Université Nice SOPHIA ANTIPOLIS

JavInspector : un inspecteur d'applications Java

Lucas Charbit
Jérôme Gahide
Xavier Galbois
Gabriel Zerbib

Présentation de JavInspector

Un inspecteur non intrusif de classifieurs Java.

Une interface graphique conviviale et intuitive.

Une application finie, robuste et extensible.

Une application bien “écrite” et bien documentée.

Plan

I. Description de l'API
- Chargement de classes
- Analyse de classifieurs
- Module de recherche

II. L'interface graphique
- Présentation
- Conception (design pattern MVC)
- Extensibilité

III. Résultats - Extensions

Conclusion

I.1 Chargement des classes

- Analyse à partir d'un objet `Class`
- Pré requis :
  - Obtenir un objet `Class` représentant la classe chargée
  - Pouvoir charger une classe à partir de différents supports
- Choix d'implémentation :
  - Eviter de spécifier le répertoire de base des packages

I.1 Chargement de classes (2)

Obtention de l'objet Class

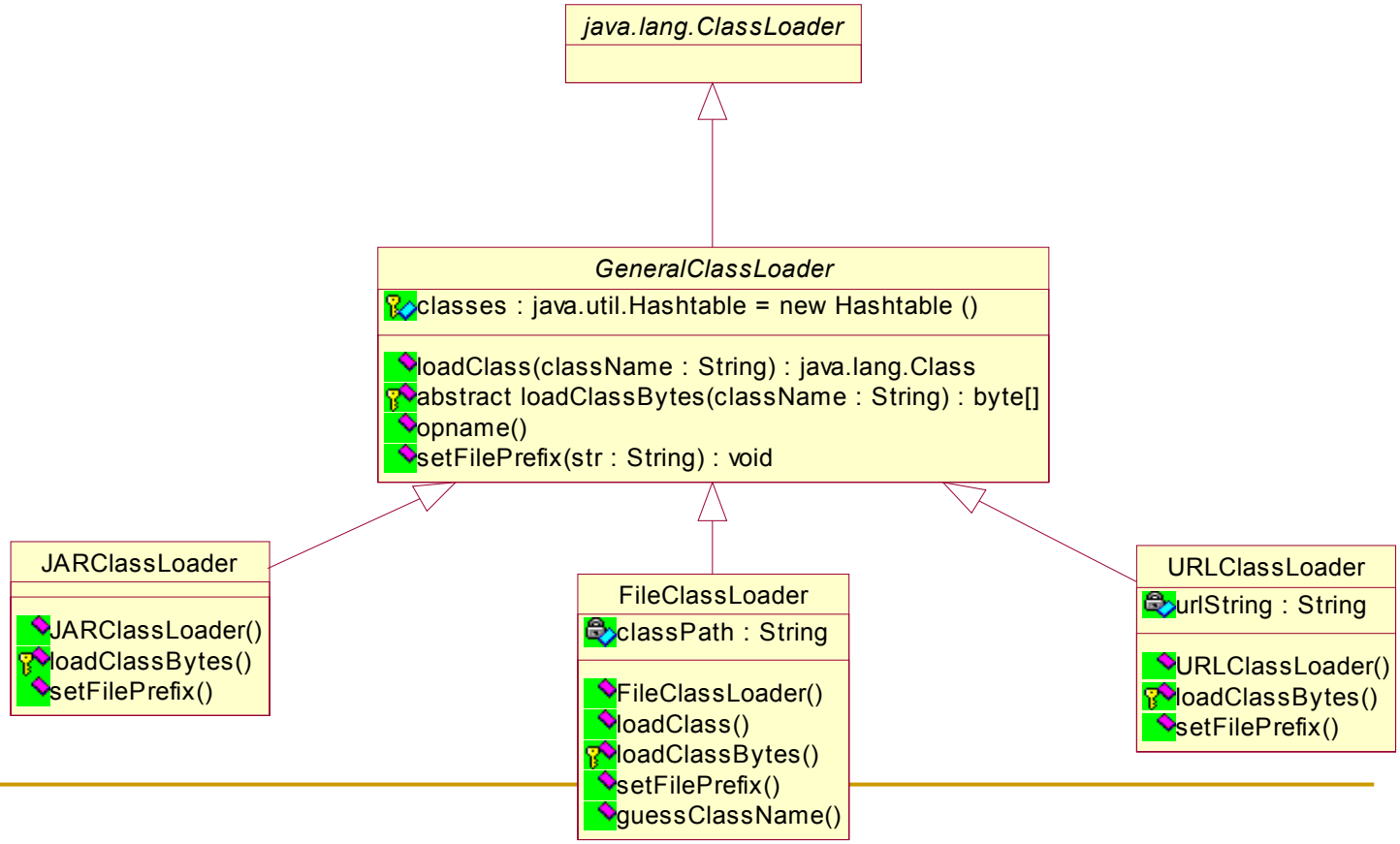
- Utilisation de `java.lang.ClassLoader`

