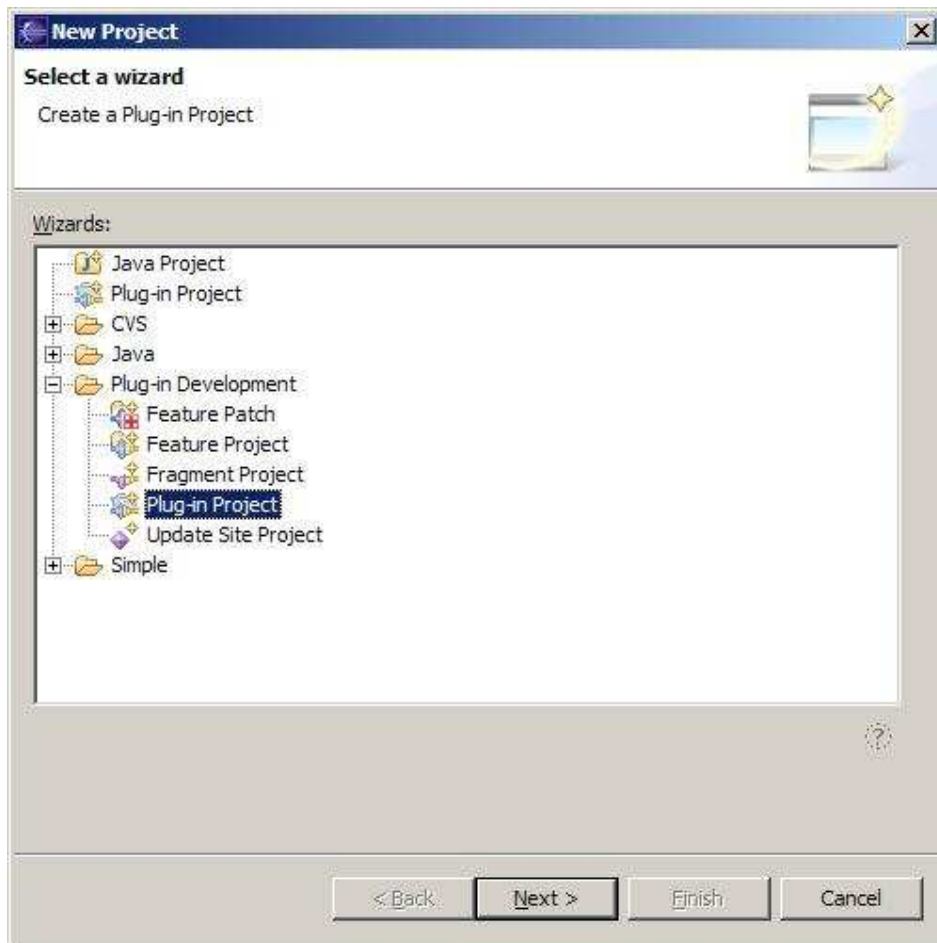


## Lab 3 Creating a 'Hello World' Plug-in

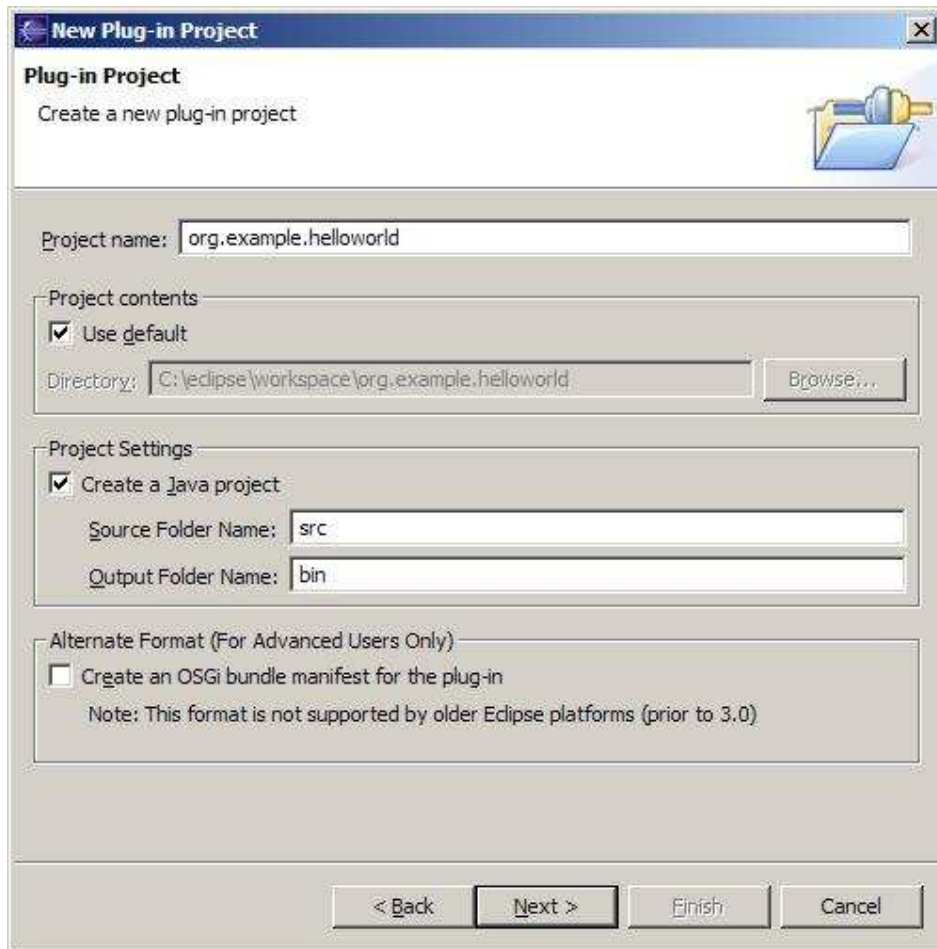
1. Select File>New>Project from the menu bar and select the options as shown and click on the 'next' button.



