

Università degli studi di Salerno

Dipartimento di informatica



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Corso di Ingegneria Gestione ed Evoluzione del Software

Aniello Florido & Alexander Minichino

Metric 3.0

Reverse Object Design Document

Ottobre 2020

Introduzione

Metric 3.0 è sprovvisto di una documentazione dettagliata e formale.

Pertanto, prima di avviare qualsivoglia attività di manutenzione, è stato ritenuto necessario attuare un processo di reverse engineering, con l'obiettivo di recuperare informazioni utili partendo dagli artefatti del sistema.

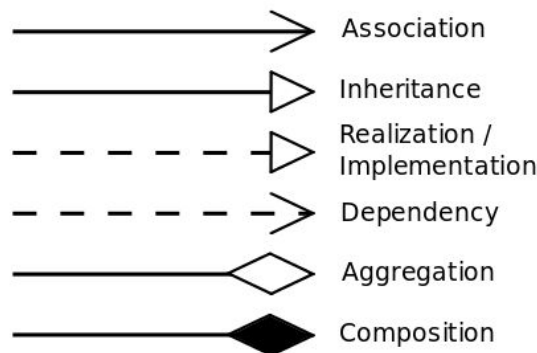
Tali informazioni saranno utilizzate per la costruzione di documenti utili che possano facilitare il processo di migrazione, avendo una base di conoscenza più strutturata.

Il processo di reverse engineering ha visto come primo step lo svolgersi di un'analisi statica del codice attraverso l'ausilio di un tool di reverse engineering.

Come tool è stato scelto il plugin UML presente all'interno dell'IDE IntelliJ IDEA che ha permesso di ottenere i diagrammi UML di Metric3.0 a diversi livelli di granularità.

Legenda

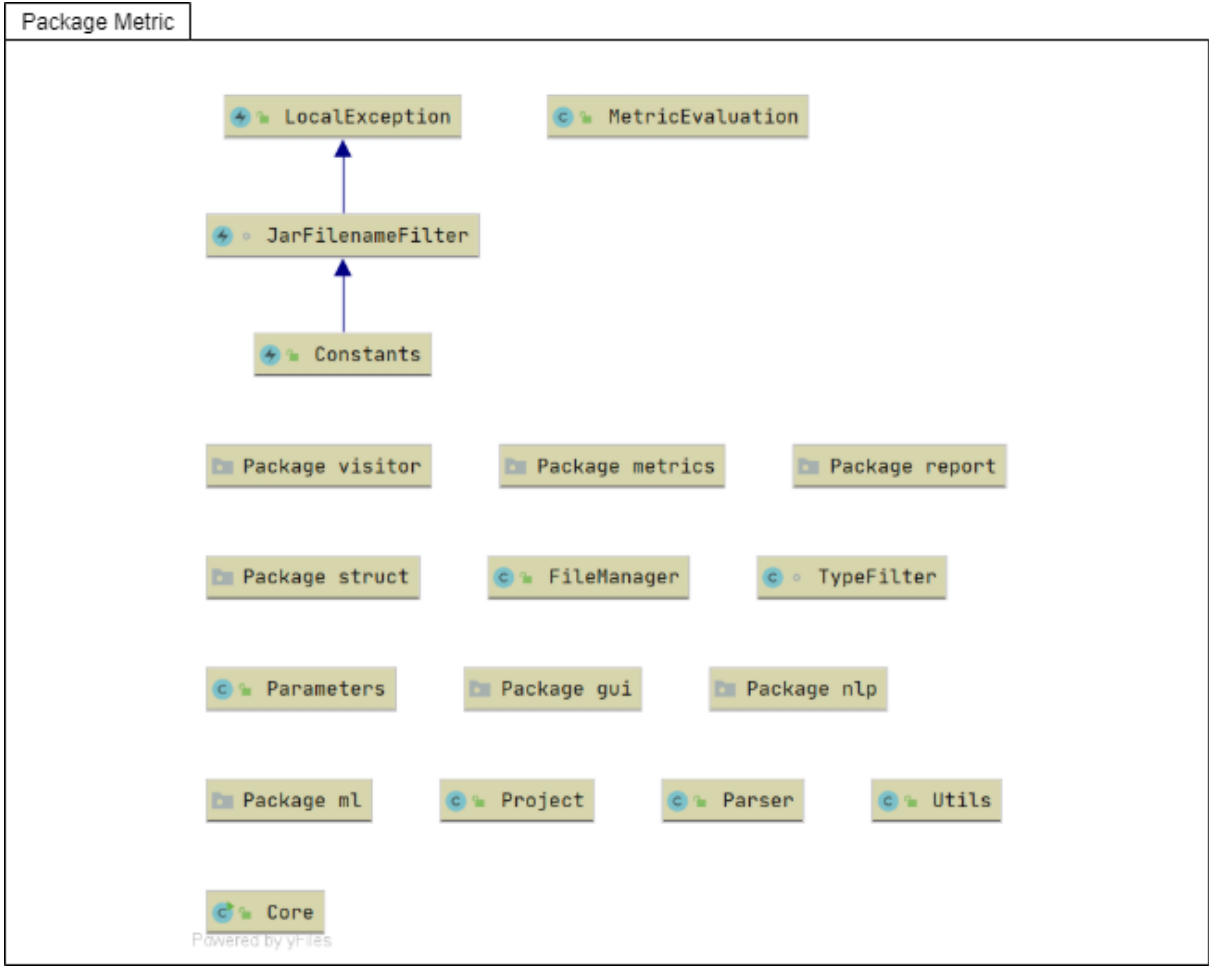
I diagrammi prodotti dal tool utilizzato rispettano la notazione UML, di seguito è riportata una legenda riassuntiva della relazioni e dei simboli utilizzati:



Di seguito sono riportati i diagrammi a livello di package con le relative dipendenze.

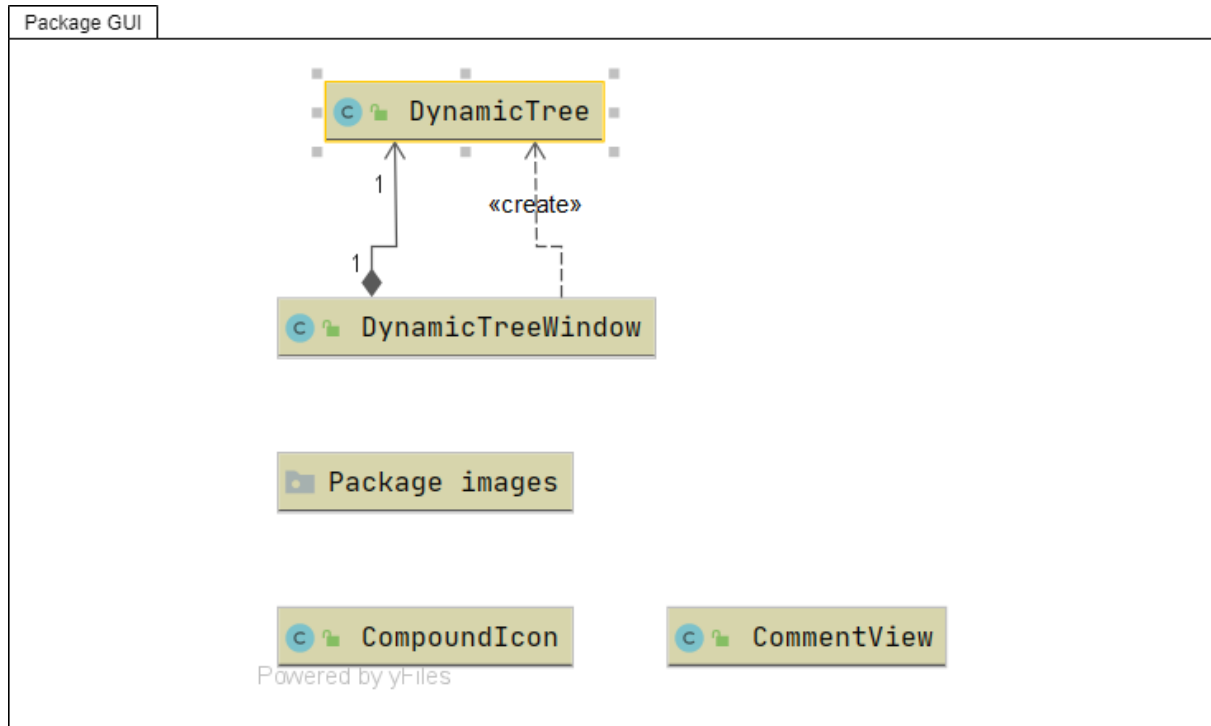
Package principale del progetto.

Visualizzazione più agevole del Package



Package GUI

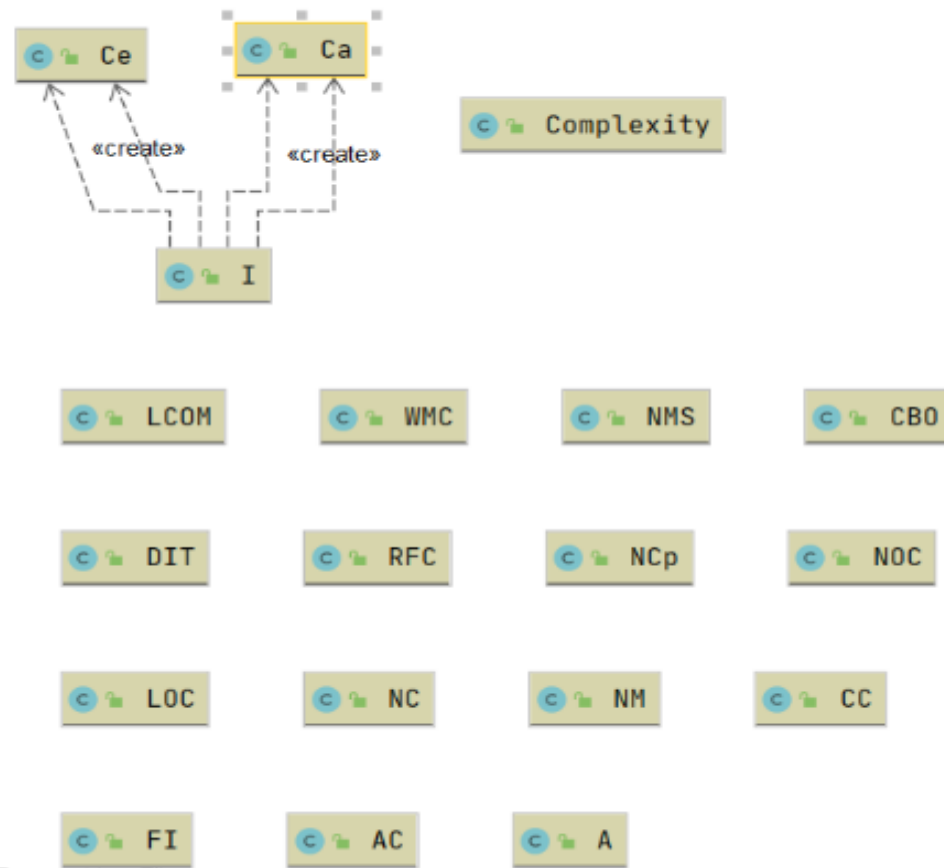
Package contenente le classi relative alla visualizzazione grafica dell'albero dei commenti.