Sources multiples (fichier, Archive, URL)

- Implémentation avec généricité

I.1 Chargement des classes (3)

Diagramme de classes



I.1 Chargement des classes (4)

Gestion des dépendances

On ne connaît pas le répertoire de base du package

- Déduire ce répertoire à partir du nom de la classe chargée (Valide pour la composition)

Impossible de charger la classe si on ne peut pas charger sa classe mère

- Connaître son nom avant le chargement
  - Extraire le nom du fichier '.class'
    - Structure trop complexe → Irréalisable

Solution : Itération sur l'arborescence du fichier jusqu'à réussite

I.1 Chargement de classes (5)

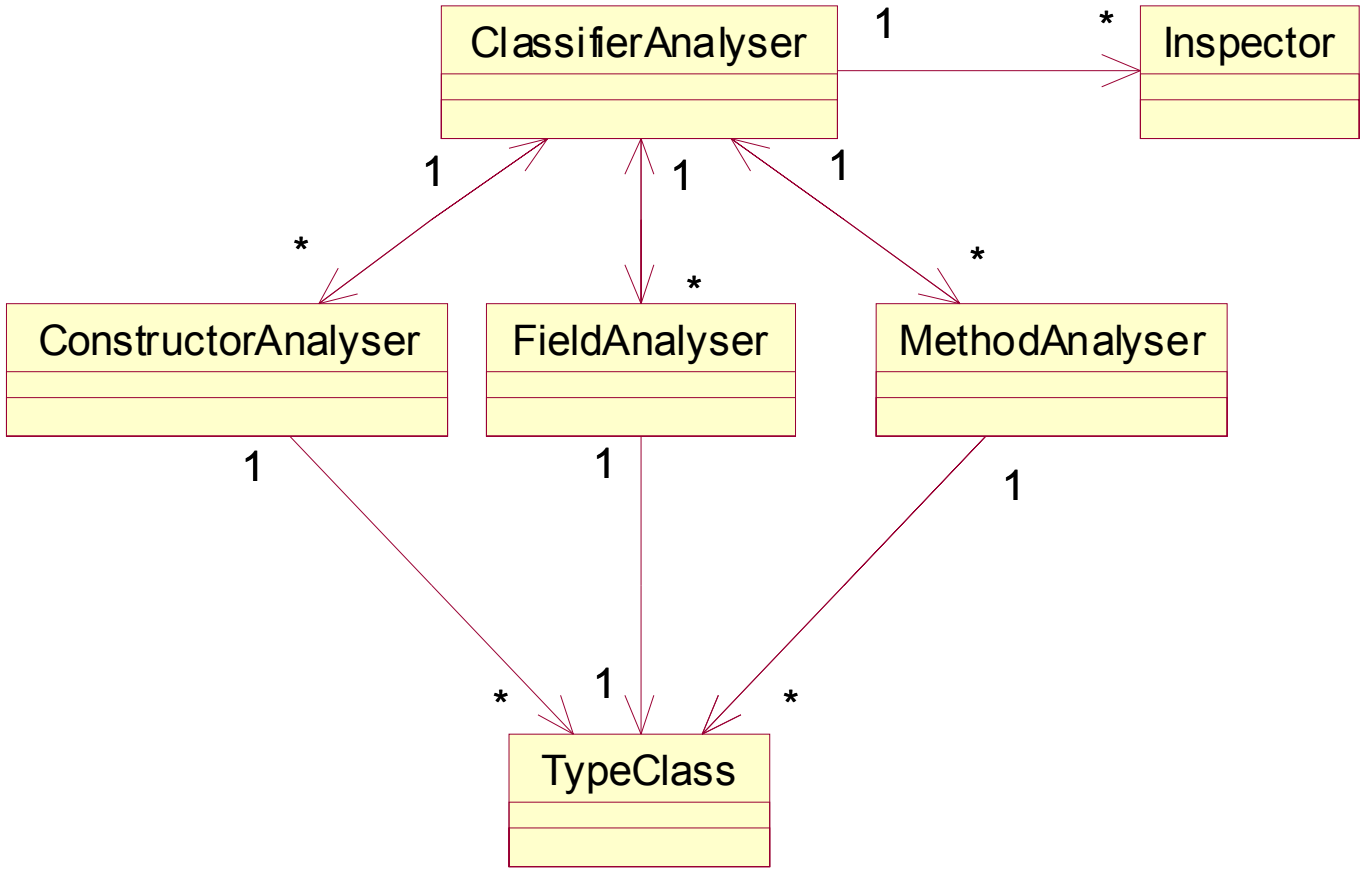
- Fonctionnalités :
  - Chargement à partir d'un fichier
    - Seul paramètre : le fichier '.class' à charger
    - Gestion complète des dépendances
  - Chargement à partir d'une Archive (JAR ou Zip)
    - Spécifier l'archive et le nom complet de la classe
    - Gestion complète des dépendances
  - Chargement à partir d'une URL
    - Dépendance limitée à un même package
    - Intérêt principal : inspection d'applets

