

Irenka Studio Users' Guide

For Eclipse JDE

Ver. 0.1.0

2007/11/11

The Ashikunep Kotan

もくじ

第 1 章 Irenka	1
第 1 節 Irenkaの目的	1
第 2 節 Irenka Studio	1
第 2 章 Irenka Studioのインストール	3
第 1 節 Eclipse JDEのインストール	3
第 2 節 Irenka Studioプラグインのインストール	3
第 3 章 Irenka Builder	10
第 1 節 簡単な使い方	10
第 2 節 Irenka Builderの設定	22
第 3 節 利用するHackの選択	24
第 4 節 Hackの詳細設定	25
第 4 章 Irenka Search	27
第 1 節 簡単な使い方	27
第 2 節 Irenka Search Queryの概要	35
第 1 項 プレースホルダへの制約	35
第 2 項 即値	36
第 3 項 プロパティ	38
第 4 項 リスト	39
第 3 節 簡単なサンプル	39
第 5 章 Hack開発の流れ	41
第 1 節 Irenka StudioでのHack	41
第 2 節 Hackの作成	43
第 3 節 Hackの検証	49
第 6 章 Hack	59
第 1 節 Hack Actionの起動	60

第 2 節 Search Queryの記述	60
第 3 節 Hack Parameterの記述	62
第 4 節 Irenka DOMの操作	64
第 5 節 Toolオブジェクトの利用	66
第 1 項 Irenka DOM要素を生成するTool	66
第 2 項 環境を操作するTool	68
第 3 項 コンパイラの機能を提供するTool	69
第 6 節 イベントオブジェクトの利用	70
第 1 項 イベントの発生とHack Action実行のタイミング	71
第 2 項 ビルドの実行に関するイベント	72
第 3 項 Hackの実行に関するイベント	72
第 7 章 Hack Packager	73
第 1 節 Hack Libraryの作成	73
第 2 節 Hack Libraryのしくみ	77
第 1 項 Hack Libraryの構造	77
第 2 項 Hack定義ファイルの構造	78
第 3 項 追加リソースファイルの扱い	79
第 8 章 その他の情報	80
第 1 節 関連ドキュメント	80
第 2 節 サンプル集	80
第 3 節 CARNIVAL	80
第 9 章 謝辞	81

第1章 Irenka

第1節 Irenka の目的

IrenkaはJava™による開発プロジェクトを管理するための環境で、プロジェクト内に含まれるプログラムを監視して、状況を通知やプログラムの改変などを行います。このようなプロジェクトを監視するためのプログラムをHackと呼び、ユーザが自由に作成してIrenkaにいくつも登録することができます。そして、これらの登録されたHackはIrenkaがプロジェクトをビルドする際などに実行され、自動的にプロジェクトを管理することができます。また、作成したHackはHack Libraryとしてまとめて配布することができ、複数の開発プロジェクトで共用することもできます。

本文書では、Irenka を利用した開発環境である Irenka Studio の使い方と、Irenka Studio を利用して Hack をプロジェクトに適用する方法、また実際に新しい Hack を作成して配布する方法などを紹介します。Hack の作成時に使用する Irenka Search Query や Irenka DOM について詳しくはそれぞれの仕様書を参照してください。

第2節 Irenka Studio

Irenka StudioはIrenkaをEclipse¹ Java Development Environment (JDE)上で利用するためのEclipse plug-inで、次のような機能を持っています。

- Irenka Builder

配布された Hack をプロジェクトに適用するコンパイラです。Eclipse のプロジェクトビルドプロセスに常駐し、プロジェクトの内容が変更されるたびに登録された Hack をプロジェクトに適用します。

- Hack Launcher

この項目は未実装です。

配布された Hack や作成した Hack をプログラムとして起動するツールです。Irenka Builder で実行するには複雑すぎる Hack や、一度だけ実行されるような Hack を実行する際に便利です。

¹ <http://www.eclipse.org/>

- Irenka Search

Irenka が提供するプログラム検出言語の Irenka Search Query を利用して、プロジェクト内のソースプログラムを検索するツールです。Eclipse JDE 標準の Java Search よりも複雑な条件を書くことができ、さまざまな条件を満たすプログラムを容易に検出することができます。

- Hack Packager

作成した Hack を Hack Library として配布可能な形式に変換するツールです。作成した Hack Library は Irenka Builder に登録することによって、他のプロジェクトからも作成した Hack が利用可能になります。

第2章 Irenka Studio のインストール

第1節 Eclipse JDE のインストール

Irenka Studio は Eclipse IDE の Java 開発環境(Eclipse JDE)を拡張するプラグインで、利用には Eclipse JDE が必要です。また、現在の Irenka Studio は Java Software Development Kit 5.0 (JDK 5.0), Eclipse 3.3 (Europa)以上を対象に設計されています。

Eclipse は公式サイト(<http://www.eclipse.org/>)からダウンロード可能です。

第2節 Irenka Studio プラグインのインストール

Irenka Studio は Eclipse の Software Update 機能を利用して簡単にインストールすることができます。Irenka Studio のアップデートサイトは <http://irenka.ashikunep.org/eclipse> です。

Eclipseを起動し、メニューバーのHelpからSoftware Updates > Find and Install...の順番に選択します(図 1)。

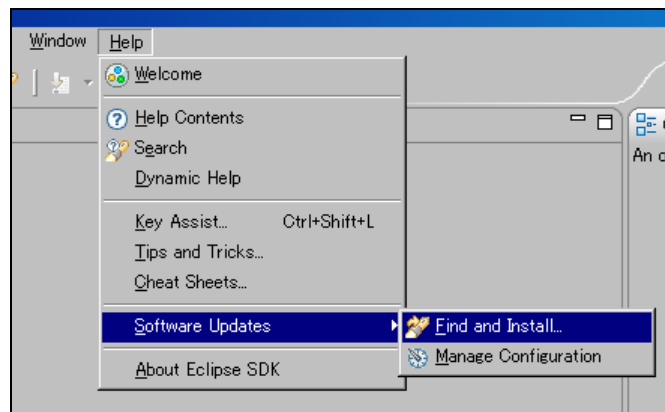


図 1. Install/Update Wizard の起動

UpdateまたはInstallを選択する画面が開きますので、Search for new features to installのラジオボタンを選択後、Nextボタンを押下します (図 2)。

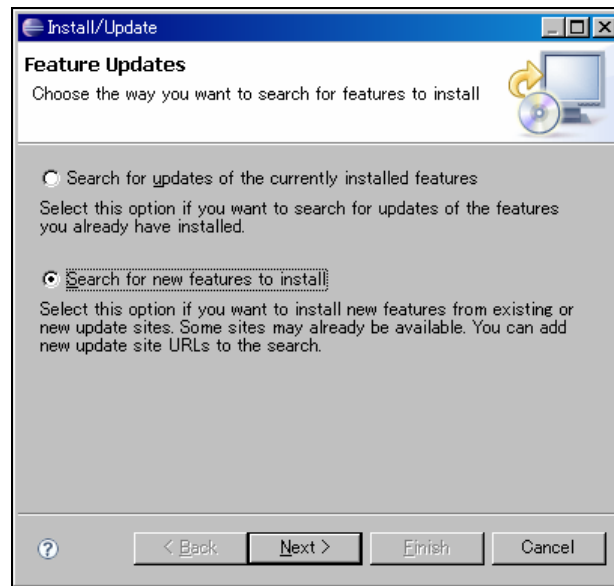


図 2. Install/Update 選択の画面

アップデートサイトの一覧が表示されます。ここではまだIrenka Studioのアップデートサイトが追加されていないので、New Remote Site...ボタンを押下してIrenka Studioの配布サイトを追加します(図 3)。

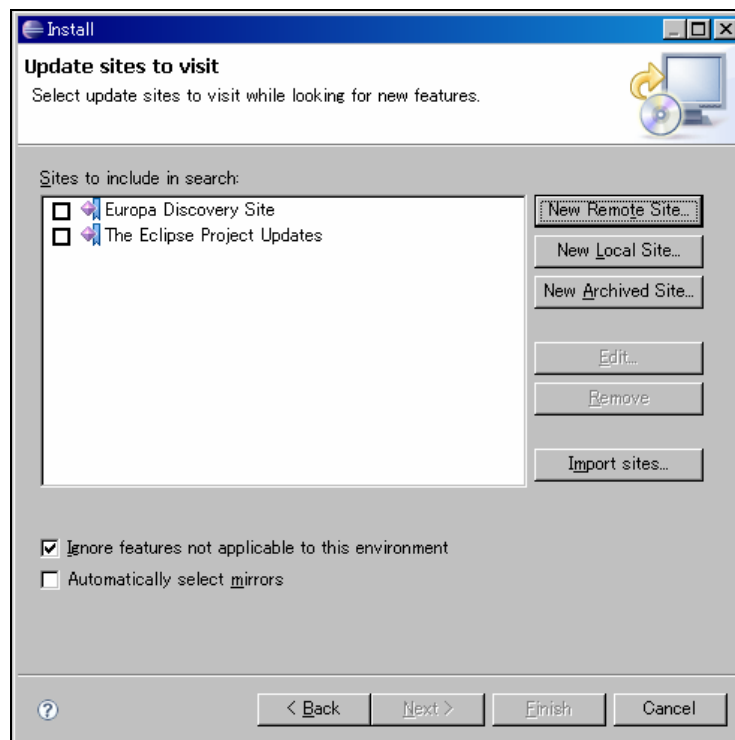


図 3. Update site の一覧画面

New Remote Site...ボタンを押下すると、Update Siteの名前とURLの入力ダイアログが表示されます。Name:に"Irenka Studio"、URL:に"http://irenka.ashikunep.org/eclipse"と入力し、OKボタンを押下します(図 4)。

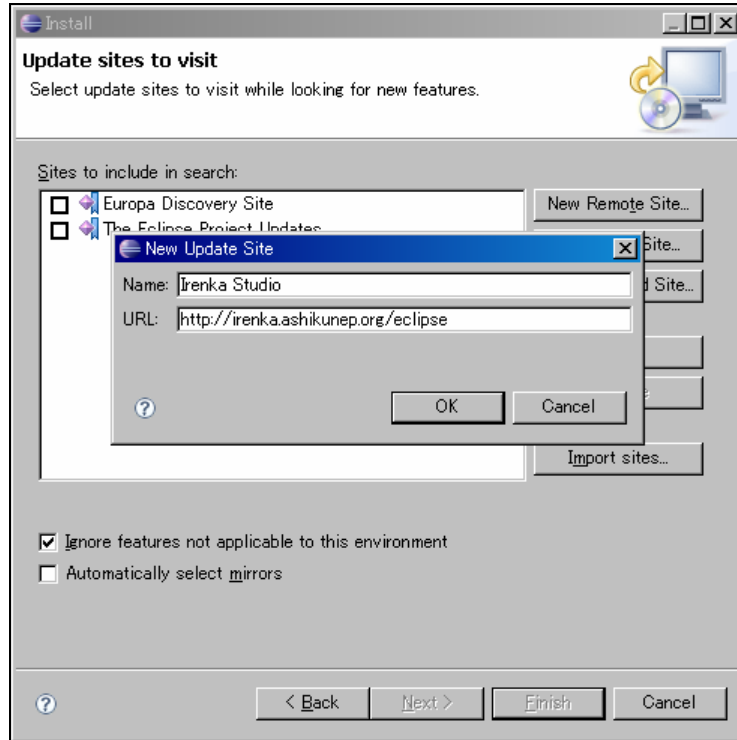


図 4. Remote Site の追加ダイアログ

Update siteの追加に成功すると、リスト上にIrenka Studioという項目が追加されます。Irenka Studioのチェックボックスにチェックを入れ、Finishボタンを押下します(図 5)。

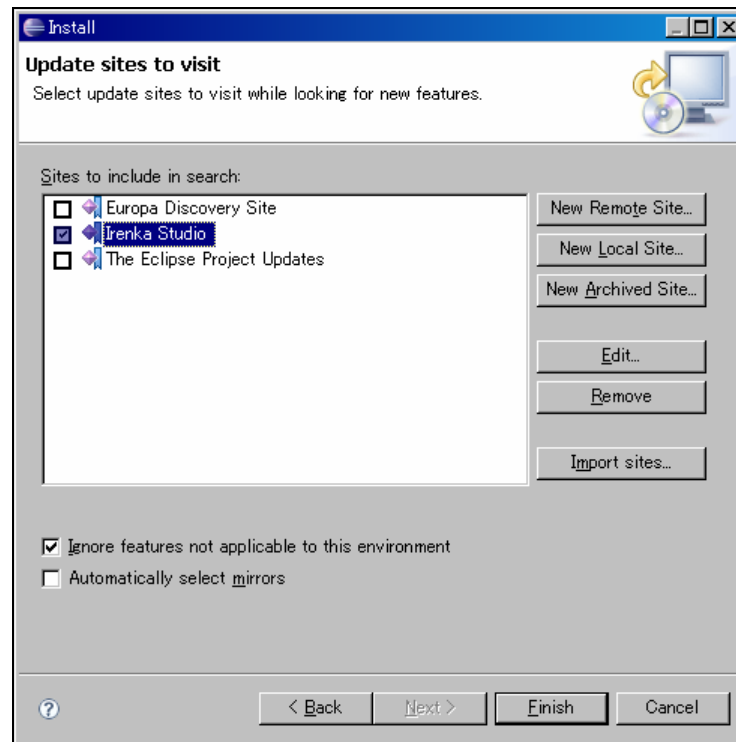


図 5. Irenka Studio 配布サイトの選択

配布サイトとの接続に成功すると、インストールするフィーチャーの一覧が表示されます。Irenka Studioを選択し、Nextボタンを押下します(図 6)。

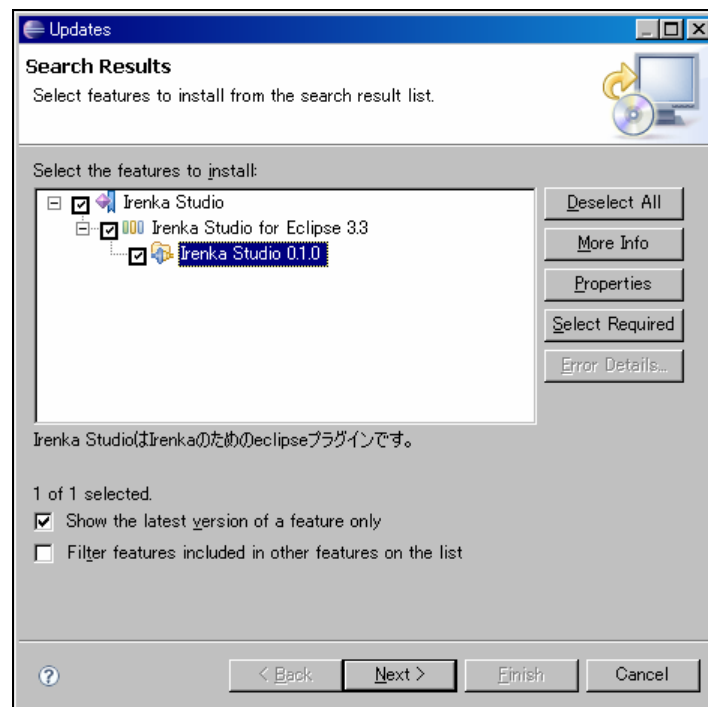


図 6. Irenka Studio プラグインの選択

インストールの前に、Irenka Studioのライセンスについて表示されます。問題がなければ"I accept the terms in the license agreement"を選択後、Nextボタンを押下します(図 7)。

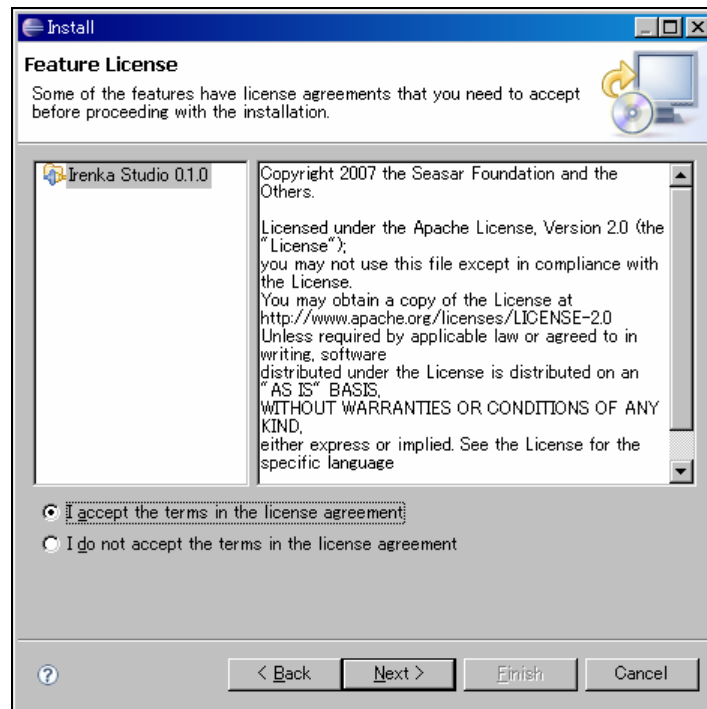


図 7. ライセンスの確認

次の画面でインストール先を設定できます。設定を完了後にFinishボタンを押下します(図 8)。

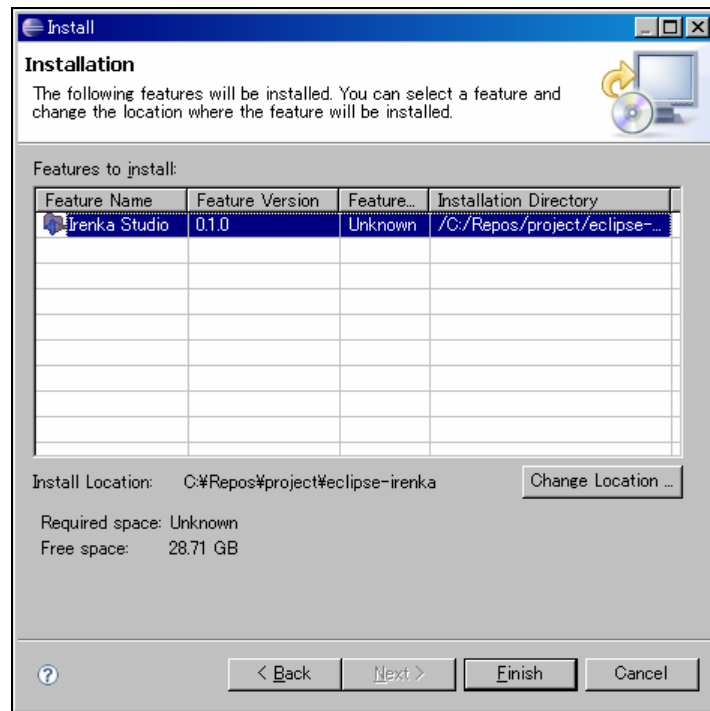


図 8. インストール先の選択

Irenka Studioはデジタル署名を添付していないため、警告画面が表示されてしまいます。いずれ準備する予定ですので、現在はそのままInstallボタンを押下してください(図 9)。

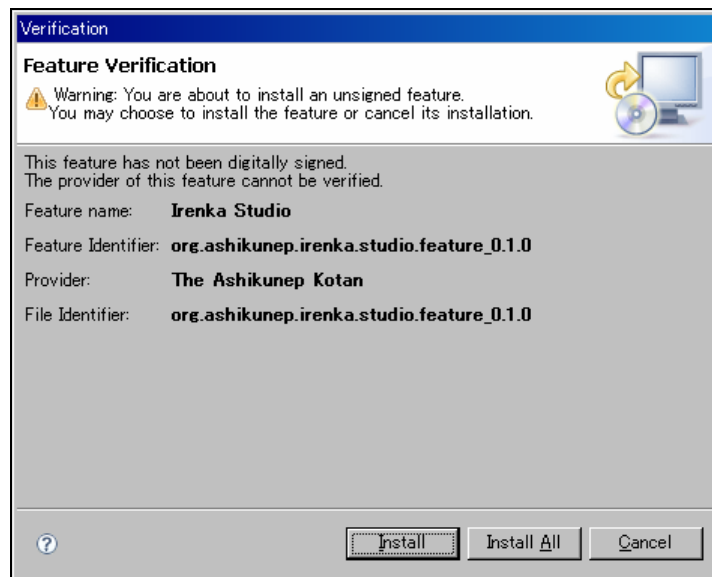


図 9. デジタル署名の確認

インストールに成功すると、Eclipse を再起動するダイアログが表示されます。Yes を選択し、Eclipse を再起動してください。

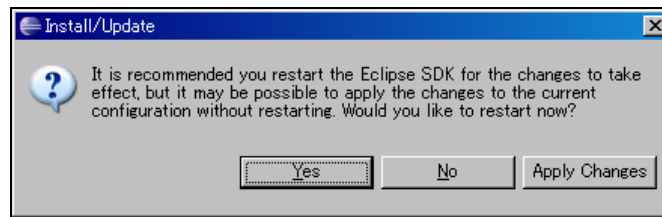


図 10. Eclipse の再起動

以上で Eclipse に Irenka Studio が導入されました。

第3章 Irenka Builder

Irenka Builder は、Eclipse のプロジェクトビルドプロセスに常駐し、プロジェクトの内容が変更されるたびに登録された Hack をプロジェクトに適用します。Hack を透過的にプロジェクトに適用できるためルールのチェックなどに向いていますが、自動的にプロジェクトに Hack が適用されてしまうため、大規模なフレームワークを生成するなどの複雑な処理を行う Hack の適用には向いていません。

この章では、Irenka Builder の利用方法を紹介します。

第1節 簡単な使い方

ここでは、Irenka Builder を利用して他人が公開する Hack をプロジェクトに適用する例を紹介します。サンプルとして、プロジェクト内の main メソッドを検出する Hack Library を <http://ireнка.ashikunep.org/example/main-detector.jar> に公開しています。これを好きな場所にダウンロードしてください。

main-detector.jar に含まれる Hack "org.ashikunep.ireнка.example.MainDetector"は、プロジェクト内に含まれる main メソッドを検出し、"main!"という情報を出力するだけの単純な機能を持ちます。Hack Library はいくつもの Hack を含めることができますが、このサンプルでは MainDetector のみからなるライブラリとして配布しています。

まず、Eclipseを起動し、新しいJava Projectを作成します。メニューバーの FileからNew > Java Project の順に選択します(図 11)。

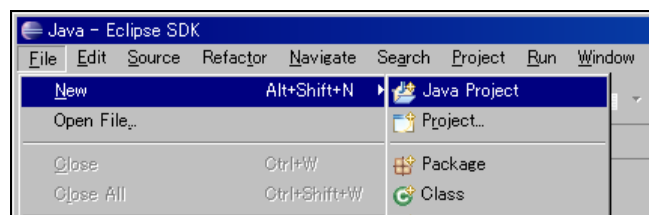


図 11. Java Project の作成

New Java Project Wizardが起動されますので、ここでは"start"という名前のJava Projectをワークスペース上に作成します(図 12)。JREは 5.0 (JDK 5.0)以降であれば何を利用しててもかまいません。

