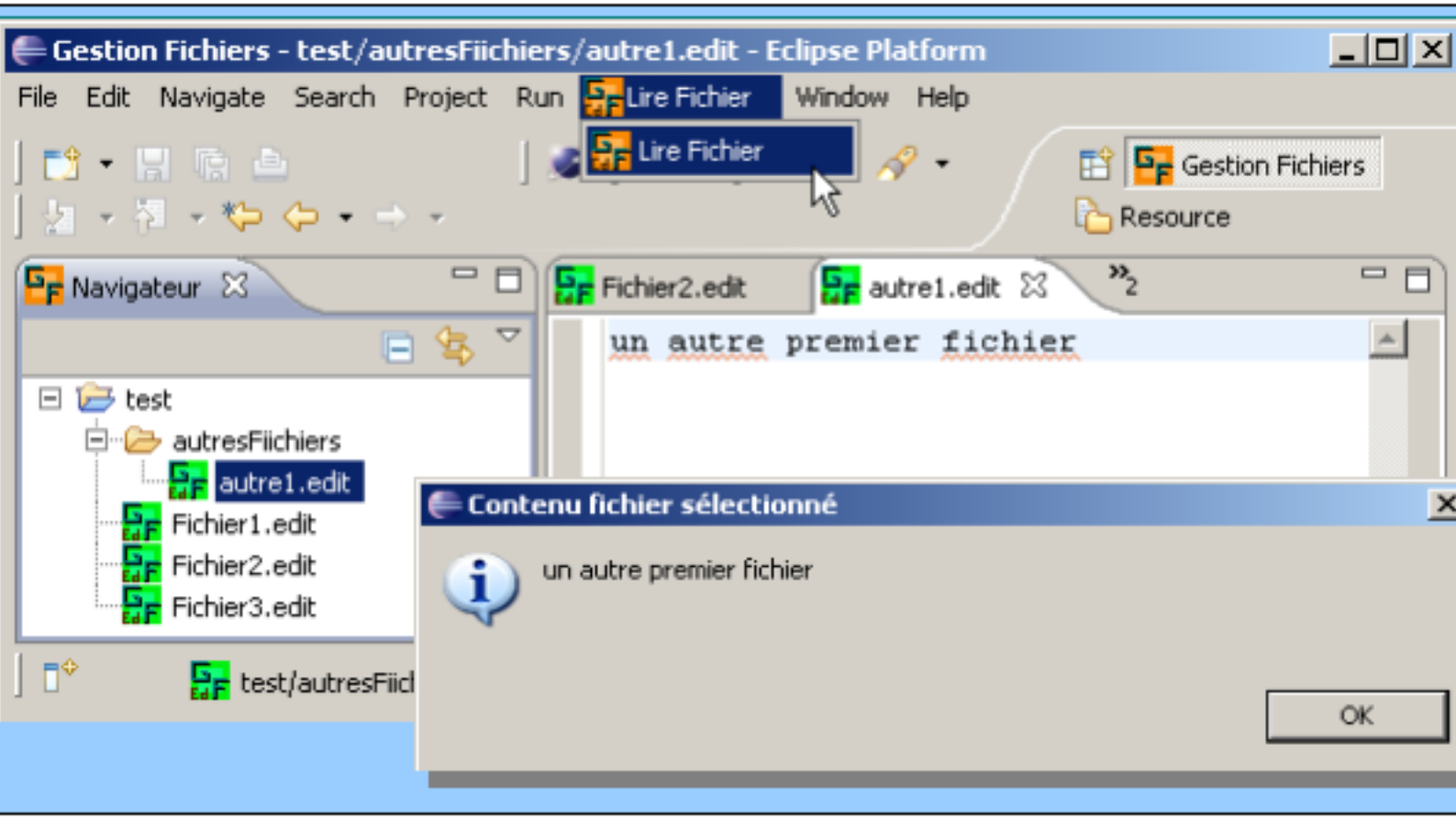
Plug-ins - Gestion edition de fichiers



par Serge Bachmann

Date de publication : 27/12/2011

Dernière mise à jour : 27/12/2011

Le plugin gére un ensemble de fichiers.

Les repertoires et fichiers sont stockés dans le workspace du plugin.

La navigation est assurée par une vue "common navigator".

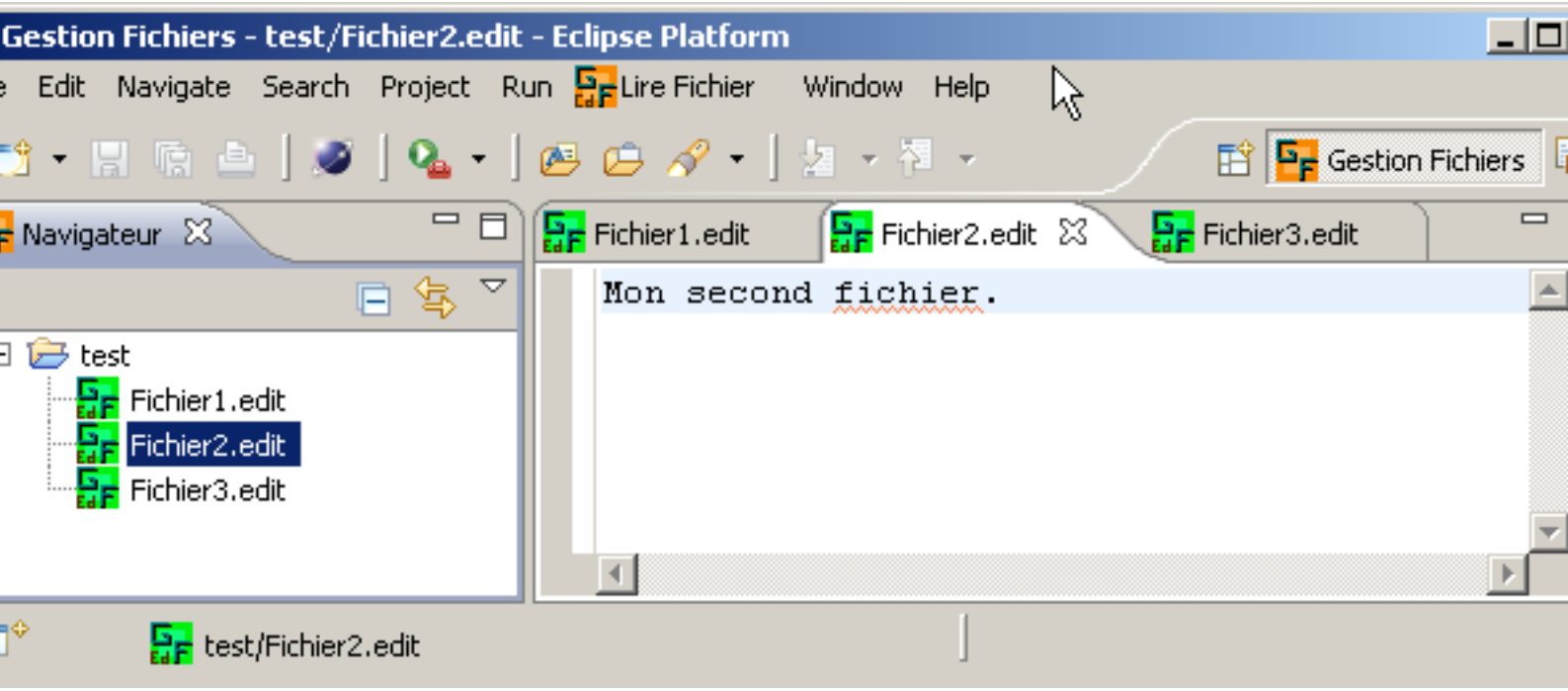
Les fichiers (*.edit) sont édités par notre éditeur.

Une commande permet d'accéder au contenu du fichier sélectionné et de le visualiser dans un message.

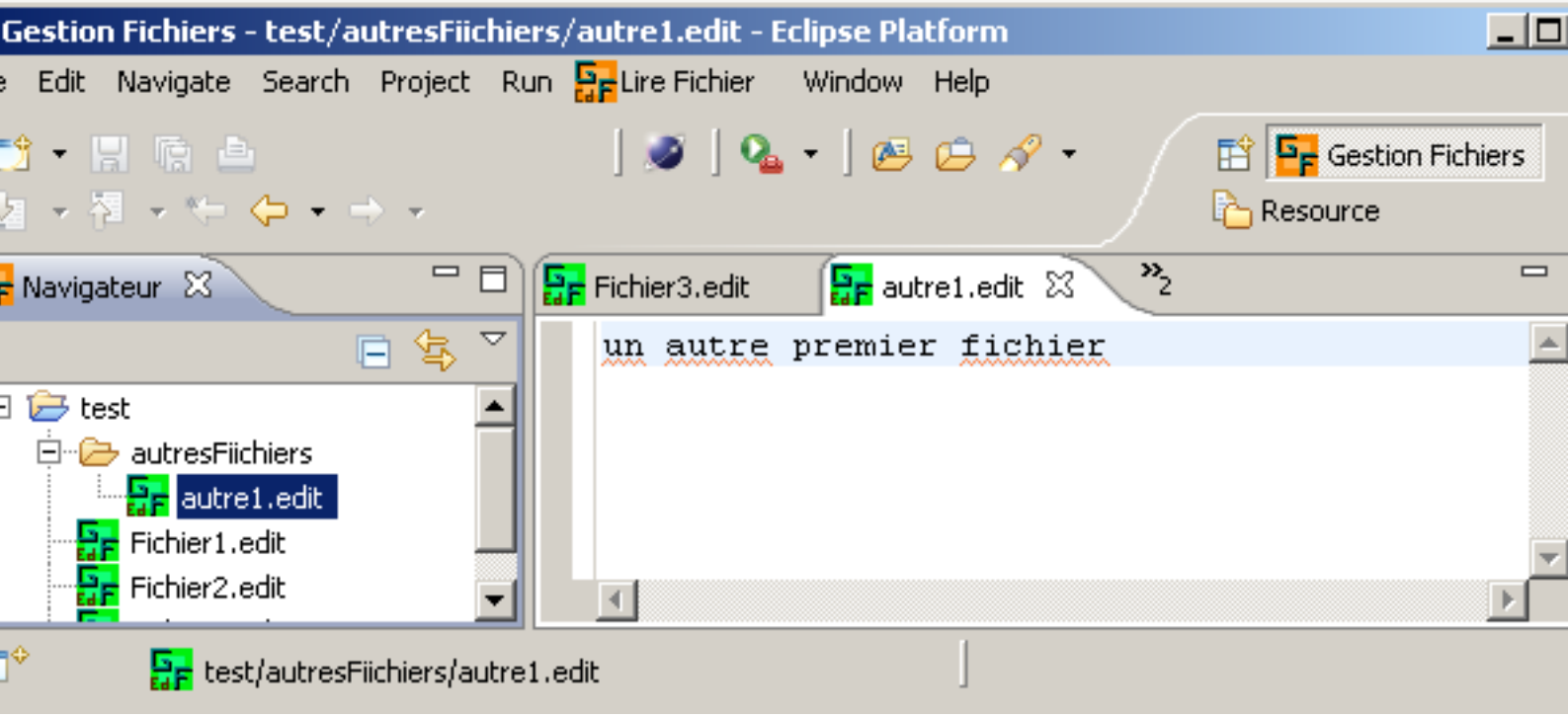
1 - Introduction

1-A - Spécification

Une vue « Navigateur » et un « éditeur de fichiers » sont mis en place dans la perspective « Gestion Fichiers ». L'éditeur est activé pour tout fichier de nom : « *.edit ».



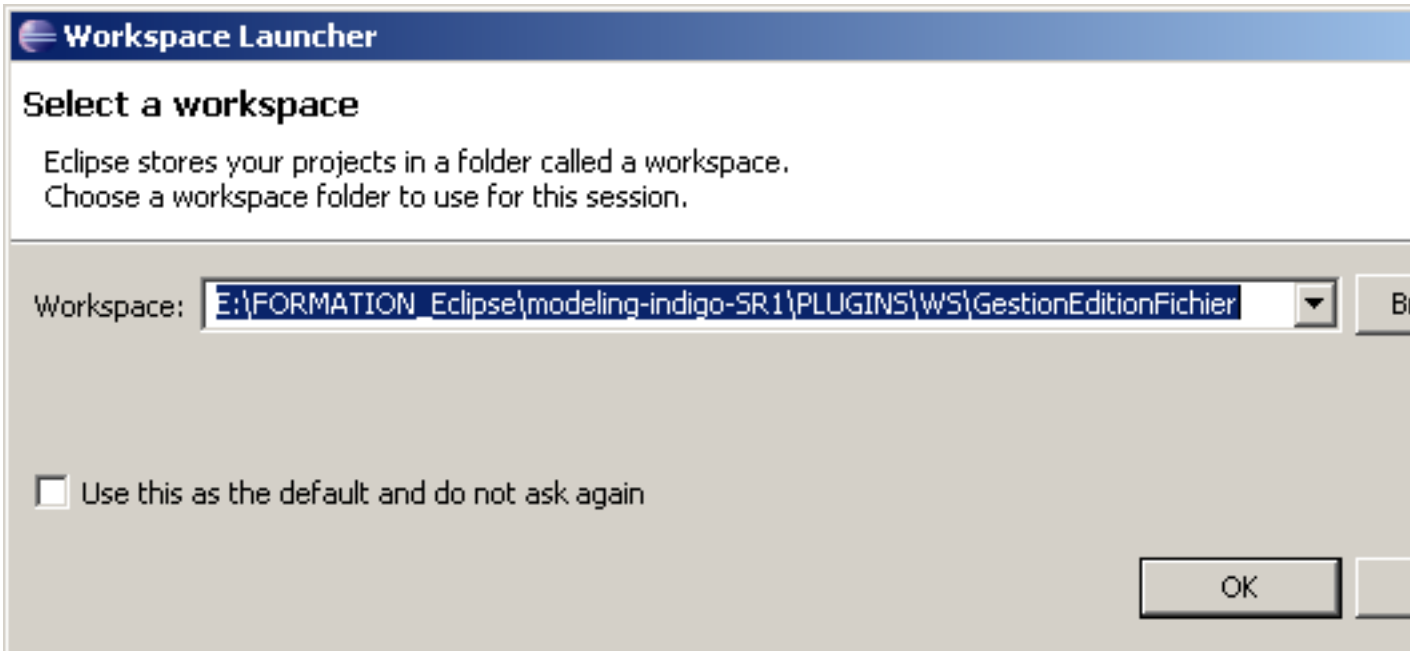
Le navigateur gère une arborescence de fichiers :



