



How to Write Your Own Eclipse Plug-ins

Beth Tibbitts

IBM

tibbitts@us.ibm.com

"This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA)
under its Agreement No. HR0011-07-9-0002"

OSCON 2009, June 2009

How to Write Your Own Eclipse Plug-ins :
OSCON 09



Before Attending

- Make sure you have a Java JRE
 - At least 5 (aka 1.5)
 - IBM or Sun's (not GCJ included with Linux)
- Download Eclipse *Classic* for your platform
 - eclipse.org/downloads
 - Save the archive file, you'll need it again in the tutorial
 - Current release “Galileo” is Eclipse 3.5



Java

- Linux: make sure you have Sun or IBM Java:
Java HotSpot == Sun

A screenshot of a Linux desktop environment showing a terminal window. The window title bar says "beth@localhost:~". The menu bar includes "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal window displays the following command and output:

```
[beth@localhost ~]$ java -version
java version "1.6.0_10"
Java(TM) SE Runtime Environment (build 1.6.0_10-b33)
Java HotSpot(TM) Client VM (build 11.0-b15, mixed mode, sharing)
[beth@localhost ~]$ ■
```

- Either Java 1.5 or 1.6 is ok (*Not gcj!*)



Outline

1. Eclipse overview and Installation
2. Installing extensions (“Plug-ins”) via Update Manager
3. New Plug-in Project Wizard, create an Action button
 - Plug-in component tour: Manifest, code, resources, etc.
 - Launching (“self-hosting”) – testing your plug-in
4. More extensions: adding a view ... modify code ... run debugger
5. Writing Help for your plug-in; including HTML and XML editing
6. Packaging your plug-in(s): Eclipse features; Update Sites & testing
7. Handmade extension: Word Counter in editor toolbar -- If time permits
8. Source code analysis: C/C++ example – and eclipse.org source code repository exploring -- If time permits
9. Summary, References, Where to go from here

Hints and Tips Along the Way



Eclipse: Overview

- Open Source IDE & Development Platform
 - Platform-independent
 - Originally donated by IBM in 2001; Foundation est. in 2004
 - Now over 180 vendor members (more than IBM)
 - 15 Strategic members – sit on Eclipse Foundation Board
 - <http://www.eclipse.org/membership/showMembersWithTag.php?TagID=strategic>
 - Now over 33 Eclipse projects in simultaneous release
 - Over 100 projects overall; more from independents
 - IDE (*develop with Eclipse*)
 - *And Platform (develop and deliver your Application on top of Eclipse)*



Platforms & Releases

Eclipse SDK +	
Status	Platform
✓	Windows (Supported Versions)
✓	Windows (x86_64) (Supported Versions)
✓	Windows (x86/WPF) **early access** (Supported Versions)
✓	Linux (x86/GTK 2) (Supported Versions)
✓	Linux (x86/Motif) **testing only**
✓	Linux (x86_64/GTK 2) (Supported Versions)
✓	Linux (PPC/GTK 2) (Supported Versions)
✓	Linux (S390/GTK 2) **early access** (Supported Versions)
✓	Linux (S390x/GTK 2) **early access** (Supported Versions)
✓	Solaris 10 (SPARC/GTK 2)
✓	Solaris 10 (x86/GTK 2)
✓	HP-UX (IA64_32/Motif)
✓	AIX (PPC/Motif)
✓	Mac OSX (Mac/Cocoa) (Supported Versions)
✓	Mac OSX (Mac/Cocoa/x86_64) (Supported Versions)
✓	Mac OSX (Mac/Carbon) (Supported Versions)
✓	Source Build (Source in .zip) (instructions)
✓	Source Build (Source fetched via CVS) (instructions)

- Releases
 - 2006: 3.2 Callisto
 - 2007: 3.3 Europa
 - 2008: 3.4 Ganymede
 - 2009: 3.5 Galileo 



Eclipse Packages

<http://eclipse.org/downloads>

Eclipse Downloads

Eclipse Packages Projects Compare Packages

Galileo Packages (based on Eclipse 3.5)

Eclipse IDE for Java EE Developers (187 MB)
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn and others. [More...](#)
Downloads: 14,912

Eclipse IDE for Java Developers (91 MB)
The essential tools for any Java developer, including a Java IDE, a CVS client, XML Editor and Mylyn. [More...](#)
Downloads: 5,240

Eclipse IDE for C/C++ Developers (78 MB)
An IDE for C/C++ developers with Mylyn integration. [More...](#)
Downloads: 2,844

Eclipse for PHP Developers (137 MB)
Tools for PHP developers creating Web applications, including PHP Development Tools (PDT), Web Tools Platform, Mylyn and others. [More...](#)
Downloads: 2,668

Eclipse for RCP/Plug-in Developers (182 MB)
A complete set of tools for developers who want to create Eclipse plug-ins or Rich Client Applications. It includes a complete SDK, developer tools and source code, plus Mylyn, an XML editor and the Eclipse Communication Framework. [More...](#)
Downloads: 1,386

Eclipse Modeling Tools (includes Incubating components) (362 MB)
This modeling package contains a collection of Eclipse Modeling Project components, including EMF, GMF, MDT XSD/OCL/UML2, M2M, M2T, and EMFT elements. It includes a complete SDK, developer tools and source code. Note that the Modeling package includes some *Incubating* components, as indicated by feature numbers less than 1.0.0 on the feature list. [More...](#)
Downloads: 923

Eclipse IDE for Java and Report Developers (218 MB)
JEE tools and BIRT reporting tool for Java developers to create JEE and Web applications that also have reporting needs. [More...](#)
Downloads: 898

Pulsar for Mobile Java Developers (112 MB)
Pulsar is a tools platform for Mobile Java Developers. It includes the Eclipse Platform, Java Development Tools (JDT), Mobile Tools for Java (MTJ), Mylyn and Plugin Development Environment (PDE). Pulsar also makes it easy to download SDK from different handset manufacturers. [More...](#)
Downloads: 344

Eclipse Classic 3.5.0 (161 MB)
The classic Eclipse download: the Eclipse Platform, Java Development Tools, and Plug-in Development Environment, including source and both user and programmer documentation. Please look also at the [Eclipse Project download page](#). [More...](#)
[Release notes](#) | [Other downloads](#) | [Documentation](#)
Downloads: 5,244

Compare Packages

Check out this table to find out what is included in each package.

	Java	Java EE	C/C++	RCP/Plugin	Modeling	Reporting	PHP	Pulsar	Classic
RCP/Platform	✓	✓	✓	✓	✓	✓	✓	✓	✓
CVS	✓	✓	✓	✓	✓	✓	✓	✓	✓
EMF	✓	✓			✓	✓			
GEF	✓	✓		✓	✓	✓			
JDT	✓	✓		✓	✓	✓			✓
Mylyn	✓	✓	✓	✓	✓	✓	✓	✓	✓
UDC	✓	✓	✓	✓	✓	✓	✓	✓	✓
Web Tools		✓				✓	✓		
Java EE Tools		✓				✓			
XML Tools	✓	✓		✓		✓	✓	✓	
RSE		✓				✓			
EclipseLink		✓				✓			
PDE		✓		✓	✓	✓			✓
Datatools		✓				✓			
CDT				✓					
BIRT							✓		
ECF					✓				
GMF						✓			
PDT								✓	
MDT					✓				
MTJ									✓

Legend:

✓ Included (with Source) ✓ Included ✓ Partially Included



Parallel Tools Platform

My committer status involvement with Eclipse:

- Eclipse.org/ptp
- Tools for High Performance Computing
- IBM funding from DARPA
- Develop, Analyze, Run, Monitor, Debug, Tune, Repeat



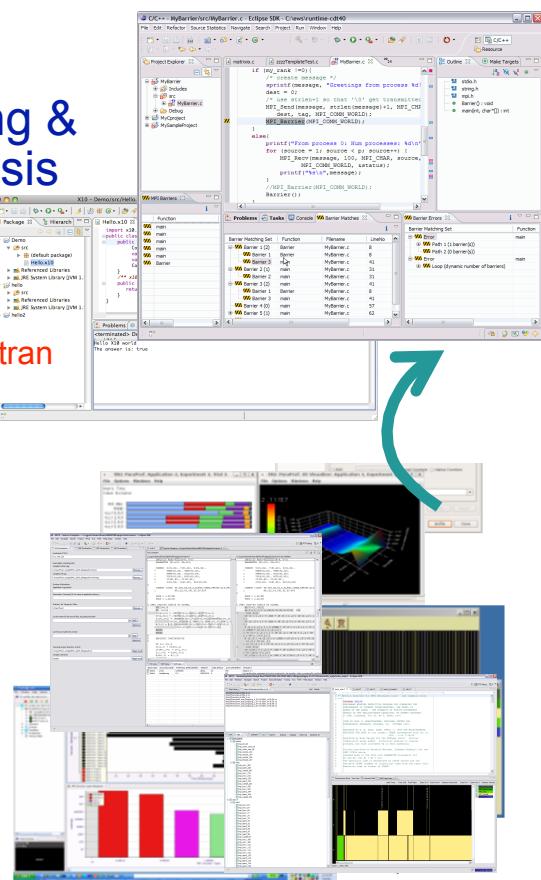
Eclipse and PTP: Parallel Tools Platform

<http://eclipse.org/ptp>

- Bring richness of commercial development tools to the parallel programmer
- Open and extensible platform to encourage further development

Coding &
Analysis

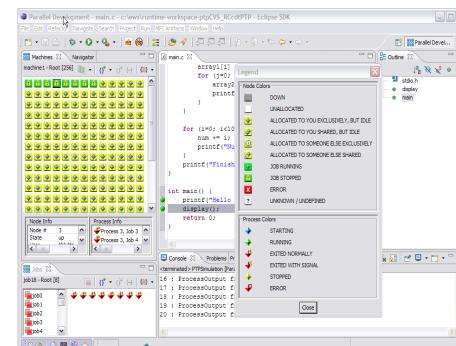
CDT: C/C++
Photran: Fortran
X10DT: X10



Productivity of
Dev. On local
machine



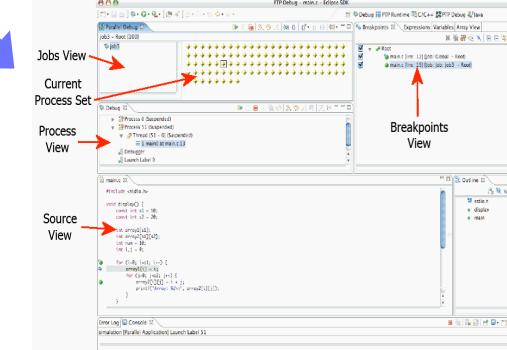
Building
Launching
Monitoring



Remote
Host



Parallel Debugging



Performance Tuning
External Tools Framework (ETFw)



Eclipse Installation

- Download from eclipse.org/downloads
 - You need a Java JRE (at least JRE 5)
 - *Not GCJ distributed with Linux! Get Sun or IBM.*
 - This tutorial is based on Eclipse 3.5 (“Galileo”)
 - Many distributions – we’ll use “Eclipse Classic”
 - Other distros target different dev. Audiences
 - C/C++, Web, Java, Java EE, etc.
 - Download and unzip/untar
 - That’s it!
 - Executable: `eclipse` or `eclipse.exe`



Hints and tips: Multiple installations

You can easily have multiple installations of eclipse.

Eclipse extracts into a single dir: 'eclipse'

Put that wherever you want

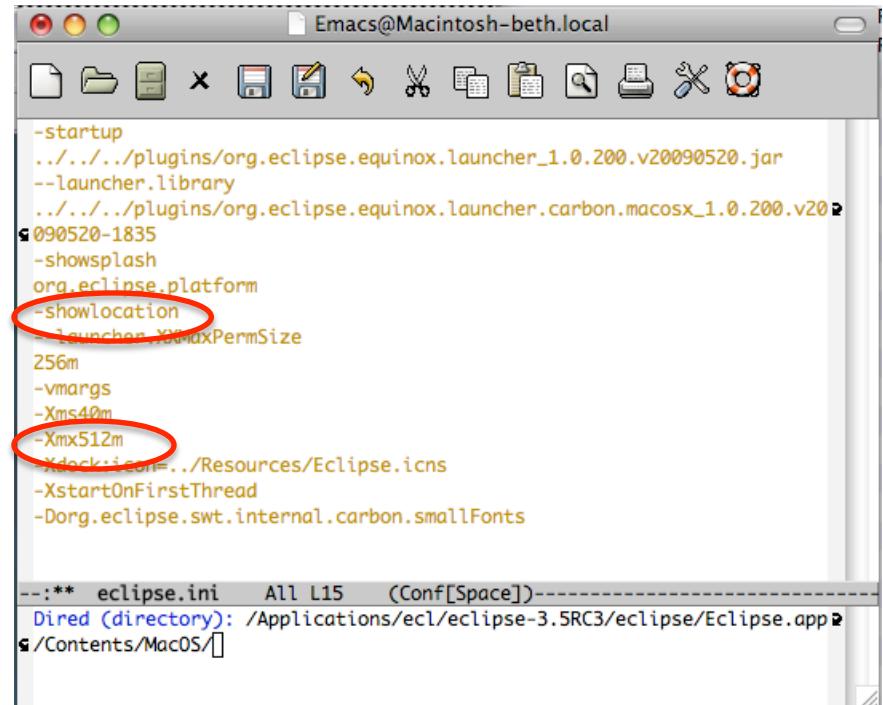
- */Users/beth/ecl/eclipse-3.5/eclipse*
 - */Users/beth/ecl/eclipse-3.4/eclipse*
 - */Users/beth/ecl/eclipse-3.3/eclipse*
-
- *Right now I have ____ (how many?) on my machine*
 - *What can I say, I'm a pack rat ☺*



Hints and tips: eclipse.ini

Before Launching

- *Hints and Tips*
 - *Edit the eclipse.ini file to set defaults.*
Suggestions:
 - *Change heap size from -Xms256m to -Xmx512m*
 - *Set -showlocation to show workspace name in title bar*
 - *Don't split another arg/value pair!*
- *Eclipse.ini location*
 - *Mac: eclipse/Eclipse.app/Contents/MacOS/eclipse.ini*
 - *Win/Linux: eclipse/eclipse.ini*



```
-startup
../../../../plugins/org.eclipse.equinox.launcher_1.0.200.v20090520.jar
--launcher.library
../../../../plugins/org.eclipse.equinox.launcher.carbon.macosx_1.0.200.v20090520-1835
-showsplash
org.eclipse.platform
-showlocation
-launcher.XmxPermSize
256m
-vmargs
-Xms40m
-Xmx512m
dock-icon=../Resources/Eclipse.icns
-XstartOnFirstThread
-Dorg.eclipse.swt.internal.carbon.smallFonts

--:** eclipse.ini  All L15  (Conf[Space])
Dired (directory): /Applications/ecl/eclipse-3.5RC3/eclipse/Eclipse.app
/Contents/MacOS/
```

See next slide for Windows/Linux



Hints and tips: eclipse.ini

Windows

```
eclipse.ini - Epsilon
File Edit Search Process Utility Window Help
File Edit Search Process Utility Window Help
-startup
plugins/org.eclipse.equinox.launcher_1.0.200.v20090520.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.0.200.v20090519
-showsplash
org.eclipse.platform
-showlocation
--launcher.XXmaxPermSize
256m
-vmargs
-Xms40m
-Xmx512m

-- \ecl\eclipse-3.5RC3\eclipse\eclipse.ini [Ini Unix]All * Line: 7 Co
Now 70 x 16
```

Linux

```
beth@localhost:~/ecl/eclipse-3.5/eclipse
File Edit Options Buffers Tools Help
File Edit Options Buffers Tools Help
-startup
plugins/org.eclipse.equinox.launcher_1.0.200.v20090520.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.gtk.linux.x86_1.0.200.v20090520
-showsplash
org.eclipse.platform
-showlocation
--launcher.XXmaxPermSize
256m
-vmargs
-Xms40m
-Xmx256m

-Xmx512m

-eu-----F1 eclipse.ini All L1 (Conf[Space])-----
Loading conf-mode...done
```



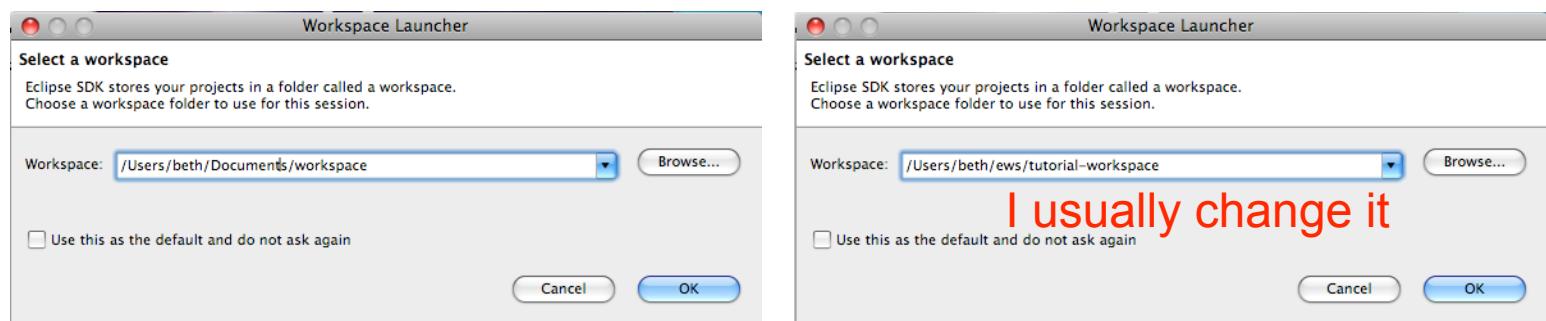
Launch Eclipse

- `./eclipse` or double-click executable
 - Warning: to assure environment is translated to the executable, you may want to run from command line (e.g. Mac OS X)
 - Or edit: `environment.plist`
 - Whatever your environment, make sure Java is in your path