Package Metrics

Package contenente tutte le classi che rappresentano le varie metriche utilizzate.

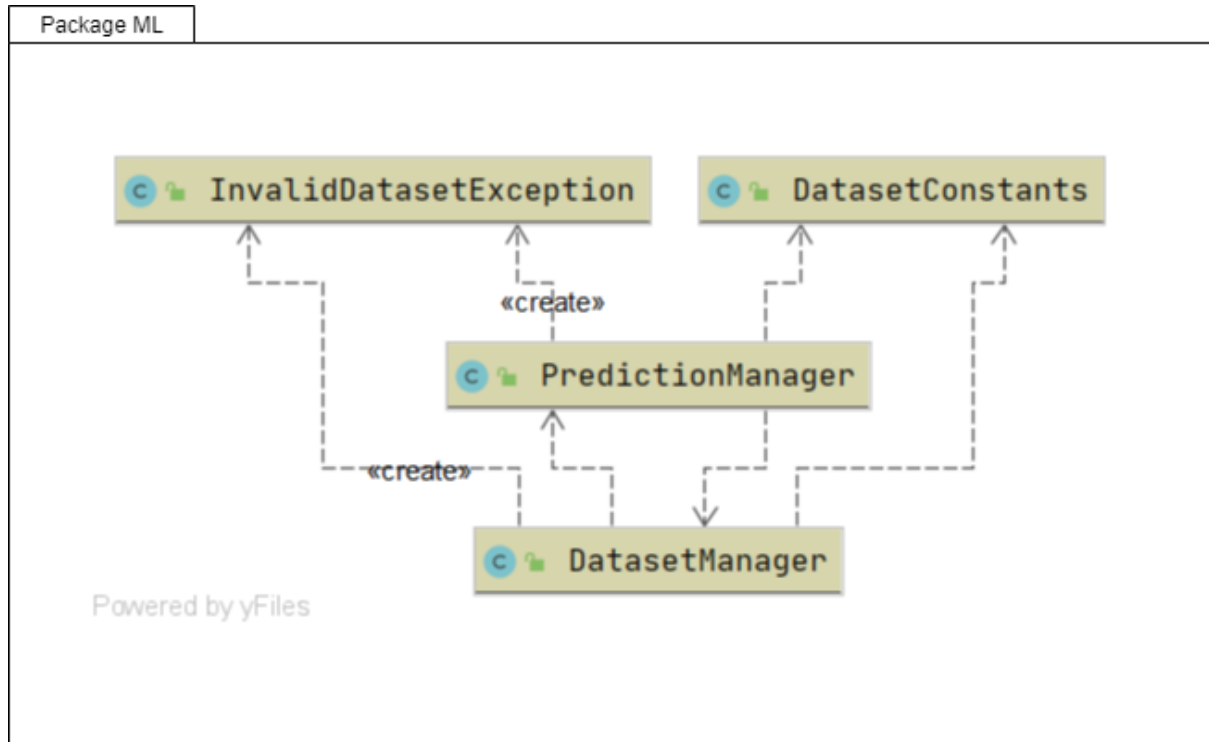
Package Metrics



Powered by yFiles

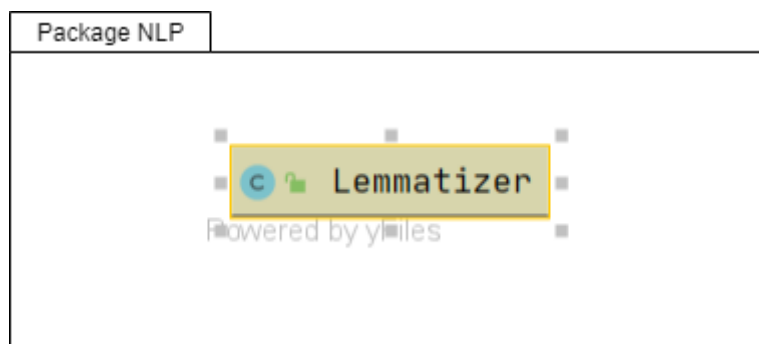
Package ML

Package contenente le classi per modellare e analizzare dataset.



