Eclipse – 整合开发工具(基础篇)

Jacky Lee

2005/03/01

目录

0.环境说明 7
1.Eclipse 简介 8
1.1 历史背景 8
1.2 开放原始码软件 8
1.3 Eclipse 版本介绍 8
1.4 跨语言、跨平台 9
2. Eclipse Platform 10
2.1 概观 10
2.2 架构 10
2.3 项目与资料夹 10
2.4 平台核心 11
2.5 工作区(workspace) 11
2.6 工作台(workbench) 11
2.6.1 视图(View) 12
2.6.2 编辑器(Editor) 14
2.6.3 视景(Perspective) 16
2.7 重新排列视图和编辑器 17
2.7.1 放置游标 17
2.7.2 重新排列视图 18
2.7.3 并列编辑器 18
2.7.4 重新排列附加标签的视图 19
2.7.5 最大化 19
2.8 菜单和工具列 20
2.8.1 菜单 20
2.8.2 图标和按钮 31
2.9 视景 36
2.9.1 新视景 36
2.9.2 新窗口 37
2.9.3 储存视景 38
2.9.4 配置视景 40
2.10 作业和标记 41
2.10.1 不相关的作业 41
2.10.2 相关的作业 42
2.10.3 开启档案 43
2.11 书签 43
2.11.1 新增和检视书签 43

2.11.2 使用书签 45
2.11.3 移除书签 45
2.12 快速视图(Fast View) 46
2.12.1 建立快速视图 46
2.12.2 使用快速视图 47
2.13 比较 48
2.13.1 简单比较 48
2.13.2 了解比较 49
2.13.3 使用比较 50
2.14 历史纪录 52
2.15 回应 UI 53
3.喜好设定(Preferences) 56
3.1 工作台(Workbench) 56
3.1.1 外观(Appearance) 58
3.1.2 功能(Capabilities) 59
3.1.3 颜色和字型(Colors and Fonts) 60
3.1.4 比较/修正(Compare/Patch) 62
3.1.5 编辑器(Editors) 64
3.1.6 档案关联(File Associations) 65
3.1.7 按键(Keys) 67
3.1.8 标签装饰(Label Decorations) 73
3.1.9 链接资源(Linked Resources) 73
3.1.10 历史纪录(Local History) 74
3.1.11 视景 75
3.1.12 搜寻(Search) 77
3.1.13 启动和关闭(Startup and Shutdown) 77
3.2 Ant 78
3.2.1 Ant 编辑器(Ant Editor) 79
3.2.2 Ant 执行时期(Ant Runtime) 80
3.3 建置次序(Build Order) 82
3.4 说明(Help) 83
3.4.1 说明服务器(Help Server) 84
3.5 自动更新(Install/Update) 85
3.6 Java 86
3.6.1 外观(Appearance) 86
3.6.2 类别路径变量(Classpath variables) 87
3.6.3 程序代码格式制作器(Code Formatter) 87
3.6.4 程序代码产生(Code generation) 89

3.6.5 编译器(Compiler)..... 91
3.6.6 Java 编辑器(Java editor)..... 95
3.6.7 JRE 安装(JRE installations)..... 100
3.6.8 JUnit..... 101
3.6.9 新专案(New project)..... 101
3.6.10 组织汇入(Organize imports)..... 102
3.6.11「重构」喜好设定(Refactoring preferences)..... 103
3.6.12 作业标示(Task Tags)..... 103
3.7 团队(Team)..... 103
3.7.1 CVS..... 104
3.7.2 忽略的资源(Ignored Resources)..... 108
3.7.3 档案内容(File Content)..... 108
4. Java 程序开发..... 109
4.1 建立 Java 项目..... 109
4.2 建立 Java 类别..... 111
4.3 程序代码完成功能..... 112
4.3.1 Code Completion..... 112
4.3.2 Code Assist..... 112
4.4 执行 Java 程序..... 113
4.5 Java 实时运算簿页面(Java Scrapbook Page)..... 115
4.6 自订开发环境..... 122
4.6.1 程序代码格式..... 122
4.6.2 程序代码产生模板..... 123
4.6.3 Javadoc 批注..... 125
4.7 产生 getter 与 setter..... 129
4.8 建立 JAR 档案..... 131
4.8.1 建立新的 JAR 档案..... 131
4.8.2 设定进阶选项..... 133
4.8.3 定义 JAR 檔的 manifest..... 134
4.8.4 重新产生 JAR 檔..... 136
4.9.建立 Javadoc 文件..... 138
4.9.1 选取产生 Javadoc 用的类型..... 138
4.9.2 为标准 doclet 配置 Javadoc 自变量..... 139
4.9.3 配置 Javadoc 自变量..... 139
4.10 工作集(Working Sets)..... 140
4.10.1 新增工作集..... 140
4.10.2 隐藏「导览器」视图中的档案..... 142
4.10.3 显示「导览器」视图中的档案..... 143

5.除错 145
5.1 错误的程序 145
5.2 设定岔断点(Breakpoints) 146
5.3 逐步除错 150
5.3.1 Step Into 150
5.3.2 Step Over 150
5.3.3 Step Return 150
5.3.4 Drop to Frame 150
5.3.5 Use Step Filters/Step Debug 151
5.4 继续执行 152
5.5 设定岔断点的 Hit Count 154
5.6 岔断点组态设定 161
5.7 监视点(Watchpoint) 162
5.8 方法岔断断点(Method Breakpoint) 165
5.9 异常岔断点(Exception Breakpoint) 167
5.10 Java 表示式及变更某些值 169
6.重构(Refactoring) 171
6.1 重新命名 171
6.1.1 区域变量(Local Variable) 171
6.1.2 字段(Field) 172
6.1.3 方法(Method) 174
6.1.4 类别(Class)或是接口(Interface) 175
6.1.5 套件(Package) 177
6.2 撷取(Extracting) 178
6.2.1 撷取常数(Extracting a Constant) 178
6.2.2 撷取区域变量(Extracting a Local Variable) 181
6.2.3 撷取方法(Extracting a Method) 183
6.3 列入(Inlining) 188
6.3.1 列入常数(Inlining a Constant) 188
6.3.2 列入区域变量(Inlining a Local Variable) 190
6.3.3 列入方法(Inlining a Method) 192
6.4 变更方法签章(Signature) 194
6.5 移动 Java 元素(Moving Java Elements) 197
6.5.1 字段(Field) 198
6.5.2 Static Members 199
6.6 自行封装字段(Self Encapsulating a Field) 201
7.要诀和技巧(Tips and Tricks) 204
7.1 编辑程序文件(Editing Source) 204

7.2 搜寻(Searching).....209
7.3 程序代码导览和读取(Code navigation and reading).....210
7.4 Java 视图(Java views).....213
7.5 除错(Debugging).....215
7.6 各种(Various).....218

0.环境说明

■ 操作系统

Microsoft Windows XP Professional

Service Pack 2

■ Eclipse 版本

Version：Eclipse 3.0.1 SDK (Release)

Build id：200409161125

File Name：eclipse-SDK-3.0.1-win32.zip

■ 参考数据

Eclipse's Help

O'REILLY Eclipse 整合开发工具

博硕文化 Eclipse 实作手册-活用 Java 整合开发环境

1.Eclipse 简介

Eclipse 就像软件开发者的『打铁铺』，它一开始备有火炉、铁钻与铁锤。就像铁匠会用现有的工具打造新的工具，也能用 Eclipse 打造新工具来开发软件-这些新工具可扩充 Eclipse 的功能。(Eclipse 其中一个卖点就是它的扩充性)

1.1 历史背景

Eclipse 这样功能完整且成熟的开发环境，是由蓝色巨人 IBM 所释出。IBM 花了 4 千万美金来开发这个 IDE(Integrated Development Environment)。第一版 1.0 在 2001 年 11 月释出，随后逐渐受到欢迎。

Eclipse 已经成为开放原始码计划(Open Source Project)，大部分的开发扔然掌握在 IBM 手中，但是有一部份由 eclipse.org 的软件联盟主导。(http://www.eclipse.org)

Eclipse 项目由 Project Management Committee(PMC)所管理，它综观项目全局，Eclipse 项目分成 3 个子项目：
平台-Platform
开发工具箱-Java Development Toolkit(JDT)
外挂开发环境-Plug-in Development Environment(PDE)
这些子项目又细分成更多子项目。例如 Platform 子项目包含数各组件，如 Compare、Help 与 Search。JDT 子项目包括三各组件：User Interface(UI)、核心 (Core)及除错(Debug)。PDE 子项目包含两各组件：UI 与 Core。

1.2 开放原始码软件

Eclipse 是开放原始码，结果很多人在使用的时候都不注重合法权的问题。开放原始码软件让使用者能够取得软件的原始码，有权去修改和散布这个软件。如果想修改软件，这件事的另一面就是，除非其它人对修改后的软件也有相同的权力，否则是不能散布修改后的软件，这种权利和著作权(copyright)相反，开放原始码项目中有时称之为著作义(copyleft)。

有些开放原始码许可书，坚持要求任何和其它开发原始码合组成的软件也必须是开放原始码。然而，Eclipse 使用的开放原始码许可书：公共公众许可书-Common Public License(CPL)作为授权方式，设计上是可以容许商业利益的。CPL 可以容许 Eclipse 和其它开放原始码软件合组时，能够以更严谨的许可书散布软件，以求用于商业途径。CPL 经过 Open Software Initiative(OSI)认证，其内容符合开放原始码授权的需求。

1.3 Eclipse 版本介绍

可以从 eclipse.org 网站(http://www.eclipse.org/downloads)下载，可以发现『最新』与『最好』的版本，这两种版本通常不一样，基本上有四种版本-或建置(build)可供下载：

- **释出版(Release builds)**

由 Eclipse 开发团队所宣称的主要稳定版本。Release builds 经过完整测试，并具有一致性、定义清楚的功能。它的定位就跟上市的商业软件一样。

■ **稳定版(Stable builds)**

比 Release build 新一级的版本，经由 Eclipse 开发团队测试，并认定它相当稳定。新功能通常会在此过渡版本出现。它的定位就跟商业软件的 beta 版一样。

■ **整合版(Integration builds)**

此版本的各个独立的组件已经过 Eclipse 开发团队认定具稳定度，但不保证兜在一起没问题。若兜在一起够稳定，它就有可能晋级成 Stable build。

■ 当日最新版(Nightly builds)

此版本显然是从最新的原始码产生出来的。可想而知，此版本当然不保证它跑起来没问题，搞不好还有严重的 bug。

1.4 跨语言、跨平台

多数人认为 Eclipse 是 Java IDE，不过，当下载 Eclipse 之后，除了有 Java IDE(就是 JDT)，还有 PDE。然而 Eclipse 是万用工具平台。JDT 实际上是 Eclipse 的添加品，也就是外挂程序。Eclipse 本身实际上是指 Eclipse 平台(Eclipse Platform)，除了下载时能取得 Java 工具集以外，还提供各种工具的支持，所以平台本身只是相当小的丨组软件。

如果想开发 Java 程序，用的是 Eclipse 随附的 JDT 外挂程序。如果想开发其它语言的程序，就需要拿到其它外挂程序，诸如 CDT(C Development Toolkit)就可以开发 C/C++程序。

Eclipse 跨计算机语言，也跨人类的语言。相同的外挂机制可用来增加对不同语言的支持，这里使用一种特殊的外挂，叫做外挂程序片断(plug-in fragment)。IBM 以捐出一个语言套件，支持中文(繁体与简体)、法文、德文、意大利文、日文、韩文、葡萄牙文(巴西)与西班牙文。

照理说 Eclipse 以 Java 写成，应该可以在任何的平台执行。但严格来说 Eclipse 不是跨平台的，因为它使用作业平台的原生图形来建置。因此要等 SWT(Standard Widget Toolkit)移植到该平台，Eclipse 才能在那个平台执行。但就现实而言到不是什么大问题，因为 SWT 已经被移植到数个常见平台上了，包括 Windows、Linux/Motif、Linux/GTK2、Solaris、QNX、AIX、HP-UX 与 Mac OS X。

2. Eclipse Platform

Eclipse 平台的目的，是提供多种软件开发工具的整合机制，这些工具会实作成Eclipse 外挂程序，平台必须用外挂程序加以扩充才有用处。Eclipse 设计美妙之处，在于所有东西都是外挂，除了底层的核心以外。这种外挂设计让Eclipse 具备强大扩充性，但更重要的是，此平台提供一个定义明确的机制，让各种外挂程序共通合作(透过延伸点 extension points)与贡献(contributions))，因此新功能可以轻易且无缝地加入平台。

2.1 概观

第一次执行 Eclipse 时，会在Eclipse 目录下建一个 workspace 的目录，根据预设，所有的工作都会存在此目录。若要备份工作目录，只要备份这个目录就行了。若要升级至新版的Eclipse，只要将这个目录拷贝过去即可。

用新版时得看看 release notes，确保它支持前一版的 workspace；若不支持，只要将旧的 workspace 子目录拷贝到新的Eclipse 目录下即可。所有的喜好设定都会保留。

2.2 架构

Eclipse 平台由数种组件组成：平台核心(platform kernel)、工作台(workbench)、工作区(workspace)、团队组件(tram component)以及说明组件(help)。

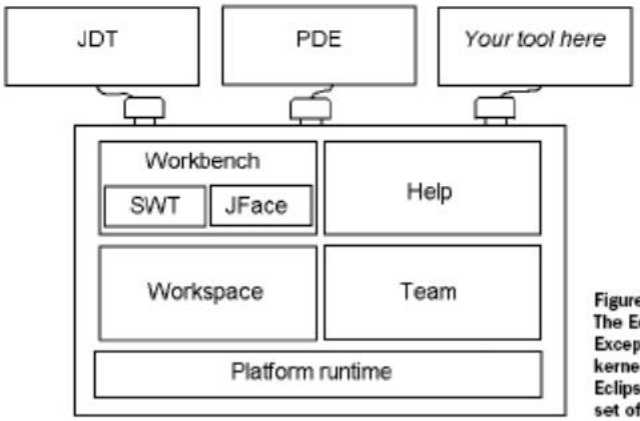


Figure 1.1
The Eclipse architecture. Except for a small runtime kernel, everything in Eclipse is a plug-in or a set of related plug-ins.

图 2.0

2.3 项目与资料夹

若想要手动操作档案、拷贝或看档案大小，就得知道档案放哪里。但原生档案系统会随操作系统而变，这对在各个操作系统均需运作一致的程序会发生问题。为了解决此问题，Eclipse 在档案系统之上提供了一个抽象层级。换句话说，它不使用内含档案的阶层式目录/子目录结构，反之，Eclipse 在最高层级使用『项目』，并在项目之下使用数据夹。

根据预设，『项目』对应到 workspace 目录下的子目录，而『数据夹』对应到项目目录下的子目录。

在 Eclipse 项目内的所有东西均是以独立与平台无关的方式存在。

2.4 平台核心

核心的任务是让每样东西动起来，并加载所需之外挂程序。当启动 Eclipse 时，先执行的就是这个组件，再由这个组件加载其它外挂程序。

2.5 工作区(workspace)

工作区负责管理使用者的资源，这些资源会被组织成一个(或多个)项目，摆在最上层。每个项目对应到 Eclipse 工作区目录下的 1 个子目录。每个项目可包含多个档案和数据夹；通常每个数据夹对应到一个在项目目录下的子目录，但数据夹也可连到档案系统中的任意目录。

每个工作区维护一个低阶的历史纪录，记录每个资源的改变。如此便可以立刻复原改变，回到前一个储存的状态，可能是前一天或是几天前，取决于使用者对历史纪录的设定。此历史纪录可将资源丧失的风险减到最少。

工作区也负责通知相关工具有关工作区资源的改变。工具可为项目标记一个项目性质(project nature)，譬如标记为一个“Java 项目'，并可在必要时提供配置项目资源的程序代码。

2.6 工作台(workbench)

Eclipse 工作台(workbench)就如图 2.1 的画面，这是操作 Eclipse 时会碰到的基本图型接口，工作台是 Eclipse 之中仅次于平台核心最基本的组件，启动 Eclipse 后出现的主要窗口就是这个，workbench 的工作很简单：让操作专案。它不懂得如何编辑、执行、除错，它只懂得如何找到项目与资源(如档案与数据夹)。若有它不能做的工作，它就丢给其它组件，例如 JDT。

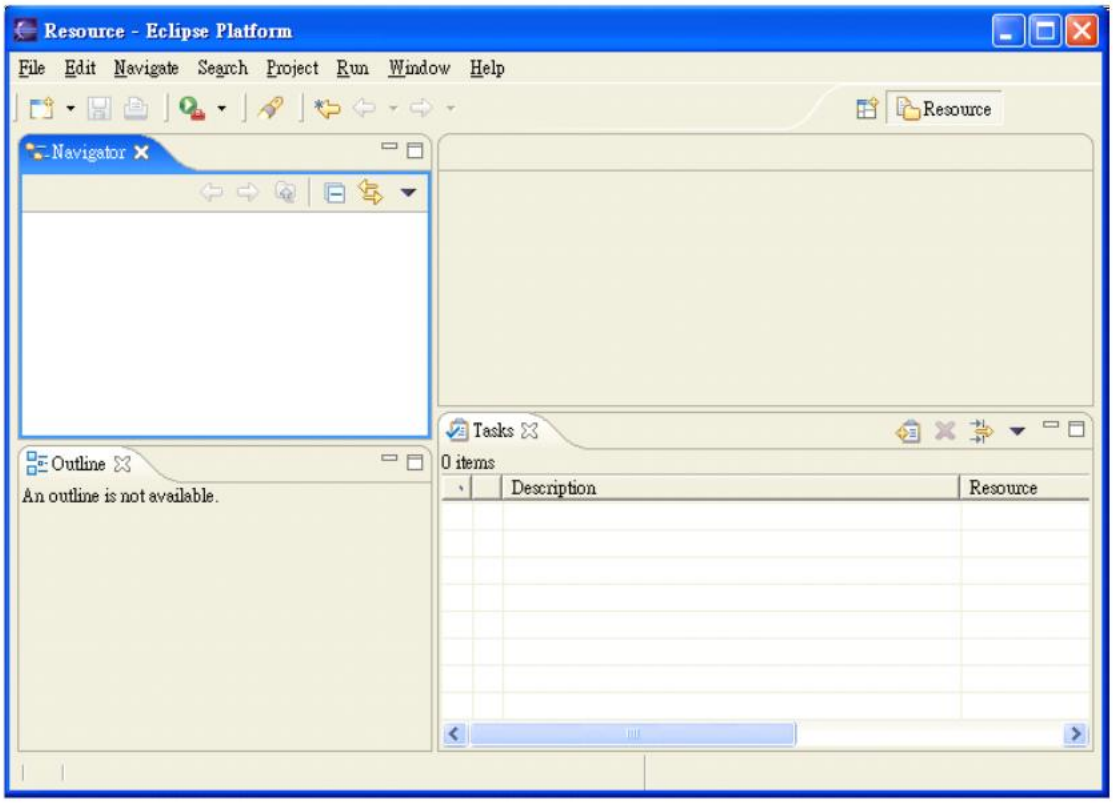


图 2.1

工作台看起来像是操作系统内建的应用程序，可以说是 Eclipse 的特点，同时也是争议点。工作台本身可以说是 Eclipse 的图形操作接口，它是用 Eclipse 自己的标准图形工具箱(Standard Widget Toolkit-SWT)和 JFace(建立在 SWT 之上)的架构。SWT 会使用操作系统的图形支持技术，使得程序的外观感觉(look-and-feel)随操作系统而定。这一点和过去多数 Java 程序的做法很不同，即使是用 Swing，也没有这样过。

2.6.1 视图(View)

工作台会有许多不同种类的内部窗口，称之为视图(view)，以及一个特别的窗口-编辑器(editor)。之所以称为视图，是因为这些是窗口以不同的视野来看整各项目，例如图 2.1，Outline 的视图可以看项目中 Java 类别的概略状况，而 Navigator 的视图可以导览整各项目。

视图支持编辑器，且可提供工作台中之信息的替代呈现或导览方式。比方说：「书签」视图会显示工作台中的所有书签且会附带书签所关联的文件名称。「Navigator」视图会显示项目和其它资源。在已附加卷标的笔记本中，视图可独自呈现，也可以与其它视图形成堆栈。

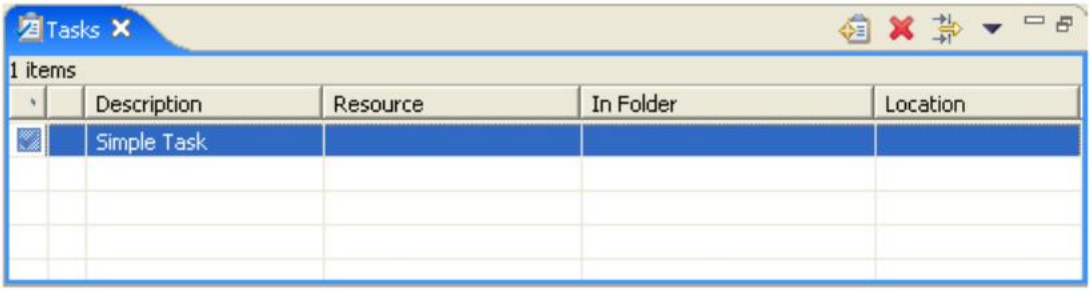


图 2.2

如果要启动在附加卷标的笔记本中的视图，只要按一下标签就行了。工作台会提供了许多又快又简单的方式供配置环境，其中包括卷标在笔记本的底端或顶端。

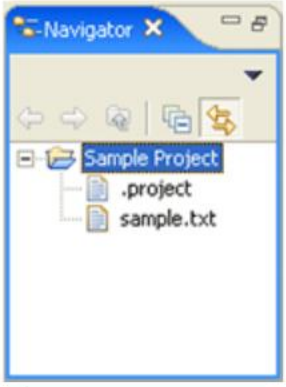


图 2.3

视图有两个菜单，第一个是用鼠标右键按一下视图卷标来存取的菜单，它可以利用类似工作台窗口相关菜单的相同方式来操作视图。

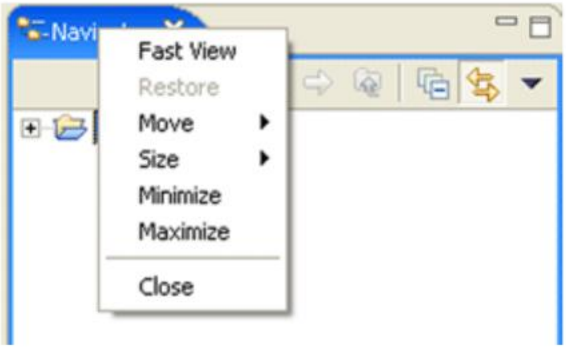


图 2.4

第二个菜单称为「视图下拉菜单」，存取方式是按一下向下箭头 。视图下拉菜单所包含的作业通常会套用到视图的全部内容，而不是套用到视图中所显示的特定项目。排序和过滤作业通常可在检视下拉菜单中找到。

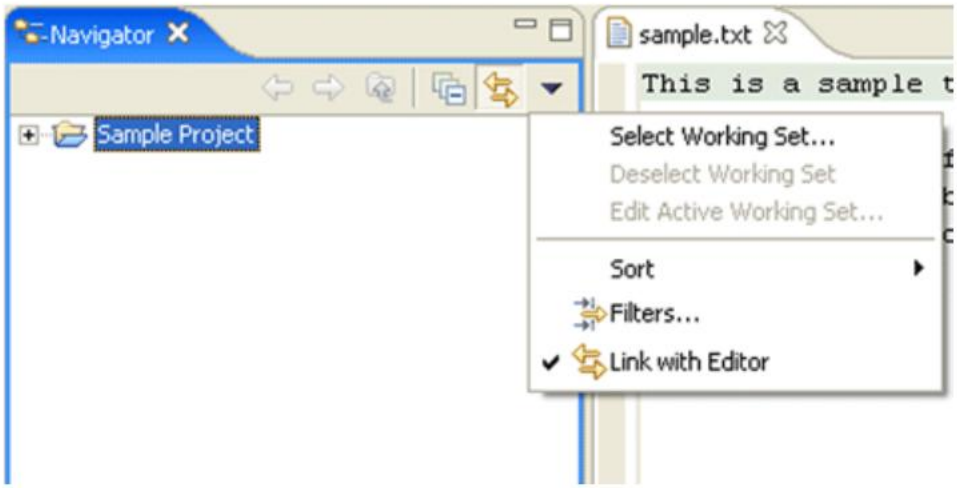


图 2.5

自订工作台是使用「Window」→「Reset Perspective」菜单作业的好时机。重设作业会将布置还原成程序状态。

可以从「Window」→「Show View」菜单中选取一个视图来显示它。视景决定了哪些视图是必要的，它会将这些视图显示在「Show View」子菜单中。选择「Show View」子菜单底端的「Other...」时，就可以使用其它的视图。这只是可用来建立自订工作环境的许多功能之一。

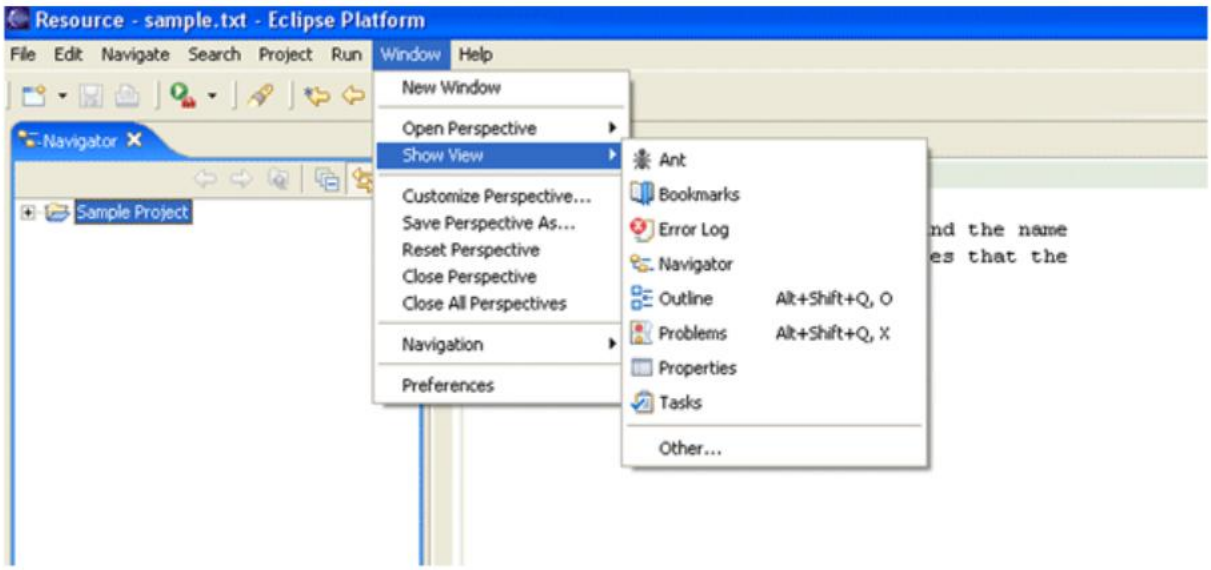


图 2.6

2.6.2 编辑器(Editor)

编辑器是很特殊的窗口，会出现在工作台的中央。当打开文件、程序代码或其它资源时，Eclipse 会选择最适当的编辑器打开文件。若是纯文字文件，Eclipse 就用内建的文字编辑器打开(例如图 2.7)；若是 Java 程序代码，就用 JDT 的 Java 编辑器打开(例如图 2.8)；若是 Word 文件，就用 Word 打开(例如图 2.9)。此 Word 窗口会利用 Object Linking and Embedding-OLE，内嵌在 Eclipse 中。

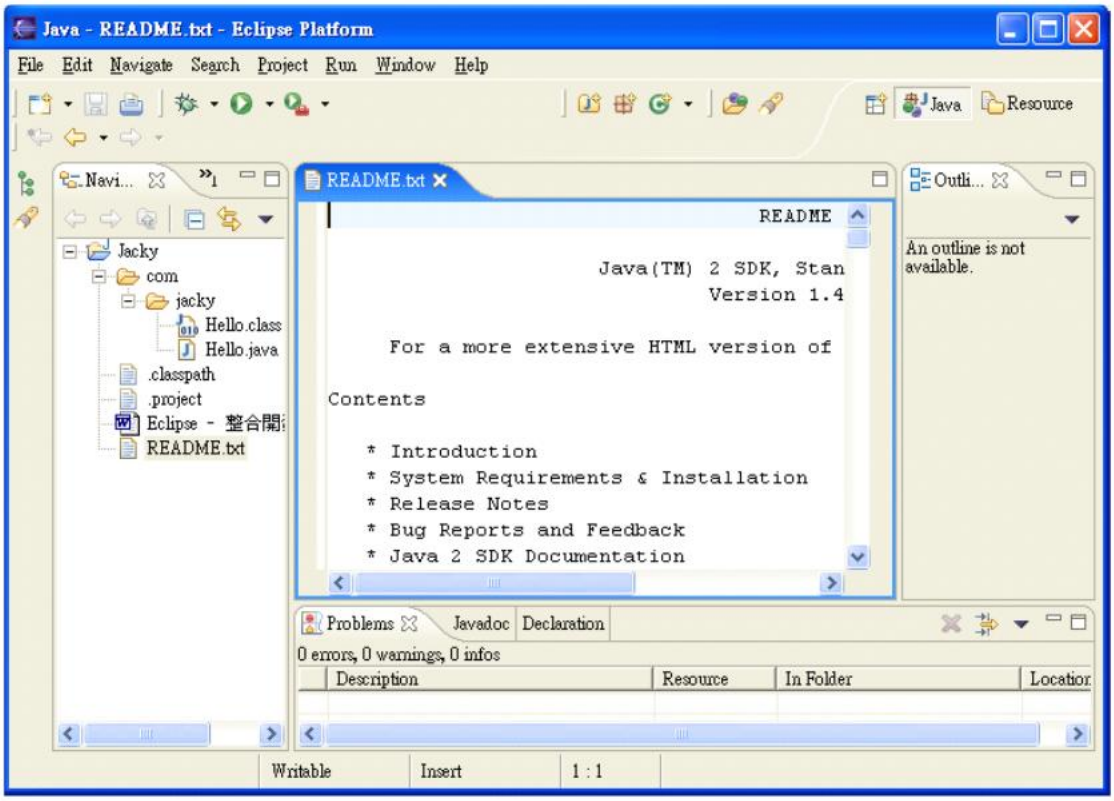


图 2.7

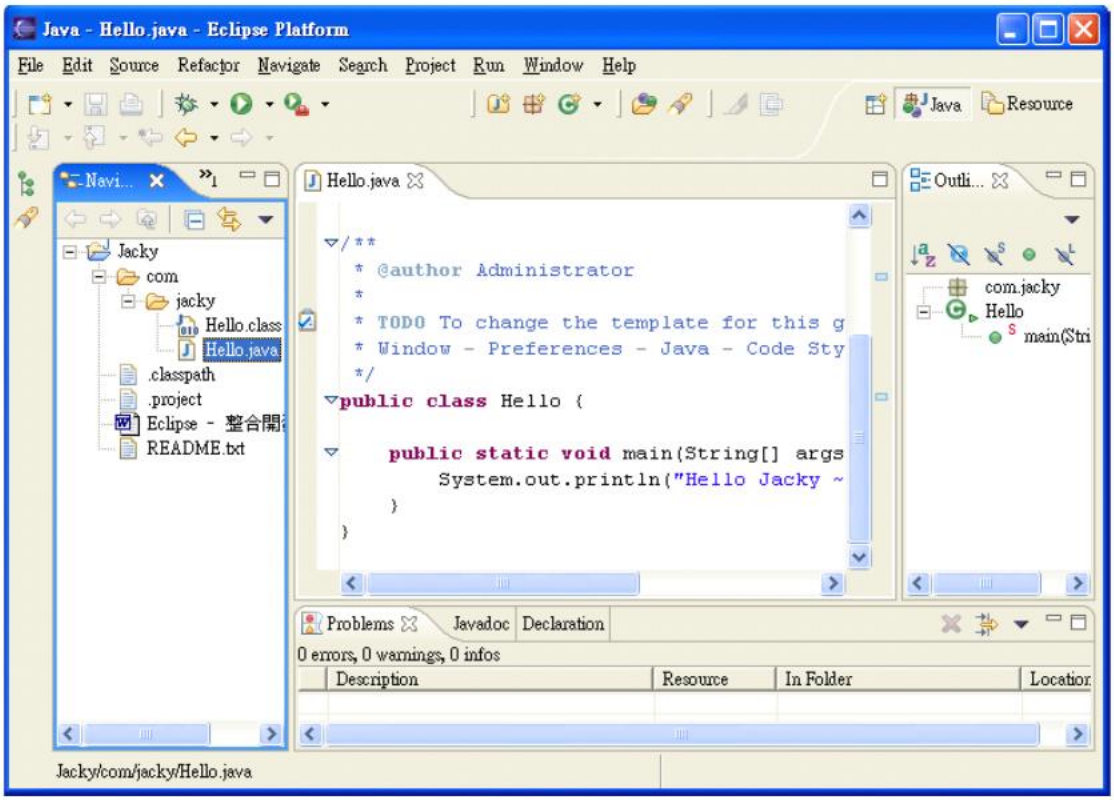


图 2.8

在 Windows 中，工作台会试图启动现有的编辑器，如 OLE(Object Linking and Embedding)文件编辑器。比方说，如果机器中安装了 Microsoft Word，编辑 DOC 档案会直接在工作台内开启 Microsoft Word(例如图 2.9)。如果没有安装 Microsoft Word，就会开启 Word Pad。

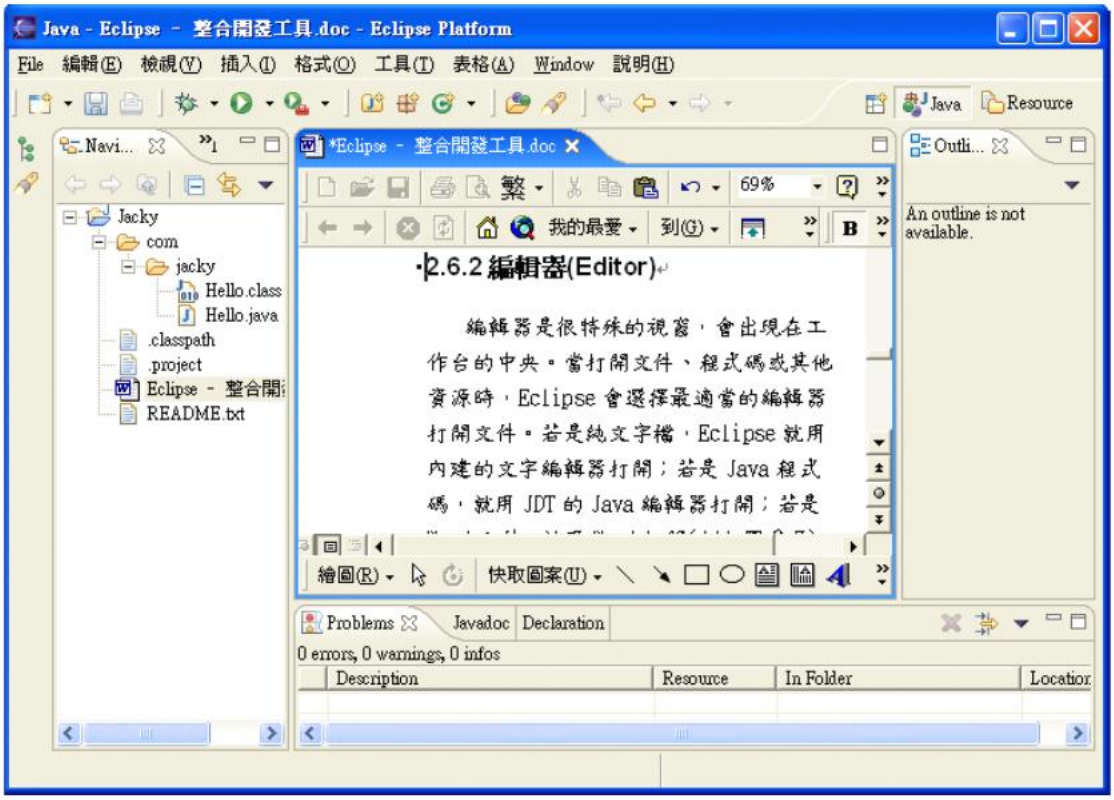


图 2.9

如果标签左侧出现星号 (*)(例如图 2.9)，就表示编辑器有未储存的变更。如果试图关闭编辑器或结束工作台，但没有储存变更，就会出现储存编辑器变更的提示。

工具列中的向后和向前箭头按钮，或利用 Ctrl+F6 加速键来切换编辑器。箭头按钮会移动通过先前的鼠标选取点，可以先通过档案中的多个点，之后才移到另一个点。 Ctrl+F6 会蹦现目前所选取的编辑器清单，依预设，会选取在现行编辑器之前所用的编辑器。(在 Macinosh 中，加速键是 Command+F6。)

2.6.3 视景(Perspective)

Eclipse 提供数群育先选定的视图，并已事先定义好的方式排列，称之为视景(perspective)。所有视景的主要组件式编辑器。

每个视景的目的是执行某特定的工作，如编写 Java 程序，在每个视图以各种不同的观点处理工作，例如图 2.10。

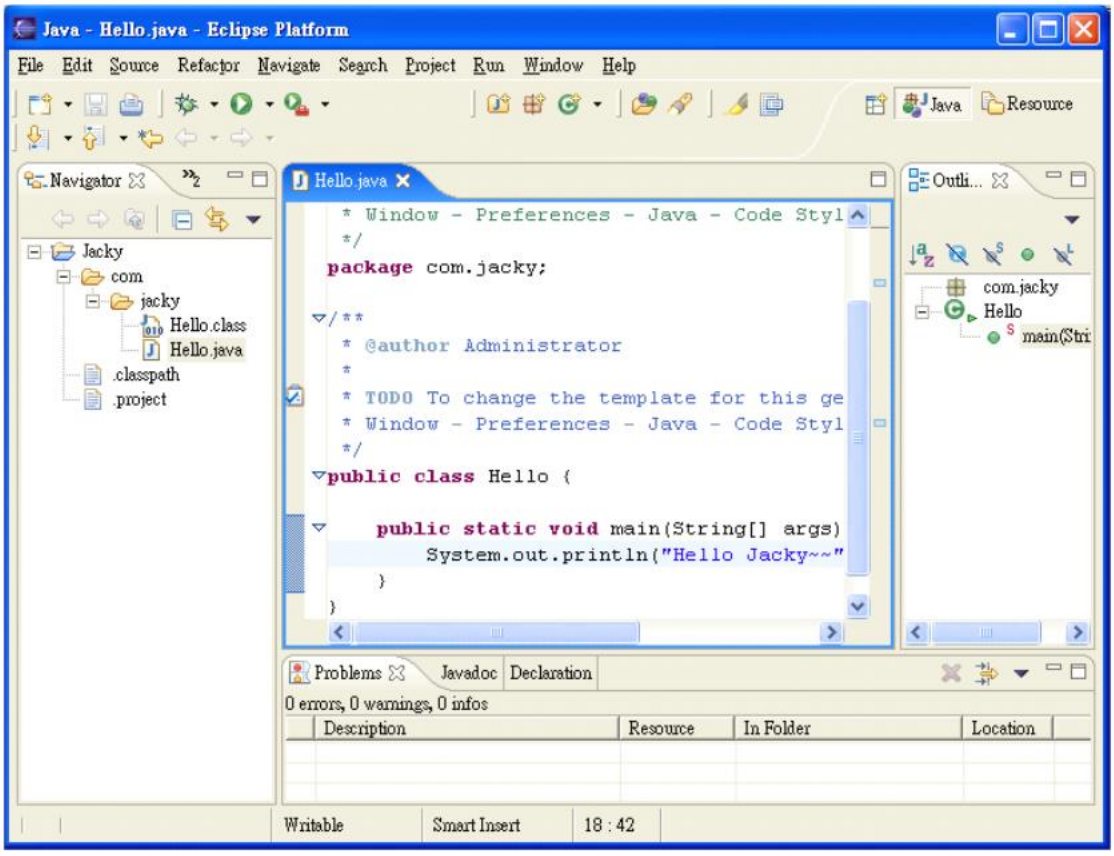


图 2.10

若在 Debug 的视景中，其中一个视图会显示程序代码，另一个可能换显示变量目前的值，还有一个可能会显示程序的执行结果。例如图 2.11。

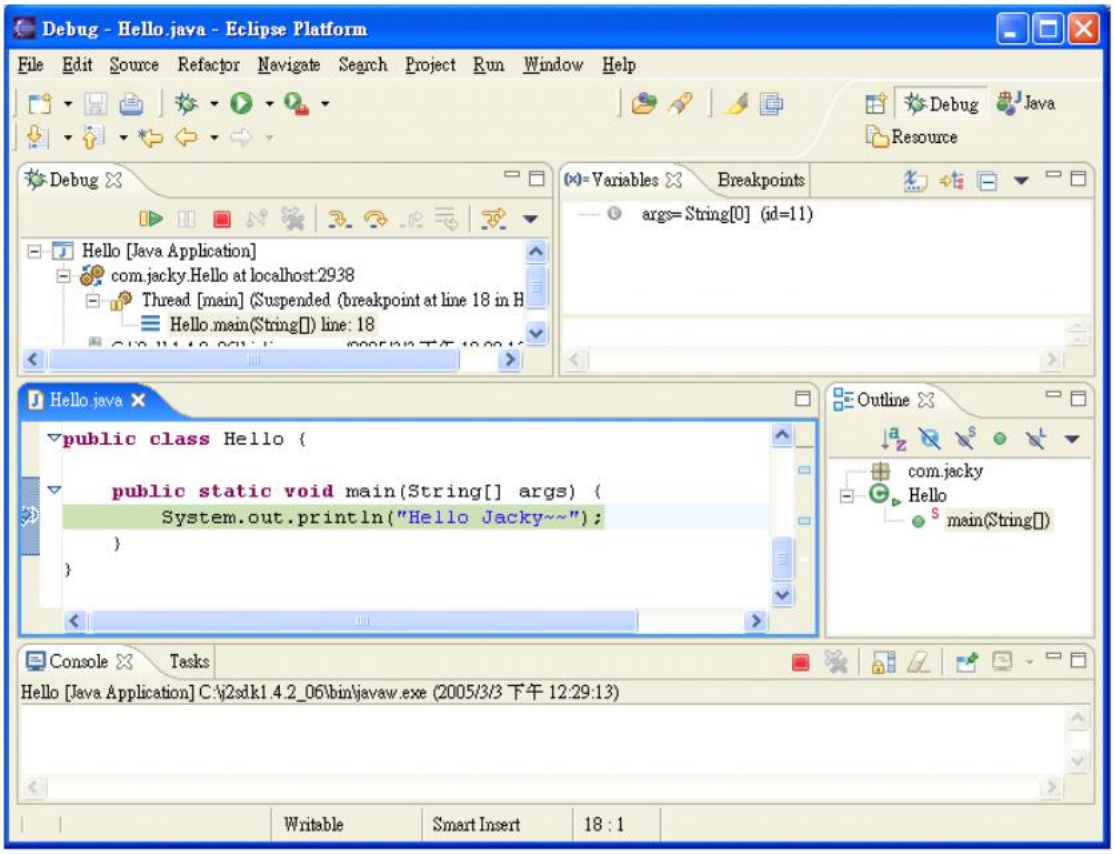


图 2. 11

2.7 重新排列视图和编辑器

2.7.1 放置游标

放置光标表示视图可以定置在工作台窗口的哪里。当重新排列视图时，可能会出现几种不同的放置光标。

图示	说明
⬆	定置上方：如果在显示定置上方光标时放开鼠标按钮，视图会放在光标所在视图的上面。
⬇	定置下方：如果在显示定置下方光标时放开鼠标按钮，视图会放在光标所在视图的下面。
➡	定置右侧：如果在显示定置右侧光标时放开鼠标按钮，视图会放在光标所在视图的右侧。
⬅	定置左侧：如果在显示定置左侧光标时放开鼠标按钮，视图会放在光标所在视图的左侧。
🗗	堆栈：如果在显示堆栈光标时放开鼠标按钮，视图会变成与光标下面的视图同一个窗格中的标签。
⊘	限制：如果在显示限制光标时放开鼠标按钮，视图不会定置在这个位置。比方说，视图不能定置在编辑区。

2.7.2 重新排列视图

可以变更「Navigator」视图在工作台窗口中的位置。

I. 按一下「Navigator」视图的标题列，并且拖曳视图以横跨工作台窗口。目前还不要放开鼠标按钮 。

II. 当仍在工作台窗口的顶端拖曳视图时，请注意，各种放置光标时会出现。这些放置游标（请参阅上一节）表示当放开鼠标按钮 时，视图会关联于光标所在的视图或编辑区而定置在哪里。请注意，这时会绘制用来强调显示的矩形，以提供视图将定置在哪里的其它回馈。

III. 将视图定置在工作台窗口中的任何位置，再检视这个动作的结果。

IV. 按一下并且拖曳视图的标题列，将视图重新定置在工作台窗口中的其它位置。请观察这个动作的结果。

V. 最后，将「Navigator」视图拖曳到「Outline」视图上面。这时会显示一个堆栈光标。如果放开鼠标按钮 ，「Navigator」就会和「Outline」视图一起堆放到附加卷标的笔记本中。

2.7.3 并列编辑器

工作台可以在编辑区中建立两组或更多组编辑器。也可以调整编辑区的大小，但不能将视图拖曳到编辑区。

I. 在「Navigator」视图中按两下可编辑的档案，以在编辑器区中开启至少两个编辑器。

II. 按一个编辑器的标示，将它拖曳到编辑器区域之外。不要放开鼠标按钮 。

III. 请注意，如果试图将编辑器放到任何视图的顶端，或放在工作台窗口之外，就会出现限制光标。

IV. 仍按住鼠标按钮 ，将编辑器拖曳到编辑器区，沿着编辑器区的四边移动光标，以及在编辑器区中央另一开启的编辑器上移动光标。请注意，沿着编辑器区域的边缘会出现有方向箭头的放置光标，编辑器区域中央会出现堆栈放置光标。

V. 将编辑器定置在有方向箭头的放置光标上，使两个编辑器都出现在编辑器区域中。

VI. 请注意，必要时，也可以调整各编辑器和整个编辑区的大小来容纳编辑器和视图。

VII. 请务必观察编辑器标签的颜色（下图中有两个群组，一个群组在另一群组的上面）

蓝色 - 表示编辑器目前在作用中。

默认值（在 Windows XP 中呈灰色）- 表示编辑器是前次作用中的编辑器。如果有作用中的视图，它就是作用中视图目前在使用的编辑器。当使用会密切搭配编辑器的「Outline」和「内容」这类视图时，这一点非常重要。

VIII. 拖曳编辑器，将它定置在编辑器区的其它位置，请注意定置各种放置光标时所产生的行为。请继续尝试定置编辑器和视图及调整其大小，直到工作台的安排符合要求为止。图 2.12 说明将一个编辑器拖放到另一编辑器之下的布置。

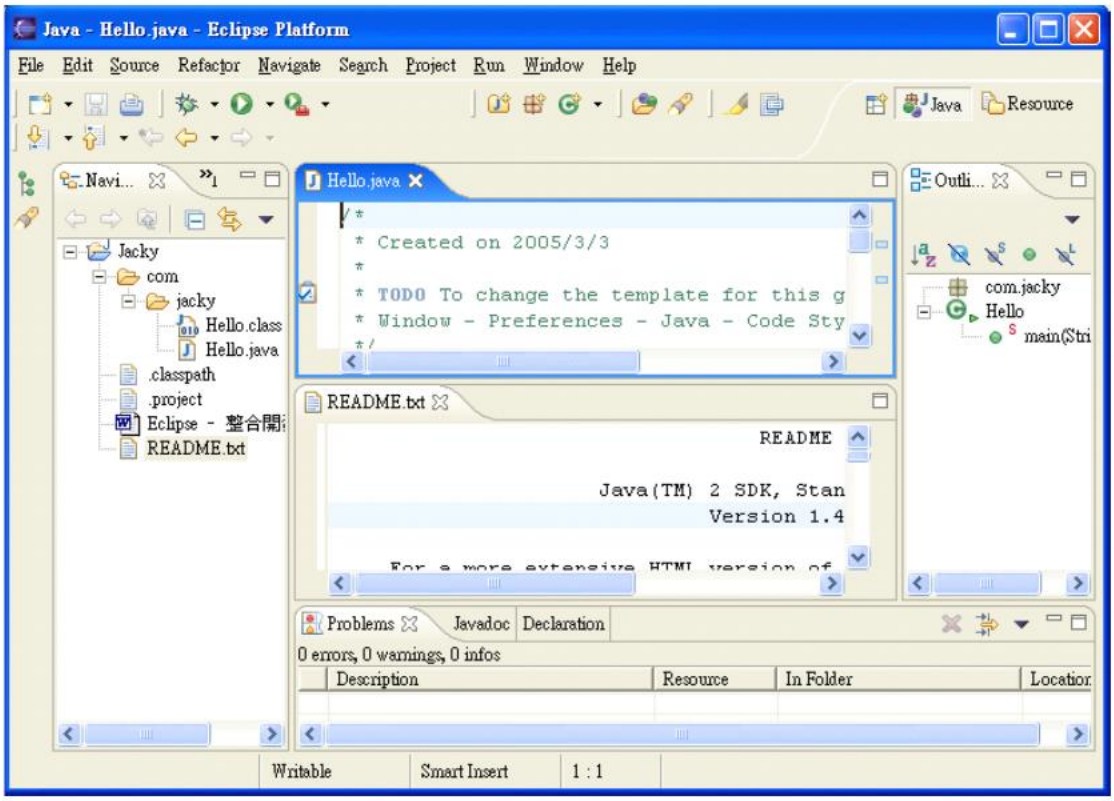


图 2.12

2.7.4 重新排列附加标签的视图

除了在工作台中拖放视图之外，也可以在附加卷标的笔记本内重新排列视图的次序。

I. 选择「Window」→「Reset Perspective」，将「Resource」视景重设回程序布置。

II. 按一下「Outline」标题列，然后在「Navigator」视图顶端加以拖曳。现在「Outline」将会堆栈在「Navigator」的顶端。

III. 按一下「Navigator」标签，将它拖曳到「Outline」标签的右侧。

IV. 光标到了「Outline」卷标右侧且变成堆栈光标之后，放开鼠标按钮。

请观察「Navigator」卷标，它现在已在「Outline」标签的右侧。

2.7.5 最大化

能够将视图或编辑器最大化，有时非常有用。将视图和编辑器两者最大化很容易。

- 如果要将视图最大化，请按两下它的卷标，或从卷标的蹦现菜单中选取「Maximize」。
- 如果要将编辑器最小化，请按两下编辑器卷标，或从卷标的蹦现菜单中选取「Minimize」。

将视图还原至程序大小的方法也类似（按两下或从菜单中选择「Restore」）。

2.8 菜单和工具列

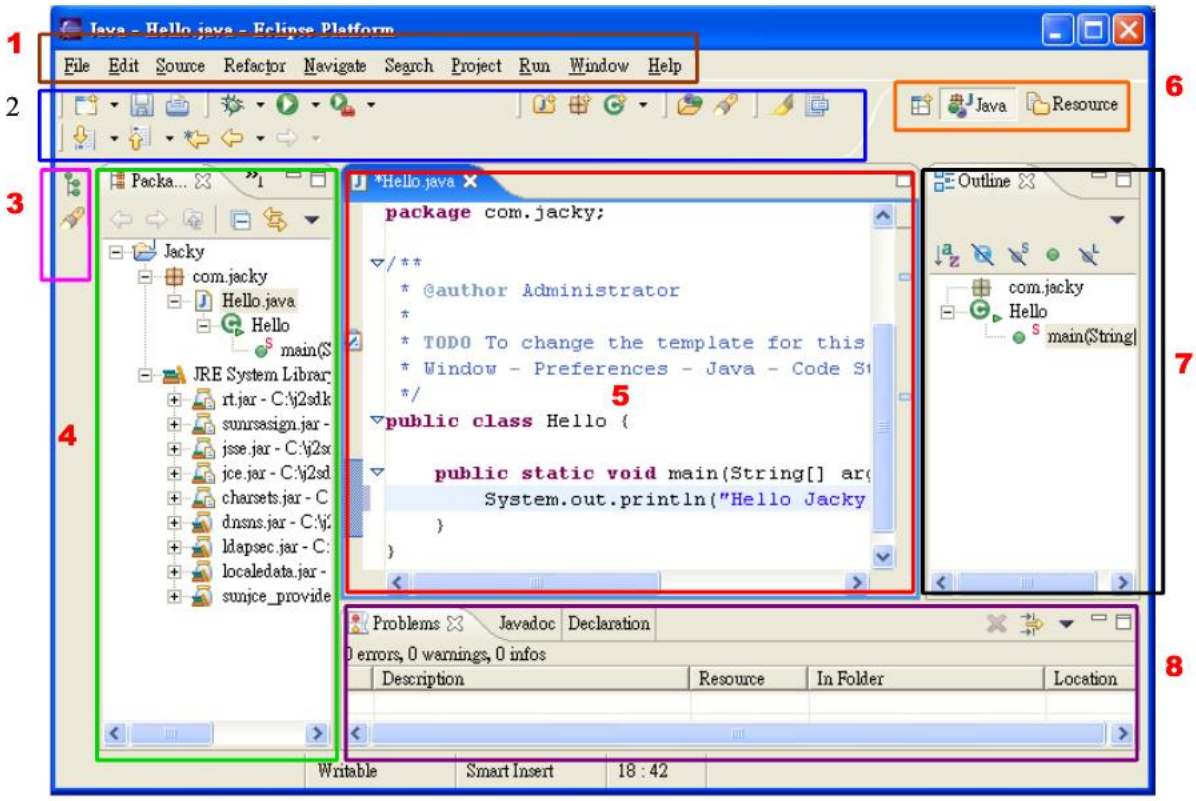


图 2.13

1.菜单(Menu Bar)

2.工具列(Tool Bar)

3.快速视图(Fast View)

4.Package Explorer 视图

5.Editor 视图

6.快捷方式工具列(Shortcut Toolbar)

7.Outline 视图

8.Tasks 视图和 Console 视图

2.8.1 菜单

「File」菜单

这个菜单可以建立、储存、关闭、打印、汇入及汇出工作台资源以及结束工作台本身。

名称	功能
New(新建)	建立 Java 元素或新资源。配置哪些元素会显示在「Window」→「Preferences」的子菜单中。在 Java 视景中，依预设，会提供项目、套件、类别、接口、来源数据夹、实时运算簿、档案和数据夹的建立动作。
Close(关闭)	关闭现行编辑器。如果编辑器中有资料尚未储存，则会显示一个储存要求对话框。

Close All(全部关闭)	关闭所有编辑器。如果编辑器中有资料尚未储存，则会显示一个储存要求对话框。
Save(储存)	储存现行编辑器的内容。如果编辑器中没有未储存的变更，则会停用。
Save As(另存新檔)	以新名称储存现行编辑器中的内容。
Save All(全部储存)	储存所有编辑器内容以及未储存的变更。如果没有编辑器中有未储存的变更，则会停用。
Revert(回复)	将现行编辑器的内容回复成已储存档案中的内容。如果编辑器中没有未储存的变更，则会停用。
Move(移动)	移动资源。如果是 Java 元素则会停用。如果要移动 Java 元素，请使用「Refactor」→「Move」（如此会更新档案的所有参照），或使用「Edit」→「Cut/Paste」（如此不会更新参照）。
Rename(重新命名)	将资源重新命名。如果是 Java 元素则会停用。如果要重新命名 Java 元素，请使用「Refactor」→「Rename」（如此会更新档案的所有参照）。
Refresh(重新整理)	以本端档案系统来重新整理所选元素的内容。如果不是从特定选项启动，这个指令会重新整理所有项目。
Print(打印)	打印现行编辑器的内容。会在编辑器成为焦点时启用。
Switch workspace(切换工作区)	这个指令可以切换至不同的工作区这会重新启动工作台
Open external file(开启外部档案)	这个指令可以在文字编辑器中开启不在工作区中的档案
Import(汇入)	开启汇入精灵对话框。JDT 不会提供任何汇入精灵。
Export(汇出)	开启汇出精灵对话框。JDT 会提供 JAR 档汇出精灵和 Javadoc 产生精灵。
Properties(内容)	开启所选元素的「内容」页面。依据 Java 项目开启 Java 建置路径页面，且可使用 Javadoc 位置页面。如果是 JAR 保存文件，请在这个配置 JAR 的程序文件附加与 Javadoc 位置。
Recent file list(最近使用的档案清单	「File 底端维护了一份最近在工作台中存取的档案的清单只要选取文件名，就可以从「File 开启这其中的任何档案。

Exit(结束)	结束 Eclipse

「Edit」菜单

这个菜单可协助操作编辑器区域中的资源

名称	功能
Undo(复原)	回复成编辑器中的前一次变更
Redo(重做)	回复已取消的变更
Cut(剪下)	将目前所选取的文字或元素复制到剪贴簿中，并移除元素。就元素而言，在贴到剪贴簿前不会移除。
Copy(复制)	将目前所选取的文字或元素复制到剪贴簿中。
Paste (贴上)	将目前的内容当成文字贴到编辑器中，或当成同层级或下层元素，贴到目前所选的元素中。
Delete(删除)	删除目前的文字或元素选项。
Select All(全选)	选取所有的编辑器内容。
Find / Replace(寻找/取代)	开启「寻找/取代」对话框。限编辑器。
Find Next(寻找下一个)	寻找目前所选文字下一个搜寻结果。限编辑器。
Find Previous(寻找上一个)	寻找目前所选文字上一个搜寻结果。限编辑器。
Incremental Find Next(增量寻找下一个)	启动增量寻找模式。在呼叫后，请按照状态列中的指示来输入搜寻文字。限编辑器。
Incremental Find Previous(增量寻找上一个)	启动增量寻找模式。在呼叫后，请按照状态列中的指示来输入搜寻文字。限编辑器。
Add Bookmark(新增书签)	为目前的文字选项或所选取的元素新增书签。
Add Task(新增作业)	为目前的文字选项或所选取的元素新增使用者定义的作业。
Expand Selection to(展开选项至)	含括元素：选取程序代码中的含括表示式、区块、方法。这个动作会注意 Java 语法。如果程序代码的语法有错，可能无法运作正常。（上移键） 下一个元素：选取现行与下一个元素。（右移键） 上一个元素：选取现行与上一个元素（左移键） 还原前次的选择：在呼叫展开选项至之后，还原先前的选项。（下移键）
Show Tooltip Description(显示工具提示说明)	以浮动说明方式显示出现在现行光标位置上的值。对话框可以卷动，因而不会缩短说明。
Content Assist(内容辅助)	在现行光标位置开启一个内容辅助对话框，以显示 Java 程序代码的辅助提议与范本。请参阅「模板」喜好设定页面，以取得可用的模板（「Window」→「Preferences」→「Java」→「Editor」→「Templates」），然后移至「编辑器」喜好设定页面（「Window」→「Preferences」→「Java」→「Editor」→「Code Assist」），来配置程序代码辅助的行为。

Quick Fix(快速修正)	如果光标位于有出现问题指示之处，则这个动作会在现行光标处开启一个内容辅助对话框，以提供可能的更正动作。
Parameter Hints(参数提示)	如果光标位于方法参照的参数规格处，这个动作会以浮动说明的方式显示参数类型信息。现行光标处的参数会以粗体字显示。
Encoding(编码)	切换目前所示文字内容的编码。

「Source」菜单

名称	功能
Toggle Comment(批注)	标注出内含现行选择项的所有字行。
Add Block Comment(批注区块)	标注出内含现行选择项的区块。
Remove Block Comment(解除批注区块)	取消标注内含现行选择项的区块。
Shift Right(向右移位)	增加目前所选字行的内缩层次。只有在选择项涵盖多行或一整行时才会启用。
Shift Left(向左移位)	减少目前所选字行的内缩层次。只有在选择项涵盖多行或一整行时才会启用。
Format(格式)	可使用程序代码格式制作器，来设定目前文字选择项的格式。格式设定选项是在「Code Formatter」喜好设定页面（「Window」→「Preferences」→「Java」→Code Formatter)）中配置
Format Element(格式成员)	格式化成员
Sort Members(排序成员)	「Window」→「Preferences」→「Java」→「Appearance」→「Members Sort Order」中指定的排序次序，来排序类型中的成员
Organize Imports(组织汇入)	组织目前开启或所选编译单元中的汇入宣告。会移除不必要的汇入宣告，且会按照「Organize Import」喜好设定页面（「Window」→「Preferences」→「Java」→「Organize Import」）中的指定，来排列必要的汇入宣告。「Organize Import」可执行于不完整的程序文件上，并且会在所参照的类型名称无法唯一对映至现行项目中的某个类型时提示。 也可以组织多个编译单元，其做法是对某个套件呼叫动作，或选取一组编译单元。
Add Import(新增汇入)	为目前所选的类型参照建立一项汇入宣告。如果类型参照完整，则会移除资格。如果所参照的类型名称无法唯一对映至现行项目中的某个类型，将会提示指定正确的类型。「Add Import」会试着遵循「Organize Import」喜好设定页面中指定的汇入顺序。
Override/Implement Methods(置换/实作方法)	会开启「Override Method」对话框，可以置换或实作现行类型中的方法。适用于类型或类型中的某个文字选择项。
Generate Getter and Setter(产生 Getter 和 Setter)	开启「Generate Getter and Setter」对话框，可以为现行类型中的字段，建立 Getter 和 Setter。适用于字段与类型或类型中的某个文字选择项。

Generate Delegate Methods(产生委派方法)	开启「Generate Delegate Methods」对话框，可以为现行类型中的字段建立方法委派。可用在字段。
Add Constructor from Superclass(新增 Super 类别中的建构子)	为目前所选的类型新增 Super 类别中所定义的建构子。适用于类型或类型中的某个文字选择项。
Surround with try/catch(以 try/catch 包覆)	针对所选的陈述式，评估所有必须捕捉到的异常状况。这些表示式会包覆 try catch 区块。可以使用编辑菜单中的展开选项至，以取得有效的选项范围。
Externalize Strings(将字符串提出)	开启「Externalize Strings」精灵。这个精灵可以藉由会存取内容档的陈述式，来更换程序代码中的所有字符串。
Find Strings to Externalize(寻找要提出的字符串)	会出现一个对话框，其中显示未提出字符串数目的摘要。适用于项目、来源资料夹与套件。
Convert Line Delimiters To(将行定界字符转换成)	在目前开启的编辑器中，变更所有行定界字符，而采用下列操作系统中所用的行定界字符： CRLF(Windows) LF（Unix、MacOSX） CR（传统 MacOS） Java 编辑器容许混合使用行定界字符。不过，其它某些工具会要求使用和 OS 一致的行定界字符，或者要求至少行定界字符要一致。

「Refactor」菜单

重构指令也可以在一些视图的快速菜单与 Java 编辑器中找到。

名称	功能
Undo(复原)	「Undo」前次的重构作业。重构复原缓冲区，共在执行重构后程序文件未变更的状况下有效。
Redo(重做)	重做前次复原的重构作业。重构复原/重做缓冲区的有效期，仅限于执行重构后到没有其它程序文件变更的这段时间。
Rename(重新命名)	启动「Rename Refactoring」对话框：重新命名所选的元素，并且（如果有启用的话）更正元素的（以及其它档案中的）所有参照。适用于方法、字段、区域变量、方法参数、类型、编译单元、套件、来源数据夹、项目，并且适用于可解析成这些元素类型之一的文字选项。
Move(移动)	启动「Move」重构对话框：移动所选的元素，并（如果有启用的话）更正元素的（以及其它档案中的）所有参照。可套用至一或多个 Static 方法、Static 字段、类型、编译单元、套件、来源数据夹与项目，并且套用于可解析成这些元素类型之一的文字选择项。
Change Method Signature(变更方法签章)	启动「Change Method Signature」重构对话框。变更参数名称、参数类型、参数顺序，并更新对应方法的所有参照。此外，可以移除或新增参数，也可以变更方法

	传回类型及其可见性。这个重构作业可套用至方法或套用在解析成方法的文字选项。
Convert Anonymous Class to Nested(将匿名类别转换成巢状)	启动「Convert Anonymous Class to Nested」重构对话框。协助将匿名内部类别转换成成员类别。这个重构作业可套用至匿名内部类别。
Convert Nested Type to Top Level(将巢状类型转换成最上层)	启动「Convert Nested Type to Top Level」重构对话框。为所选成员类型建立新的 Java 编译单元，同时依需要更新所有参照。对于非 static 成员类型，将新增一个字段，以容许存取先前含括的实例。这个重构作业可套用至成员类型或解析成成员类型的文字。
Push Down(下推)	启动「Push Down」重构对话框。将类别中的一组方法和字段移至它的子类别。这个重构作业可套用至一个或多个以相同类型宣告的方法和字段，或套用在字段或方法内的文字选项。
Pull Up(上拉)	启动「Pull Up」重构精灵。将字段或方法移至其宣告类别的 Super 类别，或（如果是方法）将方法宣告成 Super 类别中的 abstract。这个重构作业可套用至一个或多个以相同类型宣告的方法、字段和成员类型，或套用在字段、方法或成员类型内的文字选项。
Extract Interface(撷取界面)	启动「Extract Interface」重构对话框。以一组方法建立新的接口，并使所选类别实作接口，同时选择性将类别参照变更为新接口（在可能的情况下）。这个重构作业可套用至类型。
Use Supertype Where Possible(适当时使用 Super 类型)	启动「Use Supertype Where Possible」重构对话框。在识别所有可能发生这个取代的位置后，将出现的类型换成其 Super 类型之一。这个重构作业可用在类型之上。
Inline(列入)	启动「Inline」重构对话框。列入区域变量、方法或常数。这个重构作业可用在方法、static final 字段，以及解析为方法、static final 字段或区域变量的文字选项。
Extract Method(撷取方法)	启动「Extract Method」重构对话框。会建立一个内含目前所选之陈述式或表示式的新方法，并将选择项换成新方法的参照。可以使用编辑菜单中的展开选项至，以取得有效的选项范围。 这项特性非常适合用来清除冗长、杂乱和太复杂的方法。
Extract Local Variable(撷取区域变量)	启动「Extract Local Variable」重构对话框。会建立一个新变量，以指定给目前所选的表示式，并将选择项换成新变量的参照。这个重构作业可用在解析为区域变量的文字选项。可以使用编辑菜单中的展开选项至，以取得有效的选项范围。
Extract Constant(撷取常数)	启动「Extract Constant」重构对话框。从所选表示式中建立 static final 字段并替代字段参照，以及选择性地重新写入其它出现相同表示式的位置。这个重构作业可用在 static final 字段，以及解析为 static final 字段的文字选项。
Convert Local Variable to Field(将区域变量转换成	启动「Convert Local Variable to Field」重构对话框。将区域变量转换成字段。如果在建立时已起始设定变量，则作业会将起始设定移至新字段的宣告，或移至类别

字段)	的建构子。这个重构作业可用在解析为区域变量的文字选项。
Encapsulate Field(封装字段)	启动「Encapsulate Field」重构对话框。会将字段的所有参照换成 getting 与 setting 方法。适用于所选的字段或可解析成字段的文字选择项。

「Navigate」菜单

这个菜单可以寻找及导览工作台中显示的资源及其它成品。

名称	功能
Go Into(进入)	将视图输入设定在目前所选的元素上。「套件浏览器」视图可支持这项。
Go To(移至)	上一页：将视图输入设定在历程中的上一个输入上：必须有历程，才能用到这项（已使用「Go Into」） 下一页：将视图输入设定在历程中的下一个输入上：必须有历程，才能用到这项（已使用「Go Into」、「 Go Into」→「Back」） 往上移一层：将现行视图的输入设定在其输入的母元素上。 参照测试：浏览以找出所有参照目前选取之类型的 JUnit 测试 类型：浏览以找出类型，并在现行视图中显示它。「Package Explorer」视图支援这项。 套件：浏览以找出套件，并在现行视图中显示它。「Package Explorer」视图支援这项。 资源：浏览以找出资源，并在现行视图中显示它。
Open(开启)	试着解析现行程序代码选项上所参照的元素，并开启宣告该参照的档案。
Open Type Hierarchy(开启类型阶层)	试着解析现行程序代码选项上所参照的元素，并在「Type Hierarchy」视图中开启该元素。针对元素呼叫，并开启元素的类型阶层。显示 Java 元素的 Java 编辑器与视图中可支持这项。
Open Call Hierarchy(开启呼叫阶层)	试着开启呼叫现行程序代码选项上所参照的元素，并在「Call Hierarchy」视图中开启该元素。
Open Super Implementation(开启 super 实作)	开启一个编辑器，以显示目前所选方法或现行光标位置旁之方法的 super 实作。如果未选取方法，或者方法没有 super 实作，则不会开启编辑器。
Open External Javadoc(开启外部 Javadoc)	开启目前所选元素或文字选项的 Javadoc 文件。JAR 或项目的 Javadoc 位置是在项目或 JAR 的「Javadoc Location」内容页面中指定。请注意，这个外部 Javadoc 文件可能未以现行程序代码中指定的 Javadoc 加以更新。可以使用 Javadoc 汇出精灵，在 Java 项目中为程序文件建立 Javadoc 文件。
Open Type(开启类型)	显示「Open Type」选择对话框，以便在编辑器中开启一个类型。「开启类型」选择对话框中显示工作区中的所有现有类型。
Open Type In Hierarchy(在「阶层」中开启类型)	显示「Open Type」选择对话框，以便在编辑器与「Type Hierarchy」视图中开启一个类型。「Open Type」选择对话框中显示工作区中的所有现有类型。

Show in → Package Explorer(显示在→套件浏览器)	在「Package Explorer」视图中显示目前所选的元素（或现行光标位置旁的元素）。
Quick Outline(显示概要)	为目前选取的类型开启小型概要器。
Quick Type Hierarchy(显示类型阶层)	为目前选取的类型开启小型类型阶层器。
Next Annotation (移至下一个问题)	选取下一个问题。Java 编辑器中支持这项。
Previous Annotation (移至上一个问题)	选取上一个问题。Java 编辑器中支持这项。
Go to Last Edit Location(移至前次编辑位置)	显示前次发生编辑的位置。
Go to Line(移至指定行号)	开启对话框，以输入指示编辑器应移至的行号。限编辑器。
Back(向后)	这个指令会导览至之前在编辑器中检视的前一个资源。这个指令和 Web 浏览器的上一页按钮相同。
Forward(向前	这个指令会导览并复原之前的上一页指令所造成的效果。这个指令和 Web 浏览器的下一页按钮相同。

「Search」菜单

名称	功能
Search...(搜寻...)	开启搜寻对话框
File...(档案...)	针对「档案搜寻」页面开启搜寻对话框
Java...(Java...)	针对「Java 搜寻」页面开启搜寻对话框
References(参照)	寻找所选 Java 元素的所有参照
Declarations(宣告)	寻找所选 Java 元素的所有宣告
Implementors(实作者)	寻找所选接口的所有实作者。
Read Access(读取权)	寻找所选字段的所有读取权
Write Access(写入权)	寻找所选字段的所有写入权
Referring Tests...()	寻找所选 Java 元素的所有测试参照
Occurrences in File(档案中的搜寻结果)	寻找所选 Java 元素在其档案中的所有出现项目
Exception Occurrences(抛出例外中的搜寻结果)	寻找所选 Java 元素在其抛出例外中的所有出现项目

Search Scopes Submenu(搜寻范围子菜单)：

范围	可用性	说明
Workspace(工作区)	所有元素	在整个工作区中搜寻
Project(专案)	所有元素	在含括所选元素的项目中进行搜寻
Hierarchy(阶层)	类型和成员	在类型的阶层中搜寻
Workings Set(工作集)	所有元素	在工作集中搜寻

工作集对话框可以储存并命名范围。「搜寻范围」子菜单中亦会显示工作集的现有实例。

可在下列视图中透过所选资源与元素的快速菜单，来执行 Java 搜寻：

- 「Package Explorer」
- 「Outline」视图
- 「Search Result」视图
- 「Hierarchy」视图
- 「Browsing」视图

Java 编辑器中亦提供「Search」快速菜单。目前所选文字必须可解析成 Java 元素，才能执行搜寻。

所选 Java 元素的类型会定义所能使用的「Search」快速菜单。Java 编辑器不会根据选项而限制可用的 Java 搜寻项清单。

「Project」菜单

「项目」菜单可以对工作台中的项目执行动作（建置或编译）。

名称	功能
Open Project(开启专案)	显示对话框，可以选取开启已关闭的项目
Close Project(关闭专案)	关闭目前所选取的项目
Build All(全部建置)	这个指令会对工作台中的所有项目执行增量(incremental)建置。也就是说，它会建置（编译）自从前次增量建置后，工作台中受到任何资源变更所影响的所有资源。自动建置关闭时，才可使用这个指令。
Build Project(建置专案)	这个指令会对目前选取的项目执行增量(incremental)建置。也就是说，它会建置（编译）自从前次建置后，受到任何资源变更所影响的项目中的所有资源。自动建置关闭时，才可使用这个指令。
Build Workings Set(重新建置工作集)	这个菜单可以在工作集上执行增量(incremental)建置。也就是说，它会建置（编译）前次建置之后，受到任何资源变更所影响之工作集中的所有资源。自动建置关闭时，才可使用这个指令。
Clean(清除)	这个指令会舍弃先前的所有建置结果。如果自动建置是开启的，这会呼叫完整的建置。
Build Automatically(自动建置)	自动建置工作区中的所有项目。这个指令可以切换自动建置喜好设定。
Generate Javadoc...(产生 Javadoc...)	对目前选取的项目开启「Generate Javadoc」精灵。
Properties(内容)	对目前选取的项目开启内容页面。

「Run」菜单

名称	功能
Toggle Line Breakpoint(切换行岔断点)	这个指令可以在目前于作用中 Java 编辑器中所选之行处，新增或移除 Java 行岔断点。
Toggle Method Breakpoint(切换方法岔断点)	这个指令可以针对目前的二进制方法，新增或移除方法岔断点。可在 Java 类别档编辑器的来源中选取二进制方法，或在其它任何视图中选取（像是「Outline」视图）。
Toggle Watchpoint(切换监视点)	这个指令可以针对目前的 Java 字段，新增或移除字段监视点。可在 Java 编辑器的来源中选取字段，或在其它任何视图中选取（像是「Outline」视图）。
Skip All Breakpoints(忽略所有的岔断点)	这个指令可以忽略所有的岔断点
Add Java Exception Breakpoint(新增 Java 异常状况岔断点)	这个指令可以建立一个异常状况岔断点。可藉由指定异常状况岔断点，而在掷出异常状况时，暂停执行绪或 VM 的执行。可设为在未捕捉到或捕捉到（或两者）异常状况时暂停执行。
Add Class Load Breakpoint	这个指令可让以建立一个 Class Load 岔断点。
Run Last Launched(执行前一次的启动作业)	这个指令可以在执行模式下迅速重复最近一次的启动作业（如果有支持该模式的话）。
Debug Last Launched(除错前一次的启动作业)	这个指令可以在除错模式下迅速重复最近一次的启动作业（如果有支持该模式的话）。
Run History(执行历程)	呈现在执行模式下启动的启动配置之最近历程的子菜单
Run As(执行为)	呈现所登录之执行启动快捷方式的子菜单。启动快捷方式可支持工作台或作用中编辑器选项的感应式启动。
Run...(执行...)	这个指令会了解启动配置对话框，以管理执行模式下的启动配置。
Debug History(除错历程)	呈现在除错模式下启动的启动配置之最近历程的子菜单
Debug As(除错为)	呈现所登录之除错启动快捷方式的子菜单。启动快捷方式可支持工作台或作用中编辑器选项的感应式启动。
Debug...(除错...)	这个指令会了解启动配置对话框，以管理除错模式下的启动配置。
Inspect(视察)	当执行绪暂停时，这个指令会使用「表示式」视图，显示在该执行绪之堆栈框或变量的环境定义下，视察所选表示式或变量的结果。
Display(显示)	当执行绪暂停时，这个指令会使用「Display」视图，显示在该执行绪之堆栈框或变量的环境定义下，评估所选表示式的结果。如果目前作用中的部分是「Java Snippet Editor(Java 片段编辑器)」，则会在其中显示结果。
Execute(执行)	执行
Step into Selection	这些指令可以逐步执行所要除错的程序代码。

名称	功能
Externakl Tools(外部工具)	外部工具

「Windows」菜单

这个菜单可以显示、隐藏，以及另行在工作台中操作各种视图、视景和动作。

名称	功能
New Window(开新窗口)	这个指令会开启一个新的工作台窗口，其中含有与现行视景相同的视景。
Open Perspective(开启视景)	这个指令会在此工作台窗口中开启新视景。可以在「Window」→「Preferences」→「Workbench」→「Perspectives」页面中变更这个喜好设定。在工作台窗口内开启的所有视景都会显示在快捷方式列上。
Show View(显示视图)	这个指令会在现行视景中显示选取的视图。可以在「Window」→「Preferences」→「Workbench」→「Perspectives」页面中配置开启视图的方式。可能会想开启的视图会最先列示；这份清单视现行视景而定。从其它... 子菜单中，可以开启任何视图。视图会依照「Show View」对话框中的各个种类来排序。
Customize Perspective(自订视景)	每个视景包含一组预先定义的动作，可以从菜单列和工作台工具列存取这些动作。
Save Perspective As(另存新视景)	这个指令可以储存现行视景，以及建立自己的自订视景。储存视景之后，可以使用「Window」→「Show View」→「Other...」菜单项目来开启更多这类型的视景。
Reset Perspective(重设视景)	这个指令会将现行视景的布置变更为其原始的配置。
Close Perspective(关闭视景)	这个指令会关闭作用中的视景。
Close All Perspectives(关闭所有视景)	这个指令会关闭工作台窗口中的所有已开启视景。
Navigation(导览)	这个子菜单包含用于在工作台窗口中的视图、视景及编辑器之间导览的按键。 显示系统菜单：显示用来重新调整大小、关闭或固定现行视图或编辑器的菜单。 显示视图菜单：显示可在作用中视图的工具列中存取的下拉菜单。 将作用中的视图或编辑器最大化：使作用中的部分占用整个画面，如果已占用整个画面，就使它返回原始状态。 启动编辑器：使现行编辑器作用中。 下一个编辑器：启动最近使用的编辑器清单中的下一个开启的编辑器。 上一个编辑器：启动最近使用的编辑器清单中的上一个开启的编辑器。 切换至编辑器：显示一个对话框，用来切换到已开启的编辑器。显示一

名称	功能
	个对话框，用来切换到已开启的编辑器。 下一个视图：启动最近使用的视图清单中的下一个开启的视图。 上一个视图：启动最近使用的编辑器清单中的上一个开启的编辑器。 下一个视景：启动最近使用的视景清单中的下一个开启的视景。 上一个视景：启动最近使用的视景清单中的上一个开启的视景。
Preferences(喜好设定)	这个指令可以指出在使用工作台时的喜好设定。其中有各式各样的喜好设定可用来配置工作台及其视图的外观，以及用来自订在工作台中安装的所有工具的行为。

「Help」菜单

这个指令提供有关使用工作台的说明。

名称	功能
Welcome(欢迎使用)	这个指令会开启欢迎使用内容。
Help Contents(说明内容)	这个指令显示说明视图。说明视图含有说明书籍、主题，以及与工作台和已安装特性的相关信息。
Tips and Tricks(要诀和技巧)	这个指令会开启可能尚未探索之有兴趣的生产力特性的清单。
Cheat Sheets(提要)	这个指令会开启选取提要的对话框。
Software Updates(软件更新)	这个指令群组可以更新产品以及下载及安装新特性。
About Eclipse Platform(关于 Eclipse 平台)	这个指令显示产品、已安装特性及可用外挂程序的相关信息。

2.8.2 图标和按钮

「Navigator」视图图示

「Navigator」视图中可能会出现下列图示：

图示	说明
	专案（开启）
	数据夹（开启）
	专案（已关闭）
	一般檔

编辑区标记列

标记列（编辑区左侧）中可能会出现下列标记：

图示	说明
	书签
	岔断点
	作业标记
	搜寻结果
	错误标记
	警告标记
	信息标记

「Tasks」视图

「Tasks」视图可能会出现下列标记：

图示	说明
	信息作业
	高优先级作业
	低优先级作业
	已完成作业
	警告作业
	错误作业

工具列按钮

下列按钮可能会出现在工作台工具列、视图的工具列以及快捷方式列中：

图示	说明	图示	说明
	开启新视景		储存作用中的编辑器内容
	储存所有编辑器的内容		以新的名称或位置储存编辑器内容
	开启搜寻对话框		打印编辑器内容
	开启资源建立精灵		开启档案建立精灵
	开启数据夹建立精灵		开启项目建立精灵
	开启「汇入」精灵		开启「汇出」精灵

图示	说明	图示	说明
	执行增量建置		执行程序
	除错程序		执行外部工具或 Ant
	剪下选择至剪贴簿		复制选择至剪贴簿
	从剪贴簿贴上选择		复原最近的编辑
	重做最近的复原编辑		导览至清单中的下一个项目
	导览至清单中的上一个项目		向前导览
	向后导览		导览上一层
	新增书签或作业		开启视图的下拉菜单
	关闭视图或编辑器		固定编辑器以防止自动重复使用
	过滤作业或内容		移至编辑器中的作业、问题或书签
	还原预设内容		以树状结构显示项目
	重新整理视图内容		按字母顺序排序清单
	取消执行过久的作业		删除选取的项目或内容

外部工具和 Ant 图示

物件

图示	说明
	Ant 建置檔
	包含错误的 Ant 目标
	无效的专案建置器
	预设目标
	公用 Ant 目标（含说明的目标）
	Ant 内部目标（不含说明的目标）
	Jar 檔
	Ant 内容
	Ant 作业
	Ant 类型
	Ant 汇入作业
	Ant macrodef 作业

启动配置

图示	说明

	启动外部工具
	Ant 启动配置
	程序启动配置
	「主要」标签
	「重新整理」标签
	「建置」标签
	「目标」卷标
	「内容」栏标
	「类别路径」卷标

Ant 视图

图示	说明
	Ant 视图
	新增建置檔
	透过搜寻来新增建置档
	执行选取的建置文件或选取的目标文件
	移除选取的建置档
	移除所有的建置檔
	内容

除错视图

指令	名称	说明
	回复	这个指令会让已暂停的执行绪恢复执行。
	暂停	这个指令会暂停执行目标中所选取的执行绪，可以浏览或修改程序代码、视察资料、逐步执行等。
	终止	这个指令会终止所选取的除错目标。
(仅快速菜单)	终止并移除	这个指令会终止所选取的除错目标，并将之从视图中移除。
(仅快速菜单)	全部终止	这个指令会终止视图中所有作用中的启动作业。
	切断联机	这个指令会切断除错器和所选取的除错目标间的联机（如果是远程除错的话）。
	移除全部终止的启动	这个指令会将所有已终止的除错目标从视图显示中清除。

	使用逐行过滤器	这个指令会切换逐行过滤器（开/关）。当它开启时，所有的逐行功能都会套用逐行过滤器。
	进入副程序	这个指令会进入强调显示的陈述式。
	跳过副程序	这个指令跳过强调显示的陈述式。在下一行会以相同方法继续执行或（如果位于方法结尾）使用呼叫现行方法的方法继续执行。 游标会跳到方法的宣告处，并选取这一行。
	执行到 Return	这个指令会跳出现行方法的副程序。这个选项会在结束现行方法后停止执行。
	显示完整名称	这个选项可以切换成显示或隐藏完整名称。
(仅快速菜单)	复制堆栈	这个指令会将已暂停执行绪中所选取的堆栈以及执行中之执行绪的状态，复制到剪贴簿中。
	放到页框	这个指令可以放回与重新输入指定的堆栈框。这项特性类似「回头执行」再整个重新启动程序。 如果要放回堆栈框，再重新输入指定的堆栈框，请选取要「放置」的指定堆栈框，再选取放入堆栈框。 请注意下列有关这项特性的警告： 不能在堆栈中放入原生方法。 全体数据不受影响，仍维持其现行值。举例来说，不会清除内含元素的 Static 向量。 附注：只有在基础 VM 支持这项特性时，才会启用这个指令。
(仅快速菜单)	重新启动	这个指令会重新启动所选除错目标。
(仅快速菜单)	内容	这个指令会显示所选取的启动作业的内容。此外，也可以检视所选程序的完整指令行。

2.9 视景

2.9.1 新视景

有几种方法可在这个工作台窗口内开启新视景：

- 利用快捷方式列中的「Open Perspective」按钮 。
- 从「Window」→「Open Perspective」菜单中选取一个视景。

如果要利用快捷方式列按钮来开启一个视景，请执行下列动作：

I. 按一下「Open Perspective」按钮 。

II. 这时会出现一个菜单，显示和「Window」→「Open Perspective」菜单相同的选项。请从菜单中选择

「Other...」。

图 2.14

III. 在「Select Perspective」对话框中，选择 Debug，然后按一下 OK。

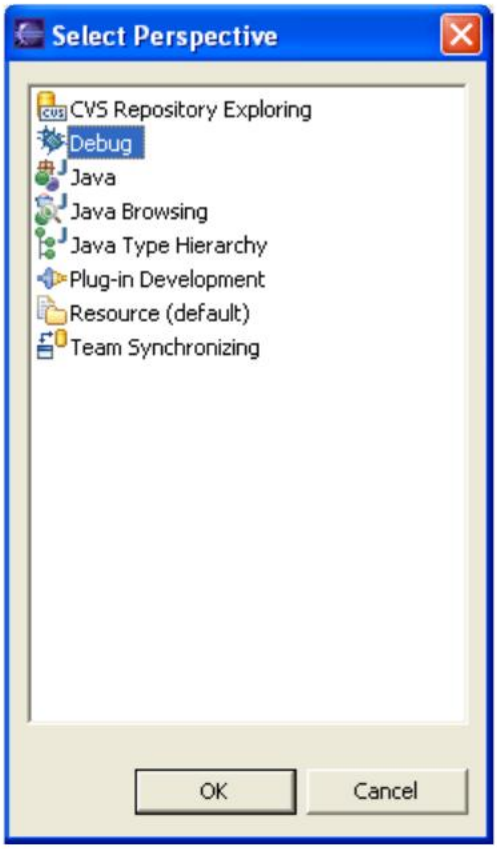


图 2.15

这时会显示「Debug」视景。

IV. 另外还有些有趣的事情值得注意。

现在，窗口标题会指出「Debug」视景正在使用中。

快捷方式列包含几个视景：原始「Resource」视景、新的「Debug」视景，以及少数几个其它视景。「Debug」视景按钮是已经下按的，表示它是现行视景。

- 如果要显示视景的完整名称，请用鼠标右键按一下视景列，再勾选 Show Text。

图 2.16

V. 在快捷方式列中，按一下「Resource」视景按钮。这时「Resource」视景又会成为现行视景。请注意，每个视景所拥有的一组视图各不相同。

2.9.2 新窗口

除了在现行工作台窗口中开启视景之外，也可以在另一个窗口中开启新的视景。

依预设，新视景会开启在现行窗口中。可以利用「Window」→「Preferences」→「Workbench」→「Perspectives」来配置这个预设行为。

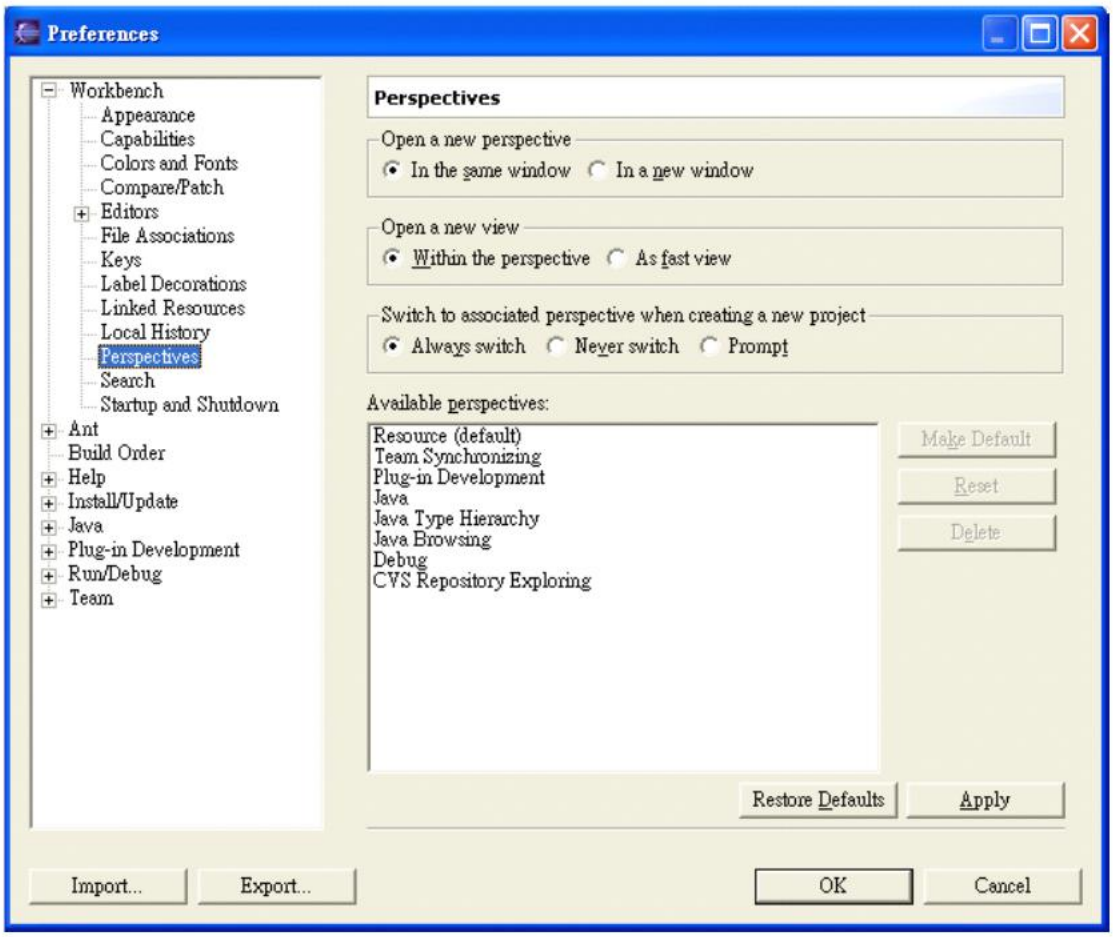


图 2.17

2.9.3 储存视景

可以利用工作台来储存自己喜好的布置，供未来使用。

I. 在快捷方式列中，按一下「Resource」视景。现在「Resource」视景是在作用中。

II. 拖曳「Outline」视图，将它和「Navigator」视图堆放在一起。

III. 选择「Window」→「Save Perspective As...」

IV. 「Save Perspective As...」对话框可用来重新定义现有的视景，或建立新视景。

按一下 OK 来更新「Resource」视景，并在后续的确认对话框中按一下 OK。如果重设视景或开启新视景，就会使用新的视景布置。

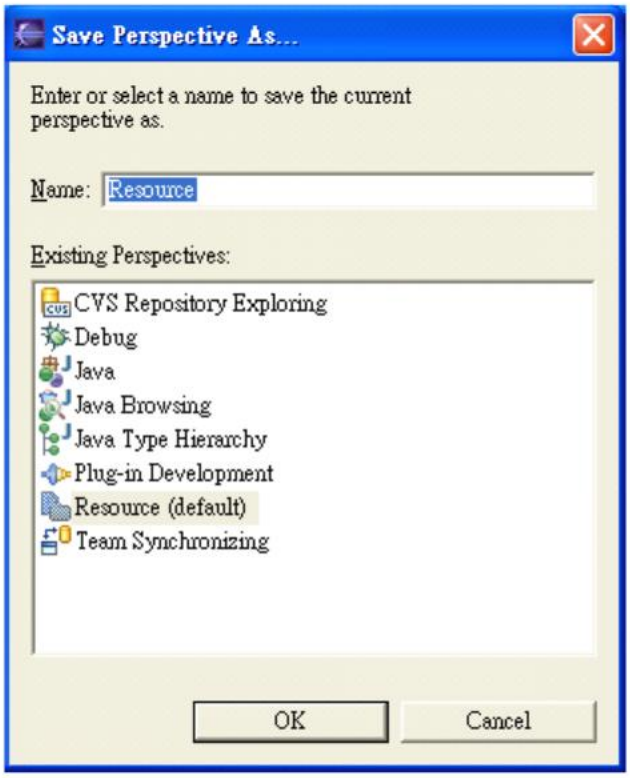


图 2.18

V. 在「Resource」视景中移动「Outline」视图，现在，它和「作业」视图堆放在一起。

VI. 选择「Window」→「Reset Perspective」。请注意，「Outline」视图会和「Navigator」堆放在一起。原先，当第一次启动工作台时，它是在导览器下面，但由于储存视景时将「储存库」和「Outline」堆栈起来，因此，它现在就以此为起始布置。

