

Starting In Java with JPT in Eclipse

1. Installing Java and Eclipse

Both *Java* from Sun Microsystems and the *Eclipse* development environment are free to download. It is important that *Java* be installed first and then *Eclipse* because *Eclipse* looks for an installation of *Java*.

Java may be downloaded from:

<http://java.sun.com/j2se/downloads/index.html>

Eclipse may be downloaded from:

<http://www.eclipse.org/downloads/index.php>

2. Installing the Java Power Tools in Eclipse

The *Java Power Tools* are installed by installing the library *jpt.jar* into an appropriate folder in the *Eclipse* directory tree. The current version 2.3.2 of *jpt.jar* may be downloaded from:

http://www.ccs.neu.edu/jpt/jpt_2_3/lib/jpt.jar

Let us explain where to store *jpt.jar* using Windows folder notation. Use a similar location for Mac OS X or Linux. Open the following folder:

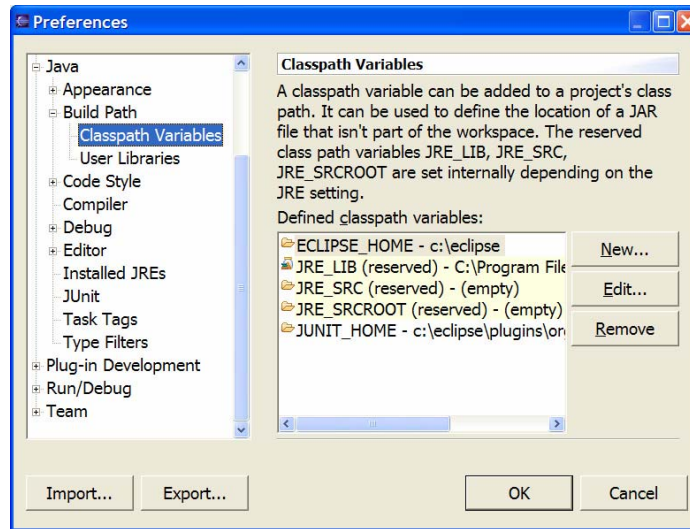
C:\eclipse\plugins

In this folder, create the subfolder:

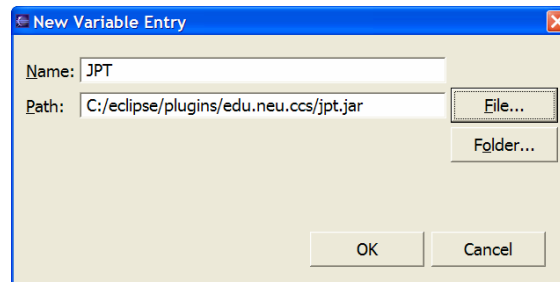
edu.neu.ccs

Place *jpt.jar* in this subfolder.

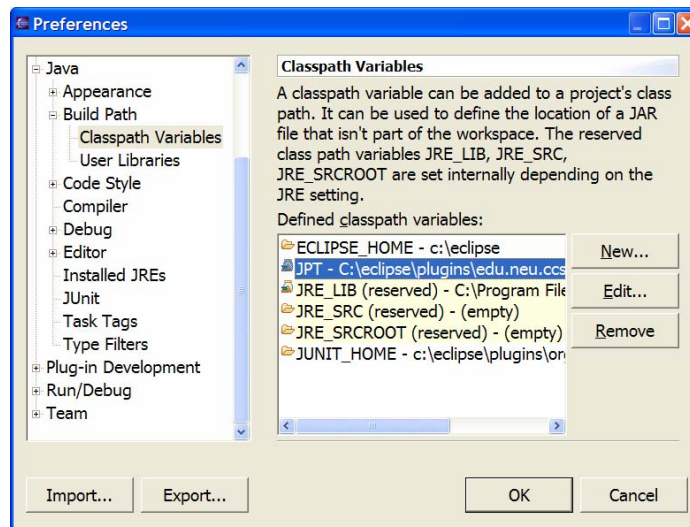
The next step is to install a *classpath variable* in *Eclipse* that will make it easy to attach *jpt.jar* into any desired *Eclipse* project. In *Eclipse*, select the menu item *Window* → *Preferences*. In the dialog that is displayed, select the option *Java* → *BuildPath* → *ClassPath Variables*.



