

第 編

高度情報化支援ソフトウェアシーズ育成事業

「オブジェクト指向ソフトウェア向け
リファクタリングツールの開発」

ソフトウェア設計書

- 目 次 -

1 .	システムの概要.....	1
1 . 1	開発目的.....	1
1 . 2	概要説明.....	1
1 . 3	全体構成.....	1
1 . 4	開発対象範囲.....	2
2 .	設計指針	2
3 .	システム構成	3
3 . 1	ハードウェア環境.....	3
3 . 2	ソフトウェア環境.....	3
3 . 3	ネットワーク環境.....	3
3 . 4	機能説明.....	3
3 . 5	機能ブロックの構成	3
3 . 6	機能ブロック間の関連.....	4
3 . 7	機能ブロックの概要説明	5
4 .	入出力仕様.....	7
4 . 1	画面仕様.....	7
4 . 2	帳票仕様.....	33
4 . 3	エラーメッセージ.....	34
5 .	機能仕様	37
5 . 1	JAVA ソースコードモデル.....	37
5 . 1 . 1	JavaComponent クラス.....	37
5 . 1 . 2	JavaFile クラス	39
5 . 1 . 3	JavaClass クラス.....	40
5 . 1 . 4	JavaMethod クラス.....	42
5 . 1 . 5	JavaStatement クラス.....	44

5.1.6	JavaVariable クラス	45
5.2	グラフ	47
5.2.1	Graph クラス	48
5.2.2	GraphNode クラス	49
5.2.3	GraphEdge クラス	50
5.3	制御フローグラフ	50
5.3.1	CCFG クラス	51
5.3.2	CFG クラス	51
5.3.3	CFGNode クラス	53
5.3.4	CFGClassEntryNode クラス	54
5.3.5	CFGMethodEntryNode クラス	55
5.3.6	CFGExitNode クラス	56
5.3.7	CFGMergeNode クラス	56
5.3.8	CFGStatementNode クラス	56
5.3.9	CFGAssignmentNode クラス	57
5.3.10	CFGBranchNode クラス	57
5.3.11	CFGCallNode クラス	57
5.3.12	CFGParameterNode クラス	58
5.3.13	Flow クラス	58
5.4	プログラム依存グラフ	59
5.4.1	PDG クラス	60
5.4.2	PDGNode クラス	60
5.4.3	PDGClassEntryNode クラス	61
5.4.4	PDGMethodEntryNode クラス	61
5.4.5	PDGStatementNode クラス	61
5.4.6	Dependence クラス	62

5.4.7	CD クラス	62
5.4.8	DD クラス	63
5.4.9	BindingEdge クラス	64
6 .	リファクタリング部	65
6.1	構文解析部	65
6.1.1	JavaModelFactory クラス	66
6.1.2	CFGFactory クラス	68
6.2	依存解析部	68
6.2.1	CDGFactory クラス	69
6.2.2	DDGFactory クラス	69
6.2.3	Slice クラス	69
6.3	コード変換部	69
6.3.1	Refactoring クラス (抽象クラス)	70
6.3.2	ClassRefactoring クラス	72
6.3.3	MethodRefactoring クラス	72
6.3.4	FieldRefactoring クラス	72
6.3.5	VariableRefactoring クラス	73
6.3.6	MiscellaneousRefactoring クラス	73
6.3.7	RefactoringImpl クラス	73
6.3.8	RefactoringVisitor クラス	77
6.3.9	PrintVisitor クラス	78
6.4	リファクタリング操作部	79
6.4.1	Refactor クラス	79
6.4.2	PositionVisitor クラス	80
6.5	操作列検索部	81
6.5.1	RefactoringHistory クラス	81

7 .	ユーザインタフェース(GUI)部.....	83
7 . 1	編集機能.....	83
7 . 1 . 1	TextPane クラス.....	83
7 . 1 . 2	TextUndoManager クラス.....	85
7 . 2	イベント取得機能.....	85
7 . 2 . 1	TabbedTextPane クラス.....	86
7 . 2 . 2	RefactoringMenu クラス.....	87
8 .	性能	88
9 .	運用方法	88

1. システムの概要

1.1 開発目的

本研究開発の目的は、プログラマの負担を増加させずに、高品質なソフトウェアの開発および既存ソフトウェアの保守ができるリファクタリング支援ツールを提供することである。リファクタリングとは、ソフトウェアの外部的振る舞いを保存したまま、内部のソフトウェアの構造を変換することで、設計やソースコードの理解性や変更容易性を改善することである。

リファクタリングの有効性は従来から指摘されているが、その操作は人間が行うには非常に煩雑である。さらに、手動リファクタリングでは、その作業中に誤りが混入する恐れがあり、プログラマはリファクタリング後に必ずテストを行うことになる。リファクタリングの自動化ツールを開発し、普及させることで、このような面倒な作業からプログラマは解放され、高品質なソフトウェアの開発および既存ソフトウェアの保守が容易になる。

1.2 概要説明

開発ソフトウェアは、Java 言語で記述されたオブジェクト指向ソフトウェアに対して、プログラマの意図するリファクタリング操作を自動実行するツールである。従来のソフトウェア開発環境とほぼ同等の編集機能に、プログラム解析に基づく自動リファクタリング機能を追加した。

1.3 全体構成

開発ソフトウェアの構成を図1に示す。以下、このソフトウェアを JRB(Java Refactoring Browser)と呼ぶ。

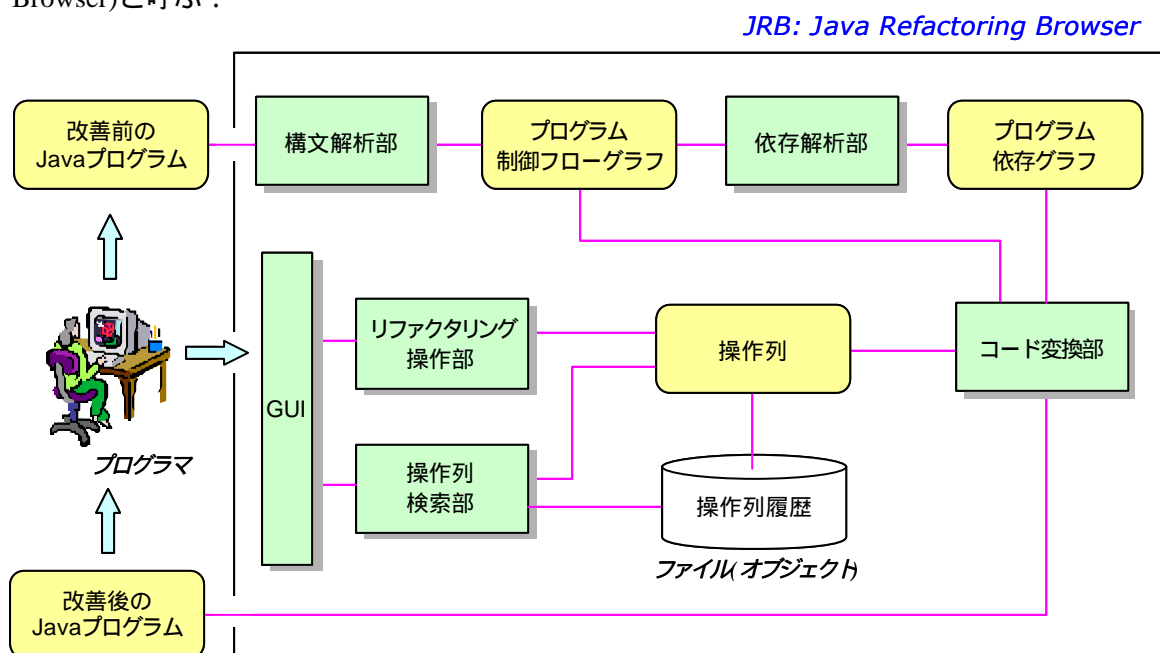


図1 研究開発システムの機能構成

JRB は大きく (1) リファクタリング部と (2) ユーザインタフェース(GUI)部で構成されている。さらに、リファクタリング部は次に示す 5 つのモジュールで構成されている。

- (1) 構文解析部
- (2) 依存解析部
- (3) コード変換部
- (4) リファクタリング操作部
- (5) 操作列検索部

1.4 開発対象範囲

本研究開発では、図 1 において四角形で表した 6 つのモジュール (1.3 の 5 つのモジュールとユーザインタフェース部) についてソフトウェアを開発する。

2. 設計指針

(1) 基本方針

リファクタリングの普及を目指し、できるかぎりプログラマが簡単に実行できる環境を想定し、ソースコードエディタからマウスによるコード(テキスト)指定とメニュー選択でリファクタリングができるようにする。また、自動化の効果をより多く体験できるように、単純なものから複雑なものまで幅広くリファクタリングメニューを提供する。

(2) 研究的要素

リファクタリング研究の可能性を示唆する目的で、自動化が難しいとされているリファクタリング操作を、プログラム解析技術により実現することを試み、メニュー項目に試験的に組み入れる。

(3) 実装方式

今後の予定として開発ソフトウェア自身をリファクタリングの実験対象とすることを考え、オブジェクト指向言語 Java を用いて実装する。

(4) 拡張性

あらかじめリファクタリングメニューを決定するのではなく、プログラマが自分にあつたリファクタリングメニューを用意できるリファクタリングビルダーへの拡張を考慮し、個々の操作の実現モジュールを単独で起動できるようにする。

(5) 「操作列検索部」(発注仕様書「4.1.5 操作列生成部」) に関して

リファクタリング計画の提示に関する研究開発において、リファクタリング目標を利用者が直接指示するよりも、その利用者の直前の操作に基づき妥当な操作を提示する方が、実際のリファクタリング支援において有効であることがわかった。そこで、本研究開発シ

システムでは、利用者からの明示的な指示がなくとも、適切なリファクタリング手順を計画する方法として、直前（１つあるいは２つ前）の操作をキーとして、あらかじめ蓄積されているリファクタリング履歴内の操作列から一致するものを検索することを考えた。一致する操作が過去に行われている場合、その操作の次の操作を、利用者が次に行うべき操作であると判断し、メニューを介して利用者に提示する。このような仕組みを取り入れるため、「操作列生成部」の入力情報を「リファクタリング目標」から「利用者の直前（１つおよび２つ前）の操作」に、出力情報を「操作列」から「（提案する）操作の集合」に変更した。さらに、機能を的確に表現しているという点から、名称を「操作列生成部」から「操作列検索部」とした。この変更に伴い、リファクタリング履歴の管理を一箇所で行うように、「リファクタリング操作部」と「操作列検索部」の設計を改善し、操作列を履歴としてライブラリ（ファイル）に蓄積する機能を、「リファクタリング操作部」から「操作列検索部」に移動した。

３． システム構成

３．１ ハードウェア環境

- (a) CPU: Intl Pentium 互換, SunSparc
- (b) メモリ: 128MB 以上
- (c) ハードディスク: 300MB 以上（ただし, OS,JDK を除く）
- (d) グラフィックカード: VGA 互換ボード, 256 色以上

３．２ ソフトウェア環境

- (a) Windows2000/98/Me, Solaris8
- (b) Sun Microsystems Java2 SDK Standard Edition, 1.3
- (c) Borland JBuilder4 Foundation(エディタとして利用する場合のみ)

３．３ ネットワーク環境

本ソフトウェアはスタンドアロンでの利用を想定しているため、ネットワークに接続する必要はない。ただし、ネットワークにつながっていてもかまわない。

３．４ 機能説明

開発ソフトウェアは、オブジェクト指向言語 Java で記述されたソースコードに対して、リファクタリング操作を自動実行するツールである。プログラム解析技術を積極的に取り入れることで、より多くのリファクタリング操作の自動化を実現する。

３．５ 機能ブロックの構成

図 2 に開発ソフトウェアの機能ブロックの構成を示す。角に小さい四角形を有する長方

3.6 機能ブロック間の関連

JRB の利用者は、エディタ上でソースコードを編集する．リファクタリングを行う場合は，ソースコードの改善対象部分をエディタ上で指定し，メニューからリファクタリング操作を選択する．図 2 において，利用者の操作は，以下のように実現される．

利用者は、メニューを介して、希望するリファクタリング対象部分と操作を選択する。本ソフトウェアでは、そのイベントをイベント取得機能(gui パッケージの TabbedTextPane クラスと RefactoringMenu クラス)において取得する。リファクタリング操作に関するイ

ベントであった場合，gui パッケージの PositionVisitor クラスによりテキストの選択範囲から選択されているソースコードの字句（トークン）の種類を判別し，その内容をリファクタリング操作部（gui パッケージの Refactor クラス）に送る．Refactor クラスは，送られた内容から適切なリファクタリングコマンドを生成し，コード変換部（refactor パッケージの Refactoring クラス）にその実行を依頼する．Refactoring クラスは，あらかじめ構文解析部（parser パッケージの JavaModelFactory と graphs.cfg パッケージの CFGFactory クラス）により構築された構文木(AST)と制御フローグラフ(CFG)を用いて，ソースコードの変換を行う．また，必要に応じて，依存解析部(graphs.pdg パッケージの CDGFactory および DDGFactory クラス)により構築されたプログラム依存グラフ(PDG)を用いる．ここで，GraphFactory クラスは，CFGFactory クラス，CDGFactory クラス，DDGFactory クラスにメソッドの処理を転送するクラス(Façade)である．

変換されたコードは，Refactor クラスを介して，gui パッケージの TextPane クラスに送られ，利用者の画面に表示される．ソースコードの変換実行後に，Refactor クラスは，操作列検索部(gui パッケージの RefactoringHistory クラス)に操作履歴の蓄積を依頼し，RefactoringHistory クラスが履歴(gui パッケージの RefactoringRecord クラスのオブジェクト)をファイルに保存する．また，イベント取得部において，リファクタリングメニューが表示されたときには，RefactoringMenu クラスから RefactoringHistory クラスに履歴の検索要求が起こり，直前の操作と履歴内の操作列でマッチングが行われる．検索結果は，RefactoringMenu クラスを介して利用者に提示される．

（２） ファイル操作

利用者は，メニューを介して，ファイルのオープン，クローズ，保存などを行う．オープンされたファイルはイベント取得機能の TabbedTextPane クラスで読み込まれ，編集機能を持つ TextPane クラスにより利用者の画面に表示される．さらに，編集機能では，ファイルの状態に応じて，構文解析部(JavaModelFactory クラスと CFGFactory クラス)を呼び出し，リファクタリング対象ソースコードの構文解析を行う．parser パッケージの JavaModelFactory クラスは，parser.ast パッケージ内のクラスを用いて AST を，model パッケージの JavaComponent のサブクラスを用いて CFG を構築する．

（３） 編集操作

画面上の個々ソースコードは，編集機能の TextPane クラスによって管理される．また，編集操作の履歴は gui パッケージの TextUndoManger クラスによって管理される．

3.7 機能ブロックの概要説明

（１） リファクタリング機能

プログラマが編集集中のソースコードに対して，メニューより選択したリファクタリング操作を適用し，変換後のソースコードを編集画面に出力する．

- 構文解析部： 入力として指定された Java ソースコードの語句と構文を解析し，各クラスに対して構文木(AST: Abstract Syntax Tree)および制御フローグラフ(CFG: Control Flow Graph)を作成する．
- 依存解析部： Java ソースコードから作成した CFG に対して，データ依存解析と制御依存解析を適用し，プログラム依存グラフ(PDG: Program Dependence Graph)を作成する．CFG の各節点に対して，制御の従属関係を調査して，制御依存関係(Control Dependence)を抽出し，制御依存グラフを作成する．さらに，変数の定義と参照の関係（データの到達可能性）を調査して，データ依存関係(Data Dependence)を抽出し，データ依存グラフを作成する．JRB では，オブジェクト指向プログラムが依存解析対象であるため，PDG を拡張したクラス依存グラフ(CIDG: Class Dependence Graph)の依存関係矢印を追加する．さらに，与えられた PDG(CIDG)において，スライシング基準として指定された節点と変数に関する依存関係矢印をたどることでスライスを計算する．ここで用いるスライスとは，着目節点の特定変数の値に影響を与えるコード断片を集めたものである
- コード変換部： 入力 Java ソースコードの AST, CFG, PDG に対して，与えられたリファクタリング操作列に従い変換操作を適用し，入力された Java ソースコードを整形する．
- リファクタリング操作部： GUI を介して与えられたリファクタリング要求（メニューから選択）に対して適切な変換操作コマンドを生成する．
- 操作列検索部： 直前(1 あるいは 2 つ前の)リファクタリング操作に応じて，過去の操作列を検索することで，利用者が行うリファクタリング手順の計画を支援する．また，リファクタリング操作を履歴としてライブラリに蓄積する．

(2) ユーザインタフェース (G U I) 部

Java ソースコードを編集するためのエディタおよびファイル操作メニューを提供する．ソースコードの編集機能を持ち，マウスおよびキーボードイベントを取得する機能を提供する．与えられたイベントに従い，リファクタリング要求を取得し，それぞれリファクタリング操作部あるいは操作列検索部に引き渡す．

- 編集機能：ソースコード（ファイル）の新規作成・読み込み・保存・名前変更・印刷機能，ソースコードの切り取り・コピー・貼り付け・挿入・削除機能，ソースコード上での文字列検索・置換機能，ソースコードに対する編集の取り消し，やり直し機能を提供する．さらに，リファクタリング前後のソースコードを組にして表示する．
- イベント取得機能： マウスイベントおよびキーボードイベントを取得する機能を提供する．与えられたイベントに従い，リファクタリング要求や変更目標を取得する．

4 . 入出力仕様

4 . 1 画面仕様

(1) JRB の基本画面

JRB の基本画面を図 3 に示す . JRB は 3 つの画面を持つ .

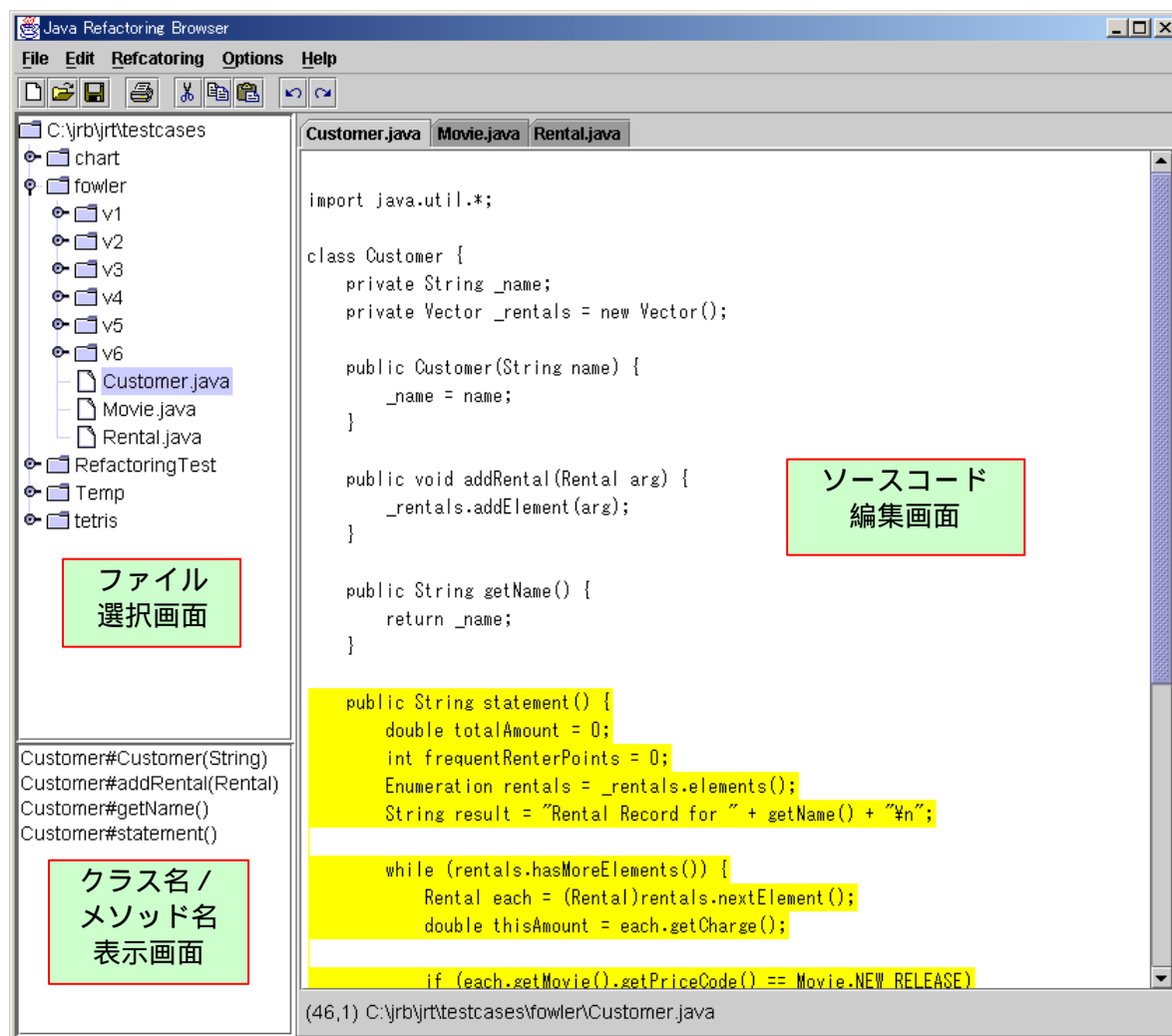


図 3 JRB の基本画面

(a) ファイル選択画面

探索ディレクトリ (環境変数 `Root.Dir` で指定されたディレクトリ) 以下の Java ソースコード (拡張子が `java` である) ファイルがディレクトリの階層に従い表示される . この画面上でファイル名をクリックすると , 指定されたファイルがファイル編集画面に読み込まれる . すでに読み込み済みのファイルの場合は , 指定ファイルの編集画面が最前面に呼び出される .