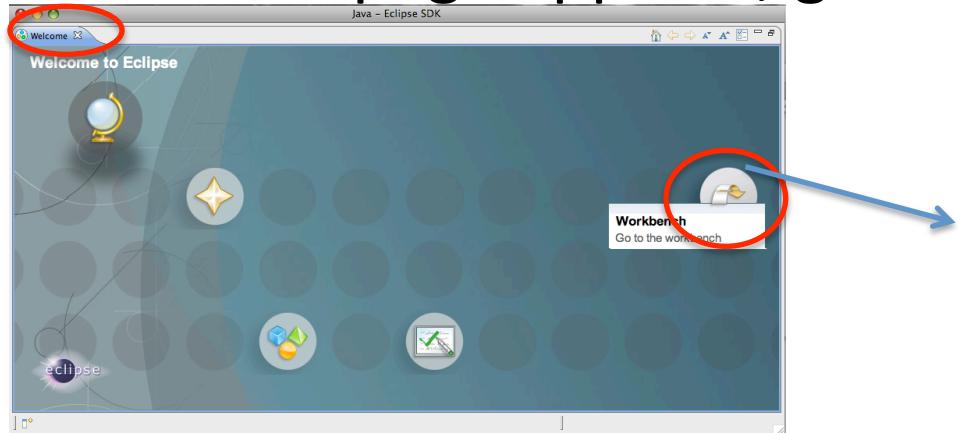


Launch Eclipse

- Launch eclipse; default workspace probably OK



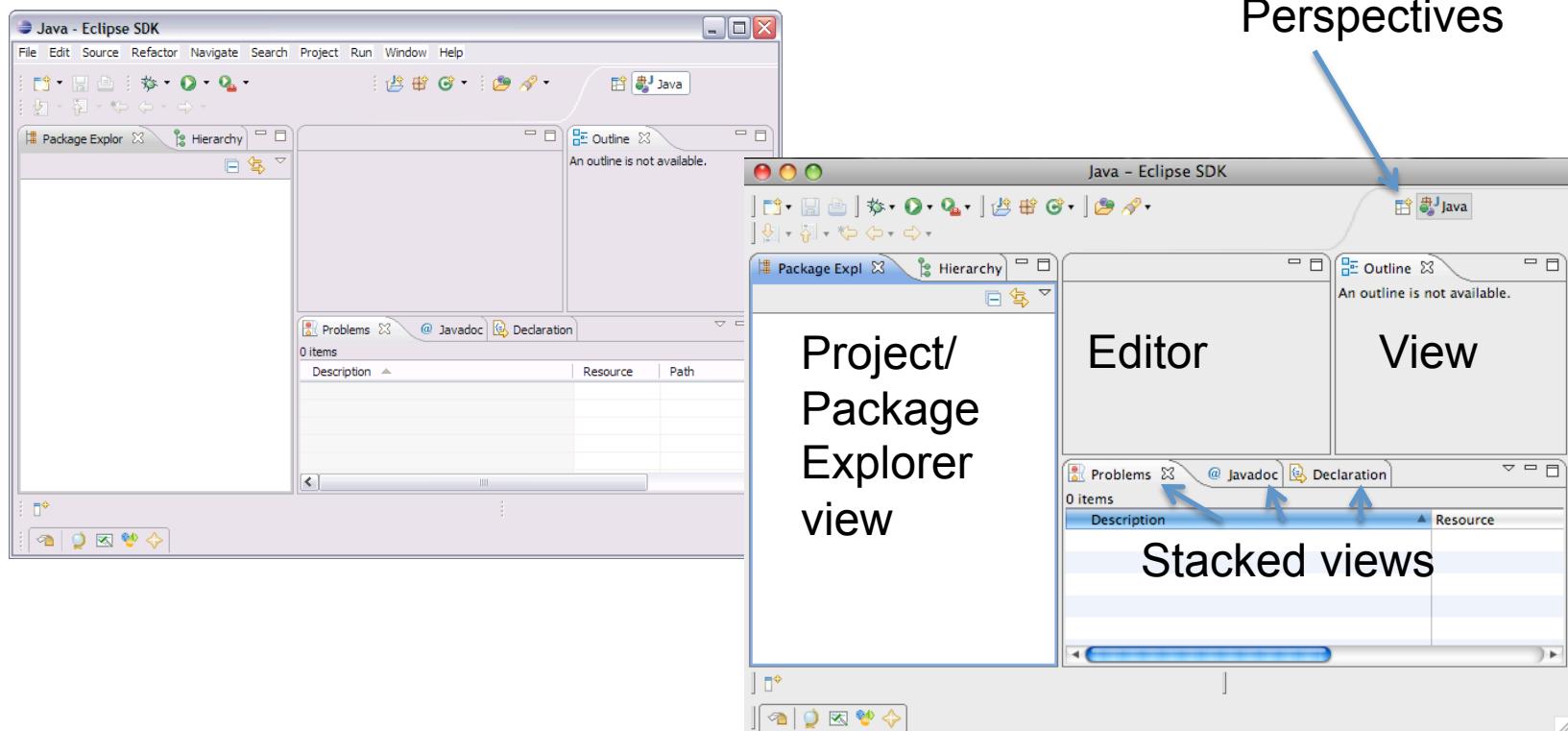
- Welcome page appears; go to workbench



- Close Welcome tab
or
- Select “Go to workbench” icon



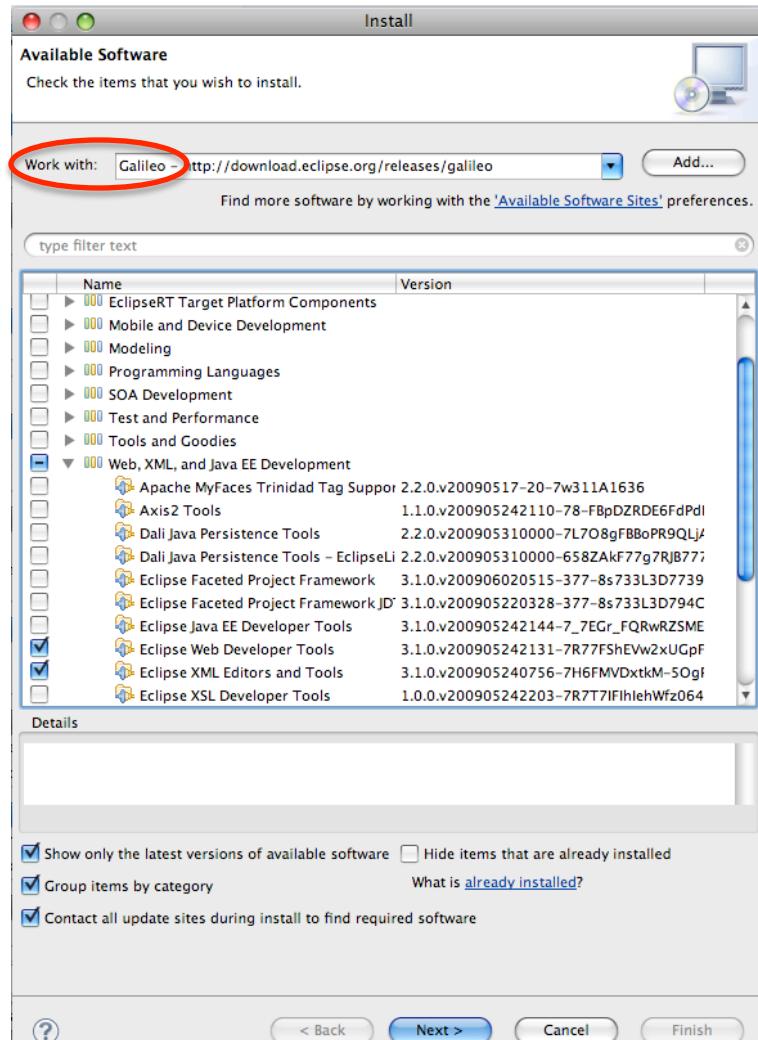
Eclipse Workbench





Installing extensions

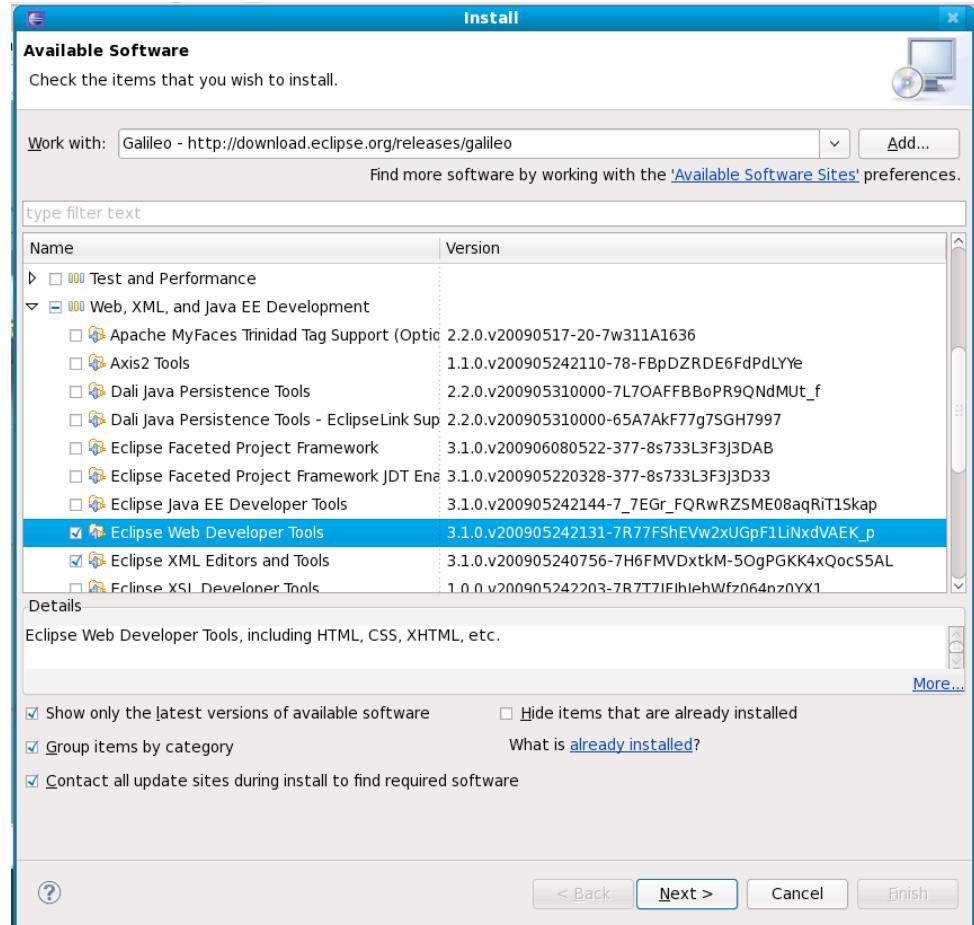
- Install other eclipse tools
- Help > Install New Software..
- No need to do anything now.





Installing extensions

- Linux
Update Mgr
- Manual
Installation:
- dropins/ folder





What is a plug-in?

- An Eclipse plug-in is a unit of eclipse function
- It “extends” the eclipse workbench.
- Has a special type of manifest and (usually) a plugin.xml file
- May depend on other plug-ins
- Can (usually does) contribute to other plug-ins
- Lots of plug-ins already exist (LOTS)
 - <http://www.eclipseplugincentral.com/>
 - <http://eclipse-plugins.info> and on <http://eclipse.org>



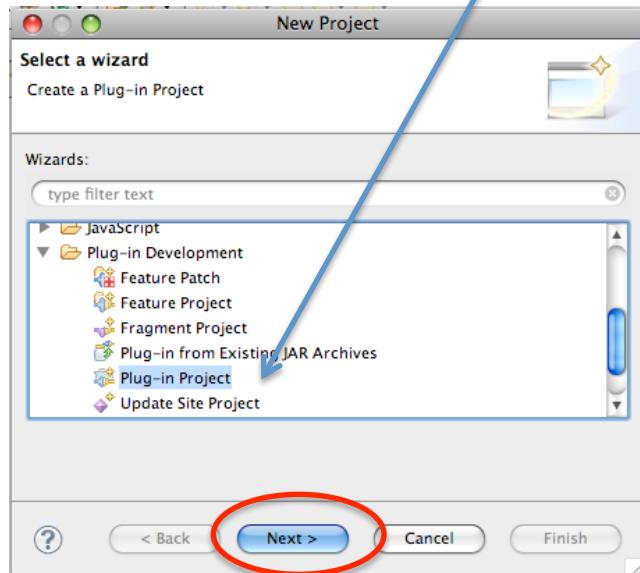
Eclipse examples

- We will develop Eclipse plug-ins in this tutorial via the built-in example wizards
- How much can you learn from “hello world” ?
 - A lot, actually
 - Basic plug-in structure, how to hook in, where to put your code, what your code can do
 - Then we’ll go from there.



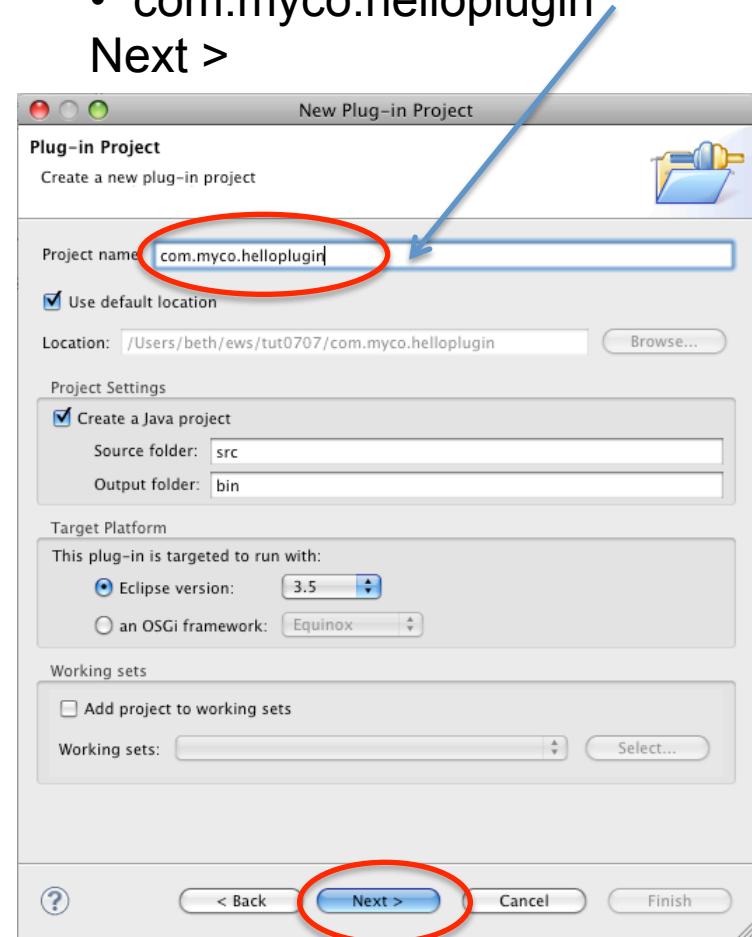
New Plug-in project

- File > New > Project...
- Under Plug-in Development
 - Select Plug-in Project; Next >



Hints and tips:

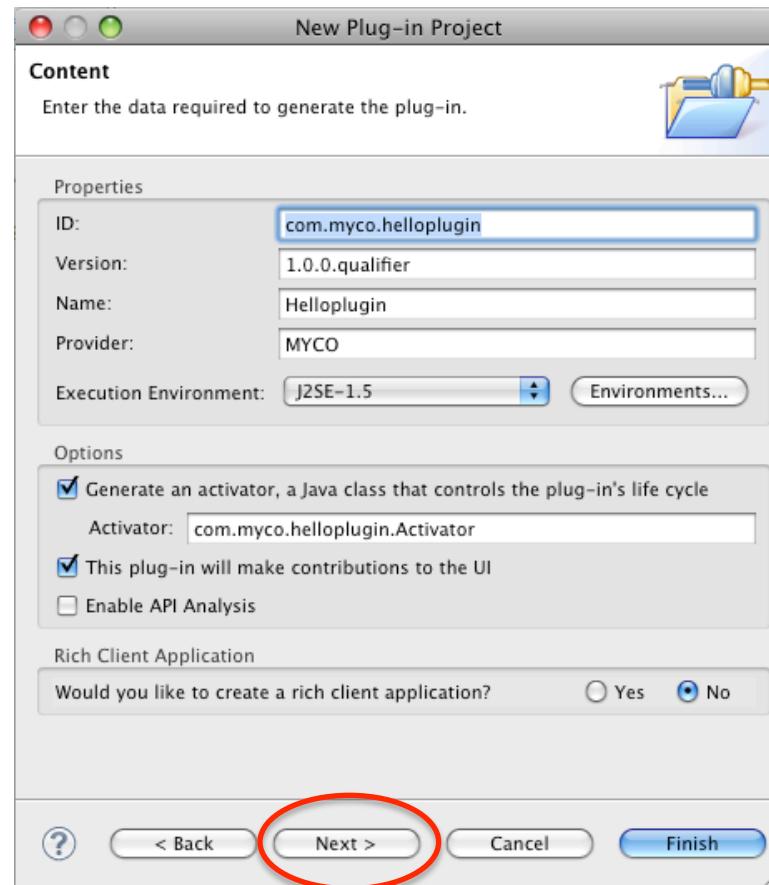
Make the project name the base package name - Helps with linking up IDs (can be difficult to get right later)





New Plug-in project (2)

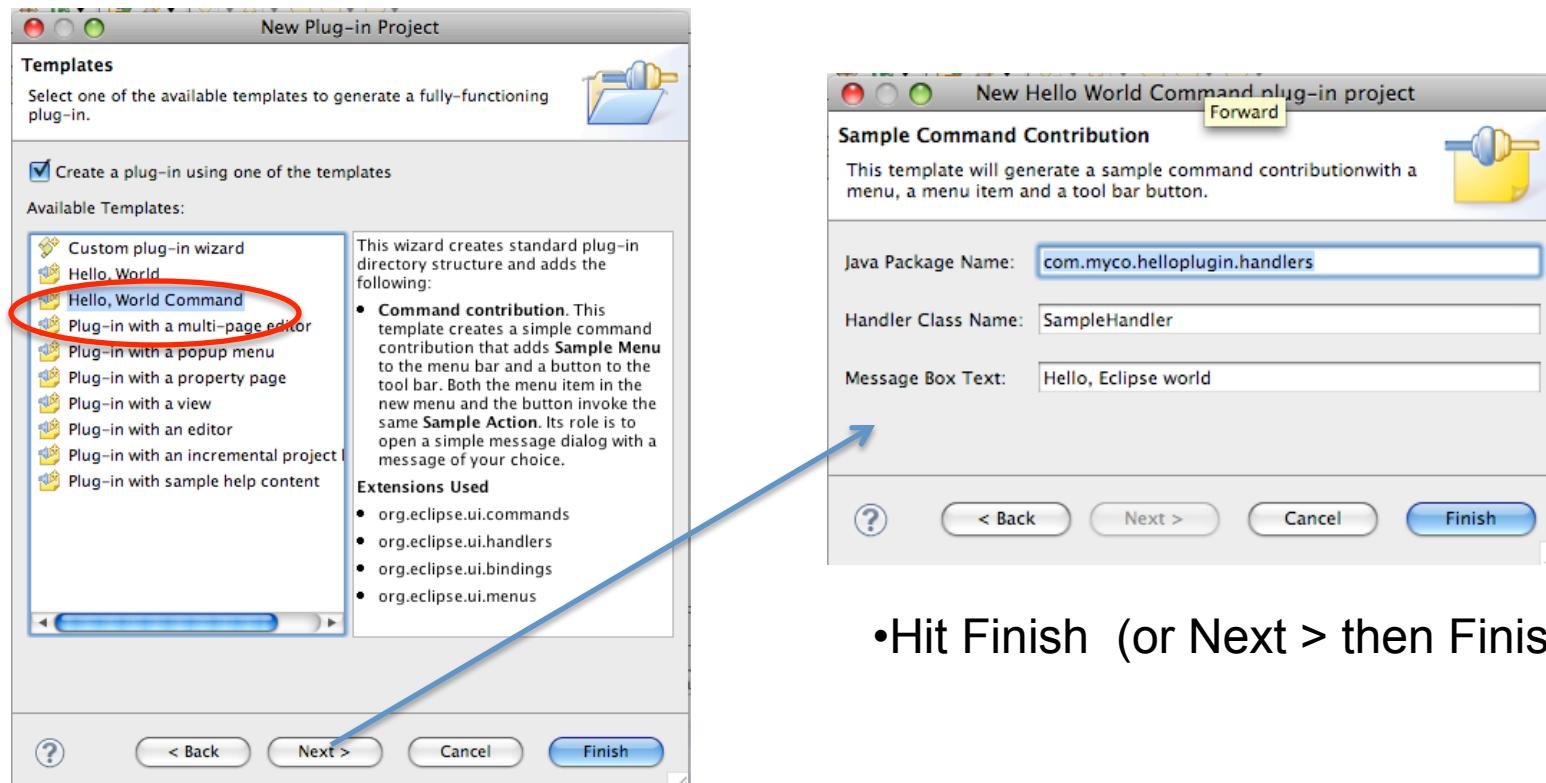
- Content page
- Defaults are OK
- Next >