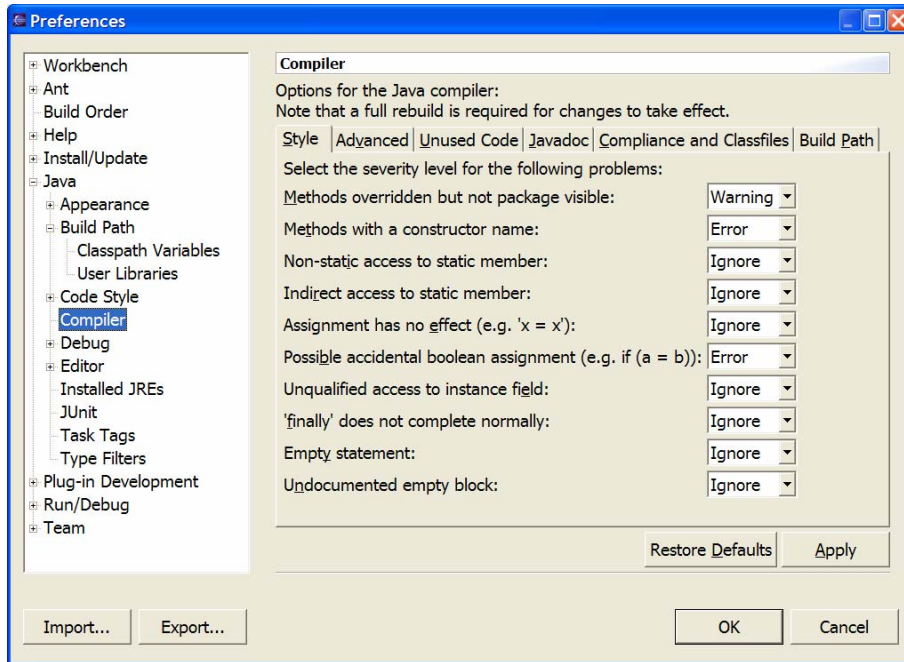
Click *New...* and fill in the resulting dialog as follows.



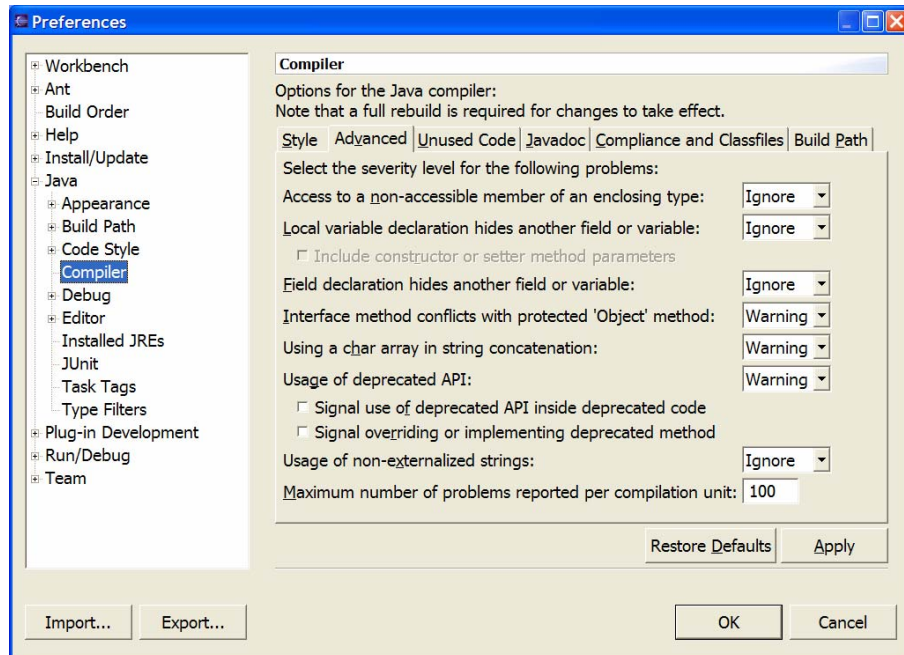
You will then see the *classpath variable* JPT installed in the original dialog.



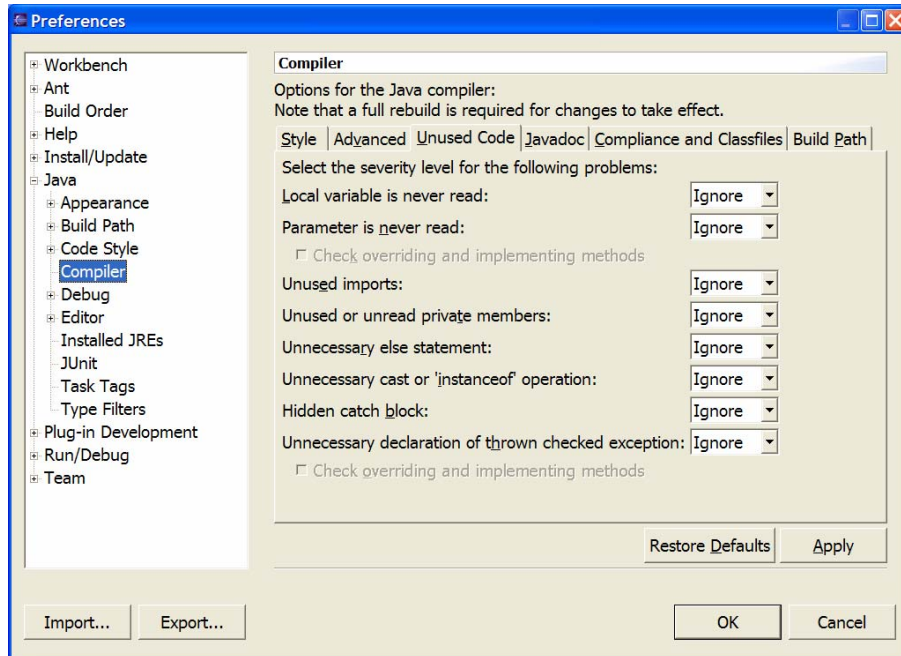
Before closing the *Preferences* dialog, select *Compiler* to set some compiler options. Set the options in the *Style* tab as follows.