(b) ソースコード編集画面

利用者に指定されたファイルの内容 (ソースコード) が表示される . 複数のソースコード

が読み込まれている場合、それぞれのソースコードのタブをクリックすることで、指定ソースコードを最前面に呼び出すことができる。利用者は、この画面内でマウスをドラッグすることにより対象テキストを選択し、ソースコードの編集、検索、置換、リファクタリング操作を行う。

(c) クラス名/メソッド名表示画面

最前面のソースコード内に存在するクラスとメソッドを表示する。表示形式は、「クラス名#メソッド名(引数の型)」である。

(2) メニュー画面

JRB のメニュー画面を図 4 ～ 9 に示す。メニュー項目選択時の動作については、「操作説明書 3.2」に詳しい記述があるのでそちらを参考のこと。

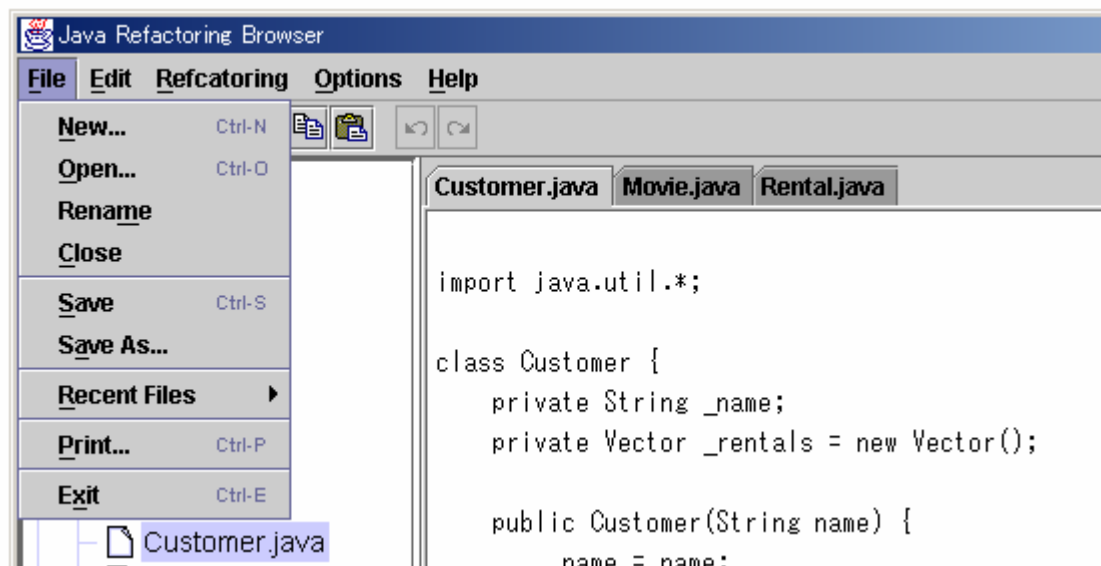


図 4 File メニュー

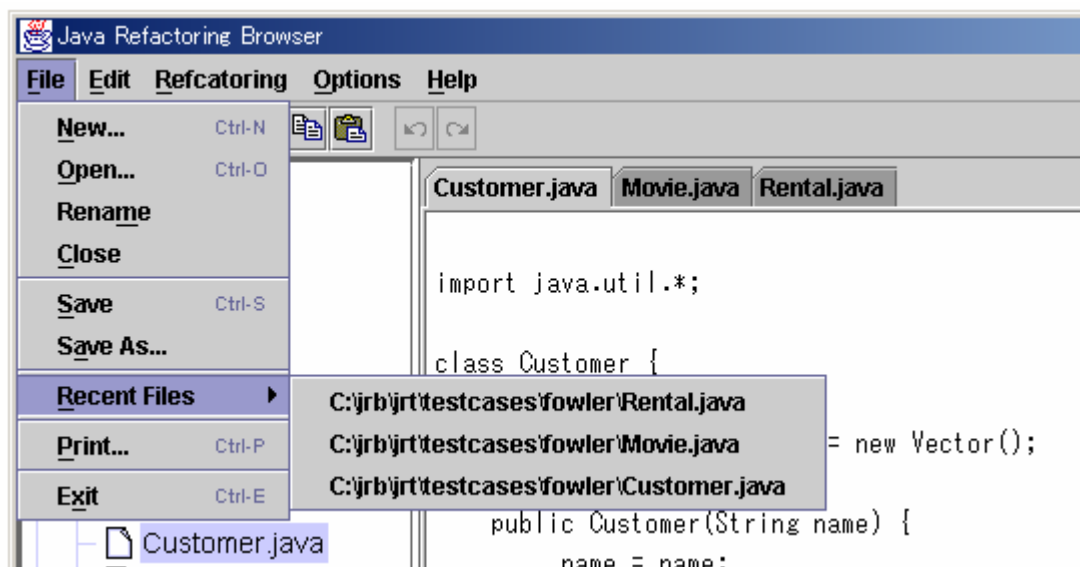


図 5 Recent Files メニュー

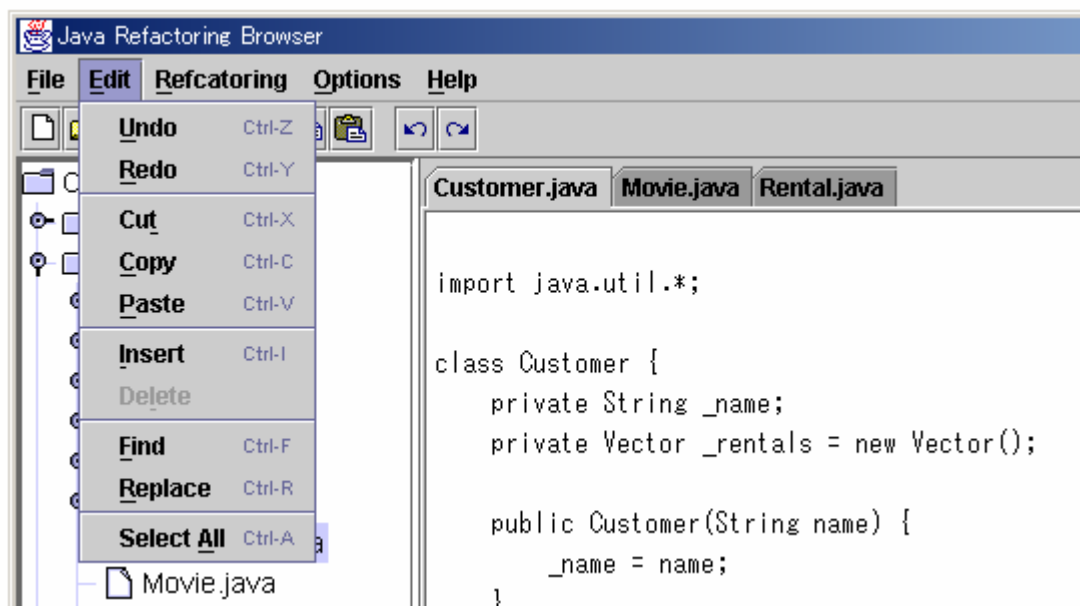


図 6 Edit メニュー

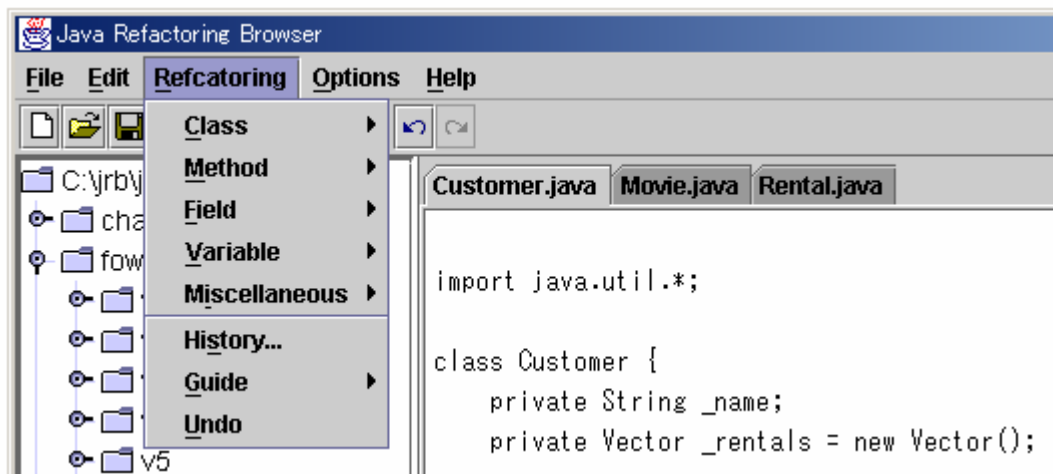


図 7 Refactoring メニュー

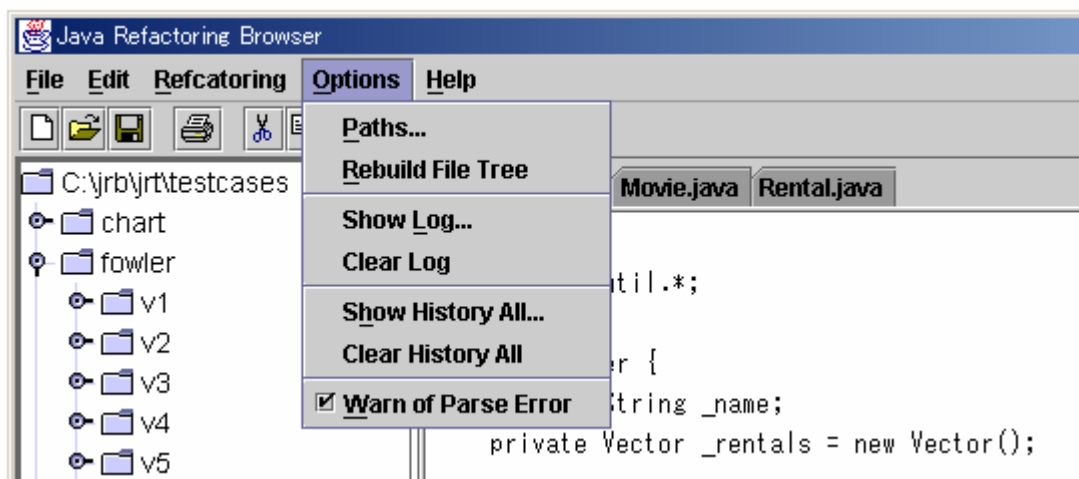


図 8 Options メニュー

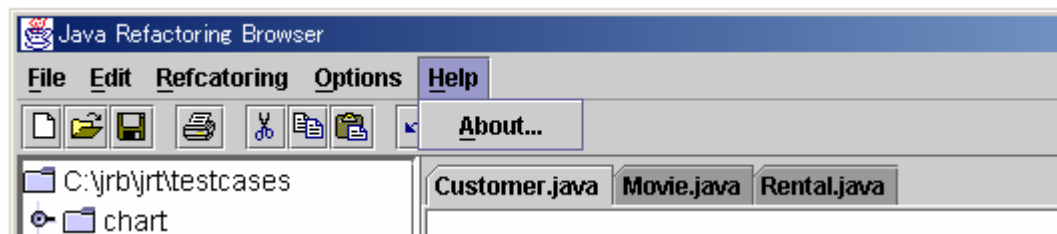


図 9 Help メニュー

(3) ダイアログ / ウィンドウ

JRB のダイアログとウィンドウを図 1 0 ~ 2 2 に示す。ダイアログとは、その画面での操作を終了しないと制御が基本画面に戻らないもの、ウィンドウとは表示したままで別の操作が可能なものを指す。各ダイアログおよびウィンドウに対する操作については、「操作説明書 3 , 4 章」を参考のこと。

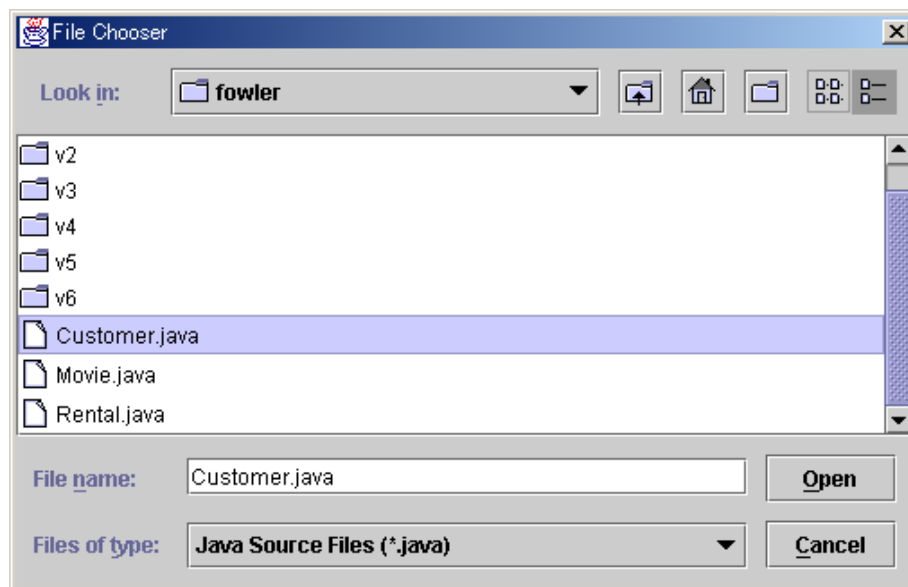


図 1 0 ファイル選択ダイアログ (File メニューから起動)

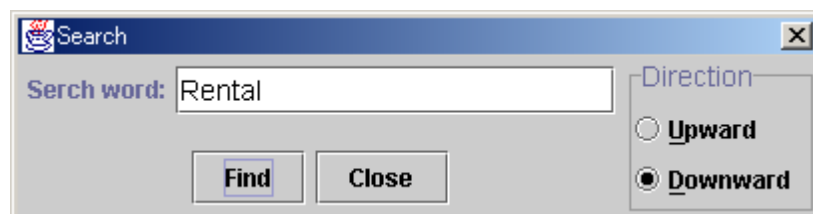


図 1 1 文字列検索ダイアログ (Edit メニューから起動)

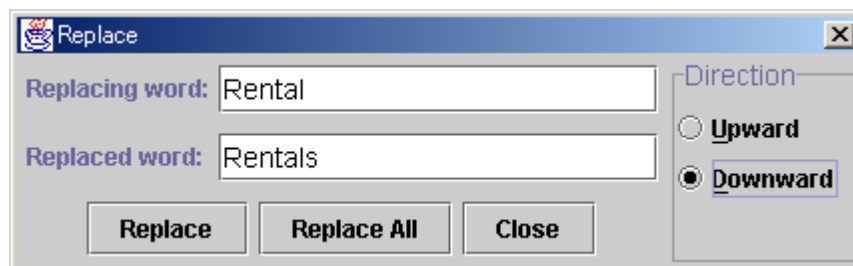


図 1 2 文字列置換ダイアログ (Edit メニューから起動)

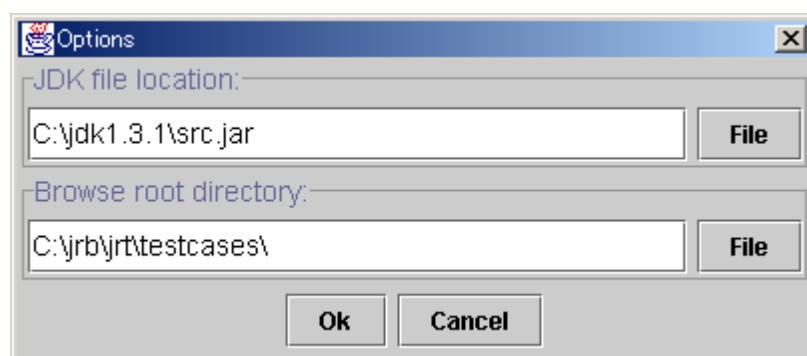


図 1 3 パス設定ダイアログ(Options メニューから起動)

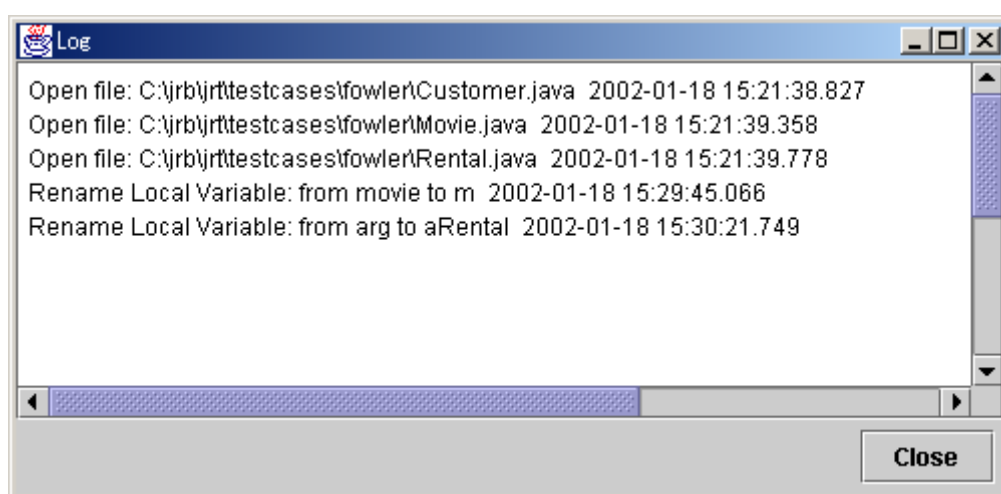


図 1 4 ログ表示ダイアログ(Options メニューから起動)

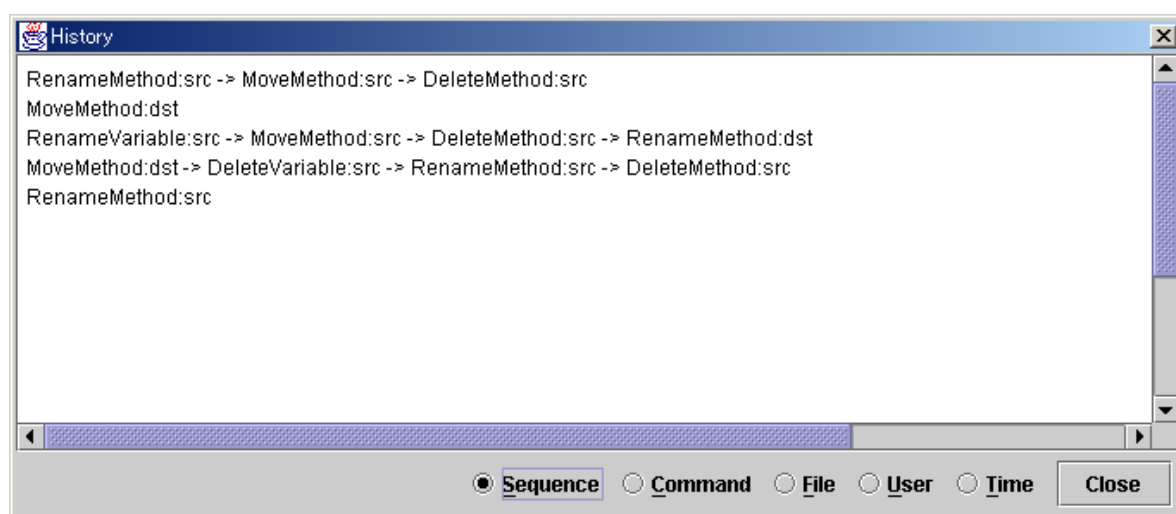


図 1 5 リファクタリング履歴表示ウィンドウ(Sequence を選択) (Option メニューから起動)

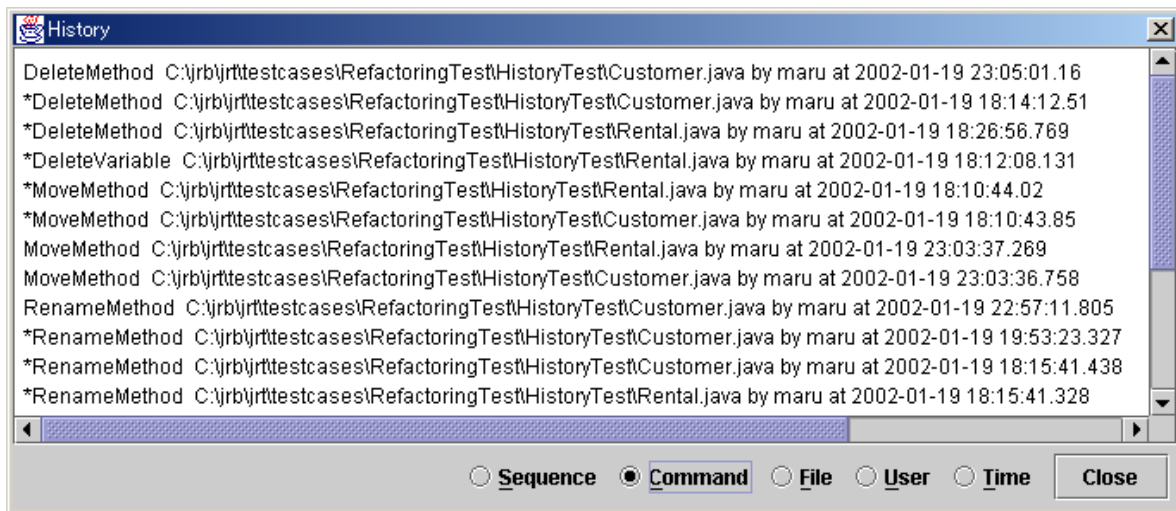


図 1 6 リファクタリング履歴表示ウィンドウ(Command を選択) (Option メニューから起動)

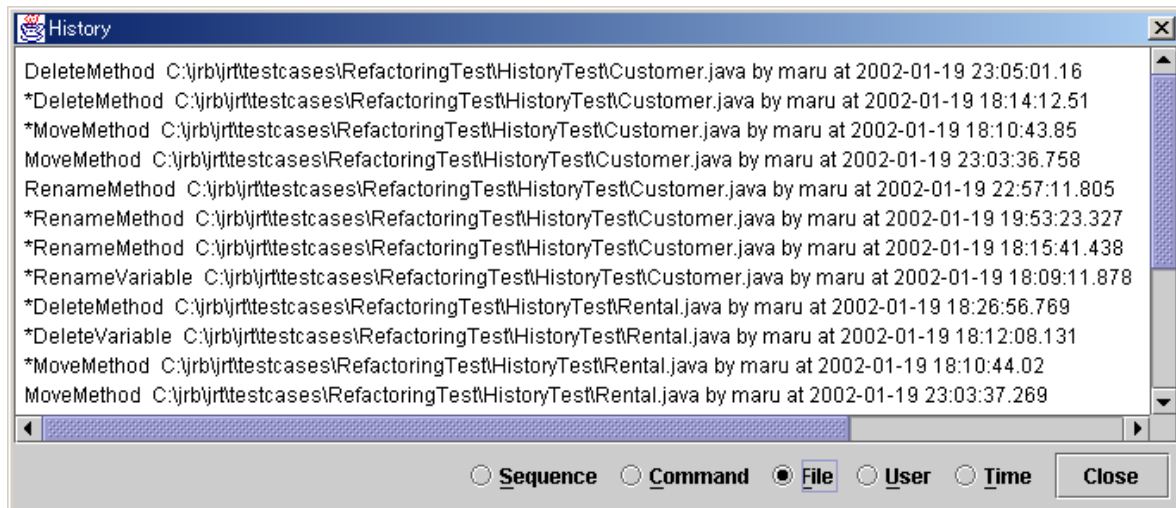


図 1 7 リファクタリング履歴表示ウィンドウ(File を選択) (Option メニューから起動)

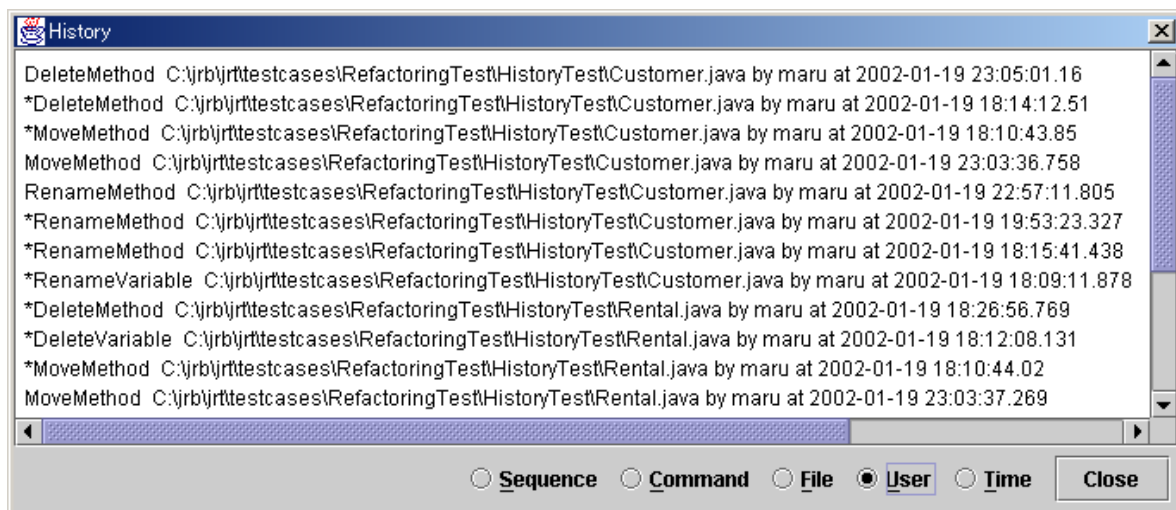


図 1 8 リファクタリング履歴表示ウィンドウ(User を選択) (Option メニューから起動)

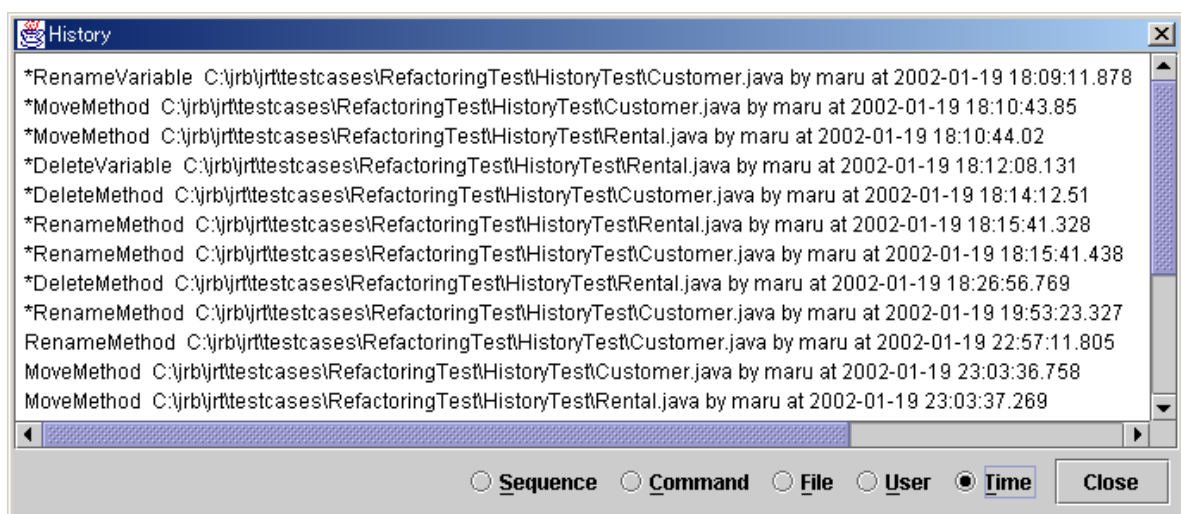


図 1 9 リファクタリング履歴表示ウィンドウ(Time を選択) (Options メニューから起動)