VII. 选择「Window」→「New Window」来开启第二个窗口，以显示资源视景。请观察它，它在使用新储存的布置。

VIII. 关闭第二个窗口。

在变更过「Resource」视景之后，还有一个方法可用来回复原始布置。如果要将「Resource」视景重设回程序布置：

I. 选择「Window」→「Preferences」。

II. 展开 Workbench，然后选取 Perspective。

III. 选取 Resource 视景， 然后按一下 Reset 按钮，再按一下 OK。

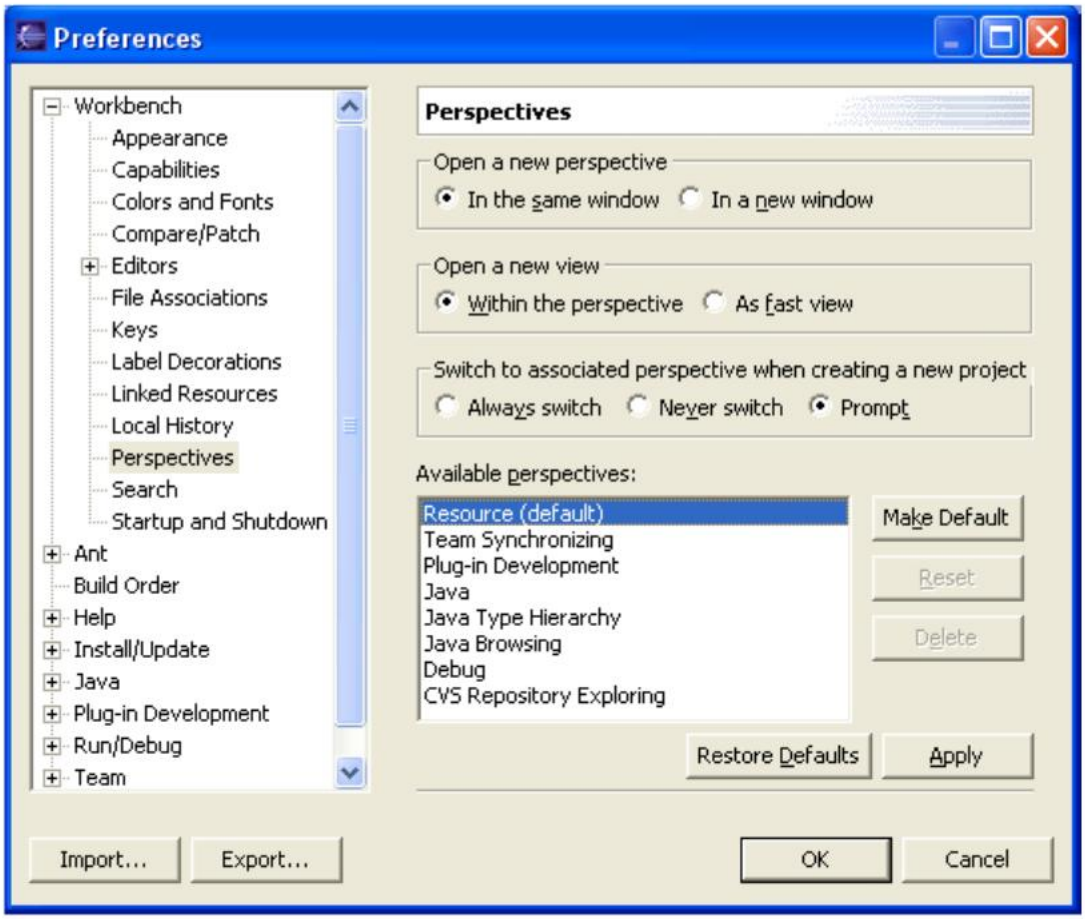


图 2.19

IV. 现在，视景储存状态的任何变更尚未完成。如果要更新正在处理的「Resource」视景现行复本，请从工作台的菜单列中选取「Window」→「Reset Perspective」。

2.9.4 配置视景

除了配置视景的布置之外，也可以控制视景的若干其它主要方面。其中包括：

- 「New Window」。
- 「Window」→「Open Perspective」。
- 「Window」→「Show View」。
- 工具列所显示的各组动作。

请尝试自订这些项目之一。

I. 在快捷方式列中，按一下「Resource」视景。

II. 选取「Window」→「Customize Perspective...」

III. 选取 Commands 标签。

IV. 勾选 Launch，然后按一下 OK。

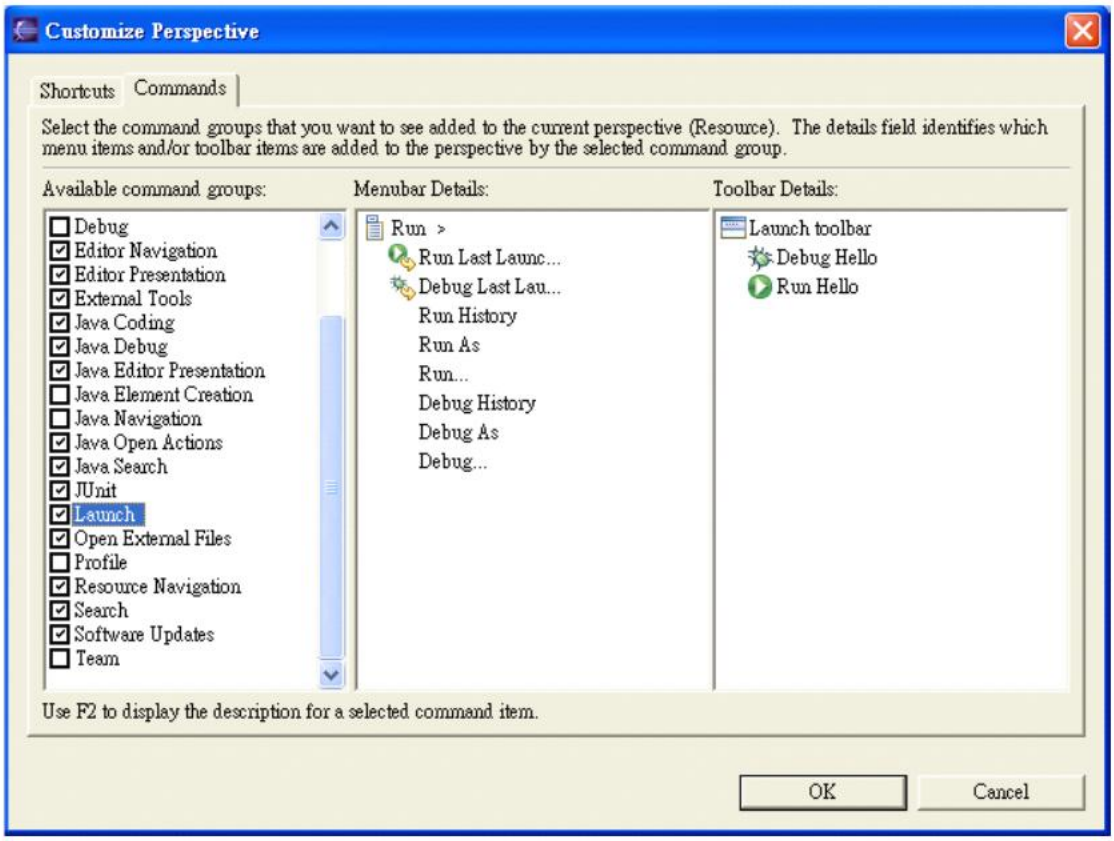


图 2.20

V. 观察工具列，现在它含有除错 / 执行启动的按钮。

VI. 尝试过「Customize Perspective...」对话框中的其它选项之后，请选择「Window」→「Reset Perspective」，让视景返回原始状态。

2.10 作业和标记

标记有许多类型，包括书签、作业标记、除错岔断点和问题。这一节的重点是作业和「Tasks」视图。

「Tasks」视图会显示工作台中的所有作业。这个视图会显示与特定档案、特定档案中的特定行的相关作业，以及没有与任何特定档案相关的一般作业。

2.10.1 不相关的作业

未关联的作业不会关联于任何特定资源。如果要建立未关联的作业：

I. 在「Tasks」视图中，按一下「Add Task」按钮 。这时会出现新作业对话框。

II. 输入作业的简要说明，再按 Enter 键。如果在输入说明时要取消对话框，请按 Esc。这时「Tasks」视图中会出现新的作业。

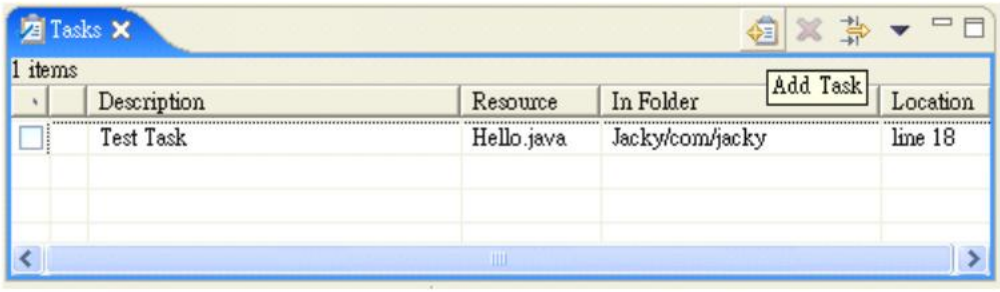


图 2.21

2.10.2 相关的作业

相关作业会关联于资源中的某特定位置。如果要将作业关联于 Hello.java，请执行下列动作：

I. 在「Navigator」视图中，按两下开启 Java 程序 (Hello.java)。

II. 在文字编辑器任何一行左侧的编辑器区域中，从标记列来存取快速菜单。标记列主要文字区域左侧的垂直列。

III. 从标记列的快速菜单中，选取 Add Task。

标记列会显示包括书签、（相关作业的）作业标记和/或除错岔断点在内的任何标记。可以直接从档案中特定行左侧的标记列中，存取快速菜单来将各种标记关联于特定行。

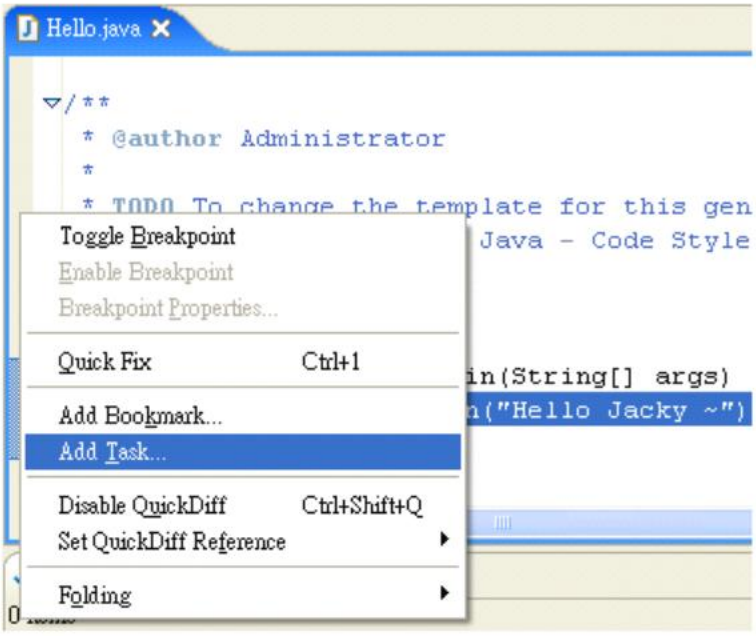


图 2.22

IV. 在 Description 字段中，输入要关联于文字文件中的这一行之作业的简要说明。

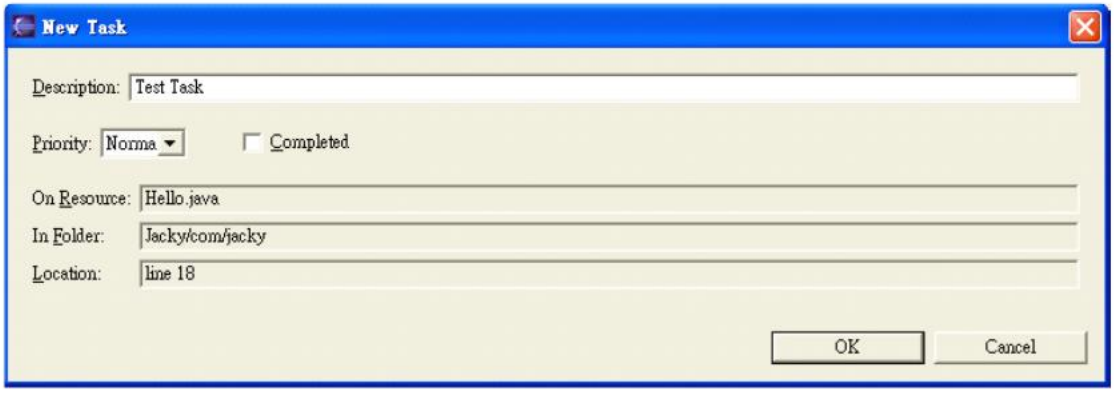


图 2.23

V. 完成之后，按一下 OK。

VI. 请注意，新的作业标记会出现在标记列中，就在新增作业的那一行左侧。另外，也请注意，新作业会出现在「Tasks」视图。

VII. 新增作业之后，请按一下编辑器中的第一行，或新作业所关联的行上面的任何其它行。

VIII. 在这点上，新增若干文字行到档案中。

IX. 请注意，当在上面新增了若干文字行时，作业标记会跟着档案中相关的行而在标记列中下移。当储存档案时，「Tasks」视图中的行号会被更新。

X. 在「Tasks」视图中，存取刚建立之作业的快速菜单。

XI. 选取 Mark Completed。

XII. 现在，从标记的快速菜单选取 Delete Completed Tasks。

XIII. 请注意，这时标记列的作业标记会消失，且会从「Tasks」视图中移除作业。

2.10.3 开启档案

「Tasks」视图提供两个开启作业的相关档案的方法：

- 选取作业，然后从快速菜单中，选择 Go To
- 按两下作业

这两种方法都会开启档案编辑器，且会标示出选取的作业所关联的那一行。

2.11 书签

书签是导览至常用资源最简单的方式。这一节要看看书签的设定和移除，以及在「Bookmarks」视图中检视它们。

「Bookmarks」视图会显示工作台中的所有书签。如果要显示「Bookmarks」视图，请在「Resource」视景中，选取「Window」→「Show View」→「Bookmarks」。

2.11.1 新增和检视书签

工作台可以用书签来标示个别档案或档案内的位置。这一节要示范如何利用「Bookmarks」视图来设定若干书签及检视它们。

I. 从菜单列中，选取「Window」→「Show View」→「Bookmarks」。这时「Bookmarks」视图会出现在工作台中。

II. 编辑 Hello.java 檔。

III. 将光标放在档案中任何一行旁的编辑器标记列上。然后，从标记列的快速菜单中，选取 Add Bookmark。

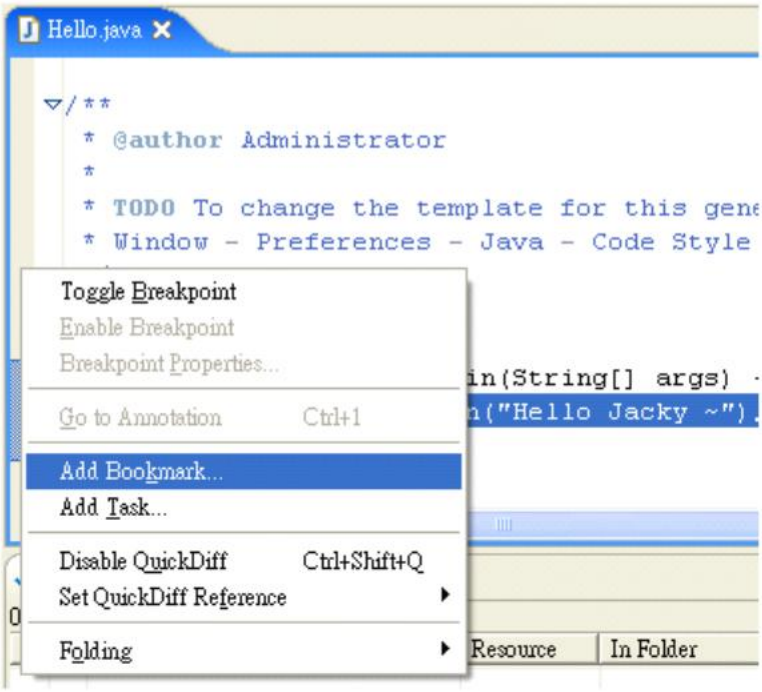


图 2.24

当「Add Bookmark」对话框开启时，输入这个书签的说明。请输入「我的书签」。

IV. 请注意，标记列中会出现一个新书签。

图 2.25

新书签也会出现在「Bookmarks」视图中。

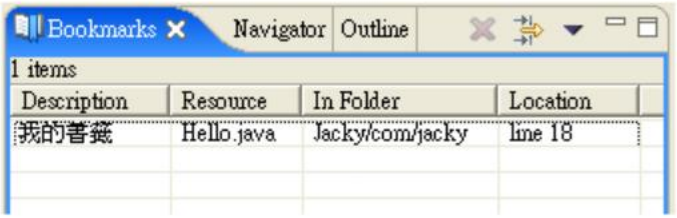


图 2.26

V. 在「Navigator」视图中，选取 Hello.java 檔。从主工作台菜单中，选取「Edit」→「Add Bookmark」。这将会使用文件名称说明书签，来建立档案的书签。现在，请看看「Bookmarks」视图，其中有两个书签。

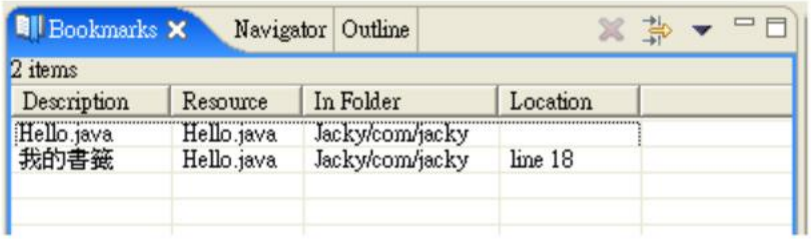


图 2.27

2.11.2 使用书签

建立好若干书签之后，现在，将提供一些指示来说明如何取得书签相关档案的指示。

I. 在编辑器区域中，关闭所有档案。

II. 在「Bookmarks」视图中，按两下所建立的第一个书签（我的书签）。

III. 请注意，这时会有开启的编辑器显示书签所关联的档案，且会标示出书签所关联的那一行。

附 注：「Bookmarks」视图支持用另一种方式来开启所选书签的相关档案，只要从书签的快速菜单中选取 Go To 就行了。

在「Bookmarks」视图中，选取导览器中的相关档案。

I. 在「Bookmarks」视图中，选取「我的书签」。

II. 从书签的快速菜单中，选择 Shoe in Navigator。

III. 请注意，现在可以见到「Navigator」视图，且会自动选取 Hello.java 檔。Hello.java 是「我的书签」所关联的档案。

2.11.3 移除书签

I. 在「Bookmarks」视图中，选取 Hello.java（我们建立的第二个书签），再执行下列其中一项动作：

- 按一下视图工具列中的「删除」按钮 。
- 从书签的快速菜单中，选取「删除」。
- 按一下键盘上的 Delete 键。

请注意，书签已从「Bookmarks」视图中移除。

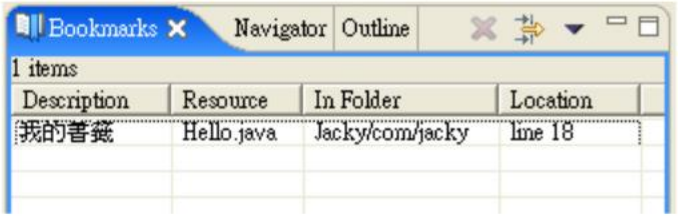


图 2.28

II. 这时应该还有一个书签。这个书签与 Hello. java 档其中一行相关。另外还有两种方法可以移除这个书签。

- 使用 Hello.java 编辑器标记列中的 Remove Bookmark。请记住，最初建立书签时，在标记列中使用 Add Bookmark。

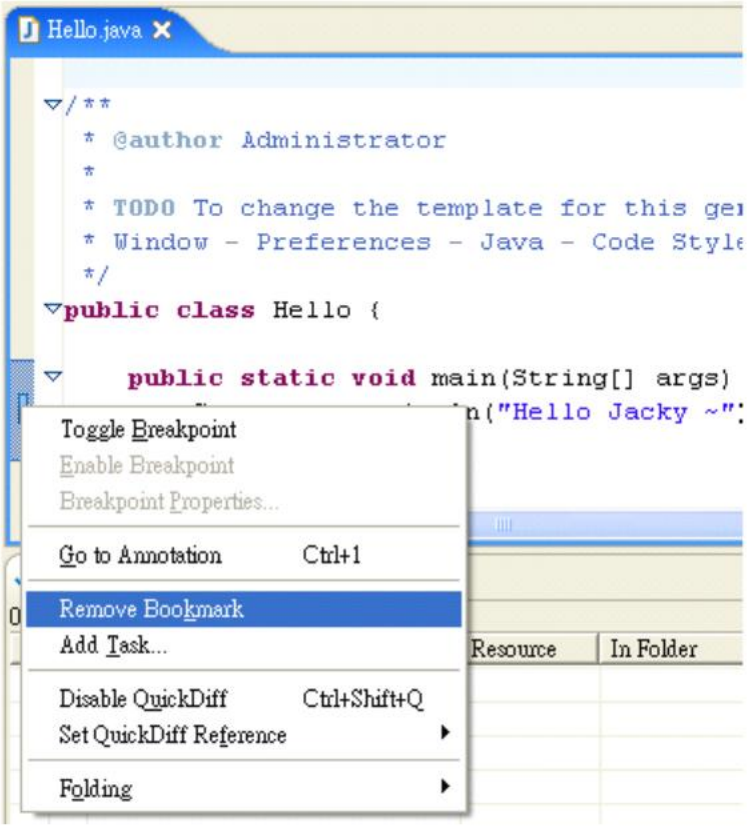


图 2.29

- 在「Bookmarks」视图中，利用书签蹦现菜单中的删除来删除书签（如上面所执行的动作）。

以下是第二个方式。

III. 确定有编辑器开启了 Hello.java。

虽然编辑器实际上不需要开启，但删除书签时，可以检视编辑器更新。

IV. 在「Bookmarks」视图中，选取 Hello. java（剩下的书签）。 按一下视图工具列中的「删除」按钮。请注意，书签已从「Bookmarks」视图及 Hello. java 编辑器中移除。

2.12 快速视图(Fast View)

快速视图是隐藏而可以快速显示的视图。它们的运作方式和一般视图相同，唯一不同之处是它们在隐藏时不会占据工作台窗口的画面空间。

这一节要说明如何将「Navigator」视图转换成快速视图。

2.12.1 建立快速视图

快速视图是隐藏而可以快速显示的视图。这些指示开始于从「Navigator」视图建立快速视图，之后，再说明完成快速视图之后要如何使用它。

以下是两个建立快速视图的方法

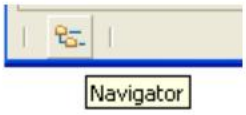
■ 使用拖放技术。

■ 使用视图「系统」菜单所提供的菜单作业。

请依照下列方式，利用拖放技术来建立快速视图。

I. 在「Navigator」视图中，按一下标题列，将它拖曳到窗口左下方的快捷方式列中。

II. 当游标到了快捷方式列，它会变成一个"快速视图"游标。请放开鼠标按钮，将导览器放在快捷方式列中 。现在，快捷方式列中会有导览器快速视图的按钮



如果要利用第二种方法来建立快速视图，首先在「Navigator」视图的卷标上蹦现快速菜单。请从这个菜单中，选取 Fast View。

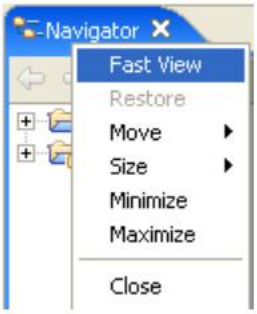
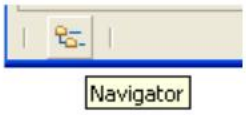


图 2.30

2.12.2 使用快速视图

现在，导览器已转换成快速视图。这一节要示范它现在能做什么。

请确认窗口左下方的快捷方式列仍有「Navigator」视图，且外观如下：



I. 在快捷方式列中，按一下「Navigator」快速视图按钮。

II. 观察「Navigator」视图从窗口左侧出现。

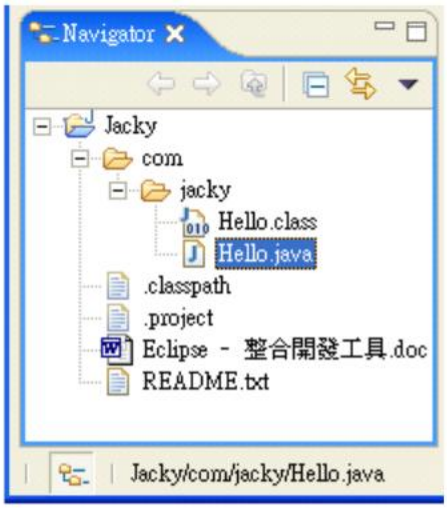


图 2.31

III. 可以依照正常方式来使用「Navigator」快速视图。如果要调整快速视图的大小，请将鼠标移到快速视图右缘，光标在该处会变成双箭头。之后，请按住鼠标左键来移动鼠标。

IV. 如果要隐藏快速视图，请按一下另一个视图或编辑器，或在快速视图的工具列中按一下最小化按钮。

附注： 如果从「Navigator」快速视图开启档案，快速视图会自动隐藏起来，让能够使用档案。

如果要将快速视图转换回正规视图，请执行下列动作之一：

- 从视图左上角图标的快速菜单中，选择快速检视。
- 从工具列拖曳快速检视图示，然后将它放置在工作台窗口某处。

2.13 比较

工作台可用来比较多项资源以及在特殊的比较编辑器中呈现结果。

开始比较之前，必须建立一些档案。

I. 利用项目的蹦现菜单来建立一个叫做 file1.txt 的档案。

在 file1.txt 的编辑器中，输入下面这几行文字，再将档案储存起来：

This is line 1.

This is line 2.

This is line 3.

This is line 4.

This is line 5.

II. 在导览器中，选取 file1.txt，再利用 Ctrl+C 来复制档案。

III. 使用 Ctrl+V（贴上）来建立副本。在出现的名称冲突对话框中， 将档案重新命名为 file2.txt。（在 Mac 中，请使用 Command+C 和 Command+V。）

现在，有两个相同的档案 file1.txt 和 file2.txt。

2.13.1 简单比较

在导览器中，选取 file1.txt 和 file2.txt，然后从快速菜单中，选取「Compare With」→「Each Other」。这时会出现一个对话框，指出两个档案相同。

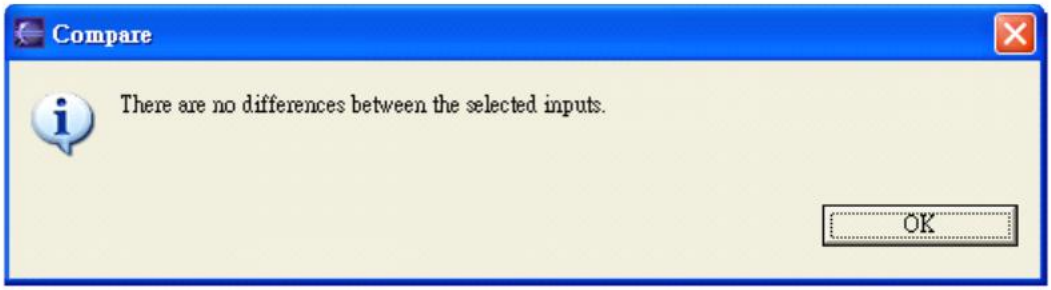


图 2.32

依照下列方式来编辑 file1.txt：

I. 删除第 1 行："This is line 1."

II. 将第 3 行改成 "This is a much better line 3."

III. 插入第 4a 行（在第 5 行之前），内容为："This is line 4a and it is new"

现在，档案 (file1.txt) 的内容应该如下：

This is line 2.

This is a much better line 3.

This is line 4.

This is line 4a and it is new

This is line 5.

储存档案的内容，方法是选取「File」→「Save」（或按 Ctrl+S）。

如果要比较档案，请再次选取 file1.txt 和 file2.txt，从「导览器」的快速菜单中，选取「Compare With」→「Each Other」。

这时会开启一个特殊比较编辑器。下一节将说明如何使用这个比较编辑器。

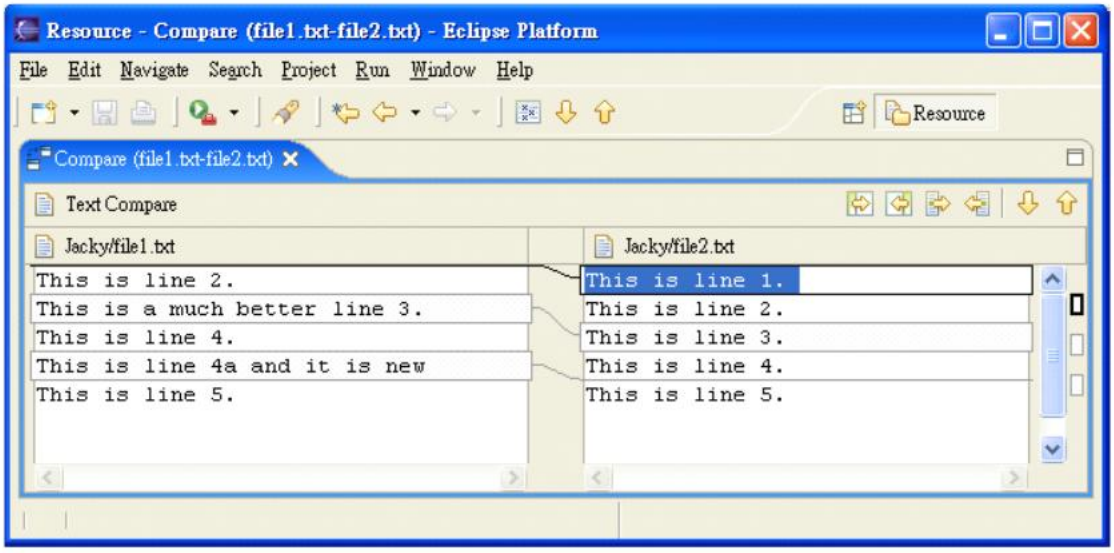


图 2.33

2.13.2 了解比较

请比较在下列比较编辑器中产生的 file1.txt 和 file2.txt。左侧显示 file1.txt 的内容，右侧显示 file2.txt 的内容。连接左侧和右侧的线表示档案之间的差异。

如果需要更多空间来查看比较，可以按两下编辑器标签，将编辑器最大化。

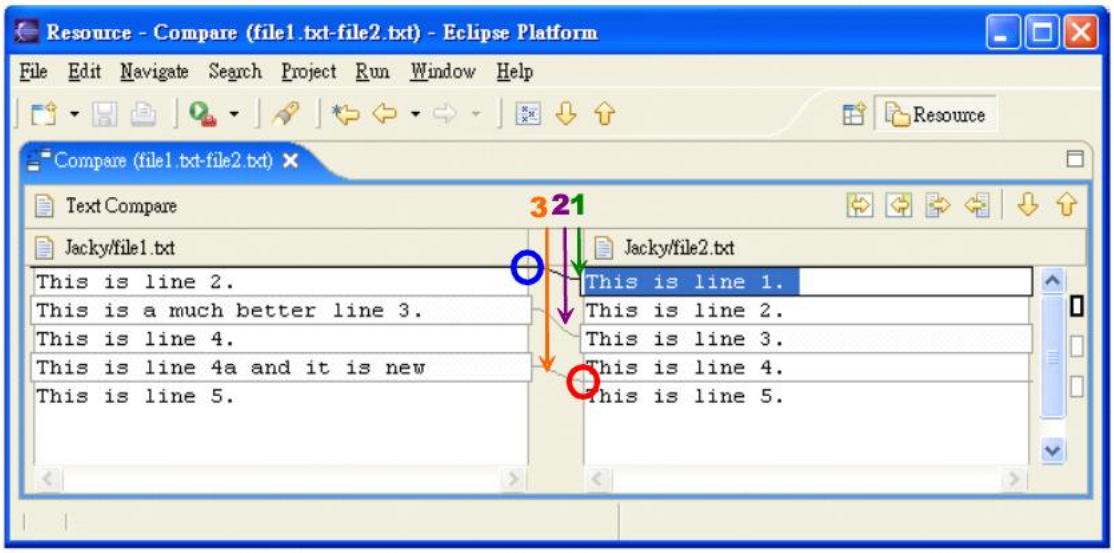


图 2.34

不同编辑器左侧的编号变更如下：

I. 从最上面一行开始（左窗格），差异列（在蓝圈区）指出左侧档案的最顶端遗漏了什么。请遵循右侧档案的差异群（请参阅 #1）。它含有 "This is line 1"。

II. 下一行 "This is line 2." 是白色，指出它符合右侧档案。

III. 移至下一行（背景颜色是彩色），可以看到左侧档案和右侧档案这一行的内容不同（请参阅 #2）。

IV. 下一行（第 4 行）又是白色，因此，可以跳过它。

V. 下一行是在左侧档案中，但由于它使用背景颜色，可以沿着它的右侧差异列（请参阅#3），注意到右侧档案并没有包含这一行（请参阅红色圆圈）。

开始时，比较编辑器会有点令人气馁，但当沿着左侧向下作业，将焦点放在有灰色标示的项目以及左侧中所没有的项目时，就不会像原先那么不好处理。

2.13.3 使用比较

请比较在下列比较编辑器中产生的 file1.txt 和 file2.txt。这一节要示范如何使用比较编辑器来解析两个档案之间的差异。

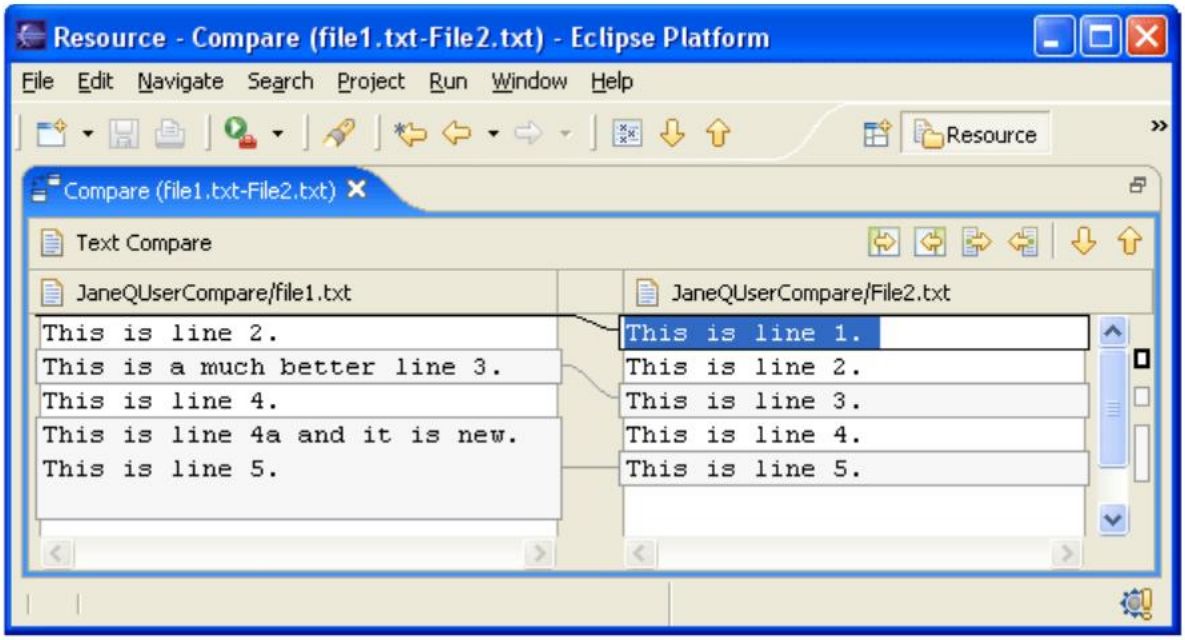


图 2.35

比较编辑器的本端工具列有两个部分。请利用右侧的本端工具列按钮群组来移至下一个或上一个变更。

I. 按一下「选取下一个变更」按钮。请观察它如何选取下一个差异。

II. 再按一次「选取下一个变更」按钮，移至下个变更。

III. 按一下「选取上一个变更」按钮。

如果要将左侧档案的变更合并到右侧档案中，请使用左侧的本端工具列按钮群组，反之亦然。可以执行四类型的合并：

- 由左向右复制整份文件。
- 由右向左复制整份文件。
- 由左向右复制现行变更。
- 由右向左复制现行变更。

通常，当左或右侧的整个档案可由其它档案的内容来取代时，都会使用复制整份文件的动作。

「复制现行变更」按钮可以合并单一变更。

I. 确定已选取第二个差异（如下所示）：

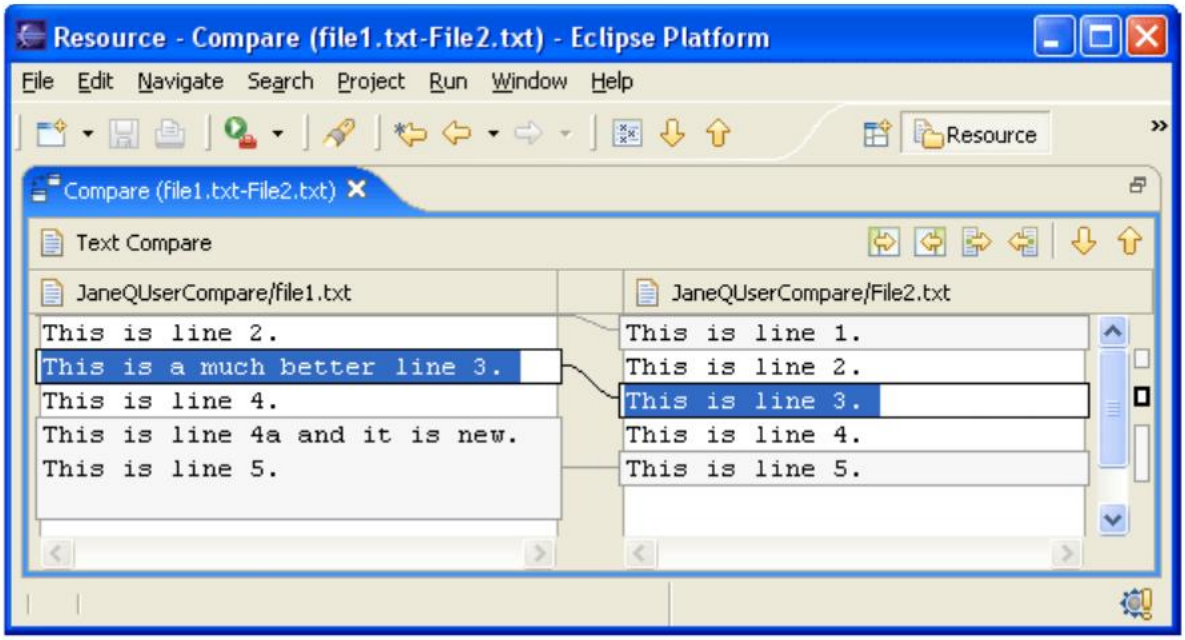


图 2.36

II. 按一下从右向左复制现行变更 。观察右侧档案中的所选文字，现在已复制到左侧档案中。

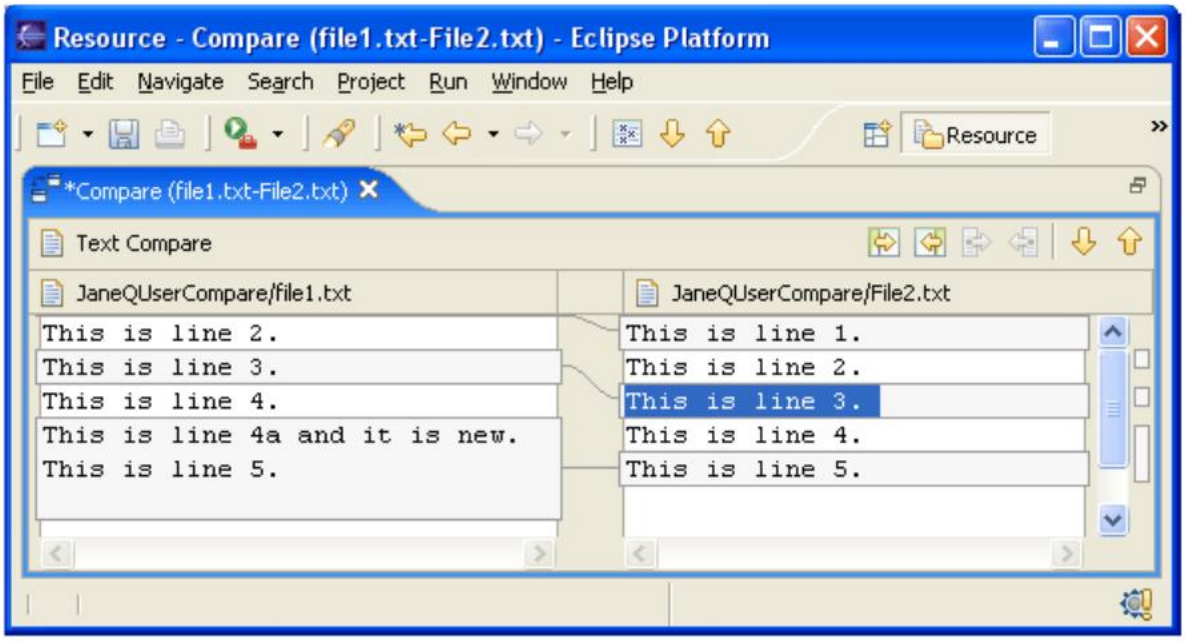


图 2.37

III. 关闭比较编辑器，然后选择 OK 来储存变更。另外，也可以选择「File」→「Save」 (Ctrl+S) 来储存变更。

2.14 历史纪录

每次在工作台中储存可编辑的档案时，工作台都会更新这个档案的历史纪录，且会将变更记载下来。之后，只要所需状态不是太久以前，仍在储存历程中，就可以存取档案的历史纪录且可以回复到先前所储存的档

案复本。

I. 建立名称为 sampleFile.txt 的新档案。

II. 在 sampleFile.txt 的编辑器中修改资源，先新增 "change1" 这一行，再将档案储存起来。

III. 输入新的一行 ″change2″，再重新储存它，以重复这个动作。

IV. 新增第三行 ″change3″，再重新储存它。

V. 从「Navigator」视图中的资源的快速菜单中，选取「Replace With」→「Local History...」。

VI. 这时会开启「从历史纪录取代」对话框，且会显示档案先前的历史纪录。

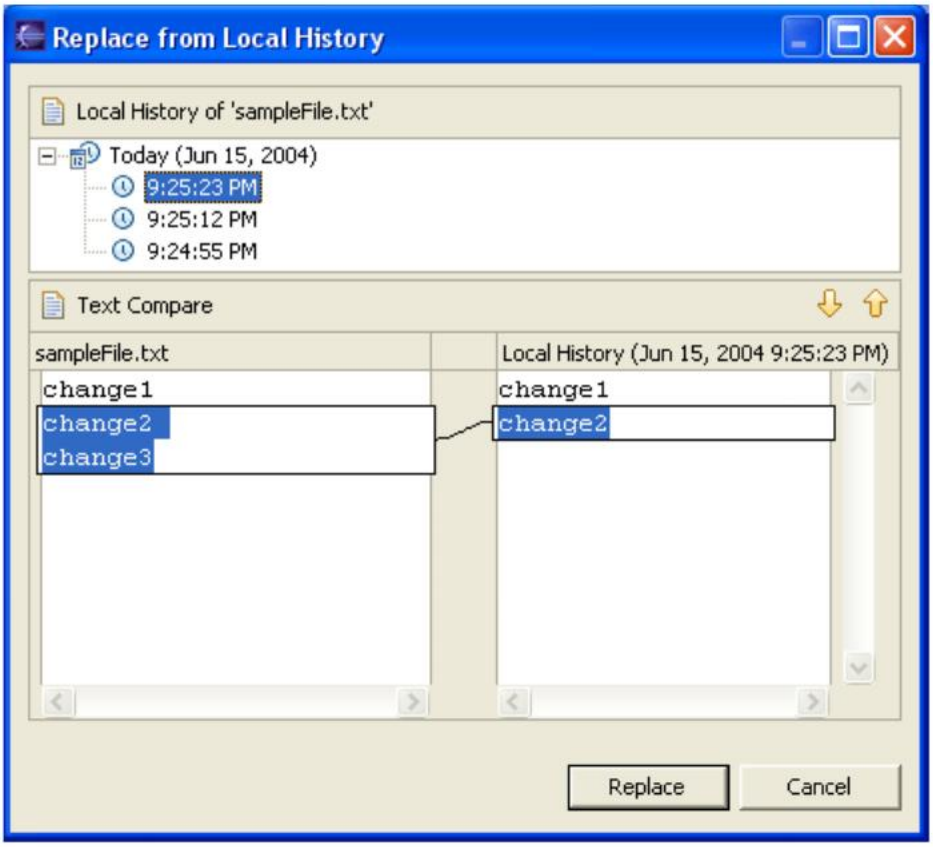


图 2.38

对话框左窗格含有档案的工作台副本。上图显示工作台包含的复本有完整的三行，也就是工作台编辑区目前所显示的相同复本。

历史纪录中的第一个项目（请参阅上面）含有最后储存的档案副本。这是只有两行文字的副本。树状结构中最终项目是档案的第一个副本。

对话框底端区域会显示工作台档案和历史纪录中所选取的特定档案复本的差异。

VII. 选取历史纪录中的第一个项目（如上所示）。右窗格应该会显示一行的文字。

VIII. 按一下取代。这会以所选历史纪录项目来取代 sampleFile.txt 的工作台副本。

IX. 观察 sampleFile.txt 编辑器，现在它有两行。

2.15 回应 UI

依预设，所有 Eclipse 作业都是在使用者接口执行绪中执行的。当使用接受序列化程序代码执行绪作业的响应 UI 时，仍可以在 Eclipse 的其它位置中作业。如果没有响应 UI 支持，当遇到速度慢的作业时，会

被锁定，因而无法执行任何其它动作。

虽然部分作业会自动在背景中执行（如自动建置），但在许多情况下，都会显示一个对话框来提供在背景中执行作业的选项。比方说，手动建置项目有时要多花一些时间，在这期间，仍可以在 Eclipse 中继续使用其它功能。

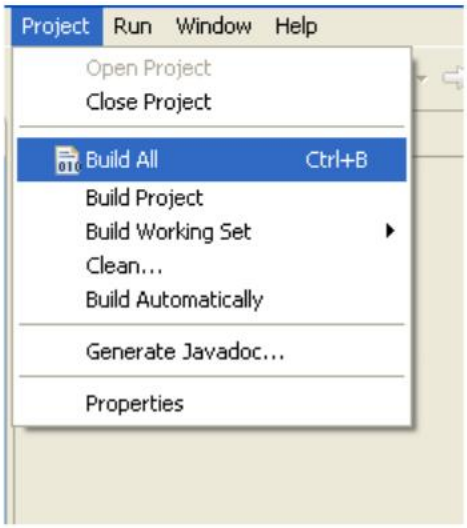


图 2.39

当建置项目时，请从「Project」对话框中选取 Run in Background，响应 UI 可以在 Eclipse 中执行其它作业。

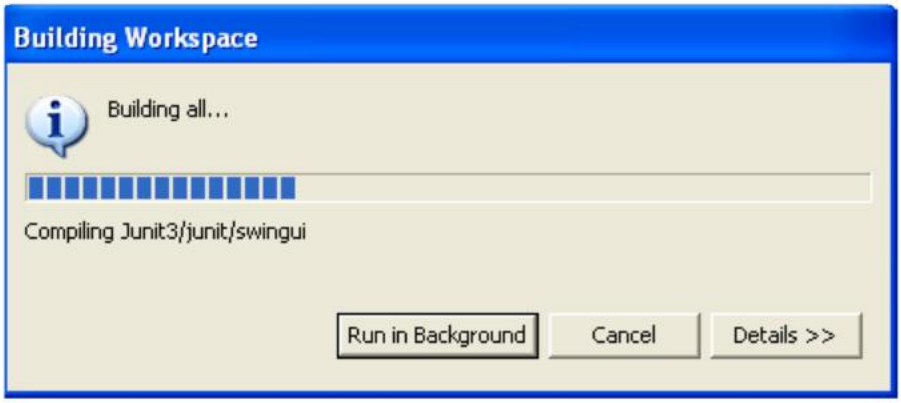
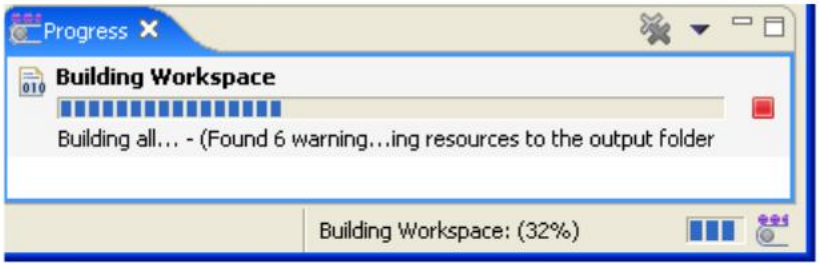


图 2.40

如果需要动作状态及目前在执行的其它作业的相关信息，请按一下 Details。



Detail 画面会显示即将执行的作业以及可能同时在执行中的任何其它作业的状态信息。

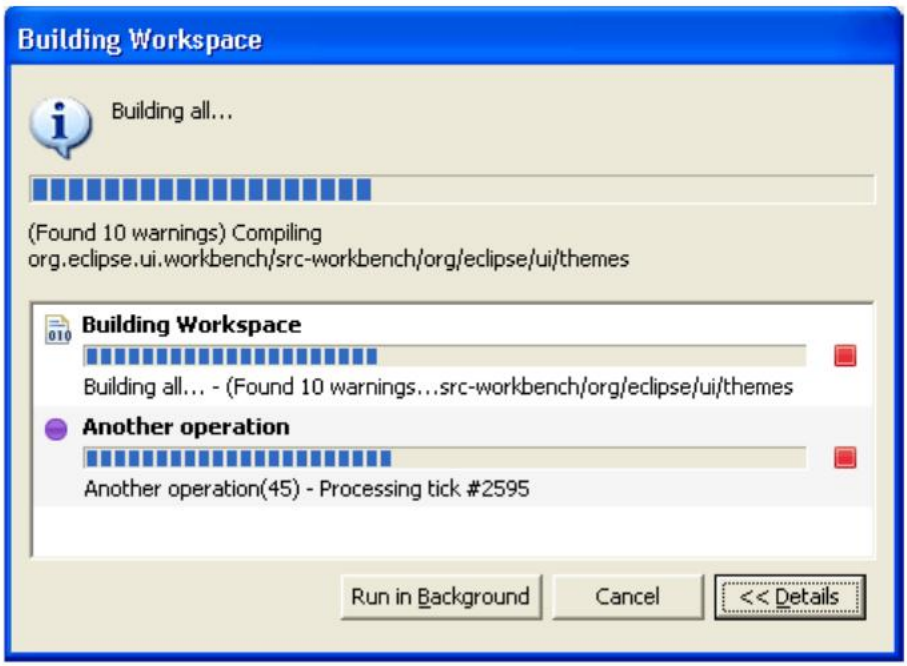


图 2.41

当某项作业被其它作业锁定时，「Progress Information」对话框也会加以指示。

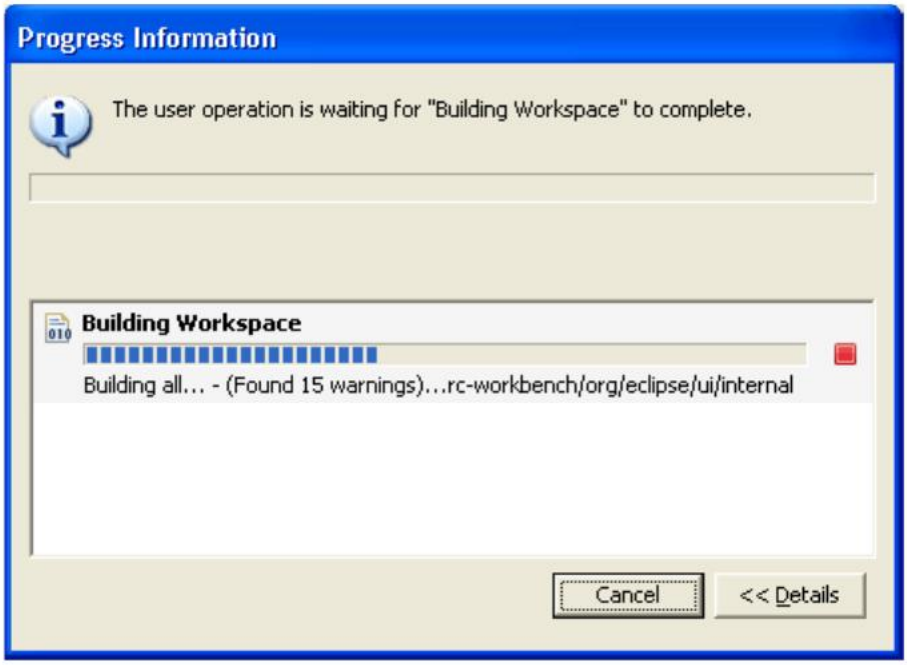


图 2.42

如果要将在背景中执行的作业设为默认值，请选取「Window」→「Preferences」→「Workbench」，再勾选 Always run in background。

图 2.43

3.喜好设定(Preferences)

「Preferences」对话框是用来设定使用者喜好设定的对话框。可以从「Window」→「Preferences」找到这个对话框。由外挂程序组成的喜好设定页面也可以在这个对话框中找到。

喜好设定大部分的功能都是由其个别页面所定义，但对话框提供了两个一般功能：

Import：汇出会将对预设喜好设定的任何变更写入到使用者指定的档案中。

Export：汇入会套用使用者指定的档案中的喜好设定。

喜好设定对话框的外观如下：

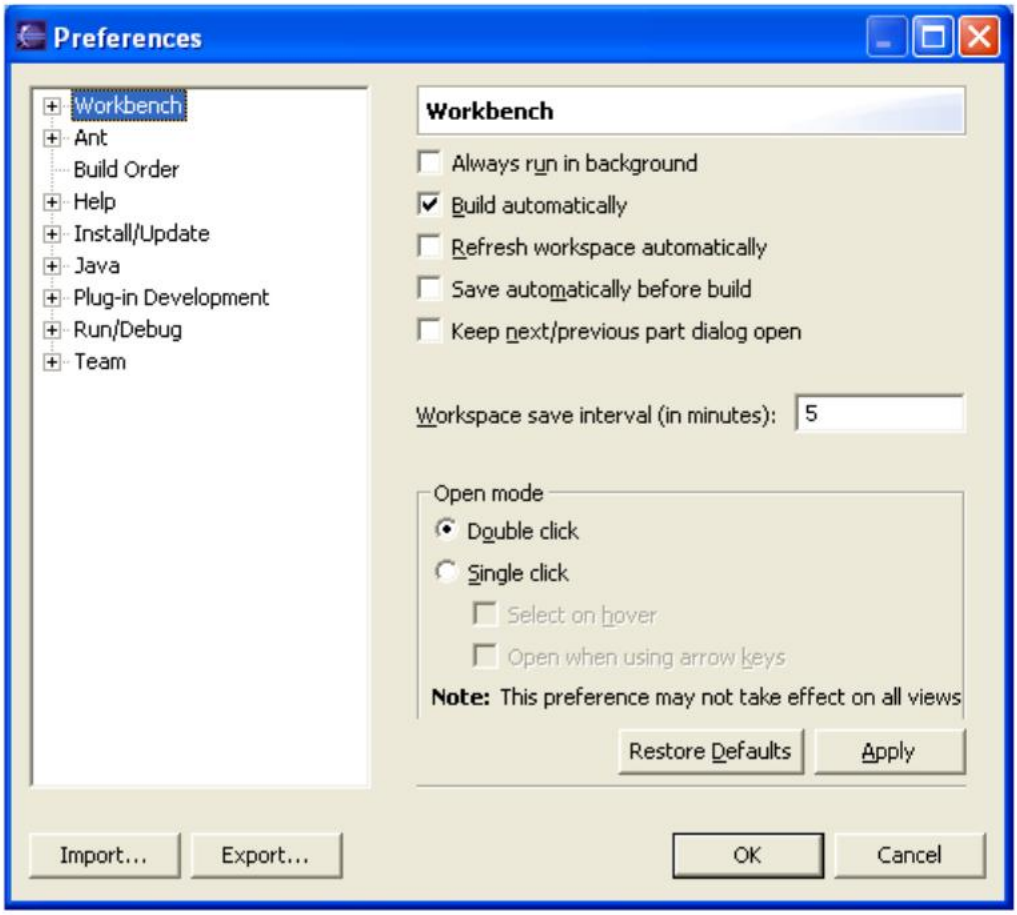


图 3.1

3.1 工作台(Workbench)

*工作台*一词指的是桌面开发环境。

每一个工作台窗口都含有一个或多个视景。视景则包含视图和编辑器，以及控制在某些菜单和工具列上出现的项目。在任何给定的时间里，可有多个工作台窗口存在于桌面上。

可以在工作台页面中变更下列喜好设定。

选项	说明	默认值

选项	说明	默认值
Always run in background(一律在背景中执行)	如果开启这个选项，工作台会在背景中执行特定动作，不会打扰到使用者。	关闭
Build automatically(自动建置)	如果开启这个选项，每次储存修改过的资源时，工作台都会自动执行建置动作。	开启
Refresh workspace automatically(自动重新整理工作区)	如果开启了这个选项，工作区资源自动与档案系统中对应的资源同步化。 附注：这有可能会是一项冗长的作业，这会随着工作区中的资源数目而不同。	关闭
Save automatically before build(建置之前自动储存)	如果开启这个选项，每次执行手动建置（从菜单列中，选取可用选项的「项目」）时，工作台都会自动储存前次执行建置之后又修改过的所有资源。	关闭
Keep next/previous part dialog open(保持开启下一个/上一个组件对话框)	如果开启了这个选项，当编辑器和视图循环对话框的启动键放开时，对话框仍会维持开启状态。通常在按键组合放开时，会立即关闭对话框。	关闭
Workspace save interval (in minutes)(工作区储存间隔（以分钟为单位）)	这个数字指出工作区的状态自动储存至磁盘的频率。	5
Open mode...(开启模式...)	可以选取下列方法之一，来开启资源： 按两下 - 按一下资源将会选取它，按两下资源则会在编辑器中开启它。 按一下（浮动说明时选取）- 将鼠标光标横越在资源上将会选取它，在资源上按一下则会在编辑器中开启它。 按一下（使用方向键时开启）- 以方向键选取资源时，会在编辑器中开启它。 请注意，视哪一个视图具有焦点而定，选取及开启资源可能有不同的行为。	按两下

工作台喜好设定页面看起来如下：

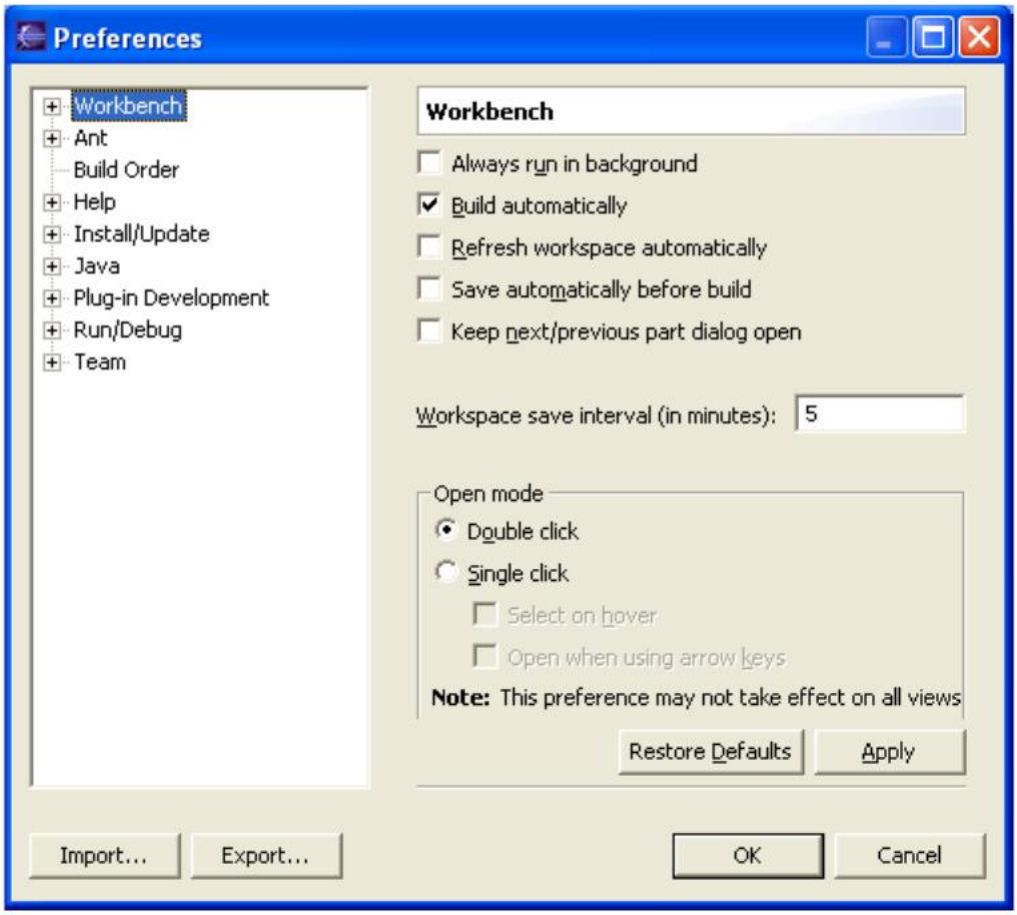


图 3.2

3.1.1 外观(Appearance)

可以在「外观」页面中变更下列喜好设定。

选项	说明	默认值
Tab positions - Editors(卷标位置 - 编辑器)	指定顶端或底端，以指出希望堆栈的编辑器的卷标出现的位置。	顶端
Tab positions - Views(卷标位置 - 视图)	指定顶端或底端，以指出希望堆栈的视图的卷标出现的位置。	顶端
Perspective switcher position(视景切换器位置)	请指定视景切换器列的位置	右上
Current presentation(现行呈现方式)	请指定目前作用中的呈现方式（外观和操作方式）。	3.0 呈现方式
Current theme(现行主题)	请指定目前在作用中的主题（颜色和字型集）。	3.0 主题
Show text on perspective bar(在视景列显示文字)	请指定应不应该在视景列及图标中显示卷标。	已启用

选项	说明	默认值
Show traditional style tabs(显示传统样式卷标)	请指定应不应该用传统（方块）卷标来取代曲线卷标	已停用

「外观」喜好设定页面看起来如下：

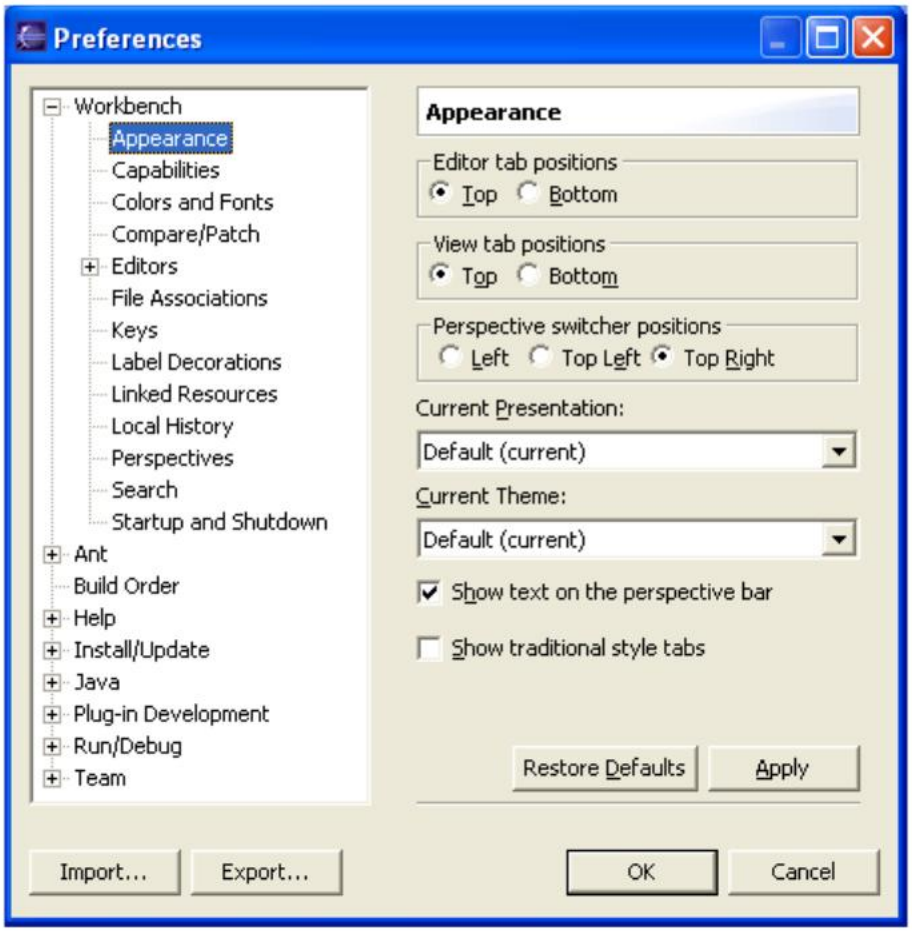


图 3.3

3.1.2 功能(Capabilities)

「功能」喜好设定页面可以启用或停用各种产品组件，如 Java 开发和外挂程序开发。

附注：部分功能选项会相依于其它功能，停用某必要功能，却仍启用相依的功能，结果只会重新启用它们。当取消选取 Java 开发和核心团队支持时，就是如此。

「功能」喜好设定页面看起来如下：

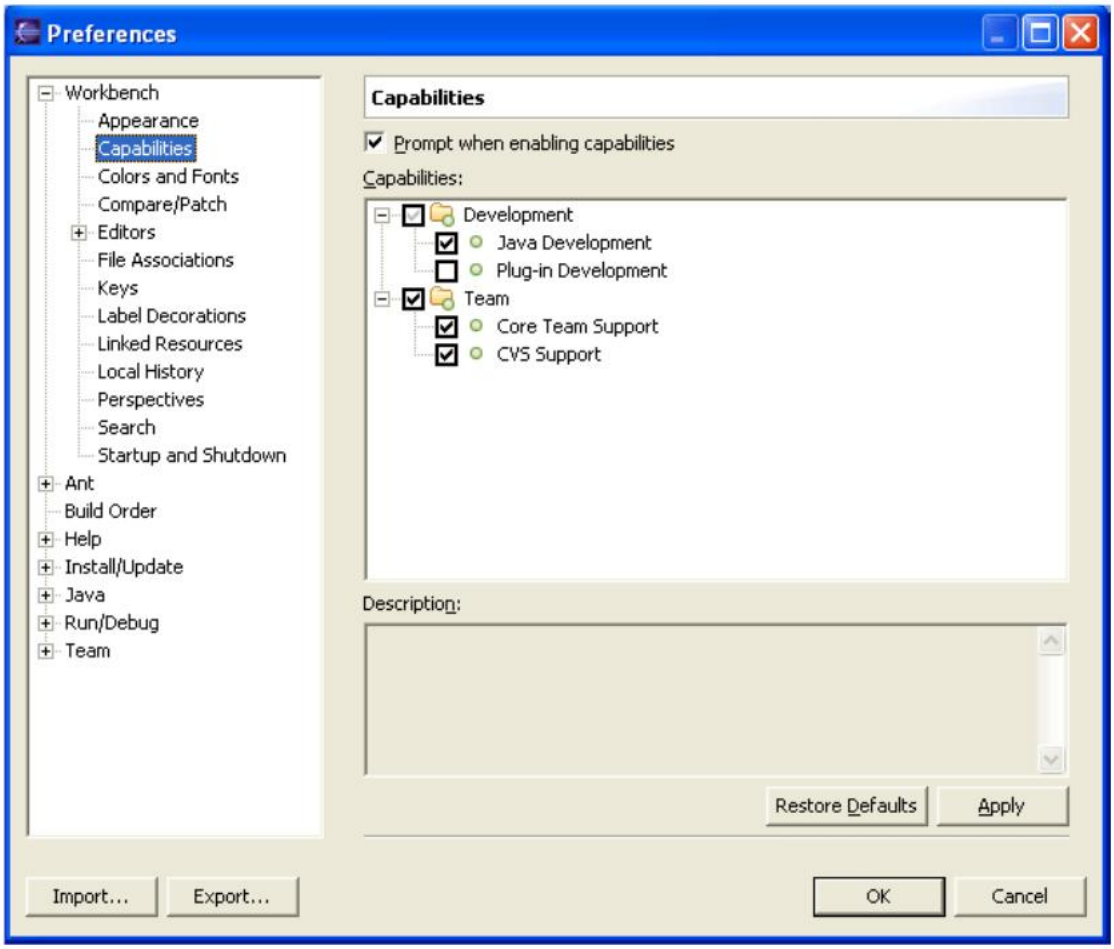


图 3.4

当试图启用某个动作，但它的功能先前已停用或尚待喜好设定页面予以启用时，会出现下列 Confirm Enablement 提示，供确认确实要启用必要的功能。请按一下 Details 来显示功能的说明。

图 3.5

3.1.3 颜色和字型(Colors and Fonts)

可以利用「颜色和字型」喜好设定页面来设定 Eclipse 组件所用的许多字型和颜色。

树状结构用来导览各种颜色和字型，以及显示各种颜色和字型的预览。任何字型的现行样式（不是大

小）预览都会出现在它的标签中。颜色的预览则会出现在卷标的相关图标中。另外，部分种类（尤其是工作台）会提供更详细的构成要素预览。这个预览如果可用的话，会显示在说明之下。

可以从清单中选取字型区，再按一下 Use System Font 来选取操作系统字型设定，或按一下 Change 来开启选取字型的对话框，以变更字型设定。Reset 可用来返回默认值。

当选取了某个字型时，可以按一下树状结构区右侧的颜色按钮来变更颜色设定。Reset 可用来返回默认值。

「颜色和字型」喜好设定页面看起来如下：

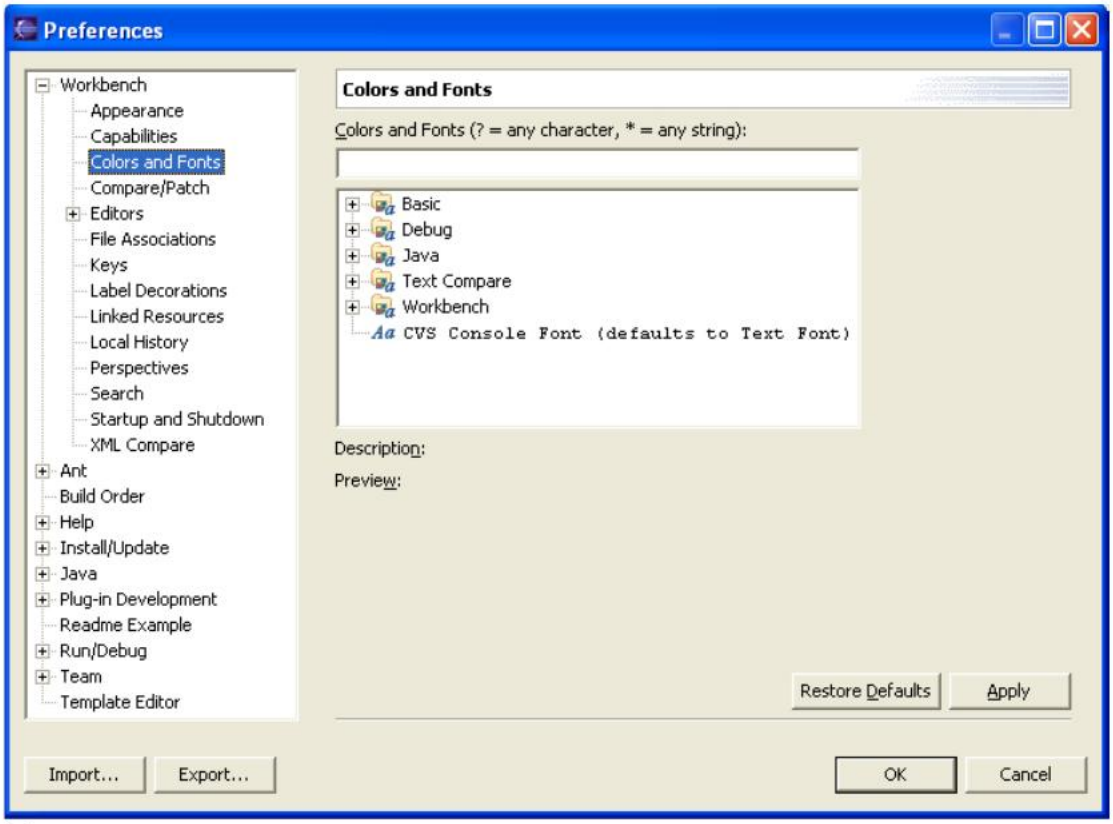


图 3.6

「颜色和字型」文字字段可用来过滤内容。只需要输入一个项目，任何相符的结果都会保留在树状视图中 。当选取工作台颜色和字型设定时，会提供说明和预览。

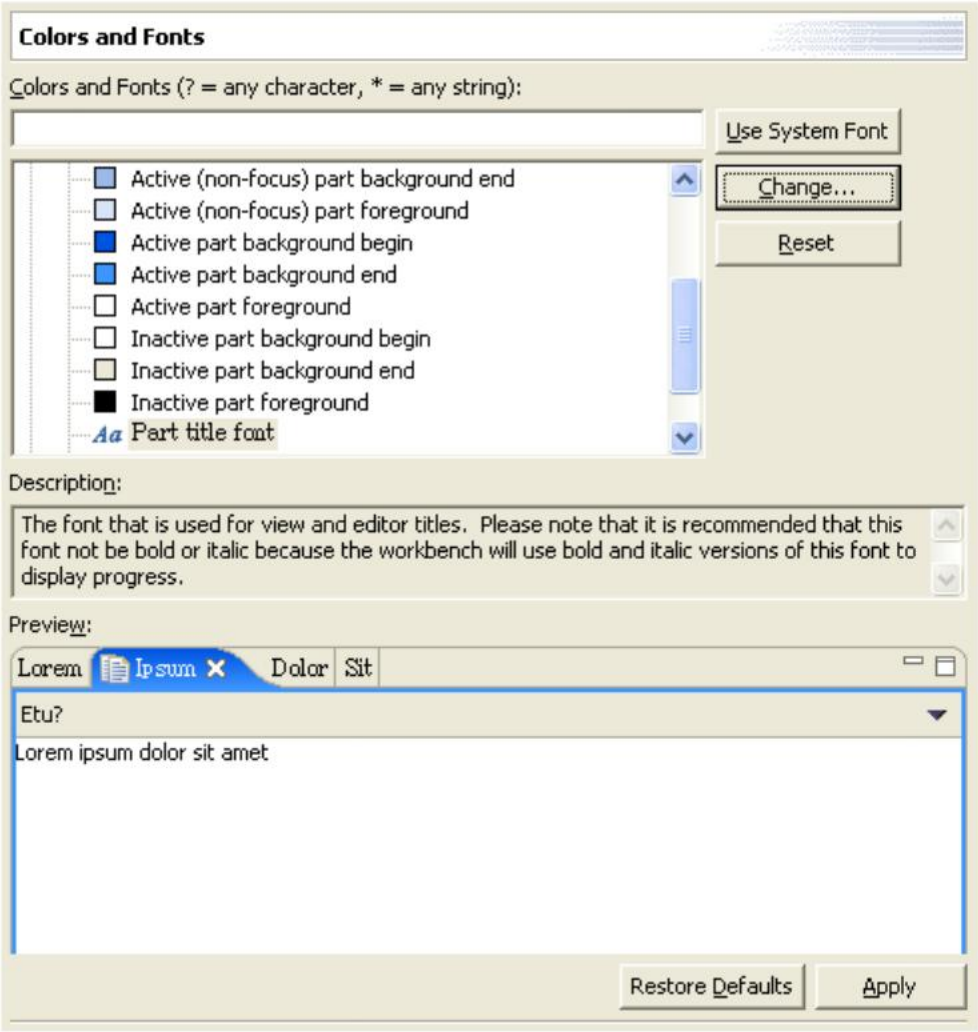


图 3.7

3.1.4 比较/修正(Compare/Patch)

可以在「比较/修正」页面中变更下列喜好设定。

一般选项(General)

选项	说明	默认值
Open structure compare automatically(自动开启结构比较)	这个选项控制是否要在每当内容完成时自动执行结构比较。如果不想看到结构上的差异，请关闭这个选项。	开启
Show additional compare information in the status line(在状态行显示其它的比较信息)	如果开启这个选项，则状态行中会显示关于变更的其余信息。如果有意了解变更的其余信息，请开启这个选项。	
Off(关闭)		

选项	说明	默认值
Ignore white space(忽略空格)	这个选项控制比较检视器中是否要显示空白变更。 如果想看到空白变更，请开启这个选项。	关闭
Automatically save dirty options before patching(在修正前自动储存变动过的选项)	这个选项可控制在套用修正前是否要自动储存任何尚未储存的变更。 如果要自动储存变更，请开启这个选项。	关闭

文字比较选项

选项	说明	默认值
Synchronize scrolling between panes in compare viewers(同步化卷动比较检视器中的窗格)	两个比较检视器将会与对方一起「锁定卷动」(lock scroll)，以便使每一个窗格内的程序代码的相同及对应部分并列显示。 如果不希望比较检视器锁定卷动，请关闭这个选项。	开启
Initially show ancestor pane(起始显示上代窗格)	有时会想比较资源的两个版本与衍生它们的先前的版本。这称为它们的共同上代，在三向比较期间，它会出现在其自己的比较窗格内。 如果希望上代窗格固定在比较开始时出现，请开启这个选项。	关闭
Show pseudo conflicts(显示虚拟冲突)	显示虚拟冲突，此冲突是在两位开发人员进行相同的变更（例如，两者都新增或移除完全相同的程序代码行或批注）时发生。 如果要虚拟冲突出现在比较浏览器中，请开启这个选项。	关闭
Connect ranges with single line(包含单行的联机范围)	控制不同的范围是否要由单行或两行所区隔的范围来进行可视化联机。	开启

「比较」喜好设定页面看起来如下：

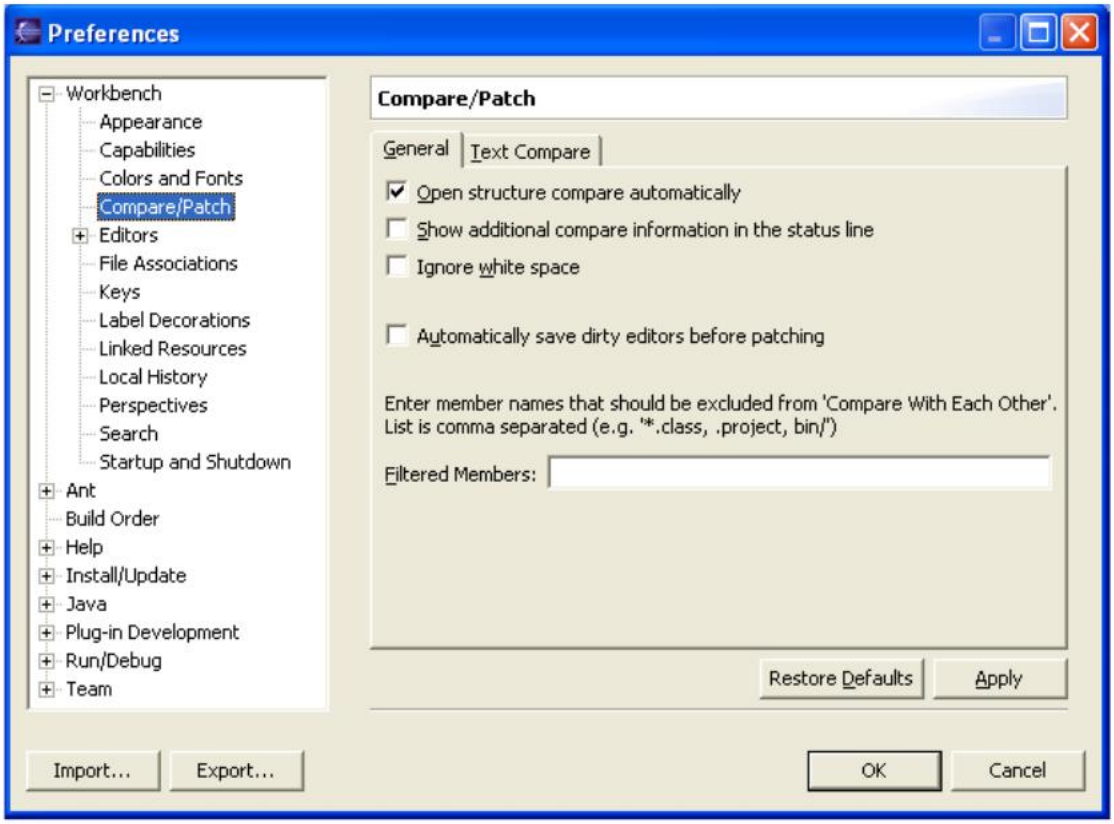


图 3.8

3.1.5 编辑器(Editors)

可以在「编辑器」页面中变更下列喜好设定。

选项	说明	默认值
Size of recently opened files list(最近开启的档案清单的大小)	以在编辑器中开启的每一个档案而言，它会被储存在最近使用档案的清单中。这个选项控制显示在「档案」菜单中的这个清单中的档案数目。	4
Show multiple editor tabs(显示多重编辑器卷标)	指定是否要显示多重编辑器卷标。如果关闭的话，编辑器活页簿会有一个大标签，所有不可见的编辑器都只能从 chevron 存取。	开启
Close all editors on exit(结束时关闭所有的编辑器)	这个选项是用来指定是否要在结束工作台时关闭所有的编辑器。启用这个选项时，可以加速 Eclipse 的启动， 因为它会减少启动 Eclipse 所需要的工作量。	关闭
Close editors automatically(自动关闭编辑器)	这个选项用来指定是否要在工作台中重复使用编辑器。如果开启，则可以指定在循环使用编辑器之前要使用的编辑器数目（默认值是 8）。也可以指定当所有的编辑器都已用过时，是否要开启提示对话框或新的编辑器。一旦开启，就会将「固定编辑器」动作新增至工具列和编辑器卷标菜单。固定编辑器并不会循环	关闭

选项	说明	默认值
	使用。	
Text File Encoding(文字文件编码)	请使用这个选项来指定在编辑器中储存文字文件时要使用的编码。	默认值(CP1252)

「编辑器」喜好设定页面看起来如下：

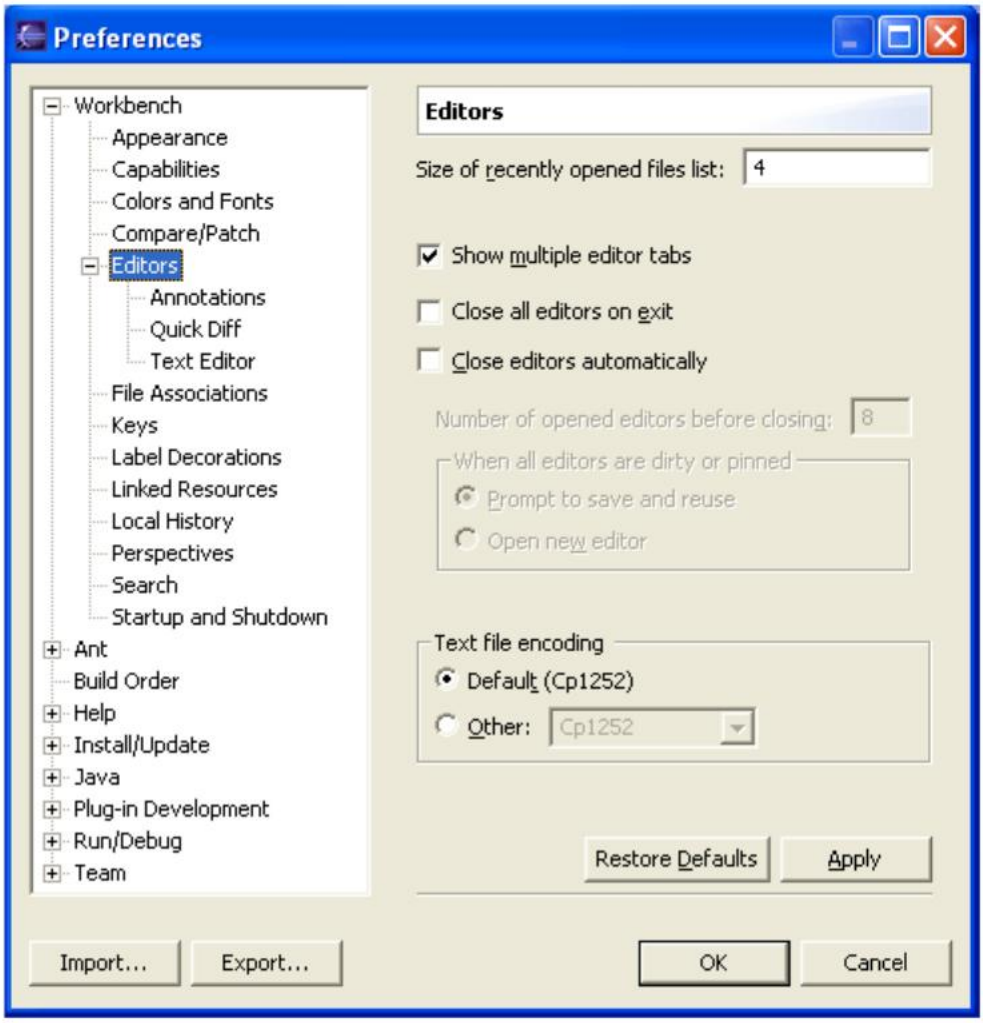


图 3.9

3.1.6 档案关联(File Associations)

在「档案关联」喜好设定页面上，可以新增或移除工作台可辨识的档案类型。也可以建立编辑器与档案类型清单中的档案类型的关联性。

档案类型清单(File Associations)

新增：将新的档案或档案类型（扩展名）新增至预先定义的清单。在产生的「新档案类型」对话框中，输入档案的名称或扩展名。如果要新增扩展名，则必须在档案类型前面输入一点或 "*."（例如，".xml" 或 "*.xml"，而非只是 "xml"）。