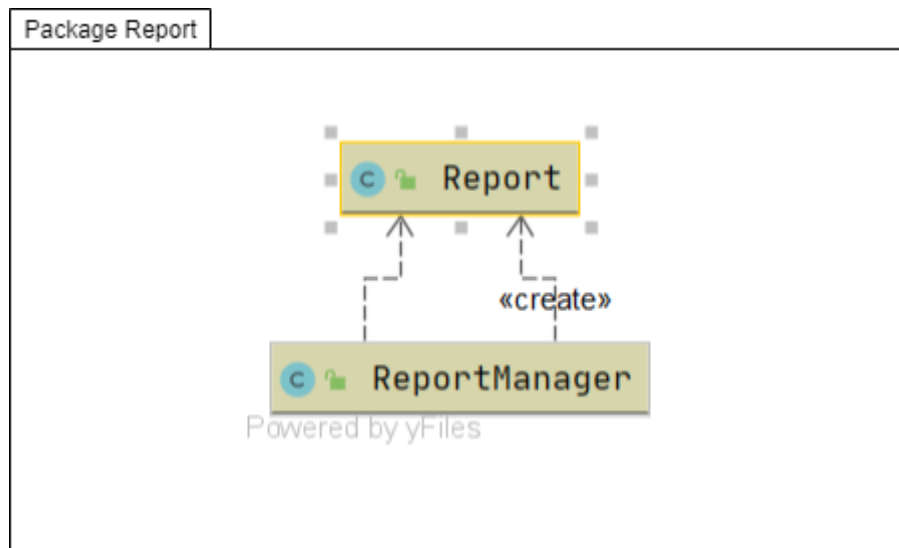
Package NLP

Package relativo al lemmatization



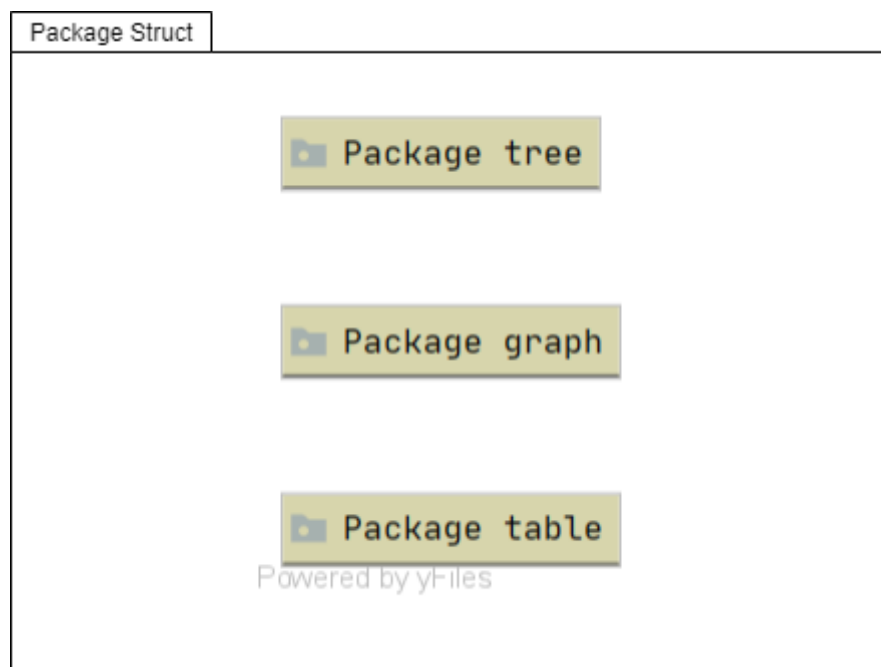
Package Report

Package relativo all'analisi di file di bug report



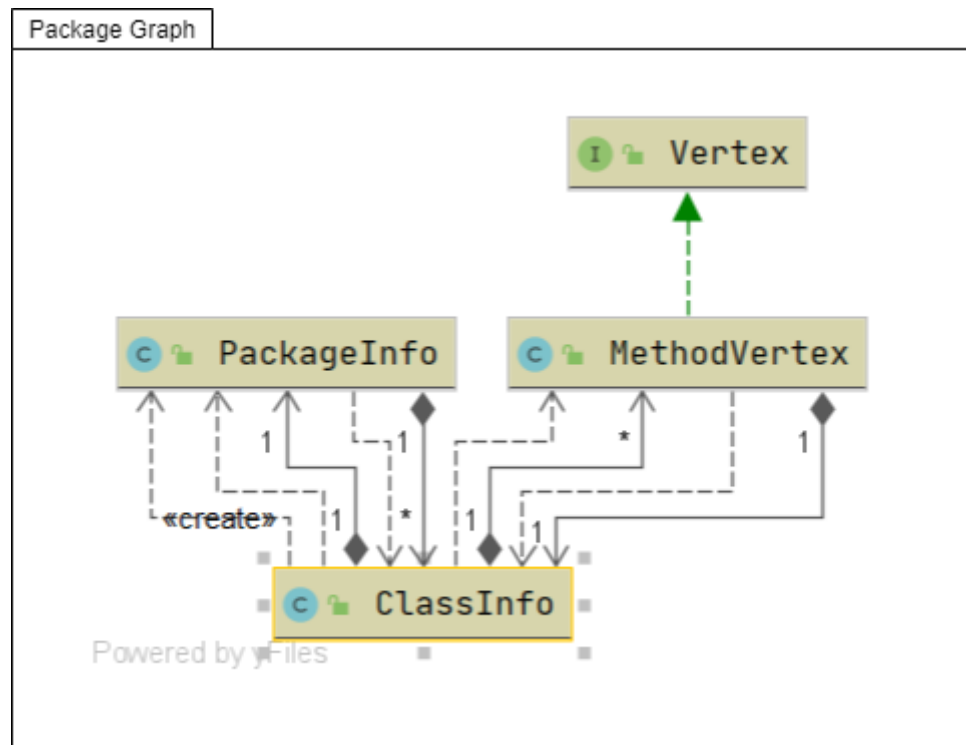
Package Struct

Package che contiene tutte le strutture dati utilizzate



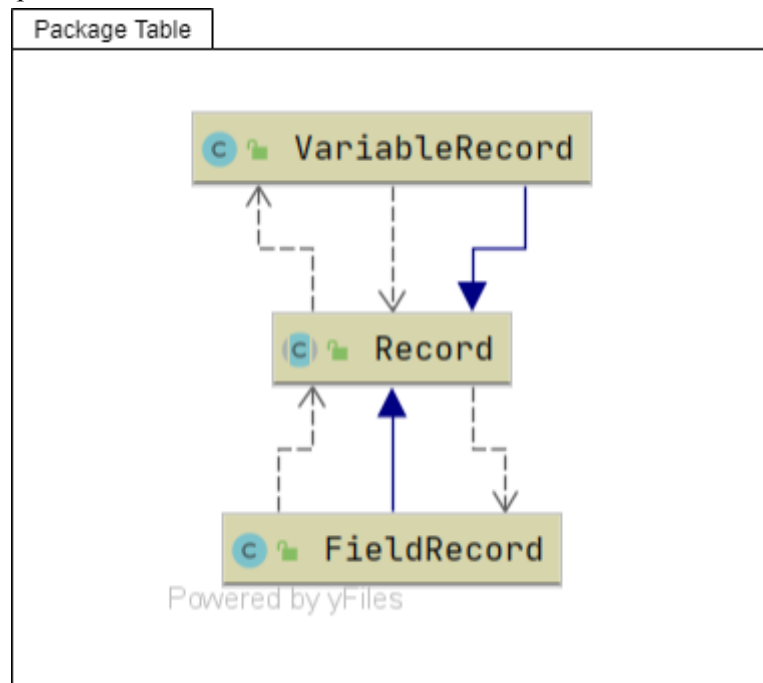
Package Graph

Contiene le classi relative alla struttura dati a grafo utilizzata per rappresentare i metodi.



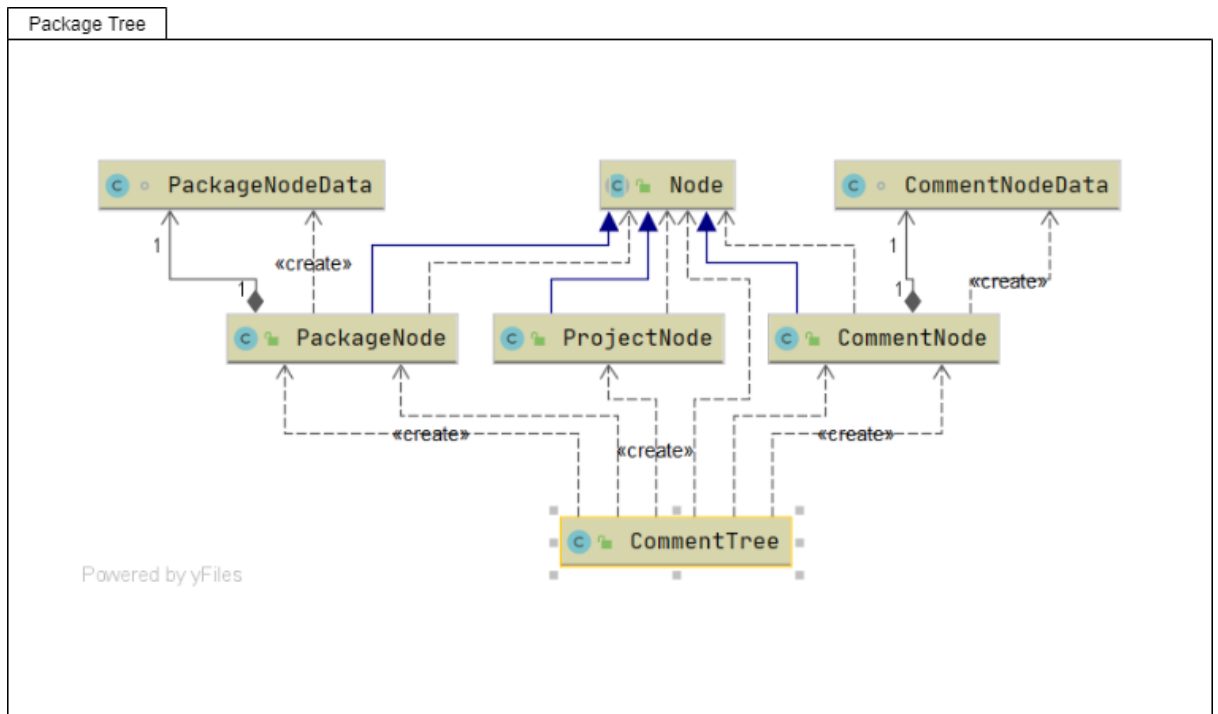
Package Table:

Contiene tutte le classi per memorizzare i dati relativi a variabili e attributi di classe



Package Tree


Contiene le classi relative alla struttura dati ad albero utilizzata per i commenti.





Package Visitor


Contiene tutti gli ASTVisitor, ovvero i metodi che ci permettono di visitare i nodo dell'AST e raccogliere informazioni.


Package Visitor


 MethodDeclarationVisitor


 MethodInvocationVisitor

 FieldAndVariableVisitor

 SimpleNameVisitor

 StatementVisitor

 ElementVisitor

 CommentVisitor

Powered by yFiles

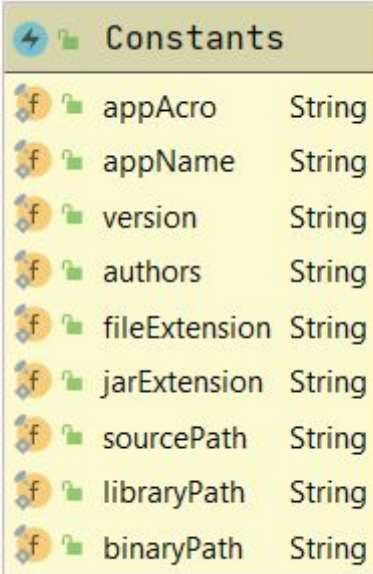
Classes

Di seguito sono riportate le classi relative ai package mostrati precedentemente.
Le classi sono ordinate per package.

Package metric:

Classe Constants

Classe utilizzata per racchiudere tutte le costanti utilizzate nel programma.



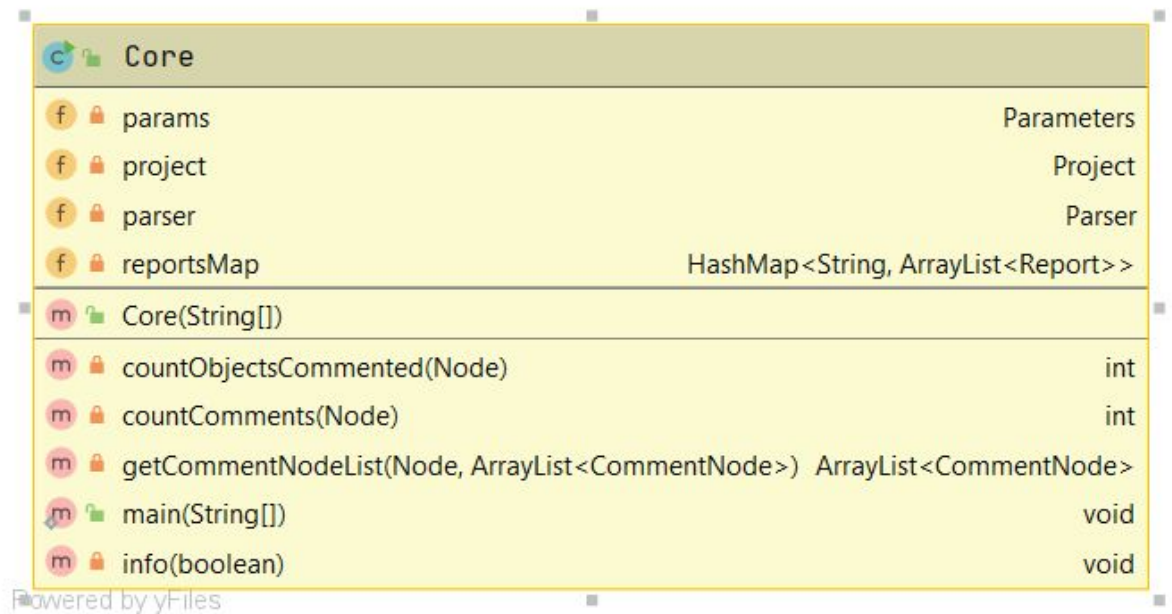
The screenshot shows a window titled 'Constants' with a list of static variables. Each variable is preceded by a small icon consisting of a blue circle with a white 'f' and a green folder icon. The variables are all of type 'String'.

