User's Guide for

CaesarJ Development Tool

Software Technology Group

Version 0.4.0

Jochen Unger Daniel Zwicker

October 20, 2004

Contents

1 Introduction

This documentation describes how to use the CaesarJ-Eclipse Plugin.

1.1 What is CaesarJ?

CaesarJ is a new aspect-oriented programming language, which addresses the most important goals of the software design: modularity, reuse, flexibility and correctness. It is easy to learn because it fully integrates with the Java programming language. All new language extensions are compiled to efficient Java byte-code.

The CaesarJ highlights are:

- Virtual Classes
- Mixin Composition
- Collaboration Interfaces
- Bindings
- Aspectual Polymorphism
- Dynamic Deployment

For more detailed information please visit http://caesarj.org/.

1.2 About the CaesarJ Eclipse Plugin

CaesarJ extends the Java language with new the syntax and semantics. In order to provide a good IDE support for the CaesarJ programming language, we have extended the Eclipse's JAVA Development Tool (JDT) plugin with Caesar specific features (For details about the Eclipse platform, please visit http://www.eclipse.org/).

Some of the CaesarJ plugin highlights are:

- Editor support with keyword highlighting. (Figure 1)
- Outline view showing structural members and crosscutting relationships. Also from an advice declaration to the places it advises. (Figure 2)
- New CaesarJ-project wizard. This wizard helps you to start a new CaesarJ-project. (Figure 3)

- CAESARJ hierarchy view. This view shows the multiple inheritance and nested class relations of an CAESARJ top level class. (Figure 4)
- Debugging support. (Figure 5)

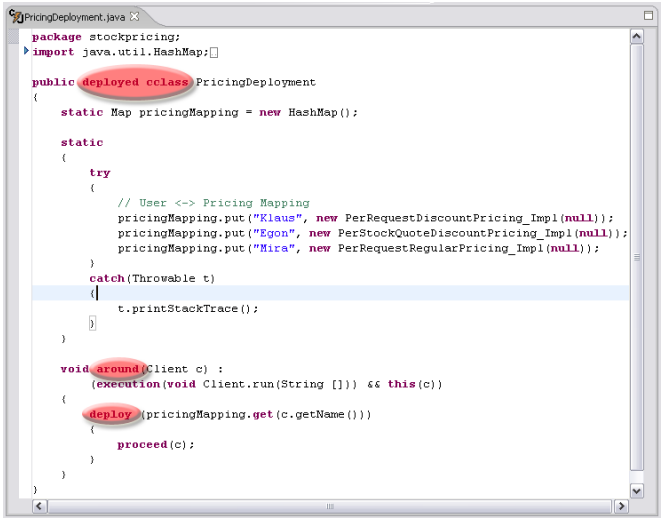

Figure 1: Codehighlighting in CAESARJ Development Tool

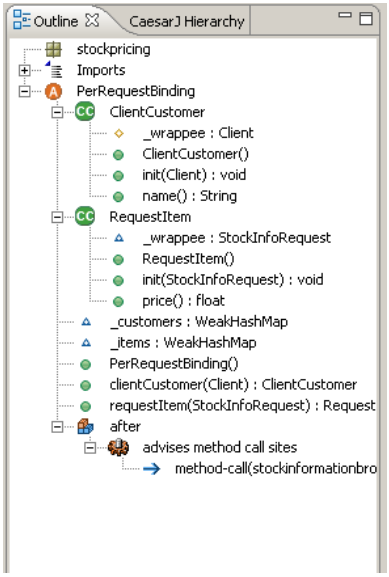

Figure 2: Outline view with advice relations

For detailed description please see Section 3.