移除：从清单中移除所选取的档案类型

建立新档案类型的对话框：

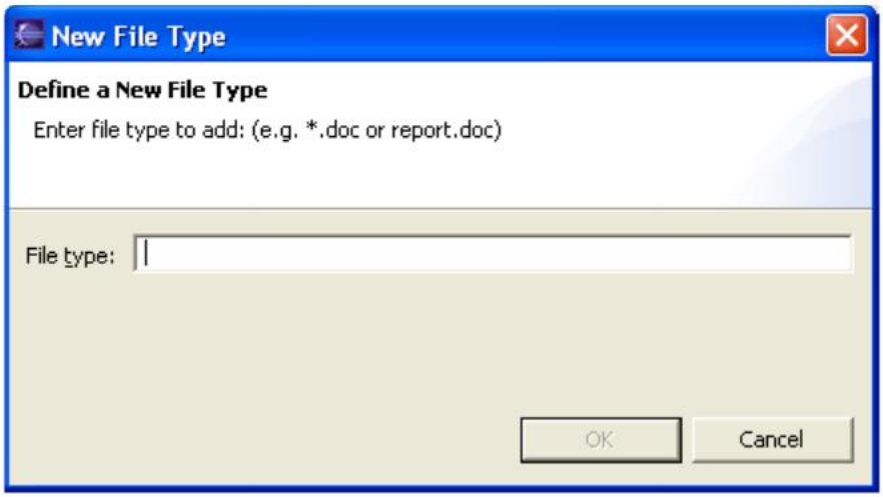


图 3.10

相关编辑器清单(Associated Editors)

新增：将新编辑器新增至与上述选取之档案类型相关联的编辑器清单。在产生的「编辑器选择」对话框中，可以选择编辑器，以在工作台内（内部）或工作台外（外部）启动；如果所要的编辑器未显在清单中，请按一下 Browse 来自行寻找编辑器。

移除：移除编辑器与上述选取之档案类型之间的关联。

默认值：将所选取的编辑器设为上述选取之档案类型的预设编辑器。该编辑器会移至「相关联的编辑器」清单的顶端，以表示它是该档案类型的预设编辑器。

建立新档案关联的对话框：

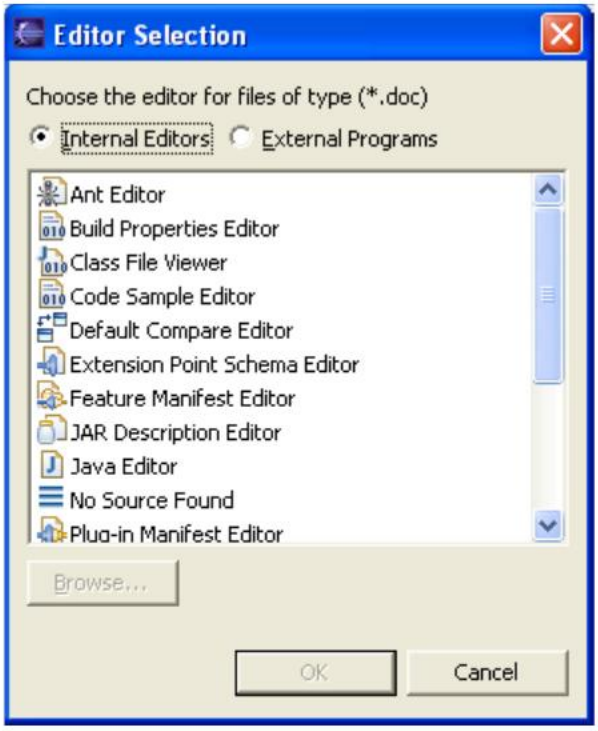


图 3.11

「档案关联」喜好设定页面看起来如下：

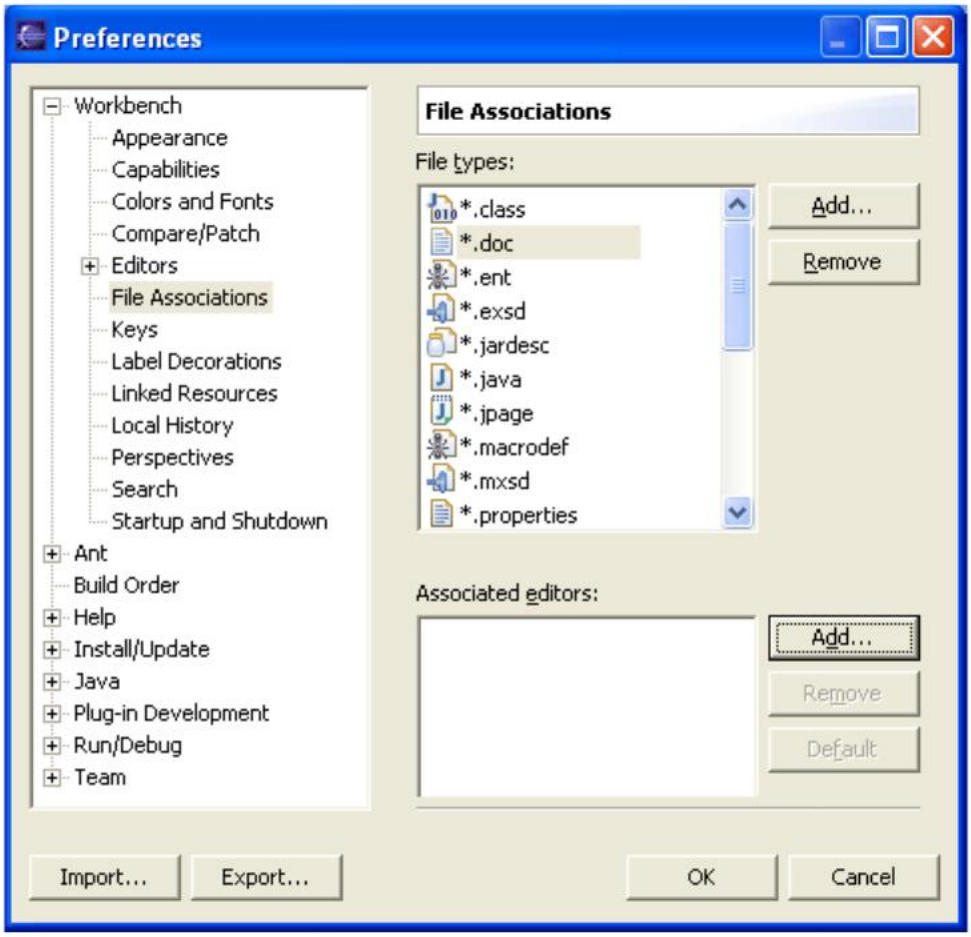


图 3.12

3.1.7 按键(Keys)

在 Eclipse 中，可以广泛自订键盘的功能。Eclipse 中指定了许多按键作用和按键顺序来呼叫特定的指令。

按键作用(Key Strokes)、按键顺序(Key Sequences)和按键连结(Key Bindings)

「按键作用」是指按下键盘上的某个按键，同时选择性按住一或多个下列修正键： Ctrl、Alt（在 Macintosh 上为 Option)、Shift 或 Command（只有 Macintosh 才有。）比方说，先按住 Ctrl，然后按 A 的时候，就会产生按键作用 Ctrl+A。单独按下修正键时，并不会组成按键作用。

「按键顺序」是指一或多个按键作用。传统上，emacs 会指定两个或三个按键作用的按键顺序给特定的指令。例如，在 emacs 中，指派给全部关闭的正常按键顺序是 Ctrl+X Ctrl+C。如果要输入这个按键顺序，必须按下按键作用 Ctrl+X，然后按下按键作用 Ctrl+C。虽然 Eclipse 支持任意长度的按键顺序，但键盘快捷方式的长度最好不超出四键。

「按键连结」是将按键顺序指派给指令。

配置(Configurations)

「配置」是指一组按键连结。Eclipse 包含两种配置：

- 默认值

- Emacs（延伸预设）

预设配置包含一组通用的按键连结，在许多情况下，使用者可以将它们视为传统的按键顺序。例如，*Ctrl+A* 是指派给*全选*，*Ctrl+S* 是指派给*储存*等。

Emacs 配置包含一组 Emacs 使用者非常熟悉的按键连结。例如，Ctrl+X H 是指派给全选，Ctrl+X S 是指派给储存等。

必须了解为什么 ***Emacs*** 配置说它「延伸预设」，这一点很重要。与***预设***配置不同，***Emacs*** 配置并不是一组完整的按键连结。相反的，它会尽可能借用***预设***配置，而且只会针对与***预设***配置不同的地方来定义明确的 Emacs 样式按键连结。通常，诸如*全选*、*储存*等常用的指令才会关联于特定的 Emacs 按键顺序。

使用者可以变更按键喜好设定页面中的「作用中的配置」设定，来决定最喜欢使用的配置。如果使用者选择***预设***配置，便会忽略所有 ***Emacs*** 按键连结。如果使用者选择 ***Emacs*** 配置，则明确指派的 Emacs 样式按键顺序会优先于***预设***配置中任何冲突的指派。

环境定义(Contexts)

按键连结可能会因为 Eclipse 的现行环境定义而有所不同。

有时候，例如，作用中的组件可能会是 Java 档案编辑器，这时指派不同组的按键顺序可能会比作用中的组件是 HTML 档案编辑器更合适。*Ctrl+B* 是一项特定的范例，在 Java 档案编辑之类的环境定义中，*Ctrl+B* 通常是指派给*建置*，而在 HTML 档案编辑之类的环境定义中，则是指派给*将文字变为粗体字*。这个环境定义通常是由作用中的组件来决定，但它也可能受到作用中的窗口或对话框的影响。如果作用中的组件没有选择特定的环境定义，工作台会将作用中的环境定义设定为***在窗口中***。

Eclipse 包含七种不同的环境定义。它们是：

- 在对话框和窗口中
- 在窗口中（延伸「在对话框和窗口中」）
- 在对话框中（延伸「在对话框和窗口中」）
- 编辑文字（延伸「在窗口中」）
- 编辑 Java 程序代码（延伸「编辑文字」）
- 除错（延伸「在窗口中」）
- Java 除错（延伸「除错」）

环境定义与配置类似，它们可以延伸其它的环境定义。比方说，***编辑 Java 程序代码***环境定义会借用***编辑文字***环境定义的按键连结，后者又会从***在窗口中***环境定义借用按键连结。

附注：不建议将按键连结提升到它所延伸的环境定义。比方说，最好不要将***编辑文字***按键连结移到***在对话框和窗口中***环境定义。这可能会有非预期的结果。

可以让某些按键连结在对话框中运作。这些按键连结会指派给***在对话框和窗口中***环境定义。「剪下」的按键连结就是这类按键连结的一个范例。可以变更这些按键连结。比方说，可以用 Ctrl+X 当成对话框的「剪下」功能，而以 Ctrl+W 作为窗口中的「剪下」功能。

平台和语言环境(Platform and Locale)

在不同的平台和语言环境下，按键连结也会不同。在 Macintosh 平台上，*Command+S* 是指派给*储存*，而不是常用的 *Ctrl+S*。在中文语言环境中 (zh)，*Alt+/* 是指派给*内容辅助*，而不是常用的 *Ctrl+空格键*。当 Eclipse 启动时，会决定现行的平台和语言环境，而且在 Eclipse 实例过程中并不会改变。

自订按键连结(Customizing Key Bindings)

在自订按键连结时，如果有多键的按键顺序、配置和环境定义，则会有许多事项须记住。为了简化，所有的按键自订都是在「按键」喜好设定页面中完成。

选取「Window」→「Preferences」→「Workbench」→「Keys」来进入「按键」喜好设定页面。

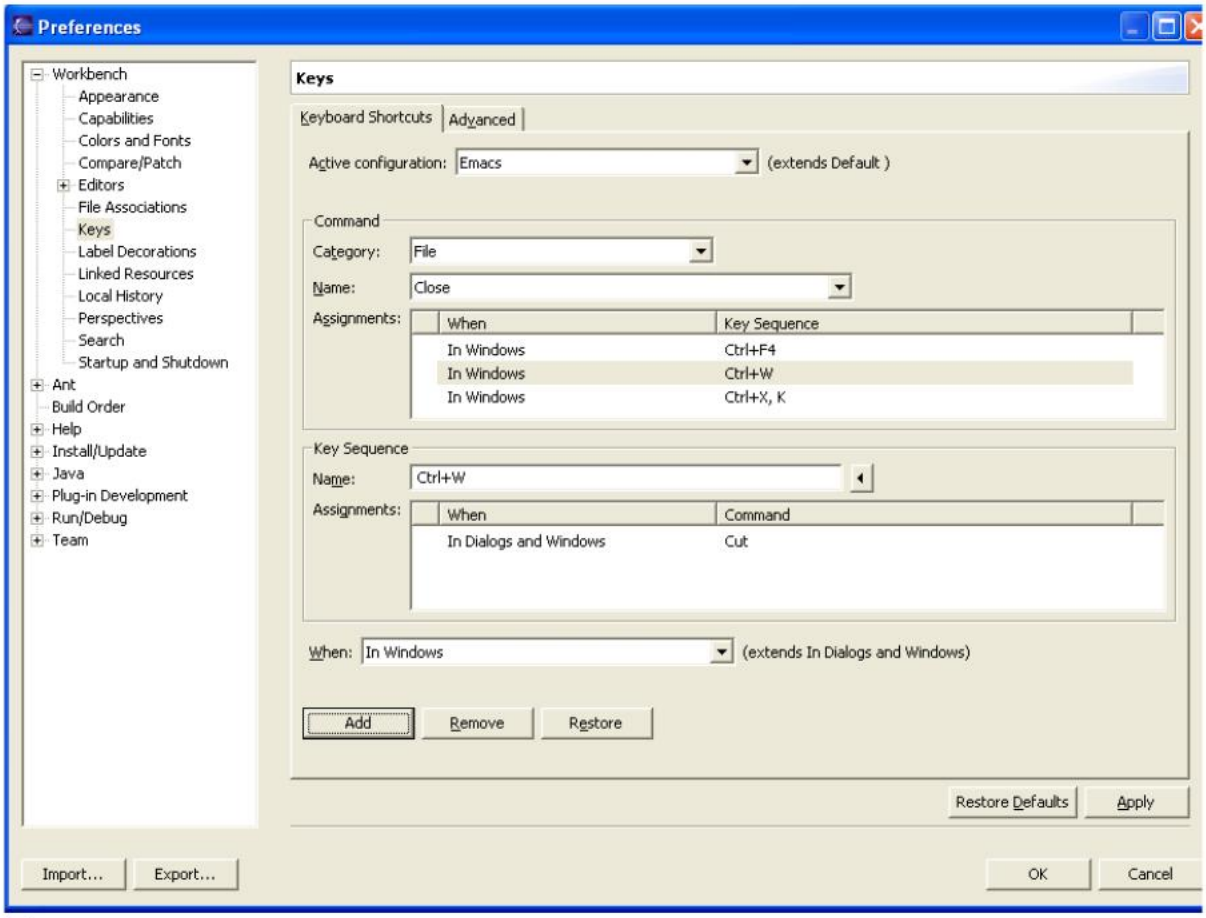


图 3.13

在这个范例中，我们已经选择 Emacs 作为作用中的配置，而且已经从指令清单中选择关闭指令。这个指令的信息以及指令的现行按键连结都会显示出来。

请注意，*关闭*已经指派了三个按键顺序：***预设***配置中的 *Ctrl+F4* 和 *Ctrl+W* 以及 ***Emacs*** 配置中的 *Ctrl+X K*。两者的指派环境定义都是***在窗口中***。因此，如果使用者将作用配置设为***预设***，就会将 *Ctrl+F4* 和 *Ctrl+W* 指派给*关闭*，而 *Ctrl+X K* 则不会。不过，如果使用者将作用中的配置设定为 ***Emacs***， *Ctrl+X K* 就会指派给*关闭*。同时，由于 ***Emacs*** 配置也会从***预设***配置借用按键连结，因此，*Ctrl+F4* 和 *Ctrl+W* 也会指派给*关闭*，不过，这些按键连结必须尚未指派给 ***Emacs*** 配置中的另一个指令。在这个范例中，"Ctrl+W" 连结于 Emacs 按键配置中的*剪下*。

以下是指派给*关闭*的按键顺序清单，还有一个地方可以新增或移除按键连结。依预设，它选取的环境定义是***在窗口中***。

输入按键顺序 *Ctrl+W*，就会启用「新增」按钮。同时，指定按键顺序 *Ctrl+W* 的所有指令的清单会显示在「新增」按钮下面。这时可以看到 *Ctrl+W* 目前是在***在窗口和对话框中***的环境定义中指派给*剪下*指令。按一下「新增」指令来将 *Ctrl+W* 指派给*关闭*。

现在可以看到 *Ctrl+W* 已经新增至指派给*关闭*的按键顺序清单中。请注意，小型的「变更」图形 Δ 表示这个按键连结会变更现有的按键连结。请确定*关闭*的 *Ctrl+W* 按键连结在 *Emacs* 按键配置中能够

运作。*剪下*的连结仍存在，但只能在对话框中运作（也就是说，我们是「在对话框和窗口中」，而不是「在窗口中」）。我们可以随时移除这个变更，方法是选取新的按键连结，然后按一下「移除」按钮。这时就会自动还原将 *Ctrl+W* 指派给*剪下*的指派。

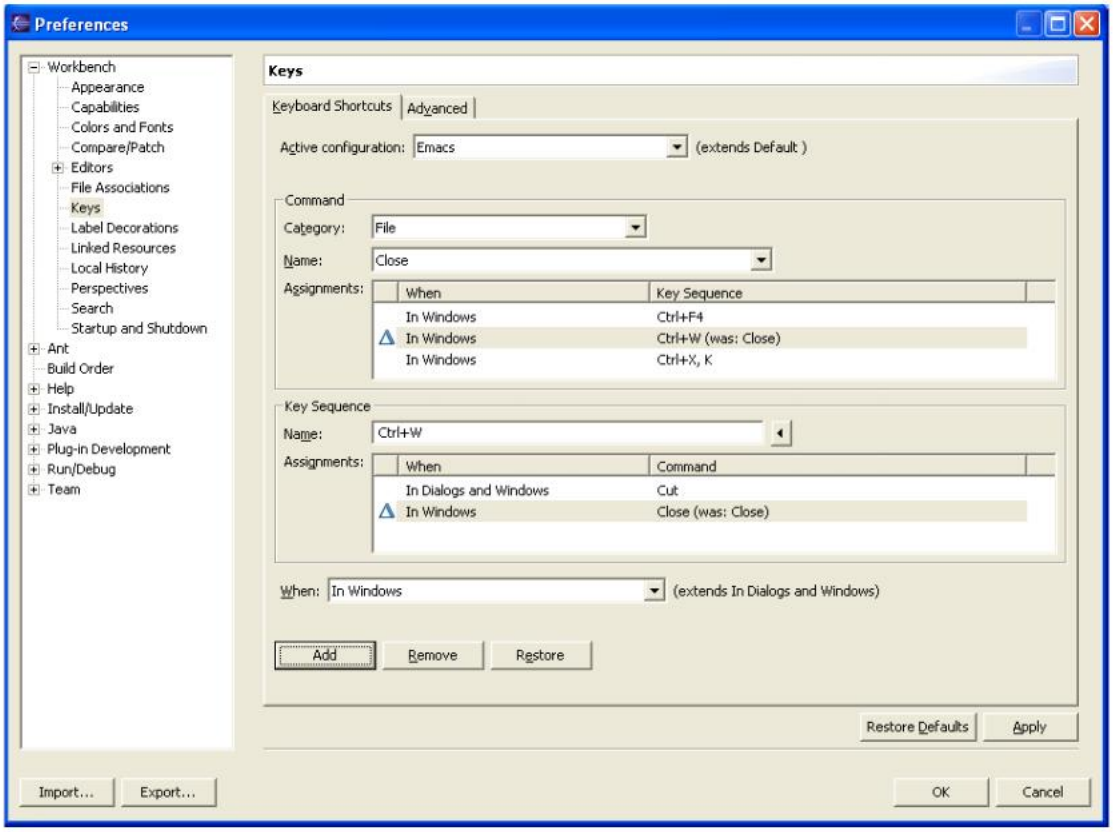


图 3.14

选取剪下指令，就可以看到变更的结果。请注意，这个图 ✖ 表示按键连结已经移除。我们可以随时还原这个按键连结，方法是在此处选取它，然后按一下「还原」按钮，这样就能有效移除在前一个步骤中新增的按键连结。

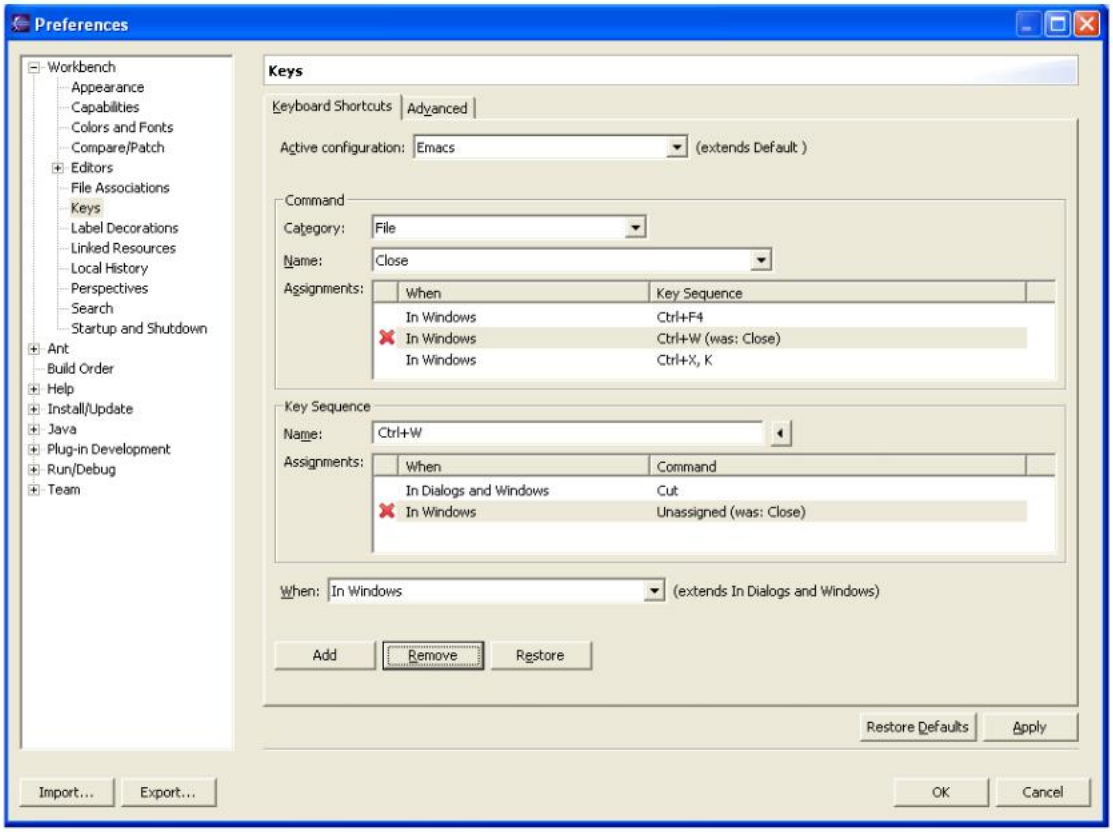


图 3.15

假设在 Emacs 配置中已经选择指定另一个按键给剪下（例如，Ctrl+Alt+W），以前一个方法来新增这个按键时，会产生下列结果。请注意，小型的「加号」图形 表示使用者已新增按键连结，而这个按键连结之前并未指定：

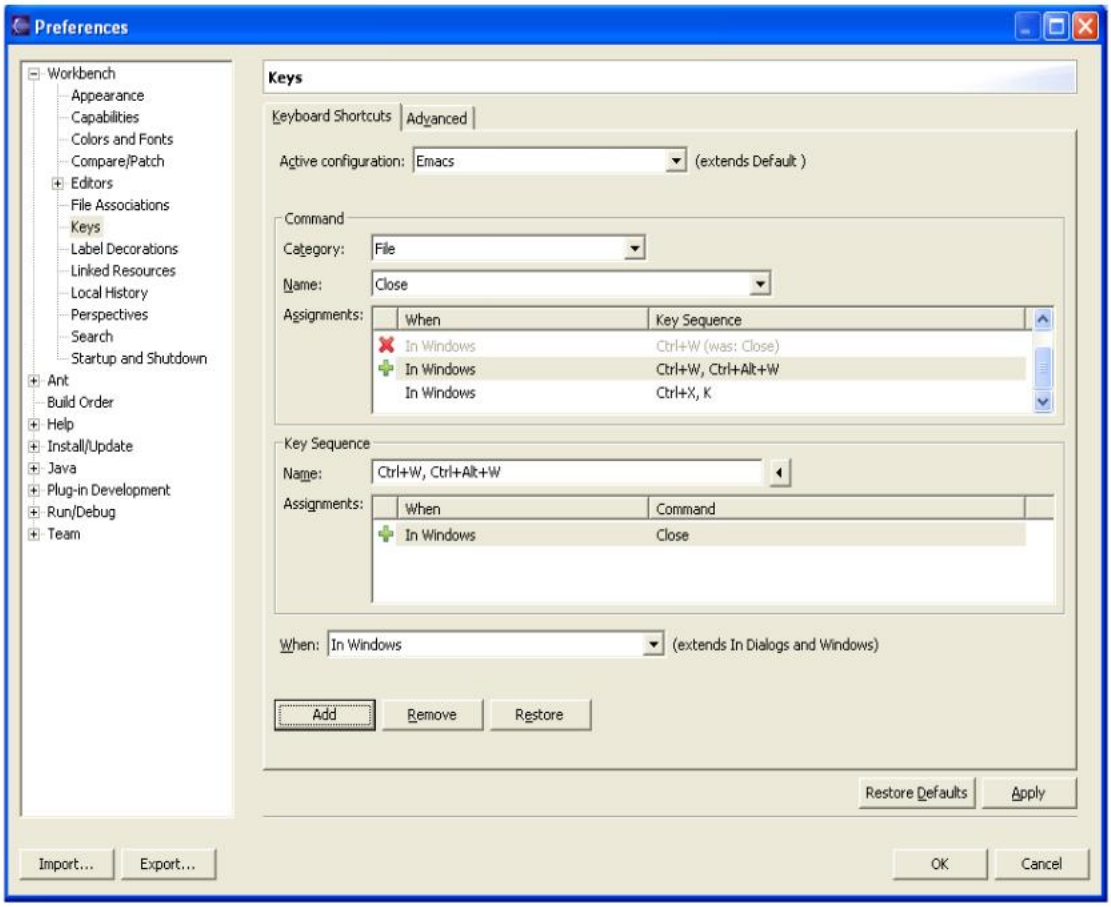


图 3.16

按键连结的动态本质(The Dynamic Nature of Key Bindings)

按键连结是由外挂程序提供，而在 Eclipse 中，可以新增或移除外挂程序。这样就能新增或移除由这些外挂程序所宣告的按键连结。Eclipse 在储存自订按键连结时，可以自动补偿这个问题。比方说，在上面的范例中，在 ***Emacs*** 配置中，*Ctrl+Alt+W* 是指派给*剪下*。假设使用者安装一个新的外挂程序，将 *Ctrl+Alt+W* 指定至特定指令。Eclipse 会将使用者的指派保留给*剪下*，但是会显示有小型「变更」图型的按键连结，而不会显示含有「加号」图型的按键连结。

冲突解决(Conflict Resolution)

只有少数简单、常用的按键作用可以指派给多个指令。许多配置、环境定义、平台和语言环境的所有分割键顺序在指派到网域中时，并没有彼此冲突。如果环境定义不存在，请考虑上述 *Ctrl+B* 的情况。有一个外挂程序将 *Ctrl+B* 指派给*建置*，则其它的外挂程序会将 *Ctrl+B* 指派给*将文字变为粗体字*。那么 Eclipse 将如何正确地解决这个冲突呢？

虽然可藉由上述的机制来大量减少冲突，但冲突仍然可能发生。两个相对独立的外挂程序可以将相同按键顺序指派给含相同环境定义、配置、平台和语言环境的不同指令。请考虑如果外挂程序于***在窗口中***环境定义中指派了 *Ctrl+F4*，且将***预设***配置指派给它的其中一个指令。这会与将 *Ctrl+F4* 指派给相同环境定义和配置中之关闭指令的 Eclipse 造成直接冲突。

这就是冲突。同时呼叫两个指令是不正确的，也不能只选择其中一个指令来接收按键作用。唯一能做的，就是忽略这两个按键连结，使 Ctrl+F4 在这个环境定义和配置中实际上没有用。

「按键」喜好设定页面显示这个本质的冲突。请注意红色的文字和 "[冲突]" 一字：

如果要解决这类冲突，使用者可以将按键顺序明确指派给其中一个指令。

另一类的冲突可能是因为按键顺序有多重按键作用。例如，在 ***Emacs*** 配置中，有许多多重按键作用的按键顺序是以 *Ctrl+X* 的按键作用作为开头。*Ctrl+H K* 是指派给*关闭*。*Ctrl+X H* 是指派给*全选*。

如同之前的说明，***Emacs*** 配置会从***标准***配置借用按键连结。在标准配置中，*Ctrl+X* 是指派给*剪下*。虽然 ***Emacs*** 配置没有明确重新定义 *Ctrl+X*，但是它的许多按键连结都需要按下 *Ctrl+X*。在 ***Emacs*** 配置中，按下 *Ctrl+X* 时，就等于要进入其中一个可能已经指定的按键顺序。但我们并不希望在这时候呼叫*剪下*动作。

对于这类冲突，其规则是忽略已指派给*剪下*的 *Ctrl+X*。否则，就无法完成 ***Emacs*** 配置中的许多按键连结。

3.1.8 标签装饰(Label Decorations)

「卷标装饰」可让其余信息显示在项目的卷标和图标中。

「标签装饰」喜好设定页面提供每一个装饰的说明，并可以选择要让哪些装饰看得到。

「标签装饰」喜好设定页面看起来如下：

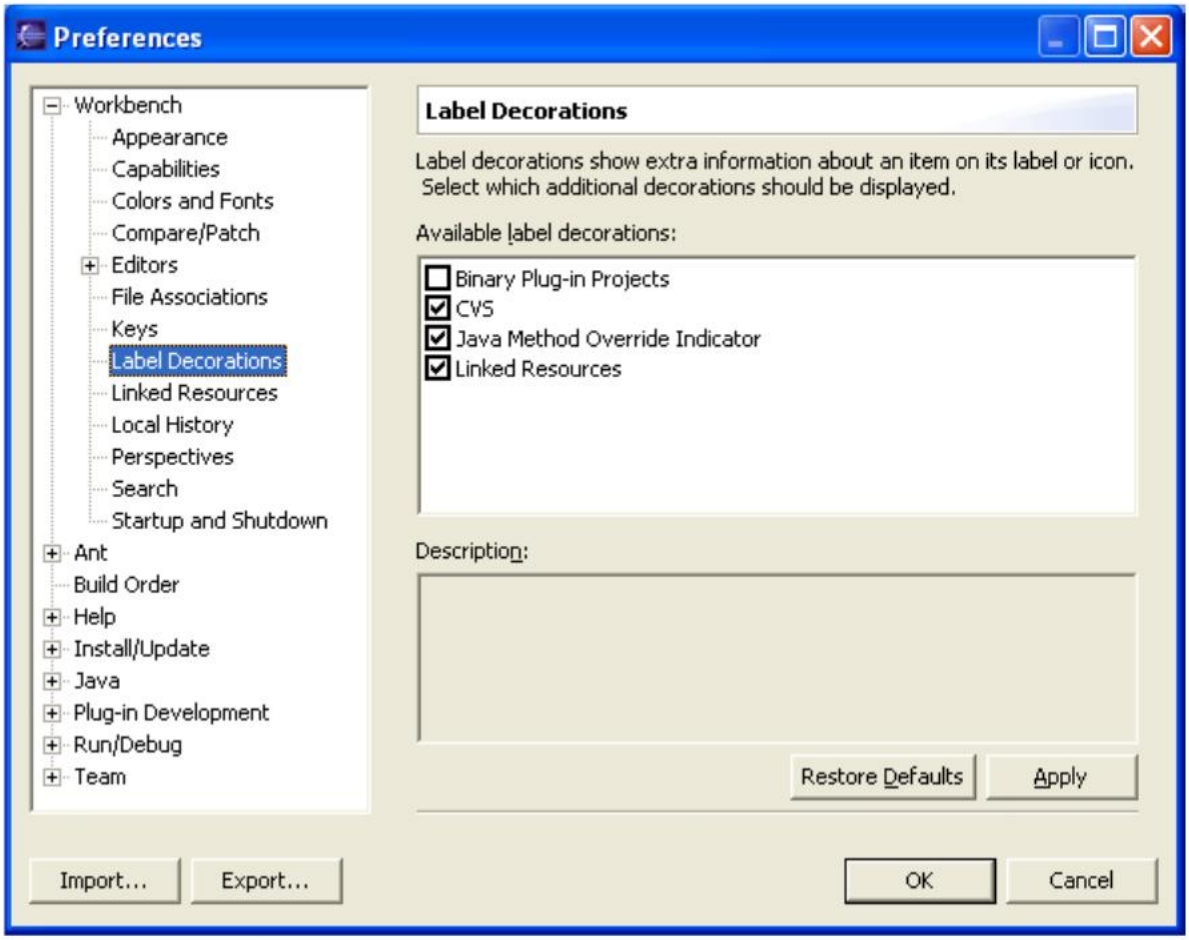


图 3.17

3.1.9 链接资源(Linked Resources)

在使用链接资源时，会使用这个喜好设定页面。Enable linked Resources 喜好设定是用来整体启用或停用整

个工作区的链接资源特性。依预设，链接资源是启用的。如果停用链接资源，就无法建立任何新的链接资源或汇入含有链接资源的现有项目。

并非所有的工作台版本都支持链接资源并且可将它们识别为链接信息。如果打算与其它使用者共享工作区数据，可能不要使用链接资源。如果其它使用者无法使用链接资源，请停用这个喜好设定。

这个页面的其它部分是用来定义在建立链接资源时所使用的路径变量。请使用 New 按钮来定义新的变量，也可以使用 Edit 按钮来变更现有变量的值，或者使用 Remove 按钮来移除现有的变量。请注意，如果变更的路径变量正在使用中，就需要对这些项目执行本端重新整理，以 "探索" 档案系统中是否有任何不同之处。可以开启该项资源的「导览器」快速菜单，然后选取 Refresh，来重新整理资源。建议不要移除目前正在使用的路径变量。

「链接资源」喜好设定页面的外观如下：

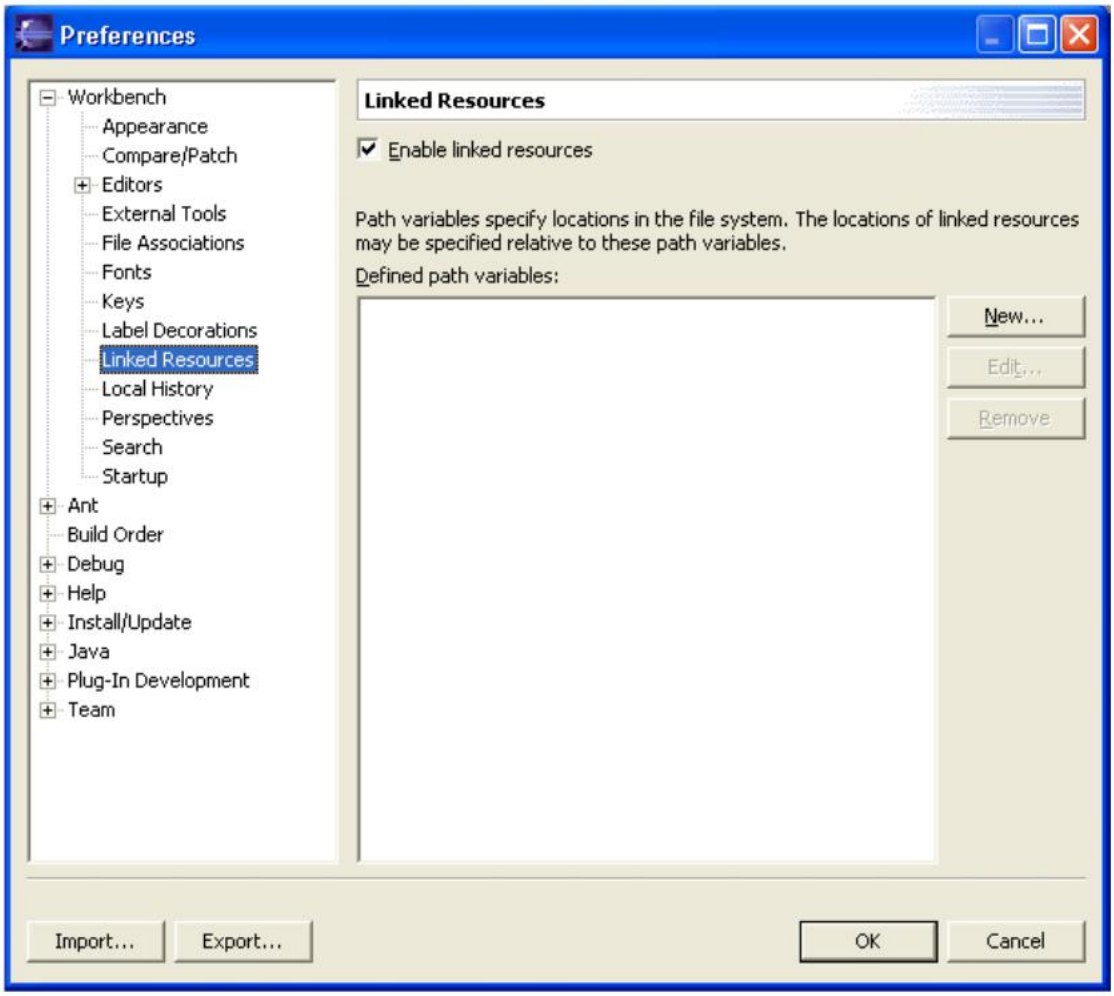


图 3.18

3.1.10 历史纪录(Local History)

可以在「历史纪录」页面中变更下列喜好设定。

选项	说明	默认值
Days To Keep Files(档	指出要在历史纪录中维护变更多少天。这个值以外的历程状态将会	7 天

选项	说明	默认值
案的天数)	流失。	
Entries Per File(档案的项目数)	指出在历史纪录中每个档案要维护多少历程状态。如果超过这个值，将会失去较旧的历程，以挪出空间供新历程使用。	50 个项目
Maximum File Size(MB)(最大档案大小(MB))	指出历程储存库中的个别状态的大小上限。如果档案超过这个大小，它将不会被储存。	1 MB

「历史纪录」喜好设定页面看起来如下：

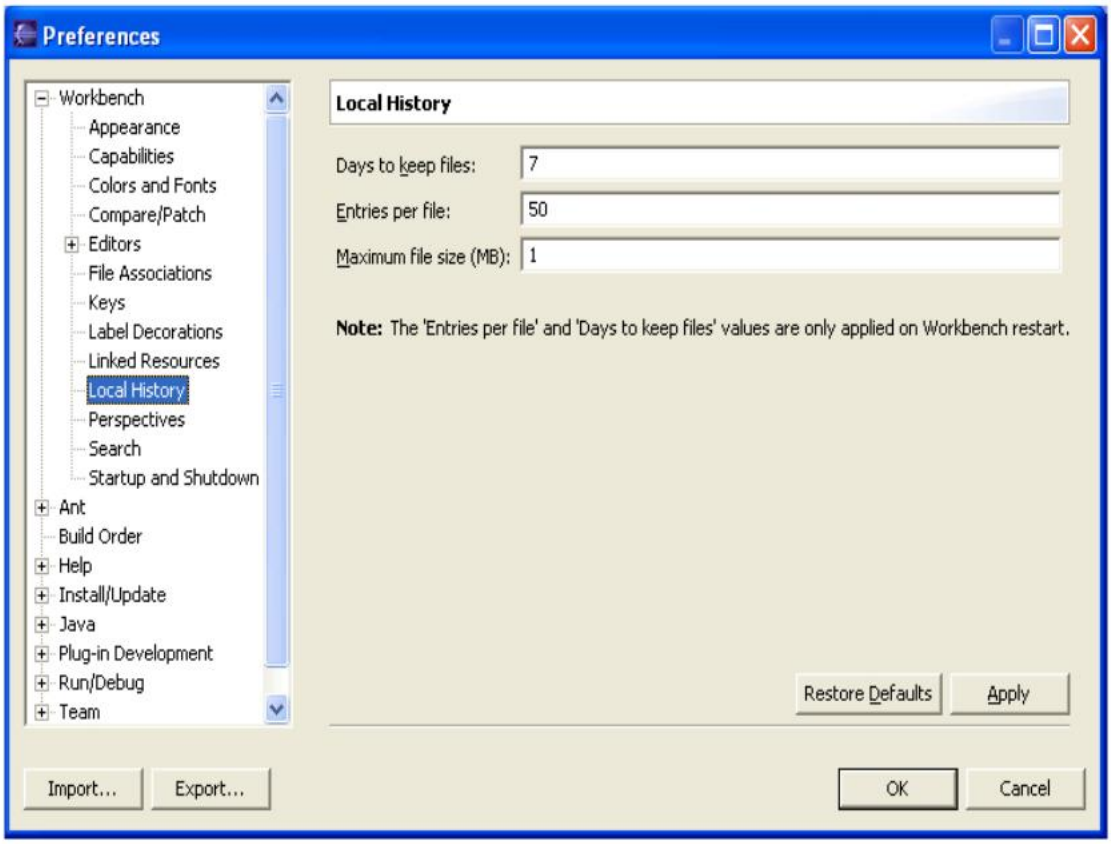


图 3.19

3.1.11 视景

在「视景」喜好设定页面上，可以管理各种定义于工作台的视景。

选项	说明	默认值
Open a new perspective(开启新视景)	请使用这个选项来设定开启新视景时会发生的情况。要在现行工作台窗口内或是新窗口内开启视景？	在相同窗口中
Open a new view(开启新视图)	请使用这个选项来指定新的视图开启时会发生什么情况。它会在其位于现行视景内的预设位置上开启，或是开启为快速视图并定置到现行视景的侧边。	在视景内

选项	说明	默认值
New project options(新项目选项)	请使用这个选项来指定建立新项目时的视景行为。可以将它设定为：将现行视景切换至与项目类型相关联的视景，并在同一个工作台窗口中开启视景以作为现行视景、切换视景并在新的工作台窗口中加以开启，或是完全不切换视景。	在同一窗口开启视景

可用的视景选项：

选项	说明	默认值
Make Default(设为默认值)	将所选取的视景设为预设视景。	资源
Reset(重设)	将选取视景的定义重设成预设配置。这个选项仅适用于已经使用「窗口」→「另存新视景...」来改写的内建视景。	n/a
Delete(删除)	删除所选取的视景。这个选项仅适用于使用者定义的视景（无法删除内建视景）。	n/a

「视景」喜好设定页面看起来如下：

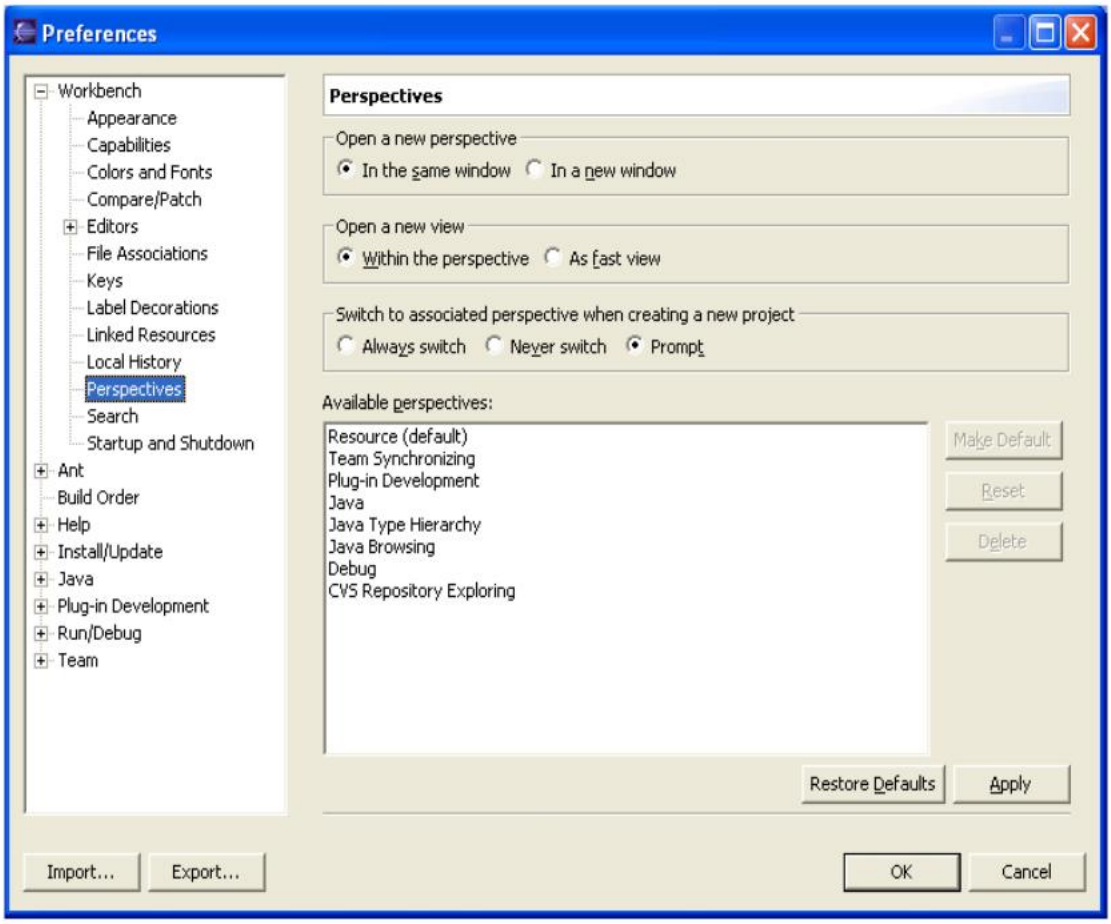


图 3.20

3.1.12 搜寻(Search)

「搜寻」喜好设定页面可让使用者设定搜寻的喜好设定。Reuse editors to show matches(重复使用编辑器来显示相符项目)选项可让使用者继续使用同一个编辑器来搜寻结果，以减少开启的编辑器数目。Emphasize inexact matches(强调不完全相符)的项目是一个选项，可以在「搜寻」视图中强调显示这些项目。如果搜寻引擎不完全确定相符项目，则该相符项目会被视为不精确。也可以设定 Foreground color for inexact matches (不完全相符项目的前景颜色)。如果只要察看完全相符的项目，请勾选 Ignore inexact matches (忽略不完全相符的项目)。Default perspective for the Search view(视图的预设视景)可以定义有新搜寻结果时要将哪一个视景移至最前面。

「搜寻」喜好设定页面看起来如下：

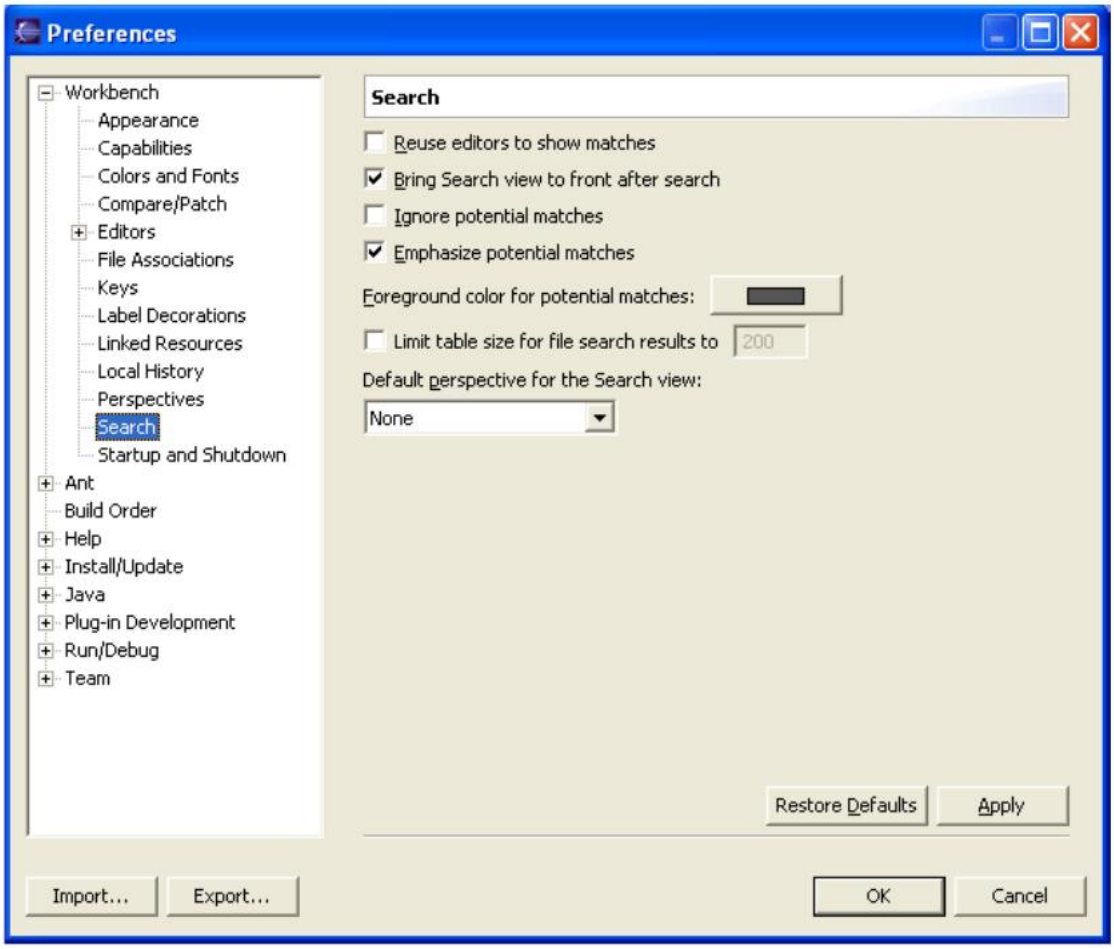


图 3.21

3.1.13 启动和关闭(Startup and Shutdown)

「启动和关闭」喜好设定页面可以选择要在工作台启动期间自动启动的外挂程序。

一般而言，要到需要外挂程序时才会加以启动。不过，某些外挂程序可能会指定它们希望在启动期间被启动。这个喜好设定页面可以选择在启动期间将会实际启动这其中的哪些外挂程序。

选项	说明	默认值
Prompt for workspace	如果开启这个选项，每次启动工作台时，工作台都会提示使用者要	开启

选项	说明	默认值
on startup(启动时发出工作区提示)	用哪个工作区。	
Refresh workspace on startup(启动时重新整理工作区)	如果开启这个选项，在启动时，工作台会与档案系统同步化它的内容。	关闭
Confirm exit when closing last window(关闭最后一个窗口时确认结束)	如果开启这个选项，当关闭最后一个窗口时，工作台都会问使用者是否要结束。	开启

「启动和关闭」喜好设定页面看起来如下：

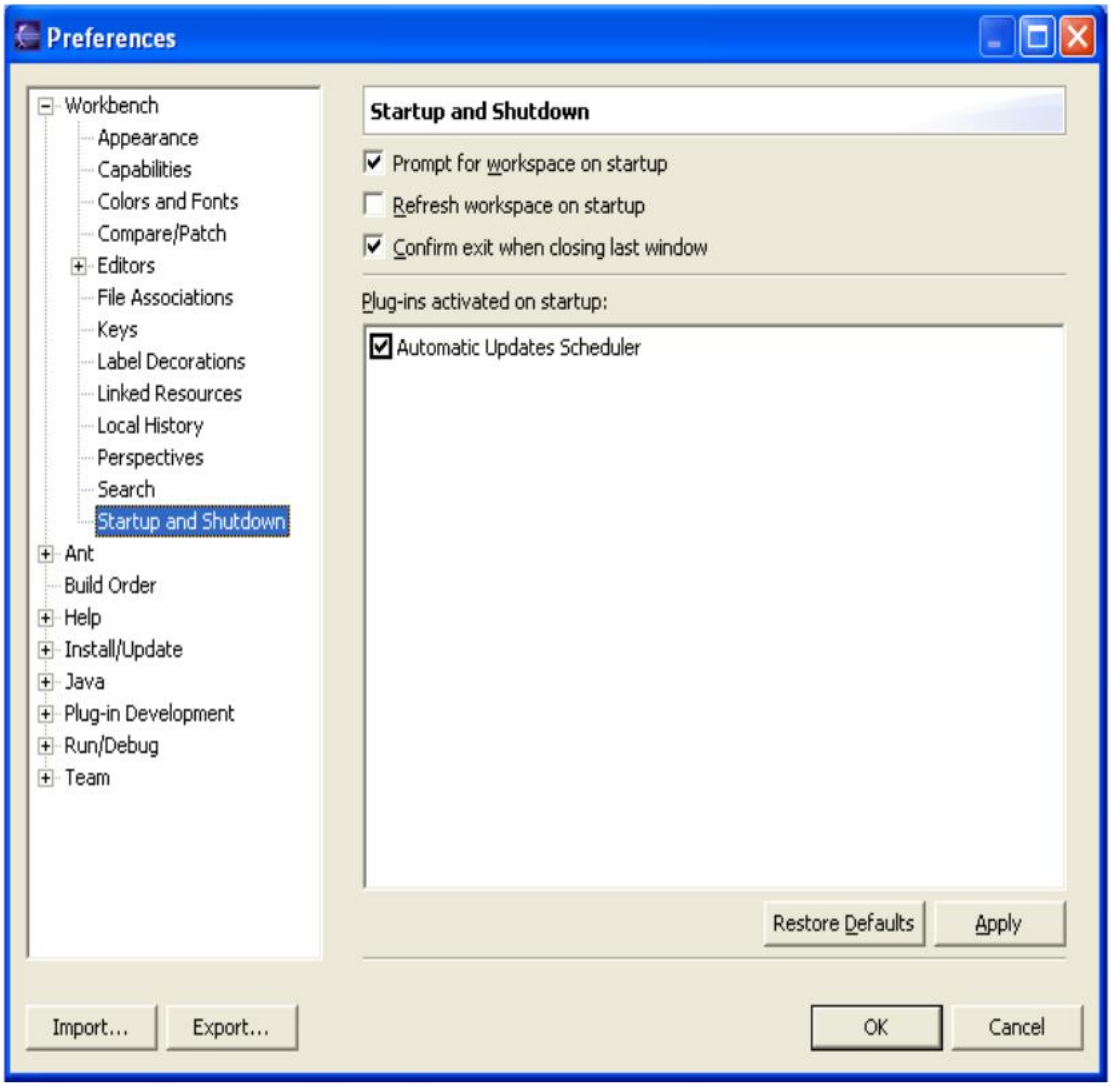


图 3.22

3.2 Ant

可以在 Ant 页面中变更下列喜好设定。

可以配置 Ant 的 "-find" 模拟要搜寻的建置档。

也可以配置 Ant 建置输出的颜色。

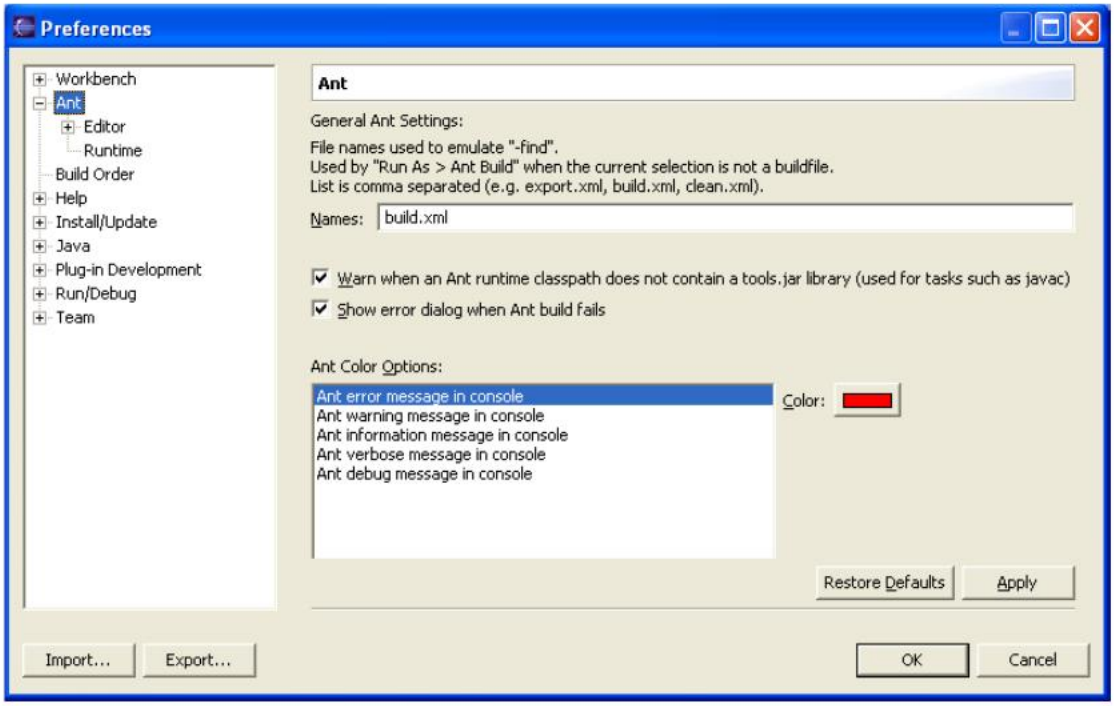


图 3.23

3.2.1 Ant 编辑器(Ant Editor)

可以在「Ant 编辑器」页面中变更下列喜好设定。

外观选项(Appearance Options)

选项	说明	默认值
Print margin column(打印边距直栏)	这个选项可以设定 Ant 编辑器的打印边距。	80
Displayed tab width(显示的栏标宽度)	这个选项用来控制在 Ant 编辑器中，要用多少空格来显示栏标。	4
Insert spaces for tabs when typing(在输入时插入栏标空格)	这个选项用来控制在 Ant 编辑器中输入时，是否要用空格来取代栏标。	关闭
Show overview ruler(显示概观标尺)	这个选项用来控制 Ant 编辑器右侧是否要显示概观标尺。	开启
Show line numbers(显示行号)	这个选项用来控制 Ant 编辑器左侧是否要显示行号。	关闭
Highlight current line(高亮度显示现行行)	这个选项用来控制是否要强调显示现行行。	开启
Show print margin(显示	这个选项控制是否可以看到打印边距。	关闭

选项	说明	默认值
打印边距)		
Appearance color options(外观颜色选项)	这个选项控制各种外观颜色。	

在外观页面中，提供 Ant 编辑器的外观选项。

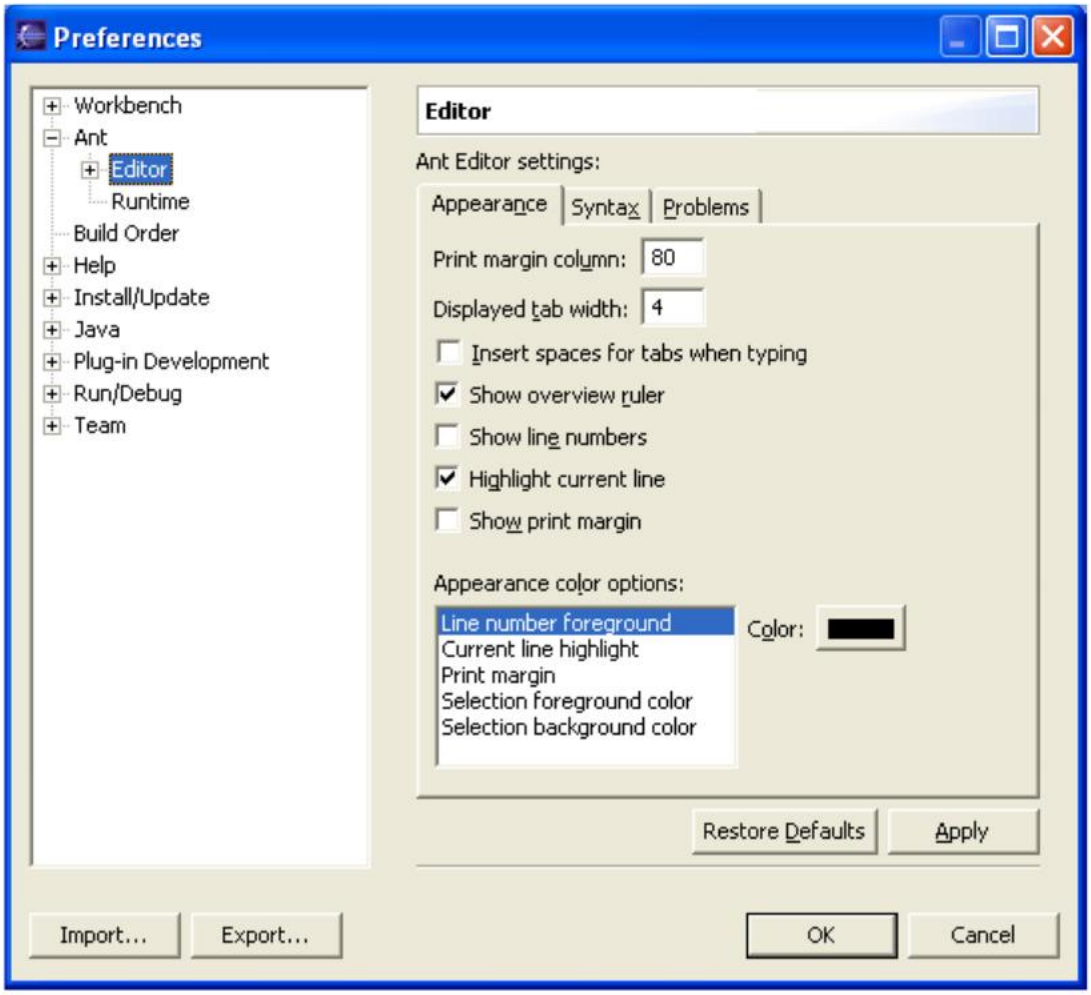


图 3.24

3.2.2 Ant 执行时期(Ant Runtime)

在类别路径页面上，可以将定义作业和类型的其它类别新增至 Ant 类别路径。

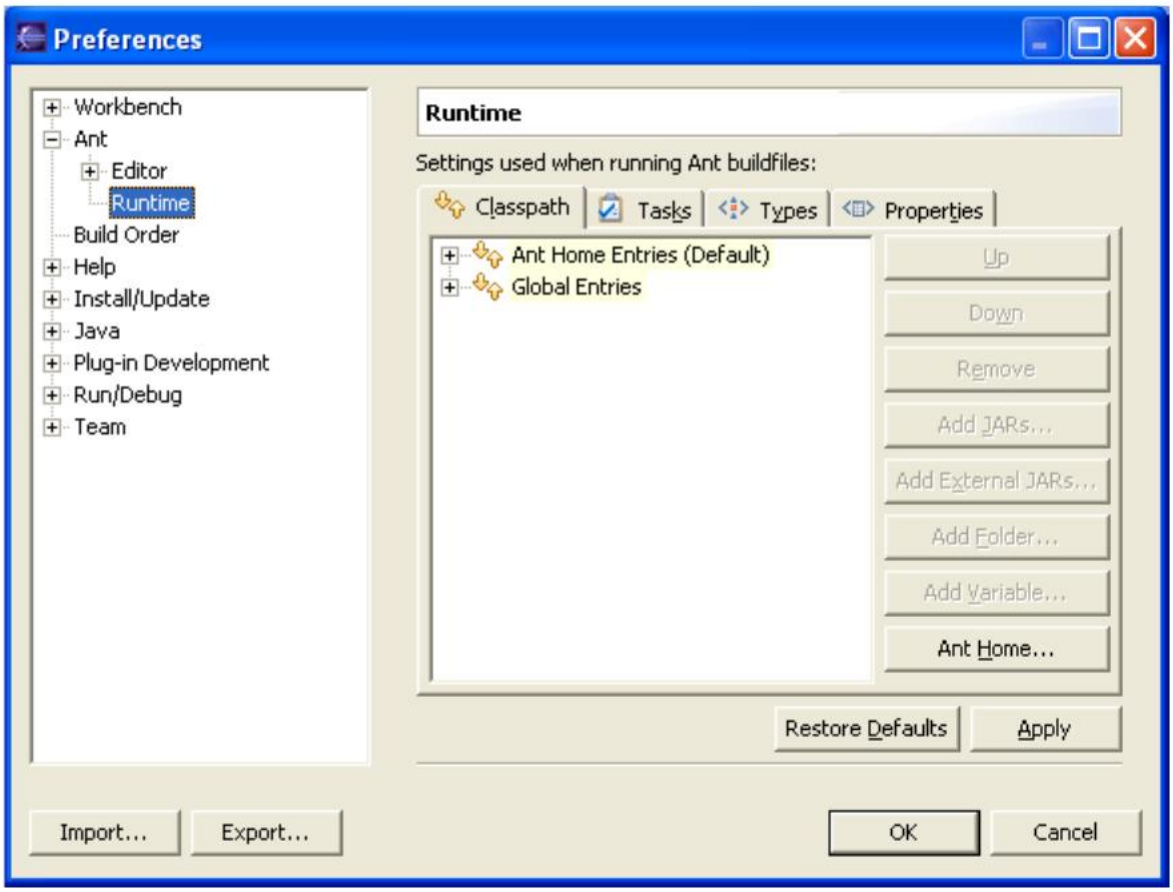


图 3.25

在作业页面上，可以新增定义于类别路径上的其中一个类别的作业。

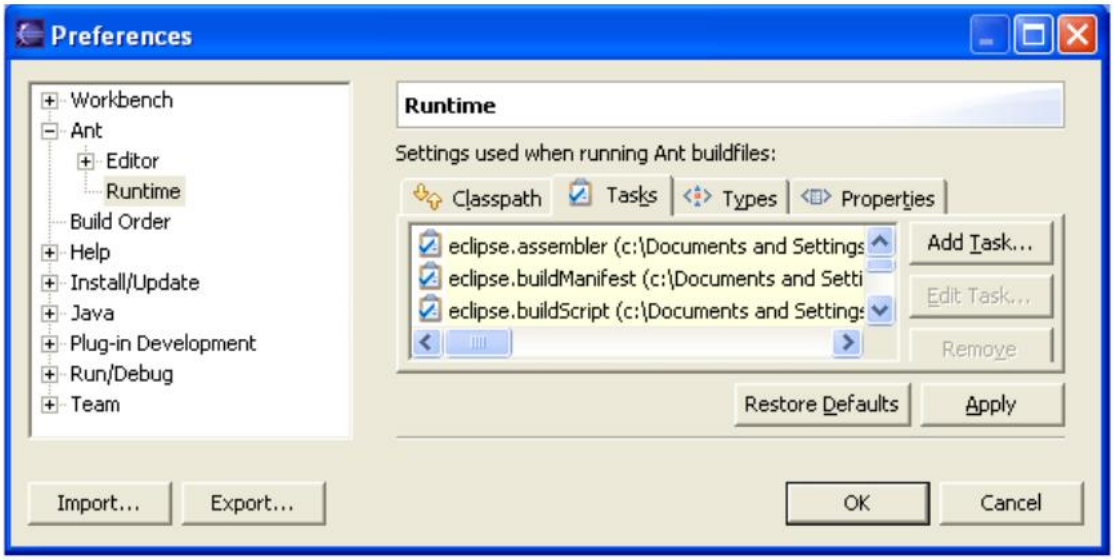


图 3.26

在类型页面上，可以新增定义于类别路径上的其中一个类别的类型。

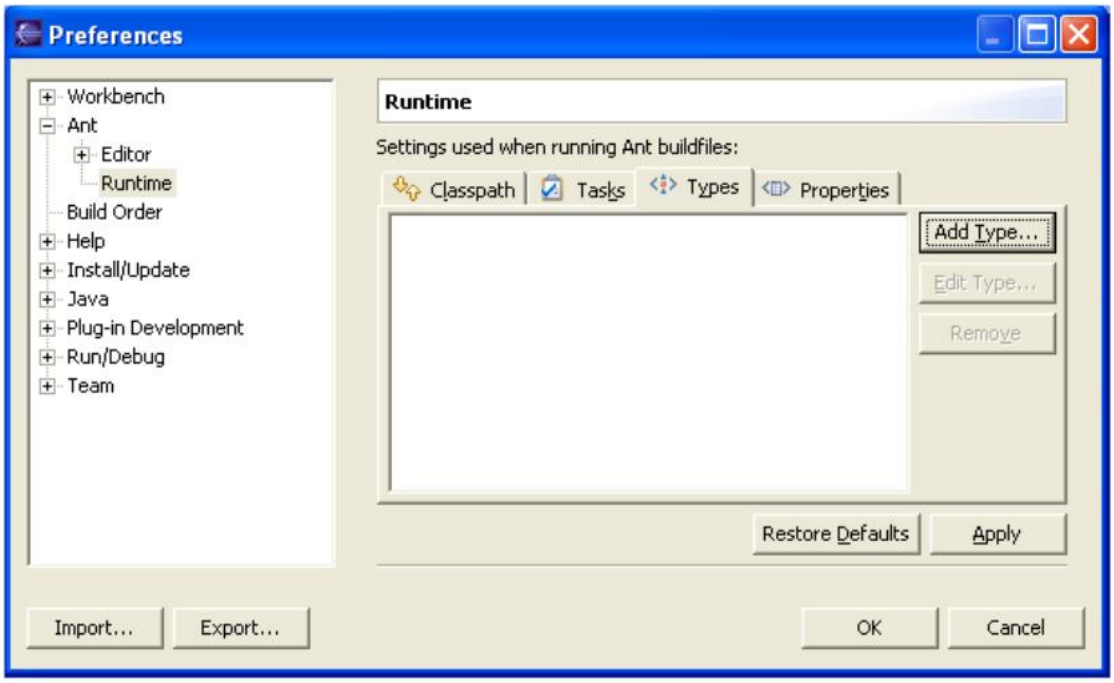


图 3.27

在内容页面中，可以新增要传送到 Ant 的内容和内容档。

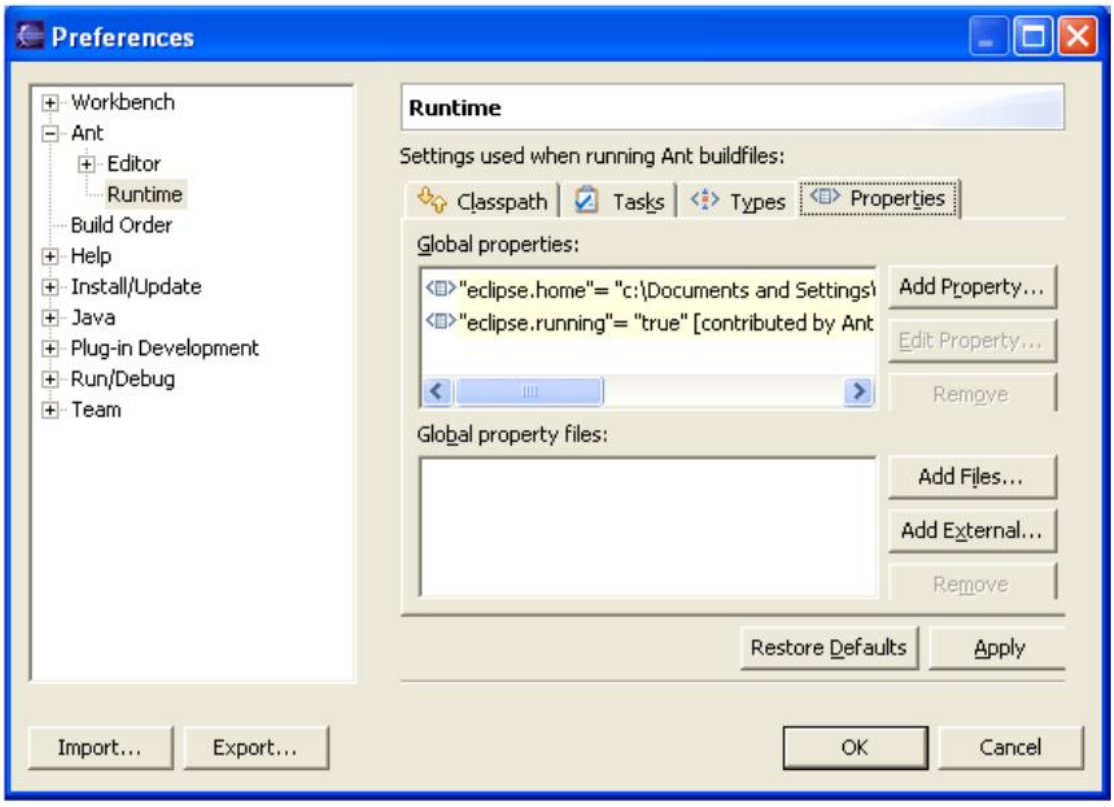


图 3.28

3.3 建置次序(Build Order)

通常，项目建置的次序是很重要的。比方说，如果某个项目需要在另一个项目中定义的 Java 类别，则必

须先建置第一个项目的必备类别，然后再建置第一个项目。工作台可以让使用者明确地定义建置项目的次序。此外，使用者可以让平台藉由将项目参照解译为必备关系，来计算建置次序。在建置整个工作区或项目群组上，都会套用建置次序。

可以在「建置次序」喜好设定页面中变更这个次序。一开始，*使用预设建置器次序*选项是开启的，在此情况下，平台会计算建置次序。关闭这个选项时，就可以存取项目清单，而且可以操作项目的次序。请选取项目并使用上和下按钮来变更建置次序。可以使用 Add Projec 及 Remove Project 按钮，在建置次序中新增和移除专案。从建置次序中移除的项目*将*会被建置，但是会在建置次序中的所有项目都已建置完成之后才会建置。

在这个页面底端，有一个喜好设定可用来处理包含循环的建置次序。在理想的状况下，应该避免在项目之间使用循环参照。包含循环的项目在逻辑上仍属于单一项目，所以它们会尽可能收合成为一个项目。然而，如果一定要有循环，建置次序可能需要好几个迭代，才能正确地建置每一个项目。变更这个喜好设定时，会改变工作台尝试在建置次序上进行迭代多少次之后，才会放弃。

「建置次序」喜好设定页面看起来如下：

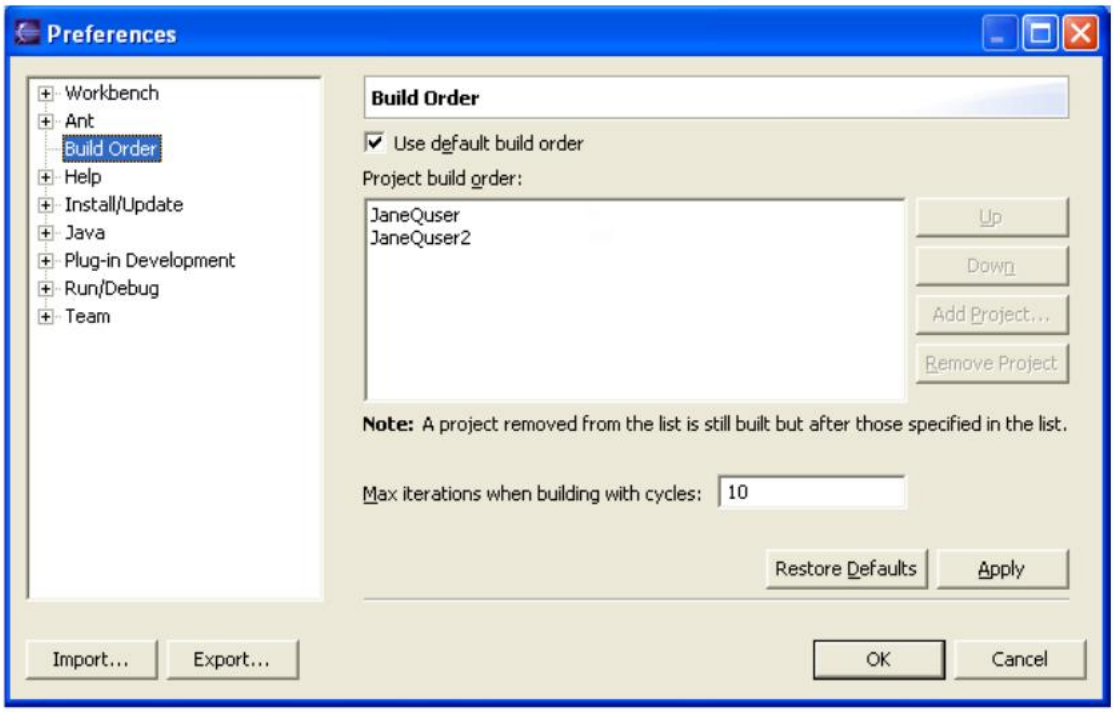


图 3.29

3.4 说明(Help)

在「说明」喜好设定页面上，可以指出如何显示说明书籍。

选项	说明	默认值
Always use external browsers(固定使用外部浏览器)	如果的系统支持内嵌 Web 浏览器，可能的话，说明会利用内嵌说明浏览器来显示说明，但仍可以使用这个选项。请选取它来强迫说明使用清单中的外部浏览器。	关闭

Current web browser adapter(现行 Web 浏览器配接器)	说明系统利用 Web 浏览器配接器，在外部浏览器中显示在线说明。如果有多个配接器可以在的系统中开启浏览器，就会使用选取的配接器来显示说明。	视操作系统而定
Custom browser command(自订浏览器指令)	如果从浏览器配接器清单中选取「自订浏览器」，就会使用这个字段来指定要启动浏览器程序的指令。如果 URL 不是浏览器程序可以接受的最后一个参数，请使用 "%1" 字符串来指出 URL 在指令中的位置。	"C:\Program Files\Internet Explorer\IEXPLORE.EXE" %1 - Windows，mozilla %1 - 其它平台（会显示这个字段）

「说明」喜好设定页面看起来如下：

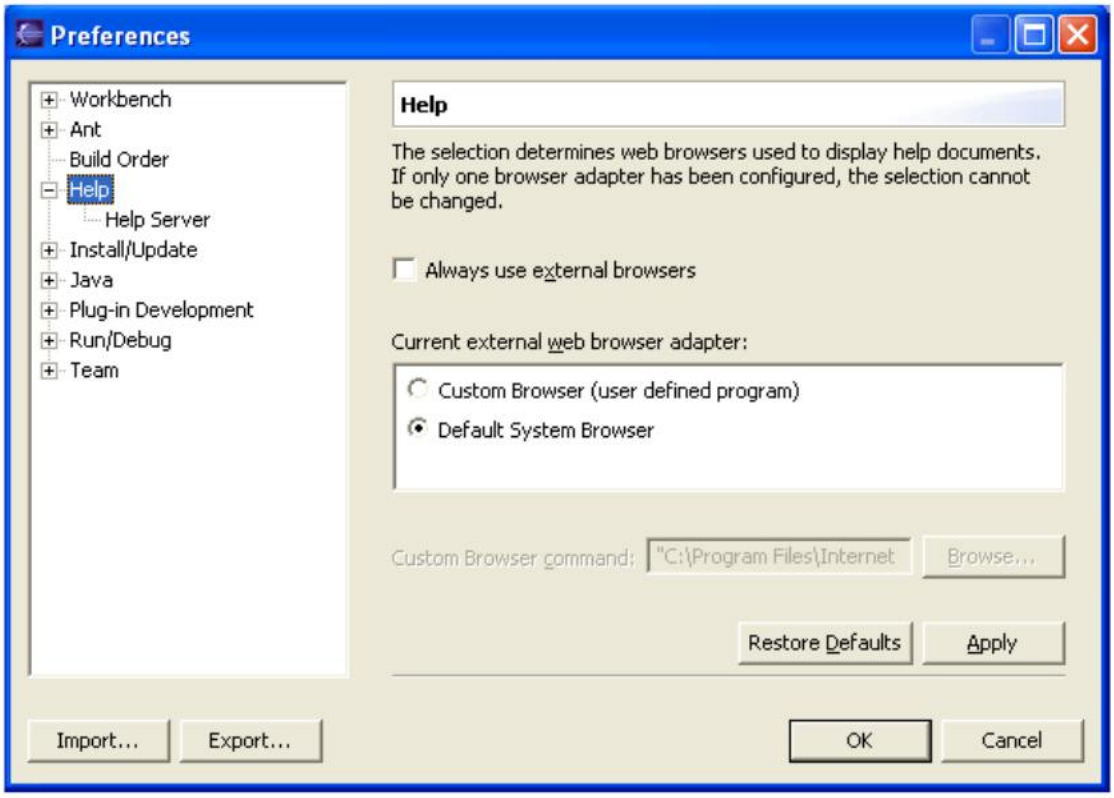


图 3.30

附注：在这个页面中所执行的选项会影响说明视图的呈现方式。如果选取的浏览器没有和 Internet Explorer 或 Mozilla 完全兼容，或者已经停用 JavaScript，则浏览器中所显示的说明视图可能是简化的版本。

3.4.1 说明服务器(Help Server)

说明系统包含一个内部服务器，可提供说明内容给浏览器。可以使用这个喜好设定页面来变更服务器所使用的接口和端口。只有当遇到问题且无法使用预设的喜好设定来检视说明时，才应该变更这些设定。

选项	说明	默认值

Host(主机)	服务器所使用的本端 IP 接口的名称或地址。	空白
Port (埠)	服务器要接听的 IP 埠。如果将这个值指定成 0，就会由操作系统来指定这个端口。	0

「说明服务器」喜好设定页面的外观如下：

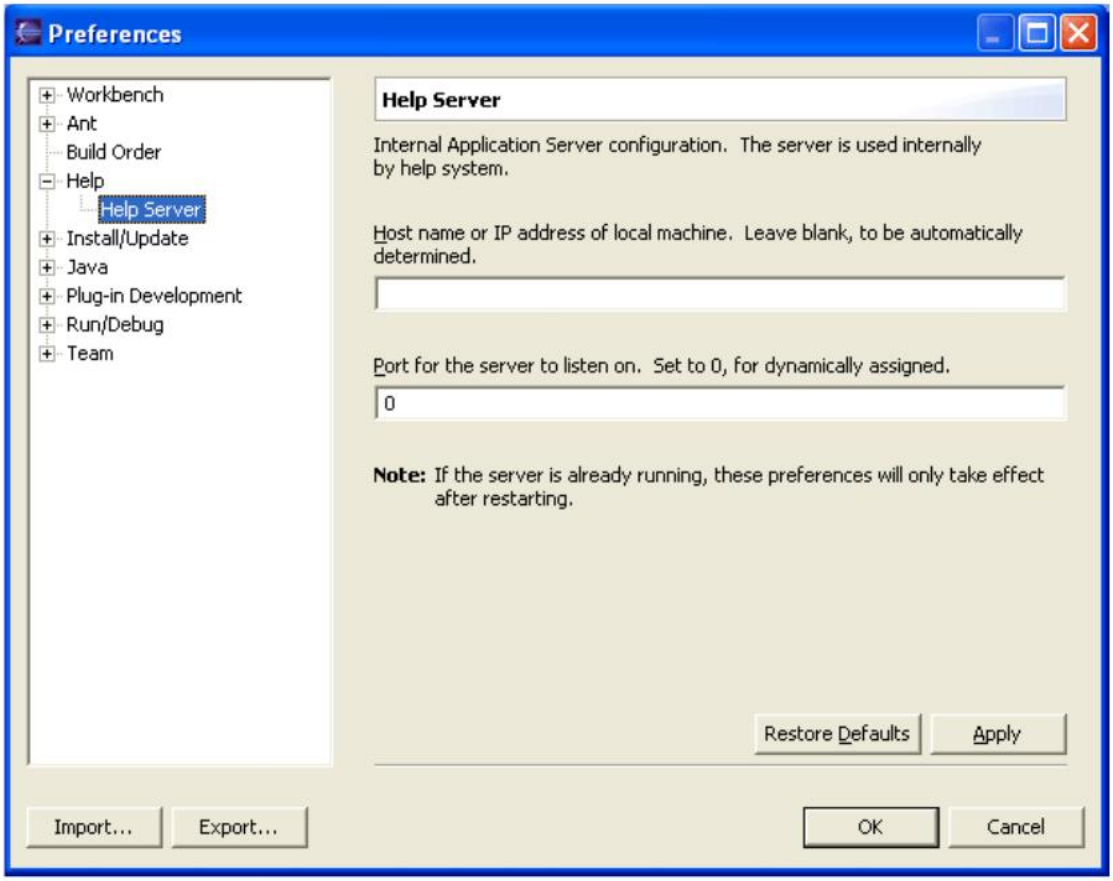


图 3.31

3.5 自动更新(Install/Update)

可以在「自动更新」页面中变更下列喜好设定：

选项	说明	默认值
Automatically search for updates and notify me (自动搜寻更新项目并通知我)	当选取时，更新管理程序会依照更新时程表所定义来自动搜寻更新项目	未选取
Update Schedule (更新时程表)	每次启动时寻找更新项目，或在每天或每周某天的某个预定时间寻找更新项目。	在启动时

3.6 Java

这个页面可以指出在一般 Java 设定方面的喜好设定。

选项	说明	默认值
Update Java views(更新 Java 视图)	**只在储存时(On save only)：** 在储存编译单元前，不会更新「概要」视图以外之所有视图中显示的 Java 元素。视图会反映工作区的现行状态，但不会顾及到工作副本。 **编辑时(While editing)：** 所有视图中显示的 Java 元素恒会反映工作区的现行状态，包括工作副本。	编辑时
Action on double click in the Package Explorer(在「套件浏览器」中按两下后的动作)	**进入所选元素(Go into the selected element)：** 当按两下储存器时，就会执行 Go Into 指令。 请从导览菜单查看 Go Into。 **展开所选元素(Expand the selected element)：** 当按两下储存器时，会将之展开，并显示其子项。	展开所选取的元素
When opening a Type Hierarchy(当开启类型阶层时)	**开启新的「类型阶层」视景(Open a new Type Hierarchy Perspective)** 只要一开启「类型阶层」视图，即开启新的「类型阶层」视景。 **在现行视景中显示「类型阶层」视图(Show the Type Hierarchy View in the current perspective)** 在现行视景中显示「类型阶层」视图。 附注：在「工作台」喜好设定页面中，可以选择要在新窗口或现行窗口中开启新视景，或者以新视景取代现有的视景。	在现行视景中显示「类型阶层」视图

3.6.1 外观(Appearance)

在这个喜好设定页面上，可以配置视图中之 Java 元素的外观。选项如下：

选项	说明	默认值
Show method return types(显示方法传回类型)	当启用时，视图中的方法会显示传回类型。	关闭
Show override indicators in outline and hierarchy(以概要及阶	当启用时，则在「概要」与「类型阶层」视图中，会针对已置换与实作的方法显示一个指示器。	开启

选项	说明	默认值
层显示置换指示器)		
Show members in Package Explorer(在套件浏览器中显示成员)	当启用时，亦会显示 Java 文件与类别文件层次下的 Java 元素。	开启
Compress package name segments(压缩套件名称区段)	当启用时，则会根据压缩型样压缩套件名称。	关闭
Stack views vertically in the Java Browsing perspective(垂直堆栈「Java 浏览」视景中的视图)	当启用时，则「Java 浏览」视景中的视图会采垂直（而非水平）方式堆栈。	关闭

3.6.2 类别路径变量(Classpath variables)

可配置的变量(Configurable variables)

可在「Java 建置路径」中使用类别路径变量，以免参照本端档案系统。当使用变量项目时，类别路径中只含有一个变量，且建置路径可供整个团队共享。必须在这个页面中配置变量的值。

指令	说明
New...(新建...)	新增变量项目。在产生的对话框中，指定新变量的名称和路径。可以按一下 File 或 Folder 按钮来浏览以找出路径。
Edit...(编辑...)	可让编辑所选取的变量项目。在产生的对话框中，编辑该变量的名称与（或）路径。可以按一下 File 或 Folder 按钮来浏览以找出路径。
Remove(移除)	移除选取的变量项目。

保留类别路径变量(Reserved class path variables)

某些类别路径变量会设定在内部，且无法在「类别路径变量」喜好设定中变更：

- JRE_LIB：这个保存档中含有目前所用之 JRE 的执行时期 JAR 檔。
- JRE_SRC：为目前所用 JRE 的程序文件保存文件。
- JRE_SRCROOT：为目前所用 JRE 之程序文件保存文件中的根路径。

3.6.3 程序代码格式制作器(Code Formatter)

这个页面中的预览窗格会示范这些选项对编辑器中之 Java 程序代码所产生的结果。

选项	说明	默认值
Insert a new line before an opening brace(在左大括号前插入新行)	编辑器在新的左大括号前插入换行符号。换句话说，左大括号恒起自新行的开头。	关闭
Insert new lines in control statements(在控制陈述式中插入新行)	编辑器在新的控制陈述式前插入一行。换句话说，控制陈述式（如 if、else、catch、finally 等）恒起自新行的开头。	关闭
Clear all blank lines(清除所有空白行)	编辑器删除档案中的所有空白行。	关闭
Insert new line between 'else if'(在 'else if'之间插入新行)	编辑器在 else-if 陈述式中之 "else" 与 "if" 字之间插入一行新行。	关闭
Insert a new line inside an empty block(在空区块内插入新行)	编辑器在空大括号间插入一个换行。换句话说，在一个空大括号组中的左右大括号会出现在不同行。一个例外情况是，如果右大括号后跟着一个关键词，则两个大括号会出现在同一行。	开启
Maximum line length(行长度上限)	这是任何单行的最大长度。当字行超过这个长度时，则会折行。如果输入 0，则完全停用折行特性。	80
Compact assignment(精简指派)	编辑器会移除变数与指派陈述式间的任何空格，使其不对称（如 a=b;）。	关闭
Insert a space after a cast(在强制转型后插入空格)	编辑器会在强制转型与下列表示式之间插入一个空格。	开启
Insert tabs for indentation, not spaces(插入 tab（而非空格）以内缩)	编辑器使用 tab（而非空格）来呈现内缩。	开启
Number of spaces representing an indentation level(代	如果编辑器是使用空格而非 tab 来呈现内缩，则这是指出一个内缩是由多少空格组成。	4

选项	说明	默认值
表内缩层次的空格数目)		
Preview pane(「预览」窗格)	使用这个页面目前所示的设定值，以范例显示 Java 程序代码的模样。	n/a

3.6.4 程序代码产生(Code generation)

程序代码产生喜好设定分成两个区段：

- 名称
- 程序代码和批注

名称(Names)

这个页面定义字段（Static 和非 Static）、参数和区域变量的命名惯例。对于每一个变量类型，有可能配置前缀或字尾清单，或两者的清单。

产生 Getter 和 Setter 动作，以及所有建立字段、参数和区域变量的动作和「快速修正」提议，都会使用命名惯例。

动作	说明
Edit...(编辑...)	开启一个对话框，编辑目前选取之变量类型的前缀和字尾清单

程序代码和批注(Code and Comments)

