Software Methods and Tools

Fall 2015

Instructor: Yongjie Zheng

Lab #4: Eclipse Extensions and Extension Points

Description:

In this lab, we are going to do exercises on Eclipse plug-in development. Specifically, we are going to work on the tasks of creating Eclipse plug-in projects, creating extension points and extensions, and running and installing the developed plug-ins. At the end of the lab, we will create two plug-ins with one extending the extension point declared by the other. As part of the exit exercise, we will also run these two plug-ins and install them in Eclipse.

Instructions:

1. Creating an Eclipse plug-in project

In the Eclipse workbench, select File > New > Plug-in Project.

A pane will pop up. Enter your project name and use the default setting for the other fields. Click Next.

The next pane is as shown in the following figure: plug-in ID and Version are both automatically populated and you do not need to change them; Name can be any text you like; you could use your name as Vendor.

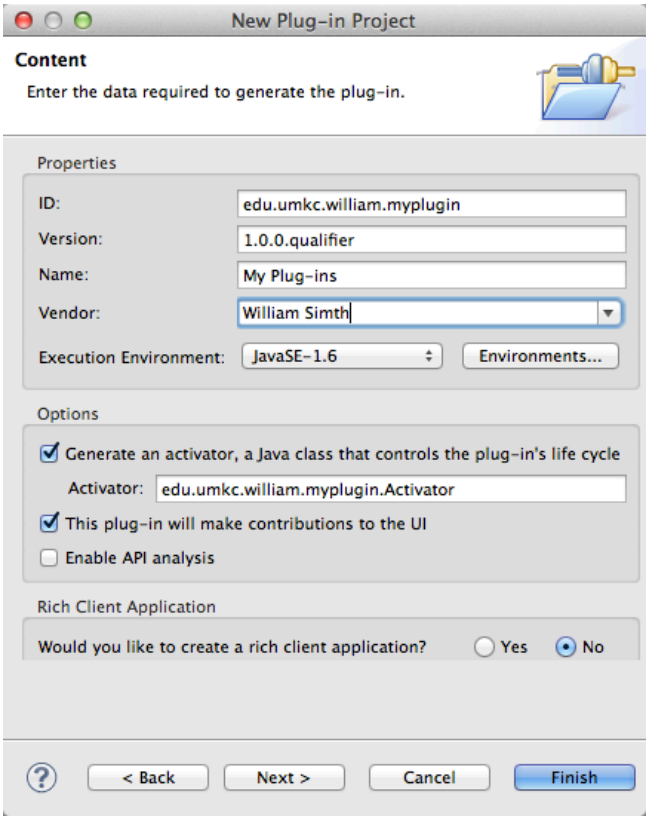
If the plug-in is UI-based (e.g. adding new menu items, new views, etc), click the checkbox next to “This plug-in will make contributions to the UI”. In our exit exercise, you will need to select this when creating the first plug-in.

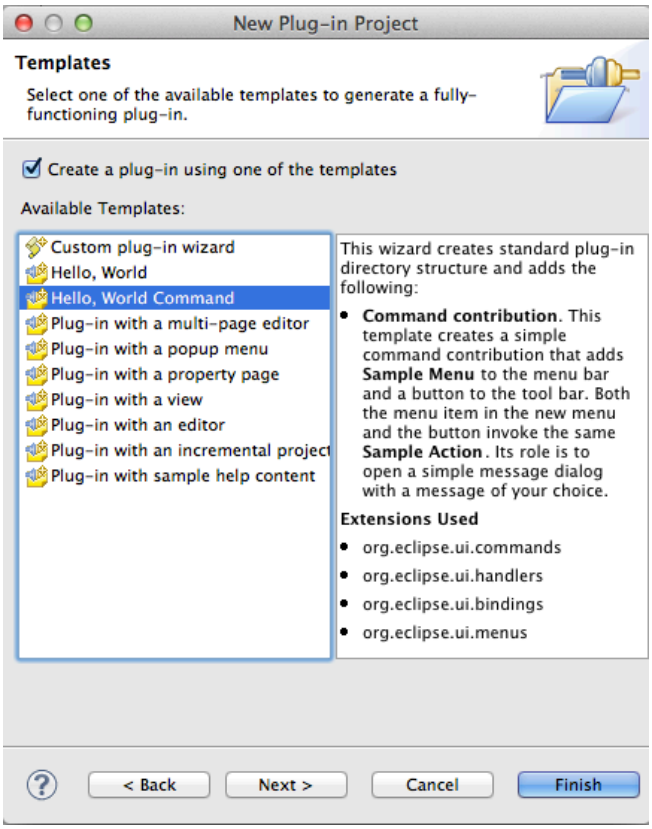
Click Next.

