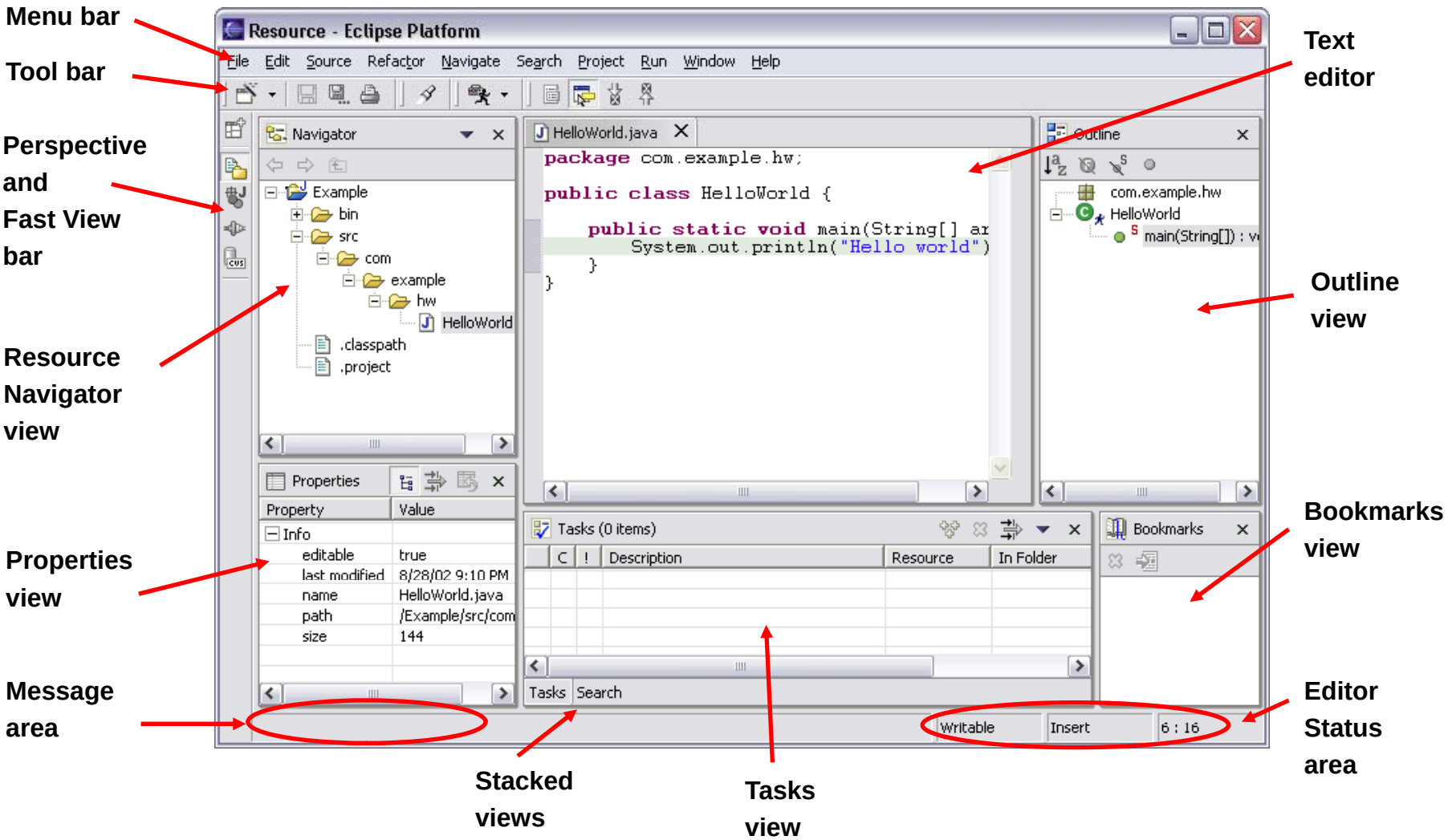
Extending JTransformer

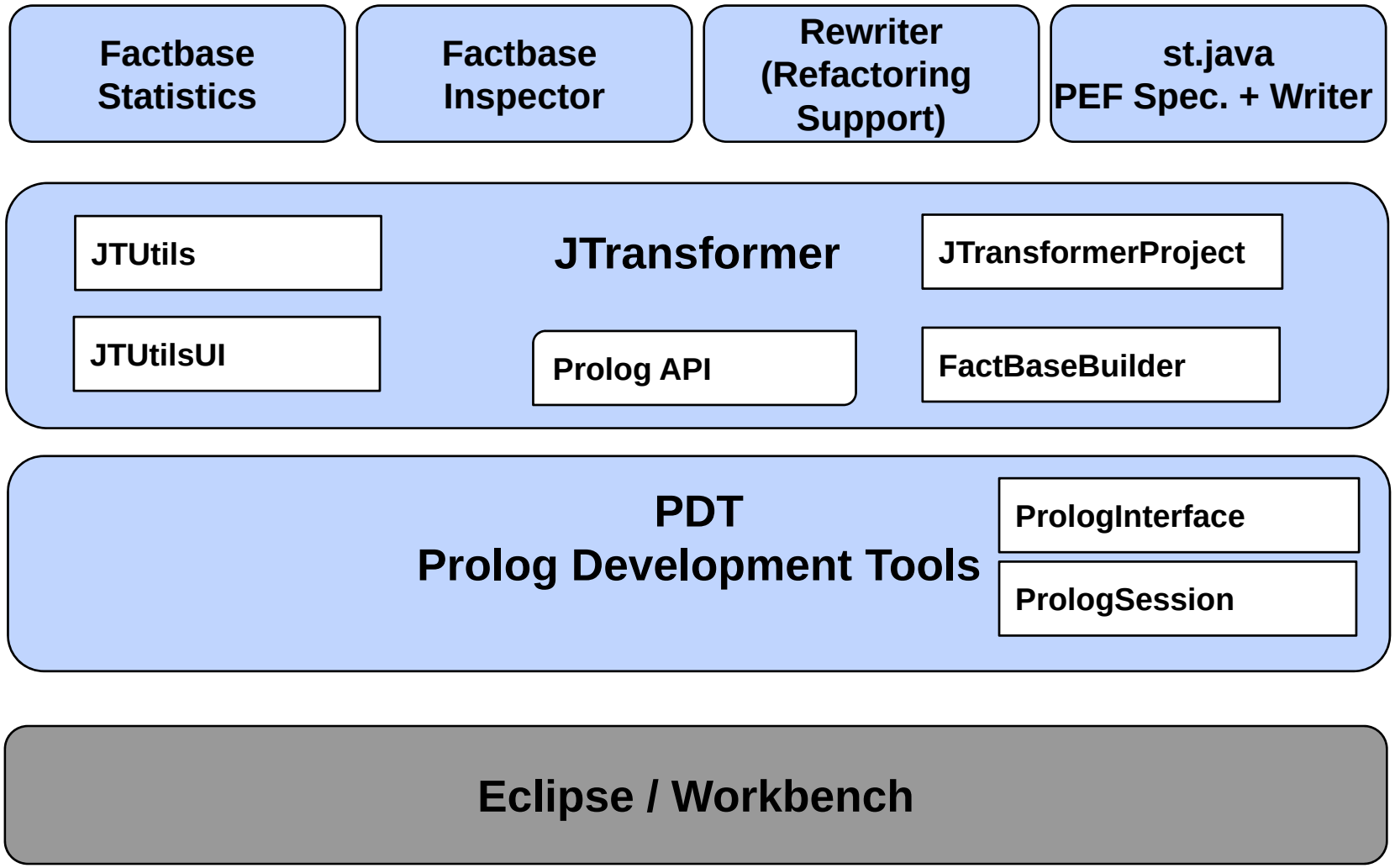
Tobias Rho
rho@cs.uni-bonn.de

ROOTS/SAM Group
Computer Science Department III
University of Bonn

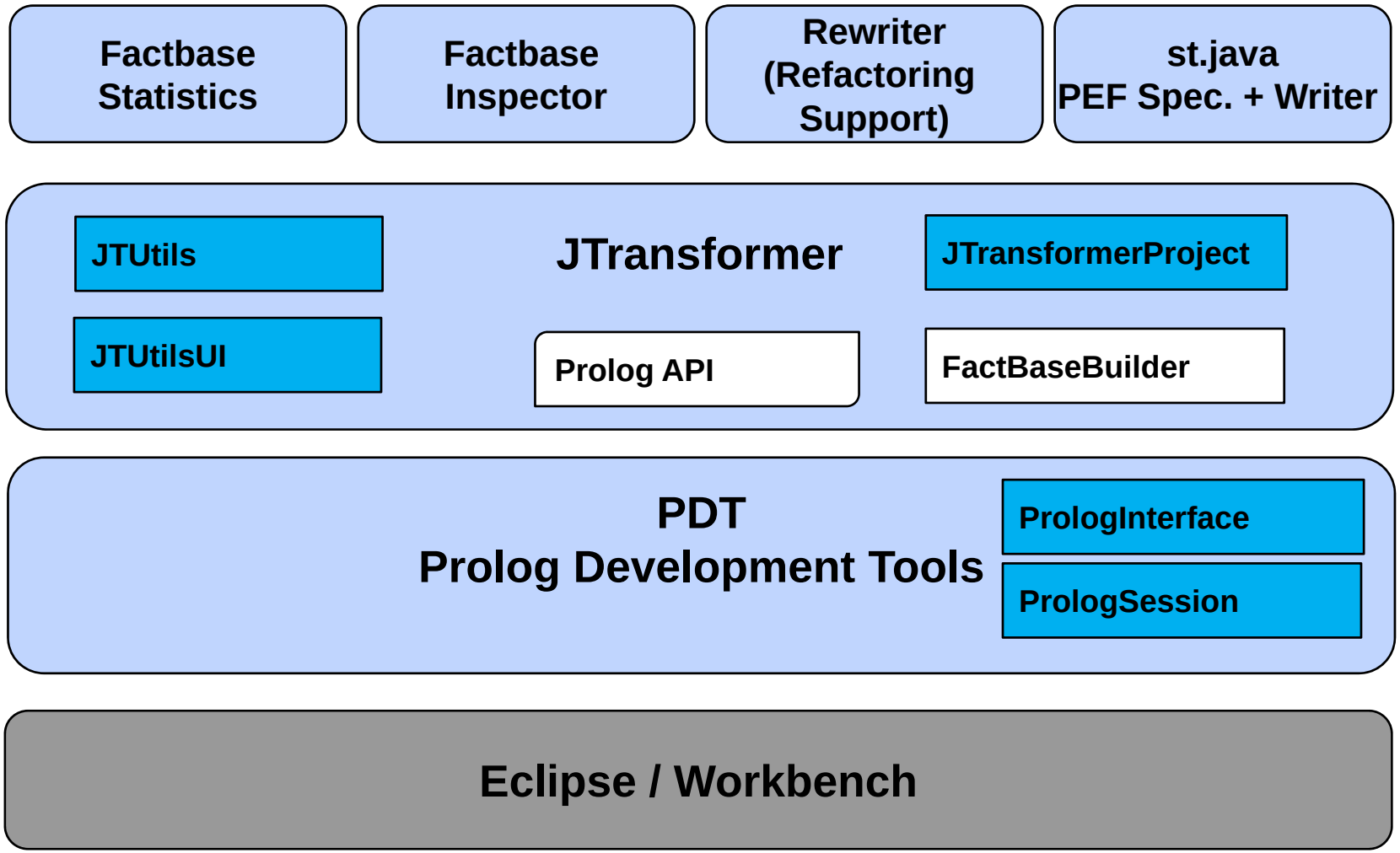
Workbench Terminology



Important Plug-ins and Classes



JTransformer Core APIs



PrologInterface + PrologSession

- PrologInterface
  - Java Interface to a Prolog fact base
  - Managed by the PDT
    - Initalization, loading of necessary prolog files (more at a later time in the course)
  - Uses a PrologSession to interact with the prolog system
    - PrologInterface.getSession()
    - The first time the getSession method is called the corresponding prolog process (the fact base) is started and initalized (lazy initialization)
- PrologSession
  - Most important methods:
    - queryOnce(String query)
    - queryAll(String query)
    - dispose()