程序代码和批注页面含有产生程序代码之动作所使用的程序代码模板。模板含有当套用模板时将替代的变量。某些变量可用在所有模板中，某些变量则是模板特有的。

动作	说明
Edit...(编辑...)	开启一个对话框，编辑目前选取的程序代码范本。
Import...(汇入...)	从档案系统汇入程序代码模板。
Export...(汇出...)	汇出所有选取的程序代码模板至档案系统。
Export All...(汇出全部...)	汇出所有程序代码模板至档案系统。
Automatically add comments for new methods and types(自动新增方法和类型的	这个设定指定批注程序代码模板是否会自动新增至所有新的方法。如果停用，仅在明确地新增批注（如使用新增 Javadoc 批注动作）时，才会使用批注程序代码模板。请注意，这个设定并不套用至程序代码模板（如新建 Java 档案）中所含的批注

动作	说明
批注)	

批注范本(Comment templates)

批注模板可包含 ${tags} 变量，这个变量将被已加注元素的标准 Javadoc 标示 (@param, @return..) 所替代。此外，「置换方法」批注可包含 ${see_to_overridden} 范本

- 建构子批注：指定新建构子批注的范本
- 类型批注：指定新类型批注的模板。请注意，可在「新建 Java 文件」范本中参照这个范本
- 方法批注：指定新方法（不置换基础类别中的方法）批注的范本
- 置换方法批注：指定新方法（置换基础类别中的方法）批注的范本。依预设，批注会定义成非 Javadoc 批注（Javadoc 将把这个批注换成已置换方法的批注）。如果想要的话，可以将这个批注变更为真实的 Javadoc 批注

新建 Java 文件范本(New Java files template)

当建立新档案时，「类别」和「接口」精灵就会使用「新建 Java 文件」模板。模板可以指定要新增批注之处。请注意，模板可以含有 ${typecomment} 变量，这个变量将被类型批注模板的评估值所替代。

Catch 区块主体范本(Catch block body template)

当建立 catch 区块主体时，就会使用「Catch 区块主体」模板。它可以使用 ${exception_type} 和 ${exception_var} 变数。

方法主体范本(Method body template)

当建立含有主体的新方法时，就会使用「方法主体」模板。它含有解析为 return 陈述式或/和 super 呼叫的 ${body_statement} 变量。

建构子主体范本(Constructor body templates)

当建立含有主体的新方法或建构子时，就会使用「建构子主体」模板。它含有解析 super 呼叫的 ${body_statement} 变量。

「程序代码范本」对话框(Code Template dialog)

对话框中的字段与按钮如下：

选项	说明
Description(说明)	范本的说明
Pattern(型样)	范本的型样。
Insert Variables...(插入变数...)	显示预先定义之模板特有变量的清单。

3.6.5 编译器(Compiler)

下列各段将分别说明编译器的喜好设定：

- 问题

- 样式
- 相容和类别档
- 建置路径

问题(Problems)

选项	说明	默认值
Unreachable code(无法呼叫到的程序代码)	无法呼叫到程序代码，可选择性地报告成错误、警告，或者加以忽略。字节码一律产生最佳化程序代码。请注意，根据 Java 语言规格，无法呼叫到的程序代码应该是一个错误。	错误
Unresolvable import statements(无法解析的 import 陈述式)	无法解析的 import 陈述式可选择性地报告成错误、警告，或加以忽略。请注意，根据 Java 语言规格，无法解析的 import 陈述式应该是一个错误。	错误
Unused local variables (i.e. never read)(未使用的区域变量（如从未读取）)	当启用时，编译器会针对未用的区域变量（亦即：从未读取的变量），发出错误或警告。	忽略
Unused parameters (i.e. never read)(未使用的参数（如从未读取）)	当启用时，编译器会针对未用的方法参数（亦即：从未读取的参数），发出错误或警告。	忽略
Unused imports(未用的汇入)	当启用时，编译器会针对未用的汇入参照，发出错误或警告。	警告
Unused private types, methods or fields (未用的 private 类型、方法或字段)	当启用时，每当宣告 Private 方法或字段时，但从未在同一单元内使用时，编译器将发出错误或警告。	忽略
Usage of non-externalized strings(未提出字符串的用法)	当启用时，编译器将为未提出的字符串文字发出错误或警告（如，未标示的 //$NON-NLS-<n>$）。	忽略
Usage of deprecated API(已停用的 API 的用法)	当启用这个选项时，编译器会将使用已停用的 API 标为错误或警告。	警告
Signal use of deprecated API inside deprecated code(已停用的程序代码内之已停用的 API 的信号使用)	一旦启用，编译器将发出信号，指出在已停用的程序代码内使用已停用的 API。问题的严重性是由「已停用的 API 的用法」选项来控制。	关闭
Maximum number of problems reported per compila	指定各编译单元所能报告的问题数上限。	100

tion unit(各编译单元所能报告的问题数上限)		

样式(Style)

选项	说明	默认值
Methods overridden but not package visible(已置换但套件看不到的方法)	套件的预设方法在另一套件中看不到，因此无法置换。当启用这个选项时，编译器会将这类情况标为错误或警告。	警告
Methods with a constructor name(建构子名称中的方法)	如果以建构子名称来命名方法，通常会被视为较差的程序设计风格。当启用这个选项时，编译器会将这类情况标为错误或警告。	警告

Conflict of interface method with protected 'Object' method(接口方法与受保护的「对象」方法发生冲突)	当启用时，每当接口定义一个与非继承「对象」方法不兼容的方法时，编译器将发出错误或警告。直到解决这个冲突之前，将无法实作如此的接口，如
	interface I {
	int clone();
	}
		警告

Hidden catch blocks(隐藏的 catch 区块)	在本端环境下对于 try 陈述式而言，某些 catch 区块可能会隐藏其它者，例如：
	try { throw new java.io.CharConversionException();
	} catch (java.io.CharConversionException e) {
	} catch (java.io.IOException e) {}.
	当启用这个选项时，编译器会针对对应至所检查之异常状况的快取区块隐藏，发出错误或警告。
		警告

Non-static access to a static member(Static 成员的非 Static 存取权)	当启用时，每当以表示式接收器存取 Static 字段或方法时，编译器将发出错误或警告。应该以类型名称限定 Static 成员的参照。	警告
Access to a non-accessible member of an enclosing type(存取含括类型中无法存取的成员)	当启用时，只要其模拟存取含括类型中无法存取的成员，编译器即会发出错误或警告。这类存取可能涉及效能。	忽略
Assignment has no effect (e.g. 'x = x')(指定没有生效（例如'x=x'）)	当启用时，每当指派没有效果（如 'x = x'）时，编译器将发出错误或警告。	警告

Using a char array in string concatenation(在字符串连结中使用 char 数组)	当启用时，每当在下列「字符串」连结中使用 char[] 表示式时，编译器就会发出错误或警告： "hello" + new char[]{'w','o','r','l','d'}	警告

相容和类别档(Compliance and Class files)

选项	说明	默认值
Compiler compliance level(编译器兼容层次)	指定 JDK 编译器兼容层次。	1.3
Use default compliance settings(使用预设兼容设定)	当启用时，在编译器的兼容层次方面，会套用预设的兼容设定。	开启
Generated class files compatibility(所产生的类别档兼容性)	指定所产生的类别档兼容性。	1.1
Source compatibility(程序文件兼容性)	指定程序文件和 1.3 或 1.4 兼容。从 1.4 开始，"assert" 为保留给主张支持的关键词。	1.3
Report 'assert' as identifier(将'assert'报告成识别码)	当启用时，只要 'assert'（为 JDK 1.4 中的保留关键词）被当成识别码使用，编译器即会发出错误或警告。	忽略
Add variable attributes to generated class files(新增变量属性到产生的类别文件中)	当启用时，会在类别文件中新增变量属性。这会让区域变量名称显示在除错器中（位于明确指定变量之处）。产生的 .class 档会变大。	开启
Add line number attributes to generated class files(新增行号属性到产生的类别文件中)	当启用时，会在类别文件中新增行号信息。这会在除错器中强调显示出程序代码。	开启
Add source file name to generated class file(新增程序文件名称到产生的类别档中)	当启用时，会在类别文件中新增程序文件名称。这会让除错器显示对应的程序代码。	开启
Preserve unused local variables(保留未用的区	当启用时，则不会将未用的区域变量（亦即，从未读取）从类别档中除去。如果除去这项，有可能会改变除错。	开启

域变量)		

建置路径(Build Path)

选项	说明	默认值
Incomplete build path(不完整的建置路径)	指出当类别路径上的项目不存在、不合规定或看不见（如关闭了参照项目）时，所报告的问题的严重性。	错误
Circular dependencies(循环相依项)	指出在循环中并入项目时所报告的问题的严重性。	错误
Duplicated resources(重复的资源)	指出当多次出现的资源将复制到输出位置时所报告的问题的严重性。	警告
Abort building on build path errors(当建置路径错误时中止建置)	容许如果类别路径无效，将建置器切换至中止。	开启
Scrub output folders on full build(进行完整建置时清除输出数据夹)	指出是否容许「Java 建置器」在执行完整建置作业时清除输出数据夹。	开启
Enable using exclusion patterns in source folders(启用在来源数据夹中使用排除型样)	当停用时，项目类别路径上没有项目可与排除型样相关联。	开启
Enable using multiple output locations for source folders(启用对来源数据夹使用多个输出位置)	当停用时，项目类别路径上没有项目可与特定输出位置相关联，因而防止使用多个输出位置。	开启
Filtered resources(过滤的资源)	以逗点分格方式列出不复制到输出数据夹中的档案型样。	"

3.6.6 Java 编辑器(Java editor)

这个页面可以设定如下的 Java 编辑器喜好设定：

- 外观
- 语法
- 程序代码协助

■ 问题指示

外观(Appearance)

外观指定 Java 编辑器的外观。

选项	说明	默认值
Displayed tab width(Tab 的显示宽度)	指定 Tab 的显示宽度（以空格为单位）。	4
Print margin column(打印边距直栏)	指定其后要显示打印边距的直栏。 如果要显示打印边距，请启 Show print margin 选项；如果要指定打印边距的颜色，请使用 Appearance color options 喜好设定。	80
Synchronize outline selection on cursor move(在光标移动时将概要选项同步化)	当启用时，「概要」视图恒会在 Java 编辑器中选取含括光标的 Java 元素。	关闭
Show overview ruler(显示概观标尺)	当启用时，Java 编辑器右边框中会出现概观标尺，并显示整个可视文件的问题。	开启
Show line numbers(显示行号)	当启用时，Java 编辑器左边框中的垂直标尺会显示可视文件的行号。 可在 Appearance color options 中指定行号的颜色。	关闭
Highlight matching brackets(强调显示对称的括号)	当启用时，只要光标是位于小括号、方括号或大括号旁，即会强调显示其相对的左或右括号。 可在 Appearance color options 中指定括号的强调显示颜色。	开启
Highlight current line(强调显示现行行)	当启用时，则会强调显示光标所在现行行的背景。 可在 Appearance color options 中指定现行行的背景颜色。	开启
Show print margin(显示打印边距)	当启用时，会显示打印边距。 可使用 Print margin column 和 Appearance color options 喜好设定，来判定打印边距的位置与颜色。	关闭
Appearance color options(外观颜色选项)	可在这里指定各种 Java 编辑器外观特性的颜色。 Line number foreground(行号前景)：行号的颜色。 Matching brackets highlight(相符的方括号强调显示)：括号的强调显示颜色。 Current line highlight(现行行强调显示)：现行行的强调显示颜色。 Print margin(打印边距)：打印边距的颜色。 Find scope(寻找范围)：寻找范围的颜色。 Linked position(链接的位置)：程序代码辅助中所用链接位置的颜色。	预设颜色

选项	说明	默认值
	Link(链接)：链接颜色	

语法(Syntax)

语法指定如何展现 Java 程序代码。

选项	说明	默认值
Background color(背景颜色)	System default(系统默认值)：操作系统所提供的预设背景颜色。 Custom(自订)：使用者定义的背景颜色。	系统默认值
Foreground(前景)	下列的 Java 程序文件片段可以不同的颜色与样式展现： Multi-line comment(多行批注)：批注的格式为'/*...*/' Single-line comment(单行批注)：批注的开头为'//' Keywords(关键词)：所有 Java 关键词。 Strings(字符串)：Java 字符串和字符，以单引号与双引号括住 Others(其它)：预设 Java 程序代码 Task tags(作业标示)：批注中的作业标示 Javadoc keywords(Javadoc 关键词)：Javadoc 中所用的关键词，开头为'@' Javadoc HTML tags(Javadoc HTML 标示)：Javadoc 中所用的 HTML 标示 Javadoc links(Javadoc 链接)：{@link reference}标示 Javadoc others(Javadoc 其它)：预设 Javadoc 文字	预设颜色和样式
Preview(预览)	显示和现行颜色和样式有关之 Java 程序代码的预览。	n/a

程序代码辅助(Code assist)

程序代码辅助指定程序代码辅助的行为与外观。

选项	说明	默认值
Completion inserts/Completion overwrites(完成插入/完成改写)	如果开启了完成插入(Completion inserts)，完成文字将插入在脱字符号 (^) 位置，所以它绝不会改写任何现有的文字。 如果开启了完成改写(Completion overwrites)，完成文字将取代脱字符号 (^) 位置之后直到字尾的字符。	完成插入
Insert single proposals automatically(自动插入单一提议)	当启用时，程序代码辅助会自动选择与插入单一提议。	开启
Show only proposals visible in the invocation	当启用时，则会使用显示规则来限制 Java 元素的提议。例如，不显示其它类别的 private 字段提议。	开启

选项	说明	默认值
context(在呼叫环境定义中只显示可见的提议)		
Present proposals in alphabetical order(按字母顺序来提供提议)	当启用时，则提议会按字母顺序排序。	关闭
Automatically add import instead of qualified name(自动新增汇入取代完整名称)	当启用时，则位于其它套件中的类型提议会呼叫以新增对应的汇入宣告。否则，会完整插入类型。	开启
Fill argument names on method completion(方法完成时填入自变量名称)	当启用时，则在选择方法提议时，会插入该方法之宣告中指定的自变量名称。	关闭
Enable auto activation(启用自动启动)	当启用时，可自动呼叫程序代码辅助。 可在自动启动延迟(Auto activation delay)、自动启动 Java 触发(Auto activation triggers for Java)以及自动启动 Javadoc 触发(Auto activation triggers for Javadoc)中指定自动呼叫的条件。	开启
Auto activation delay(自动启动延迟)	一旦从遇到自动启动触发字符起到输入新字符止的时间，超过自动启动延迟，即会呼叫程序代码辅助。	500
Auto activation triggers for Java(自动启动 Java 触发)	如果 Java 程序代码内键有一个触发字符（但不是键于 Javadoc 批注内），则一旦在自动启动延迟逾时前未输入其它字符，即会呼叫程序代码辅助。	'.'
Auto activation triggers for Javadoc(自动启动 Javadoc 触发)	如果 Javadoc 内键有一个触发字符，则一旦在自动启动延迟逾时前未输入其它字符，即会呼叫程序代码辅助。	'@'
Code assist color options(程序代码辅助颜色选项)	下列程序代码辅助 UI 元素所用的颜色： Completion proposal background(完成提议背景)：完成提议窗口的背景颜色 Completion proposal foreground(完成提议前景)：完成提议窗口的前景颜色 Method parameter background(方法参数背景)：参数窗口的背景颜色 Method parameter foreground(方法参数前景)：参数窗口的前景颜	预设颜色

选项	说明	默认值
	色 Completion overwrite background(完成改写背景)：完成改写窗口的背景颜色 Completion overwrite foreground(完成改写前景)：完成改写窗口的前景颜色	

附注(Annotations)

附注指定何时及如何显示附注。

选项	说明	默认值
Analyze annotations while typing(输入时分析附注)	如果启用，则在使用者输入时，就会更新附注。否则，直到编译 Java 档案之前，都不会更新附注。	开启
Indicate annotations solvable with Quick Fix in vertical ruler(在垂直标尺中指出可利用快速修正解决的注释)	针对每一个可透过快速修正解决的注释，在 Java 编辑器左边框的垂直标尺中，显示一个灯泡。	开启
Annotation presentation(附注呈现方式)	对于每一类型的附注，可以指定 以文字、以概观标尺或两者显示附注 以何种颜色展现附注	

范本(Templates)

「模板」喜好设定页面可以建立新模板与编辑现有模板。模板可方便程序设计师快速插入常重复出现的程序代码型样。

下列按钮可以操作与配置模板：

动作	说明
New...(新建...)	开启对话框以建立新模板。
Edit...(编辑...)	开启对话框以编辑目前所选取的范本。
Remove(移除)	移除所有选取的范本。
Import...(汇入...)	从档案系统汇入模板。
Export...(汇出...)	将选取的所有模板汇出至档案系统。
Export All...(汇出全部...)	将所有模板汇出至档案系统。

动作	说明
Enable All(全部启用)	启用所有范本。
Disable All(全部停用)	停用所有范本。
Use Code Formatter(使用程序代码格式制作器)	当启用时，在插入之前，会先根据程序代码格式制作器(Code Formatter)喜好设定中指定的程序代码格式设定规则，来设定模板的格式。否则，范本会按原样插入，但内缩将不正确。 请参阅「程序代码格式制作器」喜好设定页面

「范本」对话框(Template dialog)

新建模板与编辑现有模板所用的对话框相同，以下是其说明。

对话框中的字段与按钮如下：

选项	说明
Name(名称)	模板的名称。
Context(环境定义)	环境定义是决定哪些地方可使用模板，以及决定可用的一组预先定义的模板变量。 Java：Java 环境定义 Javadoc：Javadoc 环境定义
Description(说明)	模板的说明，会在使用者选择模板时显示。
Pattern(型样)	范本的型样。
Insert Variables...(插入变数...)	列出预先定义的环境定义特定变量。

模板变量(Template variables)

Java 和 Javadoc 环境定义皆会定义下列变量：

变数	说明
${cursor}	指出当离开模板编辑模式时的光标位置。当离开模板编辑模式时，如果希望光标应移至另一位置，而非移至模板尾端，则可善用这项。
${date}	评估成现行日期。
${dollar}	评估成钱币符号 '$'。 另外，可以使用两个货币符号：'$$'。
${enclosing_method}	评估成含括名称的名称。
${enclosing_method_arg	评估成含括方法的以逗点区隔之自变量名称清单。这个变量有助于为许多方法

变数	说明
uments}	产生 log 陈述式。
${enclosing_package}	评估成含括套件的名称。
${enclosing_project}	评估成含括项目的名称。
${enclosing_type}	评估成含括类型的名称。
${file}	评估成档案的名称。
${return_type}	评估成含括方法的传回类型。
${time}	评估成现行时间。
${user}	评估成使用者名称。

此外，Java 环境定义会定义下列变量：

变数	**说明**
${array}	评估成已宣告之数组名的提议。
${array_element}	评估成已宣告之数组的元素名称的提议。
${array_type}	评估成已宣告之数组的元素类型的提议。
${collection}	评估成实作 java.util.Collection 之已宣告集成的提议。
${index}	评估成未宣告之数组索引迭代的提议。
${iterator}	评估成未宣告之集成迭代的提议。

3.6.7 JRE 安装(JRE installations)

「类别路径变量」喜好设定

选项	**说明**
Add...(新增...)	将新的 JRE 定义新增至工作台中。在产生的对话框中，指定下列： JRE 类型：（从下拉列表中选取一种 VM 类型） JRE 名称：输入这个 JRE 定义的名称 JRE 起始目录：输入或浏览以选取这项 JRE 安装的根目录 Javadoc URL：输入或浏览以选取 URL 位置。这个位置供 Javadoc 汇出精灵做为默认值，并供「开启外部 Javadoc」动作使用。除错器逾时值：输入这个 JRE 之除错器的预设逾时值（以毫秒为单位） 如果要让这个 JRE 采用预设链接库位置，请勾选这个勾选框；如果要输入或

选项	说明
	浏览以便为下列档案指定链接库位置，请清除这个勾选框： JAR 檔（例如：classes.zip） 程序文件（例如：source.zip） 可以按一下「Browse」按钮，以浏览并找出所要的路径。
Edit...(编辑...)	可让编辑所选取的 JRE。
Remove(移除)	将所选取的 JRE 从工作台中移除。
Search...(搜寻...)	自动搜寻已安装在本端档案系统的 JRE，并在工作区中建立对应的 JRE 定义。

3.6.8 JUnit

选项	说明	默认值
Show the JUnit results view only when an error or failure occurs (只有在发生错误或失败时，才显示 JUnit 结果视图)	当启用时，则只有在发生错误或失败时才会将 JUnit 视图带至前面。	开启
Stack trace filter patterns(堆栈追踪过滤器型样)	测试失败时，不应该显示在堆栈追踪的套件、类别或型样。	预设过滤器型样

3.6.9 新专案(New project)

选项	说明	默认值
Source and output folder(程序文件与输出数据夹)	专案：将程序文件与输出位置皆设为项目的根目录。 资料夹：可个别设定程序文件和输出位置。 如果要指定程序文件与输出位置，请设定来源数据夹名称(Source folder name)与输出位置名称(Output location name)。	专案
Source folder name(来源资料夹名称)	程序文件的位置。	'src'
Output location name(输出位置名称)	输出文件的位置。	'bin'
As JRE library use(使用 JRE 链接库时)	指定所要使用的 JRE 链接库。 JRE 储存器 JRE 储存器。 JRE_LIB 变数 JRE_LIB 变数指定的 JRE。	JRE 储存器

3.6.10 组织汇入(Organize imports)

下列的喜好设定是定义「组织汇入」指令要如何在编译单元中产生 import 陈述式。

「组织汇入」喜好设定

选项	说明	默认值
Import order list(汇入顺序清单)	这份前缀清单显示套件汇入到 Java 编译单元中的顺序。每一个项目皆定义出一个区块。区块之间各空一行。	java javax org com
New...(新建...)	新增套件名称前缀到汇入顺序清单中。在产生的对话框中，输入套件名称或套件名称前缀。	n/a
Edit...(编辑...)	变更现有套件名称前缀的名称。在产生的对话框中，输入套件名称或套件名称前缀。	n/a
Up(上)	在汇入顺序清单中，将所选套件名称前缀往上移。	n/a
Down(下)	在汇入顺序清单中，将所选套件名称前缀往下移。	n/a
Remove(移除)	将套件名称前缀从汇入顺序清单中移除。	n/a
Load...(载入...)	从档案加载套件名称前缀清单。	n/a
Save...(储存...)	将套件名称前缀清单储存至档案。	n/a
Number of imports needed before .* is used(使用.*之前所需的汇入数目)	在使用 <package>.* 之前，可来自相同套件的完整 import 陈述式的数目。	99
Do not create imports for types starting with a lower case letter(不为以小写字母开头的类型建立汇入)	当启用时，则不会汇入以小写字母开头的类型。	开启

3.6.11「重构」喜好设定(Refactoring preferences)

可在「重构」喜好设定页面中设定下列的喜好设定。(「Window」→「Preferences」→「Java」→「Refactoring」。)

选项	说明	默认值
Confirm the execution of the refactoring if(如果发生下列情况，请确认执行重构)	在这个区段中，请选取哪些种类的问题会使精灵在按下 Finish 之后，仍维持开启状态并显示问题： 会阻止实际重构执行的问题 搜寻工作台 工作台中的警告 前置条件检查所产生的信息	错误

选项	说明	默认值
	当问题的严重性比所选的层次还低时，则可让进行重构，而不必先预览结果。	
Save all modified resources automatically prior to refactoring(在重构前自动储存所有修改过的资源)	如果启用这个选项，则每当执行重构动作时，工作台会自动储存自前次储存以来所有已修改过的资源。	不勾选

3.6.12 作业标示(Task Tags)

在这个喜好设定页面上，可以配置作业标示。当标示清单不是空的时候，每当编译器在 Java 程序代码中的任何批注内遇到其中一个对应标示时，编译器将发出作业标记。所产生的作业讯息将包括标示，以及直到下一个字行分隔字符或批注结尾的范围。

指令	说明
New...(新建...)	新增作业标示。请在出现的对话框中，指定新作业标示的名称和优先级。
Remove(移除)	移除所选作业标示。
Edit...(编辑...)	可以编辑所选作业标示。请在出现的对话框中，编辑作业标示的名称和/或优先级。

有一个名为 TODO、优先级为 Normal 的预设作业标示。

3.7 团队(Team)

团队喜好设定页面包含会影响版本管理团队支持的选项。

选项	说明	默认值
Use compressed folders as default Synchronize view layout(利用压缩数据夹作为预设的「同步化」视图布置)	在初次将同步化加入「同步化」视图时，请利用这个选项来配置所用的预设布置。可以在视图下拉菜单中后续变更这个布置。	已启用
Show all synchronization information in a resource's text label(将所有同步化信息显示在资源的文字卷标中)	请利用这个选项，将资源的同步化状态显示成资源卷标中的文字。依预设，只会用一个图示装饰元来识别资源的同步化状态。	已停用
Switch to the associated	请利用这个选项来配置执行同步化作业时会发生的状况。选项	提示

选项	说明	默认值
perspective when a synchronize operation completes(当同步化作业完成时，切换到相关的视景)	如下： Always(固定)：一律切换视景 Never(绝不)：绝不切换视景 Prompt(提示)：提示切换视景	
Perspective Switching(视景切换)	请利用这个选项来配置执行同步化作业时所显示的视景。	「团队同步化」视景

3.7.1 CVS

在 CVS 喜好设定页面中，可以自订 CVS 外挂程序的若干项目。

一般 CVS 喜好设定(General CVS Preferences)：

选项	说明	默认值
Prune empty directories(删改空目录)	可以使用这个选项来指定在更新以及同步处理视图中删改空白的目录。虽然删改的目录不会显示在工作台中，在储存库中仍会存有一个空目录。这是有帮助的，因为 CVS 不会让客户端从服务器移除目录。	已启用
Consider file contents in comparison(在比较时考虑档案内容)	当比较 CVS 资源时，请利用这个选项来比较所找到已变更的档案之内容。通常会利用时间戳记来比较 CVS 档案，这是目前最快的方法。不过，在某些情况下，经由比较档案内容可得到较为精确的比较。停用这个选项会加快比较的速度，但可能会产生内容相同的比较项目。这个选项只适用于比较，不适用于合并和工作区同步化。	已启用
Delete unmanaged resources on replace(取代时删除未管理的资源	可以使用这个选项，在取代为储存库的资源时，允许删除不在 CVS 控制下的资源。	已启用
Treat all new files as binary(将所有新档案视为二进制文件)	可以使用这个选项来置换档案内容设定以及将所有新档案视为二进制文件。	已停用
Validate server version compatibility on first connection(第一次联机时验证服务器版本的兼容性)	可以使用这个选项，在第一次联机时查询 CVS 服务器版本，以判断服务器的兼容性。 服务器版本会输出到主控台，如果侦测到不兼容，就会在联机时记载警告讯息。	已启用
Confirm move tag on tag operation(确认在标	请利用这个选项，以便在选取「移动标示」选项时得到提示。	已启用

选项	说明	默认值
示作业上移动标示)		
Display detailed protocol output to stdout(将详细的通讯协议输出显示在标准输出中)	请利用这个选项来显示工作台和 CVS 服务器之间的通讯追踪。	已停用
Convert text files to use platform line neding(转换文字文件来使用平台行尾)	请利用这个选项，将文字文件的行尾转换成平台所用的行尾。如果将资源移出到 Windows 机器所装载的 *nix 磁盘驱动器，可以停用这个选项。	已启用
Show revision comparisons in dialog(在对话框中显示修订比较)	请利用这个选项，将修订比较显示在对话框中，而不是显示在比较编辑器中。	已停用
Communication timeout(通讯逾时)	可以使用这个选项来配置在逾时前要等待联机到 CVS 服务器的时间总数（以秒为单位）。	60 秒
Quietness level(安静层次)	设定针对指令的 CVS 打印状态信息总数。在有点安静(Somewhat quiet)模式中，会抑止打印不重要的参考信息。重要性的考虑是根据每个指令。在非常安静(Very quiet)模式中，除了完成指令的必要输出外，所有的输出都会被抑止。在非常安静(Very quiet)模式中，有些 CVS 服务器可能无法告知一些已经发生的错误的重要信息。可能要考虑改用有点安静(Somewhat quiet)模式。	细节
Default keyword substitution(预设的关键词替代)	可以使用这个选项来设定文字文件的预设关键词替代。	包含关键词展开项 -kkv 的 ASCII
Save dirty editors before CVS operations(在 CVS 作业之前储存变动过的编辑器)	可以使用这个选项来配置在执行 CVS 作业时，如果已开启的编辑器包含未储存的变更时，会发生什么情况。 选项包括： Never(绝不)：继续执行 CVS 作业，即使已开启的编辑器中有尚未储存的变更。 Prompt(提示)：询问使用者要如何处理已开启的编辑器中的未储存变更。 Auto-save(自动储存)：在每一个 CVS 作业前自动储存已开启编辑器中尚未储存的变更。	提示

主控台喜好设定(Console preferences)：

选项	**说明**	**默认值**

选项	说明	默认值
Console text color settings(主控台文字的颜色设定)	可以使用这些选项来变更「CVS 主控台」中所显示的文字的颜色。 指令行文字（黑色） 消息正文（蓝色） 错误文字（红色）	
Show CVS output in Console view(在「主控台」视图中显示 CVS 输出)	请利用这个选项，将 CVS 指令输出显示在「主控台」视图中。启用这个选项可以显示非常有用的信息，但指令作业速度会变慢。	已停用

「Ext 联机方法」喜好设定(Ext Connection Method preferences)：

Use external program vs. Use internal connection method(使用外部程序和使用内部联机方法)	这个页面可以配置 ext 联机方法来使用外部程序，或配置另一个联机方法来连接到服务器。后面一个选项是要让 extssh 之类的自订联机方法保持与外部 CVS 客户端工具兼容。	使用外部程序
CVS_RSH(CVS_RSH)	可以使用这个选项来配置在在联机到远程 CVS 服务器时要呼叫的程序。呼叫 RSH 指令时会使用下列呼叫型样： *CVS_RSH 参数 CVS_SERVER*	ssh
Parameters(参数)	可以使用这个选项来配置传送到 CVS_RSH 程序的参数。预设参数型样为 {host} -l {user}。可使用 {host}、{user}、{password} 与 {port} 等变量来加以修改。	{host} -l {user}
CVS_SERVER(CVS_SERVER)	可以使用这个选项来配置要执行的远程 CVS 服务器程序的名称。只有当远程 CVS 服务器二进制名称与默认值不同时，才变更这个设定	cvs
Connection type(联机类型)	如果启用使用另一个联机方法的选项，请利用这个选项来设定使用 ext 联机方法的储存库位置所用的联机方法。	

「标签装饰」喜好设定(Label Decorations preferences)：

Text(文字)	可以使用这个页面中的选项来配置如何将 CVS 信息新增至文字卷标。
Icons(图示	可以使用这个页面中的选项来配置可用哪些图标作为覆盖，以便在视图中显示 CVS 特定的信息。
General(一般)	可以使用这个页面中的选项来配置有关装饰元的几个喜好设定： 计算数据夹的深度送出状态(Compute deep outgoing state for folders) 可以使用这个选项来配置是否应该计算数据夹的送出指示器。停用这个选项时，可增进 装饰元的效能，因为计算数据夹的已用过状态时，需要计算所有子项资源的已用过状态。（预设是启用的）

密码管理(Password Management)：

这个喜好设定页面可以查看哪些储存库位置的密码快取在金钥环档案中，且可以除去这些密码。

SSH2 联机方法(SSH2 Connection Method)：

General(一般)	请利用这个页面中的选项来配置 ssh 金钥目录的位置，以及联机时要将哪些金钥传给服务器。
Proxy	请利用这个页面中的选项来配置 HTTP 或 SOCKS5 Proxy。
Key Management(金钥管理)	请利用这个页面中的选项来建立、管理和汇出金钥

监视/编辑喜好设定(Watch/Edit preferences)：

Configure projects to use Watch/edit on checkout(配置项目以便在移出时使用监视/编辑)	可以使用这个选项来指出从储存库移出的档案都要变成只读。	停用
When read-only files are modified in an editor(在编辑器中修改只读文件的时机)	可以使用这个选项来配置当另一个工具或已开启的编辑器在修改只读档时，会发生何种状况。 传送 cvs 编辑通知给服务器(Send a cvs edit notification to the server)：在容许写入档案前发出 cvs 编辑通知给服务器。 如果档案中有其它编辑器，就会提示使用者是否要继续或取消。 编辑档案而不通知服务器(Edit the file without informing the server)：使档案变成只读而不通知服务器。	传送 cvs 编辑通知给服务器
Before a CVS edit notification is sent to the server(在传送 CVS 编辑通知给服务器之前)	请利用这个选项来配置当开启的编辑器或另一个工具在修改只读档，且启用了将 CVS 编辑通知传给服务器(Send a cvs edit notification to the server)时，会发生何种状况。 Always Prompt(固定提示)：固定提示使用者进行确认 Only prompt if there are other editors(只在有其它编辑器时才提示)：向使用者显示现行编辑器的清单，让使用者确认或取消编辑。 Never Prompt(绝不提示)：传送编辑通知，但不提示	只在有编辑器时才提示

3.7.2 忽略的资源(Ignored Resources)

在「Team」→「Ignored Resources」喜好设定页面中，可以指定要排除在版本控制管理系统之外的文件名称型样。它有一份档案型样清单，资源必须符合这些档案型样，才能成为版本控制候选项。这些型样可含有万用字符 "*" 和 "?"。型样 "*" 代表任何零或多个字符的序列。型样 "?" 代表任何一个字符。例如，可以指定型样 "*~"，其将会比对任何以 "~" 结尾的暂存档。在更新或确认作业期间，将会忽略任何符合任一型样的档案或目录。

如果要将档案类型新增至忽略清单，只需要按一下 Add 按钮。在对话框中，请输入档案类型（例如

*.class）。如果要从忽略清单中移除档案类型，只需要选取忽略清单中的档案类型，按一下 Remove 按钮 。可以从清单中取消选取某个档案型样，暂时停用这个档案型样的忽略功能。不必从清单中移除指定的档案型样，就可以暂时停用它。

3.7.3 档案内容(File Content)

在「Team」→「File」喜好设定页面中，可以将扩展名关联到档案所包含的数据类型。档案内容类型的两个选项是 ASCII 和二进制。然后，像 CVS 这类的储存库提供者就可以使用这项信息来为内容类型提供适当的行为。例如，在 ASCII 档案中，CVS 可确保行终止器符合 OS 平台的行终止器。

将项目新增至「档案内容」页面的方法有两种。第一个方法是透过工作台外挂程序的帮助。整合到工作台的工具可以为工作台提供工具专用的扩展名的档案内容类型。工作台本身也会定义在工作台中经常使用和出现的扩展名的档案内容类型（例如，html、gif 等等）。

第二个方法是让使用者在「档案内容」喜好设定页面中明确地新增档案内容类型。如果要这么做，使用者只需要按一下 Add 按钮，然后输入扩展名。之后，他们可以切换类型与扩展名之间的关联，方法是在表格中选取扩展名的项目，然后按一下 Change。使用者可以选取项目，然后按一下 Remove，将项目从清单中移除。

4. Java 程序开发

在 Eclipse 中做任何事之前，都必须新增一个项目。Eclipse 可透过外挂支持数种项目(如 EJB 或 C/C++)，预设支持下列三种项目：

- **Java Project** – Java 开发环境
- **Plug-in Project** – 自行开发 plug-in 的环境
- **Sample Project** – 提供操作文件的一般环境

如图 4.1

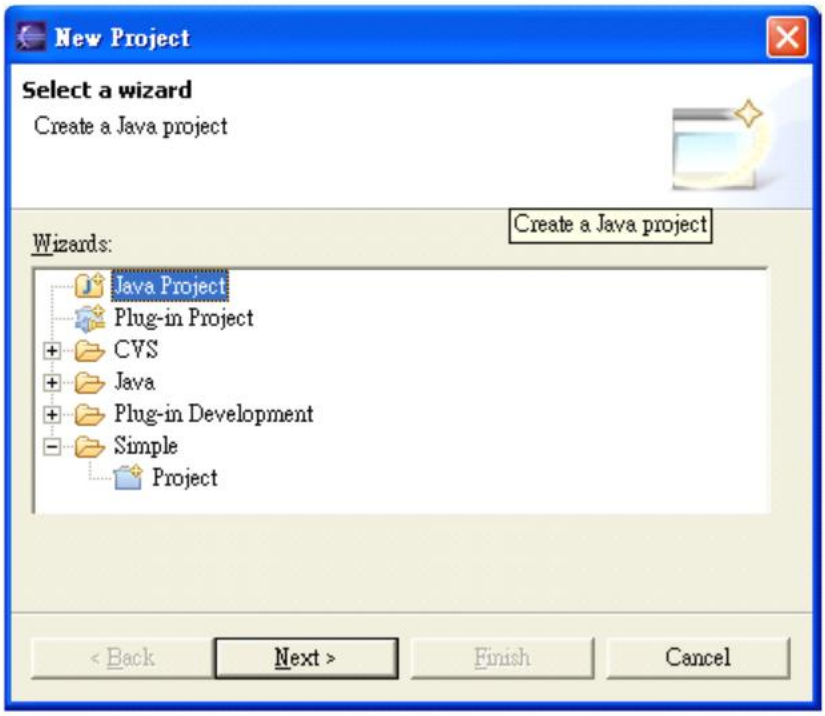


图 4.1

4.1 建立 Java 项目

新增 Java 项目的步骤：

I. 选择「File」→「New」→「Project」

(或是在『Package Explorer』窗口上按鼠标右键，选择「New」→「Project」选单选项)

(或是按工具列上 New Java Project 的按钮)

II. 在 New Project 对话框(图 4.1)，选 Java Project

(或是展开 Java 的数据夹，选 Java Project，如图 4.2)

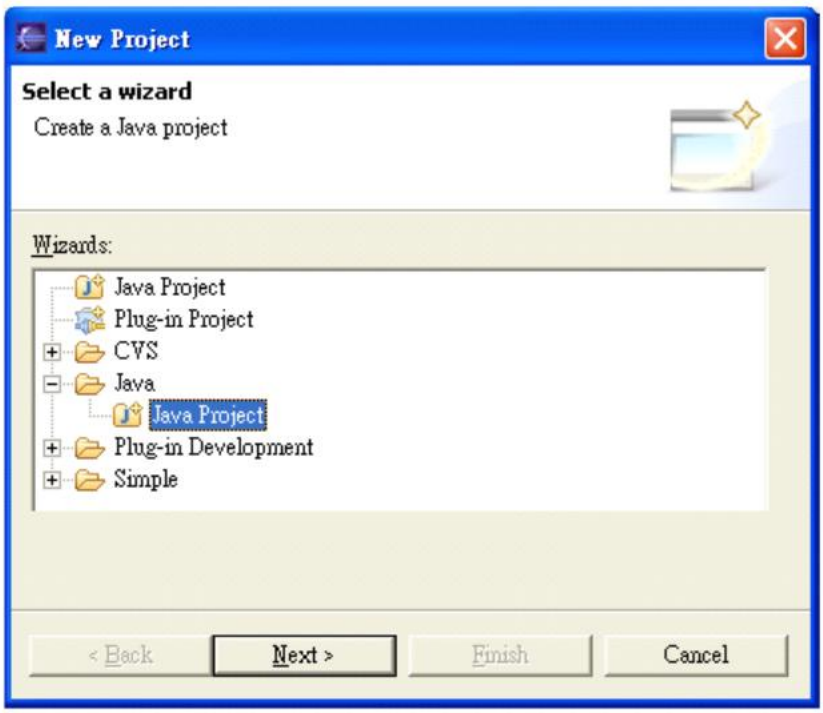


图 4.2

III. 在 New Java Project 的窗口中输入 Project 的名称，如图 4.3

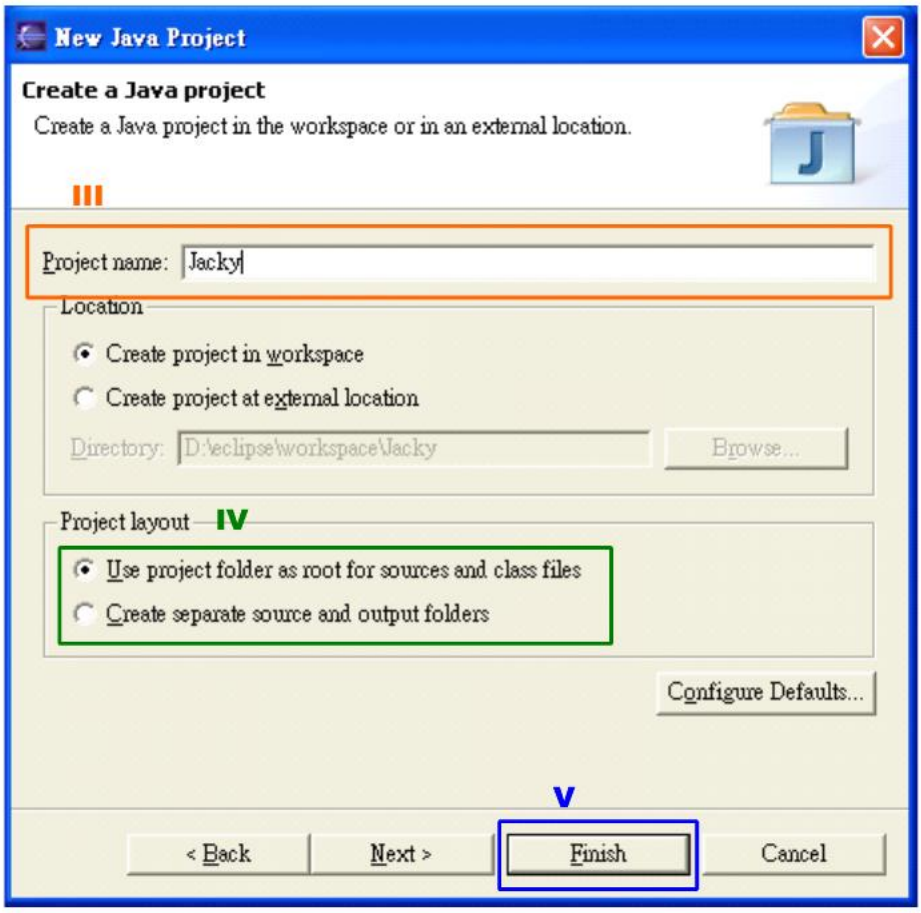


图 4.3

IV. 在 Project Layout 中可以选择编译好的档案是否要和原始档放在同一个目录下，如图 4.3

V. 按下 Finish

4.2 建立 Java 类别

新增 Java 类别的步骤：

I. 选择「File」→「New」→「Class」

(或是在『Package Explorer』窗口上按鼠标右键，选择「New」→「Class」选单选项)

(或是按工具列上 New Java Class 的按钮)

II. 在 New Java Class 窗口中，Source Folder 字段默认值是项目的数据夹，不需要更改。

III. Package 字段输入程序套件的名称

IV. Name 字段输入 Class Name

V. 在 Which method would you like te creat 的部份，有勾选 public static void main(String[] args)的话，会 generate main method

VI. 按 Finish，会依套件新增适当的目录结构及 Java 原始文件

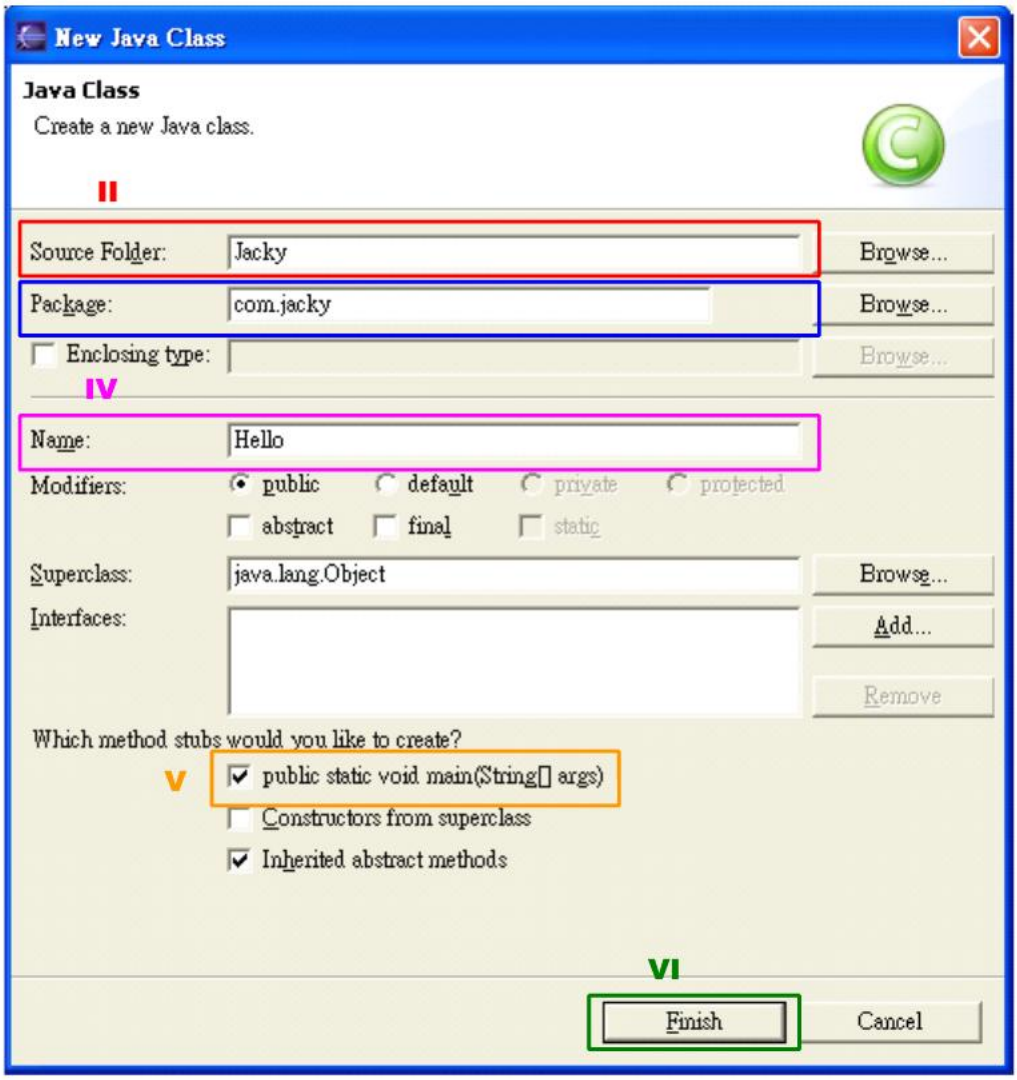


图 4.4

在 Package Explorer 的视图中可以看到程序的结构

在 Navigator 的视图中可以看到套件的目录架构

4.3 程序代码完成功能

4.3.1 Code Completion

在 Eclipse 中打左括号时会立刻加上又括号；打双引号(单引号)时也会立刻加上双引号(单引号)。

4.3.2 Code Assist

在输入程序代码时，例如要打 System.out.println 时，打完类别名称后暂停一会儿，Eclipse 会显示一串建议清单，列出此类别可用的方法和属性，并附上其 Javadoc 批注。可以直接卷动选出然后按 Enter。

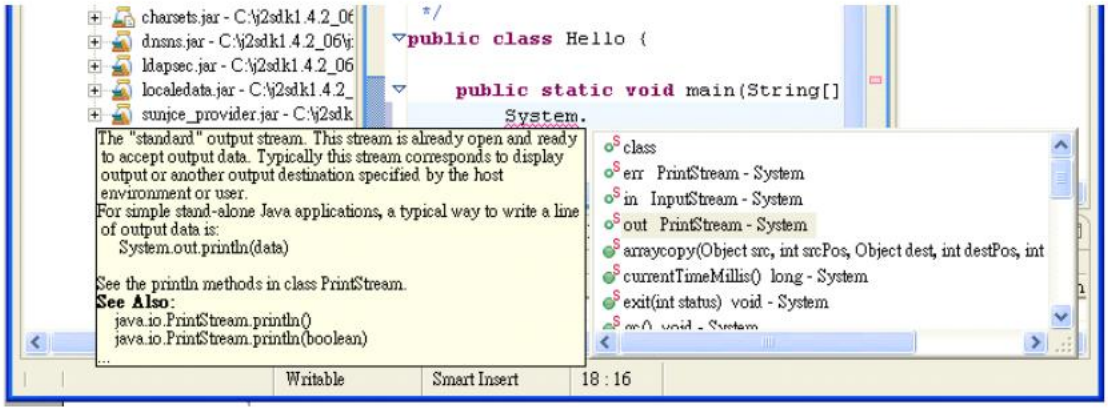


图 4.5

也可以只打类别开头的字母，然后按 Alt - /，一样会显示一串建议清单。

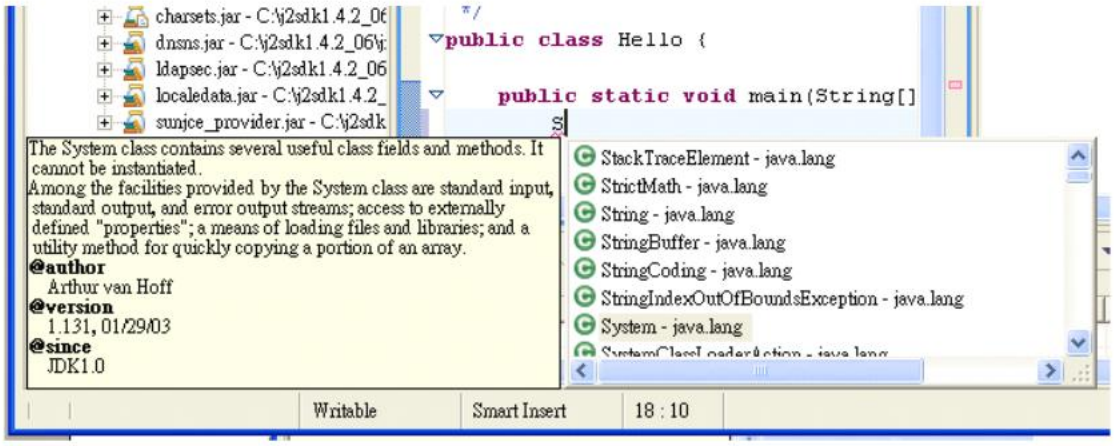


图 4.6

Alt - /这个组合键不仅可以可以显示类别的清单，还可以一并显示已建立的模板程序代码，例如要显示数组的信息，只要先打 for，在按 Alt - /这个组合键，就会显示模板的请单。

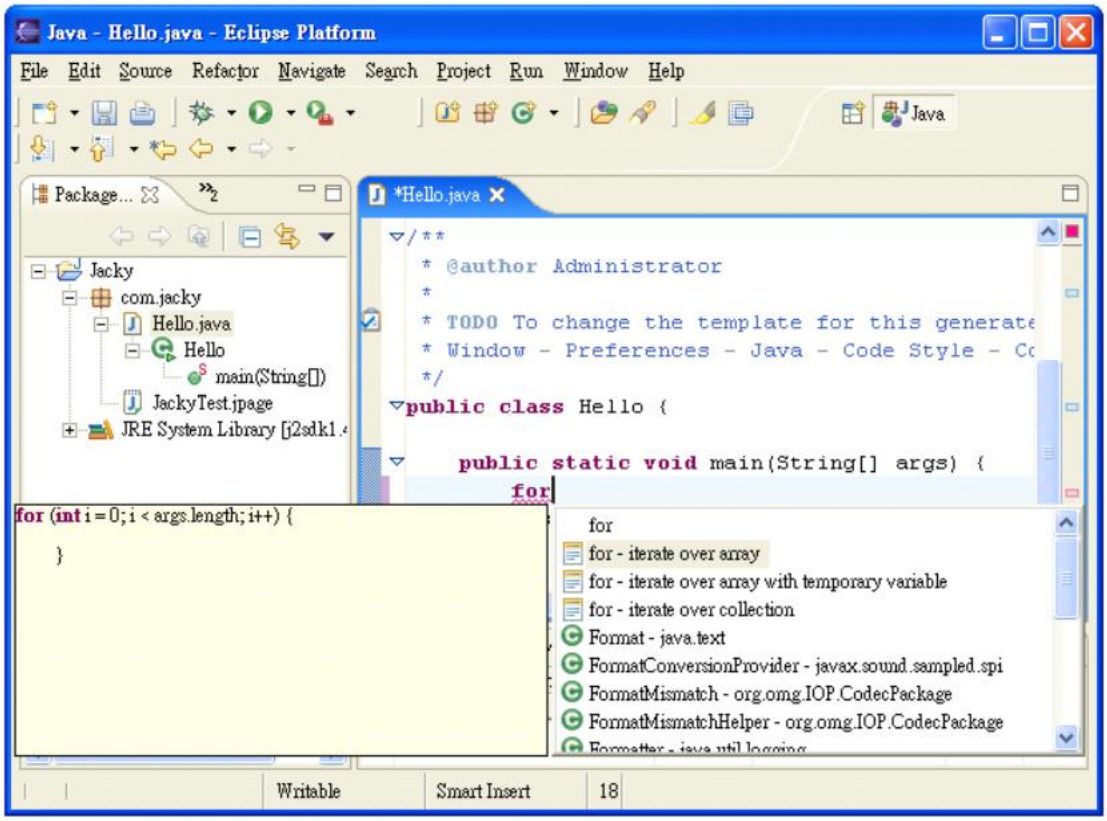


图 4.7

4.4 执行 Java 程序

大多数的程序不需特定的启动组态(Launch Configuration)，首先确定要执行的程序代码在编辑器中有选到(页签变蓝色)，再执行下列步骤：

I. 选单选「Run」→「Run as」→「Java Application」

II. 若有修改过程序，Eclipse 会询问在执行前是否要存档

III. Tasks 试图会多出 Consol 页签并显示程序输出

程序若要传参数、或是要使用其它的 Java Runtim...等等，则需要设定程序启动的相关选项，执行程序前，新增一个启动组态或选用现有的启动组态。

I. 选单选「Run」→「Run」，开启 Run 的设定窗口

- Main 标签用以定义所要启动的类别。请在项目字段中，输入内含所要启动之类别的项目名称，并在主要类别字段中输入主要类别的完整名称。如果想要程序每当在除错模式中启动时，在 main 方法中停止，请勾选 Stop in main 勾选框。

附注：不必指定一个项目，但这样做可以选择预设类别路径、来源查阅路径，以及 JRE。

- 自变量(Arguments)标签用以定义要传递给应用程序与虚拟机器（如果有的话）的自变量。也可以指定已启动应用程序要使用的工作目录。
- JRE 卷标用以定义执行或除错应用程序时所用的 JRE。可以从已定义的 JRE 选取 JRE，或定义新的 JRE。
- 类别路径(Classpath)卷标用以定义在执行或除错应用程序时所用类别文件的位置。依预设，使用者和

bootstrap 类别位置是从相关联项目的建置路径衍生而来。可以在这里置换这些设定。

- 程序文件(Source)卷标用以定义当除错 Java 应用程序时，用来显示程序文件之程序文件的位置。依预设，这些设定是从相关联项目的建置路径衍生而来。可以在这里置换这些设定。
- 环境(Environment)标签会定义在执行 Java 应用程序或者对它进行除错时，所要使用的环境变量值。依预设，这个环境是继承自 Eclipse 执行时期。可以置换或附加至继承的环境。

共享(Common)卷标定义有关启动配置的一般信息。可以选择将启动配置储存在特定档案，以及指定当启动配置启动时，哪些视景将变成作用中。

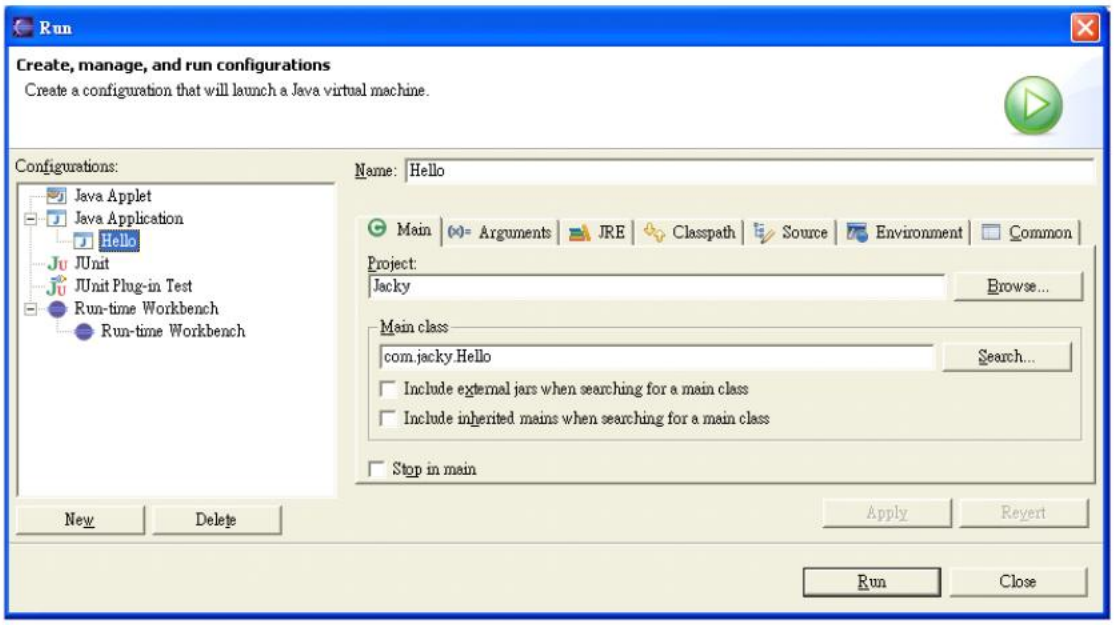


图 4.8

II. 在 Arguments 的页签中输入要传入的值，若是多值的话，用空格键隔开

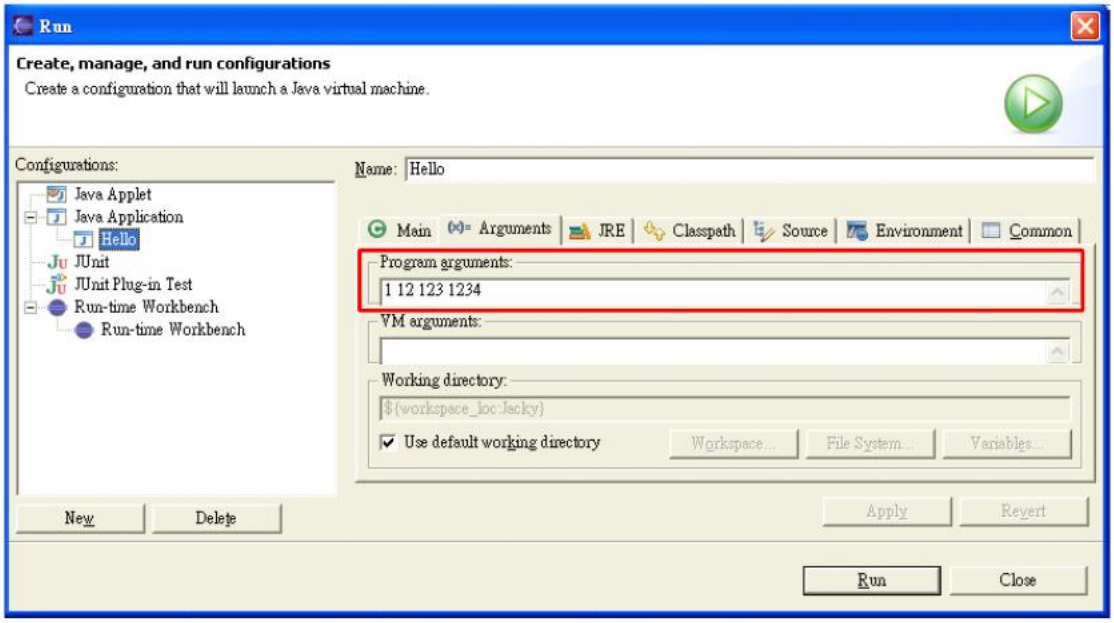


图 4.9

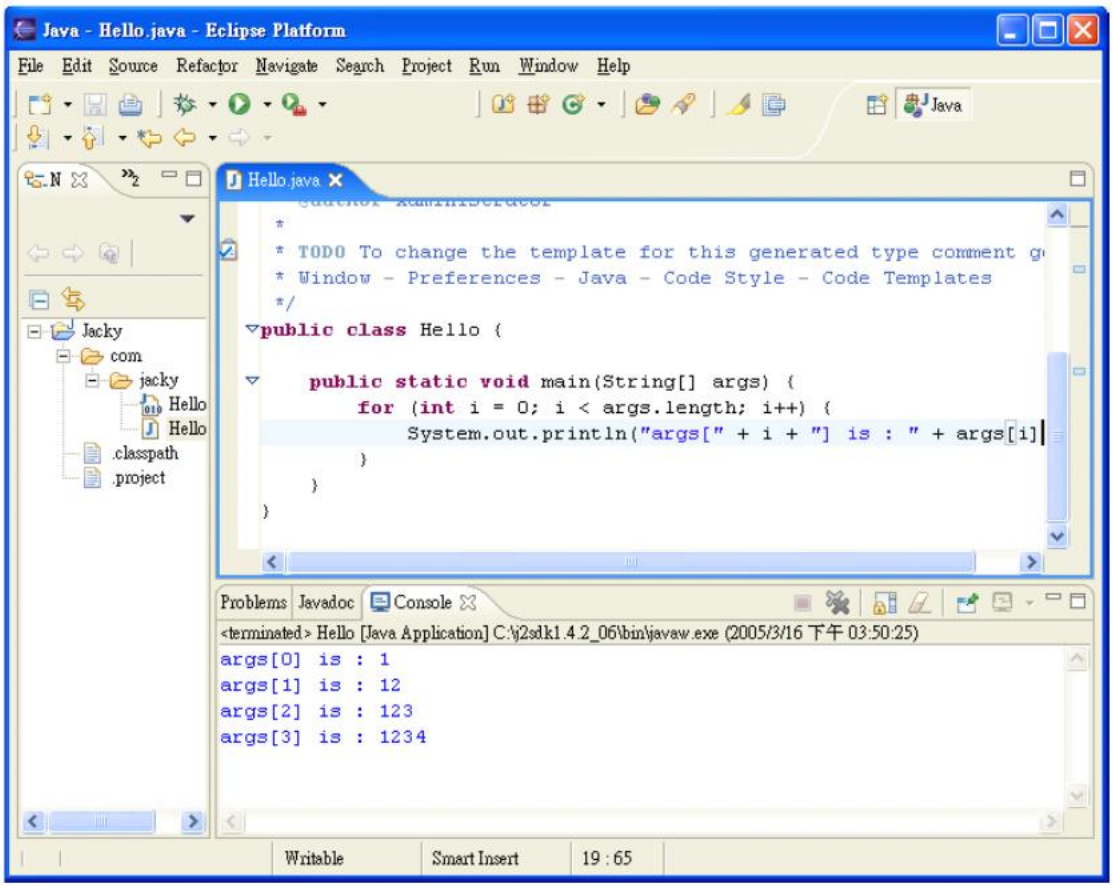


图 4.10

4.5 Java 实时运算簿页面(Java Scrapbook Page)

写程序时可能会些其它的想法，但不知是否可行：多数情况是直接写到程序再来 debug，或是另外写各小程序。Eclipse 提供一种轻巧的替代方式，Java 实时运算簿页面(Java Scrapbook Page)，藉由渐进式编译器，可以在实时运算簿写入任意的 Java 程序代码并执行，不需另写在类别或方法中。

I. 切换至 Java 视景

II. 「File」→「New」→「Other...」→「Java」→「Scrapbook Page」

(在专案上按右键，「New」→「Other...」→「Java」→「Scrapbook Page」)

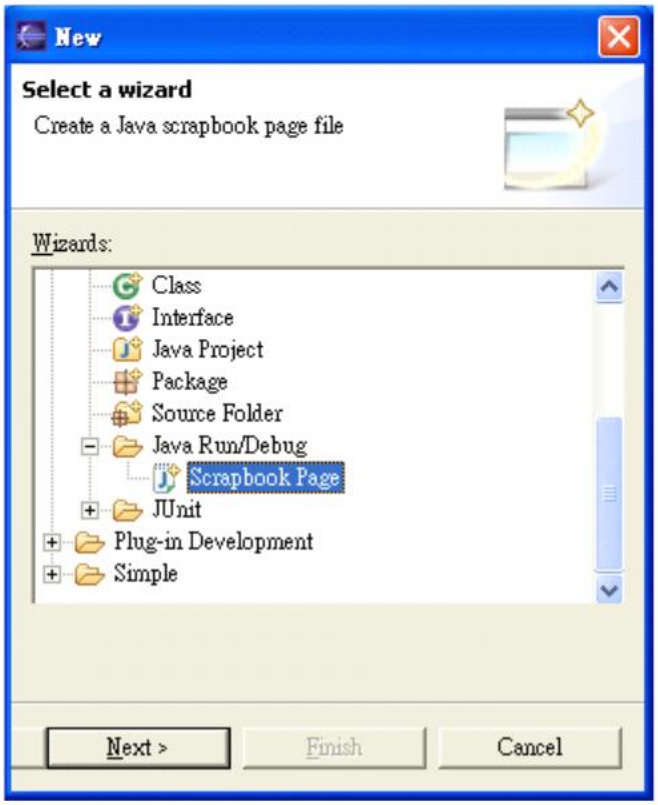


图 4.11

III. 选择要存放的地方

IV. 输入档名

IV. 按下 Finish

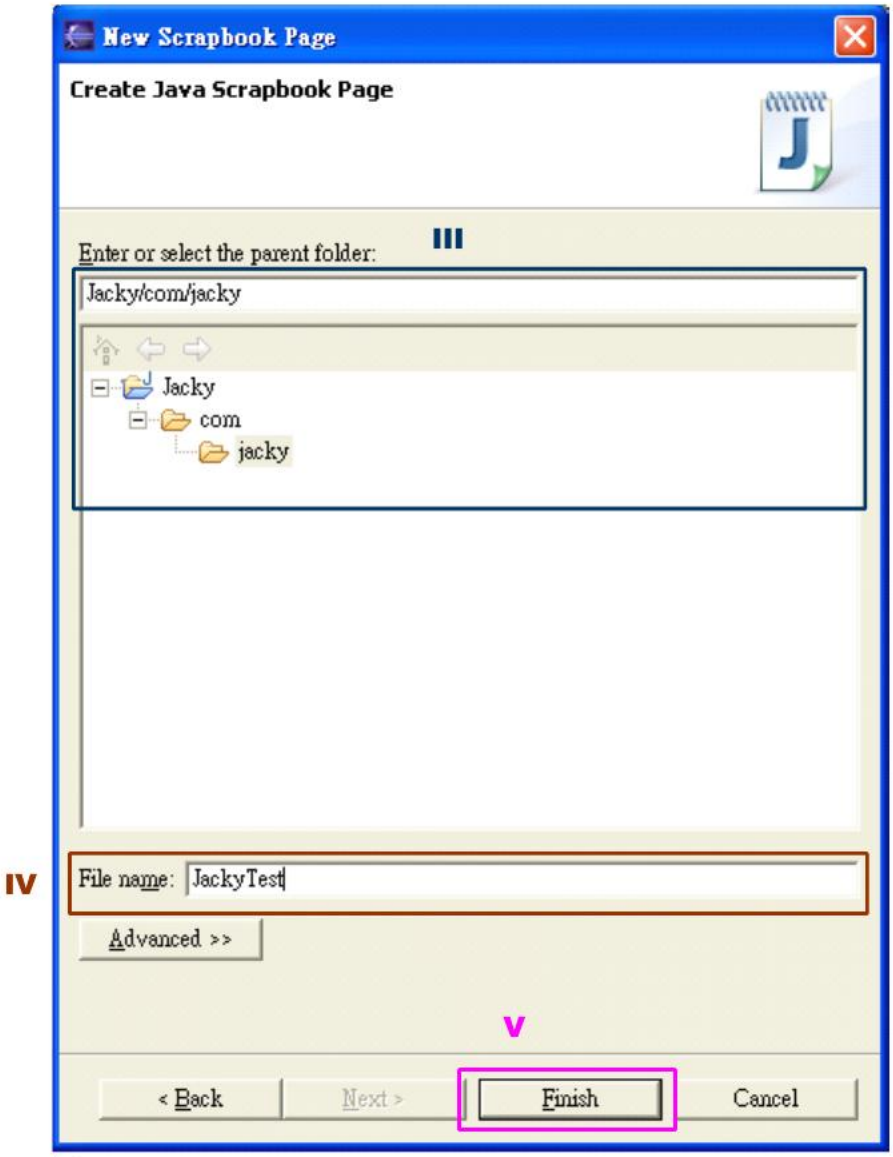


图 4.12

在「Package Explorer」或是「Navigator」视图会显示刚刚建立的 JackyTest.jpage 档案。

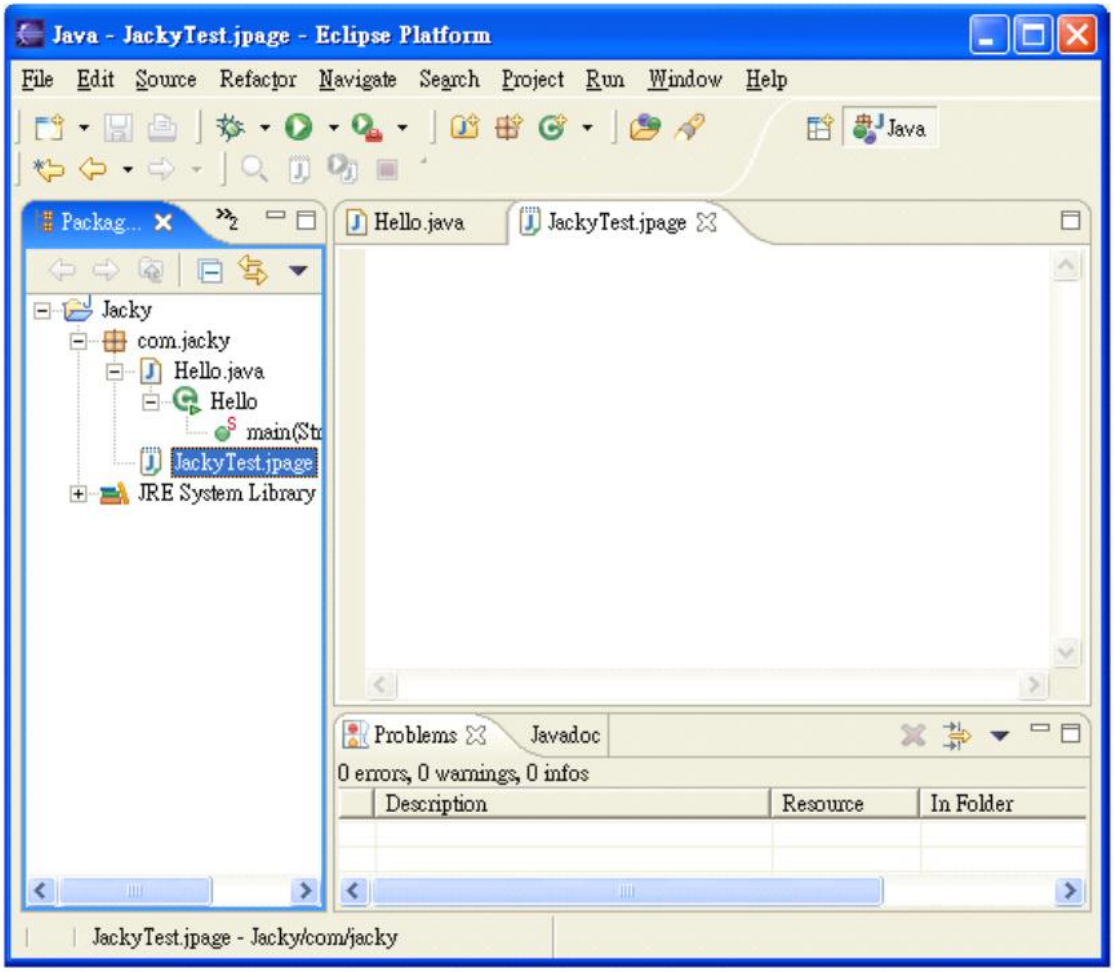


图 4.13

可以输入要测试的 Java 程序代码，例如

```
for (int i = 0; i < 5; i++) {
    System.out.println(Integer.toString(i));
}
```

I. 将这段程序代码反白

II. 在这段程序上按右键，选择 Execute

III. Consol 视图会显示执行的结果

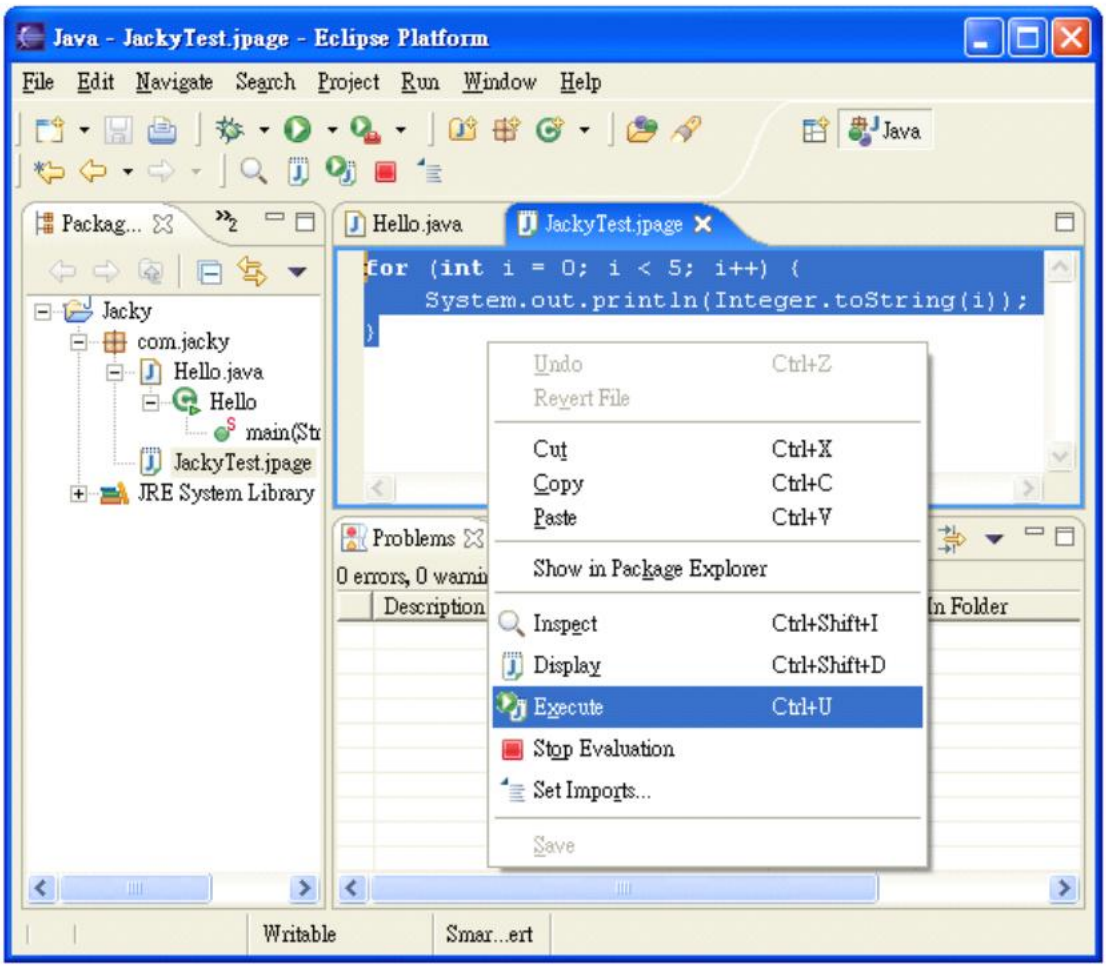


图 4.14

若需要汇入套件；

I. 在编辑器窗口内按右键，选 Set Import...

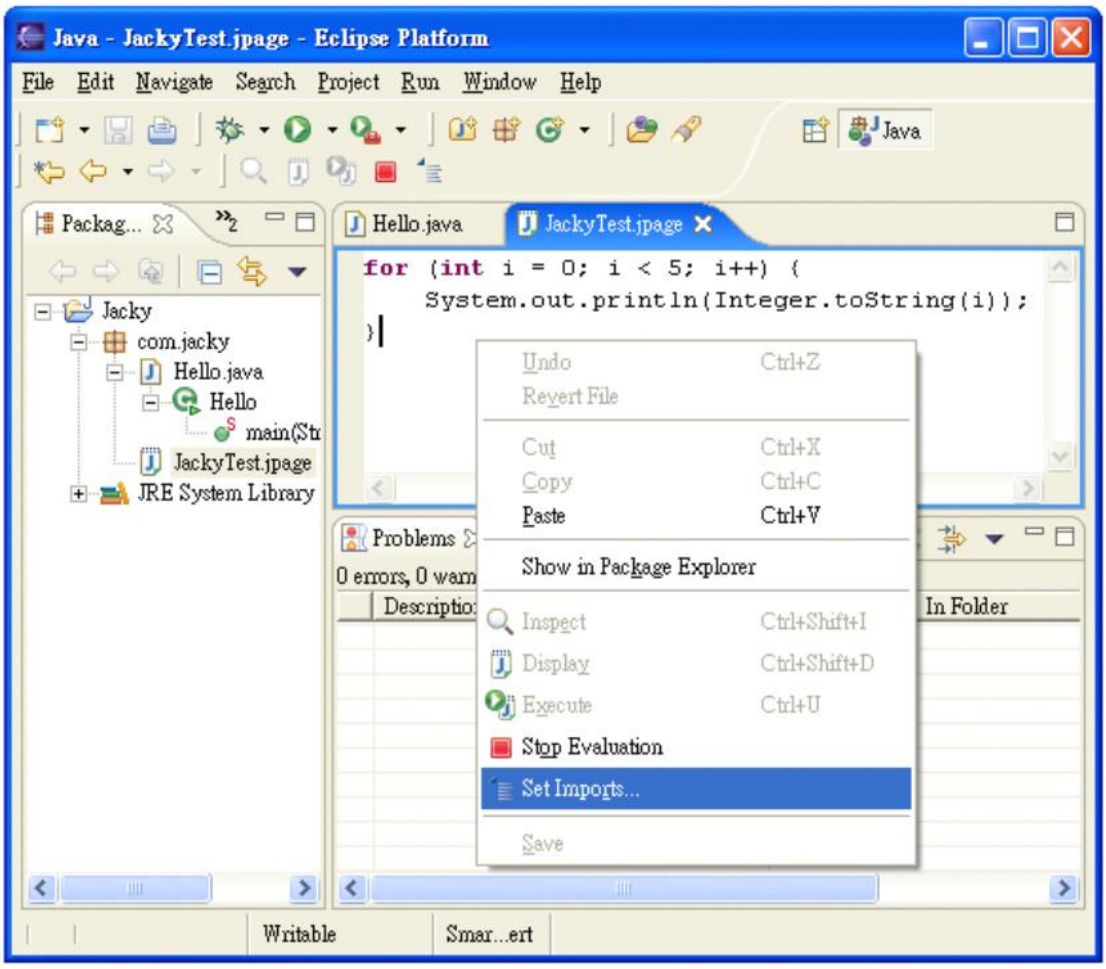


图 4.15

II. 在 Java Snippet Imports 窗口中，按 Add Packages 的按钮

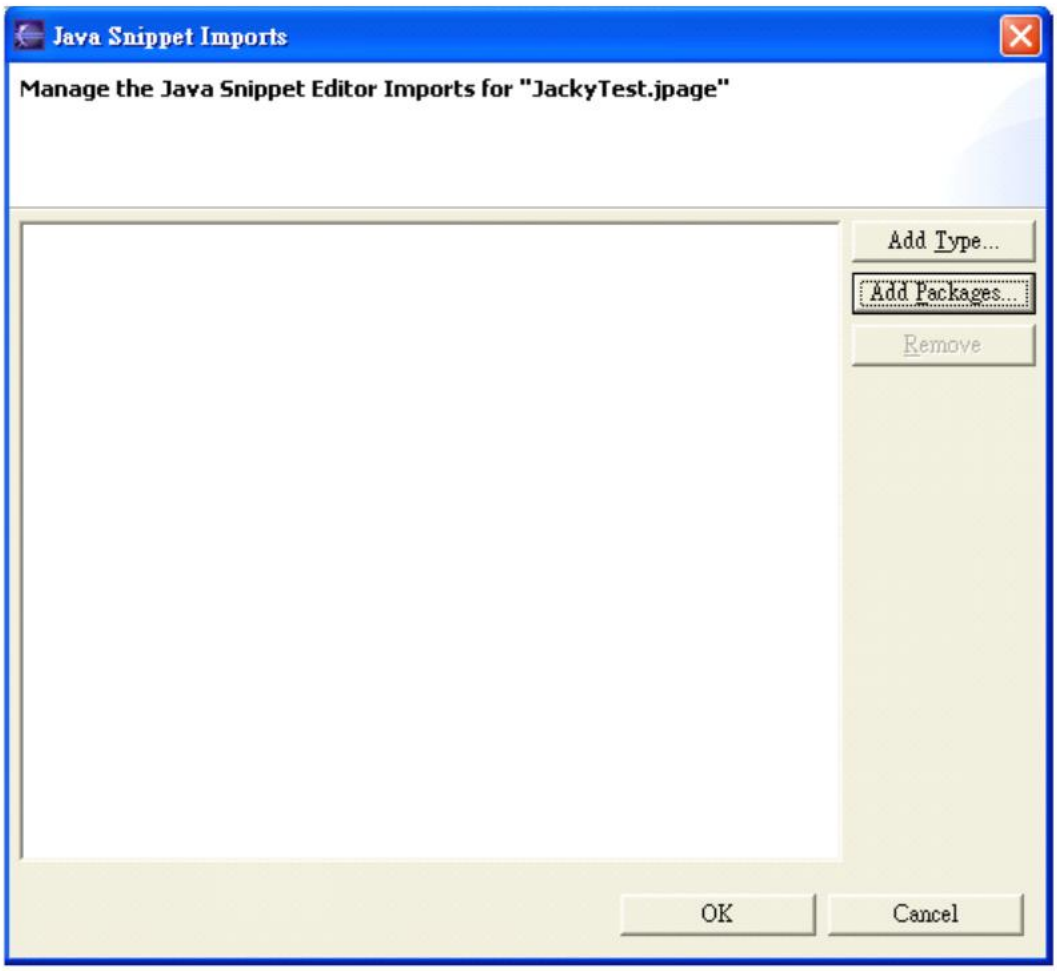


图 4.16

III. 在 Add Packages as Imports 的窗口中，挑选要 import 的 package

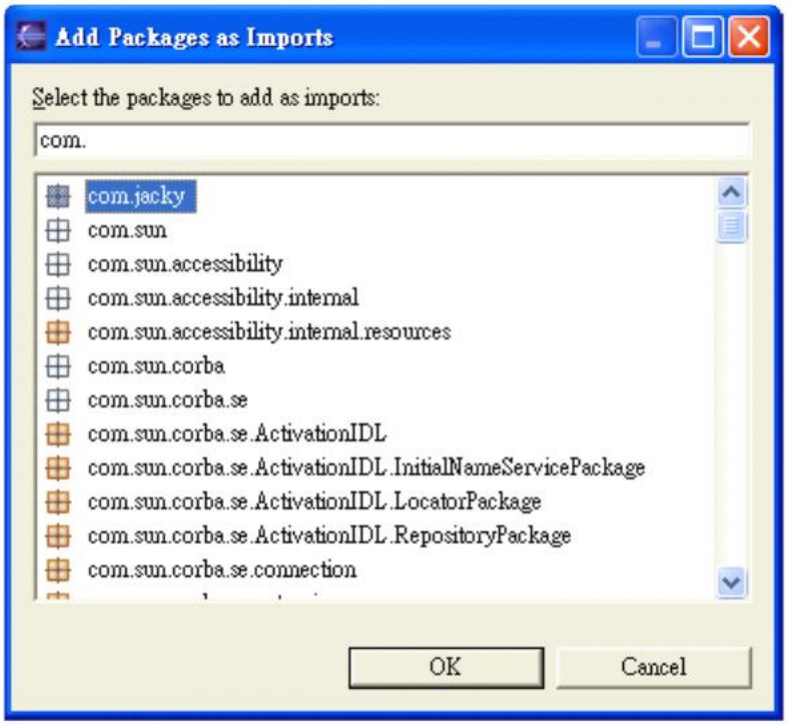


图 4.17

IV. 按 OK

4.6 自订开发环境

4.6.1 程序代码格式

在「Window」→「Preferences」→「Java」→「Code Style」→「Code Formatter」

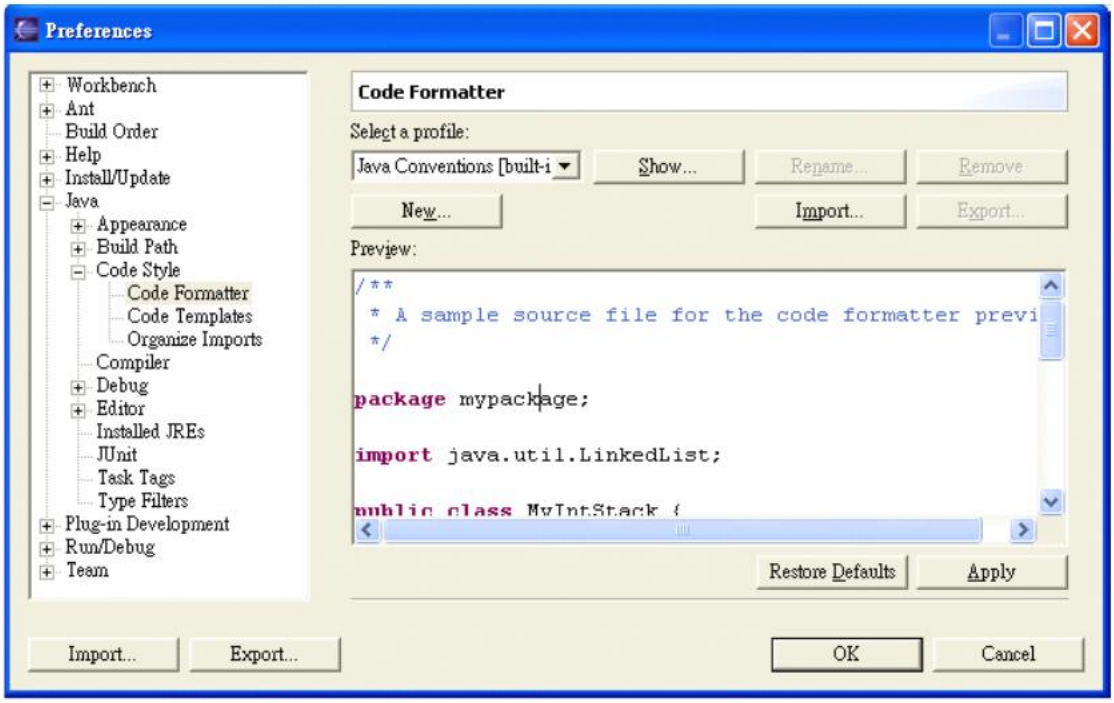


图 4.18

按 Show 的按钮，出现 Show Profile 的窗口，里面的各个页签，可以设定 Java Code Style

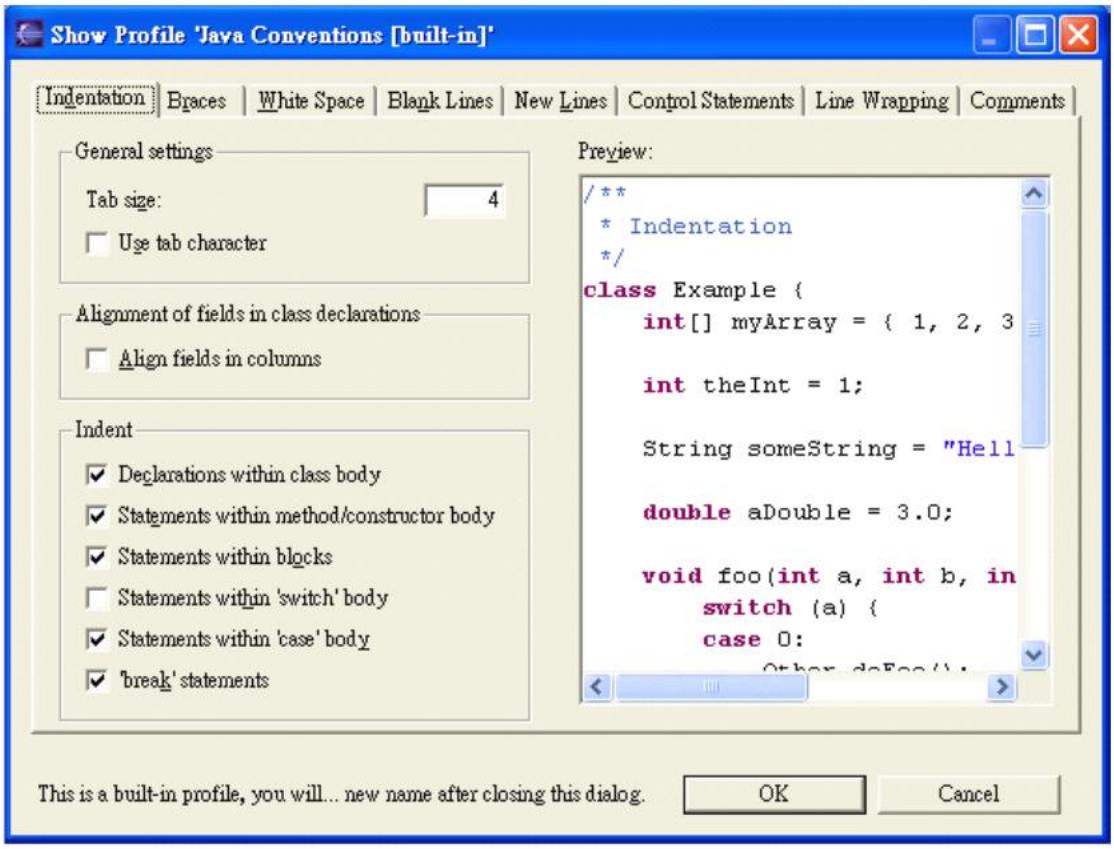


图 4.19

设定完成后，可以 Export 成一个档案；以利下次设定 Java Code Style 时，可以利用 Import 的方式，产生一致的程序风格。

4.6.2 程序代码产生模板

在开发 Java 时，可以把常用的流程控制建构式或是常用到的 API，建立成一个模板，可以加速程序开发。接下来以 System.out.println()为例子，来说明如何建立模板：