Options メニューにおいて、「Warn of Parse Error」項目にチェックをいれておくと、構文解析時にエラーが生じた場合、エラーとなった字句とエラー箇所（行と列）を示すダイアログが表示される。図 2 0 にエラーダイアログを示す。

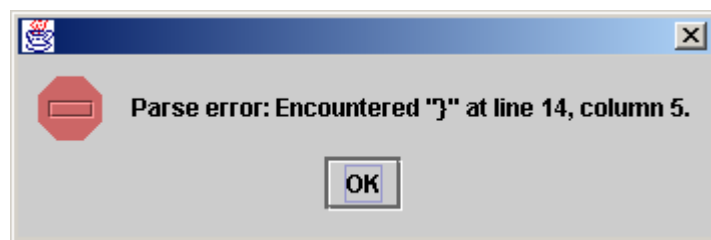


図 2 0 解析エラーダイアログ

JRB では、解析に時間を要する場合、図 2 1 に示すようなプログレスバーが表示される。Cancel ボタンを押すと解析を中止することができるが、リファクタリングの結果は保証されなくなる。

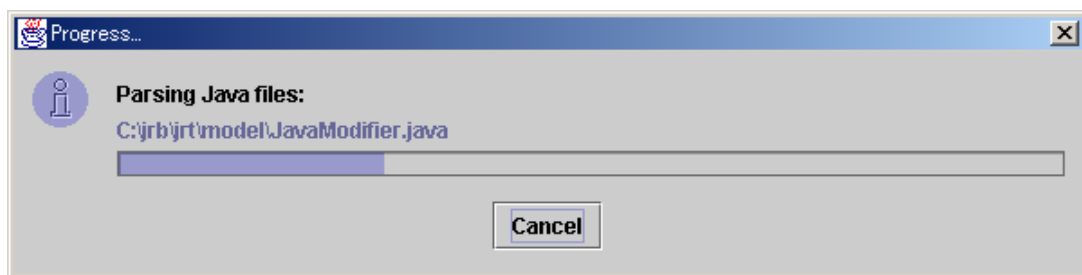


図 2 1 プログラム解析時のプログレスバーダイアログ

JRB に関する情報を表示するダイアログを図 2 2 に示す。



図 2 2 JRB に関する情報ダイアログ(Help メニューから起動)

(4) ポップアップメニューとツールボタン

JRB では操作性の向上のために、一部の編集メニューとリファクタリングメニューが、ソースコード編集画面において、マウスの右ボタンをクリックすることによってポップアップする。ポップアップ時の様子を図 2 3 に示す。

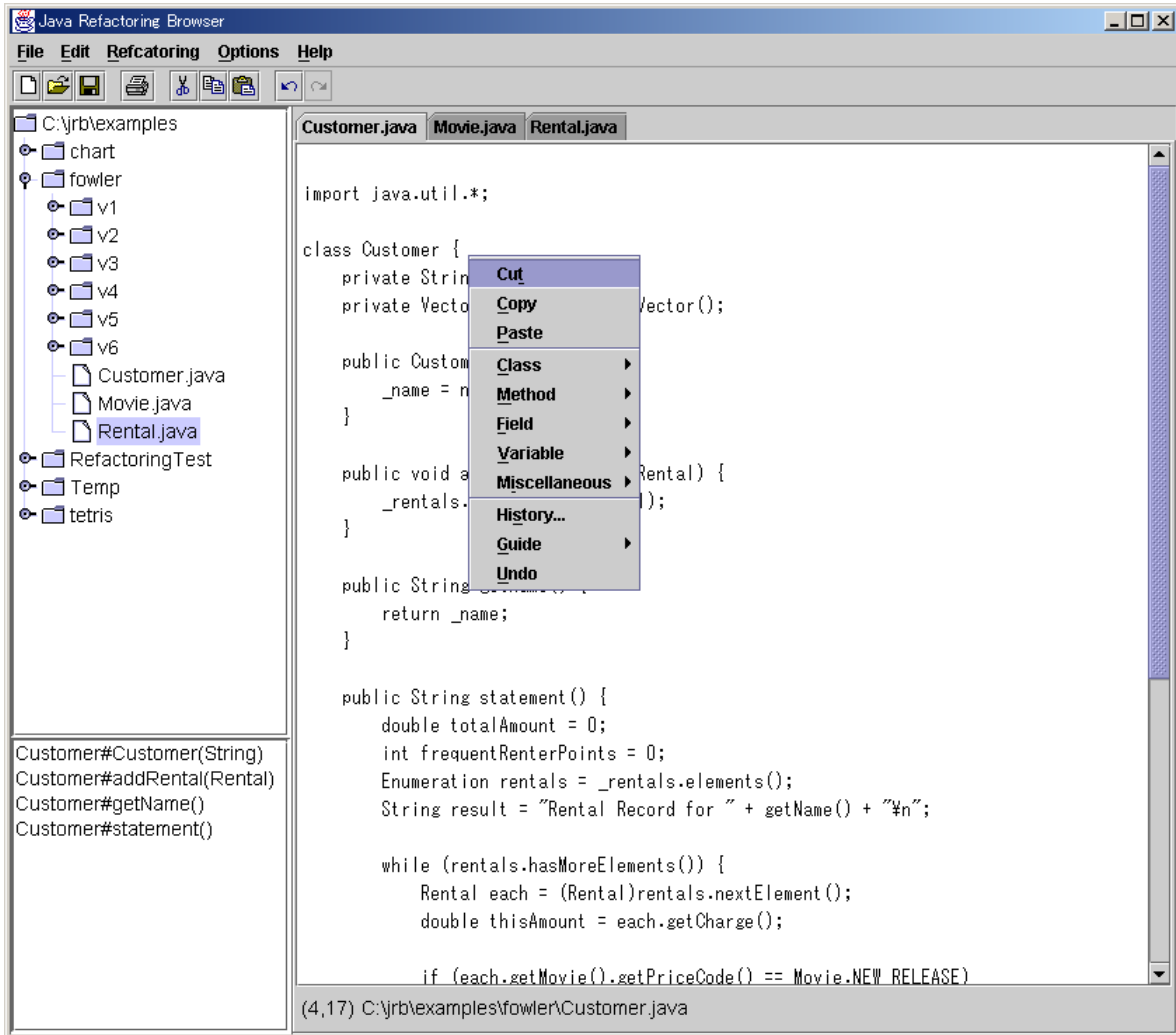


図 2 3 ポップアップメニュー

JRB では、利用者がリファクタリングを実行しようとした際、すなわち、Refactoring メニューを表示させた場合、対象ソースコードの解析を始める（ソースコードに変更があったときのみ）。この解析時にエラーが生じた（たとえば、構文エラーなど）が生じた際には、ポップアップメニューのリファクタリング項目が選択不可能状態となる（Options メニューで、Warn of Parse Error をチェックしておく、この時点でエラーダイアログがでる）。図 2 4 にエラー時のポップアップメニューを示す。

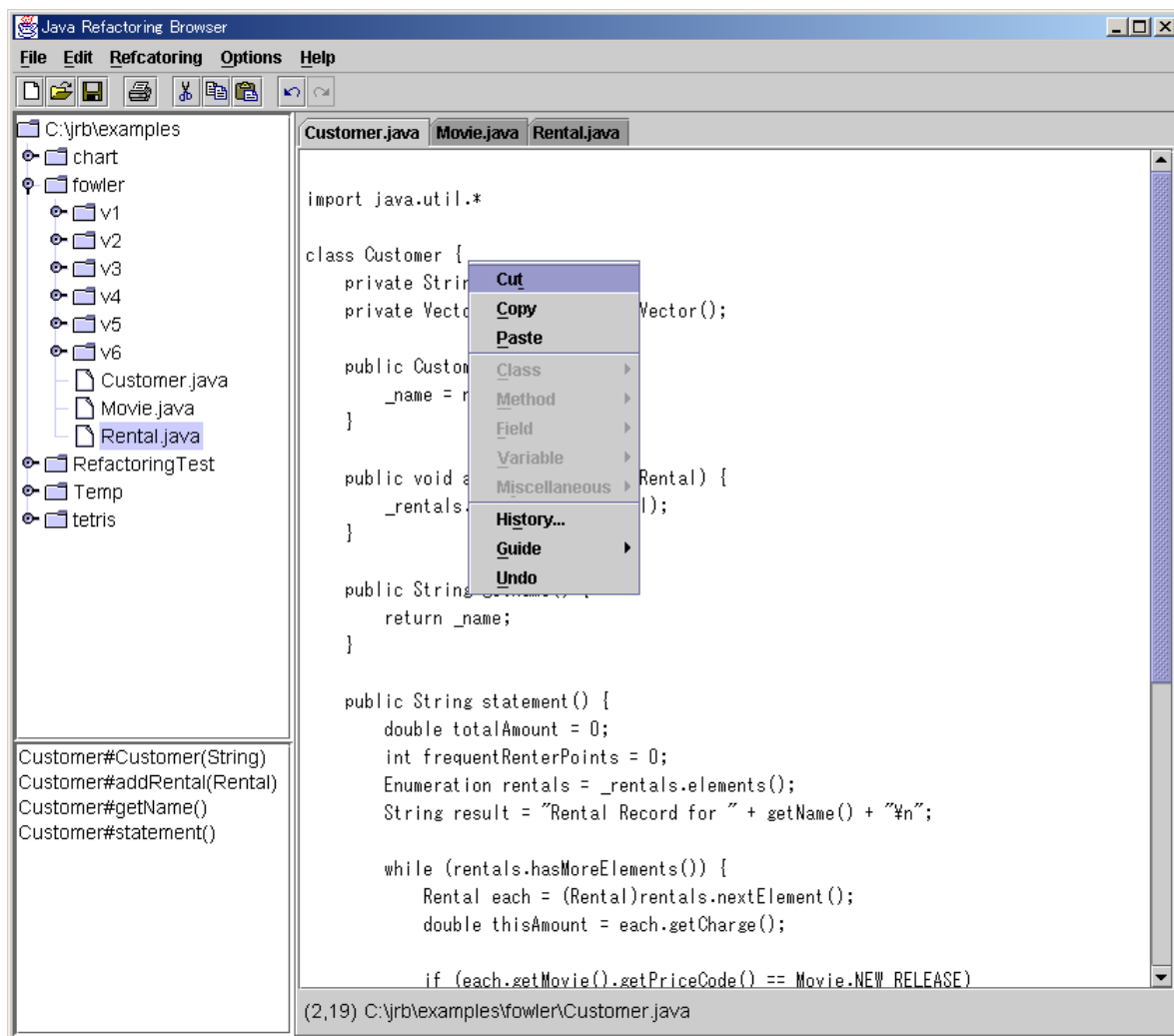


図 2 4 ポップアップメニュー(エラー時)

また，さらなる操作性の向上のために，主ないくつかの項目に関して，メニューの直下にボタンを用意している．ツールボタンを図 2 5 に示す．ボタンを押した際の動作は，メニューで選択した際と同じである．各ボタンとメニュー項目の対応については，「操作説明書 3.3」を参考のこと．

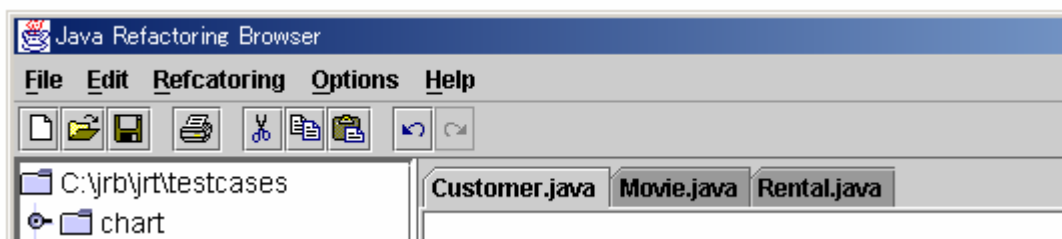


図 2 5 ツールボタン

(5) リファクタリング操作に関する画面出力

JRB においてリファクタリングを実行する際のメニュー画面を図 2 6 ~ 3 0 に示す。リファクタリング操作の実行手順については、「操作説明書 4 章」に詳しい記述があるのでそちらを参考のこと。