Variable	Type
appAcro	String
appName	String
version	String
authors	String
fileExtension	String
jarExtension	String
sourcePath	String
libraryPath	String
binaryPath	String

Powered by yFiles

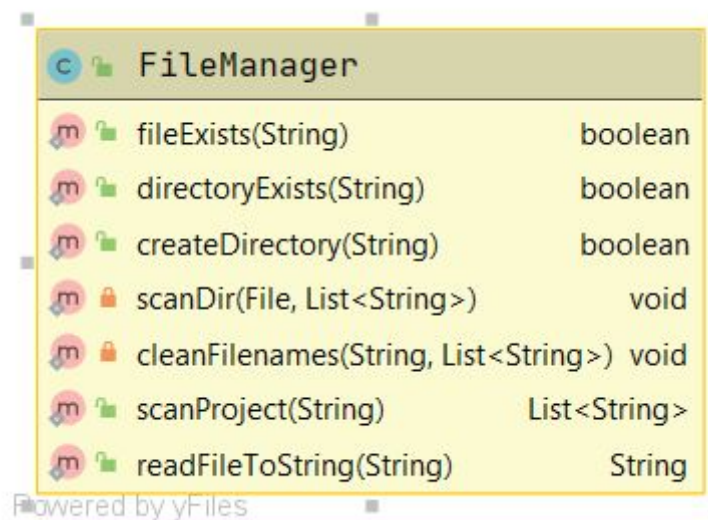
Classe Core

La classe che contiene il main del progetto e dalla quale vengono richiamate tutte le funzionalità.









Classe FileManager

Contiene tutte le funzionalità che occorrono per gestire i file e trasformarli in stringa.



Classe LocalException

Un'eccezione che serve a mandare messaggi all'utente nel caso di un errato setup nell'utilizzo del tool.

		LocalException	
		serialVersionUID	long
		LocalException(String)	

Powered by yFiles

Classe MetricEvaluation

Raccoglie informazioni dalle strutture dati e calcola il valore di diverse metriche rispettivamente ai domini in cui queste metriche sono definite (metodi, classi, packages, project)

MetricEvaluation		
f	tree	CommentTree
f	graph	DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>
m	MetricEvaluation(DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>, CommentTree)	
m	metrics()	void
m	stampComplexity()	void
m	stampLoc()	void
m	stampClassInfo()	void
m	ditcEvaluation()	void
m	wmccEvaluation()	void
m	nocEvaluation()	void
m	cboEvaluation()	void
m	loccEvaluation()	void
m	lcomcEvaluation()	void
m	ncpEvaluation()	void
m	ceEvaluation()	void
m	caEvaluation()	void
m	ccEvaluation()	void
m	acEvaluation()	void
m	aEvaluation()	void
m	iEvaluation()	void
m	ditpEvaluation()	void
m	wmcpEvaluation()	void
m	nmpEvaluation()	void
m	nmcEvaluation()	void
m	rfcEvaluation()	void
m	nmsEvaluation()	void
m	ficEvaluation()	void
m	fipEvaluation()	void
m	locpEvaluation()	void
m	lcompEvaluation()	void
m	ncEvaluation()	void
m	ncpackageEvaluation()	void
m	stamPackageInfo()	void

Class Parameters

La classe che si occupa di gestire il parsing dei parametri passati in input da riga di comando.

Parameters		
f	inputClasses	StringHolder
f	classFilter	BooleanHolder
f	path	StringHolder
f	java	IntHolder
f	parser	ArgParser
f	reportFilePath	StringHolder
f	createDataset	BooleanHolder
f	predictBugPresence	BooleanHolder
f	commentsGUI	BooleanHolder
f	sourcePath	StringHolder
Parameters(String[], String)		
m	init()	void
m	classFilterEnabled()	boolean
m	usage()	void
m	print()	void
m	haveReportFilePath()	boolean
m	createDatasetEnabled()	boolean
m	predictBugPresenceEnabled()	boolean
m	commentsGUIEnabled()	boolean
m	haveSourcePath()	boolean
p	sourcePath	String
p	inputClasses	String[]
p	reportFilePath	String
p	javaVersion	int
p	projectPath	String

Classe Parser

Contiene tutti i metodi per fare il parsing di diverse informazioni all'interno dei file sorgenti.

Parser		
f	_javaVersion	String
f	_classpath	String[]
f	_classpathSeparator	String
f	_jreHome	String
m	Parser(int)	
m	addClasspath(String)	void
m	addClasspaths(String[])	void
m	addClasspaths(String)	void
m	getClasspath(boolean)	String
m	parseVariables(String, String, String, String, Hashtable<Integer, Record>)	void
m	parseMethodDeclaration(String, String, String, String, DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>)	void
m	parseMethodInvocation(String, String, String, String, DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>)	void
m	parseComment(String, String, String, String, CommentTree)	void
m	print()	void

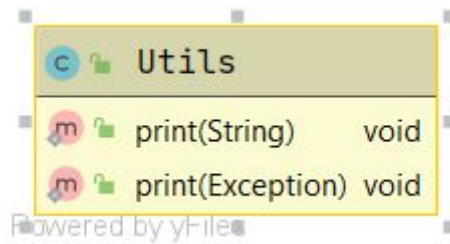
Classe Project

La classe che si occupa della gestione del progetto dato in input al tool.

Project		
m	Project(Parameters)	
m	print()	void
m	scan()	void
p	projectDir	String
p	sourcePath	String
p	libraryPath	String
p	projectPath	String
p	binaryPath	String
p	sourceFiles	List<String>
p	projectName	String

Classe Utils

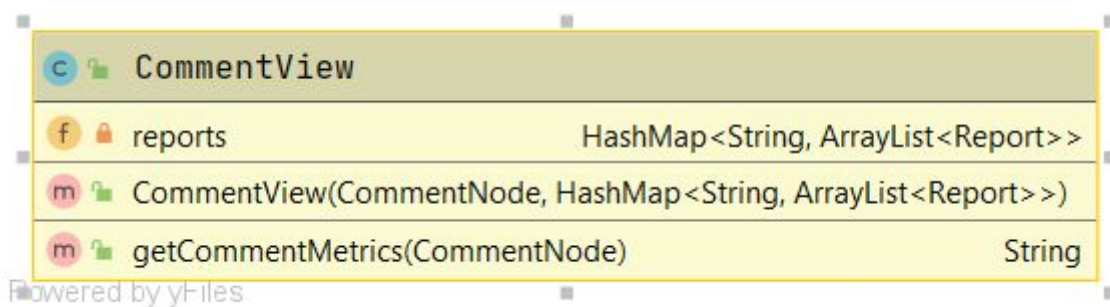
Contiene le utility per l'applicazione come la stampa formattata.



Package GUI

Classe CommentiView

Permette di visualizzare in dettaglio le metriche e le informazioni dei commenti (oggetti di tipo `CommentNode`)



Classe CompoundIcon

Implementa l'interfaccia Icon e permette di ottenere un'icona composta da due o più diverse icone, dando la possibilità di scegliere (attraverso i metodi esposti) allineamento, spazio tra le icone e asse su cui posizionare le icone aggiuntive, permettendo di creare icone in modo dinamico in base ai modificatori degli elementi.

CompoundIcon		
f	TOP	float
f	LEFT	float
f	CENTER	float
f	BOTTOM	float
f	RIGHT	float
f	icons	Icon[]
m	CompoundIcon(Icon...)	
m	CompoundIcon(Axis, Icon...)	
m	CompoundIcon(Axis, int, Icon...)	
m	CompoundIcon(Axis, int, float, float, Icon...)	
m	getIcon(int)	Icon
m	paintIcon(Component, Graphics, int, int)	void
m	getOffset(int, int, float)	int
p	iconWidth	int
p	alignmentY	float
p	iconHeight	int
p	alignmentX	float
p	iconCount	int
p	axis	Axis
p	gap	int

Classe Dynamic Tree

Espone i metodi per creare, modellare ed esplorare l'albero, ovvero l'oggetto JTree in esso contenuto.

DynamicTree		
f	rootNode	DefaultMutableTreeNode
f	treeModel	DefaultTreeModel
f	tree	JTree
f	toolkit	Toolkit
f	sourceCodeArea	RSyntaxTextArea
f	editorPaneComments	JEditorPane
f	textScrollPaneInfo	RTextScrollPane
m	DynamicTree(DefaultMutableTreeNode, TreeWillExpandListener, TreeSelectionListener, MouseListener)	
m	clear()	void
m	removeCurrentNode()	void
m	addObject(Object)	DefaultMutableTreeNode
m	addObject(DefaultMutableTreeNode, Object)	DefaultMutableTreeNode
m	addObject(DefaultMutableTreeNode, Object, boolean)	DefaultMutableTreeNode
m	createImageIcon(String)	ImageIcon
m	getFormattedComment(String)	String
m	displayNodeTextInfo(String, int)	void
m	displayNodeTextComments(String)	void

Classe Dynamic TreeWindow

la classe che crea la GUI vera e propria (istanziando un oggetto JFrame) ed è delegata all'interazione tra la GUI e la business logic.

DynamicTreeWindow		
f	treePanel	DynamicTree
f	tree	CommentTree
f	reports	HashMap<String, ArrayList<Report>>
m	DynamicTreeWindow(CommentTree, HashMap<String, ArrayList<Report>>)	
m	createAndShowGUI()	void
m	appendRoot()	DefaultMutableTreeNode
m	appendChildOf(DefaultMutableTreeNode)	void

Package Metrics

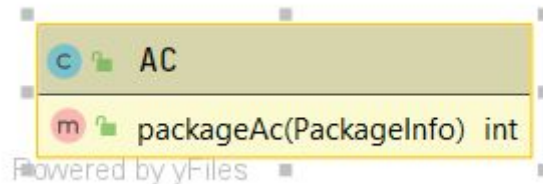
Classe A

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta l'astrattezza di un package.