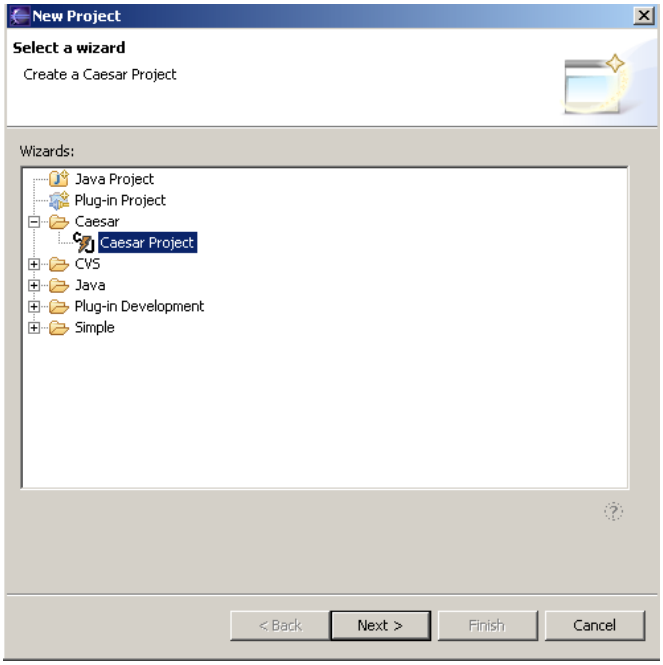

Figure 3: New CAESARJ-project wizard

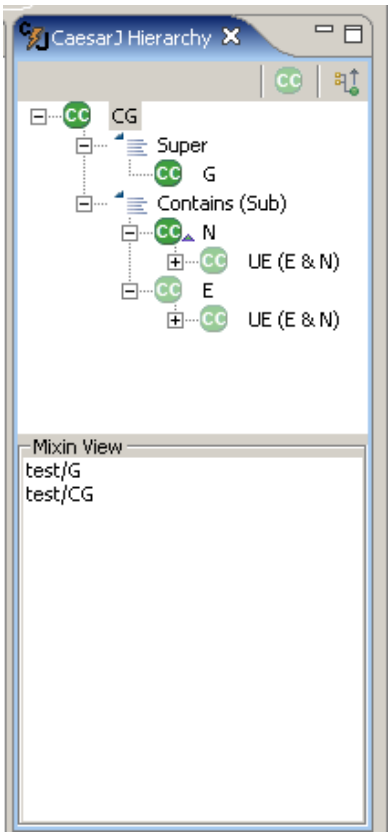

Figure 4: CAESARJ hierarchy view

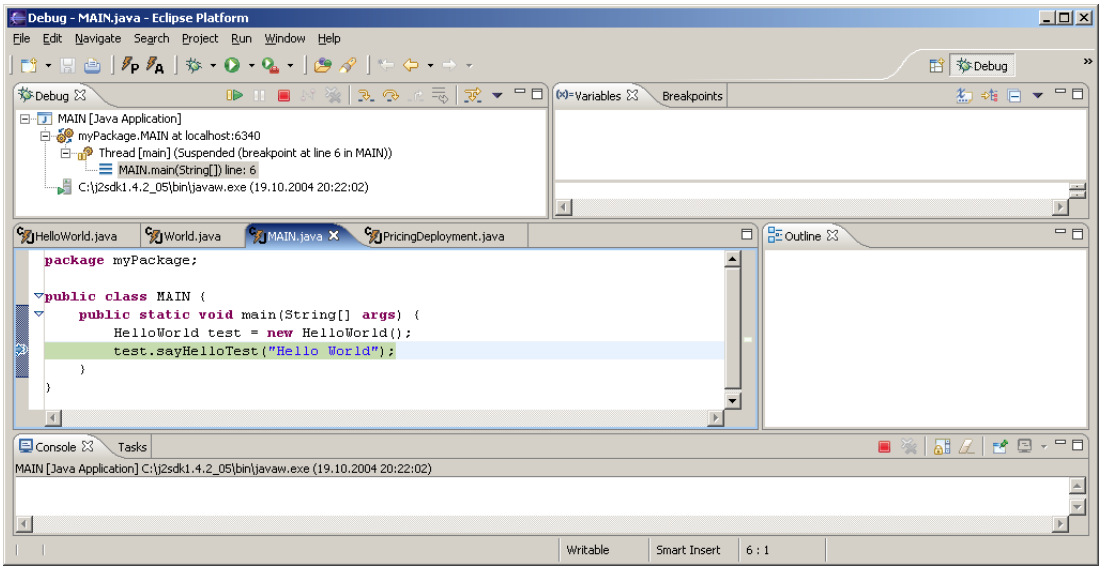

Figure 5: Debugging an CAESARJ-project

2 CaesarJ Development Tool Installation

The following two sections describe the installation of the CAESARJ eclipse plugin. Two scenarios are possible: clean installation and updating an existing installation.

2.1 Clean Installation

The CAESARJ Development Tool is installed by using the Eclipse Update Manager. We recommend you to use Eclipse 3.x.

2.1.1 Using A Proxy Server

If you need to use a proxy server to access the internet, the first thing to do is to configure the proxy preference details, so that the update manager can contact the CAESARJ Development Tool update site. From the **Window** menu select **Preferences** and then the **Install/Update** tab. Please enter your proxy server details as shown in figure 6.

2.1.2 Installing via Update Manager

Create an update site bookmark for the CAESARJ Development Tool update site, and start the install procedure. From the help menu, select **Software Updates** → **Find and Install**. Then select **Search for new features to install** and

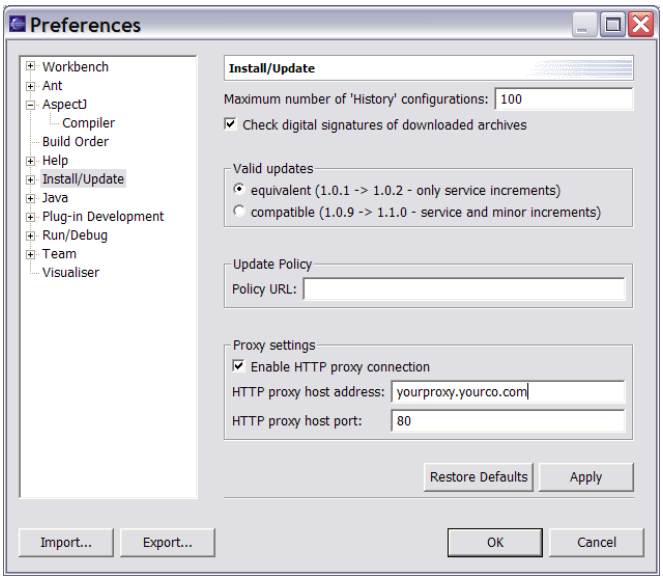

Figure 6: Setting up your proxy server

click **Next**. Afterwards click **Add Update Site** and enter the name **"CAESARJ update site"** and the following URL:

http://cage.st.informatik.tu-darmstadt.de/caesar/updatesite/0.4.0/

Click **OK**. Fully expand the appearing CAESARJ Development Tool update site node and select **CAESARJ**. Pick **Next**. Select **org.caesarj.feature** as shown in figure 7 and click **Next**.