2. Enter the project name: 'org.example.helloworld' , leaving the Project settings as default, and select the Next button.

**Source Folder** lets you specify the subdirectory where the Java of your project are kept.

**Output Folder** specifies where Eclipse place the generated class files.



**New Plug-in Project**

**Plug-in Project**  
Create a new plug-in project

Project name:

Project contents

☒ Use default

Directory:

Project Settings

☒ Create a Java project

Source Folder Name:

Output Folder Name:

Alternate Format (For Advanced Users Only)

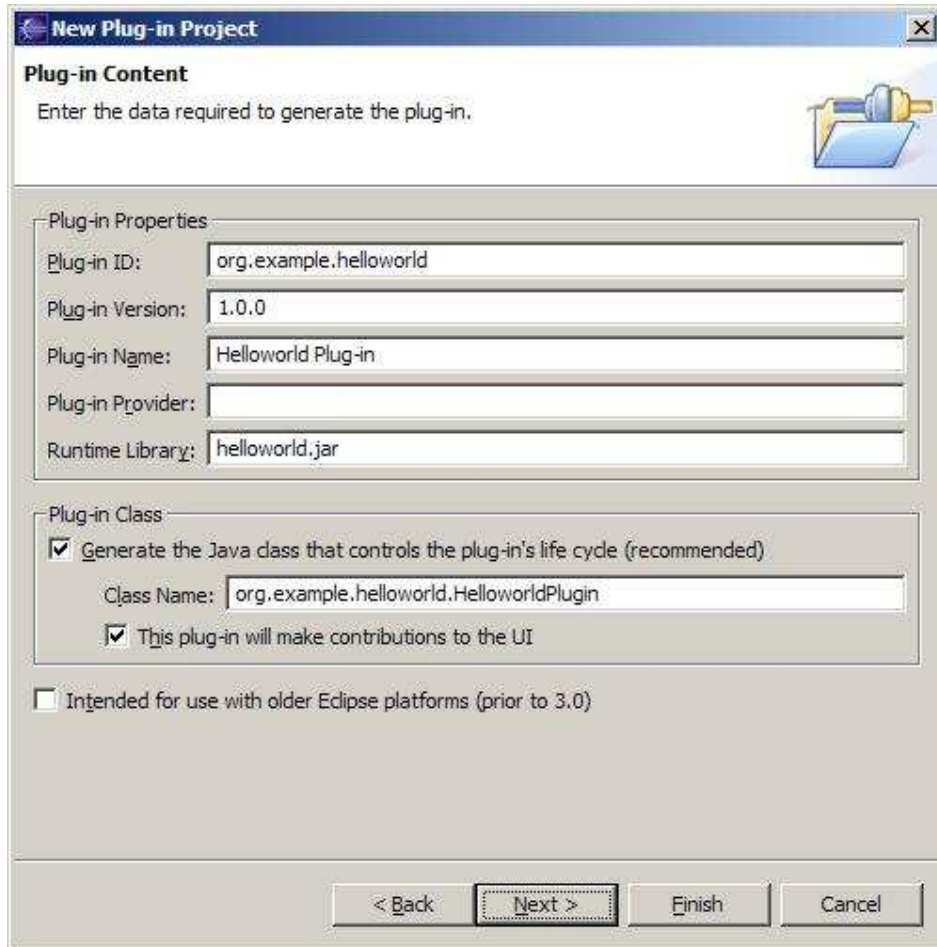
☐ Create an OSGi bundle manifest for the plug-in

