Eclipse と AJDT でアスペクト指向の Java アプリケーションを作成する

初心者のみならず熟練者もアップデートされたツールで AOP を簡単にできる

Matt Chapman
AJDT contributor
IBM

Helen Hawkins
AJDT contributor
IBM

2004年 9月 21日

AspectJ は、クロスカット・コンサーン (crosscutting concerns) のモジュラー実装を可能にする Java™ 言語のアスペクト指向拡張です。静的でも動的でもあり得るこのクロスカットの振る舞いは、AspectJ をサポートするツールに更なる難題を提供します。AJDT プロジェクトは、(Eclipse の Java Development Tools (JDT) に従うかたちで) アスペクト指向のアプリケーションの持つクロスカットの本質を明瞭化し理解する追加機能とともに、AspectJ 用の Eclipse プラットフォームをベースにしたツールの提供を試みます。この記事では、AJDT の貢献者であり IBM アスペクト指向ソフトウェア開発チームのメンバーでもある、Matt Chapman 氏そして Helen Hawkins 氏が AJDT を読者に紹介します。ツールのインストール方法、AspectJ アプリケーションの作成・実行・デバッグ、そしてアスペクト指向プログラミングに固有のクロスカットの構造を視覚化そしてナビゲートする方法について学びます。

EclipseのAspectJ Development Tools（AJDT）は、AspectJ アプリケーションを作成そして実行するうえで必要なツールを提供するオープン・ソースのEclipse Technology Project です。良質のツールがアスペクト指向プログラミングの完全な利益（特に新参者の関連する概念に対する理解の支援）を実現する上で重要な役割を担うと確信を持っています。

この記事を読まれた後に、読者の皆様がAspectJを使用するうえでAJDTが提供するサポートに対する理解を深めることを願っています。まずAJDTをどのようにして立ち上げて実行するかを紹介します。それから題目に深く入り込み、AspectJアプリケーションを無から作成します。どのようにしてアスペクトとJavaクラスを作成し、どのようにしてアプリケーションを実行するかを理解することでしょう。作成を助けるためにAJDTが提供するビジュアルそしてナビゲーションの異なる機能を紹介します。ツールへの焦点を保つために、最初の例はとても簡単です。簡単であっても、

この簡単なアプリケーションは実世界でのアスペクトの使用法の側面のうちのひとつを説明します。

そのあとに、より詳細にわたりツールを探究できるように多少複雑な例に移り、現存のプロジェクトとどのように関わるべきかを示します。（アプリケーションを実際にデバッグするのであれば）包括することのみを望むデバッグのアスペクトのみならず、アプリケーションに不可欠な複数のアスペクトをその例は含みます。このデバッグのアスペクトを選択的に応用そして除去することを、どのようにAJDTが可能にするかを示します。そのあとに、多数の異なるアスペクトがアプリケーションの別の部分にどのようにして影響をおよぼすかについてのより高レベルな理解を得るためにも、強力な Visualization パースペクティブを見に行きます。最後に、（AspectJ アプリケーションのデバッグそして文書作成を含む）ツールのより先進的な機能に触れます。

この記事を紹介するにあたり、読者が基本的にAspectJ とアスペクト指向プログラミング（AOP、Aspect-Oriented Programming）にある程度は慣れていることを前提とします。AOPそしてAspectJのプログラミングに関する入門的な資料をお求めでしたら、参考文献を参照してください。別の手段として、AJDTをインストールしてからHelpのリンクを介してAspectJに行き、AspectJのドキュメントを見付けます。

AJDT をインストール

この記事を理解するには、（Eclipse 3.0（最新版）を必要とする）バージョン1.1.11以降のAJDTを実行する必要があります。Eclipse 2.1と一緒に機能するAJDT の過去バージョンもありますが、この記事で説明する機能の多くはバージョン1.1.11に新たに追加されました。

Eclipse 3.0のために最新のAJDTをインストールするには、次のステップを踏みます。

1. Eclipseを起動し、Help>Software Updates>Find and Install...の順番で移動する。
2. インストールする新機能を探し、URL（http://download.eclipse.org/technology/ajdt/30/update）を使い、新規リモート・サイト（AJDT Update Site）を追加する。Internetにアクセスするのにプロキシー・サーバーを必要とする場合は、Window>Preferences>Install/Updateの順番で移動してプロキシー設定を設定する。
3. AJDT Update Siteのノードを展開し、AspectJを選択する。Eclipse のAspectJ Development Tools のインストールを選択し、著作権文(copyright statement)を読んで同意する。
4. Finishをクリックする。

バージョン1.1.11の時点では、AJDT はEclipse 3.0 のフレームワークに完全に統合されていますので、それが適切にインストールされているかどうかを確認する方法がいくつかあります。この中でも最も役立つのは、AJDTの ドキュメントの（Eclipse にあるhelp そしてEclipse Platform にあるwelcome のページとの）統合です。AspectJ 1.2 のドキュメントそしてAJDTのhelpの両方をナビゲートするには、Help>Help Contentsの順番で選択します。AJDTの追加物を見るには、Help>Welcomeの順番で選択してから Overview、Tutorials、Samples、またはWhat's Newのリンクのうちの1つを選びます。特に、Tutorials>Build a simple AspectJ application のリンクをたどればSimple AspectJ Applicationcheat sheetを呼び出す事ができます。cheat sheetはEclipse 3.0の新機能で、AspectJ cheat sheet はJava cheat sheetにて詳細にわたり述べられるJavaの「Hello World」アプリケーションを拡張します。（Help>Cheat Sheets の順番をたどり、求めているcheat sheetを選択してアクセスすることも可能です。）

第一段階

AJDTをインストールしましたので、AspectJ アプリケーションの作成と実行をどのようにしてサポートするかを探究しましょう。そうするには、1つのアスペクトと1つのクラスのみから成る簡素なAspectJ アプリケーションを作成することにより、それを達成するのに必要なAJDT の機能を調べます。そのアプリケーションは多少不自然ですが、目的を果たすにはそれで十分です。それを使い、2つの数字の二乗を単に計算します。最初は、プログラムが数字を与えられるたびに、その数字の二乗を最初から計算します。このメソッドに`Thread.sleep()`呼び出しを追加して、時間が掛かるようにします。しかしながら、キャッシュのアスペクトを追加することにより、与えられた入力に対して結果がすでに計算されているかどうかをチェックできます。もしもそうだとすれば、キャッシュ値は戻されます。もしもそうでなければ、通常どおりに二乗は計算され結果はキャッシュに追加されます。かくして、キャッシュのアスペクトはアプリケーションのパフォーマンスの向上につながります。（キャッシュのためにアスペクトを使用することに関する論考をお求めでしたら、参考文献にあるAspects Blog のリンクをたどってください。）