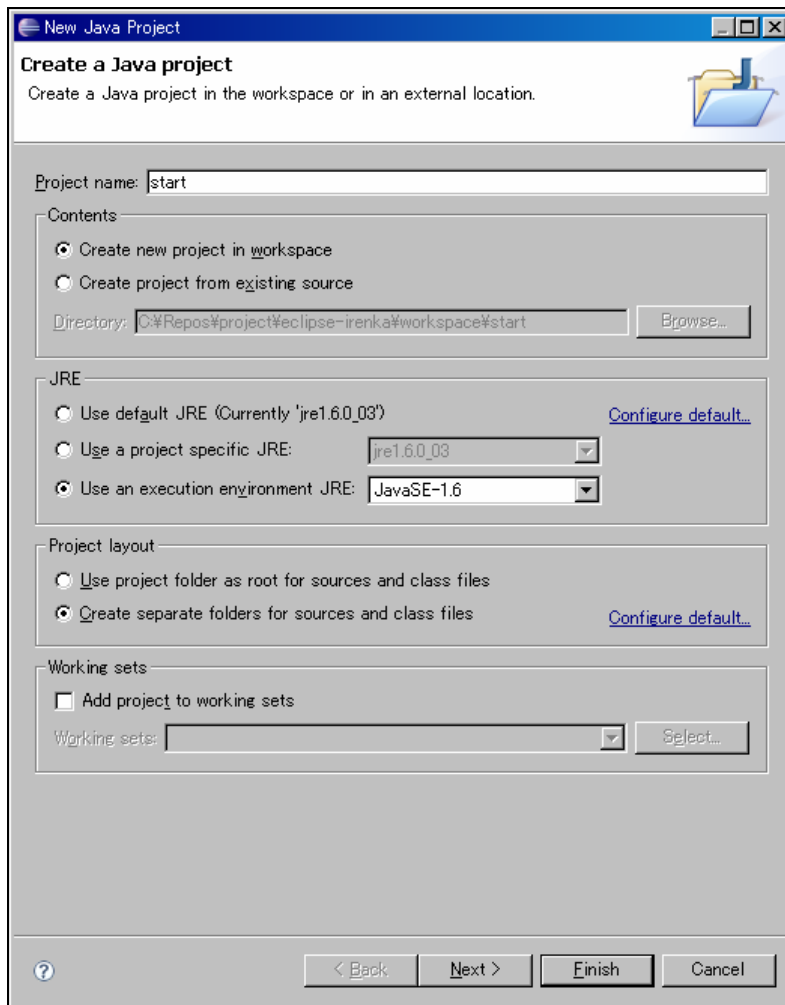


図 12. Java Project 作成ウィザード

プロジェクトを作成したら、プロジェクト内にHack Libraryを保存しておくためのフォルダを作成します。プロジェクトを右クリックし、New > Folderと選択します(図 13)。この操作は必須ではありません。

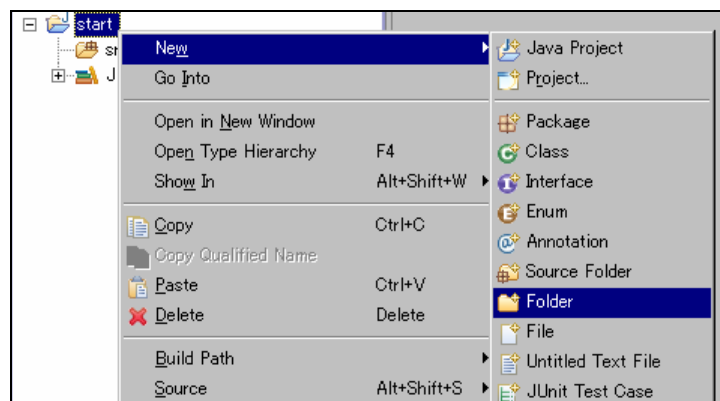


図 13. Hack Library を配置するフォルダの作成

Hack Libraryをまとめて保管するフォルダを作成します。特に制約はありませんが、わかりやすいようにhackという名前を付けてFinishボタンを押下します(図 14)。



図 14. フォルダの作成ダイアログ

作成したフォルダに、ダウンロードしてきたmain-detector.jar²をコピーします(図 15)。

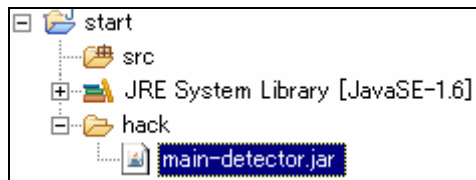


図 15. サンプル Hack Library の配置

次に、プロジェクトにHackを適用させるための設定を行います。プロジェクトを右クリックし、Propertiesを選択します(図 16)。

² <http://irenka.ashikunep.org/example/main-detector.jar>

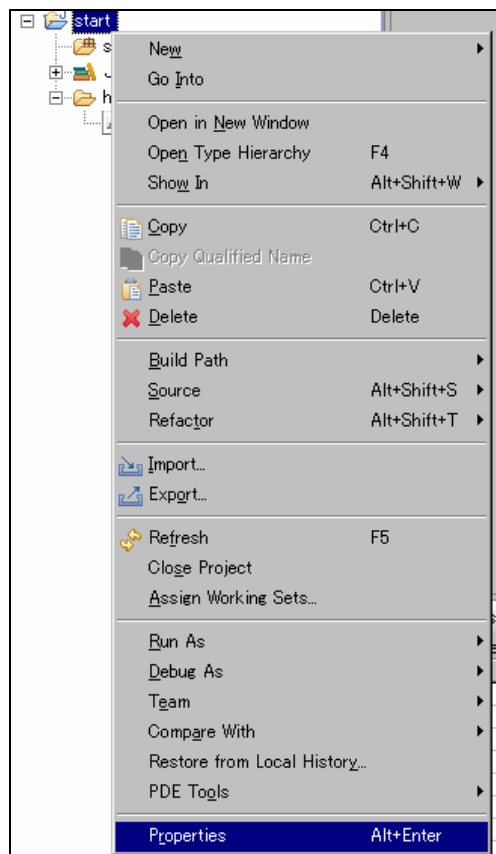


図 16. Java Project プロパティを開く

プロジェクトのプロパティウィンドウが表示されますので、左側のペインからJava Build Pathを選択し、右側のペインでLibrariesタブを開きます(図 17)。

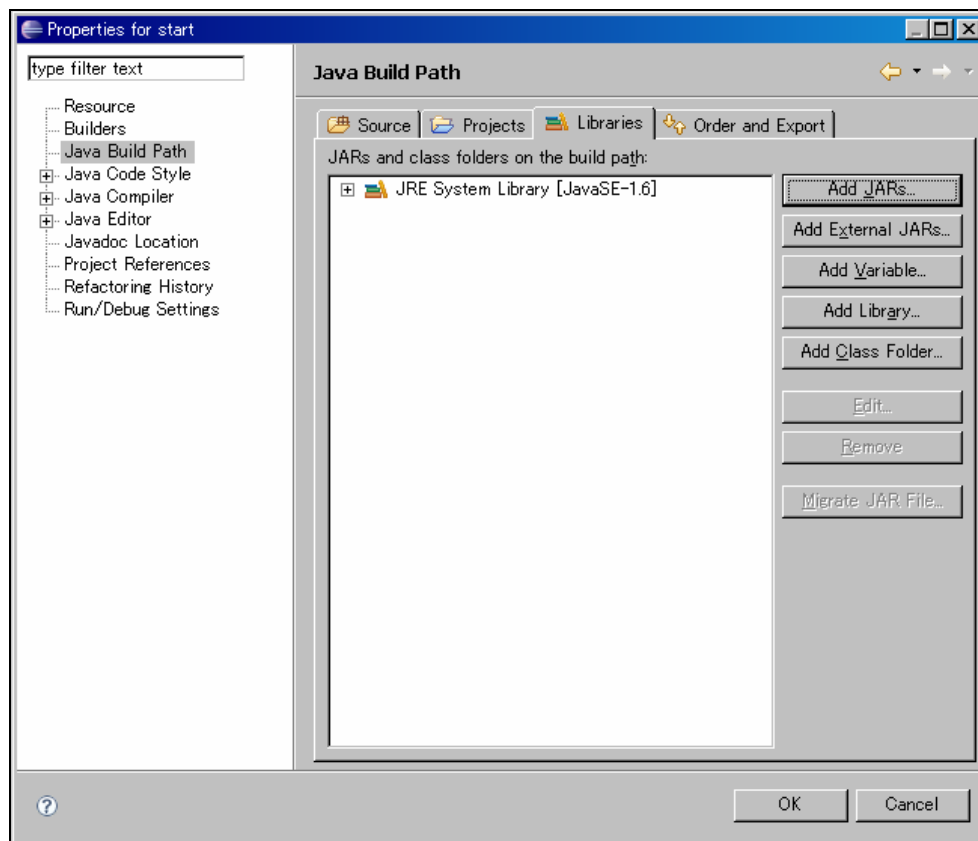


図 17. Build path ヘライブラリの登録

図 17 の右側にあるAdd JARs...ボタンを押下すると、追加するJARファイルの選択画面が表示されます。ここでは、先ほどプロジェクト上にコピーしたhack/main-detector.jarを選択し、OKボタンを押下します(図 18)。

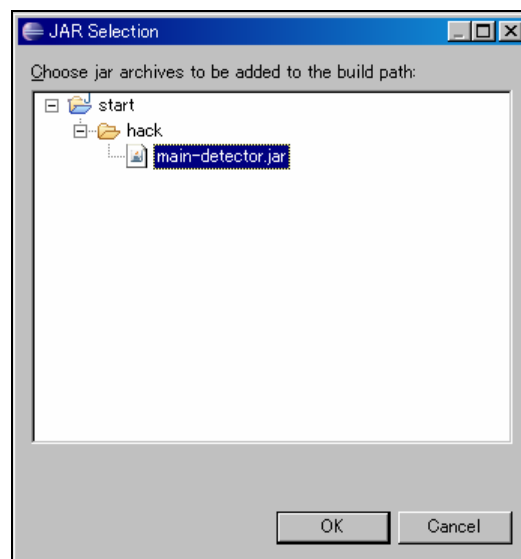


図 18. 追加する JAR ファイルの選択

成功すると、"main-detector.jar - start/hack"という表示が追加されます(図 19)。

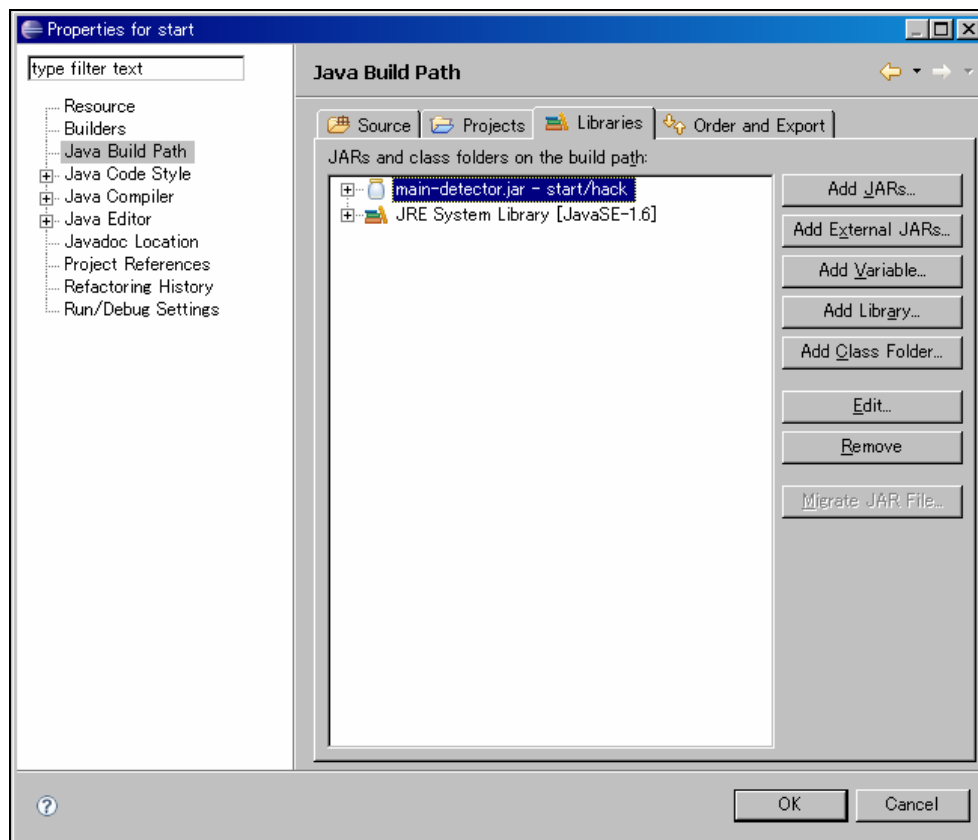


図 19. Hack Library がビルドパスに追加された状態

次に、Irenka Builder の設定を行います。Irenka Builder はプロジェクトのビルドパス上に配置された JAR ファイルの中から Hack Library であるものを自動的に検出します。Irenka Builder の設定画面では、検出した Hack Library の中から実際に適用したい Hack の一覧をユーザが選択して設定することができます。

まず、プロジェクトプロパティ画面の左ペインから、Java Compiler > Irenkaを選択します。このとき、Java Build Pathの設定直後であった場合には、図 20のようなダイアログが表示されますので、Applyを選択してプロジェクトのクラスパスを確定させてください。

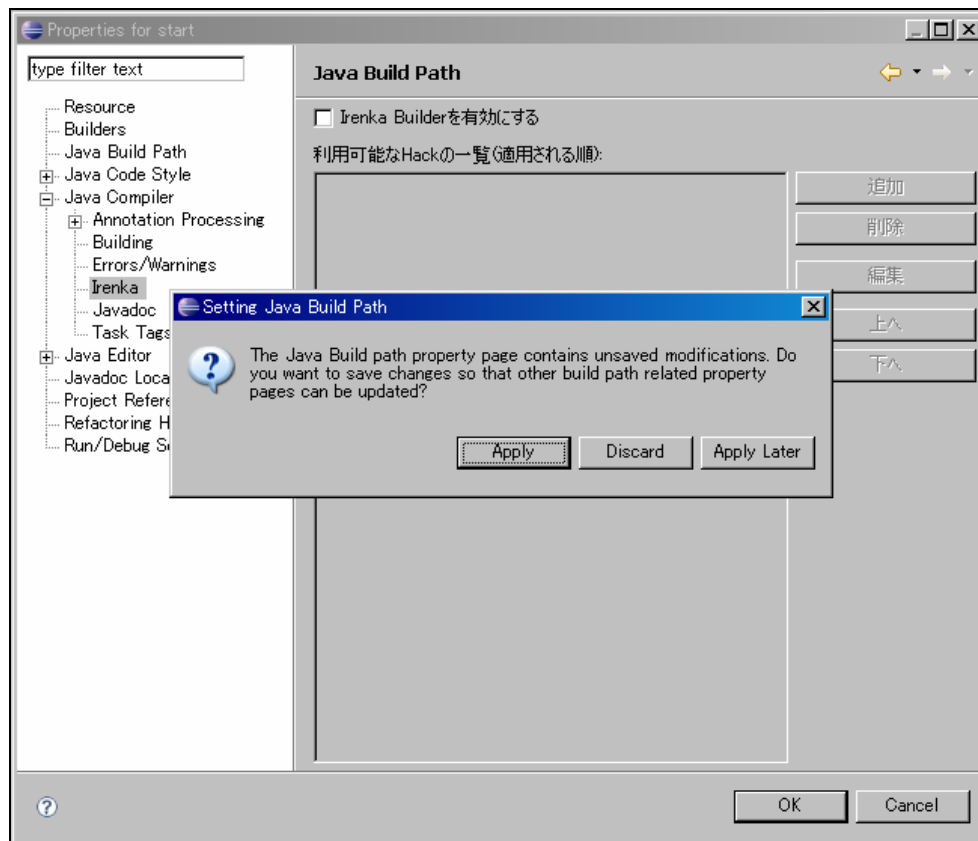


図 20. Irenka Builder の設定/Java build path の保存

Java Compiler > Irenkaのページ上部にある"Irenka Builderを有効にする"というチェックボックスをクリックします(図 21)。これによって、プロジェクトにIrenka Builderが登録されます。

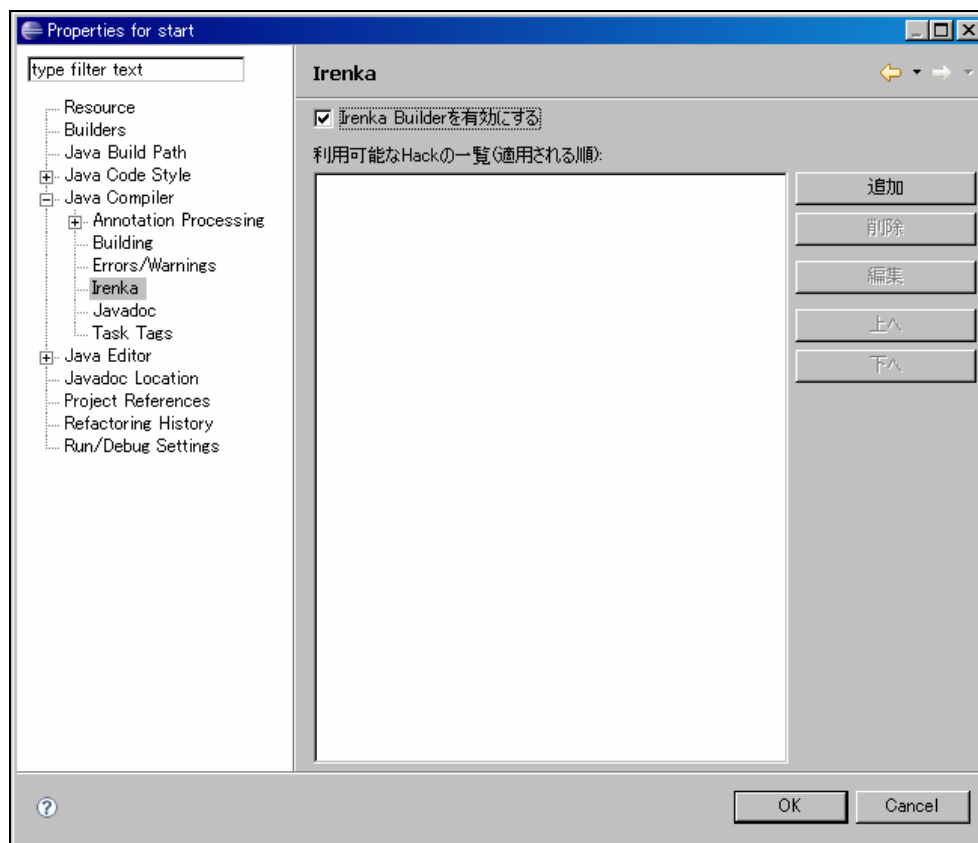


図 21. Irenka Builder を有効にする

図 21の右にある追加ボタンを押下すると、利用可能なHackの一覧がダイアログ上に表示されます。今回は"main-detector.jar"に"org.ashikunep.irenka.example.MainDetector"というHackが登録されていますので、それを選択してOKボタンを押下します(図 22)。

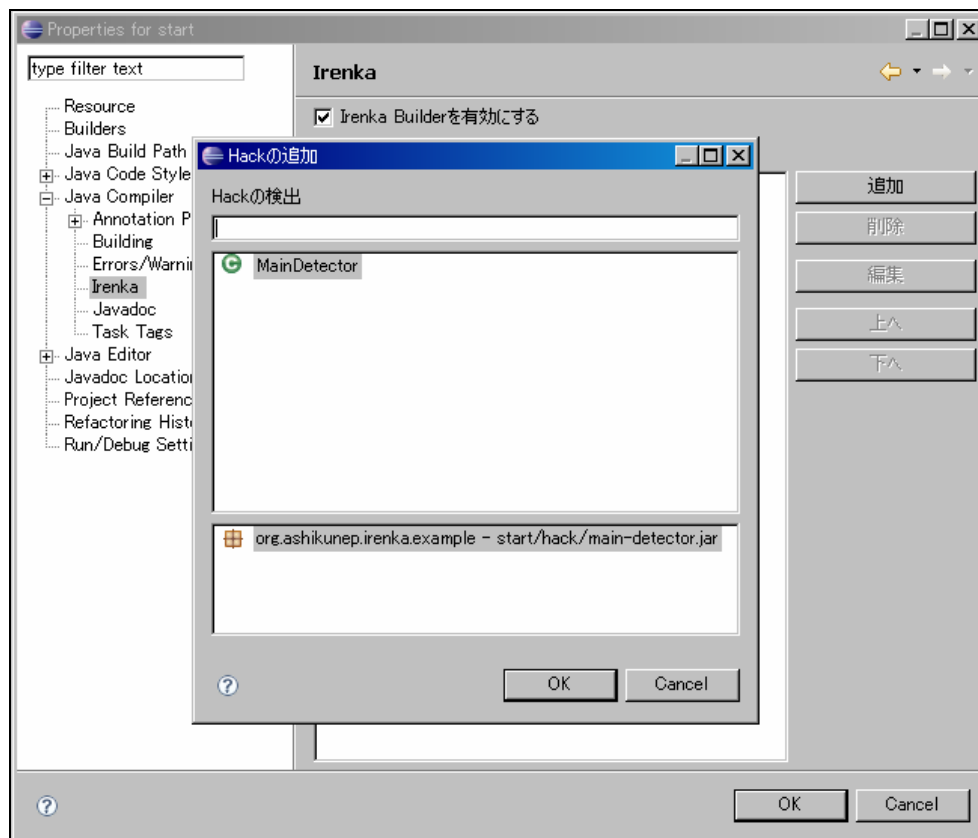


図 22. Hack Library から Hack を追加する

利用可能なHackの一覧にMainDetectorが追加されました。利用可能なHackは今回これだけですので、OKボタンを押下してIrenka Builderの設定を確定します(図 23)。