1-B - Lancement de la plateforme Eclipse

Double cliquer : eclipse.exe ou le raccourci vers cet exécutable si vous l'avez créé dans le répertoire destiné à recevoir les « workspaces ». La plate-forme « Eclipse » est lancée:

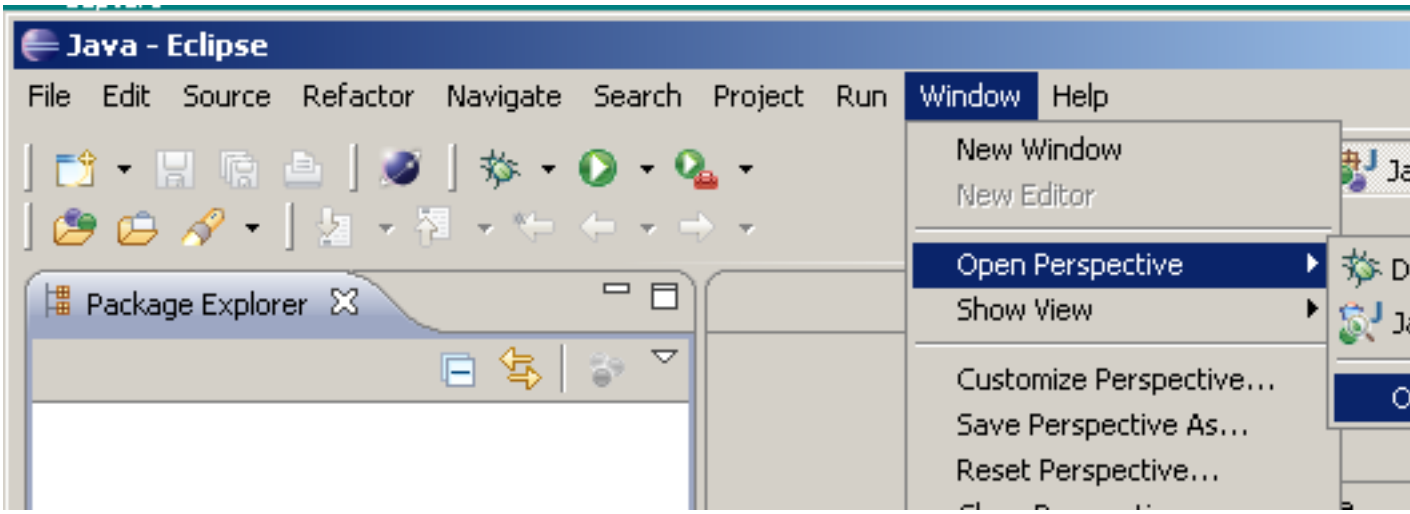
On sélectionne le « workspace »:



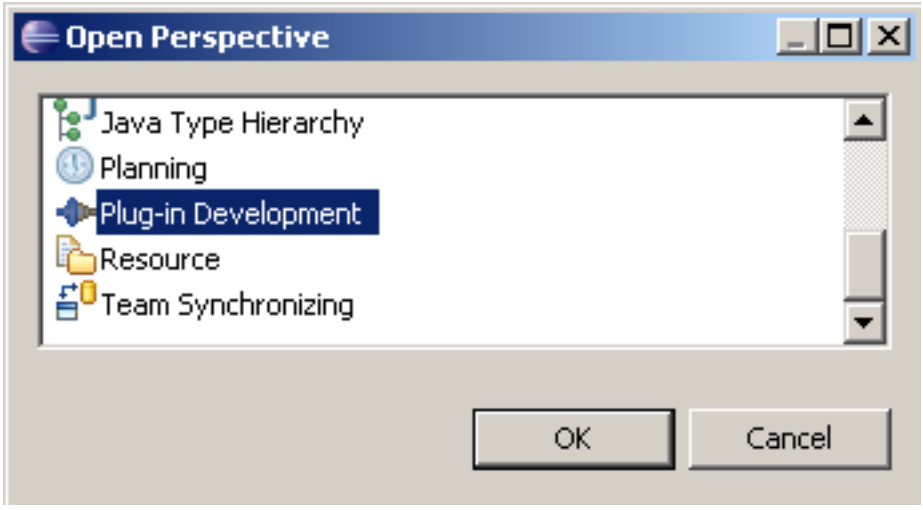
Cliquer « OK », Fermer la fenêtre « Welcome »

1-C - Passage en perspective « Plug-in Development »

Faire : « Window - Open Perspective > Other... » :

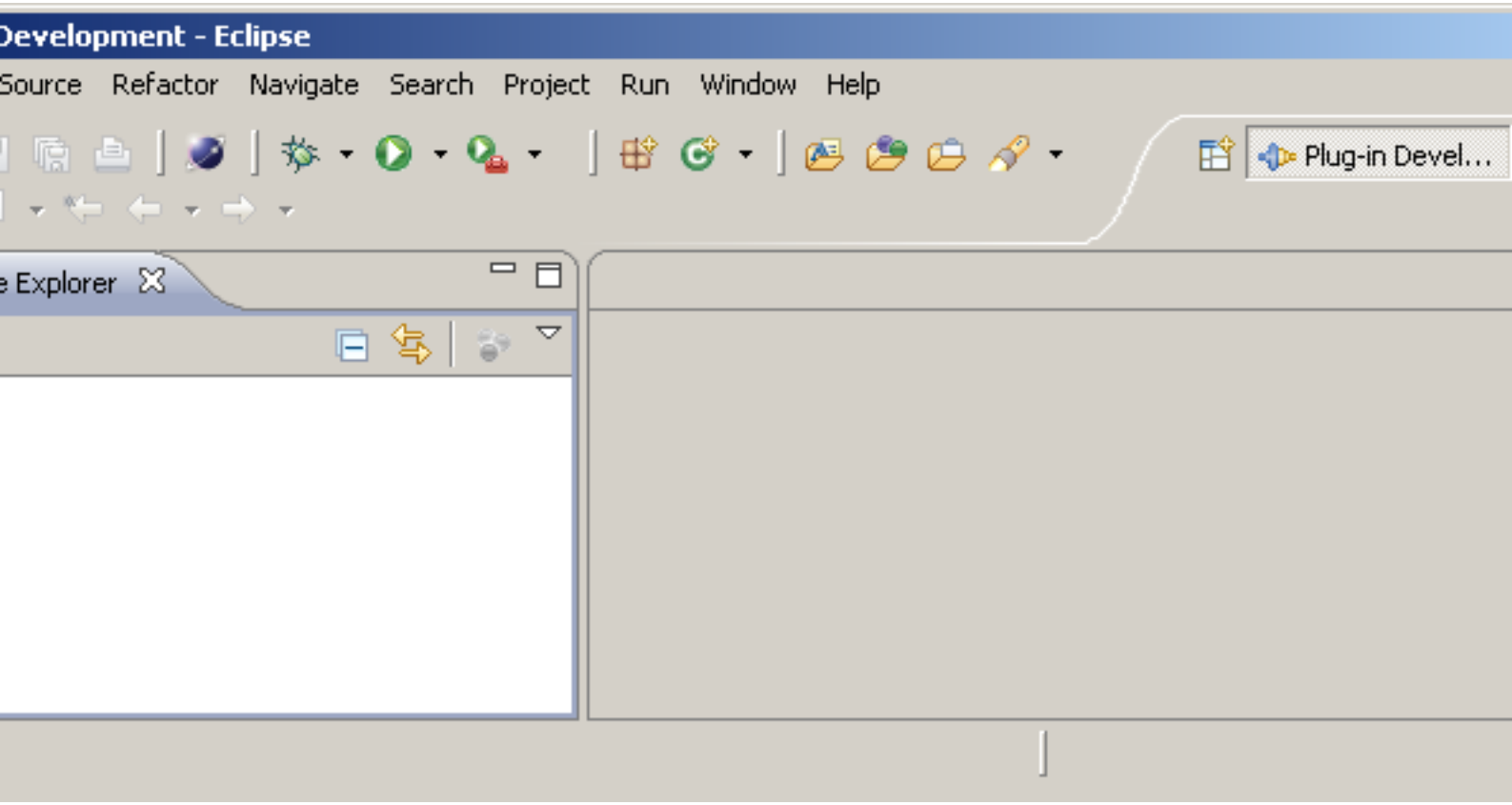


Dans « Open Perspective » sélectionner « Plug-in Development »:



Cliquer « OK »

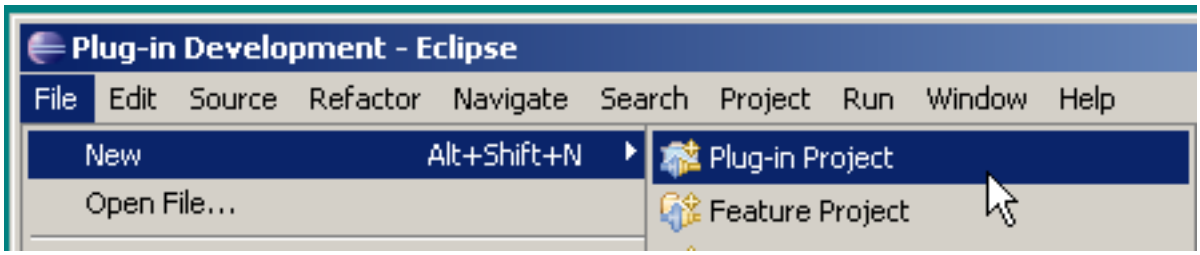
Pour simplifier le « Workbench » on ferme les vues: Outline, Task List, Error Log, Problems, Tasks, Plug-ins). On obtient:



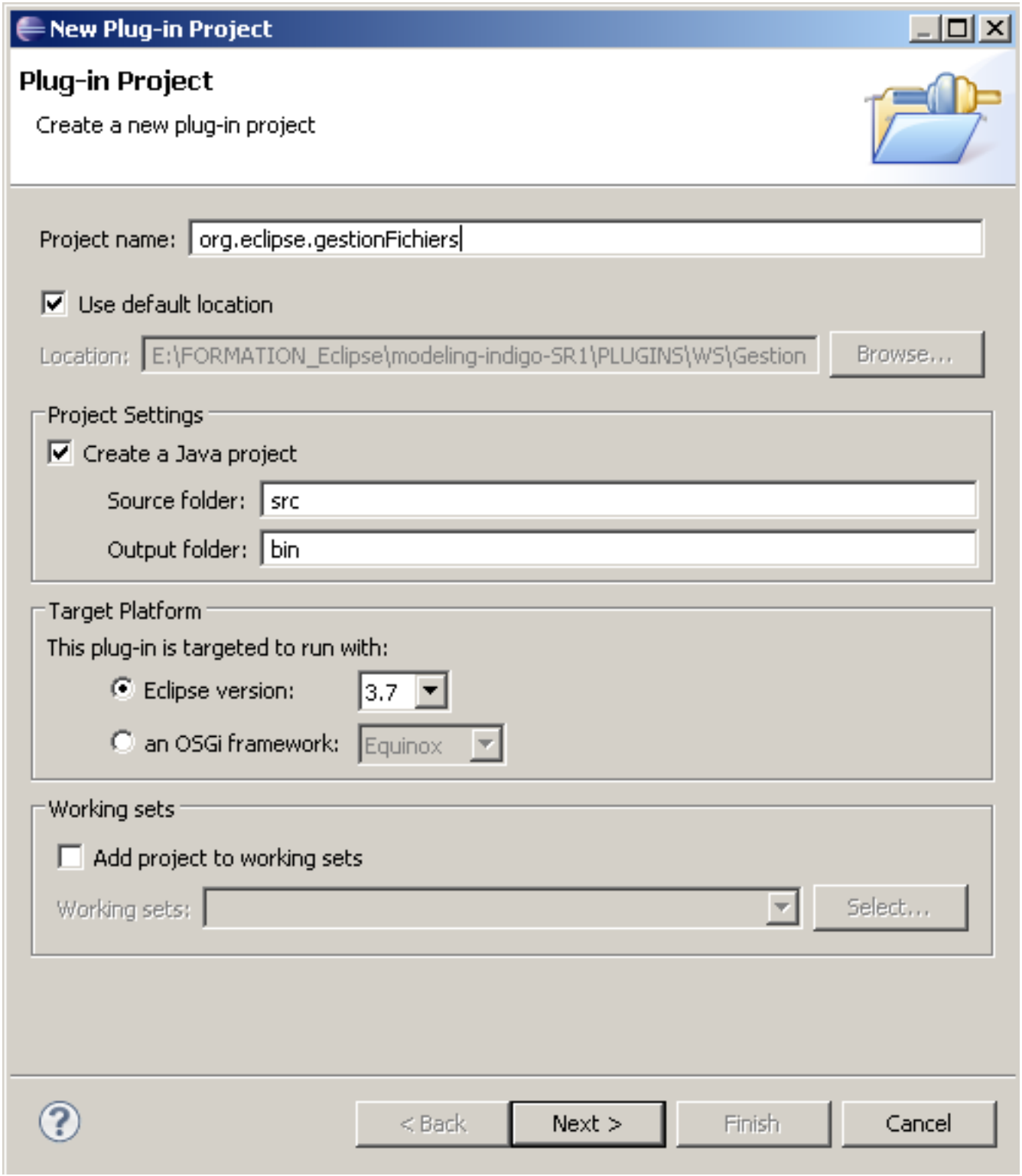
2 - Initialisation du Plug-in

2-A - Création du projet

Sélectionner « File > New > Plug-in Project »

