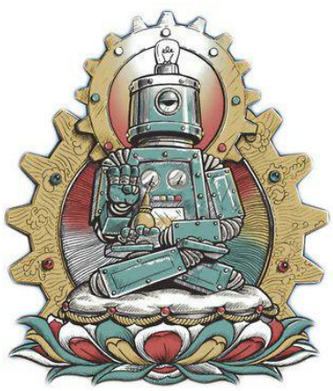
UNIVERSITÀ DEGLI STUDI DI SALERNO
CORSO DI INGEGNERIA DEL SOFTWARE

ASCETIC

Automated Code Smell Identification and Correction

OBJECT DESIGN DOCUMENT

VERSION 1.1



4 febbraio 2019

Coordinatore Progetto:

Nome	**Matricola**
Manuel De Stefano	0522500633

Partecipanti:

Nome	**Matricola**
Amoriello Nicola	0512104742
Di Dario Dario	0512104758
Gambardella Michele Simone	0512104502
Iovane Francesco	0512104550
Pascucci Domenico	0512102950
Patierno Sara	0512103460

Revision History:

Data	**Versione**	**Descrizione**	**Autore**
27/01/2019	1.0	Prima stesura	Tutto il Team
01/02/2019	1.1	Rifinitura struttura packages	Tutto il Team

Indice

1 Introduction

1.1 Object Design Trade-offs

Con la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi.
Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.
In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design.
Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo

Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del progetto.

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in modo da essere semplice, chiara e concisa, fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.

1.2 Guidelines for the Interfaces Documentation

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

- **Naming Convention**

  È buona norma utilizzare nomi:

  1. Descrittivi
  2. Pronunciabili
  3. Di uso comune
  4. Lunghezza media-corta
  5. Non abbreviati
  6. Evitando la notazione ungherese
  7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

- **Variables**

  1. I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una perriga e devono essere tutte allineate per facilitare la leggibilità.

     Esempio: smellName

  2. È inoltre possibile, in alcuni casi, utilizzare il carattere underscore "_", ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

     Esempio: MISPLACED_CLASS

- **Methods**

  1. I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola.
  2. Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto.
  3. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile().

4. Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitare la leggibilità.

   Esempio: getId(), setId()

5. I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

- **Classes**

1. I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo.

   Esempio: RefactoringManager.java

2. Ogni file sorgente java contiene una singola classe e dev'essere strutturato in un determinato modo. Al suo interno devono essere presenti:
   - Una breve introduzione alla classe. L'introduzione indica, oltre ad una breve descrizione del compito della classe, l'autore, la versione e la data di implementazione.
   - L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.
   - La dichiarazione di classe caratterizzata da:
     (a) Dichiarazione della classe pubblica
     (b) Dichiarazioni di costanti
     (c) Dichiarazioni di variabili di classe
     (d) Dichiarazioni di variabili d'istanza
     (e) Costruttore
     (f) Commento e dichiarazione metodi.

1.3 Definitions, acronyms e abbreviation

Acronims:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document

Abbreviations:

- DB: DataBase

Definitions:

- Code Smell: porzioni di codice scritte in maniera non ottimale, le quali non compromettono il funzionamento del sistema ma introducono debolezze di progettazione, riducendo la qualità complessiva del codice.

1.4 References

- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto Ascetic
- Documento RAD del progetto Ascetic

2 Packages

Il sistema utilizza il modello architetturale MVC: prevede un livello di memorizzazione dati, uno di controllo e uno di presentazione.
La gestione del sistema è quindi suddivisa in tre livelli:

- Presentation layer
- Business layer
- Data layer

Il package Ascetic contiene sottopackage che a loro volta inglobano classi atte alla realizzazione degli scopi del plug-in. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

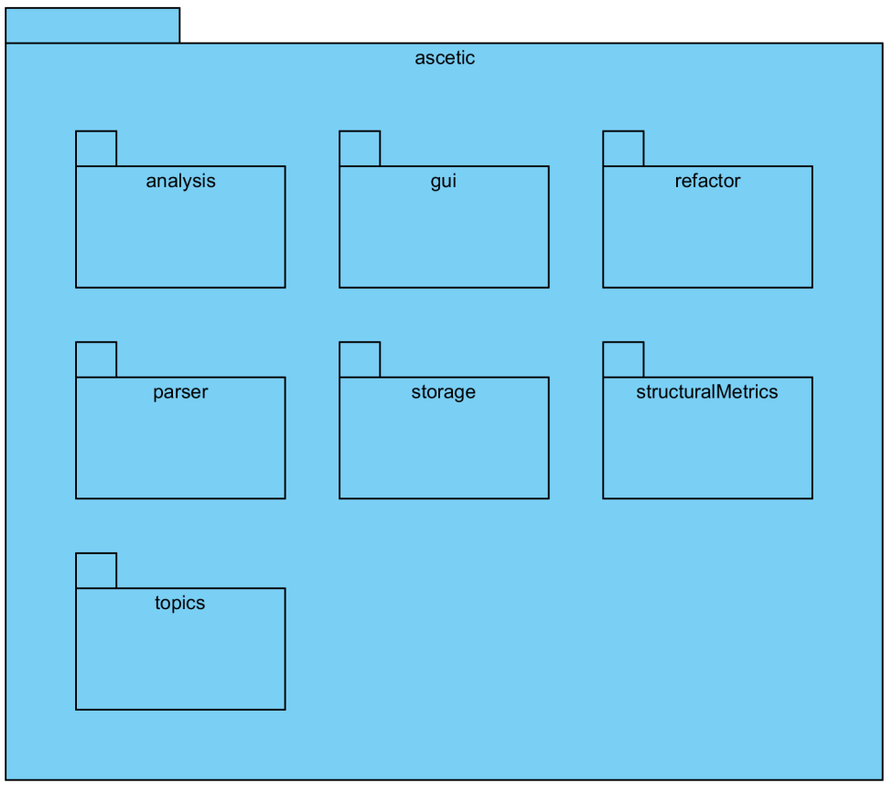
Presentation layer	Rappresenta l'interfaccia del sistema ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di immettere, in input, un progetto da analizzare che di visualizzare, in output, i dati prodotti.

Business Layer	Include tutti gli oggetti relativi all'elaborazione dei dati. Questo avviene interrogando il DB, tramite la componente Storage, per accedere a dati persistenti, cioè il file strutturato preso in analisi. Si occupa di varie gestioni, quali: 1. **Gestione Analisi** 2. **Gestione Refactoring** 3. **Gestione Code Parser** 4. **Gestione Topic** 5. **Gestione Metriche di qualità**

Data Layer	Ha il compito di memorizzare i dati utili al sistema, ossia il file contenente il codice sorgente da prendere in esame e il project structured file, ossia una componente che implementa una meccanica di cache realizzata in SQLite come singolo DB di cache per ogni progetto.

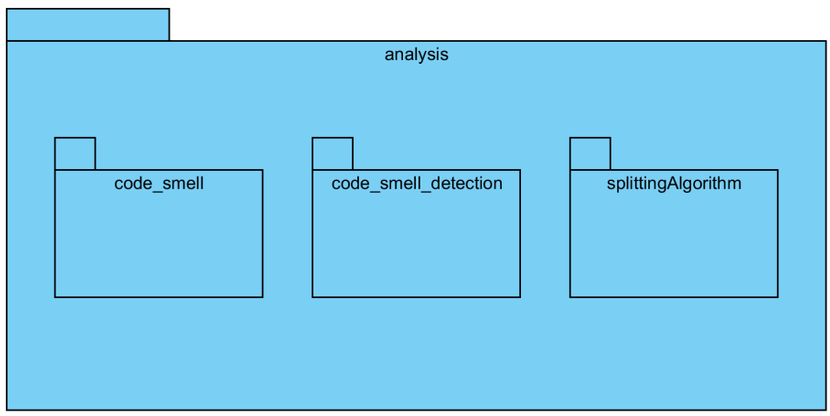
2.1 Package ascetic

Il package ascetic contiene i seguenti sottopackage:

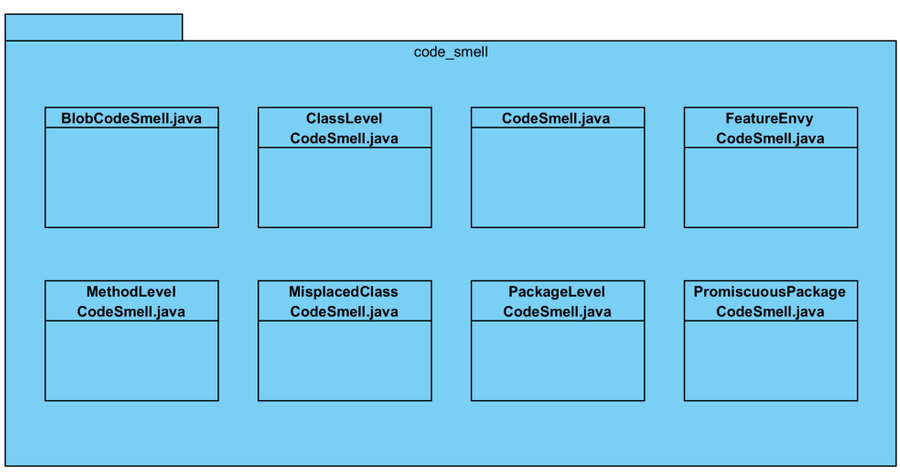


2.1.1 Package analysis

Il sottopackage analysis si suddivide a sua volta in tre sottopackage, ossia:



2.1.1.1 Package code_smell

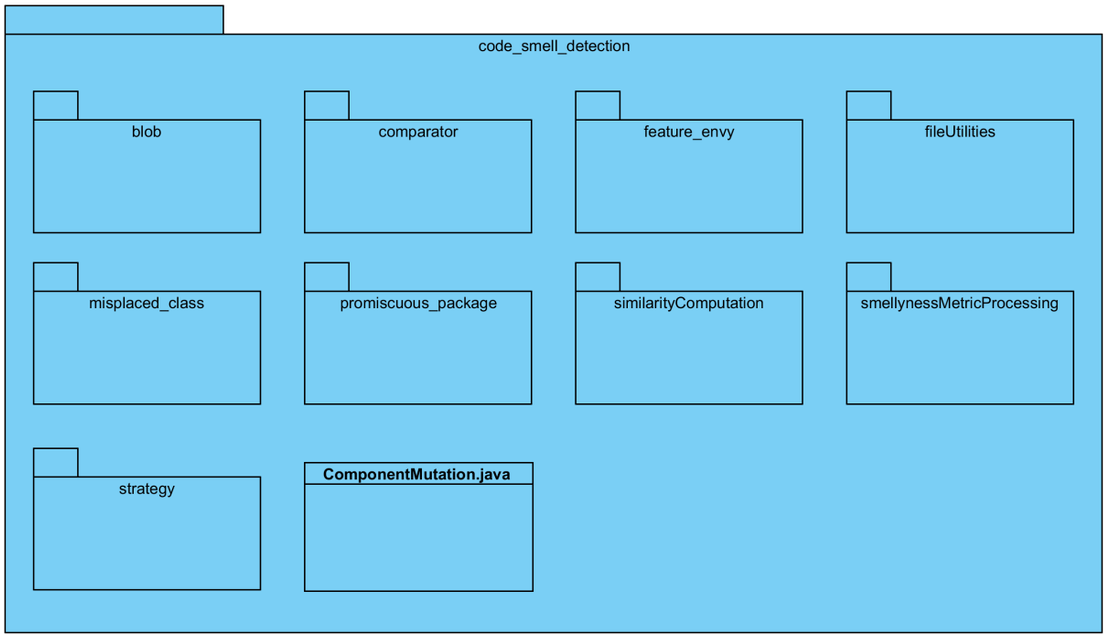


Di seguito è riportata la tabella delle descrizioni di ogni classe appartenente al package code_smell:

Classe	Descrizione
BlobCodeSmell.java	Permette di istanziare il code smell Blob.
ClassLevelCodeSmell.java	Permette di istanziare tutti gli oggetti di tipo CodeSmell che riguardano l'ambito delle classi.
CodeSmell.java	Permette di tenere traccia nel database dei code smell individuati durante l'analisi del codice.
FeatureEnvyCodeSmell.java	Permette di istanziare il code smell Feature Envy.
MethodLevelCodeSmell.java	Permette di istanziare tutti gli oggetti di tipo CodeSmell che riguardano l'ambito dei metodi.
MisplacedClassCodeSmell.java	Permette di istanziare il code smell Misplaced Class.
PackageLevelCodeSmell.java	Permette di istanziare tutti gli oggetti di tipo CodeSmell che riguardano l'ambito dei package.
PromiscuousPackageCodeSmell.java	Permette di istanziare il code smell Promiscuous Package.

