**[Eclipse JDT Tutorials](http://www.programcreek.com/2011/01/best-java-development-tooling-jdt-and-astparser-tutorials/)**

  By [X Wang](https://plus.google.com/+XWang)

JDT is supported by 3 pillars: Java Model, Search Engine, and AST. It is a handy tool for manipulating Java source code. However, the handy tool comes with a steep learning curve even for an experienced Java developer and there are just not enough proper code examples. This page summarizes the code examples for using Eclipse JDT.

* Java Model
  + [Java Model – Concept and Design](http://www.programcreek.com/2011/01/2012/02/eclipse-jdt-tutorial-java-model-concept-and-design/)
  + [Java Model – Create, Access, Load Projects](http://www.programcreek.com/2011/01/2011/05/eclipse-jdt-tutorial-java-model/)
  + [Traverse .jar File by Using Eclipse JDT](http://www.programcreek.com/2012/06/traverse-jar-file-by-using-eclipse-jdt/)
  + [Count Total Number of Methods in a Java Project](http://www.programcreek.com/2012/06/count-total-number-of-methods-in-a-java-project/)
* Java Search Engine
  + [A very simple example](http://www.programcreek.com/2012/02/eclipse-jdt-tutorial-java-search-engine-example/)
  + [Search method references](http://www.programcreek.com/2012/05/eclipse-jdt-tutorial-find-all-references-of-a-method/)
* ASTParser - Parse projects, files, and methods
  + [Use ASTView to explore AST of a Java class](http://www.programcreek.com/2012/04/represent-a-java-file-as-an-astabstract-syntax-tree/)
  + [Parse a Java Project in Workspace](http://www.programcreek.com/2011/01/2011/08/code-to-parse-a-java-project/)
  + [Parse Single .java files](http://www.programcreek.com/2011/01/2011/11/use-jdt-astparser-to-parse-java-file/)
  + [Parse a Java method](http://www.programcreek.com/2011/01/2011/01/parse-a-java-method-by-using-astparser/)
  + [Parse Java statements](http://www.programcreek.com/2012/08/parse-a-sequence-of-java-statements-by-using-jdt-astparser/)
  + [A complete standalone example of ASTParser](http://www.programcreek.com/2011/01/2011/01/a-complete-standalone-example-of-astparser/)
* More Examples and Possible Issues
  + [Get Internal Comments By Using Eclipse JDT ASTParser](http://www.programcreek.com/2013/03/get-internal-comments-by-using-eclipse-jdt-astparser/)
  + [How to format source code by using CodeFormatter?](http://www.programcreek.com/2013/04/how-to-format-java-code-by-using-eclipse-jdt/)
  + [Insert Blank Lines to source code](http://www.programcreek.com/2012/06/insert-blank-lines-to-source-code-by-using-eclipse-jdt/)
  + [Add a comment to source code](http://www.programcreek.com/2012/06/add-comments-by-using-eclipse-jdt-astrewrite/)
  + [Add a statement to source code](http://www.programcreek.com/2012/06/insertadd-statements-to-java-source-code-by-using-eclipse-jdt-astrewrite/)
  + [Dynamically load a large number of open source project to workspace](http://www.programcreek.com/2012/02/eclipse-jdt-tutorial-dynamic-load-projects-to-workspace/)
  + [Find all callers of a method – get all methods that call a particular method](http://www.programcreek.com/2011/01/2011/07/find-all-callers-of-a-method/)
  + [Count Number of Statements in a Java Method By Using Eclipse JDT ASTParser](http://www.programcreek.com/2011/01/2011/07/java-count-number-of-statements-in-a-method/)
  + [java.lang.IllegalStateException: Workspace is closed.](http://www.programcreek.com/2011/01/2011/05/java-lang-illegalstateexception-workspace-is-closed/)
* ===================================================

# [Eclipse JDT tutorial – Java Model – Concept and Design](http://www.programcreek.com/2012/02/eclipse-jdt-tutorial-java-model-concept-and-design/)

  By [X Wang](https://plus.google.com/+XWang)

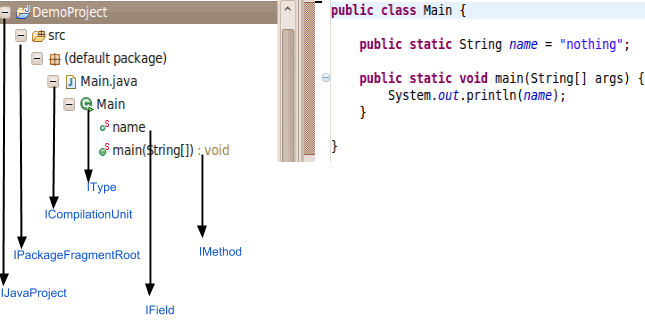
This article belongs to [Eclipse JDT Tutorial Series](http://www.programcreek.com/2011/01/best-java-development-tooling-jdt-and-astparser-tutorials/).