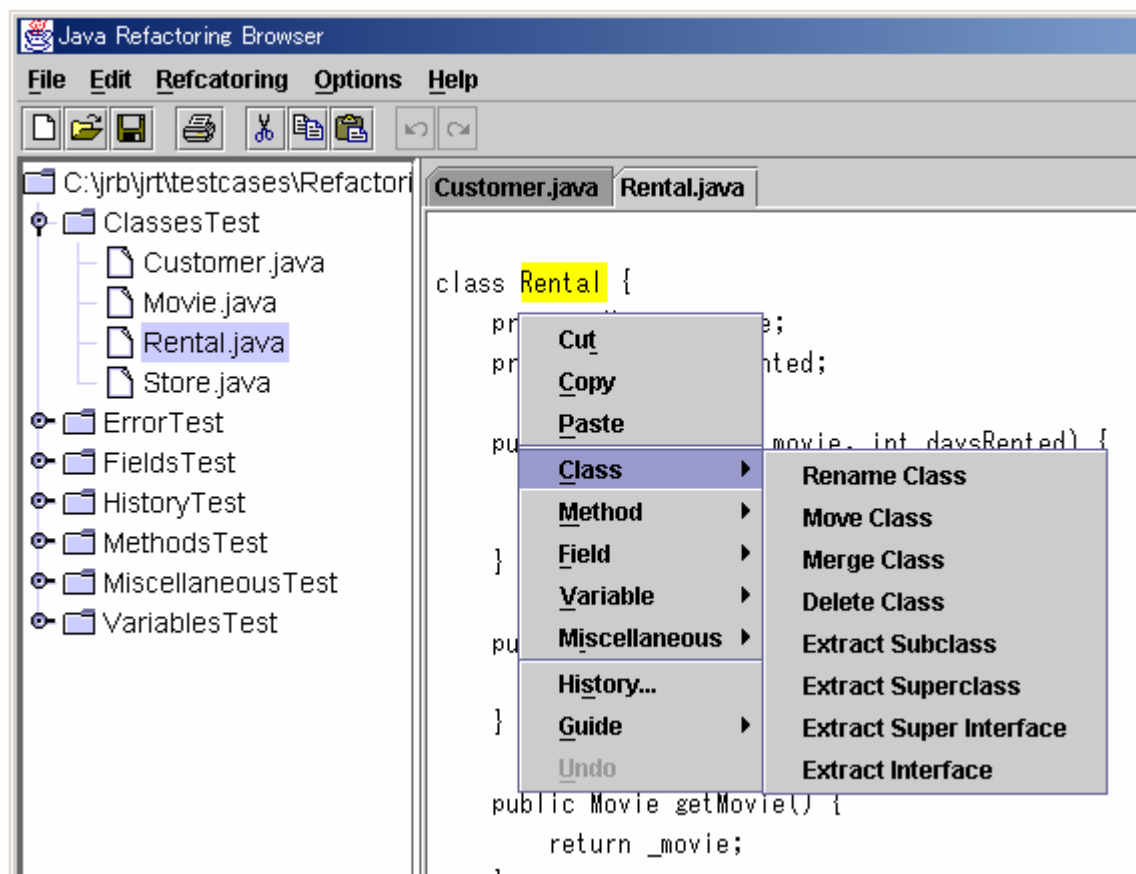


図 2 6 クラス(Class)リファクタリングメニュー

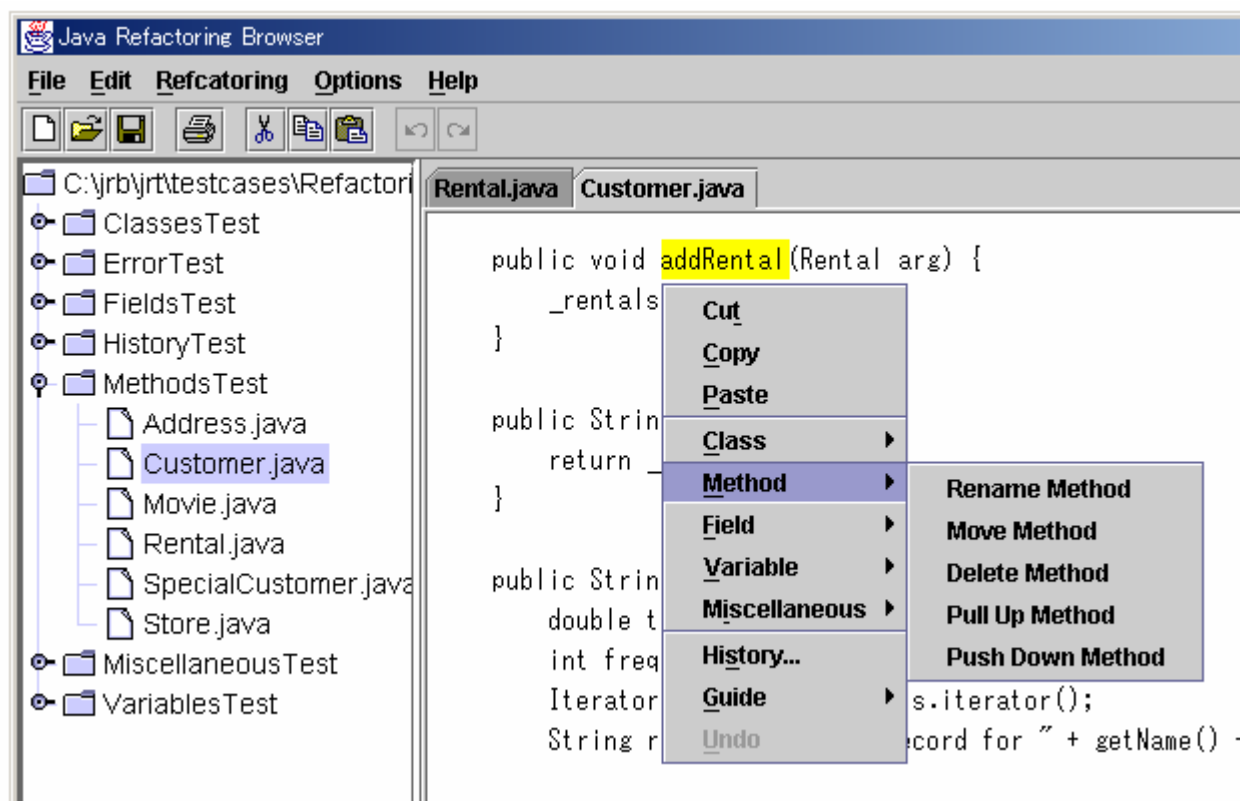


図 2 7 メソッド(Method)リファクタリングメニュー

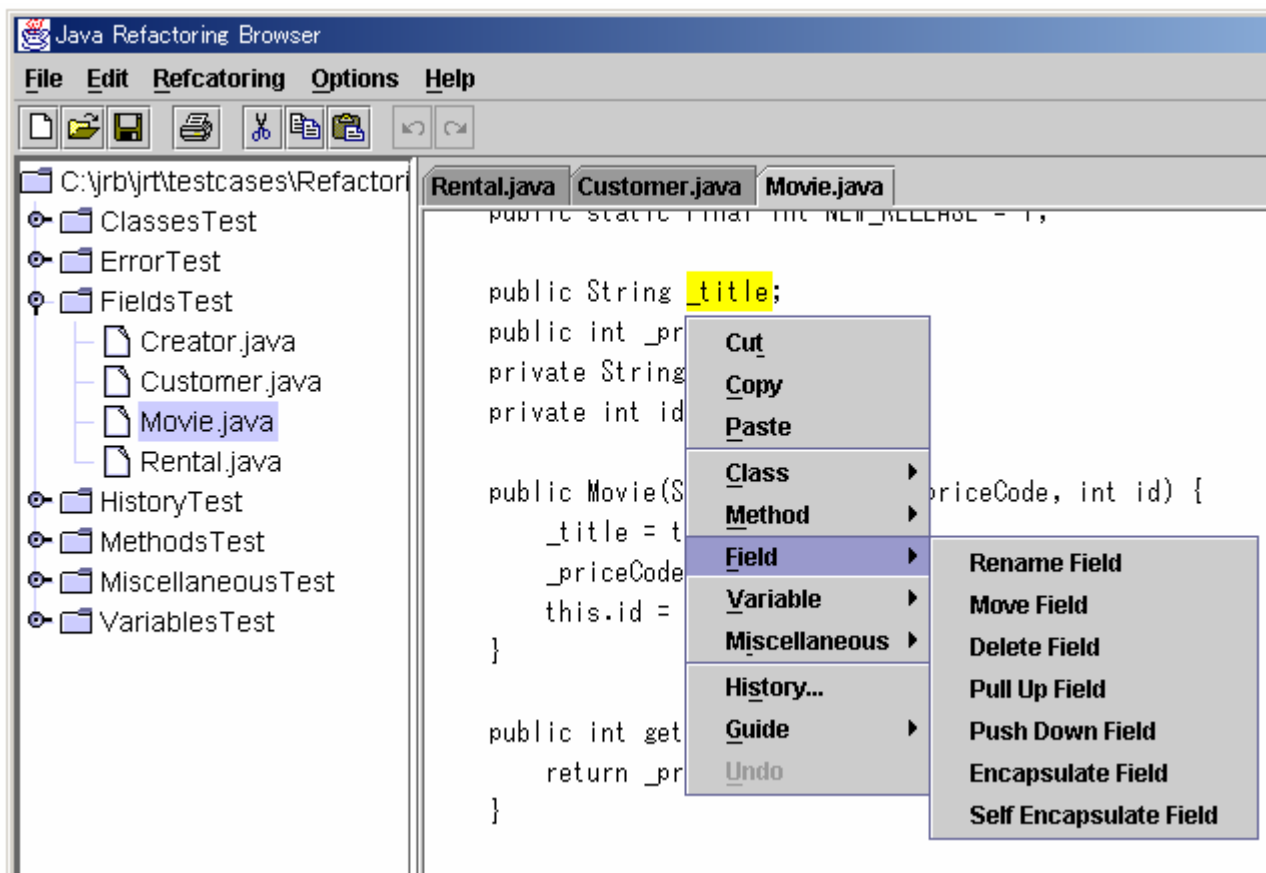


図 2 8 フィールド(Field)リファクタリングメニュー

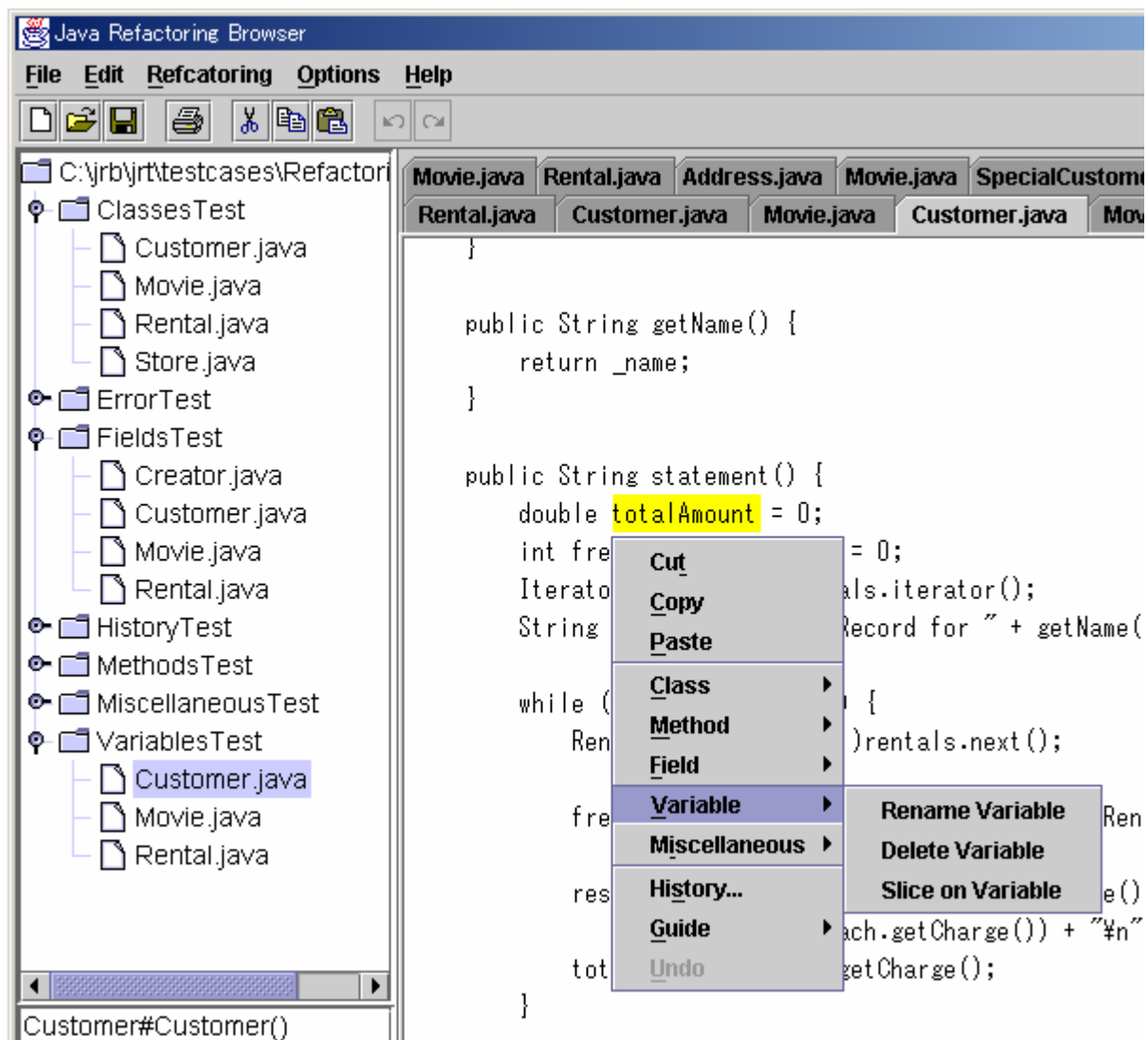


図 2 9 ローカル変数(Variable)リファクタリングメニュー

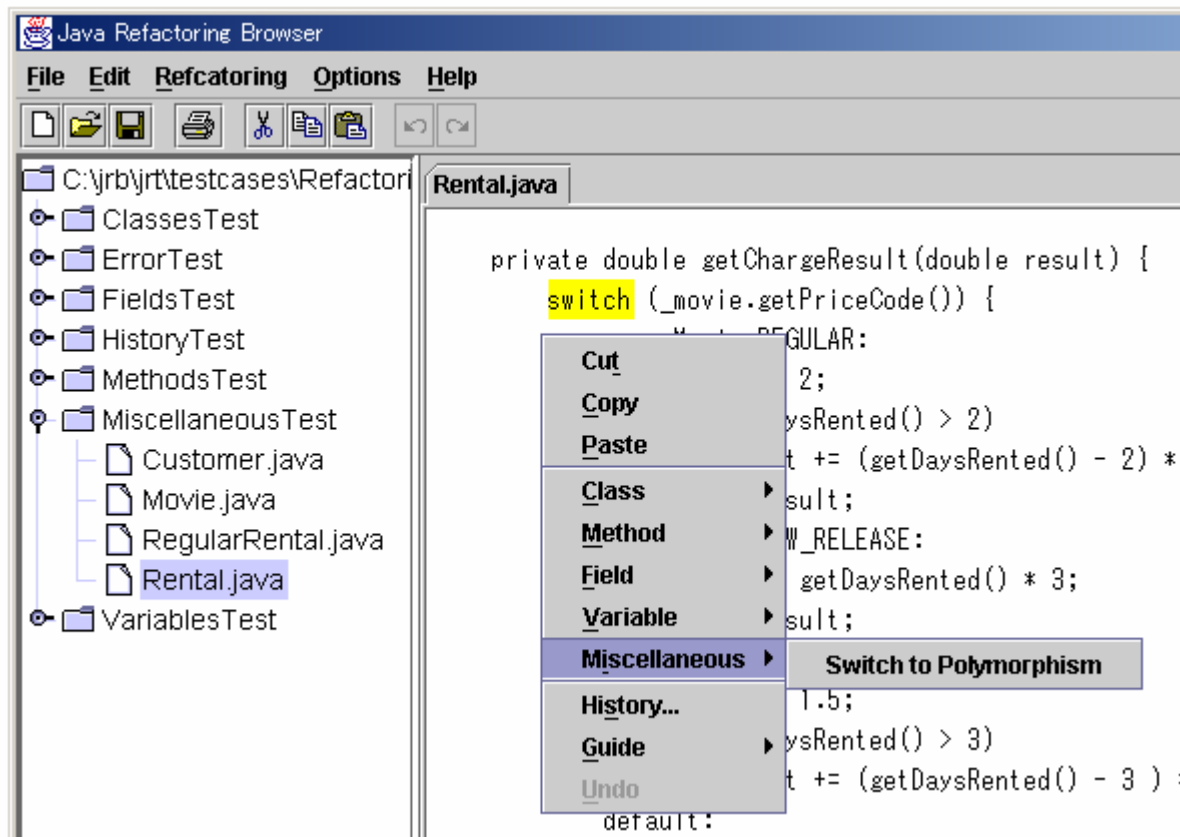


図 3 0 その他(Miscellaneous)のリファクタリングメニュー

(6) リファクタリング操作に関するダイアログ

JRB においてリファクタリングを実行する際に出力されるダイアログを，簡単な説明とともに図 3 1 ~ 4 6 に示す．リファクタリング操作実行時の各画面に対する操作については，「操作説明書 4 章」を参考のこと．

- (a) 名前の変更操作実行時に，変更後のクラス名，メソッド名，フィールド名，ローカル変数の入力を促すダイアログを図 3 1 に示す．前の名前(Old Name)は JRB が表示する．クラス，メソッド，フィールド，ローカル変数の名前は，空白を含まない文字列である．

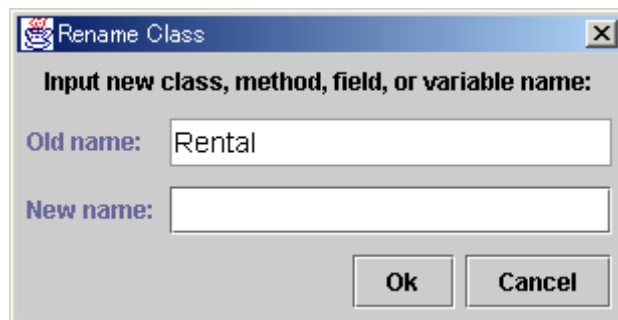


図 3 1 変更後の名前入力ダイアログ

- (b) クラス移動において，移動先ソースコード(ファイル)を指定するダイアログを図 3 2 に示す．File ボタンをクリックすることで，Options で指定した Root.Dir ディレクトリ以下に存在するファイルを一覧から選択することができる．ファイル選択ダイアログを図 3 3 に示す．このファイル選択ダイアログでは，Root.Dir ディレクトリより上位のファイルは選択できないようになっている．

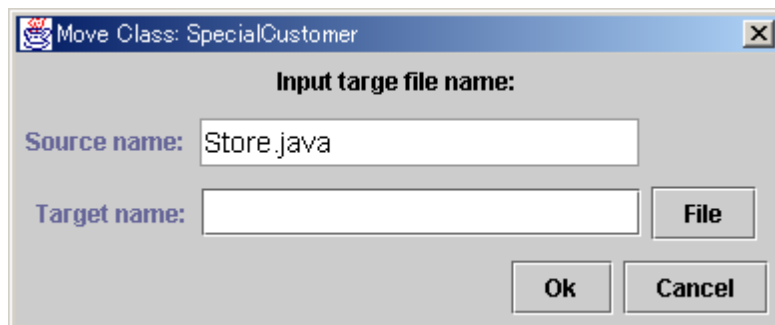


図 3 2 移動先ソースコード指定ダイアログ

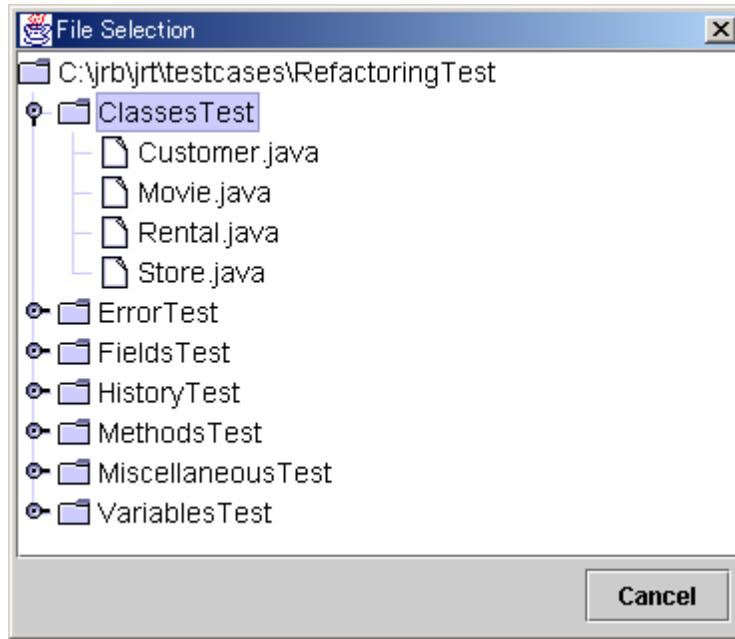


図 3 3 ファイル選択ダイアログ

- (c) クラスを合併する際に合併先のクラスを指定するダイアログを図 3 4 に示す．Class ボタンをクリックすることで，Options で指定した Root.Dir ディレクトリ以下に存在するクラスの一覧から選択することができる．クラス選択ダイアログを図 3 5 に示す．

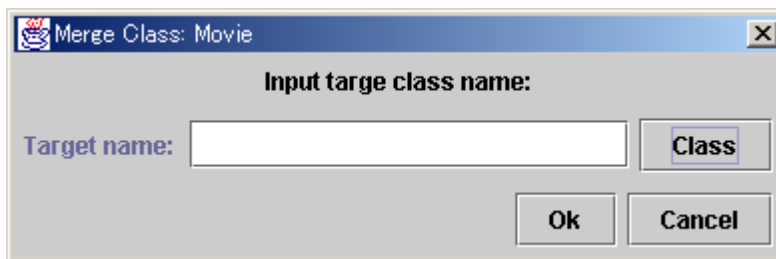


図 3 4 合併先クラス指定ダイアログ

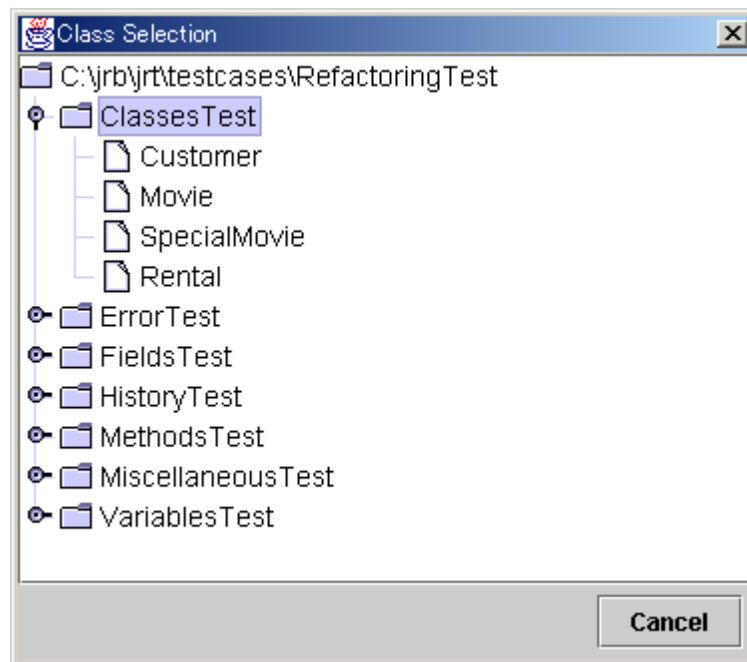


図 3 5 クラス選択ダイアログ

- (d) クラスを抽出する際に新規に抽出したクラスの名前を入力するダイアログを図 3 6 に示す .

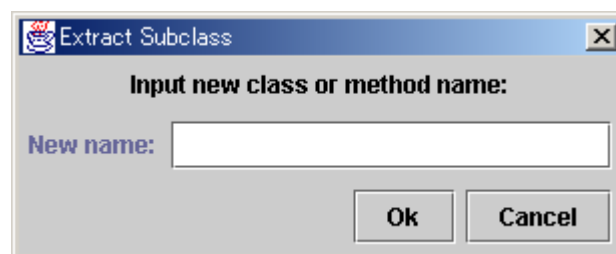


図 3 6 新規クラス名入力ダイアログ

- (e) メソッドおよびフィールドを移動する際に移動先のクラスを指定するダイアログを図 3 7 に示す . メソッドの移動とフィールドの移動では , タイトルバーのメッセージが異なるだけである . このダイアログでも Class ボタンをクリックすることで , 図 3 5 と同様にクラスを一覧から選択することができる .

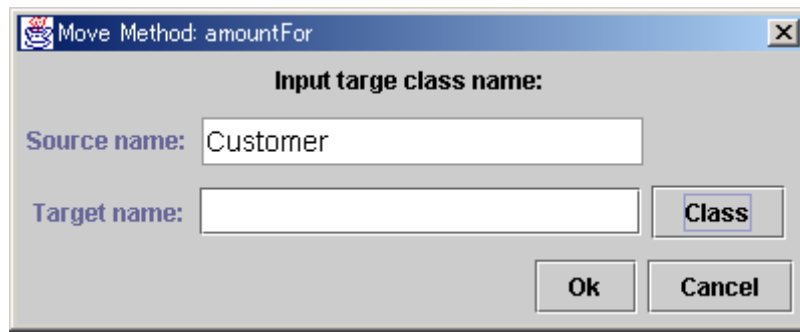


図 3 7 移動先クラス指定ダイアログ

- (f) メソッドおよびフィールドを引き上げる際に、そのスーパークラスを表示し、引き上げ操作の実行を確認するダイアログを図 3 8 に示す。メソッドの引き上げとフィールドの引き上げでは、出力メッセージが異なるだけである。

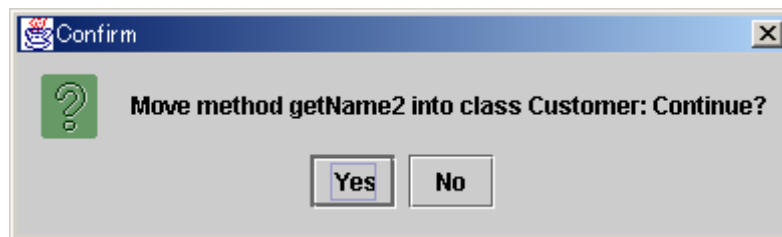


図 3 8 引き上げ先クラスの確認ダイアログ

- (g) メソッドおよびフィールドを引き下げる際に、そのサブクラスを表示し、引き下げ操作の実行を確認するダイアログを図 3 9 に示す。メソッドの引き下げとフィールドの引き下げでは、出力メッセージが異なるだけである。

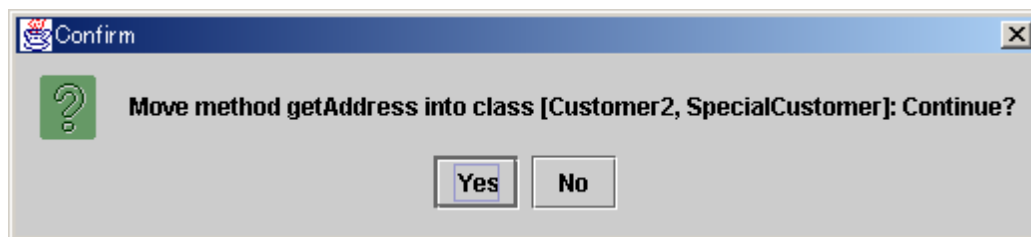


図 3 9 引き下げ先クラスの確認ダイアログ

- (h) フィールドの移動、カプセル化、自己カプセル化において、アクセッサを指定するダイアログを図 4 0 に示す。アクセッサはデフォルトで、フィールド名の先頭一文字を大文字に変え、Getter name は文字列 set、Setter name は文字列 get を、フィールド名の

前に付加したものとなる。ただし、フィールド名がアンダーライン(_)で開始されている場合は、それを取り除いてから、上記の処理を行う。



図 4 0 アクセッサの指定ダイアログ

- (i) メソッドの移動およびフィールドの移動の際には、移動先クラスの参照オブジェクト名を決定する必要がある。JRB では、メソッド移動に関してのみ、そのメソッド内で唯一移動先オブジェクトが現れるかどうかを検査し、この条件を満たす場合のみ参照オブジェクトを自動的に決定する。自動決定に失敗した場合は、利用者にそのオブジェクト名をたずねる。その際に表示される参照オブジェクトの入力ダイアログを図 4 1 に示す

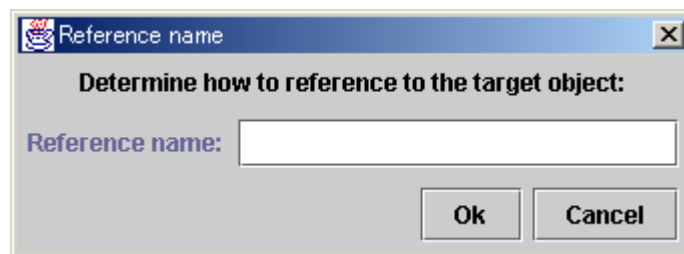


図 4 1 参照オブジェクト名入力ダイアログ

- (j) 条件分岐をポリモーフィズムに置き換えるリファクタリング操作実行の際に、条件とその出力先サブクラスの対応をとる必要がある。この対応を指定するダイアログの様子を示す。実際に指定する際の手順は、「操作説明書 4.5」に詳しい記述があるのでそちらを参考のこと。ここでは、指定開始時の画面を図 4 2、指定途中の画面を図 4 3、指定完了時の画面を図 4 4 に示す。

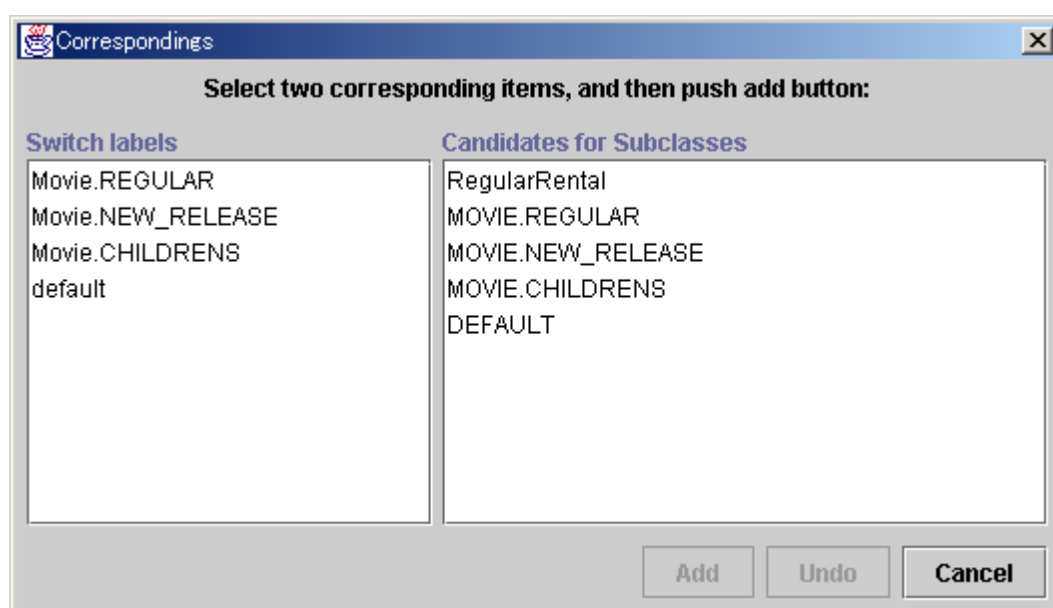


図 4 2 分岐条件とサブクラスとの対応指定ダイアログ（開始時）

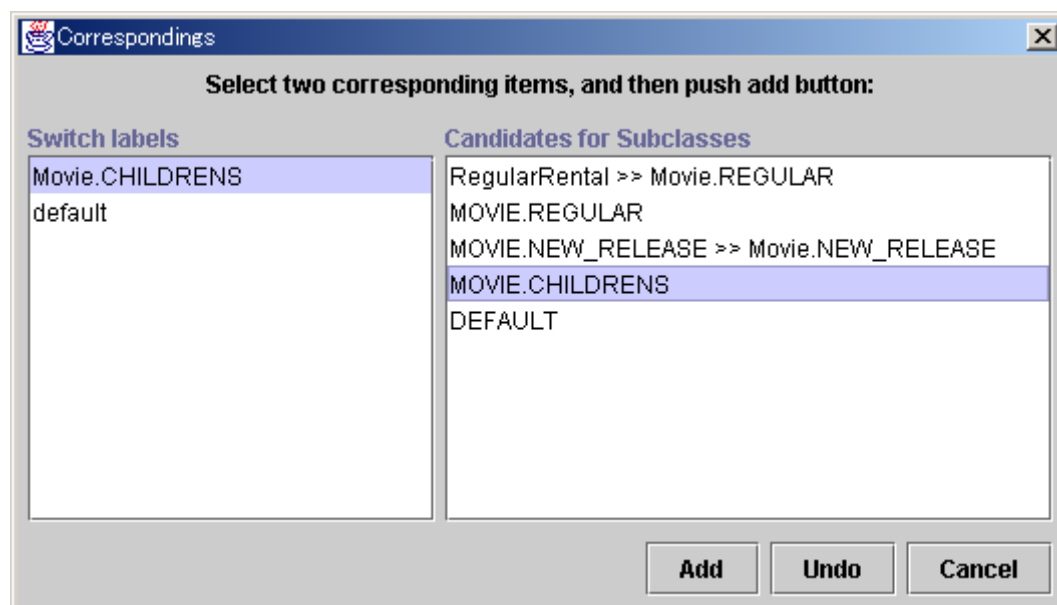


図 4 3 分岐条件とサブクラスとの対応指定ダイアログ（指定途中）

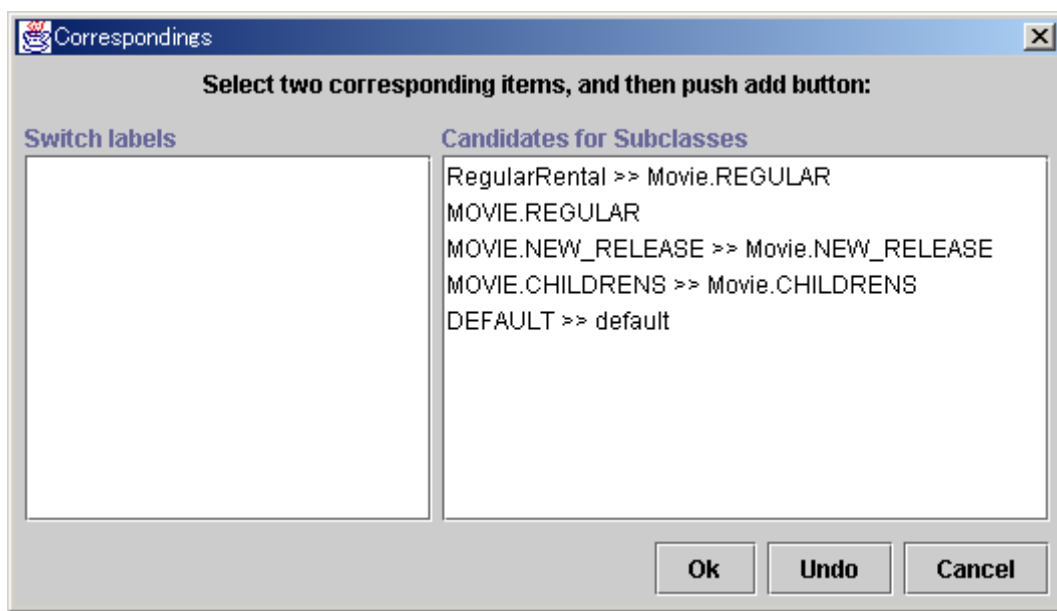


図 4 4 分岐条件とサブクラスとの対応指定ダイアログ（指定完了時）

- (k) JRB では，リファクタリングの自動化に関する可能性を示す目的で，依存関係を用いたメソッドの抽出リファクタリングを試験的に導入した．残念ながら，現時点では，このリファクタリング操作が安全である保証はない．そこで，JRB ではスライスによるメソッド抽出リファクタリングが試験的であることを通知する．その際のダイアログを図 4 5 に示す．



