

Java Application Development using Eclipse

Richard Cole

7th March 2006

Session: 8358

Abstract



- ❖ Learn how to use the powerful features of Eclipse to aid software development.
 - 'Ease-of-use' features such as code generation, automatic syntactic checking and code completion reduce the time needed to write software
 - Debugging options including, step through execution, breakpoints, display of variable values, memory, registers, ... make finding those algorithmic bugs easier
 - Remote debugging facilities will leave you open mouthed.

Objectives

- ❖ An overview of Eclipse and its Architecture
- ❖ The Debugger
- ❖ Plugin's

Agenda

❖ Introduction

- Eclipse Overview and History
- Eclipse Architecture
- Perspectives

❖ The Resource Perspective

- Editing and syntax checking
- Running Programs

❖ The Debug Perspective

- Local Debugging
- Remote Debugging

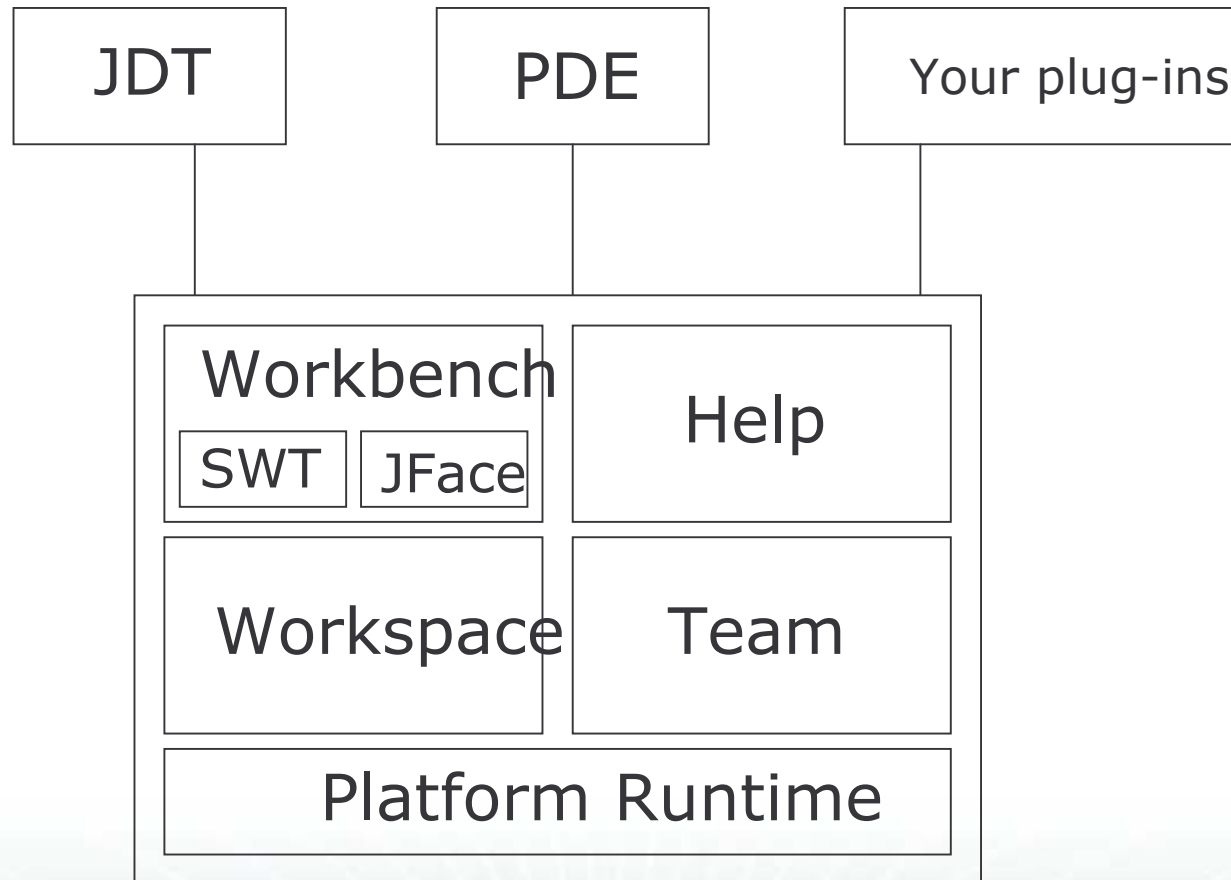
❖ Plug-In's

Eclipse Overview and History



- ❖ Eclipse is an extensible software development framework based on a plug-in architecture.
- ❖ Eclipse is open source, released under a common public license (CPL).
- ❖ Originally developed by IBM, version 1.0 being released in November 2001, Eclipse is now under the stewardship of an independent consortium including: IBM, SAP, Sybase, Fujitsu, HP, BEA...

Eclipse Architecture



Eclipse Architecture explained



Platform runtime

Primary job is to discover the available plug-ins. Each plug-in has an XML manifest file which lists the connections the plug-in requires.

The workspace

Manages the user's resources which are organised into projects. The workspace maintains a history of changes to resources.

The workbench

Eclipse's Graphical User Interface, menus, toolbars and perspectives. Perspectives provide a GUI for specific areas of functionality such as debugging, plug-in development,...

The Standard Widget Toolkit (SWT) are graphics toolkits which map directly to the OS native graphics.

JFace is a toolkit built on SWT.

Team Support

Version control. Eclipse provides a client for Concurrent Version System (CVS)

Help

An extensible help component

Java Development Toolkit (JDT)

Toolkit for the writing and debugging of Java programs. C Development Toolkit is the equivalent plug-in for C development.

Plug-in Development Environment (PDE)

Developing plug-ins for extending ECLIPSE.

Perspectives

- ❖ Eclipse perspective is a pre-selected set of views arranged in a predetermined way.
- ❖ This presentation is structured around describing functionality of the following perspectives:
 - Resource
 - Debug