In the following pane, you can use any template you want to facilitate the creation of your plug-in project. In our exit exercise, you should select the “Hello World Command” template for the first UI-based plug-in.

Click Next, accept all default settings, and click Finish.

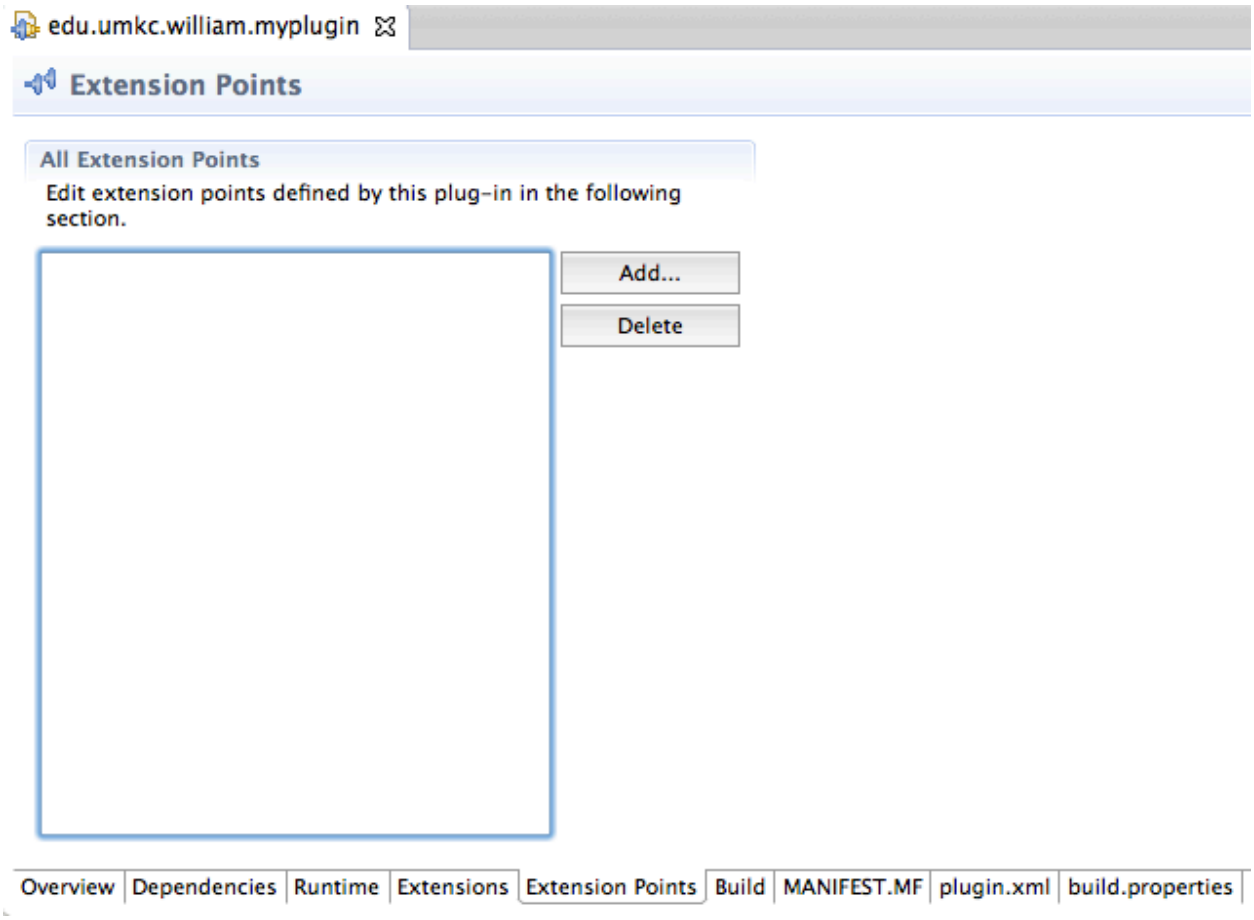
The project is created in the workspace, and the manifest editor is opened automatically.