Note: This format is not supported by older Eclipse platforms (prior to 3.0)

< Back   Next >   Finish   Cancel

3. In the next window define the plug-in content as follows:

**Runtime Library** specifies the JAR file that contains your Java code.



The screenshot shows the 'New Plug-in Project' dialog box. The title bar reads 'New Plug-in Project'. Below the title bar, the section 'Plug-in Content' is visible, with the instruction 'Enter the data required to generate the plug-in.' and a small icon of a folder with a plug. The main area is divided into two sections: 'Plug-in Properties' and 'Plug-in Class'. In the 'Plug-in Properties' section, there are five text fields: 'Plug-in ID:' with the value 'org.example.helloworld', 'Plug-in Version:' with '1.0.0', 'Plug-in Name:' with 'Helloworld Plug-in', 'Plug-in Provider:' which is empty, and 'Runtime Library:' with 'helloworld.jar'. The 'Plug-in Class' section contains a checked checkbox 'Generate the Java class that controls the plug-in's life cycle (recommended)', a text field 'Class Name:' with 'org.example.helloworld.HelloworldPlugin', and another checked checkbox 'This plug-in will make contributions to the UI'. At the bottom of the dialog, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a dashed border.

**New Plug-in Project**

**Plug-in Content**  
Enter the data required to generate the plug-in.

**Plug-in Properties**

Plug-in ID:

Plug-in Version:

Plug-in Name:

Plug-in Provider:

Runtime Library:

**Plug-in Class**

☒ Generate the Java class that controls the plug-in's life cycle (recommended)

Class Name:

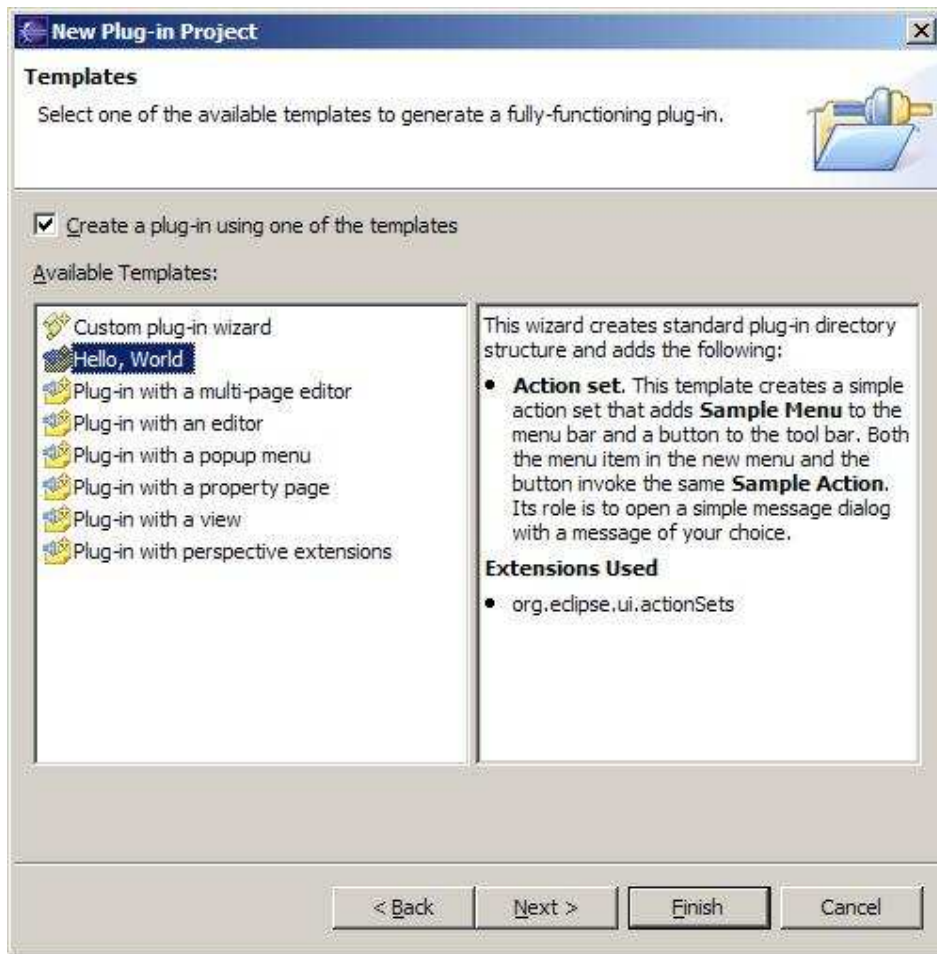
☒ This plug-in will make contributions to the UI