Set the options in the *Advanced* tab as follows.



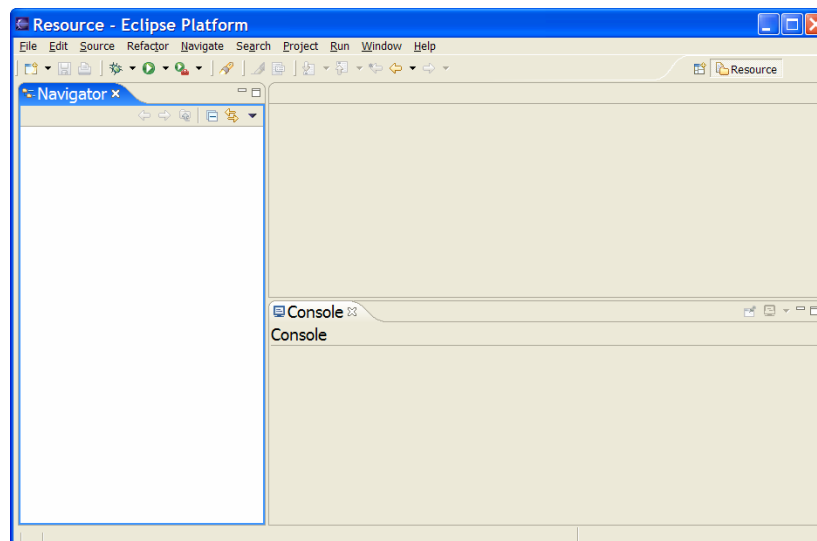
Set the options in the *Unused Code* tab as follows.



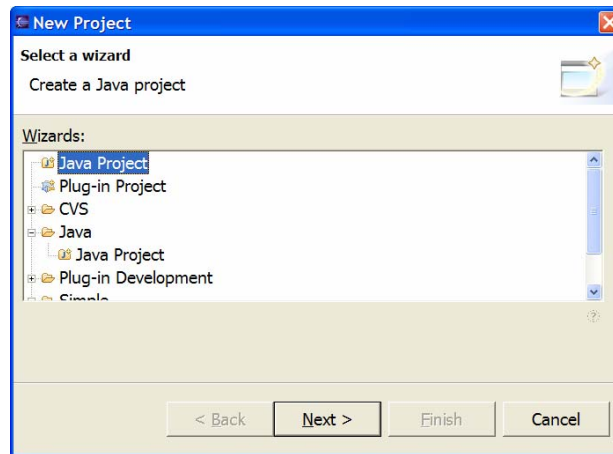
All of the particular settings in the *Unused Code* tab are set to *Ignore* in order to permit experimental code in which some code or imports are present for possible later use.

3. Creating a Project in Eclipse

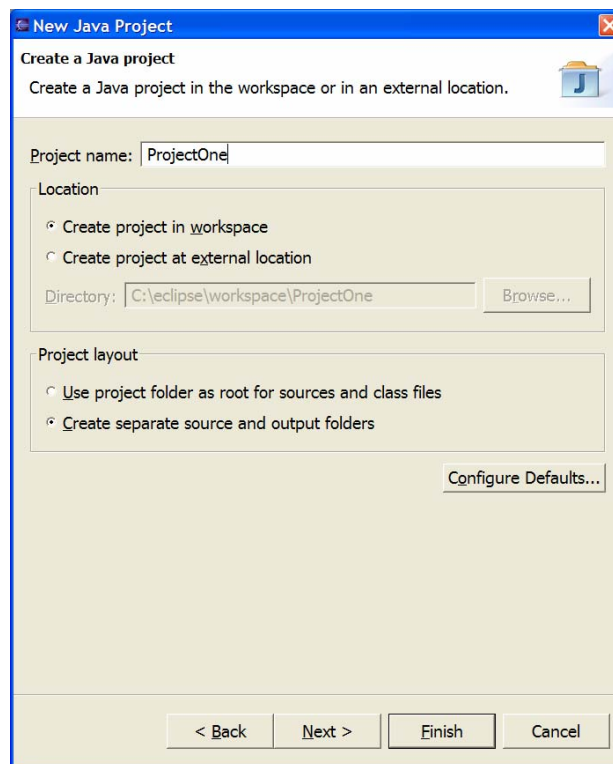
Here is a snapshot of *Eclipse* with no projects yet defined.



To create a project, select the menu item *File* → *New* → *Project*.