New Plug-in Project
Content
Enter the data required to generate the plug-in.
Properties
ID:
edu.umkc.william.myplugin
Version:
1.0.0.qualifier
Name:
My Plug-ins
Vendor:
William Simth
Execution Environment:
JavaSE-1.6
Environments...
Options
Generate an activator, a Java class that controls the plug-in's life cycle
Activator:
edu.umkc.william.myplugin.Activator
This plug-in will make contributions to the UI
Enable API analysis
Rich Client Application
Would you like to create a rich client application?
Yes
No
< Back
Next >
Cancel
Finish

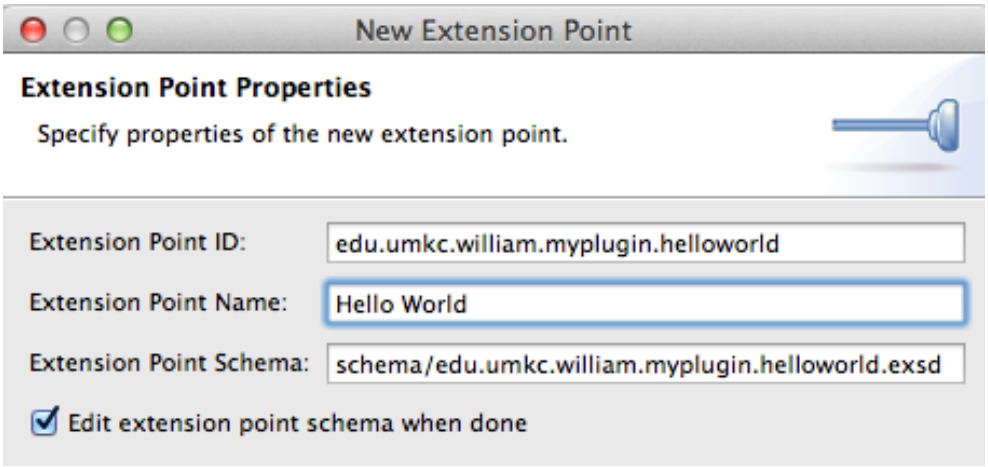

New Plug-in Project
Templates
Select one of the available templates to generate a fully-functioning plug-in.
Create a plug-in using one of the templates
Available Templates:
Custom plug-in wizard
Hello, World
Hello, World Command
Plug-in with a multi-page editor
Plug-in with a popup menu
Plug-in with a property page
Plug-in with a view
Plug-in with an editor
Plug-in with an incremental project
Plug-in with sample help content
This wizard creates standard plug-in directory structure and adds the following:
Command contribution. This template creates a simple command contribution that adds Sample Menu to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same Sample Action. Its role is to open a simple message dialog with a message of your choice.
Extensions Used
org.eclipse.ui.commands
org.eclipse.ui.handlers
org.eclipse.ui.bindings
org.eclipse.ui.menus
< Back
Next >
Cancel
Finish