☐ Intended for use with older Eclipse platforms (prior to 3.0)

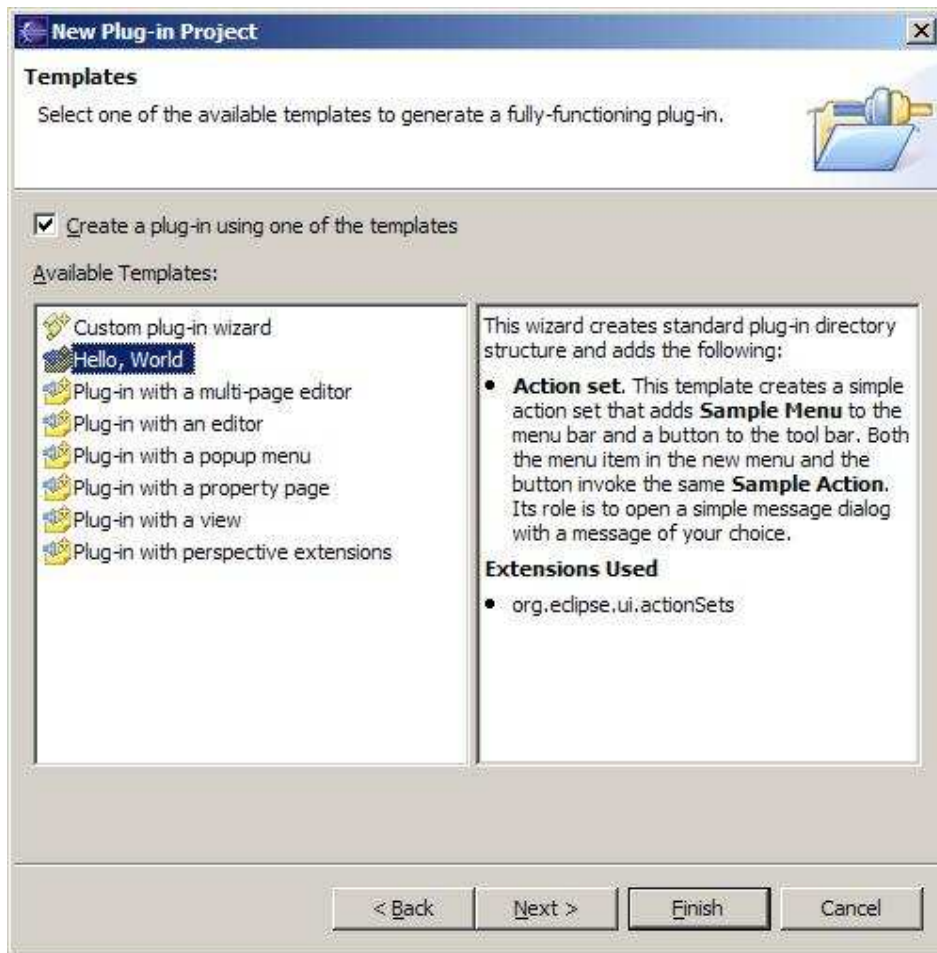
< Back   **Next >**   Finish   Cancel

Click 'Next' .