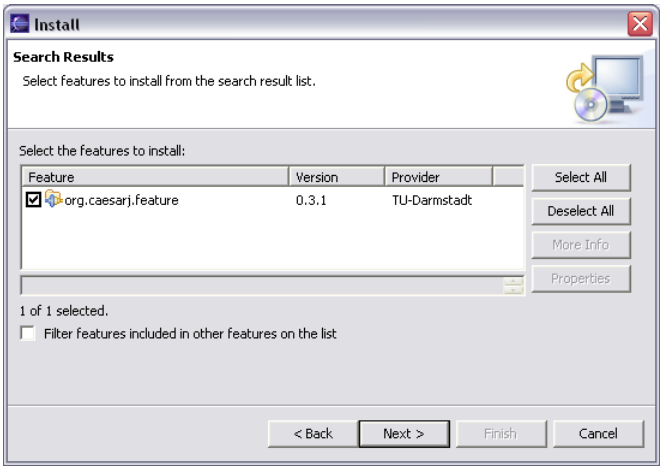

Figure 7: Selection of the CAESARJ-plugin

Accept the **license agreement** and proceed to the installation.

2.2 Updating an Existing Installation

Proceed as as in section clean install, except that in this case the CAESARJ Development Tool update site bookmark is already existing. You only need to expand the bookmark node and go on. If the version you have installed is the same as the version on the update site (or even more recent), then you will not be confronted by any installing options.

2.3 Testing the Installation

Restart the Eclipse workbench. Try to open a new perspective by clicking **Window** → **Open Perspective**. Pick **other** and select **CaesarJ Perspective** in the upcoming list. When the perspective opens successfully, the installation of your CAESARJ Development Tool works fine.

3 Features

The following section describes the additional features of the CaesarJ Development Tool Plugin.

3.1 Opening the CaesarJ-perspective

First of all you need to open the CaesarJ-perspective. It includes some new features like the CaesarJ-editor, the new outline view or the CaesarJ-hierarchy view.
You can open this perspective by selecting: **Window** → **Open Perspective** → **other** → **CaesarJ perspective**.
If this is the first time you are using the plugin, you will see a dialog popup as shown in figure 8.

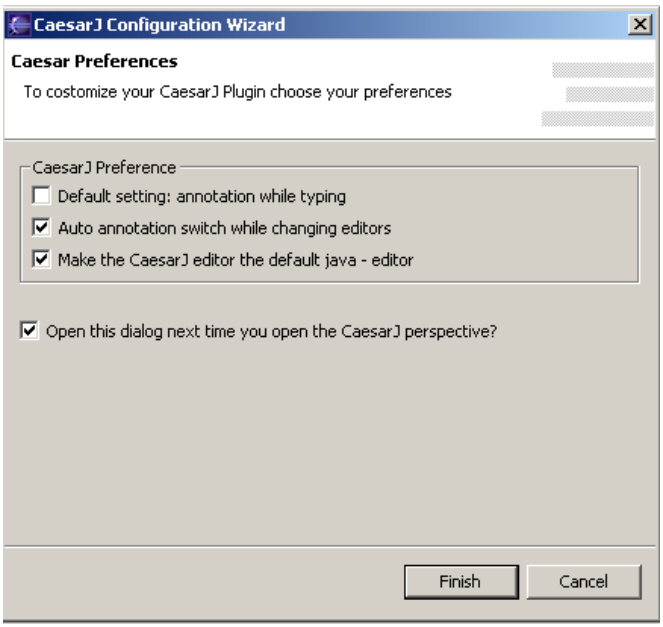

Figure 8: The CaesarJ Preferences

This dialog configures some Eclipse settings, which will make your life much easier when working with CaesarJ-projects. Leave everything as selected and click **Finish**.

3.2 Creating a new CaesarJ project

From the File menu select **New** → **Project**. Pick **Caesar Project** in the list and select **Next** as shown in figure 9.