2. Creating an Eclipse extension point

Open the Manifest editor, go to the Extension Points tab, and click Add

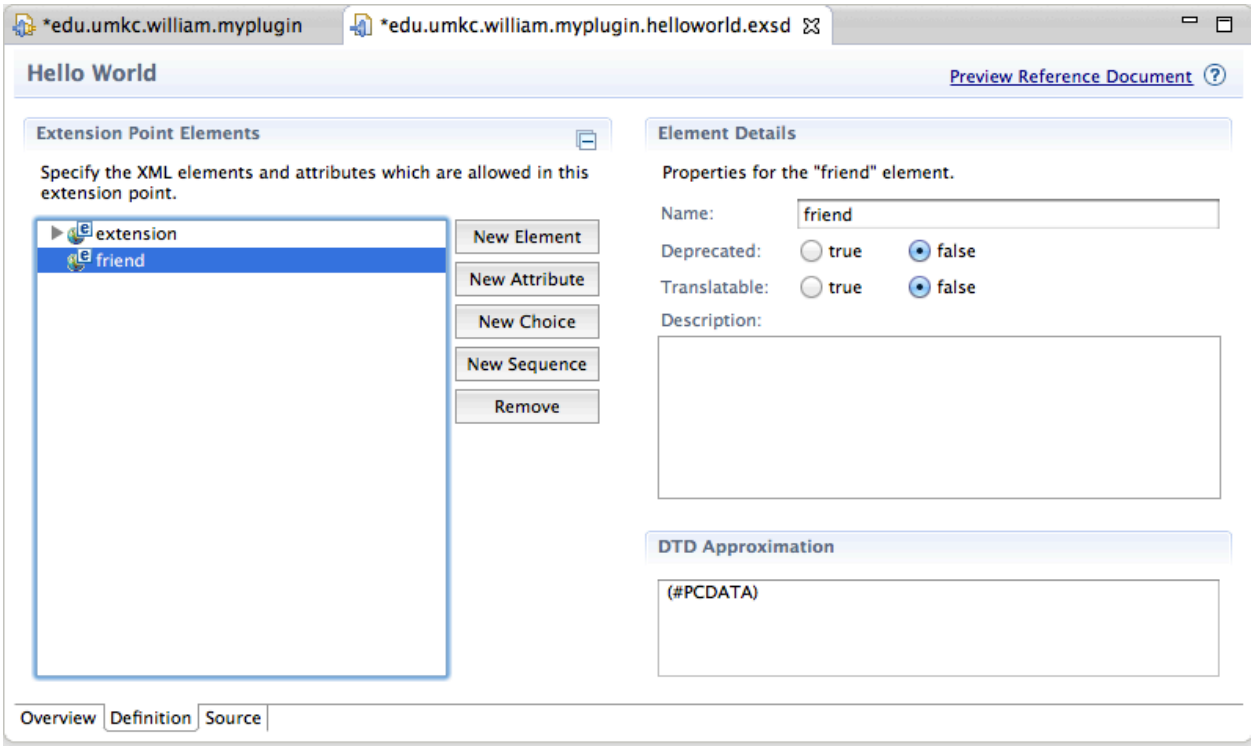


In the next pane, enter Extension Point ID and Extension Point Name.

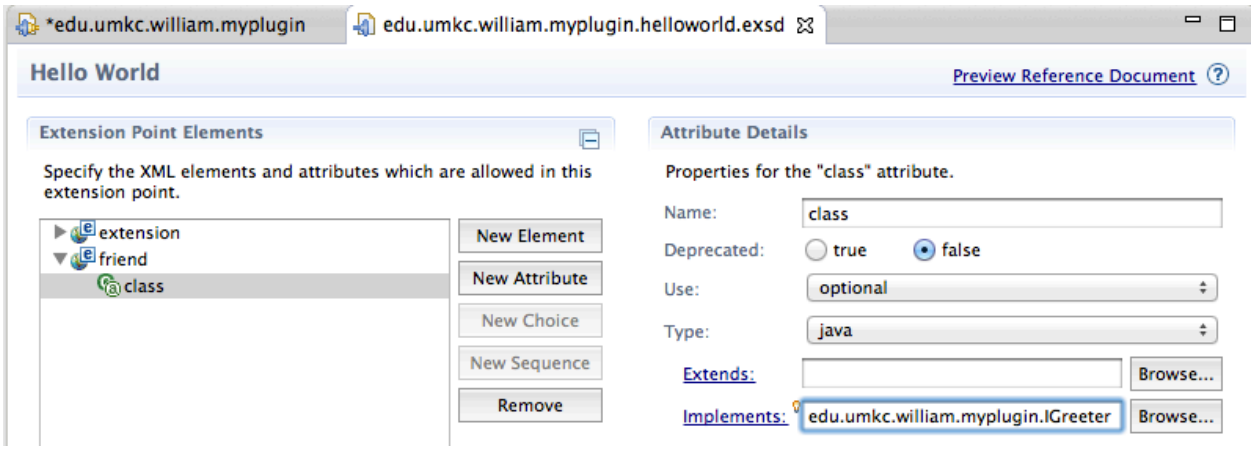


Go to the Definition tab, here you can add more elements or attributes to the extension point you created. In the exit exercise, you will need to do the following.