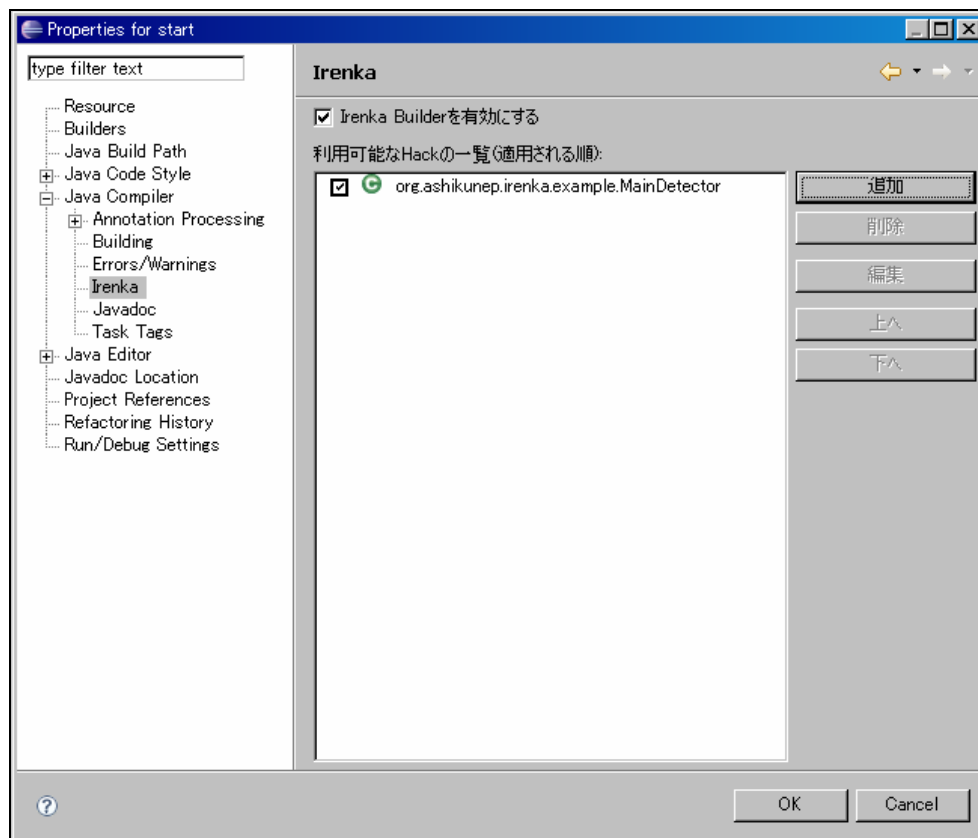


図 23. Hack: MainDetector が追加された状態

Irenka Builderを有効にしたプロジェクトには、Irenka LibrariesというIrenkaの基本クラスライブラリが自動的に追加されます。これはHackをプロジェクトに適用する際に必要です(図 24)。

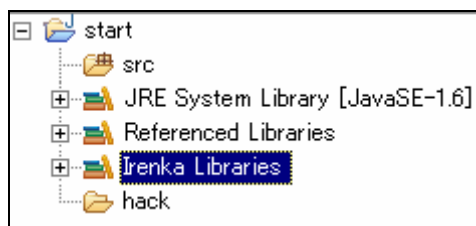


図 24. Irenka Libraries が追加された状態

次に、Irenka Builderに登録したHackを実際に検証してみます。MainDetectorはクラス内のmainメソッドを検出するHackですので、まずは新しくクラスを作成します。ソースフォルダの右クリックからNew > Classを選択します(図 25)。

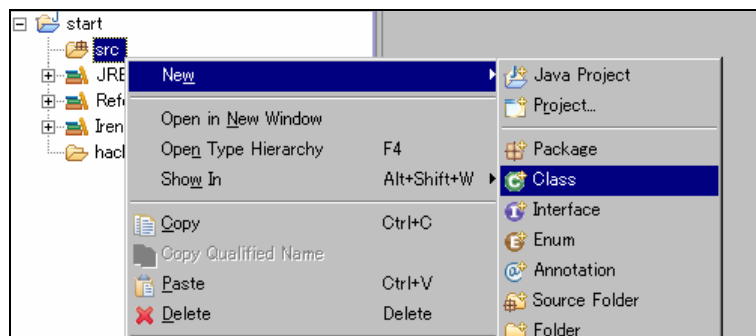


図 25. テスト対象クラスの追加

New Java Class Wizardが起動されますので、テスト用にクラスを作成してみます。ここでは、com.exampleパッケージにMainという名前の空のクラスを作成しています(図 26)。

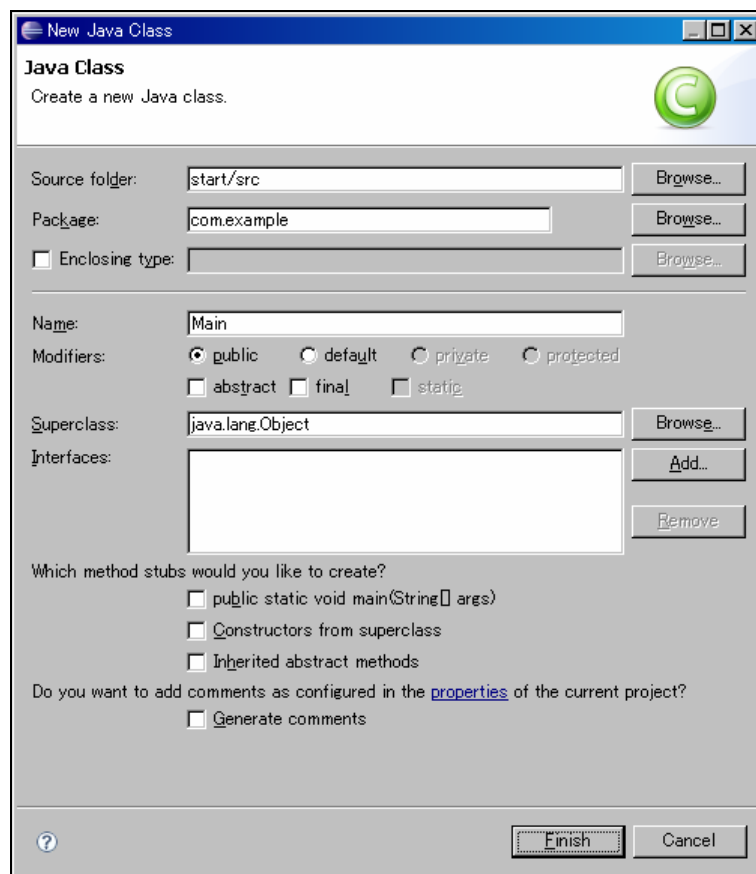


図 26. com.example.Main クラスの作成

空のクラスが作成されました。このクラスにはmainメソッドが存在していないため、MainDetectorは何も行いません(図 27)。

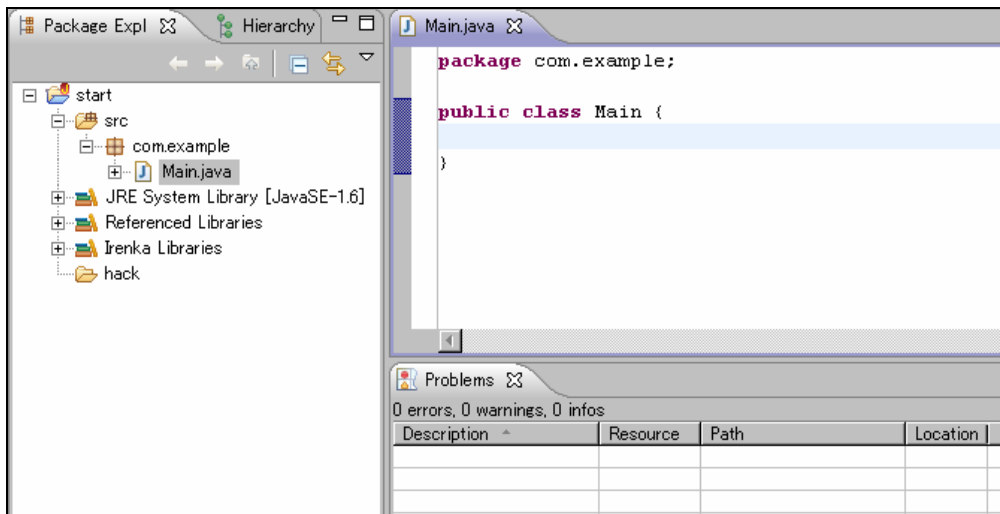


図 27. 空の Main クラスが生成された状態

簡単なmainメソッド(`public static void main(String[])`)を作成します。保存するまで Irenka Builderは実行されないため、ここでもまだ何も起こりません(図 28)。

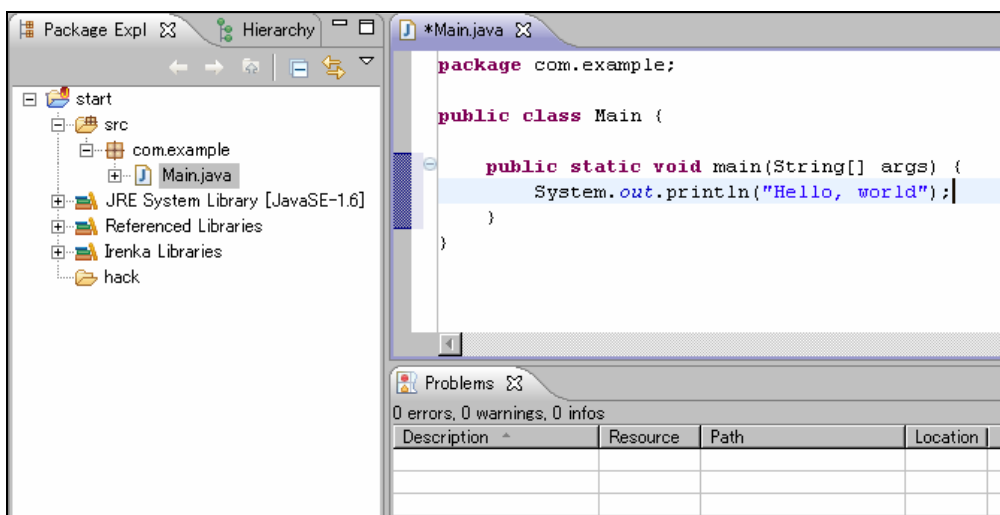


図 28. Main クラスに main メソッドを追加した状態(未保存)

図 28の状態ですらCtrl+Sを押下すると、作成したプログラムが保存されてIrenka Builderのコンパイル対象となります。コンパイル終了後にProblemsタブを見ると、Infosカテゴリに"main!"と表示された項目が追加されています。これはHack MainDetectorがmainメソッドを発見した際に行うアクションで、Irenka Builderによってビルド時にHackが適用されていることが確認できます(図 29)。

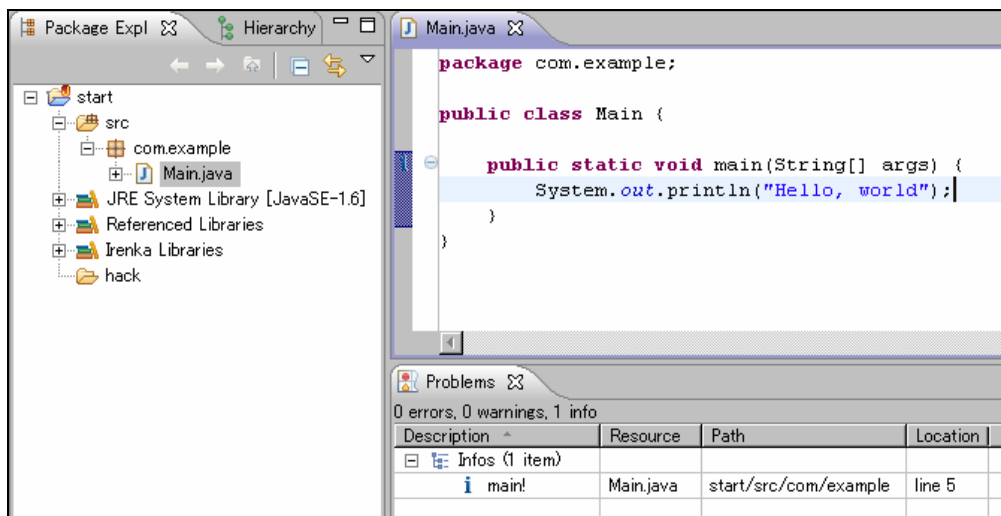


図 29. Irenka Builder によって Hack が適用された状態

このように、配布されている Hack Library をダウンロードし、次の 3 ステップでプロジェクトに Hack を適用することができます。

1. Hack Library をダウンロードする
2. Java Build Path に Hack Library を追加する
3. Irenka Builder に Hack を追加する

他にも簡単な Hack のサンプルを <http://irenka.ashikunep.org/example> に用意しています。

第2節 Irenka Builder の設定

Irenka Builder の設定は Java Project の Properties 内、Java Compiler > Irenka というページから行うことができます。

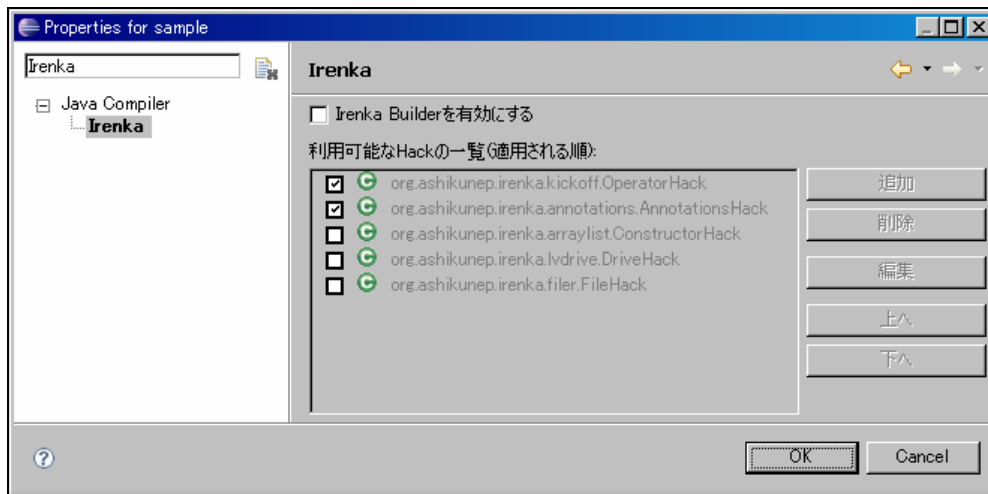


図 30. Irenka Builder の設定画面(Irenka Builder 無効)

図 30の上部に配置された"Irenka Builderを有効にする"という項目をチェックすると、対象のプロジェクトに対して次の操作が行われます。

- Irenka Nature の登録

Irenka Nature は対象のプロジェクトを Irenka Studio が管理していることを表すマークのようなものです。Irenka Nature 自体がアクションを実行することはありませんが、後述の Irenka Builder や Irenka Libraries を利用する際に必要となります。

- Irenka Builder の登録

Irenka Builder は対象プロジェクト内でビルドイベントが発生する際(リソースの変更時やクリーンビルドの実行時)に実行されるプログラムで、登録された Hack をプロジェクトに適用させる機能を持ちます。

- Irenka Libraries の登録

Irenka Libraries は各 Hack が利用するインターフェース等を定義したクラスライブラリ群です。Irenka Builder が Hack をプロジェクトに適用する際、または Hack を開発する際などに利用されます。

"Irenka Builder を有効にする"という項目のチェックを除去すると、上記の 3 つはプロジェクト上から削除されます。Hack を利用する際、および開発する際のどちらにもこれらが有効になっている必要があります。

Irenka Builder を有効にしたら、Irenka Builder に Hack を登録することができます。Irenka Builder に登録された Hack は、対象のプロジェクトのビルド時に実行され、さまざまな操作をプロジェクトに対して適用することができます。

第3節 利用する Hack の選択

Irenka Builderを有効にすると、Irenka Builderの設定画面(第2節)上で、Irenka Builderがプロジェクトに対して適用するHackの一覧を設定することができます。

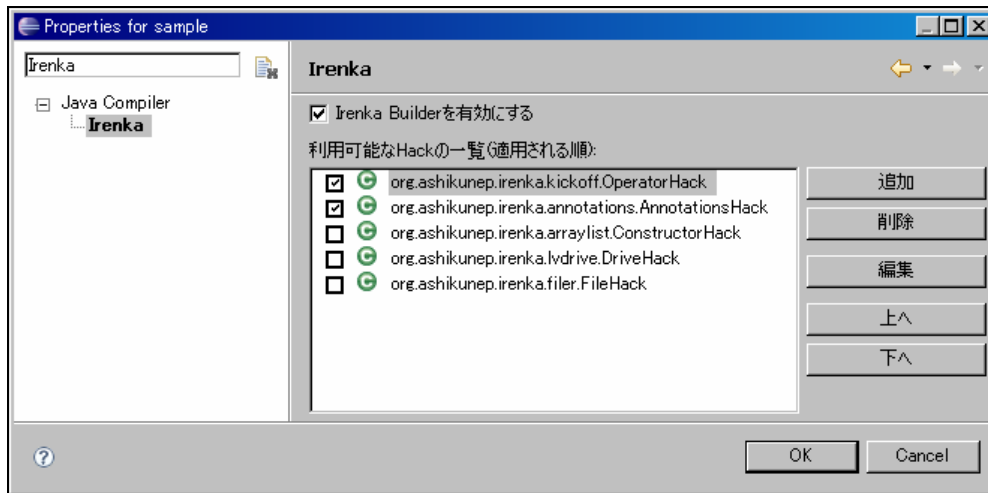


図 31. Irenka Builder の設定画面(Irenka Builder 有効)

図 31の中央部に配置されたペインは、現在Irenka Builderに登録されたHackの一覧が表示されます。

1. チェックボックス(Hack 一覧)

各 Hack の左に配置されたチェックボックスは、対象の Hack の有効/無効を切り替えることができます。無効になった Hack は Irenka Builder の実行時に無視されます。

2. アイコン(Hack 一覧)

現在利用可能であるHackはクラスを表すアイコンで表示され、利用不可能であるHackにはアイコンが表示されません³。利用不可能であるHackはIrenka Builderの実行時に無視されます。

3. 追加ボタン

Irenka Builder に新しい Hack を追加します。このとき、Irenka Studio は対象プロジェクト上に存在する Hack を検出し、有効な Hack の一覧を表示します。

4. 削除ボタン

³ 特別なアイコンを用意する予定です。

Irenka Builder に追加された Hack を削除します。削除した Hack はリスト上から消滅し、再度利用するには追加ボタンで登録しなおす必要があります。

5. 編集ボタン

各Hackの詳細設定を行います(第 4 節)。

6. 上へボタン

選択した Hack をリストの一つ上へ移動します。Irenka Builder は各 Hack をリストの上から順に適用します。依存関係のある Hack などは適切にソートして利用する必要があります。

7. 下へボタン

選択した Hack をリストの一つ下へ移動します。

第4節 Hack の詳細設定

この項目は未実装です。

図 31にある編集ボタンを押下すると、現在選択中のHackに対して細かい設定を行うことができます。

1. 名前

Hack の名前が表示されます。自プロジェクト上のソースコードで作成された Hack の場合、このフィールドを設定することによって Hack にわかりやすい名前を与えることができます。Hack Library などに含まれる Hack では、この項目は編集できません。

2. 解説

Hack の解説が表示されます。自プロジェクト上のソースコードで作成された Hack の場合、このフィールドを設定することによって Hack に対する解説を与えることができます。Hack Library などに含まれる Hack では、この項目は編集できません。

3. プロパティ

外部から設定可能な Hack のプロパティ一覧と、割り当てられた値が表示されます。テーブル上にある"値"の項目は自由に編集することができ、"型"で指定された値を与えることができます。

Hack がプロパティを公開しない場合、このテーブルは編集できません。

第4章 Irenka Search

Irenka Search は Irenka が提供する Search Query というプログラム検索言語を利用してプロジェクト内のソースコードを検索するツールです。Eclipse JDE 標準の Java Search よりも複雑な条件を書くことができ、さまざまな条件を満たすプログラムを容易に検出することができます。また、Irenka Studio では Eclipse が提供する検索機構を拡張して Irenka Search が作成されているため、IDE と連携した様々な機能を利用することができます。

この章では、Irenka Search の利用法について紹介します。

第1節 簡単な使い方

ここでは、Irenka Search を利用してプログラム内の細かい構造を検出する方法を紹介します。Irenka Search の完全な利用には Irenka Search Query (第 2 節) の習得が必要ですが、いくつかパターンを作って再利用することもできます。

まず、検索対象のクラスとそのためのプロジェクトを新規に作成します。メニューバーの File から New > Java Project の順に選択します(図 32)。

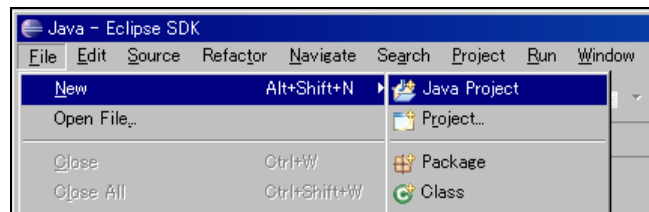


図 32. 検索対象プロジェクトの作成

New Java Project Wizardが開きますので、プロジェクト名に"search"(名前は自由)と入力してFinishボタンを押下します(図 33)。

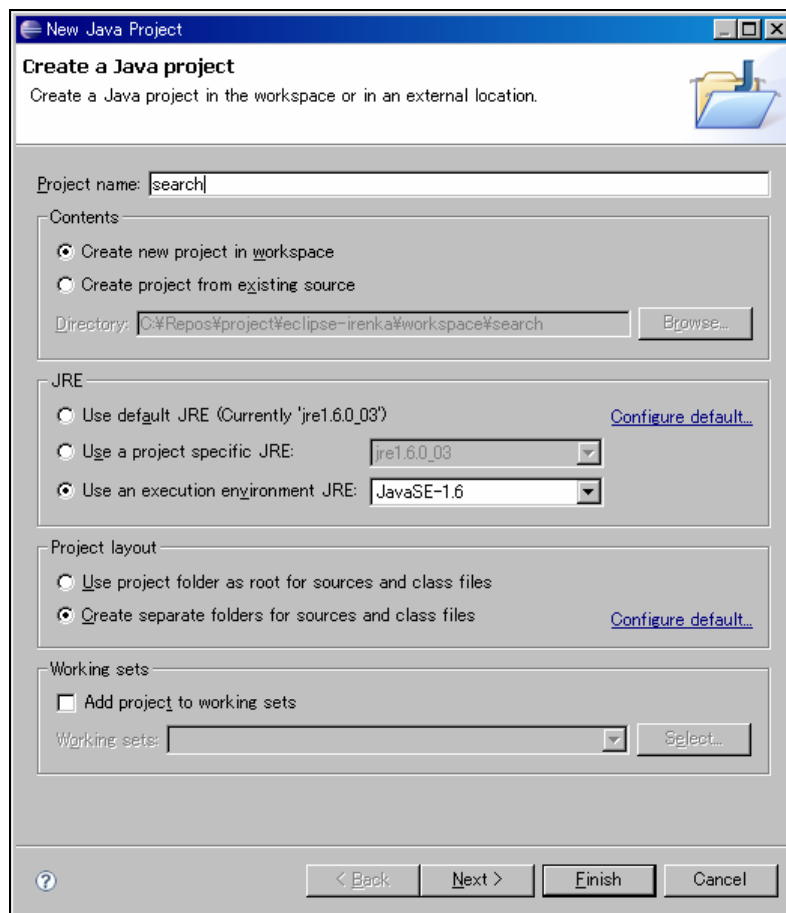


図 33. プロジェクト"search"の作成

現在のIrenka StudioでのIrenka Searchは、検索対象プロジェクトのクラスパスにIrenka Librariesが追加されている必要があります⁴。作成した"search"プロジェクトを右クリックし、Propertiesを選択します(図 34)。