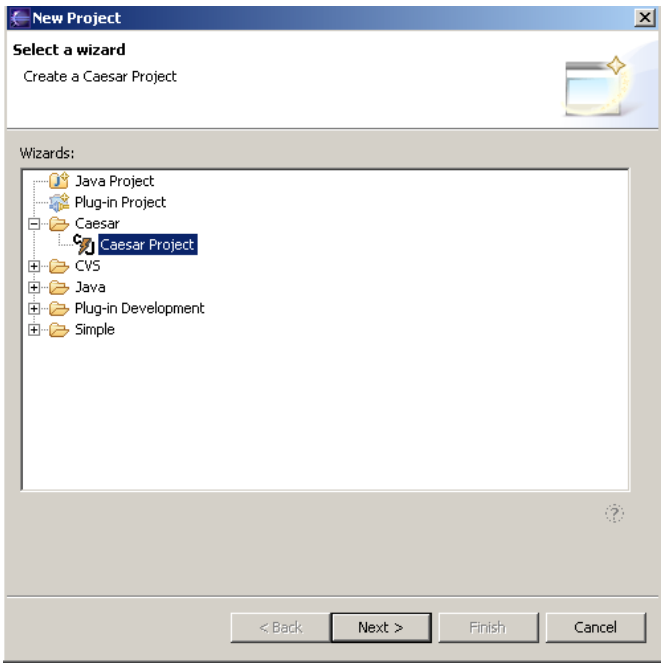

Figure 9: Coosing the New Project Wizard

If the item doesn't appear in the list, this is probably because you use the plugin for the first time. Select **Other** and then **Caesar** and **Caesar Project**. The wizard opens up. Here specify a name for your project as shown in figure 10.

This wizard has identical behavior to the new Java project wizard (with the exception that it creates a project with the Caesar nature).
When you click **Finish**, your project will be created.

3.3 Adding a Class to Your Project

First you have to create a package for your class files. Select the project you created in the section 3.2 in the package explorer. Right click on it and select **New** → **Other** from the context menu. You have to look for **Package** in the **Java** subsection as you can see in figure 11.

Name the package **"myPackage"** then click **Finish**.
Right-click on the package you have just created and select **New** → **Class** from the context menu. Name the class **"HelloWorld"** and activate the option

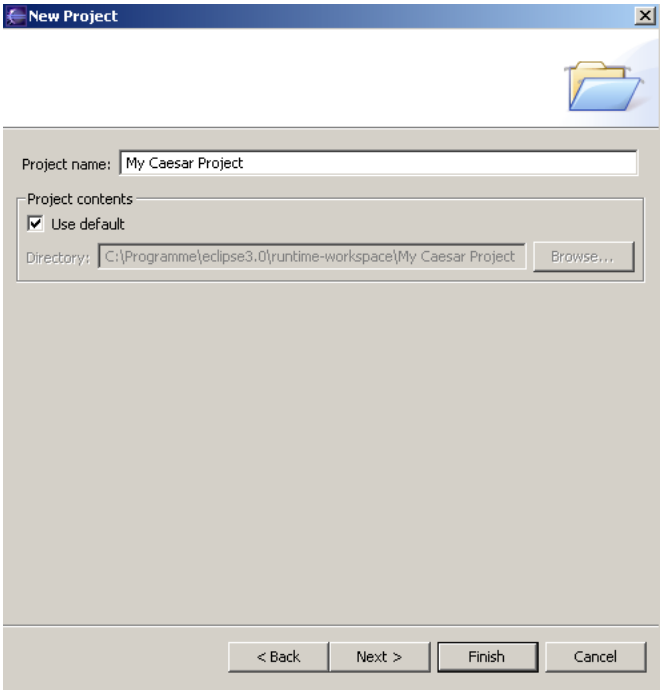

Figure 10: The New Project Wizard

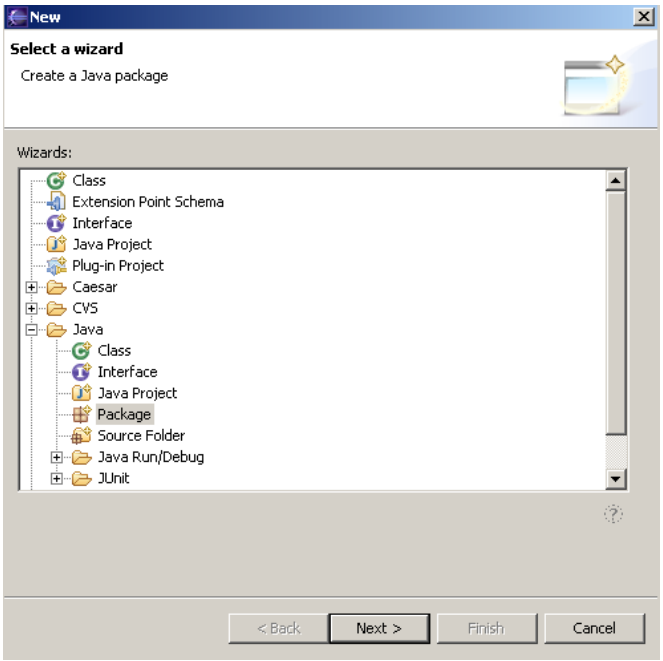

Figure 11: Creating a package

to let Eclipse create a new main method for you. Click **Finish**. Edit the text in the editor so that it looks like this:

Listing 1: HelloWorld.java

```
1 package myPackage;
2
3 public cclass HelloWorld {
4  private static HelloWorld instance = new HelloWorld();
5
6  public void sayHelloTest(String message) {
7   System.out.println(message);
8  }
9 }
```

Save the file.
Notice that unlike in a Java project, there was no eager parsing of the buffer while you were typing. Also the outline view didn't update.[1] Your Eclipse workbench should be looking somehow like in figure 12.