Click New Element, and give it a name (e.g. friend).



Add a new attribute, “class”, to the friend element. Make the type of the attribute “java”. Enter the fully qualified name of the interface that you want extension plug-ins to implement.

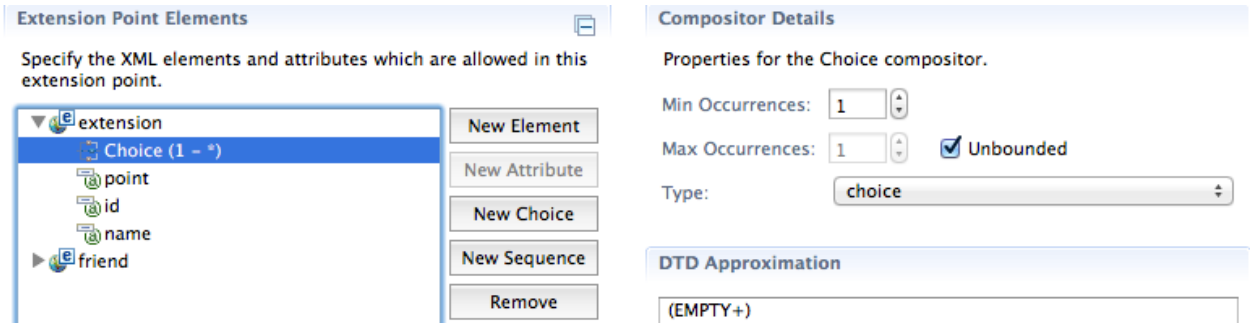


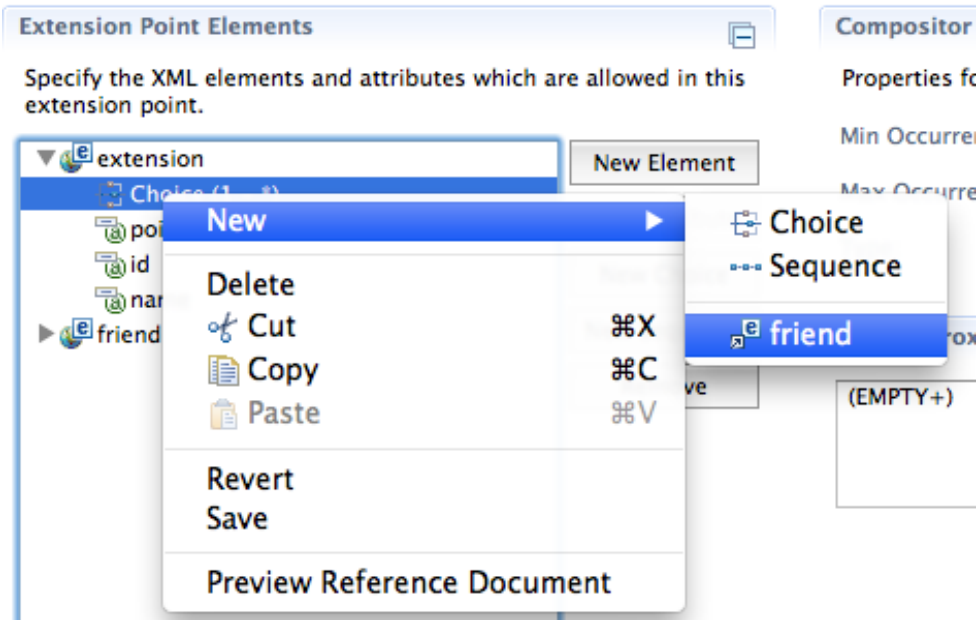
Click the link of “Implements”, and edit the code of the interface in the editor that opens.