図 4 5 試験的リファクタリングであることを通知するダイアログ

- (l) JRB では，利用者の実行したリファクタリング操作により影響を受けるファイルを Options で指定したディレクトリ以下の全ファイルに対して検査する．影響を受けるソースコードが発見された場合，図 4 6 に示すダイアログにより利用者に通知する．

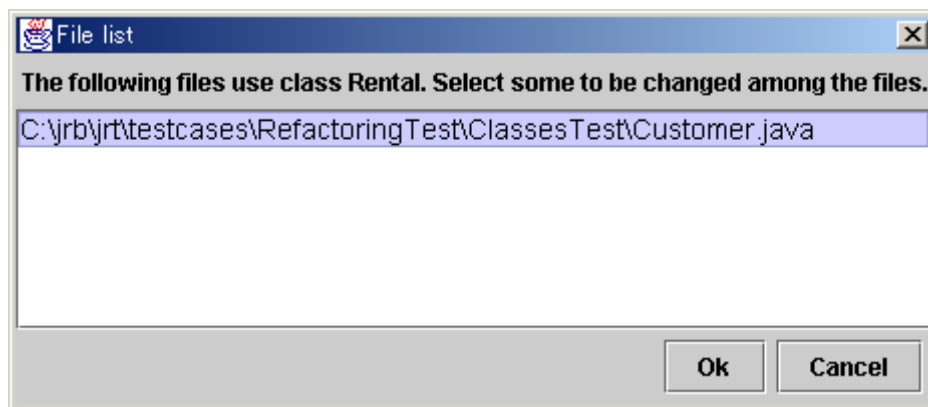


図 4 6 変換の影響を受けるソースコードの一覧

このダイアログに表示された一覧に対して、利用者が指定したファイルと同時に変換を適用したいソースコード（ファイル）を選択することができる。

(7) リファクタリング結果の画面出力

JRB では、リファクタリングにより変換されたソースコードを利用者に確認する。その際、変更箇所をハイライトで明示する。リファクタリング結果の確認ダイアログの画面出力を図 4 7 に示す。図 4 7 は、クラス名の変更後の確認ダイアログである。JRB において、すべての変換結果はこのダイアログで出力される。

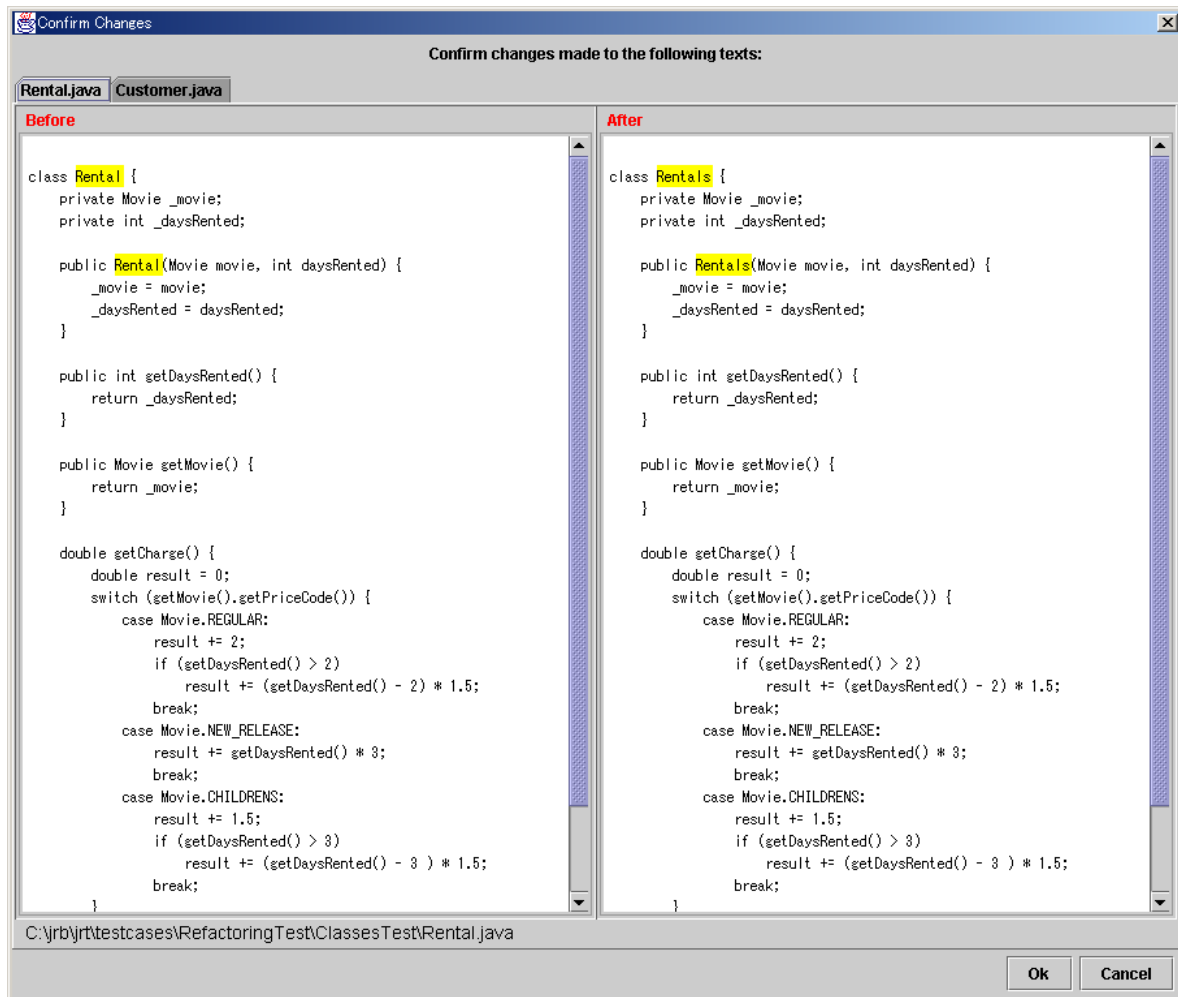


図 4 7 リファクタリング結果確認ダイアログ

左側が変換前のソースコード、右側が変換後のソースコードである。変換ソースコード(ファイル)が複数存在する場合は、上部のタブをクリックすることで、指定ソースコードを前面に表示することができる。このダイアログにおいて、変換を受理する場合は Ok ボタンを押す。受理されたソースコードは、基本画面のソースコード編集画面に移動する。反対に、この時点で Cancel ボタンを押すと、変換はすべて破棄され、履歴にも残らない。

(8) Undo 操作と履歴の検索 , 表示

JRB では , ファイルごとの取り消し操作の他に , リファクタリング操作の一括取り消しを提供する . Undo 実行時の制約については , 「操作説明書 4.8」を参考のこと . 取り消しを実行する際の確認ダイアログを図 4 8 に示す .

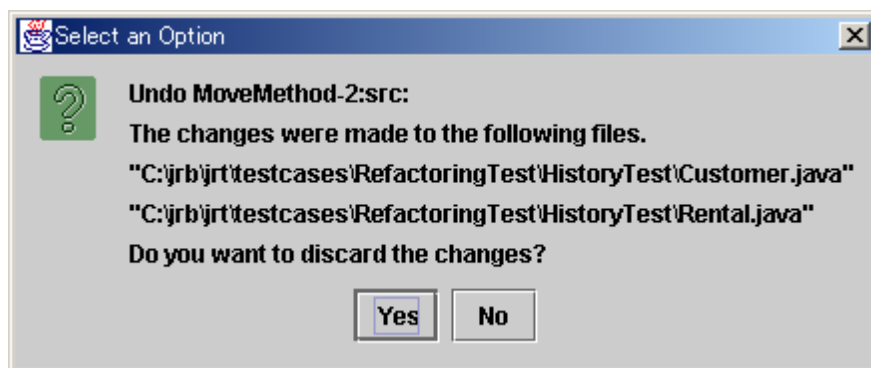


図 4 8 取り消し操作の実行確認ダイアログ

さらに , JRB ではファイルごとのリファクタリング操作の履歴を表示する機能と , 過去の操作から直前に行った操作を検索することで , 次の操作を提案する機能を有する . 履歴表示ダイアログを図 4 9 に示す . また , 検索した結果として提案する操作の出力例を図 5 0 に示す .

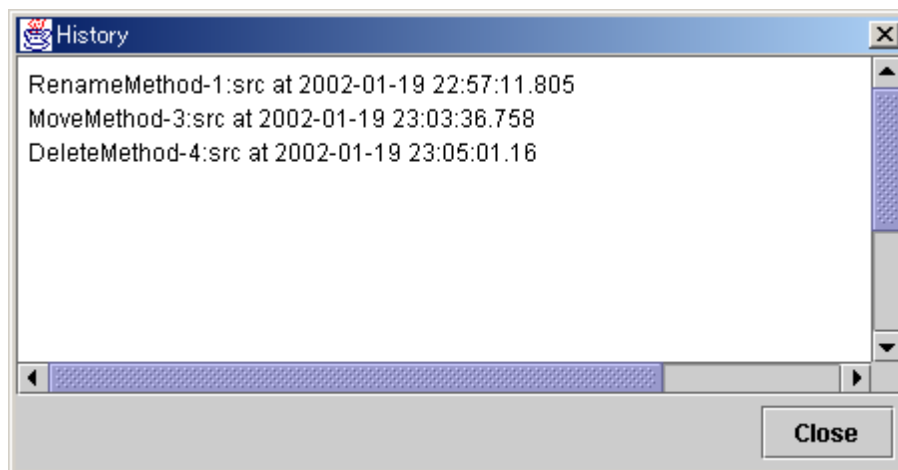


図 4 9 履歴表示ダイアログ

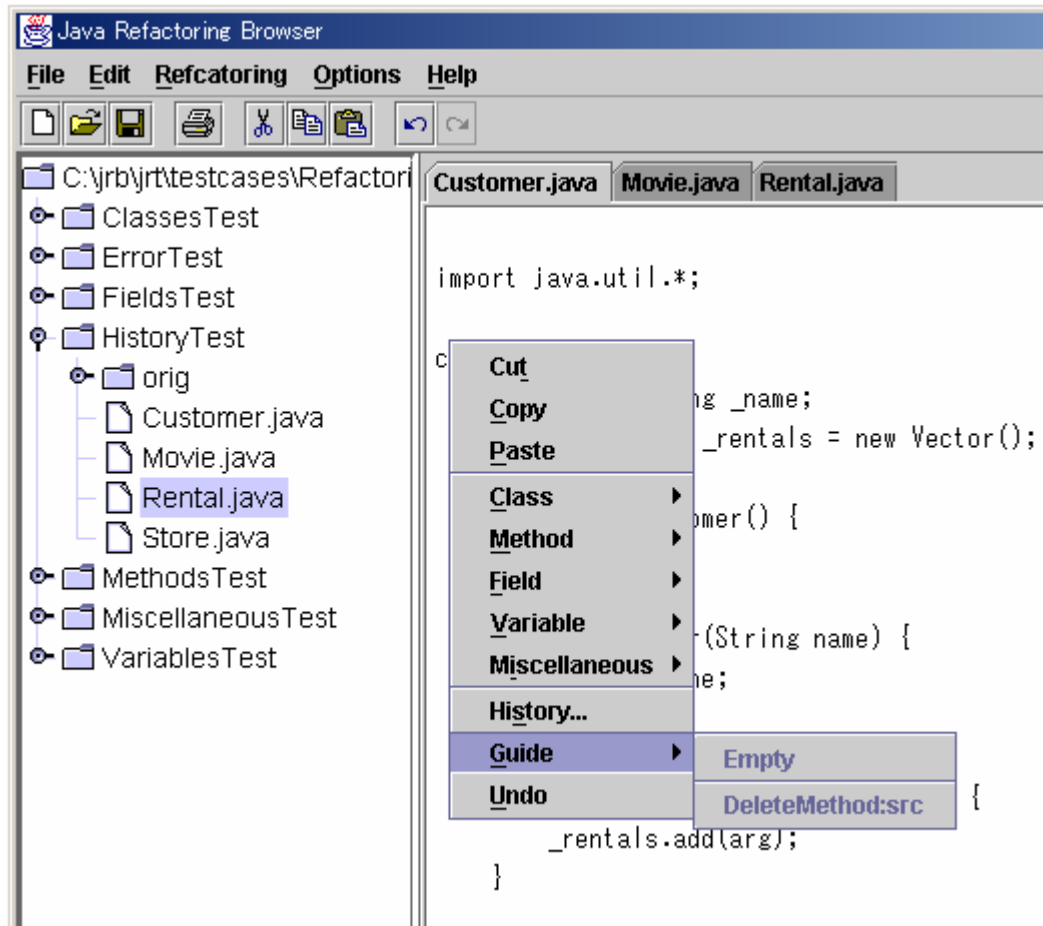


図 5 0 検索結果の出力例

Guide メニューの右側に表示されている操作名は、JRB が提案する次リファクタリング操作である。2 つ前までの操作が一致した場合、メニューの上段（図 5 0 では Empty）に、1 つ前の操作が一致した場合は、下段に表示される。一致する操作が複数存在する場合は、それらの操作の一致回数を検査し、上段および下段のそれぞれに対して回数の多いものから 5 つずつ表示される（頻度の多いものが上位になる）。

4.2 帳票仕様

JRB では、リファクタリング履歴を Java オブジェクトとしてファイルに蓄積する。クラス RefactoringRecord が蓄積されるオブジェクトのクラスである。このクラスのフィールドを以下に示す。

```
public class RefactoringRecord implements Serializable {
    private String command;        // 操作の名前
    private String fileName;       // 変換ファイル
    private Timestamp timestamp;   // 実行時刻
    private String userName;       // 利用者
}
```

4.3 エラーメッセージ

JRB においてエラーメッセージは基本的にダイアログで表示される。エラーの出力例を図 5 1 に示す。

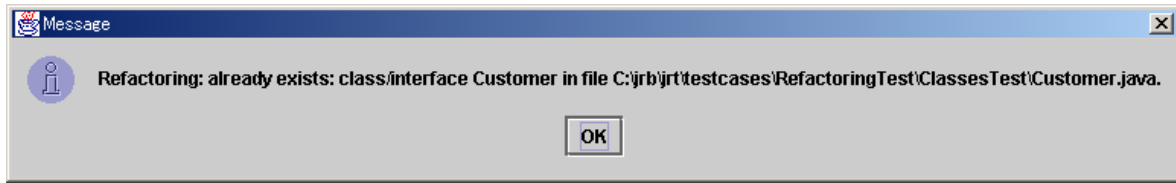


図 5 1 エラーダイアログの例

JRB で表示されるエラーメッセージの一覧を以下に示す。すべて大文字の文字列は、実際にエラーの対象となる文字列に置き換えられて表示される。たとえば、上記のエラーメッセージは、2.に相当する。エラーメッセージの前の番号は、単に通し番号である。

(1) リファクタリング時のエラー

リファクタリング操作の実行において、各操作の前提条件を満たさない場合、あるいは、確認が必要な場合に表示されるエラーメッセージを以下に示す。各リファクタリング操作の前提条件については、「論文 5.1」を参照のこと。

1. Refactoring: already exist(s): accessor(s) GETTERNAME and/or SETTERNAME in class CLASSNAME.
2. Refactoring: already exists: class/interface CLASSNAME in file FILENAME.
3. Refactoring: already exists: field FIELDNAME in class CLASSNAME.
4. Refactoring: already exists: method METHODNAME in class CLASSNAME.
5. Refactoring: already exists: variable NAME in method METHODNAME.
6. Refactoring: already used: type TYPE in file FILENAME.
7. Refactoring: cannot create abstract method METHODNAME called by METHODNAME2.
8. Refactoring: class CLASSNAME is used in the package.
9. Refactoring: field FIELDNAME is used in class CLASSNAME.
10. Refactoring: field FIELDNAME is used in the package.
11. Refactoring: field conflict exists: classes CLASSNAME and CLASSNAME2.
12. Refactoring: method METHODNAME is called in class CLASSNAME.
13. Refactoring: method METHONAME is used in the package.
14. Refactoring: method conflict exists: classes CLASSNAME and CLASSNAME2.
15. Refactoring: not Java file: FILENAME.
16. Refactoring: not change: field FIELDNAME is not defined in a simple assignment.
17. Refactoring: not change: field FIELDNAME is not private.

18. Refactoring: not change: field FIELDNAME is private.
19. Refactoring: not change: method METHODNAME contains multiple statements.
20. Refactoring: not extract: class CLASSNAME is final.
21. Refactoring: not found: accessor(s) GETTERNAME and/or SETTERNAME in class CLASSNAME.
22. Refactoring: not found: class CLASSNAME.
23. Refactoring: not found: class CLASSNAME.
24. Refactoring: not move: field FIELDNAME is directly accessed in class CLASSNAME.
25. Refactoring: not move: field FIELDNAME is not private.
26. Refactoring: not move: field FIELDNAME uses other fields of its ancestors.
27. Refactoring: not move: field FIELDNAME uses other fields of its class.
28. Refactoring: not move: field FIELDNAME uses private fields in class CLASSNAME.
29. Refactoring: not move: method METHODNAME calls other methods in class/superclass of CLASSNAME.
30. Refactoring: not move: method METHODNAME calls private methods in class CLASSNAME.
31. Refactoring: not move: method METHODNAME has no/multiple objects referring to class CLASSNAME.
32. Refactoring: not move: method METHODNAME is called by other methods in class CLASSNAME.
33. Refactoring: not move: method METHODNAME uses fields except the object referring to class CLASSNAME.
34. Refactoring: not move: method METHODNAME uses fields in class CLASSNAME.
35. Refactoring: not move: method METHODNAME uses private fields in class CLASSNAME.
36. Refactoring: variable VARIABLENAME is used in method METHODNAME.
37. Invalid string: "GETTERNAME" and/or "SETTERNAME".
38. Invalid string: "NAME".
39. No such file: FILENAME.
40. File FILENAME exists outside directories under DIRNAME.
41. Move field FIELDNAME into class CLSSSNAME: Continue?
42. No class uses field FIELDNAME: Continue to move it into all children CLASSES?
43. Exists the same method declaration in superclasses CLASSES and/or subclasses CLASSES: Continue?
44. Move method METHODNAME into class CLASSNAME: Continue?
45. No subclass calls method METHODNAME: Continue to move it into all children

CLASSES?

- 46. Slice on variable refactoring is on trial, so it generates code as a comment.
- 47. Undo COMMAND:「
The changes were made to the following files..」
FILENAMES.」
Do you want to discard the changes?

(2) 共通エラー

各メニュー(File, Edit, Refactoring, Options)の実行時に共通するエラーメッセージを以下に示す .

- 48. File exists: FILENAME.
- 49. Already opened: FILENAME.
- 50. The changes were made to this file. Do you want to re-open the file?"
- 51. Failed to open: FILENAME.
- 52. Failed to read: FILENAME.
- 53. Failed to write: FILENAME.
- 54. Do you want to save the changes you have made to this file?
- 55. The changes were made to a file on disk. Do you want to save this file?
- 56. Not found: WORD.
- 57. Fail to store property: FILENAME.
- 58. Fail to store history.
- 59. Do you want to discard the history of all refactorings?
- 60. Failed to find: FILENAME.
- 61. Not found: WORD.
- 62. Not found: WORD in this text.
- 63. Fail to load properties. Please re-install JRB.
- 64. Not found the specified application directory: DIRECTORY.

5 . 機能仕様

本ソフトウェアは、オブジェクト指向で開発されている。このため、各機能に対する仕様を説明する前に、これらの機能で使用するデータモデルを示す。主なデータモデルは、以下の4つである。

- a) Java ソースコードモデル
- b) グラフ
- c) 制御フローグラフ
- d) プログラム依存グラフ

各モデルの説明を5.1～5.4で述べる。

5.1 Java ソースコードモデル

Java ソースコードモデルは以下のクラスで構成されている。これらのクラスは、model パッケージに格納されている。これらのクラス構成を図5.2に示す。

- a) JavaComponent
- b) JavaFile
- c) JavaClass
- d) JavaMethod
- e) JavaStatement
- f) JavaVariable

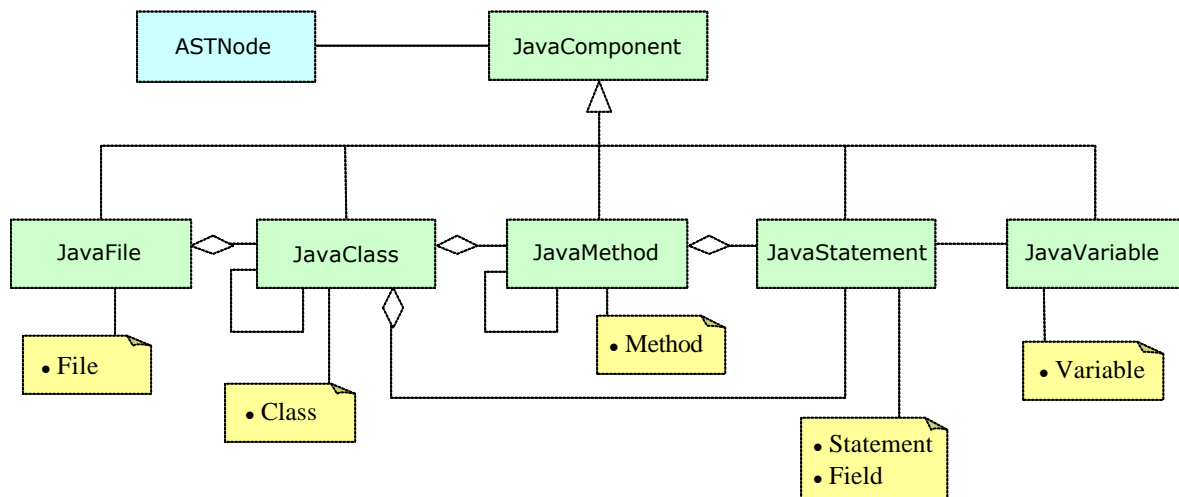


図5.2 Java ソースコードモデルのクラス構成

5.1.1 JavaComponent クラス

(1) 機能説明

Java ソースコードモデルにおける各構成要素の情報を保持する。

(2) 属性

- SimpleNode astNode
AST におけるノードへの参照
- int beginLine, beginColumn, endLine, endColumn
解析ソースコードのテキスト上での位置(順に開始行, 開始列, 終了行, 終了列)
- GraphNode cfgNode
CFG におけるノードへの参照

(3) メソッド

- void setASTNode(SimpleNode node)
AST のノードを node に設定する
- SimpleNode getASTNode()
AST のノードを取得する
- void setCFGNode(GraphNode node)
CFG のノードを node に設定する
- GraphNode getCFGNode()
CFG のノードを取得する
- void setResponsive(int beginLine, int beginColumn, int endLine, int endColumn)
テキスト上での位置を設定する (引数の順に, 開始行, 開始列, 終了行, 終了列)
- int getBeginLine()
テキスト上での開始行を取得する
- int getBeginColumn()
テキスト上での開始列を取得する
- int getEndLine()
テキスト上での終了行を取得する
- int getEndColumn()
テキスト上での終了列を取得する
- boolean isJavaFile()
この構成要素が JavaFile オブジェクトかどうか
- boolean isJavaClass()
この構成要素が JavaClass オブジェクトかどうか
- boolean isJavaMethod()
この構成要素が JavaMethod オブジェクトかどうか
- boolean isJavaStatement()
この構成要素が JavaStatement オブジェクトかどうか
- boolean isJavaVariable()
この構成要素が JavaVariable オブジェクトかどうか