⁴ 1.0 までに改修して、通常のプロジェクトも検索対象にできるようにする予定

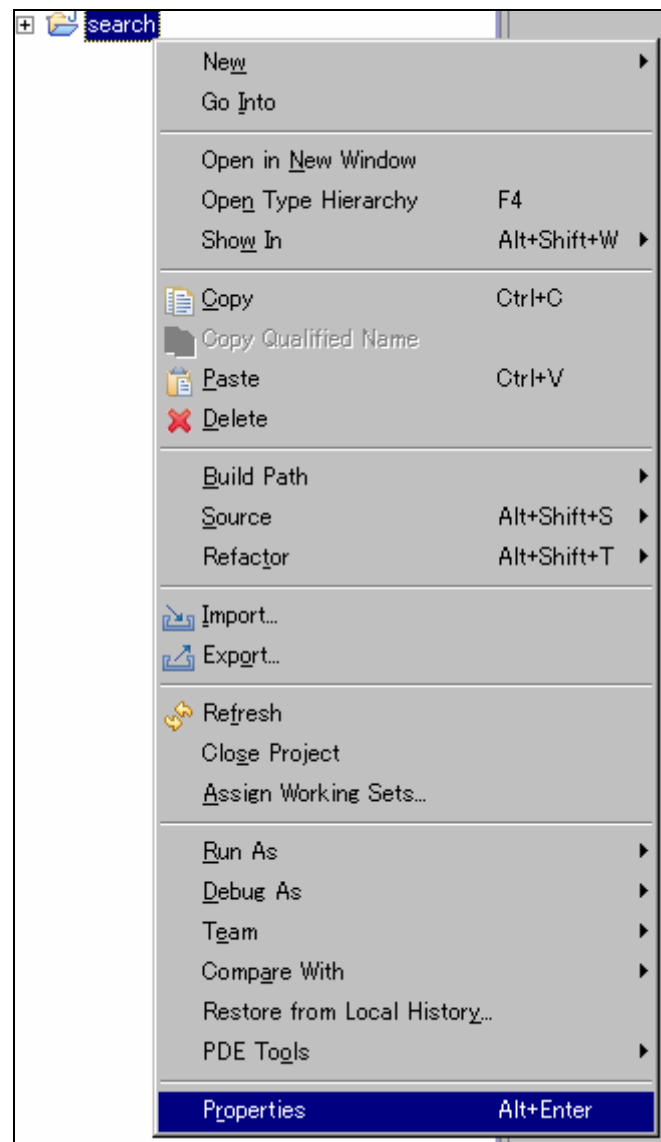


図 34. "search"プロジェクトのプロパティ

左側のペインからJava Compiler > Irenkaと選択し、右側のペインで"Irenka Builderを有効にする"というチェックボックスをチェックし、OKボタンを押下します(図 35)。

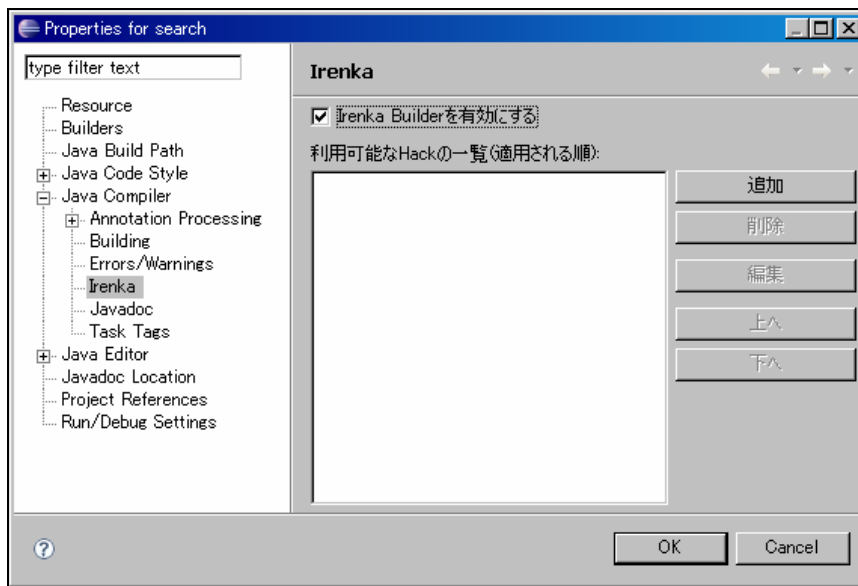


図 35. "search"プロジェクトの Irenka Builder を有効に

以上で、Irenka Librariesが登録されたプロジェクトが完成しました。次は、検索対象となるサンプルのクラスを作成します。ソースフォルダを右クリックし、New > Classと選択します(図 36)。

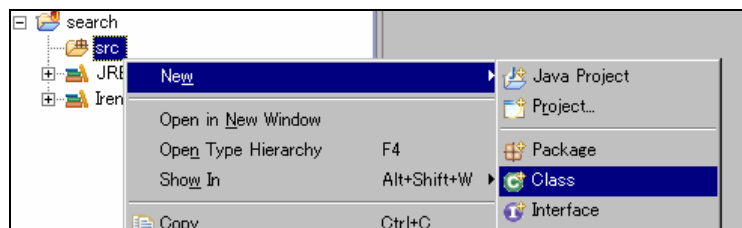


図 36. 検索対象クラスの作成

New Java Class Wizardが開かれますので、パッケージ名に"com.example"、クラス名に"Target"と入力します(好きな名前でもよい)。入力後、Finishボタンを押下します(図 37)。

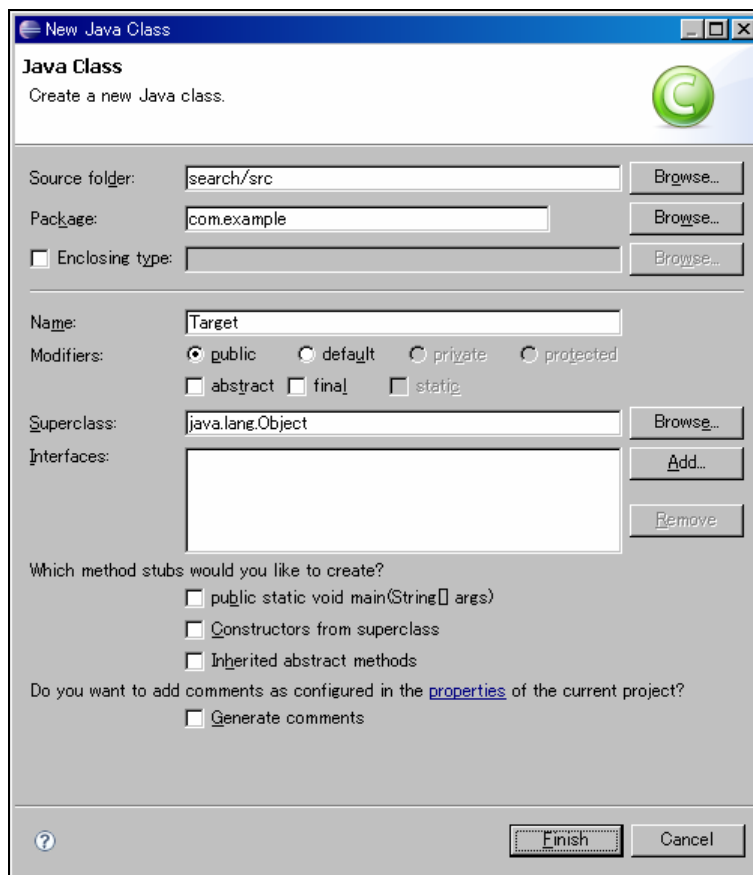


図 37. com.example.Target クラスの作成

空のクラスが作成されたら図 38の内容をコピーします。これで、サンプルの検索対象が用意されました。

```

package com.example;

public class Target {

    public int plus(int a, int b) {

        return a + b;

    }

    public int minus(int a, int b) {

        return a - b;

    }

    public int zero() {

        return 0;

    }

}

```

図 38. com.example.Target クラスの内容

メニューバーからSearch > Irenkaを選択します(図 39)。見つからない場合は、Search > Search...を開いたのち、Irenka Searchタブを選択する方法もあります。

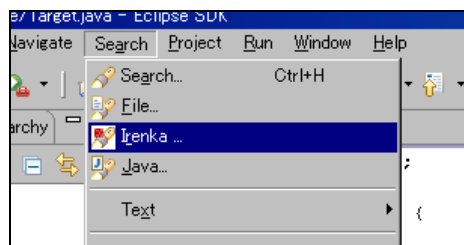


図 39. Irenka Search ウィンドウの起動

Irenka Searchを起動すると、図 40のようなダイアログが開かれます。例として、図 41の内容をダイアログ左上のテキストエリアにコピーします。これは、任意のメソッドを検索するためのクエリです。

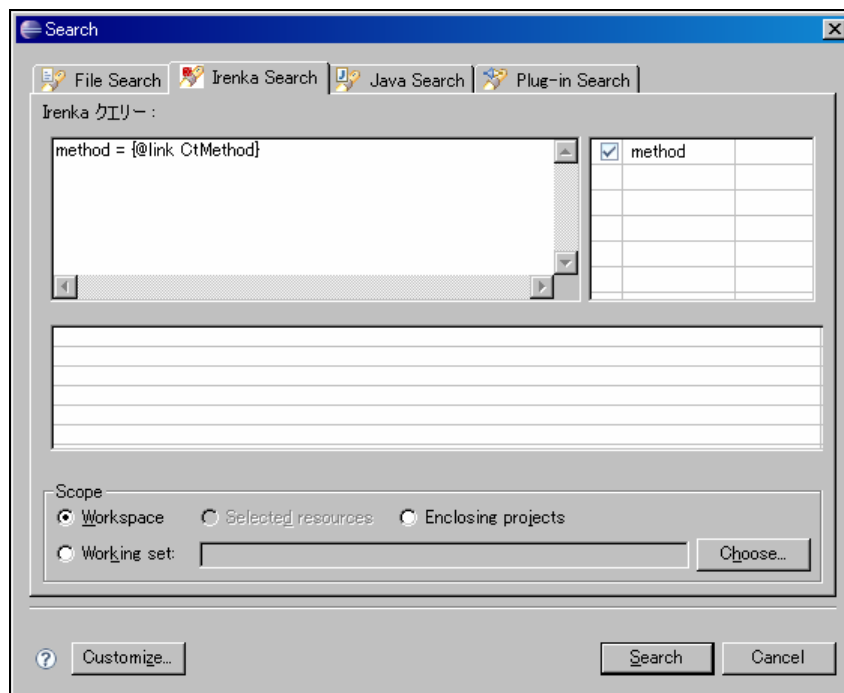


図 40. Irenka Search ウィンドウ

```
method = {@link CtMethod}
```

図 41. すべてのメソッドを探す Irenka Search Query

図 40の左上のエリアに入力したクエリ、"method = {@link CtMethod}"のうち、"method"は検索結果が格納されるプレースホルダで、Irenka Searchのエンジンが使用する変数のようなものです。このプレースホルダに対して様々な制限を掛けていくことによってIrenkaは対象を絞り込んでいきます。"method"の右側にある"= {@link CtMethod}"は、"method"に"CtMethod"というメソッドを表すDOMしか格納させないという制限を与えるためのプログラムで、結果として"method"には全ての要素の中からメソッドのみが選択されて格納されます。

右上のエリアには、入力されたクエリに含まれるプレースホルダの一覧が表示されます。ここでは、唯一のプレースホルダ"method"が表示されており、その左側に配置されたチェックボックスにチェックを入れることでプレースホルダに格納された内容を検索結果に含めることができます。今回は検索対象が全てのメソッドですので、メソッドが格納されるプレースホルダ"method"にチェックを入れています。

クエリを入力後、図 40の右下にあるSearchボタンを押下すると、指定のクエリを利用した検索が開始されます。今回は全てのメソッドを検出するというクエリを利用したので、図 42のようにすべてのメソッドが検索結果に含まれます。

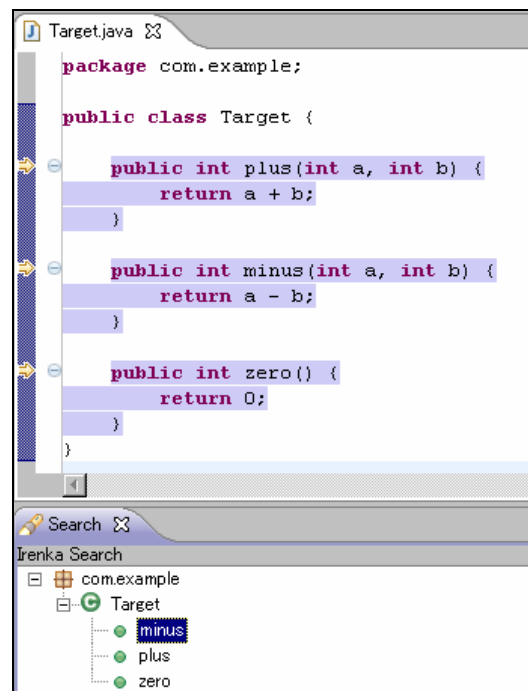


図 42. メソッドの検出結果

もう少し複雑な例として図 43を挙げます。これは、図 41にあったプレースホルダ"method"に対してさらに制限を掛けたもので、"method.parameters.type = (int, int)"という行を追加しています。まず、"x.parameters"という文法は"xの引数宣言一覧"を表し、さらに"y.type"という文法は"yの型"を表します。これを組み合わせた"method.parameters.type"は、"methodの引数一覧の型"になります。"="は左辺と右辺が同じであるという制限を掛ける記号で、その右辺は"(int, int)"というint型 2 つからなるリストです。

上記を組み合わせると、"methodの引数リストの型は、int, intである"という制限を表しています。これをクエリの 2 行目に追加することによって、検索対象が (int, int) を引数の型に持つメソッドになりました。

```

method = {@link CtMethod}
method.parameters.type = (int, int)

```

図 43. 引数の型が(int, int)である全てのメソッドを探すクエリ

実際にこのクエリを利用してIrenka Searchを実行すると、図 44のようにminus(int, int)とplus(int, int)が検索結果に含まれ、引数を持たないメソッドzero()は検索結果から除外されています。

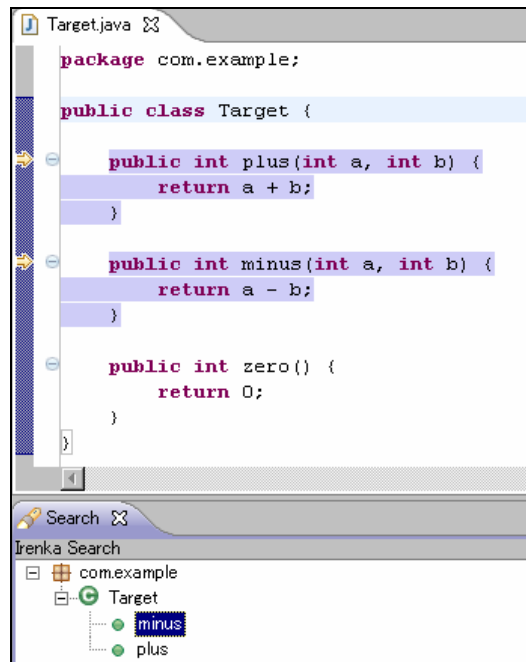


図 44. *(int, int)メソッドの検索結果

より複雑な例は、第 3 節や第 8 章第 2 節で紹介します。

第2節 Irenka Search Query の概要

Irenka Search Query についての完全な解説は Irenka Search Query Specification にありますが、ここでは簡単な紹介を行います。

第1項 プレースホルダへの制約

Irenka Search Query はプレースホルダに様々な制約を与えて格納される要素を制限し、複雑な構造を持つプログラムの一部を特定することができます。プレースホルダに格納される要素は基本的にプログラムに含まれる任意の構造で、クラスやメソッドの宣言、if 文、ローカル変数の宣言、String リテラルなど、Java のプログラムとして記述できるすべての要素を含めることができます。

簡単な例として、次のようなものがあります。

- a = {@link CtIf}
 - プレースホルダ"a"の内容は"CtIf の構造(if 文)"を持つ
- b = 1
 - プレースホルダ"b"の内容は整数"1"である
- c in method.body

➤ プレースホルダ"c"の内容はプレースホルダ"method"のプロパティ"body"に含まれる

上記のように、Irenka Search Query は「<式> <演算子> <式>」という形式で制約を記述し、左右の式に対して演算子による演算が成り立つものに検索を限定します。

また、同じ名前を持つプレースホルダがクエリ中に複数回出現した場合、それらは同一のプレースホルダとみなされます。

- public in method.modifiers

➤ 修飾子 public はプレースホルダ"method"のプロパティ"modifiers"に含まれる

- static in method.modifiers

➤ 修飾子 static はプレースホルダ"method"のプロパティ"modifiers"に含まれる

上記 2 つの文はどちらも method に対して言及しています。プロパティ"modifiers"は修飾子の一覧を表すプロパティで、二つを同一クエリ上に併記すると、"修飾子 public および修飾子 static は、プレースホルダ"method"のプロパティ"modifiers"にどちらも含まれる"という、より厳しい制約をプレースホルダに対して与えることができます。

プレースホルダの名前は、Java の変数名と同じルールで付けることができます。Java と同様に、public などの予約語はプレースホルダとして利用できません。

また、制約を掛けるための演算子は、大まかに次のようなものがあります(表 1)。

表 1. 主な制約演算子

演算子	概要
a = b	a と b は同一
a in b	a は b に含まれる
a < b	a は b 未満
a =~ "regex"	a は正規表現"regex"で表現可能な文字列

詳しくは、Irenka Search Query Specification を参照してください。

第2項 即値

即値はそのまま値として利用可能な要素のことで、Irenka Search Query 内に出現するとその表記のまま値として利用されます。たとえば、"a = 1"という制約のうち"1"は 1 という値を表す即値で、制約全体は「プレースホルダ"a"の内容は整数リテラル 1 である」と解釈されます。

主に、表 2にある種類の即値を利用することができます。パラメータ化型や宣言型の配列型などは通常のJavaドキュメンテーションコメントの仕様から外れていることに注意が必要です。

表 2. 利用可能な即値の一覧

即値の種類	例
整数リテラル	0, 1, -5, 10000L, 0x12345678abcdefL
浮動小数点数リテラル	0.0, 3.14f, -.5, 6.02e+23
ブールリテラル	true, false
文字リテラル	'a', '掟', '¥u0041'
文字列リテラル	"The Ashikunep Kotan", "Hello, world!"
修飾子	public, abstract, synchronized
基本型	int, double, void
宣言型	{@link String}, {@link java.util.List}
プリミティブ型の配列型	int[], char[][]
パラメータ化型	{@link java.util.Set<String>}, {@link Map<?, ?>}
宣言型の配列型	{@link String[]}, {@link Date[][]}
フィールド	{@link Math#PI}
メソッド	{@link Math#max(double, double)}
DOM 要素型	{@link CtMethod}, {@link CtClass}

DOM 要素型以外の即値は、プログラム上の即値と同じように扱われます。たとえば、`ifStatement.condition = false`は、`if (false) ..`という構造 `ifStatement` を検出することができますし、`public in decl.modifiers`は、`public` で宣言された構造 `decl` を検出することができます。

DOM要素型が出現した場合、対象のプレースホルダに格納されるDOM要素の種類を制限します。メソッドを表すCtMethodが出現する`method = {@link CtMethod}`がクエリ中に存在した場合、プレースホルダ`method`に格納されるDOM要素はメソッドに限られます。利用可能なDOM要素型のうち、主に利用されるものを表 3に紹介します。

表 3. 主な DOM 要素の種類

DOM 要素の型	概要
CtAnnotationInstance	注釈
CtAssignment	代入文
CtClass	クラス
CtDeclaredType	任意の宣言型(class, interface, enum, @interface)
CtElement	任意の要素
CtExpression	任意の式
CtField	フィールド
CtFor	for 文
CtIf	if 文
CtJavadoc	Javadoc
CtMethod	メソッド
CtModifier	修飾子
CtParameter	メソッド引数
CtReference	任意の宣言、またはそれらの参照
CtReturn	return 文

DOM 要素の型	概要
CtStatement	任意の文
CtType	任意の型

DOM 要素の種類については、Irenka DOM Specification により詳しい解説があります。

第3項 プロパティ

Irenka DOMはJavaのプログラムを表すツリー構造をしていて、それぞれの子要素は親要素のプロパティとして公開されています。Irenka Search Queryからもこのプロパティを利用することによって、任意の子要素を参照することができます。たとえば、図 45のような単純なif-else文は、図 46のような要素のツリー構造で表現することができ、それぞれの辺にあるプロパティ名を利用して子要素を参照できます。

```

if (true)
    return 1;
else
    return 2;

```

図 45. 単純な if-else 文

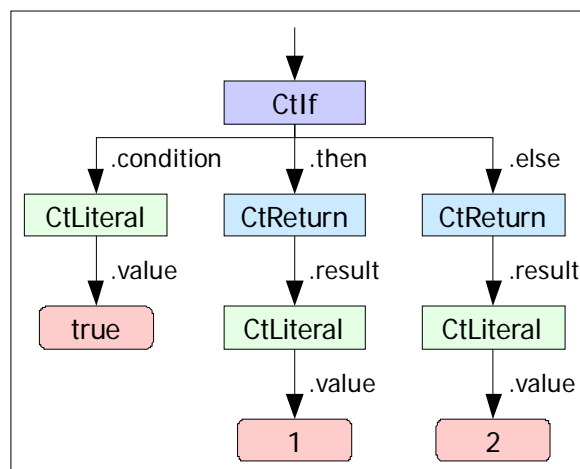


図 46. 図 45のツリー構造

プロパティを利用するには、"<式>.<プロパティ名>"の形式でクエリ内に記述します。図 45の文全体が"self"というプレースホルダに格納されていたとすると、"self.then"は"return 1;"を指し、"self.else"は"return 2;"を指します。さらに"self.then"や"self.else"もまたIrenka Search Queryの式であるため、"self.then.result"、"self.else.result"といった具合にプロパティを連結して書くこともできます。