Agenda

❖ Introduction

- Eclipse Overview and History
- Eclipse Architecture
- Perspectives

❖ The Resource Perspective

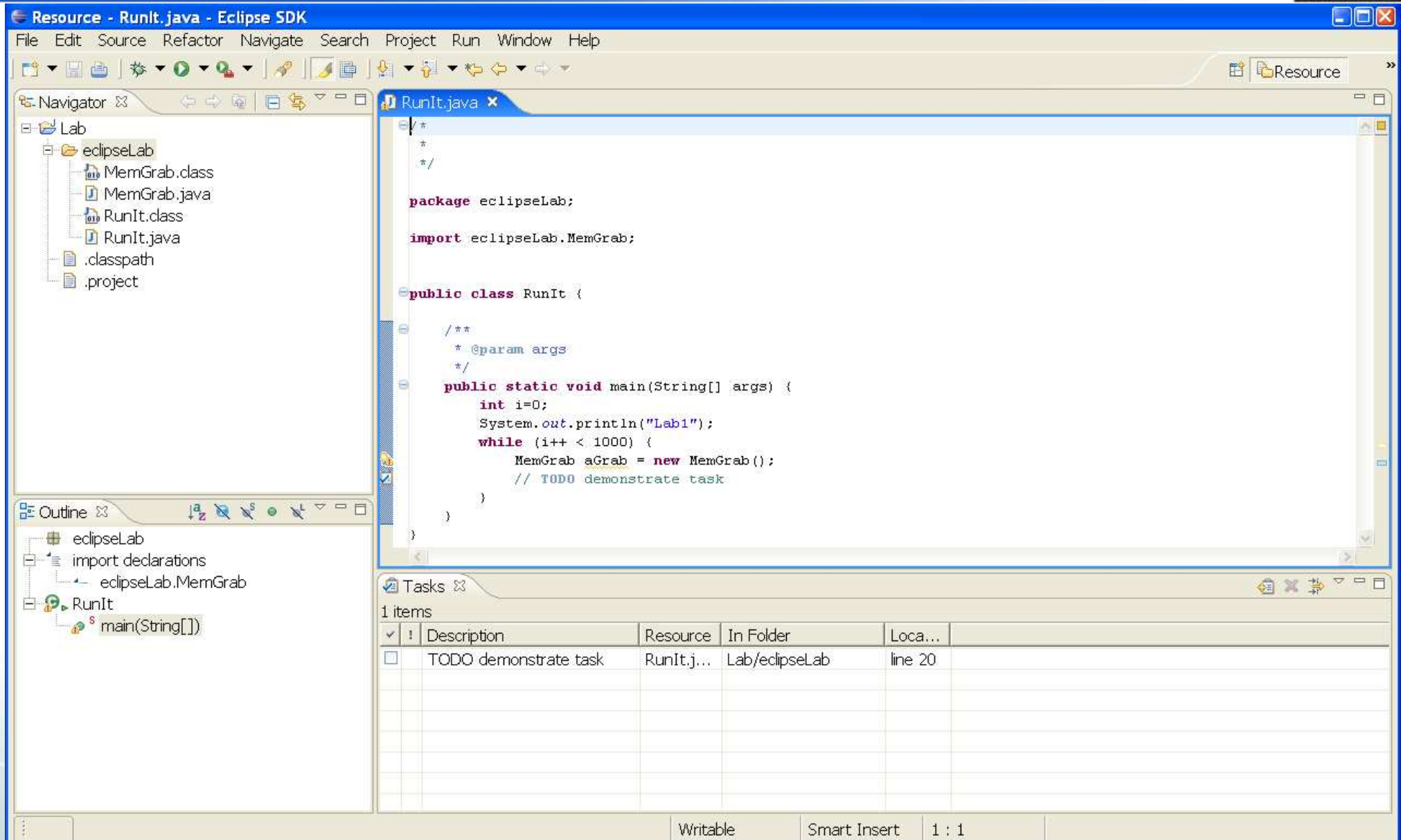
- Editing and syntax checking
- Running Programs

❖ The Debug Perspective

- Local Debugging
- Remote Debugging

❖ Plug-In's

Perspectives and Eclipse 1



The screenshot displays the Eclipse IDE interface for a project named "eclipseLab". The main editor window shows the source code for "RunIt.java". The code includes package and import statements, a public class definition, and a main method that prints "Lab1" and enters a loop to demonstrate a task.

```
/**
 *
 */
package eclipseLab;

import eclipseLab.MemGrab;

public class RunIt {

    /**
     * @param args
     */
    public static void main(String[] args) {
        int i=0;
        System.out.println("Lab1");
        while (i++ < 1000) {
            MemGrab aGrab = new MemGrab();
            // TODO demonstrate task
        }
    }
}
```