I.1 Chargement des classes (6)

- Extensions possibles :
  - Chargement à partir d'une URL :
    - Gestion complète des dépendances
  - Actualisation d'une classe déjà chargée
    - Impossible avec le ClassLoader de java
      - Analyser directement le bytecode

I.2 L'API d'analyse de classifieurs

Diagramme de classes

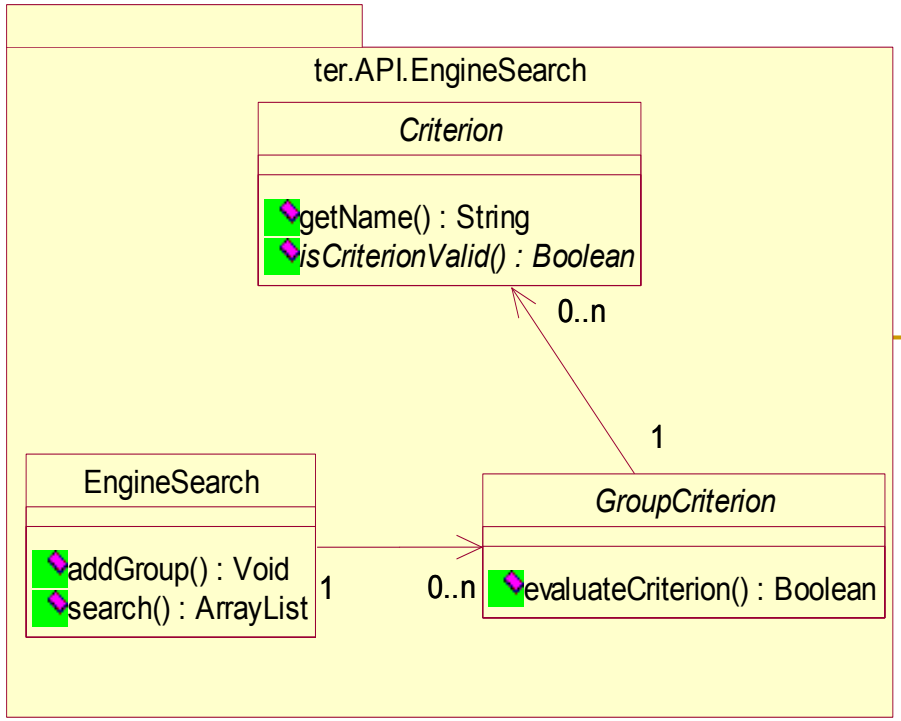


I.2 L'API d'analyse de classifieurs (2)