Irenka Search Queryで利用可能なプロパティのうち、主に利用されるものを表 4に提示します。

表 4. 主なプロパティの一覧

プロパティ名	対象	概要
annotations	任意の宣言	注釈一覧
arguments	メソッドの起動	引数一覧
condition	条件文(if, while など)	条件式
constructors	クラス、列挙	コンストラクター一覧
elements	注釈	注釈要素一覧
else	if 文	else 節
javadoc	任意の宣言	Javadoc コメント
methods	宣言型	メソッド一覧
modifiers	任意の宣言	修飾子一覧
parameters	メソッド、コンストラクタ	引数宣言一覧
returnType	メソッド、注釈要素	戻り値型
simpleName	名前を持つ任意の宣言	単純名
target	メソッドの起動など	対象のメソッド、コンストラクタなど
then	if 文	then 節
type	注釈、new 式など	対象の型
variable	変数参照	参照先の変数

他のプロパティは Irenka Search Query Specification に詳しい解説があります。

第4項 リスト

Irenka Search Query は DOM 要素だけでなく、その 1 次元のリストも使用することができます。たとえば、宣言を持つ修飾子の一覧はプロパティ"modifiers"で取得できますが、これは修飾子を表す DOM 要素の 1 次元のリストとして表現されています。

リストに含まれる要素を取り出すには、"list[index]"といった Java の配列参照のような文法でクエリ内に書くことができます。たとえば"method.modifiers[0]"は、プレースホルダ"method"に含まれる先頭の修飾子を表します。また、リストを作成するには"(a, b, c)"のように丸括弧の中にカンマ区切りで式を書きます。たとえば"(1, 2, 3)"は、即値"1", "2", "3"からなる 1 次元のリストを表します。

第3節 簡単なサンプル

いくつかの Irenka Search Query のサンプルを提示します。

```
// stmt (CtReturn)の値は0
stmt = {@link CtReturn}
stmt.result = 0
```

図 47. クエリの例:"stmt は'return 0;'の形式である"

```
// stmt (CtReturn)はmethod (CtMethod)の本体に含まれる
method = {@link CtMethod}
stmt = {@link CtReturn}
stmt in method.body
```

図 48. クエリの例:"stmt は method に含まれる任意の return 文である"

```
// Deprecated型は、プレースホルダdeclの注釈一覧の型に含まれる
{@link Deprecated} in decl.annotations.type
```

図 49. クエリの例:"decl は@Deprecated アノテーションが付与されている"

```
// assign (CtAssignment)の左辺の変数はparam (CtParameter)である
param = {@link CtParameter}
assign = {@link CtAssignment}
assign.leftHandSide.variable = param
```

図 50. クエリの例:"assign はメソッド引数への代入を行っている"

```
// プレースホルダmethod (CtMethod)は
// 修飾子にpublicを含み、
// かつ修飾子にstaticを含み、
// かつ戻り値の型はvoidであり、
// かつ単純名は"main"であり、
// かつ引数の型は(String[])である
method = {@link CtMethod}
public in method.modifiers
static in method.modifiers
void = method.returnType
"main" = method.simpleName
( {@link String[]} ) = method.parameters.type
```

図 51. クエリの例:"method は main メソッドである"

第5章 Hack 開発の流れ

ここまでの章では、Irenka Studioの基本的な機能を利用する方法について紹介しました。ここより先は、Java コンパイラフレームワークである Irenka の機能を最大限に利用し、自分の Hack Library を開発する方法について紹介します。

この章ではまず、Hack 単体を開発および検証する流れについて簡単に紹介します。

第1節 Irenka Studio での Hack

Irenka Studio は次のような要件をすべて満たすソースプログラム内のクラス宣言を Hack とみなします。Hack として認識可能なのはクラスのみで、インターフェース、列挙、注釈ではありません。

1. ビルドパス上のファイル内で定義されており、同ファイルは拡張子が".java"である
2. public であり、かつ abstract でないトップレベルのクラスとして宣言されている
3. 同クラス内で引数を持たない public コンストラクタを宣言する
4. 同クラス内で 1 つ以上の Hack Action と解釈可能なメソッドを宣言する

定義 1. ソースプログラム内の Hack

Hack Action とは Hack を構成するプログラムで、それぞれの Hack は必ず 1 つ以上の Hack Action を含む必要があります。Irenka では、次の要件をすべて満たすメソッドを Hack Action とみなします。

1. Java ドキュメンテーションコメントが直接付与されている
2. Java ドキュメンテーションコメント内に Irenka Search Query を記述する"@when"タグブロックを有する
3. public であり、かつ static でないメソッドとして宣言されている
4. 1つ以上の引数を宣言し、いずれも次の型のうちいずれかを有する
 - org.ashikunep.irenka.dom.CtElement 型、またはそのサブタイプ
 - org.ashikunep.irenka.toolkit.Tool 型のサブタイプ
 - org.ashikunep.irenka.event.CtEvent 型、またはそのサブタイプ

定義 2. ソースプログラム内の Hack Action

定義 1を満たすクラスの単純な例を図 52に提示します。これはJavadocに空の"@when"ブロックを持つメソッド"action"のみからなるクラスで、この記述だけでIrenka Studioはクラス com.example.SimpleHackをHackとみなすことができます。なお、この単純なHack Actionを実装するメソッド"action"は、Hack対象の全ての要素が順次引数elementにドライブされ、要素数と同じ回数だけメソッドが起動されます。

```

package com.example;
import org.ashikunep.irenka.dom.CtElement;
public class SimpleHack {
    /**
     * @when
     */
    public void action(CtElement element) {
        // ...
    }
}

```

図 52. 単純な Hack の例

Irenka Studio 内で Hack とみなされたクラスは、Irenka Builder に登録したり Hack Packager を利用して配布したりすることができます。

第2節 Hack の作成

本節では、実際に簡単なHackを作成する方法を紹介します。複雑なHackの作成方法は第6章、作成したHackの検証方法は第3節、作成したHackの配布方法は第7章をそれぞれ参照してください。

まず、Hackを作成するためのプロジェクトを新規に作成します。Hackの作成はEclipseのJava開発環境をそのまま利用します。メニューバーのFileからNew > Java Projectの順に選択します(図 53)。

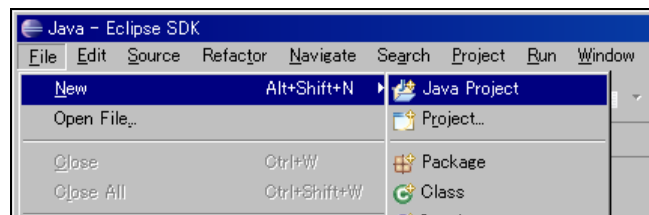


図 53. Hack を作成するプロジェクトの作成

New Java Project Wizardが開きますので、プロジェクト名に"hack"(名前は自由)と入力してFinishボタンを押下します(図 54)。

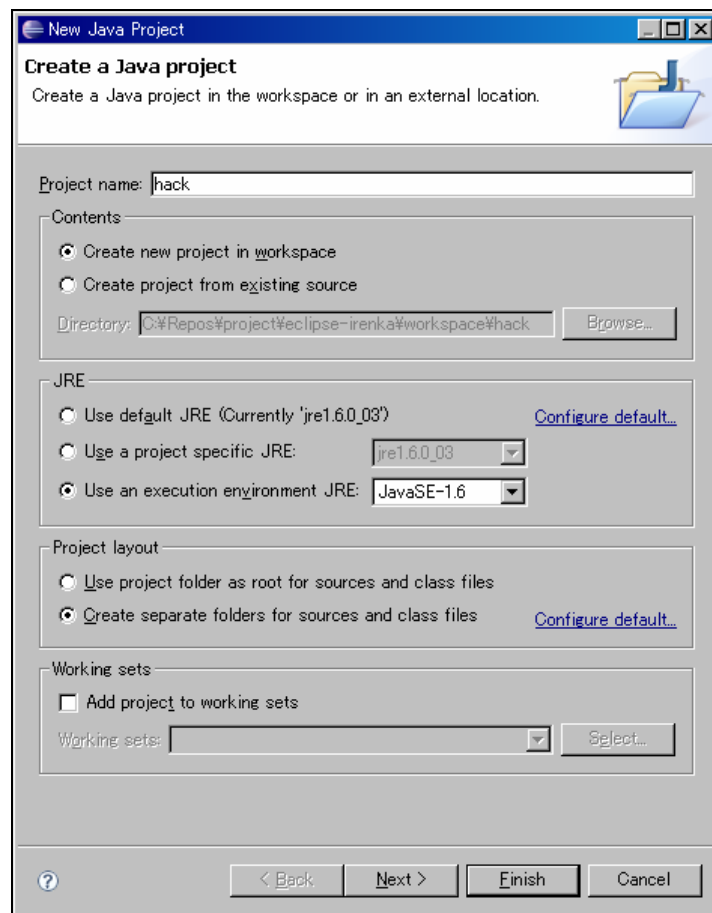


図 54. プロジェクト"hack"の作成

Hackを開発する際には、Irenka Librariesに含まれるIrenka End User APIを利用します。これはHackに必要なインターフェースなどが含まれています。作成した"hack"プロジェクトを右クリックし、Propertiesを選択します(図 55)。

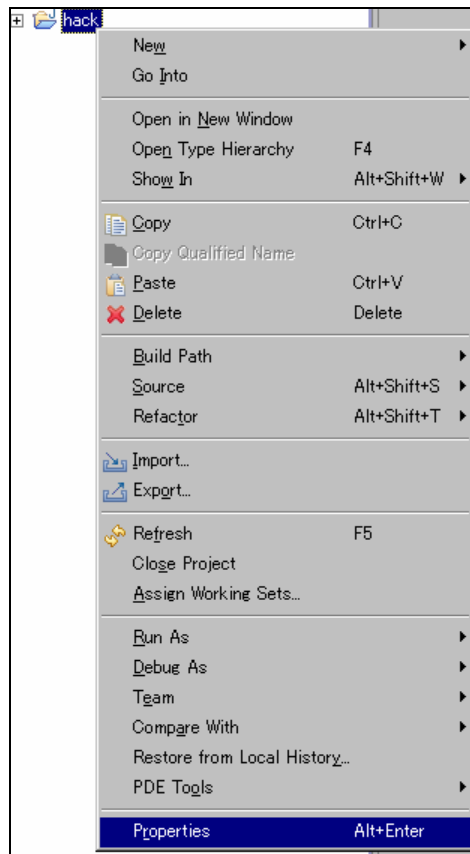


図 55. "hack"プロジェクトのプロパティ

左側のペインからJava Compiler > Irenkaと選択し、右側のペインで"Irenka Builderを有効にする"というチェックボックスをチェックし、OKボタンを押下します(図 56)。

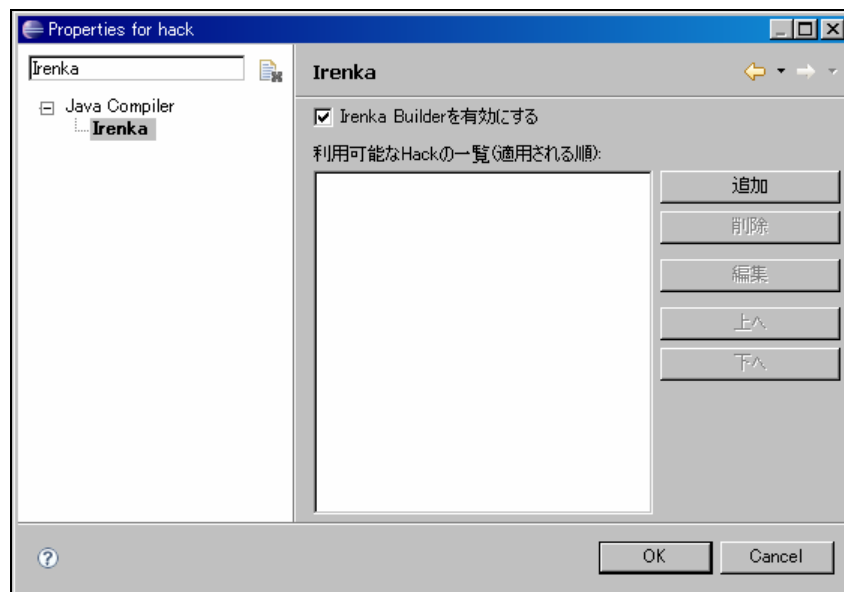


図 56. "hack"プロジェクトの Irenka Builder を有効に

以上の作業で、プロジェクト"hack"のビルドパスにIrenka End User APIのライブラリであるorg.ashikunep.ireнка.specプラグインが追加されます(図 57)。

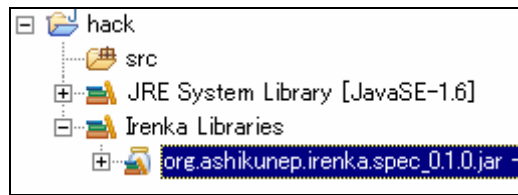


図 57. Irenka ライブラリが追加された様子

次は、「引数を持たないメソッド宣言を検出し、それらを警告する」という単純なHackを作成します。ソースフォルダを右クリックし、New > Classと選択します(図 58)。

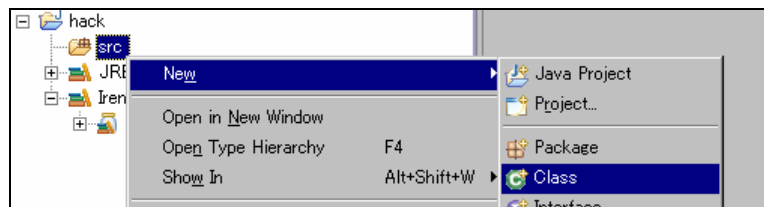


図 58. Hack 実装クラスの作成

New Java Class Wizardが開かれますので、パッケージ名に"com.example"、クラス名に"DetectEmptyParams"と入力します(好きな名前でもよい)。入力後、Finishボタンを押下します(図 59)。

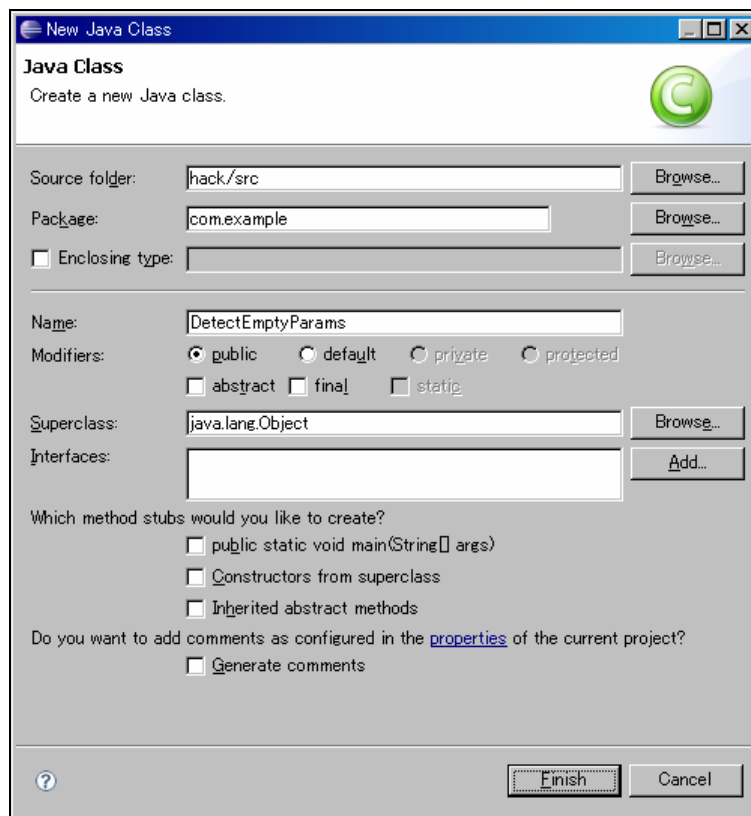


図 59. com.example.DetectEmptyParams クラスの作成

空のクラスが作成されたら、それを図 60の内容で上書きします。

```

package com.example;
import org.ashikunep.irenka.dom.CtMethod;
import org.ashikunep.irenka.toolkit.Messenger;
public class DetectEmptyParams {
    /**
     * @when
     *     method.parameters = ()
     */
    public void found(CtMethod<?> method, Messenger msgr) {
        msgr.warn(method, "no parameters");
    }
}

```

図 60. com.example.DetectEmptyParams クラスの内容

このように、Javadocに"@when"ブロックを持ったメソッドをクラス内に作成すると、そのクラスは Hack として取り扱われます。@when ブロックには Irenka Search Query を記述することができ、ここでは"method.parameters = ()"という、「"method"の引数宣言リストが空」である構造を

探すクエリを記述しています。@when ブロックを持つメソッド found は、2 つの引数を持ちます。これらはそれぞれ、下のような意味を持っています。

- org.ashikunep.ireнка.dom.CtMethod<?> method
 - CtMethod<?>は CtElement のサブタイプでメソッドを表す DOM 要素
 - @when ブロック中のプレースホルダ"method"に格納される値が代入される
 - @whenブロック中のプレースホルダ"method"はCtMethod型、およびそのサブタイプ⁵に限定される
- org.ashikunep.ireнка.toolkit.Messenger msgr
 - Messenger は Tool のサブタイプでメッセージを通知するためのオブジェクト
 - メソッドが呼び出されるたびに、Irenka Studio が適切なインスタンスを代入する

結果として、メソッド found は引数宣言リストが空であるようなメソッド、つまり「引数を持たないメソッド宣言」の DOM が引数 method に次々と代入されて呼び出されます。その際、引数 msgr にはメッセージを通知するためのオブジェクトが Irenka Studio から自動的に代入されます。found の本体では、Irenka Studio から与えられた Messenger を利用して、警告(warn)メッセージを Irenka Studio に通知します。これらをまとめると、作成した DetectEmptyParams は、「引数を持たないメソッド宣言を検出し、警告を行う」という Hack になりました。

引数にCtElementのサブタイプが与えられた場合の詳しい解説は第 6 章第 4 節に、Toolのサブタイプが与えられた場合の詳しい解説は第 6 章第 5 節にそれぞれあります。詳しくはそちらも参照してください。

最後に、"hack"プロジェクトのIrenkaのプロパティ画面を確認すると、作成した DetectEmptyParamsクラスがHackとして追加できるようになっています(図 61)。

⁵ 現在のバージョンでは、型引数は無視されてしまいます

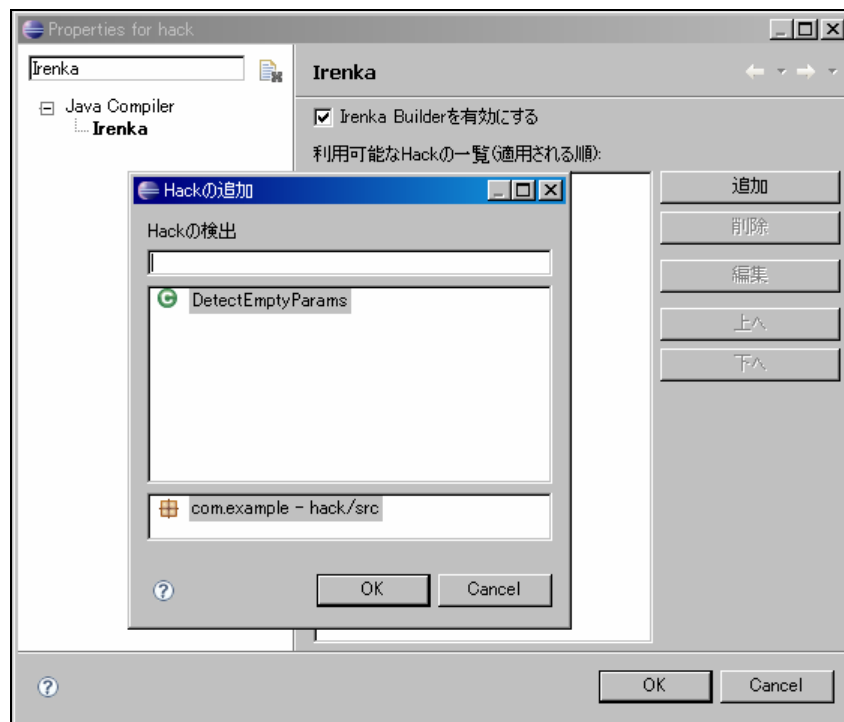


図 61. com.example.DetectEmptyParams が Hack の一覧に表示された状態

次節では、DetectEmptyParams の動作を確認する方法を紹介します。

第3節 Hack の検証

本節では、Irenka Builderを利用して第 2 節で作成したHackの動作を確認する方法を紹介します。実際の運用時にはHack Libraryを作成(第 7 章第 1 節)し、Irenka Builderに登録(第 3 章)などの方法をとるのがよいのですが、Hackの確認と修正を素早く行うにはここで紹介する方法が便利です。

まず、Hackを検証するためのプロジェクトを新規に作成します。わざわざ別のプロジェクトを用意するのは、Hackもまたクラスとして定義されており、同一のプロジェクト上で検証するとHackを実現するクラス自体にHackが適用されてしまい問題の切り分けが難しくなる恐れがあるためです。メニューバーのFileからNew > Java Projectの順に選択します(図 62)。

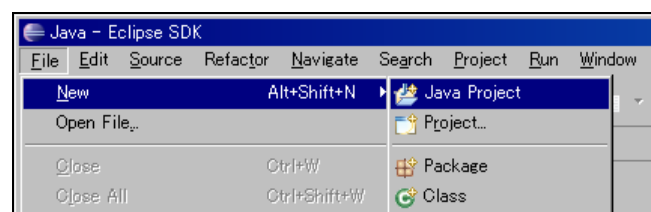


図 62. Hack テストプロジェクトの作成

New Java Project Wizardが開きますので、プロジェクト名に"hack-test"(名前は自由)と入力してFinishボタンを押下します(図 63)。

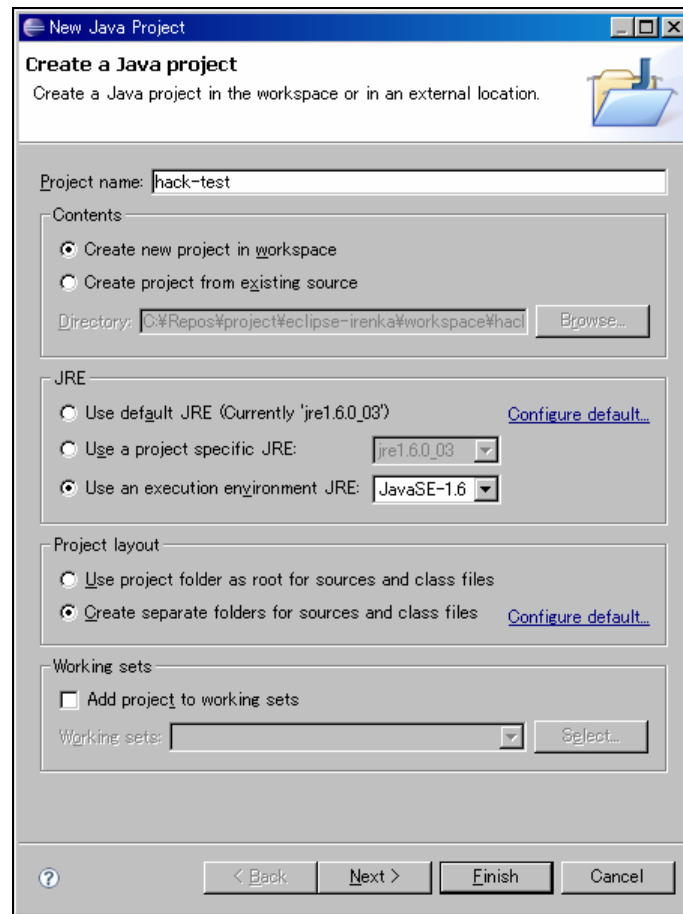


図 63. プロジェクト"hack-test"の作成

次に、Irenka Builderにテスト対象のHackを登録します。まず、作成したプロジェクト"hack-test"を右クリックし、Propertiesを選択します(図 64)。