**What is Java Model?**

*The Java model is the set of classes that model the objects associated with creating, editing, and building a Java program.*

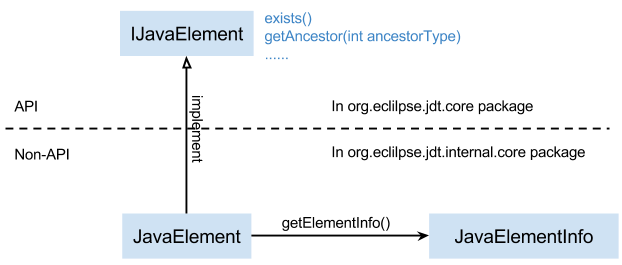
It is defined in org.eclipse.jdt.core package. Accordingly, non-API classes are defined in package org.eclipse.jdt.internal.core. You will see their relation in next section.

In the diagram below, a simple project is displayed in the Package Explore view. Each element is mapped to some IJavaElement. All those classes directly or not directly extend IJavaElement class, and there are many other classes other than what are shown in the diagram. If you want to see more, you can go to reference 2 to check out the entire hierarchy.



**The Handle/Info Design**

Let's take IJavaElement for example, since it is the common protocol for all elements provided by the Java model. Other Java elements have the same design(e.g. IType, IMethod, etc). Java model elements are exposed to clients as handles to the actual underlying element.



Actually, the design pattern here is a mixture of [Proxy](http://www.programcreek.com/2009/10/proxy-design-pattern-in-a-funny-story/) and [Bridge](http://www.programcreek.com/2011/10/java-design-pattern-bridge/). Those patterns are not exactly the same as how they are defined. You can follow the links to see what is a canonical implementations of Proxy and Bridge pattern.

The advantage of this design:  
1. The handle (IJavaElement) is a key for an element. They define behaviors of each element, but do not keep any state information.  
2. An info object stores the element's state. There is only one implementer (JavaElement) for a handle, so it is a simplified Bridge.

\* Eclipse core resources have the similar design.

References:

1. [JDT APT Specification](http://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Foverview-summary.html)

2. [Hierarchy For Package org.eclipse.jdt.core](http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/package-tree.html)

# [Eclipse JDT tutorial – Java Model – Create, Access, Load Projects](http://www.programcreek.com/2011/05/eclipse-jdt-tutorial-java-model/)

  By [X Wang](https://plus.google.com/+XWang)

In this article, we are going to use Eclipse JDT to create, access and load projects. I assume that you know how to create a simple Eclipse plug-in project which adds a menu item that you can click and trigger some actions. If you don't know, you can go to this [article](http://www.programcreek.com/2011/07/eclipse-plugin-development-add-a-menu/). The reason why we need a plug-in project is that Java Model only work inside of a Plug-in, a standalone application will not support Java Model.

This article only focus on JDT Java Model. The following three topics will be explored:

* Create projects in workspace
* Access projects in workspace
* Dynamically import existing projects into workspace

Those are essentially important when you want to process a large number of Java projects.

**1. Create projects**

We can use Java Model to create a new project in the work space.

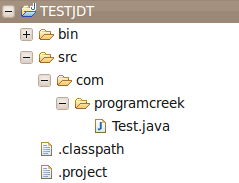
The following here requires the following dependencies:

|  |
| --- |
| **import** org.eclipse.core.resources.IFolder;  **import** org.eclipse.core.resources.IProject;  **import** org.eclipse.core.resources.IProjectDescription;  **import** org.eclipse.core.resources.IWorkspaceRoot;  **import** org.eclipse.core.resources.ResourcesPlugin;  **import** org.eclipse.core.runtime.CoreException;  **import** org.eclipse.jdt.core.IClasspathEntry;  **import** org.eclipse.jdt.core.ICompilationUnit;  **import** org.eclipse.jdt.core.IJavaProject;  **import** org.eclipse.jdt.core.IPackageFragment;  **import** org.eclipse.jdt.core.IPackageFragmentRoot;  **import** org.eclipse.jdt.core.IType;  **import** org.eclipse.jdt.core.JavaCore;  **import** org.eclipse.jdt.core.JavaModelException;  **import** org.eclipse.jdt.launching.JavaRuntime; |

Add code to run method. The code ignores the try/catch statements, Eclipse will ask you to add exception handling code.