Plug-in templates

- Choose “Hello, World Command” Template

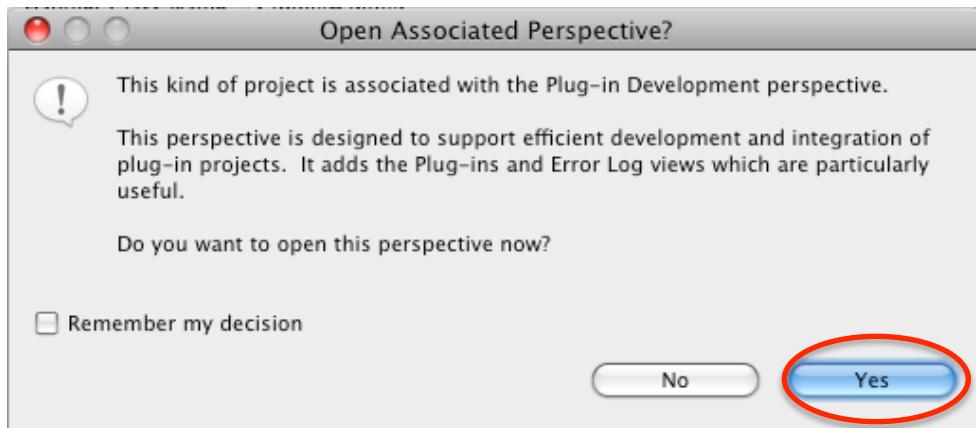


- Hit Finish (or Next > then Finish)



Switch Perspectives

- Accept switch to Plug-in Development Perspective



- If you were already in the Plug-in Development Perspective, you won't see this dialog.



Perspectives

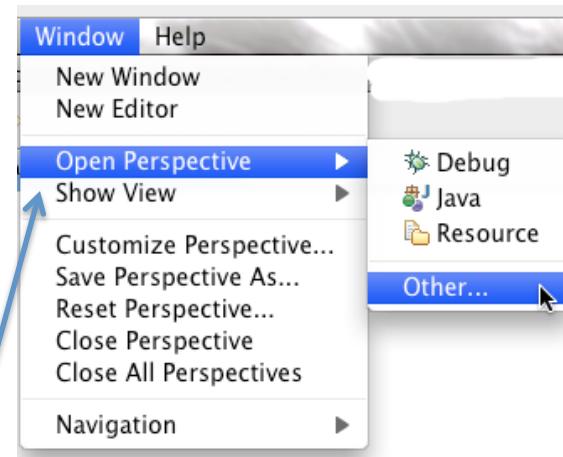
Perspectives provide combinations of views and editors that are suited to performing a particular set of tasks.

Hints and tips:

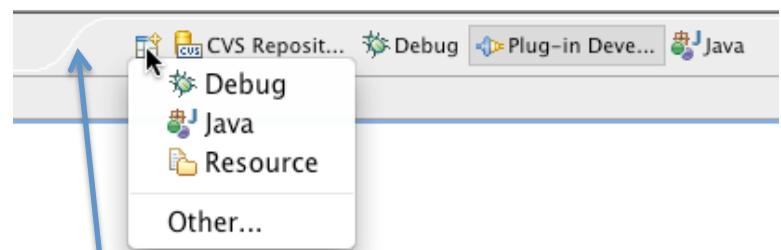
You can easily switch between perspectives via:

1. *Window > Open Perspective, or*
2. *Click the “Open Perspective” button on the shortcut bar,  or*
3. *Click one of the perspective icons in the shortcut bar*

You may need to drag the shortcut bar’s left edge to make it wider, in order to show all the perspective icons that have been active.



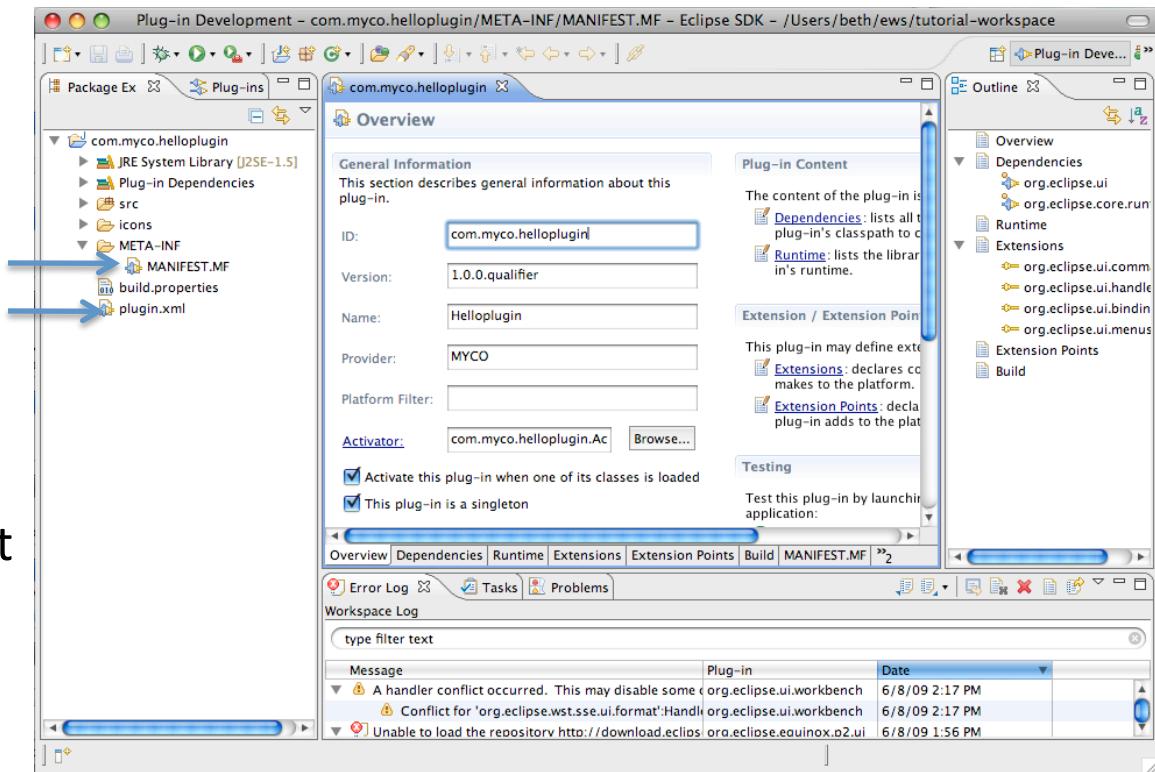
Shortcut bar





Simple Plug-in created

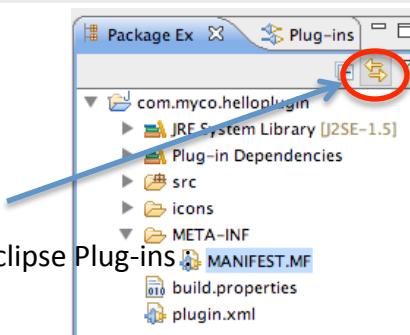
- Edit the plug-in manifest:
MANIFEST.MF
basics
plugin.xml
extensions
- See XML of manifest
- OSGi vs plug-in extensions



Hints and Tips:
To link current editor with location in project:

7/13/09

How to Write Your Own Eclipse Plug-ins
OSCON 09



Link with editor

26



Plug-in source code

- Activator controls plug-in life cycle
- Java code for “Hello world” action
- Double-click file to open in editor

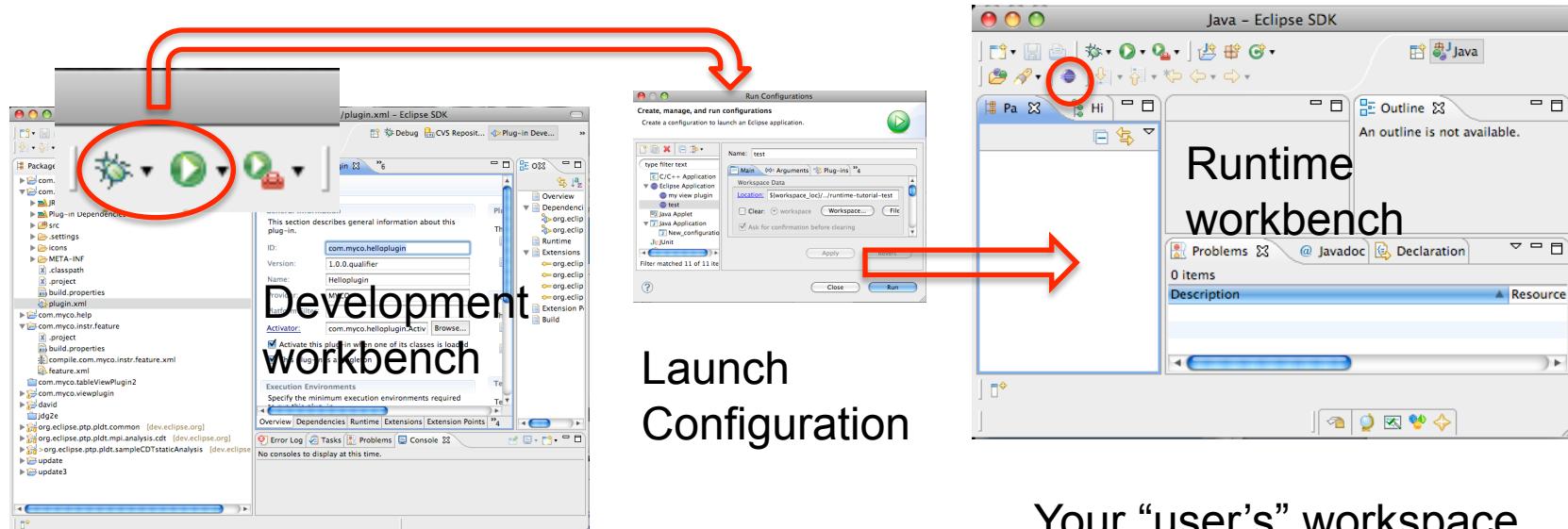
```
public SampleHandler() {  
}  
  
/**  
 * the command has been executed, so extract e  
 * from the application context.  
 */  
public Object execute(ExecutionEvent event) th  
IWorkbenchWindow window = HandlerUtil.get  
MessageDialog.openInformation(  
window.getShell(),  
"Helloplugin",  
"Hello, Eclipse world");  
return null;  
}
```

Enough of a tour. Let's run with our new extension!



Running/Testing plug-ins

- “Self-hosting”- launch another instance of Eclipse
- Runtime workbench launches with (usually) everything in the development workbench, plus the new plug-ins being developed



Your plug-in code is here
In your workspace

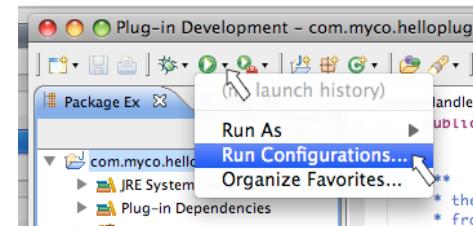
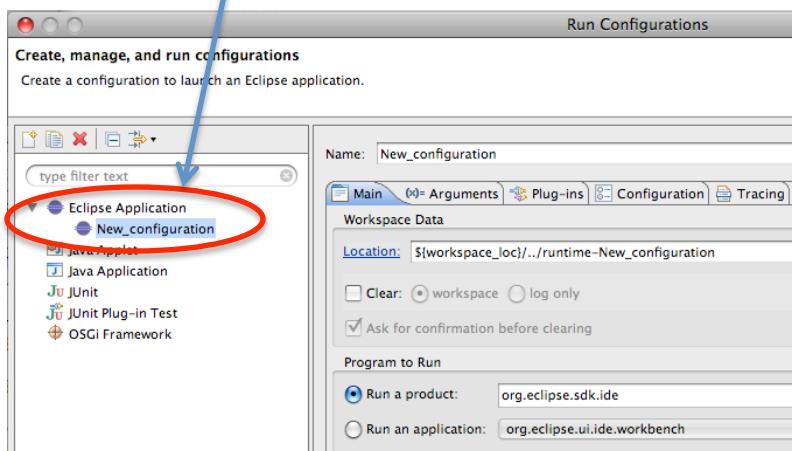
7/13/09

How to Write Your Own Eclipse Plug-ins :
OSCON 09



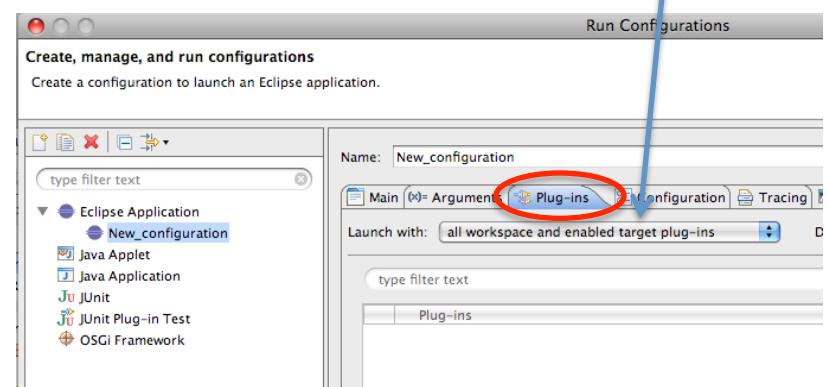
Run

- Run > Run Configurations...
- New Eclipse Application



Or use Run icon menu

On Plug-ins tab
Default selection is what we want:
Launch with all ... plugins



Run!



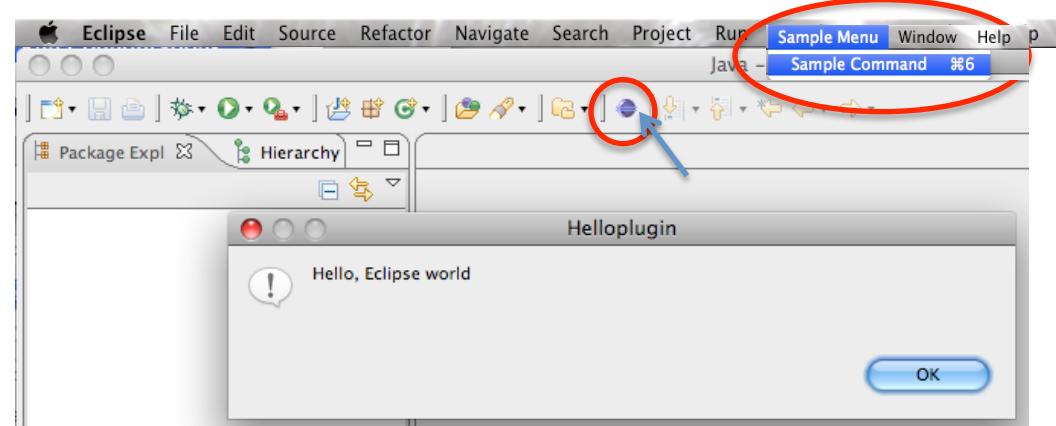


Running your new plug-in

- Another instance of Eclipse is launched
- Go to workbench

Execute:

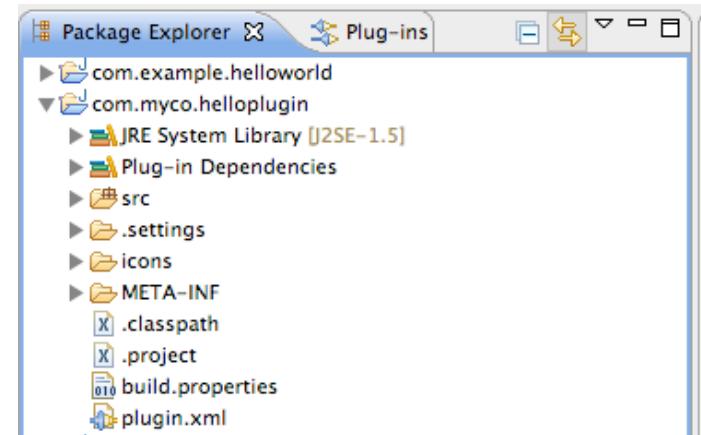
- Toolbar icon
or
Menu





Basic plug-in tour

- Manifest
 - MANIFEST.MF and plugin.xml
- Java code
 - Usually under /src
- Other resources
 - Many are optional
 - Icons, html files, etc.





You're done!