I. 「Window」→「Preferences」→「Java」→「Editor」→「Templates」

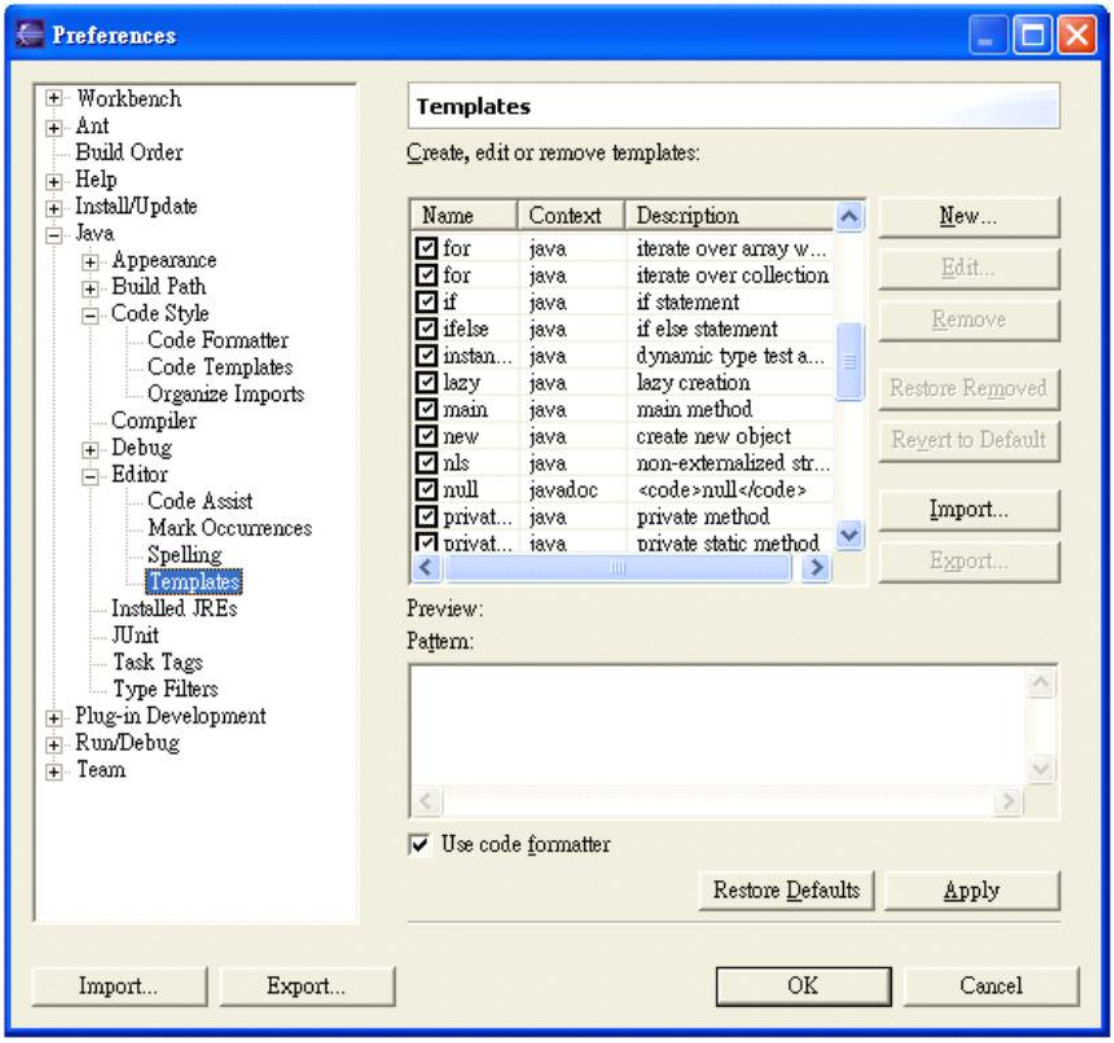


图 4.20

II. 在 Preferences 窗口按 New 的按钮

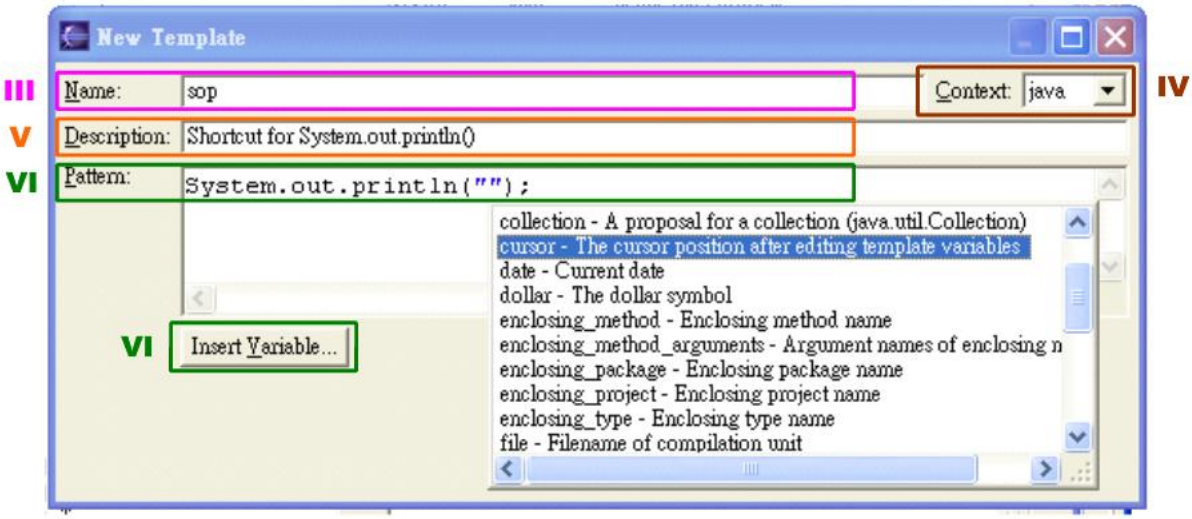


图 4.21

III. 在 Name 的字段输入自己想要的名称

IV. Context 选 java

V. 在 Description 的字段输入简短的说明

VI. 在 Pattern 的字段输入 System.out.println("")后；把光标移到两个双引号的中间，再按下面 Insert Variable 的按钮，选择 cursor

VII. 再按两次 OK

这里的${cursor}变量代表插入模板的程序代码后，光标所在的位置。

使用此新模板，打 s(或是 sop)再按 Alt - /，从清单中选 sop，再按 Enter 即可。

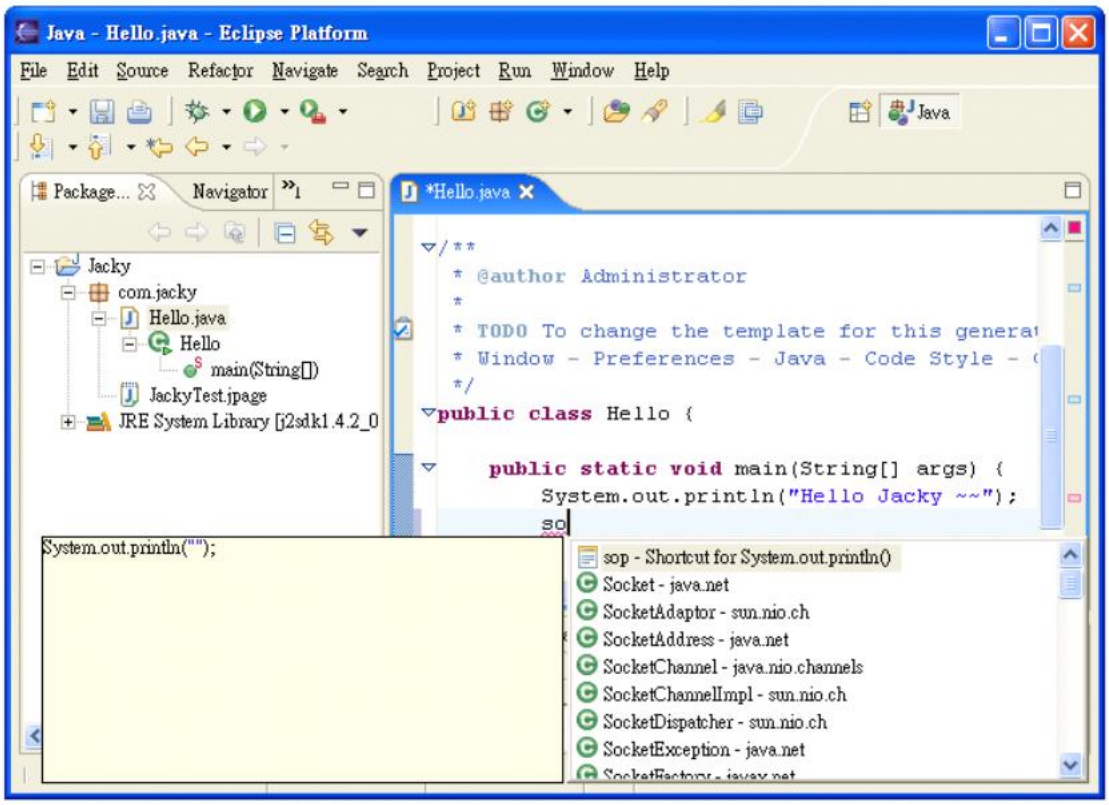


图 4.22

4.6.3 Javadoc 批注

编辑新增类别后出现的文字。移除" To change the template for this generated..."这段前置文字，并自行扩充 Javadoc 批注。

I. 「Window」→「Preferences」→「Java」→「Code Style」→「Code Templates」

II. 选右边画面的「Code」→「New Java files」，按 Edit 按钮

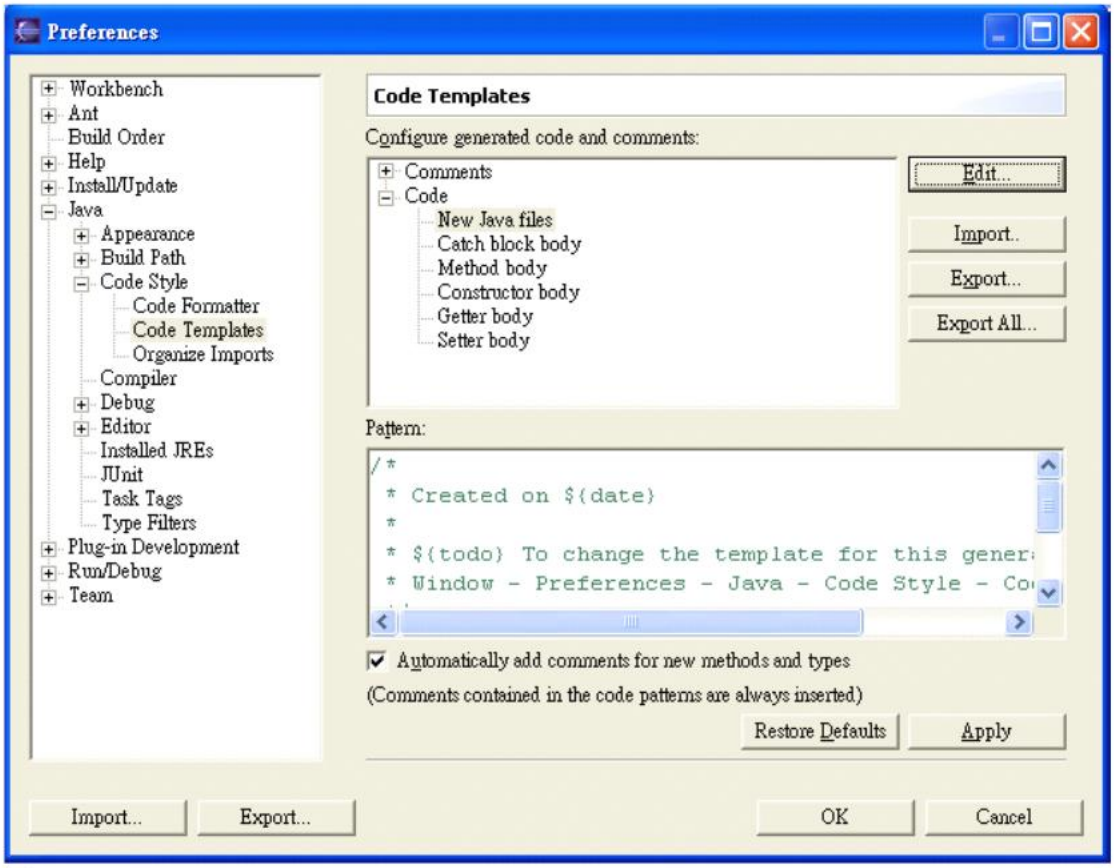


图 4.23

III. 修改成需要的格式

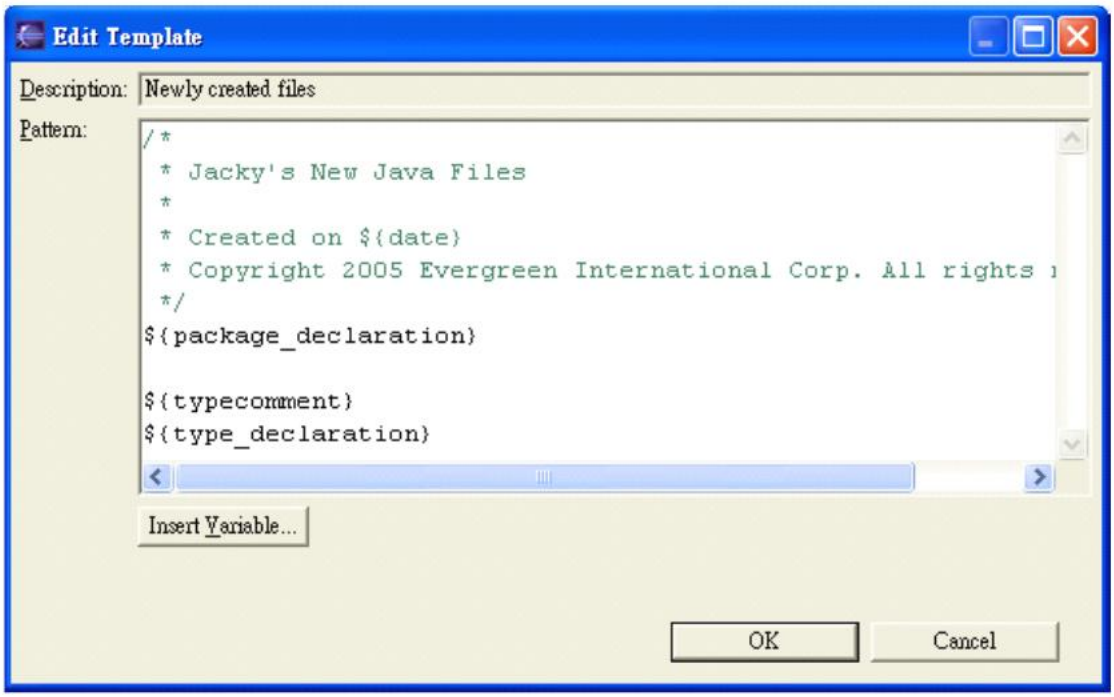


图 4.24

IV. 按 OK

除了 New Java file 的模板外，还需要修改另一个模版-类型批注(Typecomment)。

I. 「Window」→「Preferences」→「Java」→「Code Style」→「Code Templates」

II. 选右边画面的「Comments」→「Types」，按 Edit 按钮

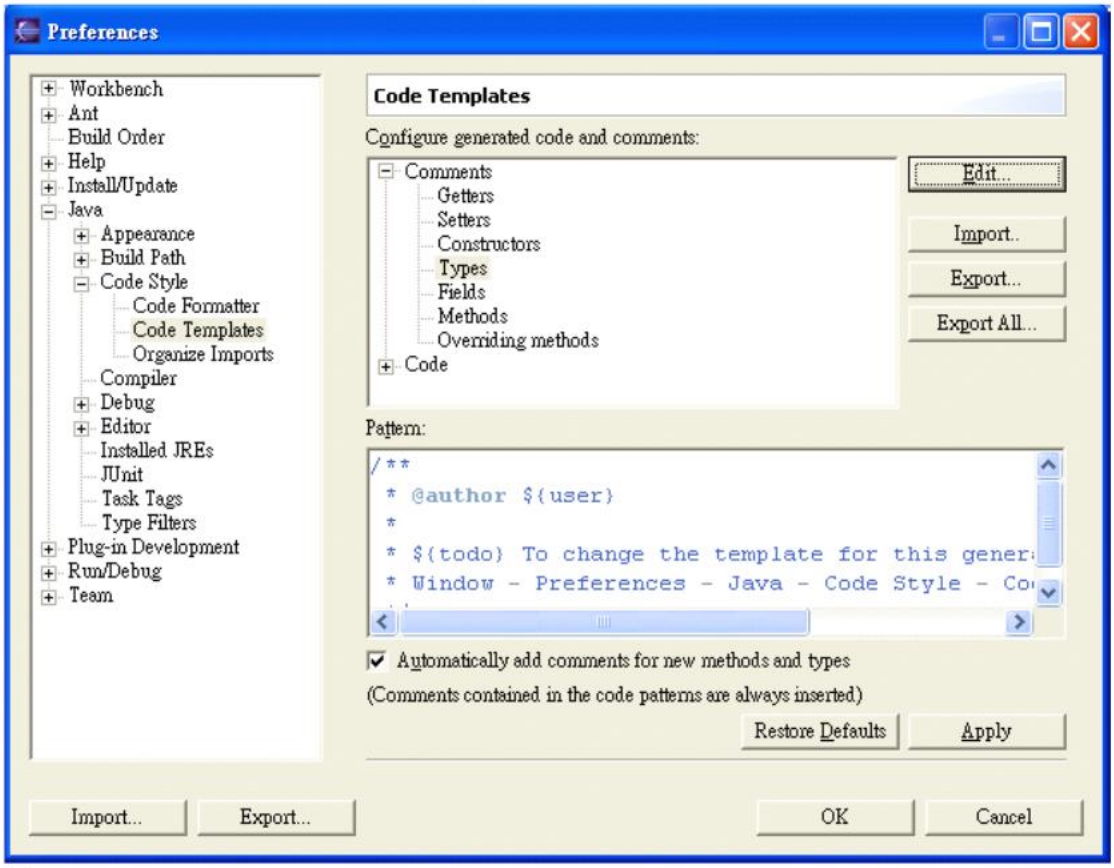


图 4.25

III. 修改成需要的格式

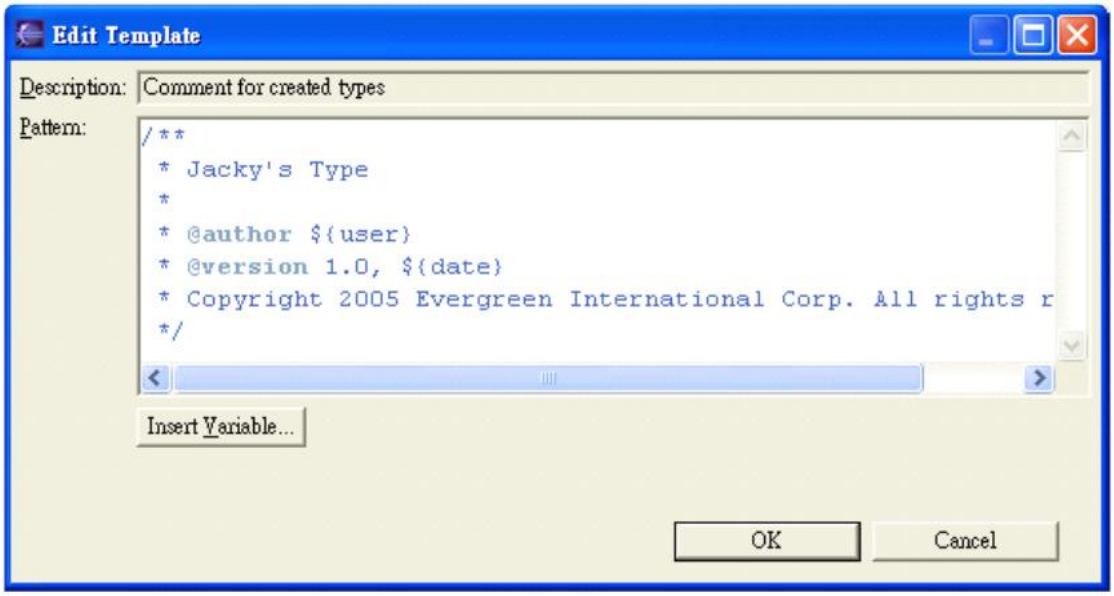


图 4.26

IV. 按 OK

往后新增的类别档案，就会套用现在批注。

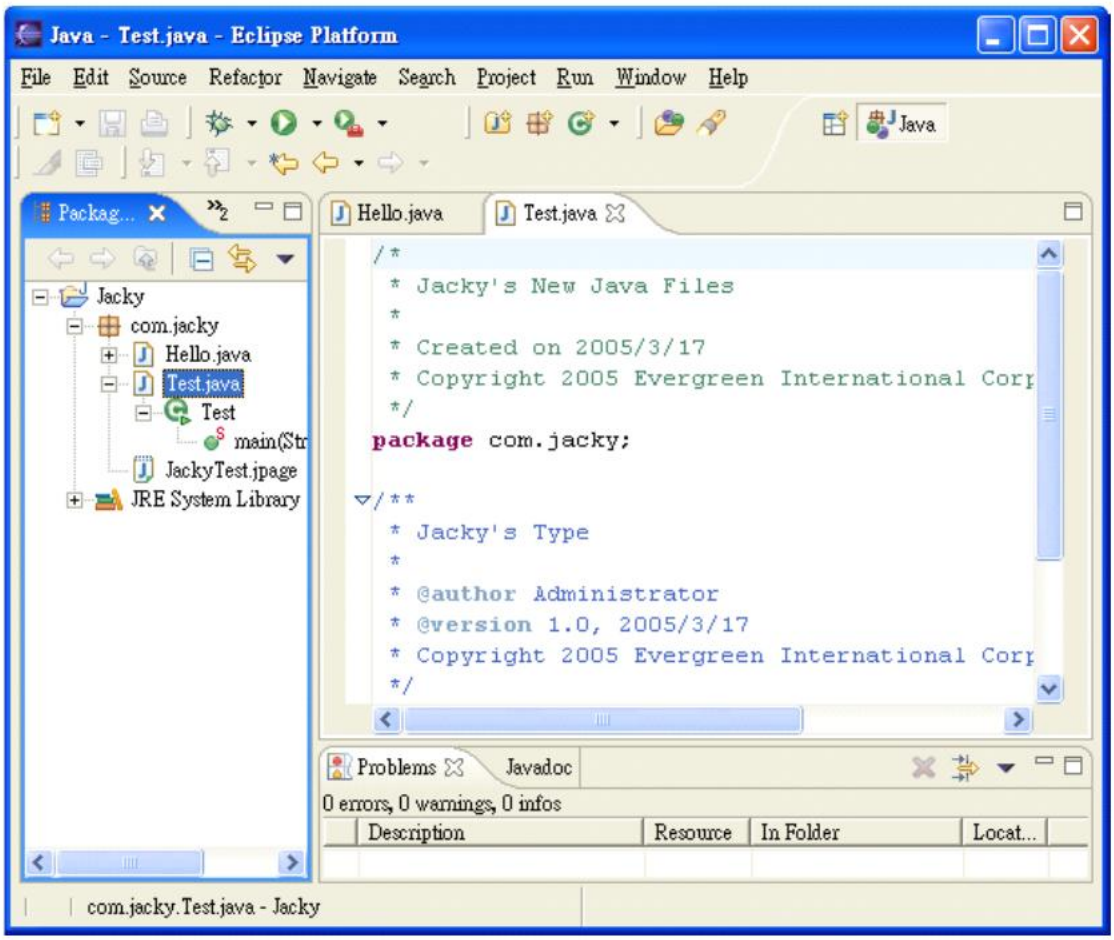


图 4.27

Javadoc 也可以产生模板，做法跟 4.6.2 程序代码产生模板类似，差别在于 Context 改选 javadoc。

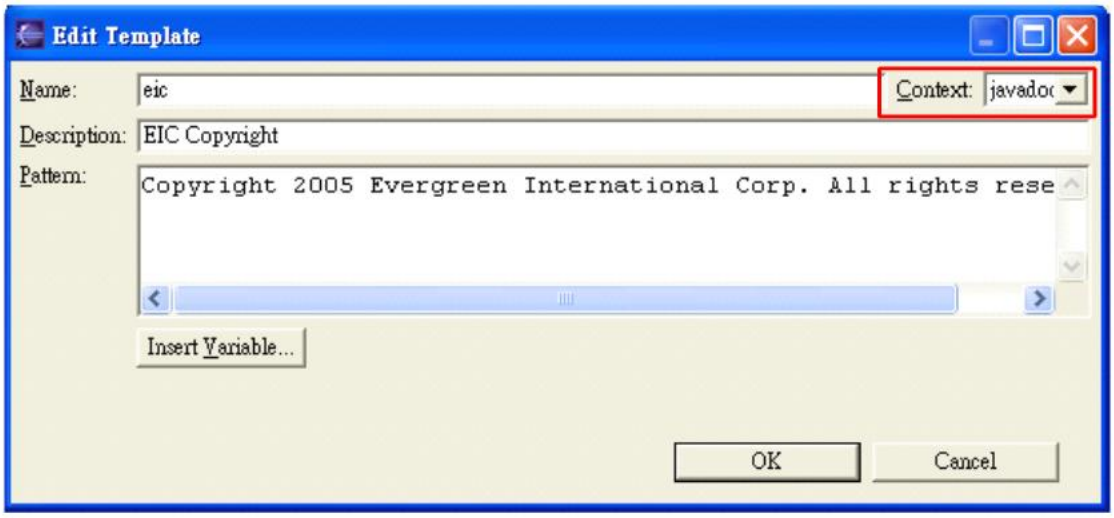


图 4.28

在程序批注的地方，打 eic 再按 Alt - /，就可以出现清单可以选择。

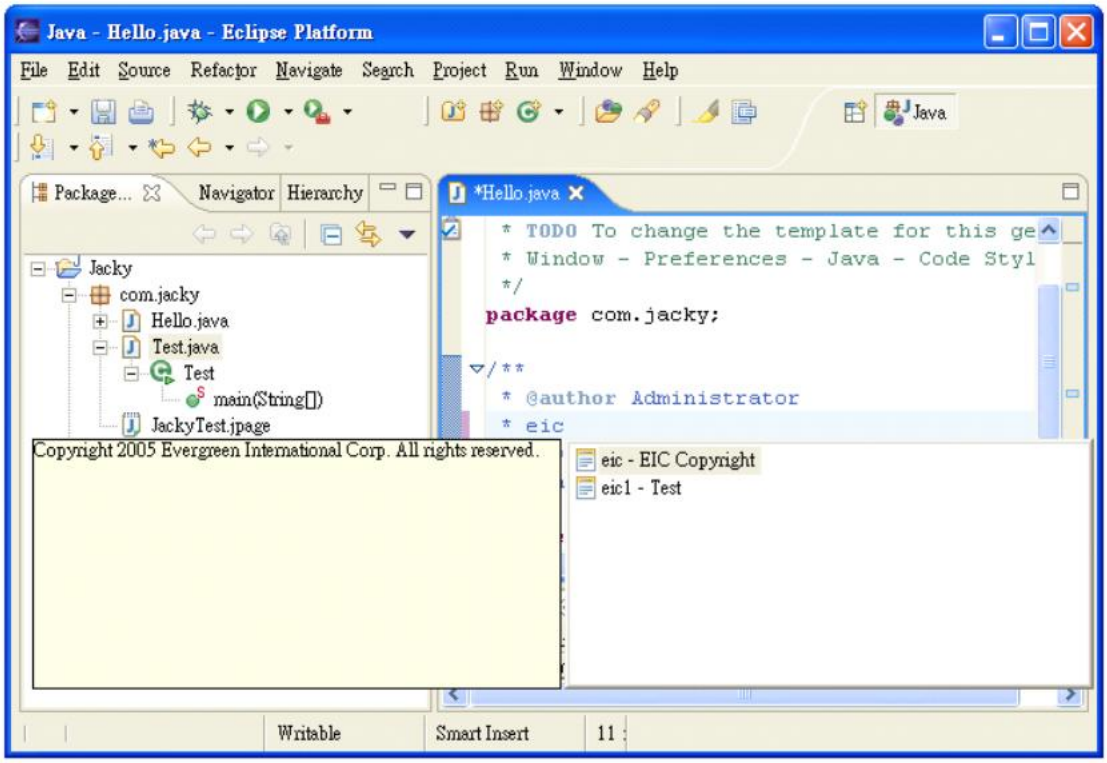


图 4.29

4.7 产生 getter 与 setter

Java 编辑器可以为编译单元内的类型字段，产生存取元(accessors，也就是 getter 和 setter 的 method)。

I. 「Source」→「Generate Getter and Setter...」

(或是在 Java 编辑器按右键，「Source」→「Generate Getter and Setter...」)

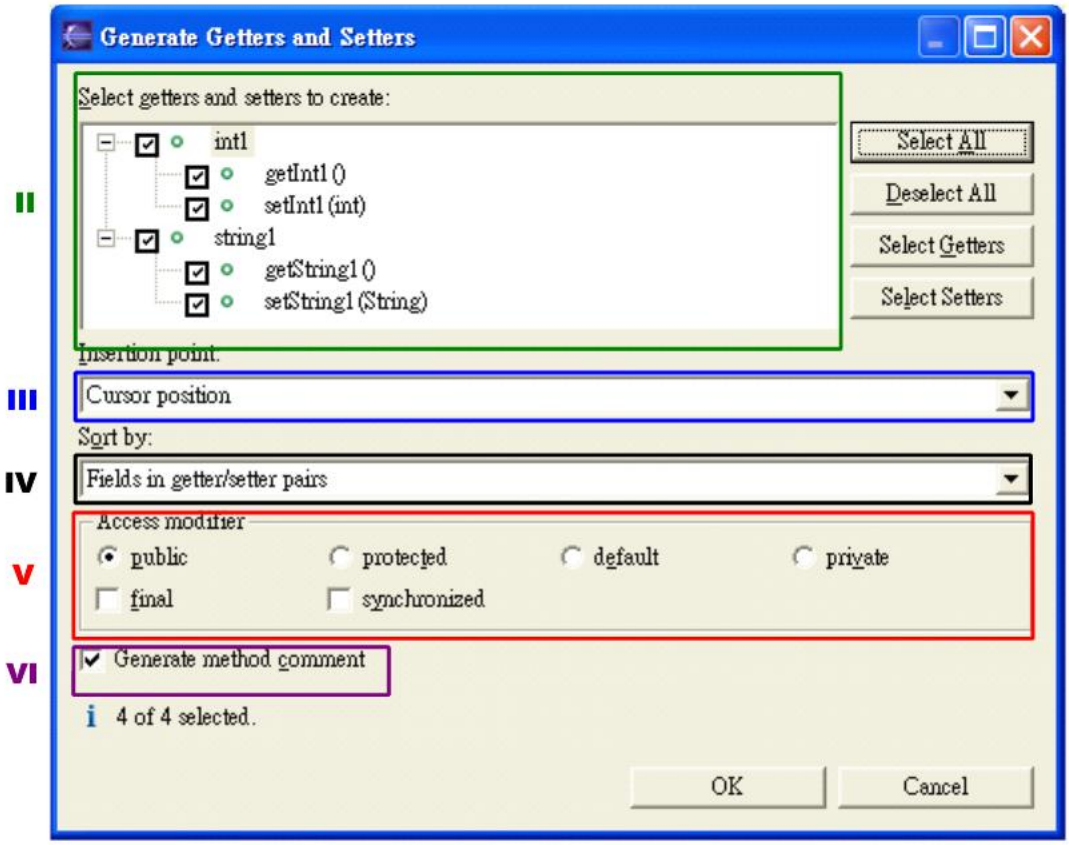


图 4.30

II. 挑选哪些需要建立 getter 和 setter 的 method

III. 选择 method 要建立的地方

IV. 排序的方式

V. 选择 Access modifier

VI. 选择是否需要建立批注

VII. 按 OK

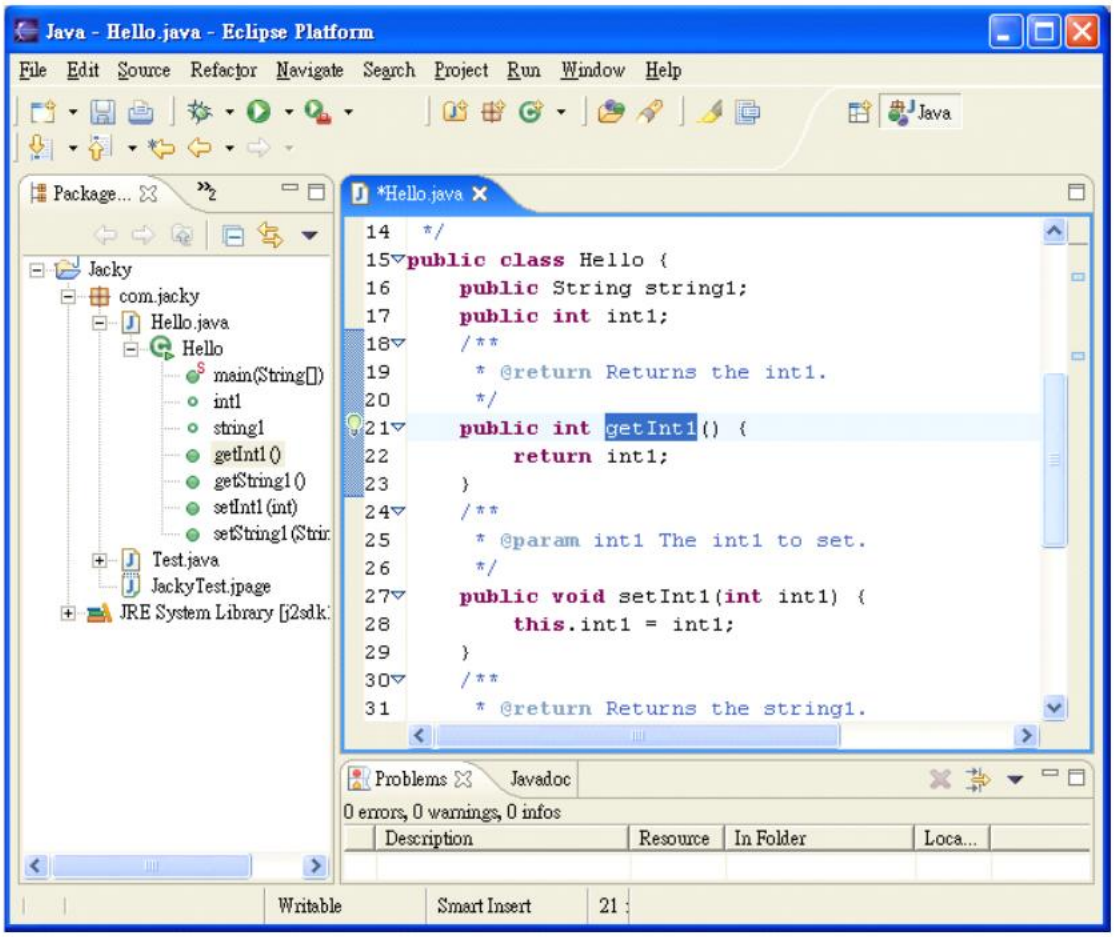


图 4.31

4.8 建立 JAR 档案

4.8.1 建立新的 JAR 档案

如果要在工作台中建立新 JAR 档，请执行下列动作：

I. 在「Package Explorer」中，可以选择性地预选一或多个要汇出的 Java 元素。(在步骤 IV 中，这些会在 JAR Package Specification 精灵页面中自动选出。)

II. 从快速菜单或从菜单列的 File 菜单，选取 Export。

III. 选取 JAR file，然后按一下 Next。

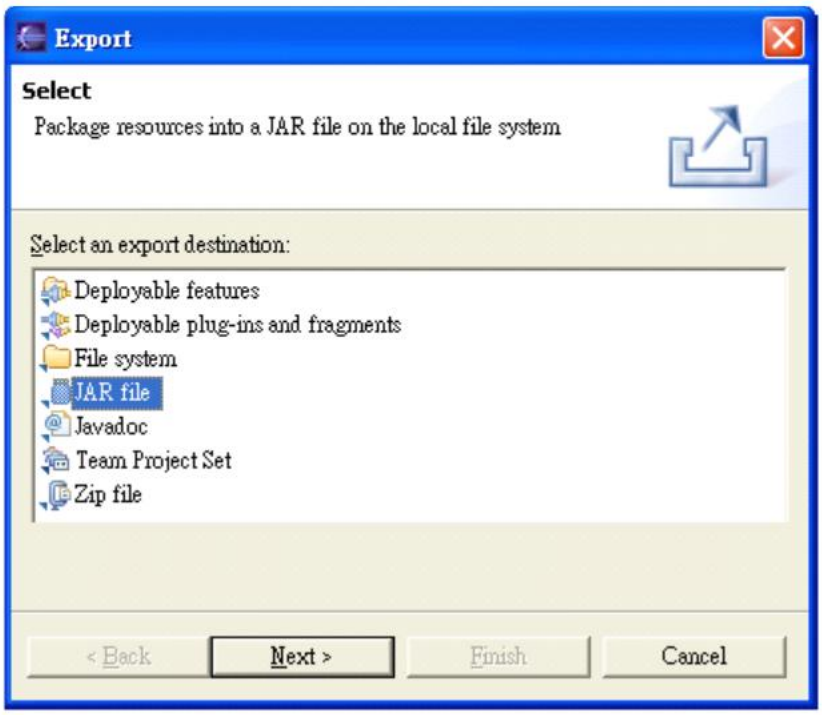


图 4.32

IV. 在 JAR Package Specification 页面的 Select the resources to export 字段中，选取要汇出的资源。

V. 选取适当的勾选框，以指出想 Export generated class files and resourcess 或 Export java source files and resources。**附注：**这两种情况皆会汇出所选的资源。

VI. 在 Select the export destination 字段中，输入或按一下 Browse 以选取 JAR 文件的位置。

VII. 选取或清除 Compress the contents of the JAR file 勾选框。

VIII. 选取或清除 Overwrite existing files without warning 勾选框。如果清除这个勾选框，则会提示确认是否要更换每一个将被改写的档案。

IX. **附注：**在撰写 JAR 檔、JAR 说明与 Manifest 档时，会套用改写选项。

X. 有两项选择：

- 按一下 Finish 来立即建立 JAR 檔。
- 按一下 Next，使用「JAR 套装选项」页面，以设定进阶选项，建立 JAR 说明，或变更预设 manifest。

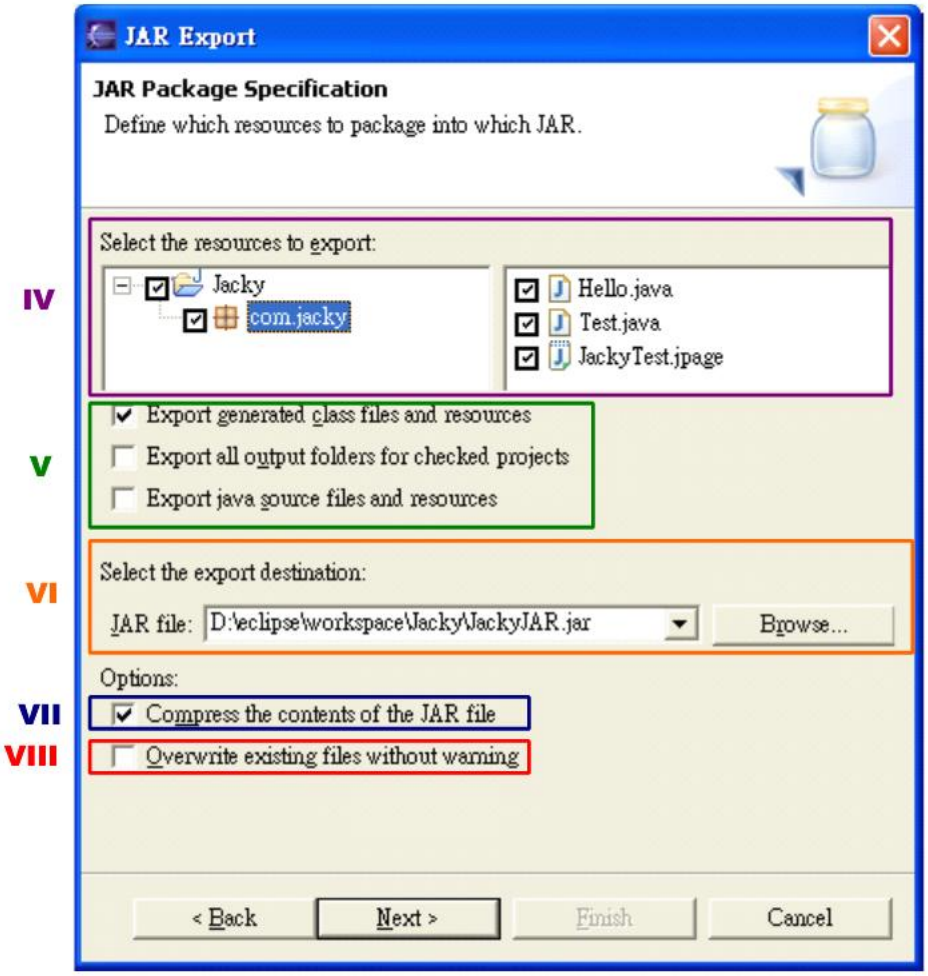


图 4.33

4.8.2 设定进阶选项

I. 遵循建立 JAR 文件的程序进行，但在最后一个步骤中按一下 Next，以移至「JAR 套装选项」页面 。

II. 如果想储存 JAR 档说明，请选取 Save the description of this JAR in the workspace 勾选框。

III. 编译器能产生 CLASS 文件，即使程序文件中有错误。可以选择排除内含编译错误的 CLASS 文件（但非程序文件）。如果有启用报告特性的话，则结束时将会报告这些档案。

IV. 可以选择排除内含编译警告的 CLASS 文件（但非程序文件）。在结束时将会报告这些档案。

附注：这个选项不会自动排除内含编译错误的类别档。

V. 可以选择包含来源数据夹路径，方法是选取 Create source folder structure 勾选框。

VI. 如果希望让汇出作业在建立 JAR 文件前先执行建置，请选取 Build projects if not built automatically 勾选框。

VII. 按一下 Finish，以立即建立 JAR 檔；如果想变更预设 manifest，请按一下 Next。

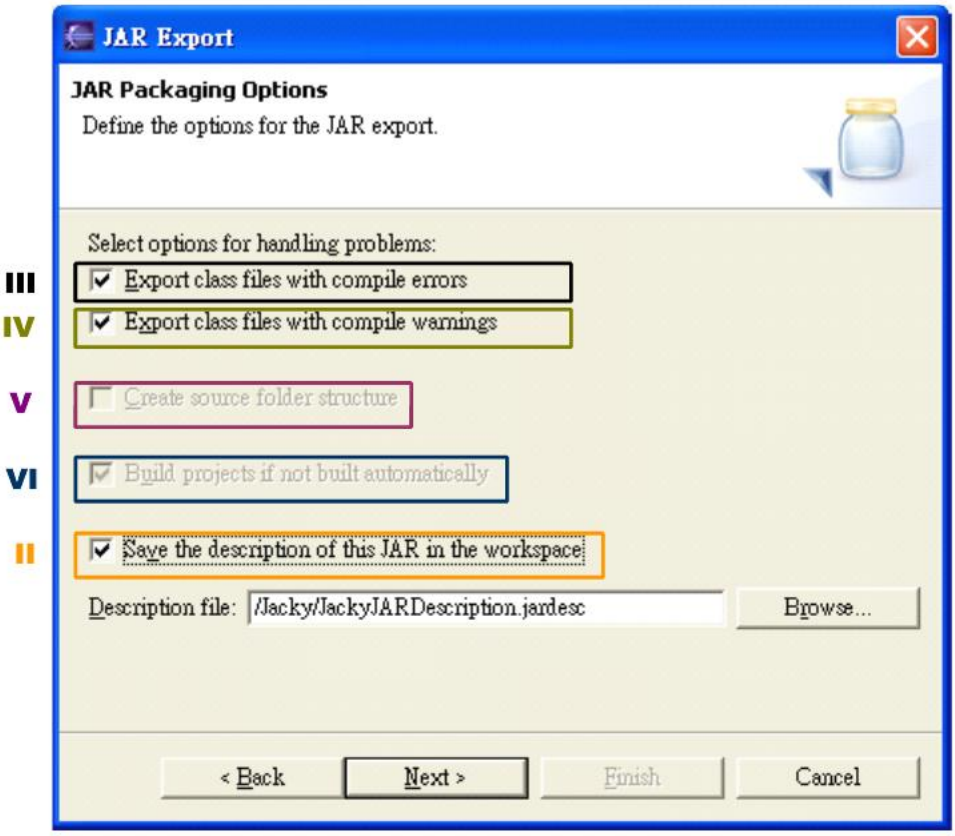


图 4.34

4.8.3 定义 JAR 檔的 manifest

可以直接在精灵中定义 JAR 檔 Manifest 的重要部分，或使用已存在于工作台中的 Manifest 檔。

建立新 Manifest

I. 遵循建立 JAR 文件的程序进行，但在最后一个步骤中按一下 Next，以移至「JAR 套装选项」页面 。

II. 设定任何要设定的进阶选项，再按一下 Next，移至「JAR manifest 规格」页面 中。

III. 如果尚未选取，请按一下 Generate the manifest file 按钮。

IV. 这时可以选择将 Manifest 储存在工作台中。这会储存 Manifest 以便日后使用。按一下 Save the manifest in the workspace，然后按一下 Manifest file 字段旁的 Browse，以指定 Manifest 的路径与文件名。

V. 如果在前一步骤中决定储存 Manifest 档，并在先前的精灵页面中选择储存 JAR 说明，可以选择在 JAR 说明中重复使用 Manifest（做法是选取 Reuse and save the manifest in the workspace 勾选框）。这表示当从 JAR 说明重建 JAR 档时，将会使用所储存的档案。如果想先修改或更换 Manifest 档，然后再从说明重建 JAR 档，请善用这个选项。

VI. 可以选择密封 JAR，以及选择性地将某些套件排除在密封之外，或指定密封套件清单。依预设，不会进行任何密封。

VII. 按一下 Main Class 字段旁的 Browse 按钮来指定应用程序的进入点。

附注：如果的类别不在清单中，表示在一开始时忘了选取它。

VIII. 按一下 Finish。这会建立 JAR，并选择性地建立 JAR 说明与 Manifest 檔。

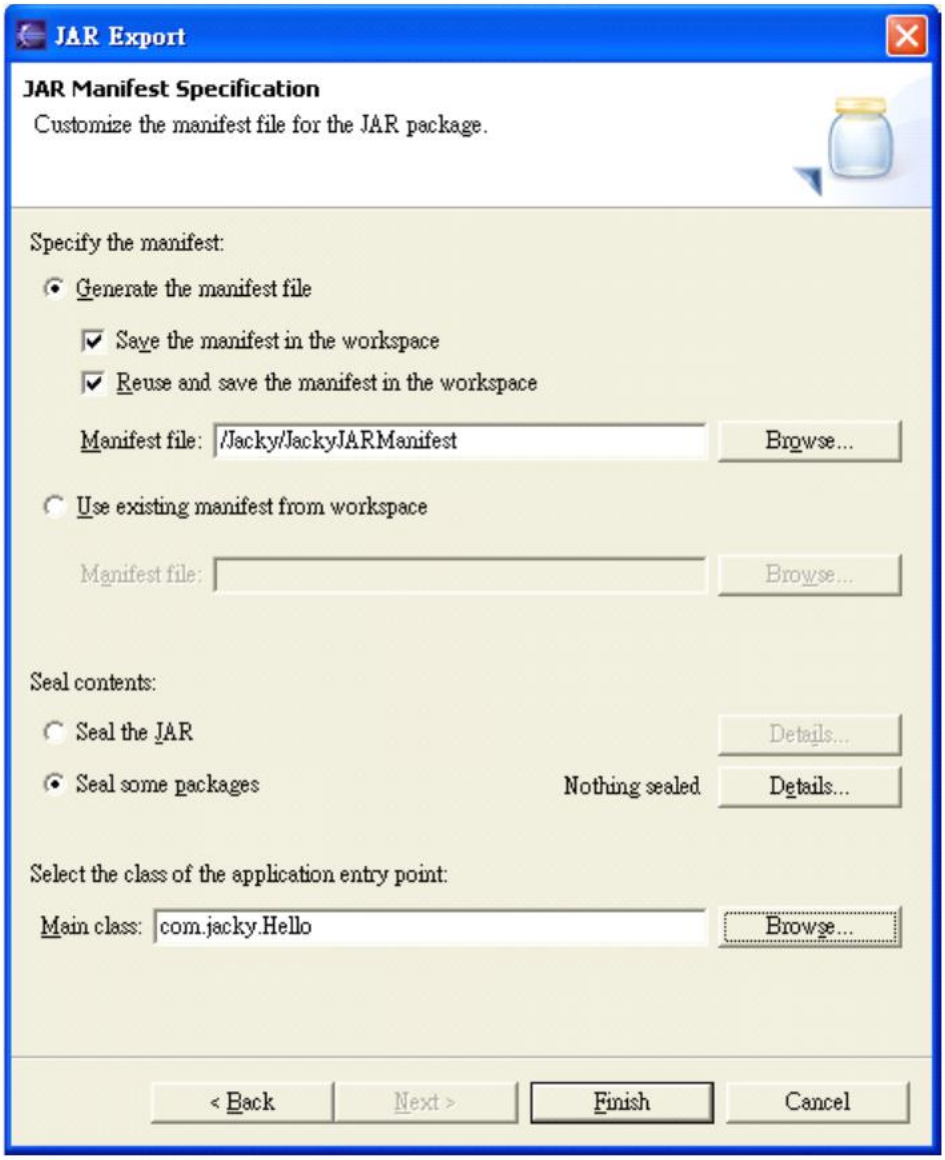


图 4.35

使用现有的 manifest

可以使用已存在于工作台中的现有 Manifest 檔。

I. 遵循建立 JAR 文件的程序进行，但在最后一个步骤中按一下 Next，以移至「JAR 套装选项」页面 。

II. 设定任何要设定的进阶选项，再按一下 Next，移至「JAR manifest 规格」页面中。

III. 按一下 Use existing manifest from workspace 圆钮。

IV. 按一下 Browse 按钮来从工作台中选取 Manifest 檔。

V. 按一下 Finish。这会建立 JAR，并选择性地建立 JAR 说明。

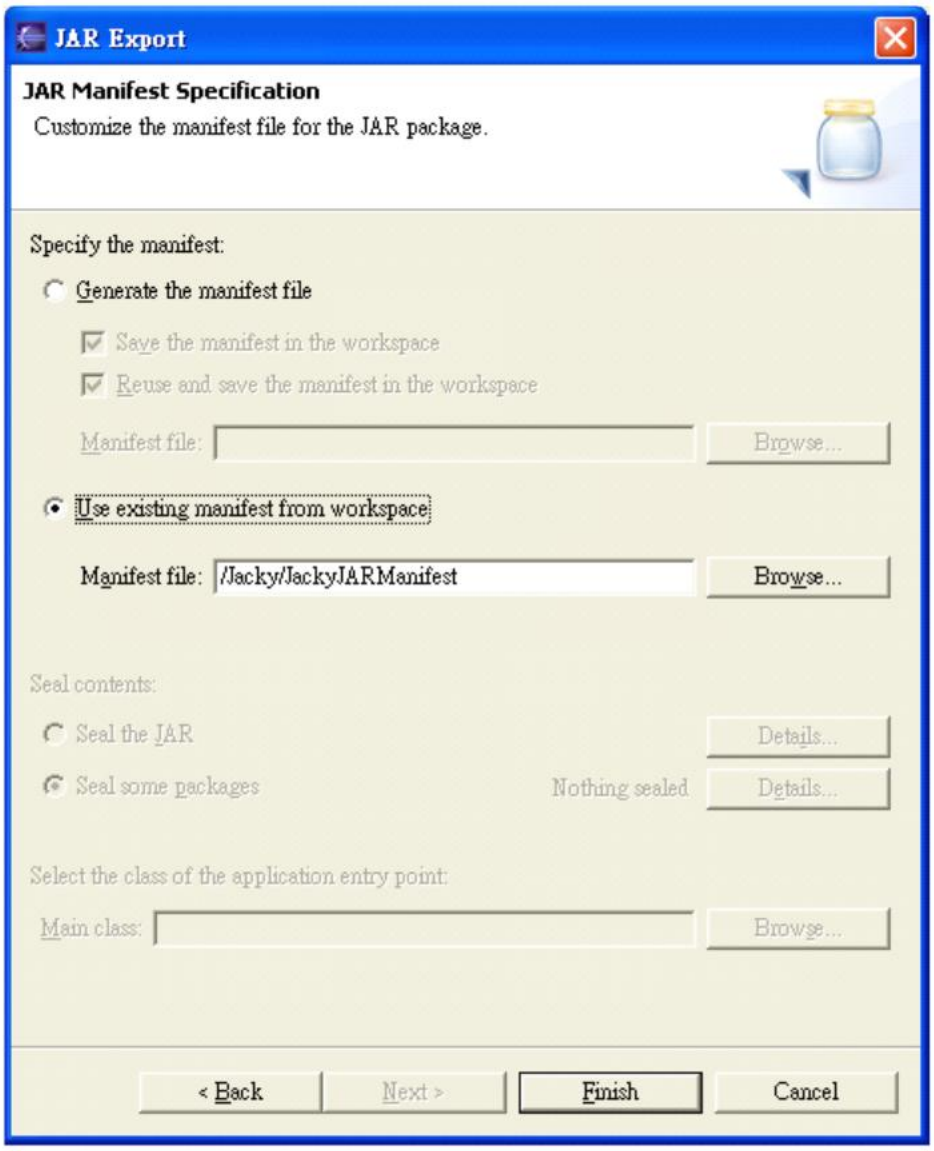


图 4.36

4.8.4 重新产生 JAR 檔

可以使用 JAR 档说明来重新产生先前所建立的 JAR 檔。

从选项的蹦现菜单中，选取 Create JAR。这时会重新产生 JAR 檔。

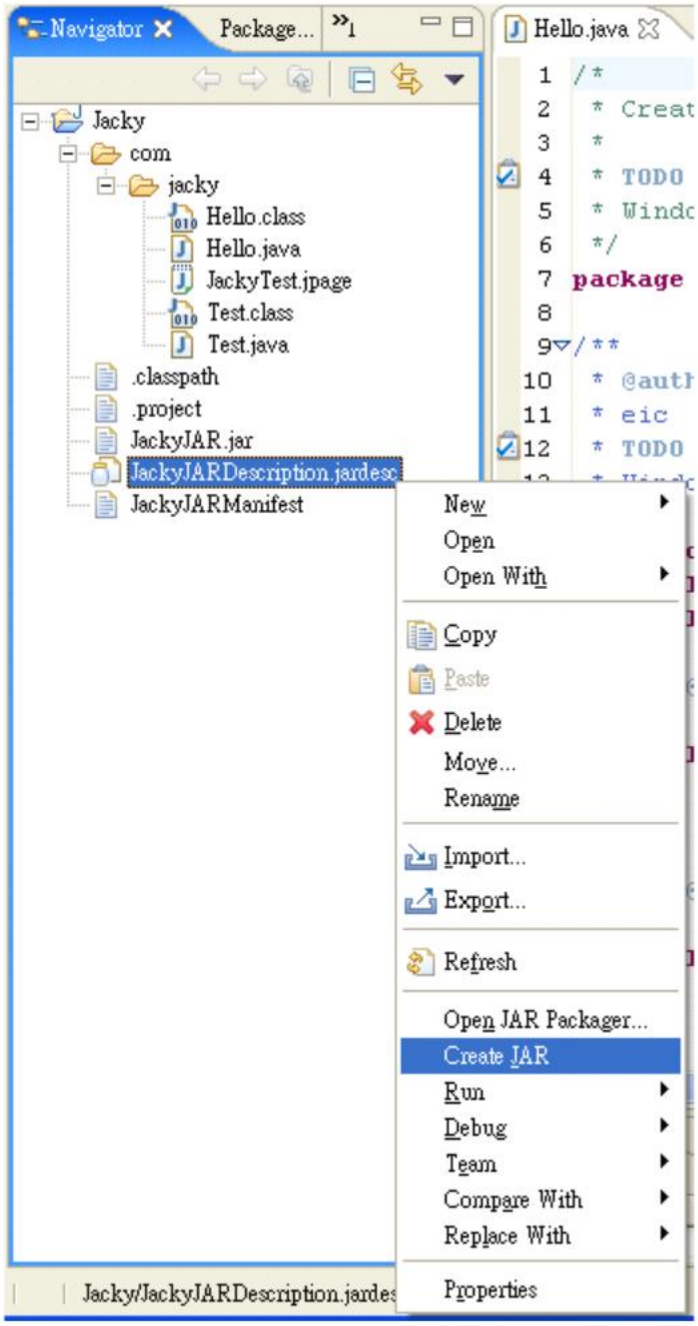

Navigator
Package...
Hello.java
Jacky
com
jacky
Hello.class
Hello.java
JackyTest.jpage
Test.class
Test.java
.classpath
.project
JackyJAR.jar
JackyJARDescription.jardesc
JackyJARManifest
New
Open
Open With
Copy
Paste
Delete
Move...
Rename
Import...
Export...
Refresh
Open JAR Packager...
Create JAR
Run
Debug
Team
Compare With
Replace With
Properties
Jacky/JackyJARDescription.jardes

图 4.37

4.9.建立 Javadoc 文件

选取所要的套件、来源资料夹或项目组（内含一或多个元素），以便为其产生 Javadoc 文件。

执行下列之一，以开启「汇出」精灵：

- 选取选择项之蹦现菜单中的汇出；或
- 从菜单列中选取「File」→「Export...」。

在出现的对话框中，从清单中选取 Javadoc，并按下下一步。

4.9.1 选取产生 Javadoc 用的类型

I. 指定 Javadoc 指令的位置。

II. 在树状结构控制中，选取要的元素，以为其产生 Javadoc。

III. 使用为具有可见性的成员建立 Javadoc 下的圆钮，选取可见性。

IV. 维持选取使用标准 doclet 圆钮。(或是自定 doclet)

V. 按下完成，为所选的元素产生 Javadoc，或按下一步，指定其它选项。

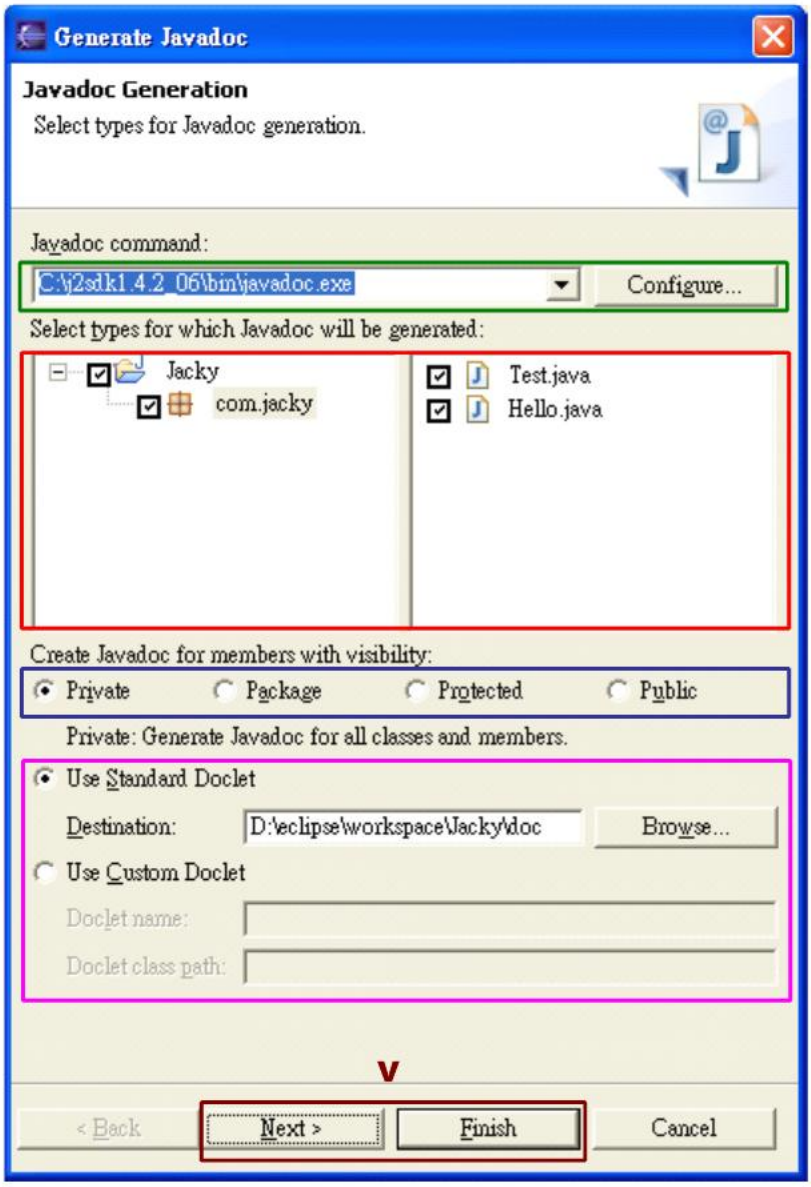


图 4.38

4.9.2 为标准 doclet 配置 Javadoc 自变量

I. 标题名称

II. 请使用基本选项下的勾选框，以指定 Javadoc 选项。

可以使用阐明这些标示群组中的勾选框，以变更所要阐明的标示。

III. 如果想让链接库中之类别的参照，链接至链接库的 Javadoc，请在清单中选取该链接库，并按下配置，以指定链接库之 Javadoc 的位置。

IV. 选择 CSS 档案

V. 按下完成，以产生 Javadoc；或按下一步，以指定其它的 Javadoc 产生选项。

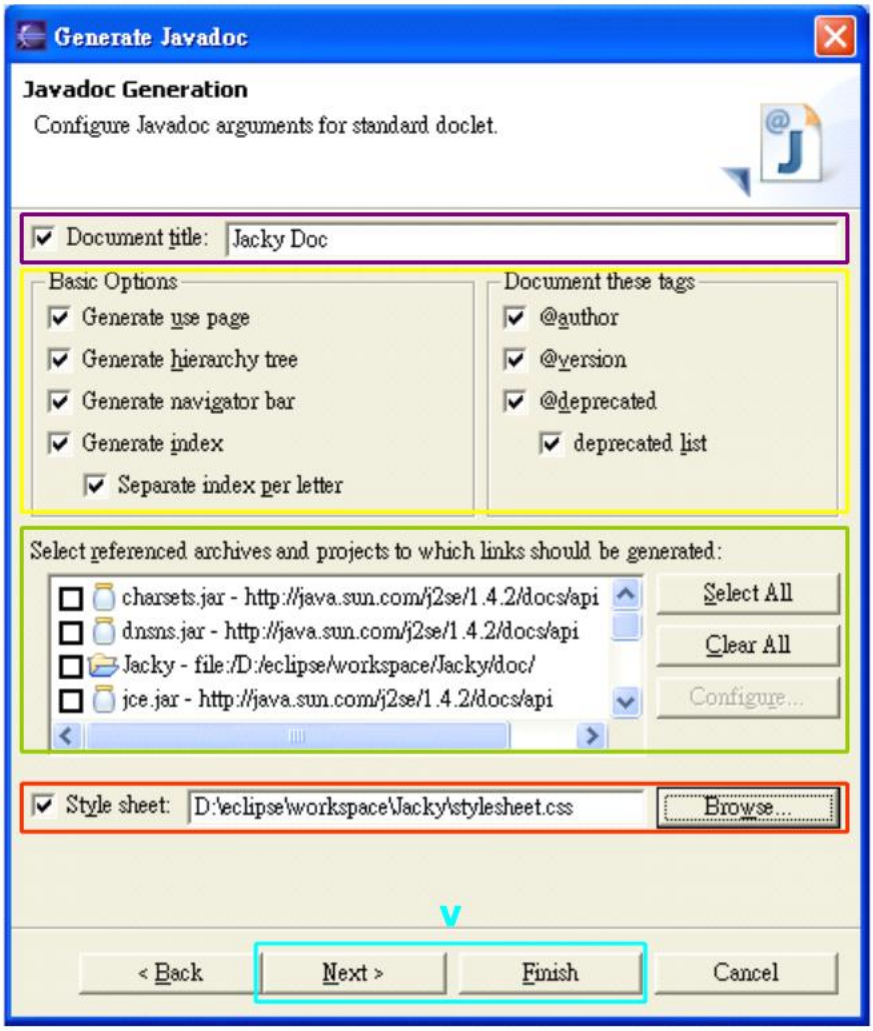


图 4.39

4.9.3 配置 Javadoc 自变量

I. 选取在浏览器中开启产生的索引文件勾选框。

II. 可以为 Javadoc 指令指定多个特定选项，方法是在文字区中输入这些选项。

III. 选取将这个 Javadoc 汇出的设定储存成 Ant Script 勾选框。

IV. 指定 Ant Script 的位置。

V. 按完成，以产生 Javadoc。

附注：Javadoc 指令产生的输出（包括错误与警告）会显示在「Console」视图中。

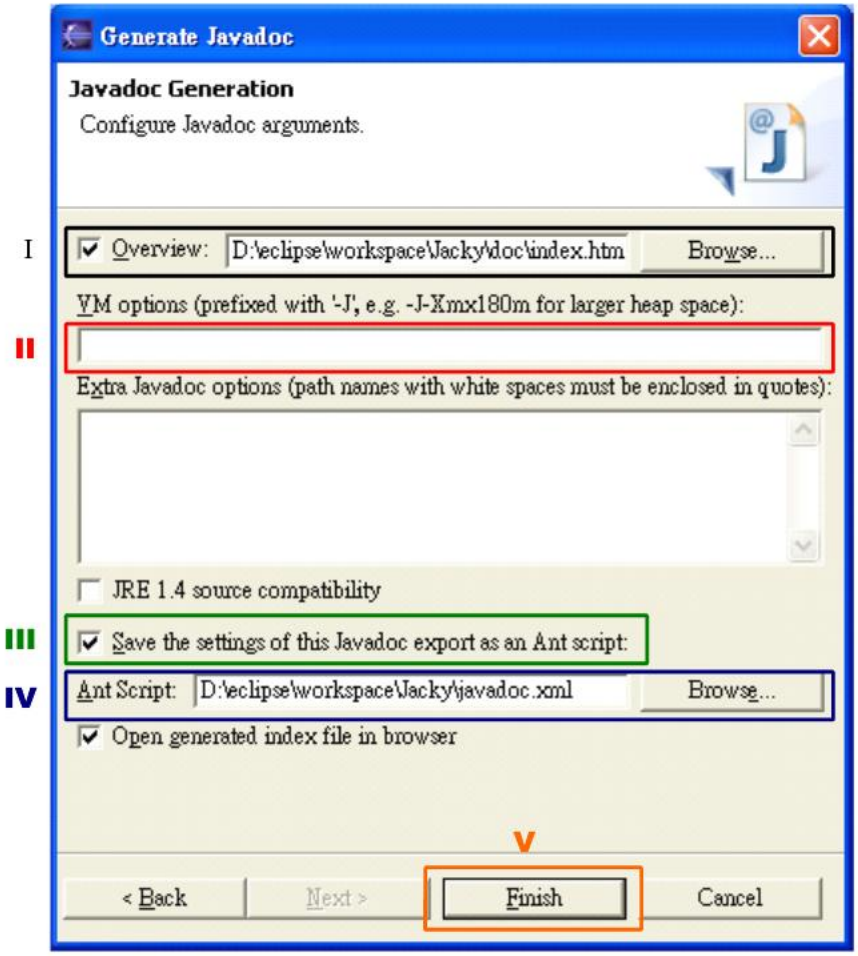


图 4.40

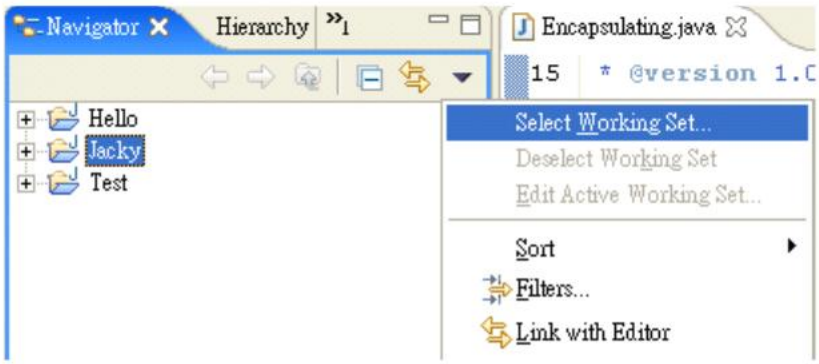
4.10 工作集(Working Sets)

工作集将元素分组，以显示在视图中，或对元素集进行作业。「Navigate」视图使用工作集来限制显示的资源集。如果在导览器中选取了工作集，只会显示资源、资源的子项，以及资源的母项。

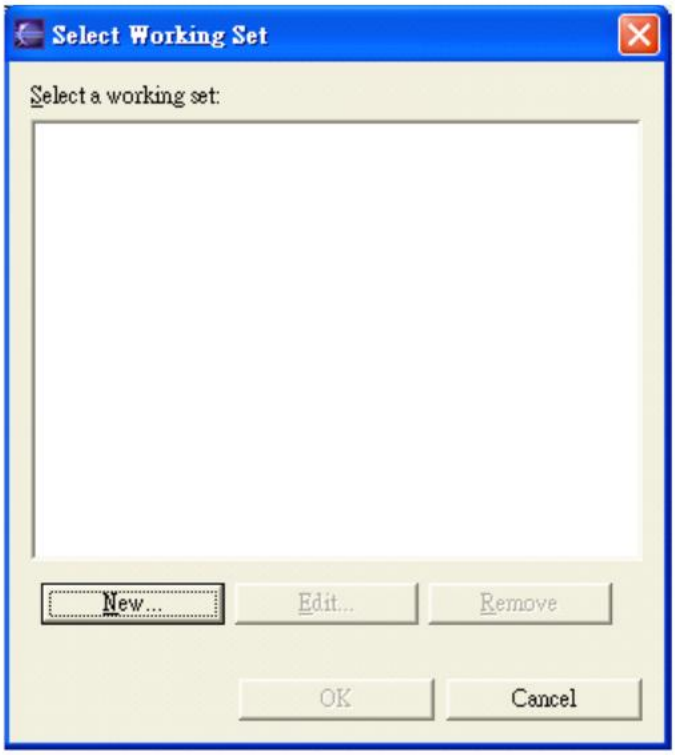
可以在「Tasks」视图中使用工作集来限制作业的显示，方法与「Navigate」视图类似。在使用工作台搜寻机能时，可以使用工作集来限制可搜寻的元素集。不同的视图提供不同的指定工作集方法。不过，它们通常使用下列工作集选项对话框来管理现有的工作集以及建立新的工作集。

4.10.1 新增工作集

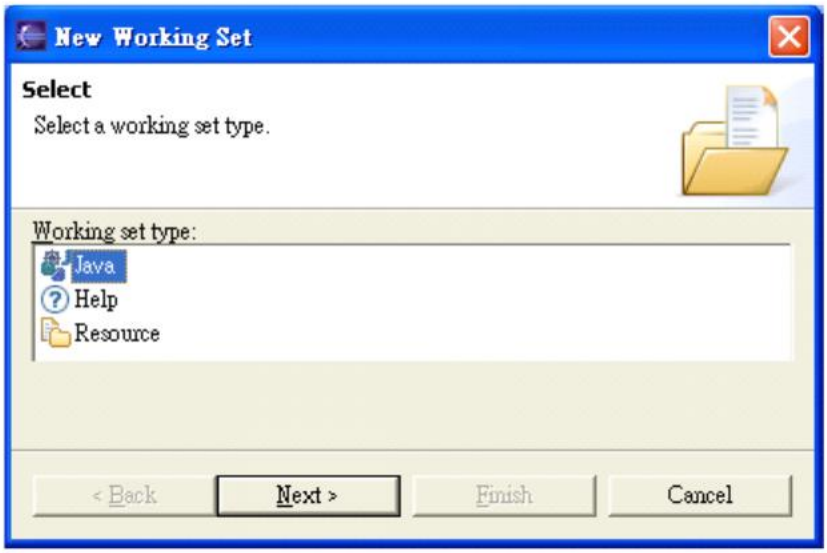
I. 在「导览器」视图的工具列上，按一下菜单按钮 ▾ ，开启显示选项的下拉菜单。



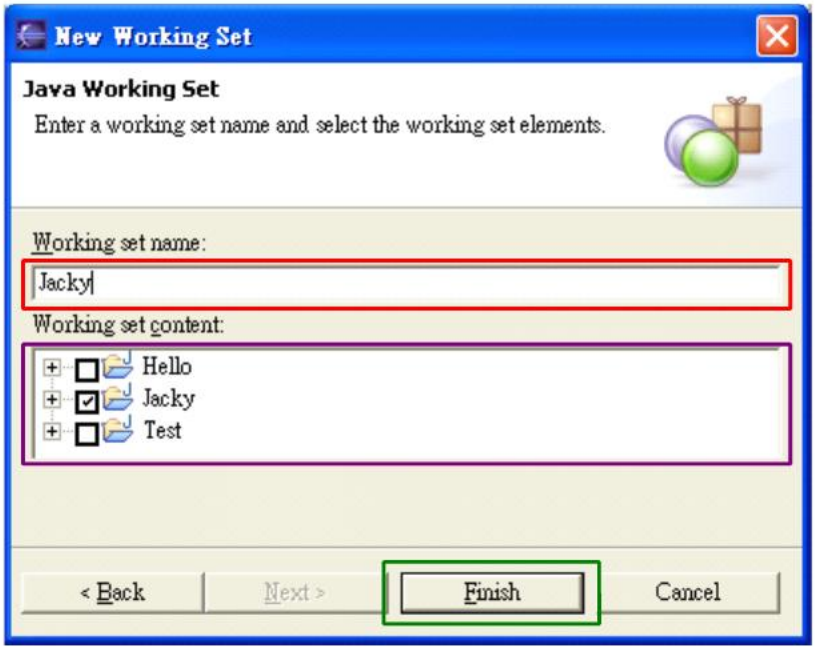
II. 选 Select Working Set 后，出现 Select Working Set 的窗口



III. 选择 Java 后，按 Next



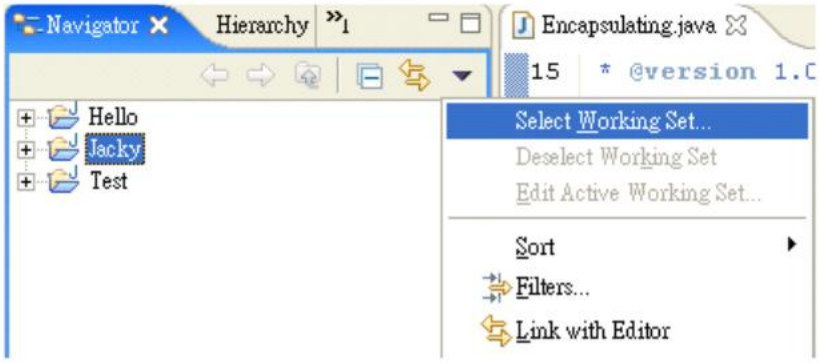
VI. 在 New Working Set 窗口输入名称和勾选所需的项目，最后按下 Finish 即可



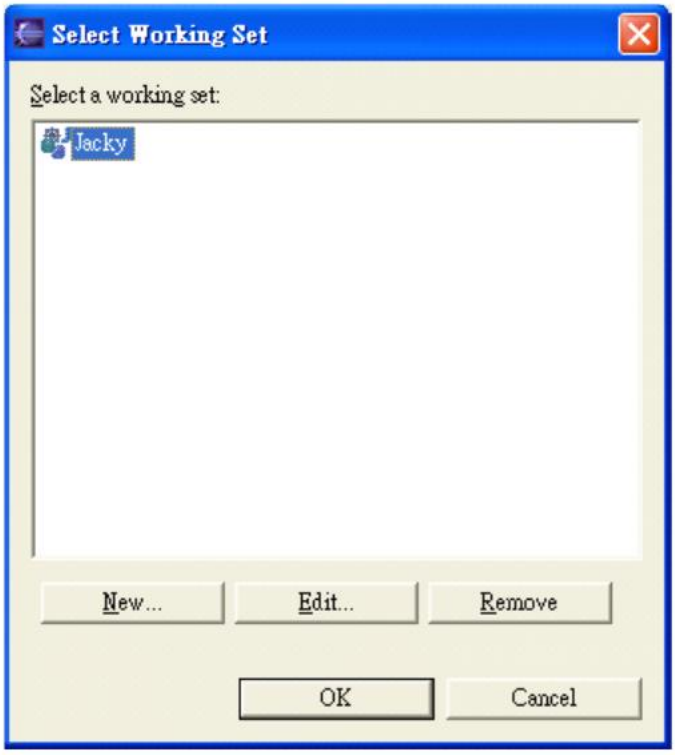
附注：新建的资源不会自动并入作用中的工作集。如果它们是现有的工作集元素的子项，则会被隐含地并入工作集中。如果要在建立其它资源之后并入它们，必须明确地将它们新增至工作集。

4.10.2 隐藏「导览器」视图中的档案

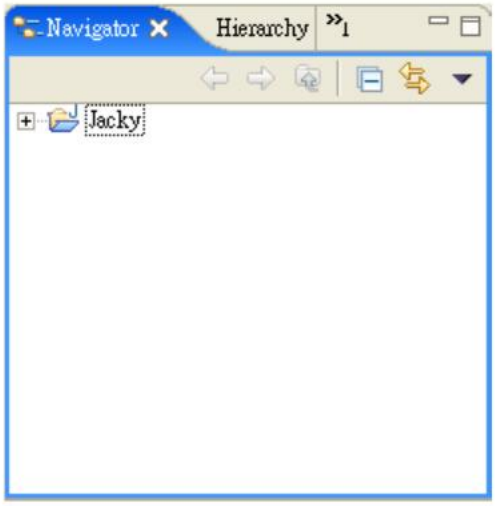
I、在「导览器」视图的工具列上，按一下菜单按钮 ▼，开启显示选项的下拉菜单。



II、选 Select Working Set 后，出现 Select Working Set 的窗口

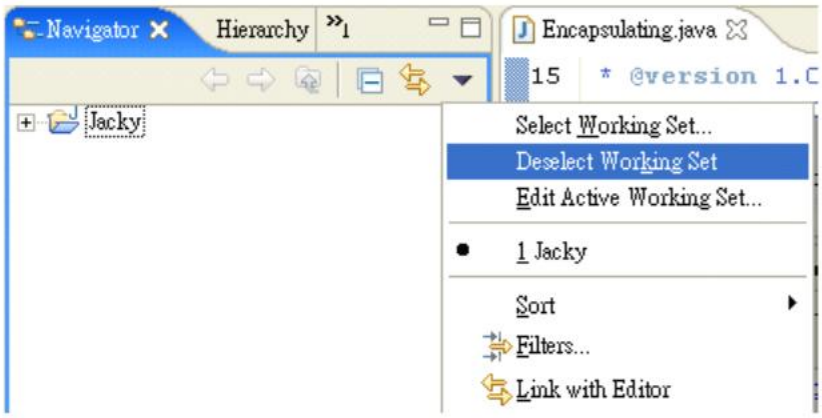


III. 从清单中选取一个现有的工作集，来隐藏「导览器」视图中的档案

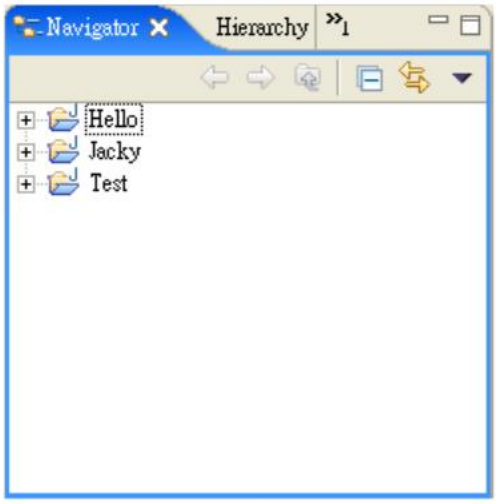


4.10.3 显示「导览器」视图中的档案

I. 在「导览器」视图的工具列上，按一下菜单按钮 ▼，开启显示选项的下拉菜单。



II. 选 Deselect Working Set 后，就可以出现原有的档案



5.除错

我们的说明是采用逻辑错误，藉此追踪下去；范例之后，要谈一些更进阶的除错主题，例如设定除错启动组态，使用 Hot Code Replacement，暂停执行中且不会中断的程序(例如无穷回圈)等等。对 IDE 而言，能够和程序做交互式的除错，是应该具备的功能。

5.1 错误的程序

错误的范例程序是要做阶乘(n! = n * (n-1) * (n-2) *... * 1)。此范例会建立多层的堆栈框(stack frame)。

```
public class ErrorTest {
    public static void main(String[] args) {
        System.out.println(factorial(6));
    }
    public static int factorial(int value) {
        if (value == 0) {
            return value;
        } else {
            return value * factorial(value - 1);
        }
    }
}
```

此例中，是求 factorial(n)，这个方法会递归的呼叫自己，直到此阶乘被算尽为止。此例是要找出 6 的阶乘，也就是 720，可惜第一次执行此例的结果是 0。

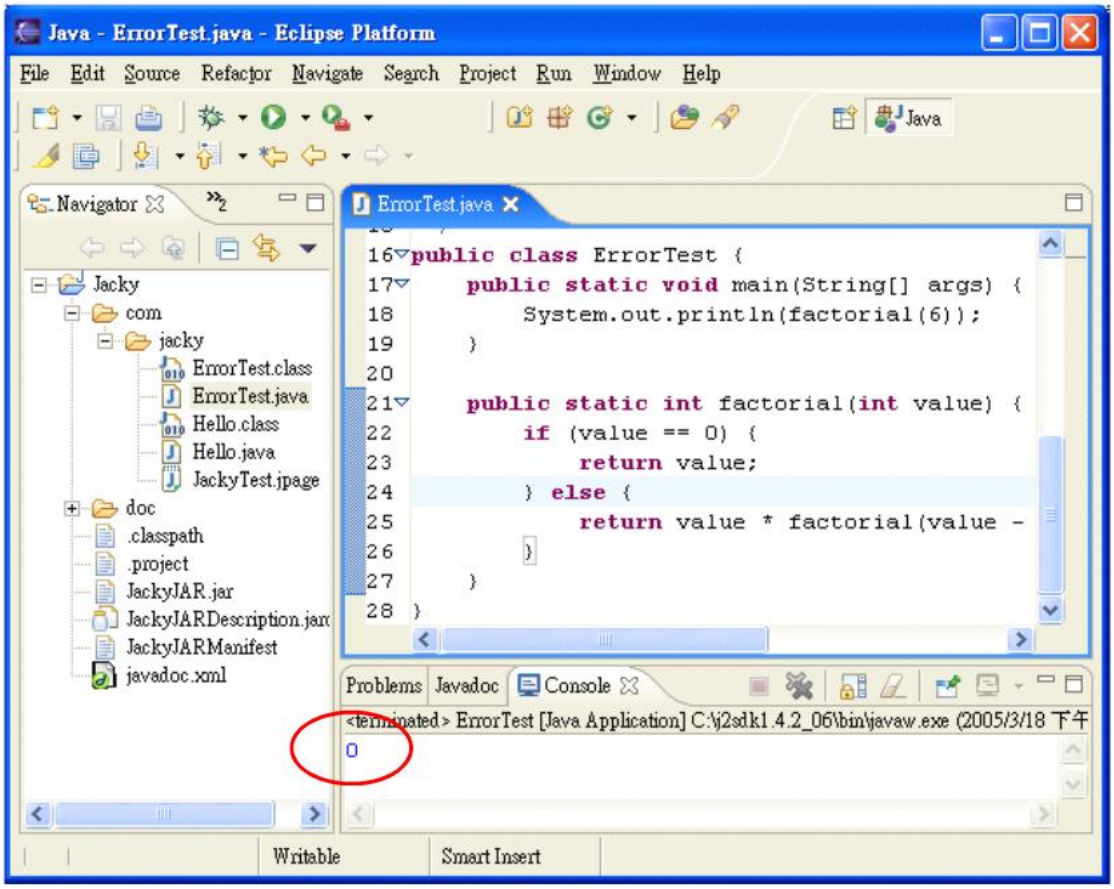


图 5.1

5.2 设定岔断点(Breakpoints)

由于没有抛出任何的例外事件，所以问题是在程序的逻辑。要在程序执行期间检视程序，所以要设定岔断点(Breakpoints)来暂停程序。在要暂停的程序代码前面的「Marker Bar」点两下(或是「Run」→「Toggle Line Breakpoint」)来设定岔断点，稍后要移除岔断点，只要再对该岔断点按两下即可。

要安插一个岔断点到 return value * factorial(value - 1)的旁边，这样才能观看连续呼叫 factorial()方法而建立的阶乘值。在「Marker Bar」上有一个蓝点。

开始除错，「Run」→「Debug As」→「Java Application」(或是按 旁边的箭头选「Debug As」→「Java Application」)，还开启「Debug」视景。

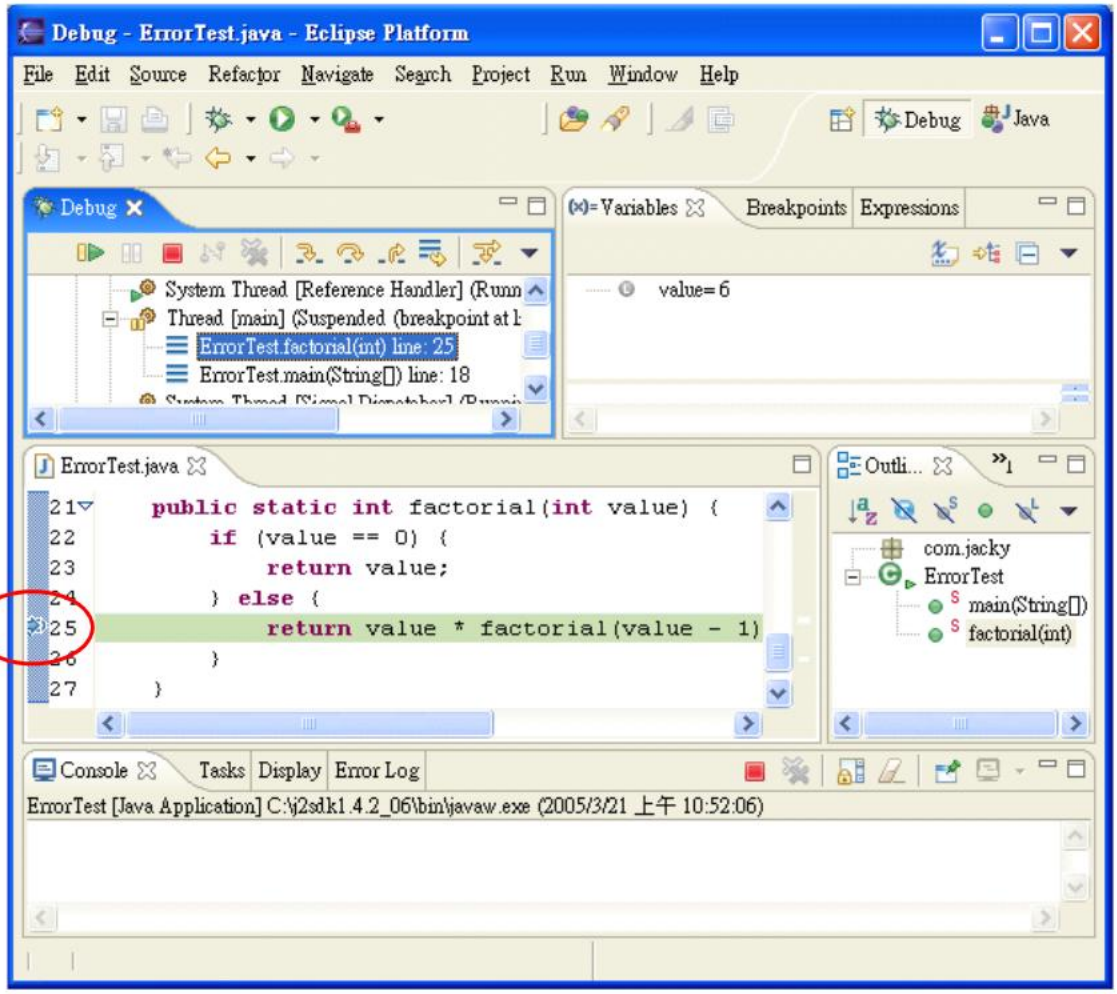


图 5.2

程序执行到岔断点会暂停，执行暂停处的该列程序会出现在「Debug」编辑器中，标上一个箭头。

先了解「Debug」视景。在左上角的「Debug」视图中可以看见正在除错的程序构成项目。这里的堆栈框都有标上三条横棒 ≡ 。此例中，我们正在 factorial()方法中，已经由 mail()方法所呼叫了。「Debug」视图中由左到右的按钮分别是， Resume 按钮(在开始执行程序)、 Suspend 按钮(暂停程序)、 Terminate 按钮(中止除错)、 Disconnet 按钮、 Remove All Terminated Launches 按钮(除去先前 debug session)。