|  |
| --- |
| *// create a project with name "TESTJDT"*  IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();  IProject project = root.getProject("TESTJDT");  project.create(**null**);  project.open(**null**);    *//set the Java nature*  IProjectDescription description = project.getDescription();  description.setNatureIds(**new** String[] { JavaCore.NATURE\_ID });    *//create the project*  project.setDescription(description, **null**);  IJavaProject javaProject = JavaCore.create(project);    *//set the build path*  IClasspathEntry[] buildPath = {  JavaCore.newSourceEntry(project.getFullPath().append("src")),  JavaRuntime.getDefaultJREContainerEntry() };    javaProject.setRawClasspath(buildPath, project.getFullPath().append(  "bin"), **null**);    *//create folder by using resources package*  IFolder folder = project.getFolder("src");  folder.create(**true**, **true**, **null**);    *//Add folder to Java element*  IPackageFragmentRoot srcFolder = javaProject  .getPackageFragmentRoot(folder);    *//create package fragment*  IPackageFragment fragment = srcFolder.createPackageFragment(  "com.programcreek", **true**, **null**);    *//init code string and create compilation unit*  String str = "package com.programcreek;" + "**\n**"  + "public class Test {" + "**\n**" + " private String name;"  + "**\n**" + "}";    ICompilationUnit cu = fragment.createCompilationUnit("Test.java", str,  **false**, **null**);    *//create a field*  IType type = cu.getType("Test");    type.createField("private String age;", **null**, **true**, **null**); |

When you trigger the action, the following project will be created.

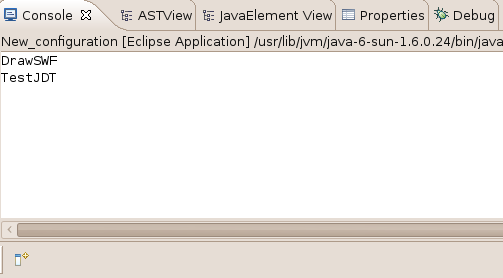


**2. Access projects**

If there are already projects in our work space, we can use Java Model to loop through each of them.

|  |
| --- |
| **public** **void** run(IAction action) {  *// Get the root of the workspace*  IWorkspace workspace = ResourcesPlugin.getWorkspace();  IWorkspaceRoot root = workspace.getRoot();  *// Get all projects in the workspace*  IProject[] projects = root.getProjects();  *// Loop over all projects*  **for** (IProject project : projects) {  System.out.println(project.getName());  }    } |

If we import some projects or create some, and click the menu item we created, the projects names will show up as follows.

[](http://www.programcreek.com/wp-content/uploads/2011/05/9.png)

**3. Dynamically load/import existing projects into workspace**

In the previous step, we need manually import existing projects to work space. If the number is larger, this would not be applicable.

Eclipse JDT provide functions to do this dynamically. Now let's see how to import a large number of existing projects into the work space. It does not copy files to the workspace root directory, but only point to the projects in the external directory. In the example, I use the flash drive to hold my open source projects. In this way, you can parse thousands of projects and get useful information you need without copying anything.

|  |
| --- |
| IWorkspaceRoot root= ResourcesPlugin.getWorkspace().getRoot();    **final** IWorkspace workspace = ResourcesPlugin.getWorkspace();    System.out.println("root" + root.getLocation().toOSString());    Runnable runnable = **new** Runnable() {  **public** **void** run() {  **try** {  IPath projectDotProjectFile = **new** Path("/media/flashx/TestProjectImport" + "/.project");  IProjectDescription projectDescription = workspace.loadProjectDescription(projectDotProjectFile);  IProject project = workspace.getRoot().getProject(projectDescription.getName());  JavaCapabilityConfigurationPage.createProject(project, projectDescription.getLocationURI(), **null**);  *//project.create(null);*  } **catch** (CoreException e) {  e.printStackTrace();  }  }  };    *// and now get the workbench to do the work*  **final** IWorkbench workbench = PlatformUI.getWorkbench();  workbench.getDisplay().syncExec(runnable);      IProject[] projects = root.getProjects();    **for**(IProject project: projects){  System.out.println(project.getName());  } |

What if the project we want to load does not contain a .project file? This is the complicated case, we need dynamically create all those projects by using its source code.

**Notes**

When you practice the examples above, you may got error message like "The type org.eclipse.core.runtime.IAdaptable cannot be resolved. It is indirectly referenced from required .class files". The solution is adding org.eclipse.core.runtime through Plug-in Menifest Editor. Simply adding to build path will not work.

If you think this article is useful and want to read more, you can go to [Eclipse JDT Tutorial Series](http://www.programcreek.com/2011/01/best-java-development-tooling-jdt-and-astparser-tutorials/) I wrote.