Classe AC

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il numero di classi astratte in un package.



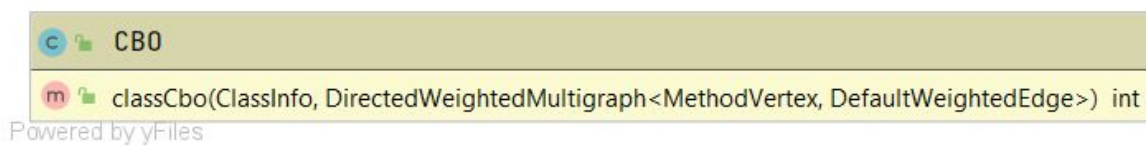
Classe Ca

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il coupling afferente di un package



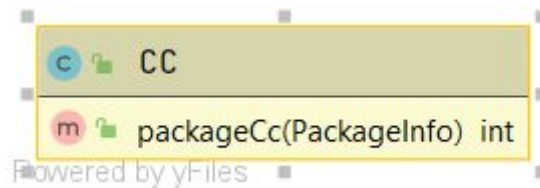
Classe CBO

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta la coesione tra tutti gli oggetti di una classe.



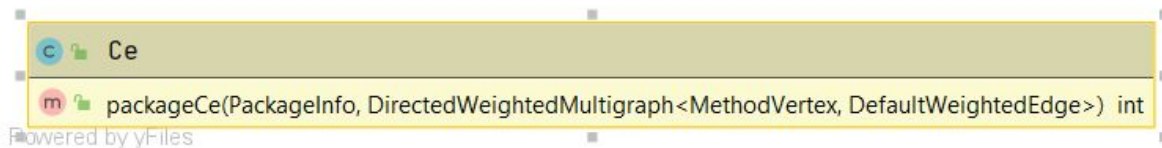
Classe CC

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il numero di classi concrete di un package.



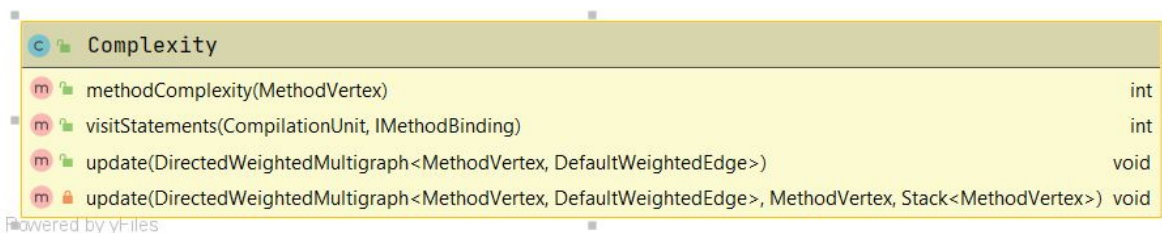
Classe Ce

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il coupling efferente di un package.



Classe Complexity

Classe che contiene tutte le operazioni per il calcolo della complessità ciclomatica.



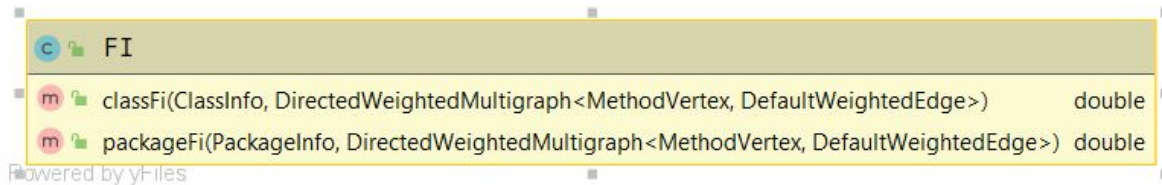
Classe DIT

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta la profondità dell'albero dell'ereditarietà delle classi.



Classe FI

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il valore di Fan In e Fan Out.

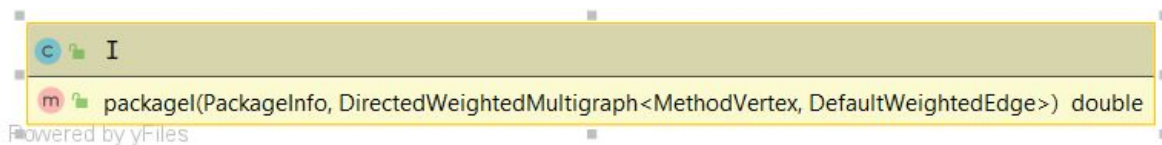


UML class diagram for the FI class. The class is named FI and contains two methods: classFi and packageFi. Both methods take ClassInfo and DirectedWeightedMultigraph as parameters and return a double value.

Method	Parameters	Return Type
classFi	ClassInfo, DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>	double
packageFi	PackageInfo, DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>	double

Classe I

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta l'instabilità di un package.

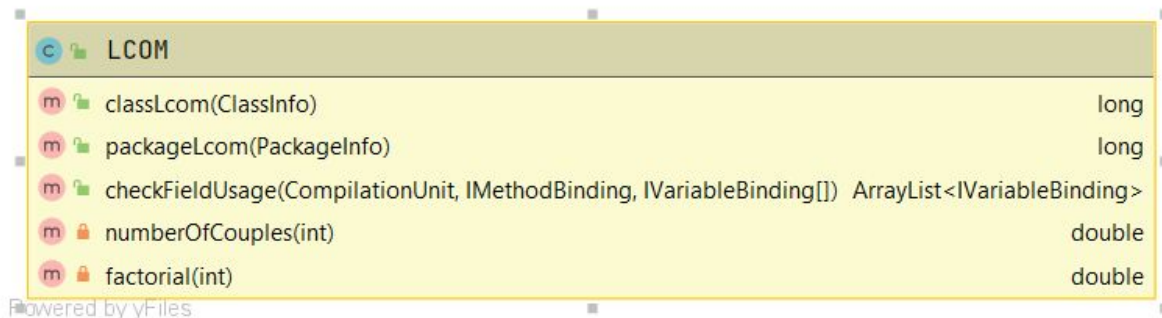


UML class diagram for the I class. The class is named I and contains one method: packageI. The method takes PackageInfo and DirectedWeightedMultigraph as parameters and returns a double value.

Method	Parameters	Return Type
packageI	PackageInfo, DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>	double

Classe LCOM

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta la mancanza di coesione tra i metodi della classe.

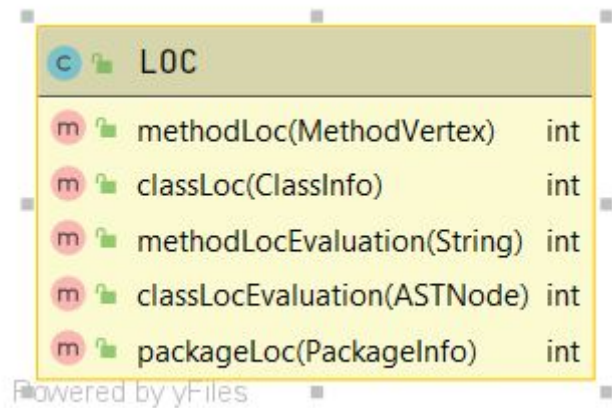


UML class diagram for the LCOM class. The class is named LCOM and contains five methods: classLcom, packageLcom, checkFieldUsage, numberOfCouples, and factorial. The first two methods return long, checkFieldUsage returns ArrayList<IVariableBinding>, and the last two return double.

Method	Parameters	Return Type
classLcom	ClassInfo	long
packageLcom	PackageInfo	long
checkFieldUsage	CompilationUnit, IMethodBinding, IVariableBinding[]	ArrayList<IVariableBinding>
numberOfCouples	int	double
factorial	int	double

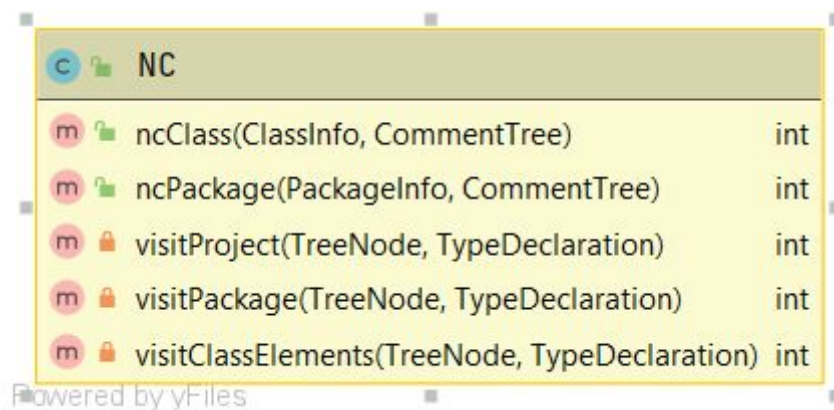
Classe LOC

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia le linee di codice.



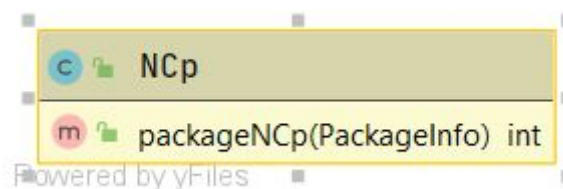
Classe NC

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia i commenti.



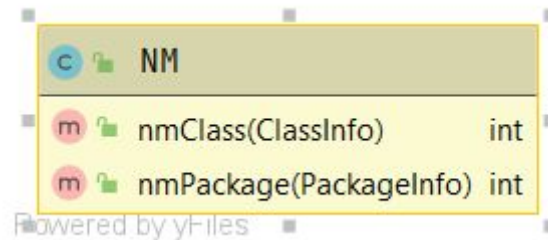
Classe NCp

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia il numero di classi di un package.