2.1.1.2 Package code_smell_detection

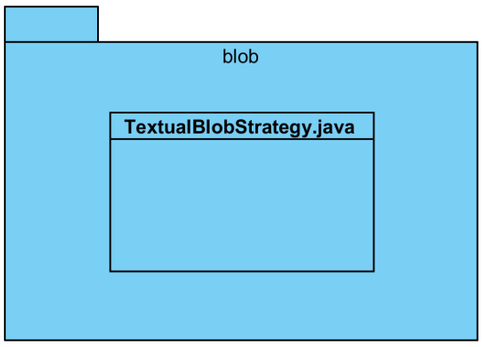
Il sottopackage code_smell_detection si suddivide a sua volta in altri nove sottopackage, ossia:



Di seguito è riportata la tabella della descrizione della classe appartenente al package code_smell_detection:

Classe	Descrizione
ComponentMutation.java	Permette di convertire un qualsiasi Bean in String, semplicemente inserendo il contenuto testuale completo del Bean in una stringa.

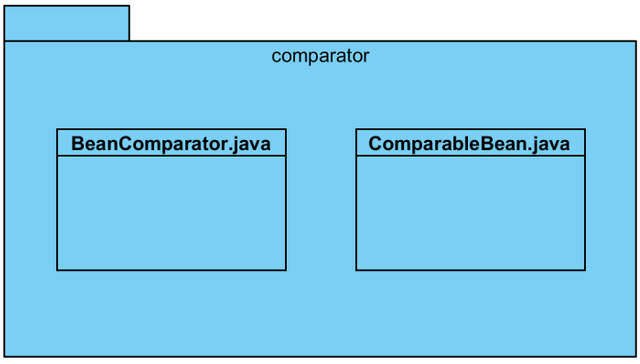
2.1.1.2.1 Package blob



Di seguito è riportata la tabella della descrizione della classe appartenente al package blob:

Classe	Descrizione
TextualBlobStrategy.java	Permette di rilevare attraverso il contenuto testuale, implementando lo Strategy adeguato, i code smell di tipo Blob.

2.1.1.2.2 Package comparator

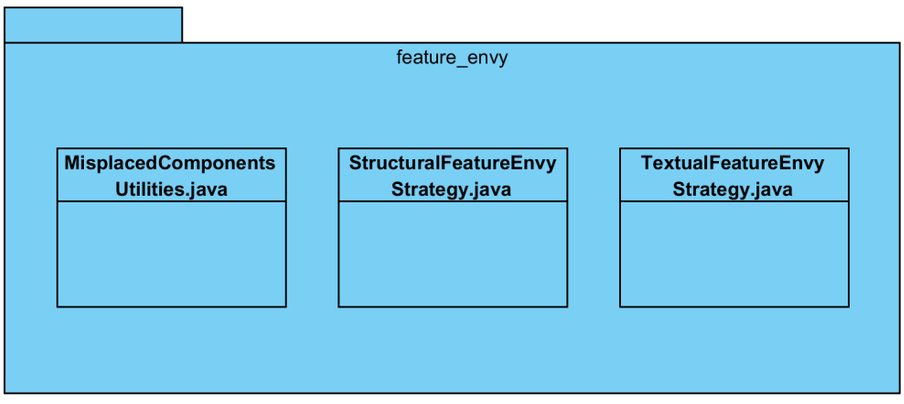


Di seguito sono riportate le tabelle delle descrizioni della classe e dell'interfaccia appartenenti al package comparator:

Classe	Descrizione
BeanComparator.java	Permette di comparare due Bean.

Interfaccia	Descrizione
ComparableBean.java	Dichiara il metodo che permette di trovare uguaglianze fra i Bean.

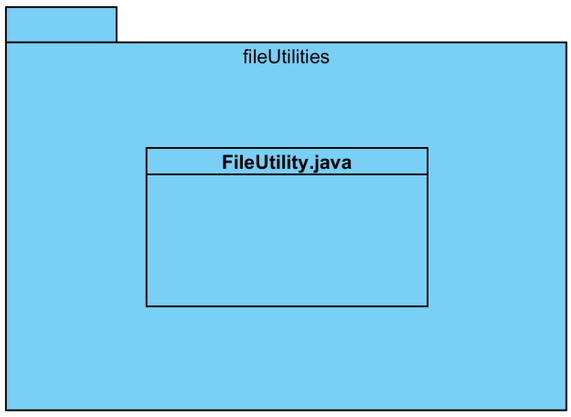
2.1.1.2.3 Package feature_envy



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package feature_envy:

Classe	Descrizione
MisplacedComponentsUtilities.java	Permette di identificare le possibili classi affette da code smell di tipo Feature Envy in base al calcolo del coseno.
StructuralFeatureEnvyStrategy.java	Permette di identificare, a livello strutturale, i code smell di tipo Feature Envy.
TextualFeatureEnvyStrategy.java	Permette di rilevare attraverso il contenuto testuale i code smell di tipo Feature Envy.

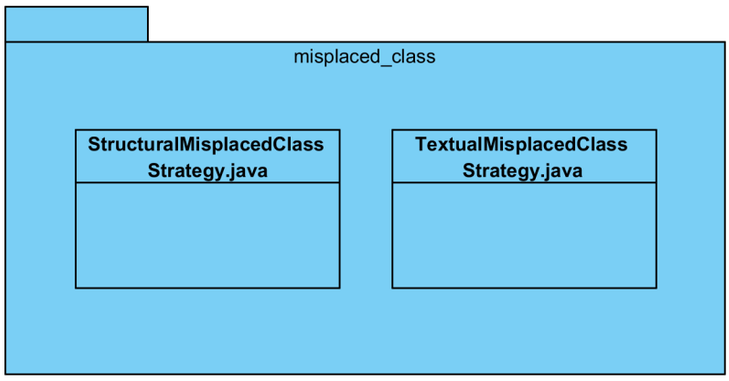
2.1.1.2.4 Package fileUtilities



Di seguito è riportata la tabella della descrizione della classe appartenente al package fileUtilities:

Classe	Descrizione
FileUtility.java	Permette di effettuare operazioni su file.

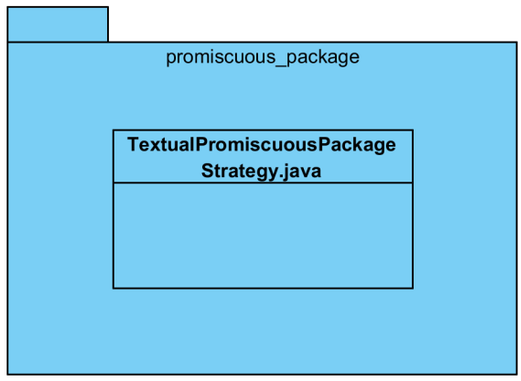
2.1.1.2.5 Package misplaced_class



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package misplaced_class:

Classe	Descrizione
StructuralMisplacedClassStrategy.java	Permette di identificare, a livello strutturale, i code smell di tipo Misplaced Class.
TextualMisplacedClassStrategy.java	Permette di rilevare attraverso il contenuto testuale i code smell di tipo Misplaced Class.

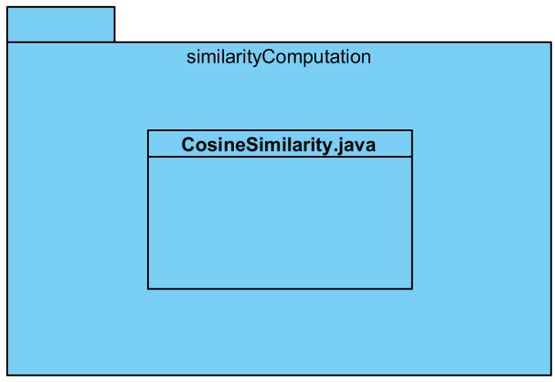
2.1.1.2.6 Package promiscuous_package



Di seguito è riportata la tabella della descrizione della classe appartenente al package promiscuous_package:

Classe	Descrizione
TextualPromiscuousPackageStrategy.java	Permette di rilevare attraverso il contenuto testuale i code smell di tipo Promiscuous Package.

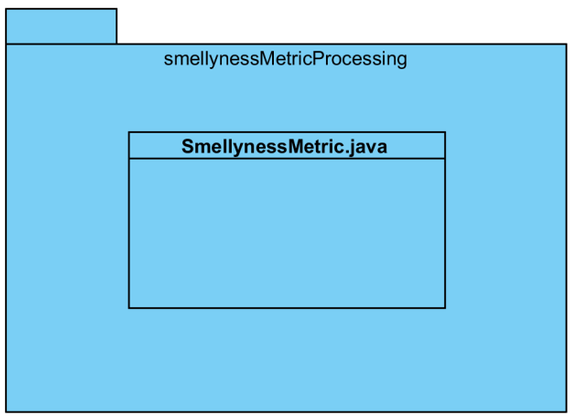
2.1.1.2.7 Package similarityComputation



Di seguito è riportata la tabella della descrizione della classe appartenente al package similarityComputation:

Classe	Descrizione
CosineSimilarity.java	Permette di calcolare la differenza fra due Bean attraverso il calcolo del coseno per verificare la presenza di un code smell.