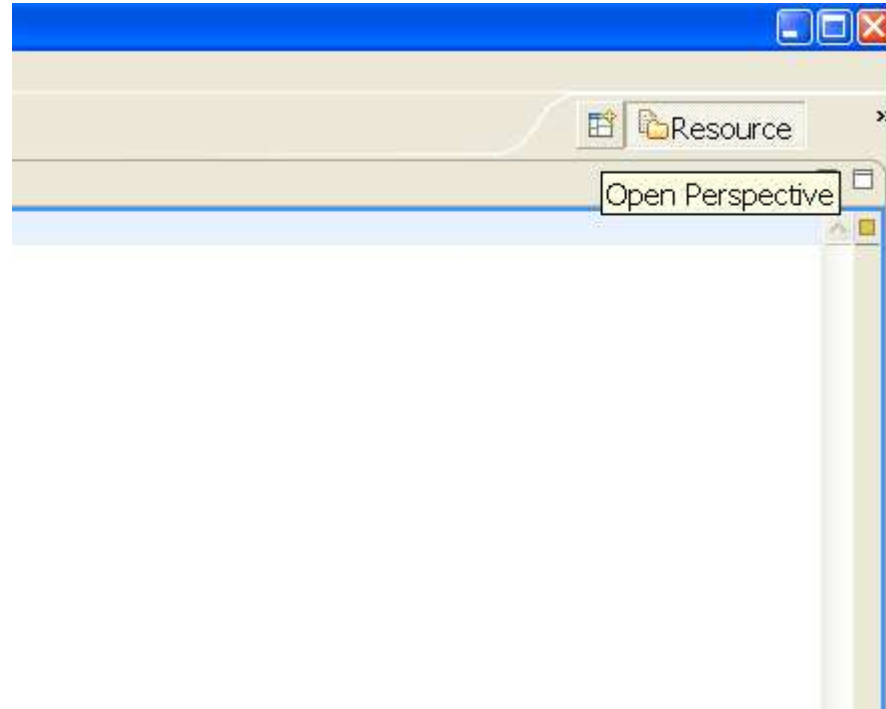
The Navigator view on the left shows the project structure, including the "eclipseLab" folder and its contents: "MemGrab.class", "MemGrab.java", "RunIt.class", "RunIt.java", ".classpath", and ".project". The Outline view shows the class hierarchy, including "eclipseLab", "import declarations", "eclipseLab.MemGrab", and "RunIt" with its "main(String[])" method.

The Tasks view at the bottom shows a table with 1 item:

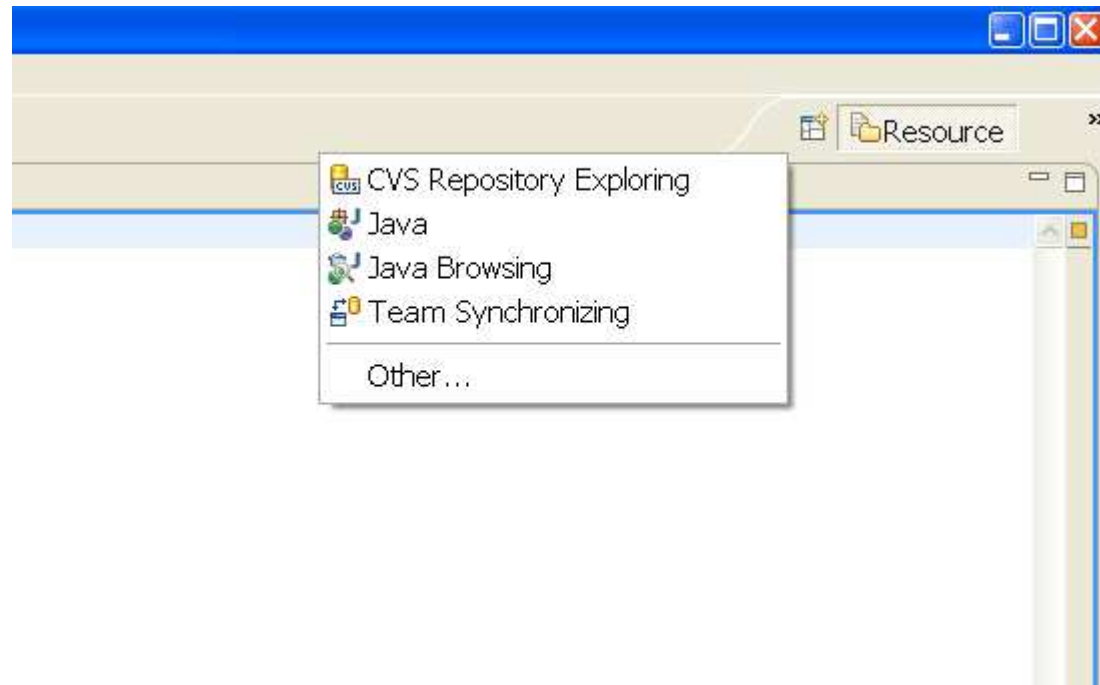
✓ !	Description	Resource	In Folder	Loca...
<input type="checkbox"/>	TODO demonstrate task	RunIt.j...	Lab/eclipseLab	line 20

The status bar at the bottom indicates "Writable", "Smart Insert", and "1 : 1".