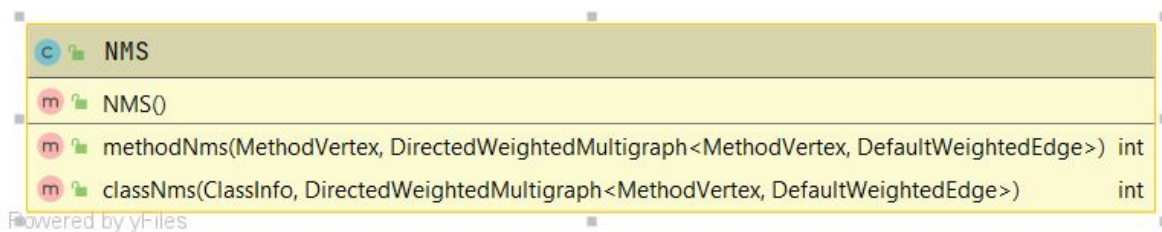
Classe NM

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia il numero di metodi.



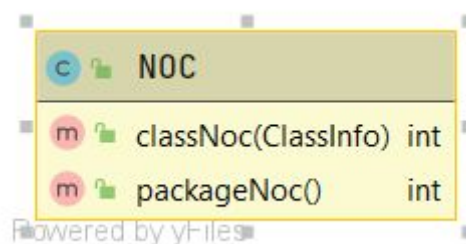
Classe NMS

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia il numero di messaggi inviati da una classe.



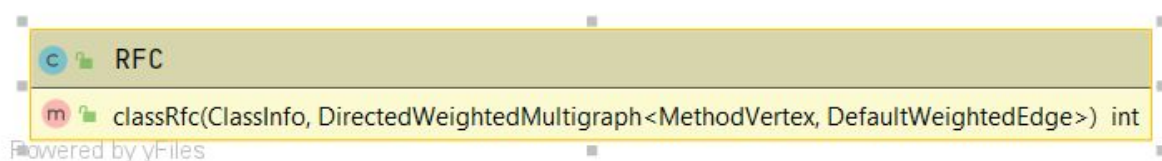
Classe NOC

Classe che contiene tutte le operazioni per il calcolo della metrica che conteggia il numero di figli di una classe.



Classe RFC

Classe che contiene tutte le operazioni per il calcolo della metrica che valuta quanti messaggi invia una classe.



Classe WMC

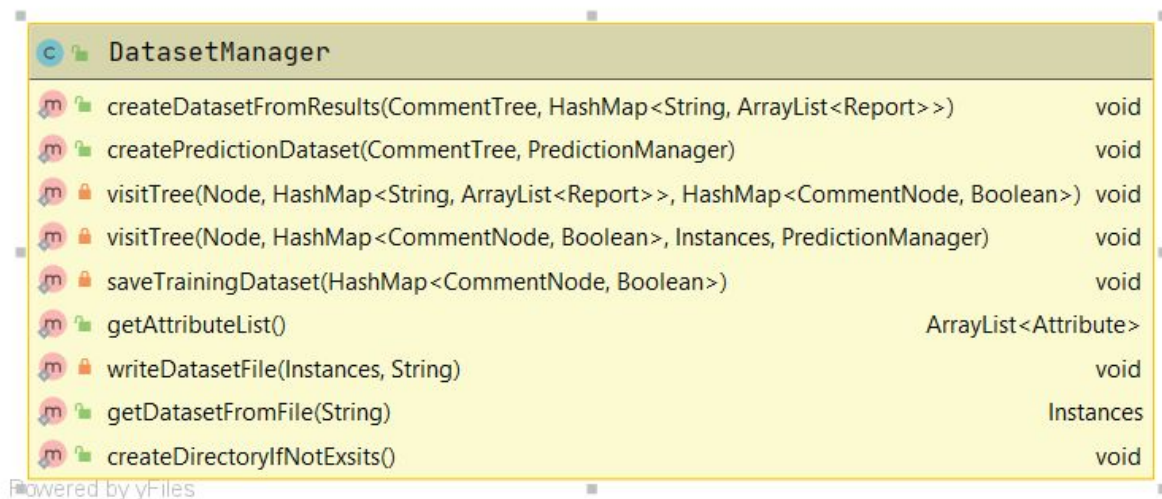
Classe che contiene tutte le operazioni per il calcolo della metrica che valuta il peso dei metodi di una classe.



Package ML

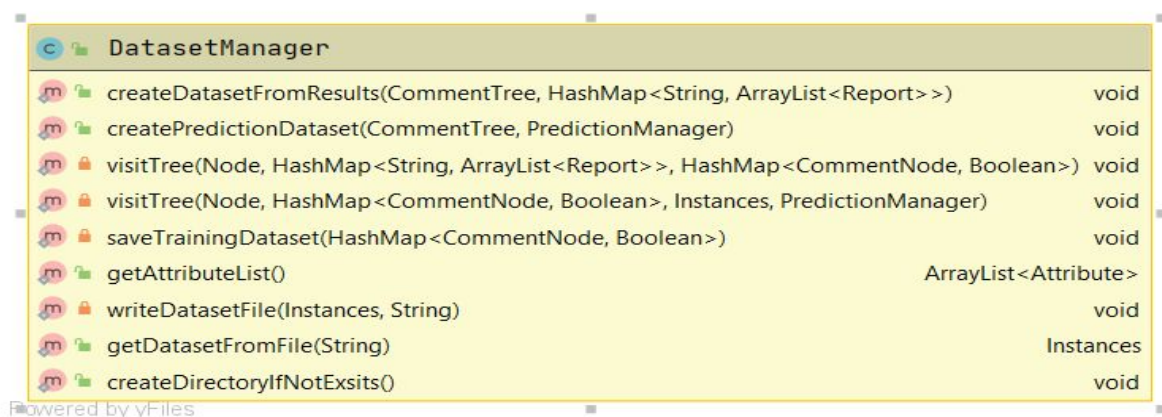
Classe DatasetConstants

Contiene le costanti utilizzate nel package.



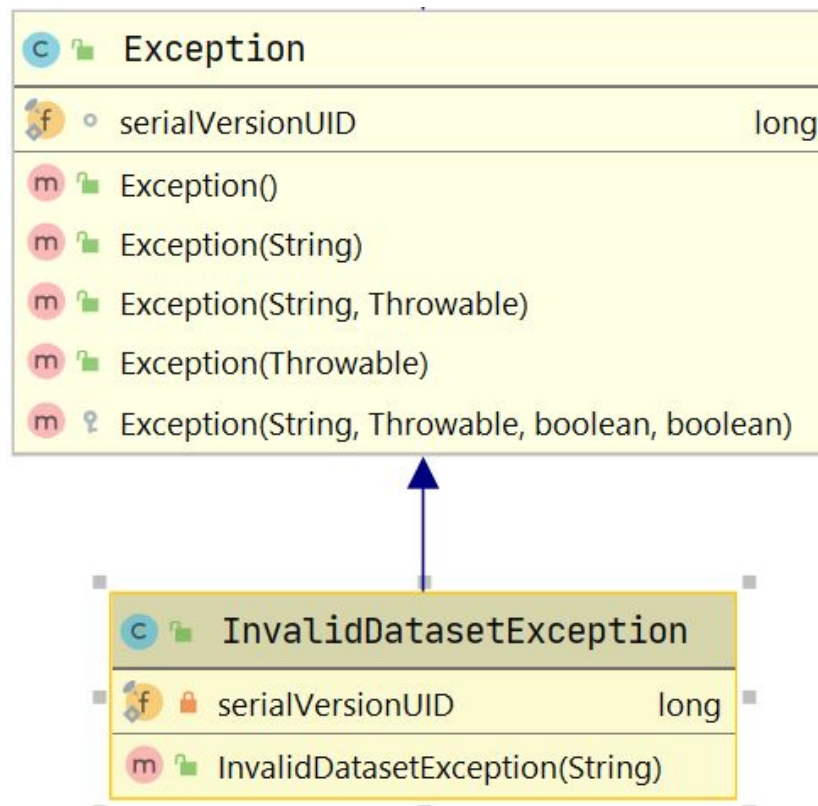
Classe DatasetManager

Espone i metodi per creare e modellare datasets partendo dai dati individuati nella fase di parsing e successivamente di visit.



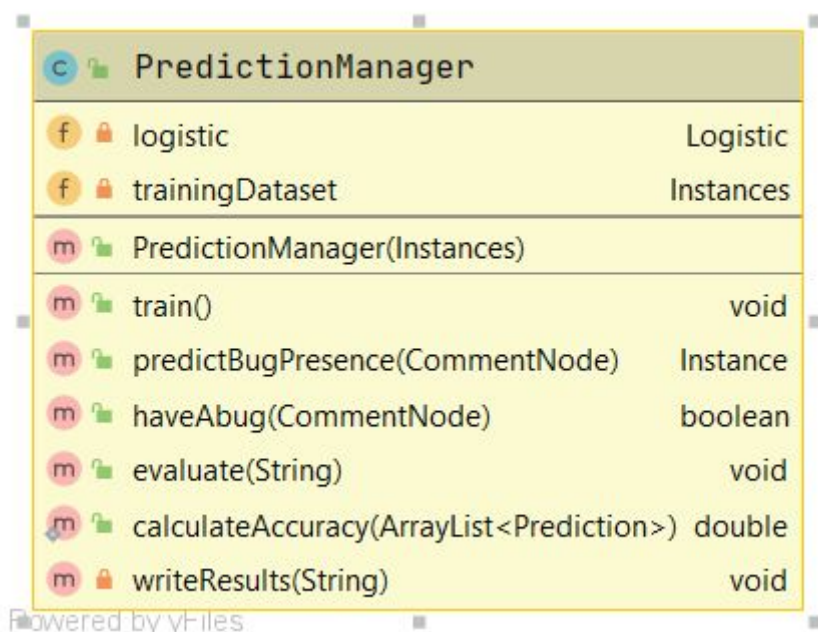
Classe InvalidDatasetException

Rappresenta le eccezioni che si possono verificare durante l'analisi o la costruzione di un dataset.