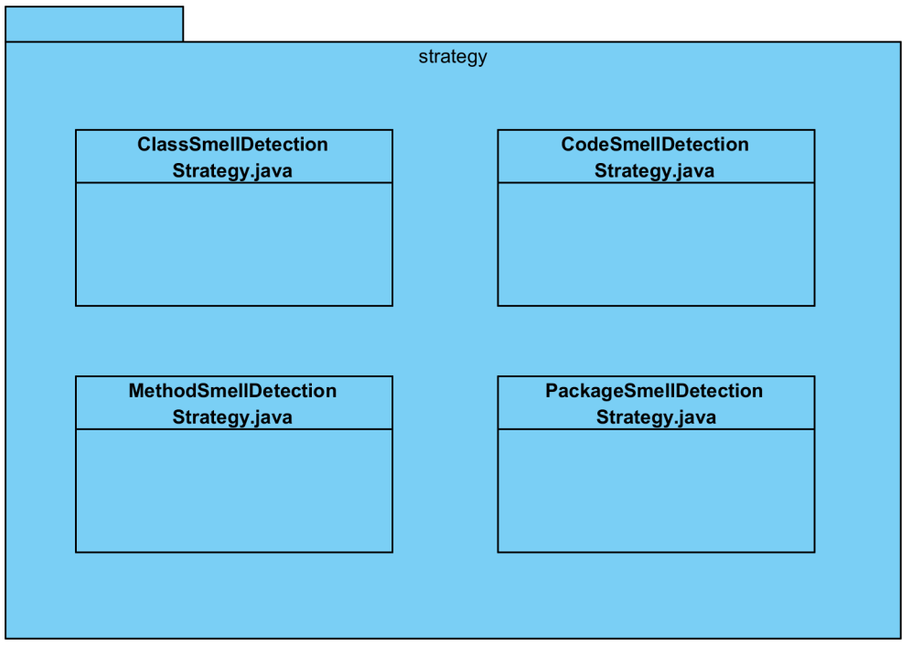
2.1.1.2.8 Package smellynessMetricProcessing



Di seguito è riportata la tabella della descrizione della classe appartenente al package smellynessMetricProcessing:

Classe	Descrizione
SmellynessMetric.java	Permette di calcolare le metriche all'interno del testo di un Bean.

2.1.1.2.9 Package strategy

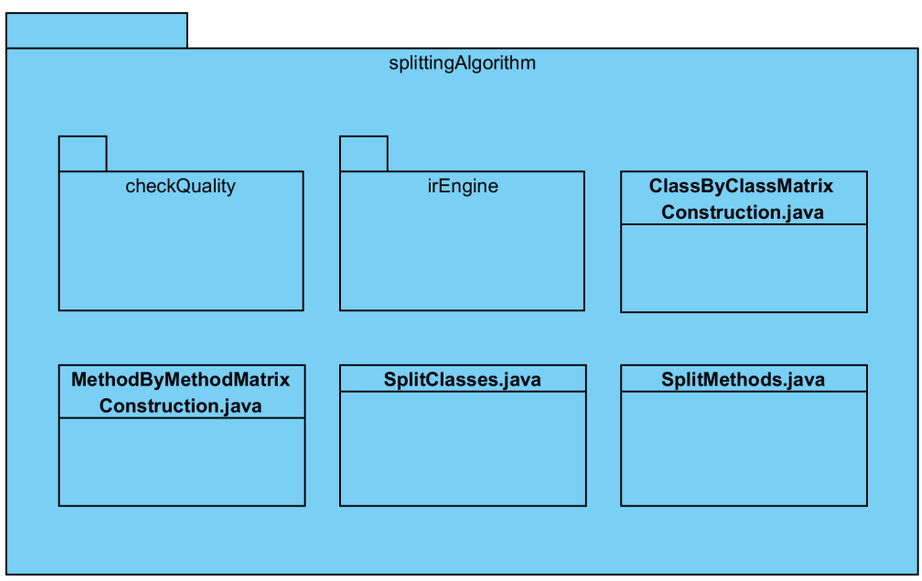


Di seguito è riportata la tabella delle descrizioni delle interfacce appartenenti al package strategy:

Interfaccia	Descrizione
ClassSmellDetectionStrategy.java	Dichiara i metodi da implementare negli Strategy addetti al rilevamento di code smell nelle classi.
CodeSmellDetectionStrategy.java	Dichiara i metodi da implementare negli Strategy.
MethodSmellDetectionStrategy.java	Dichiara i metodi da implementare negli Strategy addetti al rilevamento di code smell nei metodi.
PackageSmellDetectionStrategy.java	Dichiara i metodi da implementare negli Strategy addetti al rilevamento di code smell nei package.

2.1.1.3 Package splittingAlgorithm

Il sottopackage splittingAlgorithm si suddivide a sua volta in un altro sottopackage, ossia:

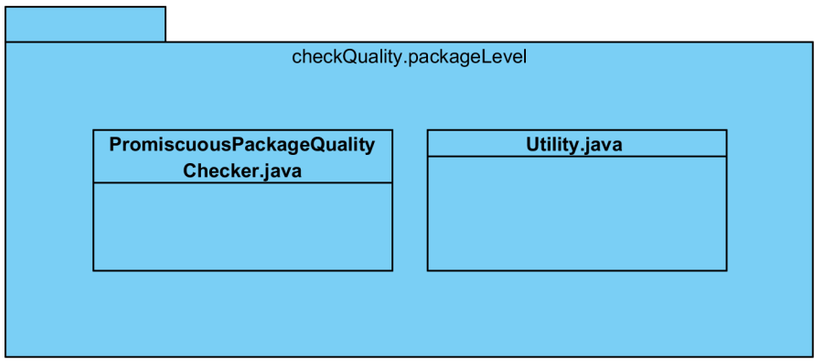


Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package splittingAlgorithm:

Classe	Descrizione
ClassByClassMatrixConstruction.java	Permette di creare una matrice classe per classe.
MethodByMethodMatrixConstruction.java	Permette di creare una matrice metodo per metodo.
SplitClasses.java	Permette di dividere una classe in più classi.
SplitMethods.java	Permette di dividere un metodo in più metodi.

2.1.1.3.1 Package packageLevel

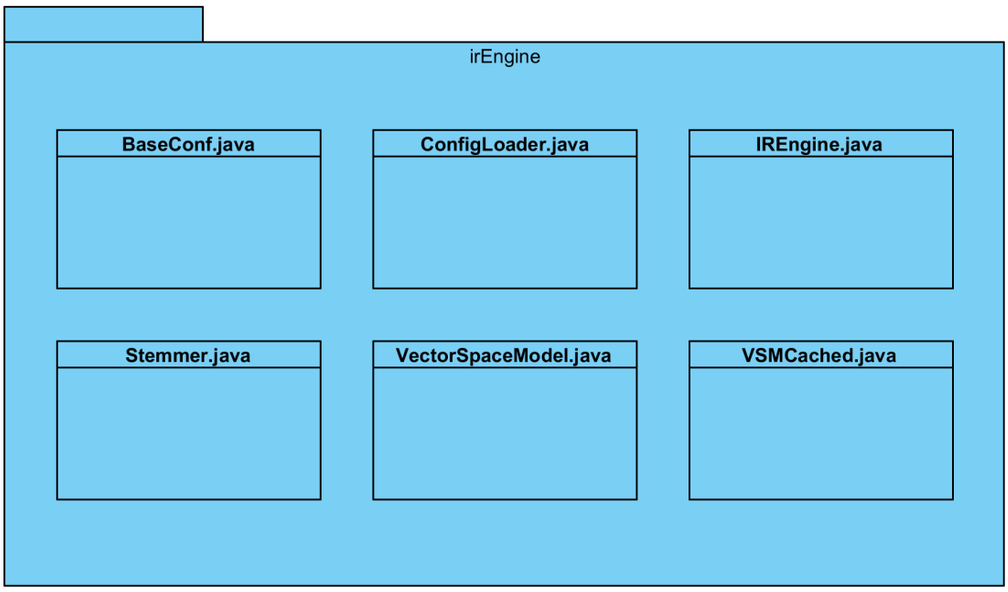
Il sottopackage packageLevel è contenuto nel package checkQuality:



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package packageLevel:

Classe	Descrizione
PromiscuousPackageQualityChecker.java	Permette di calcolare delle metriche su package per controllarne la qualità.
Utility.java	Permette di effettuare operazioni su file.

2.1.1.3.2 Package irEngine



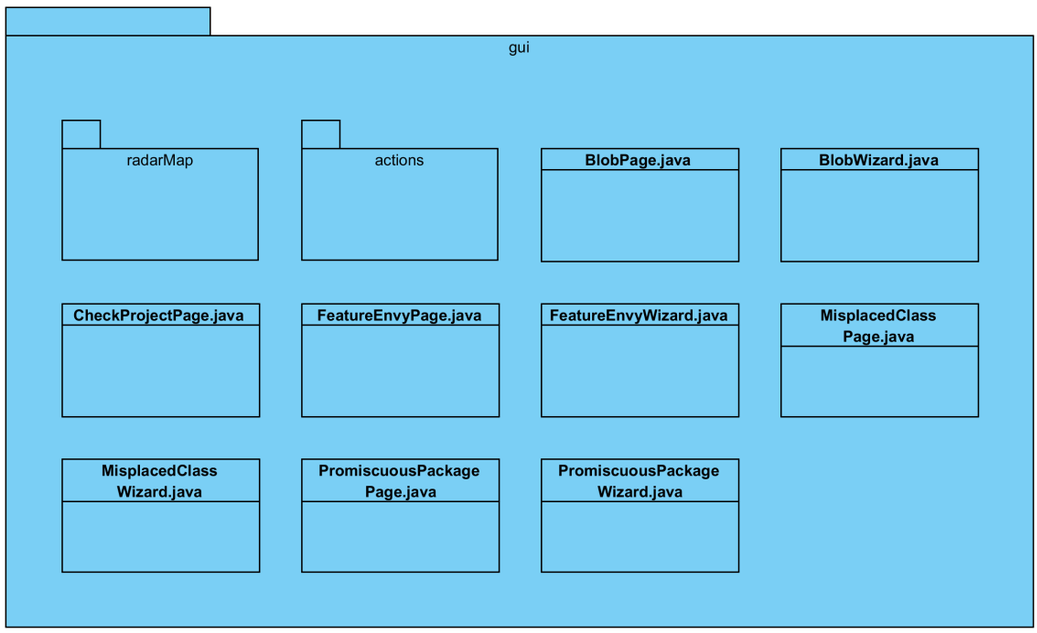
Di seguito è riportata la tabella delle descrizioni delle classi e dell'interfaccia appartenenti al package irEngine:

Classe	Descrizione
BaseConf.java	Permette di creare la configurazione standard del plug-in.
ConfigLoader.java	Permette di caricare il file della configurazione standard del plug-in.
Stemmer.java	Permette di ridurre termini derivati alle parole da cui derivano.
VectorSpaceModel.java	Permette di effettuare operazioni di conteggio e di estrazione dei termini dai documenti contenenti matrici di termini e di generare quest'ultime.
VSMCached.java	Permette di, estendendo VectorSpaceModel, effettuare operazioni sull'istanza dell'oggetto matrice.

Interfaccia	Descrizione
IREngine.java	Dichiara i metodi per la creazione di documenti contenente matrici di termini e per il calcolo delle somiglianze fra documenti.