- Utililsation du package `java.lang.reflect`
- Fonctionnalités
  - Récupération des classes mères jusqu'a `Object`
  - Récupération des super interfaces
  - Récupération des champs, constructeurs et méthodes.
  - Récupération de la valeur des champs statiques
  - Récupération des classes internes et anonymes

I.3 Le moteur de recherche simple

Architecture générale

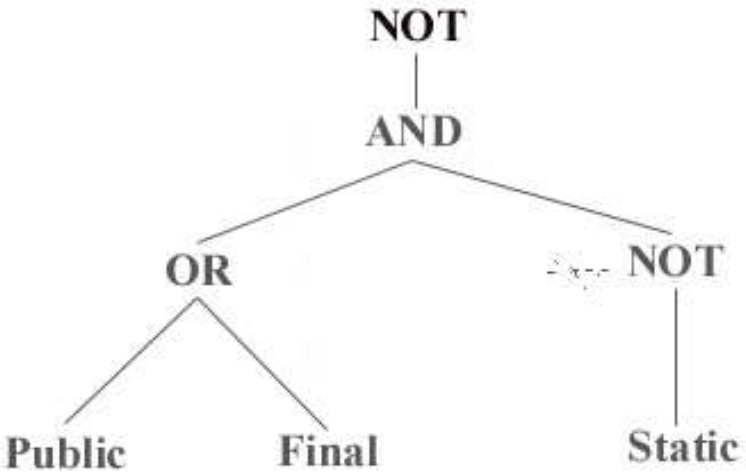


- Permet une recherche selon des critères.
- Organisation dans deux classes abstraites :
  - Criterion
  - GroupCriterion

- Un moteur (Engine Search) qui utilise les groupes de critères implémentés.
- Renvoie un résultat en fonction de l'évaluation des critères.
- Facilement extensible

I.3 Le moteur de recherche avancée

Basé sur une nouvelle idée d'execution d'une recherche : évaluation d'arbres d'expressions



! ((Public OR Final) AND NOT Static)

Screen shoot de la fenêtre de saisie d'expressions

But : permettre d'effectuer des recherches à partir de combinaisons logiques de critères.

I.3 Le moteur de recherche avancée (2)

Réutilisation des bases posées dans la recherche simple. (Criterion et GroupCriterion)

Outils implémentés pour cette recherche

- Utilisation d'un language d'expressions logiques et de la grammaire construite à partir de ce langage.
- Implémentation d'un analyseur lexico syntaxique et d'un analyseur sémantique pour cette grammaire.

II.1 L'interface graphique

- Difficultés
  - Encapsulation des données :
    - Séparation code/interface
  - Affichage des données
    - Quelles données afficher ?
    - Comment les afficher ?
    - Gestion des interactions
  - Extensibilité
    - Permettre le développement de nouveaux composants

II.2 Le design pattern MVC (Model, View, Controler)

Bien adapté :

- Séparation données (modèle)
- Plusieurs représentations graphiques des données (vues)
- Besoin de contrôler ces représentations (contrôleurs)

Particularités :