「Debug」视图右边是层迭的视图。分别是「Variables」、「 Breakpoints」和「Expressions」。

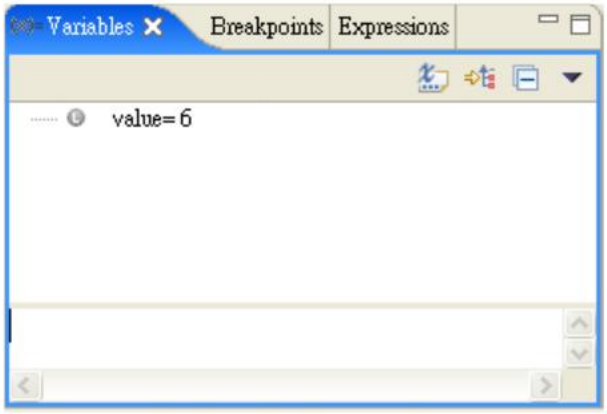


图 5.3

「Variables」视图可以检视区域变量之值。在除错程序时，可以编修区域变量之值(稍后会做)，这样可以和程序互动以修正问题。Eclipse 会监视这些变量值，当这些变量值有变时，会改变颜色(改成红色)。「Variables」视图底端的部分称为**详细资料窗格**(Detail Pane)，会显示更完整的信息。

「Breakpoints」视图管理程序中的岔断点，对清单中的某各岔断点按右键，在从选单中选择「Enable」、「Disable」、「 Remove」或「Remove All」。

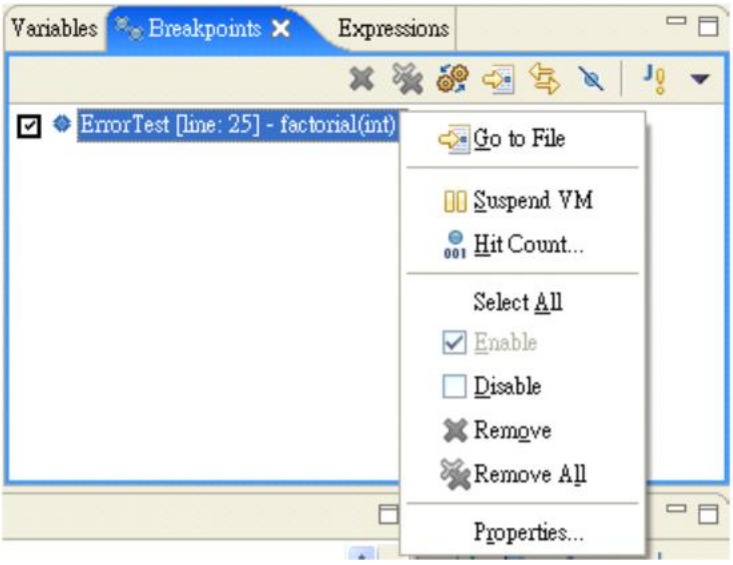


图 5.4

「Expressions」视图可以计算表示式(稍后会做)，在编辑器中选取一道表示式，按右键，选择 Inspect 选项，就可以在「Expressions」视图中予以计算。

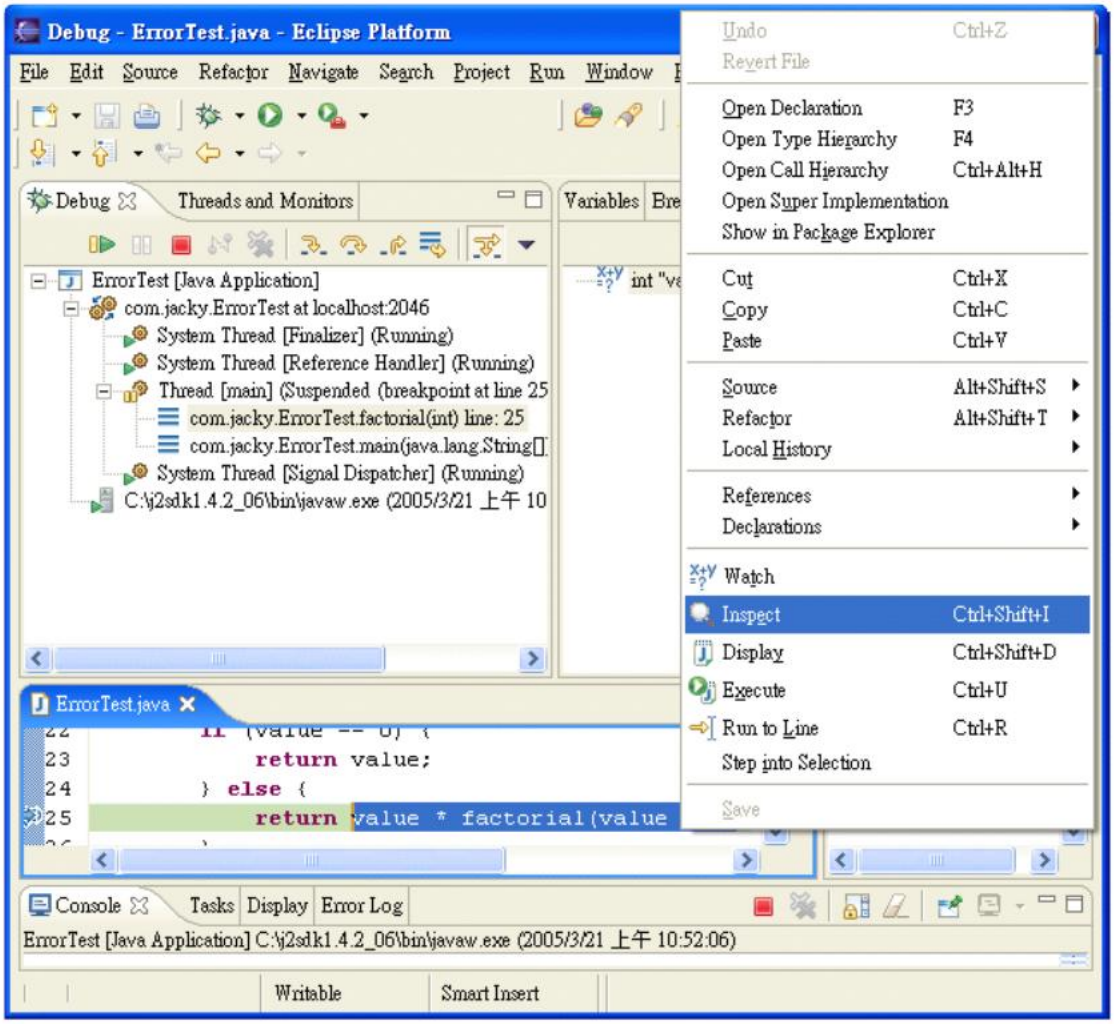


图 5.5

若选 Display 选项时，计算的结果会显示在「Display」视图中。

「Debug」视景中的编辑器和「Java」视景中的编辑器本质上一样的，但是「Debug」视景的编辑器可以检视变量的值，只要鼠标移到变量上即可。

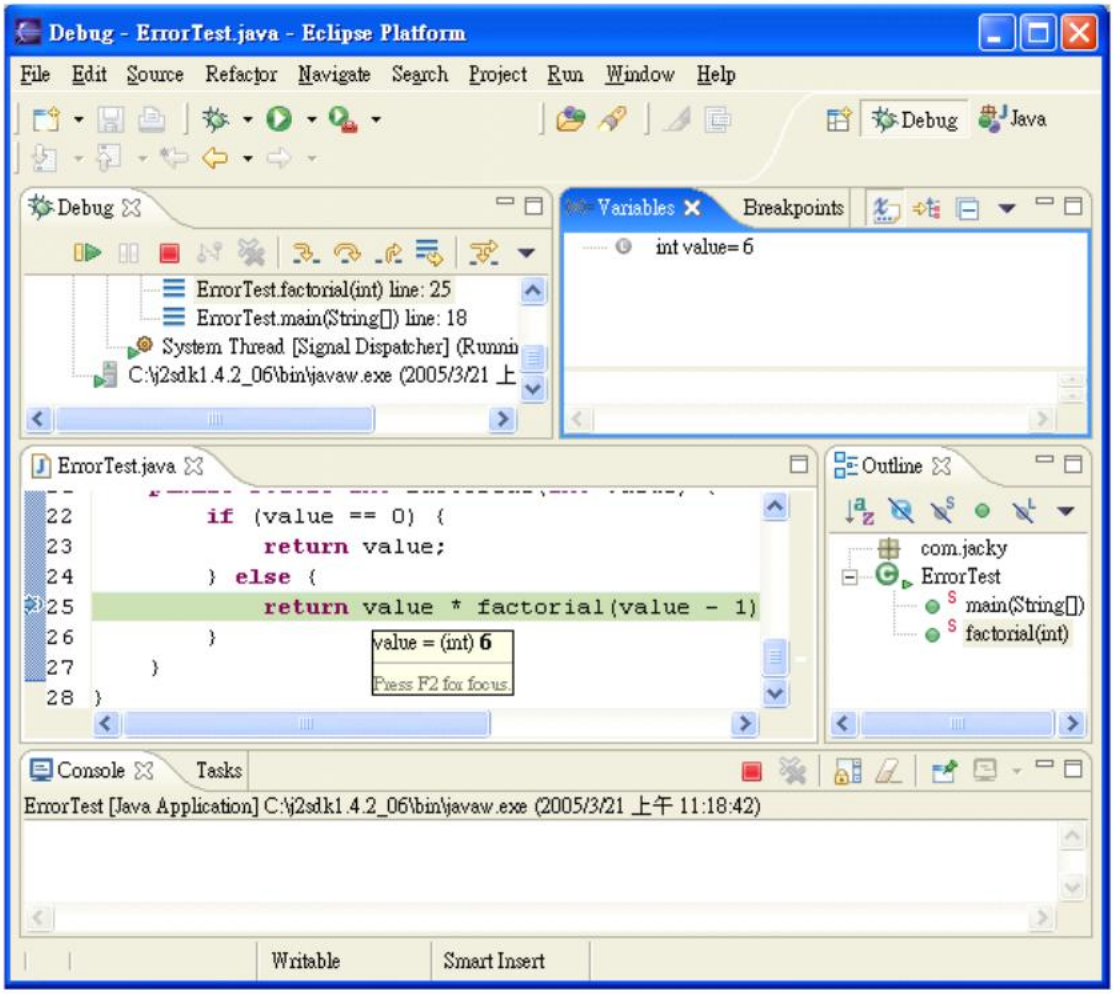


图 5.6

5.3 逐步除错

在暂停的程序中移动最基本的作法是采逐步法(single-steping)。Eclipse 提供下列的选项：

5.3.1 Step Into

按按钮(也可以按 F5)，进入选取的叙述内。如果该叙述是呼叫某方法，则进入执行该方法。

5.3.2 Step Over

按按钮(也可以按 F6)，掠过选取的叙述内。如果该叙述是呼叫某方法，则不会进入该方法。

5.3.3 Step Return

按按钮(也可以按 F7)，执行将回复，直到现行方法中下一个 return 陈述式要执行为止，且执行会暂停于下个可执行行上。

5.3.4 Drop to Frame

按按钮，这个指令可以放回与重新输入指定的堆栈框。这项特性类似「回头执行」再整个重新启动程序。如果要放回堆栈框，再重新输入指定的堆栈框，请选取要「放置」的指定堆栈框，再选取 **Drop to Frame**。

请注意下列有关这项特性的警告：

- 不能在堆栈中放入原生方法。
- 全体数据不受影响，仍维持其现行值。举例来说，不会清除内含元素的 Static 向量。

附注：只有在基础 VM 支持这项特性时，才会启用这个指令。

5.3.5 Use Step Filters/Step Debug

按按钮(也可以按 Shift - F5)，当动作切换为开启时，每一个逐行动作（over、into、return）都会套用使用者喜好设定所定义的逐行过滤器集（请参阅「Window」→「Preferences」→「Java」→「Debug」→「Step Filtering」）。当呼叫逐行动作时，逐行作业会一直进行，直到到达未经过滤的位置，或是遇到岔断点为止。

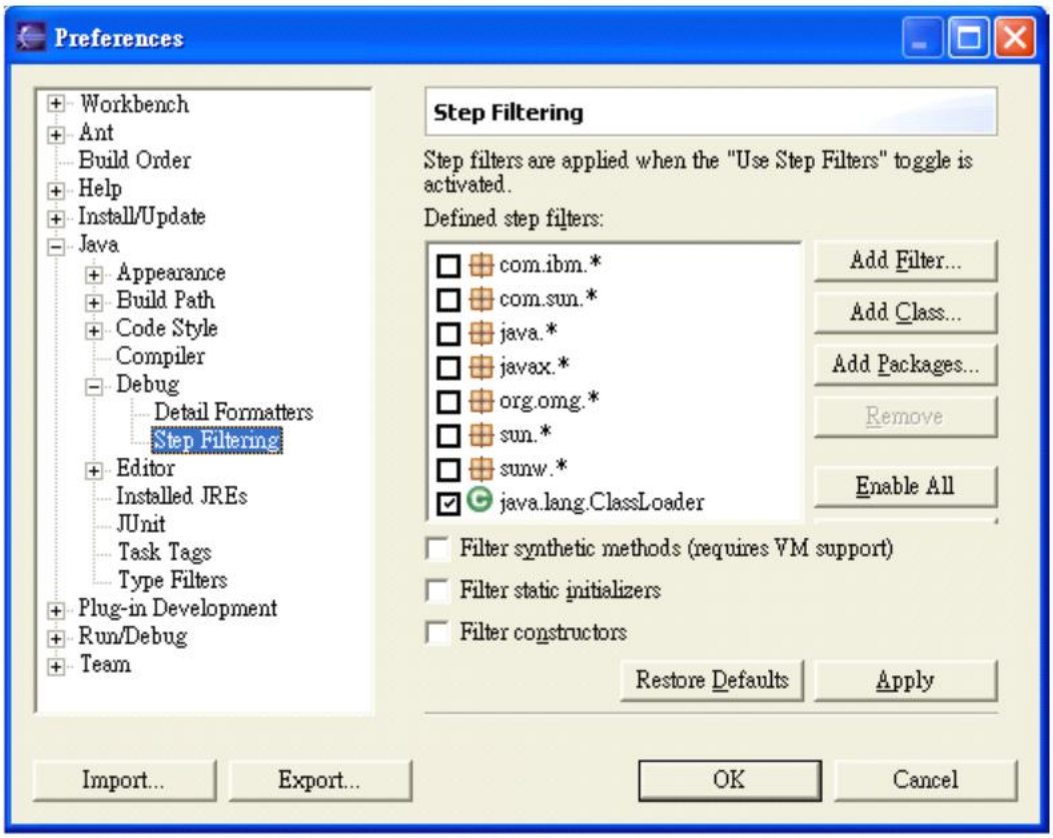


图 5.7

例如，我们的 debug session 线在暂停在 return value * factorial(value - 1)这一列程序代码，按 F5，就会走进该列，也就是说会开始执行 factorial(value - 1)的呼叫，value 的变量之新值为 5。

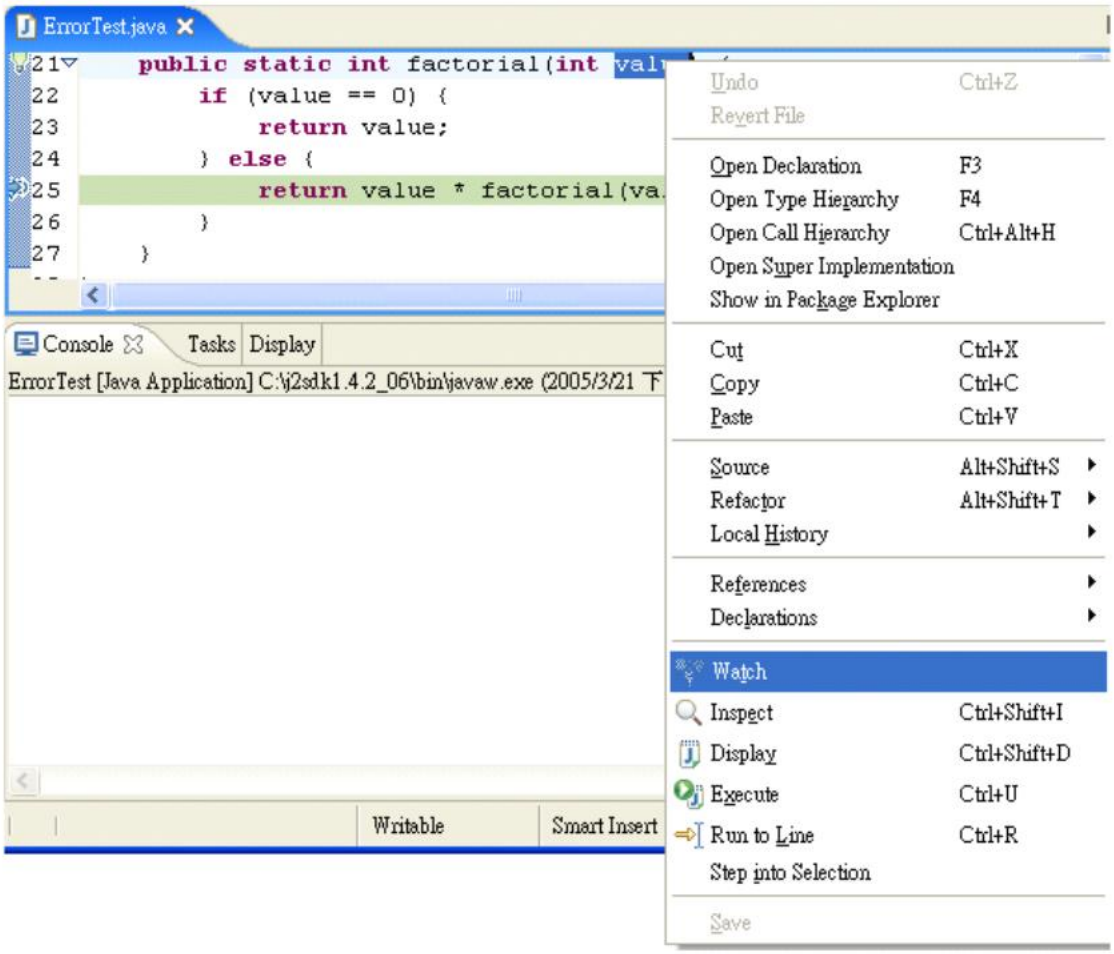


图 5.9

现在点 Resume 按钮，程序会继续执行，直到碰到下一个岔断点，看一下 value 的值为 5 之后，会发现扔然在同一个 factorial()之内，只要重复不断按 Resume 按钮，可以看出 value 值的变化。

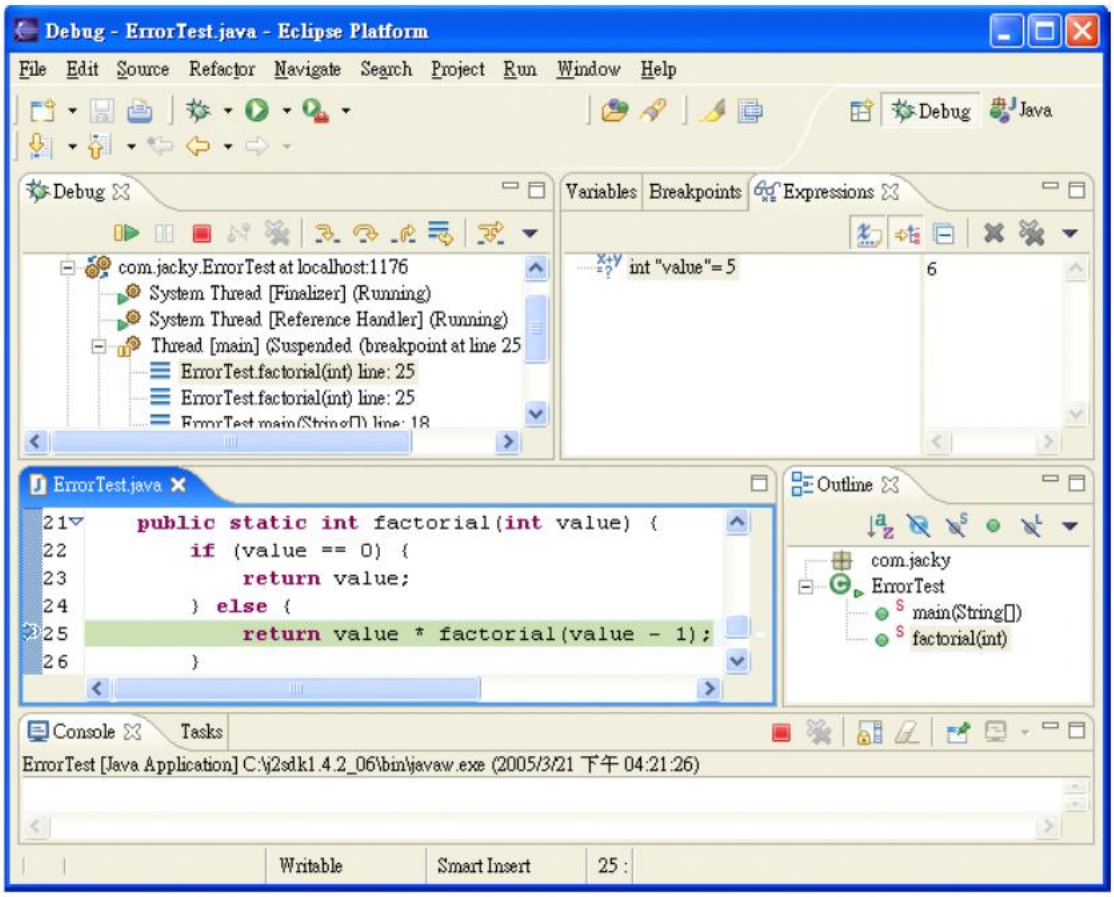


图 5.10

5.5 设定岔断点的 Hit Count

这个 factorial()方法要跑 6 次，所以需要按 6 次 Resume 按钮；也可以设定 Hit Count 来节省时间。有 2 种方式设定：在「Breakpoints」视图的岔断点按右键

选择 Properties

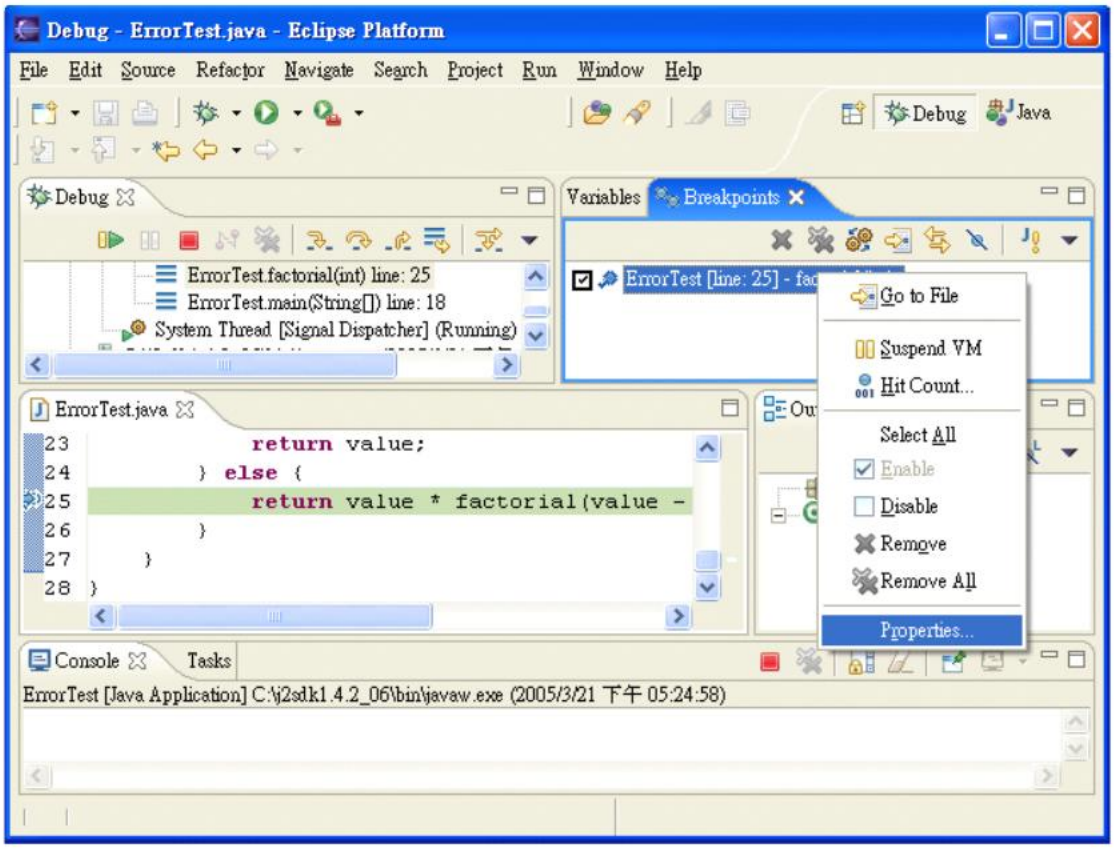


图 5.11

开启 Breakpoints Properties 窗口，勾选 Hit Count，并输入 6

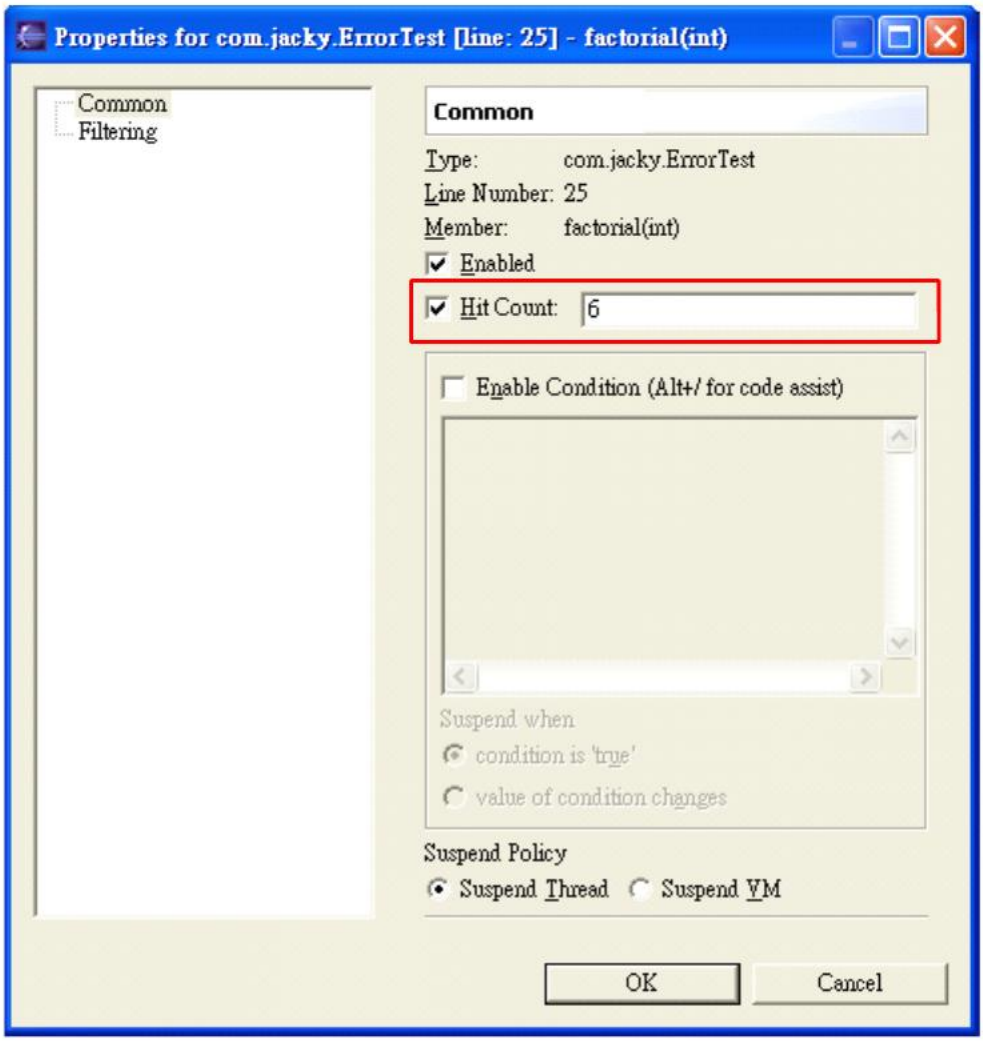


图 5.12

(或是选择 Hit Count。

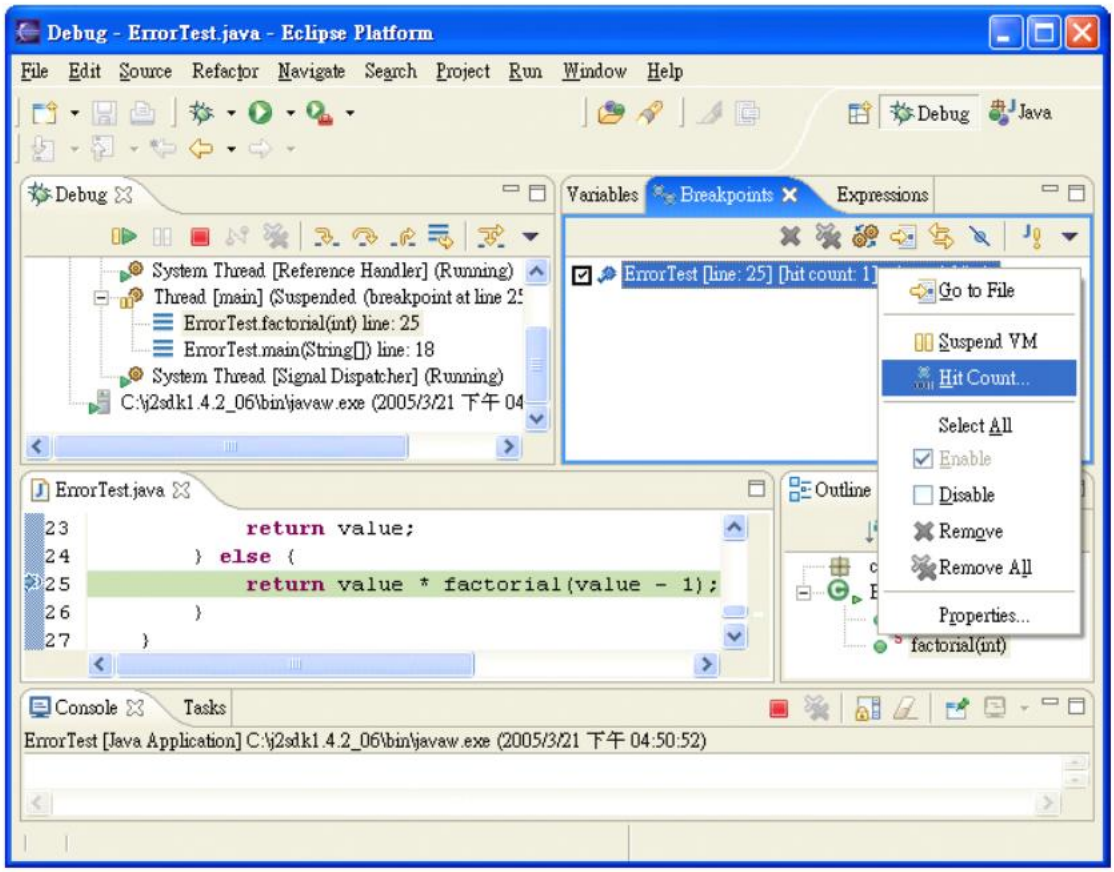


图 5.13

开启 Set Breakpoint Hit Count 窗口，并输入 6

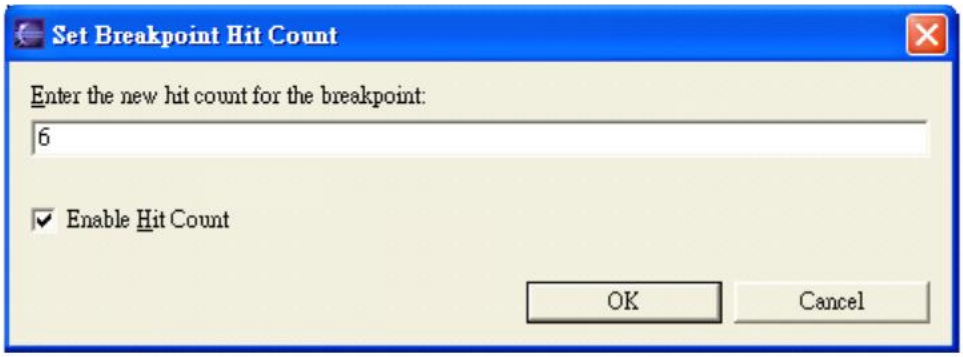


图 5.14)

重新开始此 debug session，程序执行会在第 6 次碰上岔断点时暂停，注意到 value 的值是 1，而在「Debug」视图中可以看见 factorial()连续呼叫的堆栈框。

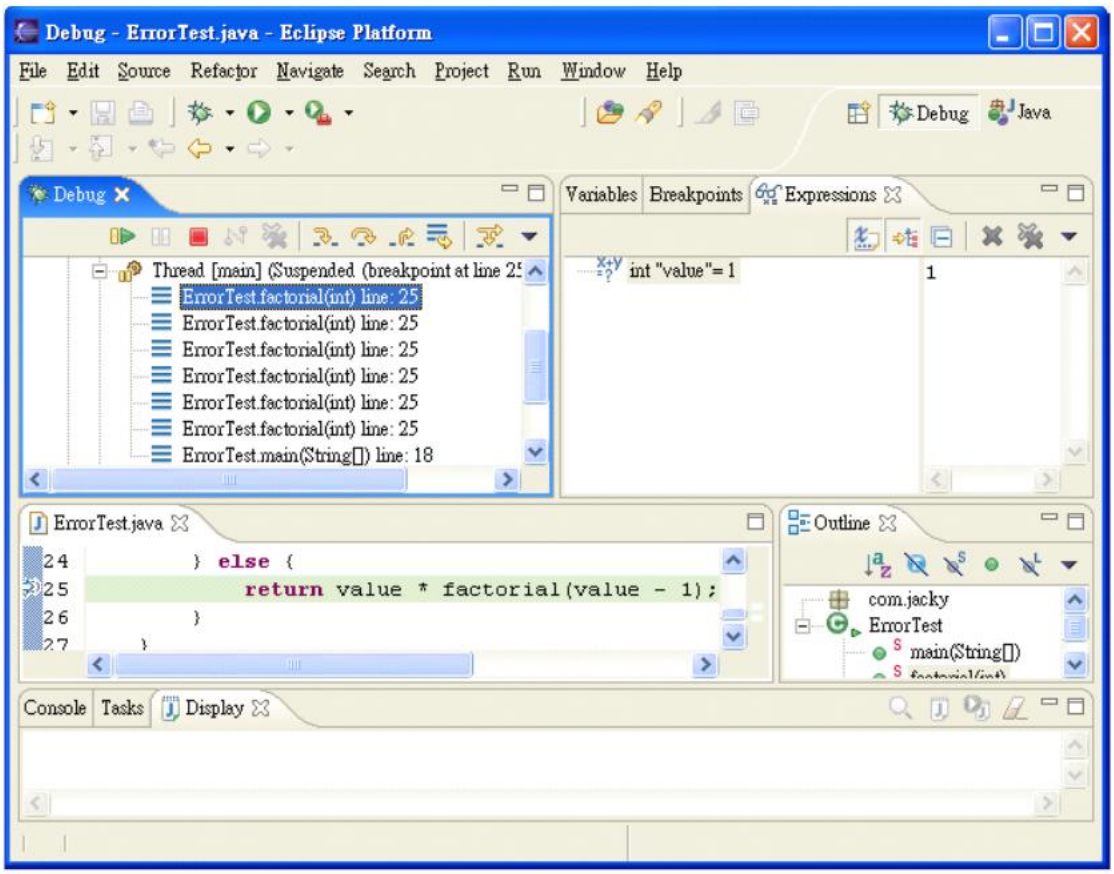


图 5.15

要检视「Debug」视图中任何的堆栈框，以及其中区域变量的值，只要对某堆栈框点一下，使其成为现行堆栈框(active frame)。该区域变量都随着堆栈框保留下来，例如当前堆栈框的 value = 1，下一个堆栈框 value = 2。

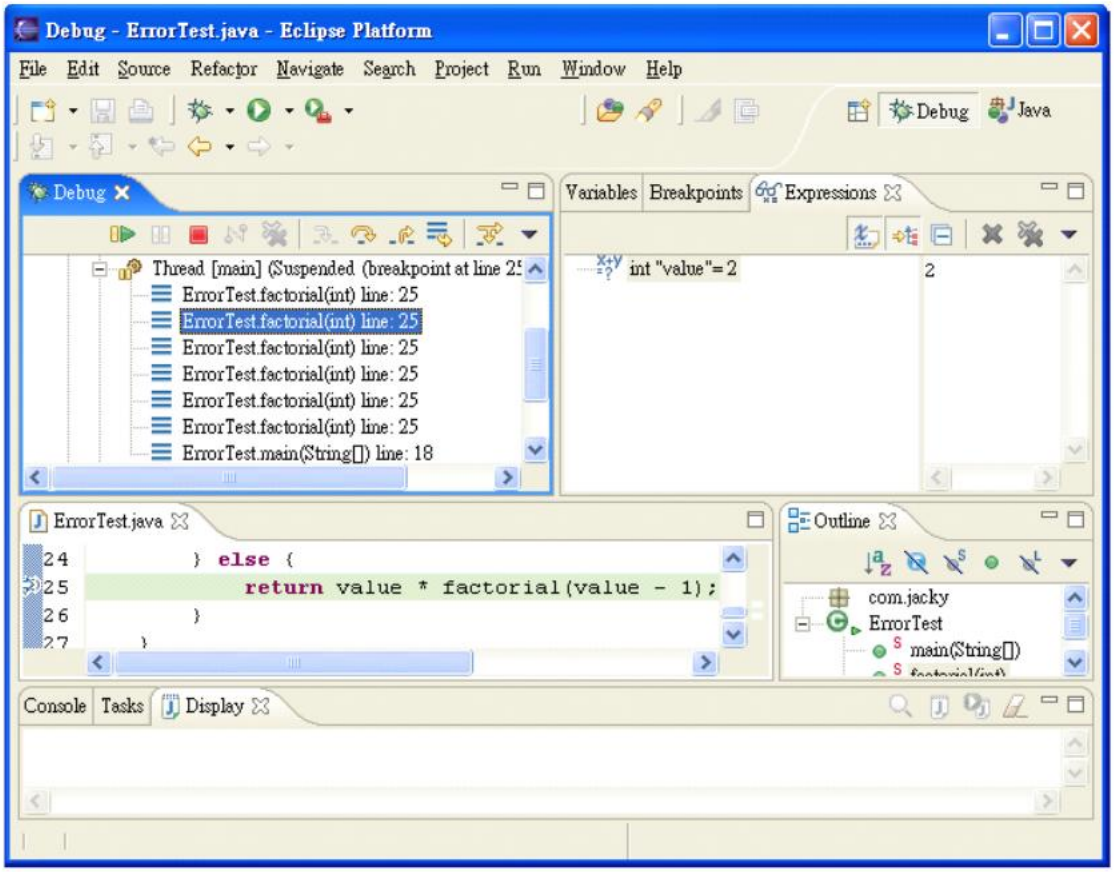


图 5.16

以上的执行一切都没问题，然而接下来再按下 Step Into 按钮时，程序却又跑进 factorial()方法，而「Expressions」视图中的 value 之值变成 0，这是不对的。Value 值永远不能为 0，因为阶乘只能用到正整数。

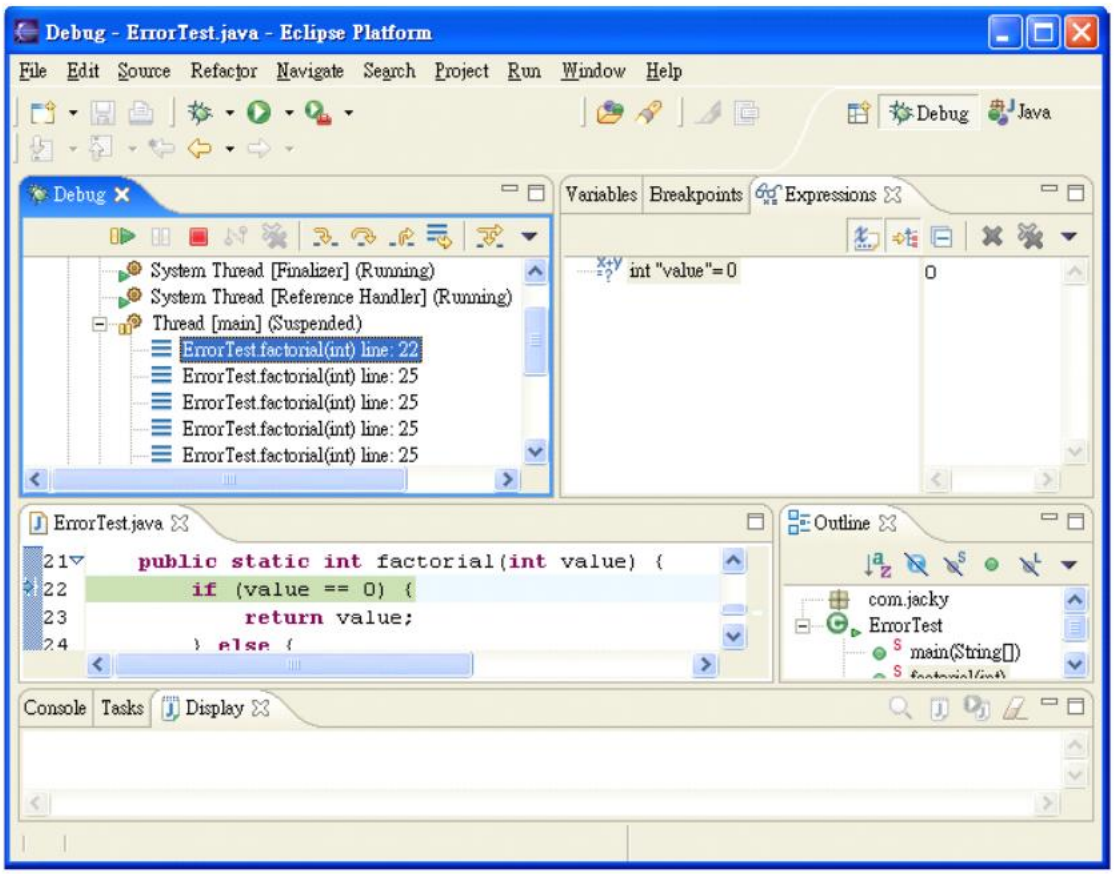


图 5.17

终于找到问题的所在，让 factorial()呼叫重复到 value = 1 的时候，而不是设为 0。

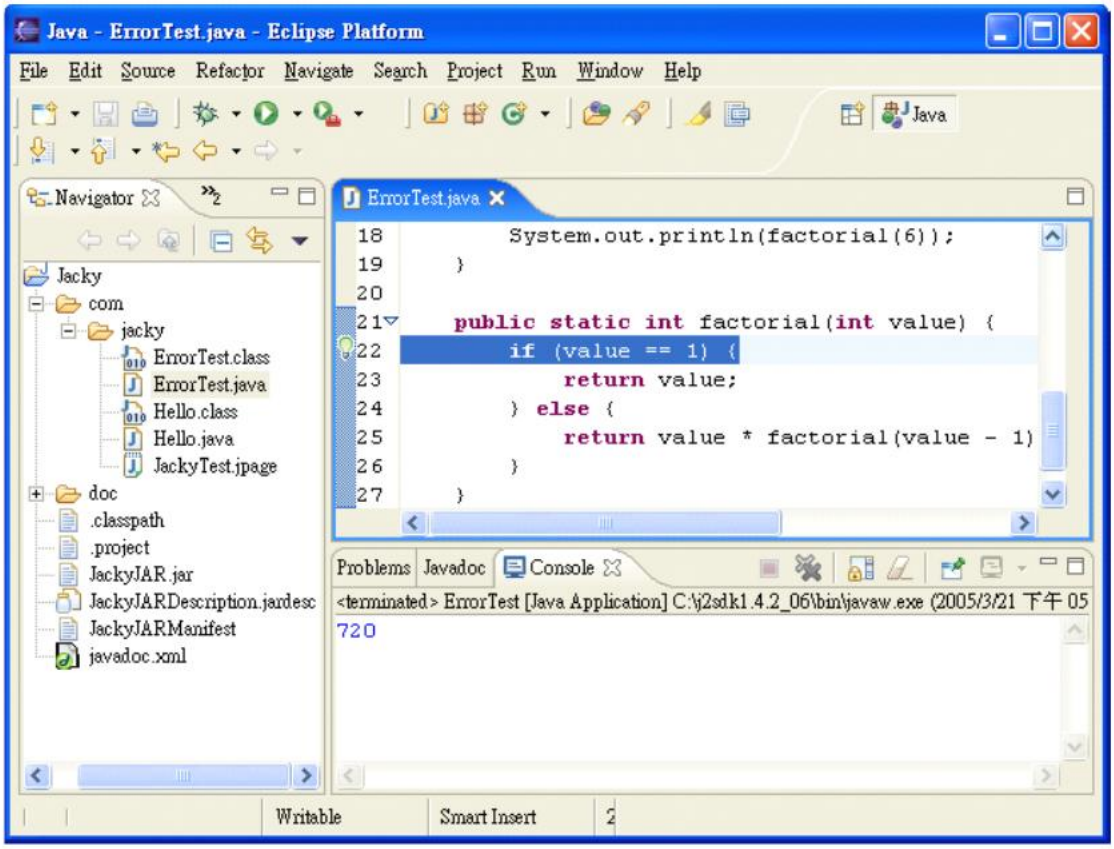


图 5.18

5.6 岔断点组态设定

利用 Hit Count 让除错方便一点，也可以用其它的做法来设定岔断点的组态，一样达到方便除错的目的。在「Breakpoints」视图对岔断点按右键，选择 Properties。

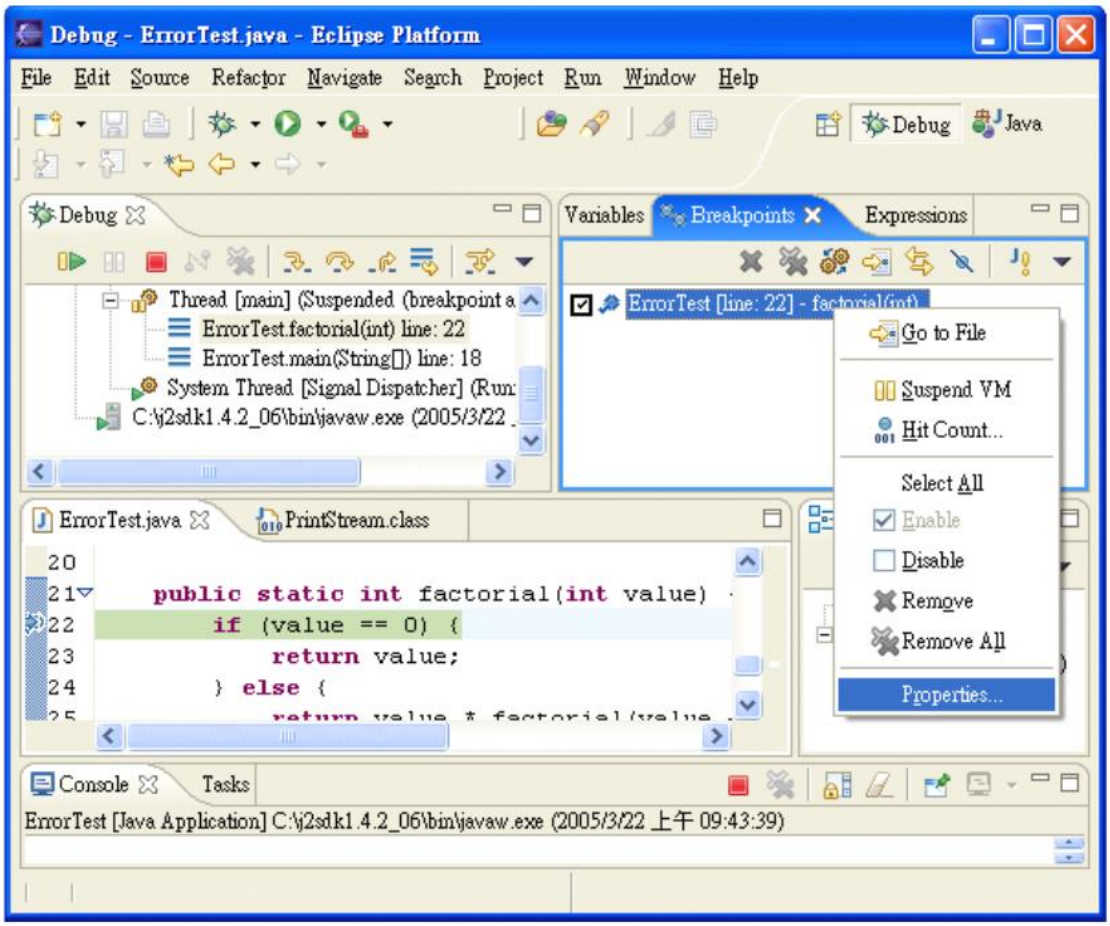


图 5.19

开启 Breakpoints Properties 窗口，选取 Enable Condition 的复选框，然后就可以输入条件式来暂停程序。

Suspend when 的选项中：

condition is‘true’(条件式成立)

value of condition changes(值改变时)

Suspend Policy

Suspend Thread 表示只暂停错误发生时的 Thread，其它 Thread 继续执行。

Suspend VM 表示暂停整个虚拟机器。

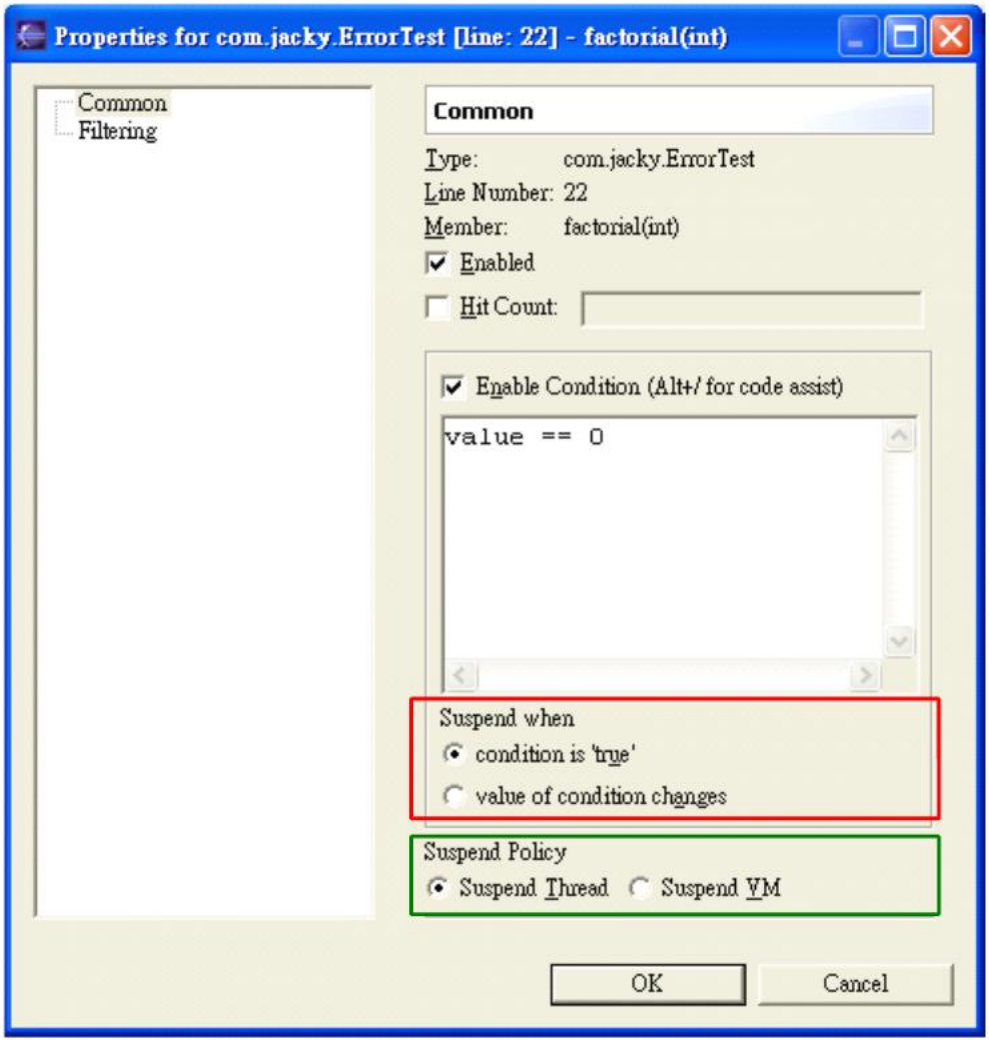


图 5.20

例如阶乘只能使用正整数，当 value == 0 时，就不符合阶乘的条件，就让程序暂停。

5.7 监视点(Watchpoint)

之前使用的岔断点称为 Line Breakpoint，除了 Line Breakpoint 以外，也支持监视点(Watchpoint)、方法岔断点(Method Breakpoint)以及异常岔断点(Exception Breakpoint)。

设定监视点，表示当程序准备去存取或修改某字段时，就会暂停执行。监视点不能设在区域变量身上，只能在字段身上。

设定监视点，在「Java」视景的编辑器中，选取一个字段，然后再选「Run」→「Toggle Watchpoint」。

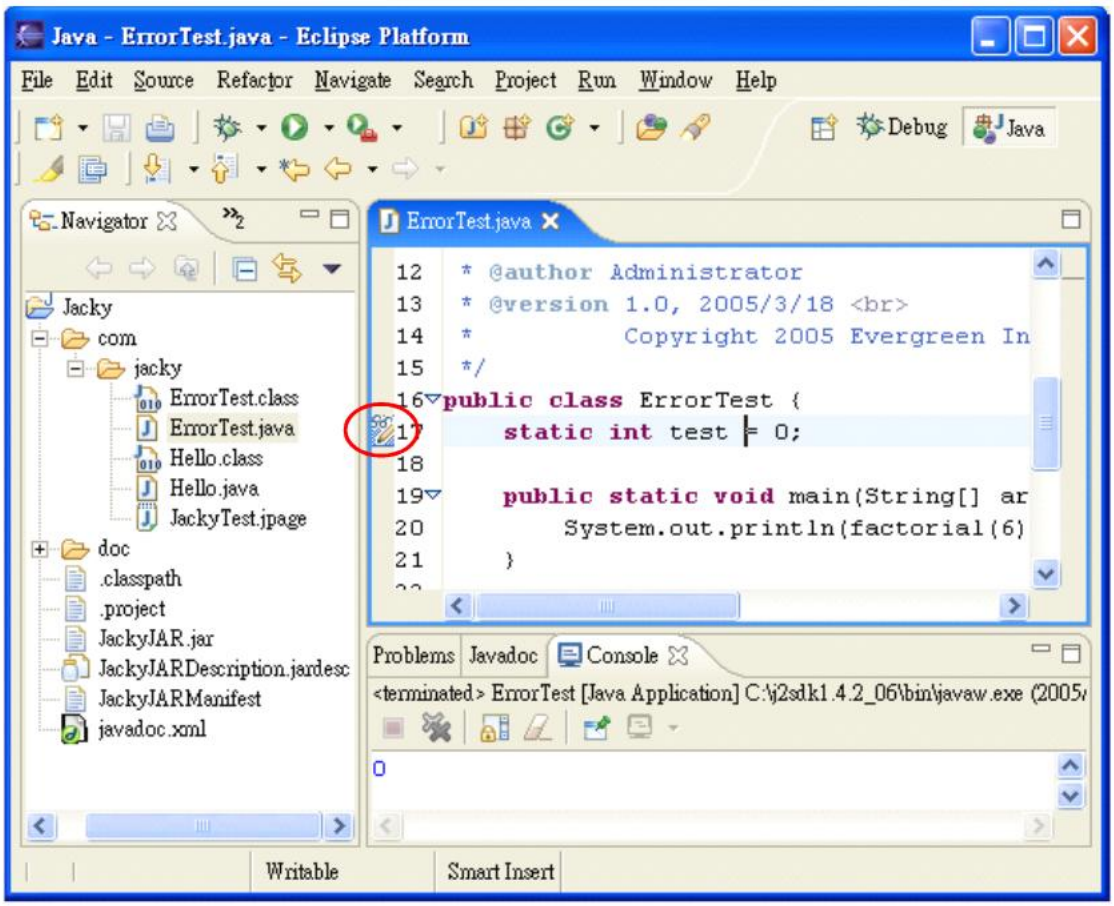


图 5.21

设定完成后，在「Marker Bar」会出现这个图示。

新的监视点会出现在「Debug」视景中的「Breakpoints」视图里，对该监视点按右键，选择 Properties。

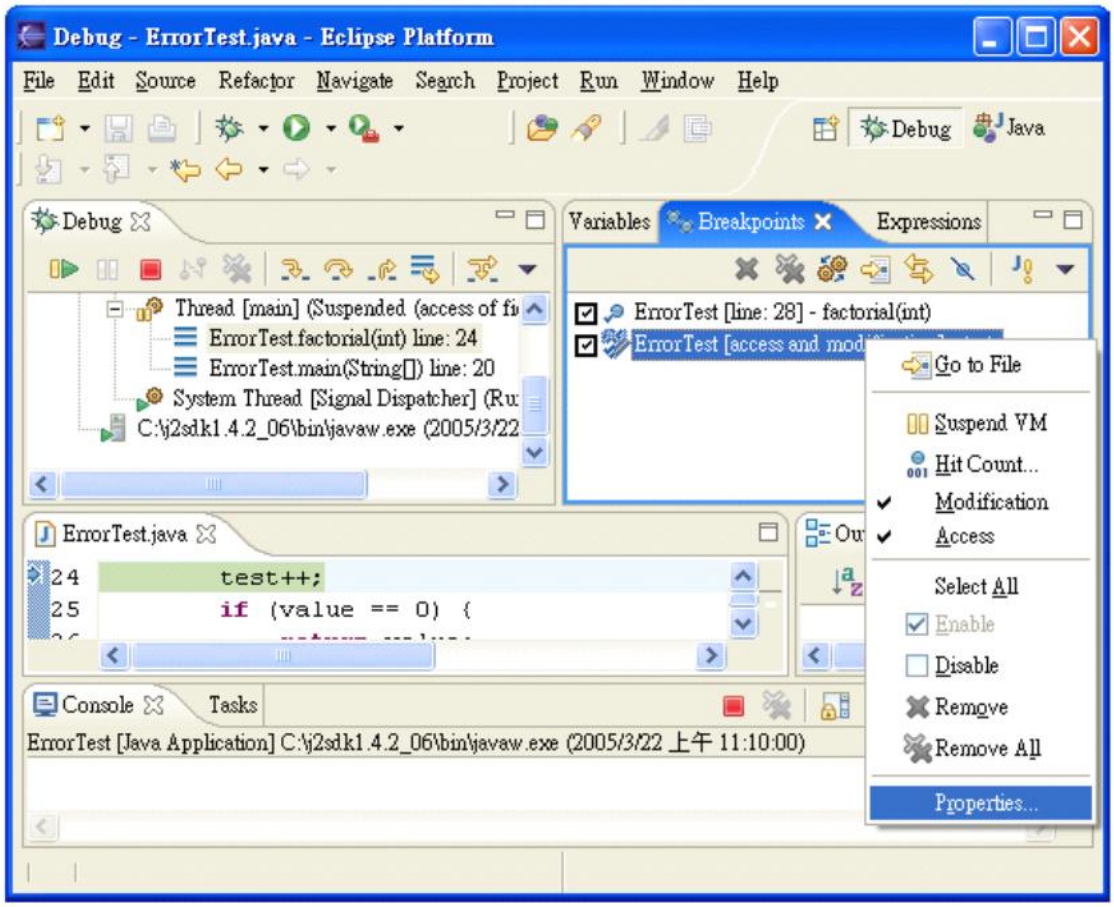


图 5.22

开启 Breakpoints Properties 窗口，选项设定跟之前的岔断点差不多，特别的是 Suspend on 的选项

Field Access 暂停程序之依据是当字段被存取

Field Modification 暂停程序之依据是当字段被修改

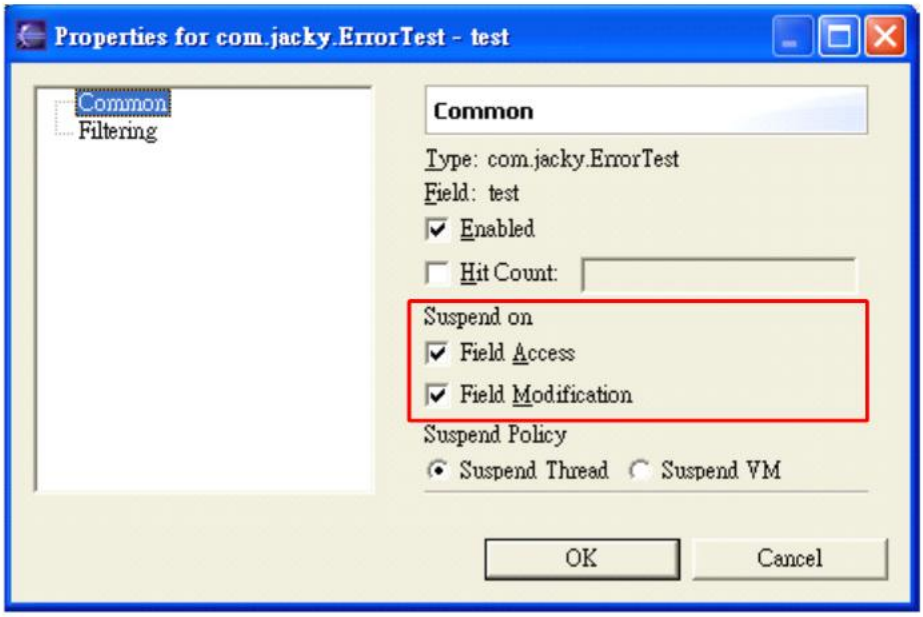


图 5.23

5.8 方法岔断断点(Method Breakpoint)

进入或离开某方法时，方法岔断点(Method Breakpoint)会暂停程序执行，至于是进入之时或是离开之时，依据组态的设定。

设定方法岔断点，在「Java」视景的编辑器中，把光标放在要监视的方法前，然后再选「Run」→「Toggle Method Breakpoint」。

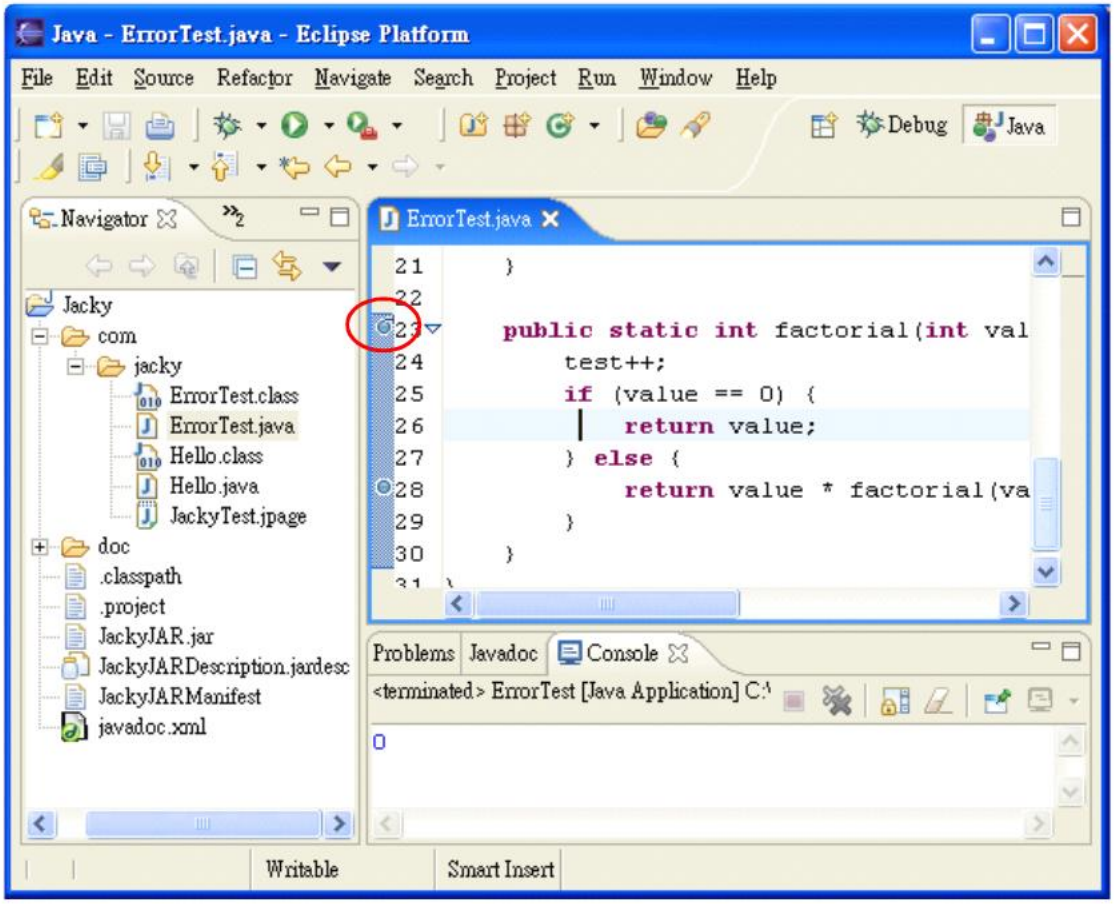


图 5.24

设定完成后，在「Marker Bar」会出现这个图示。

新的方法岔断点会出现在「Debug」视景中的「Breakpoints」视图里，对该岔断点按右键，选择 Properties。

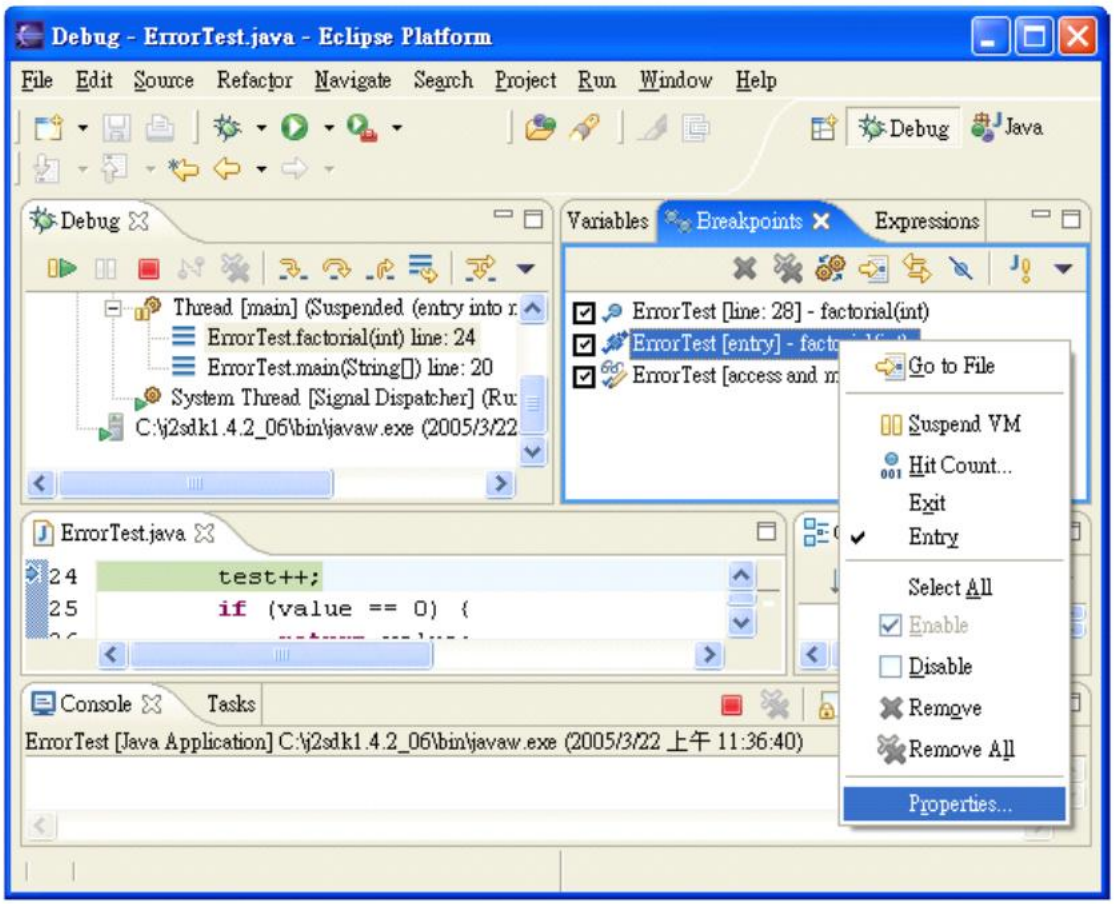


图 5.25

开启 Breakpoints Properties 窗口，选项设定跟之前的岔断点差不多，特别的是 Suspend on 的选项

Method Entry 决定岔断点生效之时是在进入该方法

Method Exit 决定岔断点生效之时是在离开该方法

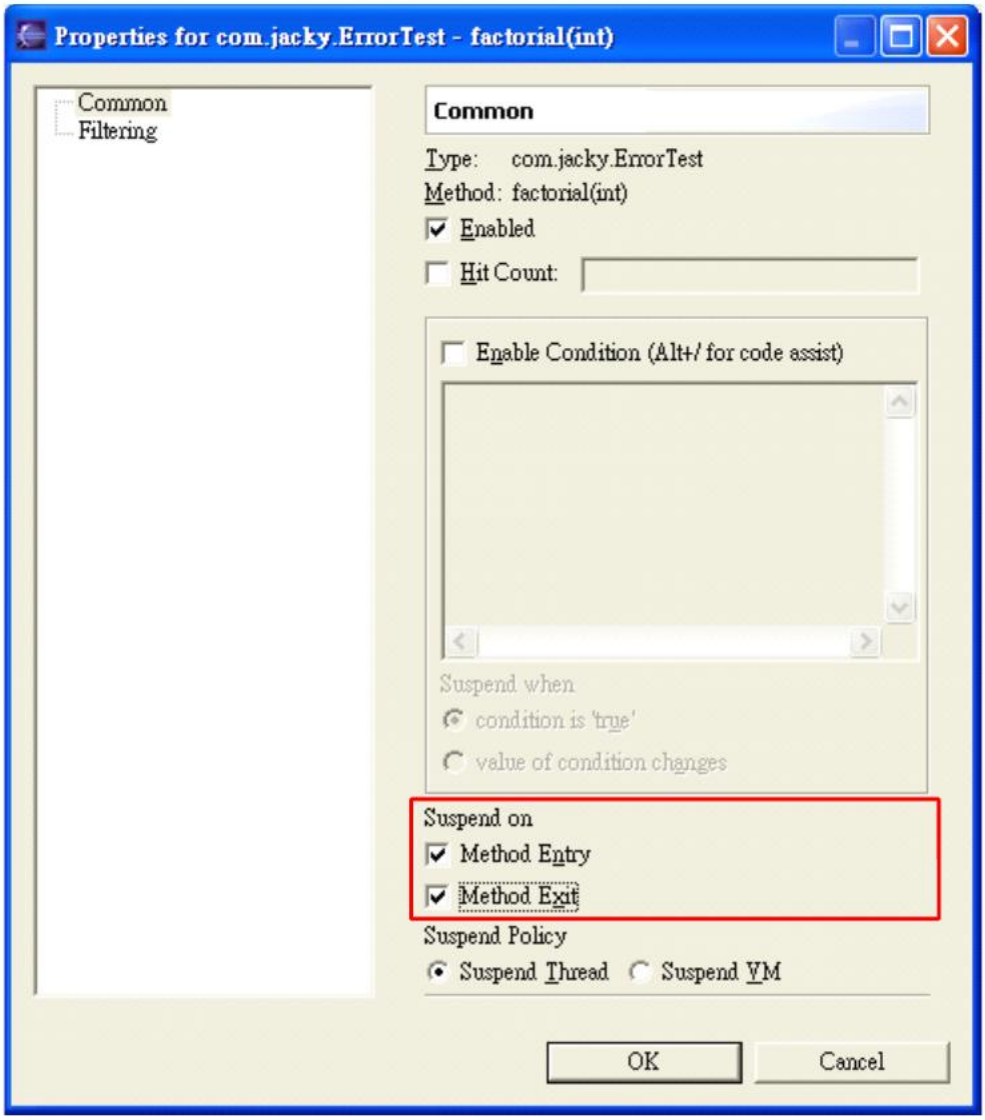


图 5.26

5.9 异常岔断点(Exception Breakpoint)

当例外发生时，可以暂停程序执行。如果程序会抛出例外事件，诸如 Null 例外事件，而且不知道这个例外事件是从何时(或是何处)发生的，这个岔断点就很有用。可以暂停程序，观看程序中抛出例外事件时，出了什么事。

设定异常岔断点，在「Java」视景的编辑器中，选「Run」→「Add Java Exception Breakpoint」。

新的异常岔断点会出现在「Debug」视景中的「Breakpoints」视图里，对该岔断点按右键，选择 Properties。

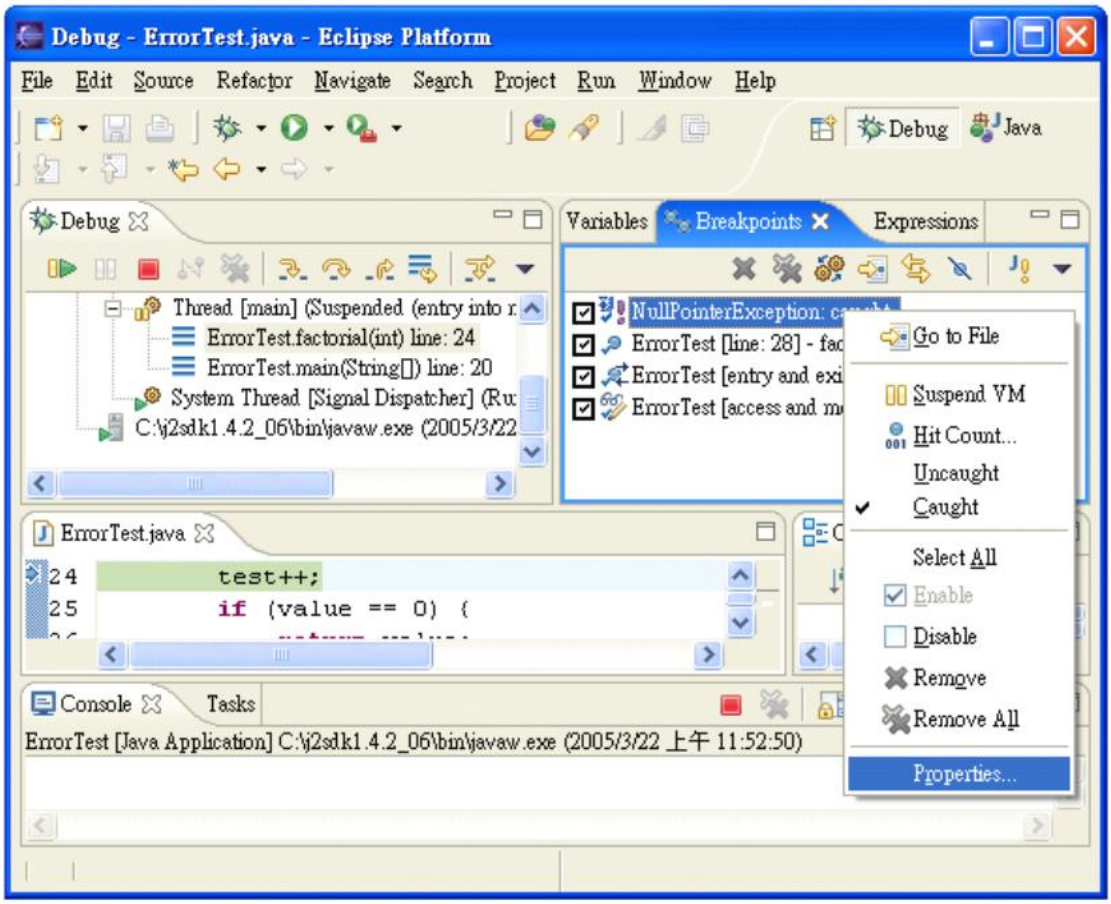


图 5.27

开启 Breakpoints Properties 窗口，选项设定跟之前的岔断点差不多，特别的是 Suspend on 的选项

Caught Exception 决定岔断点生效之时是例外事件被捕捉

Uncaught Exception 决定岔断点生效之时是例外事件没被捕捉

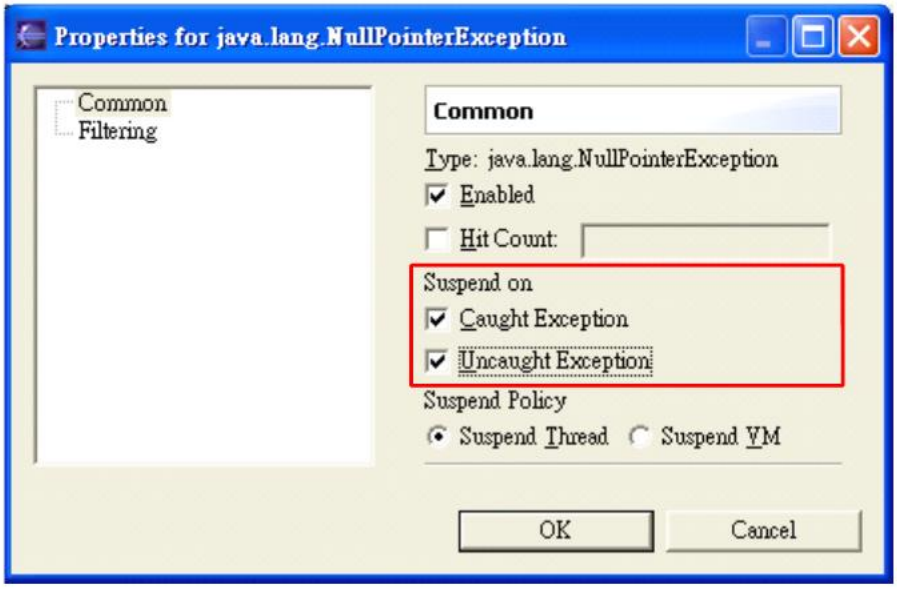


图 5.28

5.10 Java 表示式及变更某些值

在除错时，可以在「Expressions」视图中的详细资料窗格内输入表示式，选取表示式，按右键选择 Inspect。例如现在变量值是 6，在详细资料窗格内输入表示式 value + 1，选取表示式，按右键选择 Inspect

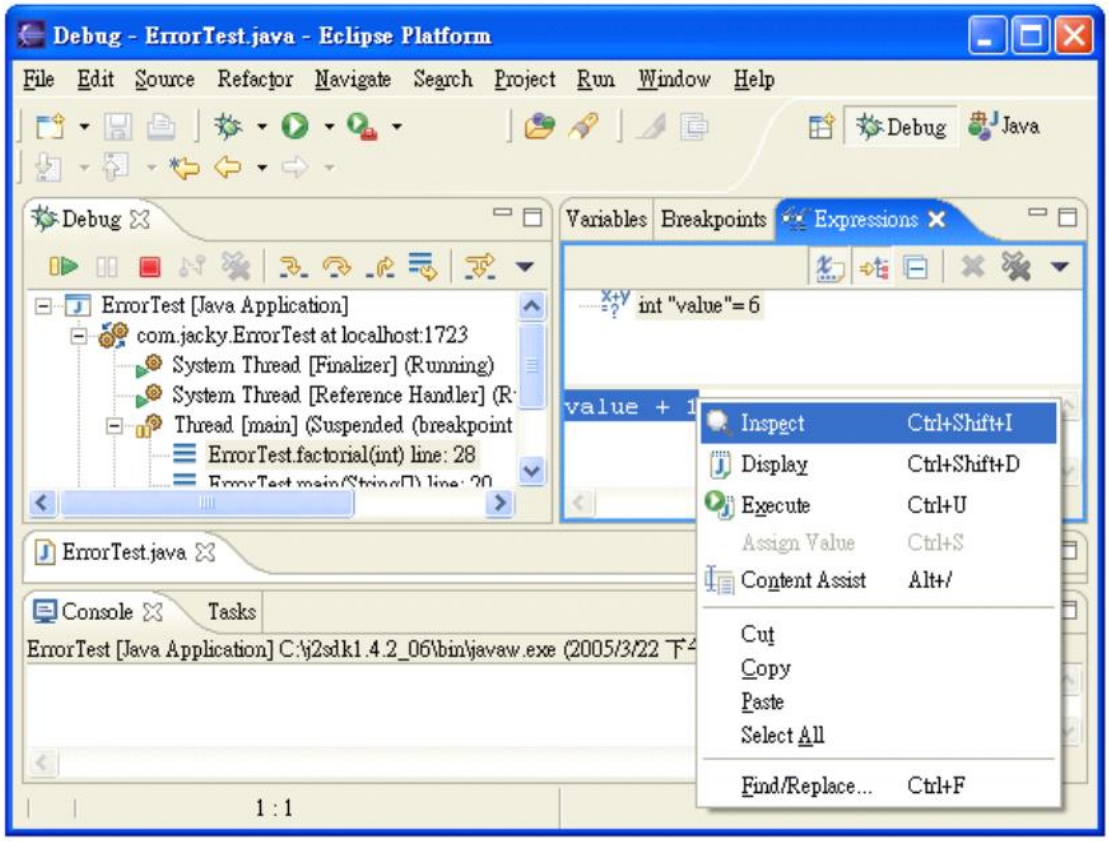


图 5.29

这样做会把 value + 1 加进「Expressions」视图中的表示清单。

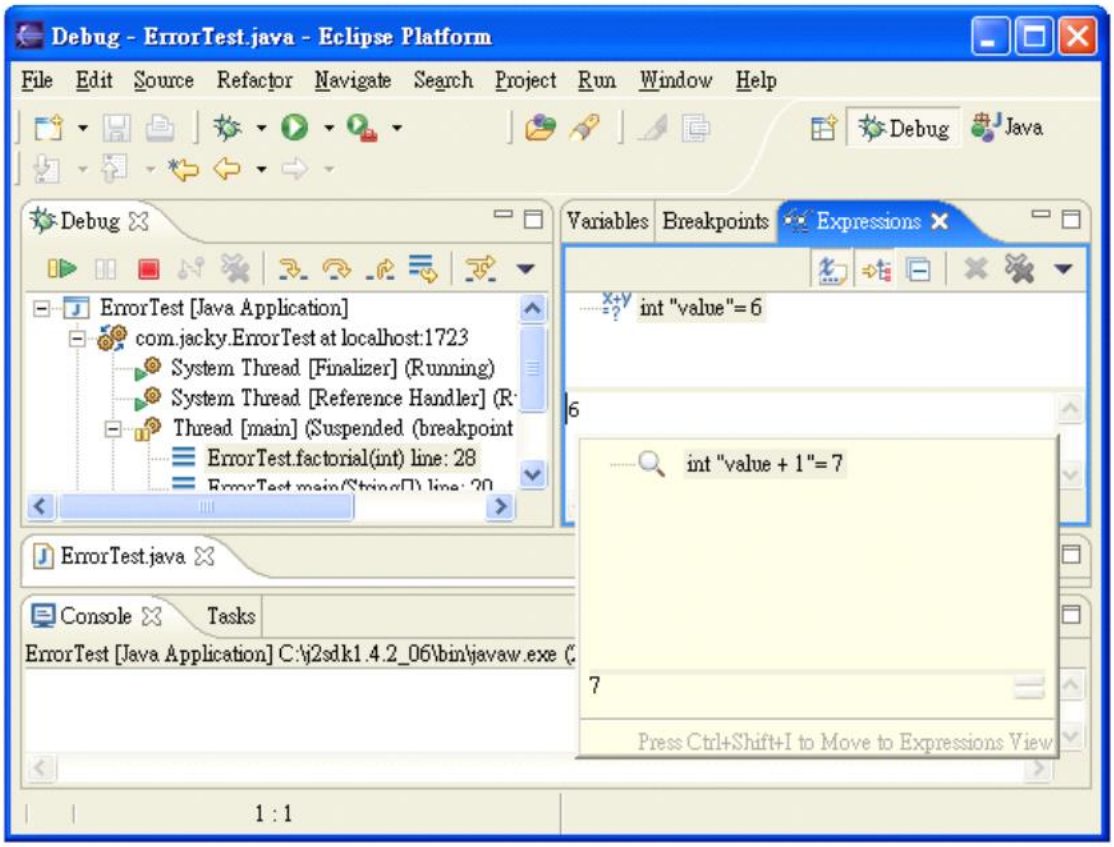


图 5.30

在除错时编辑字段和变量的值，只要对「Variables」视图中的字段或变量点两下，开启 Set Value 窗口，输入新值。例如在执行期间变更 value 的值为 5。

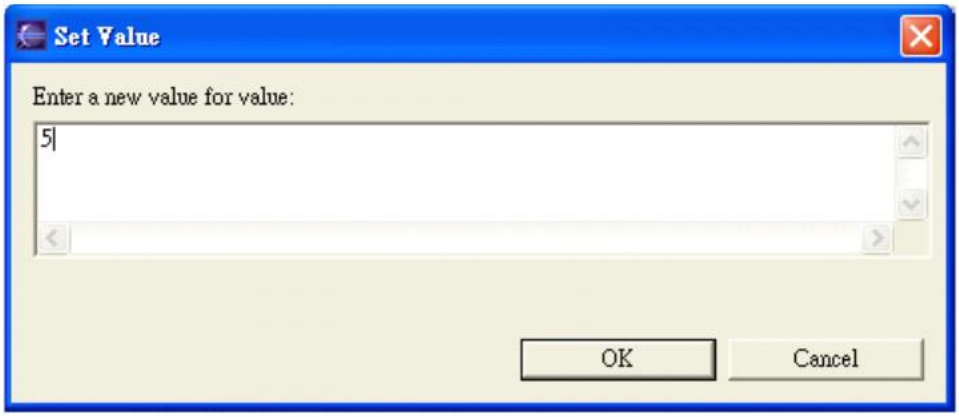


图 5.31

想检查程序针对不同测试值的反应，或是避开某些有问题的值，这个功能非常有用。

6.重构(Refactoring)

Java 程序重构的目标就是进行全系统程序代码变更，但不会影响程序的行为。Eclipse 提供有易于重构程序代码的协助。

重构工具支持若干在 Martin Fowler 所著的 Refactoring: Improving the Design of Existing Code, Addison Wesley 1999 一书中描述的转换，如撷取方法、列入区域变量等。

在执行重构作业时，可以先选择性地预览所有因某个重构动作而发生的变更，然后再决定是否实行。当预览重构作业时，系统将通知潜在的问题，而且将呈现一个清单，列出重构动作将执行的变更。如果未预览重构作业，系统将完整地进行变更，而且将显示任何产生的问题。如果侦测到不容许重构作业继续的问题，则这个作业将会中止，并显示问题清单。

重构指令可在一些 Java 视图（如：套件浏览器、概要）与编辑器的内容菜单中找到。有许多「看似简单」的指令，如移动和重新命名，实际上是重构作业，因为移动 Java 元素以及将它重新命名，通常都需要变更相依档。

6.1 重新命名

6.1.1 区域变量(Local Variable)

如果要将区域变量（或方法参数）重新命名，请执行下列动作：

I. 在 Java 编辑器中选取变数（或其参照）

II. 「Refactor」 → 「Rename」

(或是在编辑器按右键，选取「Refactor」 → 「Rename」)

出现 Rename Local Variable 窗口

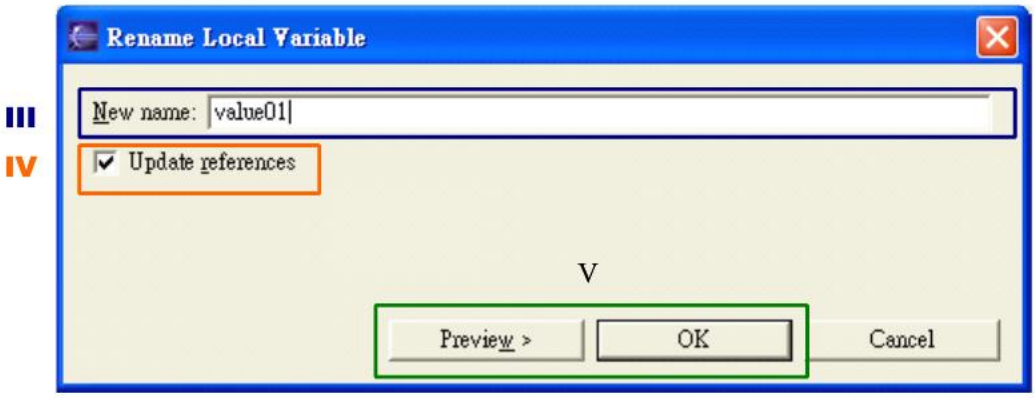


图 6.1

III. 设定新的 Name

IV. 如果不想更新已重新命名之区域变量的参照，请取消选取更新已重新命名之元素的参照勾选框。

V. 按一下 OK 以执行快速的重构作业，或按一下 Preview 以执行受控制的重构作业。

VI. 预览窗口会显示重构要更动的部份

VII. 下半部的窗格显示两者的比较

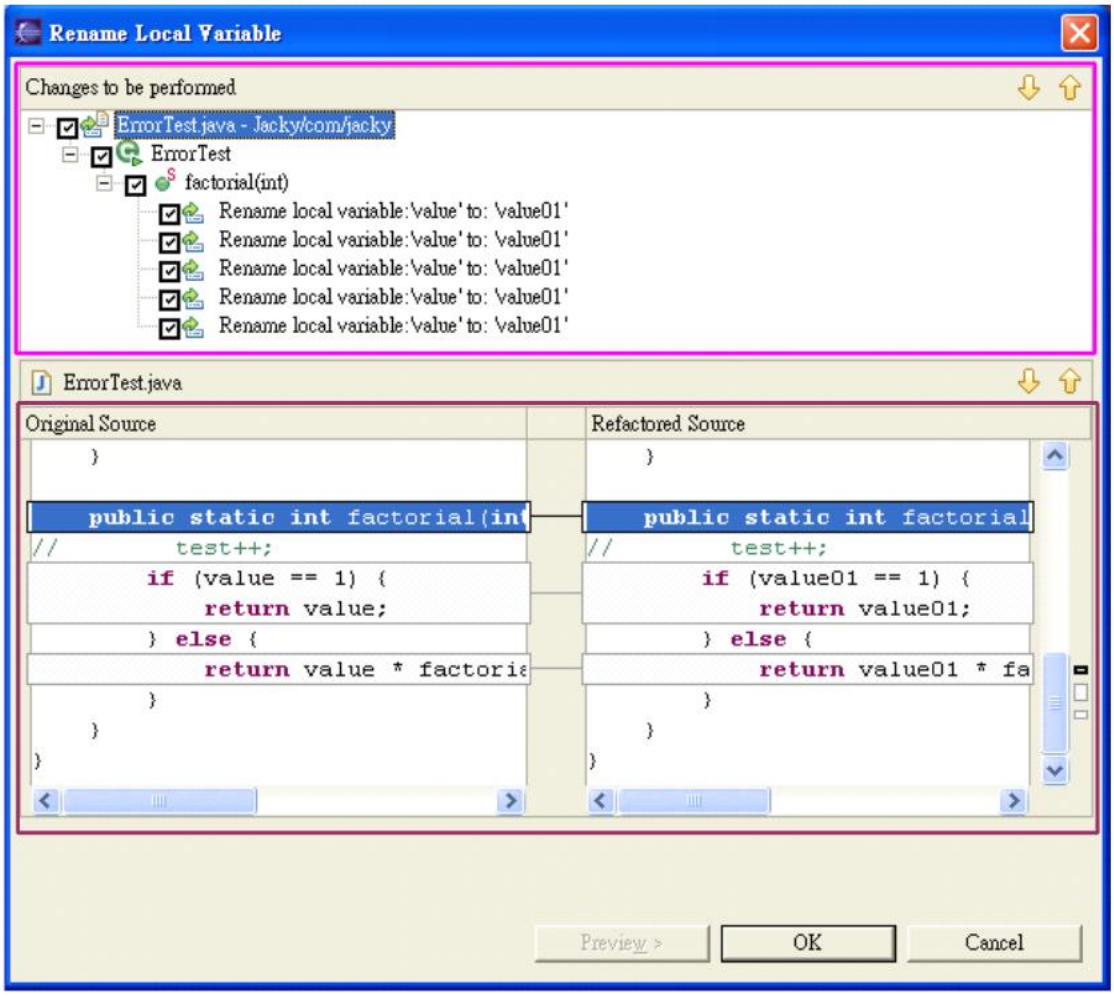


图 6.2

6.1.2 字段(Field)