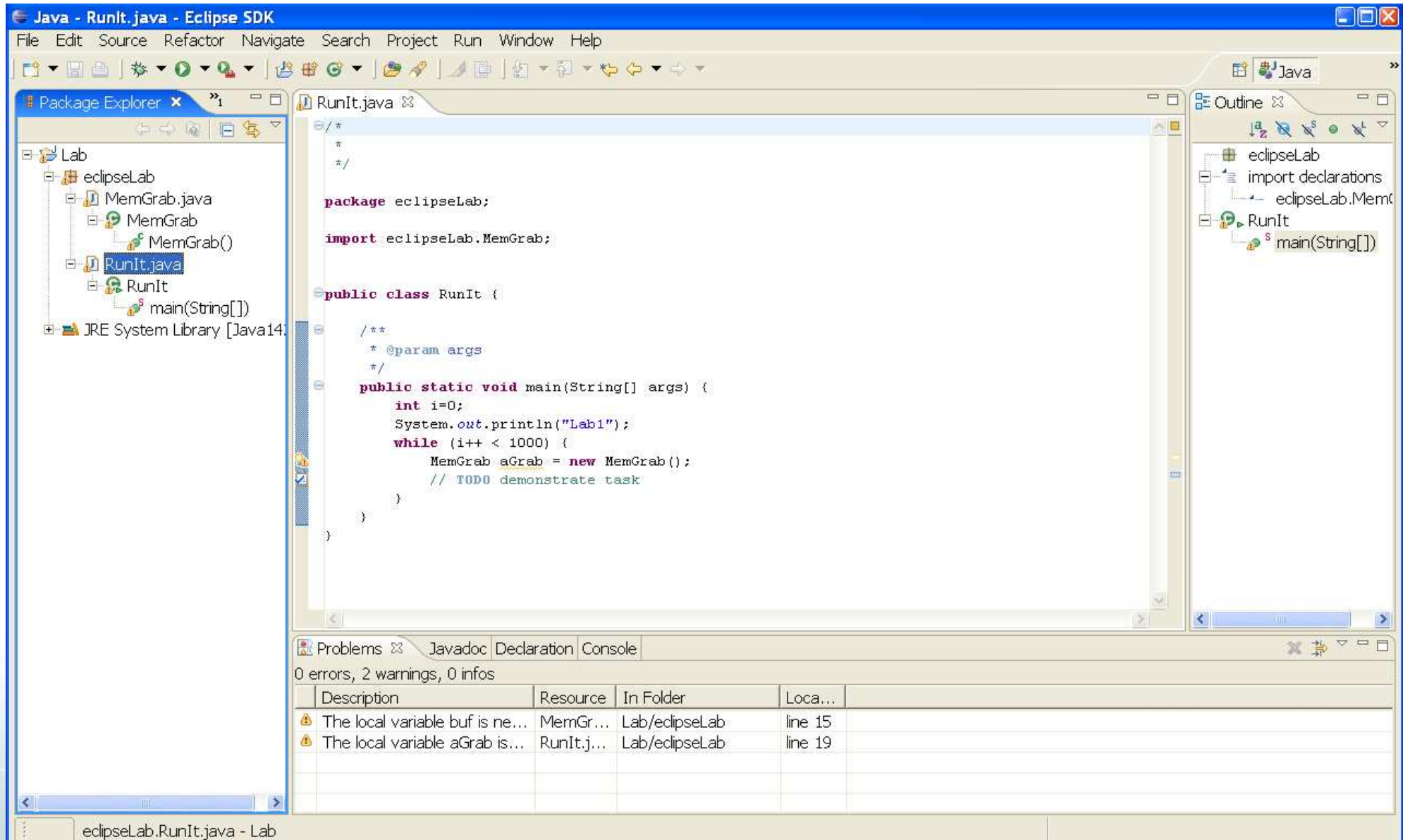
Perspectives and Eclipse 2



Perspectives and Eclipse 3



Java perspective



The screenshot displays the Eclipse IDE in the Java perspective. The main editor shows the source code for `RunIt.java` in the `eclipseLab` package. The code includes a `main` method that prints "Lab1" and enters a loop that creates `MemGrab` objects. The Package Explorer on the left shows the project structure, and the Outline on the right shows the class and method structure. The Problems view at the bottom shows two warnings related to local variables.

```
/*
 *
 */
package eclipseLab;

import eclipseLab.MemGrab;

public class RunIt {

    /**
     * @param args
     */
    public static void main(String[] args) {
        int i=0;
        System.out.println("Lab1");
        while (i++ < 1000) {
            MemGrab aGrab = new MemGrab();
            // TODO demonstrate task
        }
    }
}
```

Description	Resource	In Folder	Loca...
The local variable buf is ne...	MemGr...	Lab/eclipseLab	line 15
The local variable aGrab is...	RunIt.j...	Lab/eclipseLab	line 19