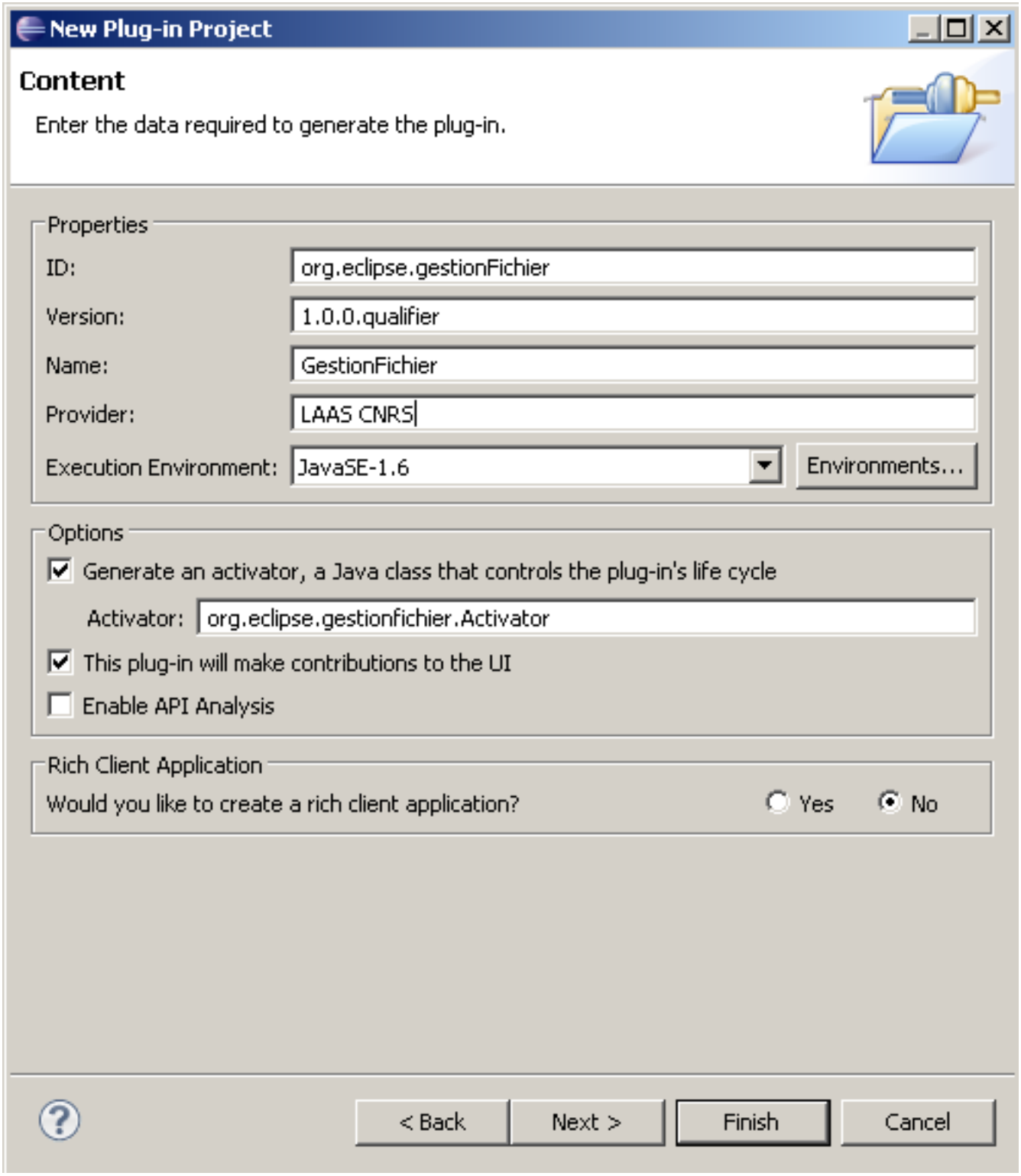


Dans « New Plug-in Project - **Plug-in Project** » nommer le projet :



Cliquer « Next > ».

Dans « New Plug-in Project - **Content** » donner le nom du « provider » :



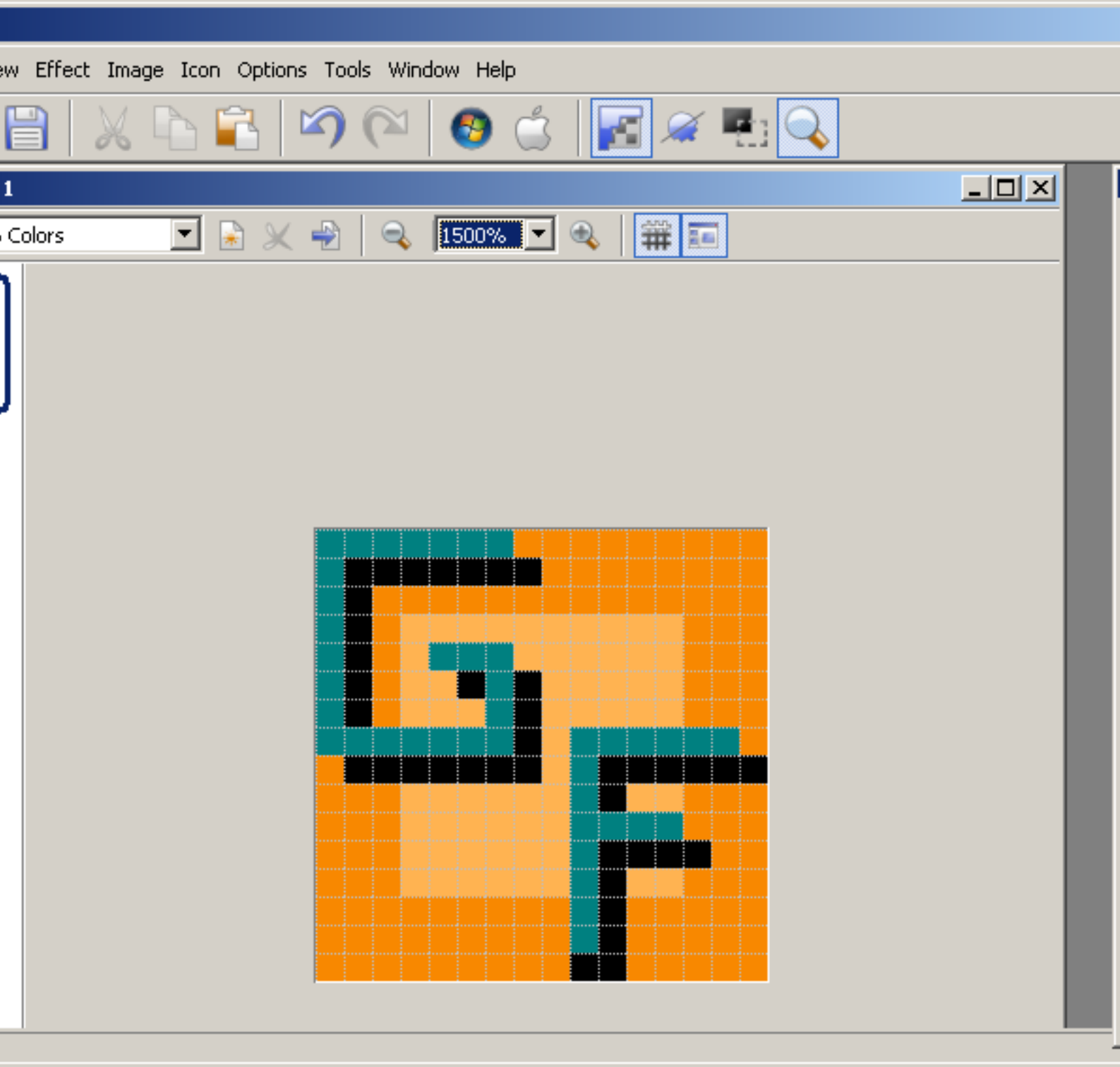
Cliquer « Finish ». Faire une sauvegarde.

3 - Construction du Plug-in

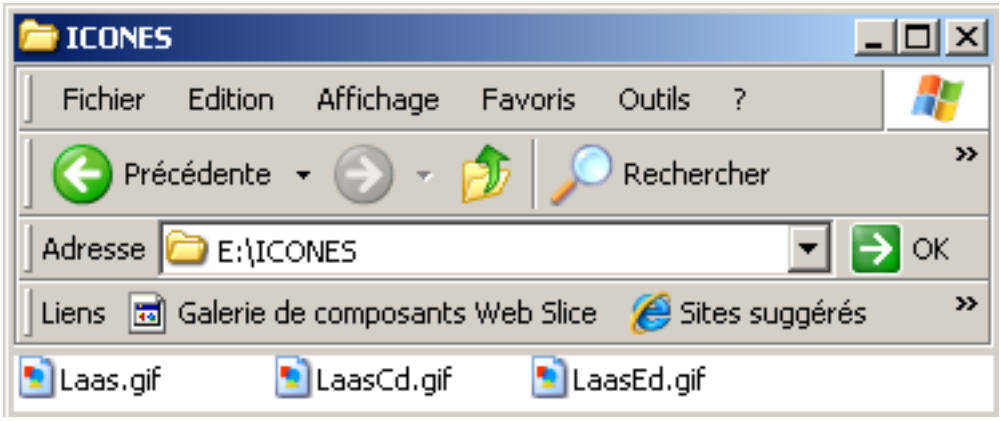
3-A - Création des icônes

Dans le « Package explorer » faire un clic droit sur « org.eclipse.gestionFichiers » pour créer un « folder » (commande : « New > Folder ») le nommer « icons »

Éditer vos icônes (par exemple avec IcoFX) :

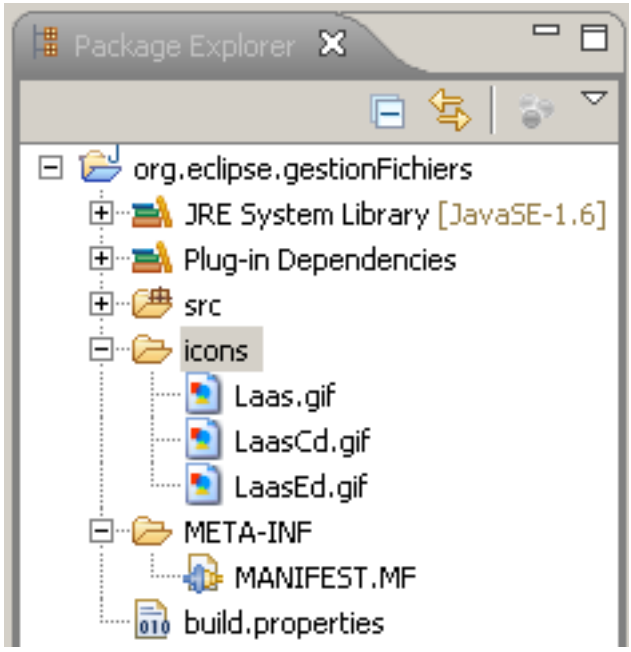


Sauvegarder vos icônes dans un répertoire temporaire :



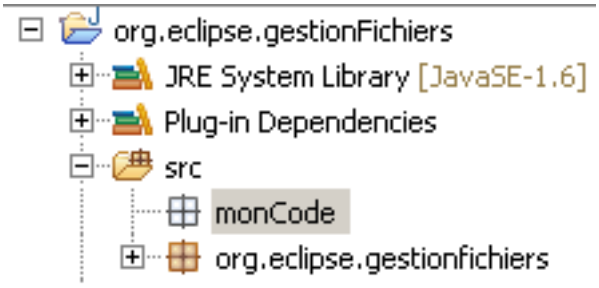
Recopier les icônes sous « icons » par un simple « copier - Paste).