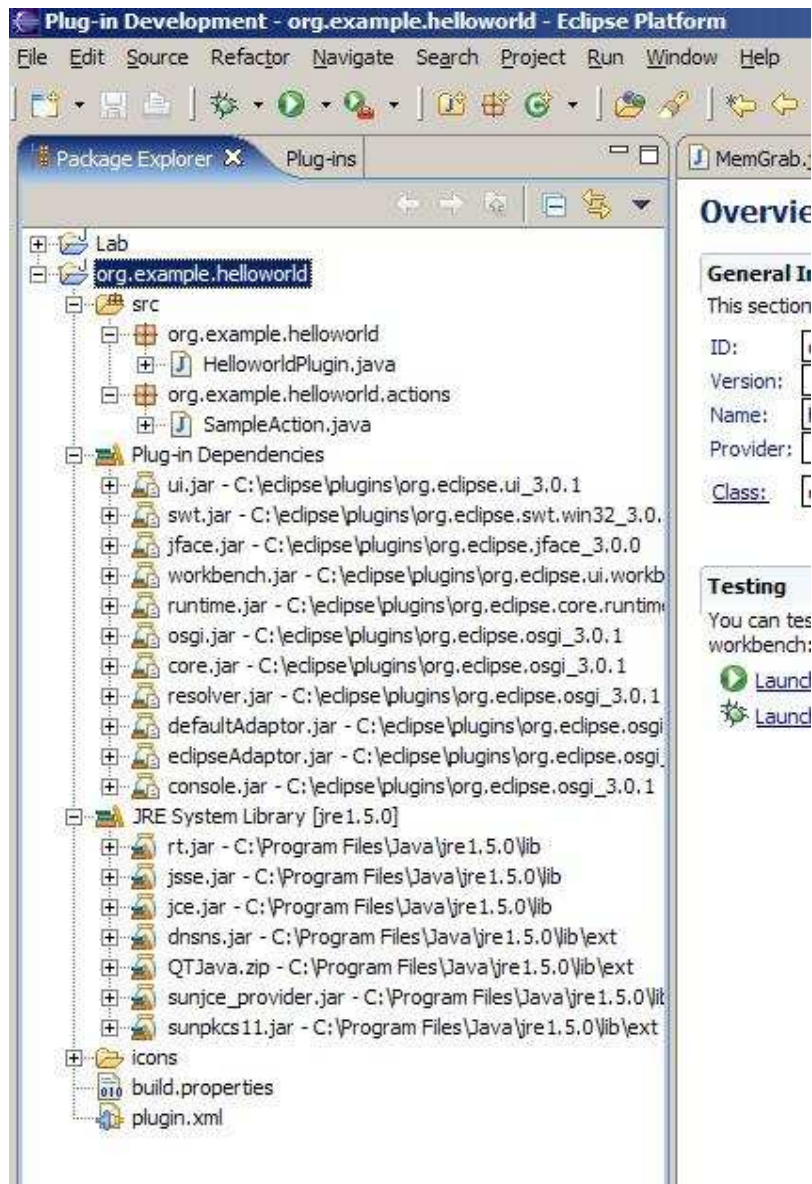
4. The next window enables you to select the plug-in code generator.  
Firstly enable **Create a plug-in project using a code generation wizard**.  
In the case of 'Hello World' a sample code generator is supplied. Select this.  
Select next.



5. Fill in the fields to define the plug-in content as below and select **Finish**



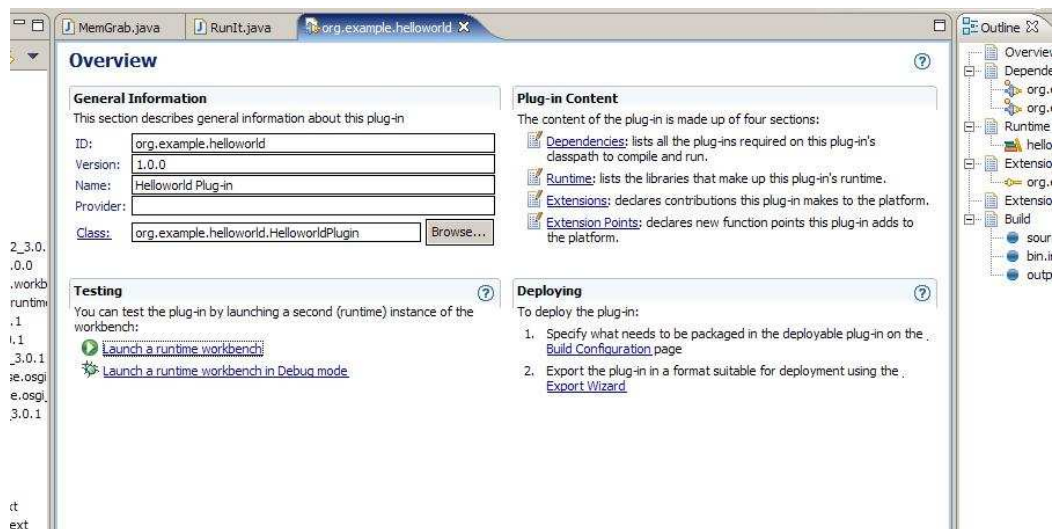
6. The plug-in has been created. The file structure ...





7. The plug-in can be run either by deploying the created files into the plug-ins directory, which is the subject of Lab 4, or run in a new spawned Eclipse image. The latter approach is used for plug-in development and is described here:

1. Select **Run > Run As > Run-time Workbench**. A new Workbench appears that includes your new plug-in menu and toolbar button.
2. Click on the 'Launch a runtime workbench'



8. Click on the 'Sample Menu', then 'Sample Action' or Click on the Eclipse Icon circled below;