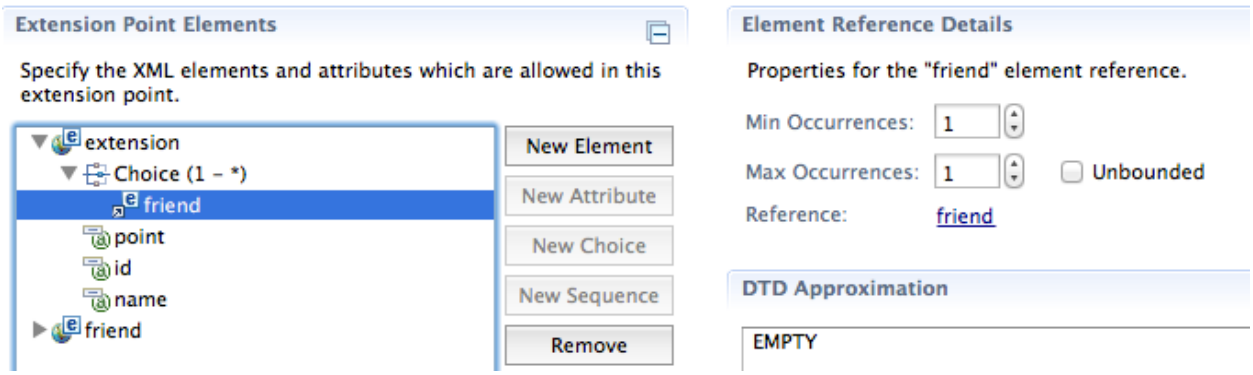
*edu.umkc.william.myplugin *edu.um

```
package edu.umkc.william.myplugin;

public interface IGreeter {

    String sayHello();

}
```

Go back to your extension point definition and add a choice to the extension point you just created. Make its Max Occurrences “Unbounded”. This defines how many elements can be defined in the plug-in that extends or contributes to this extension point. Add the “friend” element to this choice compositor.

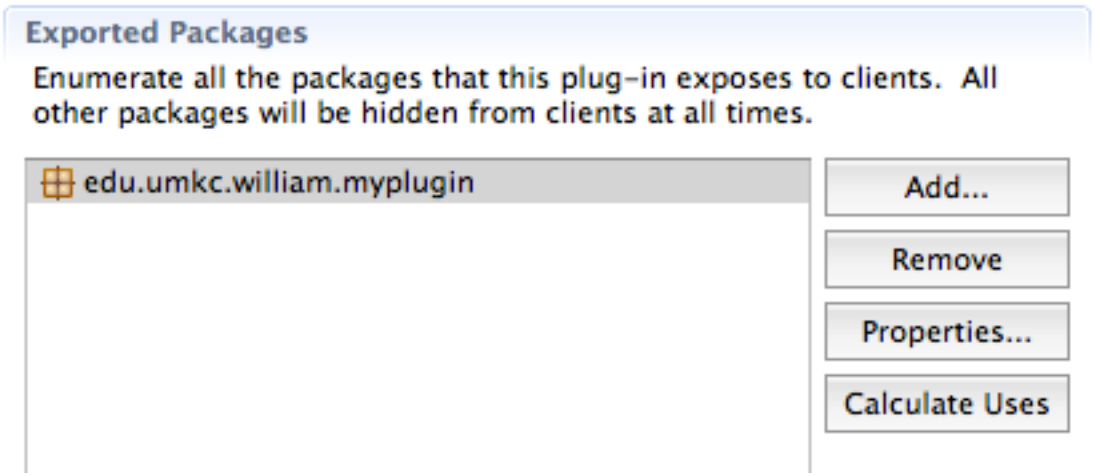






Go to the Runtime tab in your manifest editor, add the package that contains your interface file into the list of Exported Packages.

Save all the changes you made above.