On obtient :



3-B - Création « package » pour recevoir le code de l'application

Dans « src » ajouter le package « monCode » :

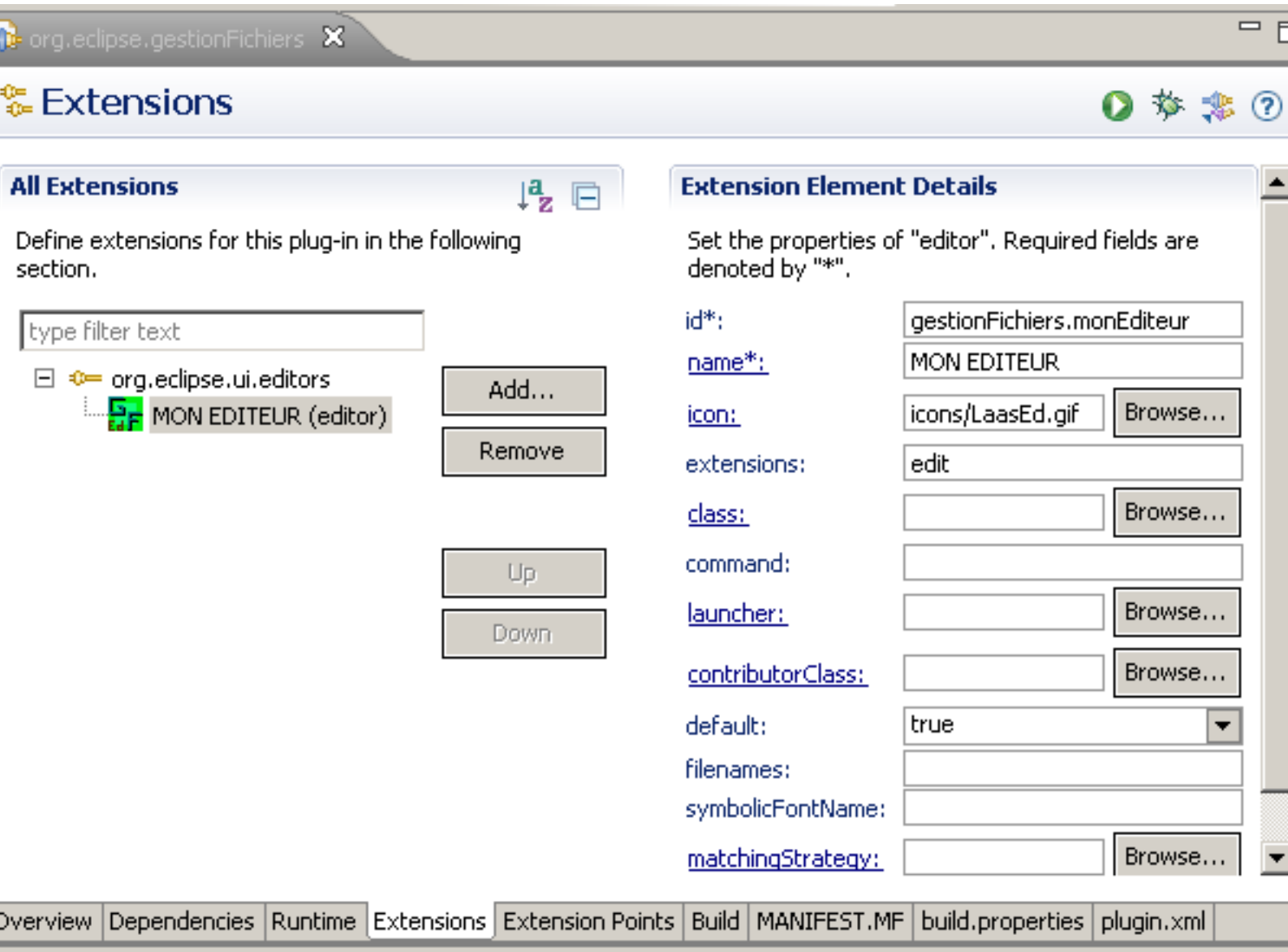


3-C - Création éditeur

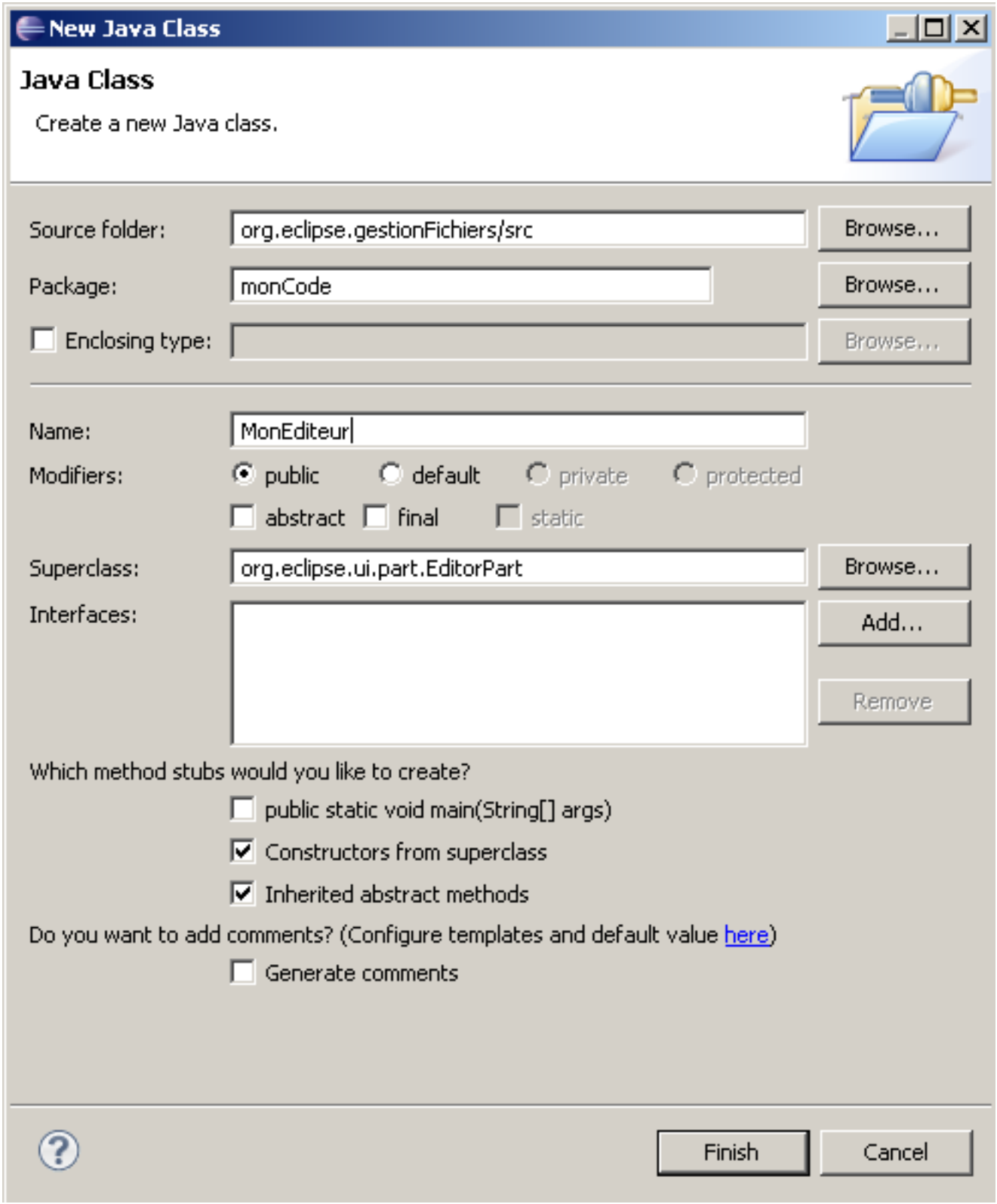
Dans le « Plug-in Manifest Editor » onglet « Extensions » faire « Add.. ». Dans « New Extension - **Extension Point Selection** » sélectionner l'extension « org.eclipse.ui.editors » faire « Finish », on met à jour l'éditeur:

(Si une erreur apparaît dans le fichier « MANIFEST.MF faire une sauvegarde).

L'éditeur est défini par:



Cliquer « class: », initialiser la classe :



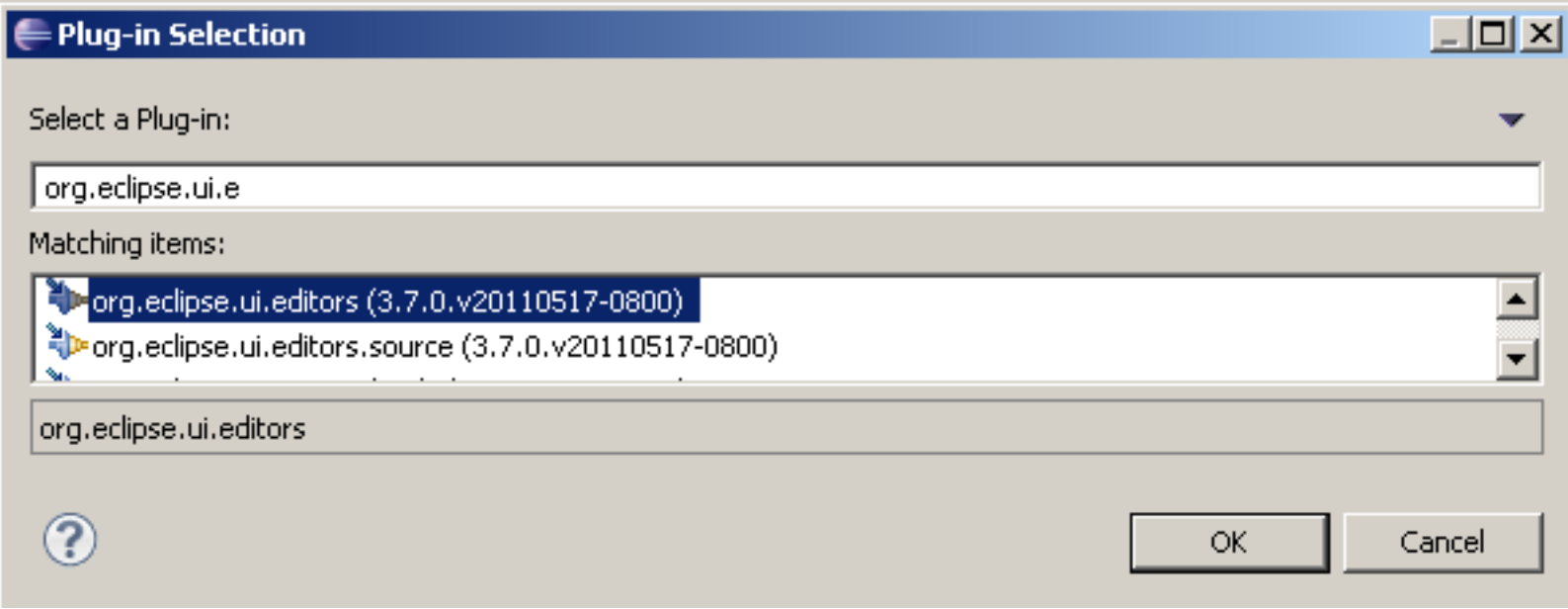
Cliquer « Finish ». Editer la classe « MonEditeur »:

```
package monCode;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.editors.text.TextEditor;
```

```
import org.eclipse.jface.dialogs.MessageDialog; // Pour la partie en commentaires


publicclass MonEditeur extends TextEditor {

    public IWorkbenchWindow[] windows;

    public MonEditeur() {
        super();
        windows = org.eclipse.gestionfichiers.Activator.getDefault().
            getWorkbench().getWorkbenchWindows();
    }



    publicvoid doSave(IProgressMonitor monitor) {
        super.doSave(monitor);
        /*
        // Quelques information obtenueslorsdelasauvegarde
        String message = new String("================================");
        message = message +"\nRegisteredName: "+ this.getSite().getRegisteredName();
        message = message +"\nId: "+ this.getSite().getId();
        message = message +"\nTitle: "+ this.getSite().getPart().getTitle();
        message = message +"\nTitleToolTip: "+ this.getSite().getPart().getTitleToolTip();
        message = message +"\nPluginId: "+ this.getSite().getPluginId();
        message = message +"\nLabel: "+ this.getSite().getPage().getLabel();
        message = message +"\nWindows nb: "+ windows.length;
        MessageDialog.openInformation(windows[0].getShell(),
                "Quelques informations ", message);
        */

    }

}
```

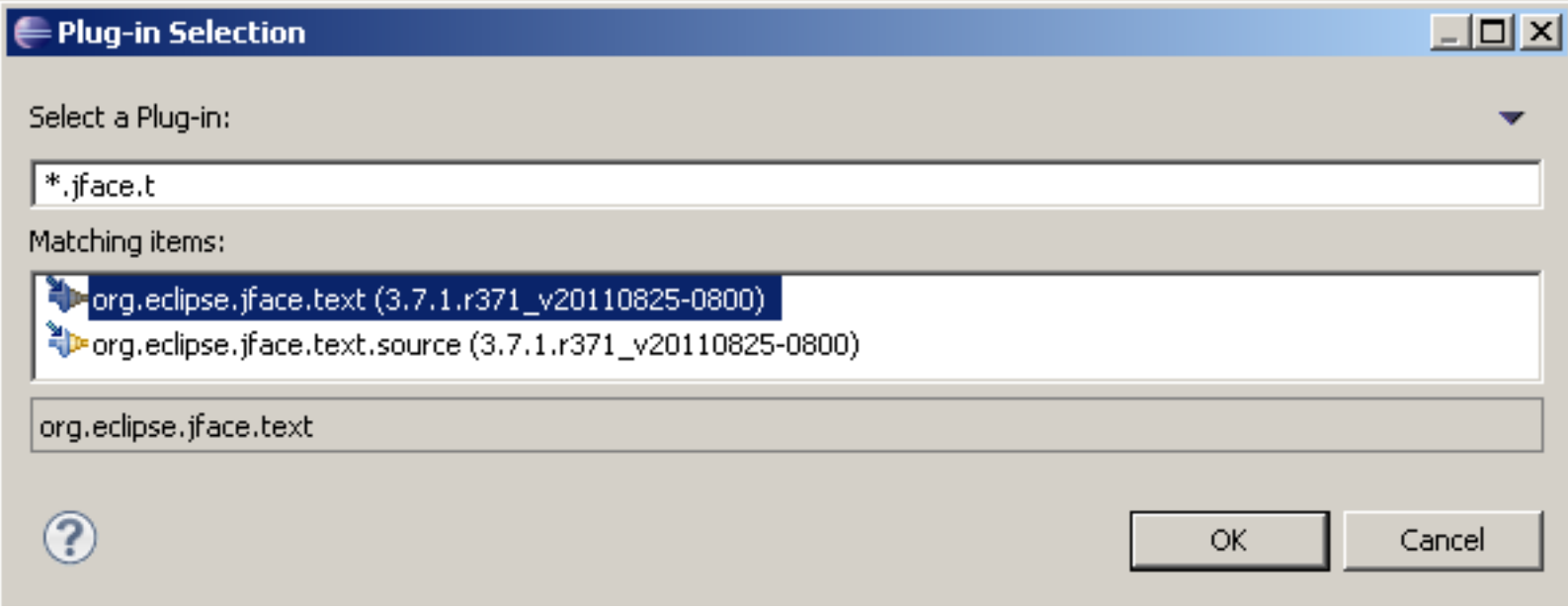
Faire une sauvegarde. Il y a des erreurs. Il faut ajouter des dépendances. Aller dans le « Plug-in Manifest Editor » onglet « Dependencies » et ajouter les dépendances (dans « Required Plug-ins » faire « Add... ») ajouter les plugins:

- « org.eclipse.ui.editors »



Cliquer « OK ».