2.1.2 Package gui

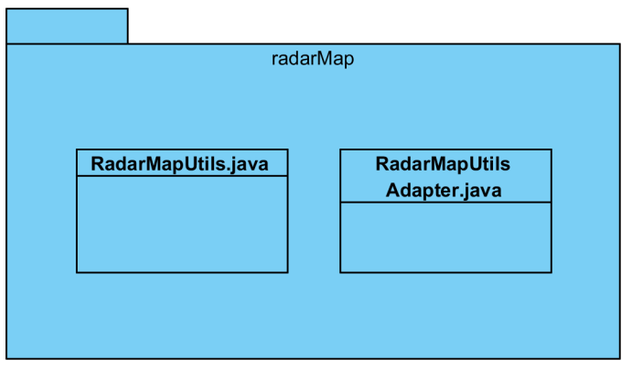
Il package gui contiene i seguenti sottopackage:



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package gui:

Classe	Descrizione
BlobPage.java	Permette di visualizzare la pagina di analisi dell'elemento affetto da code smell Blob.
BlobWizard.java	Permette di visualizzare la pagina di refactoring dell'elemento affetto da code smell Blob.
CheckProjectPage.java	Permette di avviare la schermata principale del plug-in.
FeatureEnvyPage.java	Permette di visualizzare la pagina di analisi dell'elemento affetto da code smell Feature Envy.
FeatureEnvyWizard.java	Permette di visualizzare la pagina di refactoring dell'elemento affetto da code smell Feature Envy.
MisplacedClassPage.java	Permette di visualizzare la pagina di analisi dell'elemento affetto da code smell Misplaced Class.
MisplacedClassWizard.java	Permette di visualizzare la pagina di refactoring dell'elemento affetto da code smell Misplaced Class.
PromiscuousPackagePage.java	Permette di visualizzare la pagina di analisi dell'elemento affetto da code smell Promiscuous Package.
PromiscuousPackageWizard.java	Permette di visualizzare la pagina di refactoring dell'elemento affetto da code smell Promiscuous Package.

2.1.2.1 Package radarMap

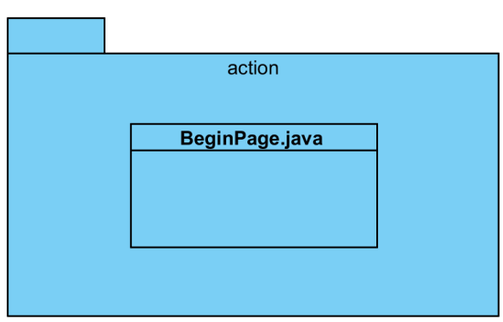


Di seguito è riportata la tabella delle descrizioni della classe e dell'interfaccia appartenenti al package radarMap:

Classe	Descrizione
RadarMapUtilsAdapter.java	Permette di creare, attraverso le metriche, le radar map dei Bean.

Interfaccia	Descrizione
RadarMapUtils.java	Dichiara i metodi per la creazione delle radar map.

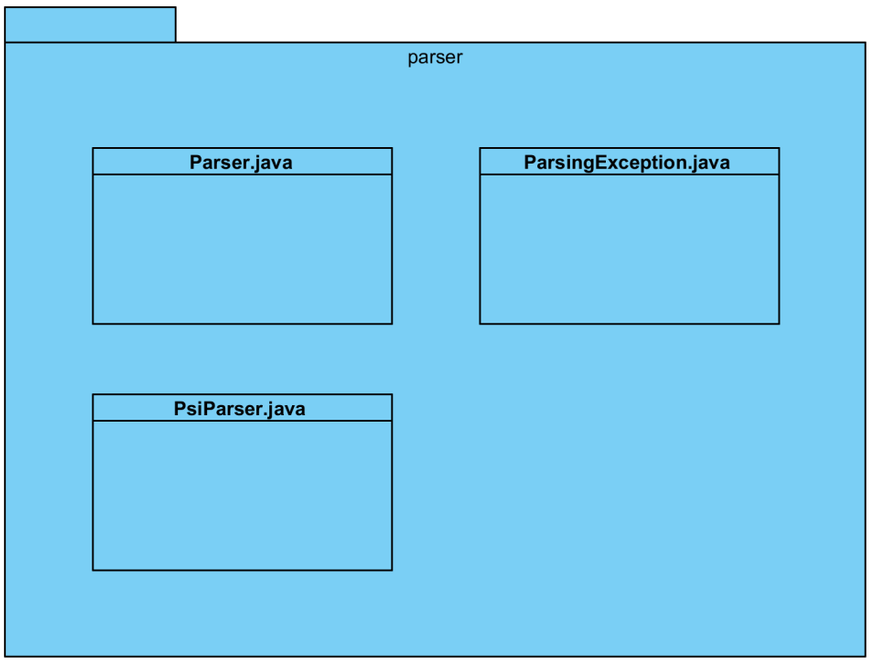
2.1.2.2 Package action



Di seguito è riportata la tabella della descrizione della classe appartenente al package action:

Classe	Descrizione
BeginPage.java	Permette di avviare l'esecuzione del plug-in.

2.1.3 Package parser



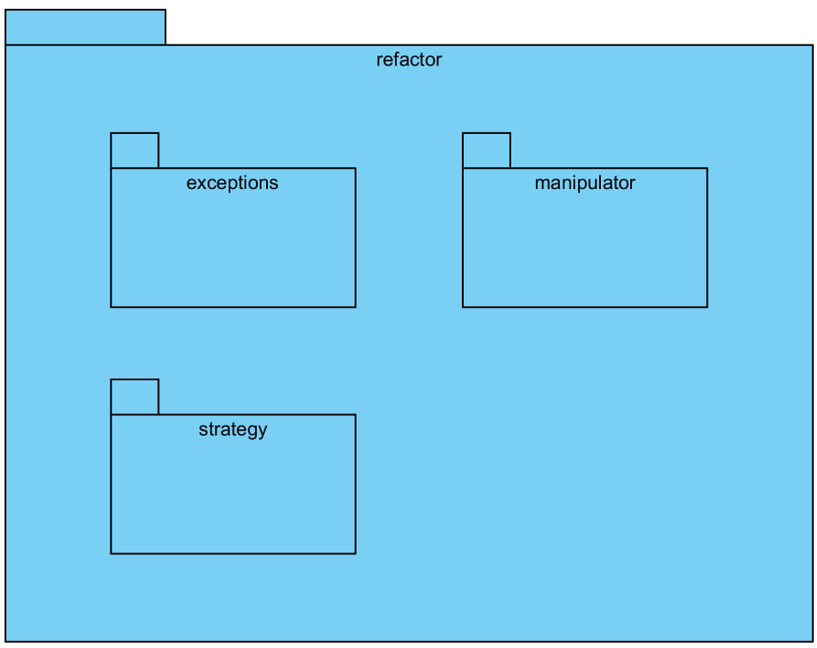
Di seguito è riportata la tabella delle descrizioni delle classi e dell'interfaccia appartenenti al package parser:

Classe	Descrizione
ParsingException.java	Permette di definire l'eccezione lanciata dal Parser.
PsiParser.java	Permette di effettuare la conversione da Psi a Bean.

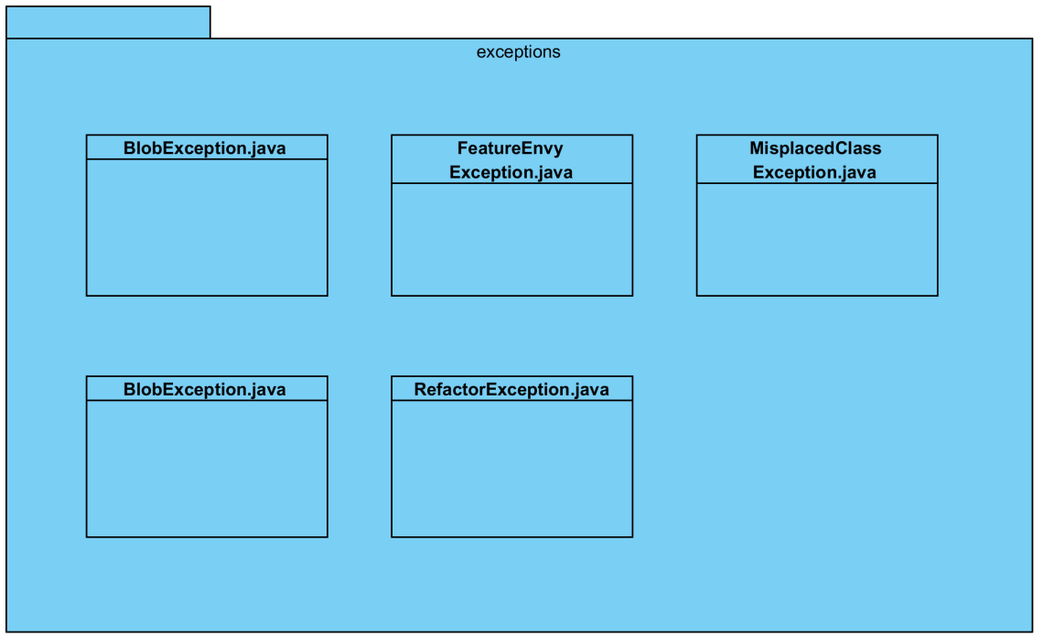
Interfaccia	Descrizione
Parser.java	Dichiara il metodo che avvia il processo di conversione da Psi a Bean.

2.1.4 Package refactor

Il sottopackage refactor si suddivide a sua volta in altri tre sottopackage, ossia:



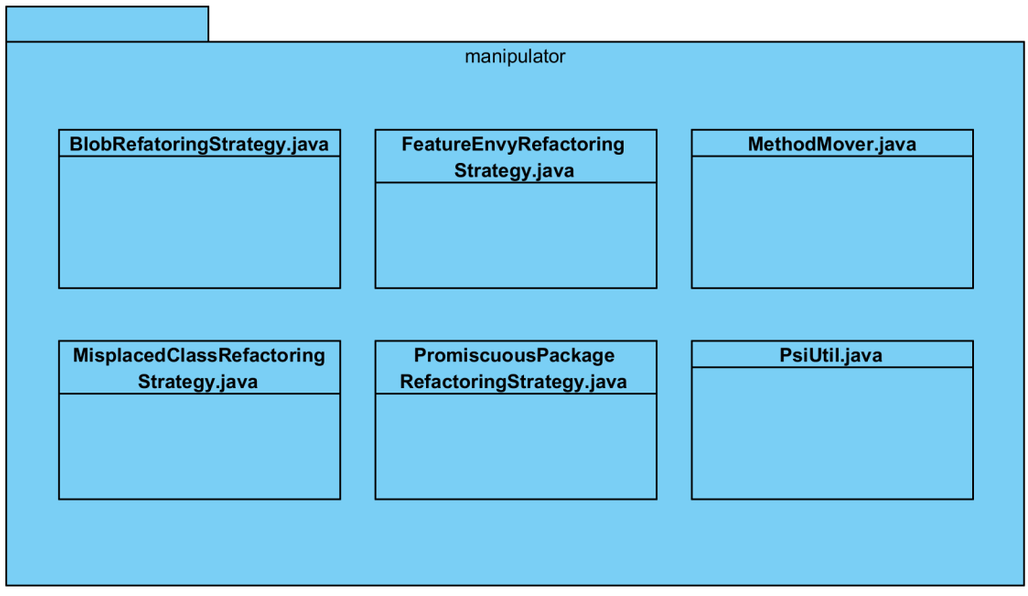
2.1.4.1 Package exceptions



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package exceptions:

Classe	Descrizione
BlobException.java	Permette di definire l'eccezione lanciata dallo Strategy riguardante il code smell Blob.
FeatureEnvyException.java	Permette di definire l'eccezione lanciata dallo Strategy riguardante il code smell Feature Envy.
MisplacedClassException.java	Permette di definire l'eccezione lanciata dallo Strategy riguardante il code smell Misplaced Class.
PromiscuousPackageException.java	Permette di definire l'eccezione lanciata dallo Strategy riguardante il code smell Promiscuous Package.
RefactorException.java	Permette di definire l'eccezione estesa dalle altre Exception già elencate.