Javaの作成にJDT Eclipse のツールを使用したことがあれば、AspectJ の作成にAJDT を使用するのには簡単に慣れることでしょう。新規のAspectJ プロジェクトを作成するには、次のステップを介してください。

1. Window>Open Perspective>Java の順番で選択し、Javaパースペクティブを開く。
2. File>New>AspectJ Project の順番で選択し、New AspectJ Projectウィザードを開く。
3. Caching Exampleのプロジェクトを呼び出し、Nextをクリックし、そしてsrcと名付けられたソース・フォルダーを追加する。
4. ワーク・スペースにて新規のプロジェクトを作成するには、Finish をクリックする。

注釈: もしもこれがワーク・スペースにて作成した初めてのAspectJ プロジェクトでしたら、AJDT Preferences Configurationウィザードはステップ4の後に現われます。人生を楽にしてしまうほどに、このウィザードはEclipse 設定を簡単にします。ウィザードのデフォルトを受け入れ、Finishをクリックします。

Package Explorer の中にあるプロジェクトのノードを展開すれば、JREシステム・ライブラリーと同様にAspectJ ランタイム・ライブラリー（aspectjrt.jar）が追加されることがわかります。ここで、プロジェクトを選択してFile>New>Packageの順番をたどって`caching`と呼ばれるパッケージを作成し、File>New>Class を順番にたどって`Application` と呼ばれるクラスを`caching`のパッケージの中に作成します。これはプロジェクトがJavaプロジェクトであるかの様です。

そしてリスト1のようにクラスを編集します。

リスト 1. アプリケーションの主要なクラス

```
package caching;

public class Application {

    public static void main(String[] args) {
        System.out.println("The square of 45 is " + calculateSquare(45));
        System.out.println("The square of 64 is " + calculateSquare(64));
        System.out.println("The square of 45 is " + calculateSquare(45));
        System.out.println("The square of 64 is " + calculateSquare(64));
    }

    private static int calculateSquare(int number) {
        try {Thread.sleep(7000);}
        catch (InterruptedException ie) {}
        return number * number;
    }
}
```

リスト1のコードから、`calculateSquare()`メソッドへの呼び出しを4回実行する`main()`メソッドがクラスにある（最後のメソッド2つは最初のメソッド2つの繰り返しです。）のがわかります。`calculateSquare()` メソッドは少しの間何もせず、それから入力引き数の二乗を返します。

キャッシュのアスペクトを追加する前に、アプリケーションを実行しましょう。Javaプロジェクトでの Javaアプリケーションと同様に、アプリケーションを実行するには`main()` メソッドを含む（Package Explorer 内の）Javaクラスを選択する必要があります。この記事では、Application.javaが唯一のクラスです。このクラスを一度選択すれば、右クリックすることによりコンテキスト・メニューを開きRun>Java Applicationを選択します。二乗を計算するのに時間が結構かかることに気付かれることでしょう。

それでは、アプリケーションにアスペクトを追加しましょう。アスペクト作成のプロセスはJavaクラス作成のプロセスと同じ感触と見栄えを持っています。`Cache`と呼ばれるアスペクトを作成するには、File>New>Aspect をたどり新規のNew Aspect ウィザードを開きます。New Java Class ウィザードと同様に、アスペクトを作成するのに使うパッケージを選択し、それに命名し、そしてFinishをクリックします。Package Explorerにて、Cache.aj と呼ばれるファイルが作成されたことに気付いて下さい。.java（拡張子）のファイルにもアスペクトを入れることは可能ですが、デフォルトとしてアスペクトは.aj 拡張子と共にファイルにて作成されます。

リスト2のコードを含むように、Cache.aj を編集しましょう。

リスト 2. アプリケーションにキャッシュを追加するアスペクト

```
package caching;

import java.util.Hashtable;

public aspect Cache {
    private Hashtable valueCache;

    pointcut calculate(int i) : args(i)
        && (execution(int Application.calculateSquare(int)));

    int around(int i) : calculate(i) {
        if (valueCache.containsKey(new Integer(i))) {
            return ((Integer) valueCache.get(new Integer(i))).intValue();
        }
```

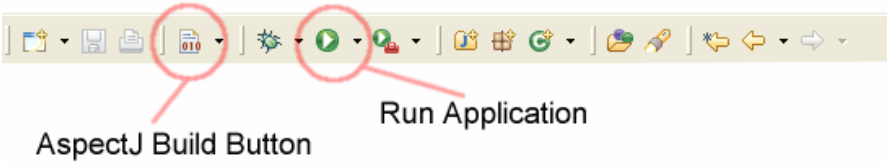
```
        int square = proceed(i);
        valueCache.put(new Integer(i), new Integer(square));
        return square;
    }

    public Cache() {
        valueCache = new Hashtable();
    }
}
```

リスト2のコードから、Cache アスペクトにて、最初に数字とそれらの二乗を記録するために hashtable を作成しているのがわかります。それからcalculateと言うポイントカットを指定します。整数を受け入れて戻すApplication.calculateSquare() と言う名のメソッドの実行と一致します。与えられた数字の二乗がまだ計算されていなければ単にApplication.calculateSquare() メソッドを実行したいだけなので、around advice を使います。advice は整数引き数を受け取り、整数値を戻します。advice の本体にて入力値がhashtableにあるかどうかをチェックします。あれば、対応する結果を戻します。なければ、calculateSquare()メソッドの実行に移行します。around advice から戻る前に、これらの新規の値でhashtable をアップデートします。

Eclipse がプロジェクトを自動的にビルドするように設定されているのであれば、AspectJ アプリケーションを実行するのみです。もしもそうではないのだとすれば、図1に示されるようにPackage Explorer 内のAspectJ プロジェクトを選択してBuild ボタンをクリックしてください。これで準備完了です。このアプリケーションの旧バージョンを既に実行しておりますので、実行の構成は全て設定されているはずです。このため、AspectJ アプリケーションを実行するには、タスクバー（ 図1 ）からRun ボタンを選択してからApplicationを選択します。2度目である今回においては、二乗を計算するのにはるかに少ない時間しか要しないことがおわかりでしょう。

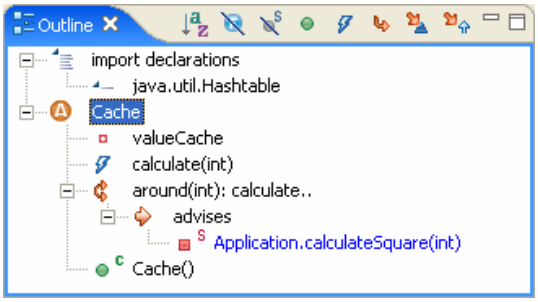
図 1. ビルドと実行のボタン



Outline ビューそしてエディター・マーカー

簡素なAspectJ アプリケーションを作成そして実行しましたので、どのようにしてAJDT がそれに対する理解と作業を支援するかに注目しましょう。エディターにあるCache アスペクトが開かれた状態で、（ 図2にあるような ）Eclipse Outline のビューをご覧になってください。