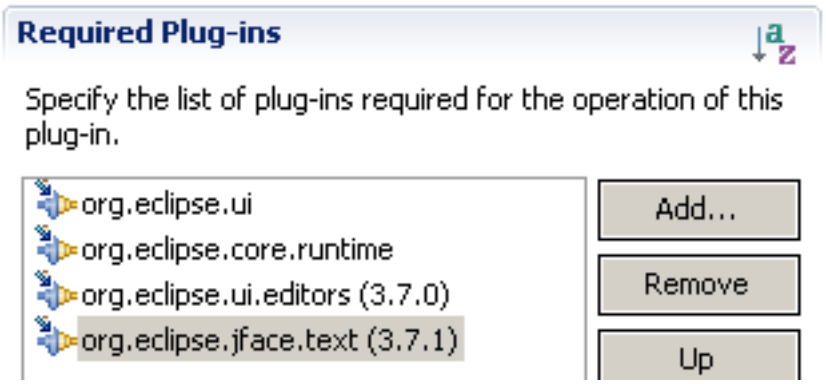
- « org.eclipse.jface.text »:



Cliquer « OK »

Faire une sauvegarde.

Les dépendances sont:

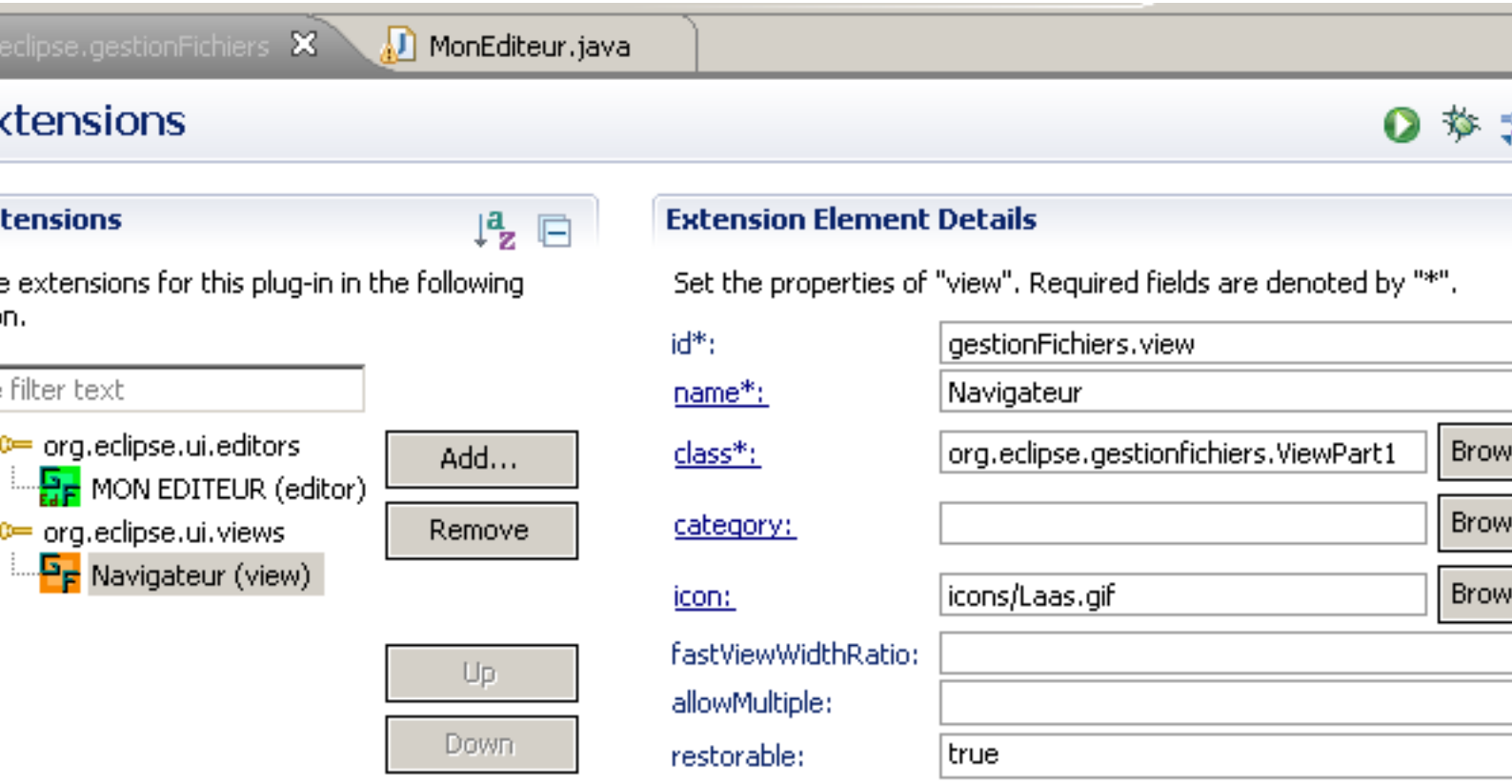


Il n'y a plus d'erreur dans « monCode.MonEditeur ».

3-D - Création de la vue

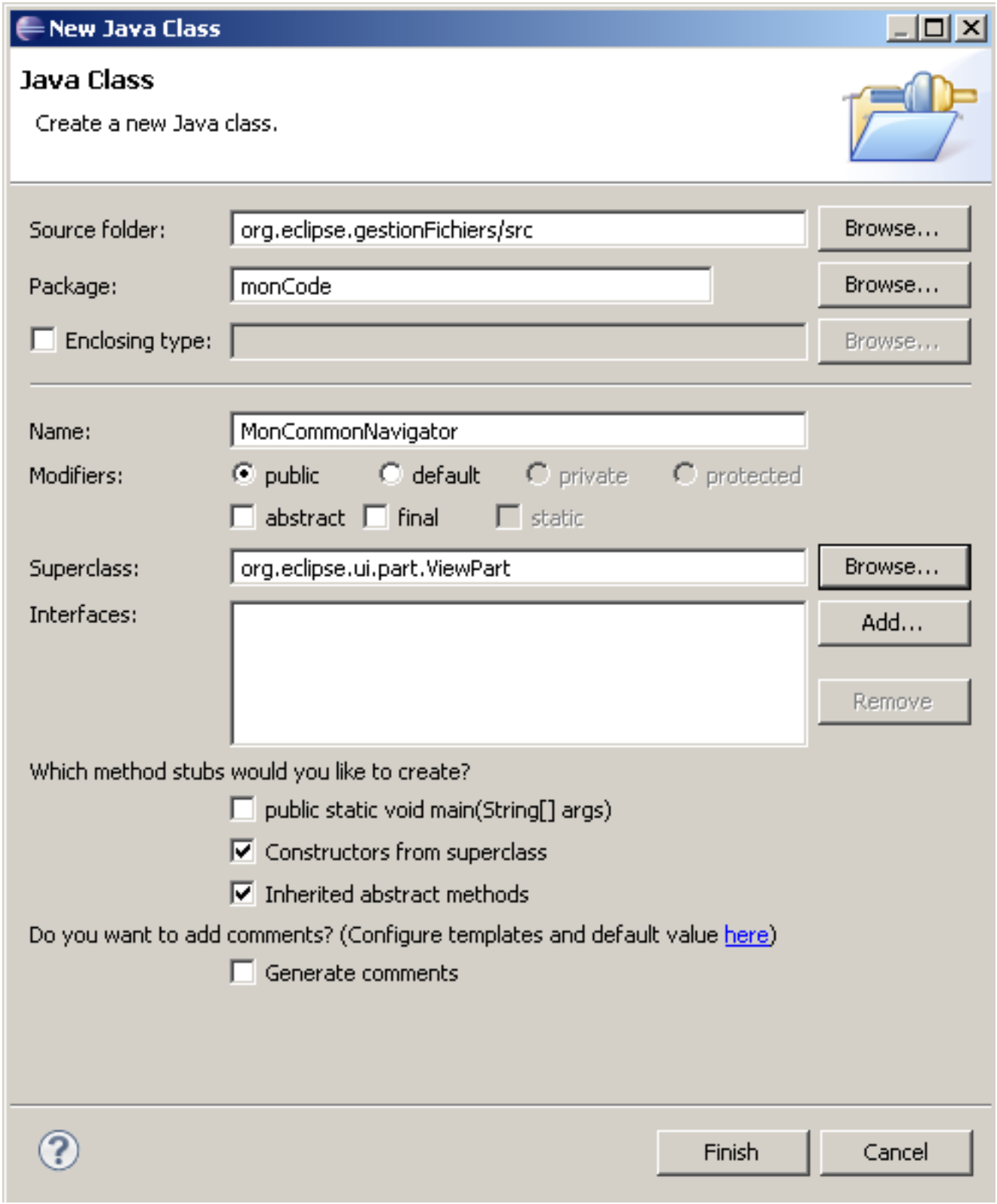
Dans le « Plug-in Manifest Editor » onglet « Extensions » faire « Add.. » sélectionner l'extension « org.eclipse.ui.views » faire « Finish ».

Sous « org.eclipse.ui.views » créer une vue, la mettre à jour :



Cliquer « class: ».

Initialiser la classe :

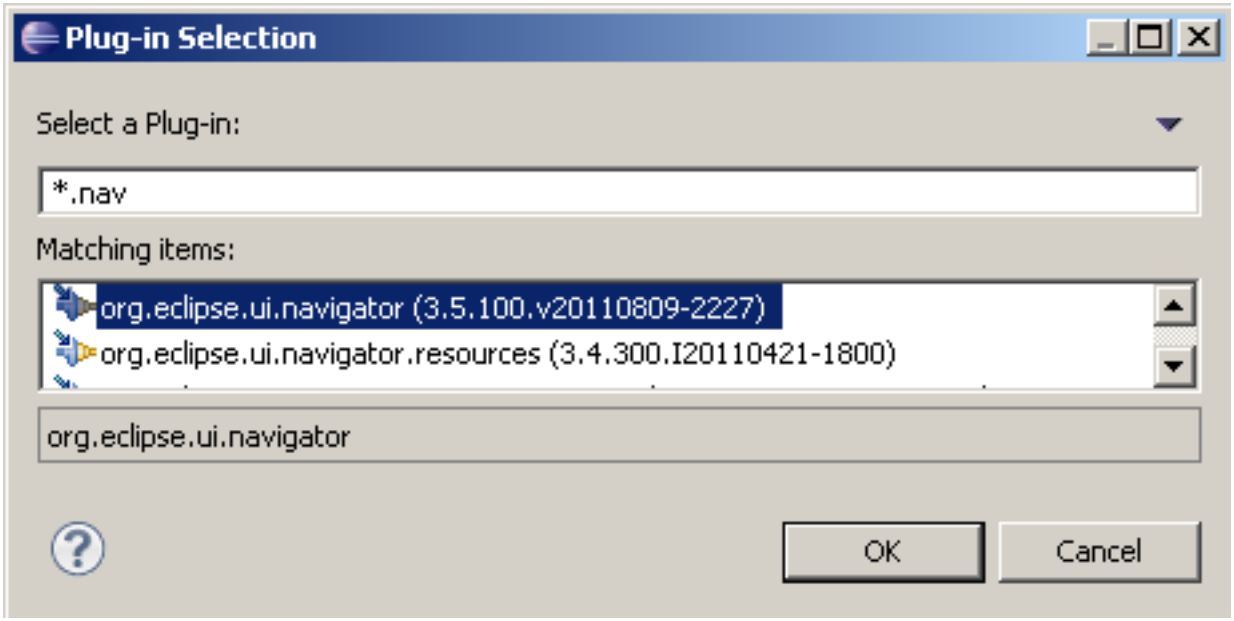


Cliquer « Finish ». Éditer la classe « monCode.MonCommonNavigator »

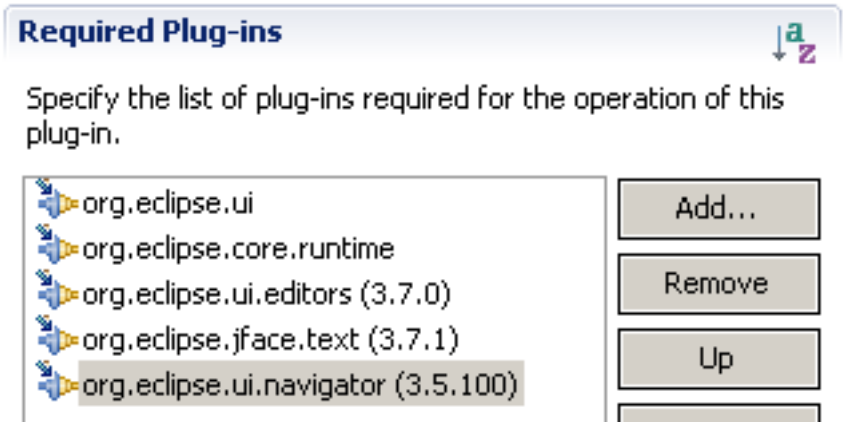
```
package monCode;

import java.util.regex.Pattern;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.SelectionChangedEvent;
```

```
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.swt.widgets.Composite;
importorg.eclipse.ui.navigator.CommonNavigator;

publicclass MonCommonNavigator extendsCommonNavigator {

    static String fullPath;

    static String[] selectionFichier;

    public MonCommonNavigator() {
        super();
    }

    publicvoid createPartControl(Composite parent){
        super.createPartControl(parent);
        System.out.println ("CREATE PART CONTROL");
        Viewer viewer = this.getCommonViewer();

        viewer.addSelectionChangedListener(
                new ISelectionChangedListener(){
                    publicvoid selectionChanged(SelectionChangedEvent event){
                        IStructuredSelection selection =
                            (IStructuredSelection) event.getSelection();
                        fullPath = new String(selection.getFirstElement().toString());
                        System.out.println("Selection " + fullPath);
                        selectionFichier = decomposition(fullPath);
                    }
                }

        );

    }

    publicvoid setFocus() {
    }

    /**
     * result[0] nedoit pas etreprisencompte
     * ladernierevaleurestlenomdufichier
     * Lesvaleursintermediairessontdes "folders"
     */
    private String[] decomposition(String fullPath){
        return Pattern.compile("/").split(fullPath);
    }
}
```

Il y a des erreurs. Il faut ajouter une dépendance. Aller dans le « Plug-in Manifest Editor » onglet « Dependencies ». Dans « Required Plug-ins » faire « Add... » pour ajouter le plugin: « org.eclipse.ui.navigator»:



Les dépendances sont:



Faire une sauvegarde. Il n'y a plus d'erreur.