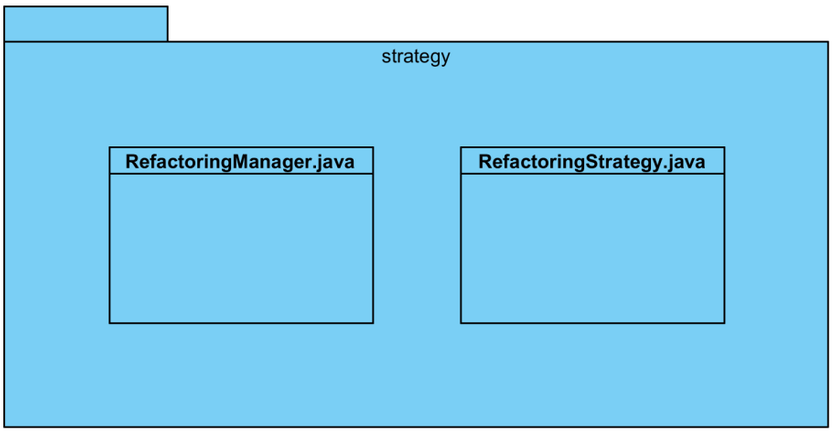
2.1.4.2 Package manipulator



Di seguito è riportata la tabella delle descrizioni delle classi appartenenti al package manipulator:

Classe	Descrizione
BlobRefatoringStrategy.java	Permette di effettuare il Refactor degli smell di tipo Blob.
FeatureEnvyRefactoringStrategy.java	Permette di effettuare il Refactor degli smell di tipo Feature Envy.
MethodMover.java	Permette di muovere un metodo in un'altra classe.
MisplacedClassRefactoringStrategy.java	Permette di effettuare il Refactor degli smell di tipo Misplaced Class.
PromiscuousPackageRefactoringStrategy.java	Permette di effettuare il Refactor degli smell di tipo Promiscuous Package.
PsiUtil.java	Permette di effettuare la conversione da Psi a Bean e viceversa.

2.1.4.3 Package strategy



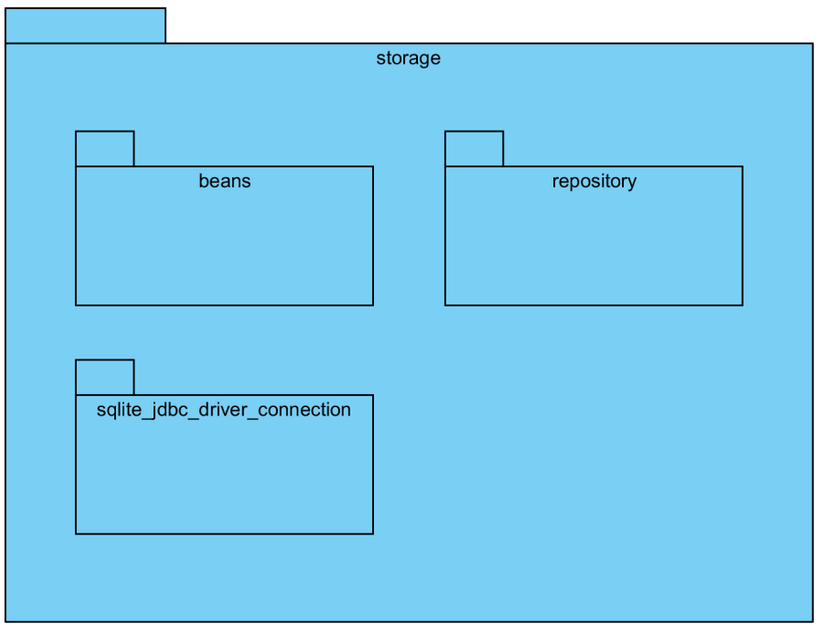
Di seguito sono riportate le tabelle delle descrizioni della classe e dell'interfaccia appartenenti al package strategy:

Classe	Descrizione
RefactoringManager.java	Permette di effettuare il Refactor.

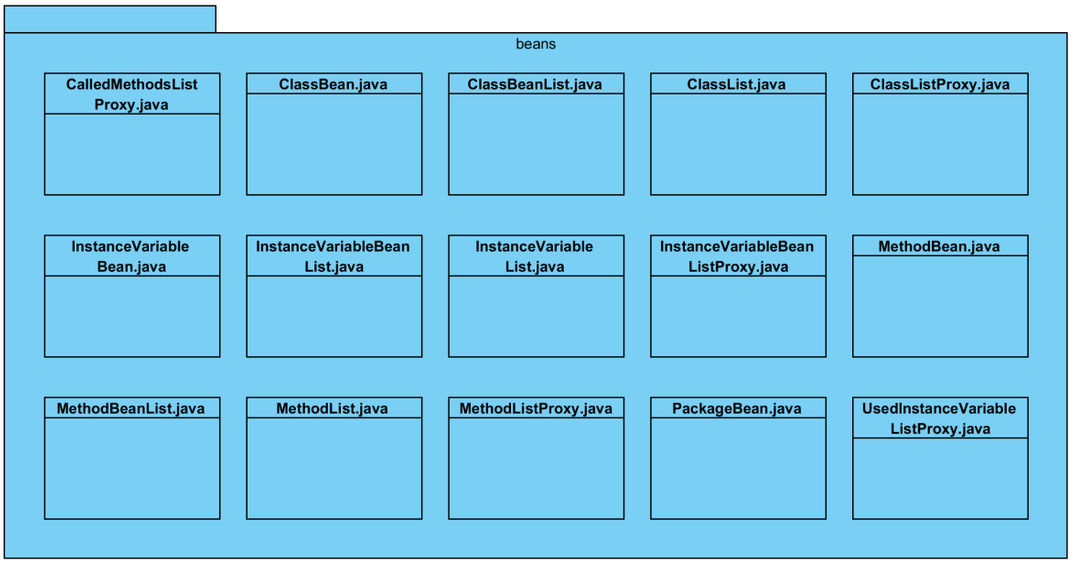
Interfaccia	Descrizione
RefactoringStrategy.java	Dichiara il metodo che effettua il Refactor, richiamato nella classe RefactoringManager.

2.1.5 Package storage

Il sottopackage storage si suddivide a sua volta in tre sottopackage, ossia:



2.1.5.1 Package beans

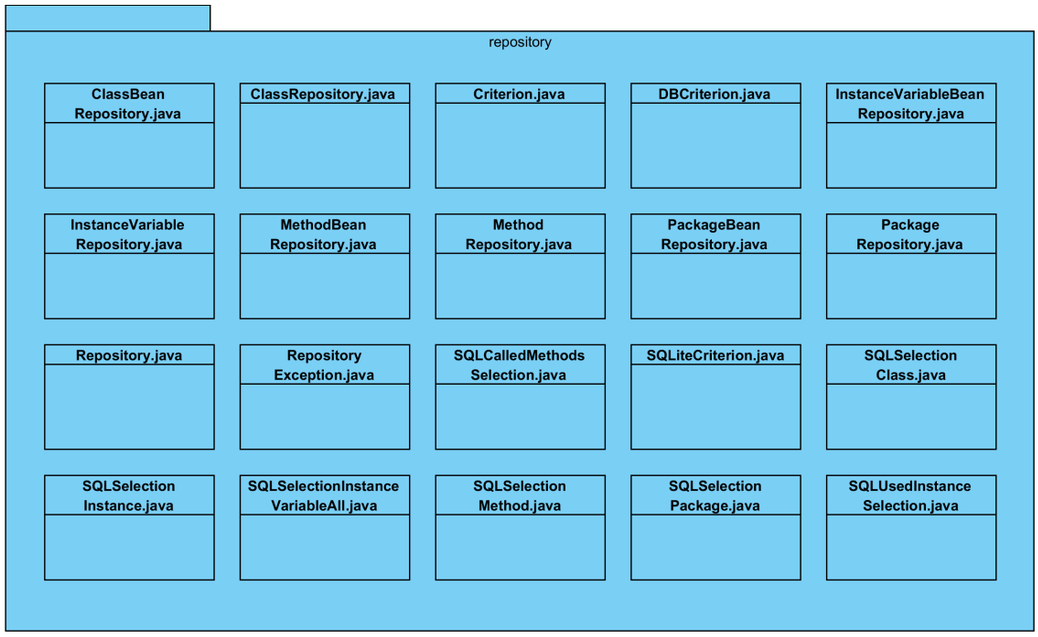


Di seguito sono riportate le tabelle delle descrizioni delle classi e delle interfacce appartenenti al package beans:

Classe	**Descrizione**
CalledMethodsListProxy.java	Permette di realizzare il lazy loading della lista dei metodi chiamata da un metodo.
ClassBean.java	Permette di istanziare una classe.
ClassList.java	Permette di istanziare una lista di classi.
ClassListProxy.java	Permette di realizzare il lazy loading per le classi.
InstanceVariableBean.java	Permette di istanziare una variabile di istanza.
InstanceVariableList.java	Permette di istanziare una lista di variabili di istanza.
InstanceVariableListProxy.java	Permette di realizzare il lazy loading per le variabili di istanza.
MethodBean.java	Permette di istanziare un metodo.
MethodList.java	Permette di istanziare una lista di metodi.
MethodListProxy.java	Permette di realizzare il lazy loading per i metodi.
PackageBean.java	Permette di istanziare un package.
UsedInstanceVariableListProxy.java	Permette di realizzare il lazy loading delle variabili di istanza usate da un metodo.

Interfaccia	**Descrizione**
ClassBeanList.java	Dichiara il metodo da implementare in ClassList e ClassListProxy.
InstanceVariableBeanList.java	Dichiara il metodo da implementare in InstanceVariableList e InstanceVariableList-Proxy.
MethodBeanList.java	Dichiara il metodo da implementare in MethodList e MethodListProxy.

2.1.5.2 Package repository

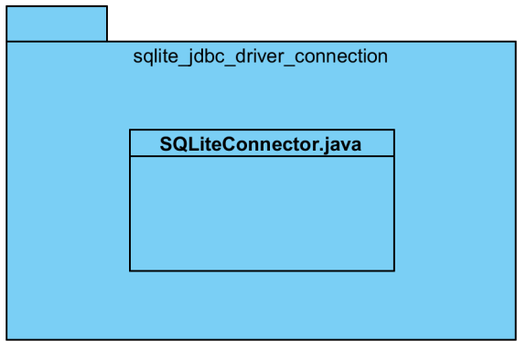


Di seguito sono riportate le tabelle delle descrizioni delle classi e delle interfacce appartenenti al package repository:

Classe	**Descrizione**
ClassRepository.java	Permette di salvare nel DB le classi.
DBCreation.java	Permette di creare il DB.
InstanceVariableRepository.java	Permette di salvare nel DB le variabili di istanza.
MethodRepository.java	Permette di salvare nel DB i metodi.
PackageRepository.java	Permette di salvare nel DB i package.
RepositoryException.java	Permette di definire l'eccezione lanciata dalle Repository.
SQLCalledMethodsSelection.java	Permette di istanziare la query da eseguire per la selezione della lista dei metodi chiamata da un metodo.
SQLSelectionClass.java	Permette di istanziare la query da eseguire per la selezione delle classi.
SQLSelectionInstance.java	Permette di istanziare la query da eseguire per la selezione delle variabili di istanza.
SQLSelectionInstanceVariableAll.java	Permette di istanziare la query da eseguire per la selezione di tutte le variabili di istanza.
SQLSelectionMethod.java	Permette di istanziare la query da eseguire per la selezione dei metodi.
SQLSelectionPackage.java	Permette di istanziare la query da eseguire per la selezione dei package.
SQLUsedInstanceSelection.java	Permette di istanziare la query da eseguire per la selezione delle variabili di istanza usate da un metodo.