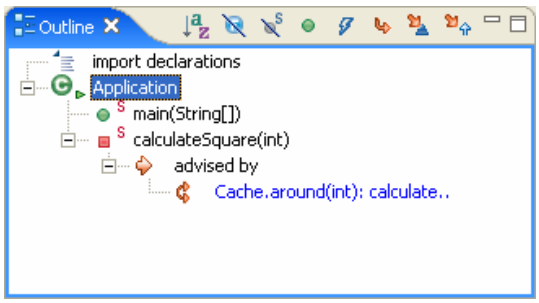
図 2. Outline ビューにあるCache アスペクト



Outline のビューは現行の文書の構造を示します。ここにある`Cache` アスペクトの場合、重要な宣言があり、それから`Cache`と呼ばれる（クラスの代わりに）アスペクトがあります。（それぞれがアスペクト内の機能に対応する）4つの異なるノードが`Cache` の直下にあります。そのうちで最初にあるのが、プライベート・フィールド（`valueCache`）です。（クラス同様、アスペクトもメソッドとフィールドを包括できることを忘れないでください。）次にあるのは、`calculate(int)`と名付けられた稲妻の形をしたアイコンであり、これは少々趣（おもむき）が異なります。これはポイントカットを意味し、クリックをすればエディターは適切な行にジャンプします。around advice を表記する次のノードは、さらに興味深いと言えます。このノードを展開すれば、`Application.calculateSquare(int)`と呼ばれる専用の静的メソッドに展開する「advises」ノードをお目にかかれます。ここでは、AspectJに展開されたOutline ビューは現行のソース・ファイル基本構造よりも奥深いものを見せ、（around advice の）構成とシステム内の別の構成との関係を示します。このメソッドの場合においては、このメソッドは advice されています。なお、この関係のターゲットはハイパーリンクです。単に`Application.calculateSquare(int)` ノードをクリックして、エディターはソース・ファイル（Application.java）に切り替えて、advice されるメソッドをハイライトすればいいだけです。

Application.javaがエディターで開かれておりますので、Outline ビューは、図3に示されるようになるはずです。

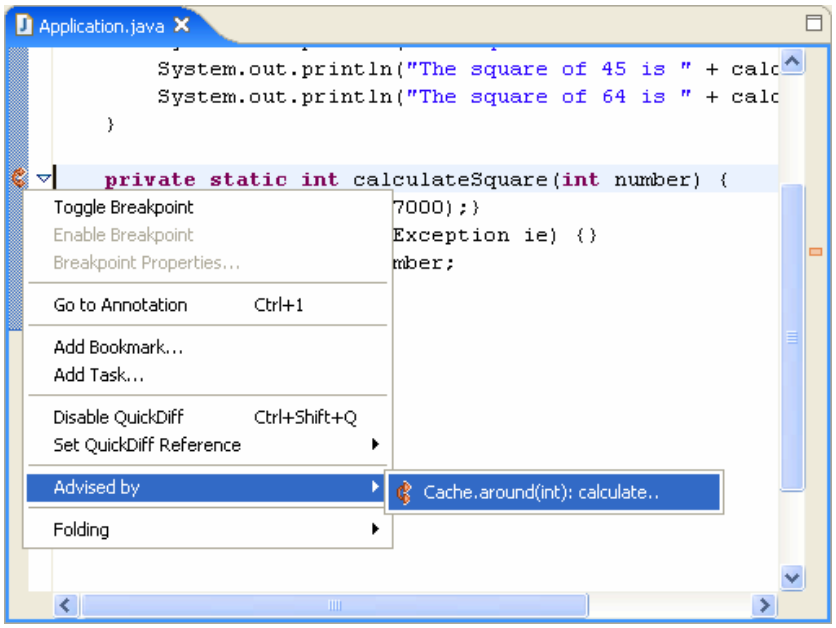
図 3. Application クラスを示すOutline ビュー



最初の時点では、このソース・ファイルのOutline ビューは、（AJDT無しで）普通のJavaの世界にて見かけるものとそっくりです。しかし、`calculateSquare()`メソッドのノードが展開されて「advised by」ノードを示してアドバイス（この場合、around advice は以前と一緒でありリンクでもある`Cache` アスペクト）のソースを披露します。

ここで`Application` クラスを表示するエディターそのものに戻りましょう。ウィンドウの左端（gutter）では、メソッド宣言（`calculateSquare()`）の隣に（Outline ビューにあるアイコンと共に）マーカーがあります。エディターの右側のバーには、同じメソッド宣言を記すオレンジ色のブロックがあります。両方とも何らかのアドバイスが影響を及ぼしている様子である証拠です。マウス・ポインターをそれらのインディケーターに移動すれば、`Cache` アスペクトにあるaround アドバイスにアドバイスされた位置をそれらが示していることを伝えるメッセージが表示されます。最後に、アドバイスのマーカーを右クリックして「Advised by」メニューを開いてから適切な advice に目を通します。図4に示されるとおり、advice を選択してナビゲートしましょう。

図 4. 「advised by」のメニュー



AJDTを使用して簡素なAspectJ プロジェクトのクロスカット構造を作成、実行、そして探究するのに必要なのは、たったこれだけです。

Spacewar の例

簡素な例でAJDT を紹介しましたので、ここで少々大掛かりなプロジェクトに関与するAspectJ プログラマーに提供するサポートを探究しましょう。Outline ビューを再訪し、アプリケーションから選択的にアスペクトを追加・削除し、そして最後に強力なVisualization パースペクティブを使用してプロジェクトの全体像をどのようにしてとらえるかを学びましょう。

現存の Java プロジェクトにアスペクトを追加

アスペクトを追加したい現存のJavaプロジェクトがあれば、それを簡単にAspectJ プロジェクトに変換できます。Package Explorerにて変換したいプロジェクトを単に選択し、右クリックをしてコンテキスト・メニューを表示させ、Convert to AspectJ Projectを選択します。同様に、何らかの理由でそれをJavaプロジェクトに戻したい（変換したい）のであれば、Package Explorer にてAspectJ プロジェクトを選択し、それからコンテキスト・メニューからRemove AspectJ Natureを選択します。

便利なことに、中核（コア）のAspectJ 配布物からの例、そして（ワークスペースにそれらを追加する作業を楽にする）ウィザードを、AJDT は含みます。Spacewar の例はその中でも大きな例であり、我々の目的に沿っています。（Spacewar の例が過去のバージョンに包括されていませんでしたので、この記事のこの部分でもバージョン1.1.11 以降のAJDT を必要とします。）それでは早速やってみましょう。

