aussi été utilisé en JAVA afin de mesurer la couverture du code du projet, mais aussi afin de trouver les points bloquants, critiques et majeurs du projet qui peuvent nuire (incohérences syntaxiques …). Cet outil nous a aidé à simplifier le code, à le nettoyer, et surtout le rendre plus performant en diminuant des problèmes de compléxité. SonarQube nous a ainsi aider à rendre notre projet robuste, avec une faible dette technique en terme de problèmes, une couverture de test supérieure à 80%, aucun problème critique, ni bloquants, et une compléxité qui a été réduite de moitié, comme l’on peut le voir ci-dessous :

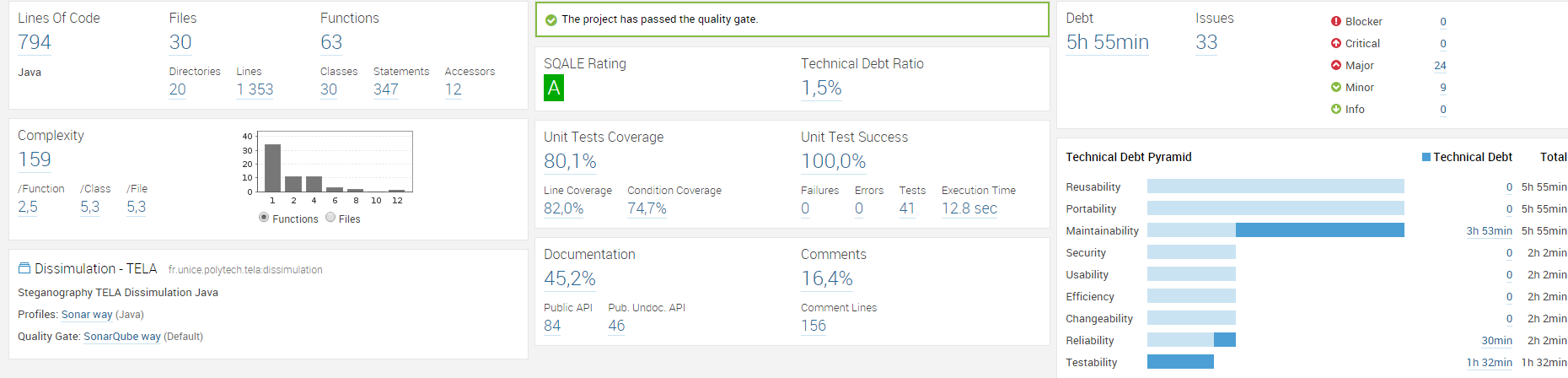


Figure : Tableau de bord Sonar

# Résultat des bancs de tests

Nous avons réalisé des bancs de tests afin de mesurer les performances de notre solution. Ces bancs de tests ont été réalisés selon trois critères : la taille de l’image, la taille de l’information à dissimuler et la puissance de la machine. Nous avons, pour cela, utilisé les images mises à disposition par NSA pour tester nos images. Nous y trouvons différentes résolutions d’images. Pour les informations à dissimuler, nous avons un …

# Conclusion

Ce projet a été intéressant sur différents points. Il a permis de nous montrer que sortir de sa zone de confort est une bonne expérience. Il nous faut s’adapter au plus vite afin de surmonter un problème et ça s’est vu lors de la création des équipes, où nous ne nous connaissions pas. Nous sommes aujourd’hui une très belle équipe avec beaucoup de potentiel et une bonne humeur qui y règne. Un des grands points fort de ce projet est le fait qu’ils soit bien aligné avec la matière de POO et de C, puisque les erreurs constatées dans ce projet sont souvent relayées au responsable de la matière POO puis expliqué en cours magistraux. De plus, le fait de réaliser un projet avec deux langages différents est une nouveauté pour beaucoup, puisque l’on aurait très bien pu réaliser ce projet avec un seul langage et le terminer pour autant.

Nous avons pu appliquer les différentes techniques apprises en cours de QGL dans ce projet, utilisé des outils comme Mockito, par exemple afin de créer des tests unitaires.

L’équipe support a été à notre écoute lors des nombreuses séances de projet, mais aussi sur Piazza qui est un outil très intéressant et devrait être, selon nous, intégré à la majorité des cours. Cela éviterait de rencontrer des problèmes d’enseignants submergés par les mails et répondre la veille d’un DS à un mail d’il y a deux semaines, mais aussi afin de répondre à des questions de manière générale et éviter une possible répétition d’une réponse.

Le seul « défaut » que l’on verrait serait d’avoir cette matière de projet à elle toute seule dans une UE, ce qui peut être dangereux pour certains. Cette matière n’est certes ni compensable, ni rattrapable, ce qui a créé un vent de panique lors des premières semaines avec l’attribution de nos premières notes. Il serait peut-être envisageable d’organiser des rattrapages pour les personnes étant vraiment très proches de la moyenne avec un mini projet totalement différent du central (vu qu’elles ont vraisemblablement raté ou mal compris le sujet de projet du semestre), ou alors les faire travailler sur une autre fonctionnalité encore qu’ils trouveraient eux même et qui soit raisonnable à réaliser en une semaine par exemple.

[Figure 1 : Schéma général du sujet 2](#_Toc439899080)

[Figure 2 :Hiérarchie des patterns 3](#_Toc439899081)

[Figure 3 : Hiérarchie de quelques exceptions 3](#_Toc439899082)

[Figure 4 : Diagramme de classe du LSB 4](#_Toc439899083)

[Figure 5 : Package de manipulation de classes 4](#_Toc439899084)

[Figure 6 : Tableau de bord Sonar 6](#_Toc439899085)