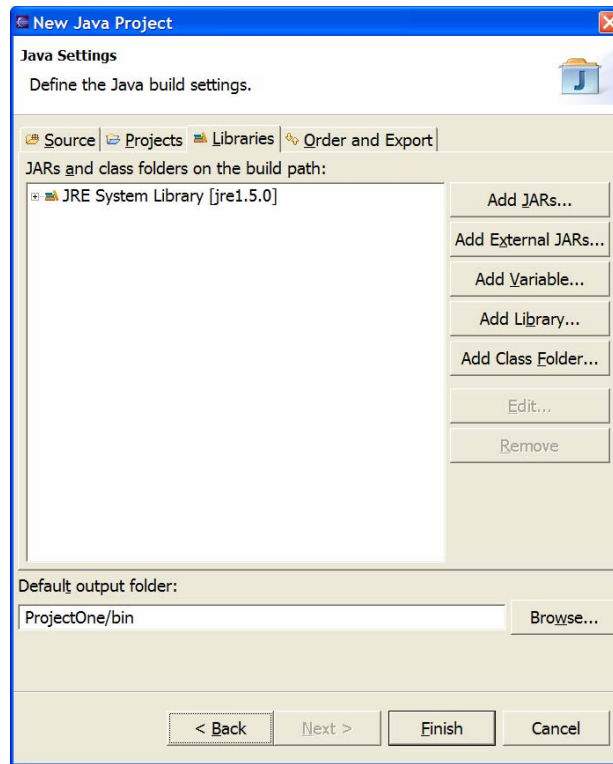


Select *Java Project* and click *Next >*.

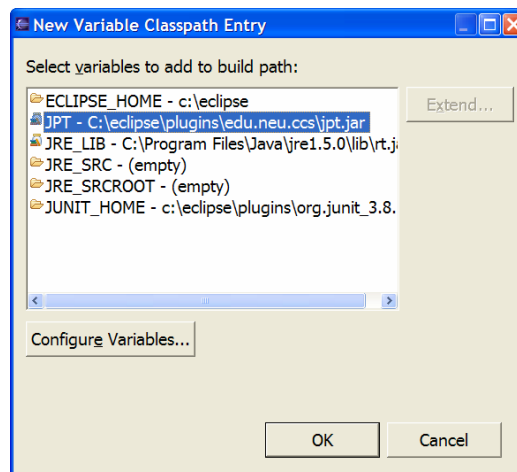


Notice that we have named the project *ProjectOne* and that we have selected the *Project layout* bullet choice *Create separate source and output folders*. This choice means that you will keep your *Java* source files separate from any compiled *Java* code class files and you will have a much less cluttered view of your project.

To allow the project to access the *Java Power Tools* library, click *Next >* rather than *Finish*. In the next stage of the dialog, click the *Libraries* tab.

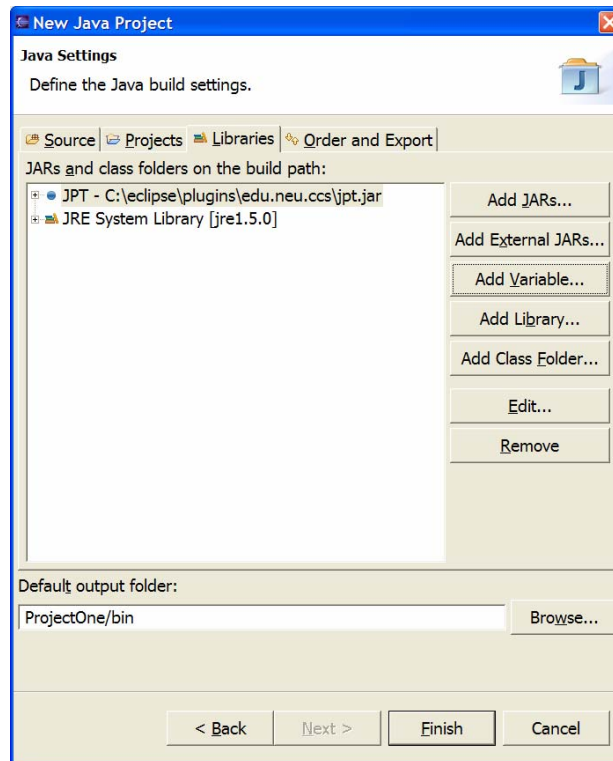


Click *Add Variable ...*



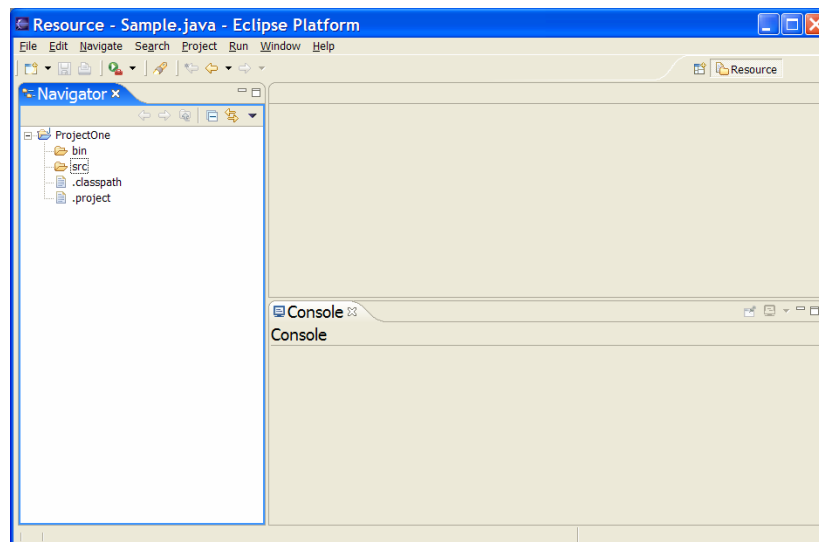
Click on the line with *JPT* to select it and then click *OK*.