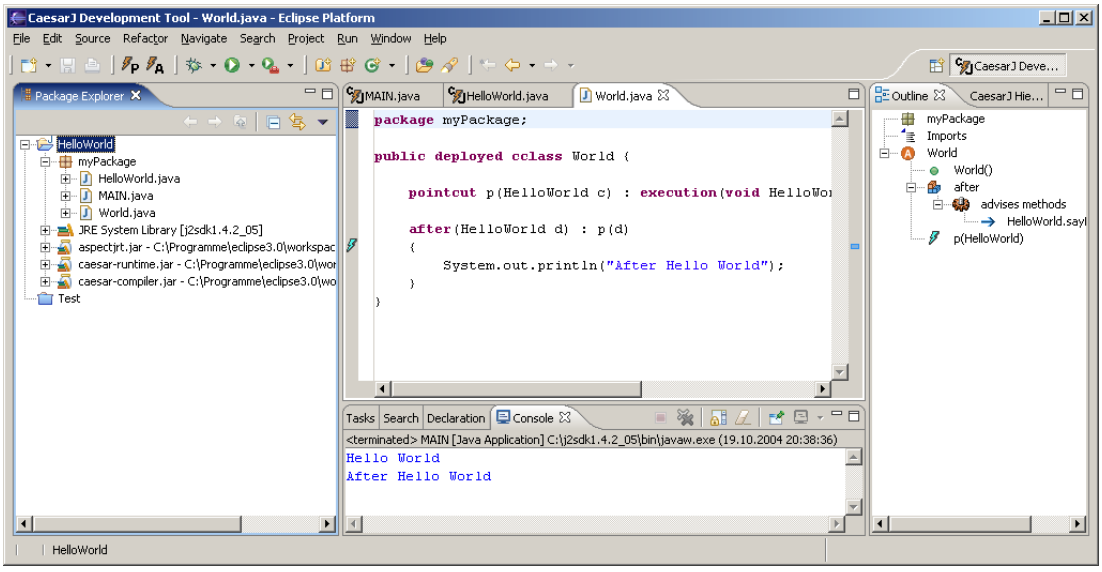

Figure 12: Workbench with HelloWorld.java

[1] The CAESARJ outline bar requires meta information from the compiler to display cross-cutting relationships.

3.4 Adding a New Aspect to Your Project

Create a new Class and name it **"World"**. Edit the buffer so it looks like listing 2 and then save it:

Listing 2: An CAESARJ-cclass including an aspect

```
1  package myPackage;
2
3  public deployed cclass World {
4
5   pointcut p(HelloWorld c) : execution(void HelloWorld.sayHelloTest(String)) && this(c);
6
7   after(HelloWorld d) : p(d)
8   {
9    System.out.println("After Hello World");
10  }
11 }
```

Furthermore you will need a "'Main-Class"' to run the project. Just create one like this:

Listing 3: An CAESARJ-java-class including an main method

```
1  package myPackage;
2
3  public class MAIN {
4   public static void main(String[] args) {
5    HelloWorld test = new HelloWorld();
6    test.sayHelloTest("Hello World");
7   }
8  }
```

Make a clean Build of the project, and the outline view populates like in figure 13. Expand the **"after()"** node.

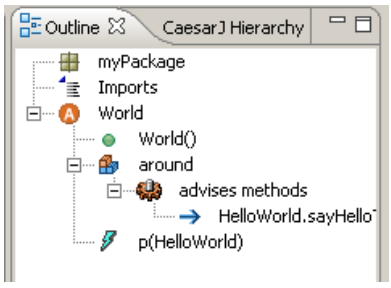

Figure 13: Outline view with content

You can see that this advice is affecting the **"HelloWorld.sayHello()"** method. Clicking on the **"HelloWorld.sayHello()"** node in the outline takes

you to the declaration of **"HelloWorld.sayHello()"**.
Notice the *advice annotation* in the editor buffer (highlighted) and that the **"sayHello"** method in the outline view shows that it is advised by the *World aspect*. It should look like in figure 14.

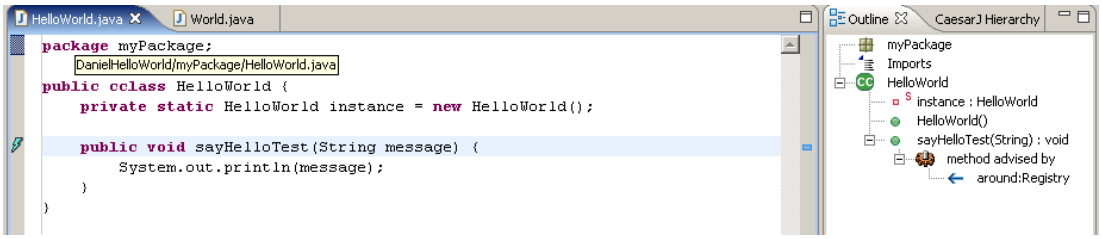

Figure 14: Advice relationship

Selecting the **"World.after()"** node in the outline view takes you back to the advice declaration. Right-clicking on the advice annotation brings up a context menu that also allows you to navigate to the advice.