5.1.2 JavaFile クラス

(1) 機能説明

Java ソースコードモデルにおけるファイルに関する情報を保持する。

(2) 属性

- String name
ファイルの名前
- String text
ファイルの内容 (ソースコードのテキスト)
- String packageName
解析ソースコードの所属するパッケージの名前
- ArrayList imports
解析ソースコードがインポートするパッケージ名のリスト
- ArrayList classes
ファイル内に存在するクラス (JavaClass)への参照のリスト
- long lastModifiedTime
ファイルの最終更新時刻

(3) メソッド

- void setName(String name)
名前に name を設定する
- String getName()
名前を取得する
- void setText(String text)
テキストに text を設定する
- String getText()
テキストを取得する
- boolean isJavaFile()
JavaFile かどうか(常に true)
- boolean isValid()
このファイルは有効であるかどうか
- void setPackageName(String name)
パッケージ名を name に設定する
- String getPackageName()
パッケージ名を取得する
- void addImportFiles(String name)
インポートパッケージ name をインポートパッケージリストに追加する
- ArrayList getImports()
インポートパッケージリストを取得する

- void addJavaClass(JavaClass jclass)
jclass をクラスリストに追加する
- JavaClass getJavaClass(String name)
名前 name を持つクラスへの参照を取得する
- ArrayList getJavaClasses()
クラスリストを取得する
- void setLastModified()
ファイルの最終更新時刻を設定する
- long getLastModified()
ファイルの最終更新時刻を取得する
- boolean isParsed()
このファイルはすでに解析済みであるかどうか
- boolean hasChanged()
このファイルが解析後に変更されているかどうか
- String getQualifiedNameInPackage(String name)
このファイルと同一のパッケージ内における名前 name の限定名を取得する

5.1.3 JavaClass クラス

(1) 機能説明

Java ソースコードモデルにおけるクラス情報を保持する。

(2) 属性

- String name
クラスの名前
- String qualifiedName
クラスの限定名
- boolean isInterface
このクラスはインタフェースかどうか
- JavaModifier modifier
このクラスの修飾子
- String superClassName
スーパークラス (親クラス) の名前
- String superClassNameList
スーパーインタフェースの名前の並び
- ArrayList methods
このクラスが所有するメソッドのリスト
- ArrayList fields
このクラスが所有するフィールドのリスト

- ArrayList usedTypes
このクラス内で利用されている型のリスト

(3) メソッド

- boolean isJavaClass()
JavaClass オブジェクトかどうか(常に true)
- void setName(String name)
名前を name に設定する
- String getName()
名前を取得する
- String getQualifiedName()
限定名を取得する
- void setInterface(boolean bool)
インタフェースであるかどうかを設定する
- boolean isInterface()
インタフェースかどうか
- void setModifier(JavaModifier modifier)
修飾子を modifier に設定する
- JavaModifier getModifier()
修飾子を取得する
- void setSuperClassName(String name)
スーパークラスの名前を name に設定する
- String getSuperClassName()
スーパーインタフェースの名前を取得する
- void setSuperClassNameList(String nameList)
スーパーインタフェースの名前の並びを nameList に設定する
- String getSuperClassNameList()
スーパーインタフェースの名前の並びを取得する
- void addJavaMethod(JavaMethod jmethod)
メソッド jmethod をメソッドリストに追加する
- ArrayList getJavaMethods()
メソッドリストを取得する
- JavaMethod getJavaMethod(String sig)
シグニチャ sig を持つメソッドへの参照を取得する
- void addJavaField(JavaStatement jst)
メソッド field をフィールドリストに追加する
- ArrayList getJavaFields()
フィールドリストを取得する

- `JavaStatement getJavaField(String name)`
名前 `name` を持つフィールドへの参照を取得する
- `JavaStatement getJavaField(JavaVariable jvar)`
変数 `jvar` に関するフィールドへの参照を取得する
- `void addUsedType(String type)`
クラス内で利用されている型を型リストに登録する
- `ArrayList getUsedTypes()`
クラス内で利用されている型リストを取得する
- `boolean isChildOf(JavaClass jclass)`
このクラスがクラス `jclass` のサブクラスであるかどうか
- `boolean isStatic()`
このクラスの修飾子は `static` かどうか
- `boolean isAbstract()`
このクラスの修飾子は `abstract` であるかどうか
- `boolean isFinal()`
このクラスの修飾子は `final` であるかどうか
- `boolean isPublic()`
このクラスの修飾子は `public` であるかどうか
- `boolean isProtected()`
このクラスの修飾子は `protected` であるかどうか
- `boolean isPrivate()`
このクラスの修飾子は `private` であるかどうか
- `boolean isDefault()`
このクラスの修飾子は `default`(アクセス制限に関する指定なし)であるかどうか

5.1.4 JavaMethod クラス

(1) 機能説明

Java ソースコードモデルにおけるメソッド情報を保持する .

(2) 属性

- `String name`
メソッドの名前
- `boolean isConstructor`
コンストラクタかどうか
- `JavaModifier modifier`
このメソッドの修飾子
- `String type`
このメソッドの型

- String qualifiedType
このメソッドの型の限定名
- ArrayList parameters
このメソッドの引数のリスト

(3) メソッド

- boolean isJavaMethod()
JavaMethod オブジェクトであるかどうか(常に true)
- void setName(String name)
メソッドの名前を name に設定する
- String getName()
メソッドの名前を取得する
- void setModifier(JavaModifier modifier)
このメソッドの修飾子を modifier に設定する
- JavaModifier getModifier()
このメソッドの修飾子を取得する
- void setType(String type)
このメソッドの型を type に設定する
- String getType()
このメソッドの型を取得する
- String getQualifiedType()
このメソッドの型の限定名を取得する
- void addParameter(JavaStatement jst)
引数 jst を引数リストに追加する
- void setParameters(ArrayList params)
引数リストを params に設定する
- ArrayList getParameters()
このメソッドの引数リストを取得する
- int getParameterNumber()
このメソッドの引数の数を取得する
- boolean equalsParameterTypes(JavaMethod jmethod)
このメソッドの引数の型の並びがメソッド jmethod と同一かどうか
- String get Signature()
このメソッドのシグニチャを取得する
- boolean equalsSignature(JavaMethod jmethod)
このメソッドと jmethod のシグニチャが同一であるかどうか
- void setConstructor(boolean bool)
コンストラクタかどうかを設定する

- `boolean isConstructor()`
コンストラクタかどうか
- `boolean isVoid()`
このメソッドの型が `void` かどうか
- `boolean isPublic()`
このメソッドの修飾子は `public` であるかどうか
- `boolean isProtected()`
このメソッドの修飾子は `protected` であるかどうか
- `boolean isPrivate()`
このメソッドの修飾子は `private` であるかどうか
- `boolean isDefault()`
このメソッドの修飾子は `default` であるかどうか
- `boolean isStatic()`
このメソッドの修飾子は `static` であるかどうか
- `boolean isAbstract()`
このメソッドの修飾子は `abstract` であるかどうか
- `boolean isFinal()`
このメソッドの修飾子は `final` であるかどうか

5.1.5 JavaStatement クラス

(1) 機能説明

Java ソースコードモデルにおける文 (変数宣言を含む) の情報を保持する .

(2) 属性

- `JavaVariableList defs`
定義変数リスト
- `JavaVariableList uses`
参照変数リスト
- `private boolean declaration`
変数宣言のみであるかどうか

(3) メソッド

- `boolean isJavaStatement()`
JavaStatement オブジェクトかどうか
- `void addDefVariable(JavaVariable jvar)`
定義変数 `jvar` を定義変数リストに追加する
- `void addUseVariable(JavaVariable jvar)`
参照変数 `jvar` を参照変数リストに追加する

- void addDefVariables(ArrayList jvars)
定義変数リストにリスト vars 内の変数を追加する
- void addUseVariables(ArrayList jvars)
参照変数リストにリスト vars 内の変数を追加する
- void setDefVariables(JavaVariableList list)
定義変数リストをリスト vars に設定する
- void setUseVariables(JavaVariableList list)
参照変数リストをリスト vars に設定する
- void clearDefVariables()
定義変数リストをクリアする
- void clearUseVariables()
参照変数リストをクリアする
- JavaVariableList getDefVariables()
定義変数リストを取得する
- JavaVariableList getUseVariables()
参照変数リストを取得する
- boolean containsDefVariable(JavaVariable v)
変数 v がこの文で定義されているか
- boolean containsUseVariable(JavaVariable v)
変数 v がこの文で参照されているか
- void setDeclaration(boolean bool)
変数宣言のみであるかどうかを設定する(代入がないか)
- boolean isDeclaration()
変数宣言のみであるかどうか
- JavaVariable getDeclaration()
変数宣言の変数を取得する

5.1.6 JavaVariable クラス

(1) 機能説明

Java ソースコードモデルにおける変数情報を保持する。

(2) 属性

- String name
変数の名前
- int id
変数の識別番号
- Token token
解析ソースコードにおいてこの変数に対応する字句 (トークン)

- `JavaModifier modifier`
変数の修飾子
- `String type`
変数の型
- `String qualifiedType`
変数の型の限定名
- `int sort`
変数の種類

(3) メソッド

- `boolean isJavaVariable()`
`JavaVariable` オブジェクトかどうか(常に `true`)
- `void setName(String name)`
変数の名前を `name` に設定する
- `String getName()`
変数の名前を取得する
- `void setID(int id)`
変数の識別番号を設定する
- `int getID()`
変数の識別番号を取得する
- `void setToken(Token token)`
この変数に対応する字句 (トークン) を設定する
- `Token getToken()`
この変数に対応する字句 (トークン) を取得する
- `void setModifier(JavaModifier modifier)`
この変数の修飾子を `modifier` に設定する
- `JavaModifier getModifier()`
この変数の修飾子を取得する
- `void setType(String type)`
この変数の型を設定する
- `String getType()`
この変数の型を取得する
- `String getQualifiedType()`
この変数の型の限定名を取得する
- `boolean isPrimitive()`
この変数の型がプリミティブであるかどうか
- `boolean equals(JavaVariable v)`
この変数は変数 `v` と同一 (同一の名前かつ同一の識別番号を持つ) かどうか

- `boolean isPublic()`
この変数の修飾子は `public` かどうか
- `boolean isProtected()`
この変数の修飾子は `protected` かどうか
- `boolean isPrivate()`
この変数の修飾子は `private` かどうか
- `boolean isDefault()`
この変数の修飾子は `default` かどうか
- `boolean isStatic()`
この変数の修飾子は `static` かどうか
- `boolean isFinal()`
この変数の修飾子は `final` かどうか
- `void setSort(int s)`
この変数の種類を `s` に設定する
- `int getSort()`
この変数の種類を取得する
- `void setField()`
この変数はフィールド変数に設定する
- `boolean isField()`
この変数はフィールド変数かどうか
- `void setLocal()`
この変数をローカル変数に設定する
- `boolean isLocal()`
この変数はローカル変数かどうか
- `boolean isParameter()`
この変数はパラメータ変数かどうか
- `void setFormal()`
この変数を仮引数に設定する
- `boolean isFormal()`
この変数は仮引数かどうか
- `void setActual()`
この変数を実引数に設定する
- `boolean isActual()`
この変数は実引数かどうか

5.2 グラフ

本ソフトウェアにおいて，グラフは制御フローグラフとプログラム依存グラフを抽象化

したものである．グラフは以下のクラスで構成されている．これらのクラスは，graph.util パッケージに格納されている．

- a) Graph
- b) GraphNode
- c) GraphEdge

グラフに関するクラス構成を図 5 3（含む CFG, PDG のクラス）に示す．

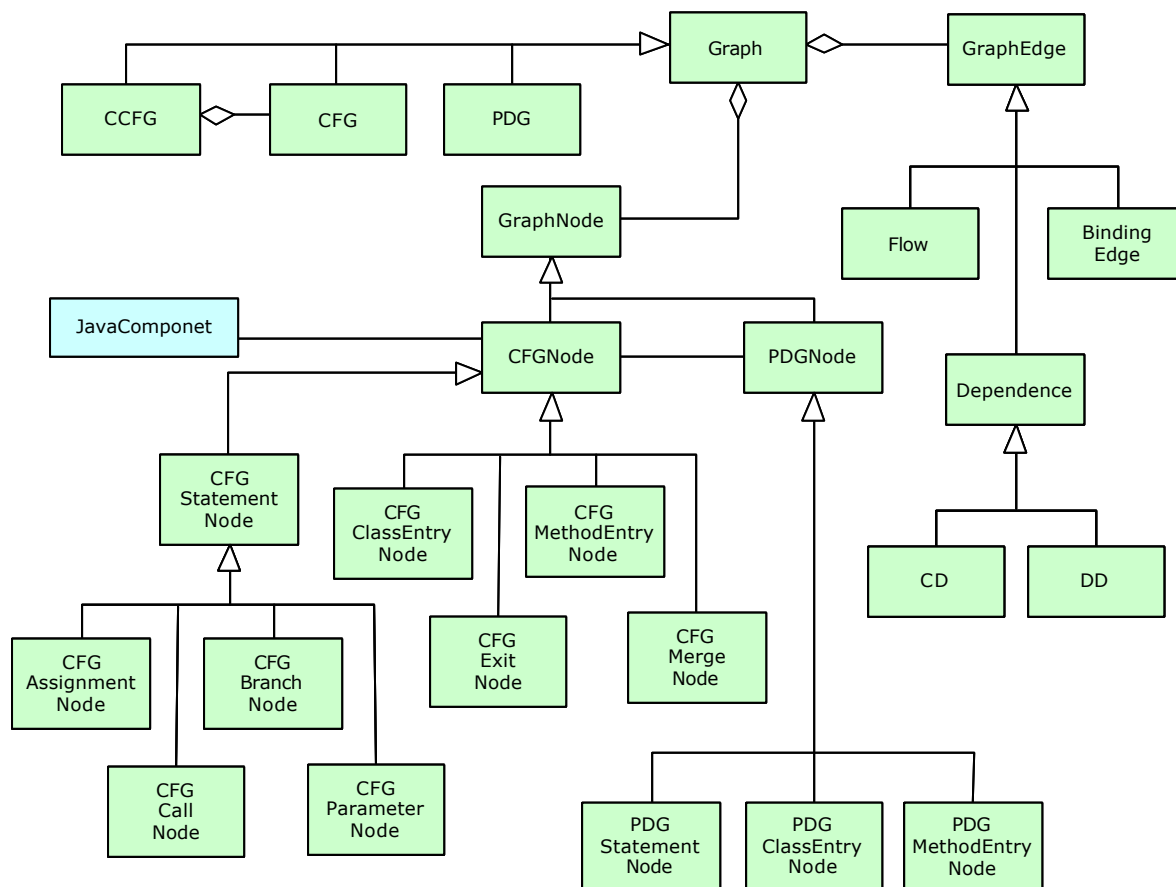


図 5 3 グラフ，制御フローグラフ，プログラム依存グラフのクラス構成

5.2.1 Graph クラス

(1) 機能説明

Java ソースコードモデルにおけるグラフ情報を保持する．

(2) 属性

- GraphComponentSet nodes
ノード集合
- GraphComponentSet edges
矢印集合

(3) メソッド

- void setNodes(GraphComponentSet set)
このグラフのノード集合を set に設定する
- GraphComponentSet getNodes()
このグラフのノード集合を取得する
- void setEdges(GraphComponentSet set)
このグラフの矢印集合を set に設定する
- GraphComponentSet getEdges()
このグラフの矢印集合を取得する
- void clear()
ノード集合と矢印集合をクリアする
- void add(GraphNode node)
このグラフにノード node を追加する
- void add(GraphEdge edge)
このグラフに矢印 edge を追加する
- void removeNode(GraphNode node)
このグラフからノード node を削除する
- void removeNode(GraphEdge edge)
このグラフから矢印 edge を削除する
- boolean contains(GraphNode node)
このグラフがノード node を含んでいるかどうか
- boolean contains(GraphEdge edge)
このグラフが矢印 edge を含んでいるかどうか

5.2.2 GraphNode クラス

(1) 機能説明

Java ソースコードモデルにおけるグラフのノード情報を保持する。

(2) 属性

- int sort
ノードの種類

(3) メソッド

- void setSort(int sort)
このノードの種類を sort に設定する
- int getSort()
このノードの種類を取得する

5.2.3 GraphEdge クラス

(1) 機能説明

Java ソースコードモデルにおけるグラフの矢印情報を保持する。

(2) 属性

- GraphNode src
接続元ノード
- GraphNode dst
接続先ノード
- int sort;
矢印の種類

(3) メソッド

- GraphEdge(GraphNode src, GraphNode dst)
ノード src からノード dst に矢印を作成する
- void setSort(int sort)
この矢印の種類を sort に設定する
- int getSort()
この矢印の種類を取得する
- GraphNode getSrcNode()
この矢印の接続元ノードを取得する
- GraphNode getDstNode()
この矢印の接続先ノードを取得する

5.3 制御フローグラフ

制御フローグラフ(CFG: Control Flow Graph)は以下のクラスで構成されている。これらのクラスは、graph.cfg パッケージに格納されている。

- a) CCFG
- b) CFG
- c) CFGNode
- d) CFGClassEntryNode
- e) CFGMethodEntryNode
- f) CFGExitNode
- g) CFGMergeNode
- h) CFFGStatementNode
- i) CFGAssignmentNode
- j) CFGBranchNode
- k) CFGCallNode
- l) CFGParameterNode

m) Flow

制御フローグラフに関するクラス構成を（ 5.2 の ）図 5 3 に示す .

5.3.1 CCFG クラス

（ 1 ） 機能説明

CCFG(Class CFG)情報を保持する .

（ 2 ） 属性

- ArrayList cfgs
CCFG に対応するクラス内に存在するメソッド群の CFG リスト
- CFGClassEntryNode startNode
CCFG の開始ノード
- CFGExitNode endNode;
CCFG の終了ノード

（ 3 ） メソッド

- setStartNode(CFGClassEntryNode node)
開始ノードを node 設定する
- CFGClassEntryNode getStartNode()
開始ノードを取得する
- void setEndNode(CFGExitNode node)
終了ノードを node に設定する
- public CFGExitNode getEndNode()
終了ノードを取得する
- public String getName()
名前を取得する
- public Flow getFlow(CFGNode src, CFGNode dst)
この CCFG において , ノード src を接続元 , ノード dst を接続先とするフローへの参照を取得する

5.3.2 CFG クラス

（ 1 ） 機能説明

CFG 情報を保持する .

（ 2 ） 属性

- CFGMethodEntryNode start
開始ノード
- CFGNode end
終了ノード

(3) メソッド

- void setStartNode(CFGMethodEntryNode node)
開始ノードを設定する
- CFGMethodEntryNode getStartNode()
開始ノードを取得する
- void setEndNode(CFGNode node)
終了ノードを設定する
- public CFGNode getEndNode()
終了ノードを取得する
- String getName()
名前を取得する
- void add(CFGNode node)
この CFG にノードを追加する
- void add(Flow edge)
この CFG にフローを追加する
- boolean isBranchNode(CFGNode node)
ノード node が分岐ノードかどうか
- boolean isLoopNode(CFGNode node)
ノード node がループノードかどうか
- boolean isJoinNode(CFGNode node)
ノード node が合流ノードかどうか
- Flow getFlow(CFGNode src, CFGNode dst)
この CFG において、ノード src を接続元、ノード dst を接続先とするフローへの参照を取得する
- public GraphComponentSet getFlowsTo(CFGNode dst)
この CFG において、ノード dst を接続先とするフローの集合を取得する
- GraphComponentSet getFlowsFrom(CFGNode src)
この CFG において、ノード src を接続元とするフローの集合を取得する
- GraphComponentSet getSrcNodes(CFGNode dst)
この CFG において、ノード dst を接続先とするノードの集合を取得する
- GraphComponentSet getDstNodes(CFGNode src)
この CFG において、ノード src を接続元とするノードの集合を取得する
- GraphNode getTrueSuccessor(CFGNode node)
この CFG において、ノード node の真方向のフローに対する次ノードを取得する
- GraphNode getFalseSuccessor(CFGNode node)
この CFG において、ノード node の偽方向のフローに対する次ノードを取得する

- `GraphComponentSet getCallNodes()`
この CFG におけるメソッド呼出しノードの集合を取得する。
- `GraphComponentSet getForwardReachableNodes(CFGNode from, CFGNode to)`
この CFG において、ノード from からノード to への順方向到達可能経路上のノード集合を取得する
- `GraphComponentSet getForwardReachableNodesWithoutLoopback(CFGNode from, CFGNode to)`
この CFG において、ノード from からノード to への順方向到達可能経路上のノード集合を取得する(ただしループバックフローは通過しない)
- `GraphComponentSet getBackwardReachableNodes(CFGNode fromNode, CFGNode toNode)`
この CFG において、ノード from からノード to への逆方向到達可能経路上のノード集合を取得する
- `GraphComponentSet getBackwardReachableNodesWithoutLoopback(CFGNode from, CFGNode to)`
この CFG において、ノード from からノード to への逆方向到達可能経路上のノード集合を取得する(ただしループバックフローは通過しない)

5.3.3 CFGNode クラス

(1) 機能説明

CFG におけるノード情報を保持する。

(2) 属性

なし

(3) メソッド

- `GraphComponentSet getPredecessors()`
このノードが属する CFG におけるフローの直前ノードを取得する
- `GraphComponentSet getSuccessors()`
このノードが属する CFG におけるフローの直後ノードを取得する
- `int getPredecessorsNumber()`
このノードが属する CFG における直前ノードの数を取得する
- `int getSuccessorsNumber()`
このノードが属する CFG における直後ノードの数を取得する
- `boolean isBranch()`
分岐ノードかどうか
- `boolean isJoin()`
合流ノードかどうか

- `boolean isNextToBranch()`
分岐ノードの直後ノードかどうか
- `boolean isLeader()`
基本ブロックのリーダーかどうか
- `boolean isLoop()`
ループノードかどうか
- `boolean isNormalStatement()`
文ノードかどうか
- `boolean isAssignmentSt()`
代入文ノードかどうか
- `boolean isBranchSt()`
分岐文(`if`, `switch-case`, `while`, `do`, `for-condition`) ノードかどうか
- `boolean isEntrySt()`
開始ノードかどうか
- `boolean isCallSt()`
メソッド呼出しノードかどうか
- `boolean isSwitchSt()`
`switch` 文ノードかどうか
- `boolean isReturnSt()`
`return` 文ノードかどうか
- `boolean isParameterSt()`
引数ノードかどうか
- `boolean isFormalSt()`
仮引数ノードかどうか
- `boolean isSwitchLabel()`
`switch` 文のラベル(`case`) ノードかどうか
- `boolean isMergeSt()`
合流文ノードかどうか
- `boolean hasDefVariable()`
このノードが文ノードの場合、定義変数を持つかどうか
- `boolean hasUseVariable()`
このノードが文ノードの場合、参照変数を持つかどうか

5.3.4 CFGClassEntryNode クラス

(1) 機能説明

CFG におけるクラス開始ノード情報を保持する。

(2) 属性

なし

(3) メソッド

- String getName()
クラスの名前を取得する

5.3.5 CFGMethodEntryNode クラス

(1) 機能説明

CFG におけるメソッド開始ノード情報を保持する。

(2) 属性

- ArrayList formalIns
メソッドの仮引数(formal-in)ノードのリスト
- ArrayList formalOuts
メソッドの仮引数(formal-out)ノードのリスト

(3) メソッド

- String getName()
メソッドの名前を取得する
- boolean isVoid()
メソッドの戻り値は型を持つかどうか
- boolean hasParameters()
メソッドは引数を持つかどうか
- void addFormalIn(CFGParameterNode node)
仮引数(formal-in)ノードのリストに引数ノード node を追加する
- void addFormalOut(CFGParameterNode node)
仮引数(formal-out)ノードのリストに引数ノード node を追加する
- ArrayList getFormalIns()
仮引数(formal-in)ノードのリストを取得する
- ArrayList getFormalOuts()
仮引数(formal-in)ノードのリストを取得する
- ArrayList getParameters()
すべての引数(formal-in と formal-out)ノードのリストを取得する
- int getNumParameters()
引数の数を取得する
- CFGParameterNode getFormalIn(int num)
num 番目の仮引数(formal-in)ノードを取得する
- CFGParameterNode getFormalOut(int num)
num 番目の仮引数(formal-out)ノードを取得する