You will see that the JPT library has been added.



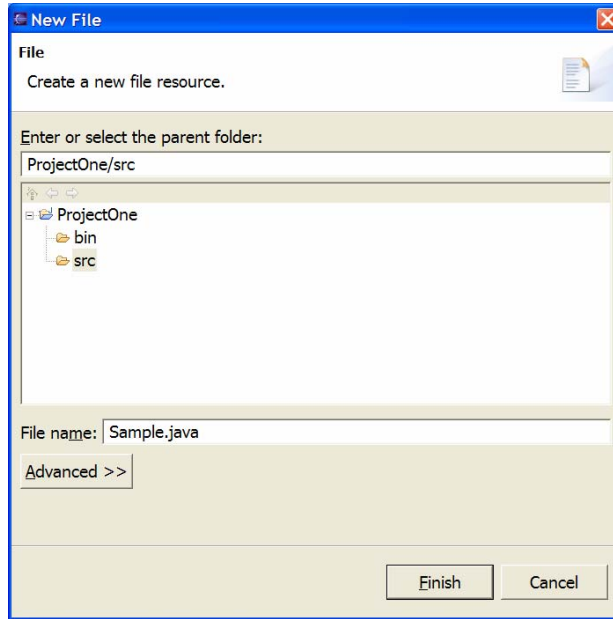
Click *Finish*.

The *Eclipse* navigator window now shows *ProjectOne*. The snapshot below shows the picture when the *ProjectOne* information has been expanded by clicking on the plus sign.

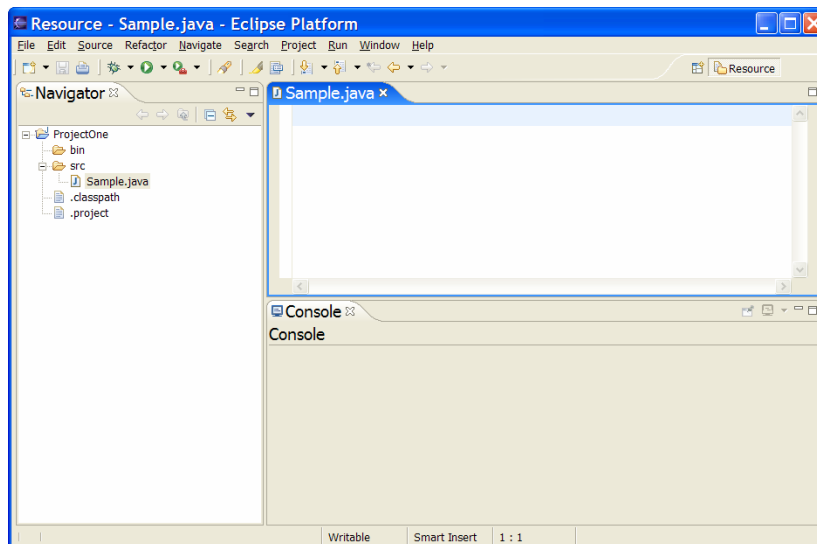


4. Creating a New Java File to Add to a Project

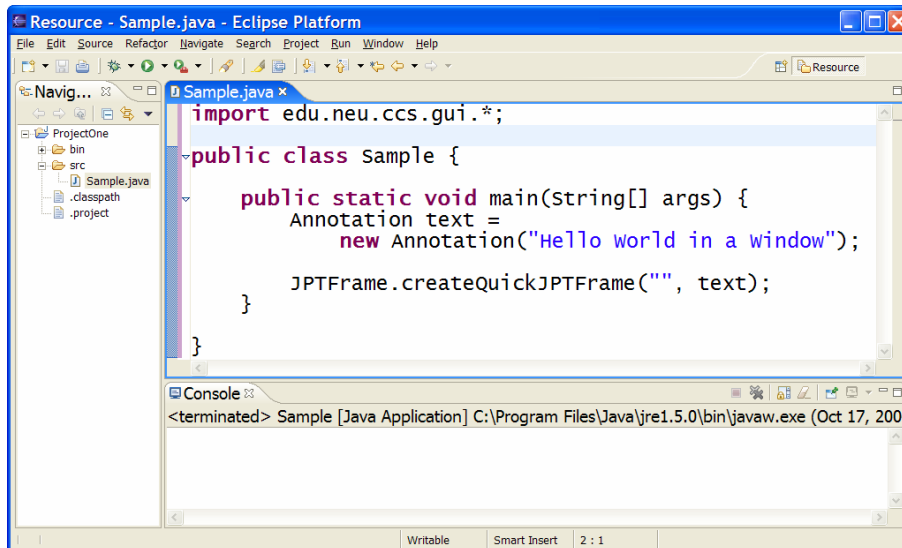
To create a new *Java* source file from scratch, click on the folder named *src* in the navigator panel and then select the menu item *File* → *New* → *File*.