Agenda



❖ Introduction

- Eclipse Overview and History
- Eclipse Architecture
- Perspectives

❖ The Resource Perspective

- Editing and syntax checking
- Running Programs

❖ The Debug Perspective

- Local Debugging
- Remote Debugging

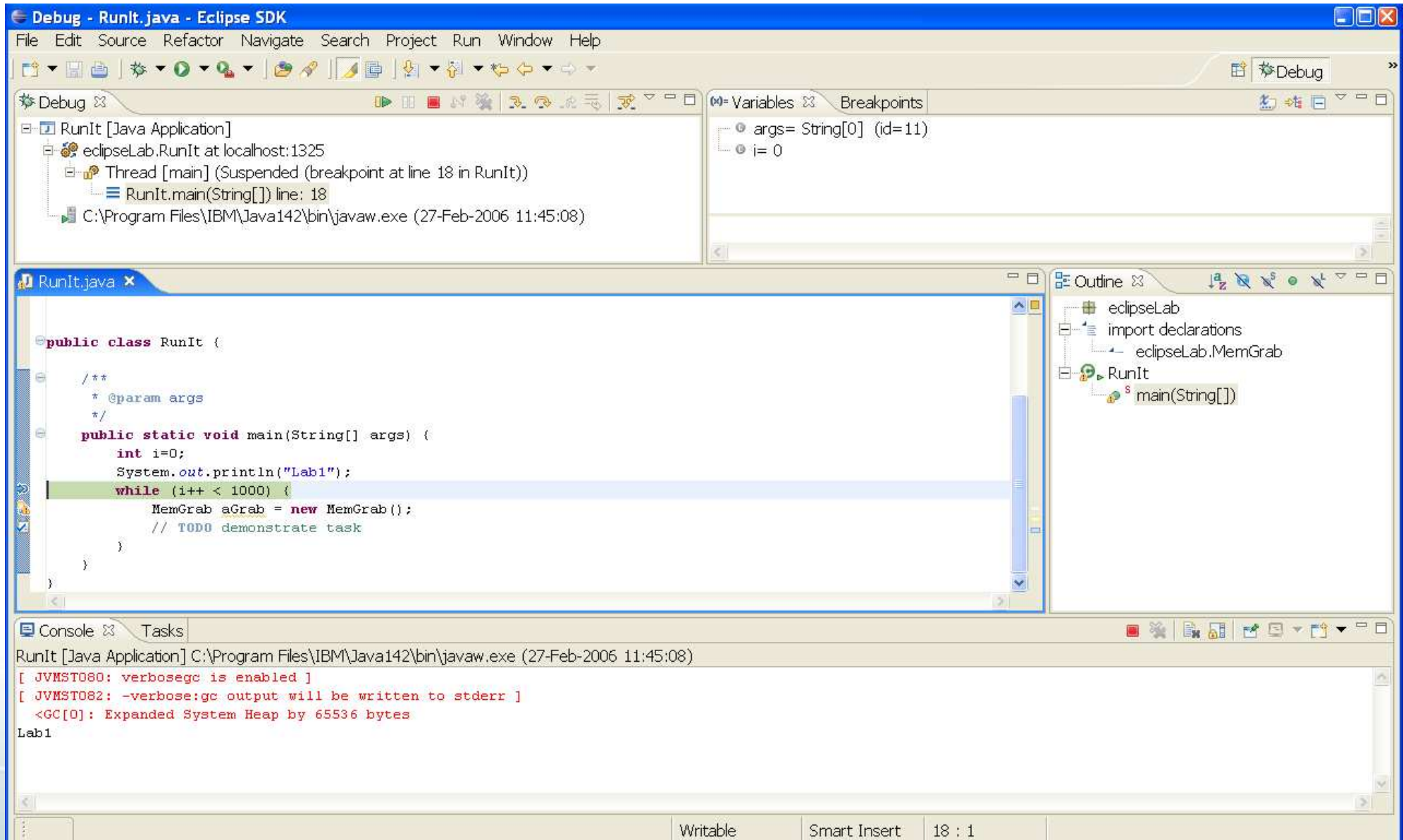
❖ Plug-In's

Debug Perspective

- ❖ The debug perspective gives you access to functionality which enables the user to run code interactively by:
 1. Stepping through the execution line by line.
 2. Setting break points at places in the code where the execution will suspend
 3. Examining program attributes, such as variables, locks, memory, registers,... at any given (break) point
 4. Modifying variables during program execution.
 5. Modifying code during program execution

- ❖ This enables the user to:
 1. Find code errors.
 2. Unit test

Debug Perspective



The screenshot shows the Eclipse IDE in the Debug perspective. The main window is titled "Debug - RunIt.java - Eclipse SDK". The interface is divided into several panes:

- Debug Console:** Shows the execution stack for "RunIt [Java Application]". The current thread is "Thread [main] (Suspended (breakpoint at line 18 in RunIt))", which is paused at "RunIt.main(String[]) line: 18". The process is "C:\Program Files\IBM\Java142\bin\javaw.exe (27-Feb-2006 11:45:08)".
- Variables:** Displays the current state of variables: "args= String[0] (id=11)" and "i= 0".
- Source Editor:** Shows the source code for "RunIt.java". The code is as follows:

```
public class RunIt {  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        int i=0;  
        System.out.println("Lab1");  