5.3.6 CFGExitNode クラス

(1) 機能説明

CFG における終了ノード情報を保持する。

(2) 属性

なし

(3) メソッド

なし

5.3.7 CFGMergeNode クラス

(1) 機能説明

CFG における合流ノード情報を保持する。

(2) 属性

- CFGNode branch
この合流ノードに対応する分岐ノード

(3) メソッド

- void setBranchNode(CFGNode node)
分岐ノードを設定する
- CFGNode getBranchNode()
分岐ノードを取得する

5.3.8 CFFGStatementNode クラス

(1) 機能説明

CFG における文ノード情報を保持する。

(2) 属性

なし

(3) メソッド

- JavaVariableList getDefVariables()
このノードで定義されている変数の集合を取得する
- JavaVariableList getUseVariables()
このノードで参照されている変数の集合を取得する
- boolean containsDefVariable(JavaVariable v)
このノードで変数 v が定義されているかどうか
- boolean containsUseVariable(JavaVariable v)
このノードで変数 v が参照されているかどうか

5.3.9 CFGAssignmentNode クラス

(1) 機能説明

CFG における代入文ノード情報を保持する。

(2) 属性

なし

(3) メソッド

なし

5.3.10 CFGBranchNode クラス

(1) 機能説明

CFG における分岐文ノード情報を保持する。

(2) 属性

なし

(3) メソッド

なし

5.3.11 CFGCallNode クラス

(1) 機能説明

CFG におけるメソッド呼出し文ノード情報を保持する。

(2) 属性

- String name;
呼び出されメソッドの名前
- ArrayList actualIns
実引数(actual-in) ノードのリスト
- ArrayList actualOuts
実引数(actual-out) ノードのリスト

(3) メソッド

- void setName(String name)
呼び出されメソッドの名前を name に設定する
- String getName()
呼び出されメソッドの名前を取得する
- boolean hasArguments()
このメソッド呼出しに引数があるかどうか
- void addActualIn(CFGParameterNode node)
実引数(actual-in) ノードのリストに引数ノード node を追加する
- void addActualOut(CFGParameterNode node)
実引数(actual-out) ノードのリストに引数ノード node を追加する

- `ArrayList getActualIns()`
実引数(actual-in)ノードのリストを取得する
- `ArrayList getActualOuts()`
実引数(actual-out)ノードのリストを取得する
- `ArrayList getArguments()`
すべての引数(actual-in と actual-out)ノードのリストを取得する
- `int getNumArguments()`
引数の数を取得する
- `CFGParameterNode getActualIn(int num)`
num 番目の仮引数(formal-in)ノードを取得する
- `CFGParameterNode getActualOut(int num)`
num 番目の仮引数(formal-out)ノードを取得する
- `boolean callSelf()`
呼出し先が自分自身かどうか

5.3.1.2 CFGParameterNode クラス

(1) 機能説明

CFG における引数ノード情報を保持する。

(2) 属性

- `int ordinal`
何番目の引数か

(3) メソッド

- `void setOrdinal(int num)`
引数の位置を num 番目に設定する
- `int getOrdinal()`
引数の位置を取得する
- `JavaVariable getDefVariable()`
この引数ノードの定義変数を取得する
- `JavaVariable getUseVariable()`
この引数ノードの参照変数を取得する

5.3.1.3 Flow クラス

(1) 機能説明

CFG における矢印情報を保持する。

(2) 属性

- `boolean loopback`
ループバックであるかどうか

(3) メソッド

- Flow(CFGNode src, CFGNode dst)
ノード src からノード dst に制御フローを作成する
- void setTrue()
このフローを真方向に設定する
- boolean isTrue()
このフローが真方向かどうか
- void setFalse()
このフローを偽方向に設定する
- boolean isFalse()
このフローが偽方向かどうか
- void setLoopBack(boolean bool)
このフローをループバックに設定する
- boolean isLoopBack()
このフローがループバックかどうか
- void setFallThrough()
このフローを fall-through に設定する
- boolean isFallThrough()
このフローが fall-through かどうか
- void setParameter()
このフローを引数渡しに設定する
- boolean isParameter()
このフローが引数渡しかどうか

5.4 プログラム依存グラフ

プログラム依存グラフ(PDG: Program Dependence Graph)は以下のクラスで構成されている。これらのクラスは、graph.pdg パッケージに格納されている。

- a) PDG
- b) PDGNode
- c) PDGClassEntryNode
- d) PDGMethodEntryNode
- e) PDGStatementNode
- f) Dependence
- g) CD
- h) DD
- i) BindingEdge

プログラム依存グラフに関するクラス構成を (5.2 の) 図 5.3 に示す。

5.4.1 PDG クラス

(1) 機能説明

PDG 情報を保持する。

(2) 属性

- PDGMethodEntryNode entryNode
この PDG の入口ノード

(3) メソッド

- public PDG(CFG cfg)
CFG cfg の PDG を構築する
- void setEntryNode(PDGMethodEntryNode node)
この PDG の入口ノードを node に設定する
- PDGMethodEntryNode getEntryNode()
この PDG の入口ノードを取得する
- String getName()
この PDG に対応するメソッドの名前を取得する
- PDGNode getNode(CFGNode node)
CFG のノード node に対応する PDG ノードを取得する
- void add(PDGNode node)
この PDG にノード node を追加する
- void add(Dependence edge)
この PDG に依存関係矢印 edge を追加する
- boolean isDominated(PDGNode node)
この PDG において、ノード node は別のノードに支配されているかどうか
- boolean isTrueDominated(PDGNode node)
この PDG において、ノード node は真方向の制御依存関係を持つかどうか
- boolean isFalseDominated(PDGNode node)
この PDG において、ノード node は偽方向の制御依存関係を持つかどうか

5.4.2 PDGNode クラス

(1) 機能説明

PDG におけるノード情報を保持する。

(2) 属性

なし

(3) メソッド

- PDGNode(CFGNode node)
CFG のノード node に対応する PDG のノードを作成する

- `boolean isBranch()`
分岐ノードかどうか
- `boolean isLoop()`
ループノードかどうか
- `boolean containsDefVariable(JavaVariable v)`
このノードにおいて変数が定義されているかどうか
- `boolean containsUseVariable(JavaVariable v)`
このノードにおいて変数が参照されているかどうか

5.4.3 PDGClassEntryNode クラス

(1) 機能説明

PDG におけるクラス入口ノード情報を保持する。

(2) 属性

なし

(3) メソッド

- `String getName()`
対応するクラスの名前を取得する

5.4.4 PDGMethodEntryNode クラス

(1) 機能説明

PDG におけるメソッド入口ノード情報を保持する。

(2) 属性

なし

(3) メソッド

- `String getName()`
対応するメソッドの名前を取得する

5.4.5 PDGStatementNode クラス

(4) 機能説明

PDG における文ノード情報を保持する。

(5) 属性

なし

(6) メソッド

- `PDGStatementNode(CFGStatementNode node)`
CFG のノード `node` に対応する PDG ノードを作成する
- `boolean isStatementNode()`
PDGStatementNode オブジェクトかどうか(常に true)

- `JavaVariableList getDefVariables()`
このノードで定義されている変数の集合を取得する
- `JavaVariableList getUseVariables()`
このノードで参照されている変数の集合を取得する
- `boolean containsDefVariable(JavaVariable v)`
このノードで変数 `v` が定義されているかどうか
- `boolean containsUseVariable(JavaVariable v)`
このノードで変数 `v` が参照されているかどうか

5.4.6 Dependence クラス

(1) 機能説明

PDG における依存関係矢印情報を保持する。

(2) 属性

なし

(3) メソッド

- `Dependence(PDGNode src, PDGNode dst)`
ノード `src` からノード `dst` に依存関係矢印を作成する
- `boolean isCD()`
制御依存関係矢印かどうか
- `boolean isDD()`
データ依存関係(Def-Def)矢印かどうか
- `boolean isDU()`
データ依存関係(Def-Use)矢印かどうか

5.4.7 CD クラス

(1) 機能説明

PDG における制御依存関係(CD: Control Dependence)矢印情報を保持する。

(2) 属性

なし

(3) メソッド

- `CD(PDGNode src, PDGNode dst)`
ノード `src` からノード `dst` に依存関係矢印を作成する
- `void setTrue()`
この矢印を真方向制御依存関係に設定する
- `boolean isTrue()`
この矢印が真方向制御依存関係かどうか

- void setFalse()
この矢印を偽方向制御依存関係に設定する
- boolean isFalse()
この矢印が真方向制御依存関係かどうか
- void setFall()
この矢印を fall 制御依存関係に設定する
- boolean isFall()
この矢印が fall 制御依存関係かどうか

5.4.8 DD クラス

(1) 機能説明

PDG におけるデータ依存関係(DD: Data Dependence)矢印情報を保持する .

(2) 属性

- private JavaVariable var
データ依存関係の変数
- PDGNode loopCarriedNode
ループ依存ノード

(3) メソッド

- DD(PDGNode src, PDGNode dst)
ノード src からノード dst にデータ依存関係矢印を作成する
- DD(PDGNode src, PDGNode dst, JavaVariable v)
ノード src からノード dst に変数 v に関するデータ依存関係矢印を作成する
- void setVariable(JavaVariable v)
この矢印に関するデータ依存変数を v に設定する
- JavaVariable getVariable()
この矢印に関するデータ依存変数を取得する
- void setLoopCarriedNode(PDGNode n)
この矢印のループ依存ノードを設定する
- PDGNode getLoopCarriedNode()
この矢印のループ依存ノードを取得する
- boolean isLoopCarried()
この矢印はループ依存かどうか
- void setDefUse()
この矢印を Def-Use データ依存関係に設定する
- boolean isDefUse()
この矢印が Def-Use データ依存関係かどうか

- void setDefOrder()
この矢印を Def-Def データ依存関係に設定する
- boolean isDefOrder()
この矢印が Def-Def データ依存関係かどうか

5.4.9 BindingEdge クラス

(1) 機能説明

CIDG(PDG)におけるクラス間およびメソッド間矢印情報を保持する。

(2) 属性

- JavaVariable var
この矢印に関連する変数

(3) メソッド

- BindingEdge(GraphNode src, GraphNode dst)
ノード src からノード dst にクラス矢印を作成する
- BindingEdge(GraphNode src, GraphNode dst, JavaVariable v)
ノード src からノード dst に変数 v に関する矢印を作成する
- void setVariable(JavaVariable v)
この矢印に変数 v を関連づける
- JavaVariable getVariable()
この矢印に関連付けされた変数を取得する
- void setClassMember()
この矢印をクラス-メンバ関係に設定する
- boolean isClassMember()
クラス-メンバ関係かどうか
- void setMethodCall()
この矢印をメソッド呼出し関係に設定する
- boolean isMethodCall()
メソッド呼出し関係かどうか
- void setParameter()
この矢印を引数渡し関係に設定する
- public boolean isParameter()
引数渡し関係に設定する
- void setArgument()
この矢印を戻り値渡し関係に設定する
- boolean isArgument()
戻り値渡し関係かどうか

6 . リファクタリング部

(1) 機能説明

利用者が入力として指定した Java ソースコードに対して ,GUI を介して入力したりファクタリング操作や変更目標を適用し , 変換後のソースコードを出力する .

(2) 入力データ

Java ソースコード , リファクタリング操作 , 変更目標 (コマンド)

(3) 出力データ

変換後の Java ソースコード

(4) 構成

各部におけるクラス間の関係は , 図 2 を参照のこと .

6 . 1 構文解析部

(1) 機能説明

入力として指定された Java ソースコード (ファイル) を先頭から読み込み , 字句に分解し , 構文解析を行い , 各クラスに対して構文木 (AST: Abstract Syntax Tree) とソースコードモデル , および制御フローグラフ (CFG: Control Flow Graph) を構築する .

(2) 入力データ

Java ソースコード

(3) 出力データ

AST (parser.ast パッケージ)

ソースコードモデル(model パッケージ)

CFG (graphs.cfg パッケージ)

(4) 制約条件

Java の文法規則において , 以下の構文は解析対象外とする .

- インナークラス(匿名クラスを含む)
- 例外処理(try-catch- finally 文, throw 文)

これらの字句を含む Java ソースコードを入力した場合 , 構文解析の実行自体は失敗しないが , 解析結果が不正確となることがある .

(5) 構成

a) JavaParser (parser パッケージ)

b) JavaModelFactory (parser パッケージ)

c) CFGFactory (graphs.cfg パッケージ)

JavaParser クラスおよび構文木の各ノードを構成するクラス (parser.ast パッケージ内のクラス) は ,JavaCC (Java Compiler Compiler)により Java 構文規則から生成するため , これらのクラスの属性 , メソッドに関する記述は省略する .

6.1.1 JavaModelFactory クラス

(1) 機能説明

Java ソースコードに対して字句解析，構文解析を実行する．さらに，解析済みのファイルを管理する．

(2) 属性

なし

(3) メソッド

- void setJDKFile(String name)
JDK ファイルの名前を name に設定する
- public String getJDKFile()
JDK ファイルの名前を取得する
- JavaFile getJavaFile(String name)
名前 name を持つ解析済み JavaFile への参照を取得する
- JavaFile getCloneOfJavaFile(JavaFile jfile)
jfile のクローンを作成し，その参照を取得する
- JavaFile getParsedFile(String name)
名前 name を持つファイルを解析し，解析後の JavaFile への参照を取得する
- boolean exists(String name)
名前 name を持つ JavaFile が解析済みファイル一覧に存在するかどうか
- boolean exists(JavaFile jfile)
jfile が解析済みファイル一覧に存在するかどうか
- addParsedFile(JavaFile jfile)
解析済ファイル一覧に jfile を登録する．
- void removeAllParsedFiles()
すべての解析済ファイルを削除する
- void removeParsedFile(JavaFile jfile)
jfile を解析済ファイル一覧から削除する
- void removeParsedSummaryFile(String name)
名前 name を持つ JavaFile を解析済ファイル一覧から削除する
- void removeAllParsedSummaryFiles()
すべての簡易解析済みファイルを削除する
- boolean isParsed(String name)
名前 name を持つファイルが解析されているかどうか
- JavaFile parseEachFile(String name)
名前 name を持つファイルを解析する
- JavaFile parseEachFile(String name, String text)
テキスト text を解析し，name という名前で解析済みファイル一覧に登録する

- SummaryJavaFile getSummaryJavaFile(String name)
名前 name を持つファイルの簡易解析情報への参照を取得する
- SummaryJavaFile parseEachSummaryFile(String name, String text)
テキスト text を簡易解析し ,name という名前で解析済みファイル一覧に登録する
- SummaryJavaFile parseEachSummaryFile(String name)
名前 name を持つファイルを簡易解析する
- boolean existsInParsedSummaryFiles(String name)
名前 name を持つファイルがすでに簡易解析されているかどうか
- void collectInformation()
簡易解析情報を集め , 解析済みファイルに反映させる
- String getQualifiedName(String name)
名前 name の限定名を取得する
- String getQualifiedNameList(String nameList)
名前の並びを限定名の並びに変換し , その文字列を取得する
- SummaryJavaClass getSummaryJavaClassInSelf(String name)
このファイル内に存在する名前 name のクラスを取得する
- SummaryJavaField getFieldType(JavaClass jclass, String name)
クラス jclass から参照可能な名前 name を持つフィールドへの参照を取得する
- SummaryJavaField getFieldType(JavaClass jclass, String cname, String name)
名前 cname を持つクラスの祖先クラス内で , クラス jclass から参照可能な名前 name を持つフィールドへの参照を取得する
- SummaryJavaField getFieldTypeAt(JavaClass jclass, String name)
クラス jclass 内で参照可能な名前 name を持つフィールドへの参照を取得する
- SummaryJavaField getFieldTypeAt(JavaClass jclass, String cname, String name)
名前 cname を持つクラス内で , クラス jclass から参照可能な名前 name を持つフィールドへの参照を取得する
- String wideningConversions(List vlist)
変数リスト vlist 内の各変数の型に対してワイドニング変換を適用した後の , 型の並びを取得する .
- SummaryJavaMethod getMethodType(JavaClass jclass, String name, ArrayList params)
クラス jclass から参照可能な名前 name と引数 params を持つメソッドへの参照を取得する
- SummaryJavaMethod getMethodType(JavaClass jclass, String cname, String name, ArrayList params)
名前 cname を持つクラスの祖先クラス内で , クラス jclass から参照可能な名前

name と引数 params を持つメソッドへの参照を取得する

- SummaryJavaMethod getMethodTypeAt(JavaClass jclass, String cname, String name, ArrayList params)
名前 cname を持つクラスの祖先クラス内で、クラス jclass から参照可能な名前 name と引数 params を持つメソッドへの参照を取得する

6.1.2 CFGFactory クラス

(1) 機能説明

AST から CFG を構築する。外部からは GraphFactory クラスを介して利用される。

(2) 属性

なし

(3) メソッド

- Object visit(ASTCompilationUnit node, Object data)
AST ノード node に合わせて CFG ノードおよびフローを作成する。data はノード間でのデータの受け渡しに使用する。実際のソフトウェアには、AST の各ノードに対応する visit メソッドがすべて記述されている。これらの処理の概要は同じであるため、ここでは説明を省略する。

6.2 依存解析部

(1) 機能説明

Java ソースコードから作成した CFG に対して、データ依存解析と制御依存解析を適用しプログラム依存グラフ(PDG: Program Dependence Graph)を作成する。本研究開発システムでは、PDG を拡張したクラス依存グラフ(CIDG: Class Dependence Graph)の矢印を追加する。さらに、PDG を用いてプログラムスライスを計算する。

(2) 入力データ

CFG

(3) 出力データ

PDG, CIDG (graphs.pdg パッケージ)

スライス(graphs.slice パッケージ)

(4) 制約条件

Java JDK 内のクラスについては、メモリ空間と実行速度の問題から依存解析の対象としない。

(5) 構成

本機能は、以下のクラスで構成されている。

- a) CDGFactory (graphs.pdg パッケージ)
- b) DDGFactory (graphs.pdg パッケージ)
- c) Slice (graphs.slice パッケージ)

6.2.1 CDGFactory クラス

(1) 機能説明

CFG から制御依存関係を抽出する．外部からは GraphFactory クラスを介して利用される．

(2) 属性

なし

(3) メソッド

- void create(PDG pdg, CFG cfg)
CFG cfg から PDG pdg の制御依存関係を求める

6.2.2 DDGFactory クラス

(1) 機能説明

CFG からデータ依存関係を抽出する．外部からは GraphFactory クラスを介して利用される．

(2) 属性なし

なし

(3) メソッド

- void create(PDG pdg, CFG cfg)
CFG cfg から PDG pdg のデータ依存関係を求める

6.2.3 Slice クラス

(4) 機能説明

スライシング基準に基づき PDG からスライスを抽出する．

(5) 属性

- PDGNode criteriaNode
スライシング基準ノード
- JavaVariable criteriaVar
スライシング基準変数

(6) メソッド

- Slice(PDGNode node, JavaVariable var)
ノード node 変数 var に関するスライスを作成する
- Slice(JavaVariable var)
変数 var に関するスライスを作成する

6.3 コード変換部

(1) 機能説明

入力 Java ソースコードの構文木，CFG, PDG に対して，与えられたリファクタリン

グ操作名に従い変換列を作成し、変換操作を適用する。その際、変換後のソースコードに対する整形も行う。

(2) 入力データ

構文木, CFG, PDG, 操作名

(3) 出力データ

リファクタリング後(変換および整形後の)Java ソースコード

(4) 構成

本機能は、以下のクラスで構成されている。

- a) Refactoring (refactor パッケージ)
- b) ClassRefactoring (refactor パッケージ)
- d) MethodRefactoring (refactor パッケージ)
- e) FieldRefactoring (refactor パッケージ)
- f) VariableRefactoring (refactor パッケージ)
- g) MiscellaneousRefactoring (refactor パッケージ)
- h) RefactoringImpl (refactor.util パッケージ)
- i) RefactoringVisitor (refactor.util パッケージ)
- j) PrintVisitor (refactor.util パッケージ)

b)~g)は、各ファクタリング操作に応じて、Refactoring クラス（抽象クラス）を実装したサブクラスである。

6.3.1 Refactoring クラス（抽象クラス）

(1) 機能説明

Java ソースコードに対してリファクタリングを適用する。

(2) 属性

- String rootDir
探索ディレクトリ
- JavaFile jfile
リファクタリング対象ファイル
- JavaClass jclass
リファクタリング対象クラス
- JavaComponent javaComp
指定トークン
- List changedFiles
リファクタリング操作によって変更されたファイルのリスト
- String logMessage
リファクタリングログメッセージ

- JavaFile originalJFile
変換前のファイル

(3) メソッド

- static RefactoringCommand create(String command, String packagePath)
メニュー名 command に基づきリファクタリングを実行するオブジェクトを packagePath に存在するクラスから生成する
- void setSource(JavaFile jf, int line, int column)
リファクタリング対象ファイル 利用者の選択テキストの位置(行 line と列 column) を設定する
- void setRootDir(String rootDir)
探索ディレクトリを設定する
- List getChangedFiles()
変更ファイルのリストを取得する
- public DisplayedFile getChangedFile(String name)
変更ファイルのリストから名前 name を持つファイルへの参照を取得する
- String getRefactoringLog()
リファクタリングログを取得する
- void execute()
リファクタリングを実行する
- boolean confirm()
利用者にリファクタリング結果を確認する
- void restore()
ソースコードの変更を破棄して、対象ファイルを元に戻す
- void clean()
解析ファイルに対する解析情報をクリアする

以下のメソッドは、上記の execute から呼び出される抽象メソッドである。Refactroing クラスのサブクラスにおいて、各リファクタリング操作は、これらの抽象メソッドを実装した具象メソッドに記述されている。

- abstract void setUp()
リファクタリング適用前の処理を実行する
- abstract void preconditions()
リファクタリングの前提条件を検査する
- abstract void transform()
ソースコードを変換する
- abstract void additionalTransformation()
実行したリファクタリングにより影響を受けるファイルを検索し、処理する

- abstract String getLog()
各リファクタリング操作に関するログを取得する

6.3.2 ClassRefactoring クラス

(1) 機能説明

クラスリファクタリングに関する共通処理を実行する。

(2) 属性

なし

(3) メソッド

- static RefactoringCommand create(String command)
クラスメニュー下のリファクタリング実行オブジェクトを生成する
- void setUp()
クラスリファクタリングに関する前処理を実行する

6.3.3 MethodRefactoring クラス

(1) 機能説明

メソッドリファクタリングに関する共通処理を実行する。

(2) 属性

- JavaMethod jmethod
リファクタリング対象メソッド

(3) メソッド

- static RefactoringCommand create(String command)
メソッドメニュー下のリファクタリング実行オブジェクトを生成する
- void setUp()
メソッドリファクタリングに関する前処理を実行する
- void showFilesCallingMethod(JavaMethod jm, List files)
リファクタリングにより変更されたメソッドを呼び出している箇所を表示する

6.3.4 FieldRefactoring クラス

(1) 機能説明

フィールドリファクタリングに関する共通処理を実行する。

(2) 属性

- JavaVariable jvar
リファクタリング対象フィールド

(3) メソッド

- static RefactoringCommand create(String command)
フィールドメニュー下のリファクタリング実行オブジェクトを生成する

- void setUp()
メソッドリファクタリングに関する前処理を実行する
- void showFilesUsingField(JavaVariable jv, String name, List files)
リファクタリングにより変更されたフィールドを参照している箇所を表示する

6.3.5 VariableRefactoring クラス

(1) 機能説明

ローカル変数リファクタリングに関する共通処理を実行する。

(2) 属性

- JavaVariable jvar
リファクタリング対象ローカル変数

(3) メソッド

- static RefactoringCommand create(String command)
ローカル変数メニュー下のリファクタリング実行オブジェクトを生成する
- void setUp()
ローカル変数リファクタリングに関する前処理を実行する

6.3.6 MiscellaneousRefactoring クラス

(1) 機能説明

その他のリファクタリングに関する共通処理を実行する

(2) 属性

なし

(3) メソッド

- static RefactoringCommand create(String command)
その他メニュー下のリファクタリング実行オブジェクトを生成する