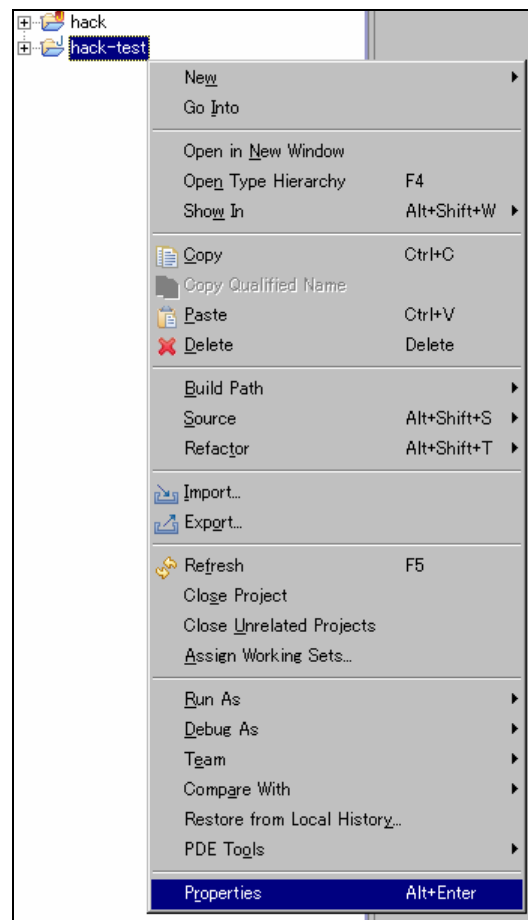


図 64. "hack-test"プロジェクトのプロパティ

プロジェクト上に存在しないHackを利用するには、対象のクラスをビルドパスに追加する必要があります。そこで、左側のペインからJava Build Pathを選択し、右側のペインでProjectsタブを選択します(図 65)。

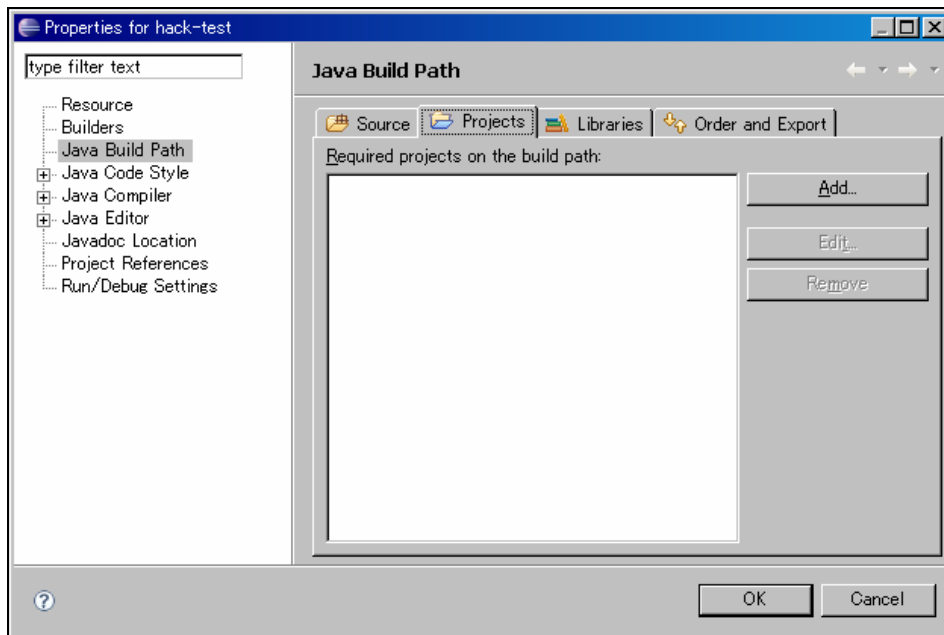


図 65. 依存プロジェクト設定タブ

Eclipseではプロジェクトに依存関係を設定すると、依存元のビルドパスに依存先のソースコードを含めることができるようになります。タブの右側に配置されたAddボタンを押下すると、プロジェクトの一覧を表示するダイアログが現れますので、テスト対象のHackを含むプロジェクトにチェックを入れ、OKボタンを押下します(図 66)。

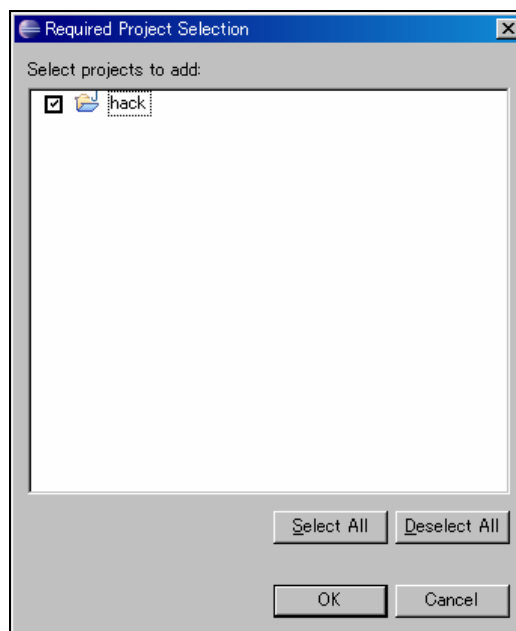


図 66. 依存するプロジェクトの選択

設定が完了すると、ビルドパス一覧に選択したプロジェクトが追加されます(図 67)。複数のプロジェクトに対する検証を実施する場合は、必要に応じて依存プロジェクトを追加します。

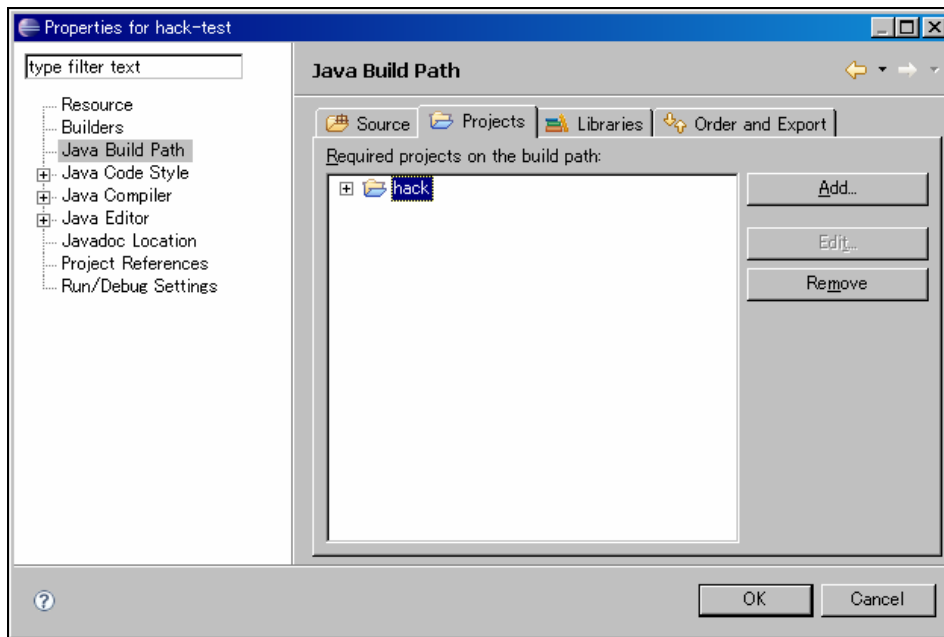


図 67. "hack"プロジェクトを依存先プロジェクトにした状態

次に、Irenka Builderの設定を行います。まず、プロジェクトプロパティ画面の左ペインから、Java Compiler > Irenkaを選択します。このとき、Java Build Pathの設定直後であった場合には図 68のようなダイアログが表示されますので、Applyを選択してプロジェクトの依存関係を確定させてください。

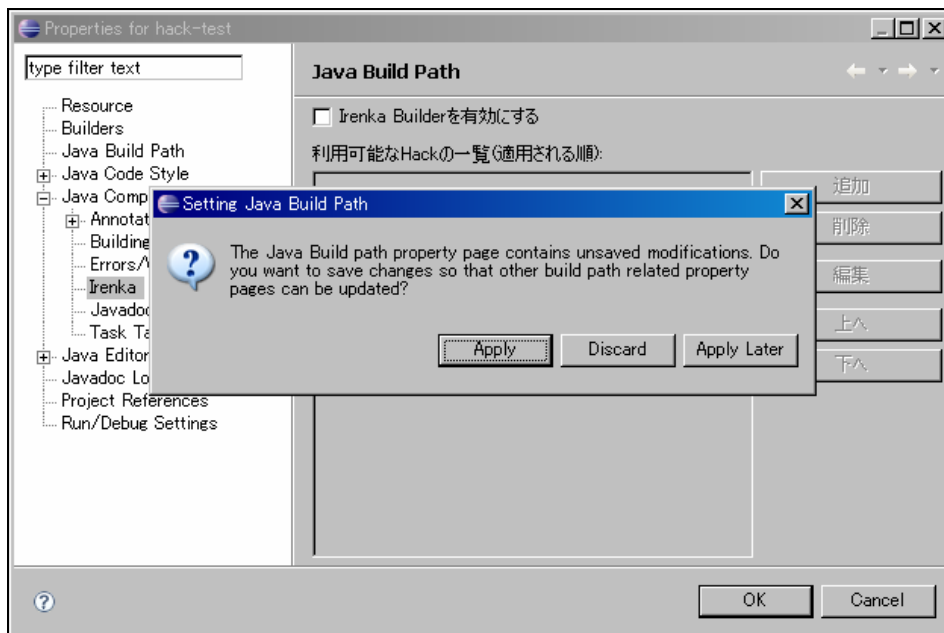


図 68. Irenka Builder の設定/依存プロジェクト設定の保存

Java Compiler > Irenkaのページ上部にある"Irenka Builderを有効にする"という項目にチェックを入れたのち、追加ボタンを押下することで検証を実施するHackを追加することができます(図 69)。

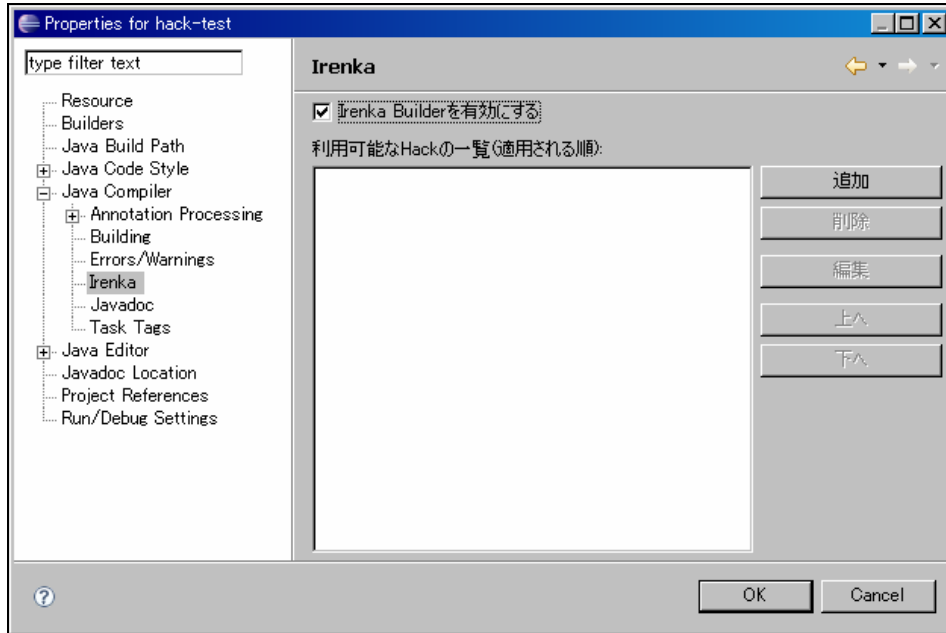


図 69. "hack-test"プロジェクトの Irenka Builder を有効に

追加ボタンを押下すると、追加するHackの一覧がダイアログ上に表示されます。このとき、検証対象のクラスが表示されなかった場合、対象クラスがHackと認められない構造をしているか、またはビルドパスの設定に誤りがあると考えられます。ここでは、第2節で作成した DetectEmptyParamsクラスを選択してOKボタンを押下します(図 70)。

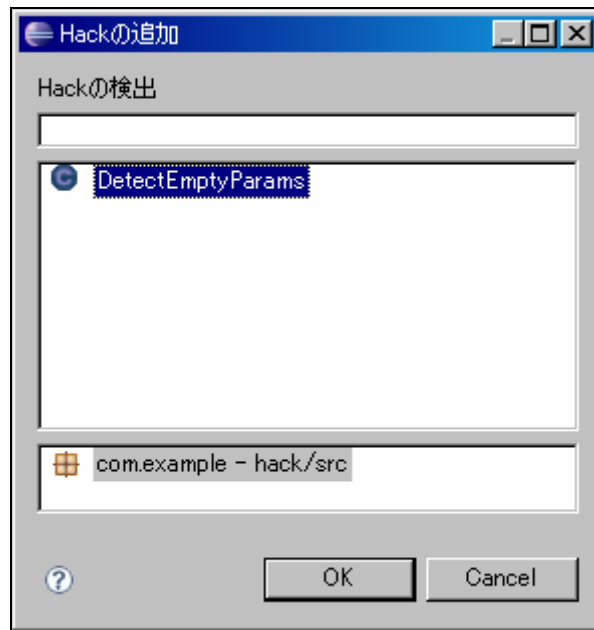


図 70. Hack 追加ウィンドウ

Hackの追加に成功すると、図 71のように一覧に対象のHackが追加されます。必要なHackをすべて追加したら、OKボタンを押下してIrenka Builderの設定を確定させます。

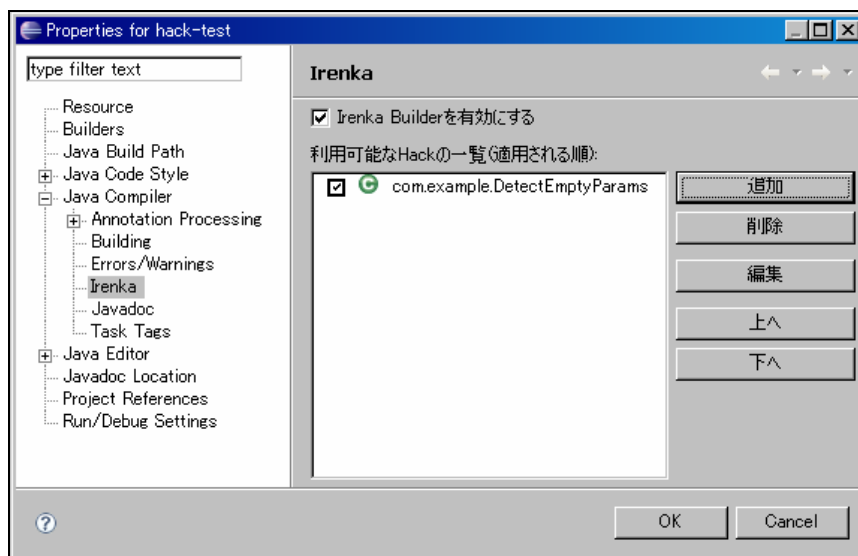


図 71. テスト対象 Hack が Irenka Builder に追加された状態

Irenka Builderの設定が完了したら、次にHackの動作を確認するためのプログラムを作成します。第 2 節で作成したHackはメソッドの宣言について動作を起こす種類のものでしたので、ここでは確認のために通常のクラスを作成します。プロジェクトのソースフォルダを右クリックし、New > Classと選択します(図 72)。なお、注釈や列挙など、クラス型以外を対象とするHackについて確認する場合には、必要に応じて対象を作成してください。

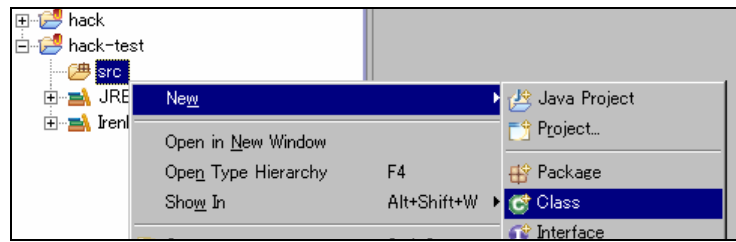


図 72. テスト対象クラスの作成

New Java Class Wizardが開かれますので、パッケージ名に"com.example"、クラス名に"SimpleHackTest"と入力します(図 73)。ここも、Hackの内容に応じて適切に設定してください。

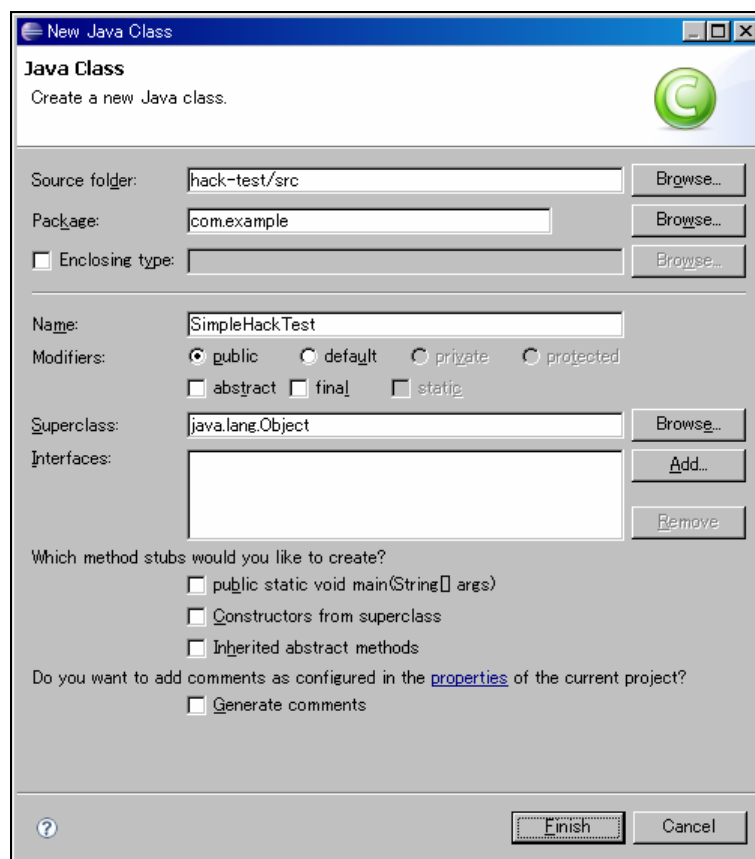


図 73. com.example.SimpleHackTest クラスの作成

今回はDetectEmptyParams - "引数を持たないメソッド宣言を検出し、警告を行う"というHackでしたので、引数を持たないメソッドと引数を持つメソッドの両方をSimpleHackTestクラス内に作成します。例として、図 74のような簡単な実装を行います。

```

package com.example;

public class SimpleHackTest {

    public int add(int a, int b) {
        return a + b;
    }

    public int one() {
        return 1;
    }
}

```

図 74. com.example.SimpleHackTest の内容

テスト対象の実装が終了したら、ファイルを保存することによってIrenka Builderを起動させます。Irenka BuilderにはDetectEmptyParamsが登録されていますので、図 75のように引数を持たないメソッドone()に対して警告を行っていることが確認できます。

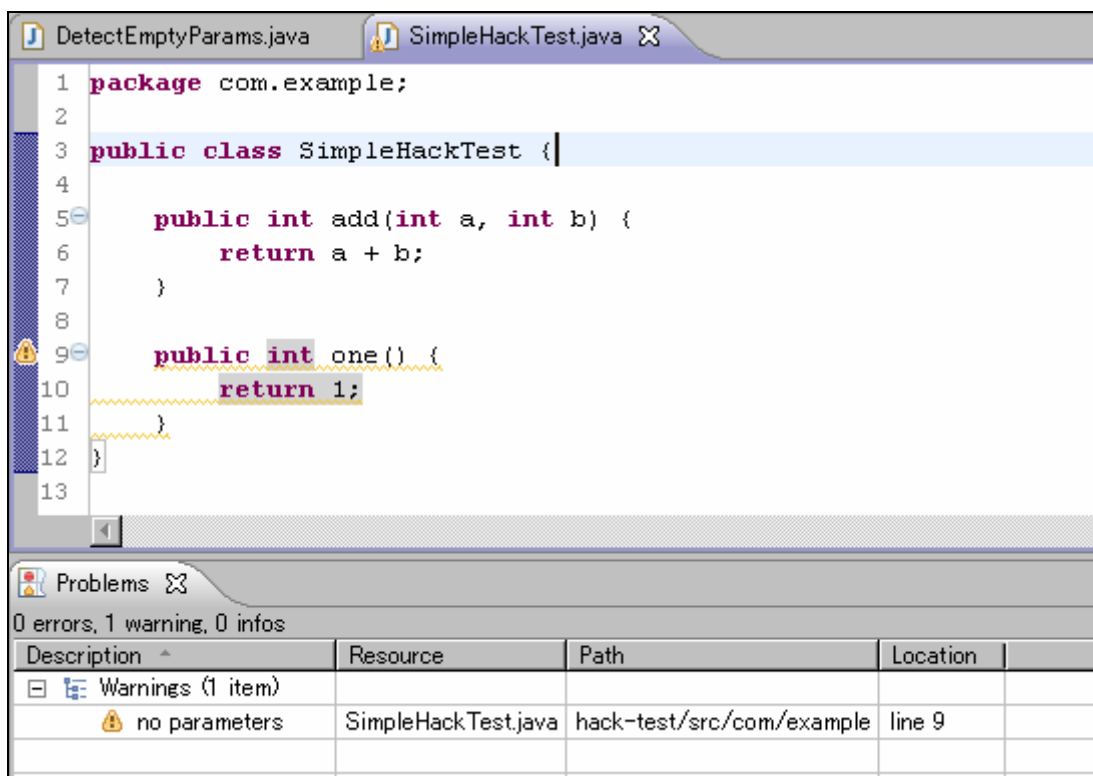


図 75. Hack: DetectEmptyParams が適用された状態

この警告は引数を持たないメソッド宣言に対して行われるものでしたので、メソッドoneに対して引数を追加してプログラムを保存することで警告を抑制することができます(図 76)。