Interfaccia	Descrizione
ClassBeanRepository.java	Dichiara i metodi da implementare nella Repository dedicata alle classi.
Criterion.java	Non dichiara nessun metodo, viene utilizzata per garantire manutenibilità.
InstanceVariableBeanRepository.java	Dichiara i metodi da implementare nella Repository dedicata alle variabili d'istanza.
MethodBeanRepository.java	Dichiara i metodi da implementare nella Repository dedicata ai metodi.
PackageBeanRepository.java	Dichiara i metodi da implementare nella Repository dedicata ai package.
Repository.java	Dichiara i metodi da implementare nelle interfacce dedicate alle Repository che la implementano.
SQLiteCriterion.java	Dichiara il metodo da implementare nelle classi che istanziano le query per la selezione nel DB.

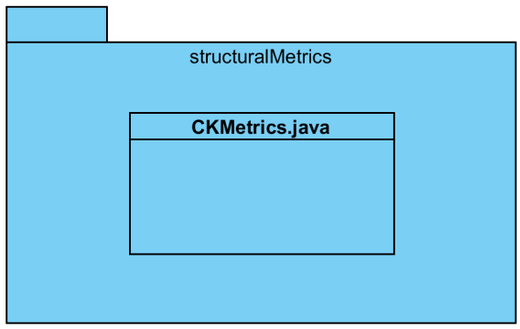
2.1.5.3 Package sqlite_jdbc_driver_connection



Di seguito è riportata la tabella della descrizione della classe appartenente al package sqlite_jdbc_driver_connection:

Classe	Descrizione
SQLiteConnector.java	Permette di istanziare la connessione col DB.

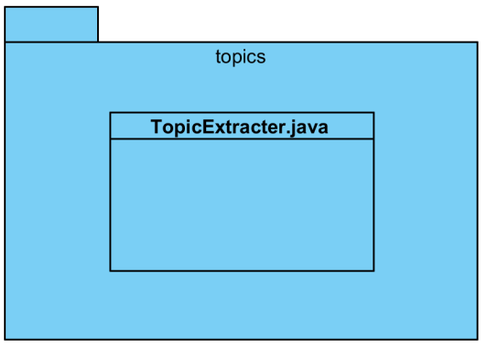
2.1.6 Package structuralMetrics



Di seguito è riportata la tabella della descrizione della classe appartenente al package structuralMetrics:

Classe	Descrizione
CKMetrics.java	Permette di calcolare le metriche di qualità di un qualsiasi Bean.

2.1.7 Package topics



Di seguito è riportata la tabella della descrizione della classe appartenente al package topics:

Classe	**Descrizione**
TopicExtracter.java	Permette di estrarre i topic, cioè cinque termini più frequenti, da un qualsiasi Bean.

3 Interfaccia delle Classi

3.1 Analysis

3.1.1 Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell durante l'analisi del codice.

- public abstract boolean affects(T component)
  Questo metodo verifica se è presente un Code Smell in un component T.

3.1.1.1 Class Level Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell in una classe durante l'analisi del codice.

- public boolean affects(ClassBean aBean)
  Questo metodo verifica se è presente un Code Smell in un Class Bean.

3.1.1.2 Method Level Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell in un metodo durante l'analisi del codice.

- public boolean affects(MethodBean aMethod)
  Questo metodo verifica se è presente un Code Smell in un Method Bean.

3.1.1.3 Package Level Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell in un package durante l'analisi del codice.

- public boolean affects(PackageBean aPackage)
  Questo metodo verifica se è presente un Code Smell in un Package Bean.

3.1.1.4 Blob Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell di tipo Blob.

- public boolean affects(ClassBean aClass)
  Questo metodo verifica se è presente un Code Smell di tipo Blob in un Class Bean.

3.1.1.5 Feature Envy Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell di tipo Feature Envy.

- public boolean affects(MethodBean aMethod)
  Questo metodo verifica se è presente un Code Smell di tipo Feature Envy in un Method Bean.

3.1.1.6 Misplaced Class Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell di tipo Misplaced Class.

- public boolean affects(ClassBean aClass)
  Questo metodo verifica se è presente un Code Smell di tipo Misplaced Class in un Class Bean.

3.1.1.7 Promiscuous Package Code Smell

Si tratta di una classe che funge da interfaccia del sottosistema che individua i Code Smell di tipo Promiscuous Package.

- public boolean affects(PackageBean aPackage)
  Questo metodo verifica se è presente un Code Smell di tipo Promiscuous Package in un Package Bean.

3.1.2 Code Smell Detection

3.1.2.1 Code Smell Detection Strategy

Si tratta di un'interfaccia del sottosistema che rileva Code Smell nei Bean.

- boolean isSmelly(T component)
  Questo metodo permette di verificare se il component T è affetto da Code Smell.

3.1.2.1.1 Class Smell Detection Strategy

Si tratta di un'interfaccia che implementa l'interfaccia del sottosistema che rilevano Code Smell nei Class Bean.

- public boolean isSmelly(ClassBean aClass)
  Questo metodo permette di verificare se un Class Bean è affetto da Code Smell.

Questa interfaccia è implementata dalle classi **Textual Misplaced Class Strategy**, **Structural Misplaced Class Rule** e **Textual Blob Strategy**.

3.1.2.1.2 Method Smell Detection Strategy

Si tratta di un'interfaccia che implementa l'interfaccia del sottosistema che rilevano Code Smell nei Method Bean.

- public boolean isSmelly(MethodBean aMethod)
  Questo metodo permette di verificare se un Method Bean è affetto da Code Smell.

Questa interfaccia è implementata dalla classe **Textual Feature Envy Strategy** e **Structural Feature Envy Rule**.

3.1.2.1.3 Package Smell Detection Strategy

Si tratta di un'interfaccia che implementa l'interfaccia del sottosistema che rilevano Code Smell nei Package Bean.

- public boolean isSmelly(PackageBean aPackage)
  Questo metodo permette di verificare se un Package Bean è affetto da Code Smell.

Questa interfaccia è implementata dalla classe **Textual Promiscuous Package Strategy**.

3.1.2.2 Comparable Bean

Si tratta di un'interfaccia del sottosistema che compara Bean fra loro per individuare Code Smell.

- public double getSimilarity()
  Questo metodo permette di ottenere le similitudini dei Beans.

3.1.2.2.1 Bean Comparator

Si tratta di una classe che implementa l'interfaccia del sottosistema che compara Bean fra loro per individuare Code Smell.

- public int compare(ComparableBean o1, ComparableBean o2)
  Questo metodo permette di comparare due Beans.

3.2 Parser

Si tratta di un'interfaccia del sottosistema che effettua la conversione in Bean.

- public void parse()
  Questo metodo effettua la conversione in Bean.

3.2.1 Psi Parser

Si tratta di una classe che implementa l'interfaccia del sottosistema che effettua la conversione da Psi a Bean.

- public void parse()
  Questo metodo effettua la conversione da Psi a Bean.

3.3 Refactoring

3.3.1 Refactoring Manager

Si tratta di una classe che funge da interfaccia del sottosistema che esegue il Refactoring.

- public void executeRefactor()
  Questo metodo permette di eseguire il Refactor.

3.3.2 Refactoring Strategy

Si tratta di un'interfaccia del sottosistema che effettua il Refactoring.

- public void doRefactor()

  Questo metodo permette di effettuare il Refactor.

3.4.2.1 Blob Refactoring Strategy

Si tratta di una classe che implementa l'interfaccia del sottosistema che effettua il Refactoring del Code Smell Blob.

- public void doRefactor()
  Questo metodo permette di effettuare il Refactor delle classi affette da Code Smell di tipo Blob.

3.4.2.2 Feature Envy Refactoring Strategy

Si tratta di una classe che implementa l'interfaccia del sottosistema che effettua il Refactoring del Code Smell Feature Envy.

- public void doRefactor()
  Questo metodo permette di effettuare il Refactor delle metodi affetti da Code Smell di tipo Feature Envy.

3.4.2.3 Misplaced Class Refactoring Strategy

Si tratta di una classe che implementa l'interfaccia del sottosistema che effettua il Refactoring del Code Smell Misplaced Class.

- public void doRefactor()
  Questo metodo permette di effettuare il Refactor delle classi affette da Code Smell di tipo Misplaced Class.

3.4.2.4 Promiscuous Package Refactoring Strategy

Si tratta di una classe che implementa l'interfaccia del sottosistema che effettua il Refactoring del Code Smell Promiscuous Package.

- public void doRefactor()
  Questo metodo permette di effettuare il Refactor dei package affetti da Code Smell di tipo Promiscuous Package.

3.4 Storage

3.4.1 Beans

3.1.1.1 Package Bean

Si tratta di una classe che funge da interfaccia del sottosistema che immagazzina i Package Bean.

- public boolean isAffected(CodeSmell smell)
  Questo metodo permette di verificare se nel Package Bean è presente un code smell.
- public void addClassList(ClassBean bean)
  Questo metodo permette di aggiungere una classe al Package Bean.
- public void removeClassList(ClassBean bean)
  Questo metodo permette di rimuovere una classe dal Package Bean.
- public void addSmell(CodeSmell smell)
  Questo metodo permette di aggiungere un code smell al Package Bean.

3.1.1.2 Class Bean

Si tratta di una classe che funge da interfaccia del sottosistema che immagazzina i Class Bean.

- public boolean isAffected(CodeSmell smell)
  Questo metodo permette di verificare se nel Class Bean è presente un code smell.
- public void addInstanceVariableList(ClassBean bean)
  Questo metodo permette di aggiungere una variabile d'istanza alla lista delle variabili d'istanza del Class Bean.
- public void removeInstanceVariableList(ClassBean bean)
  Questo metodo permette di rimuovere una variabile d'istanza alla lista delle variabili d'istanza del Class Bean.
- public void addMethodBeanList(MethodBean bean)
  Questo metodo permette di aggiungere un metodo alla lista dei metodi appartenenti al Class Bean.
- public void removeMethodBeanList(MethodBean bean)
  Questo metodo permette di rimuovere un metodo alla lista dei metodi appartenenti al Class Bean.
- public void addImports(String i)
  Questo metodo permette di aggiungere un import al Class Bean.
- public void removeImports(String i)
  Questo metodo permette di rimuovere un import dal Class Bean.
- public void addSmell(CodeSmell smell)
  Questo metodo permette di aggiungere un code smell al Class Bean.