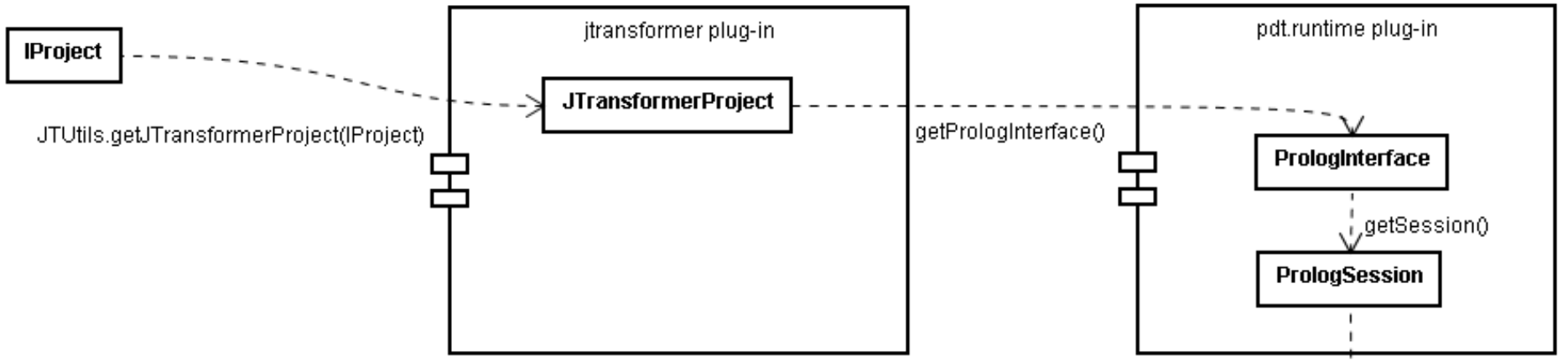
PrologSession Example

```
PrologSession session = null;
try {
  session = pif.getSession();
  Map result = session.queryOnce(
    "jt_factbase_statistics(NumAllClasses,NumAllSrcClasses,Statistics)");
  int numAllClasses = Integer.parseInt((String) result.get("NumAllClasses"));
  Object[] statistics = ((List)result.get("Statistics")).toArray();
  // ... rest of the method ...
} finally {
  if (session != null)
    session.dispose();
}
```

Accessing the Prolog Process …
… if you have the name of factbase

```
private PrologInterface pif = JTUtils.getPrologInterface(“testproject”);
```

Accessing the Prolog Process …
… if you have a reference to an Eclipse (I)Project




```
List<IProject> projects =
    JTUtils.getProjectsWithJTransformerNature();
for (IProject project : projects) {
    pif = JTUtils.getPrologInterface(project);
    // …
        session = pif.getSession();
    // …

}
```

PrologSession API

- Two main methods are provided
  - Map<String,Object> queryOnce(String) – returns the first substitution
  - Set<Map> queryAll(String) – returns all substitutions
- Map<String,Object> map =
  (Map<String,Object>)session.queryOnce(“query”);
- The Map contains variable names as keys and Strings or lists of Objects as values.
- Lists are mapped to List<Object> all other terms are mapped to String(s)
- Example

```
Map <String,Object> map =
  (Map<String,Object>)session.queryOnce("[a,b] = [V1|V2]");
String a = (String)map.get("V1");
List b = (List)map.get("V2");
```

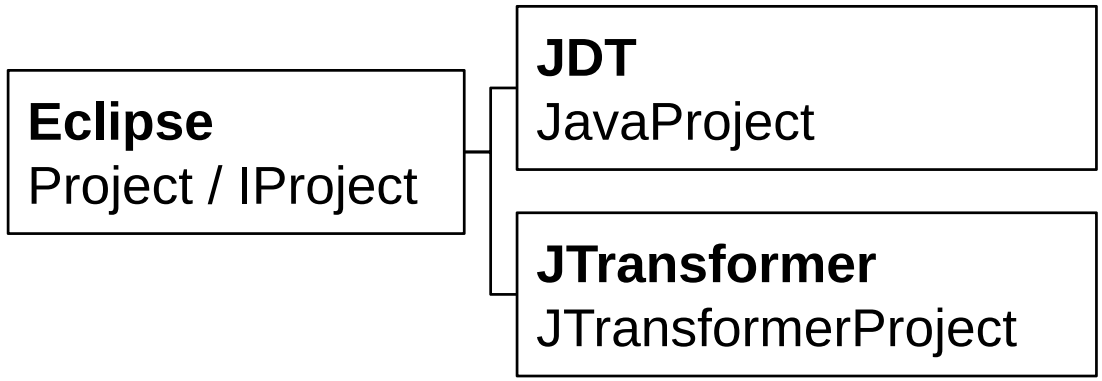
JTUtils / JTUtilsUI

- JTUtils provides helper methods for
  - Retrieving the PrologInterface in various ways
  - Finding out about existing fact bases
  - Retrieve the location of the output project
  - Delete error markers
  - String operations
  - File copy operations
  - Still needs refactorings!