如果要将字段重新命名，请执行下列动作：

I. 在 Java 编辑器中选取字段

II. 「Refactor」→「Rename」

(或是在编辑器按右键，选取「Refactor」→「Rename」)

出现 Rename Field 窗口

图 6.3

III. 设定新的 Name

IV. 如果不想更新已重新命名之字段的参照，请取消选取更新已重新命名之元素的参照勾选框。

V. 如果想更新字符串文字中的参照，请选取更新字符串文字中的参照勾选框。

VI. 如果重构作业找到要重新命名之字段的存取元(getter/setter)方法，则重构作业会建议亦重新命名这些方法（并更新其所有参照）：

如果想要重新命名 Getter，请选取重新命名 Getter 勾选框

如果想要重新命名 Setter，请选取重新命名 Setter 勾选框

VII. 按一下 OK 以执行快速的重构作业，或按一下 Preview 以执行受控制的重构作业。

VIII. 预览窗口会显示重构要更动的部份

IX. 下半部的窗格显示两者的比较

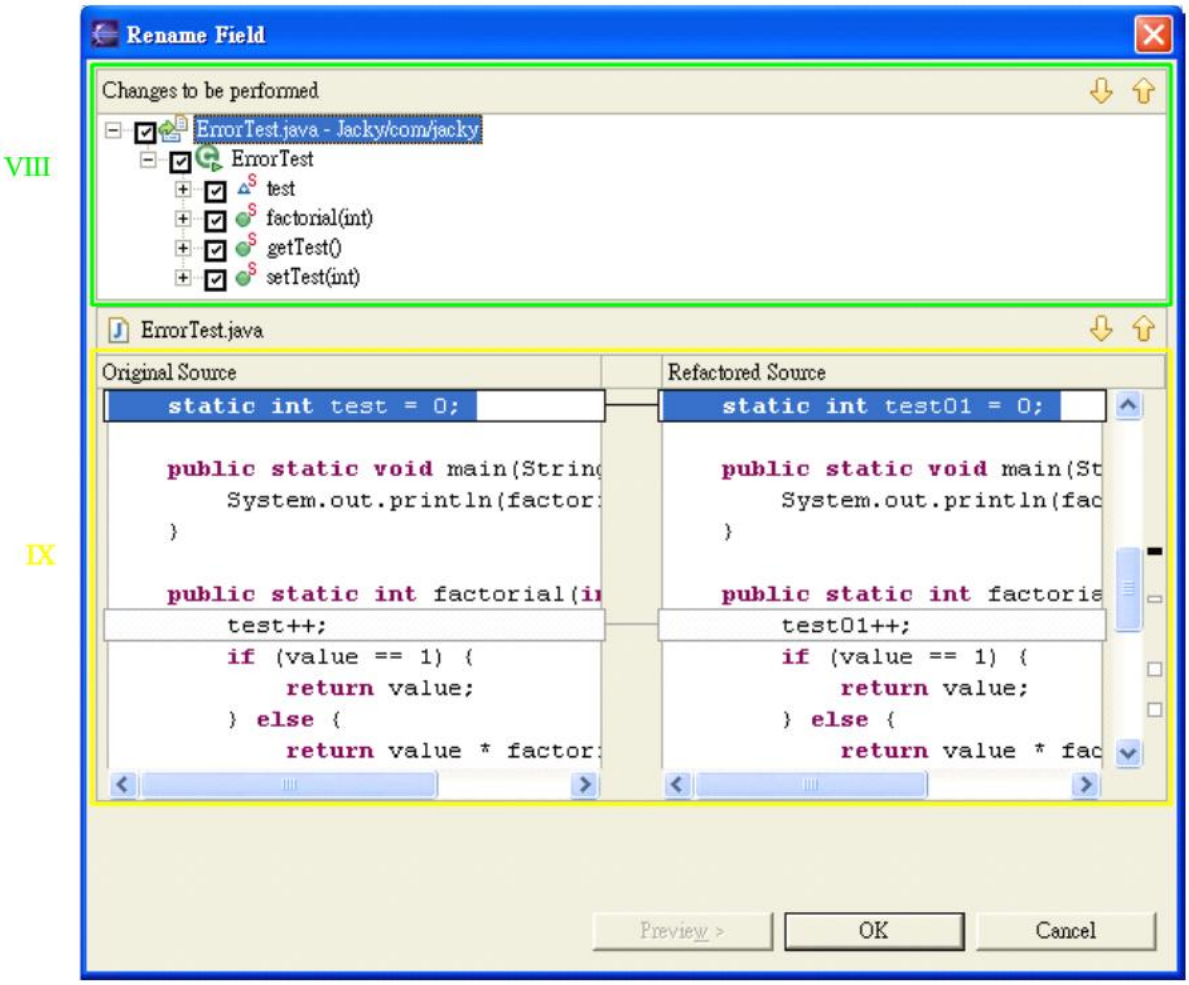


图 6.4

6.1.3 方法(Method)

如果要将方法重新命名，请执行下列动作：

I. 在 Java 编辑器中选取方法的名称

II. 「Refactor」→「Rename」

(或是在编辑器按右键，选取「Refactor」→「Rename」)

出现 Rename Method 窗口

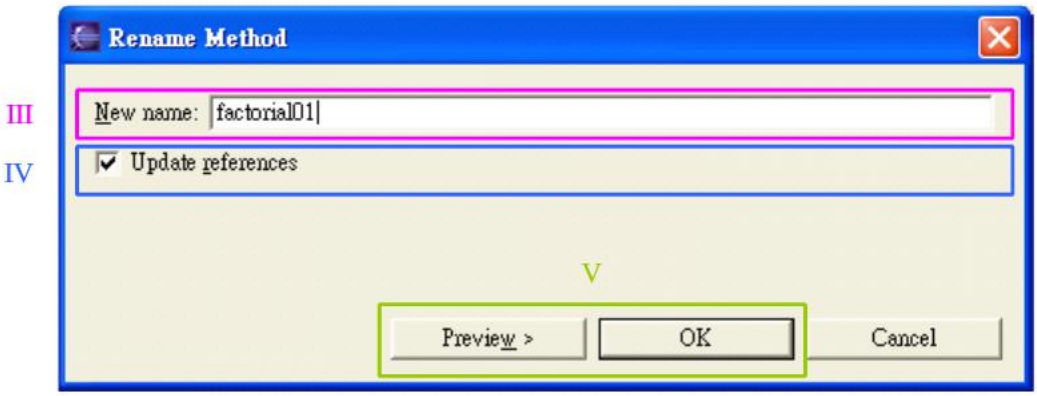


图 6.5

III. 设定新的 Name

IV. 如果不想更新已重新命名之字段的参照，请取消选取更新已重新命名之元素的参照勾选框。

V. 按一下 OK 以执行快速的重构作业，或按一下 Preview 以执行受控制的重构作业。

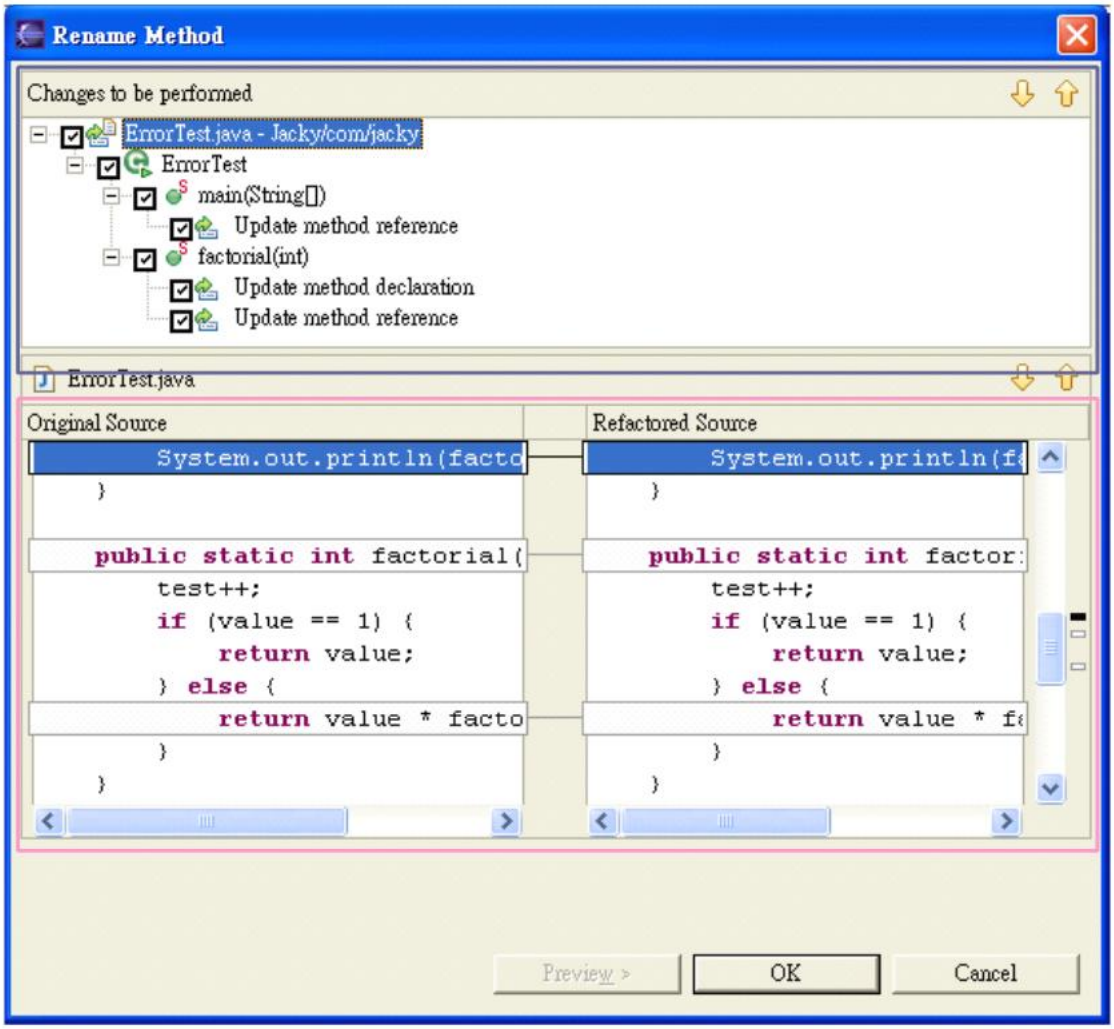


图 6.6

VI. 预览窗口会显示重构要更动的部份

VII. 下半部的窗格显示两者的比较

6.1.4 类别(Class)或是接口(Interface)

如果要将类别重新命名，请执行下列动作：

I. 在 Java 编辑器中选取类别的名称

II. 「Refactor」→「Rename」

(或是在编辑器按右键，选取「Refactor」→「Rename」)

出现 Rename Type 窗口

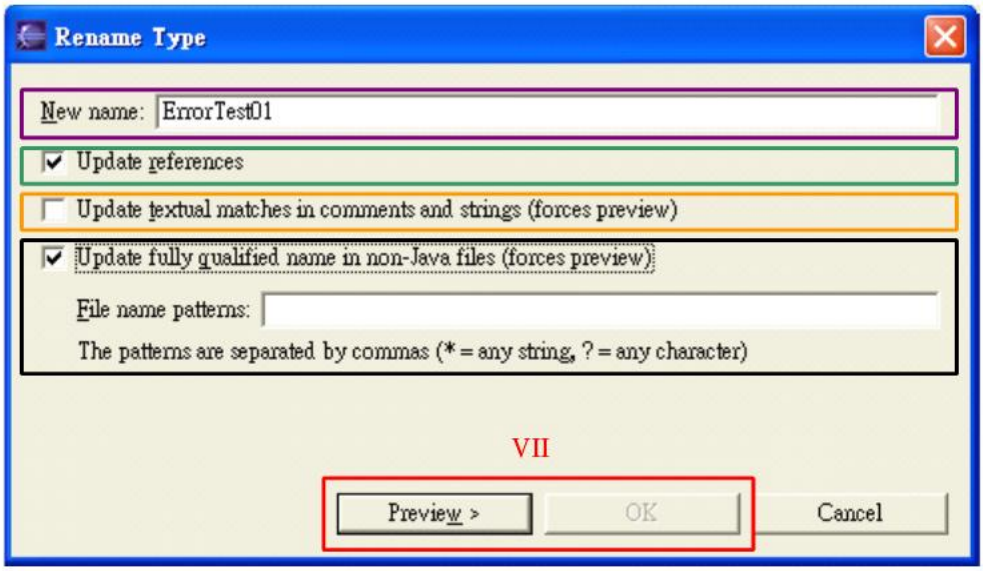


图 6.7

III. 设定新的 Name

IV. 如果不想更新已重新命名之类别或接口的参照，请取消选取更新已重新命名之元素的参照勾选框。

V. 如果想更新字符串文字中的参照，请选取更新字符串文字中的参照勾选框。

VI. 如果想更新一般（非 Javadoc）批注中的参照，请选取更新一般批注中的参照勾选框。

VII. 按一下确定以执行快速的重构作业，或按一下预览以执行受控制的重构作业。

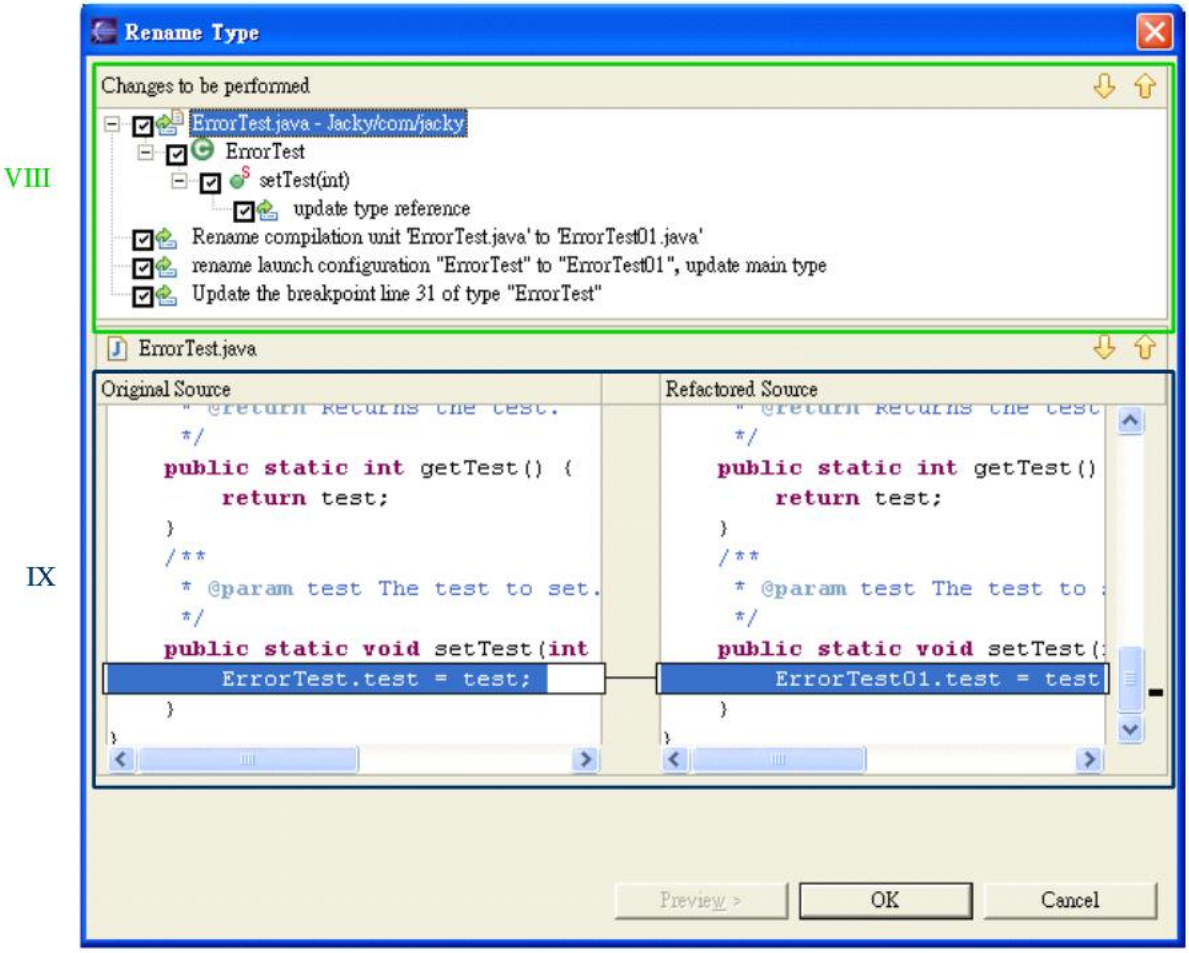


图 6.8

VIII. 预览窗口会显示重构要更动的部份

IX. 下半部的窗格显示两者的比较

6.1.5 套件(Package)

如果要将套件重新命名，请执行下列动作：

I. 在 Java 编辑器中选取套件的名称

II. 「Refactor」→「Rename」

(或是在编辑器按右键，选取「Refactor」→「Rename」)

出现 Rename Package 窗口

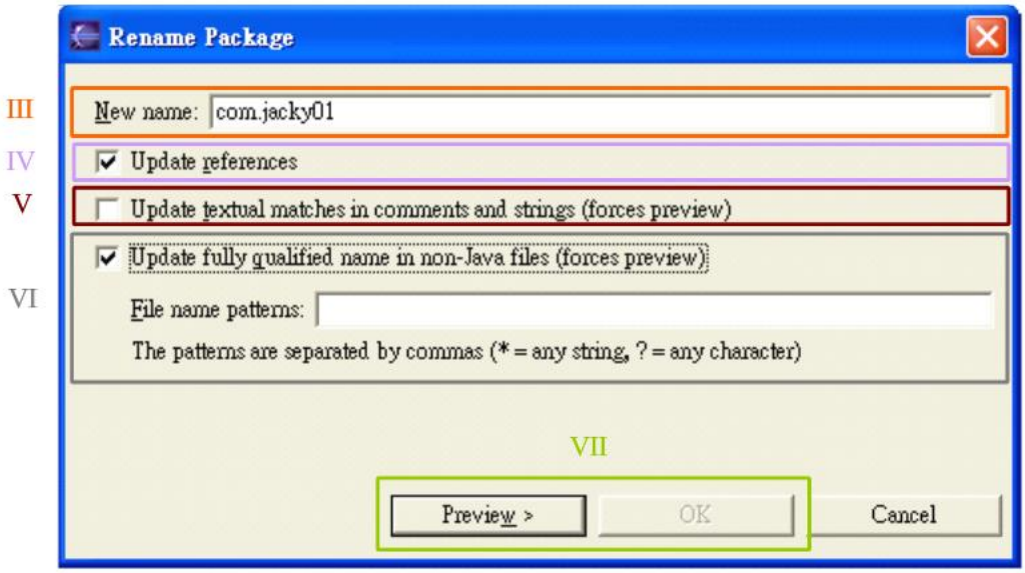


图 6.9

III. 设定新的 Name

IV. 如果不想更新已重新命名之类别或接口的参照，请取消选取更新已重新命名之元素的参照勾选框。

V. 如果想更新字符串文字中的参照，请选取更新字符串文字中的参照勾选框。

VI. 如果想更新一般（非 Javadoc）批注中的参照，请选取更新一般批注中的参照勾选框。

VII. 按下预览，以查看变更的预览，或按下确定，执行重构作业，不查看预览。

VIII

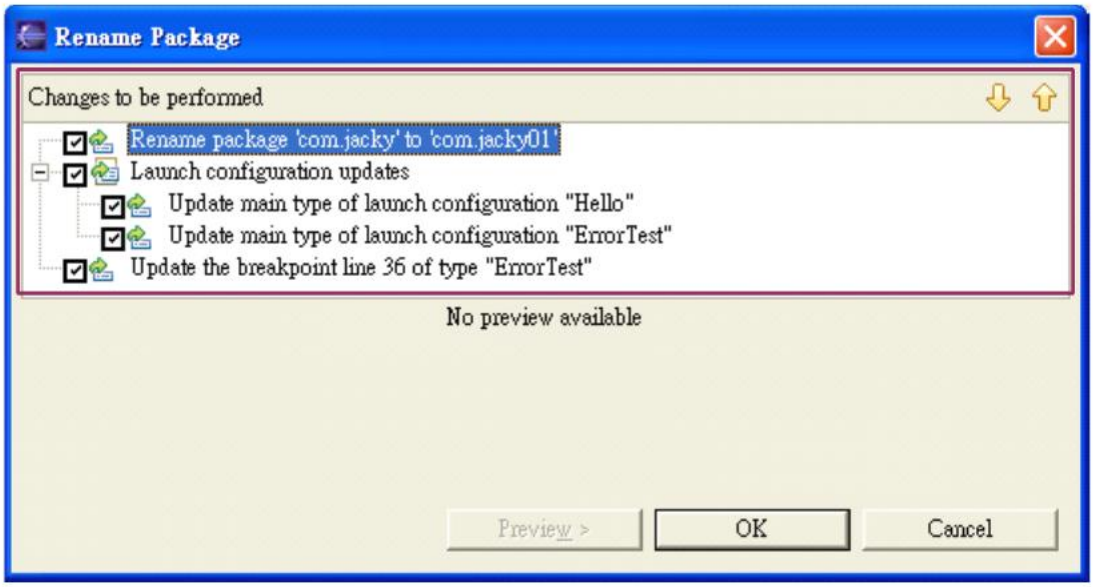


图 6.10

VIII. 预览窗口会显示重构要更动的部份

6.2 撷取(Extracting)

6.2.1 撷取常数(Extracting a Constant)

如果要撷取常数，请执行下列动作：

I. 在 Java 编辑器中选取常数

II. 「Refactor」→「Extract Constant...」

(或是在编辑器按右键，选取「Refactor」→「Extract Constant...」)

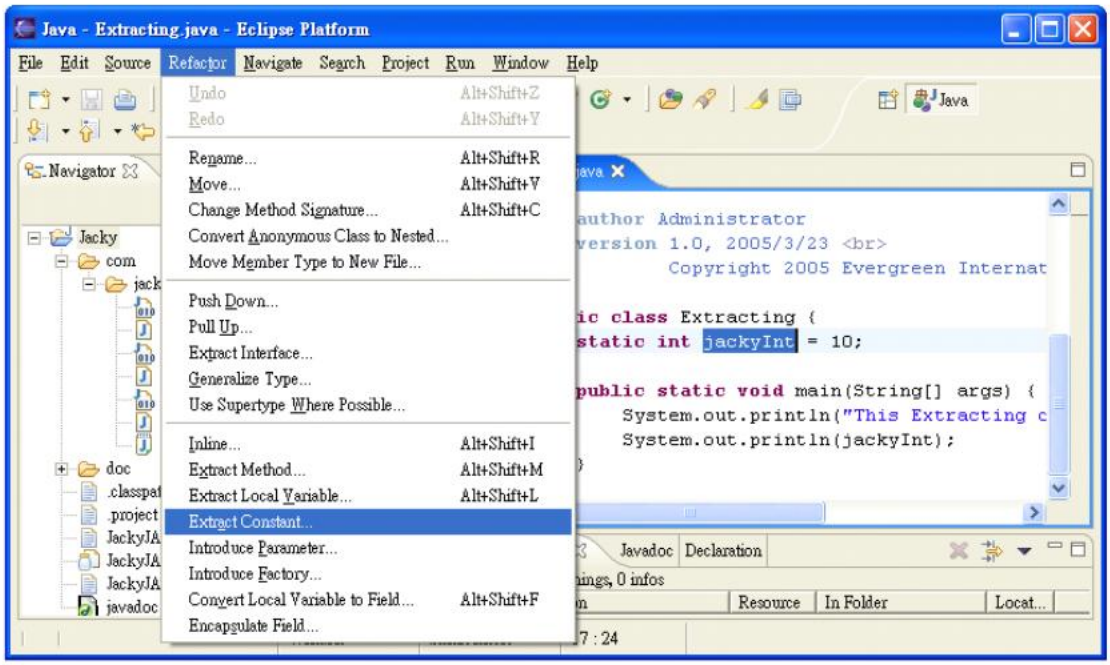


图 6.11

出现 Extract Constan 窗口

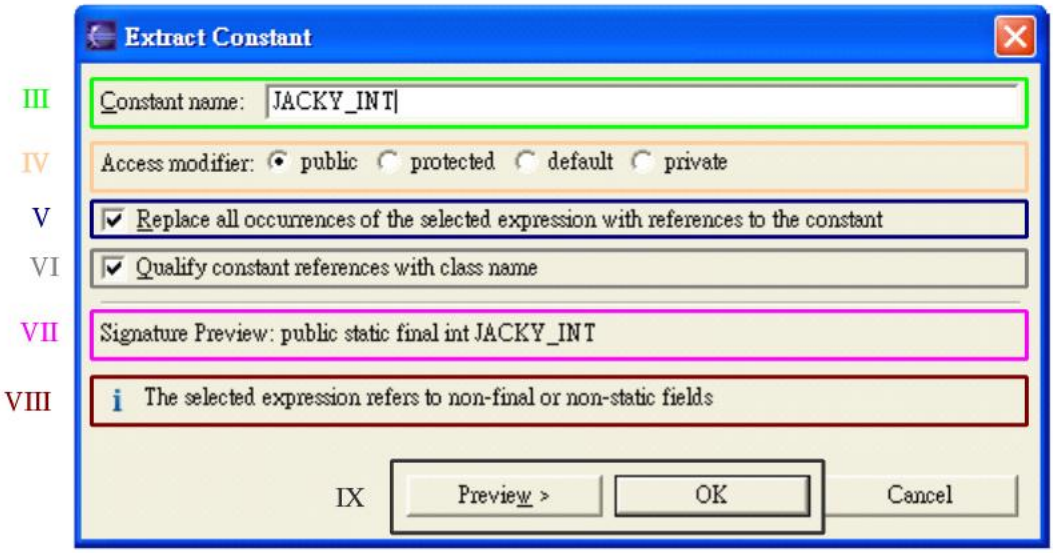


图 6.12

III. 设定新的 Name

IV. 在存取修饰元清单中，指定方法的可见性（public、default、protected 或 private）。

V. 在呼叫重构作业时，如果只想更换所选的表式示，可选择性地清除将所有出现所选表示式之处换成常数的参照勾选框

VI. 变量名称前是否要带限定者的名称。

若选取 Qualify constant reference with class name 勾选框

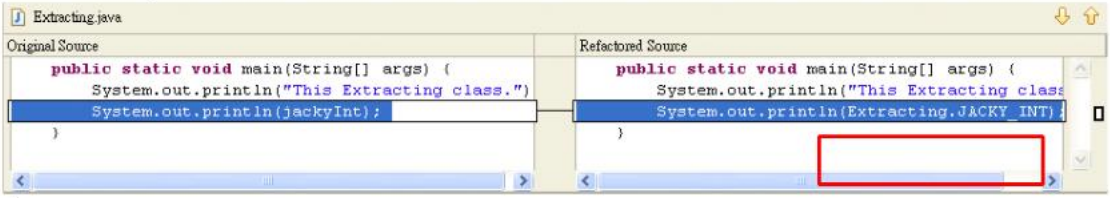


图 6.13

若没有选取 Qualify constant reference with class name 勾选框

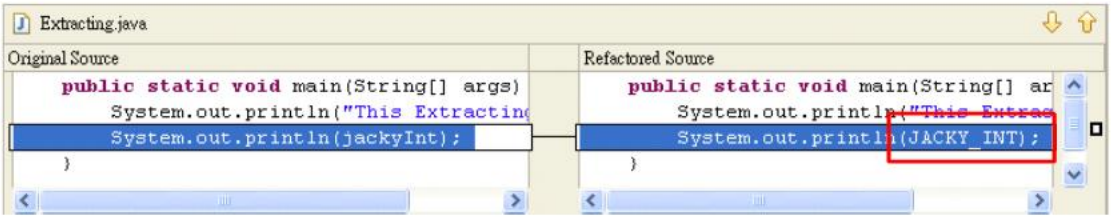


图 6.14

VII. 立即显示设定后的状态。

VIII. 讯息。

若没有符合 Java 命名规则，会出现警告的讯息

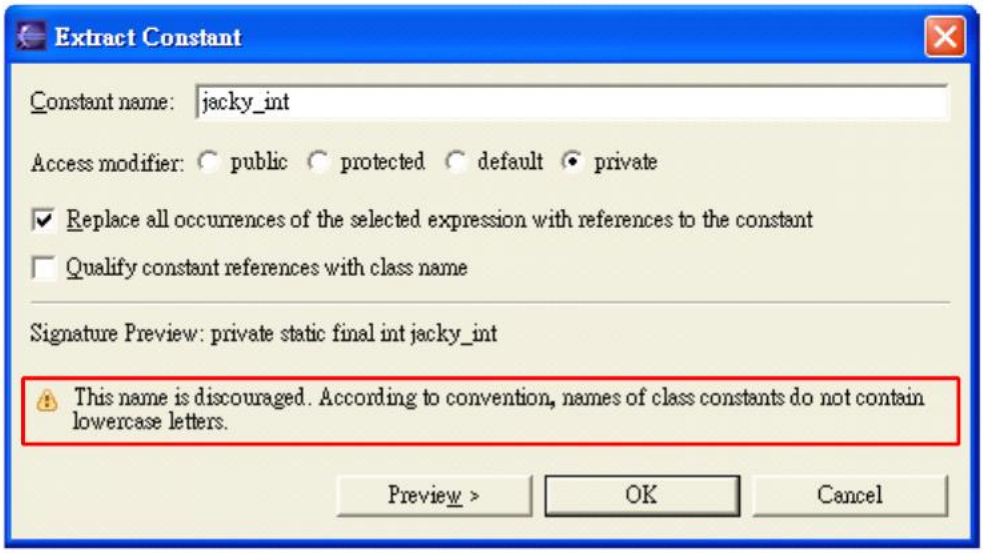


图 6.15

IX. 按一下确定以执行快速的重构作业，或按一下预览以执行受控制的重构作业。

X

XI

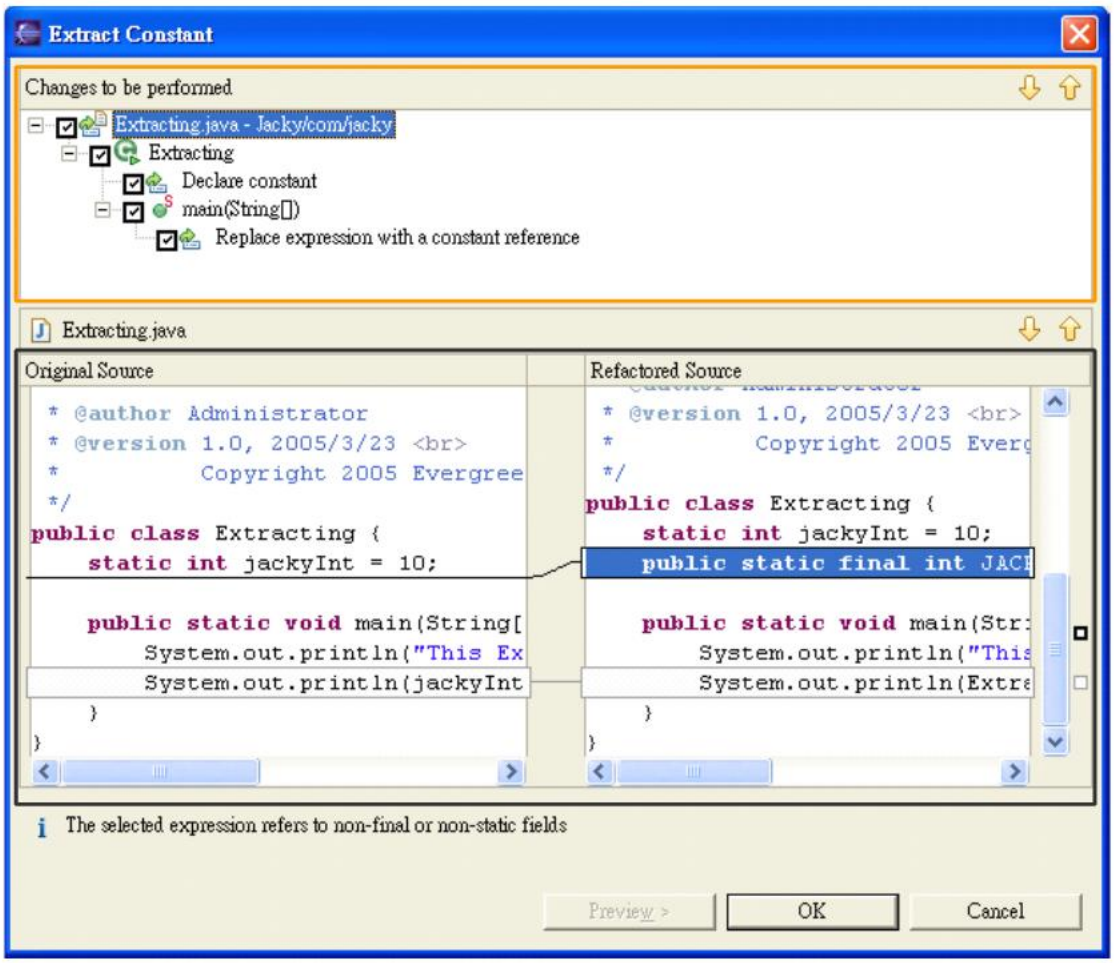


图 6.16

X. 预览窗口会显示重构要更动的部份

XI. 下半部的窗格显示两者的比较

程序代码如下：

```
public class Extracting {
    static int jackyInt = 10;
    public static void main(String[] args) {
        System.out.println("This's Extracting class.");
        System.out.println(jackyInt);
    }
}
```

6.2.2 撷取区域变量(Extracting a Local Variable)

如果要撷取区域变量，请执行下列动作：

I. 在 Java 编辑器中选取区域变量

II. 「Refactor」→「Extract Local Variable...」

(或是在编辑器按右键，选取「Refactor」→「Extract Local Variable...」)

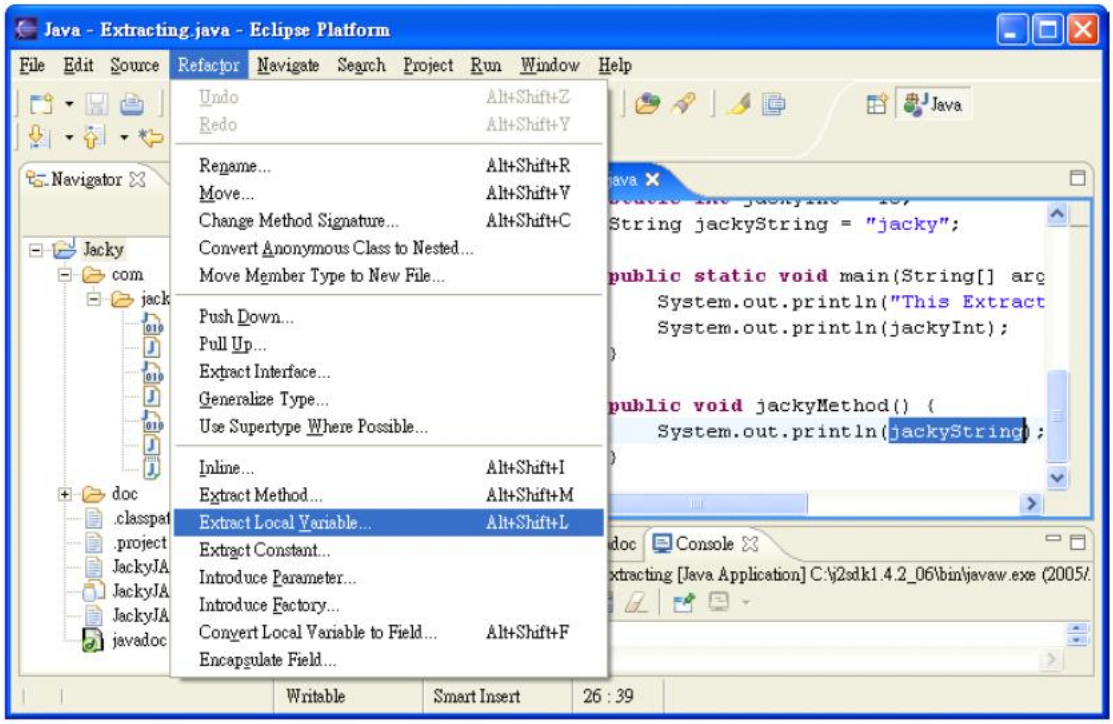


图 6.17

出现 Extract Local Variable 窗口

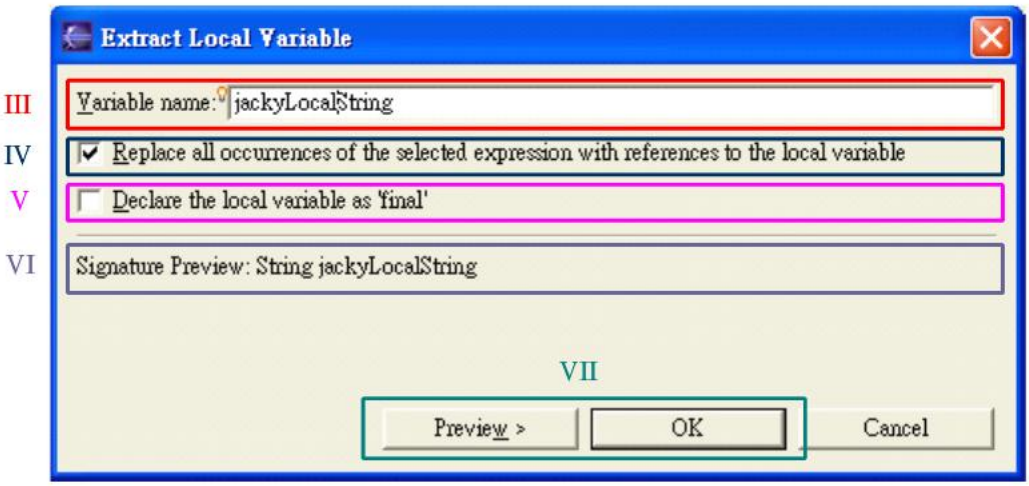


图 6.18

III. 设定新的 Name

IV. 在呼叫重构作业时，如果只想更换所选的表式示，可选择性地清除将所有出现所选表示式之处换成区域变量的参照勾选框。

V. 可选择性地选取将区域变量定义成'final'。

VI. 立即显示设定后的状态。

若没有符合 Java 命名规则，会出现警告的讯息

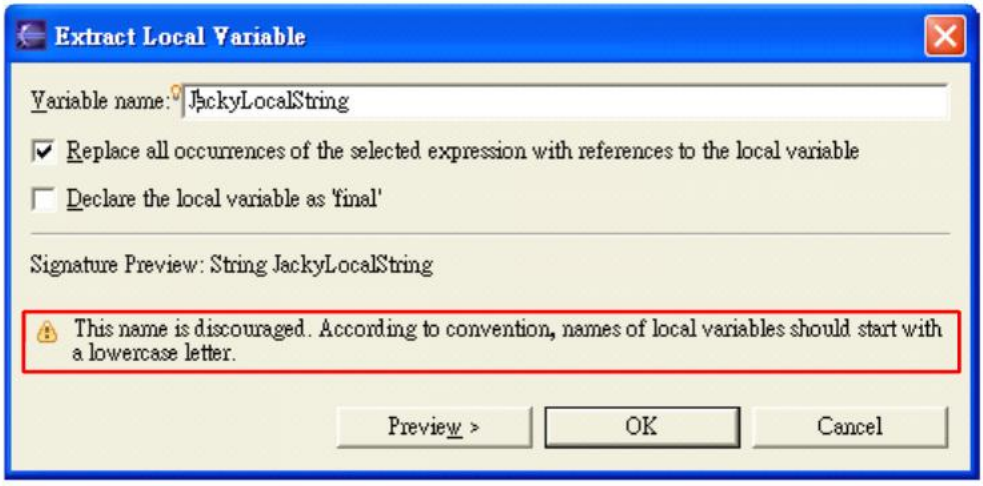
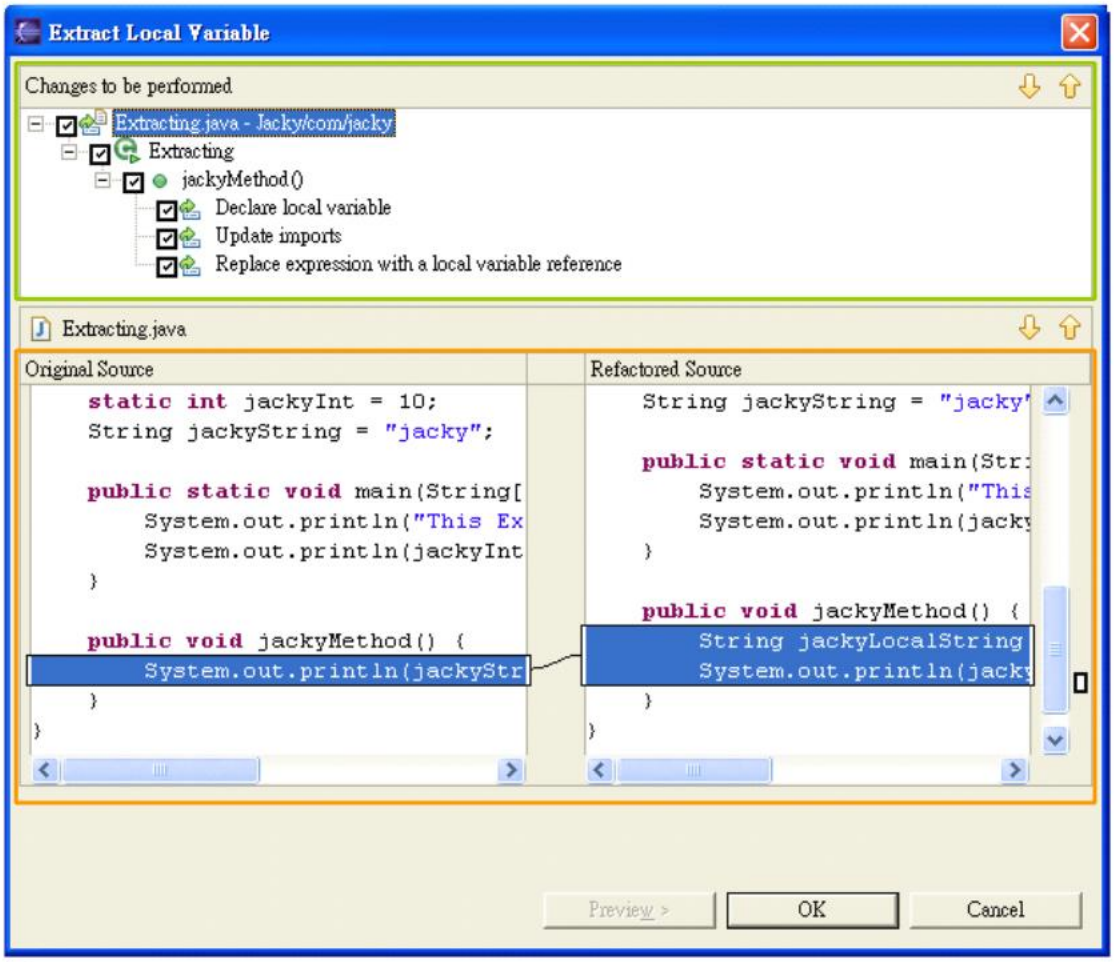


图 6.19

VII. 按下预览，以查看变更的预览，或按下确定，执行重构作业，不查看预览。



VIII

IX

图 6.20

VIII. 预览窗口会显示重构要更动的部份

IX. 下半部的窗格显示两者的比较

程序代码如下：

```
public class Extracting {
    static int jackyInt = 10;
    String jackyString = "jacky";
    public static void main(String[] args) {
        System.out.println("This's Extracting class.");
        System.out.println(jackyInt);
    }
    public void jackyMethod() {
        System.out.println(jackyString);
    }
}
```

6.2.3 撷取方法(Extracting a Method)

如果要撷取方法，请执行下列动作：

I. 在 Java 编辑器中选取方法

II. 「Refactor」→「Extract Method...」

(或是在编辑器按右键，选取「Refactor」→「Extract Method...」)

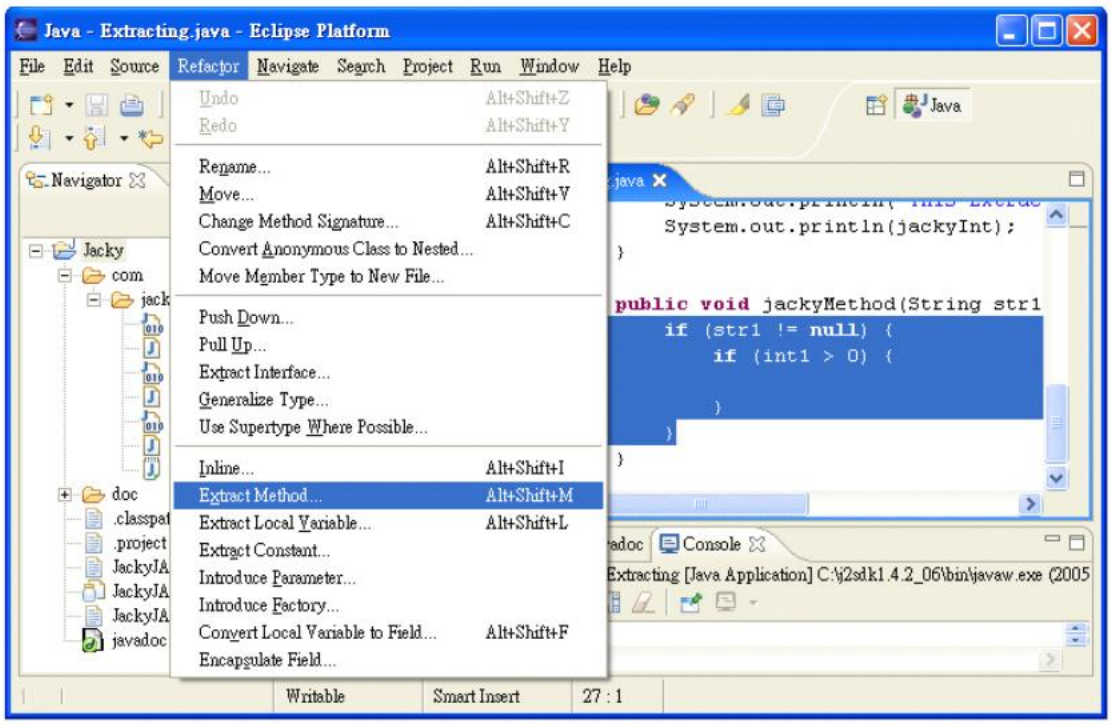


图 6.21

出现 Extract Methode 窗口

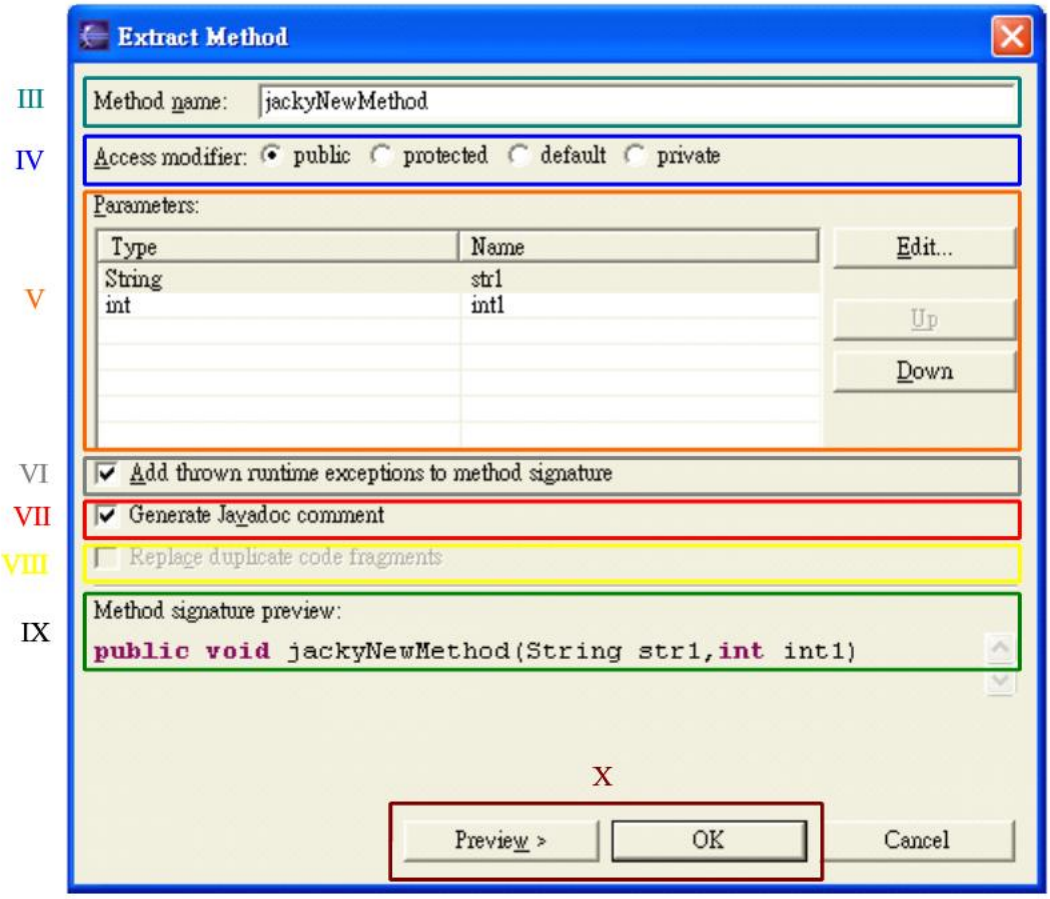


图 6.22

III. 设定新的 Name

IV. 在存取修饰元清单中，指定方法的可见性（public、default、protected 或 private）。

V. 可以重新排列新方法的参数，并且将它重新命名。

重新命名

图 6.23

重新排列

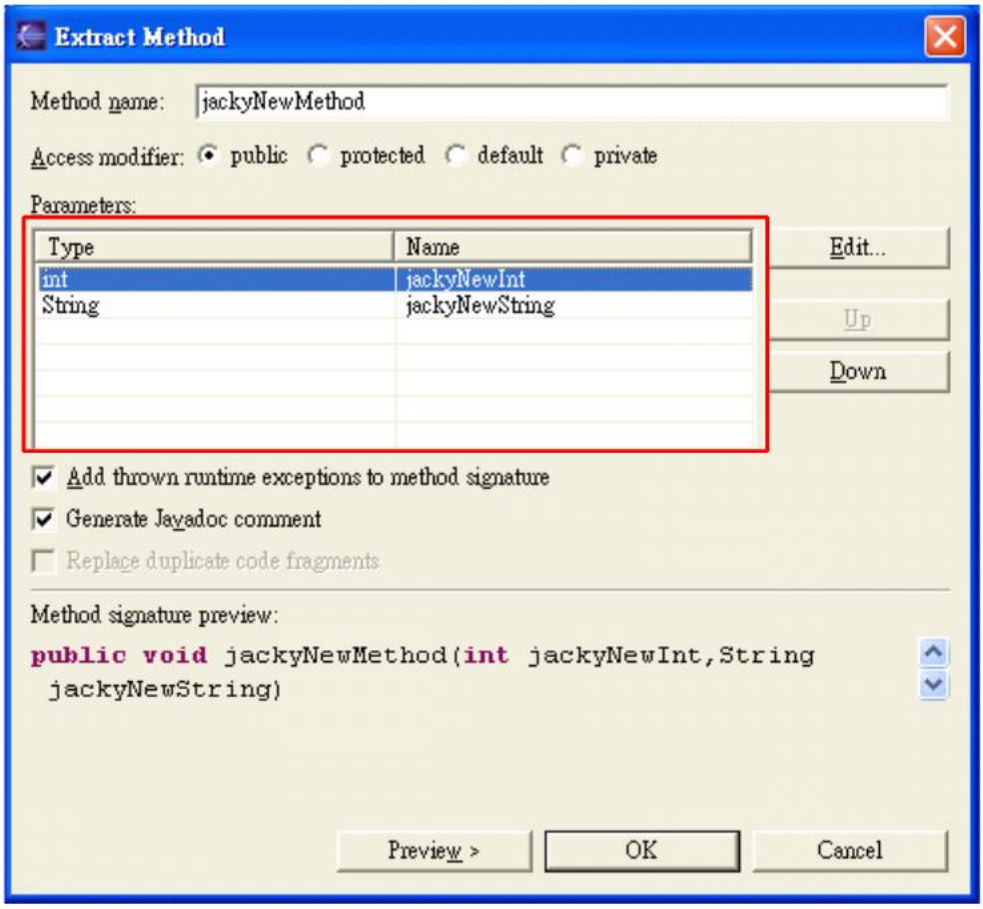


图 6.24

VI. 可以新增掷出执行时期异常状况到方法签章中，方法是选取对应的勾选框。

VII. 产生 Javadoc 批注

VIII. 取代重复的程序代码片段

IX. 立即显示设定后的状态。

若没有符合 Java 命名规则，会出现警告的讯息

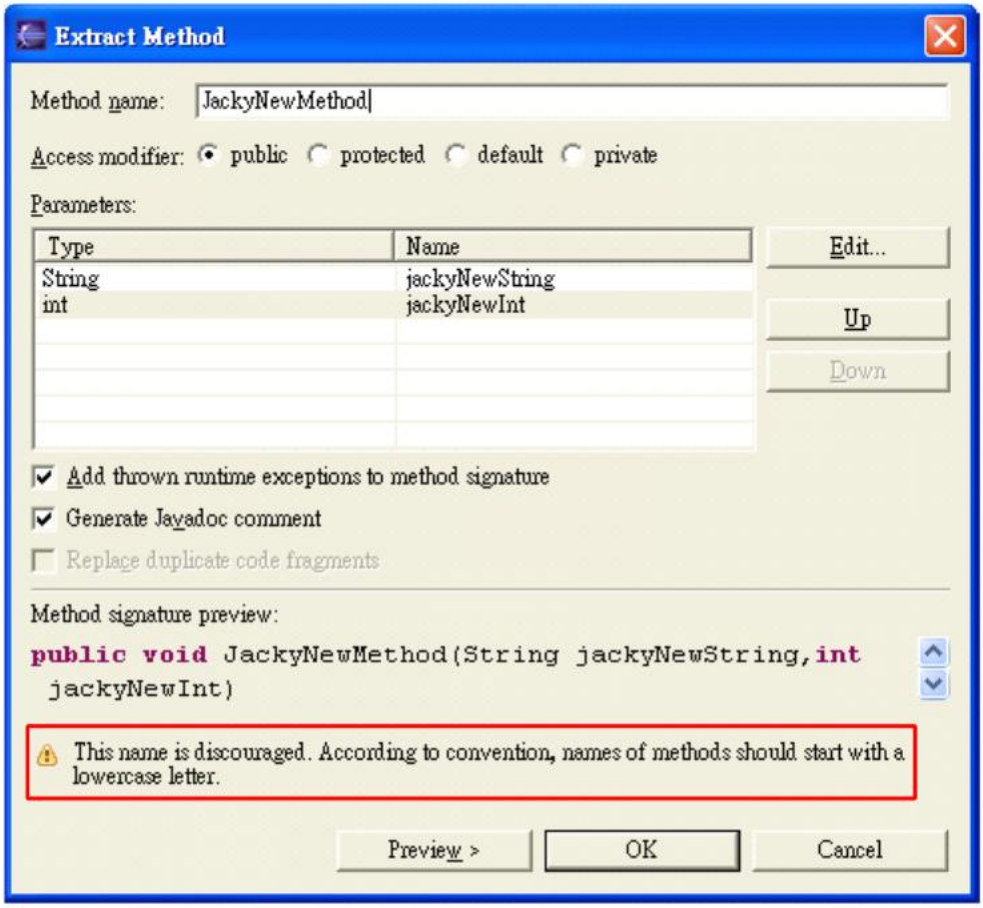
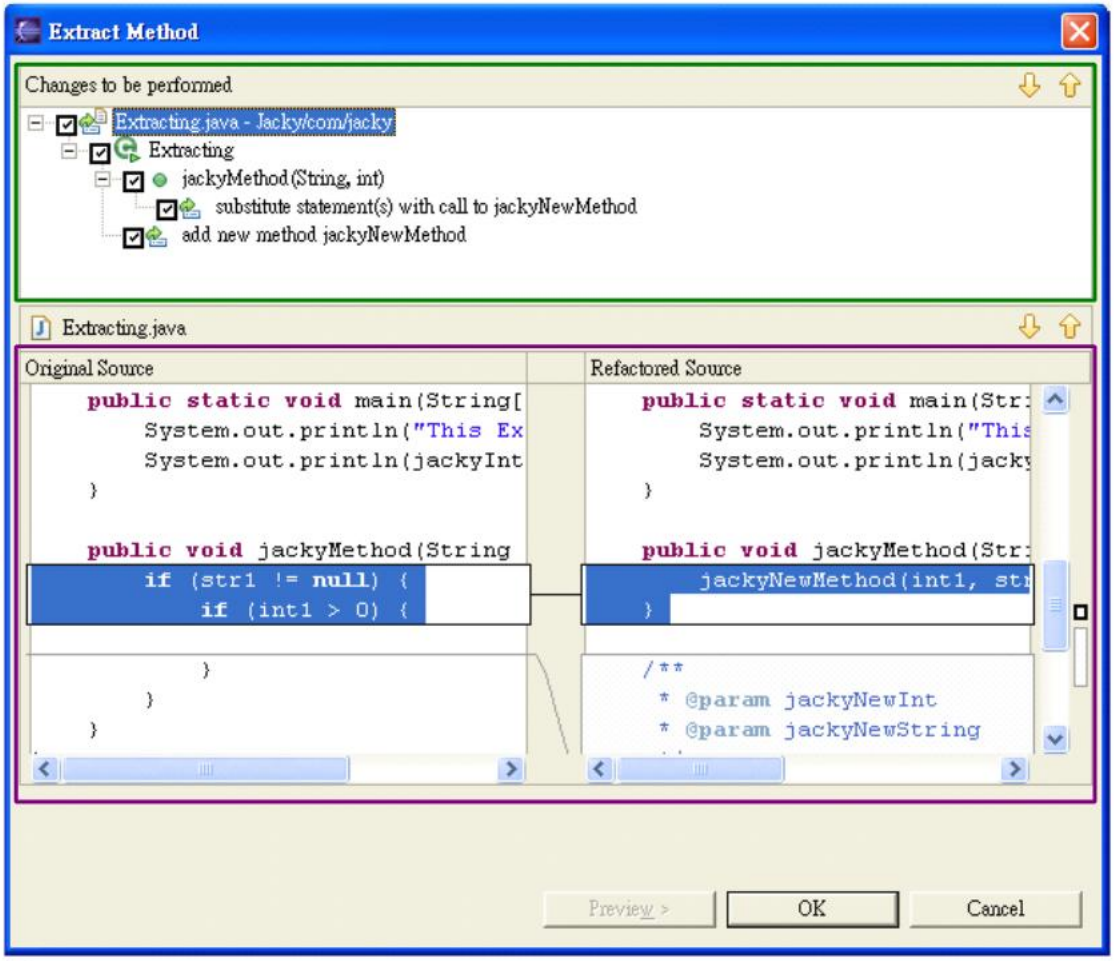


图 6.25

X. 按一下确定以执行快速的重构作业，或按一下预览以执行受控制的重构作业。



XI

XII

图 6.26

XI. 预览窗口会显示重构要更动的部份

XII. 下半部的窗格显示两者的比较

程序代码如下：

```
public class Extracting {
    static int jackyInt = 10;
    String jackyString = "jacky";
    public static void main(String[] args) {
        System.out.println("This's Extracting class.");
        System.out.println(jackyInt);
    }
    public void jackyMethod(String str1, int int1) {
        if (str1 != null) {
            if (int1 > 0) {
            }
        }
    }
}
```

6.3 列入(Inlining)

程序代码如下：

```
public class Inlining {
    static final int jackyInt = 10;
    public static void main(String[] args) {
        System.out.println("This's Inlining class.");
        System.out.println(jackyInt);
    }
    public void jackyMethod() {
        jackyMethod("jacky", 5);
    }
    public void jackyMethod(String str1, int int1) {
        String jackyString = "Jacky";
        if (jackyString != null) {
            System.out.println(jackyString);
        }
    }
}
```

6.3.1 列入常数(Inlining a Constant)

如果要列入常数，请执行下列动作：

I. 在 Java 编辑器中选取常数

II. 「Refactor」→「Extract Inlining...」

(或是在编辑器按右键，选取「Refactor」→「Extract Inlining...」)

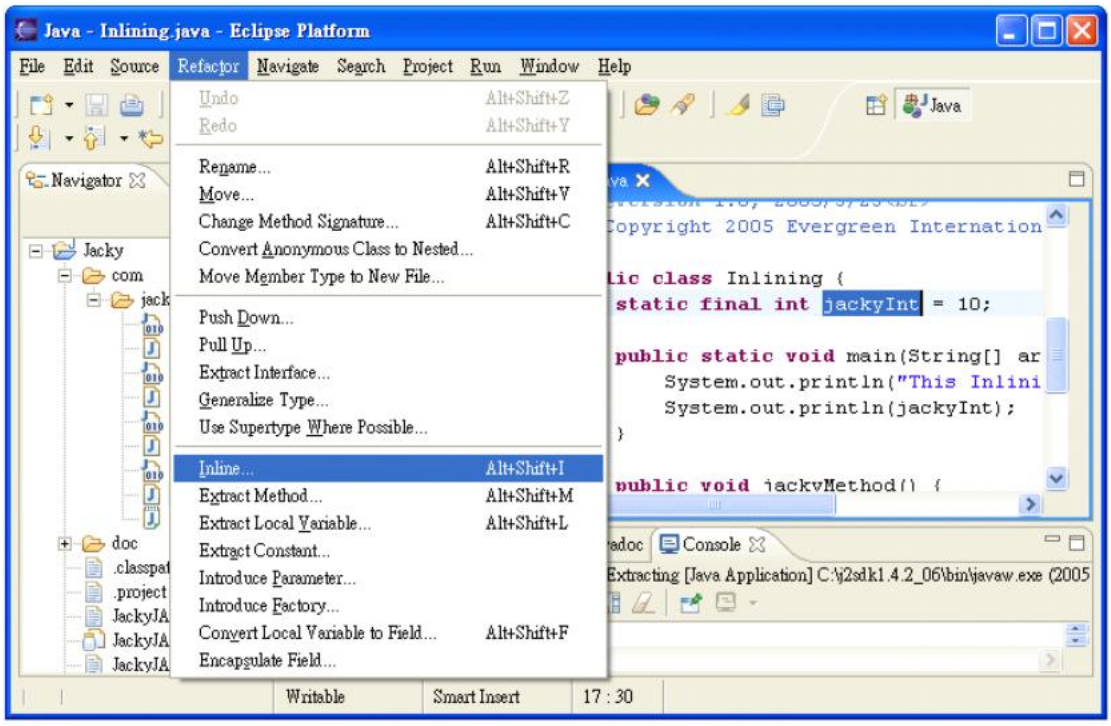


图 6.27

出现 Inline Constant 窗口

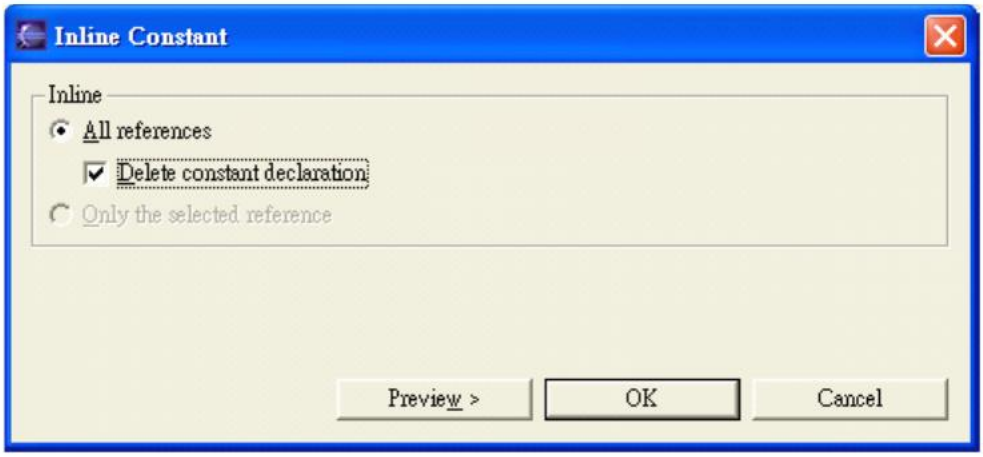


图 6.28

有选 Delete constant declaration，常数会被删除

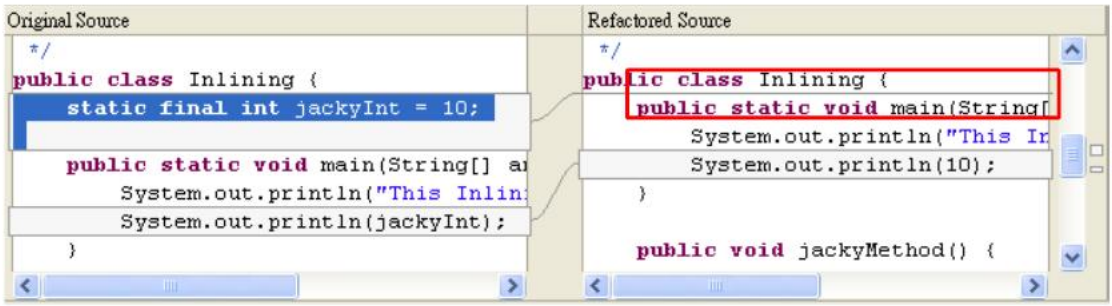


图 6.29

没有选 Delete constant declaration，常数不会被删除

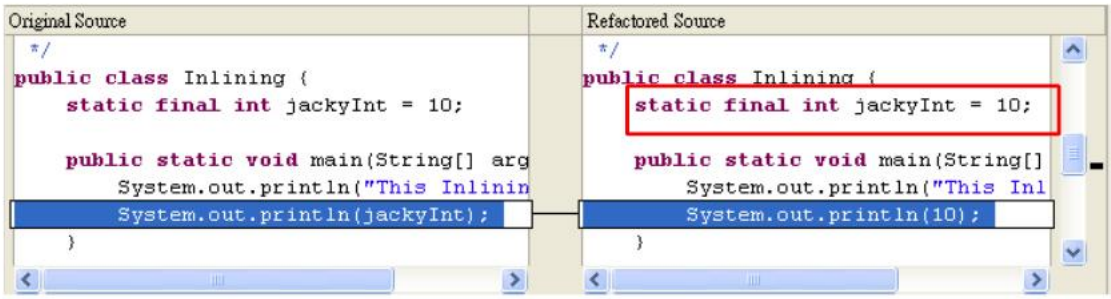


图 6.30

预览窗口会显示重构要更动的部份，下半部的窗格显示两者的比较

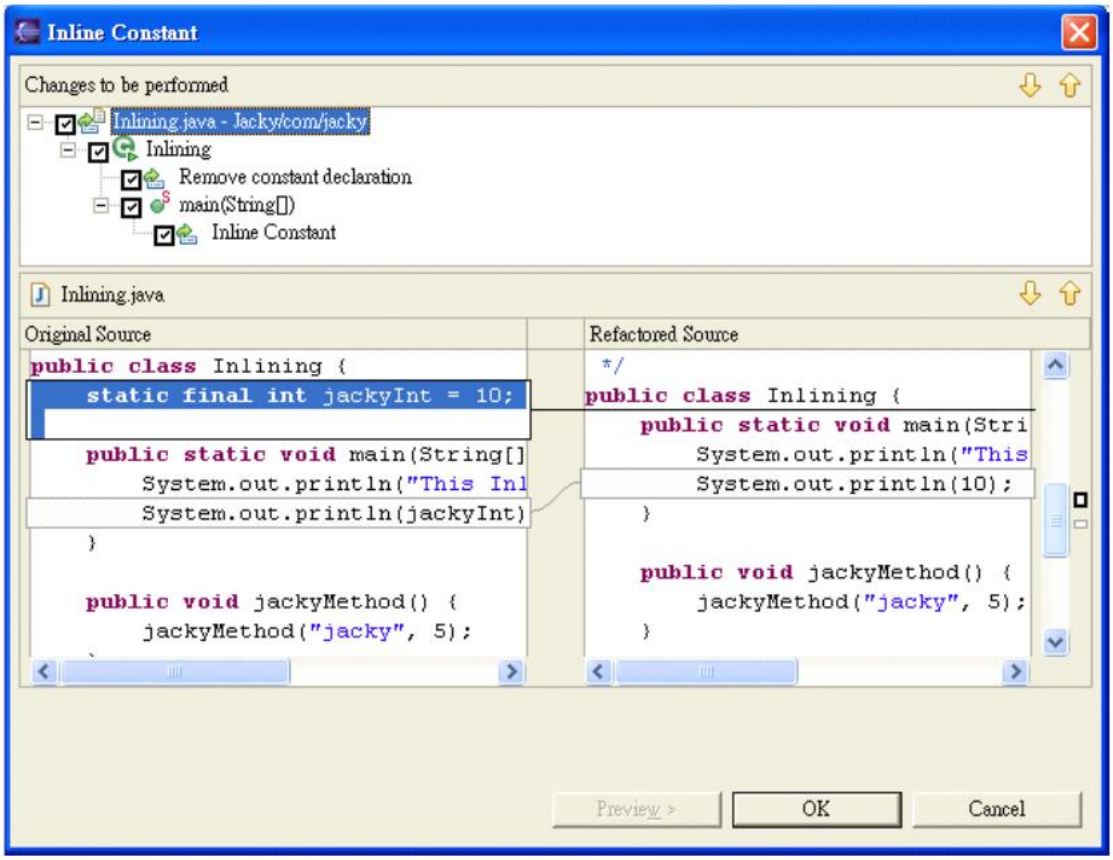


图 6.31

6.3.2 列入区域变量(Inlining a Local Variable)

如果要列入区域变量，请执行下列动作：

I. 在 Java 编辑器中选取区域变量

II. 「Refactor」→「Extract Inlining...」

(或是在编辑器按右键，选取「Refactor」→「Extract Inlining...」)

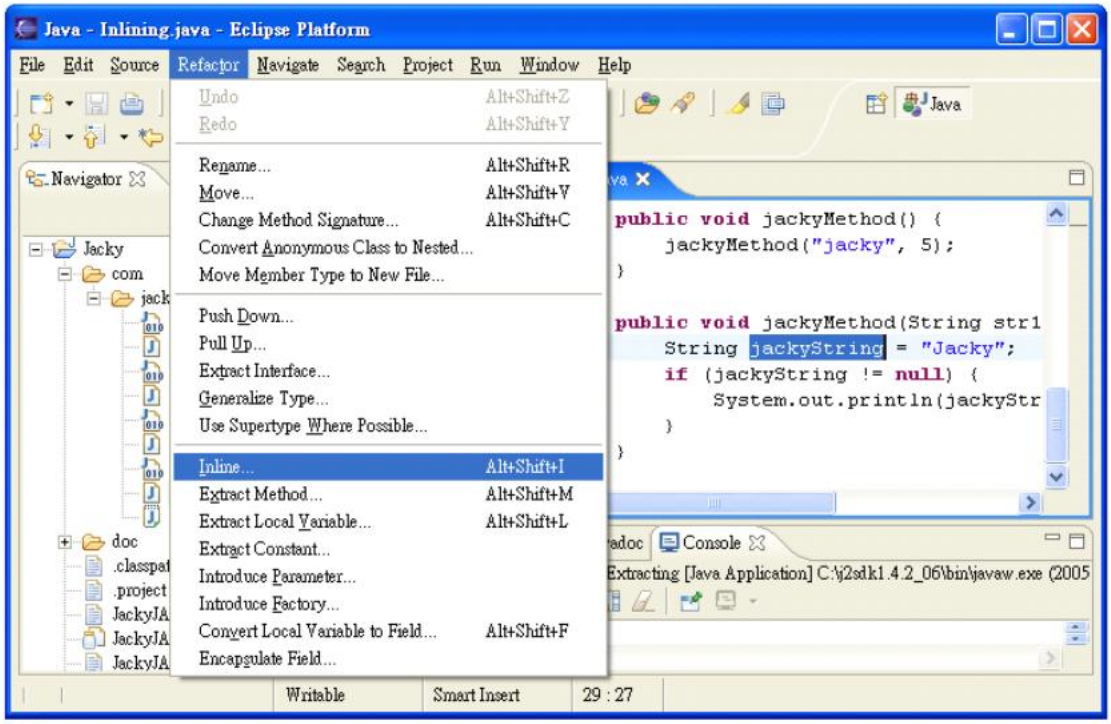


图 6.32

出现 Inline Local Variable 窗口

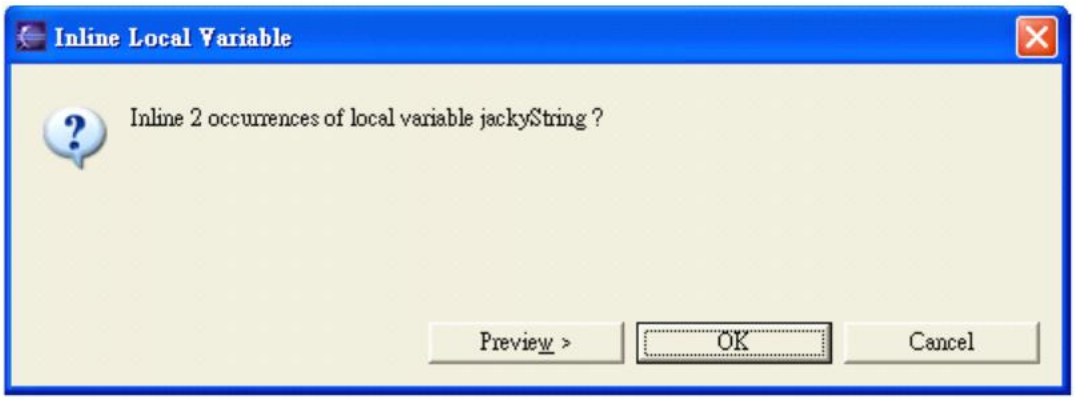


图 6.33

预览窗口会显示重构要更动的部份，下半部的窗格显示两者的比较

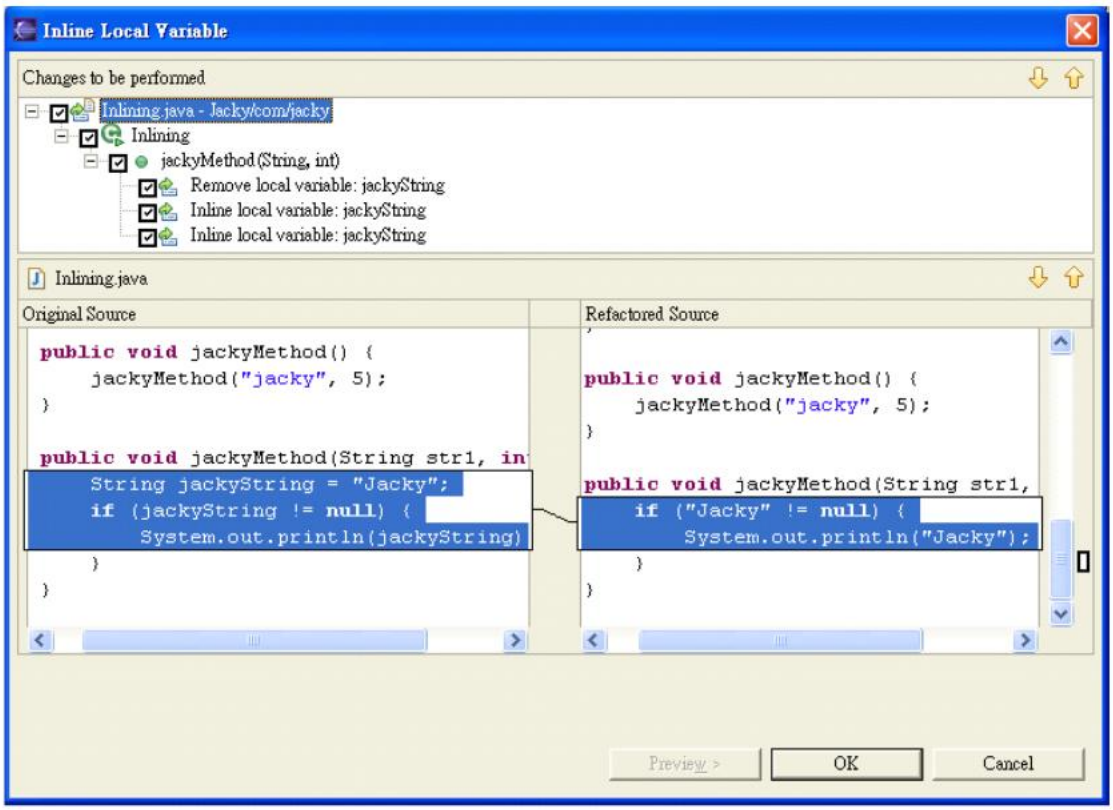


图 6.34

6.3.3 列入方法(Inlining a Method)

如果要列入方法，请执行下列动作：

I. 在 Java 编辑器中选取方法

II. 「Refactor」→「Extract Inlining...」

(或是在编辑器按右键，选取「Refactor」→「Extract Inlining...」)

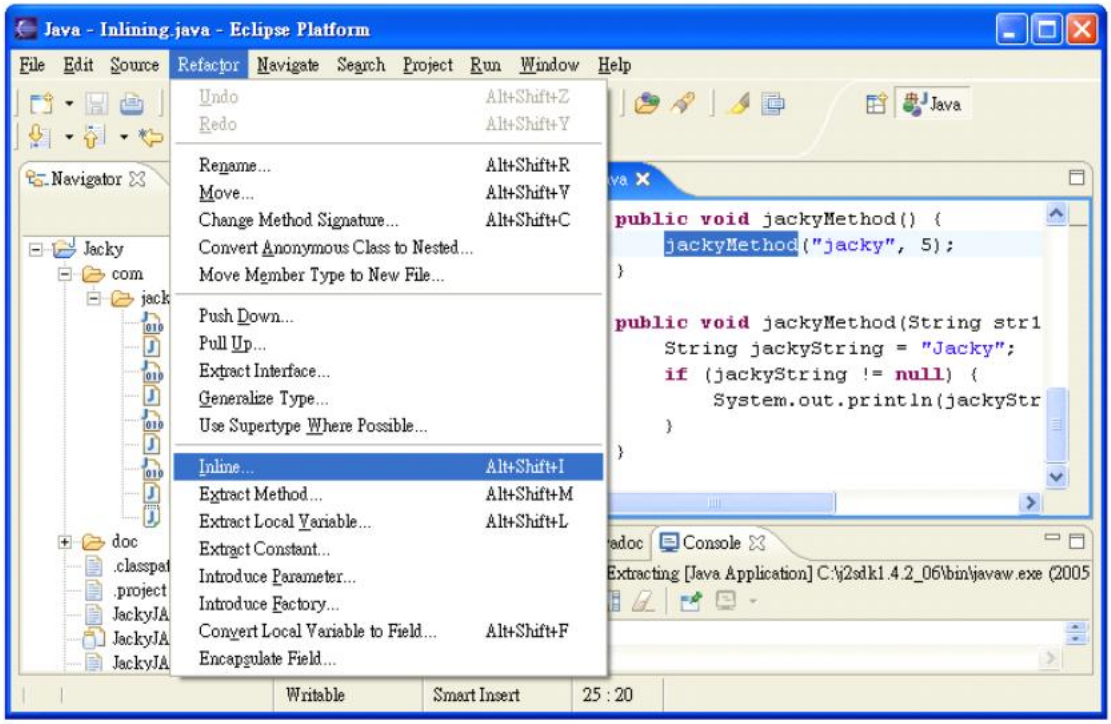


图 6.35

出现 Inline Constant 窗口

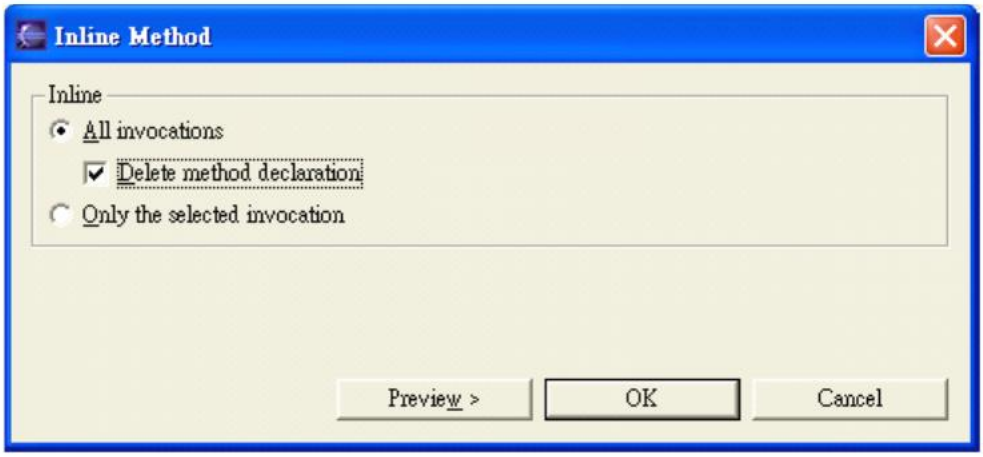


图 6.36

有选 Delete method declaration，方法会被删除

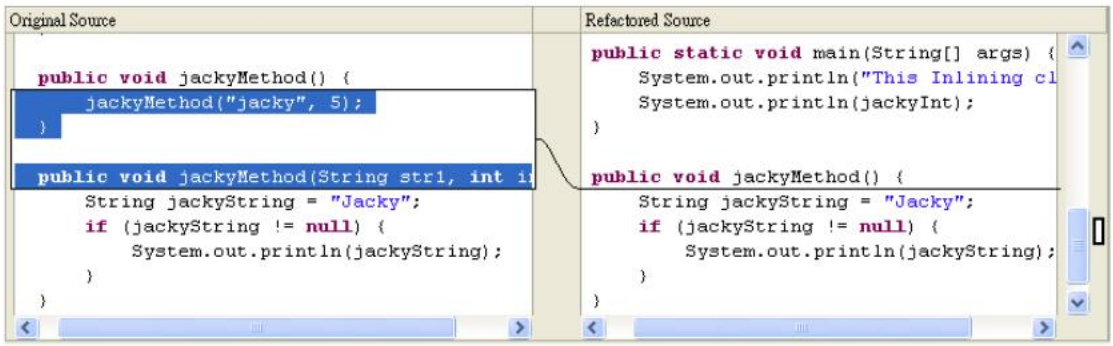


图 6.37

没有选 Delete method declaration，方法不会被删除

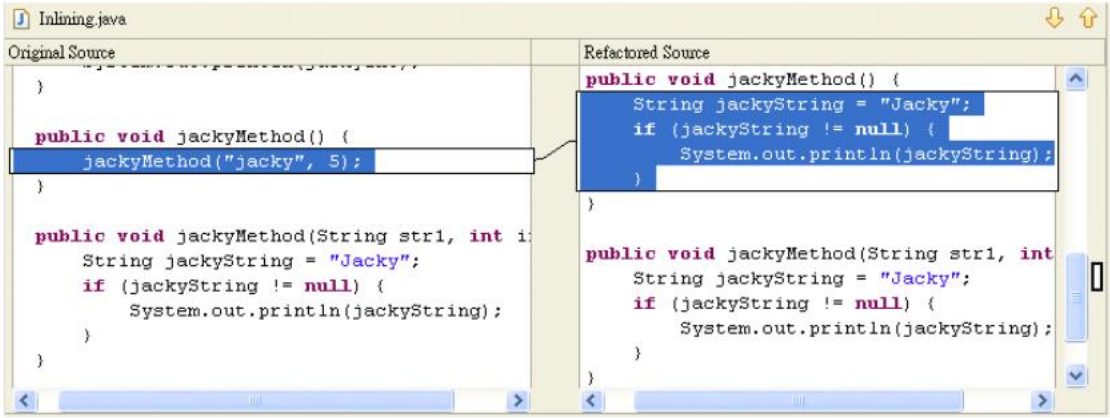


图 6.38

这里可以选择要重构的动作是针对该方法全部的呼叫或是只针对选择的部份

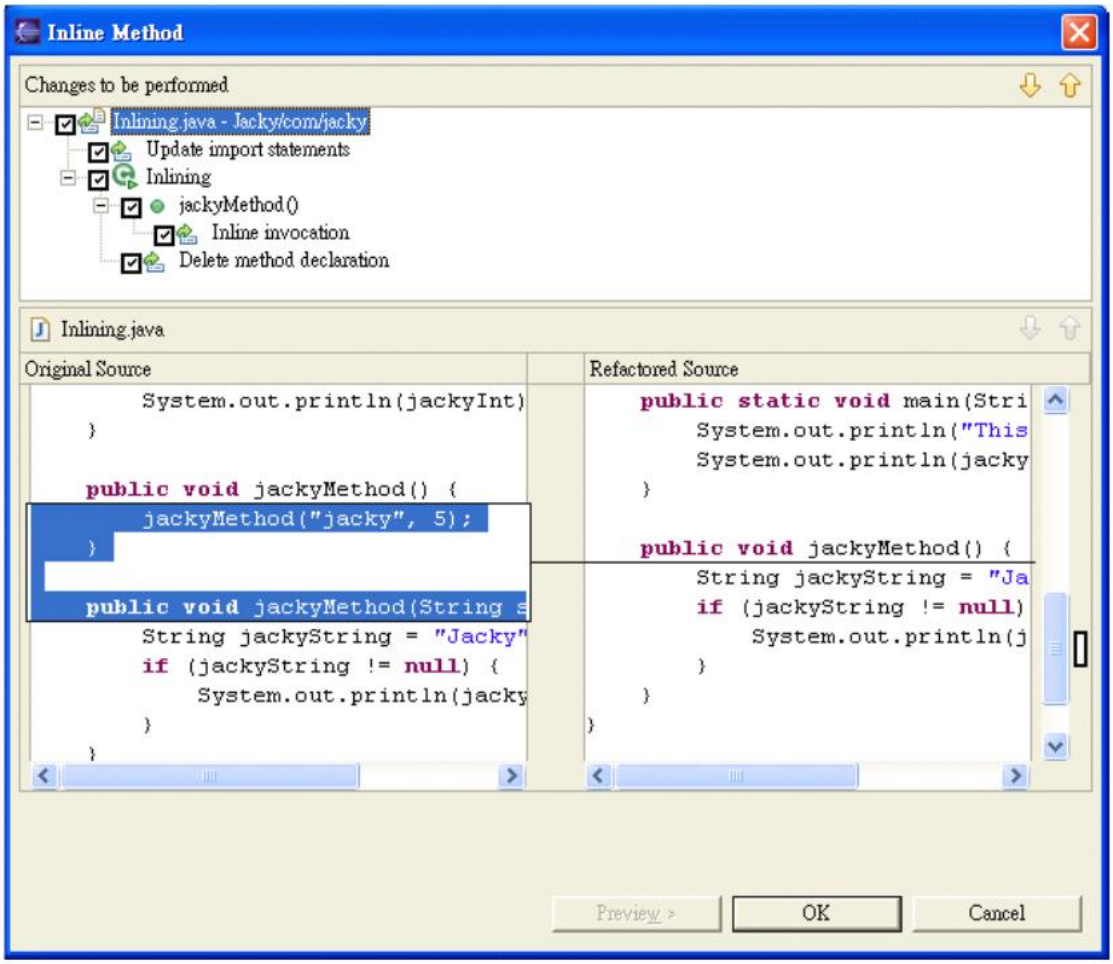


图 6.39

预览窗口会显示重构要更动的部份，下半部的窗格显示两者的比较

6.4 变更方法签章(Signature)