3-E - Création du « Navigator viewer »

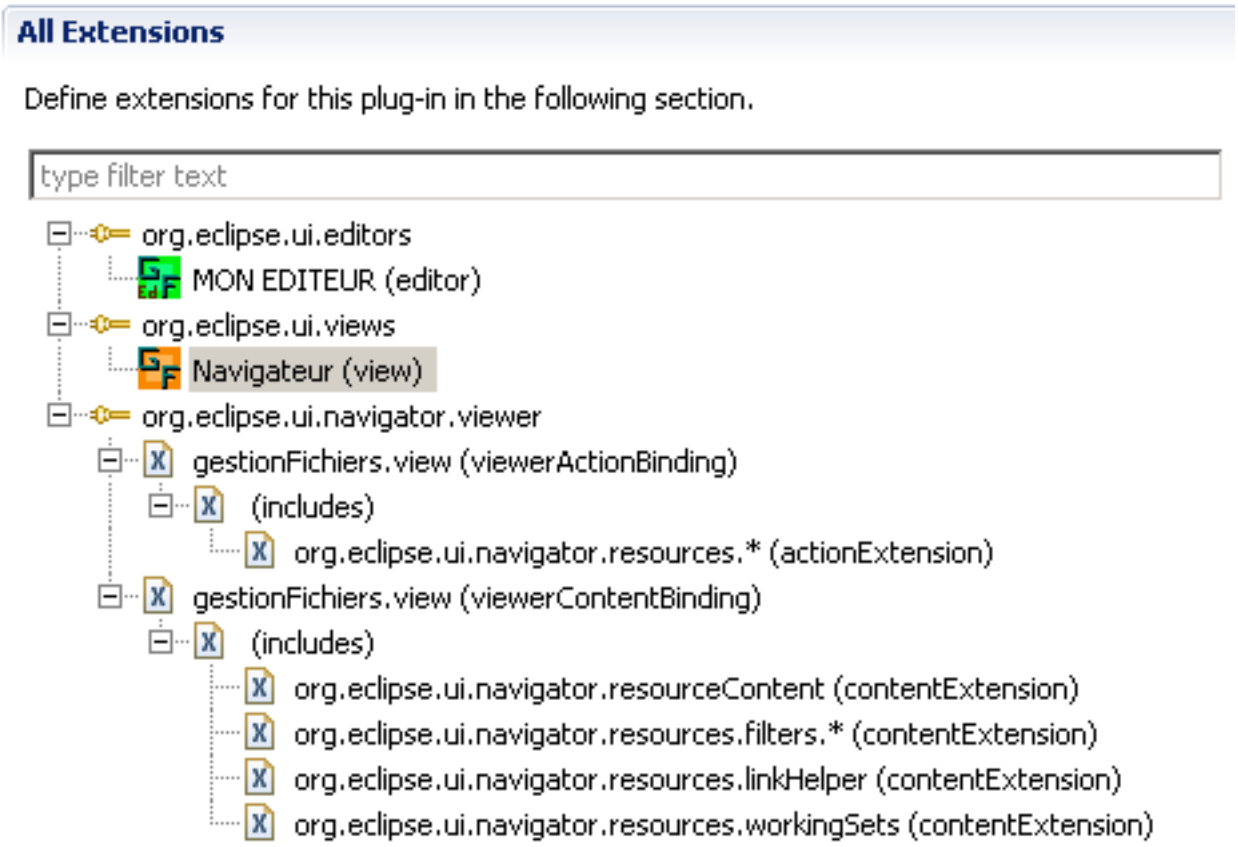
Afin de faciliter cette création laborieuse on recopie dans « plugin.xml » la définition de l'extension « org.eclipse.ui.navigator.viewer » :

Le fichier « plugin.xml » devient :

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.editors">
      <editor
            class="monCode.MonEditeur"
            default="true"
            extensions="edit"
            icon="icons/LaasEd.gif"
            id="gestionFichiers.monEditeur"
            name="MON EDITEUR">
      </editor>
```

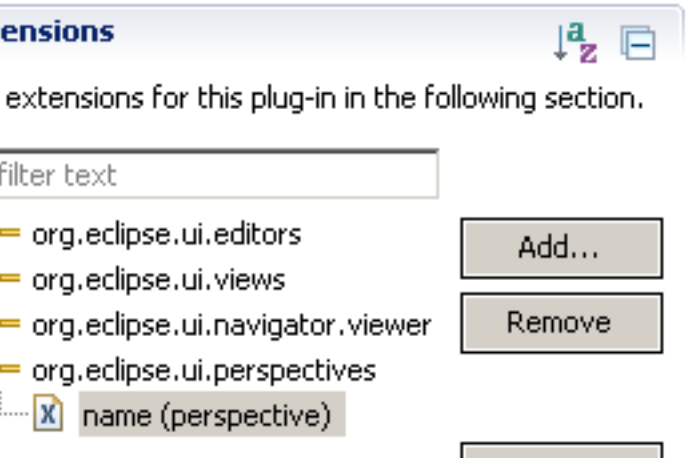
```
   </extension>
   <extension
         point="org.eclipse.ui.views">
      <view
            class="monCode.MonCommonNavigator"
            icon="icons/Laas.gif"
            id="gestionFichiers.view"
            name="Navigateur"
            restorable="true">
      </view>
   </extension>

<extension
         point="org.eclipse.ui.navigator.viewer">
      <viewerActionBinding
            viewerId="gestionFichiers.view">
         <includes>
            <actionExtension
                  pattern="org.eclipse.ui.navigator.resources.*">
            </actionExtension>
         </includes>
      </viewerActionBinding>
      <viewerContentBinding
            viewerId="gestionFichiers.view">
         <includes>
            <contentExtension
                  pattern="org.eclipse.ui.navigator.resourceContent">
            </contentExtension>
            <contentExtension
                  pattern="org.eclipse.ui.navigator.resources.filters.*">
            </contentExtension>
            <contentExtension
                  pattern="org.eclipse.ui.navigator.resources.linkHelper">
            </contentExtension>
            <contentExtension
                  pattern="org.eclipse.ui.navigator.resources.workingSets">
            </contentExtension>
         </includes>
      </viewerContentBinding>
   </extension>

</plugin>
```

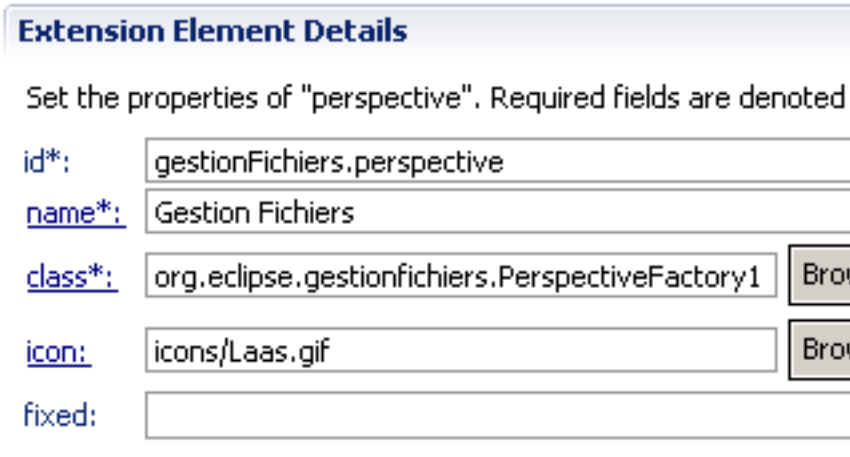
Ce qui donne dans le « Plugin Manifest Editor » les extensions :



3-F - Création de la perspective

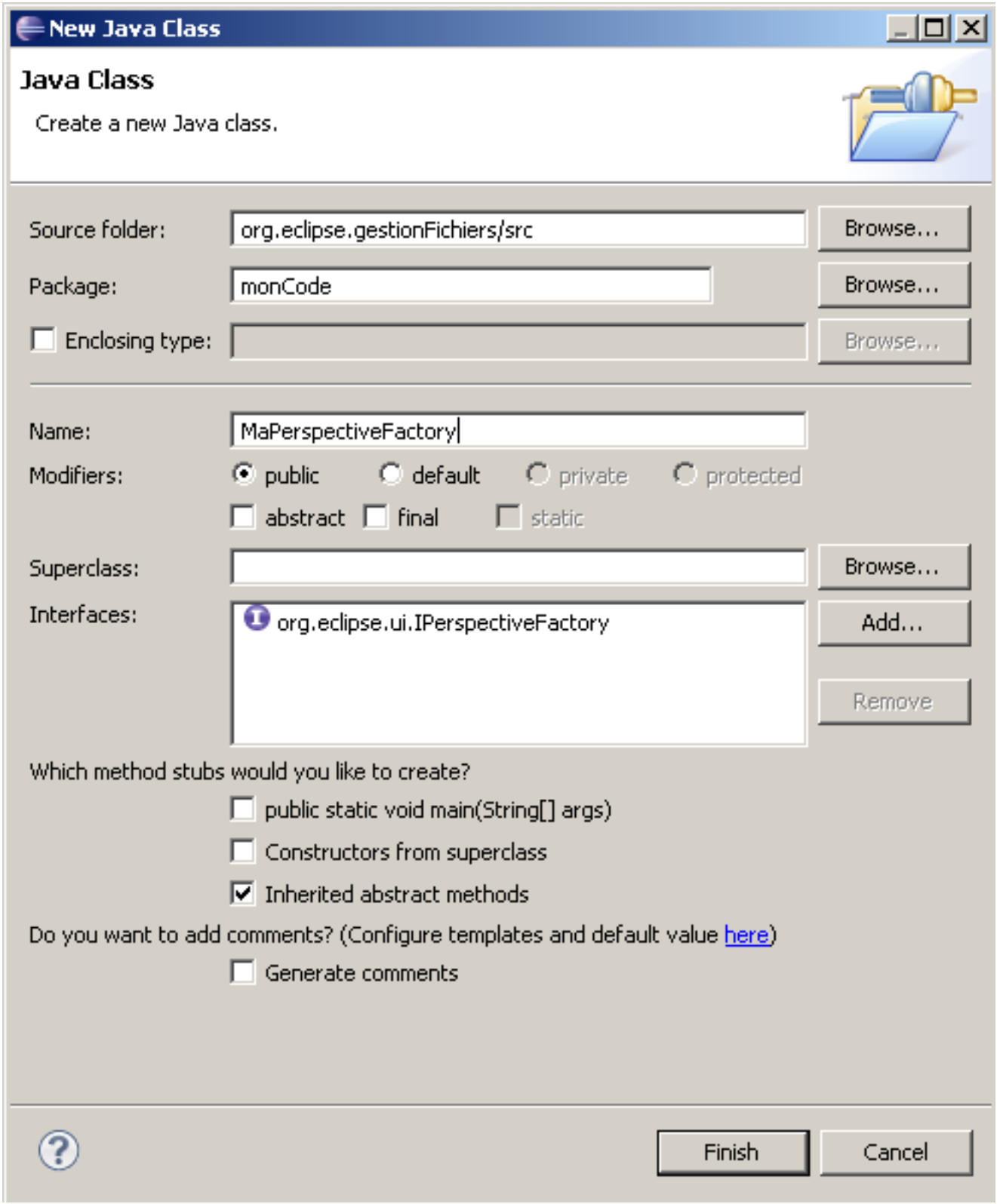
Dans le plug-in manifest Editor onglet « Extensions » faire « Add.. » sélectionner l'extension « org.eclipse.ui.perspectives » faire « Finish ». Mettre à jour la perspective créée:





Cliquer « class: ».

Initialiser la classe :



Cliquer « Finish ». Éditer : « MaPerspectiveFactory »:

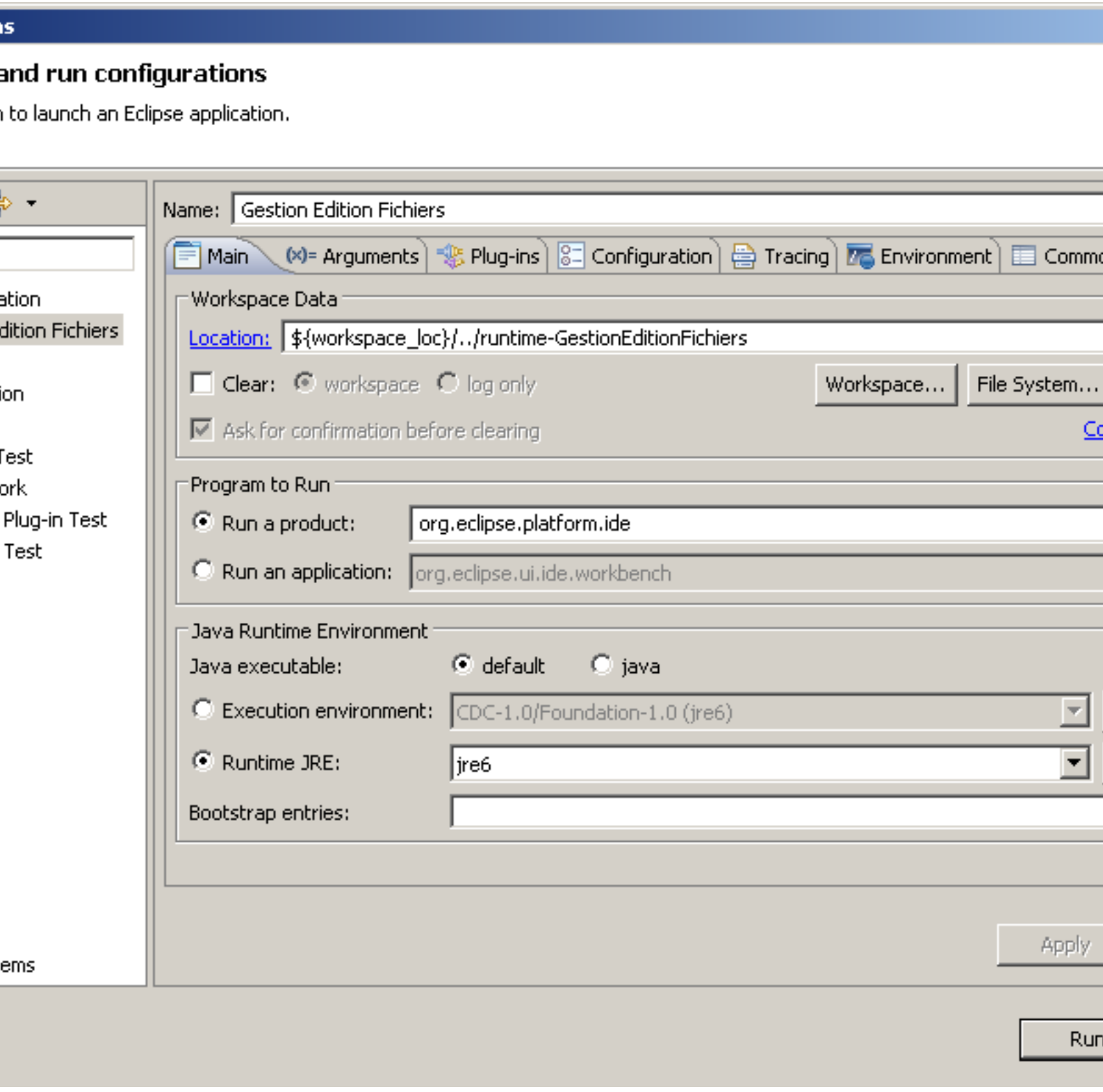
```
package monCode;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