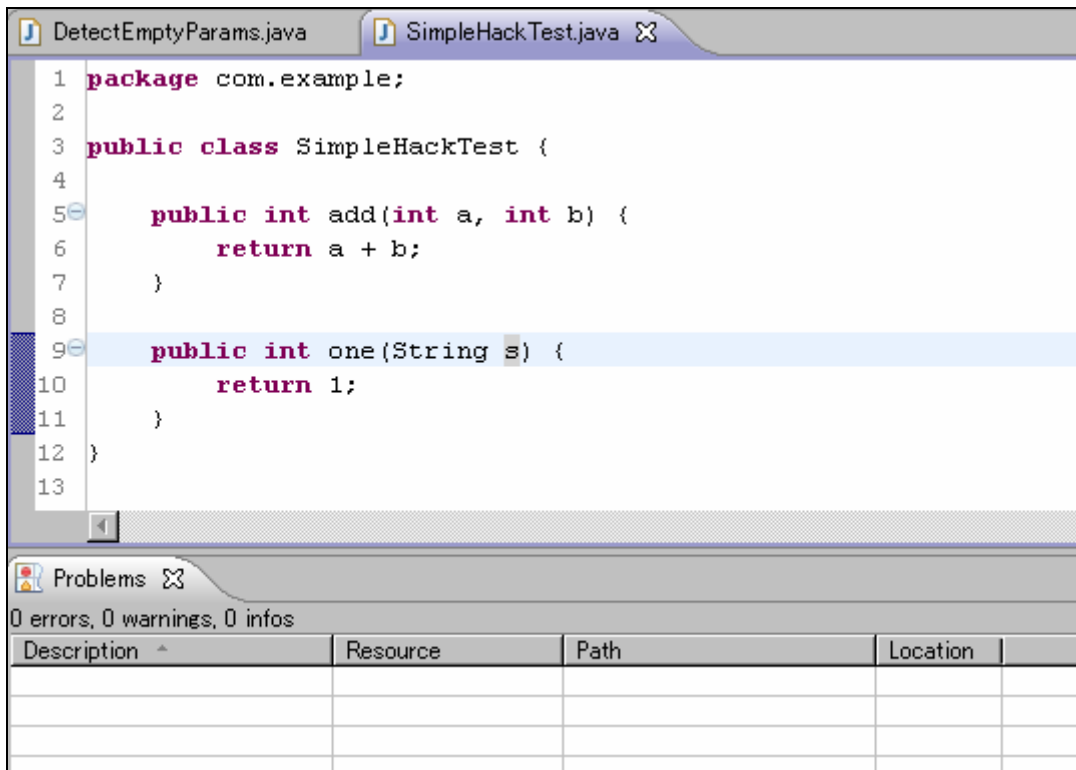


図 76. Hack: com.example.DetectEmptyParams の適用対象外となった状態

以上で、Hackの作成から簡単な検証までの流れを紹介しました。これらのHackをHack Libraryとして配布可能な形式に変換するには、Hack PackagerというIrenka Studioに内蔵されたツールを利用します。Hack Packagerの利用法については第 7 章を参照してください。

第6章 Hack

Hack は Irenka のコンパイラとしての機能を拡張するためのユーザプログラムであり、コンパイルフェーズに介入して特定プログラムの検出処理や変更処理などをユーザが自由に記述することができます。

個々の Hack は Hack Action というプログラムを 1 つ以上含んでおり、これらには検索する内容や検索結果を利用した処理を記述することができます。そして、Hack Action はさらに下記の 3 つの要素に分解できます。

- モデル内の検索条件 (Search Query)
- 検索結果を受け渡す変数群 (Hack Parameter(s))
- 検索結果に対する動作 (Match Action)

Irenka は Hack を適用する際、その Hack に含まれる個々の Hack Action を次のように処理します。

1. Search Query を発行し、コンパイル対象から合致する DOM を検出する
2. 検出した DOM を Hack Parameter に保存する
3. Match Action を実行する
4. Search Query による検出結果が残っていれば、次の検索結果に対して 2-3 を実行する

Irenka Studioでの全てのHackは、Javaの言語仕様の中で記述することができ、それぞれの要素は表 5の表記に対応します。また、Hack Actionを図解すると図 77のようになります。

表 5. Irenka Studio での Hack の表現方法

要素	Irenka Studio での表記
Hack	Hack Action を含むトップレベルクラスの宣言
Hack Action	Search Query を含む public メソッドの宣言
Search Query	'@when'タグを持つ Java ドキュメンテーションコメント
Match Action	メソッド本体ブロック
Hack Parameter	メソッドの引数宣言

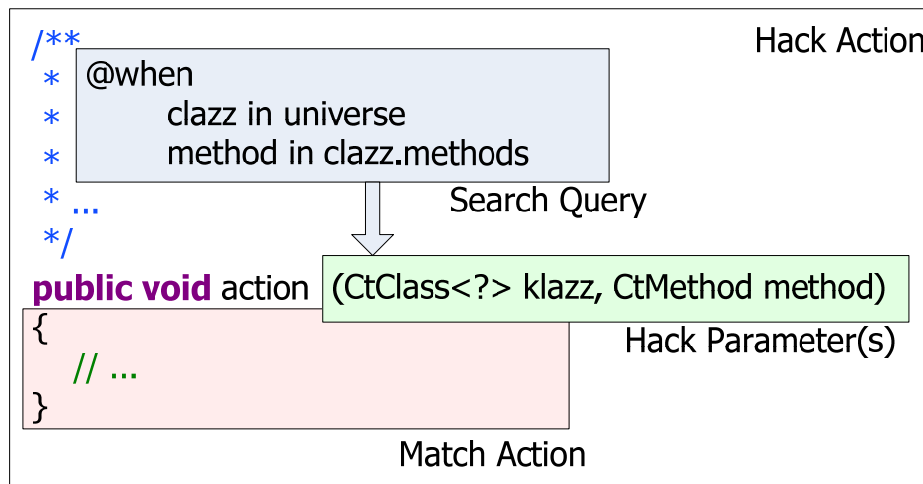


図 77. Hack Action の構造

第1節 Hack Action の起動

Irenka の Hack は、そこに含まれる個々の Hack Action が実行されて様々な効果をプロジェクトに対して及ぼすことができます。

Irenka Builder は、それぞれの Hack Action を次のような手順で実行します。

1. リソースに変更が行われると Irenka Builder が起動される
2. 変更されたリソースから DOM のツリーを生成し、そこに含まれる要素を検索対象とする
3. Irenka Builder に追加された Hack の中から、Hack Action を取り出す
4. それぞれの Hack Action が持つ Search Query で、検索対象を検索する
5. 検索に成功したら Hack Action が持つ Hack Parameter に検索結果を設定し、Match Action を実行する

第2節 Search Query の記述

Irenka StudioでHackを作成する場合、Search QueryはJavaドキュメンテーションコメント内の"@when"というタグのついたブロック内に記述します。もっとも単純な例は、図 78のように空の@whenブロックのみからなるJavaドキュメンテーションコメントです。この解釈については、第3節に説明があります。

```

/**
 * @when
 */

```

図 78. 最も単純な Search Query の例

Irenka StudioはJavaドキュメンテーションコメント内から@whenブロックを探す際、それ以外の要素を無視します。そのため、Search Queryを記述するJavaドキュメンテーションコメントは、図 79のように本来のドキュメントを併記することができます。ただし、@whenタグはブロックタグとして出現する必要があります。"{@when ...}"のようにインラインタグとして記述することはできません。また、@whenタグブロックはJavaドキュメンテーションコメント内に2つ以上存在することができません。

```
/**
 * {@code return 0;}を検出し、警告を通知するHackです。
 * @param msgr メッセージを通知するためのオブジェクト
 * @param stmt {@code return 0;}の検出結果
 * @see Messenger
 * @see CtReturn
 * @when
 *     stmt.result = 0
 */
```

図 79. 通常の Java ドキュメンテーションコメント内で Search Query を記述する例

また、Irenka Search Queryの文法の中には、"@link"インラインタグを利用して宣言や参照を利用するものがあります。Irenka Studioではこのような場合に、図 80のように参照先を単純名で指定することができます。この場合、単純名はJavaのプログラム内に記述された単純名と同様の方法で解決されます。つまり、図 80のSearch Queryを正しく解釈するには、次のいずれかを満たす必要があります。

- Search Query が記述されたコンパイル単位の import 宣言によって、List がインポートされている
- Search Query が記述されたクラスと同一パッケージ上に単純名"List"を持つ型が宣言されている
- Search Query が記述されたクラスに、"List"という名前のメンバ型が宣言されている
- java.lang.List という型が存在する

上記のいずれでもない場合、"{@link List}"は参照を解決することができません。Search Query はコンパイルエラーとなります。

```
/**
 * @when
```

```
*   listType = {@link List}
*/
```

図 80. 単純名を利用した参照

同様に、"@link"インラインタグを利用してメンバの参照を行う場合に、Search Query が記述されたクラスと同一クラス上のメンバについてはクラス名を省略し、"{@link #field}"などの表記が行えます。

Irenka Search Query についての詳しい内容は、Irenka Search Query Specification を参照してください。

第3節 Hack Parameter の記述

Hack Parameter は Search Query の検索結果を一時保存するための領域で、型と名前を持ちます。Search Query 内のプレースホルダと同名の Hack Parameter は関連付けられ、検索成功時に同名のプレースホルダに格納された内容は、それぞれ Hack Parameter に一時的に保存されます。その後、Hack Parameter は Match Action に引き渡され、検索結果を Hack Parameter を経由して参照することができます。

Hack Action は1つ以上の Hack Parameter を持ち、それぞれは次の型のいずれかで宣言されている必要があります。Hack Parameter は複数宣言することができますが、そのうち CtEvent 型であるものは高々1つである必要があります。

- org.ashikunep.ireнка.dom.CtElement、またはそのサブインターフェース
- org.ashikunep.ireнка.toolkit.Tool のサブインターフェース
- org.ashikunep.ireнка.event.CtEvent、またはそのサブインターフェース

上記のうち、CtElementは全てのIrenka DOM要素が共通して持つインターフェースで、同 Hack ParameterにはSearch Queryで検出されたDOM要素が直接保存されます(第4節)。ToolはHackを行う際に様々な機能を提供する道具を表すインターフェースで、Search Queryの結果によらずIrenka Studioから適切な実体を与えられます(第5節)。最後のCtEventはイベントの内容を表すインターフェースで、CtElementのサブインターフェースとして定義されています。そのため、CtEvent型を持つHack ParameterはCtElementの際と同様の方法でSearch Queryの結果が格納されます(第6節)。

Search Query を利用して検索を行う際、対応する Hack Parameter によって検索する内容が多少変更されることがあります。その際は、Hack Parameter ごとに下記のような手順で変更を加えます。

- Hack Parameter の型を T, 名前を N とおく
- T が CtElement、またはそのサブインターフェースであった場合

- Search Query 内に N という名前のプレースホルダが存在する場合
 - ✧ N に「T 型、またはそのサブタイプのみ格納できる」という制約を付加する
- Search Query 内に N という名前のプレースホルダが存在しない場合
 - ✧ 「T 型、またはそのサブタイプのみ格納できる」という制約を持つ N という名前のプレースホルダを作成する
- T が Tool のサブインターフェースであった場合
 - Search Query 内に N という名前のプレースホルダが存在する場合
 - ✧ 対応する Search Query は必ず検索に失敗する
 - Search Query 内に N という名前のプレースホルダが存在しない場合
 - ✧ 何も行わない
- T が CtElement、および Tool のサブインターフェースでない場合
 - 対応する Search Query は必ず検索に失敗する

たとえば、図 81 のように Search Query 上にプレースホルダが存在しない Hack Parameter を宣言した場合、図 82 のような Search Query として解釈されます。なお、これは「プレースホルダ "method" は任意のメソッド」と解釈することができます。

```
/**
 * @when
 */
public void apply(CtMethod<?> method) {
    // ...
}
```

図 81. Hack Parameter に対応するプレースホルダが存在しない例

```
/**
 * @when
 *     method = {@link CtMethod<?>}
 */
public void apply(CtMethod<?> method) {
    // ...
}
```

図 82. Hack Parameter に対応するプレースホルダが存在しない場合の解釈

なお、Search Query内に既に"method"という名前のプレースホルダが存在していた場合も、[図 82](#)と同様に"method = {@link CtMethod<?>}"という制約が追加されます。これは既存のプレースホルダ"method"に対してメソッドのみを保持させるという制約と解釈されます。

第4節 Irenka DOM の操作

DOM 要素のインターフェースを持つ Hack Parameter を記述することによって、Search Query によって検出された DOM 要素を Match Action の中で操作することができます。Irenka DOMの要素は、すべてorg.ashikunep.ireнка.dom.CtElementのサブタイプとして実装されており、DOMの種類ごとにインターフェースが用意されています。これらのインターフェースはすべてIrenka End User APIとして公開されていて、ユーザは実装を気にせずDOMを取り扱うことが可能です。

DOM を操作してプログラムの変更すると、Irenka Studio は何らかの方法で元のプログラムに対して変更を反映させます。つまり、DOM 上の対応する部分を Hack Action 上で書き換えることで、目的のプログラムを書き換えることができます。下記のいずれかの方法で、DOM の部分を差し替えることができます。

1. 現在の要素をほかの要素に置き換える場合
 - 現在の要素に対して CtElement#substitute(CtElement) を実行
2. 子要素を保持するプロパティを変更する
 - 子要素に対して CtElement#substitute(CtElement) を実行
3. 子要素のリストを保持するプロパティの内容を変更する
 - 返されるリストの内容を直接変更
 - 返されるリスト内の要素に対して CtElement#substitute(CtElement) を実行

CtElementインターフェースはsubstituteというメソッドを公開していて、これを利用することで対象のDOM要素を指定したDOM要素で置き換えることができます。[図 83](#)はsubstituteメソッドを利用してDOMを変更する例です。

```

/**
 * @when
 */
public void apply(CtCast<?> cast) {
    CtExpression<?> expression = cast.getExpression();
    CtType<?> from = expression.evalType();
    CtType<?> to = cast.getType();
    if (from.isSame(to)) {
        cast.substitute(expression);
    }
}

```

図 83. substitute メソッドを利用して不要なキャストを除去する Hack

また、CtElementのサブインターフェースの中には、プロパティ(getXxx)としてDOM要素のリストを返す機能を持つものがあります。この返されるリストは変更可能で、リストに加えられた操作はDOMの構造にそのまま反映されます。図 84はブロック文(CtBlock)のプロパティstatementsが返す文のリストに変更を加え、ブロック文の中身を変更する例です。

```

package com.example;

import org.ashikunep.irenka.dom.CtMethod;
import org.ashikunep.irenka.dom.CtStatement;

public class Greetings {

    /**
     * @when
     *   stmt = {@link #tmpl()}.body.statements[0]
     */
    public void insert(CtMethod<?> method, CtStatement stmt) {
        CtBlock body = method.getBody();
        List<CtStatement> statements = body.getStatements();
        statements.add(0, stmt);
    }

    private void tmpl() {
        System.out.println("HELLO!");
    }
}

```

図 84. リストを操作して println("HELLO!")をメソッドの先頭に挿入する Hack

Irenka DOM についての詳しい内容は、Irenka DOM Specification を参照してください。

第5節 Tool オブジェクトの利用

ToolオブジェクトはIrenka Studioが提供するツールをHackから利用するためのインターフェースです。Toolオブジェクトのインターフェースを持つHack ParameterをHack Actionに追加することによって、Match Actionが起動する際にIrenka Studioから適切なインスタンスが自動的に提供されます⁶。1つのHack Action上にToolオブジェクトを受け取るHack Parameterを複数追加することもできますし、不要であれば一つも指定しなくてもかまいません。

全てのToolオブジェクトはorg.ashikunep.ireнка.toolkit.Toolインターフェースを実装していて、ツールの目的別に様々なインターフェースが用意されています。これらは、Irenka End User APIとして公開されていますので、個々の詳しい情報についてはIrenka End User API Referenceを参照してください。

第1項 Irenka DOM 要素を生成する Tool

ソースプログラム上に存在しないDOM要素をHack内で生成して利用するには、Toolオブジェクトを利用する必要があります。生成するDOM要素の種類に応じて、いくつかのToolが用意されています。

- org.ashikunep.ireнка.toolkit.LiteralFactory

LiteralFactoryは定数リテラル(org.ashikunep.ireнка.dom.CtLiteral)、及びクラスリテラル(org.ashikunep.ireнка.dom.CtClassLiteral)を生成できます。図 85はLiteralFactoryを利用してブールリテラルfalseを生成する例です。

```
/**
 * @when
 *     stmt.condition = true
 */
public void apply(LiteralFactory tool, CtIf stmt) {
    CtLiteral<Boolean> falseValue = tool.of(false);
    stmt.getCondition().substitute(falseValue);
}
```

図 85. if(true)をすべてif(false)に置き換える Hack

⁶ 現在はIrenka Builderに新しいToolを追加することはできません

- org.ashikunep.irenka.toolkit.DeclarationFactory

DeclarationFactoryは、新しい型宣言の生成や型のロード、メンバ宣言の生成などを行います。図 86はDeclarationFactoryを利用してjava.awt.Pointクラスをロードする例です。

```
/**
 * @when
 *   clazz.superClass = {@link Object}
 */
public void apply(DeclarationFactory tool, CtClass<?> clazz) {
    CtClass<Point> pointClass = tool.classOf(Point.class);
    clazz.getSuperClass().substitute(pointClass);
}
```

図 86. POJO に java.awt.Point を継承させる Hack

- org.ashikunep.irenka.toolkit.JavadocFactory

JavadocFactoryはJavaドキュメンテーションコメントに関するDOMを生成するツールです。Javadocを表現するorg.ashikunep.irenka.dom.CtJavadocインスタンス自体はElementFactoryを利用して生成しますが、その内部構造はJavadocFactoryを利用して生成する必要があります。図 87はJavadocFactoryを利用して"@irenka Irenka was here"というブロックを生成するHackの例です。

```
/**
 * @when
 *   clazz = {@link CtClass}
 *   javadoc = clazz.javadoc
 */
public void apply(JavadocFactory tool, CtJavadoc javadoc) {
    DocBlock block = tool.newBlock("@irenka Irenka was here");
    javadoc.getBody().getBlocks().add(block);
}
```

図 87. クラスの Javadoc に"@irenka"ブロックを追加する Hack

- org.ashikunep.irenka.toolkit.ElementFactory

ElementFactoryは文や式、注釈や修飾子など他のDOMファクトリで生成できない要素を生成するツールです。図 88はElementFactoryを利用して修飾子strictfpを生成するHackの例です。

```

/**
 * @when
 *     double = method.returnType
 */
public void apply(ElementFactory tool, CtMethod<?> method) {
    final ModifierKind kind = ModifierKind.STRICTFP;
    if (!method.getModifiers().contains(kind)) {
        CtModifier strict = tool.newModifier(kind);
        method.getModifiersAndAnnotations().add(strict);
    }
}

```

図 88. double を返すメソッドに strictfp を付与する Hack

第2項 環境を操作する Tool

環境を操作するツールは、開発環境上のリソースや開発環境そのものを操作するためのツールです。

- org.ashikunep.irenka.toolkit.Filer

Filerは、ワークスペース上のファイルやフォルダを操作するためのツールです。getFolderやgetFilerでワークスペースルートからの相対パスを記述することで、対応するリソースを表すオブジェクトを取得することができます。図 89はFilerを利用してhack-test/tempというフォルダを取得する例です。

Irenka Studioでは、ワークスペースのルートはEclipse上のワークスペースと一致し、その直下にはプロジェクトを表すフォルダがあります。図 89の例では、hack-testプロジェクトのtempフォルダを利用しています。

```

/**
 * @when
 */
public void apply(Filer tool, CtClass<?> cls) throws Exception {
    CtFolder temp = tool.getFolder("hack-test/temp");
    CtFile file = temp.getFile(cls.getName());
    PrintWriter out = new PrintWriter(file.openOutputStream());
    try {
        out.write(clazz.getMembers().toString());
    }
    finally {
        out.close();
    }
}

```

図 89. クラスのメンバー一覧を hack-test/temp/<クラス名>に出力する Hack

- org.ashikunep.irenka.toolkit.Messenger

Messenger はコンパイル環境のメッセージ通知機構を利用して、ユーザに様々なメッセージを提供するツールです。メッセージは debug, info, warn, error の 4 種類があり、環境に適した方法でメッセージが通知されます。

Irenka Studio での Messenger の実装は Problems タブに情報を出力します。debug は無視されるので注意が必要です。

```

/**
 * @when
 *     stmt.condition = false
 */
public void apply(Messenger tool, CtIf stmt) {
    tool.warn(stmt, "無効なif");
}

```

図 90. if (false)に対して警告を行う Hack

第3項 コンパイラの機能を提供する Tool

コンパイラの一部の機能を切り出したツールも公開されています。テンプレートエンジンを Hack として実現する場合や、独自の方法でソースコードの分析を行うような Hack を作成する場合には利用することができます。

- `org.ashikunep.irenka.toolkit.SourceParser`

SourceParser はファイルを解析して Irenka DOM の要素に変換するツールです。Filer と組み合わせて利用します。

- `org.ashikunep.irenka.toolkit.SourceEmitter`

SourceEmitter は Irenka DOM をソースプログラムとしてストリームに書き出すツールです。コンパイル単位を表す要素、または型の宣言を表す要素をソースプログラムの形式に変換することができます。

- `org.ashikunep.irenka.toolkit.ConstantConverter`

ConstantConverter は定数として表現できる実行時の値を Irenka DOM に変換するツールです。LiteralFactory は対象が定数リテラルのみですが、ConstantConverter は定数リテラルに加えて注釈、列挙定数、クラスリテラル、およびこれらの一次元配列を Irenka DOM に変換します。これらはいずれも注釈要素に指定する値として利用できます。

- `org.ashikunep.irenka.toolkit.LiteralConverter`

LiteralConverter は定数リテラルを表現する文字列と、その実行時の値を相互に変換するツールです。

- `org.ashikunep.irenka.toolkit.TypeConversions`

TypeConversions は Java 言語仕様が規定する型の変換アルゴリズムを提供するツールです。Irenka DOM の型要素が提供する機能よりも複雑な検査や変換を行うことができます。

第6節 イベントオブジェクトの利用

イベントオブジェクトは、Irenka Studio 内で発生した何らかのイベントの情報を保持するオブジェクトです。イベントオブジェクトのインターフェースを持つ Hack Parameter を記述することによって、それぞれのイベントの発生を感知して、適切な Hack Action を起動することができます。