1. Eclipseにて、File>New>Otherの順番で移動する。
2. 結果として得られるダイアログにて、AspectJ フォルダーそれからAspectJ Examples フォルダーを展開し、Spacewar Exampleを選択する。
3. Next ボタンをクリックする。プロジェクトの名前を入力するか、デフォルトを受け入れることが可能です。

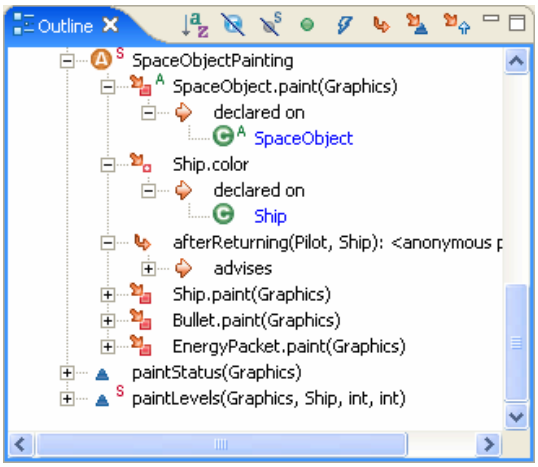
4. Finish をクリックすればNew Examples ウィザードがAspectJ プロジェクトを作成し、この例にて必要とされる全てのファイルをプロジェクトに追加し、そして（まだであれば）Javaパースペクティブに切り換わります。

もしもEclipse が自動的にビルドされるように設定されていれば、例もコンパイルされます。そうでなければ、既に簡素なアプリケーションにてそうしたように、自分自身でビルドを起動して下さい。

全てがうまく行っているかどうかをチェックして、それが実行されれば例がどうなるかを観察しましょう。例にある`main()`メソッドは、`spacewar`パッケージにあるGame.java にあります。それを起動するには、ファイルを選択し、Run>Run As>Java Applicationの順番で移動しましょう。ゲーム用のウィンドウが2枚表示されるはずです。カーソル・キーとスペース・バーを使用してゲームをコントロールします。そこから先は説明不要ですので、あとはおまかせします。

Outline ビューとエディター・マーカーが実行されている様子をより深く観察するために、Spacewar のコードを探求してみます。`Display1`クラスの内部アスペクトとして定義される`SpaceObjectPainting`に注目しましょう。図5にて示されるとおりに、エディター内のDisplay1.java を開き、Outline ビューにある`SpaceObjectPainting`アスペクトのノードを展開します。

図 5. Outline ビューにある SpaceObjectPainting アスペクト



以前に見かけたaround advice のアイコンとは異なる複数のアイコンがここにあります。最初のは抽象タイプ間の宣言（このアスペクトは`SpaceObject`クラスの代わりに`paint()`メソッドを定義します。）を表示します。ノードを展開することにより、それを観察できます。次にあるのは、ここでは`Ship.color`フィールドのためにある、別のタイプ間の宣言です。メソッドが正常に戻される（つまり、例外を投げ掛けない）場合に当てはまる匿名のポイントカット(pointcut)に宣言された advice を戻した後に来るのが、次のアイコンです。Outline ビューの中にある（そしてエディターの gutter にあるマーカーとしての）他のアイコンは、以下のとおりです。

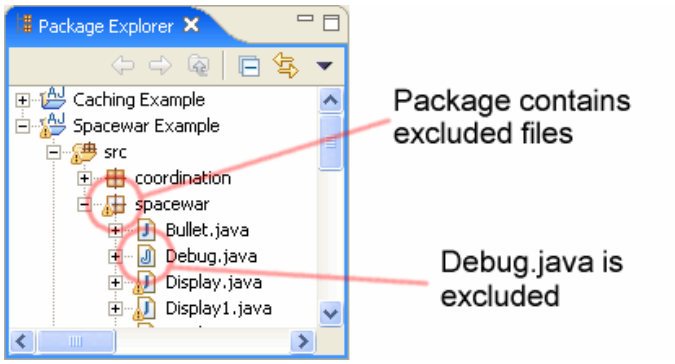
- アドバイス前 ()
- インターフェースの実装またはクラスの拡張のためにクラスが定義される親を宣言 ()
- 特定のポリシーが厳守されない場合にコンパイラー警告またはエラーをトリガーするときに使える警告()やエラー()を宣言する。

- チェックを入れられた例外がチェックを入れられない例外へとソフトになった場合、ソフト(),を宣言する。

ビルドの構成

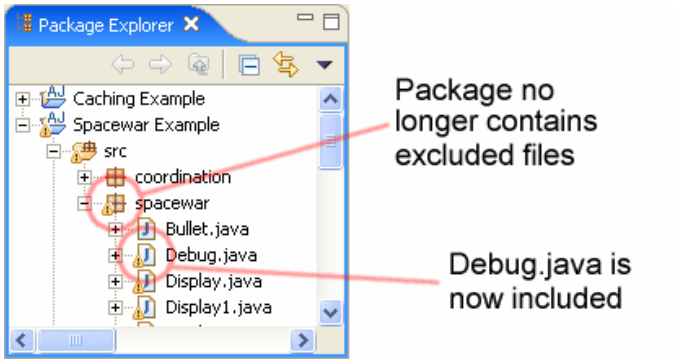
一部のアスペクトはシステムの必要不可欠な部分を担い、他のアスペクトは時々必要とされるデバッグのサポートのように特定の機能を提供します。そういうわけで、アプリケーションからアスペクトを選択的に取り除いたり追加したいと思うことでしょう。コンパイラーに受け渡されるプロジェクトの中のソース・ファイルを定義するbuild configurations（ビルドの構成）の概念を活用するその機能をAJDTは提供します。Eclipseの中にあるPackage Explorer からソース・ファイルを直接操作することを可能にすることにより、ツールはビルドの構成を楽にします。Spacewar の例には、最初からビルドの構成から除外されたデバッグのアスペクトがあります。Package Explorer に注目すれば、図6に示されるとおり、Debug.javaのアイコンがくり抜かれてその排他の状態を示しているのがわかります。それから、パッケージの（全てでは無く）一部が既に除外されていることを示すために、`spacewar`パッケージのアイコンが一部空になっていることに注意してください。

図 6. 除外されたリソースを含むPackage Explorer



単にDebug.java ファイルを右クリックしてメニューのオプション Include in "demo" configurationを選択することにより、プロジェクトにデバッグのアスペクトを追加できます。図7にて示されるとおりソースとパッケージのアイコンは満たされ、このファイルを包括するようにプロジェクトはビルドされます。（Eclipse のautobuilding が解除されていれば話は別ですが。）