- You have now written your first Eclipse plug-in!
- You're done!
- ... Just kidding!
- Close the runtime workbench



Next project

- Let's make another plug-in that creates an eclipse view
- This will use more of the
 - Eclipse extension points (how eclipse calls *your* code) and
 - APIs (the java code you write that does eclipse stuff in your plug-in)
 - And we'll modify the java code



New project

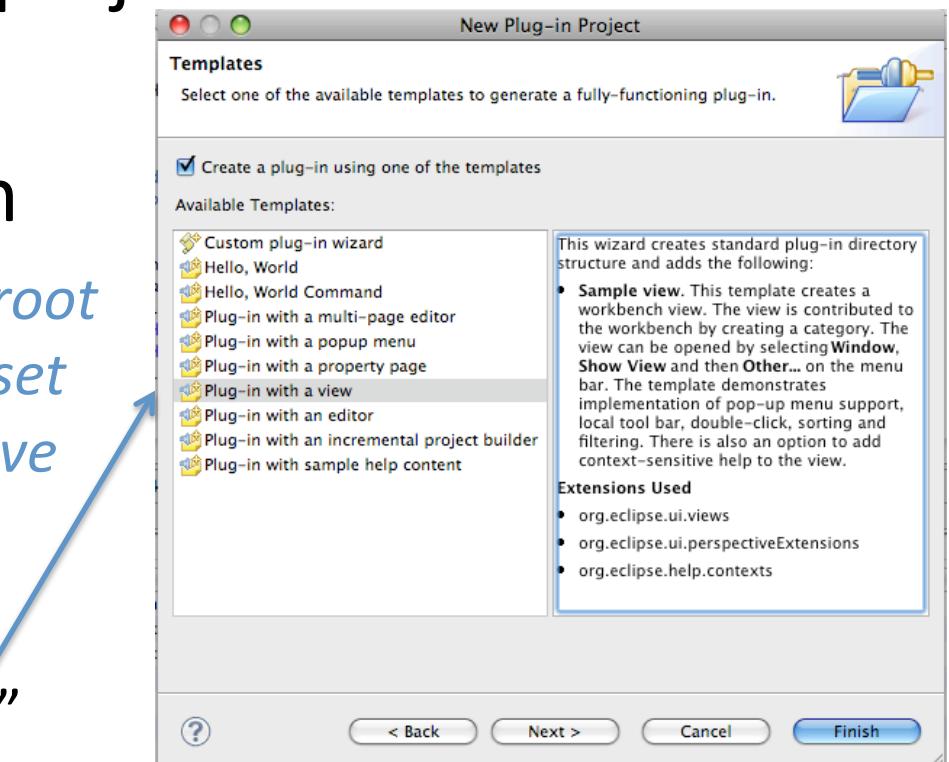
- New Project, Plug-in project

- Name:

com.myco.viewplugin

*— Project name = package root
= plug-in id. Getting this set
up at project start will save
you headaches later.*

- Next, Next
- Choose “Plug-in with a view”
- Next, Finish





Run

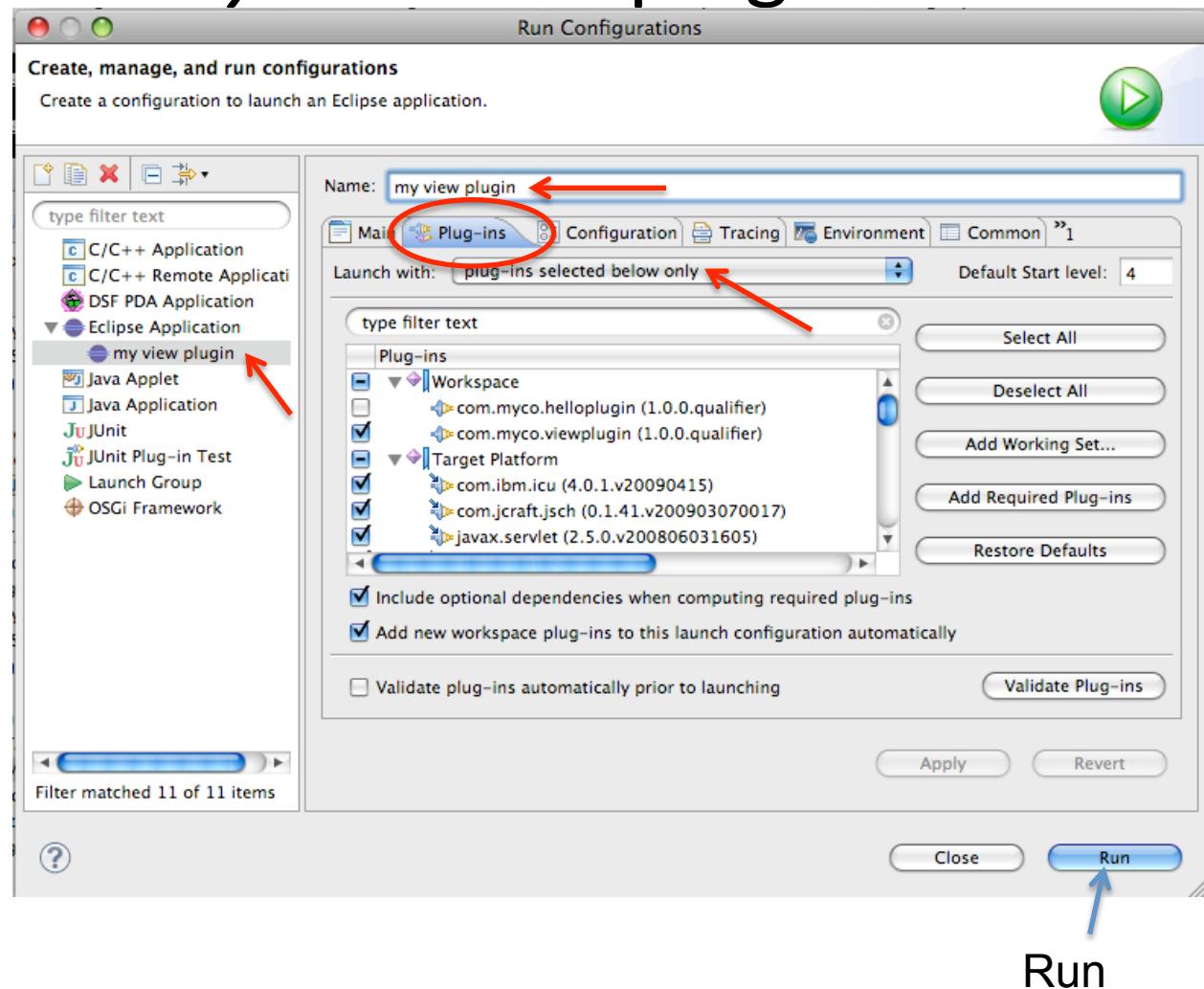
- Run with *just* the new plugin
- Run > Run Configurations
 - New Eclipse application
 - Plugins tag: Launch with: plugins selected...only
 - Under Workspace, select the viewplugin
(See next slide)



Run configuration for launching only the view plug-in

Run with *just* the new plugin

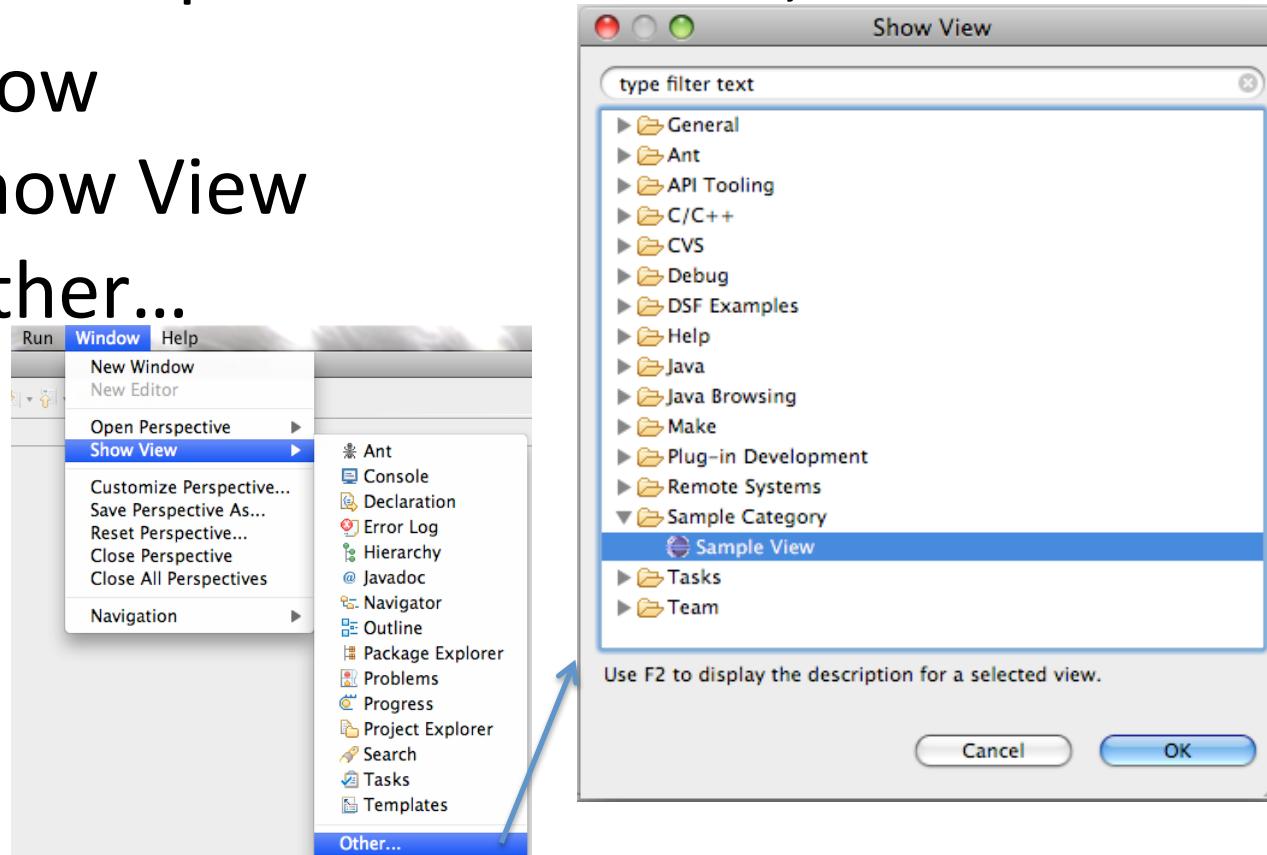
- Run > Run Configurations
- New Eclipse application
- Plugins tab:
 - Launch with: plugins selected...only
 - Under Workspace, select the viewplugin





Bring up New View

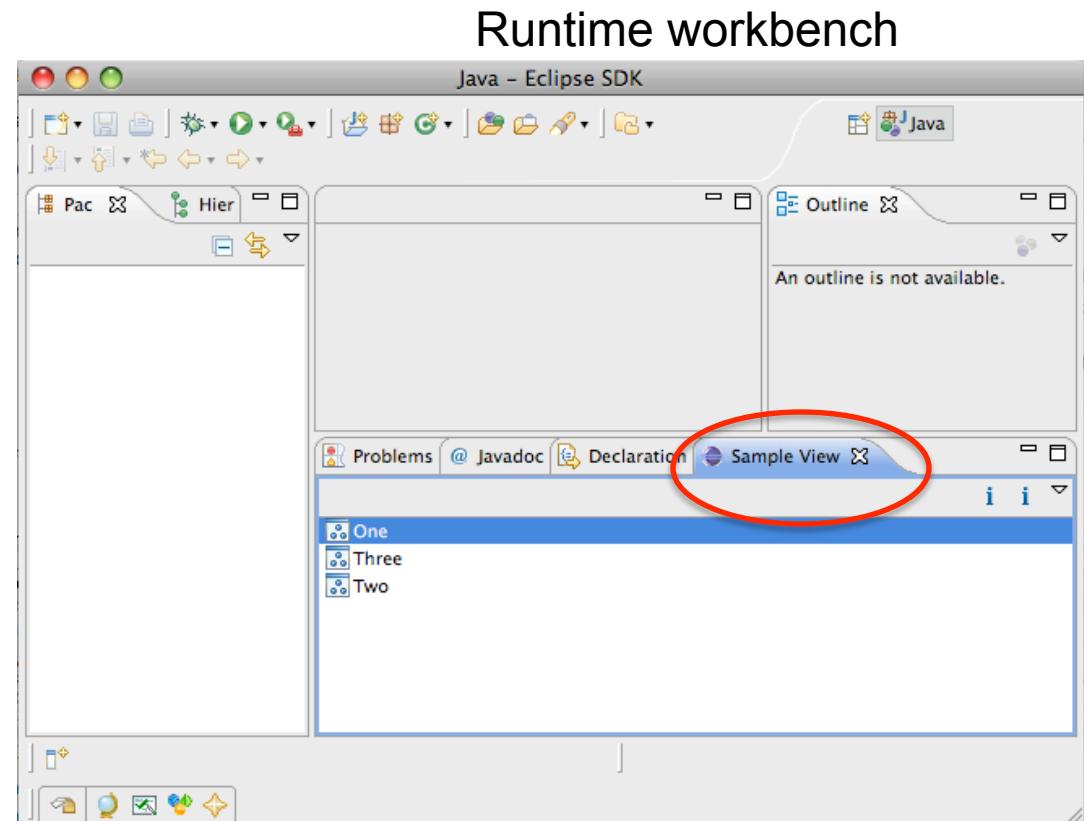
- In new Eclipse that launches,
- Window
 - > Show View
 - > Other...





Our New View

- What functions does it do?
- Click on stuff
- Close the runtime workbench





Look at code

Back in the Development workbench...

- SampleView includes Jface frameworks:
 - ContentProvider: the objects that represent items in the view
 - Label Provider: How to display the objects
 - ViewPart: the subwindow
 - Viewer: the widget for displaying objects
 - TableViewer used here
- *Java code inspection*
 - Hover, hyperlink, ctrl-space (Mac cmd-space)

See next slide



Code for new view

The screenshot shows the Eclipse IDE interface for plugin development. The title bar reads "Plug-in Development - com.myco.viewplugin/src/com/myco/viewplugin/views/SampleView.java - Eclipse SDK". The left side features the "Package Explorer" view, which displays the project structure for two plugins: "com.myco.helloplugin" and "com.myco.viewplugin". A blue arrow points from the "SampleView.java" file in the "com.myco.viewplugin" package towards the center of the screen. The central area contains the Java code for "SampleView.java", which includes imports for Action, IStructuredContentProvider, LabelProvider, and Image. The code defines a ViewContentProvider class that implements IStructuredContentProvider, returning an array of strings ["One", "Two", "Three"] for its elements. It also defines a ViewLabelProvider class that implements ITableLabelProvider, returning the text and image for each element. The right side of the interface shows the "Outline" view, listing the classes and methods defined in "SampleView.java", such as "ViewContentProvider", "ViewLabelProvider", and "SampleView()". Below the code editor are the "Error Log", "Tasks", and "Problems" views, and the "Workspace Log" view at the bottom.

```
private Action action2;
private Action doubleClickAction;

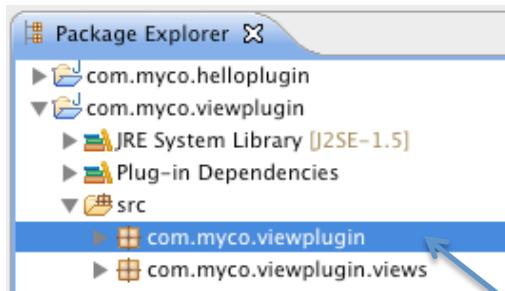
/*
 * The content provider class is responsible for
 * providing objects to the view. It can wrap
 * existing objects in adapters or simply return
 * objects as-is. These objects may be sensitive
 * to the current input of the view, or ignore
 * it and always show the same content
 * (like Task List, for example).
 */

class ViewContentProvider implements IStructuredContentProvider {
    public void inputChanged(Viewer v, Object oldInput, Object newInput) {
    }
    public void dispose() {
    }
    public Object[] getElements(Object parent) {
        return new String[] { "One", "Two", "Three" };
    }
}
class ViewLabelProvider extends LabelProvider implements ITableLabelProvider {
    public String getColumnText(Object obj, int index) {
        return getText(obj);
    }
    public Image getColumnImage(Object obj, int index) {
        return getImage(obj);
    }
    public Image getImage(Object obj) {
        return PlatformUI.getWorkbench().getSharedImages().getImage(
            ImageDescriptor.fromImage(
                new Image(
                    getClass().getResource("icons/icon.gif"))));
    }
}
```



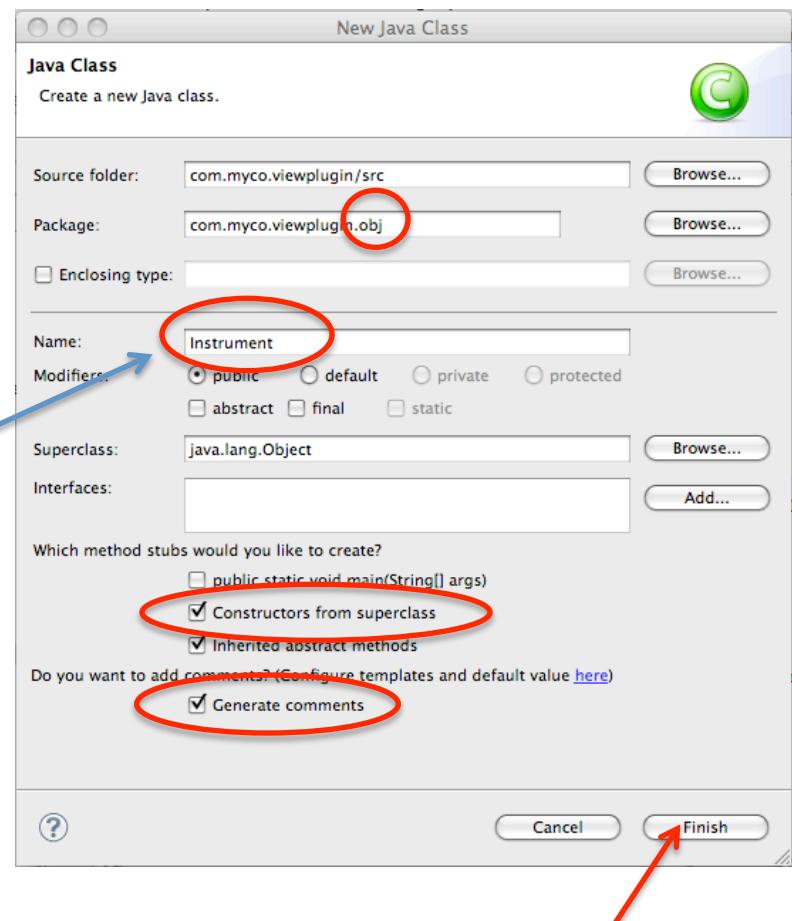
New Class

- In Package Explorer under the new project, Select package “com.myco.viewplugin”



(This makes the new class wizard automatically fill in some values)

- File, New, Class
 - Name: Instrument
 - Package: com.myco.viewplugin.obj
 - Generate comments
 - Constructors from superclass
 - Finish