        while (i++ < 1000) {  
            MemGrab aGrab = new MemGrab();  
            // TODO demonstrate task  
        }  
    }  
}
```

The line "while (i++ < 1000) {" is highlighted in green, indicating the current execution point.
- Outline:** Shows the project structure with "RunIt" selected, and its "main(String[])" method.
- Console:** Shows the output of the application, including JVM startup messages and the printed text "Lab1".

The status bar at the bottom indicates "Writable", "Smart Insert", and "18 : 1".

Remote debugging

- ❖ Debug a java application running on z/OS from Eclipse
- ❖ Configure Eclipse
 - Build the class to be debugged in a project as before.
 - The source code is the same as on the remote machine
 - Select Run>Debug>Remote Java Application and create a configuration
 - Fill in host (IP address) / port (where the remote VM is accepting connections) and source details, see next slide

Remote debugging, remote side

- ❖ Build the java application with the `-g` flag

```
Javac -g Polynomial.java
```

- ❖ Start the JVM in server/suspend mode, so that you can then attach the Eclipse Java debugger to it.

The following is the content of a script 'runMe'

```
java -Xdebug -Xnoagent -Djava.compiler=NONE  
-runjdwp:transport=dt_socket,  
server=y,suspend=y,address=8888 $* &
```

Remote debugging, pulling it together



- ❖ Start the java application on the remote machine with 'runMe Polynomial'. Polynomial is being executed but is suspended listening on port 8888
- ❖ Start the remote debug session in Eclipse. Step through the application and watch the output appear in the remote console.