Classe PredictionManager

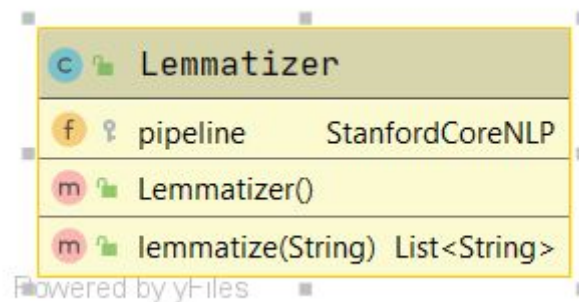
Classe che permette di costruire il modello di regressione logistica e di allenarlo allo scopo di eseguire predizioni e quindi di assegnare etichette alle istanze non etichettate.



Package NLP

Classe Lemmatizer

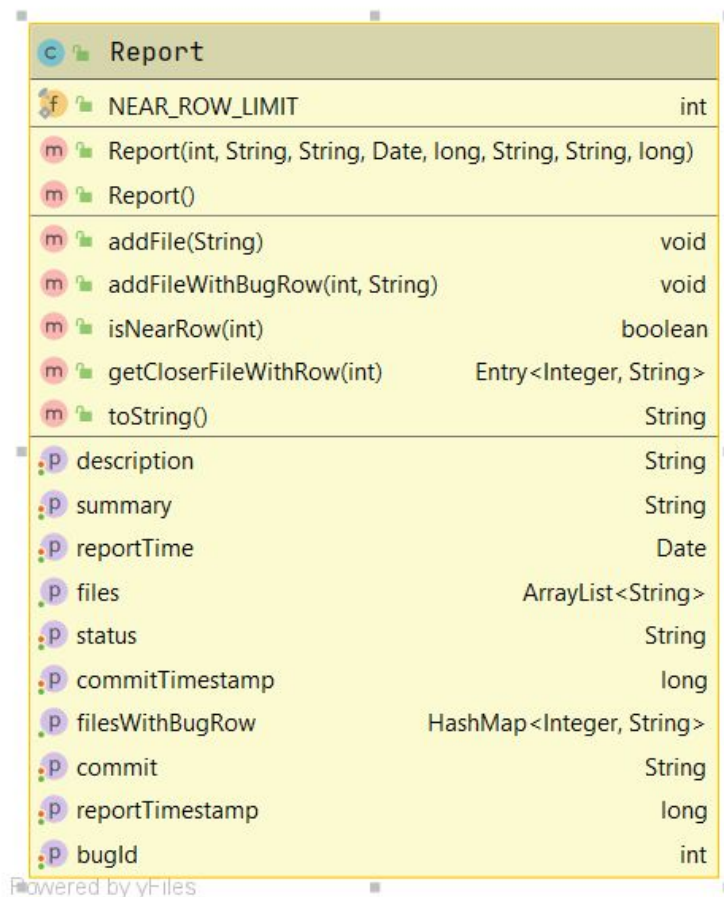
Classe, utilizzata per il calcolo di alcune metriche, che espone un metodo di lemmatization che fa uso della libreria coreNLP sviluppato dalla Stanford university.



Package Report

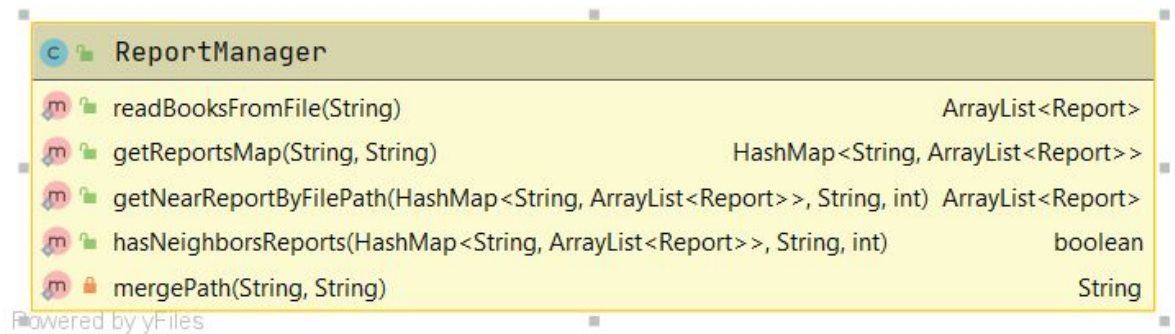
Classe Report

Rappresenta un record di bug report



Classe ReportManager

Permette di effettuare il parsing di report da file.



Package Struct

Package Graph

ClassInfo

Oggetto di supporto che rappresenta le classi del sistema analizzato.

MethodVertex

Classe che rappresenta il vertice del grafo.

PackageInfo

Oggetto di supporto che rappresenta i pacakage del sistema analizzato.

Vertex

Interfaccia che contiene alcune costanti.

Vertex		
	PUBLIC	int
	PRIVATE	int
	PROTECTED	int
	STATIC	int
	FINAL	int
	ABSTRACT	int
	SYNCHRONIZED	int
	NATIVE	int
	STRICTFP	int

MethodVertex		
	MethodVertex(CompilationUnit, MethodDeclaration, IMethodBinding)	
	MethodVertex(CompilationUnit, IMethodBinding)	
	toString()	String
	equals(Object)	boolean
	hashCode()	int
	setUpdated()	void
	methodFieldAccessed	ArrayList<IVariableBinding>
	loc	int
	line	int
	updated	boolean
	name	String
	sourceClass	ITypeBinding
	modifiersToString	String[]
	classInfo	ClassInfo
	modifiers	int
	exceptions	ITypeBinding[]
	parameters	ITypeBinding[]
	returnType	ITypeBinding
	complexity	int

Powered by yFiles

ClassInfo		
	classList	ArrayList<ClassInfo>
	updated	boolean
	ANONYMOUS_CLASS_DECLARATION	int
	ClassInfo()	
	ClassInfo(ITypeBinding, ASTNode)	
	addChildren(ClassInfo)	void
	findParent(ITypeBinding)	void
	toString()	String
	update()	void
	addMethod(MethodVertex)	void
	classList	ArrayList<ClassInfo>
	loc	int
	anonymous	boolean
	superClass	boolean
	classBinding	ITypeBinding
	name	String
	methods	ArrayList<MethodVertex>
	abstract	boolean
	parent	ClassInfo
	interface	boolean
	children	ArrayList<ClassInfo>
	concrete	boolean
	classASTNode	ASTNode
	package	PackageInfo
	complexity	int

PackageInfo		
	packageList	ArrayList<PackageInfo>
	PackageInfo()	
	PackageInfo(PackageDeclaration)	
	addClass(ClassInfo)	void
	calculateLoc()	void
	update()	void
	toString()	String
	name	String
	classes	ArrayList<ClassInfo>
	packageDeclaration	PackageDeclaration
	loc	int
	packageList	ArrayList<PackageInfo>

Package table

Classe Record

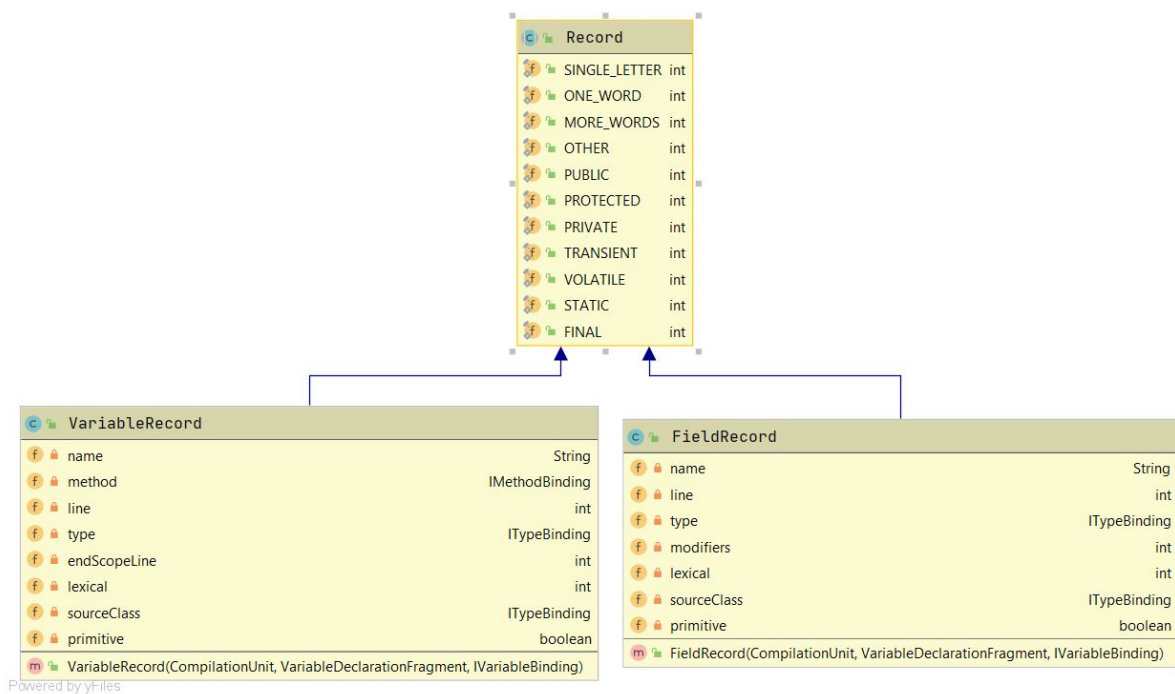
Classe astratta che contiene alcune costanti e la firma dei metodi implementati nelle sottoclassi.

Classe FieldRecord

Classe che rappresenta la entry nella tabella di un attributo di classe.

Classe VariableRecord

Classe che rappresenta la entry nella tabella di una variabile. .



Package tree

CommentTree

La classe principale che rappresenta l'albero.

Node

Classe astratta che rappresenta il nodo dell'albero.

ProjectNode

Classe che rappresenta la radice dell'albero nonché il progetto.

PackageNode

Classe che rappresenta il nodo package nell'albero.