- JTUtilsUI provides helper methods for
  - Get active workbench parts
  - Displaying log messages
  - Selecting parts in the editor
  - Showing/opening a view
  - Setting status messages

JTransformerProject

- Façade to classes associated with a JTransformer project
  - PrologInterface
  - Factbase builder
  - Attaching/Detaching Listeners for a project



JTransformer Testing Infrastructure

JUnit Runtime Test Cases with JTransformer

JTransformer Testing Infrastructure

- The junit test case org.cs3.jtransformer.tests.FactGenerationTest sets up a runtime workspace with a sample Java project and enabled JTransformer nature (assigned factbase)
- Sets up a test project and offers some convenience methods.
- It creates a project named "testproject" in the runtime workspace and offers methods to install test files into this project.
- The test project has a single package fragment root which is the project folder itself.
- To retrieve the PrologInterface for the test project use getPrologInterface()

JTransformer Testing Infrastructure
Configuring subtypes of FactGenerationTest

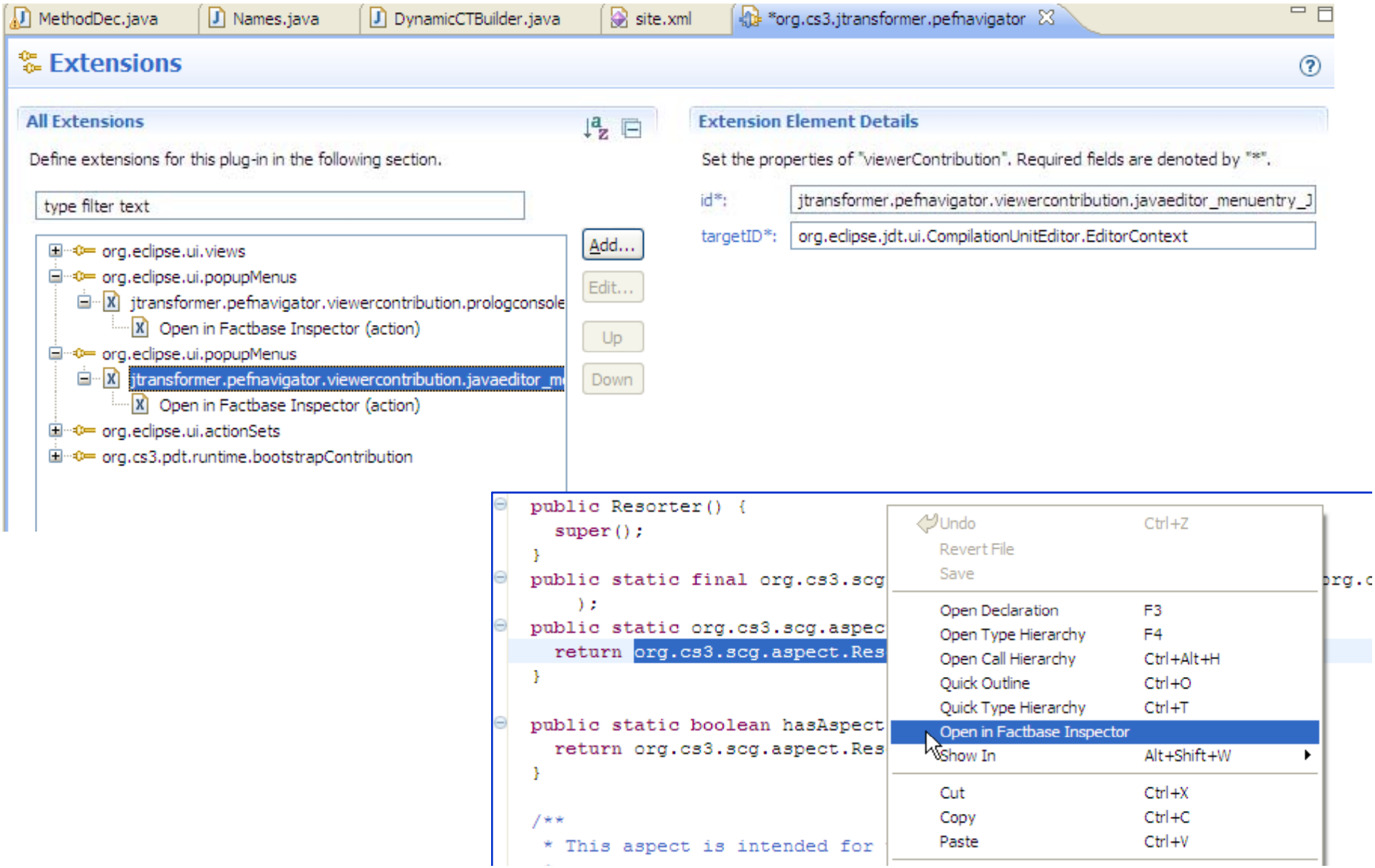
- Override the setUp method to specify the java files to add to the project
- The following snippet unpacks the directory *factbaseview* located in the zip file *testdata-project.zip* in the jtransformer plug-in into the test project's root directory

```
protected synchronized void setUp() throws Exception {
    super.setUp();
    unpackTestData(
      JTransformerPlugin.getDefault(),
      "testdata-project.zip","factbaseview");
      getTestJTransformerProject().ensurePefInitializationJobStarted();
    }
}
```