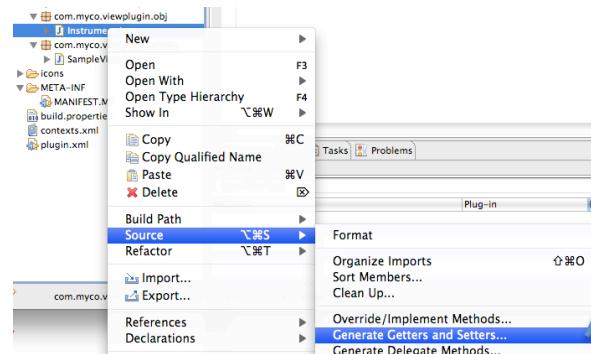




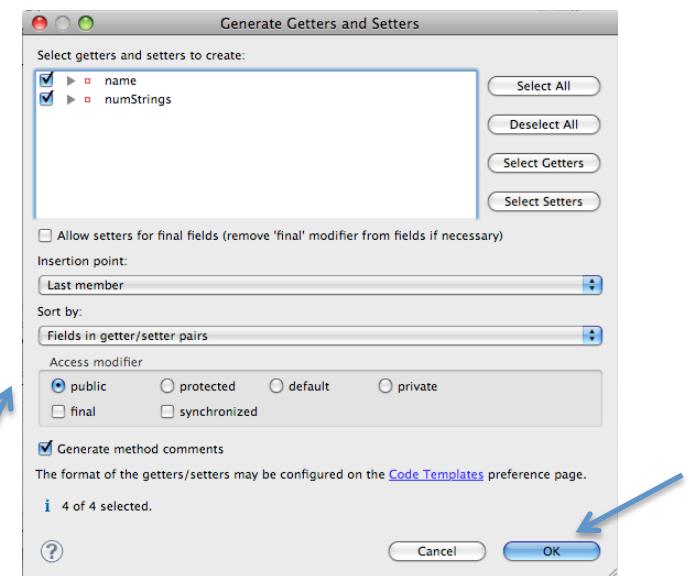
Edit code for Instrument

- Add name, numStrings
 - Add getters/setters:
 - Select ‘Instrument.java’ file
 - Context menu:
 - Source >
- Generate getters and setters

- Select all
- Insertion point last member
- Generate method comments; OK



```
public class Instrument {  
    private String name;  
    private int numStrings;
```



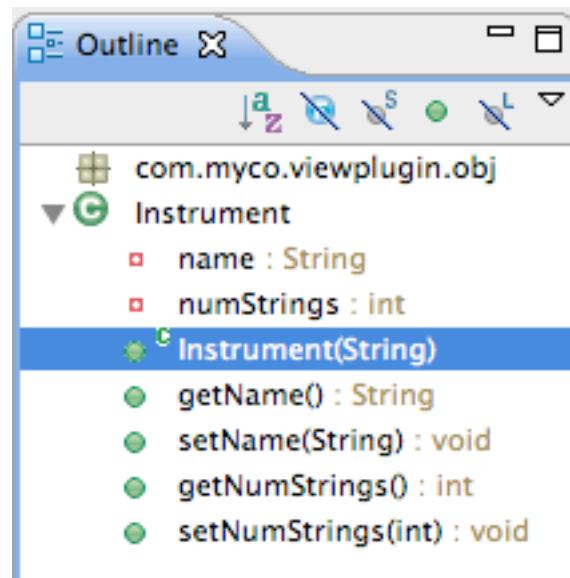


Edit code for Instrument

Add String arg to constructor; set name

```
public Instrument(String name) {  
    this.name=name;  
}
```

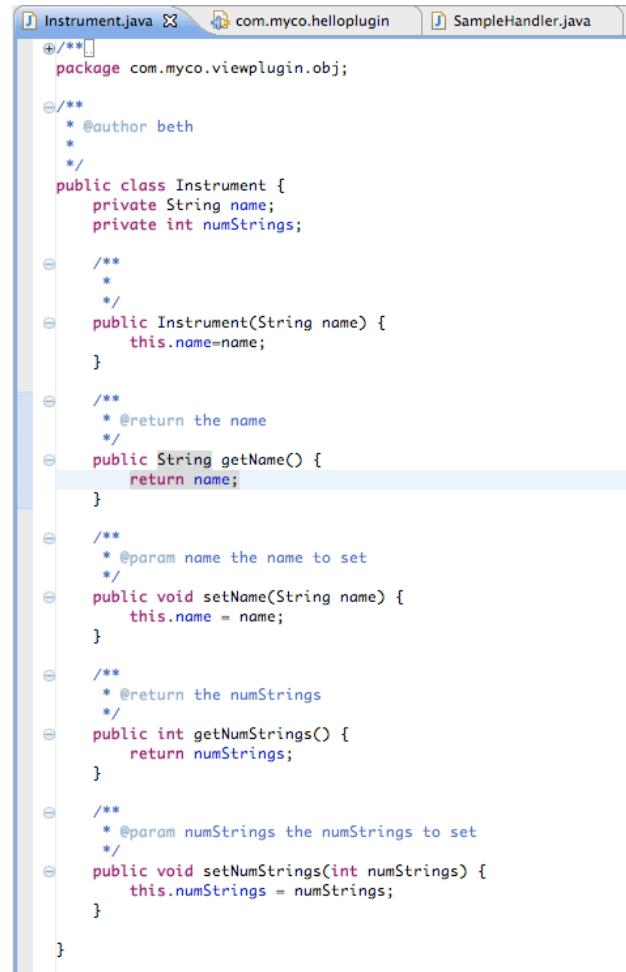
- Save file
- The outline so far





Instrument.java so far

```
// w/o comments:  
package com.myco.viewplugin.obj;  
  
public class Instrument {  
    private String name;  
    private int numStrings;  
    public Instrument(String name) {  
        this.name=name;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getNumStrings() {  
        return numStrings;  
    }  
    public void setNumStrings(int numStrings) {  
        this.numStrings = numStrings;  
    }  
}
```

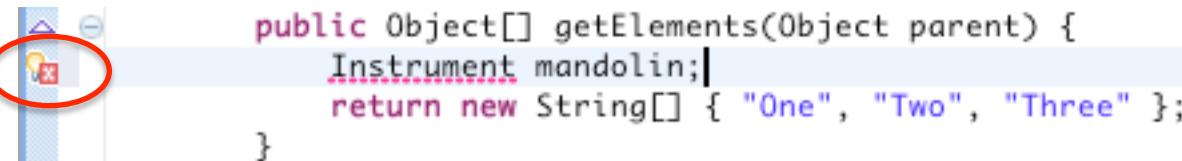


The screenshot shows the Eclipse IDE interface with the 'Instrument.java' file open in the center editor tab. The code is identical to the one shown in the previous text block. The Eclipse interface includes a top bar with tabs for 'Instrument.java', 'com.myco.helloplugin', and 'SampleHandler.java'. On the left, there's a package explorer view showing the 'Instrument' class under 'com.myco.viewplugin.obj'. The code is color-coded with syntax highlighting for keywords like 'public', 'private', and 'String'.



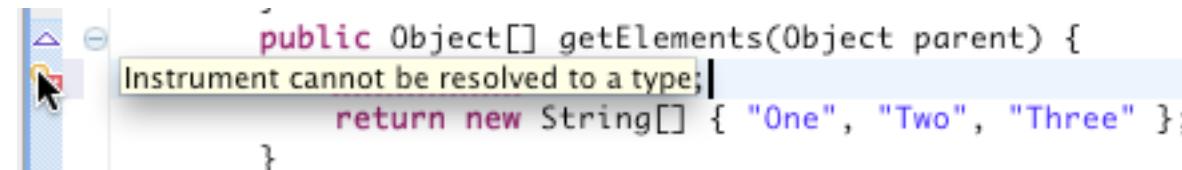
Use new objects in ContentProvider

- Edit SampleView.java (it's under com.myco.viewplugin.views package)
- In ViewContentProvider modify getElements() add 'Instrument mandolin;'



```
public Object[] getElements(Object parent) {
    Instrument mandolin;
    return new String[] { "One", "Two", "Three" };
}
```

- Hover over error marker (red X) to see “Instrument cannot be resolved to a type”



```
public Object[] getElements(Object parent) {
    Instrument cannot be resolved to a type;
    return new String[] { "One", "Two", "Three" };
}
```

- The code needs a import statement to resolve it.
- See next slide on how to get Eclipse to fix it for you!



Use Quick Fix - to fix import

- Problem: “Instrument cannot be resolved to a type”
 - Solution: need to add an import statement for the new class in different package
- Single click on Light Bulb / Problem marker – or
Use context menu (right mouse): Quick Fix – or –
(Windows Ctrl-1) (Mac: Command-1) to see suggestions:

The screenshot shows an Eclipse code editor window. A red circle highlights the 'x' icon in the left margin, indicating a problem. The cursor is over the word 'Instrument' in the code, which has turned blue, signifying it's a reference. A context menu is open at this position, listing several options:

- Import 'Instrument' (com.myco.viewplugin.obj)
- Import 'Instrument' (javax.sound.midi)
- Create class 'Instrument'
- Create interface 'Instrument'

Below the code editor, the status bar displays the text 'wpl Last Edit Location (^Q)'.

```
public Object[] getElements(Object parent) {  
    Instrument mandolin;  
}  
}  
class Vi  
publ  
import org.eclipse.swt.SWT;  
import com.myco.viewplugin.obj.Instrument;  
...  
import com.myco.viewplugin.obj.Instrument;
```

- Adds import to top of file:
- *Scroll up to see it then, to go back:*



Complete getElements()

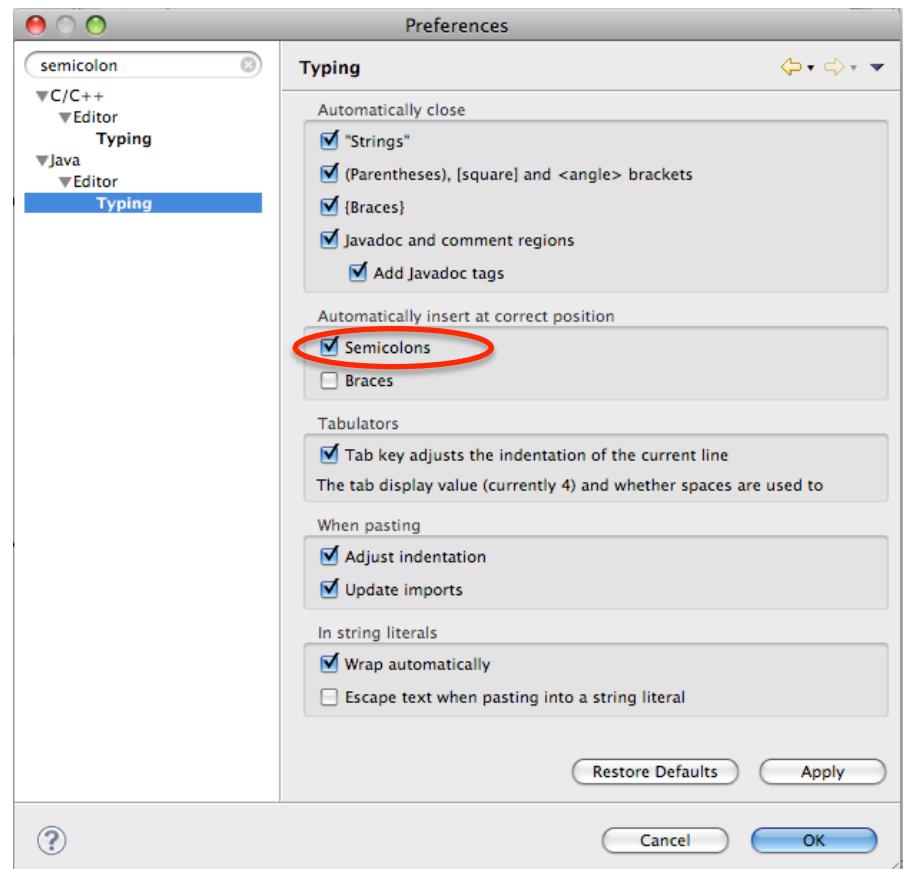
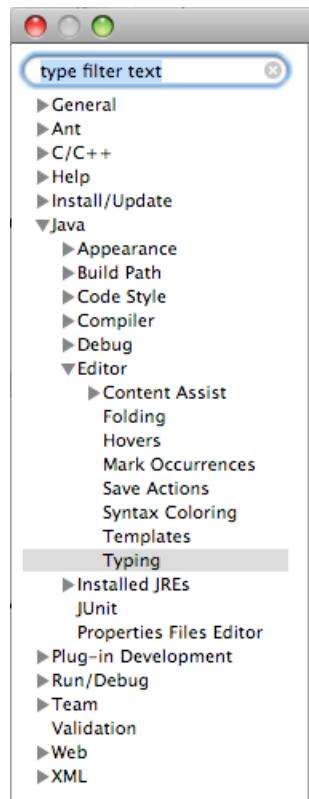
- Use completion/content assist, templates, etc to help with coding
- Change Preferences to automatically position semicolon (see next slide)

```
public Object[] getElements(Object parent) {  
    Instrument mandolin = new Instrument("mandolin");  
    Instrument fiddle = new Instrument("fiddle");  
    Instrument banjo = new Instrument("banjo");  
    return new Instrument[] { mandolin, fiddle, banjo };  
}
```



Preferences

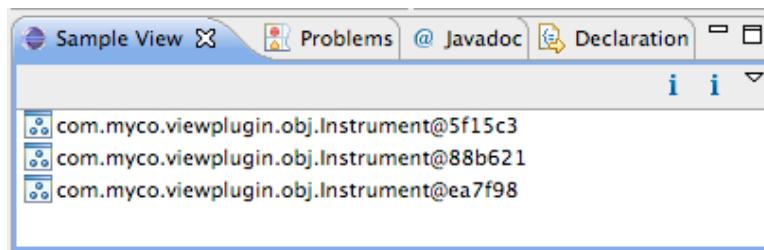
- Eclipse Preferences allow a *lot* of customization





Modify the Table View

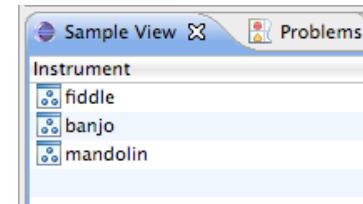
- Still in com.myco.viewplugin
- Launch again. Hmm ... not a very interesting view



- Modify the LabelProvider to provide the Instrument name

```
class ViewLabelProvider extends LabelProvider implements ITableLabelProvider {  
    public String getColumnText(Object obj, int index) {  
        return ((Instrument)obj).getName();  
    }  
}
```

- Relaunch; see nice names



- Now let's add columns to the table

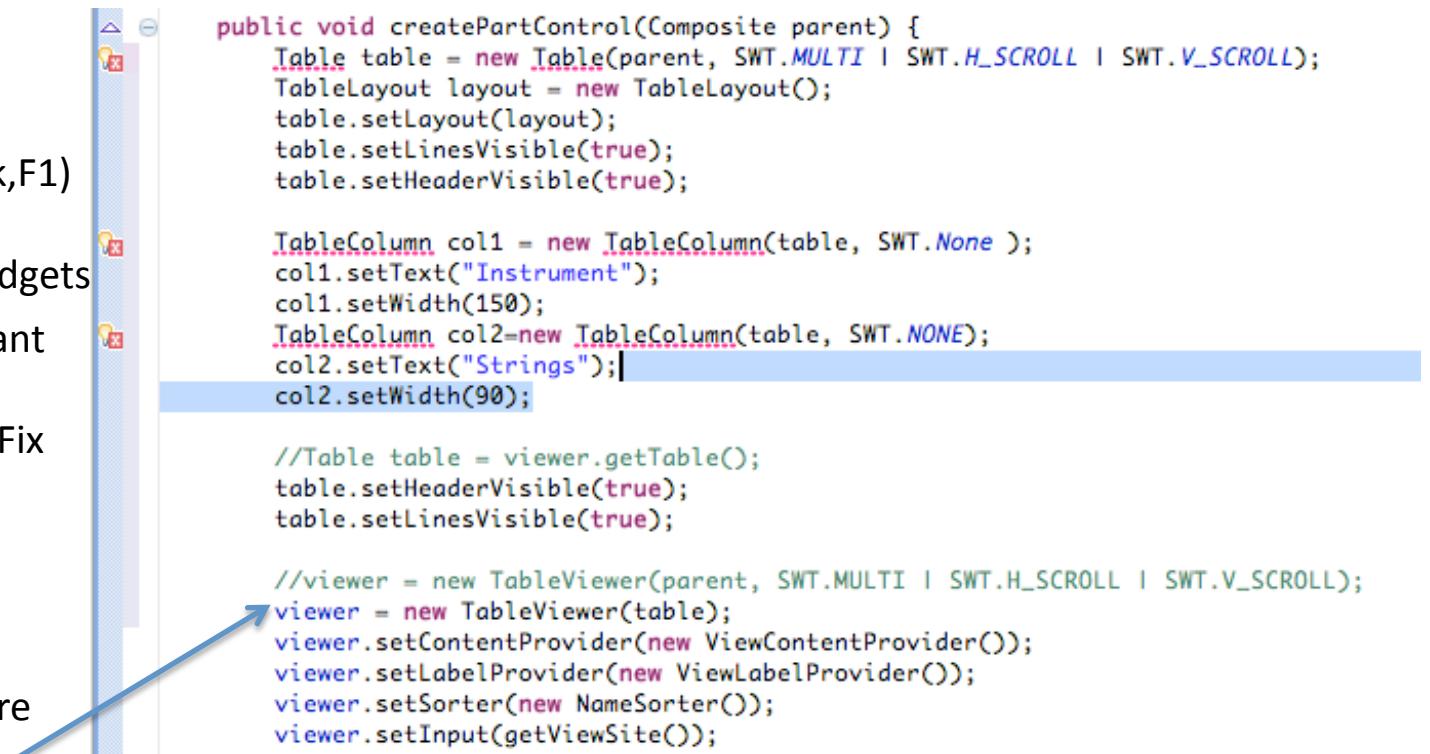


Modify the Table View

Add columns to the table

- Add to top of `createPartControl()`:

- Use QuickFix (click F1) to fix imports:
`org.eclipse.swt.widgets`
- Note you don't want the `javax.swing` classes that QuickFix will also suggest!



```
public void createPartControl(Composite parent) {
    Table table = new Table(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);
    TableLayout layout = new TableLayout();
    table.setLayout(layout);
    table.setLinesVisible(true);
    table.setHeaderVisible(true);

    TableColumn col1 = new TableColumn(table, SWT.None );
    col1.setText("Instrument");
    col1.setWidth(150);
    TableColumn col2=new TableColumn(table, SWT.NONE);
    col2.setText("Strings");
    col2.setWidth(90);

    //Table table = viewer.getTable();
    table.setHeaderVisible(true);
    table.setLinesVisible(true);

    //viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);
    viewer = new TableViewer(table);
    viewer.setContentProvider(new ViewContentProvider());
    viewer.setLabelProvider(new ViewLabelProvider());
    viewer.setSorter(new NameSorter());
    viewer.setInput(getViewSite());
```

New code down to here



Have columns, but...