6.3.7 RefactoringImpl クラス

(1) 機能説明

リファクタリングの前提条件を検査するための機能を実現したメソッド群

(2) 属性

なし

(3) メソッド

- String findSameNameFileInPackage(String name, JavaFile jf)
ファイル jf と同一のパッケージ内に存在する名前 name を持つファイルを取得する

- `boolean existsSameNameFileInPackage(String name, JavaFile jf)`
ファイル `jf` と同一のパッケージ内に存在する名前 `name` をもつファイルが存在するかどうか
- `boolean isUsedInFile(String type, JavaFile jf)`
ファイル `jf` において、名前 `type` を持つ型が利用されているかどうか
- `boolean isUsedInClass(String type, JavaClass jc)`
クラス `jc` において、名前 `name` を持つ型が利用されているかどうか
- `List collectFilesUsingClass(JavaClass jc)`
クラス `jc` を利用しているファイルの集合を取得する
- `List collectFilesCallingMethod(JavaMethod jm)`
メソッド `jm` を利用しているファイルの集合を取得する
- `List collectFilesContainingSubclassesCallingMethod(JavaMethod jm)`
子孫クラスにおいてメソッド `jm` を呼んでいるクラスを含むファイルの集合を取得する
- `List collectFilesUsingField(JavaVariable jv)`
フィールド `jv` を利用しているファイルを取得する
- `List collectFilesContainingSubclassesUsingField(JavaVariable jv)`
子孫クラスにおいてフィールド `jv` を利用しているクラスを含むファイルの集合を取得する
- `String getClassName(String name)`
名前 `name` から限定されたクラス名(ファイル名#クラス名)を取得する
- `boolean existsSameNameClassInPackage(String name, JavaFile jf)`
ファイル `jf` と同一のパッケージ内に名前 `name` を持つクラスが存在するかどうか
- `boolean existsSameNameClassInFile(String name, JavaFile jf)`
ファイル `jf` 内に名前 `name` を持つクラスが存在するかどうか
- `boolean existsSameClassInFile(JavaClass jc, JavaFile jf)`
ファイル `jf` と同一のパッケージ内に `jc` と同じクラスが存在するかどうか
- `List collectSubclasses(JavaClass jclass)`
クラス `jclass` の子孫クラスの集合を取得する
- `List collectChildren(JavaClass jclass)`
クラス `jclass` の直属の子クラスの集合を取得する
- `JavaMethod getGetter(String name, JavaVariable jv)`
名前 `name` を持つ変数 `jv` の Setter を取得する
- `JavaMethod getSetter(String name, JavaVariable jv)`
名前 `name` を持つ変数 `jv` の Getter を取得する
- `boolean existsSameNameMethodInClass(String name, JavaClass jc)`
クラス `jc` 内に同じ名前 `name` を持つメソッドが存在するかどうか

- `boolean existsSameParameterMethodInClass(JavaMethod jm, JavaClass jc)`
クラス jc 内にメソッド jm と同じパラメータを持つメソッドが存在するかどうか
- `boolean existsSameMethodInClass(JavaMethod jm, JavaClass jc)`
クラス jc 内にメソッド jm と同じ(メソッド名とシグニチャが等しい)メソッドが存在するかどうか
- `boolean existsSameMethodBetweenClasses(JavaClass src, JavaClass dst)`
クラス src とクラス dst 間に同一のメソッドが存在するかどうか
- `List collectSameMethodInSuperclasses(JavaMethod jm)`
祖先クラスに存在するメソッド jm と同じメソッドの集合を取得する
- `List collectSameMethodInSubclasses(JavaMethod jm)`
子孫クラスに存在するメソッド jm と同じメソッドの集合を取得する
- `List collectSameMethodInClasses(JavaMethod jm, List classes)`
クラス集合 classes 内のクラスに存在するメソッド jm と同じメソッドの集合を取得する
- `boolean callsMethodsInClass(JavaMethod jm, JavaClass jc)`
クラス jc 内でメソッド jm を呼び出しているかどうか
- `List collectCalledMethodsInClassOrAncestors(JavaMethod jm)`
メソッド jm の存在するクラスとその祖先クラス内で、メソッド jm を呼び出しているクラスの集合を取得する
- `List collectCalledMethodInClass(JavaMethod jm, JavaClass jc)`
クラス jc のメソッド jm から呼び出されているメソッドの集合を取得する
- `boolean callsMethodsInAncestors(JavaMethod jm)`
メソッド jm の存在するクラスの祖先クラス内で、メソッド jm を呼び出しているクラスの集合を取得する
- `boolean callsMethodsInClassOrAncestors(JavaMethod jm)`
メソッド jm の存在するクラスあるいはその祖先クラスからメソッド jm が呼び出されているかどうか
- `boolean callsPrivateMethods(JavaMethod jm)`
メソッド jm が他の private メソッドを呼び出しているかどうか
- `boolean isCalledInClass(JavaMethod jm, JavaClass jc)`
メソッド jm がクラス jc 内で呼び出されているかどうか
- `List collectCalledMethodsInClass(JavaClass jc)`
クラス jc から呼び出されているメソッドの集合を取得する
- `List collectCalledMethodsInMethod(JavaMethod jm)`
メソッド jm から呼び出されているメソッドの集合を取得する
- `List collectSubclassesCallingMethod(JavaMethod jm, JavaClass jclass)`
メソッド jm を呼び出しているクラス jc の子孫クラスの集合

- `List collectChildrenCallingMethod(JavaMethod jm)`
メソッド `jm` を呼び出しているクラス `jc` の直属の子クラスの集合を取得する
- `JavaVariableList collectVariablesReferringToClass(JavaMethod jm, JavaClass jc)`
メソッド `jm` 内でクラス `jc` のオブジェクトを参照している変数集合を取得する
- `boolean refersToPrivateField(JavaMethod jm)`
メソッド `jm` が `private` フィールドを参照しているかどうか
- `JavaVariableList collectVariablesReferringToPrivateFields(JavaMethod jm)`
メソッド `jm` 内で `private` フィールドを参照している変数の集合を取得する
- `boolean usesFieldsInClass(JavaMethod jm, JavaClass jc)`
メソッド `jm` がクラス `jc` のフィールドを利用しているかどうか
- `boolean existsSameNameFieldInClass(String name, JavaClass jc)`
クラス `jc` 内に同じ名前 `name` を持つフィールドが存在するかどうか
- `boolean existsSameFieldInClass(JavaVariable jv, JavaClass jc)`
クラス `jc` 内に `jv` と同じフィールドが存在するかどうか
- `boolean existsSameFieldBetweenClasses(JavaClass src, JavaClass dst)`
クラス `src` とクラス `dst` 間に同一のフィールドが存在するかどうか
- `boolean existsSameFieldInSuperclasses(JavaVariable jv)`
フィールド `jv` の祖先クラスに `jv` と同じフィールドが存在するかどうか
- `boolean existsSameFieldInSubclasses(JavaVariable jv)`
フィールド `jv` の子孫クラスに `jv` と同じフィールドが存在するかどうか
- `boolean existsSameFieldInClasses(JavaVariable jv, List classes)`
クラス集合 `classes` 内のクラスに `jv` と同じフィールドが存在するかどうか
- `boolean usesOtherFieldsInClassAtDeclaration(JavaVariable jv)`
フィールド `jv` がその宣言部において他のフィールドを利用しているかどうか
- `boolean existsItsDeclarationInClass(JavaVariable jv)`
フィールド `jv` の宣言が `jv` の所属するクラスに存在するかどうか
- `boolean existsItsDeclarationInClass(JavaVariable jv, JavaClass jc)`
フィールド `jv` の宣言がクラス `jc` に存在するかどうか
- `boolean usesOtherFieldsInAncestorsAtDeclaration(JavaVariable jv)`
フィールド `jv` がその宣言部において祖先クラスのフィールドを利用しているかどうか
- `boolean existsItsDeclarationInAncestors(JavaVariable jv)`
フィールド `jv` の宣言が `jv` の祖先クラスに存在するかどうか
- `JavaVariableList collectUsedFieldsInClassOrAncestors(JavaMethod jm)`
メソッド `jm` の存在するクラスおよびその祖先クラスにおいて `jm` 内で利用されているフィールドの集合を取得する

- `JavaVariableList collectUsedFieldsInClassOrAncestors(JavaVariable jv)`
フィールド `jv` の存在するクラスおよびその祖先クラスにおいて `jv` に利用されているフィールドの集合を取得する
- `boolean isDirectlyUsedInClass(JavaVariable jv, JavaClass jc)`
フィールド `jv` がクラス `jc` で直接(アクセッサを介さずに)利用されているかどうか
- `boolean isDirectlyUsedInClass(JavaVariable jv, JavaClass jc, JavaMethod gm, JavaMethod sm)`
フィールド `jv` がクラス `jc` でアクセッサ `gm` と `sm` を介さずに利用されているかどうか
- `List collectSubclassesUsingField(JavaVariable jv, JavaClass jclass)`
フィールド `jv` の存在するクラスの子孫クラスで、`jv` を利用しているクラスの集合を取得する
- `List collectChildrenUsingField(JavaVariable jv)`
フィールド `jv` の存在するクラスの子クラスで、`jv` を利用しているクラスの集合を取得する
- `boolean isUsedInClass(JavaVariable jv, JavaClass jc)`
クラス `jc` 内でフィールド `jv` が利用されているかどうか
- `boolean isUsedInClass(JavaVariable jv, JavaMethod jm)`
メソッド `jm` 内でフィールド `jv` が利用されているかどうか
- `boolean existsSameNameVariableInMethod(String name, JavaMethod jm)`
メソッド `jm` 内に同じ名前 `name` を持つ変数が存在するかどうか
- `boolean refersToPrivateField(JavaVariable jv)`
フィールド `jv` が他の `private` フィールドを利用しているかどうか
- `boolean isLocallyUsedInMethod(JavaVariable jv)`
変数 `jv` がそのメソッド内で利用されているかどうか

6.3.8 RefactoringVisitor クラス

(1) 機能説明

AST をたどり、指定された変換に基づき AST ノードを変換する。

(2) 属性

- `String text`
リファクタリング対象テキスト
- `ArrayList highlighters`
テキスト中のハイライトの集合

(3) メソッド

- ArrayList getHighlights()
テキスト内のハイライトの集合を取得する
- InsertCodeNode insertCode(Node node, int index, String code)
AST のノードの index 番目にテキスト code を挿入し , 挿入 AST ノードを返す
- InsertCodeNode insertClass(Node node, int index, String code)
AST のノードの index 番目にテキスト code を持つクラスを挿入し , 挿入 AST ノードを返す
- InsertCodeNode insertMethod(Node node, int index, String code)
AST のノードの index 番目にテキスト code を持つメソッドを挿入し , 挿入 AST ノードを返す
- InsertCodeNode insertField(Node node, int index, String code)
AST のノードの index 番目にテキスト code を持つフィールドを挿入し , 挿入 AST ノードを返す
- void deleteClass(Node node)
AST ノード node で始まるクラスを消去する
- void deleteMethod(Node node)
AST ノード node で始まるメソッドを消去する
- void deleteField(Node node)
AST ノード node で始まるフィールドを消去する
- void deleteNode(Node node)
AST ノード node を消去する
- protected void deleteNode(Node node, int index)
AST ノード node の index 番目の AST ノードを消去する
- void setHighlight(Token token)
テキスト中のトークン token をハイライト集合に追加する(表示する)
- void setHighlight(Node node)
AST ノード node に属するテキストをハイライト集合に追加する(表示する)
- void changeModifier(Node node, String oldmod, String newmod)
クラス , メソッド , フィールドの修飾子を oldmod から newmod に変更する

6.3.9 PrintVisitor クラス

(1) 機能説明

変換後の AST に基づき , テキストを出力する .

(2) 属性

- StringBuffer code
変換後のテキストの格納領域

(3) メソッド

- public String getCode(JavaComponent jc)
変換後のテキストを取得する
- String getCode(Node node)
AST ノード node を頂点とする AST からテキストを取得する

6.4 リファクタリング操作部

(1) 機能説明

GUI を介して与えられたリファクタリング要求に対して適切な操作コマンドを生成し、その実行を依頼する。また、リファクタリング操作の取り消しを実行する。

(2) 入力データ

リファクタリング要求 (リファクタリングの種類や名前)

(3) 出力データ

操作コマンド

(4) 構成

本機能は、以下のクラスで構成されている。

- a) Refactor (gui パッケージ)
- b) PositionVisitor (gui パッケージ)

6.4.1 Refactor クラス

(1) 機能説明

変換後の AST に基づき、テキストを出力する。さらに、操作の取り消しを行う。

(2) 属性

- RefactoringCommand commander
リファクタリング操作を実行するコマンド
- int refactoringId
リファクタリング操作の識別番号
- JavaFile jfile
リファクタリング対象ファイル
- JavaComponent javaComp
リファクタリング対象として指定された JavaComponent オブジェクト

(3) メソッド

- public void setJavaComponent(JavaFile jfile, int bl, int bc, int el, int ec)
テキスト選択範囲(開始行 bl, 開始列 bc, 終了行 bc, 終了列 ec)に対応する JavaComponent を設定する。
- boolean isClassRefactoring()
クラスリファクタリングの実行が可能かどうか

- `boolean isMethodRefactoring()`
クラスリファクタリングの実行が可能かどうか
- `boolean isFieldRefactoring()`
クラスリファクタリングの実行が可能かどうか
- `boolean isVariableRefactoring()`
クラスリファクタリングの実行が可能かどうか
- `boolean isSwitchRefactoring()`
クラスリファクタリングの実行が可能かどうか
- `void execute(String command)`
メニュー項目 `command` に対応するリファクタリングコマンドを生成し , コマンドを実行する .
- `String getDestinationText(String name)`
名前 `name` を持つファイルをソースコード編集画面から取得する
- `void updateTextPane(String command)`
リファクタリング結果に応じて , ソースコード編集画面を更新する
- `boolean canUndo(TextPane)`
最前面テキスト `textPane` に対してリファクタリングの取り消しが可能であるかどうか
- `void undo(TextPane)`
最前面テキスト `textPane` に対してリファクタリングの取り消しを実行する
- `boolean confirmUndo(String command, List files)`
リファクタリング操作 `command` の取り消しによって , ファイル `files` が変更されることを利用者に確認する

6.4.2 PositionVisitor クラス

(1) 機能説明

ソースコード上の選択 (カーソルあるいはマウス) 範囲に対応するトークンを探し , 対応する `JavaComponent` オブジェクトを返す .

(2) 属性

- `int beginLine`
選択範囲の開始行
- `int beginColumn`
選択範囲の開始列
- `int endLine`
選択範囲の終了行
- `int endColumn`
選択範囲の終了列

- private ArrayList javaComps
選択範囲に一致する JavaComponent の集合

(3) メソッド

- JavaComponent getTokenAt(JavaFile jfile, int line, int column)
選択位置が点(開始位置と終了位置が等しい)とき , その点(行 line, 列 column)に存在するトークンに対応する JavaComponent を探索する .
- JavaComponent getTokenAt(JavaFile jfile, int bl, int bc, int el, int ec)
選択範囲(開始行 bl, 開始列 bc, 終了行 el, 終了列 ec)に含まれるトークンに対応する JavaComponent 群を探索する
- ArrayList getFoundJavaComponents()
見つかった JavaComponent オブジェクト集合を取得する
- JavaComponent getFoundJavaComponent()
見つかった JavaComponent オブジェクトを取得する
- public SimpleNode getFoundASTNode()
見つかった JavaComponent オブジェクトの AST ノードを取得する

6.5 操作列検索部

(1) 機能説明

直前(1 つあるいは 2 つ前の)リファクタリング操作に応じて , 過去の操作列を検索することで , 利用者が行うリファクタリング手順の計画を支援する . また , その操作を履歴としてライブラリに蓄積する .

(2) 入力データ

リファクタリング操作

(3) 出力データ

履歴ファイル , 検索において一致した操作

(4) 構成

本機能は , 以下のクラスで構成されている .

- a) RefactoringHistory (gui パッケージ)
- b) RefactoringRecord (gui パッケージ)

記録される履歴の基本単位となる RefactoringRecord クラスの内部構造についてはすでに 4.2 に示したので , ここでは省略する .

6.5.1 RefactoringHistory クラス

(1) 機能説明

リファクタリング履歴の管理 , 検索を行う .

(2) 属性

- List records
過去の(ファイルに保存されている)リファクタリング操作の履歴 (記録のリスト)
- LinkedList memories
現在編集集中のソースコードに対するリファクタリング操作の履歴 (記録のリスト)
- HashMap memoriesMap
リファクタリング操作とリファクタリング記録との対応マップ
- List recordsMatched1
1 つ前の記録が一致する過去のリファクタリング操作
- List recordsMatched2
1 つ前および 2 つ前の記録が両方一致する過去のリファクタリング操作

(3) メソッド

- void clear()
過去のリファクタリング履歴を消去する
- List sortDisplayedCommand(Collection collection)
リファクタリング履歴 collection 内の記録を回数の多い順にソートする
- List getRecordsMatchedLastOne()
1 つ直前のリファクタリング操作に対応する記録を取得する
- List getRecordsMatchedLastTwo()
2 つ直前のリファクタリング操作に対応する記録を取得する
- void memorize(TextPane textPane)
ソースコード TextPane に対する履歴をメモリに格納する
- List collectAllSequences()
(ファイルとメモリに) 記録されている履歴を収集する
- public void retrieve(RefactoringRecord record1, RefactoringRecord record2)
1 つ前および 2 つ前の記録が , それぞれ record1, record2 に一致する過去の
リファクタリング操作を検索する
- boolean load()
履歴をファイルから読み込む
- boolean store()
履歴をファイルに書き出す

7. ユーザインタフェース(GUI)部

(1) 機能説明

ソースコードの編集機能を持ち、マウスおよびキーボードイベントを取得する。与えられたイベントに従い、リファクタリング要求や変更目標を取得し、それぞれリファクタリング操作部あるいは操作列検索部に引き渡す。

(2) 入力データ

マウスおよびキーボードイベント(マウスおよびキーボード入力)

(3) 出力データ

リファクタリング要求あるいはリファクタリング目標、
表示ソースコード、ファイル

(4) 制約条件

全角文字を含むソースコードを読み込む場合は、あらかじめ Unicode に変換しておくこと

(5) 構成

各機能におけるクラス間の関係は、図2を参照のこと

7.1 編集機能

(1) 機能説明

ファイルの新規作成・読み込み・書き込み、ソースコードの追加・削除・書き換え・コピー・切り取り・貼り付け機能を持つ。さらに、リファクタリング前後のソースコードを切り替え表示する。また、編集操作の取り消し、やり直し機能を持つ。

(2) 入力データ

マウスおよびキーボードイベント

(3) 出力データ

リファクタリング要求(あるいはリファクタリング目標)
表示ソースコード、ファイル

(4) 構成

本機能は、以下のクラスで構成されている。

- a) TextPane (gui パッケージ)
- b) TextUndoManager (gui パッケージ)

7.1.1 TextPane クラス

(1) 機能説明

ファイルおよびソースコード編集画面を管理する。

(2) 属性

- JTextArea textArea
ソースコードを実際に表示する Java2 のテキストペイン
- String filename
ファイルの名前
- boolean textChanged
ソースコードが変更されているかどうか
- long lastModifiedTime
ソースコードの最終変更時刻
- int selectionStart
テキスト選択範囲の開始点
- int selectionEnd
テキスト選択範囲の終了点

(3) メソッド

- void setFileName(String name)
ファイルの名前を name に設定する
- String getFileName()
ファイルの名前を取得する
- void setText(String text)
ソースコードのテキストを text に設定する
- String getText()
ソースコードのテキストを取得する
- void insert(String text)
ソースコードのカーソルの位置にテキスト text を挿入する
- void setCaretPosition(int pos)
カーソル位置を pos に設定する
- int getCaretPosition()
カーソル位置を取得する
- void setLastModified(long time)
ソースコードの最終更新時刻を time に設定する
- long getLastModified()
ソースコードの最終更新時刻を取得する
- void cut()
テキスト選択範囲をクリップボードに切り取る
- void copy()
テキスト選択範囲をクリップボードにコピーする

- void paste()
カーソル位置にクリップボードのテキストを張り付ける
- int findUpward(String word, int from)
ソースコードの位置 from より後ろのテキストにおいて、最後に存在する文字列 word を検索する

7.1.2 TextUndoManager クラス

(1) 機能説明

ファイルおよびソースコード編集画面を管理する。

(2) 属性

- LinkedList commands
実行後のリファクタリング操作を格納するリスト
- Stack redoCommands
取り消し操作を格納するスタック

(3) メソッド

- void undo()
テキストに対する取り消しを実行する
- void redo()
テキストに対する取り消しのやり直しを実行する
- void append(RefactoringRecord record)
実行したリファクタリング操作をスタックに積む
- RefactoringRecord getLastRefactoringCommand(int num)
num 番目前に実行したリファクタリング操作を取得する
- RefactoringRecord getLastRefactoringCommand()
直前に実行したリファクタリング操作を取得する
- void undoRefactoring()
リファクタリング操作の取り消しを実行する
- LinkedList getRefactoringCommands()
取り消し可能なリファクタリング操作のリストを取得する

7.2 イベント取得機能

(1) 機能説明

マウスイベントおよびキーボードイベントを取得する。与えられたイベントに従い、リファクタリングメニューを構成し、利用者の要求を取得する。

(2) 入力データ

マウスおよびキーボード入力（メニュー操作を含む）

(3) 出力データ

リファクタリング要求(コマンド)あるいはリファクタリング目標

(4) 構成

本機能は、以下のクラスで構成されている。

- a) TabbedTextPane (gui パッケージ)
- b) RefactoringMenu (gui パッケージ)

7.2.1 TabbedTextPane クラス

(1) 機能説明

ファイルおよびソースコード編集画面を管理する。

(2) 属性

- List texts
現在編集集中のソースコードのリスト
- int selectedIndex
最前面ソースコードの番号

(3) メソッド

- TextPane newFile(String name)
名前 name の新規ソースコードを作成する
- TextPane openFile(String name, String content)
名前 name のテキスト text をファイルから読み込む
- public void saveFile(TextPane textPane)
最前面のソースコードを保存する
- void closeFile()
再前面のソースコードをクローズする
- TextPane getTextPane(String name)
編集集中のソースコードから名前 name を持つソースコードを取得する
- int getOpenFileNum()
現在編集集中のソースコードの数を取得する
- List getTexts()
現在編集集中のソースコードのリストを取得する
- TextPane getCurrentTextPane()
最前面のソースコードを取得する
- void mousePressed(MouseEvent evt)
マウスが押された時の処理を実行する
- public void mouseReleased (MouseEvent evt)
マウスが放された時の処理を実行する
- showPopup(MouseEvent evt)

編集画面上にポップメニューを表示する

7.2.2 RefactoringMenu クラス

(1) 機能説明

リファクタリング操作に関するメニュー項目を構築する .

(2) 属性

なし

(3) メソッド

- JMenu initClassesMenu()
クラスリファクタリングメニューを作成する
- JMenu initMethodsMenu()
メソッドリファクタリングメニューを作成する
- JMenu initFieldsMenu()
フィールドリファクタリングメニューを作成する
- JMenu initVariablesMenu()
ローカル変数リファクタリングメニューを作成する
- JMenu initMiscellaneousMenu()
その他リファクタリングメニューを作成する
- void initGuideMenu()
ガイドメニューを作成する
- void buildGuideMenu(TextPane textPane)
ソースコード textPane に対する履歴の検索結果をメニューに反映させる
- void menuSelected(MenuEvent evt)
マウスが押されたときのメニューの構築を行う
- public void setRefactoringMenu(Point point)
リファクタリング操作の可能・不可能に応じてメニューの設定し , 位置 point に
ポップアップメニューを表示する

8 . 性能

本ソフトウェアは，Java で記述されているため，実行速度およびメモリ使用量の点で高性能であるとはいえない．今後，コードの最適化などの改善が必要である．現時点では，できるだけ高速の CPU および多くのメモリの使用を奨励する．

9 . 運用方法

本ソフトウェアに関する情報および最新版は，インターネットの以下のアドレスから取得できるようになっている．また，以下のメールアドレスで質問，意見を受け付ける．

<http://refactoring.fse.cs.ritsumei.ac.jp>

refactoring@fse.cs.ritsumei.ac.jp