Notice that we have named the file *Sample.java* in this dialog. Click *Finish*. You will then see a window for the new file to the right of the navigator panel.



Let us give some short *Java* code to type into *Sample.java*. You may consider it a variation of the classical *Hello World* program of the C programming language.



To execute this simple program, make sure the tab *Sample.java* is highlighted and then select the menu item *Run* → *Run as* → *Java Application*. If you have forgotten to save the source file, you will be prompted to do so. When this application runs, it opens the following window.



For convenience, if you want to use the above sample program, we provide its text.

```
import edu.neu.ccs.gui.*;
public class sample {
    public static void main(String[] args) {
        Annotation text =
            new Annotation("Hello world in a window");
        JPTFrame.createQuickJPTFrame("", text);
    }
}
```

The *import* line says that the program uses part of the *JPT library*, namely, the *gui* tools that include the *Annotation* class and the *JPTFrame* class. Since the class *Sample* will contain the *main* program, it must be declared as a *public class* and must contain a method of the form:

```
public static void main(String[] args) { .... }
```

This requirement for a *main* method has its origin in the C programming language and it annoys many modern *Java* programmers but that's just the way things are.

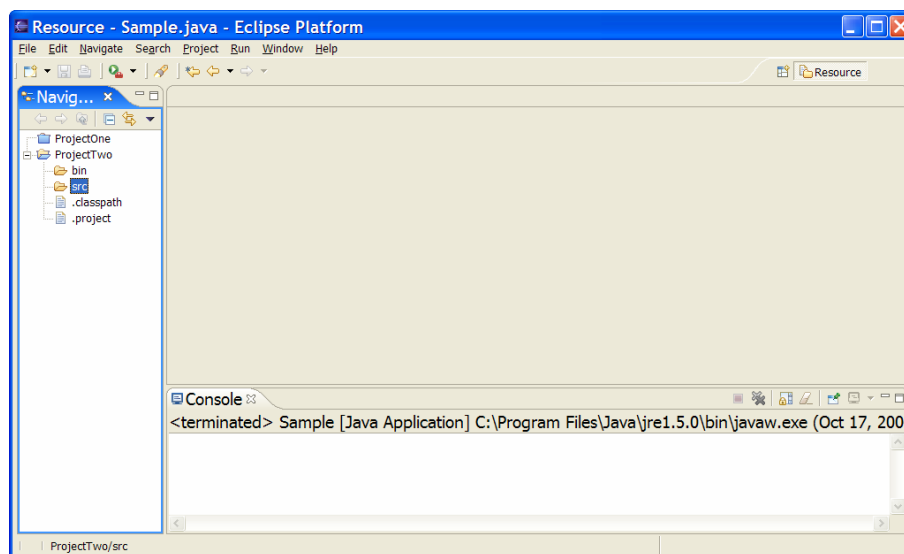
The body of *main* constructs a new *Annotation* object with the sample text and then uses the *JPTFrame* class to create and open a centered window with the text displayed.

5. Importing a Copy of an Existing File into a Project

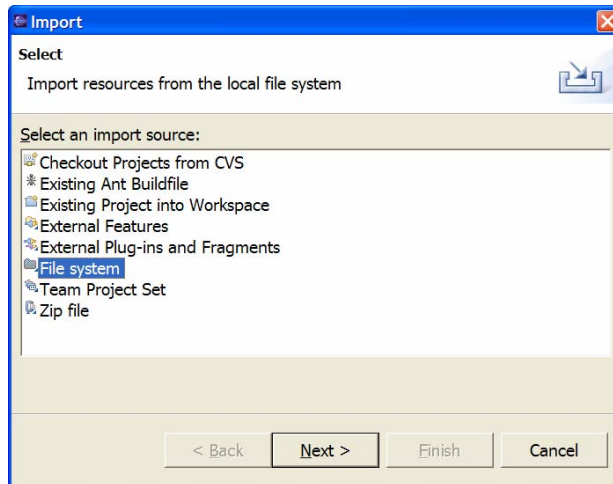
It is also possible to import existing files as part of the startup for a new project. We will do this using the *Java Power Framework* starter file *Methods.java* as the example. This file may be obtained from the web:

http://www.ccs.neu.edu/jpt/jpt_2_3/jpf_vanilla/Methods.java

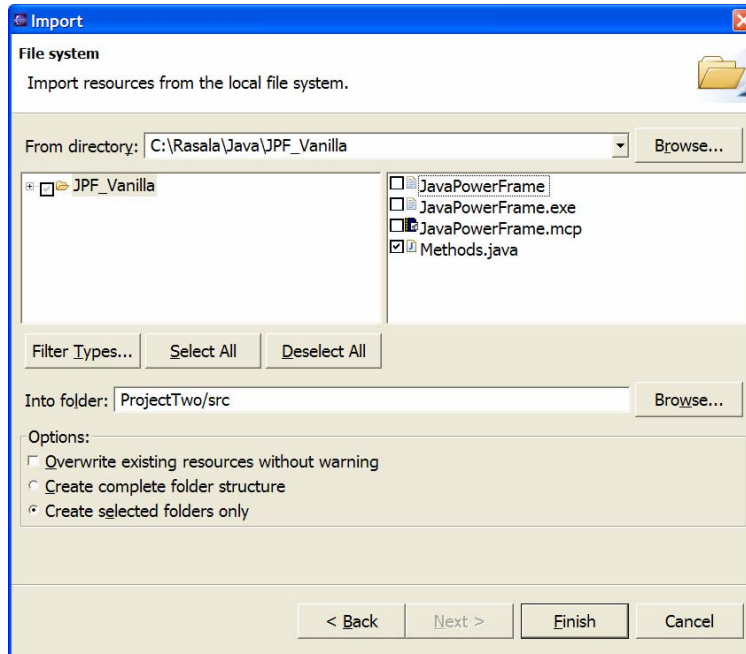
We will import this file into a new project called *ProjectTwo*. First click on *ProjectOne* in the navigator and then select the menu item *Project* → *Close Project*. This gets *ProjectOne* out of the way in the sense that *Eclipse* will consider it inactive. Then create *ProjectTwo* in the same way as we created *ProjectOne* above but do not create a new *Java* source file.



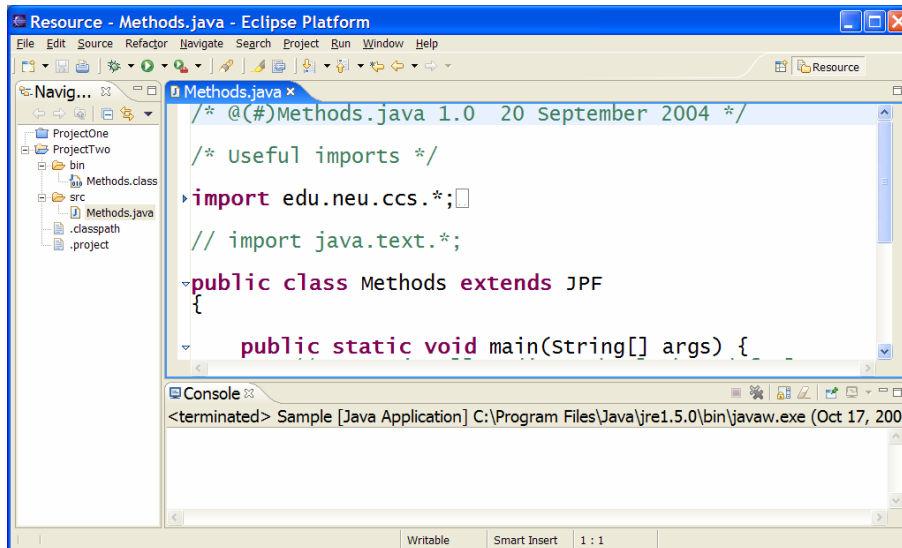
We now import the file *Methods.java*. by selecting the *src* folder and then using the menu item *File* → *Import...*. We get the following dialog.



Select *File System* and click *Next >*.



In the snapshot, I used the *Browse ...* button to locate a directory outside of the *Eclipse* directory tree where I had saved a copy of *Methods.java*. Since I wanted to import only this *Java* source, I clicked the check box in the right hand panel only for this particular file. I then clicked the *Finish* button. After opening the *bin* and *src* folders in the navigator and the file *Methods.java*, the *Eclipse* window now looks as follows.



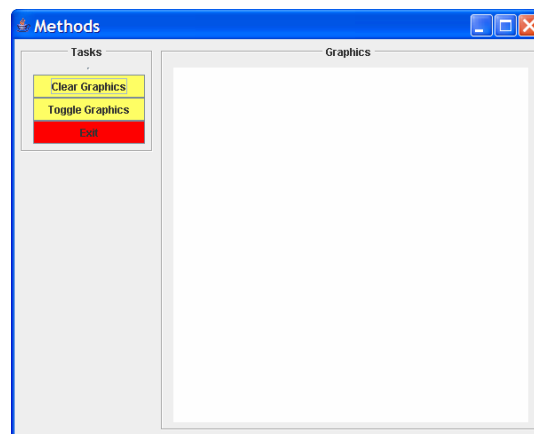
Notice that there is a *class* file in the *bin* directory. This is because *Eclipse* compiles a *Java* file immediately upon import. The file *Methods.java* has lots of *import* declarations to make it convenient to do experiments. Notice that *Eclipse* hides all but the first of these declarations but by clicking on the small triangle you may make these declarations visible if desired. Let us focus on the *class* declaration for *Methods*. Initially it has the form.

```
public class Methods extends JPF
{
    public static void main(String[] args) {
        // To optionally adjust the look and feel,
        // remove the comments from one of the two statements below.

        // LookAndFeelTools.showSelectLookAndFeelDialog();
        // LookAndFeelTools.adjustAllDefaultFontSizes(2);

        new Methods();
    }
}
```

If this program is run as is, all that is shown is a vanilla *Java Power Framework* GUI window.