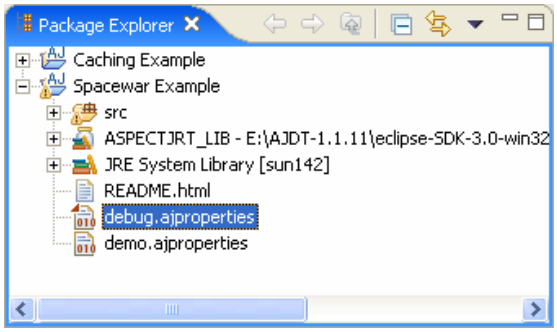
図 7. 全てを包括する Package Explorer



もう一度Spacewar プログラムを実行すれば、デバッグ関連の情報付きの新しいウィンドウを見かけることでしょう。

アスペクトを除外するには、再度ソース・ファイルを右クリックしてExclude オプションを選択します。シフト・キーやコントロール・キーを使えば、包括または除外するために複数のソース・ファイルを選択でき、全てのパッケージをも選択できます。このようにして加えた変更は現行の（または稼動中の）ビルドの構成に影響を与えますが、時には特定の構成を後の段階で使用するために保持するのが賢いと言えます。.ajproperties 拡張子付きのファイルにあるビルドの構成を保管することにより、それを実行できます。Spacewar コードを探すためにPackage Explorer の中を覗けば、図8に示されるとおり、demo.ajproperties そしてdebug.ajpropertiesと呼ばれるそのようなファイルがあるのに気付かれることでしょう。アイコンの1つには三角形が埋め込まれ、それがアクティブな構成であることがわかります。Include とExclude のコンテキスト・メニューのオプションで構成に加えられた変更は、現時点でアクティブな構成ファイルに書き込まれます。

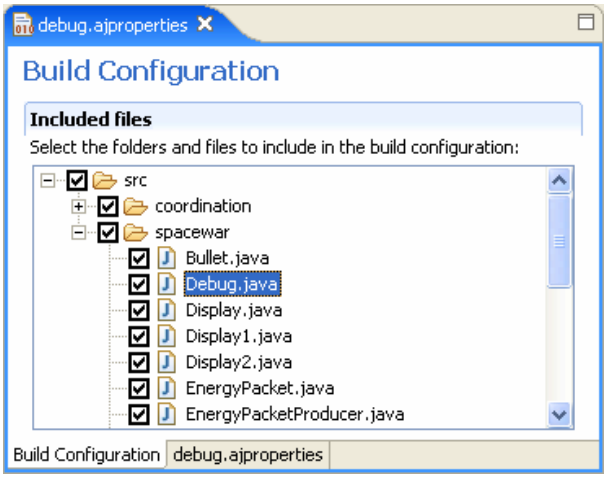
図 8. ビルドの構成ファイル



別のビルドの構成に切り換えるには、単に適切なプロパティー・ファイルを選択し、右クリックをし、コンテキスト・メニューから「Activate this configuration」を選択します。プロジェクトはそれから新規の設定で再構築されます。ビルドの構成間の切り換えを行なう別の方法は、現行のプロジェクトにて存在する全てのビルドの構成を含むサブメニューを表示するProject>Activate build configurationを選択します。このサブメニューには（新規ファイルに現行の構成を書き込むことにより新規のビルドの構成の作成を可能にする）Save as... のオプションもあります。

最後に、構成ファイルを直接編集できます。ビルドの構成ファイルをダブルクリックして、関連するビルドの構成エディターにてそれを開きます。図9に示されるとおり、この処置はプロジェクトのソース・ファイルのツリー表示（構成からリソースを除外したり構成にリソースを追加するためのチェック・ボックス付き）を表示します。

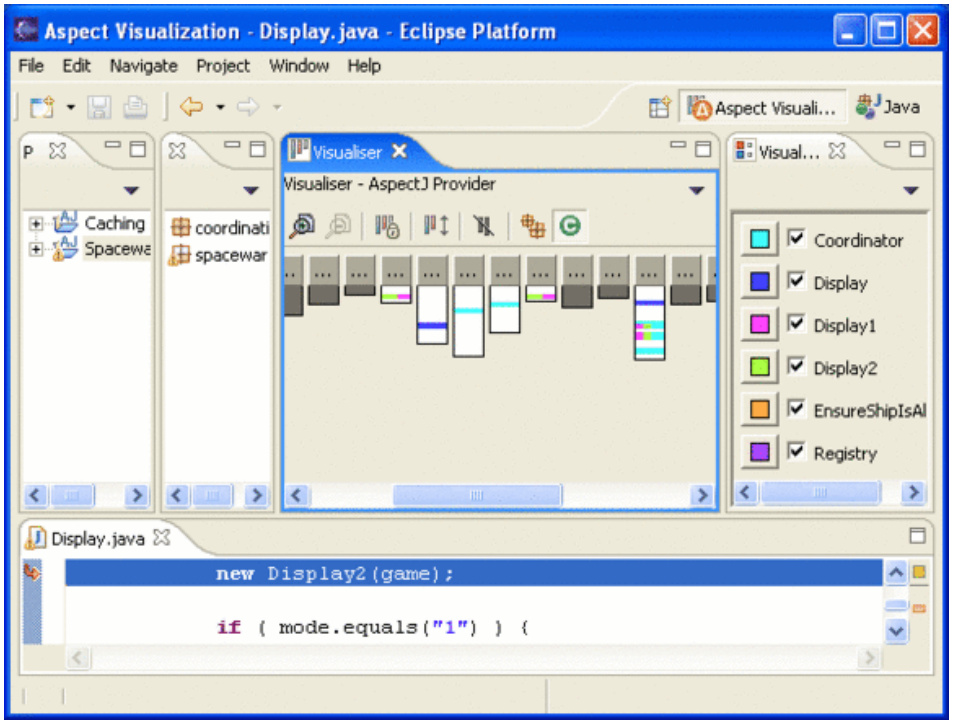
図 9. ビルドの構成エディター



Aspect Visualization のパースペクティブ

アスペクト指向のアプリケーションを作成する場合、アプリケーションの新しい視覚化の方法を採用すると得をします。そうすれば、アスペクトとクラスの間にある関係を理解し、アプリケーションがどのように振る舞うかの全体的な感触を得られます。例えば、アスペクトがどれだけの数のクラスに影響を及ぼすか？既に説明されているとおり、AJDT はコードにてナビゲートする上での先進的な機能を提供したり、何がアドバイスされているかを示すなどをします。この土台の上に構築するかたちで、AJDT はそれ自身のAspect Visualizationのパースペクティブを提供します。この機能はアプリケーションでのクロスカット・コンサーンを視覚的にとらえるために特別に設計されています。このパースペクティブを起動するには、Window>Open Perspective... に移動し、Aspect Visualizationを選択します。図10のような表示を得られるはずです。