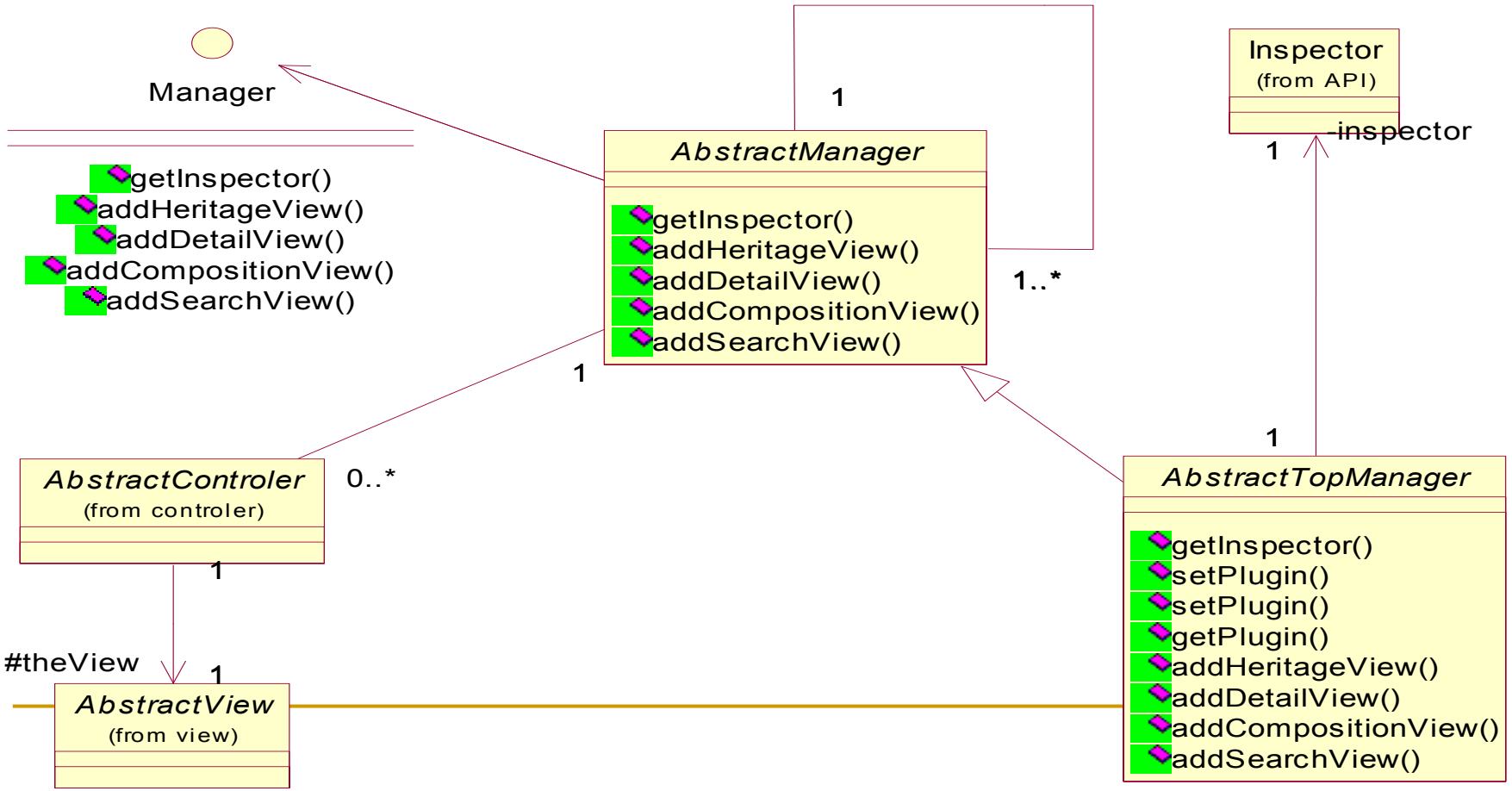
- Modèle unique et statique (ne change pas)
- Interactions multiples des vues (et des contrôleurs)

II.2 Les managers

- Buts
  - Regrouper les vues d'un même type et leurs contrôleurs
  - Faciliter l'ajout, la suppression et l'affichage de ces vues
  - Permettre la communication entre les contrôleurs
- Implémentation
  - Utilisation d'un manager central (Top Manager)
  - Architecture en couches
  - Délégation dans les méthodes

II.2 Les managers (2)