Agenda

❖ Introduction

- Eclipse Overview and History
- Eclipse Architecture
- Perspectives

❖ The Resource Perspective

- Editing and syntax checking
- Running Programs

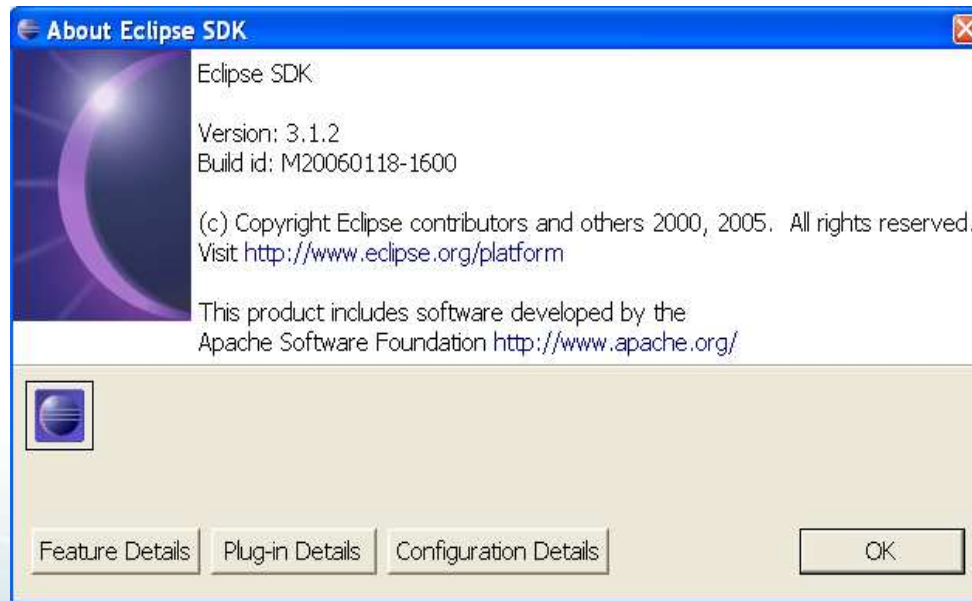
❖ The Debug Perspective

- Local Debugging
- Remote Debugging

❖ Plug-In's

Plug-In's

- ❖ Eclipse is a framework which enables plug-in operability and interoperability
- ❖ Click on 'Help>About Eclipse Platform' then 'Plug-in Details' to see currently active Plug-ins
- ❖ <http://www.eclipse.org/documentation/html/plugins/org.eclipse.platform.doc.isv/>



Where to get Plug-ins

❖ Eclipse plug-in download site
http://www.eclipseplugincentral.com/Web_Links+main.html

Main plugins categories

- | | |
|--|---|
|  Application Management (10) |  Application Server (11) |
|  Code Management (24) |  Database (26) |
|  Deployment (6) |  Documentation (8) |
|  Editor (33) |  Entertainment (6) |
|  Graphics (3) |  IDE (27) |
|  J2EE Development Platform (16) |  J2ME (4) |
|  Languages (24) |  Modeling (18) |
|  Network (8) |  Other (18) |
|  Profiling (8) |  Rich Client Applications (18) |
|  SCM (2) |  Source Code Analyzer (17) |
|  Team Development (22) |  Testing (29) |
|  Tools (71) |  UI (17) |
|  UML (14) |  Web (22) |
|  Web Services (11) |  XML (12) |

❖ Also; <http://www.eclipse-plugins.com/>

Lab Exercises



- ❖ Lab 1 The Resource Perspective
- ❖ Lab 2 The Debug Perspective
- ❖ Lab 3 Creating a HelloWorld Plug-In
- ❖ Lab 4 Deploying the HelloWorld Plug-In