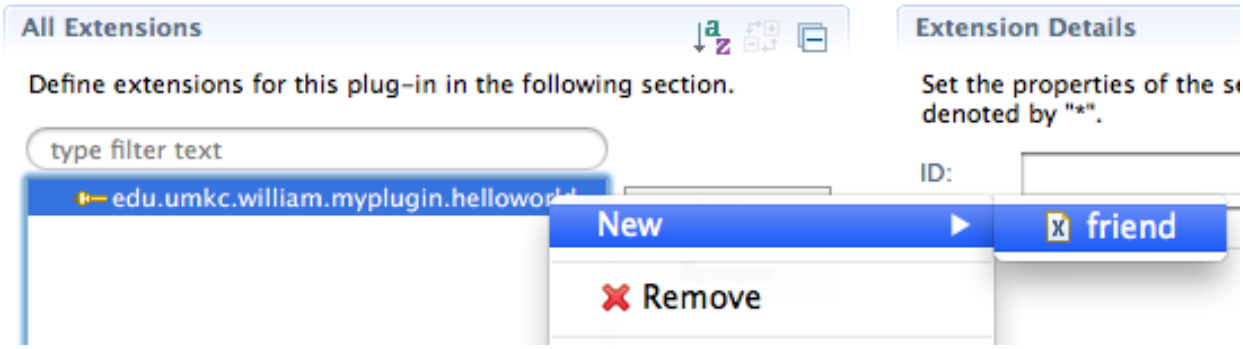
3. Creating an Eclipse extension

Create another plug-in project. In the exit exercise, this plug-in project is headless, does not contribute to the UI, and should not use any template provided.

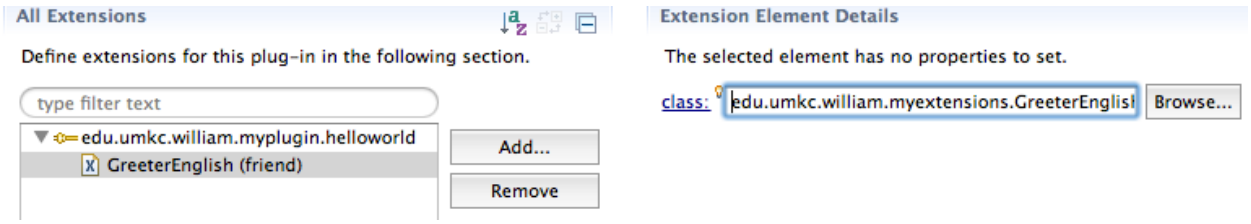
After the project is created, open its manifest editor, and go to the Dependencies tab. Add the plug-in project that declares the extension point into the dependency list. Save the file.

Go to the Extensions tab, click Add, and select the extension point that you want to extend.

Right click the extension point, and select New > friend (element name).



Enter the name of the class that implements the interface declared in the extension point.



Click the “class” link to edit the Java code as shown below.

```
package edu.umkc.william.myextensions;

import edu.umkc.william.myplugin.IGreeter;

public class GreeterEnglish implements IGreeter {

    public GreeterEnglish() {
        // TODO Auto-generated constructor stub
    }

    @Override
    public String sayHello() {
        // TODO Auto-generated method stub
        return "Hello!";
    }

}
```

Repeat the above process, you can add more elements of “friend” to this extension. For example, GreeterIndian, GreeterChinese, etc.

4. Running a plug-in

In the Eclipse workbench, select Run > Run As > Eclipse Application.

5. Installing a plug-in

Refer to the lecture notes for details.

Exit Exercise

Finish the following tasks. For each specific task, follow the corresponding instruction provided above.

Task 1: Create a plug-in project based on the Hello World Command template of Eclipse.

The name of this plug-in could be: edu.umkc.[your first name].myplugin

Task 2: Create an extension point in the plug-in project you just created. Add an element named “friend”, which has an attribute named “class” of the “java” type. At the end, add “choice” to the extensions element and make it “unbound”. Then add the “friend” element into “choice”.