We will make a few modifications to the main class above in order to show how easy it is to do something interesting in the *Java Power Framework*. The revised code is as follows.

```
public class Methods extends JPF
{
    public static void main(String[] args) {
        LookAndFeelTools.adjustAllDefaultFontSizes(10);

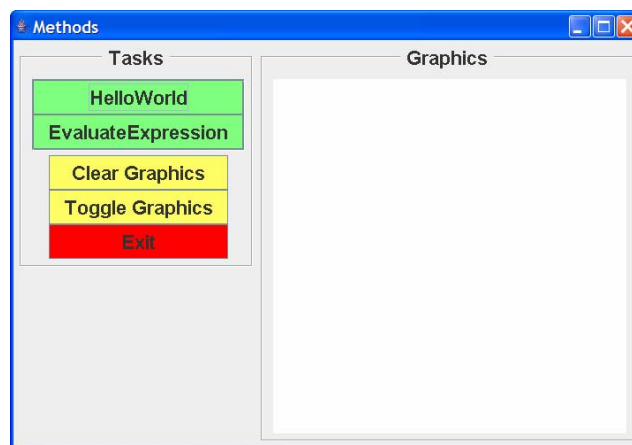
        new Methods();
    }

    public void HelloWorld() {
        Annotation text =
            new Annotation("Hello world in a window");

        JPTFrame.createQuickJPTFrame("", text);
    }

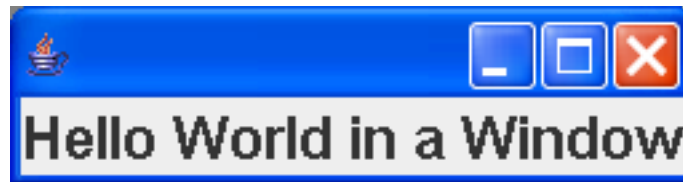
    public double EvaluateExpression(double x) {
        return x;
    }
}
```

Notice that we removed some commented lines from *main* and have activated the command to increase all default *Java* font sizes by 10. This will make the fonts in the *Java* windows easier to read. We have also added 2 *public* methods named *HelloWorld* and *EvaluateExpression*. These methods automatically give rise to buttons in the *Java Power Framework* GUI. As you can see, the *HelloWorld* method uses the same code as in the earlier sample. The method to evaluate an expression is even simpler and its functionality arises from the fact that GUI input in *Java Power Tools* automatically evaluates expressions. Let us show what happens when this program is run.



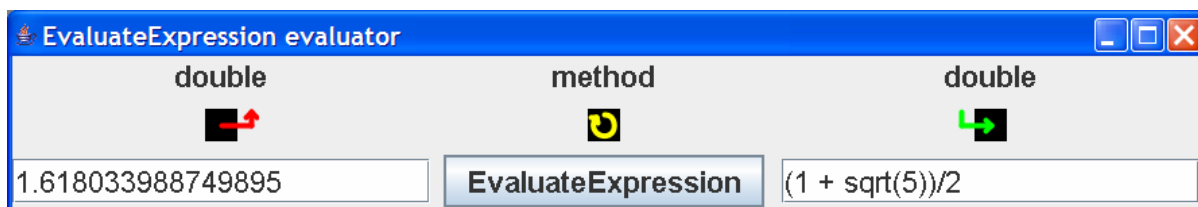
Notice the two buttons *HelloWorld* and *EvaluateExpression* that have appeared in the GUI simply due to the fact that we have introduced the two small *public* methods into the *Methods* class declaration. This is due to the power of the framework.

If we click on the *HelloWorld* button, we obtain the following window.



Notice the significant increase in the font size over the original version due to the change in the default font sizes.

If we click on the *EvaluateExpression* button, we obtain the following window which is shown here at 50% reduction in size.



The user may type in an expression at the right, for example, the famous *golden ratio* that is defined as $(1 + \sqrt{5})/2$. The button *EvaluateExpression* evaluates this to 1.618033988749895.

These examples show only the tip of the iceberg of what can be done with *Java Power Tools* in general and the *Java Power Framework* in particular. For more information, see the JPT web site:

<http://www.ccs.neu.edu/jpt/>

especially:

http://www.ccs.neu.edu/jpt/jpt_2_3/

6. Importing Data Files

Data files may also be imported into *Eclipse*. The key thing to keep in mind is that you should click on the project name, for example, *ProjectTwo*, before executing the *File* → *Import...* command when importing data files.