3.1.1.3 Method Bean

Si tratta di una classe che funge da interfaccia del sottosistema che immagazzina i Method Bean.

- public boolean isAffected(CodeSmell smell)
  Questo metodo permette di verificare se nel Method Bean è presente un code smell.
- public void addParameters(String key, ClassBean bean)
  Questo metodo permette di aggiungere un paramentro al Method Bean.
- public void removeParameters(String key, ClassBean bean)
  Questo metodo permette di rimuovere un paramentro al Method Bean.
- public void addInstanceVariableList(InstanceVariableBean bean)
  Questo metodo permette di aggiungere una variabile d'istanza alla lista delle variabili d'istanza del Method Bean.
- public void removeInstanceVariableList(InstanceVariableBean bean)
  Questo metodo permette di rimuovere una variabile d'istanza alla lista delle variabili d'istanza del Method Bean.
- public void addMethodsCalls(MethodBean bean)
  Questo metodo permette di aggiungere un Method Bean alla lista dei metodi chiamati.
- public void removeMethodsCalls(MethodBean bean)
  Questo metodo permette di rimuovere un Method Bean alla lista dei metodi chiamati.
- public void addSmell(CodeSmell smell)
  Questo metodo permette di aggiungere un code smell al Method Bean.

3.1.1.4 Class Bean List

Si tratta di un'interfaccia del sottosistema che contiene la lista di Class Bean.

- public List<ClassBean> getList()
  Questo metodo permette di ottenere la lista di classi presenti nel DB.

3.1.1.4.1 Class List

Si tratta di una classe che implementa l'interfaccia del sottosistema che crea la lista di Class Bean.

- public List<ClassBean> getList()
  Questo metodo permette di ottenere la lista di classi presenti nel DB.

- public void setList(List<ClassBean> classes)
  Questo metodo permette di settare la lista di classi.

3.1.1.4.2 Class List Proxy

Si tratta di una classe che implementa l'interfaccia del sottosistema che si occupa del lazy loading delle classi.

- public List<ClassBean> getList()
  Questo metodo permette di ottenere la lista di classi presenti nel DB.

3.1.1.5 Method Bean List

Si tratta di un'interfaccia del sottosistema che contiene la lista di Method Bean.

- public List<MethodBean> getList()
  Questo metodo permette di ottenere la lista di metodi presenti nel DB.

3.1.1.5.1 Method List

Si tratta di una classe che implementa l'interfaccia del sottosistema che crea la lista di Method Bean.

- public List<MethodBean> getList()
  Questo metodo permette di ottenere la lista di metodi presenti nel DB.
- public void setList(List<MethodBean> classes)
  Questo metodo permette di settare la lista di metodi.

3.1.1.5.2 Method List Proxy

Si tratta di una classe che implementa l'interfaccia del sottosistema che si occupa del lazy loading dei metodi.

- public List<MethodBean> getList()
  Questo metodo permette di ottenere la lista di metodi presenti nel DB.

3.1.1.5.3 Called Methods List Proxy

Si tratta di una classe che implementa l'interfaccia del sottosistema che si occupa del lazy loading dei metodi.

- public List<MethodBean> getList()
  Questo metodo permette di ottenere la lista di metodi chiamati da uno specifico metodo.

3.1.1.6 Instance Variable Bean List

Si tratta di un'interfaccia del sottosistema che contiene la lista di Instance Variable Bean.

- public List<InstanceVariableBean> getList()
  Questo metodo permette di ottenere la lista di variabili di istanza presenti nel DB.

3.1.1.6.1 Instance Variable List

Si tratta di una classe che implementa l'interfaccia del sottosistema che crea la lista di Instance Variable Bean.

- public List<InstanceVariableBean> getList()
  Questo metodo permette di ottenere la lista di variabili di istanza presenti nel DB.
- public void setList(List<InstanceVariableBean> classes)
  Questo metodo permette di settare la lista di variabili di istanza.

3.1.1.6.2 Instance Variable List Proxy

Si tratta di una classe che implementa l'interfaccia del sottosistema che si occupa del lazy loading delle variabili di istanza.

- public List<InstanceVariableBean> getList()
  Questo metodo permette di ottenere la lista di variabili di istanza presenti nel DB.

3.1.1.6.3 Used Instance Variable List Proxy

Si tratta di una classe che implementa l'interfaccia del sottosistema che si occupa del lazy loading delle variabili di istanza.

- public List<InstanceVariableBean> getList()
  Questo metodo permette di ottenere la lista delle variabili di istanza utilizzate da uno specifico metodo.

3.4.2 Repository

Si tratta di una classe che funge da interfaccia del sottosistema che contiene la Repository dei Bean.

- public void add(T toAdd)
  Questo metodo permette di aggiungere un Bean al DB.
- public void remove(T toRemove)
  Questo metodo permette di rimuovere un Bean dal DB.
- public void update(T toUpdate)
  Questo metodo permette di aggiornare un Bean contenuto nel DB.
- public List<T> select(Criterion criterion)
  Questo metodo permette di selezionare una lista di Bean dal DB.

3.1.2.1 Package Bean Repository

Si tratta di una classe che funge da interfaccia del sottosistema che contiene la Repository dei Package Bean.

- public void add(PackageBean aPackage)
  Questo metodo permette di aggiungere un Package Bean al DB.
- public void remove(PackageBean aPackage)
  Questo metodo permette di rimuovere un Package Bean dal DB.
- public void update(PackageBean aPackage)
  Questo metodo permette di aggiornare un Package Bean contenuto nel DB.
- public List<PackageBean> select(Criterion criterion)
  Questo metodo permette di selezionare una lista di Package Bean dal DB.

Questa interfaccia viene implementata dalla classe **Package Repository**.

3.1.2.2 Class Bean Repository

Si tratta di un'interfaccia del sottosistema che contiene la repository dei class bean.

- public void add(ClassBean aClass)
  Questo metodo permette di aggiungere un Class Bean al DB.
- public void remove(ClassBean aClass)
  Questo metodo permette di rimuovere un Class Bean dal DB.
- public void update(ClassBean aClass)
  Questo metodo permette di aggiornare un Class Bean contenuto nel DB.
- public List<ClassBean> select(Criterion criterion)
  Questo metodo permette di selezionare una lista di Class Bean dal DB.

Questa interfaccia viene implementata dalla classe **Class Repository**.

3.1.2.3 Method Bean Repository

Si tratta di una classe che funge da interfaccia del sottosistema che contiene la repository dei method bean.

- public void add(MethodBean aMethod)
  Questo metodo permette di aggiungere un Class Bean dal DB.
- public void remove(MethodBean aMethod)
  Questo metodo permette di rimuovere un Method Bean dal DB.
- public void update(MethodBean aMethod)
  Questo metodo permette di aggiornare un Method Bean contenuto nel DB.
- public List<MethodBean> select(Criterion criterion)
  Questo metodo permette di selezionare una lista di Method Bean dal DB.

Questa interfaccia viene implementata dalla classe **Method Repository**.

3.1.2.4 Instance Variable Bean Repository

Si tratta di una classe che funge da interfaccia del sottosistema che contiene la repository degli instance variable bean.

- public void add(InstanceVariableBean aInstance)
  Questo metodo permette di aggiungere un Instance Variable Bean al DB.
- public void remove(InstanceVariableBean aInstance)
  Questo metodo permette di rimuovere un Instance Variable Bean al DB.
- public void update(InstanceVariableBean aInstance)
  Questo metodo permette di aggiornare un Instance Variable Bean contenuto nel DB.
- public List<InstanceVariableBean> select(Criterion criterion)
  Questo metodo permette di selezionare una lista di Instance Variable Bean dal DB.

Questa interfaccia viene implementata dalla classe **Instance Variable Repository**.

3.1.2.5 Criterion

Si tratta di un'interfaccia del sottosistema che contiene la repository dei Bean.

3.1.2.5.1 SQLite Criterion

Si tratta di un'interfaccia che implementa l'interfaccia del sottosistema che contiene la repository dei Bean.

- public String toSQLquery()
  Questo metodo permette di ottenere la query da eseguire nel DB per la selezione.

3.5 CK Metrics

Si tratta di una classe che funge da interfaccia del sottosistema che calcola le metriche di qualità.

- public static int getLOC(ClassBean cb)
  Questo metodo permette di calcolare la metrica LOC di un Class Bean.
- public static int getLOC(MethodBean cb)
  Questo metodo permette di calcolare la metrica LOC di un Method Bean.
- public static int getWMC(ClassBean cb)
  Questo metodo permette di calcolare la metrica WMC di un Class Bean.
- public static int getDIT(ClassBean cb, Vector<ClassBean> System, int inizialization)
  Questo metodo permette di calcolare la metrica DIT di un Class Bean.
- public static int getNOC(ClassBean cb, Vector<ClassBean> System)
  Questo metodo permette di calcolare la metrica NOC di un Class Bean.
- public static int getRFC(ClassBean cb)
  Questo metodo permette di calcolare la metrica RFC di un Class Bean.
- public static int getCBO(ClassBean cb)
  Questo metodo permette di calcolare la metrica CBO di un Class Bean.
- public static int getLCOM(ClassBean cb)
  Questo metodo permette di calcolare la metrica LCOM di un Class Bean.
- public static int getNOM(ClassBean cb)
  Questo metodo permette di calcolare la metrica NOM di un Class Bean.
- public static int getNOA(ClassBean cb)
  Questo metodo permette di calcolare la metrica NOA di un Class Bean.
- public static int getNOPA(ClassBean cb)
  Questo metodo permette di calcolare la metrica NOPA di un Class Bean.
- public static int getNOPrivateA(ClassBean cb)
  Questo metodo permette di calcolare la metrica NOPrivateA di un Class Bean.
- public static int getNOO(ClassBean cb, Vector<ClassBean> System)
  Questo metodo permette di calcolare la metrica NOO di un Class Bean.

3.6 Topic Extracter

Si tratta di una classe che funge da interfaccia del sottosistema che calcola i topic (cioè i termini più frequenti) dei Bean.

- public TreeMap<String, Integer> extractTopicFromPackageBean(PackageBean aBean)
  Questo metodo permette di estrarre topic da un PackageBean.
- public TreeMap<String, Integer> extractTopicFromClassBean(ClassBean aBean)
  Questo metodo permette di estrarre topic da un ClassBean.
- public TreeMap<String, Integer> extractTopicFromMethodBean(MethodBean aBean)
  Questo metodo permette di estrarre topic da un MethodBean.
- public Collection<String> termsExtracter(String textContent)
  Questo metodo permette di estrarre tutte le parole di un testo.
- public String deleteComments(String pTextContent)
  Questo metodo permette di eliminare da un testo i commenti contenuti in esso.