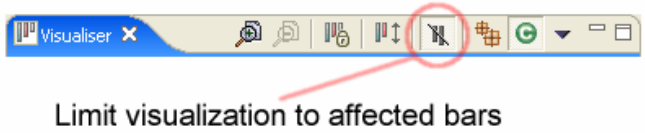
図 10. パースペクティブによるVisualization



このパースペクティブを開いた後では、コーディングよりも視覚化に重点が置かれているのがわかります。Aspect Visualization のパースペクティブに統合された主要な視覚化のツールは、パースペクティブの中央に位置するVisualiserです。過去バージョンのAJDTを使用したことがあれば、Visualiser は別段新しくも何ともありません。しかしながら、現在ではスタンドアロンなプラグインですが、過去のリリースではそれはAJDTの一部でした。（自身のデータを見るために）Visualiser を使う独自のプロバイダーを作成することが可能であると言えるのです。（Visualiser に関する詳細については、Help>Help Contents>Visualiserの順番で選択してください。）しかし、Aspect Visualization のパースペクティブでは、図10でもおわかりのとおり、AspectJ のプロバイダーを使用しました。これはプロジェクト内のクラスとアスペクトをバーで表示し、アスペクトがコードに影響を及ぼす部分をバーのストライプで表わします。バーの長さはファイル・サイズを相対的に表現します。つまり、バーが長ければ長いほど、そのバーが表示するファイル内のコードにそれだけ多くの行があると言うことです。

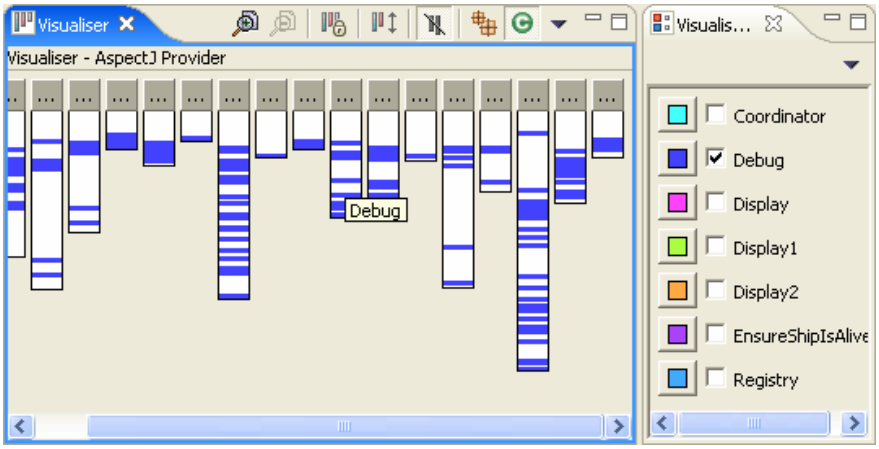
Visualiserのデータを設定するには、左端にあるProjects ビューのAspectJ プロジェクト、またはそれの右隣にあるPackage ビューのパッケージを選択します。例えばSpacewar の例を観察すれば、灰色で塗りつぶされたバー、色付きのストライプが1本付いたバー、そして2本以上の色付きストライプの付いたバーなどがあります。（ときには1本の直線上に複数のストライプが重なる場合もあります。）灰色のバーは、どのアドバイスにも影響されなかったクラスやアスペクトを表記します。図11に示されるボタンを押せば、これらの影響されていないバーにフィルターをかけて取り除くことを選べます。

図 11. Visualiser の制御



影響を受けた（図11に示されるボタンを押すことによりフィルターをかけられた後に残された）バーとそれらのストライプに注目すれば、（Visualiserの右側に位置する）Visualiserメニューにあるボタンと色が一致します。それぞれの色付きボタンの隣にあるのは、チェック・ボックスにラベルです。ラベルはその色が表示するアスペクト（内部アスペクトの場合、アスペクトを含むクラス）の名前を与え、現行の視覚化にそのアスペクトが包括されているかどうかをチェック・ボックスが示します。この様に、複数のアスペクトがある場合は、チェック・ボックスのチェックを外すことにより、興味の対象外であるアスペクトを除外できます。例えば、ビルドの構成debug.ajpropertiesでSpacewar の例を構築してからVisualiserに戻れば、`Debug` アスペクトが何に影響を与えているかを知りたいだけであると判断するかも知れません（図12）。どの色がどの特定のアスペクトを象徴しているかを調べる別の方法として、興味の対象となっているストライプの上にマウス・カーソルを移動すると言う手もあります。そうすれば、そのストライプが表示するアスペクトの名前が表示されます。もしもストライプに複数の色があれば、そのコードの行に複数のアスペクトが影響を及ぼしていることになります。

図 12. デバッグのアスペクトの効果を視覚化



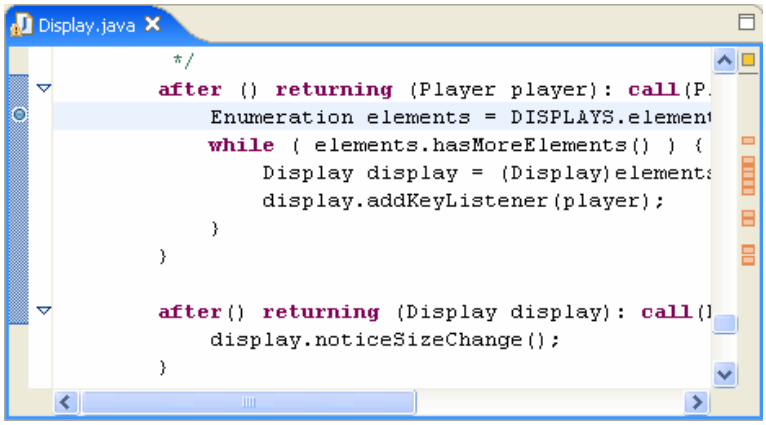
Visualiser のまた別の重要な機能は、それもコードにてナビゲートする方法を提供することです。Spacewar の例の視覚化を見れば、`Display`のアスペクトがあるのがわかります。まだコードに目を向けていませんので、このアスペクトが何をするのかに対して好奇心をいだいていることでしょう。それが4つのクラスに影響を与えるのはわかるのですが、そのクラスとは何でしょうか？それぞれの影響されたバーの天辺にある暗めな灰色のストライプにカーソルを動かせば、それらが`spacewar.Display`、`spacewar.Game`、`spacewar.SWFrame`、そして`spacewar.Timer`であることがわかります。しかし実際にはアスペクトは何をアドバイスしているのでしょうか？`spacewar.Game` のバーにあるストライプをダブルクリックすれば謎が解けます。`Display` アスペクトに影響されたGame.java のポイントに直接行き、Aspect Visualization パースペクティブの底辺にあるEditor ビューに表示されます。