全てのイベントオブジェクトは `org.ashikunep.irenka.event.CtEvent` インターフェースを実装していて、イベントの種類に応じてそれぞれサブインターフェースが用意されています。また、CtEvent は Irenka DOM の基底インターフェースである CtElement インターフェースのサブインターフェースであり、イベントの発生中は DOM の一部であるかのように振る舞います。このため、Irenka DOM の構造について分析を行う Irenka Search Query によってイベントオブジェクトに対しても制約を掛けることができ、特定のイベントだけを検出して受信することが可能です。図 91 は、Hack の開始時と終了時に発生する HackEvent のうち、終了時に発生したイベントオブジェクトだけを受け取る Hack Action の例です。

```

/**
 * @when
 *     event.phase = {@link HackPhase#FINALIZE}
 */
public void onFinalize(HackEvent event) {
    // Hackの終了処理
}

```

図 91. Hack の終了イベントだけを受信する例

第1項 イベントの発生と Hack Action 実行のタイミング

Hack ParameterにCtEvent (イベントオブジェクト) 型であるものを持つHack Actionは、通常のHack Actionとして扱われず、Event Handlerとして取り扱われることとなります。Event HandlerとはHackの実行中にCtEvent型のイベントオブジェクトを受け取って実行されるHack Actionのことで、それぞれのイベントが発生した際の処理を記述することができます。また、Event Handlerには通常のHack Actionと同様にSearch Queryを記述する必要があります。すべてのイベントオブジェクトはクエリ内でDOM要素と同様に扱うことができるため、前掲の図 91のように選択的にEvent Handlerを実行させることができます。

Event Handlerには、受信したいイベントオブジェクトの型をHack Parameterで指定します。ただし、イベントは同時に 1 種類しか発生しませんので、CtEvent型またはそのサブタイプ型のHack Parameterを 2 つ以上指定すると、そのEvent Handlerはイベントを受信することができなくなってしまいます。Tool型やCtElement型のHack Parameterを混在させることは可能です(図 92)。なお、Event Handlerでない通常のHack Actionは、イベントが発生していない場合に起動されることはありません。逆に、Event Handlerはイベントが発生していない場合に起動されることはありません。

```

/**
 * @when
 *     event.phase = {@link HackPhase#INITIALIZE}
 */
public void onFinalize(HackEvent event, Messenger msgr) {
    Hack target = event.getContext().getTarget();
    msgr.info(target + "を開始します");
}

```

図 92. Hack の開始時に情報を表示させる例

ある Event Handler が CtEvent 型のサブタイプである *E* という型の Hack Parameter を持つ場合、その Event Handler を「*E* 型の Event Handler」と呼びます。*E* 型の Event Handler は *E* 型、またはそのサブタイプ型を持つイベントオブジェクトで表現されたイベントを潜在的に受信することができます。

以下に、イベントの発生時に適切な Event Handler が選択されて実行される流れを示します。Event Handler は Hack Action の選択方法と検索対象集合が異なる以外は、通常の Hack Action と同様の手続きで起動されます。

1. イベントが発生し、その際に x という X 型のイベントオブジェクトが生成された場合、Irenka は次のようにして適切な Event Handler を実行します。
2. Hack に含まれる Event Handler のうち、 X 型のオブジェクトを潜在的に受信可能な Event Handler の集合 H を計算する
3. Irenka Search Query の検索対象集合にイベントオブジェクト x を加えた集合 T を計算する
4. H に含まれる Event Handler のそれぞれが持つ Search Query で、 T を検索する
5. Event Handler が持つ Hack Parameter に検索結果を設定し、Match Action を実行する

第2項 ビルドの実行に関するイベント

Irenka Builder は、ユーザがプロジェクト内のリソースを変更した際にビルドプロセスを実行しますが、その情報として `org.ashikunep.irenka.event.BuildEvent` を生成します。このオブジェクトは、ビルドが発生した理由として変移のあったリソースの情報を保持しています。

第3項 Hack の実行に関するイベント

何らかの Hack が適用される際、その Hack に含まれるすべての Hack Action の実行に先立って `org.ashikunep.irenka.event.HackEvent` オブジェクトが生成され、このオブジェクトをハンドル可能な Hack Action が実行されます。これは Hack 初期化イベントと呼び、その際の HackEvent オブジェクトは、`phase` プロパティに `HackPhase.INITIALIZE` という値が格納されています。

同様に、同一 Hack 内で実行可能なすべての Hack Action の実行が終わった後、Hack 終了イベントとして `org.ashikunep.irenka.event.HackEvent` オブジェクトが生成され、適切な Hack Action が実行されます。Hack 終了イベントの HackEvent オブジェクトは、`phase` プロパティに `HackPhase.FINALIZE` という値が格納されています。

第7章 Hack Packager

Hack Packager は作成した Hack を Hack Library として配布可能な形式に変換するツールです。作成した Hack Library は Irenka Builder に登録することによって、作成した Hack が他のプロジェクトからも利用可能になります。

この章では、Hack Packager の利用方法を紹介します。

第1節 Hack Library の作成

Hack Packager は Hack を含むプロジェクトをパッケージングし、Hack の配布可能形式である Hack Library を作成します。Irenka Studio に含まれる Hack Packager は通常の Eclipse の Export Wizard として提供しており、数個のステップで作成した Hack を配布可能な形式に変換できます。

ここでは、第 5 章第 2 節で作成したプロジェクト "hack" を Hack Library へ変換する手順を紹介します。

まず、変換対象のプロジェクトを右クリックし、Export... を選択します(図 93)。

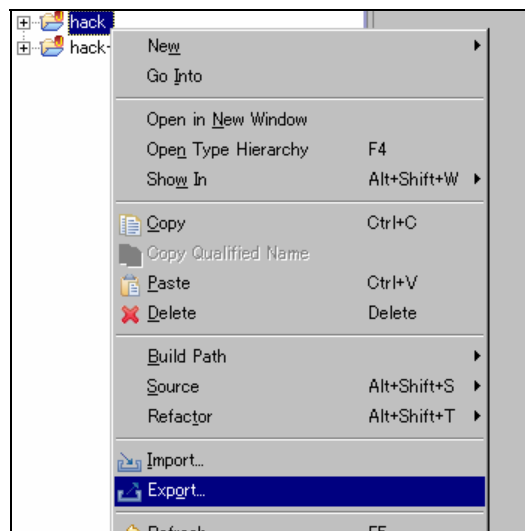


図 93. "hack"プロジェクトのエクスポート

次に、Export方法を選択するダイアログが表示されますので、その中から Irenka > ハックライブラリ を選択します(図 94)。

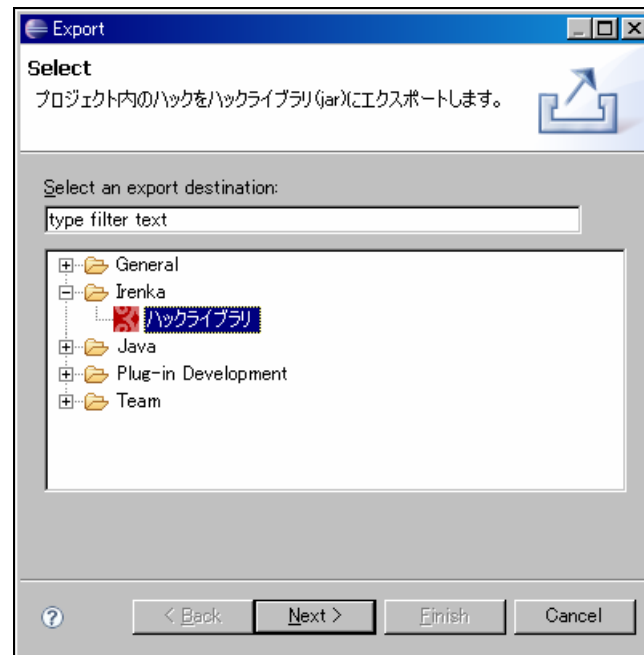


図 94. Hack Library Wizard

次に、Hack Libraryを作成する元となるプロジェクトを選択します(図 95)。図 93のようにプロジェクトを右クリックしてExport...を選択している場合、ここではすでに対象のプロジェクトが選択されています。今回は"hack"プロジェクトを元にHack Libraryを作成するため、"hack"がチェックされていることを確認した後にNextボタンを押下します。

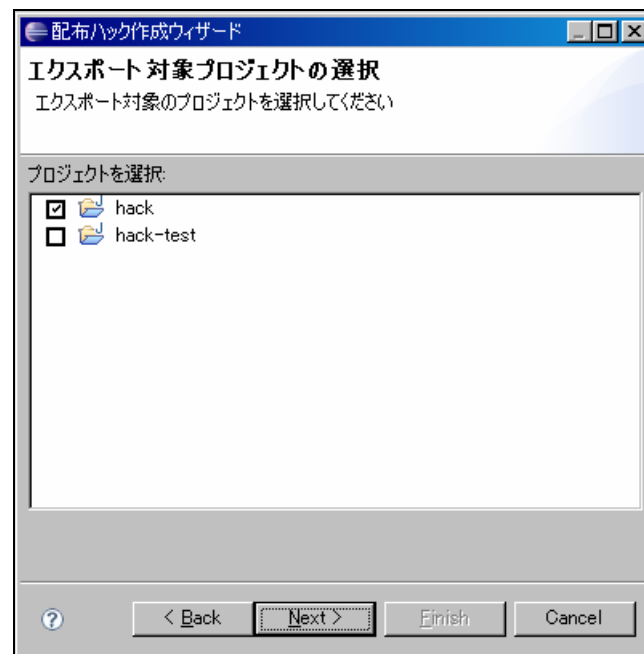


図 95. エクスポート対象プロジェクトの選択

次に、Hack Libraryに含めたいHackの一覧と、Hack Libraryファイルの出力先を選択します。ここでは、DetectEmptyParamsのみからなるHack Libraryを作成することにし、出力先は仮に"C:\tmp\sample.jar"とします(図 96)。なお、Hack Libraryは一つ以上のHackを含む必要があります。ここで選択したHackはHack定義ファイル(第 2 節第 2 項)としてHack Library内に格納されます。

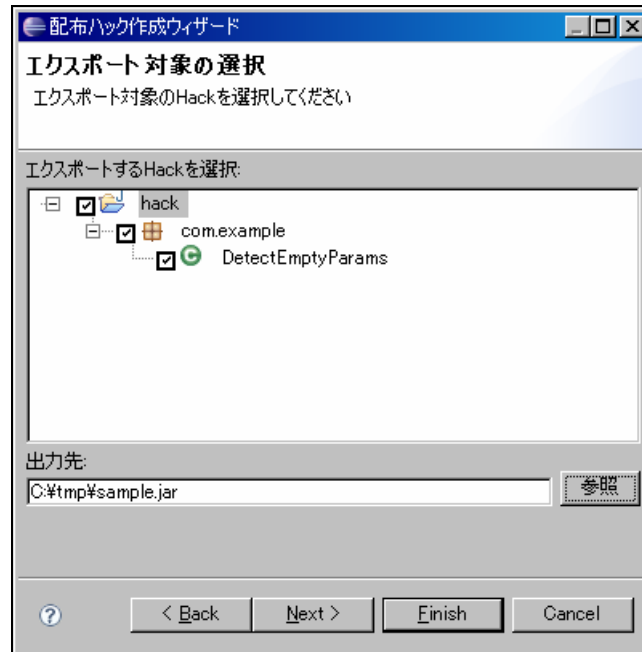


図 96. エクスポート対象 Hack の選択

最後に、Hack Libraryに含める追加リソースを選択します。HackがリソースファイルやJavaソースコードテンプレート(第 6 章第 4 節)を利用する場合には、それらのリソースファイルやJavaソースプログラムを追加リソースとしてHack Libraryに含める必要があります。DetectEmptyParamsはどちらも利用しないため、チェックを入れずにFinishボタンを押下します(図 97)。ここで選択したリソースは、追加リソース(第 2 節第 3 項)としてHack Library内に格納されます。

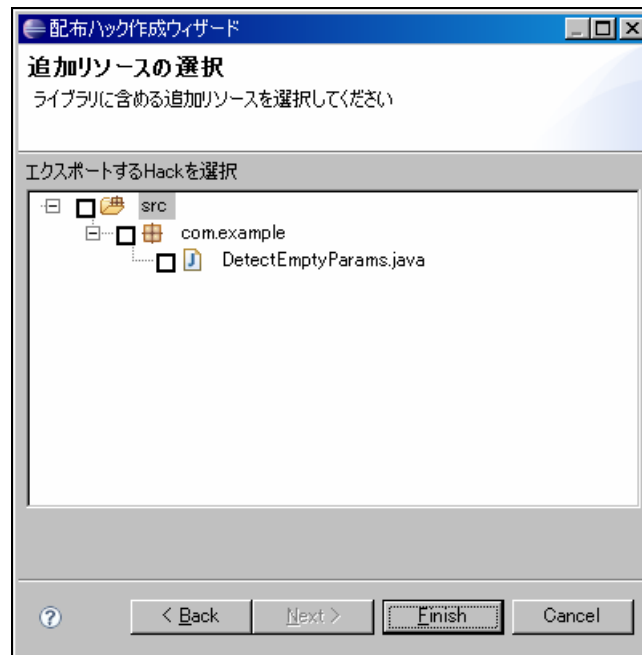


図 97. 追加リソースの選択

ここまででHack Libraryの作成は完了です。Hack PackagerはHackをコンパイルし、Irenka Builderから利用可能なHack定義ファイル(第2節第2項)へと変換します。その際、変換に多少の時間を要する場合があります(図 98)。

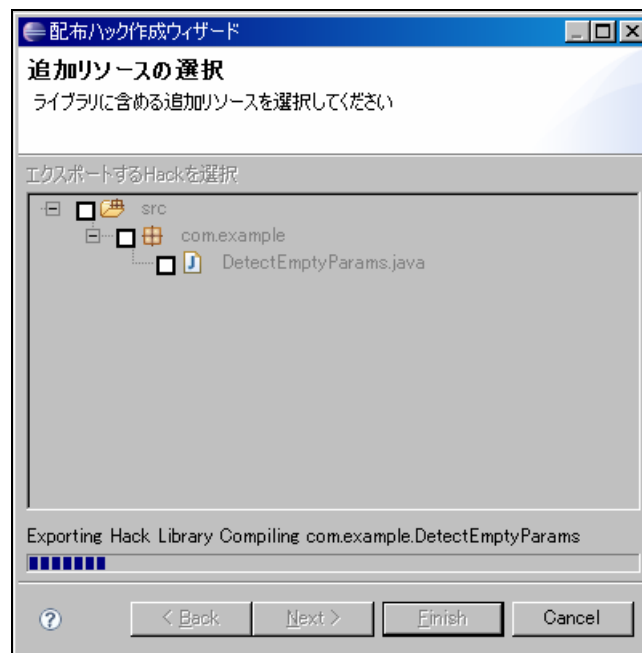


図 98. エクスポート中のダイアログ

変換に成功すると、Export Wizardのダイアログが消え、Hack Libraryファイルが保存先に指定したパス(図 96)上に作成されます(図 99)。

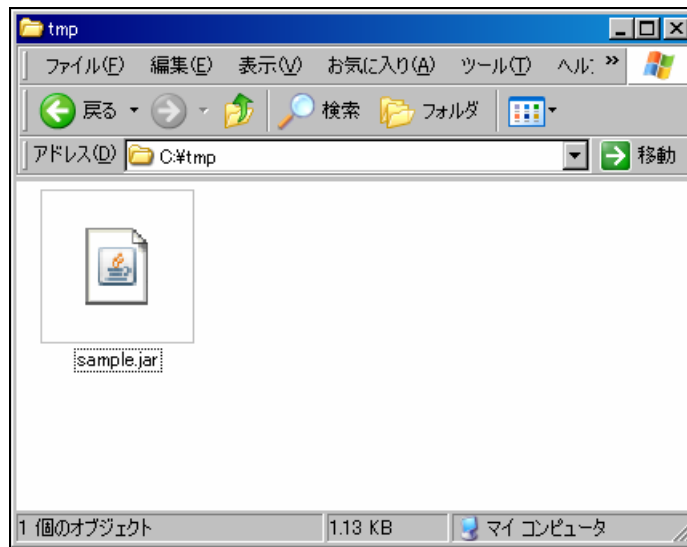


図 99. 作成された Hack Library (JAR ファイル)

この節で作成した Hack Library は、第 3 章第 1 節にある方法で Irenka Builder に登録し、ほかのプロジェクトで利用することができます。

第2節 Hack Library のしくみ

Hack Packager が生成する Hack Library は、通常の Java ARchive (JAR) を拡張し、Hack Library 特有のメタ情報を保持しています。

第1項 Hack Library の構造

Hack Library は JAR ファイルにいくつかのメタ情報を追加したもので、通常のクラスライブラリを格納したアーカイブファイルとしても利用できます。図 100 のような構造を持ち、それぞれの領域には下記のような情報が配置されます。

- 通常の Java Archive 領域

Java のクラスローダが利用する情報が配置されています。主に、クラスファイル、リソースファイル、メタ情報(META-INF/)などがあります。

- Hack 定義ファイル

Irenka Builder など、Hack を解釈するツールが利用する Hack 定義ファイル(*.hack)が配置されています。HACK-INF/hack/ 以下に作成され、同エントリを含む JAR を Irenka は Hack Library と認識します。

- 追加リソース

Irenka DOM を生成するローダが利用する情報が配置されています。任意のファイルやフォル

データをこの領域に含めることができます。HACK-INF/resources/ 以下に作成されますが、必須ではありません。

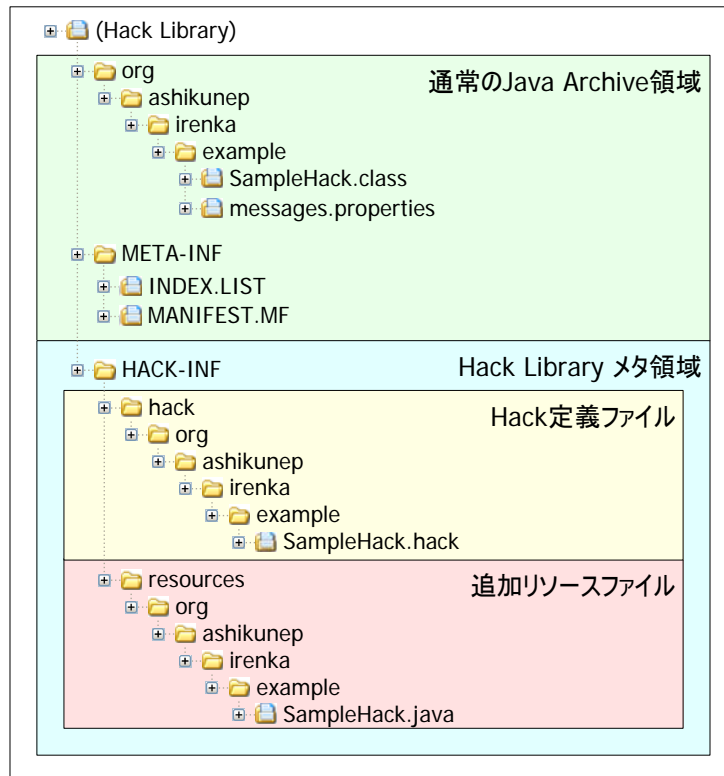


図 100. Hack Library の構造

第2項 Hack 定義ファイルの構造

Hack 定義ファイルは、Hack を通常の Java プログラムとしてコンパイルした際に、ソースコードから失われてしまう下記のような情報を保持しています。

- ソースコード全体の import 宣言
- Irenka Search Query の内容
- メソッドの引数名
- その他 Hack に関するメタ情報

Hack 定義ファイルは 1 ファイルで Hack1 つ分を表現し、下記の 3 つの規則で Hack Library 内の適切な位置に格納されます。

1. Hack 定義ファイルは/HACK-INF/hack/を起点として、Hack を実現するクラスが属するパッケージと同じ構造のフォルダの下に格納される

2. Hack の宣言を表すファイルは、対象の Hack が持つ単純名の末尾に".hack" を付与されたファイルとして上記のフォルダに格納される
3. Hackの宣言は、UTF-8 でエンコードされたテキストファイルで、その形式は図 101. Hack 定義ファイルの文法で定義される

HACK-DEFINITION	::=	HACK-HEAD HACK-ACTION+
HACK-HEAD	::=	'@class' QUALIFIED-NAME
HACK-ACTION	::=	ACTION-HEAD ACTION-ATTRIBUTE* QUERY ACTION-TAIL
ACTION-HEAD	::=	'@action' NAME
ACTION-ATTRIBUTE	::=	'@import' QUALIFIED-NAME '@param' NAME DECLARED-TYPE
QUERY	::=	QUERY-HEAD QUERY-LINE+
QUERY-HEAD	::=	'@query' QUERY-OPTION-LIST?
QUERY-OPTION-LIST	::=	'[' QUERY-OPTION ']'
QUERY-OPTION	::=	NAME '=' STRING-LITERAL
ACTION-TAIL	::=	'@endaction' NAME?
QUALIFIED-NAME	::=	NAME ('.' NAME)*
QUERY-LINE	::=	':' ~('¥r' '¥n')* ('¥r¥n' '¥r' '¥n')
NAME	::=	(<i>Identifier</i> [JLS3-3.10.5])
DECLARED-TYPE	::=	(<i>ClassOrInterfaceType</i> [JLS3-4.3])
STRING-LITERAL	::=	(<i>String Literal</i> [JLS3-3.8])

図 101. Hack 定義ファイルの文法

第3項 追加リソースファイルの扱い

追加リソースファイルは、Irenka DOM を構築するローダが利用するリソースの領域です。Irenka は Hack Library を検出すると、そこに含まれる HACK-INF/resources/エントリをビルドパスと同等に扱い、通常の Java Archive 領域に配置されたクラスに優先されてリソースをロードします。Java ソースプログラムを配置すれば、Irenka はコンパイル前の情報を失うことなく、完全な DOM を構築することができます。

Hack Library に含まれる Hack がライブラリ内のソースコードを利用する場合、それらのソースコードは追加リソース領域に配置する必要があります。たとえば、Hack が自クラス内のメソッドを DOM のテンプレートとして利用する場合、それらはコンパイルしてクラスファイルの形式になった際にはそれらの情報が除去されてしまっています。Hack Library として配布する場合は、必要なソースコードを追加リソースとして適切な位置に配置しなければなりません。

第8章 その他の情報

第1節 関連ドキュメント

より詳しいドキュメントは、下記の URL にあります。必要に応じて参照してください。

- Irenka Search Query Specification
 - <http://irenka.ashikunep.org/documents/spec/query.pdf>
- Irenka DOM Specification
 - <http://irenka.ashikunep.org/documents/spec/dom.pdf>
- Irenka End User API References
 - <http://irenka.ashikunep.org/documents/api>

第2節 サンプル集

本ドキュメントより多くのサンプルが公式ページ上で公開されています。そちらも参照してください。

- <http://irenka.ashikunep.org/examples/>

第3節 CARNIVAL

準備中

第9章 謝辞

Irenka および Irenka Studio は、独立行政法人 情報処理推進機構 未踏ソフトウェア創造事業の支援を受けて開発しています。