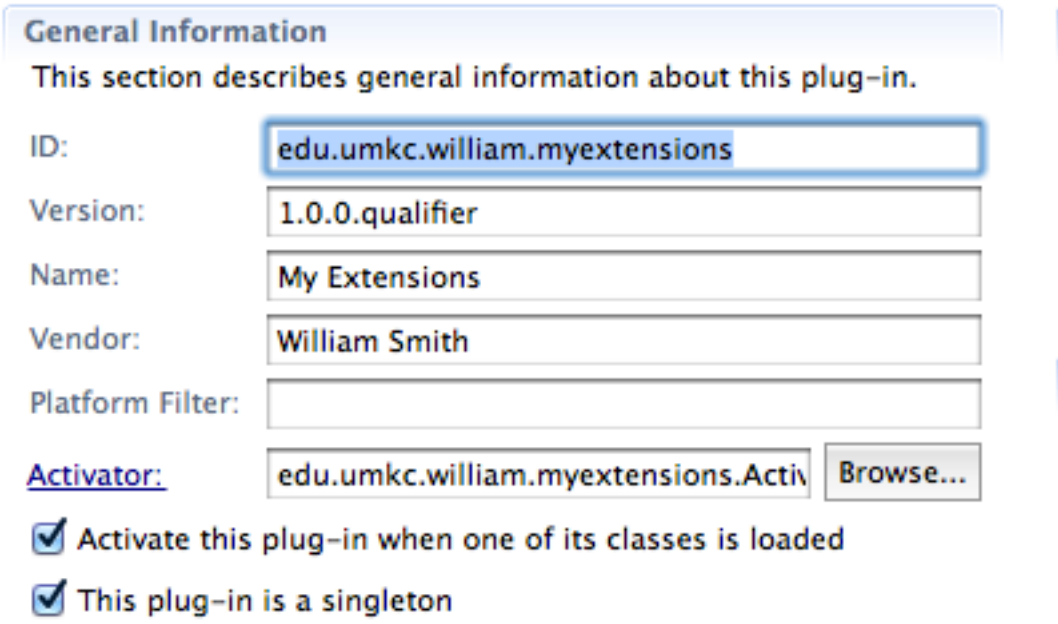
Note: do not forget to export your package.

The extension point ID could be: [your plug-in id].helloworld

Task 3: Create a headless plug-in project without using any provided template.

Suggestion: the name of your plug-in could be: edu.umkc.[your first name].myextensions

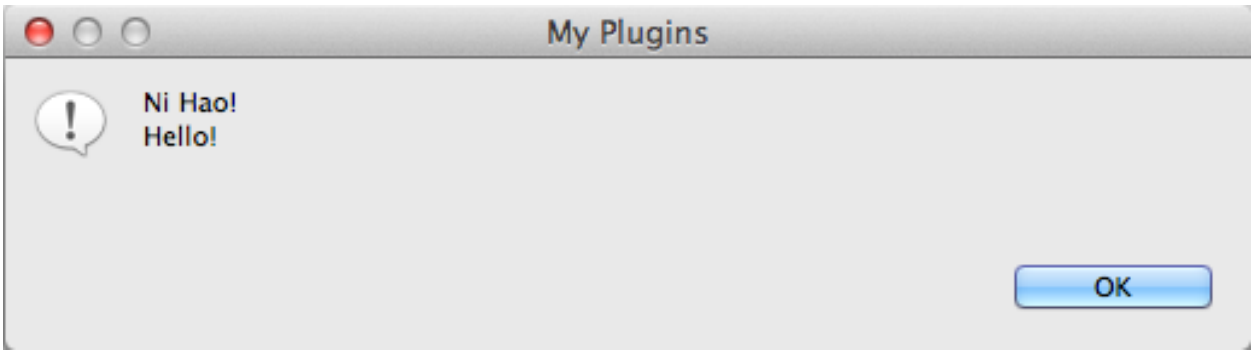
Note: In the Overview tab of the manifest editor, make sure that the checkbox “This plug-in is a singleton” is selected. This means that only one instance of the plug-in will be created when Eclipse runs.



Task 4: In the second project you created, create an extension for the extension point declared in the first point. In particular, this extension includes two elements named "friend" with different implementation classes (e.g. GreeterEnglish and GreeterIndian).

Task 5: Go back to the first project you created, find and open the file SampleHandler.java in your source code folder. Modify its code as illustrated in SampleHandler.java linked on the class website.

Task 6: Run the plug-ins by launching the Eclipse runtime workbench. In the runtime workbench, click the eclipse icon in the toolbar. If everything works well, you should see the following dialog box pops up.



Hello World! Enjoy!

Reference

Eclipse Extension Points and Extensions: http://www.vogella.com/articles/EclipseExtensionPoint/article.html