3.5 Running an CaesarJ Program

Select your CaesarJ project in the Package Explorer. Drop-down the **Run** icon on the toolbar and click **Run...**
Select **Java Application** in the left-hand tab and click **New**. Name this configuration **"HelloWorld"** and then click **Search** to find the main class. Select **"HelloWorld"** as described in figure 15.

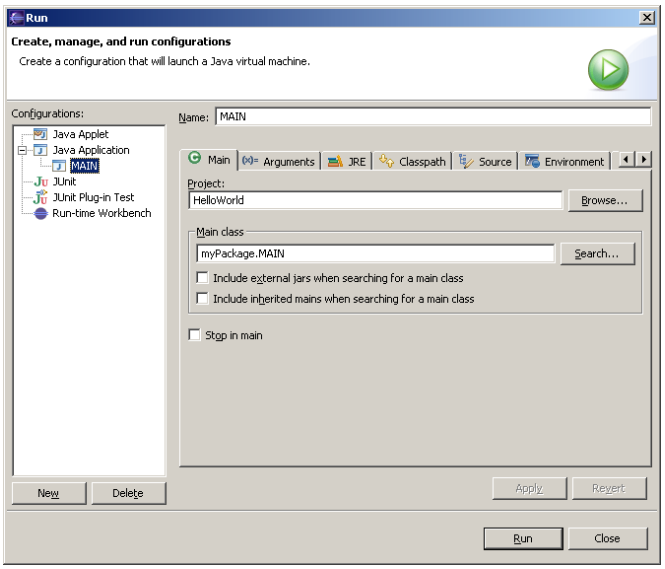

Figure 15: Running a CaesarJ program

Click **Apply** and then **Run**.
You should see the output of the **"HelloWorld"** class and the **"World"** aspect in the console as shown in figure 16.

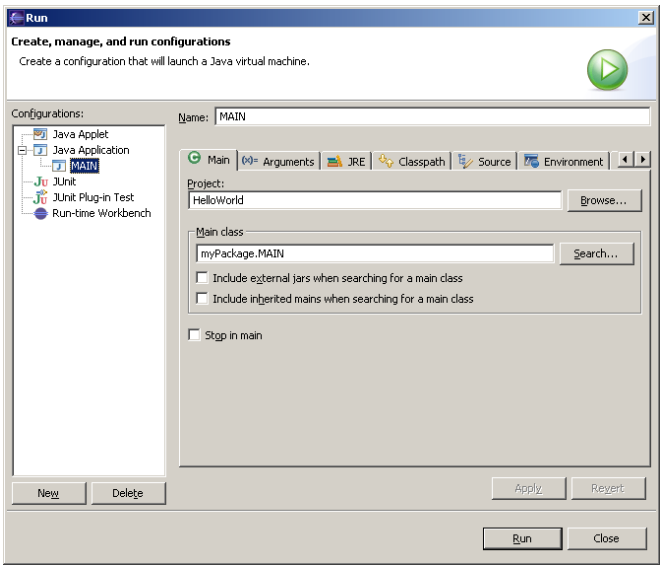

Figure 16: Programs output

To run this configuration again, just click on the **Run** icon placed on the toolbar.

3.6 Debugging CaesarJ Programs

You can debug the standard JAVA part of CAESARJ programs by using the normal Java debugger. To set a breakpoint, right-click in the gutter of the editor and choose **Toggle Breakpoint** (see figure 17). Another possibility is a simple double-click on the gutter. If it is not possible to set breakpoints the double-click will not have any affects.

After setting one or more breakpoints, you launch the Eclipse debugger in the normal way by clicking on the debug icon in the toolbar. The debugger perspective looks like figure 18.

You can use the Java Debug step filters (**Window** → **Preferences** → **Java** → **Debug** → **Step Filtering**) to make this process a little easier.
Note: A current limitation is that you cannot set breakpoints in cclasses.