Diagramme de classes de l'architecture

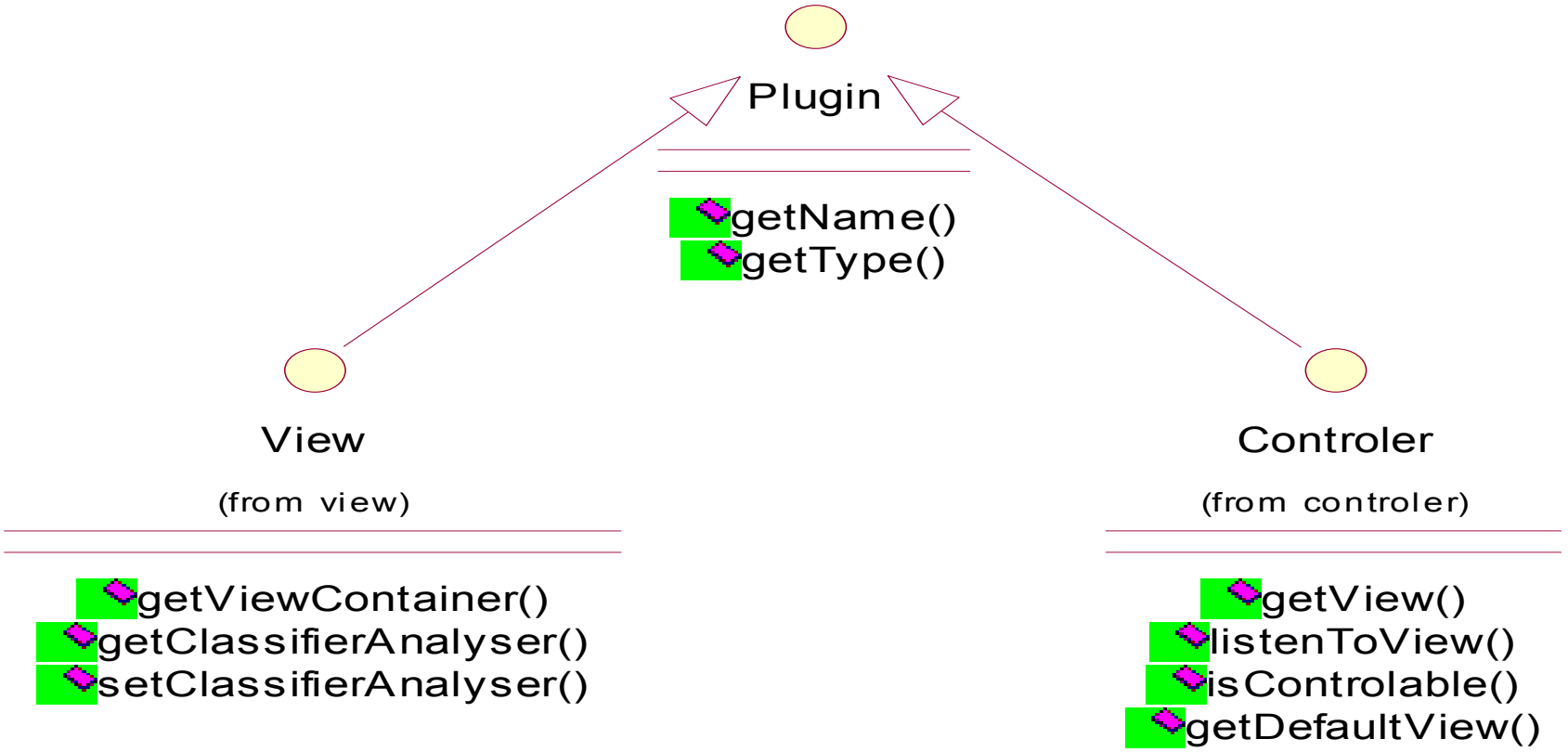


II.3 Extensibilité

- Buts
  - Permettre à l'utilisateur de paramétrer les vues
  - Permettre à l'utilisateur de développer de nouveaux composants
- Moyens
  - Les vues et les contrôleurs sont des plugins
  - Possibilité de les « clipper » sur l'application
  - Chargement dynamique dans un menu

II.3 Extensibilité (2)