PackageNodeData

Classe che rappresenta l'informazione contenuta in un PackageNode.

CommentNode

Classe che rappresenta tutti gli altri tipi di nodi non già citati, come classi, attributi, metodi, ecc.

CommentNodeData

Classe che rappresenta l'informazione contenuta in un CommentNode.

CommentTree	
serialVersionUID	long
CommentTree(Node)	
createTree(String)	CommentTree
insertNode(Node)	boolean
checkPackage(Node, String[], int)	void
getPackage(Node, String[], int)	PackageNode
findNode(ASTNode)	Node
containsNode(ASTNode)	boolean
getNodeByASTNode(Node, ASTNode)	CommentNode
replaceNode(Node, Node)	void
toString()	String
treeToString(Node, int, StringBuilder)	void
parentToString(Node, int)	String
nodeToString(CommentNode)	String

CommentNode	
data	CommentNodeData
fileName	String
CommentNode(ASTNode, String, int, int, int, String)	
CommentNode(ASTNode)	
assignType(ASTNode)	int
addComment(String)	void
createLemmaMap(String)	Map<String, Integer>
getusedWordsMap()	Map<String, Integer>
toString()	String
horizontalDistanceFromParent	int
parentLineNumber	int
filePath	String
objectCommented	ASTNode
verticalDistanceFromParent	int
realDistanceFromParent	int
numberOfWords	int
commentList	List<String>

Node	
PROJECT	int
PACKAGE	int
CLASS	int
FIELD	int
METHOD	int
ENUM	int
OTHER	int
children	List<Node>
insertChild(Node)	void
children()	Enumeration<Node>
getChildAt(int)	TreeNode
getIndex(TreeNode)	int
allowsChildren	boolean
type	int
leaf	boolean
parent	TreeNode
childCount	int

CommentNodeData	
CommentNodeData(ASTNode, String)	
CommentNodeData(ASTNode)	
addComment(String)	void
objectCommented	ASTNode
commentList	List<String>

PackageNodeData	
PackageNodeData(String)	
packageName	String

PackageNode	
data	PackageNodeData
PackageNode(String)	
packageName	String

ProjectNode	
ProjectNode(String)	
projectName	String

Package Visitors

Classe CommentVisitor

Classe che raccoglie informazioni su tutti i commenti e li associa al giusto elemento creando i nodi e inserendoli nell'albero

CommentVisitor		
f	vector	Vector<ASTNode>
f	tree	CommentTree
f	cu	CompilationUnit
f	file	String
f	filePath	String
m	CommentVisitor(CompilationUnit, String, Vector<ASTNode>, CommentTree, String)	
m	visit(LineComment)	boolean
m	visit(BlockComment)	boolean
m	visit(Javadoc)	boolean
m	getDistanceFromParent(ASTNode, Comment)	int
m	getDistanceFromParent(ASTNode, Javadoc)	int
m	getHorizontalDistanceFromParent(ASTNode, Comment)	int
m	getRealDistanceFromParent(ASTNode, Comment, int)	int
m	getInlineOrInternalDistanceFromParent(ASTNode, Comment)	int
m	rTrim(String)	String
m	rightWiteSpaces(String)	int
m	fixNodePosition(ASTNode)	void

Classe ElementVisitor

Classe che raccoglie, riga per riga, informazioni sul file sorgente, mettendo queste informazioni in una struttura dati di supporto che servirà per attribuire ogni commento al suo elemento commentato.

ElementVisitor		
f	vector	Vector<ASTNode>
m	ElementVisitor(Vector<ASTNode>)	
m	preVisit(ASTNode)	void

Classe FieldAndVariableVisitor

Classe che raccoglie informazioni su variabili e attributi di classe e li inserisce nell'apposita struttura dati.

FieldAndVariableVisitor		
f	table	Hashtable<Integer, Record>
m	FieldAndVariableVisitor(Hashtable<Integer, Record>)	
m	visit(FieldDeclaration)	boolean
m	visit(VariableDeclarationStatement)	boolean

Powered by yFiles

Classe MethodDeclarationVisitor

Classe che raccoglie informazioni su tutti i metodi di una classe e crea i vertici del grafo.

MethodDeclarationVisitor		
f	graph	DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>
m	MethodDeclarationVisitor(DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>)	
m	visit(TypeDeclaration)	boolean
m	visit(AnonymousClassDeclaration)	boolean

Powered by yFiles

Classe MethodInvocationVisitor

Classe che raccoglie informazioni su tutte le invocazioni dei metodi e crea gli archi del grafo.

MethodInvocationVisitor		
f	graph	DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>
m	MethodInvocationVisitor(DirectedWeightedMultigraph<MethodVertex, DefaultWeightedEdge>)	
m	visit(SuperMethodInvocation)	boolean
m	visit(SuperConstructorInvocation)	boolean
m	visit(ConstructorInvocation)	boolean
m	visit(ClassInstanceCreation)	boolean
m	visit(MethodInvocation)	boolean
m	findGraphVertex(MethodVertex)	MethodVertex

Powered by yFiles

Classe SimpleNameVisitor

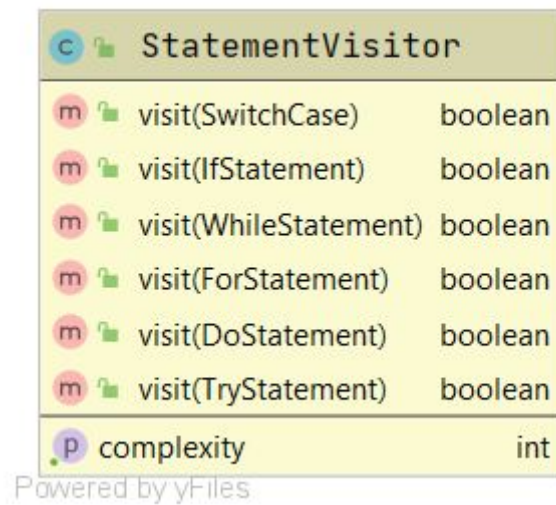
Classe che raccoglie informazioni su tutti i simple name in modo da poter analizzare l'utilizzo dei field all'interno dei metodi di ciascuna classe.

SimpleNameVisitor		
f	classFieldDeclaration	IVariableBinding[]
m	SimpleNameVisitor(IVariableBinding[])	
m	visit(SimpleName)	boolean
p	fieldAccessed	ArrayList<IVariableBinding>

Powered by yFiles

Classe StatementVisitor

Classe che raccoglie informazioni su tutti gli statements decisionali; viene utilizzata per il calcolo della complessità ciclomatica



Storage

A seguito dell'analisi effettuata nel reverse engineering, si evince che il tool Metric 3.0 non effettua nessuna storicizzazione delle analisi fatte. Non vi è nessuna presenza di file o database dove vengono memorizzati i dati.

Object design trade-offs

Analizzando il sistema, sono stati definiti dei trade-off che dovrebbero rispecchiare le scelte effettuate dai progetti originali del sistema metric 3.0.

Prestazioni vs Costi

L'intero sistema è basato su componenti open-source, tale scelta è molto significativa, in quanto abbassa i costi di realizzazione ma va a discapito delle prestazioni e dell'affidabilità dell'intero sistema.

Interfaccia vs usabilità

L'attuale interfaccia utente, la quale è un'interfaccia testuale, risulta essere molto minimale e semplice da utilizzare ma allo stesso tempo risulta essere poco intuitiva e poco human-friendly. Di conseguenza l'interfaccia risulta essere poco adatta agli utenti meno esperti.

Sicurezza vs costi

L'attuale sistema non fa uso di dati ritenuti sensibili, quindi non prevede meccanismi di sicurezza e protezione dei dati.

Linee guida per la documentazione dell'interfaccia

A seguito dell'analisi di reverse engineering, si è evidenziato che i progettisti del sistema originale hanno rispettato la tipica naming convention del linguaggio Java.

Di seguito sono riportate le linee guida nel dettaglio:

Variabili:

- I nomi delle variabili seguono la "CamelNotation";
- Ogni variabile è dichiarata su un'unica riga;
- Il carattere underscore "_" è relativo alla sola dichiarazione di costanti.

Metodi:

- I nomi dei metodi seguono la "CamelNotation";
- I metodi sono raggruppati in base alle loro funzionalità;
- Ogni metodo ha una descrizione delle sue funzionalità.

Classi:

I nomi delle classi iniziano con lettera maiuscola, così come le parole successive all'interno del nome; I nomi delle classi sono esplicativi e senza abbreviazioni.

Packages:

- I nomi dei packages iniziano con lettera minuscola;
- Non sono ammessi caratteri speciali;

La documentazione segue la convenzione di Javadoc:

- La documentazione minima dovrebbe comprendere la descrizione di ciascuna classe, interfaccia, metodo pubblico, attributo pubblico;
- Un commento Javadoc deve descrivere in modo sintetico lo scopo dell'oggetto che si sta documentando;
- Un commento Javadoc può contenere tag HTML per aiutare la formattazione, anche se si sconsiglia di adottare tag di presentazione come , <i>, in favore di tag di struttura come , , ecc.
- Il formato standard dei Javadoc è HTML e non richiede nessun file ausiliario.
- Ogni commento Javadoc deve essere formattato seguendo le seguenti linee guida:
 - Una prima parte composta da un paragrafo che riassume lo scopo del metodo;
 - Eventuali paragrafi successivi sono suddivisi da <p>;
 - Una riga vuota che separa la parte descrittiva dalla parte dei tag Javadoc. La riga vuota deve essere solo una;
 - Una seconda parte in cui sono inseriti tag Javadoc (ad esempio, @param) che identificano le componenti dell'oggetto che si sta commentando.

Definizioni, acronimi e abbreviazioni

- Package: collezione di classi e interfacce correlate.
- Javadoc: uno strumento che permette di documentare i sorgenti di un programma. all'interno dei sorgenti stessi.
- HTML: Linguaggio di mark-up per pagine web.
- Storage: persistenza dei dati all'interno di un sistema informativo.