デバッグ

アスペクトを含むアプリケーションをデバッグすることは、アスペクトを含まないアプリケーションをデバッグするのと同じくらいに簡単です。EclipseにあるJava デバッガーを使うのに慣れているのであれば、ここでは特に目新しいものには出くわさないでしょう。AJDT 1.1.11の登場以来、アドバイスの前後にブレークポイントを設置するのはメソッドにてブレークポイントを設置するのと同じことです。around advice にてブレークポイントを設置するのは少々異なります。そのようなブレークポイントにてデバッガーに停止して欲しいのであれば、インラインを切る必要があります。そうするには、Window>Preferences>AspectJ>Compiler の順番で移動し、Advanced のタブまでナビゲートします。一度そこにたどり着けば、No inline オプションにチェックを入れてOKをクリックします。

AspectJ アプリケーションのデバッグが標準のJavaアプリケーションのデバッグといかに類似しているかを示すために、Spacewar の用例にブレークポイントを設置してコードに踏み込みましょう。まず、Window>Open Perspective>Debugをたどることにより Debug のパースペクティブに切り換えましょう。Aspect Visualization のパースペクティブにて`Display` クラスに目を向けましたので、`after () returning (Player player): call(Player+.new(..))` のアドバイスにブレークポイントを挿入しましょう。Display.java クラスを開き、図13にて示されるとおりafter returningのアドバイスへとナビゲートします。それから、`Enumeration elements = DISPLAYS.elements();`の行の次にある gutter をダブルクリックするか、同じ部分を右クリックしてからToggle Breakpoint を選択します。

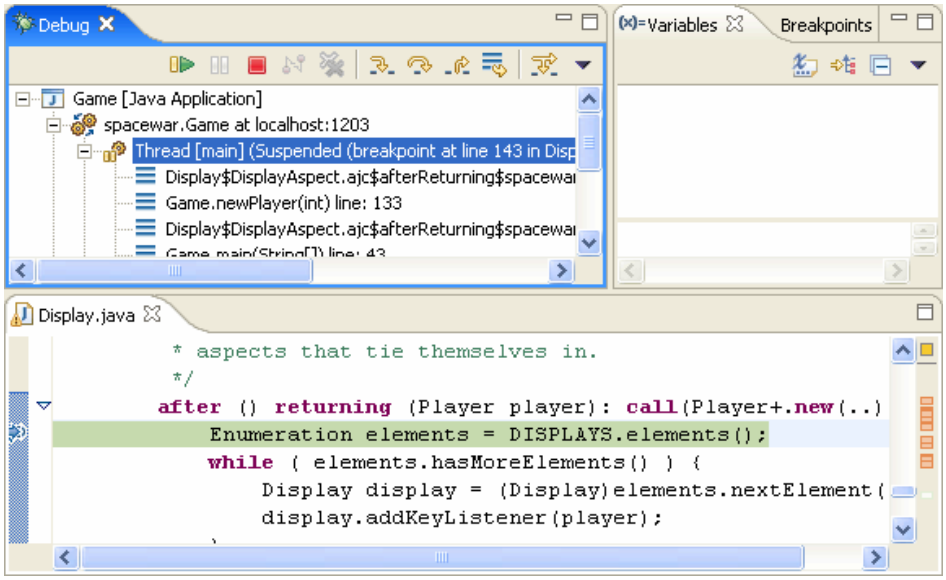
図 13. ブレークポイントを設置



ブレークポイントの設置の後、デバッグ・モードにてアプリケーションを起動するために、普通のJavaアプリケーションを立ち上げる場合と同じステップを踏まえます。（ワークベンチ・ツールバーにある緑色の昆虫（bug）のボタンの）ドロップダウン・メニューにて、Spacewar 実行の構成を選択すればよいのです。もしもSpacewar 実行の構成が設定されていなければ、同じドロップダウン・メニューからDebug... を選択します。`spacewar.Game` クラスにある`main()` メソッドを実行するJavaアプリケーションの構成を作成し、それからDebugをクリックします。

図14にて示されるとおり、ブレークポイントに達すると同時に、デバッグ・モードはSpacewar アプリケーションを一時停止させます。現行の変数に関する情報はDebug パースペクティブの右上にあるVariables ビューにあり、現行のスレッドのスタック・トレースはDebug パースペクティブの左上にあるDebug ビューにあります。Editorにて、コードのどの部分にて一時停止しているかを知ることができます。コードに踏み入れるには、Debug ビューにあるStep Overボタンをクリックしてください。このとおり、これは簡単なものです。

図 14. ブレークポイントに到達

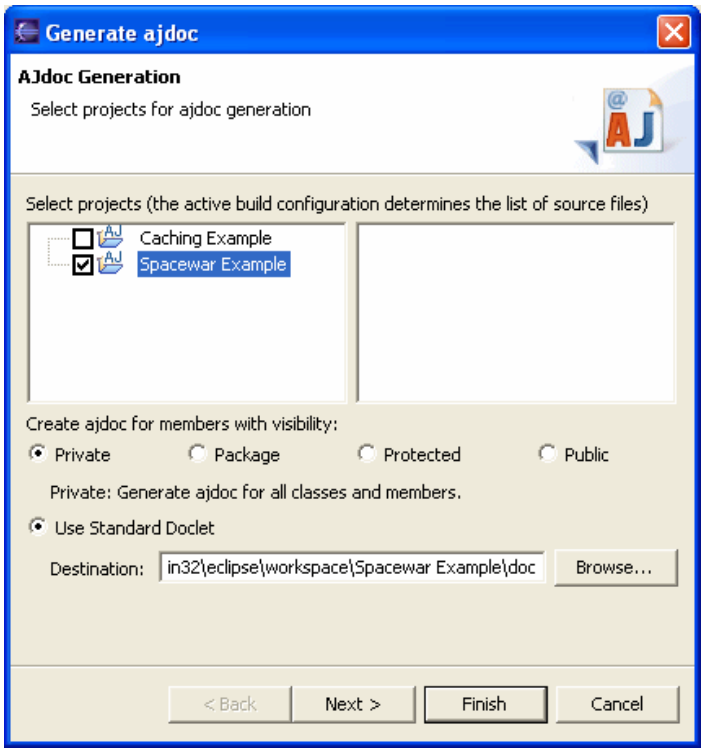


ドキュメントの生成

ソース・コードにある特別にフォーマットされたコメントを基に、Javaプロジェクト用のAPI ドキュメントを生成するのに`javadoc` ツールを使う事ができます。AspectJ 配布物のバージョン1.2 は`ajdoc`と呼ばれるツールを含みます。これは`javadoc`により生成されたドキュメントにアスペクトのクロスカットの本質に関する詳細を追加する`javadoc` の拡張です。Eclipse のJDT ツールは、`javadoc` ツールを呼び出しまたは構成するウィザードを包括します。AJDTがインストールされていれば、類似するウィザードが`ajdoc`にもあります。