publicclass MaPerspectiveFactory implements IPerspectiveFactory {
```

```
    publicvoid createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();
        //layout.setEditorAreaVisible(false);
        //layout.setFixed(true);
        //layout.addStandaloneView("gestionFichiers.view",  true /
* show title */, IPageLayout.LEFT, 0.5f, editorArea);
        layout.addView("gestionFichiers.view", IPageLayout.LEFT, 0.5f, editorArea);
    }


}
```

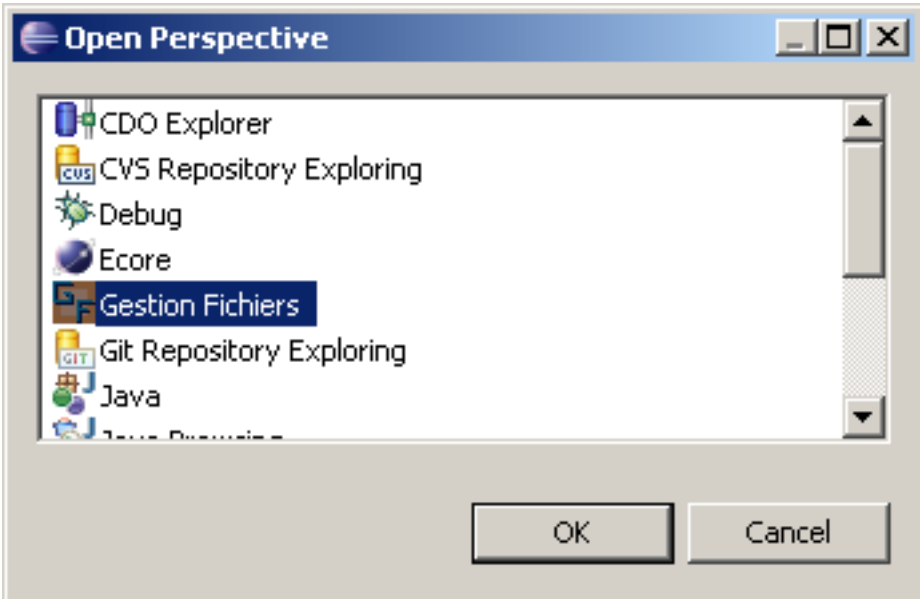
4 - Test du plugin

4-A - Test navigateur et éditeur

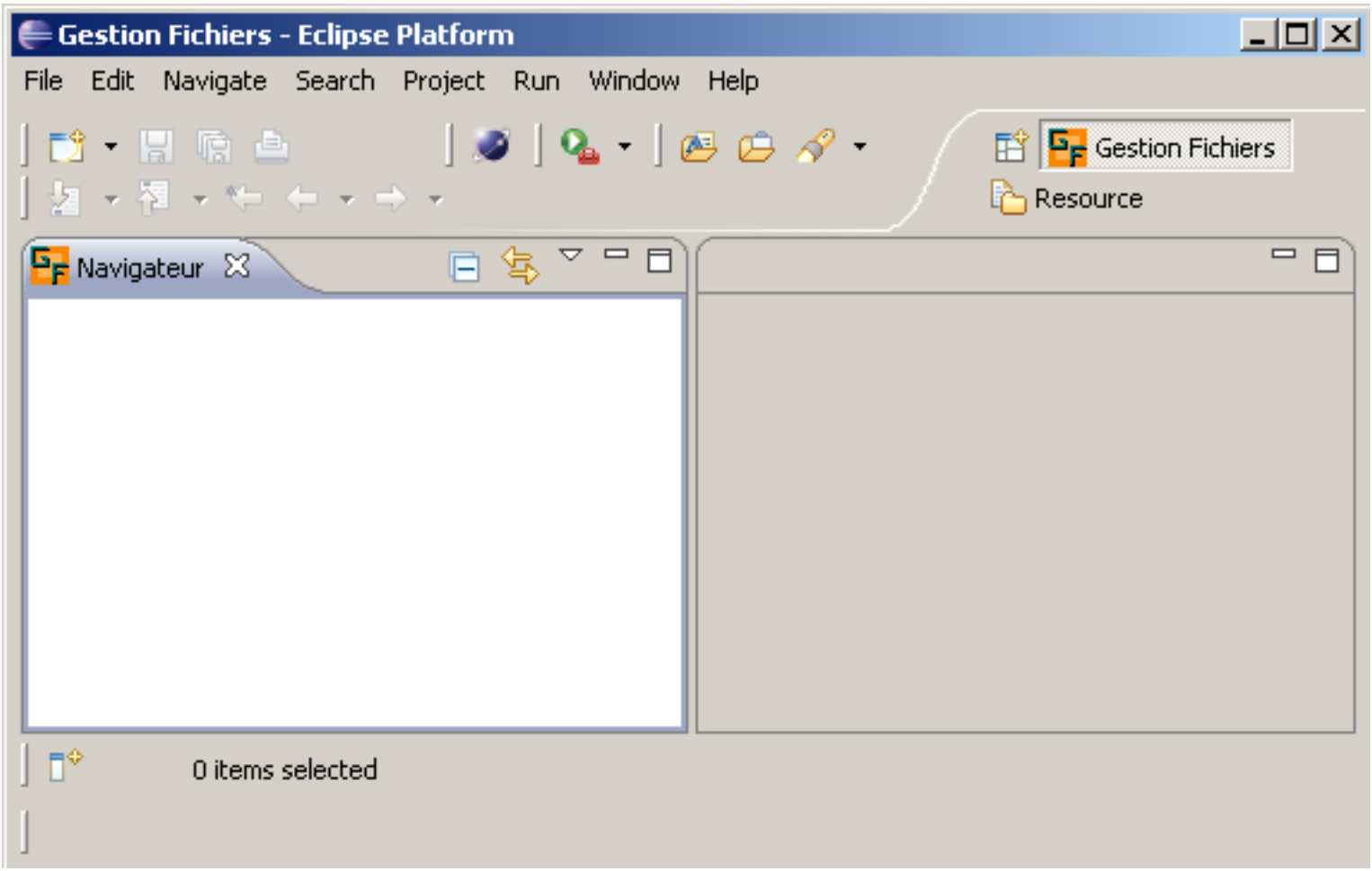
Faire un clic droit sur « org.eclipse.gestionFichiers » dans le « Package Explorer » et sélectionner la commande : « Run As > Run Configuration... ». Dans « Run Configuration - **Create,manage, and run configurations** ». Double cliquer « Eclipse Application » et nommer la configuration « Gestion Edition Fichiers », cliquer « Apply ».