4 Design Patterns

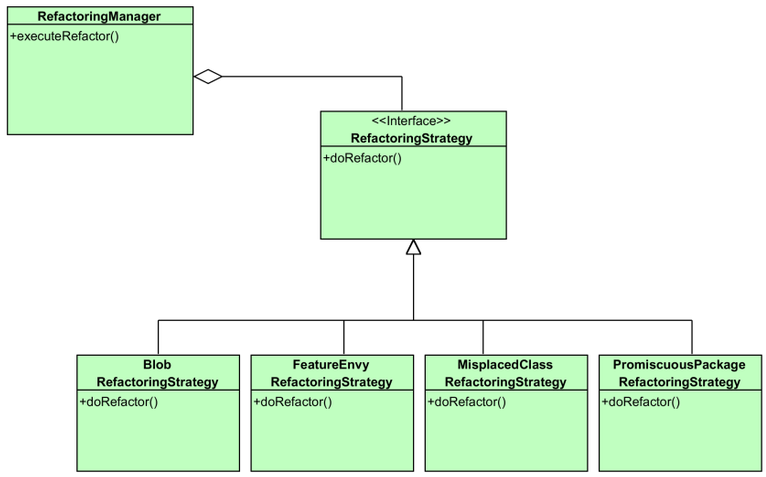
4.1 Strategy Pattern

Lo Strategy Pattern consente di isolare un algoritmo al di fuori di una classe, per far sì che quest'ultima possa variare dinamicamente il suo comportamento, rendendo così gli algoritmi intercambiabili a runtime.

Grazie allo Strategy Pattern è possibile utilizzare una qualsiasi implementazione (che viene genericamente chiamata Strategy o Strategia), scegliendo fra quelle disponibili,che si rende più opportuna in un determinato contesto, in quanto tutte le implementazioni facenti parte della stessa famiglia hanno interfaccia comune.

Questo pattern è stato usato per implementare le funzioni refactoring e analisi poichè per entrambe utilizziamo interfacce che fanno riferimento a metodi differenti.

Per il refactoring abbiamo un'interfaccia "RefactoringStrategy" che viene implementata dalle seguenti classi: BlobRefactoringStrategy, FeatureEnvyRefactoringStrategy, MisplacedClassRefactoringStrategy, PromiscuousPackageRefactoringStrategy.

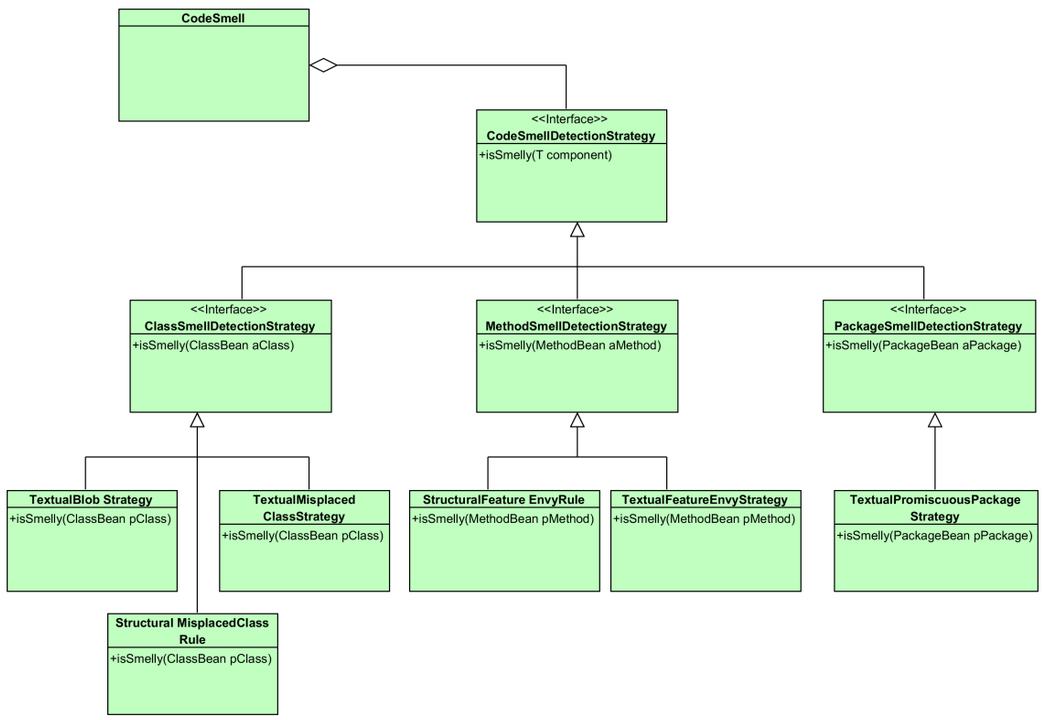


Per l'analisi abbiamo un'interfaccia "CodeSmellDetectionStrategy" che viene estesa dalle seguenti interfacce: ClassSmellDetectionStrategy, MethodSmellDetectionStrategy, PackageSmellDetectionStrategy.

L'interfaccia ClassSmellDetectionStrategy è implementata da tre classi concrete, ossia TextBlobStrategy, StructuralMisplacedClassRule e TextualMisplacedClassStrategy.

L'interfaccia MethodSmellDetectionStrategy è implementata da due classi concrete, ossia StructuralFeatureEnvyRule e TextualFeatureEnvyRule.

L'interfaccia PackageSmellDetectionStrategy è implementata da una classe concreta, ossia TextualPromiscuousPackageStrategy.

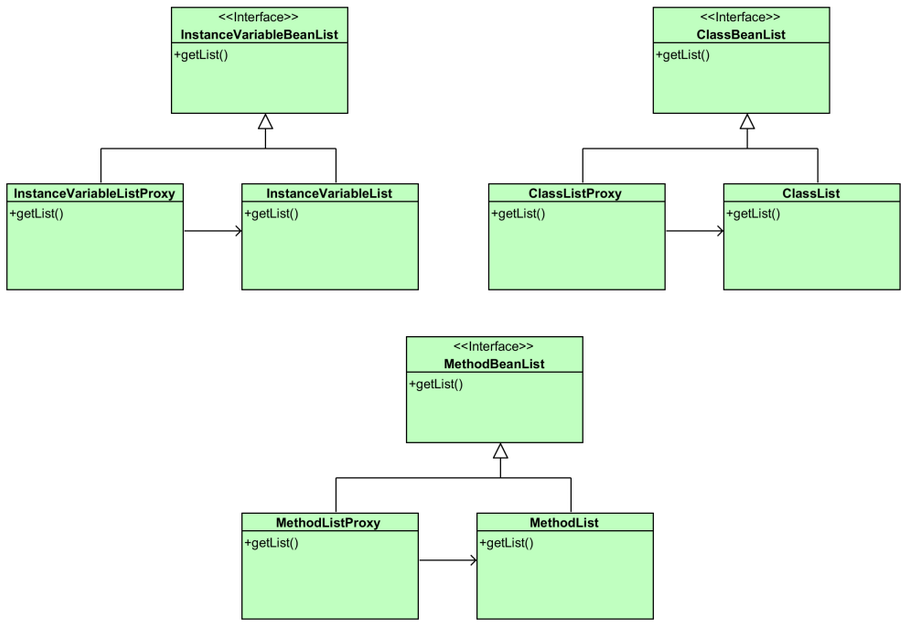


4.2 Proxy Pattern

Si tratta di un pattern strutturale basato su oggetti che viene utilizzato per accedere ad un un oggetto complesso tramite un oggetto semplice.

Questo pattern viene utilizzato per caricare una versione semplificata di oggetti di tipo InstanceVariableList, MethodList, ClassList tramite le classi proxy InstanceVariableListProxy, MethodListProxy, ClassListProxy.

Il client può accedere a queste classi tramite interfacce "InstanceVariableBeanList", "MethodBeanList", "ClassBeanList".



4.3 Builder Pattern

Si tratta di un pattern creazionale basato su oggetti e viene utilizzato per creare un oggetto senza doverne conoscere i suoi dettagli implementativi. Questo pattern consente di utilizzare un Client che non debba essere a conoscenza dei passi necessari al fine della creazione di un oggetto ma tali passaggi vengono delegati ad un Director (classe che si occupa effettivamente di costruire l'oggetto) che sa cosa e come fare.

Questo pattern è usato nel sistema per costruire gli oggetti di tipo ClassBean, PackageBean, MethodBean.

4.4 Repository Pattern

I repository sono classi o componenti che incapsulano la logica necessaria per accedere ai dati sorgente. Centralizzano le funzionalità comuni di accesso ai dati, fornendo una migliore manutenibilità e disaccoppiando l'infrastruttura o la tecnologia utilizzata per accedere ai database dal layer del modello di dominio.

Questo pattern viene utilizzato per lo storage di oggetti di tipo InstanceVariableBean, MethodBean, ClassBean, PackageBean.

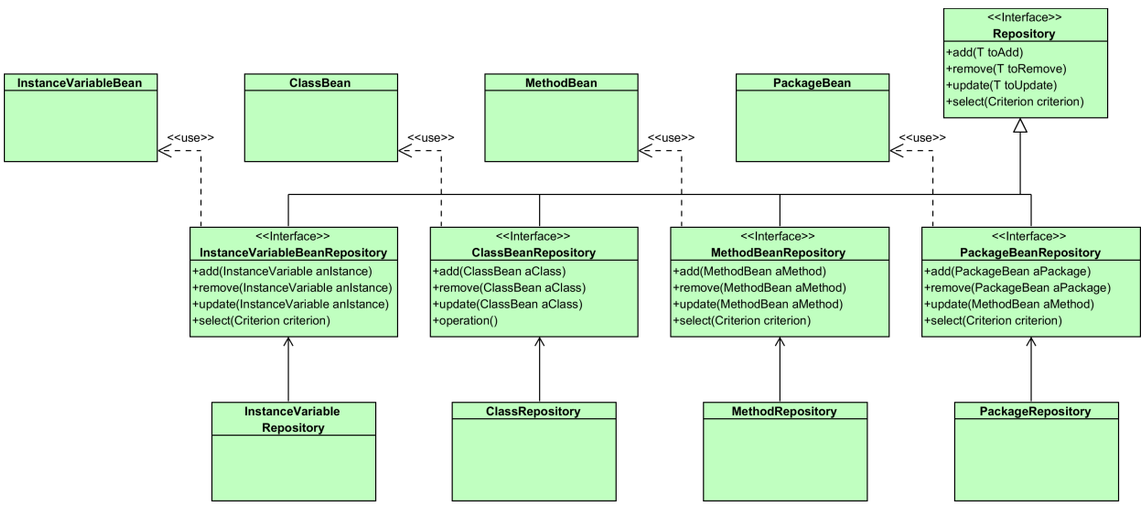
Per implementare questo pattern utilizziamo l'interfaccia "Repository" che viene estesa dalle seguenti interfacce: InstanceVariableBeanRepository, MethodBeanRepository, ClassBeanRepository, PackageBeanRepository.

L'interfaccia InstanceVariableBeanRepository è implementata dalla classe InstanceVariableRepository.

L'interfaccia MethodBeanRepository è implementata dalla classe MethodRepository.

L'interfaccia ClassBeanRepository è implementata dalla classe ClassRepository.

L'interfaccia PackageBeanRepository è implementata dalla classe PackageRepository.



4.5 Adapter Pattern

L'Adapter è un pattern strutturale che può essere basato sia su classi (Class Adapter) che su oggetti (Object Adapter) e il suo scopo è convertire le interfacce di una classe in altre interfacce, attese dai client, per far sì che classi con interfacce di differenti e incompatibili possano comunque poter comunicare tra loro.

Questo pattern viene utilizzato per costruire le RadarMap di ogni Bean.

Per questo pattern è stata definita l'interfaccia "RadarMapUtils", che viene implementata dalla classe RadarMapUtilsAdapter.