- There are now two columns

A screenshot of the Eclipse interface showing the "Sample View" window. The window title is "Sample View". Below it is a table with two columns: "Instrument" and "Strings". The data rows are:

Instrument	Strings
fiddle	fiddle
banjo	banjo
mandolin	mandolin

- Fix `getColumnText()` to return label *by column*

A screenshot of the Eclipse IDE showing the Java code for the `ViewLabelProvider` class. The code implements the `ITableLabelProvider` interface and overrides the `getColumnText` method. The code uses a switch statement to determine the string to return based on the index of the instrument.

```
class ViewLabelProvider extends LabelProvider implements ITableLabelProvider {
    public String getColumnText(Object obj, int index) {
        Instrument instr = (Instrument) obj;
        switch (index) {
        case 0:
            return instr.getName();
        case 1:
            int num= instr.getNumStrings();
            return Integer.toString(num);
        default:
            break;
        }
        return instr.getName();
    }
}
```



Make columns nicer

- Also change getColumnImage to only return image for first column

```
public Image getColumnImage(Object obj, int index) {  
    if (index == 0) return getImage(obj);  
    else return null;  
}
```

- Launch again; we now have:

Instrument	Strings
banjo	0
fiddle	0
mandolin	0



More if time permits

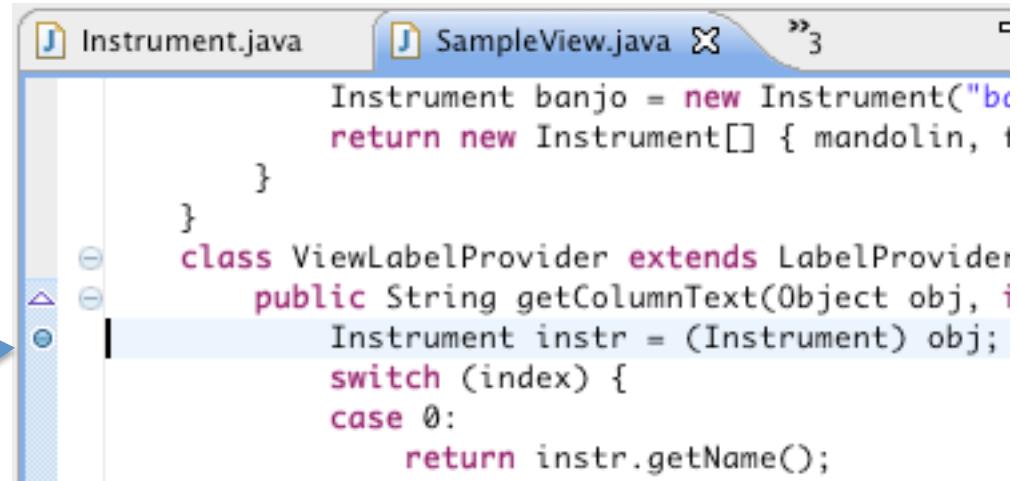
- Add number of strings to Instrument objects
- Add selection listener to col 1: open in debugger (Click on col heading to invoke/get exception. What to catch? Surround with try/catch. ClassCatchException! Show code templates that automatically appear for the required interfaces.)
- Close runtime workbench; Edit manifest, change view name and category name; relaunch
- Optional Topic: hook up to model to receive changes



Debugging

- Set breakpoint:

Double-click
at left edge



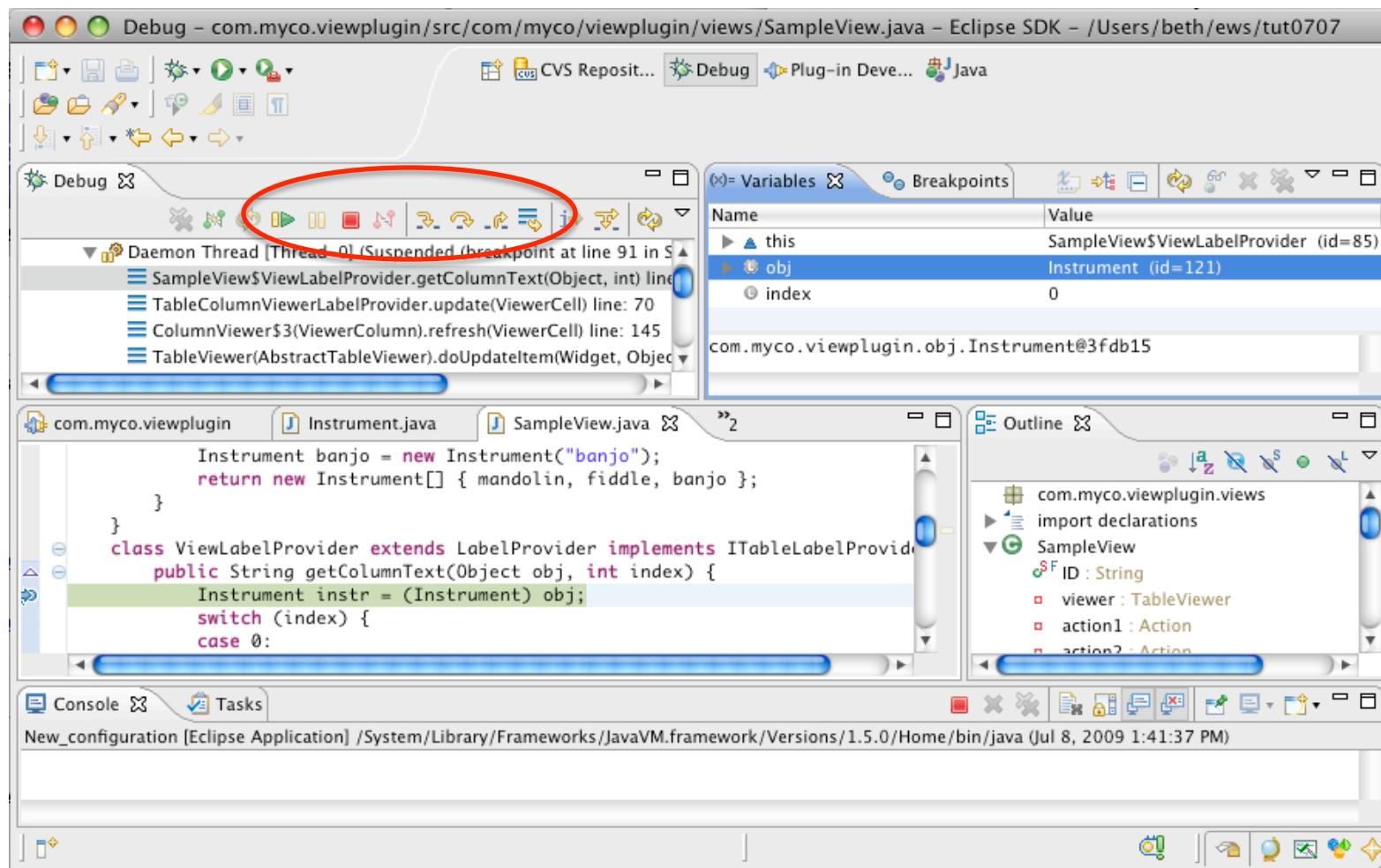
```
Instrument banjo = new Instrument("banjo");
        return new Instrument[] { mandolin, banjo };
    }
}
class ViewLabelProvider extends LabelProvider {
    public String getColumnText(Object obj, int index) {
        Instrument instr = (Instrument) obj;
        switch (index) {
        case 0:
            return instr.getName();
        
```

- Launch Eclipse debugger
- Run and step thru code





Eclipse debugger





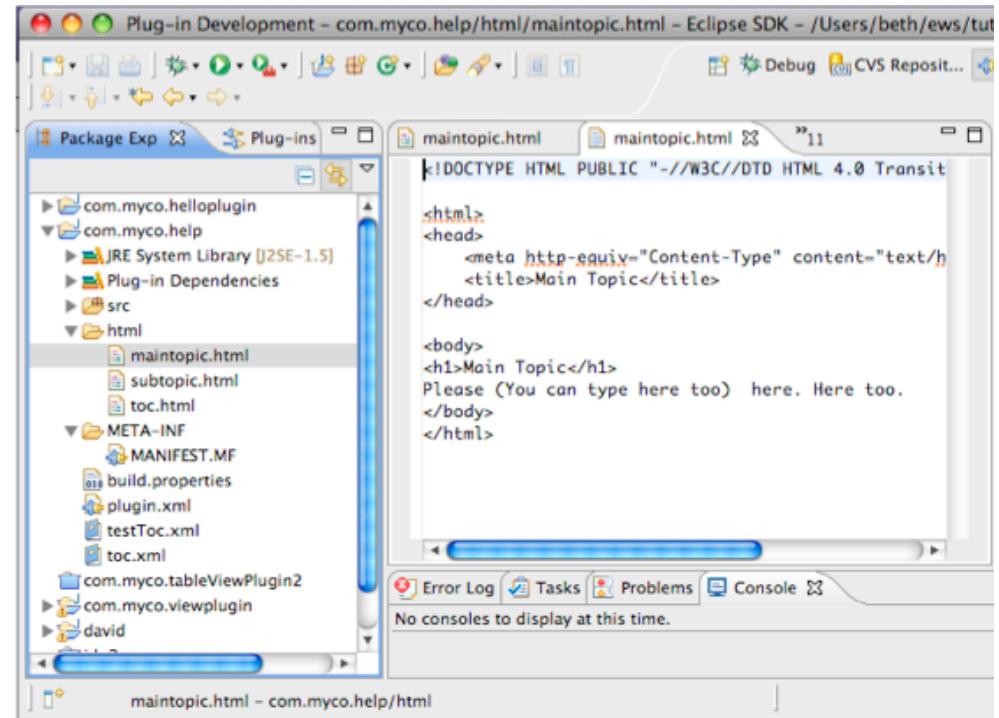
A Help Plug-in

- First, bring up Eclipse help to see what it does
 - Help > Help Contents
- For more info: See help, Platform Plug-in developer guide
>programmer's guide > user assistance support > Help
- Let's make a help plug-in project, and use the wizard as an example



Help Plugin Project

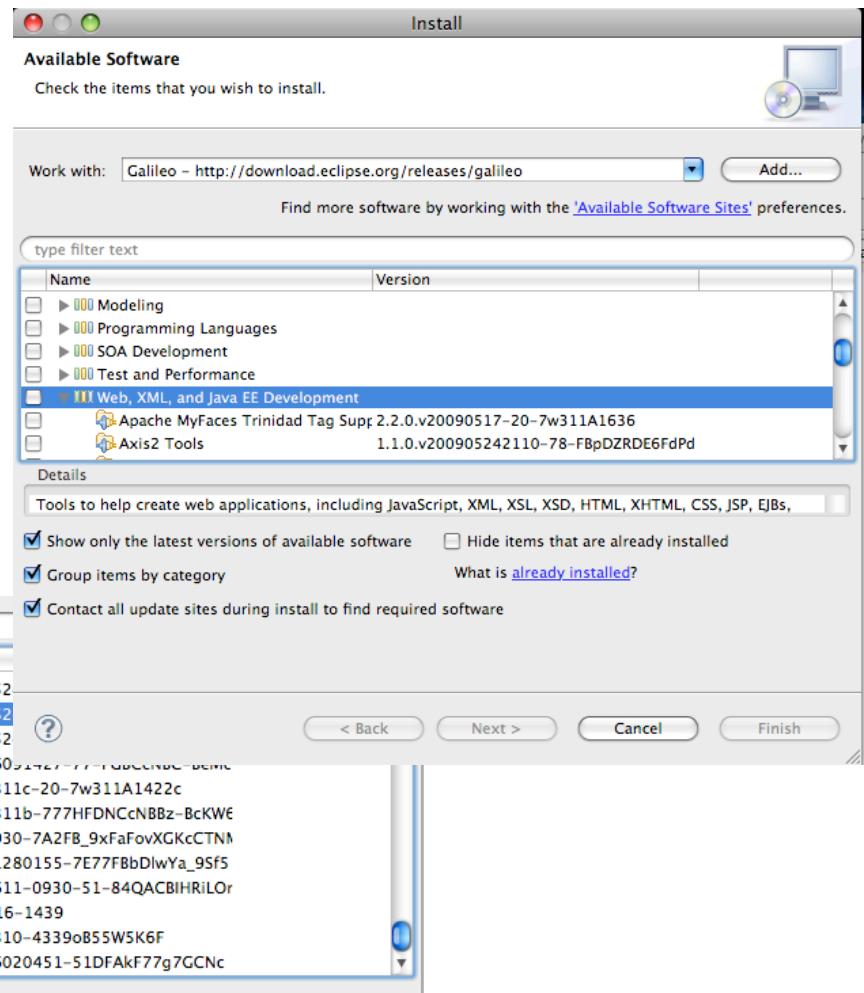
- New plugin project
- Name: com.myco.help,
Next >, Next >
- Plug-in with sample help content;
Finish
- Project is created; manifest opens
- Peruse project contents:
html, xml, etc.





Install XML/HTML Editors

- We need html/xml editor: Install them!
 - Help >Install new software
 - Work with: Galileo site
 - Under Web, XML, and Java EE Dev...
 - Choose “Eclipse XML Editors and Tools” and
 - “Web Page Editor”
 - Next >, Next >
Accept, Finish
Restart





Editing HTML

The screenshot illustrates the Eclipse IDE environment for plugin development, specifically focusing on HTML editing. It shows two main perspectives:

- Left Perspective:** Displays the Package Explorer and Plug-ins view. A file named "maintopic.html" is selected in the Package Explorer. The main editor area shows the following HTML code:

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Main Topic</title>
</head>
<body>
    <h1>Main Topic</h1>
    Please enter your text here.
</body>
</html>
```
- Right Perspective:** Shows a 2-pane view for "maintopic.html". The left pane is the "Design" view, displaying the HTML structure and a placeholder text "Please (You can type here too) here too.". The right pane is the "Preview" view, showing the rendered HTML output with the placeholder text displayed.

A blue arrow points from the text "HTML Editor" to the main editor area. Another blue arrow points from the text "Web page editor" to the 2-pane preview view. A third blue arrow points from the text "Select file, rightMouse, Open With..." to a context menu for the "maintopic.html" file in the Package Explorer. The context menu is open, showing options like "New", "Open", "Open With", "Show In", "Copy", etc. The "Open With" option is highlighted with a blue arrow.

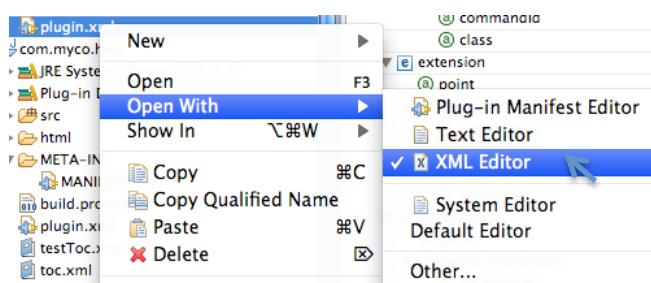
- HTML Editor:
syntax highlighting, outline, completion, etc.
- Web page editor:
• 2-pane with wysiwyg

Select file, rightMouse, Open With...

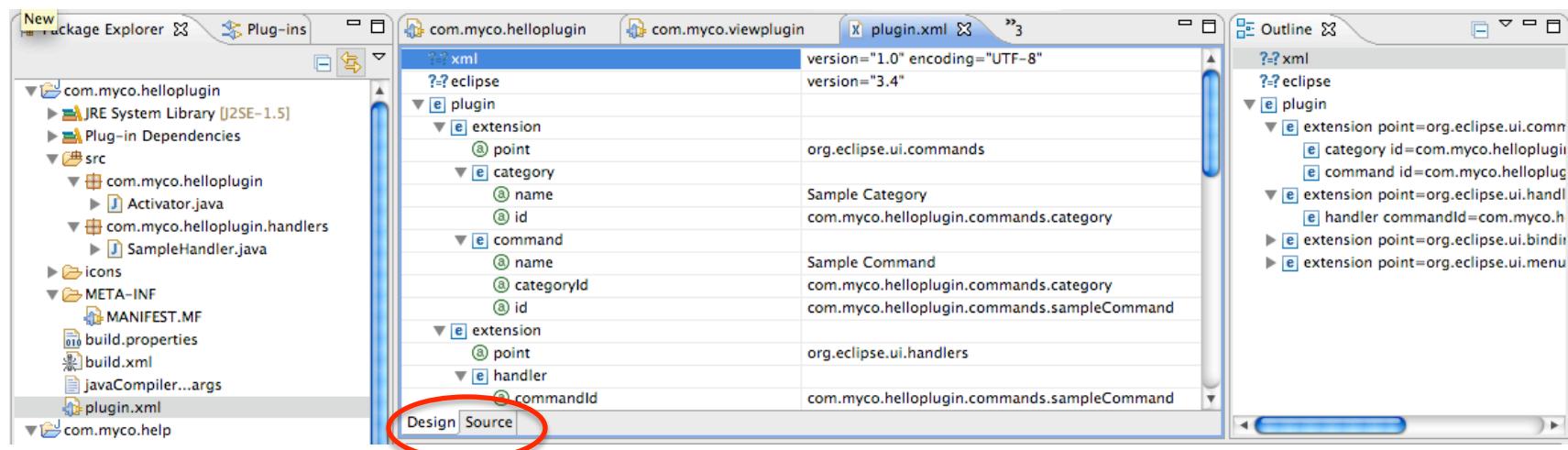


Editing XML

- Open With > XML Editor – Design and Source tabs



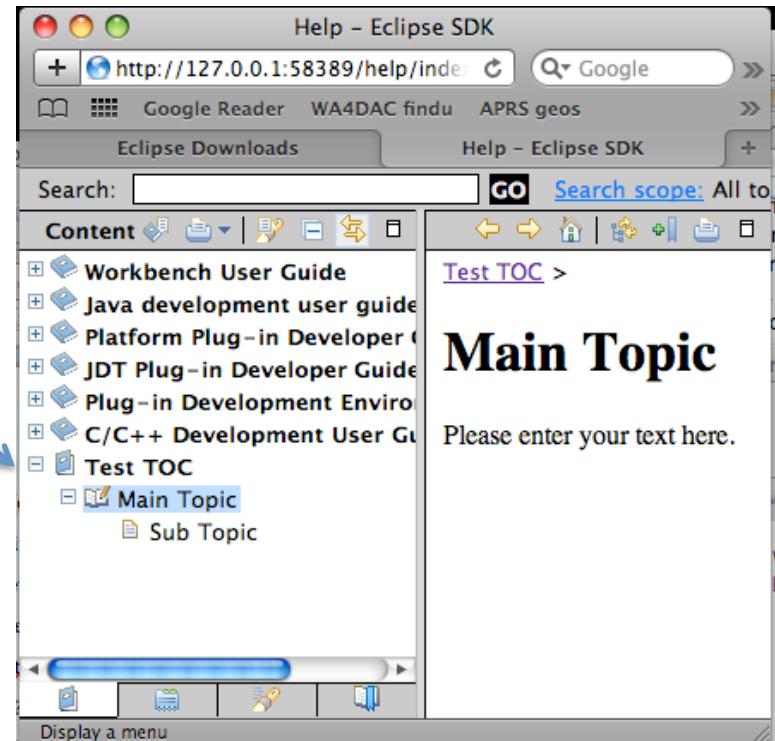
You don't really need the XML editor for editing Eclipse manifests, however. There are special editors for these.





Run the Help Plug-in

- Launch the runtime workbench -- as before
- Make sure the new plug-in is included in the plug-ins included in the launch configuration
(it probably is)
- In runtime workbench:
 - Help > Help Contents
 - See your new content!
- Other components of Eclipse “Help”
 - Cheatsheets – walk user thru steps
 - “Welcome” page content
 - ...etc





Packaging

- You want others to be able to install your plug-in(s)
- Steps:
 1. Create a feature to list the plug-ins
 2. Create an update site project with your feature(s)
 3. Build it, test the install, and distribute
 - Zip archive, or
 - On web site



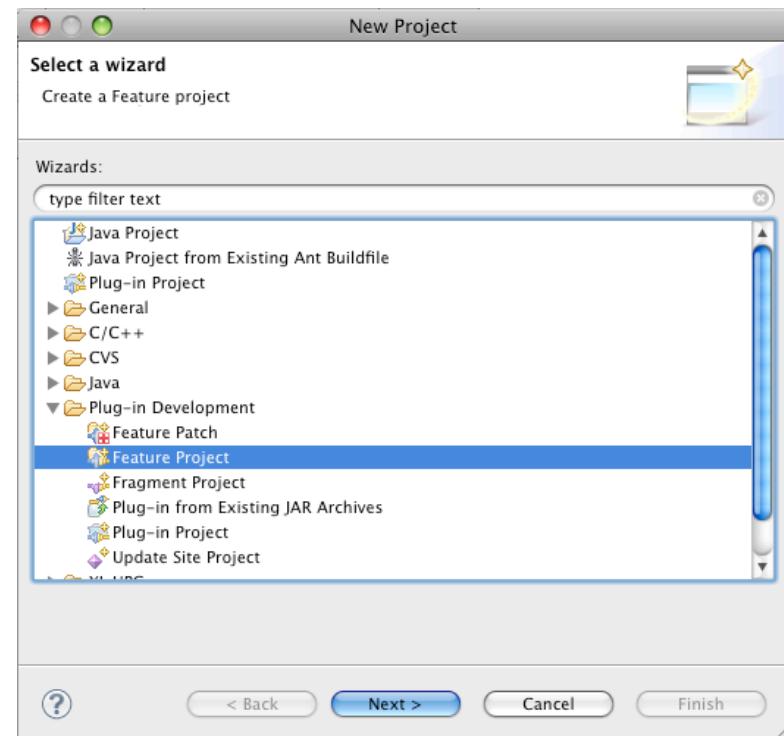
Create a feature project

- An Eclipse *feature* encapsulates one or more plug-ins. It's the unit which is *installed* by users.
- File > new Project,
Plug-in Development >
Feature Project

Next >

- Project name: e.g.
com.myco.thing.feature

Next >



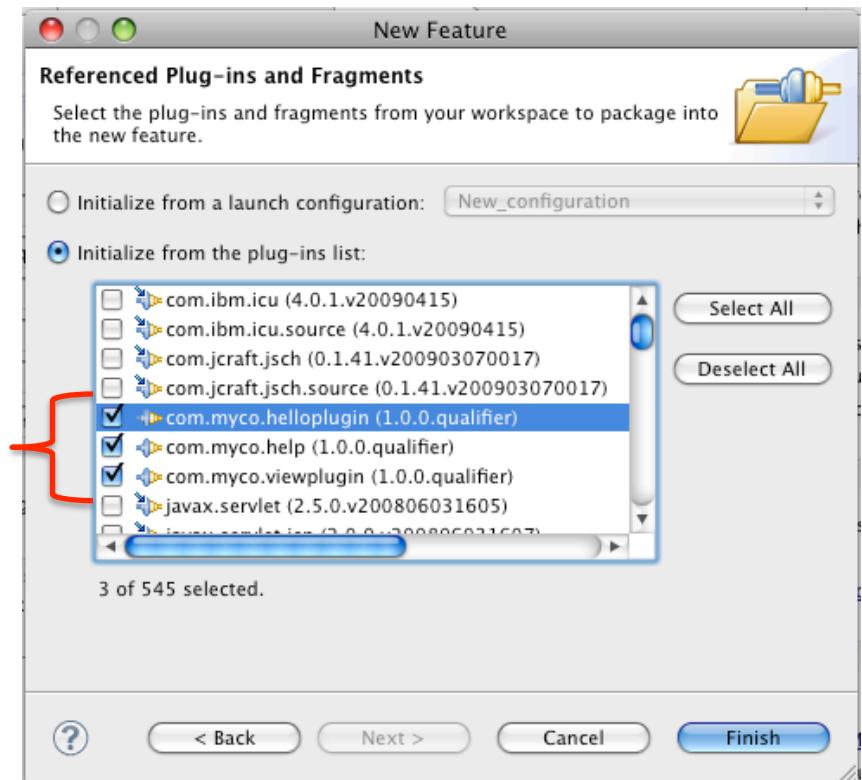


Add plugins to feature

- **Add plug-ins**

Check all the plug-ins you made

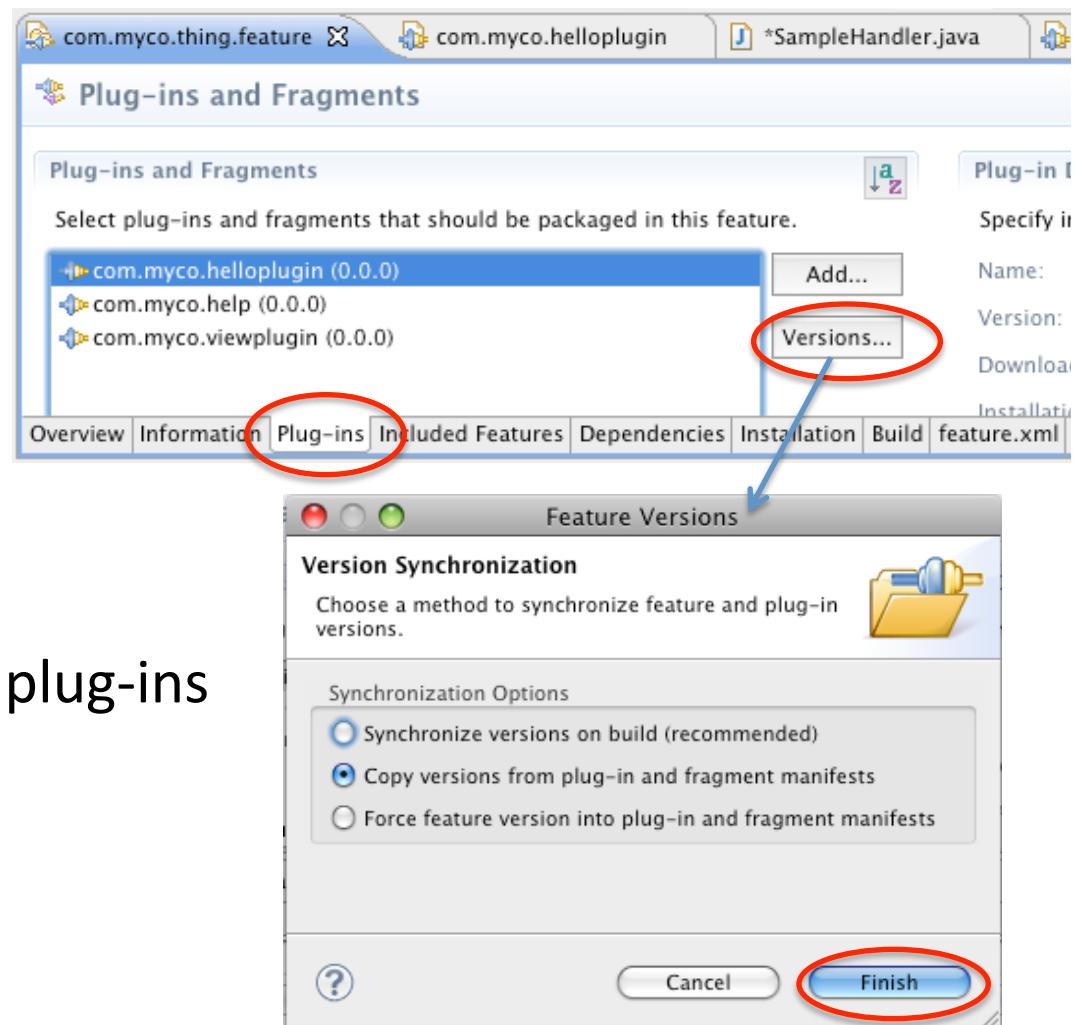
- **Finish**





Feature versions

- You are editing the feature manifest, feature.xml
- On the Plug-ins tab:
- Versions...
 - Copy versions from plug-ins
 - Finish
- Save the feature.xml





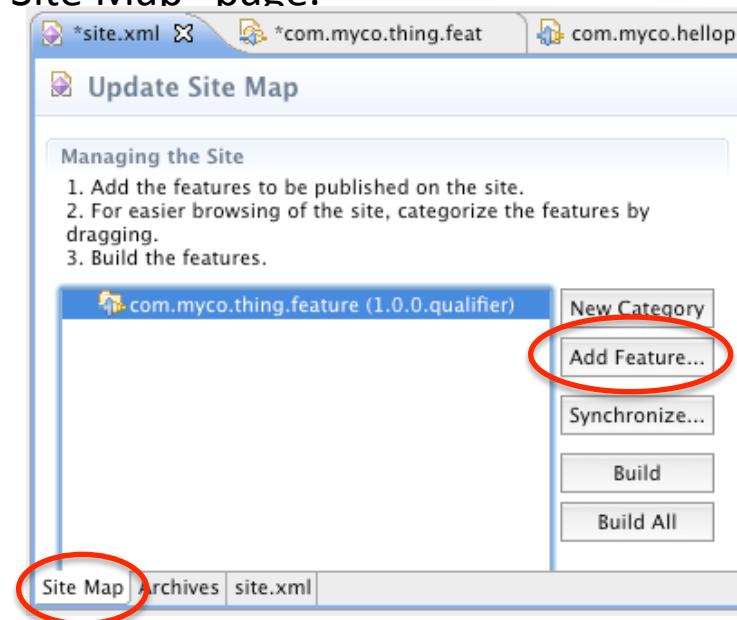
Update Site

- Create update site project

New Project, Plug-in Dev, Update Site Project

Name it e.g.: “update”

- You’re editing the site.xml file
- Add your new feature to the “Site Map” page.





Add index.html

- *Hints and tips:*
- *Add an index.html file if you plan to host the update site on a server.*
- *Then if users go to your update site URL in a browser, they won't see an error.*
- *These directions tell them to use the URL within Eclipse, not a browser.*

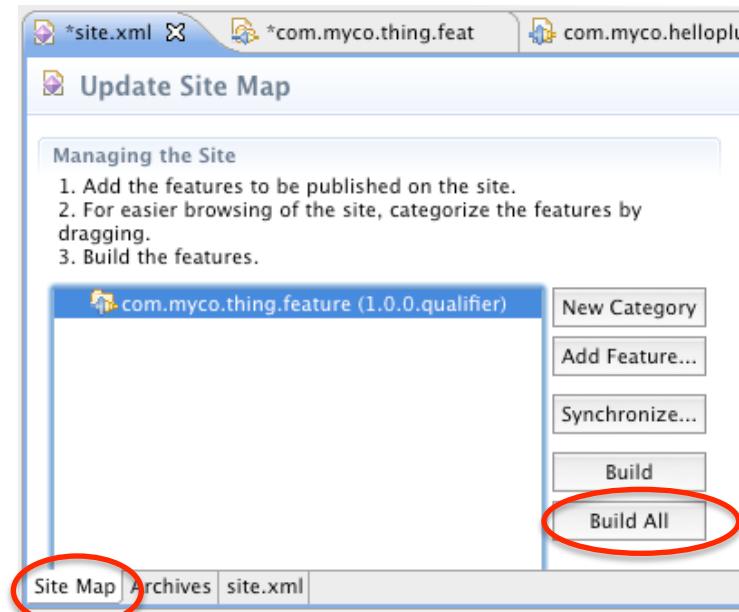
The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays several projects: com.myco.helloplugin, com.myco.help, com.myco.mything.feature, com.myco.thing.feature, com.myco.viewplugin, and an expanded 'update' folder containing index.html and site.xml. The index.html file is selected in the editor. The code editor window on the right shows the following HTML content:

```
<html>
<body>
<p>This is the Eclipse update site for com.myco.thing.
<p>
Please install this into Eclipse via Help > Install New Software...
and add this URL as a new update site.
<p>
This file assures that if a user goes to your update site URL
in a browser, they'll see something, and not an error.
</p>
</body>
</html>
```



Build the Update Site

- Build the update site
 - Click “Build All”
 - Or: site.xml > rightMouse > PDE Tools > Build site





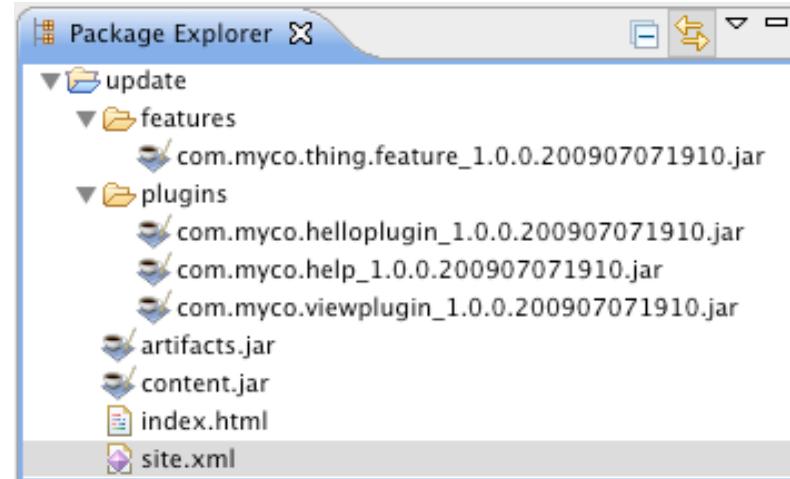
Update site: what got built

- This project contains the update site :

- site.xml
- index.html
- features dir
- plugins dir

- Notes:

- In versions, “qualifier” was replaced with timestamp
- Move these to a web/file server and you have an update site!





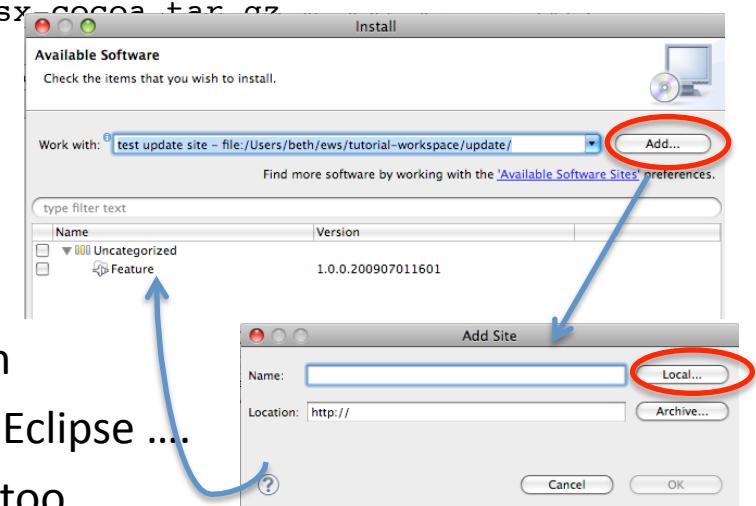
Update site: Test install

– Extract a clean Eclipse SDK

- Tar or unzip ... launch ... use new workspace name

```
$ mkdir test-install  
$ cd test-install  
$ tar -xzf ~/downloads/eclipse-SDK-3.5-macosx-cocoa-tar.gz  
$ cd eclipse  
$ ./eclipse
```

- Help > Install New Software...
- Add ... Local any name will do
- Point at your update site project location
- Check your feature, finish install, restart Eclipse
- Voila! New button in toolbar; new view too.





(Optional) Tweak feature/install

- Add a named Category
 - Update Site project: edit site.xml; on “Site Map” tab, click “New Category”, give it a name, drag your feature under it
 - Rebuild site site.xml > PDE Tools > Build Site
- Create new empty eclipse SDK (unzip again)
- Re-install: may not see the changes you made. Remove update site project & re-add, re-build.
 - Caching makes this awkward!!



(Optional) Handmade New Extension

Contributing to an Editor's toolbar – action to count words in editor selection

- In a new plugin project, or an existing one: Edit plugin.xml: Manifest editor, extensions tab
- Add: org.eclipse.ui.editorActions
- Target id: org.eclipse.ui.DefaultTextEditor (use browse To find it)
- new > action label: “MYCO: word count ”
 - Icon: select the icons/sample.gif already in project
 - Tooltip: “MYCO: show word count of selected text”
 - enablesFor: “+” (one or more selected items; this is editor so an item is a character in the editor)
 - Toolbarpath: “additions”
 - Id: (optional) change to com.example.helloworld.wordcount (just for sanity)
 - Class: click underscored “class*.” to create with wizard
 - Name: WordCounter
 - Enter code.... (see next slide or provided sample files for run() and selectionchanged() methods)
 - Manifest: Add dependency on org.eclipse.ui.editors to resolve “TextEditor”
 - Add dependency on org.eclipse.jface.text to resolve “ITextSelection”
 - Show Quick Fix, Organize Imports... to fix the imports
- Test plugin! Launch runtime workbench.
 - Select text in editor, run new Action. Note *any* selection works. Remainder is Optional:
 - Change to only enable on editor text selection – close runtime workbench.
 - Extensions, rightmouse on “MYCO: word count (action)” New > selection and browse for “ITextSelection”
 - Inspect the plugin.xml that was generated. See how the XML changed.
 - Relaunch; note action button is greyed out if text isn't selected in the editor



(part of) WordCounter.java

Past this inside the class {}