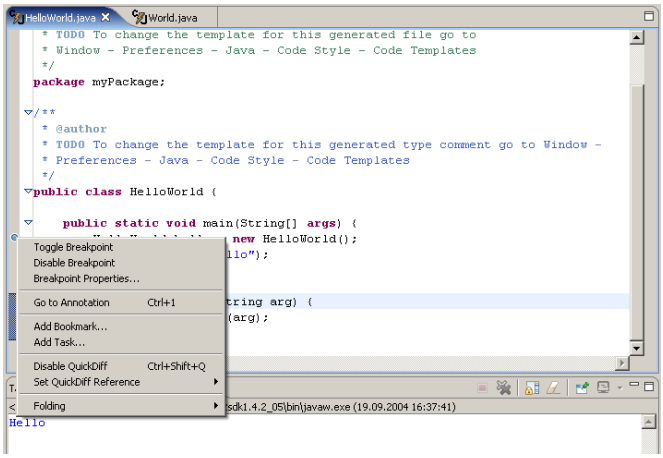

Figure 17: Toggling a debugging breakpoint

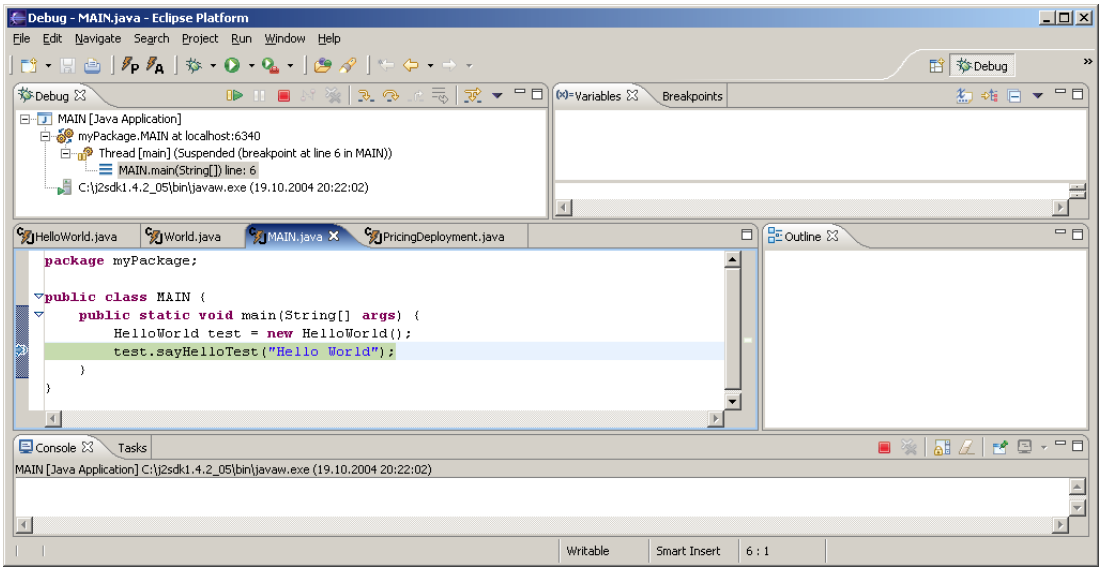

Figure 18: Debugger perspective

4 Properties and Shortcuts

If you have opened the Caesar Perspective, there are some configurations left. Open **Window** → **Customise Perspective**. Check the **Caesar** check box as shown in figure 19.

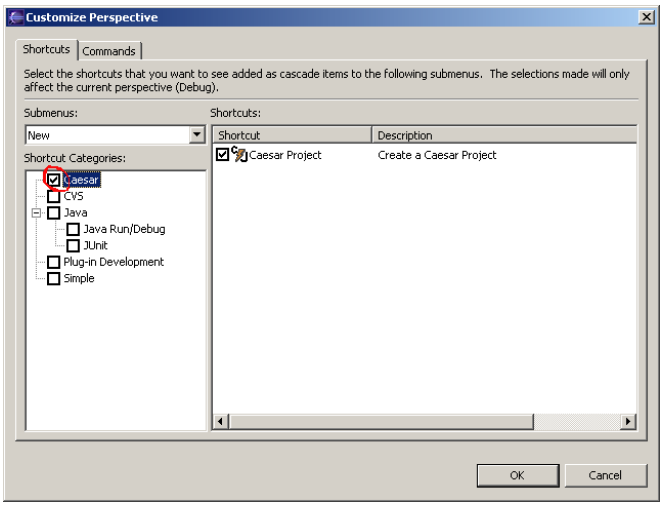

Figure 19: Selection the CAESARJ perspective

If this is done, two new Buttons will appear in the tool bar like in figure 20.

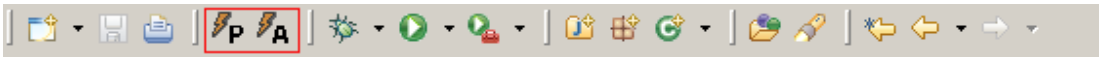

Figure 20: CAESARJ tool bar shortcuts

Figure 21 shows the CAESARJ-Configuration-Wizard, which will be displayed by pressing the **P**-Button.

The **A**-Button toggles the "Annotation-While-Typing" option on or off. Even for the Java-Editor.

A main feature of the CAESARJ Development Tool is the automatic annotation toggling while switching between the CAESARJ- and the JAVA-editor. This is a useful feature, because the CAESARJ Development Tooldoes not support live annotation yet. In this way, CAESARJ syntax are not marked as wrong expressions.

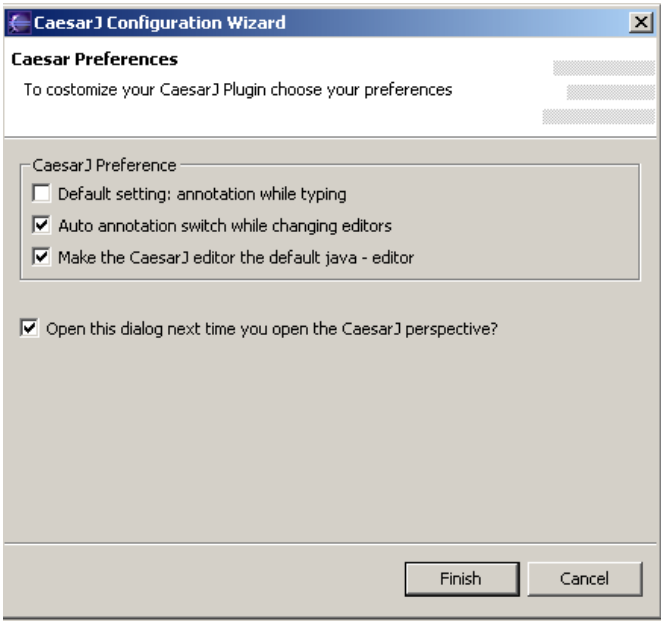

Figure 21: CAESARJ-Configuration-Wizard

5 Using the Visualisers and Views

If this is the first time you use the CAESARJ Development Tool, switch to the CAESARJ perspective by selecting **Window** → **Open Perspective** → **Other**. Pick **CaesarJDT Perspective** (see figure 22) in the list.