如果要变更方法签章，请执行下列动作：

I. 在 Java 编辑器中选取方法

II. 「Refactor」→「Change Method Signature...」

(或是在编辑器按右键，选取「Refactor」→「Change Method Signature...」)

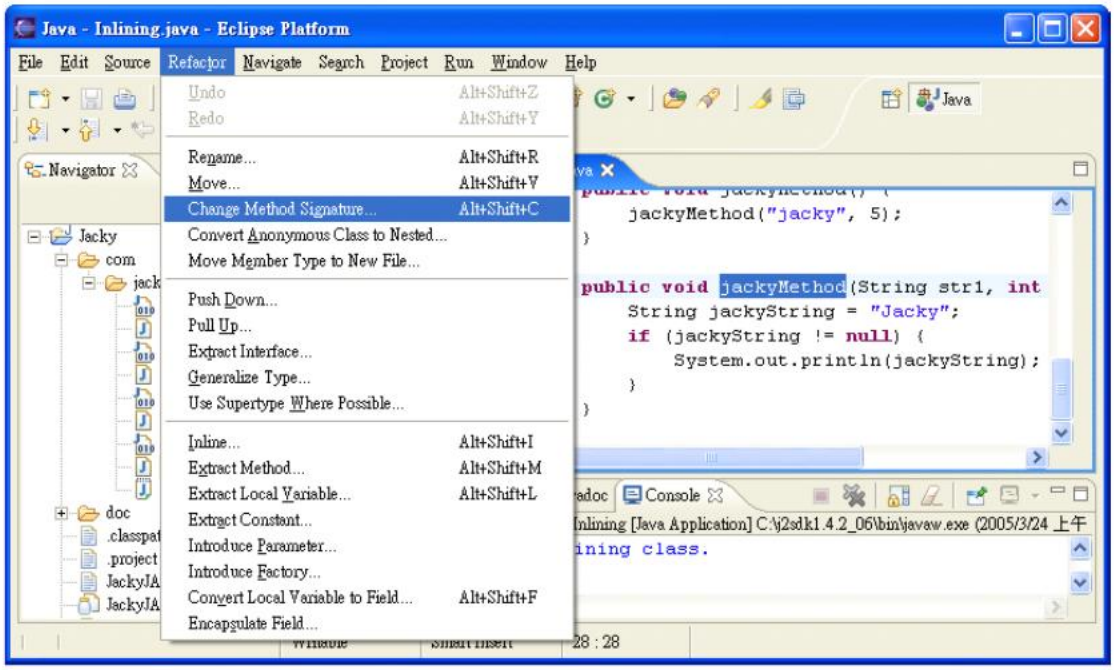


图 6.40

出现 Inline Constant 窗口

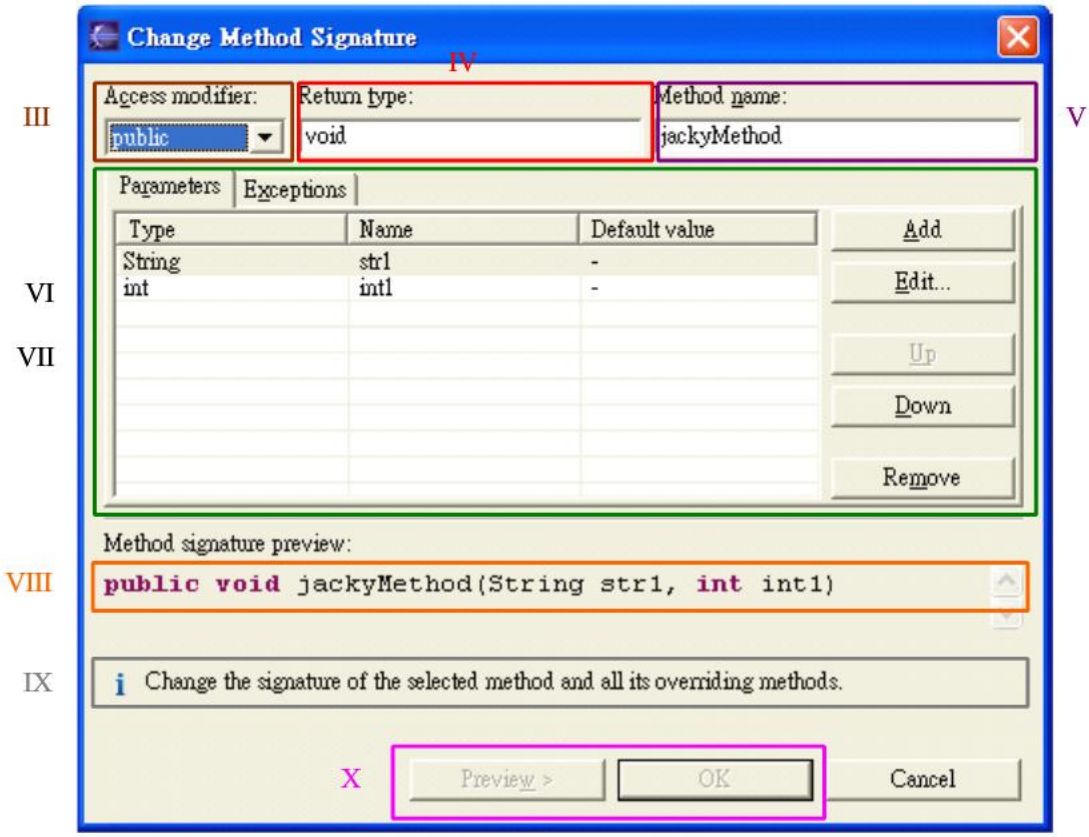


图 6.41

III. 设定 Access modifier

IV. 设定 Return type

V. 设定新的 Method Name

VI. Parameters 页签

使用新增按钮来新增参数；然后，可以在表格中编辑它的类型、名称和默认值

图 6.42

选取一个或多个参数，并使用上和下按钮，以重新排序参数（可查看参数清单下的签章预览）

VII. Exceptions 页签

可以新增或是删除 Exception

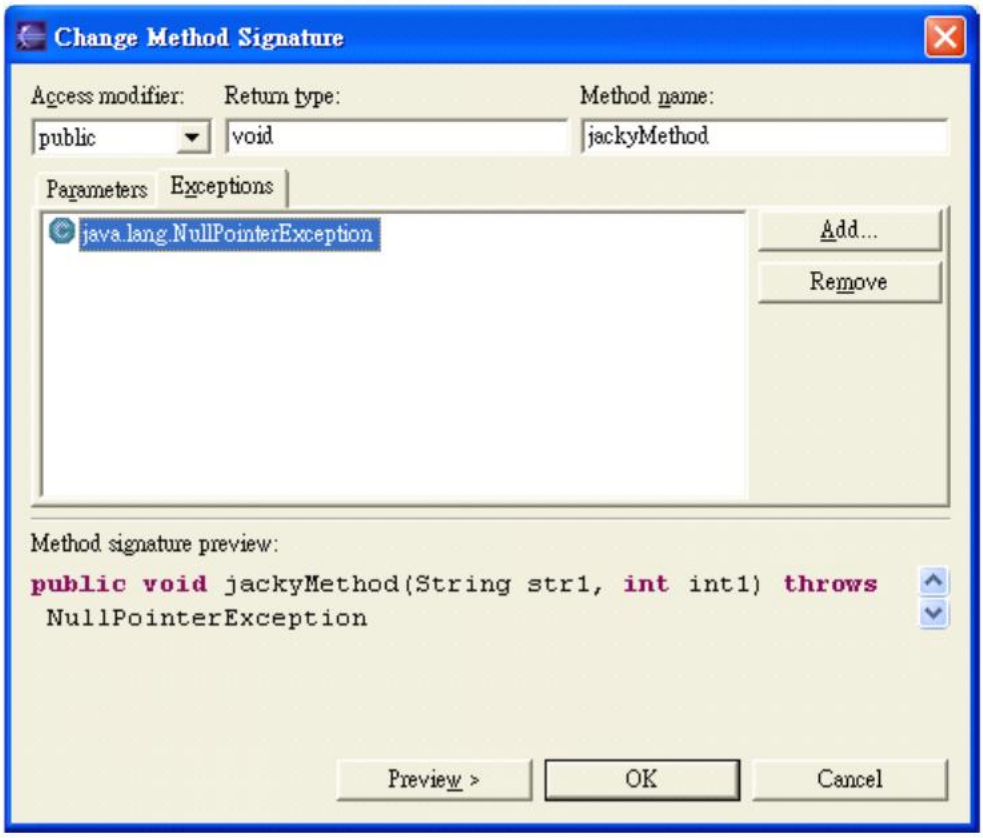


图 6.43

VIII. 立即显示设定后的状态。

IX. 讯息。

X. 按预览，以查看预览，或按确定，执行重构作业，不查看预览

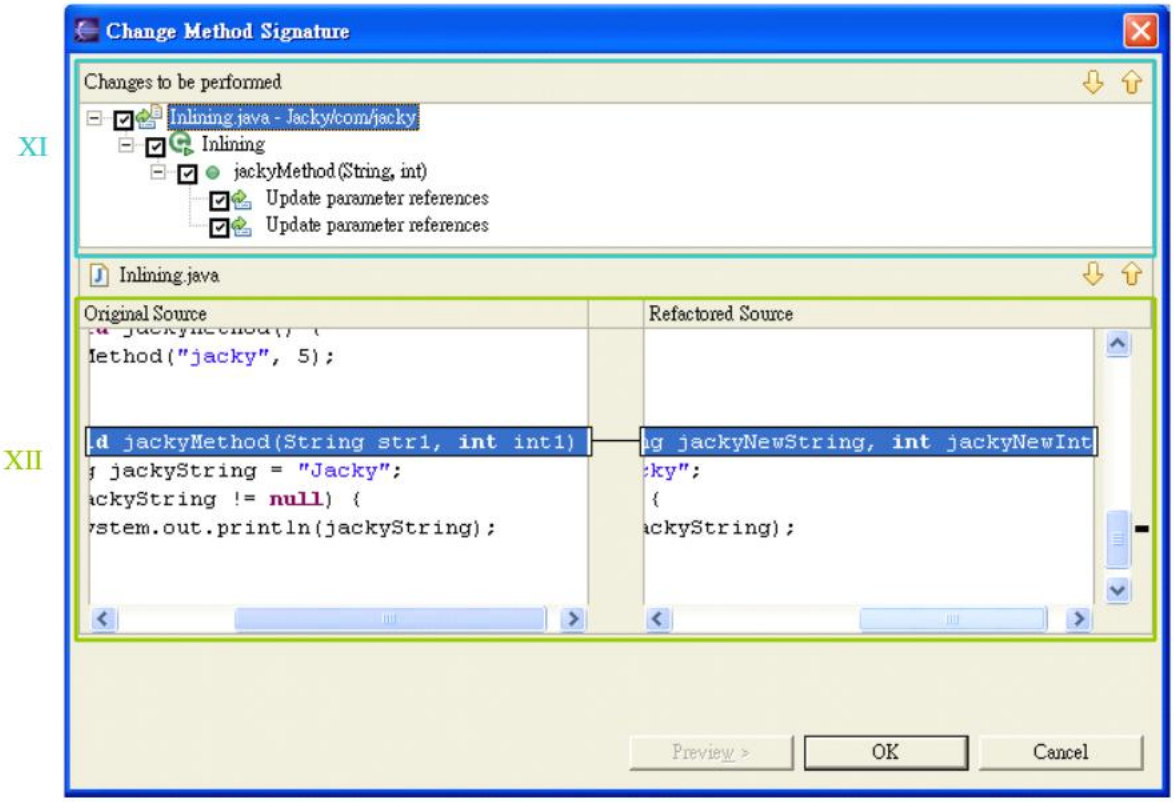


图 6.44

XI. 预览窗口会显示重构要更动的部份

XII. 下半部的窗格显示两者的比较

附注：这项重构作业会变更所选方法和所有置换它之方法的签章。此外，将更新所有参照以使用签章。

6.5 移动 Java 元素(Moving Java Elements)

程序代码如下：

```
public class Moving {
    static int jackyInt;
    String jackyString = "Jacky";

    public static void main(String[] args) {
        System.out.println("This's Moving class.");
        System.out.println(jackyInt);
    }

    public void jackyMethod() {
        System.out.println(jackyString);
    }
}
```

6.5.1 字段(Field)

如果要移动字段，请执行下列动作：

I. 在 Java 编辑器中选取字段

II. 「Refactor」→「Change Move...」

(或是在编辑器按右键，选取「Refactor」→「Move...」)

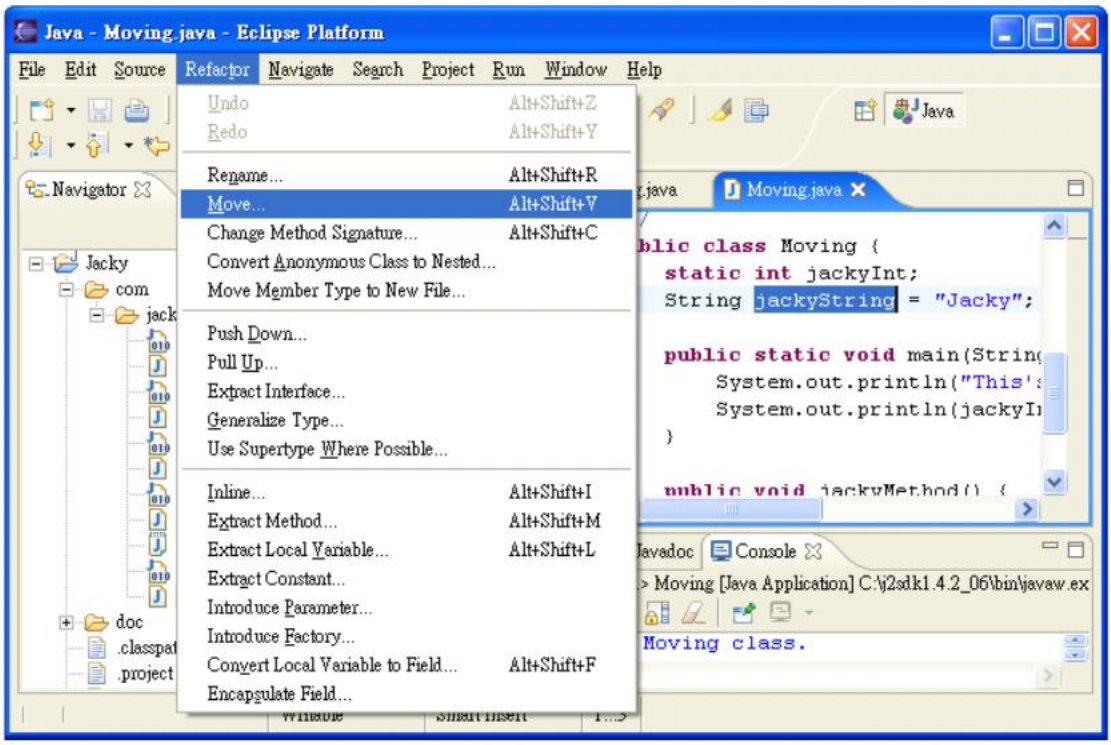


图 6.45

出现 Textual Move 窗口

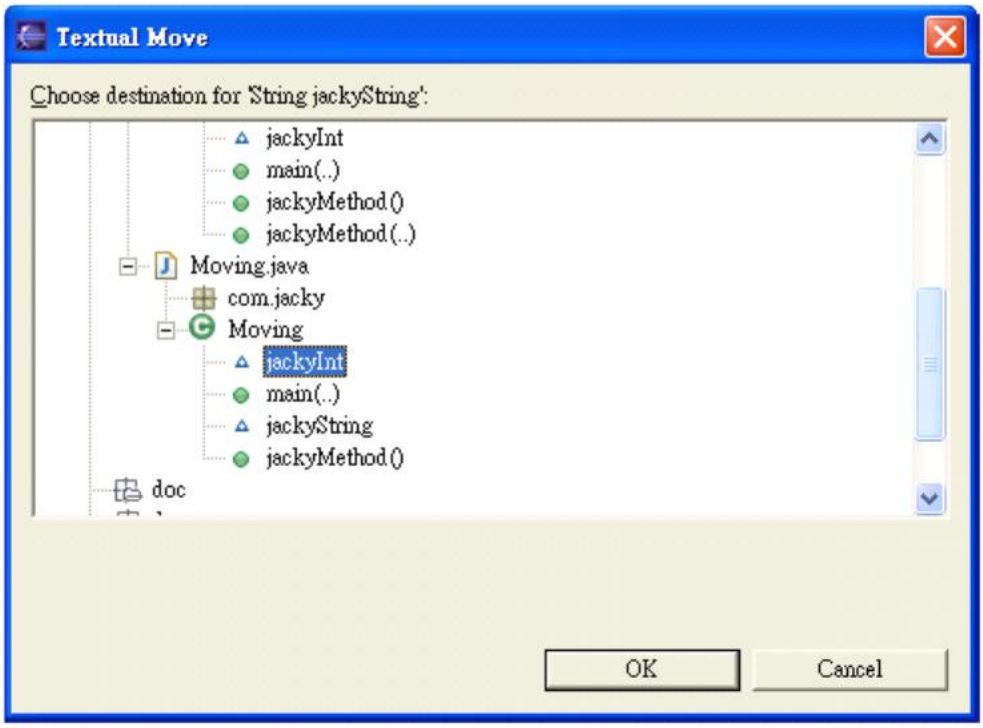


图 6.46

III. 选择要移动的目的地，按确定即可

6.5.2 Static Members

如果要移动 Static Members，请执行下列动作：

I. 在 Java 编辑器中选取 Static Members

II. 「Refactor」→「Change Move...」

(或是在编辑器按右键，选取「Refactor」→「Move...」)

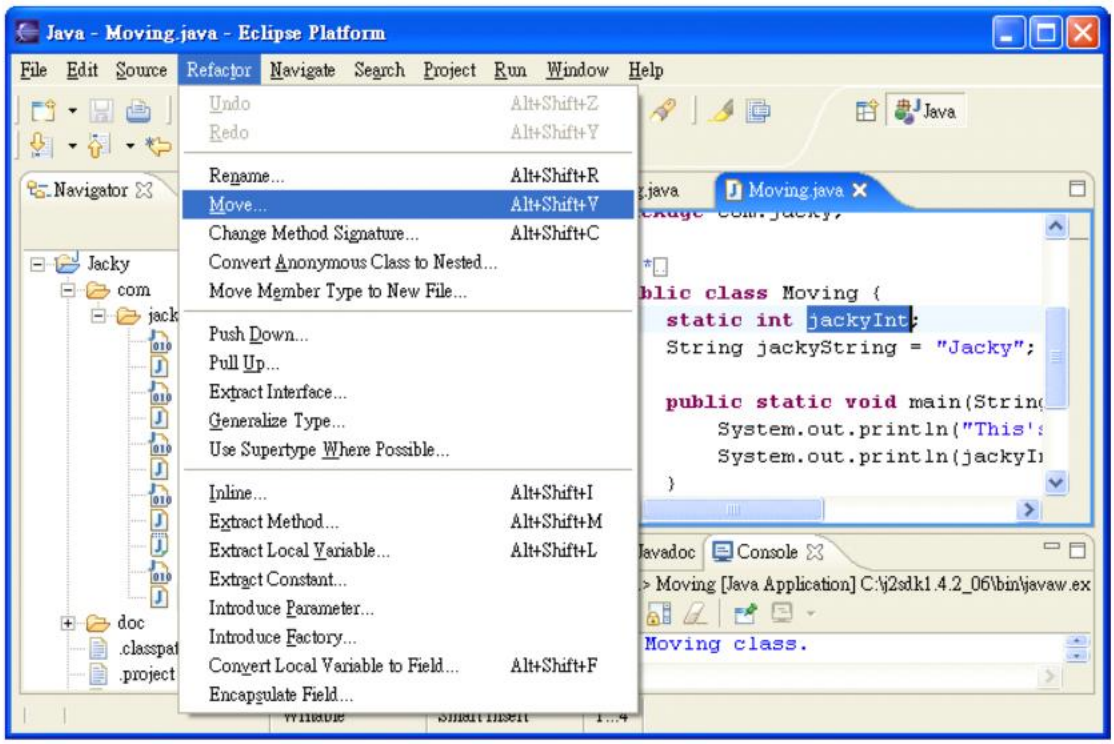


图 6.47

出现 Move Static Members 窗口

图 6.48

III. 按浏览按钮，开启 Choose Type 窗口，选择要移动的目的地

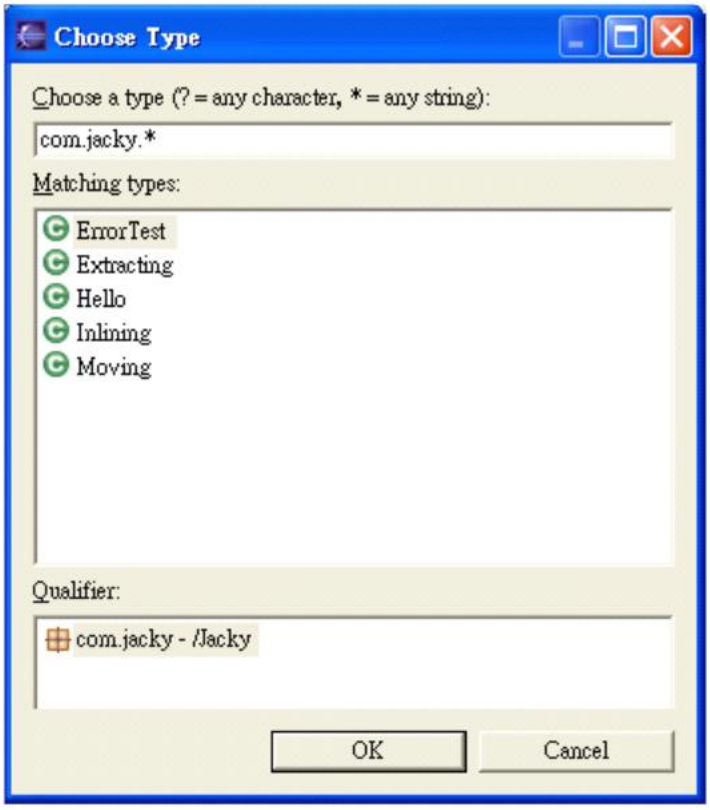


图 6.49

IV. 按预览，以查看预览，或按确定，执行重构作业，不查看预览

V. 预览窗口会显示重构要更动的部份，下半部的窗格显示两者的比较

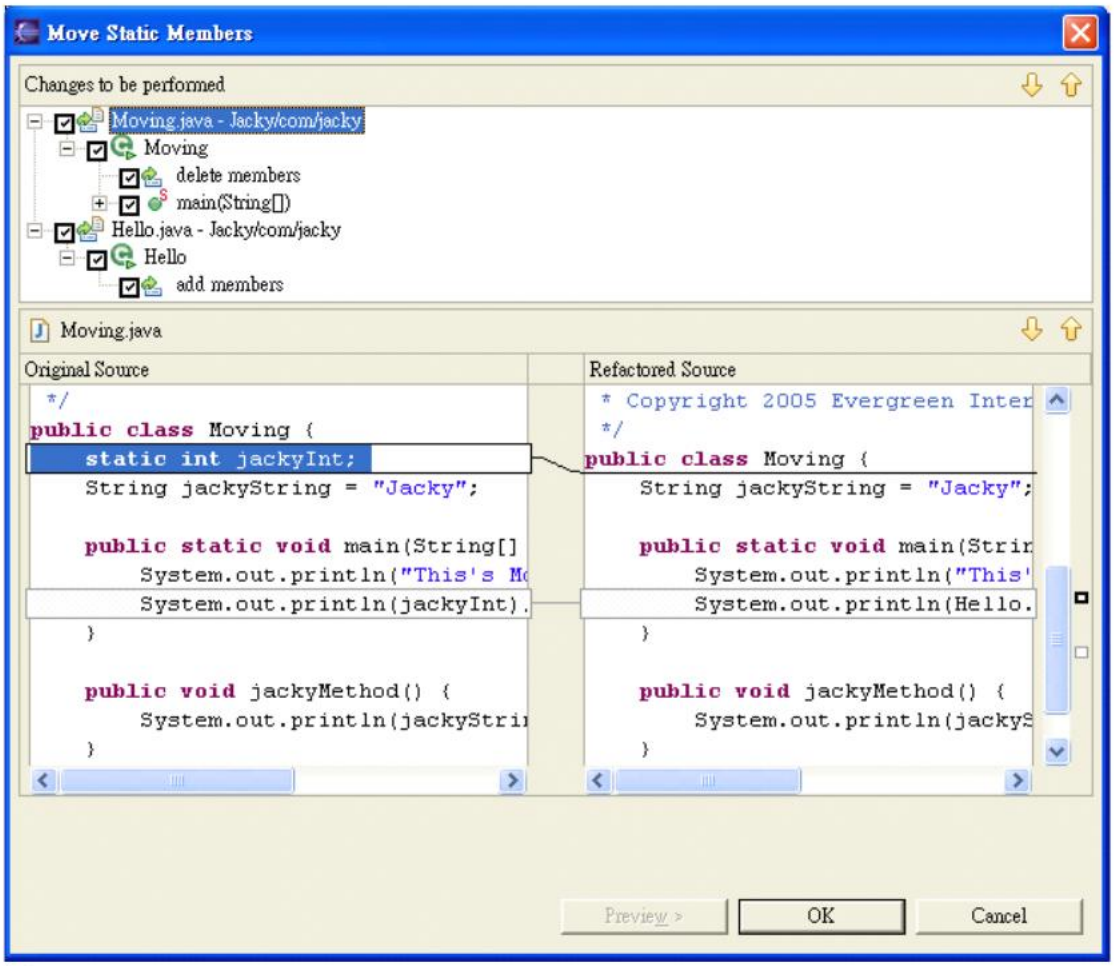


图 6.50

6.6 自行封装字段(Self Encapsulating a Field)

如果要自行封装字段，请执行下列动作：

I. 在 Java 编辑器中选取字段

II. 「Refactor」→「Encapsulate Field...」

(或是在编辑器按右键，选取「Refactor」→「Encapsulate Field...」)

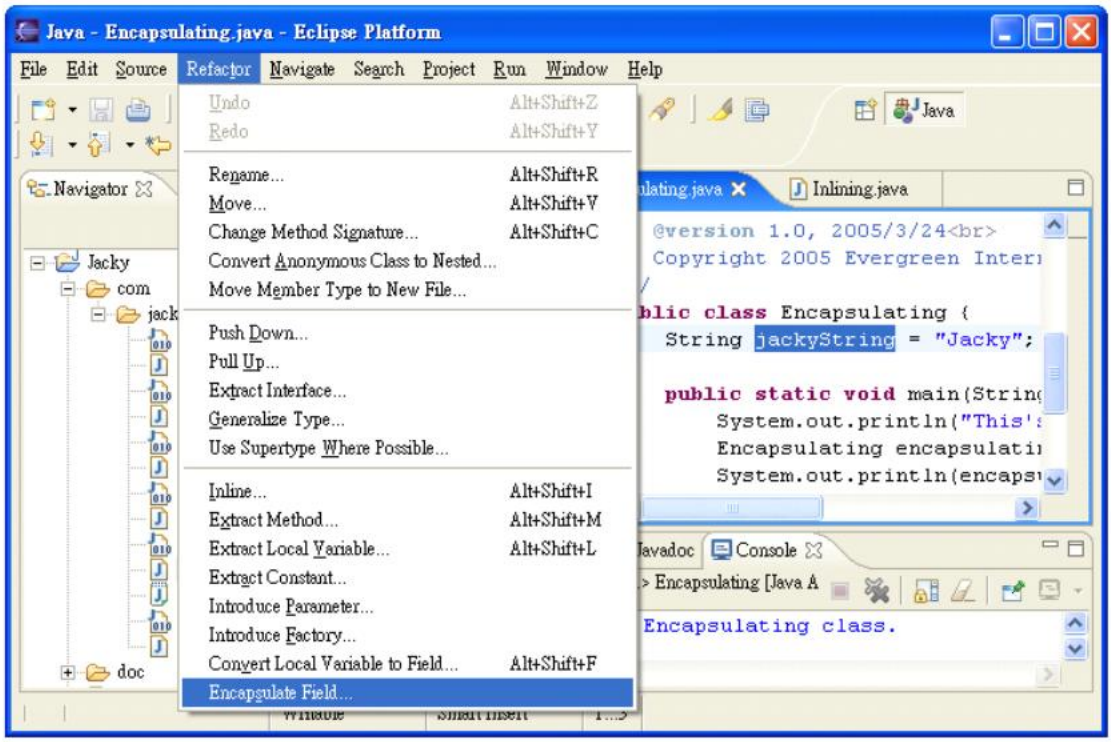


图 6.51

出现 Encapsulate Field 窗口

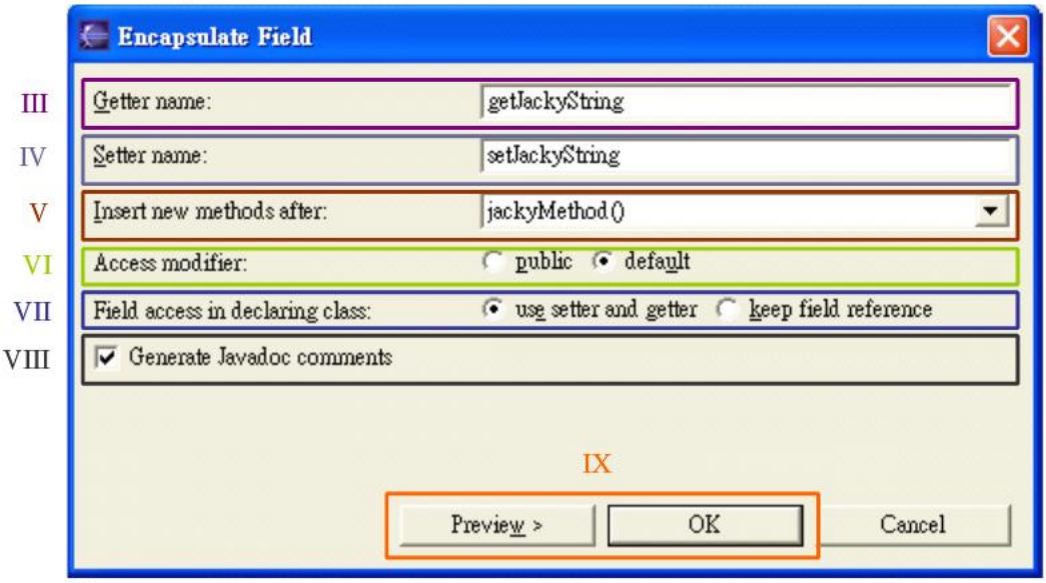


图 6.52

III. 在 Getter 名称字段中输入 Getter 的名称。

IV. 在 Setter 名称字段中输入 Setter 的名称。

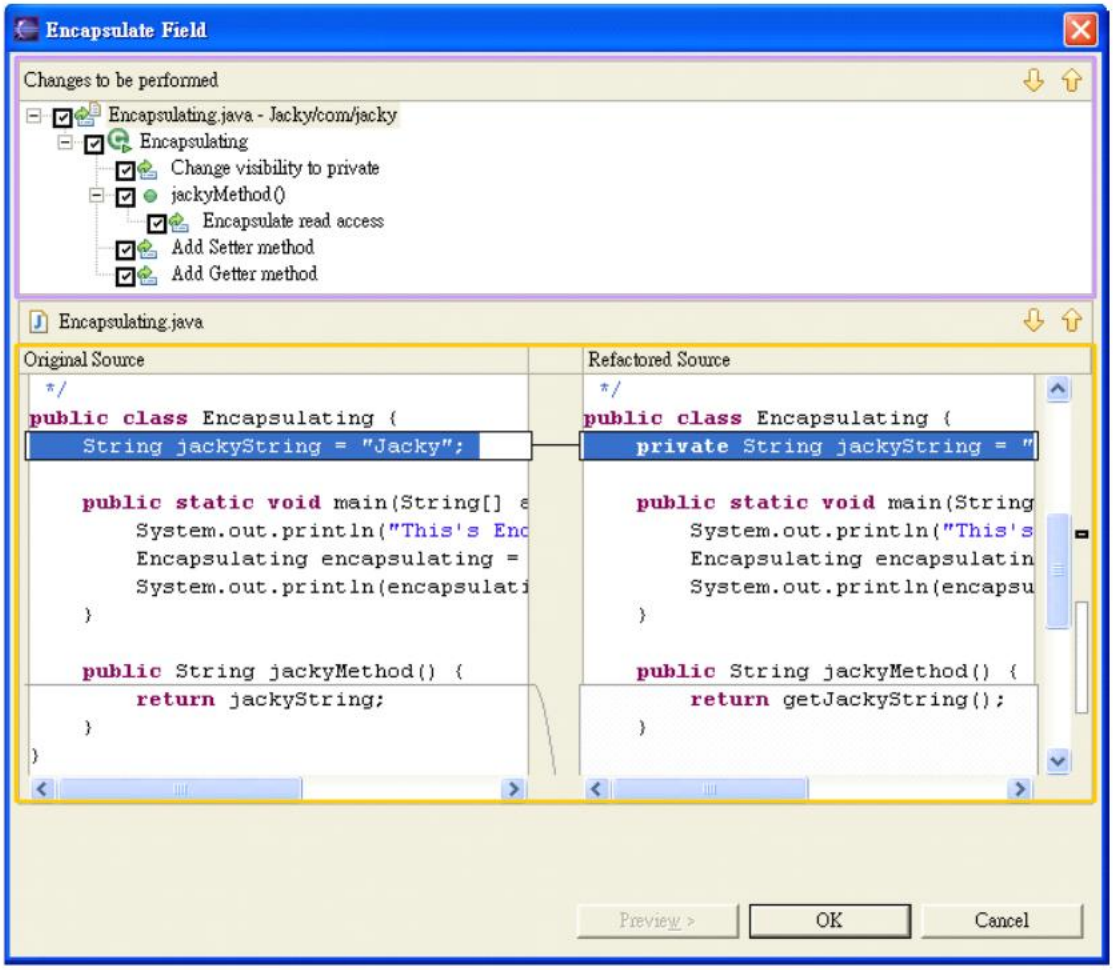
V. 使用在下列后面插入新方法组合框，指出 Getter 与（或）Setter 方法的位置。

VI. 从存取修饰元群组中选取一个圆钮，以指定新方法的可见性。

VII. 在宣告字段所在的类别中，读取权和写入权可为直接的，或者可以使用 Getter 和 Setter。如果想要重构作业将所有这些存取权转换成使用 Getter 和 Setter，请选取使用 Getter 和 Setter 圆钮。如果不想让重构作业修改宣告字段所在之类别中的现行字段存取权，请选取保留字段参照圆钮。

VIII. 若要产生 Javadoc 批注，则选取勾选框

IX. 如果要先预览再进行重构作业，请按预览，或如果要直接进行重构作业而不预览，请按确定。



X

XI

图 6.53

X. 预览窗口会显示重构要更动的部份

XI. 下半部的窗格显示两者的比较

7.要诀和技巧(Tips and Tricks)

7.1 编辑程序文件(Editing Source)

内容	说明
参数提示(Parameter Hints)	当光标位在方法自变量时，可以看到参数提示的清单。在「Java 编辑器」中，按下 Ctrl+Shift+空格键或呼叫「Edit」→「Parameter Hints」。
抑制程序代码辅助中的类型(Suppress types in code assist)	如果不要让某些类型出现在内容辅助中，请使用在「Java」→「Type Filters」喜好设定页面配置的类型过滤器功能。只要符合其中一项过滤器型样的类型，就不会出现在「开启类型」对话框中，且不供程序代码辅助、快速修正和组织汇入使用。这些过滤器型样不会影响「Package Explorer」和「Type Hierarchy」视图。 Type Filters All types in packages that match the selected filter strings will not be shown in the 'Open Type' dialog. They will also be ignored in code assist or quick fix proposals. For example 'java.awt.*' will hide all types from the awt packages. Filter list: com.ibm.* com.sun.* javax.* org.omg.* New... Add Packages... Edit... Remove
使用内容辅助来建立 Getter 和 Setter(Use content assist to create Getter and Setters)	建立 Getter 和 Setter 的另一个方法，就是使用内容辅助。可以把光标停在成员之间的类型主体，然后按 Alt+/，取得建立 Getter 或 Setter 方法 Stub 的提议。 public class Encapsulating { String jackyString = "Jacky"; set setJackyString(String) void - Setter for 'jackyString' set() void - Method stub Set - java.util SetDrawLineANY - sun.java2d.loops
连同字段一起删除 Getters 和 Setter(Delete Getters and S	当从视图删除字段时，Eclipse 可能会提议连同其 Getter 和 Setter 方法一起删除。如果在字段使用名称前缀或字尾，请务在「Code Style」喜好设定页面指定它（「Window」→「Preferences」→「Java」→「Code Style」）。

内容	说明
etters together with a field)	

建立委派方法(Create Delegate Methods)	如果要对字段建立委派方法， 请选取字段的宣告，并呼叫「Source」→「Generate Delegate Methods」。 这将新增所选方法至含有至委派的方法之转递呼叫的类型。
	`public class Encapsulating {`
	`  String jackyString = "Jacky";`
	`  public String toLowerCase() {`
	`  return jackyString.toLowerCase();`
	`  }`

使用「模板」来建立方法(Use Templates to create a method)	可以定义新的模板（「Window」→「Preferences」→「Java」→「Editor」→「Templates」）来含有方法 Stub。范本会与内容辅助(Alt+/)提议一起显示。
	也有现有的范本，如'private_method'、'public_method'、'protected_method'，以及其它等等。
	使用 Tab 键，在要输入的值（传回类型、名称和自变量）之间导览。
	`public class Encapsulating {`
	`  String jackyString = "Jacky";`
	`  private`
	`  public`
	`  Sys`
	`  Enc`
	`  Sys`
	`  }`
	private
	private_method - private method
	private_static_method - private static method
	PRIVATE_MEMBER - org.omg.CORBA
	PrivateCredentialPermission - javax.security.auth

使用「内容辅助」来置换方法(Use Quick Fix to create a new method)	在类型主体中应该加入方法的位置，呼叫内容辅助(Alt+/)。内容辅助将提供所有可以被置换的方法。将建立所选方法的方法主体。
	`public class Encapsulating {`
	`  String jackyString = "Jacky";`
	`  to`
	`  pu`
	`  }`
	toString() String - Override method in 'Object'
	to() void - Method stub
	toarray - convert collection to array
	TokenTracker - sun.security.jgss
	ToolBarUI - javax.swing.plaf

使用「快速修正」来新增未实作的方法(Use Quick Fix to change a method signature)	如果要实作新的接口，请先新增'implements'宣告至类型。即使没有储存或建置，Java 编辑器也会强调以讯号显示该类型，表示该方法遗漏，并且显示快速修正灯泡。可以按一下灯炮，或按下 Ctrl+1（「Edit」→「Quick Fix」），选择要新增未实作的方法，或者让类别成为 abstract。

内容	说明
	18 public class Encapsulating implements Iterator{ 19 String jackyS 20 21 public static 22 System.ou Add unimplemented methods Make type 'Encapsulating' abstract Rename in file
利用内容辅助来建立建构子 Stub(Use Content Assist to create a constructor stub)	在要新增建构子的位置，输入该建构子名称的第一个字母之后，再使用程序代码辅助。 public class Encapsulating { String jackyString = "Jacky"; Enc pub Enc() void - Method stub Encapsulating() - Default constructor Encapsulating - com.jacky EncapsInputStream - com.sun.corba.se.internal.corba
使用「内容辅助」来置换方法(Use Content Assist to override a method)	在类型主体中应该加入方法的位置，呼叫内容辅助 (Alt+/)。内容辅助将提供所有可以被置换的方法。将建立所选方法的方法主体。 public class Encapsulating { String jackyString = "Jacky"; to pu toString() String - Override method in 'Object' to() void - Method stub toarray - convert collection to array TokenTracker - sun.security.jgss

从参数建立新字段 (Create new fields from parameters)	需要建立新字段来存放传入建构子中的自变量吗？请在参数使用快速辅助(Ctrl+1)，来建立指派作业和字段宣告，并且让 Eclipse 根据「程序代码样式」喜好设定来提议名称。
	public String jackyMethod(int jackyInt) {
	return jackyString;
	}
	Assign parameter to new field
	Rename in file

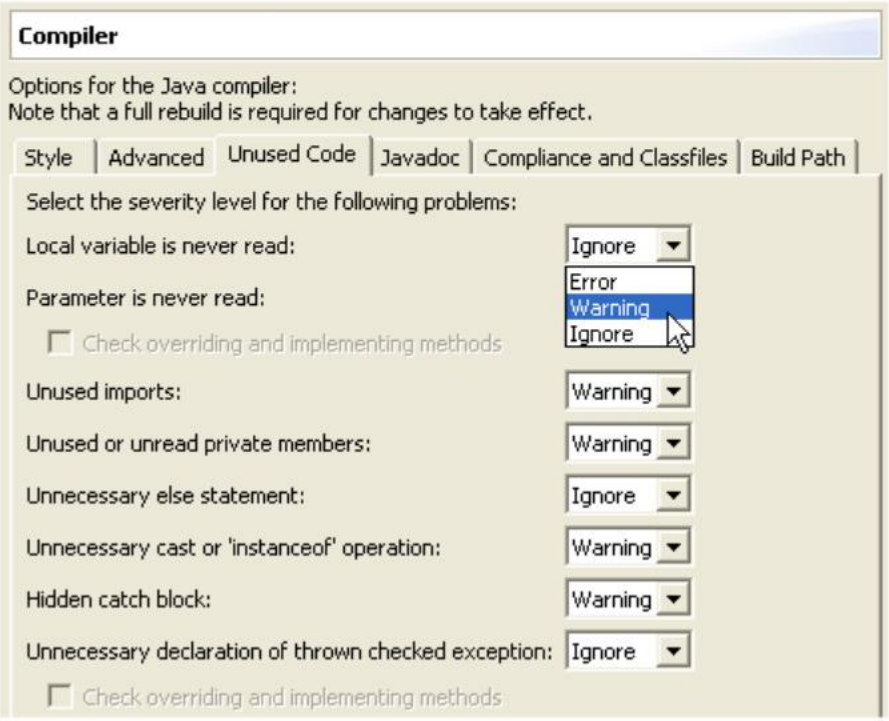
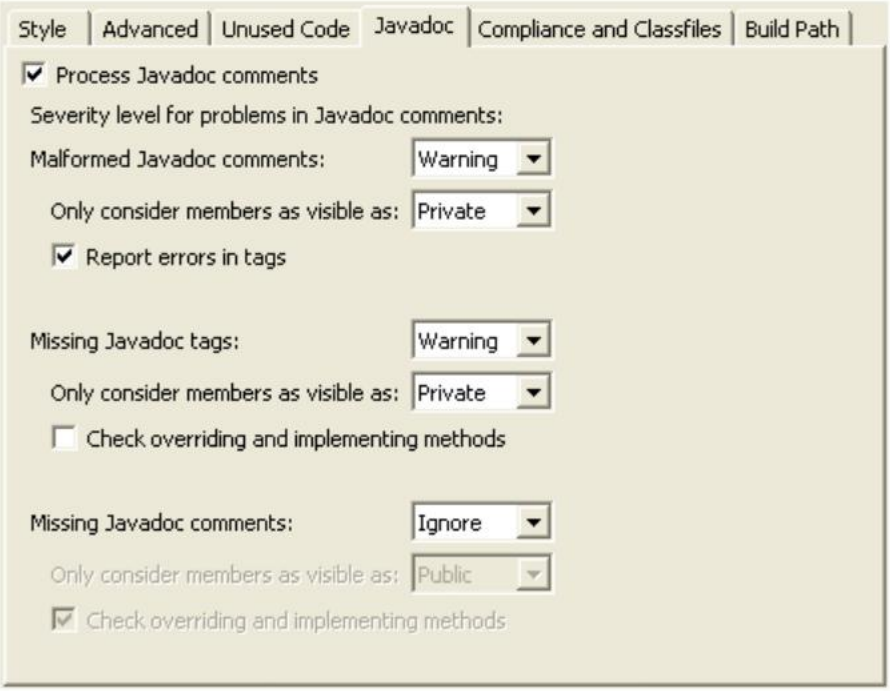
在档案中重新命名 (Rename in File)	如果要快速进行重新命名作业，可以使用「在档案中重新命名」快速辅助，而不必在其它档案完整分析其相依关系。在 Java 编辑器中，将光标定位在变量、方法或类型的识别码，然后按下 Ctrl+1（「Edit」→「Quick Fix」）
	编辑器会切换至链接的编辑模式（如同模板），而且变更识别码将同时变更该变量、方法或类型的所有其它参照。
	public void jackyMethod(int jackyInt) {
	for(int i = jackyInt; i < jackyInt +
	System.out.println(i);
	}
	}

内容	说明
减少强制转型表示式的作业时间(Less work with cast expressions)	别花太多时间在输入强制转型。先略过它们，等完成陈述式之后，再利用快速辅助加入它们。 比方说，以指派作业为例： String newString = list.get(0); return jackyStri Add cast to 'java.lang.String' Change type of 'newString' to 'Object' Rename in file ... public String jackyMethod() { String newString = **(String)** list.get(0); return jackyString;

包覆字行(Surround Lines)	如果要以 if/while/for 陈述式或区块包覆陈述式，请选取要包覆的字行，然后按下 Ctrl+1（「Edit」→「Quick Fix」）。这会列出所有含有 ${line_selection} 变量的模板。
	public void jackyMethod() {
	System.out.println("Jacky");
	Surround with 'do' (do while statement)
	Surround with 'for' (iterate over array)
	Surround with 'if' (if statement)
	Surround with 'runnable' (runnable)
	if (condition) {
	System.out.println("Jacky");
	}

移除包覆的陈述式(Remove Surrounding Statement)	如果要移除包覆的陈述式或区块，请将光标定位在右方括号，然后按下 Ctrl+1（「Edit」→「Quick Fix」）。
	public void jackyMethod() {
	if (true) {
	System.out.println("Jacky");
	}
	}
	Remove surrounding 'if' statement

寻找对称的括号(Find the Matching Bracket)	如果要寻找对称的括号，请选取左或右括号，然后按下 Ctrl+Shift+P（「Navigate」→「Go To」→「Matching Bracket」）。也可以在左方括号之前，或是右方括号之后按两下鼠标，选取这两个方括号之间的文字。
排序成员(Sort Members)	可以根据「Window」→「Preferences」→「Java」→「Appearance」→「Members Sort Order」所定义的种类顺序，针对 Java 编译单元来排序成员。 将在「Source」→「Sort Members」下找到动作
寻找未用的程序代码(Find Unused Code)	Java 编译器会侦测无法呼叫到的程序代码、未使用的变量、参数、汇入项目和未使用的私密类型、方法和字段。这个设定位在「Window」→「Preferences」→「Java 」→「Compiler」。

内容	说明
	 当输入时也会侦测这些设定，并提供一个快速修正以移除不必要的程序代码。
处理 Javadoc 批注(Javadoc Comment Handling)	Java 编译器可以处理 Javadoc 批注。搜寻报告会参照 doc 批注，而重构会更新这些参照。这项特性是从「Window」→「Preferences」→「Java 」→「Compiler」Javadoc 标签加以控制（或是利用「Project」→「Properties」→「Java Compiler」→「Javadoc」，针对个别项目加以设定）。 

内容	说明
	当它开启时，形态异常的 Javadoc 批注会在 Java 编辑器中标示出来，也可以利用「Edit」→「Quick Fix」(Ctrl+1)加以修正

7.2 搜寻(Searching)

内容	说明

寻找变数及其读写权(Locate variables and their read/write access)	可以选取一个 ID（变量、方法、类型参照或宣告），呼叫「Search」→「Occurrences in File」，来寻找变量，看看它们的读写状态。此举会标示同一档案中该识别码的所有参照。结果也会显示在搜寻视图中，以及有图标会显示变量的读取权或写入权。
	`public class Encapsulating {`
	`  String jackyString = "Jacky";`
	`  public static void main(String`
	`  System.out.println("This's`
	`  Encapsulating encapsulatin`
	`  System.out.println(encapsu`
	`  }`
	`  public String jackyMethod() {`
	`  return jackyString;`
	`  }`
	Search ✕ Problems Javadoc Declaration
	jackyString - 2 Occurrences in Encapsulating.java
	String jackyString = "Jacky";
	return jackyString;
	或者，也可以使用新的标示搜寻结果特性，动态强调搜寻结果。可以使用一般搜寻特性来搜寻数个档案（「Search」→「References」）。

搜寻具有特定传回类型的方法(Search for methods with a specific return type)	如果要搜寻具有特定传回类型的方法，请使用"*<return type>"，如下所示： 开启搜寻对话框，再按一下 Java Search 标签。 在 Search string 中输入以空格隔开的'*'和传回类型。 选取 Case sensitive 勾选框。 选取 Method 和 Declarations，然后按一下 Search。

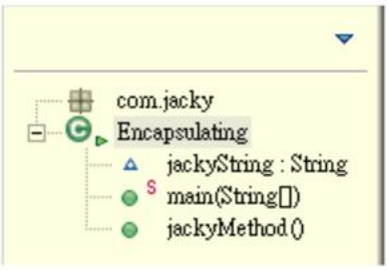
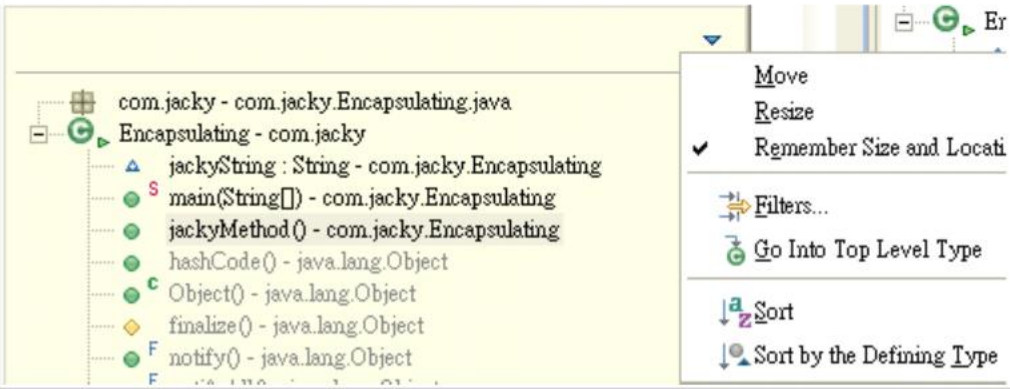
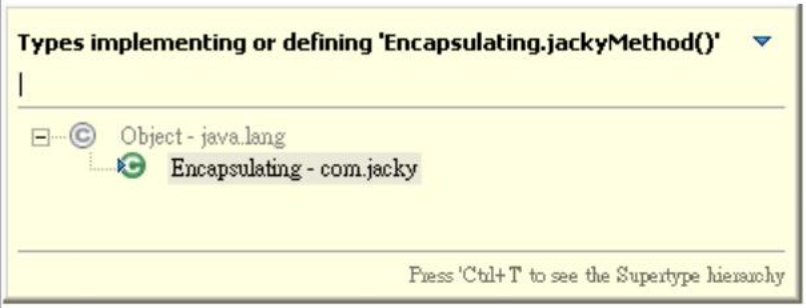
内容	说明
	File Search | Help Search | Java Search | Plug-in Search Search string (* = any string, ? = any character): * List Search For: Type, Method, Package, Constructor, Field Limit To: Declarations, References, Read Access
从 Java 搜寻移除 Javadoc 结果(Remove Javadoc results from Java search)	依预设，Java 搜寻会在 Java 程序代码和 Javadoc 中寻找参照。如果不要在 Javadoc 中寻找参照，可以取消勾选「Window」→「Preferences」→「Java」→「Compiler」→「Javadoc」的 Process Javadoc comments，停用这项行为。
以呼叫阶层来追踪方法呼叫链(Trace method call chains with the Call Hierarchy)	有没有发觉自己曾经不断的搜寻方法参照？现在有一种新的呼叫阶层，可以跟着又长又复杂的呼叫链，又不会遗漏原始的环境定义：只要选取一个方法，然后呼叫「Navigate」→「Open Call Hierarchy」(Ctrl+Alt+H)即 可 。 Call Hierarchy | Search | Problems | Javadoc | Declaration | Console Members calling 'jackyMethod()' - in Workspace jackyMethod() - com.jacky.Encapsulating main(String[]) - com.jacky.Encapsulating Line: 24 Call: jackyMethod()

7.3 程序代码导览和读取(Code navigation and reading)

内容	说明

在 Java 编辑器中依据选项开启(Open on a selection in the Java editor)	在 Java 编辑器中，有两种方法，可让从元素的参照中开启元素。
	选取程序代码中的参照，并按下 F3（「Navigate」→「Open Declaration」）
	按住 Ctrl 键，将鼠标指针移到参照之上
	`public void jackyMethod(int jackyInt) {`
	`for(int i = jackyInt; i < jackyInt +`
	`System.ou` `int jackyInt` `i);`
	`}`
	`}`

原位概要(In-place outlines)	在 Java 编辑器中按下 Ctrl+F3，以在现行光标位置蹦现元素的原位概要。或者按 Ctrl+O（「Navigate」→「Quick Outline」），以蹦现现行程序文件的原位概要。

内容	说明
	
原位概要会显示继承的成员(In-place outlines show inherited members)	再按一次 Ctrl+O 或 Ctrl+F3，把继承的成员加到一个开启的原位概要中。继承的成员附有一个灰色标签。可以利用右上角的菜单来过滤和排列概要。 
原位阶层(In-place hierarchy)	利用「Quick Hierarchy」，找出哪些是虚呼叫可能的接收端。然后把光标放在方法呼叫里面，按 Ctrl+T（「Navigate」→「Quick Outline」）。这个视图会以完整的图标，显示所有实作方法的类型。  再按一次 Ctrl+T，切换至「Supertype hierarchy」阶层。

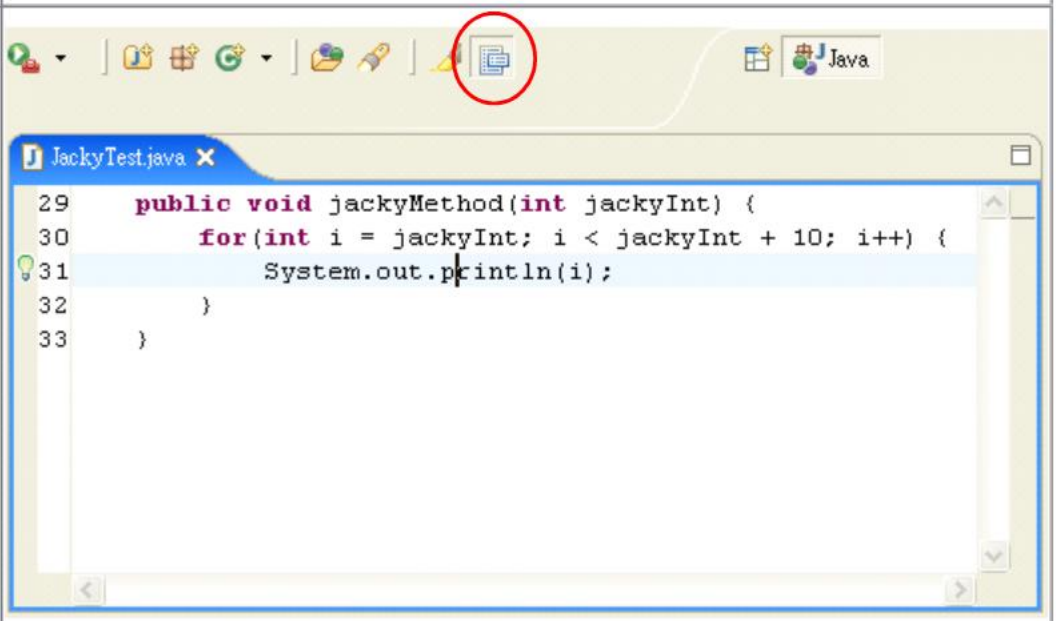
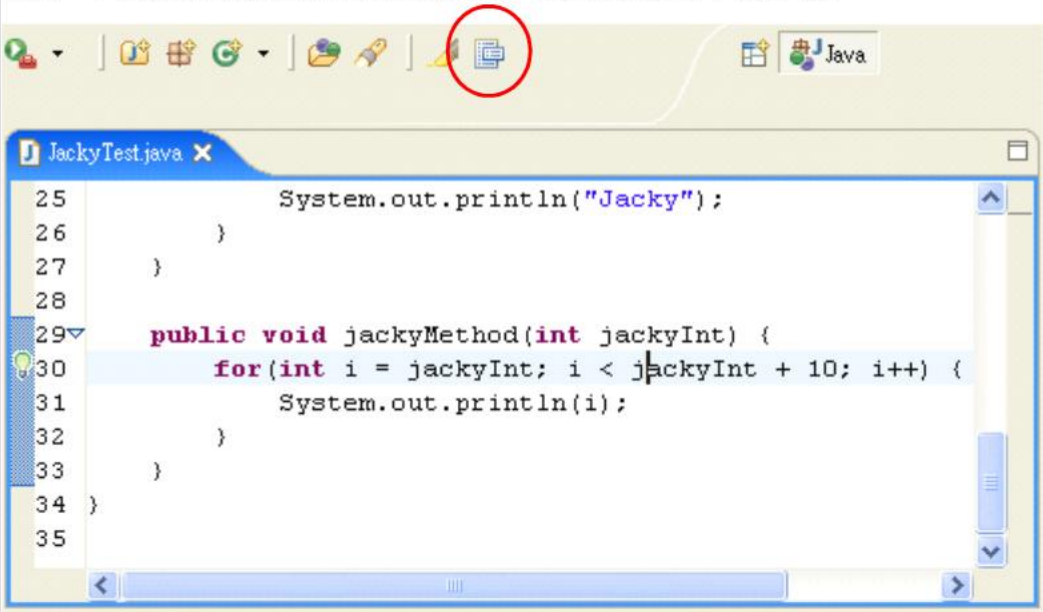
标示搜寻结果(Mark Occurrences)	当在编辑器工作时，请开启工具列上的标示搜寻结果（ ）或者按(Alt+Shift+O)。会在档案中，看到被参照的变量、方法或类型。
	`public void jackyMethod(int jackyInt) {`
	`  for(int i = jackyInt; i < jackyInt +`
	`  System.out.println(i);`
	`  }`
	`}`

移至下一个 / 上一个方法(Go to next / pr	如果要快速导览至下一个或上一个方法或字段，请使用 Ctrl+Shift+上移键（「Navigate」→「Go To」→「Previous Member」），或 Ctrl+Shift+下移键（「Navigate」→「Go To」→「Next Member」）

内容	说明
evious method)	

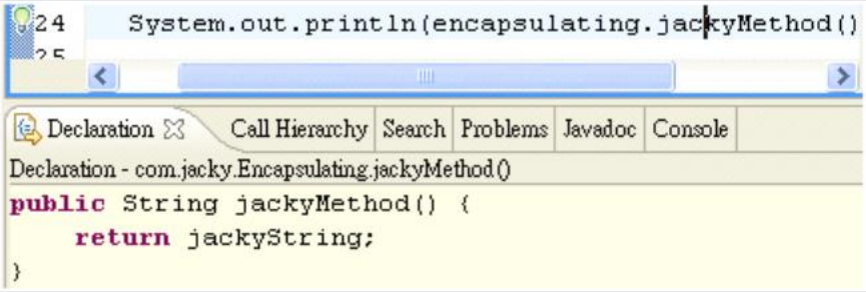
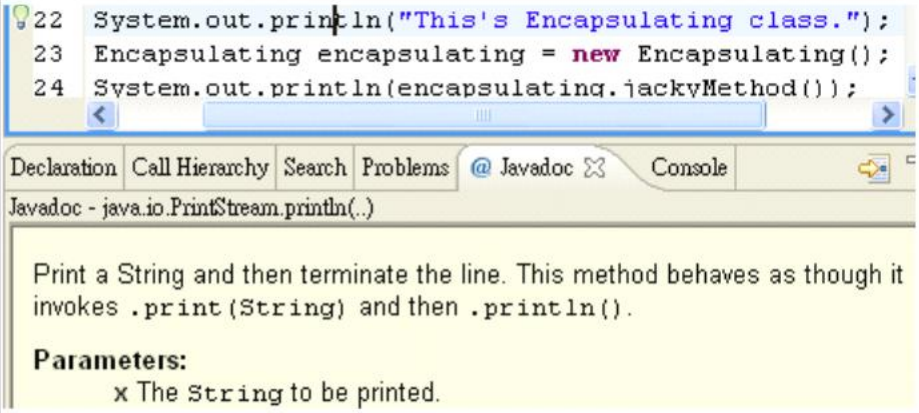
Java 编辑器中的浮动说明(Hovers in the Java editor)	可以使用修正键（Shift、Ctrl、Alt），在 Java 编辑器看到不同的浮动说明。
	当将鼠标在 Java 编辑器中的识别码上移动时，依预设，将显示一个浮动说明，其中含有从这个元素的对应程序文件撷取的 Javadoc。按住 Ctrl 键将显示程序代码。
	`System.out.println(encapsulating.jackyMethod());`
	`}`
	`public String jackyMethod() {`
	`return jackyString;`
	`}`
	`public String jackyMethod() {`
	`return jackyString;`
	`}`
	Press 'F2' for focus.
	可以在「Window」→「Preferences」→「Java」→「Editor」→「Hovers」中，变更这个行为，以及定义其它修正键的浮动说明。

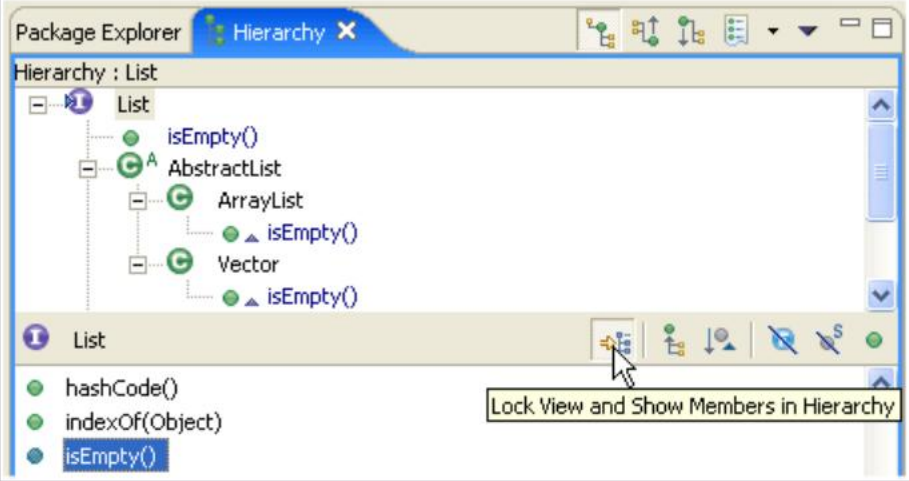
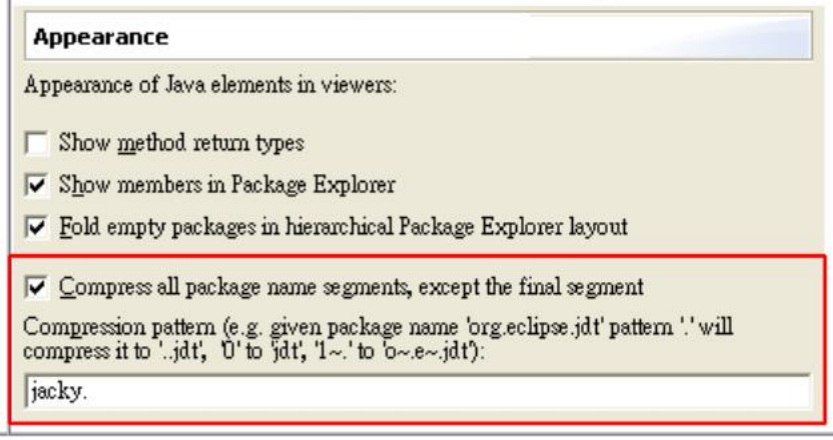
开启和配置外部 Javadoc 文件(Open and configure external Javadoc documentation)	如果想要利用 Shift+F2（「Navigate」→「Open External Javadoc」），来开启类型、方法或字段的 Javadoc 文件，必须先指定元素母项链接库（JAR、类别数据夹）或项目（来源数据夹）的文件位置。 对于链接库，请开启建置路径页面（「Project」→「Properties」→「Java Build Path」）、移至链接库，展开可以在其中编辑'Javadoc 位置'节点的链接库节点。文件可以放在本端档案系统上的一个数据夹中，也可以放在保存档或 Web 服务器中。 Java Build Path Source | Projects | Libraries | Order and Export JARs and class folders on the build path: junit.jar - org.junit Javadoc location: http://www.junit.org/junit/javadoc/3.8.1 Source attachment: junitsrc.zip - org.junit JRE System Library [ibm-java2-ws-sdk-pj9ndev-20040511] Add JA Add Extern Add Vari 对于来源数据夹中的类型、方法或字段，请移至（「Project」→「Properties」→「Javadoc Location」）。
仅显示所选元素的程序代码 Show Source of Selected Element Only)	工具列包括仅显示所选元素的程序文件按钮( )，这个按钮让只有所选概要元素的程序代码会显示在 Java 编辑器中。在底下范例中，仅显示 jackyMethod()方法。

内容	说明
	 再按一下仅显示所选元素的程序文件按钮，以重新查看整个 Java 檔。 

7.4 Java 视图(Java views)

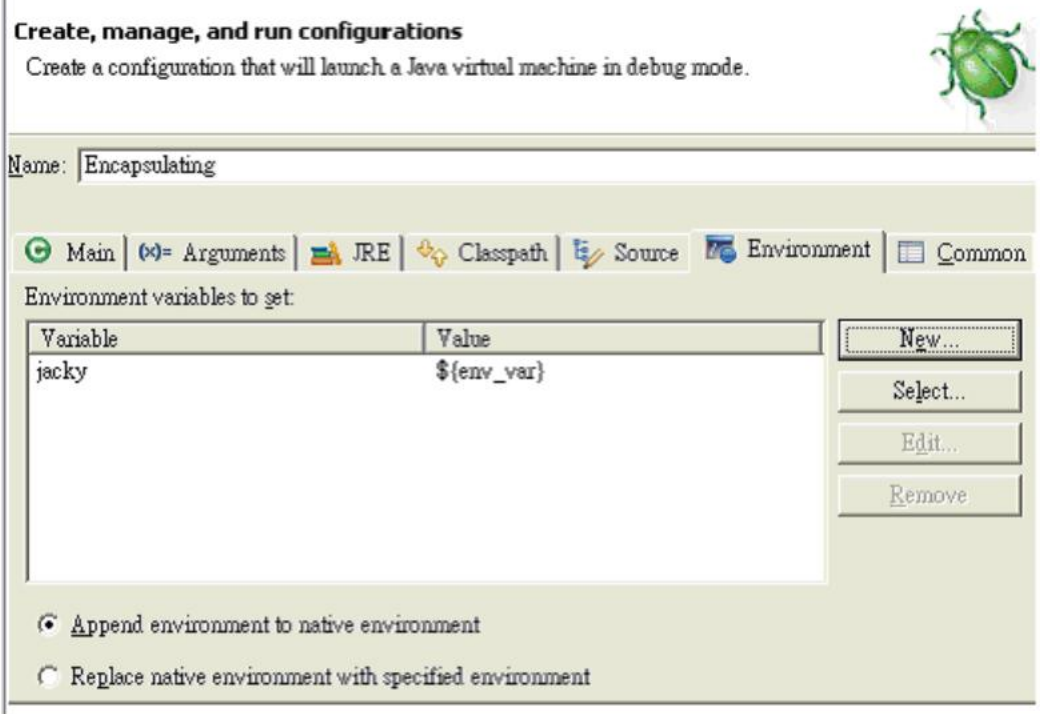
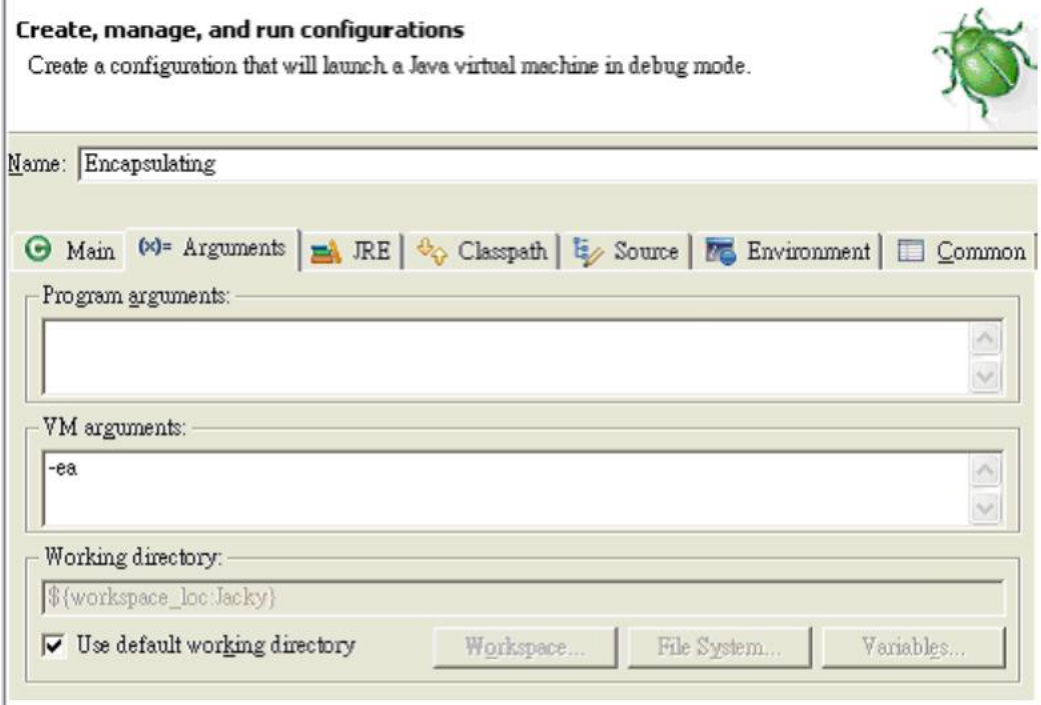
内容	说明
宣告视图(Declaration view)	新的「宣告」视图(「Window」→「Show View」→「Other...」→「Java」→「Declaration」)会显示在 Java 编辑器或 Java 视图中所选元素的程序代码。

内容	说明
	
Javadoc 视图(Javadoc view)	Javadoc 视图（「Window」→「Show View」→「Other...」→「Java」→「Javadoc」）会显示在 Java 编辑器或 Java 视图所选元素的 Javadoc。Javadoc 视图会使用 SWT 浏览器小组件，在支持它的平台上显示 HTML。 
类型阶层中的秘诀(Tricks in the type hierarchy)	在元素或所选名称上按 F4（「Navigate」→「Open Type Hierarchy」），将类型阶层的焦点放在新类型。 不单可以开启「Hierarchy」视图以显示类型，也可以显示套件、来源数据夹、JAR 保存文件与 Java 项目。 可以将元素拖放到「Hierarchy」视图中，以便将它的焦点放在该元素上。 可以从视图的工具列，变更「Hierarchy」视图的摆放方式（从预设的垂直方向到水平方向）。
找出在阶层中哪一个位置实作方法(Find out where a method is implemented in the hierarchy)	如果要了解阶层中哪些类型会置换方法，请使用「显示阶层中的成员」功能。 选取要查看的方法，然后按下 F4（「Navigate」→「Open Type Hierarchy」）。这将根据方法的宣告类型开启类型阶层视图。 如果「阶层」视图已经选取该方法，请按「Lock View and Show Members in Hierarchy」工具列按钮。 阶层视图现在仅显示实作或定义'已锁定'方法的类型。举例来说，可以看到'isEmpty()'在'List'中定义，且在'ArrayList'和'Vector'，但不在'AbstractList'中实作。

内容	说明
	
压缩套件名称 (Compress package names)	如果套件名称很长，可以配置一个在检视器显示的压缩名称。压缩型样的配置是在「Window」→「Preferences」→「Java」→「Appearance」中完成。 

7.5 除错(Debugging)

内容	说明
环境变量 (Environment Variables)	可以透过环境卷标，指定启动 Java 应用程序所用的环境。

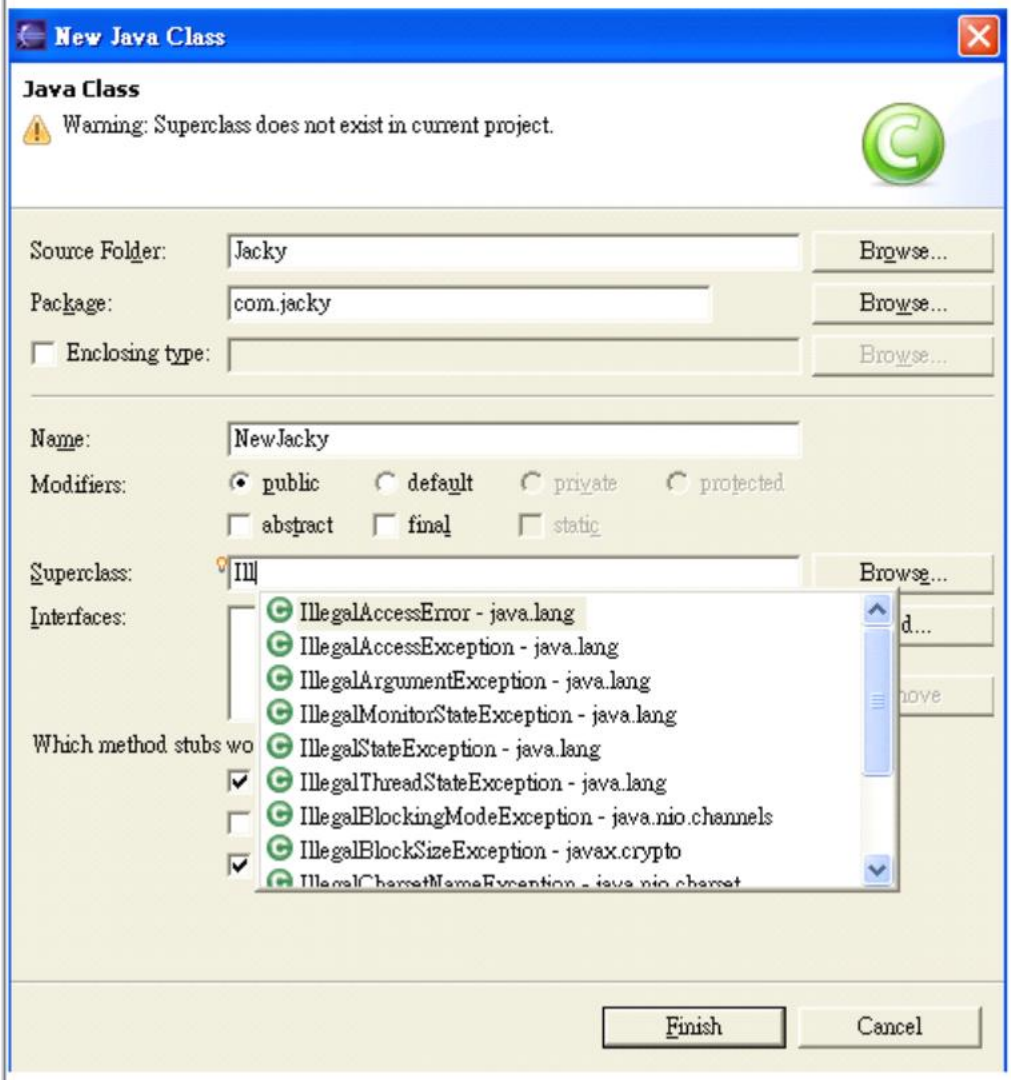
内容	说明
	
预设的 VM 自变量 (Default VM Arguments)	如果经常指定同样的自变量给特定的 VM，则可以在安装的 JRE 喜好设定页面，配置预设的 VM 自变量。比起每采用一个启动配置都要指定一次自变量，这样要方便得多。 
控制主控台 (Controlling your console)	主控台所显示的输出，可以透过「主控台」视图工具列中的 Pin 主控台动作，锁定至特定程序。另外还有一个卷动锁定动作，可以停止主控台在附加新输出时自动卷动。

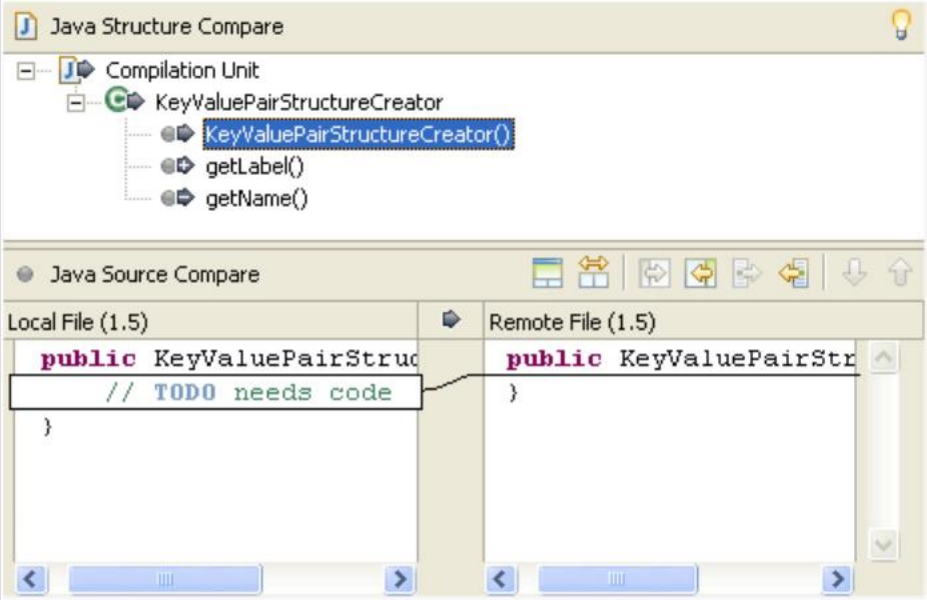
内容	说明
执行绪和监视器视图(Threads and Monitors view)	除错器的执行绪和监视器视图显示哪些执行绪正保有锁定，以及哪些正等待取得锁定。
逐行过滤器(Step filters)	逐行过滤器可以避免除错器在进入程序代码进行副程序除错时，在指定的类别和套件中暂停执行。逐行过滤器是在「Window」→「Preferences」→「Java」→「Debug」→「Step Filtering」中建立的。当使用逐行过滤器切换为开启时（在除错工具列和菜单），逐行过滤器会套用到所有的逐行动作。在「除错」视图中，所选堆栈框的套件或宣告类型可以迅速地新增至过滤器清单，方法为从堆栈框的快速菜单选取Filter Type或Filter Package。
执行有编译错误的程序代码(Running code with compile errors)	可以执行和除错并未清楚地编译的程序代码。执行有和没有编译错误的程序代码之间的唯一差异，就是如果执行一行有编译错误的程序代码，将发生下列两种情况之一： 如果在「Java」→「Debug」喜好设定页面上设定了'Suspend execution on compilation errors'，且正在进行除错，则除错阶段作业将暂停，如似遇到岔断点一般。请注意，如果 VM 支持「Hot Code Replace」，将可以修正编译错误并回复除错 否则，执行将终止，并出现'unresolved compilation'错误 有一点必须注意，只要执行路径避开有编译错误的程序代码行，就可以执行并除错，正如同平常所做一般。
堆栈追踪超链接(Stack trace hyperlinks)	主控台中会出现含超链接的 Java 堆栈追踪。当鼠标放在堆栈追踪中的某一行上时，光标就会变成手状，而且堆栈追踪之下会有一条底线出现。按下鼠标按钮将开启相关联的 Java 程序文件，并且会将光标定位在对应行中。如果在堆栈追踪顶端的异常状况名称上按一下鼠标按钮，就会建立一个异常状况岔断点。

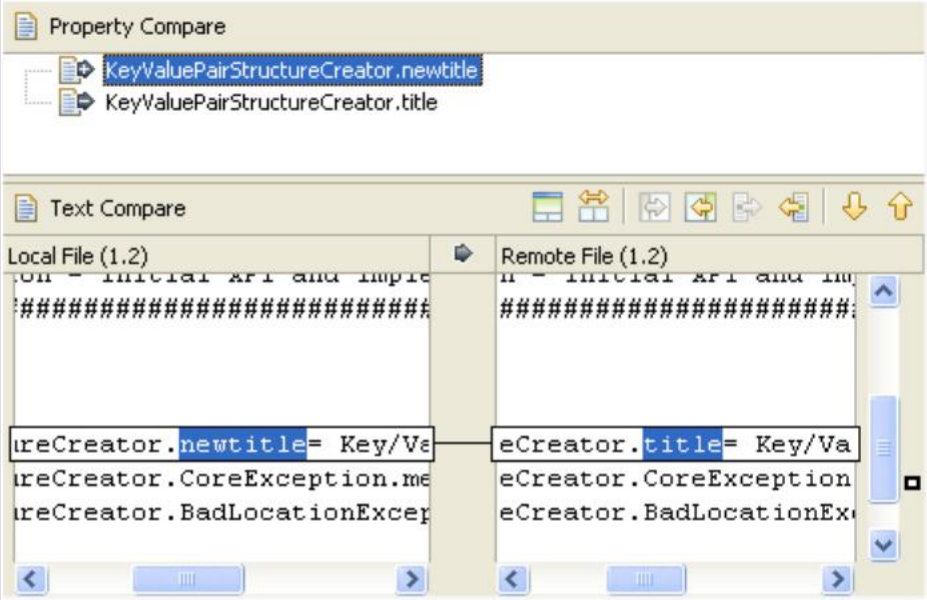
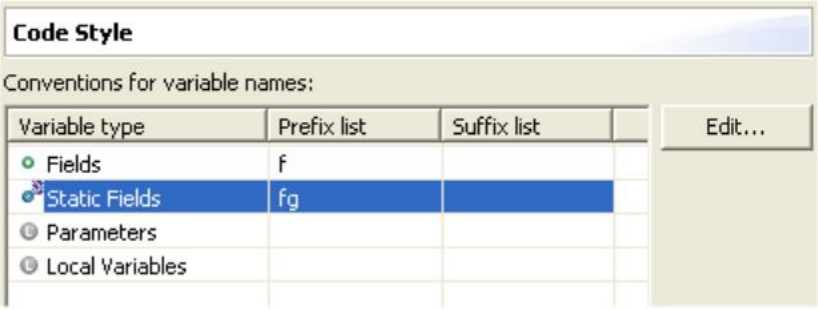
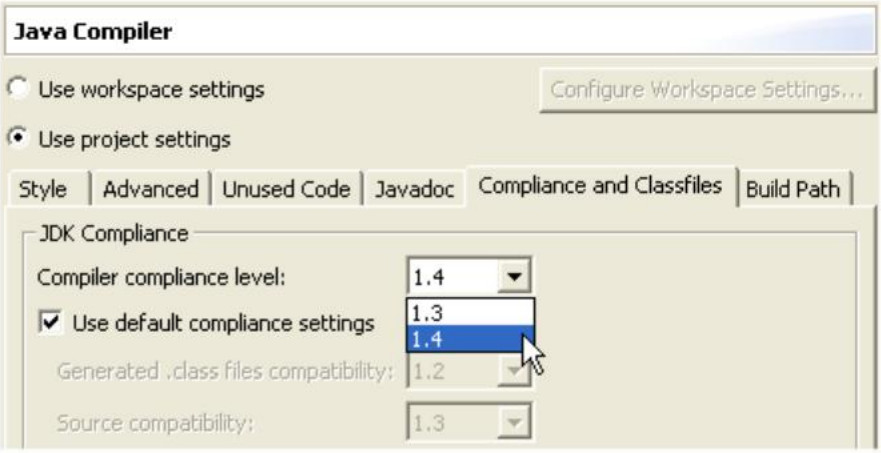
内容	说明
	Variables Console VectorTest [Java Application] C:\j2sdk1.4.2_04\bin\javaw.exe (Jun 24, 2004 3:26:43 PM) ... 16 more java.lang.ArrayIndexOutOfBoundsException: 4 >= 3 at java.util.Vector.elementAt(Vector.java:431) at junit.samples.VectorTest.testElementAt(Vecto at sun.reflect.NativeMethodAccessorImpl.invoke0 at sun.reflect.NativeMethodAccessorImpl.invoke(at sun.reflect.DelegatingMethodAccessorImpl.inv at java.lang.reflect.Method.invoke(Method.java:

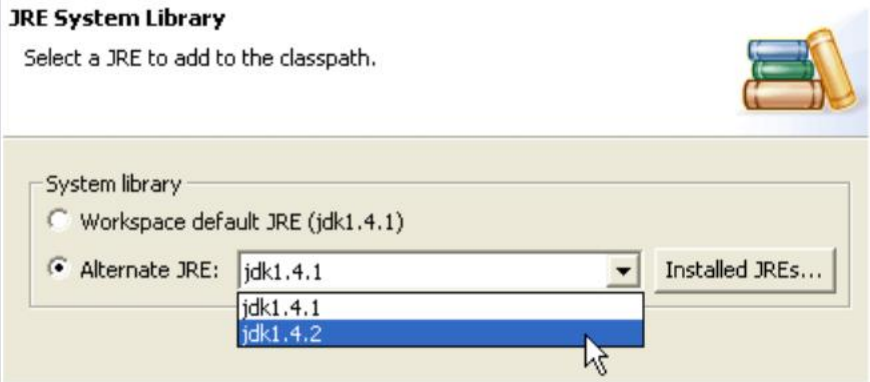
7.6 各种(Various)

内容	说明
JUnit	在视图选取 JUnit 测试方法，然后从快速菜单选取「Run」→「JUnit Test」；或者从主菜单选取「Run」→「Run As」→「JUnit Test」。这会建立一个启动配置来执行所选测试。
隐藏 JUnit 视图直到发生错误或失败(Hide JUnit view until errors or failures occur)	可以使用 JUnit 视图仅在发生错误或失败时才开启。 如此一来，可以让这个视图设成快速视图，且在没有失败测试时从不查看它。如果测试没有问题，那么在执行它们时，会看到这个图示 （小绿色方块的数目会随之增加，指出进度），而且在完成它们之后，也会看到这个图示 。不过，如果发生错误或失败，图示就会变成 （如果已经完成测试，则会变成 ），而且会出现 JUnit 视图。可以透过「Java」→「JUnit」喜好设定页面来配置这项行为。
对话框字段中的内容辅助(Content assist in dialog fields)	内容辅助(Alt+/)现在也可以在各种 Java 对话框的输入字段中使用。请寻找焦点所在的字段旁的小灯泡图示。

内容	说明
	 举个例说，内容辅助是在「New Java Class」、「 New Java Interface」和「New JUnit Test」精灵，以及在变更方法签章和移动 Static 成员的重构对话框中实作。 「Extract Local Variable」、「 Convert Local Variable to Field」以及「Introduce Parameter」重构，会提供内容辅助提议给新元素名称使用。
Java 程序文件的结构比较(Structural compare of Java source)	Java 程序文件的结构比较会忽略 Java 元素（如方法和字段）的文字次序，而更清楚的显示哪些元素已经变更、新增或是移除。 可以选择下列一种方法，来起始 Java 文件的结构比较： 选取两个 Java 编译单元(compilation units)，然后从视图的快速菜单选取「Compare With」→「Each Other」。如果档案不一样，将以「Compare Editor」开启它们。窗格顶端会显示不同的 Java 元素；如果按两下其中一个元素，便会在窗格底端显示该元素的程序文件。 在任何包含档案比较的环境定义中（如 CVS 同步化）按两下 Java 文件，不仅可以文字比较检视器显示档案内容，还会执行结构比较，并且开启新窗格来显示结果。

内容	说明
	 在执行结构比较时，甚至可以忽略批注和格式化变更：透过「Compare Editor」的工具列按钮，或「CVS Synchronization」视图的下拉菜单，来开启忽略空格选项(Ignore Whitespace)。
内容文件的结构比较(Structural compare of property files)	Java 内容档（扩展名：.properties）的结构比较会忽略内容的文字次序，而更清楚的显示哪些内容已经变更、新增或是移除。 可以选择下列一种方法，来起始内容文件的结构比较： 在「Package Explorer」或「Navigator」中选取两个档案，然后从视图的快速菜单选取「Compare With」→「Each Other」。如果档案具有差异，将以「Compare Editor」开启它们。窗格顶端显示受影响的内容；按两下它们之一将在窗格底端显示内容的程序文件。 在任何包含档案比较的环境定义中（如 CVS 同步化），按两下内容文件不仅可以在文字比较检视器显示档案内容，还会执行结构比较，并且开启新窗格来显示结果。

内容	说明
	
定义字段、参数和区域变量的前缀或字尾(Define prefixes or suffixes for fields, parameters and local variables)	除了配置字段的前缀或字尾之外，还可以指定 Static 字段、参数和区域变量的前缀或字尾。每当变量名称需要重新计算时，「Java」→「Code Style」喜好设定页面上的这些设定，就会用在内容辅助、快速修正以及重构作业。 
使用项目特有的编译器设定(Use project specific compiler settings)	每一个项目都可以决定要使用广域编译器设定，还是定义项目特有的设定。选取项目并在项目内容中开启 Java 编译器页面（「Project」→「Properties」→「Java Compiler」） 
对项目使用特定的 J	在建立新项目时预设会加入的 JRE，就是在「Preferences」→「Java」→「Installed JRE's」

内容	说明
RE(Use a specific JRE for a project)	所选的 JRE。如果要设定项目特定的 JRE，请开启项目的「Java Build Path」内容页面（「Project」→「Properties」→「Java Build Path」)、开启链接库页面、选取「Libraries」，然后按编辑。可以在「Edit Library」对话框中，选取要采用预设的 JRE，还是在新项目加入项目特有的 JRE。 
从异常不一致回复 (Recovering from abnormal inconsistencies)	在发生罕见的功能不良事件时，JDT 可能会发生一些不一致的情况，例如： Java Search 或 Open Type 漏掉结果 套件浏览器中的项目无效 如果要让它们回复一致，必须完全依照下列顺序，执行下列动作： 利用导览器关闭项目菜单动作，关闭所有的项目 结束和重新启动 Eclipse 利用导览器开启项目菜单动作，开启所有的项目 以手动方式触发重新建置整个工作区（「Project」→「Clean...」）