```
private TextEditor textEditor;
static final String WORD_DELIMITERS = ".;/?<>:[]{}\\|`~!@#$%^&*()_-+=\\n\\r";

public void setActiveEditor(IAction action, IEditorPart targetEditor) {
    textEditor = (TextEditor) targetEditor;
}

public void run(IAction action) {
    IDocument document = textEditor.getDocumentProvider().getDocument(textEditor.getEditorInput());
    ITextSelection ts = (ITextSelection) textEditor.getSelectionProvider().getSelection();
    int tokenCount;
    try {
        String text = document.get(ts.getOffset(), ts.getLength());
        tokenCount = new StringTokenizer(text, WORD_DELIMITERS).countTokens();
    } catch (BadLocationException e) {
        tokenCount = 0;
    }
    MessageDialog.openInformation(null, "JDG2E: Word Count", "Number of words: " + tokenCount);
}

public void selectionChanged(IAction action, ISelection selection) {
    if (selection != null && selection instanceof ITextSelection) {
        ITextSelection sel = (ITextSelection) selection;
        if (sel.getLength() == 0) {
            action.setEnabled(false);
        } else {
            action.setEnabled(true);
        }
    } else {
        action.setEnabled(false);
    }
}
```



SA: Source Code Analysis

.. And some source code repository fun @ dev.eclipse.org

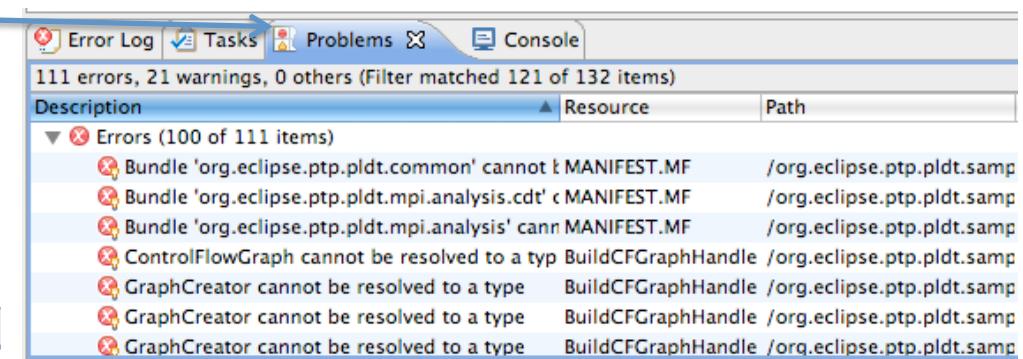
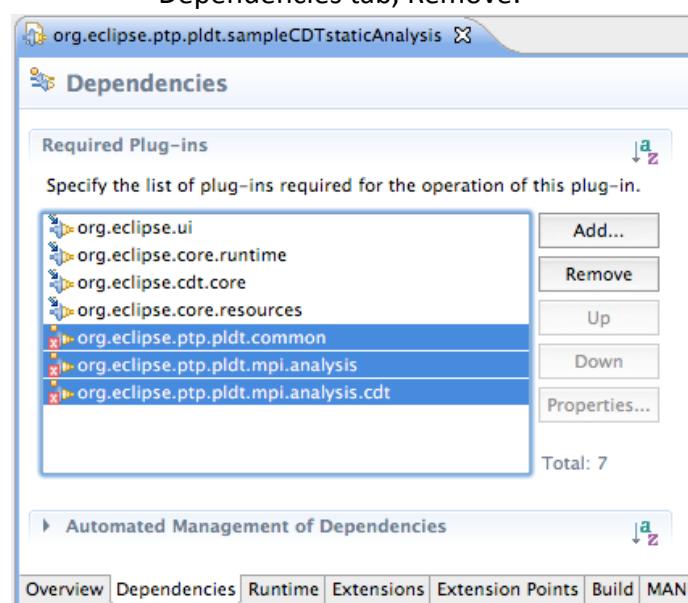
... And some plug-in dependency debugging

- We will do some source code analysis on C/C++ code
- First get CDT (Eclipse C/C++ Development Tools) : Help > Install new software, Galileo site
 - Galileo Site > Programming Languages, Eclipse C/C++ Development Tools – gets the basics
Or: Add Site: <http://download.eclipse.org/tools/cdt/releases/galileo> - Lots more incl. alt. parsers, UPC
 - Note: you must have gcc/gdb/make etc. to actually *build/run* C code – but not required for this tutorial
 - Install and Restart eclipse
- Check out a plugin project from dev.eclipse.org repository:
 - Window > Open Perspective > Other ... CVS Repository Exploring
 - In CVS Repositories view, rightMouse, New > Repository Location...
 - Host: dev.eclipse.org
 - Repository path: /cvsroot/tools
 - User: anonymous; hit Finish
 - Under the repo location, under HEAD, open org.eclipse.ptp, open tools folder, open samples folder
 - Check out org.eclipse.ptp.pldt.sampleCDTstaticAnalysis (rightMouse, Check Out)
 - Go back to PDE Perspective

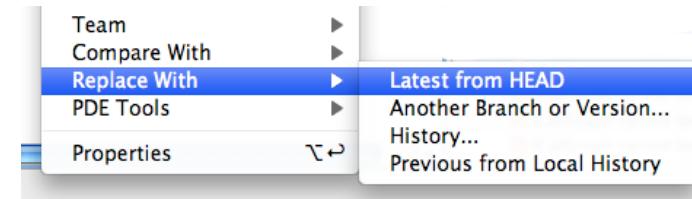


SA2: Source Code Analysis

- Look at project org.eclipse.ptp.pldt.sampleCDTstaticAnalysis you just checked out from eclipse.org
- Note red X's – open Problems View
- It seems we are missing some required plug-ins
- DETOUR: rest of this page is optional
- Try to see if we can do without them:
 - Edit META-INF/MANIFEST.MF
 - Dependencies tab, Remove:



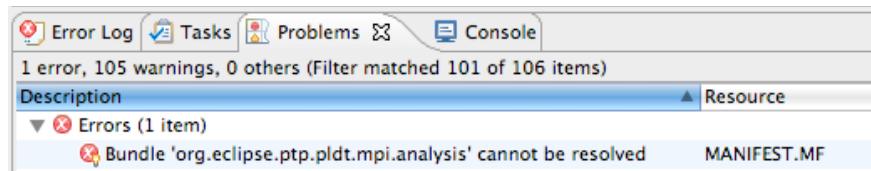
- Save manifest.
- Errors are still there. Darn.
- Get MANIFEST.MF back as we found it:
 - rightMouse, replace with,
Latest from HEAD





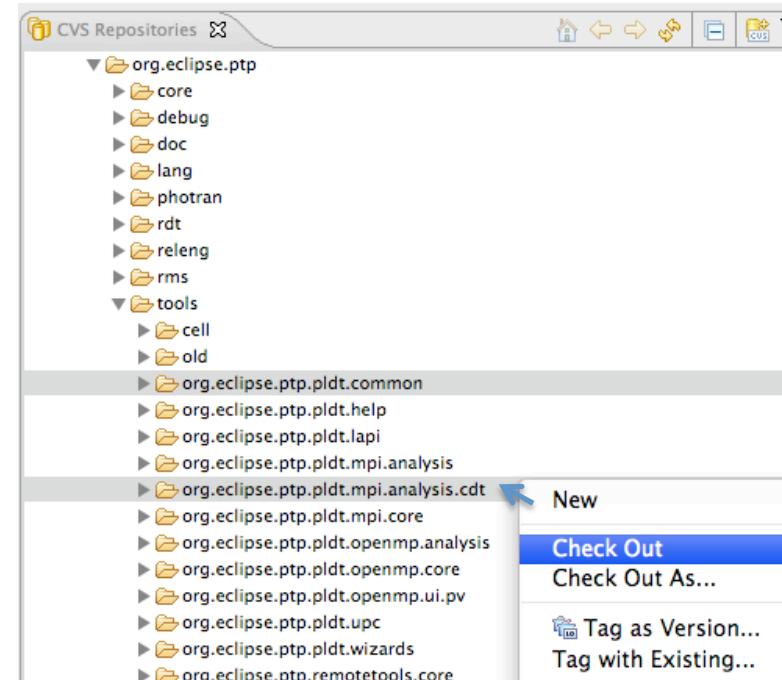
SA3: Source Code Analysis

- Let's check out the projects we need to correct the dependency errors.
- Back to CVS Repository Exploring Perspective
- Check out org.eclipse.ptp.pldt.common and org.eclipse.ptp.pldt.mpi.analysis.cdt
- Resolve errors: Force a rebuild:
 - Project > Clean...
Clean projects selected below
Check the sampleCDTstaticAnalysis project
- There is still one error:



- But Note that there are no compile errors; let's assume this dependency isn't necessary
- Edit MANIFEST.MF; Dependencies tab; Remove mpi.analysis; save manifest
- Errors gone!

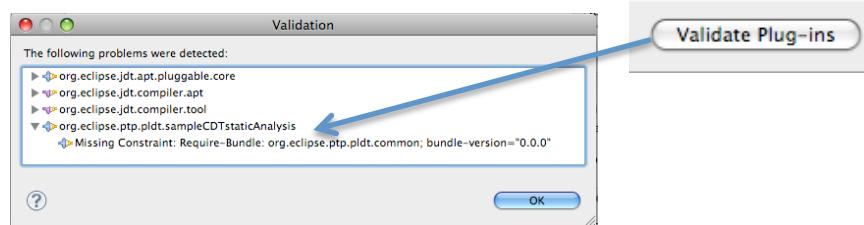
If you still get dependency errors, and you are sure you shouldn't, try (1) rebuilding project, or (2) removing and re-adding the dependencies



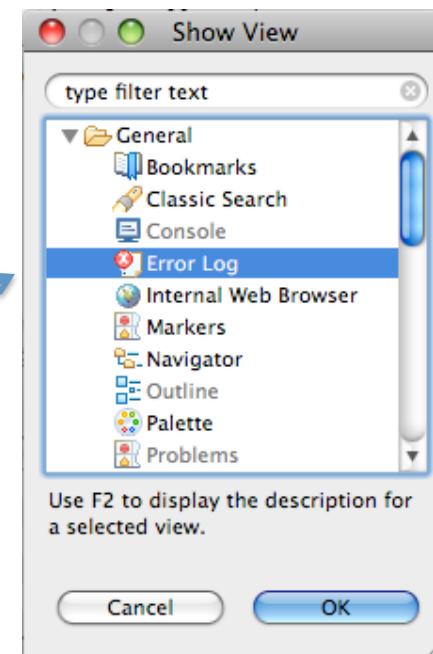


SA4: Source Code Analysis

- Launch runtime Eclipse workbench with new plugins
- Open C/C++ Perspective
 - Window >Open Perspective > Other... and select C/C++ Perspective
 - IF it's not there: Let's inspect
 - Close runtime workbench
 - Open Launch config, Plug-ins Tab, Target Platform
 - » Make sure all the CDT plugins are checked
 - » Easier: Check all at the top of Target Platform
 - » Relaunch: TA-DA! Open C/C++ Perspective
- We expect to see more toolbar buttons. Where are they?
 - Open the Error Log view: Window > Show View > Other
 - No news.
 - Close Runtime workbench. Open Launch Config.
 - On plugins tab, click “Validate Plug-ins” (lower right)



Missing plugin!! Is ptpt.common checked in the plug-ins list under Workspace? If no, check it.



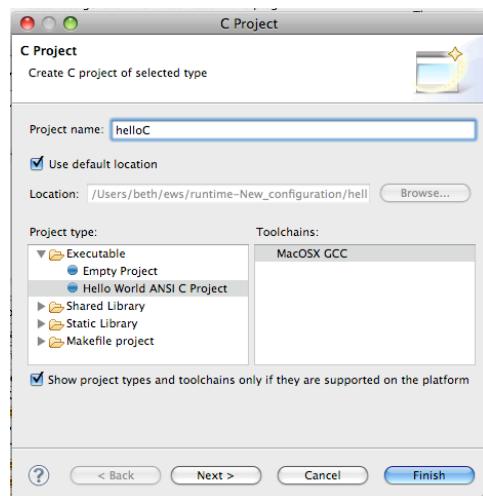


SA5: Source Code Analysis

- Re-launch runtime Eclipse workbench with all the changes
- You should now see the icons for the analysis



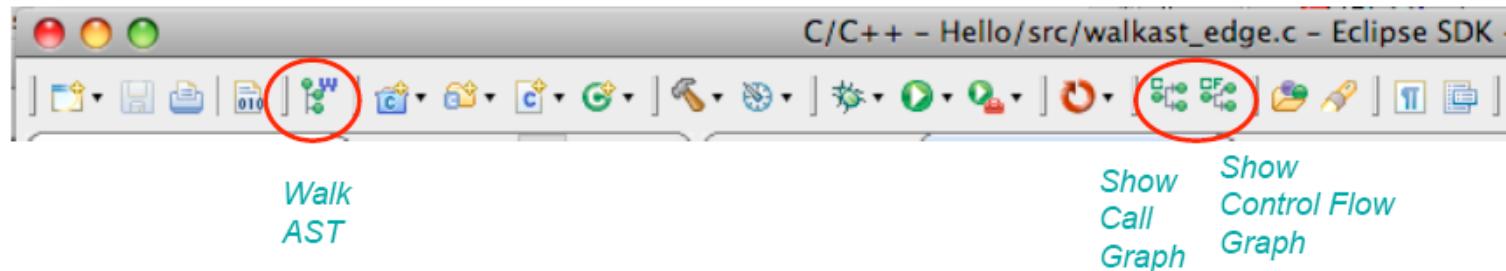
- Now make a CDT Project
- New C Project – Hello World ANSI C Project



Get source samples from cdt analysis plugin, under ‘samples’ dir
•Copy and paste from one Eclipse workbench to another!
•See
<http://www.eclipsecon.org/2008/?page=sub/&id=373> - presentation
“Static Analysis in PTP with CDT”



SA6: Source Code Analysis



- Walkast.c

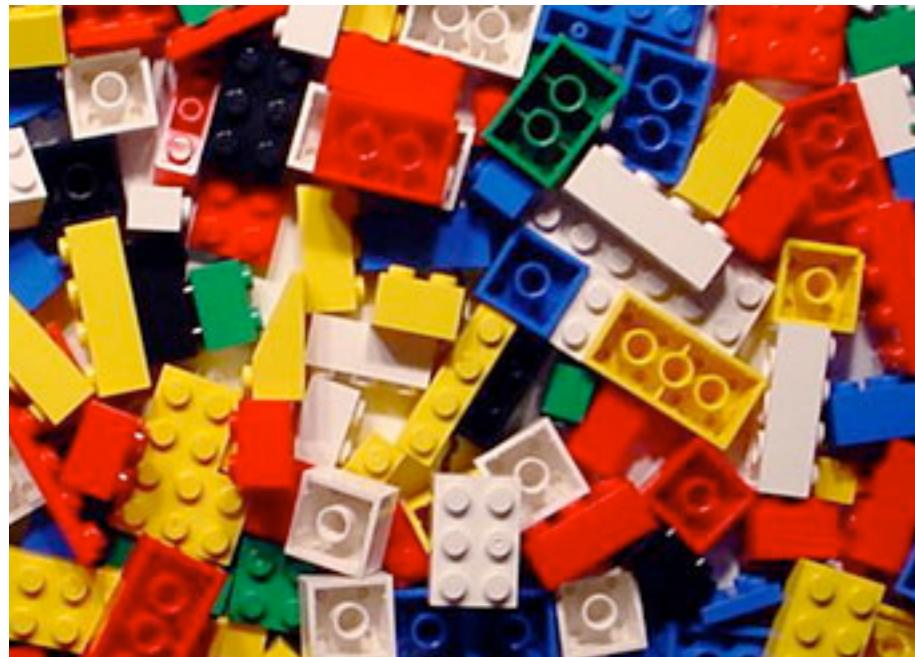
```
// p16
#include <stdio.h>
#define MYVAR 42

int main(void) {
    int a,b;
    a=0;
    b=MYVAR; // use defined
    b = b + a;
    return b;
}
int foo(int bar){
    int z = bar;
    return z;
}
```

- Select Walkast.c file in Project Explorer
- Select “Walk AST” icon in editor toolbar
- See text output in (dev. Workbench) console
- Note: tree walk includes header files (it's long!)
- See src to this plug-in for examples of C/C++ manipulation



Summary



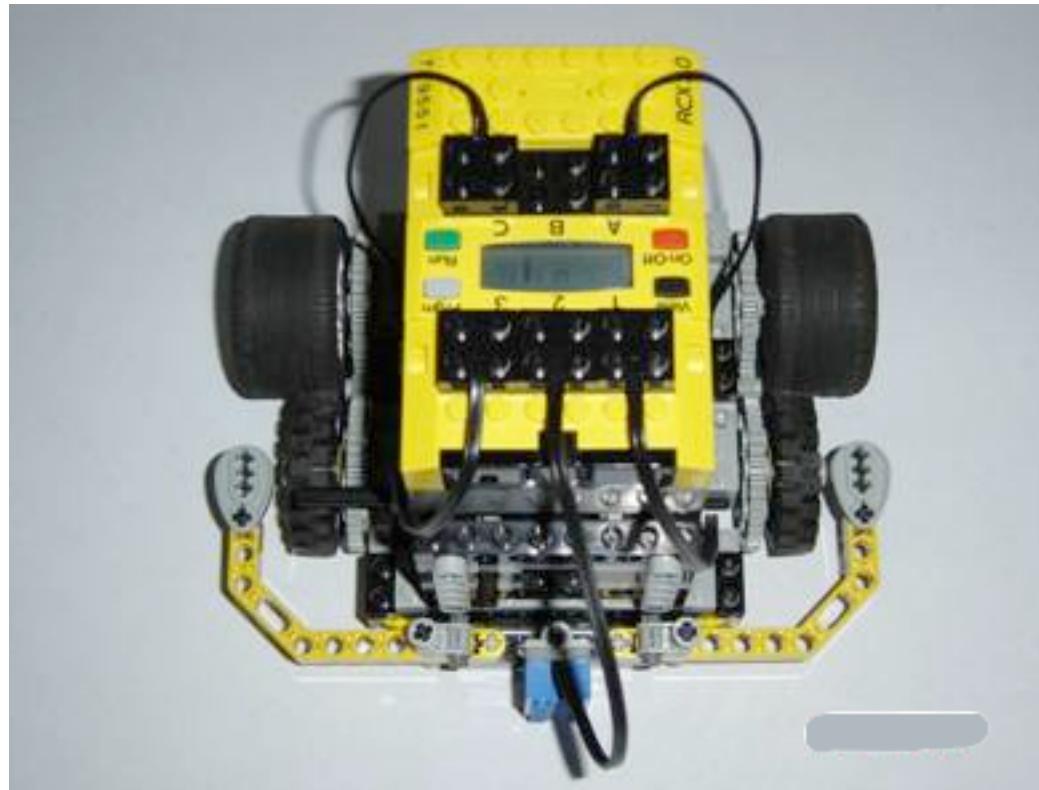


Summary





Summary





Summary

- Eclipse development has a steep learning curve
 - “swimming pool of Legos®”
 - It’s worth the investment!
 - Great set of tools, great community
- This tutorial is really just a “taste” of Eclipse development
- Take advantage of resources
 - Books, sites, help..
 - Newsgroups and mailing lists



References

- Google
- Eclipse Help – in your workbench and <http://help.eclipse.org>
- Articles on eclipse.org and IBM Developerworks
- Books
 - Eclipse plug-ins by Clayberg and Rubel (3rd ed.)
 - Java Developer's Guide to Eclipse (D'Anjou, McCarthy et.al.)