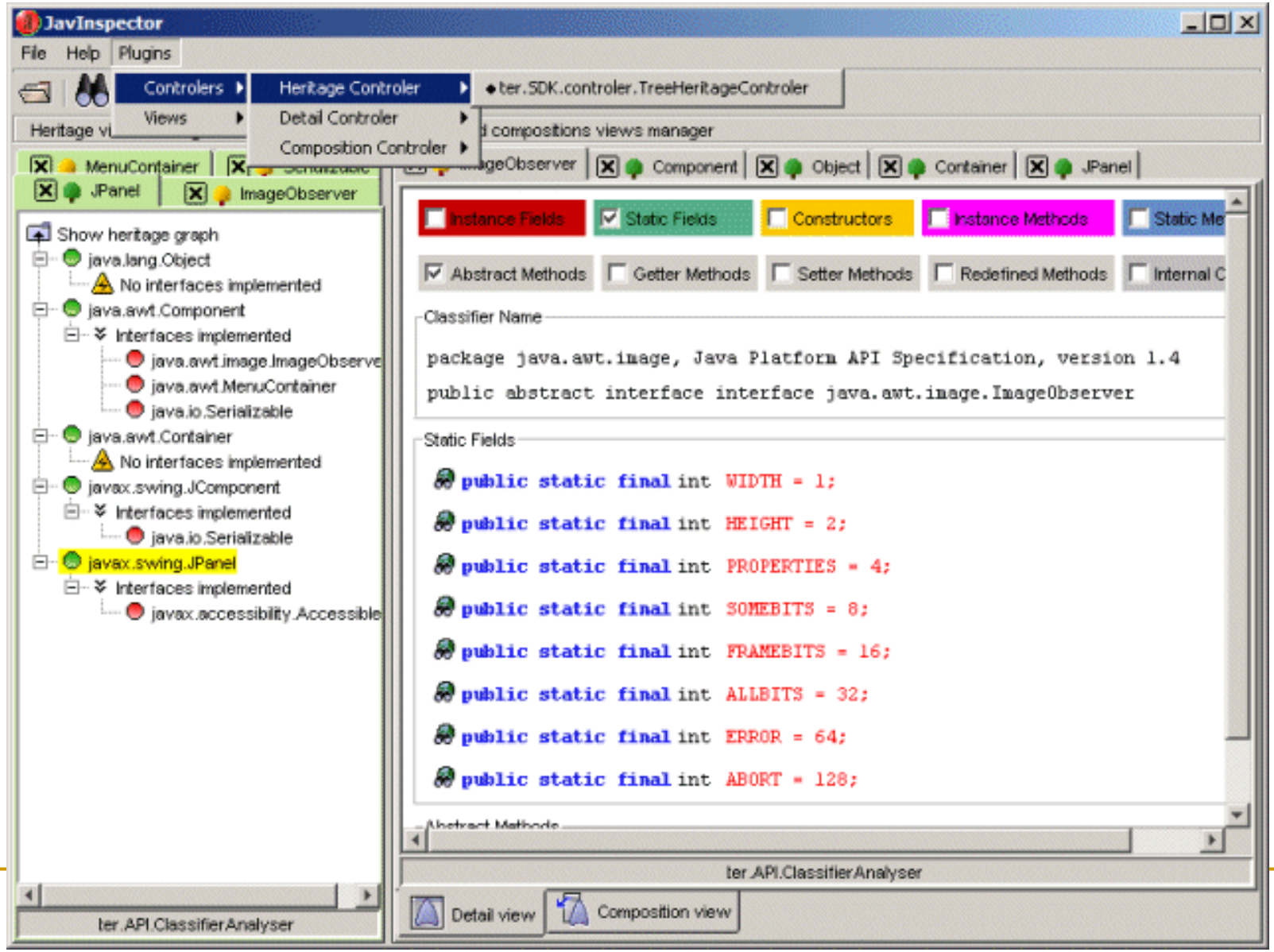
Voici le diagramme de classes de notre architecture de plugins :



III. Résultats - Extensions

- Résultats
  - Planning respecté dans son ensemble
  - Réalisation du projet dans son ensemble
  - Entièrement documenté
  - Fini et utilisable
  - Logiciel facilement extensible
- Extensions possibles
  - inspection de fichiers ‘.java’
  - analyse du byte code

III. Résultats - Extensions (2)



Conclusion

Travail intéressant et motivant

Utilisation des acquis de Licence/Maîtrise

Approfondissement de nos connaîssances en conception d'applications