- The last line waits for the completion of the fact generation

Extending Eclipse
Pop-up Menus

Extension Point popupMenus
Add a Context Menu Entry to Editors/Views



Extension Point popupMenus
jtransformer.pefnavigator/plugin.xml

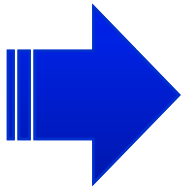
```
<extension
  point="org.eclipse.ui.popupMenus">
  <viewerContribution
    id="jtransformer.pefnavigator.viewercontribution.prologconsole"
    targetID="org.cs3.pdt.console.internal.views.PrologConsoleView">
    <action
      class="org.cs3.jtransformer.pefnavigator.internal.view.GotoPEFNavigatorAction“
      id="jtransformer.action.open_in_pef_navigator"
      label="Open in Factbase Inspector“
      menubarPath="additions"/>
   </viewerContribution>
</extension>
<extension point="org.eclipse.ui.popupMenus">
  <viewerContribution

id="jtransformer.pefnavigator.viewercontribution.javaeditor_menuentry_JTinspector“
    targetID="org.eclipse.jdt.ui.CompilationUnitEditor.EditorContext">
    <action
      class="org.cs3.jtransformer.pefnavigator.internal.view.EditorContextMenuAction“
      id="jtransformer.action.java_editor_open_in_pef_navigator“
      label="Open in Factbase Inspector"
      menubarPath="group.show">
    </action>
  </viewerContribution>
</extension>
```

Select the corresponding Source Code of a Prolog Element Fact in the Java Editor

```
int id = 10701;
Map result =
session.queryOnce("sourceLocation("+id+", File, Start, Length)");

String filename = result.get("File").toString();
int start = Integer.parseInt(result.get("Start").toString());
int length = Integer.parseInt(result.get("Length").toString());
JTUtilsUI.selectInEditor(start, length, filename);
```



```
public Resorter() {
  super();
}
public static final org.cs3.scg.aspect.Resorter aspectInstance = new org.cs3.
    );
public static org.cs3.scg.aspect.Resorter aspectOf() {
  return org.cs3.scg.aspect.Resorter.aspectInstance;
}

public static boolean hasAspect() {
  return org.cs3.scg.aspect.Resorter.aspectInstance != null;
}

/**
```

Extending JTransformer
Hello World Plug-in

Creating a new Plugin

- Create a new Plug-in Project
  - Select "File->New->Project..." from the main menu, expand the "Plug-in Development" category within the resulting "New Project" dialog box, and choose "Plug-in Project“
  - Input "com.example.helloworld" into the "Project Name" text box
  - Accept the rest of the default preloaded values on the "Plug-in Project" page and click "Next >“
  - Accept the default preloaded values on the "Plug-in Content" page and click "Next >“
  - Select the "Hello, World" template from the list of "Available Templates" and click "Finish"
- Add required plug-ins:
  - org.eclipse.core.resources
  - org.cs3.jtransformer
  - org.cs3.pdt.runtime

Modify the action

- Replace the run method body in the class SampleAction with the following code:

```
IProject project =
ResourcesPlugin.getWorkspace().getRoot().getProject("Test");

PrologInterface pif;
try {
  pif = JTUtils.getPrologInterface(project);
  PrologSession session = pif.getSession();
  Map map = session.queryOnce("atom_concat(a,b,C)");
  MessageDialog.openInformation(window.getShell(),
         "concatinated a and b",
         "result: " + map.get("C"));

} catch (Exception e) {
  e.printStackTrace();
}
```

Test the new plug-in

- Select Run->Open Run Dialog
- Right-click on “Eclipse Application”->New
- Click “run” to start a second Eclipse workspace with your new project
- Create a Java project named “Test” in your “runtime” workspace
- Right-click on the project and select “Assign JTransformer Factbase”
- Now test your new Menu Entry:
  - Sample Menu->Sample Action