This perspective extends the Java perspective. Especially a new view is available. The **CAESARJ Hierarchy View**. See section 5.2 for detailed information.

You can switch between the Java and Caesar Visualization perspectives using the perspective icons located in the top right of the menu bar.

5.1 Outline view

The outline view is showing structural members and crosscutting relationships. It extends the Java outline view by additional information (e.g advice declarations to the places it advises). A sample outline view bar is shown in figure 23.

5.2 Hierarchy View

A CAESARJ hierarchy view displays the hierarchical relationships of CAESARJ cclasses. That means, that for each cclass their super-classes are displayed under the **Super** node (see figure 24). If the class contains nested classes (**Contains** node) there are two displaying modes available for them:

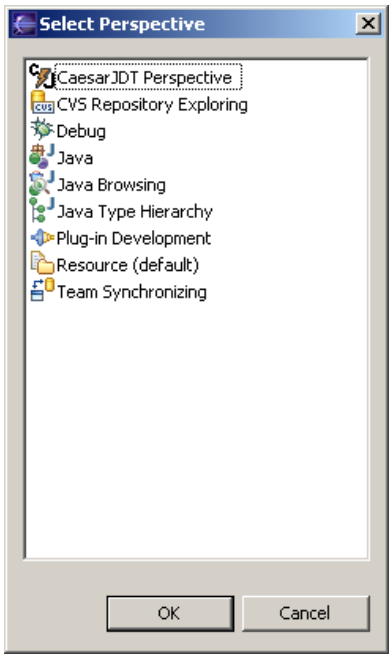

Figure 22: Perspective selection

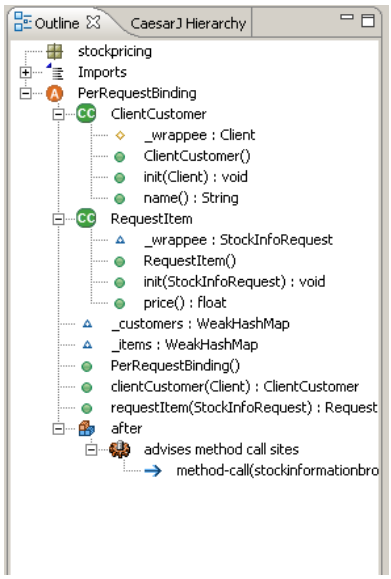

Figure 23: Outline View

Super: For each nested class their super classes are displayed.

Sub: For each nested class their sub classes are displayed. If a sub class has two super classes the linearized inheritance relation is displayed in brackets after the class name.

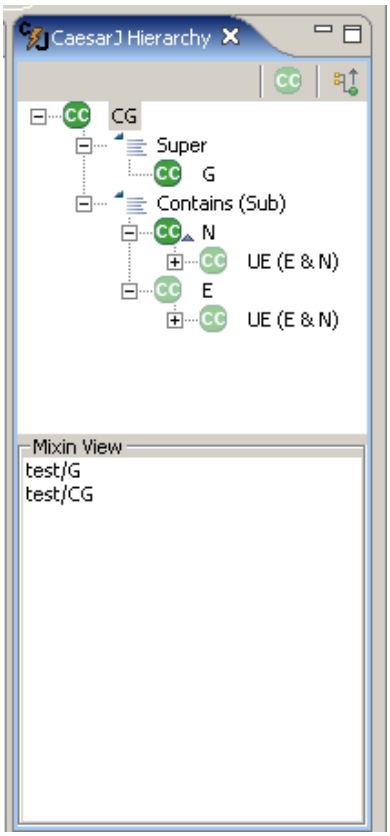

Figure 24: CAESARJ hierarchy view

The modes can be switched by pressing the control button in the upper-right of the view. The second part of the view, named "Mixin view", shows the mixin composition of the currently selected (nested-) cclass.
Note: Since this view needs meta information from the compiler, the view refreshes when a project is (re-)built successfully.

6 Common Problems and Limitations

The CAESARJ Development Tool is still under development. That is why there are some restrictions in this release. Some of these are listed below:

- This release does not support live annotation while typing. To get this

available, an CaesarJ AST[2] would have to be rebuilt while changing the code in the editor. This is not implemented yet.

- Showing the class hierarchy of an cclass marked in the editor by pressing **F4**. Only the hierarchy view of an entire source file and its included classes is supported.
- In-time refreshing of the outline bar and of the hierarchy view is not supported yet. In this release both of the views need meta information from the compiler. That is why they only refresh after a (re-) build of the entire project.
- It is not possible to declare breakpoints in cclass-es, when debugging an CaesarJ application.

[2] Abstract Syntax Tree