`ajdoc`ウィザードを呼び出すには、メニュー・オプションProject>Generate ajdoc..を選択します。表示されるダイアログでは、API ドキュメントを作成したいプロジェクトをを単に選択するのみです。図15にて示されるのは、Spacewar の例です。

図 15. ajdoc ウィザード

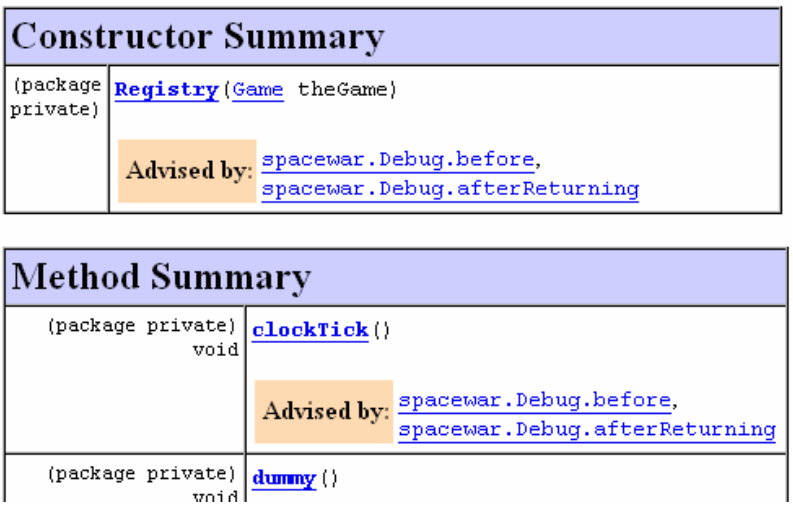


コードにあるクラスとメソッドの多くが公開されていませんので、生成されたドキュメントの使い勝手をよりよくするには、ダイアログのvisibilityのセクションにあるPrivate オプションを選択します。ダイアログの最後の方にあるDestination フィールドは（プロジェクトにあるdocディレクトリーとして初期化される）出力の位置を指定します。

（追加的なオプションの指定を可能にする）次のページに、Next ボタンをクリックして移行しましょう。Open generated index file in browserのオプションを選択することにより、生成されたドキュメントを（生成されると同時に）見られるようにします。Finishをクリックした後、`javadoc` のロケーションをアップデートしたいかどうかと質問されます。そこで「はい」と答えれば、次回同じ場所を訪れる場合にもその出力の位置が記憶されています。`ajdoc` ツールがそれから起動され、（実行されるのを観察したいのであれば）Eclipse にあるConsole ビューに切り換え

られます。それが終了すれば、Webブラウザーにインデックス・ページが表示されるはずです。`spacewar.Registry` アスペクトまでナビゲートすれば、（図16にて図解されるとおり）アドバイスの効果を表わすかたちで通常の`javadoc` 出力が拡張されているのがわかります。

図 16. 生成された文書



さらに別のオプション

他にも多くのオプションと設定がAJDTにありますが、多過ぎてこの記事で全てを取り上げることは不可能です。中でも頂点を成すものとして、（多種類のコーディングの論点を警告またはエラーとして宣言するか無視するようにコンパイラーを構成できる）先進的なコンパイラー・オプション、AspectJ InPath そしてAspect Pathのクラスパスの入力を指定する追加的なプロジェクト・プロパティーのページ、そしてProblems ビューへの情報を織り込むメッセージを送信するオプション（AspectJ 1.2にあるこの新機能を試しに使う）があります。AspectJに特化した多くのコード・テンプレートもあります。（そのリストを取得するには、Preferences ダイアログを開くためにWindows>Preferences を選択し、Java>Editor>Templatesへ移行します。）これらの機能（そしてその他）に関するより詳細にわたる情報をお求めでしたら、Eclipse Help（特にWhat's new とAJDT Highlights のページ）を参照されることをおすすめします。

まとめ

この記事では網羅しきれないほどに先進的な論題も数多くありますが、AspectJを伴うアスペクト指向プログラミングにAJDT が提供するサポートの本質に対する感触を得られたことでしょう。すでにご覧になられたとおり、エディター側のOutline ビューとマーカーから（アプリケーションのより高レベルなビューを与える）力強いVisualization パースペクティブにいたるまで、数多くの方法でツールによりアスペクト指向のアプリケーションのクロスカットの本質は浮き彫りにされます。

近年AJDT は多くの面で進化を遂げましたが（詳細に関しては、バージョン1.1.11 とバージョン1.1.12 の新しくて注目すべきリストをご覧になってください。）、さらなる拡張の余地がまだまだ多く残されていることを、この分野に与えられる難題が物語っています。JDTとのより親密な統合を達成する方法を探究するLancasterとして知られるプロジェクトにて、実験的な作業が展開さ

れています。Package Explorer と他のビューでのアスペクトの構造のコード完成、フォーマット、検索、閲覧、リファクタリング、そしてサーフェシング(surfacing)のような分野でのより完成された機能を可能にします。最近見られる進歩はなかなかのものですので、近い将来に予定されている最初の1.2.0 ベータ版発行に期待して注目しましょう。ここでの目的は　AJDT を堅実にJDTと同等にすることです。ここから先では、（ポイントカットのウィザードとアスペクトに特化したリファクタリングへのサポートのような機能で）AOP をさらに深く学んで探求することを可能にする新規の機能を開発することが焦点となります。これから先が本当に楽しみだと言えます。

最後に、AJDT はオープン・ソース・プロジェクトですので、（バグ報告、フィードバック、そして機能に対する要求を提供する）ユーザーから（パッチを提出する）開発者にいたるまで、数多くのかたちで人々が参加することを前提としています。この活動に関与したい場合は、AJDT の Webサイト（参考文献を参照）を訪れ、Bugzilla、ニュース・グループ、開発者用メーリング・リスト、そしてするべきタスクのページへのリンクを探してみてください。

著者について

Matt Chapman

Matt Chapman 氏は、英国のハースリーにある IBM アスペクト指向ソフトウェア開発チームに所属する開発者です。長年にわたり Java テクノロジーに関与した後、現在では (共に Eclipse.org のプロジェクトである) AJDT と CME (Concern Manipulation Environment) に焦点を合わせています。デスクトップの Linux そしてユーザー・インターフェースの設計にも興味をいだいています。

Helen Hawkins

Helen Hawkins 博士は、英国のハースリーにある IBM アスペクト指向ソフトウェア開発チームに所属する開発者であり、AJDT と CME (Concern Manipulation Environment) の Eclipse プロジェクトに従事しています。チームに入る前は、大型のソフトウェア・システムのテスト (特にストレス・テスト) に専念していました。