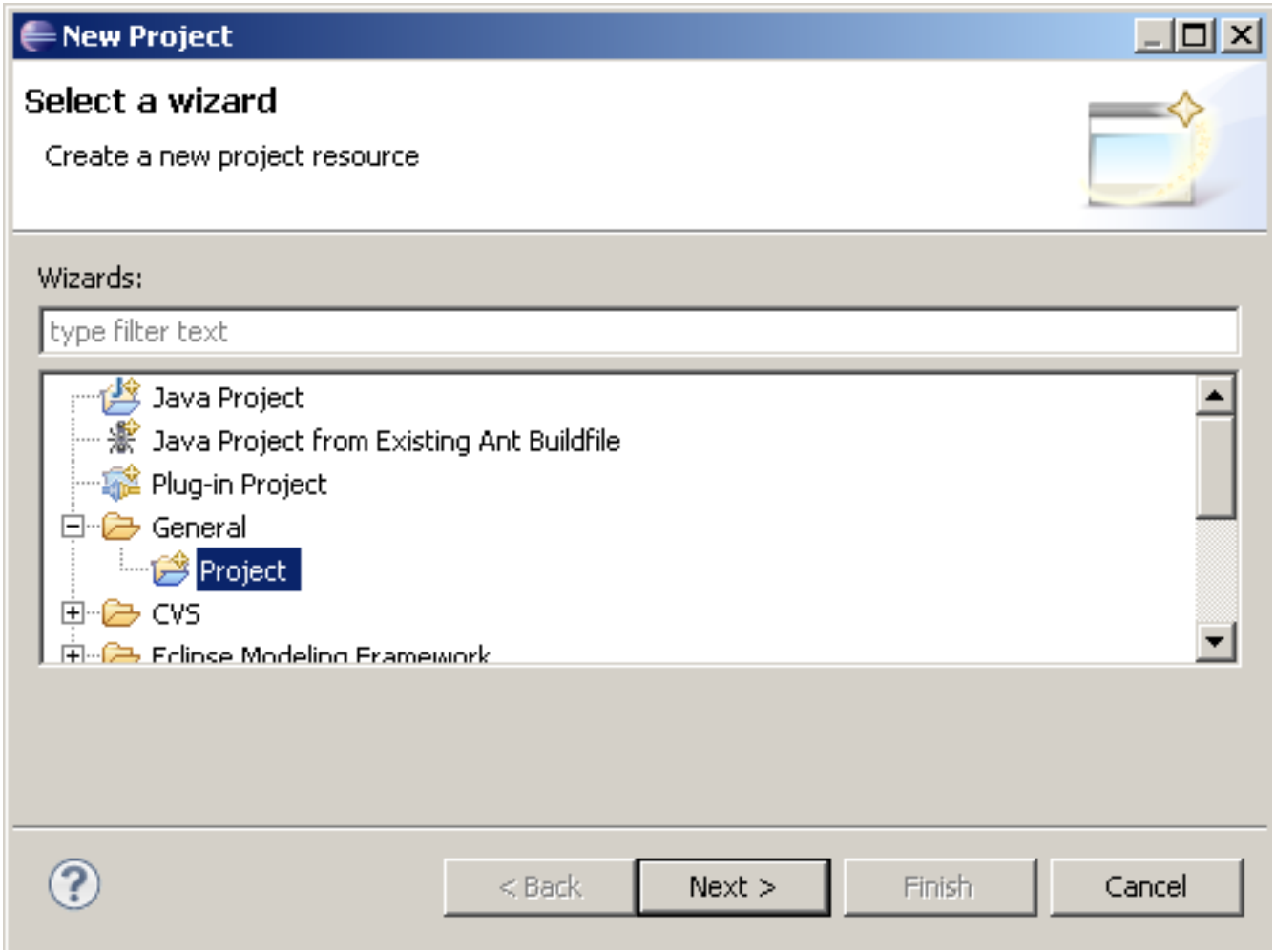
Cliquer « Run ». Fermer la fenêtre « Welcome ». Sélectionner « Window > Open Perspective > Others... ». Dans « Open Perspective » sélectionner « Gestion Fichiers »:



Cliquer « OK ». On obtient :

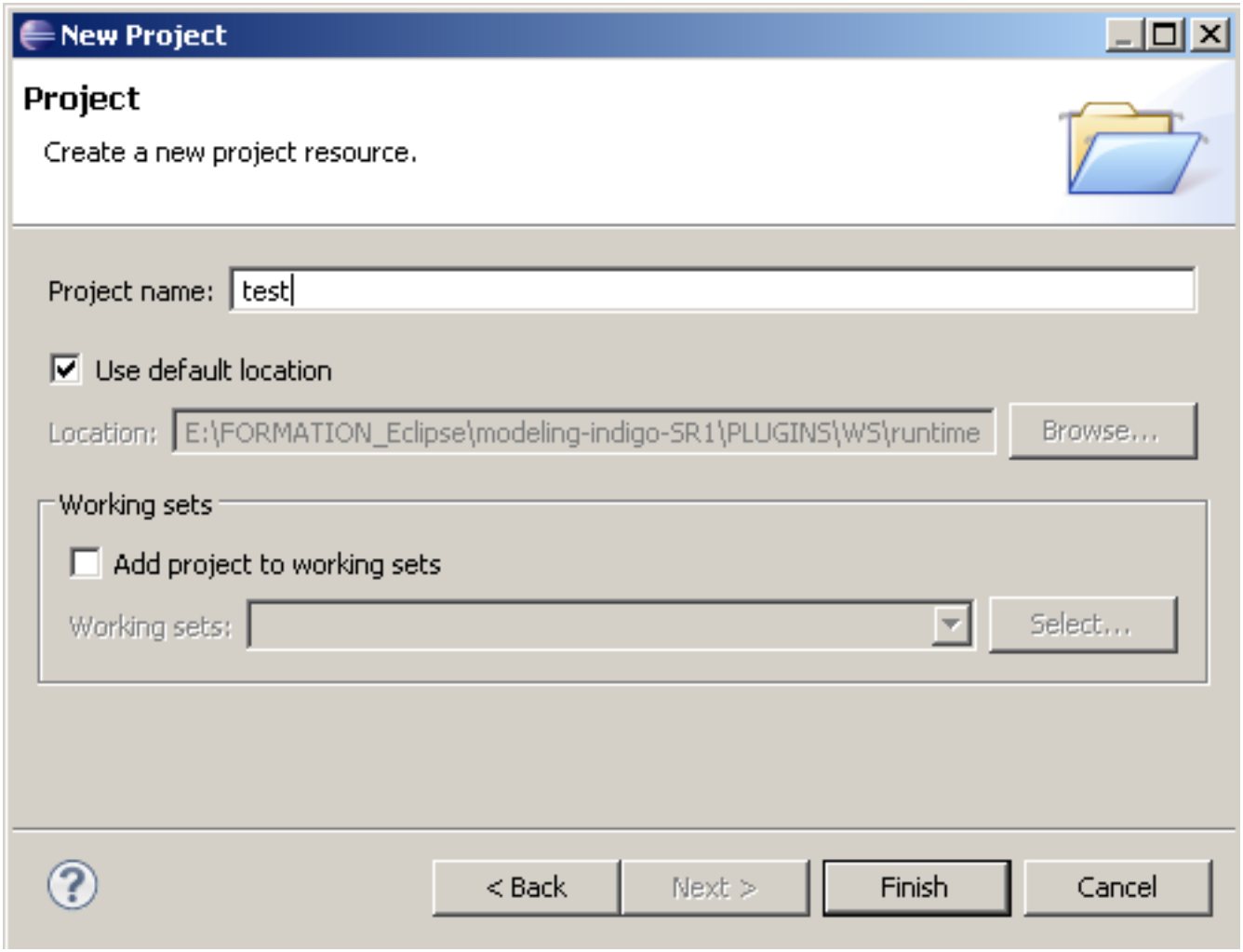


On sélectionne « File > New > Project... ». Dans « New Project - **Select a wizard** » on ouvre « General » et on sélectionne « Project » :

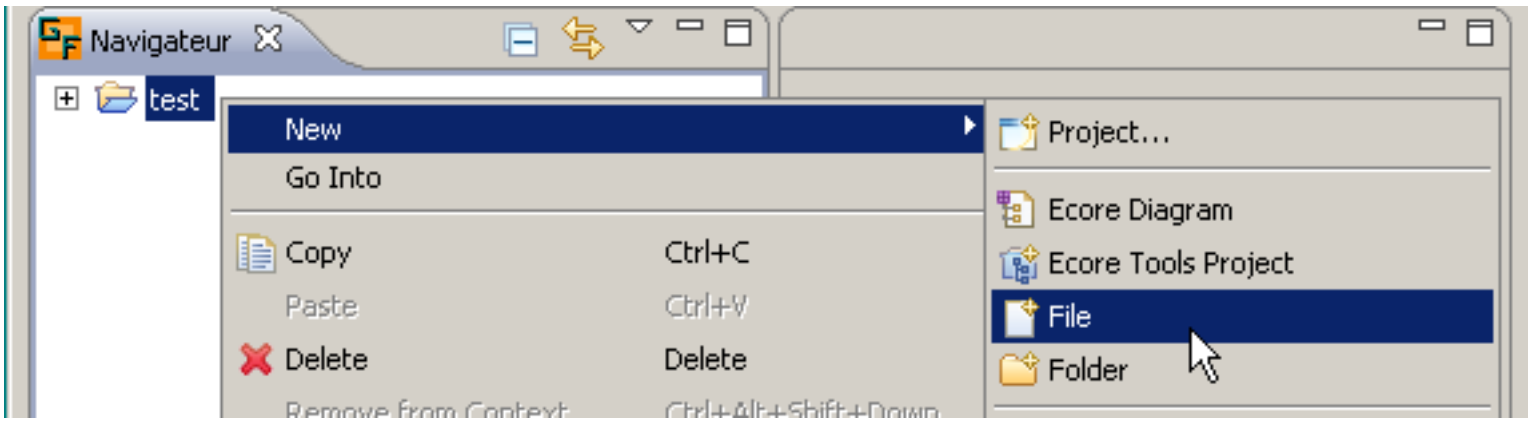


Cliquer « Next > »

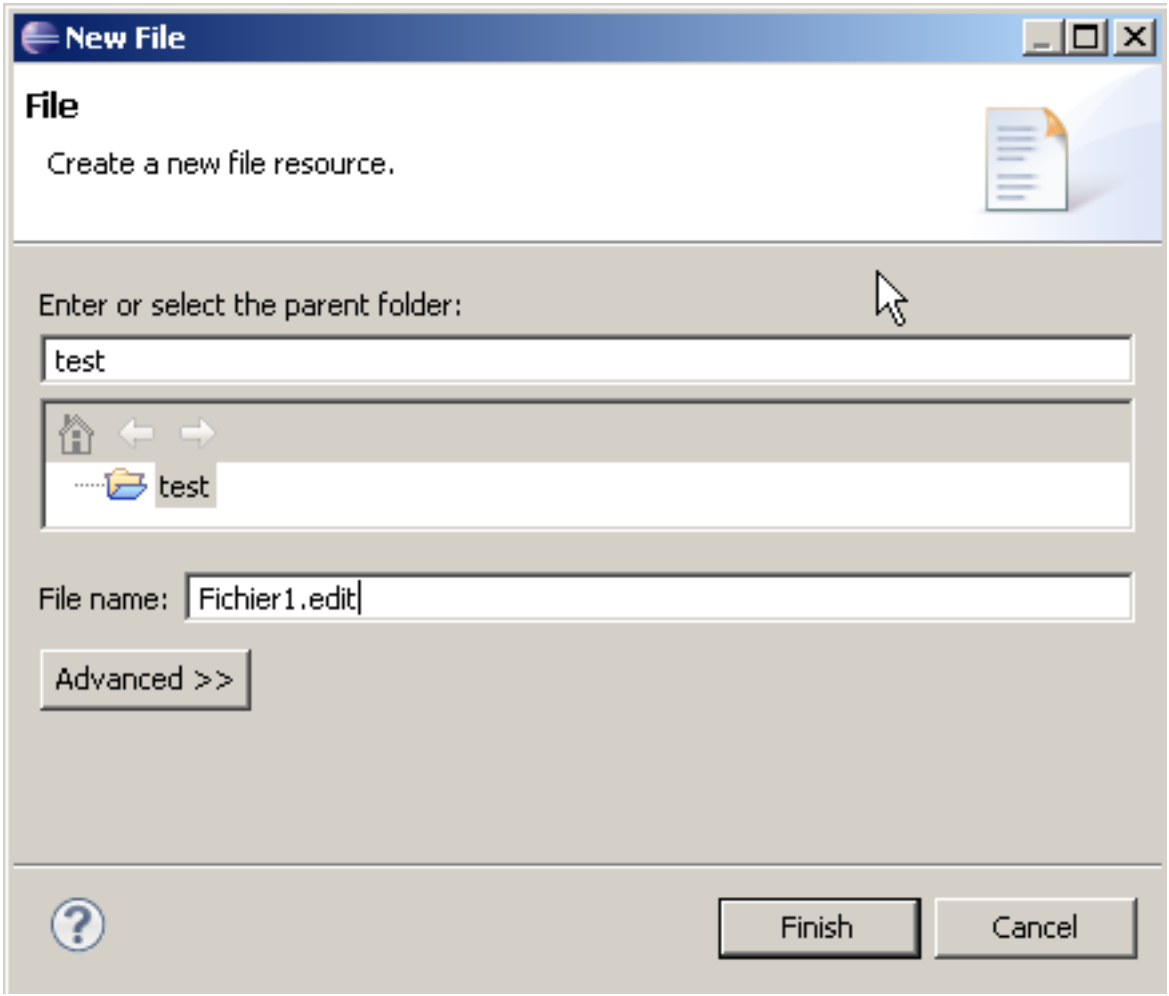
Nommer le projet :



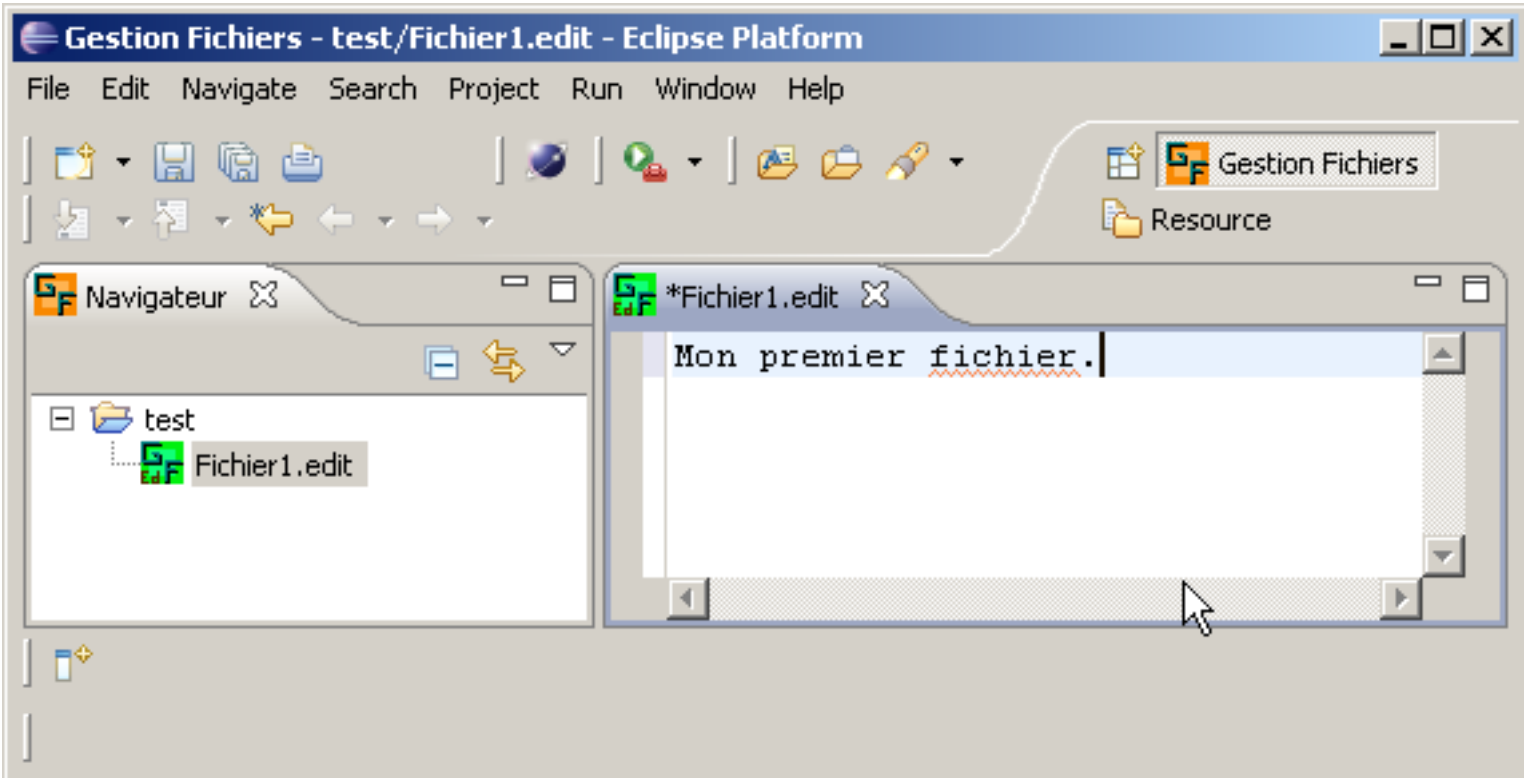
Cliquer « Finish ». Le projet est créé. Nous allons créer un fichier « *.edit » sous ce projet :



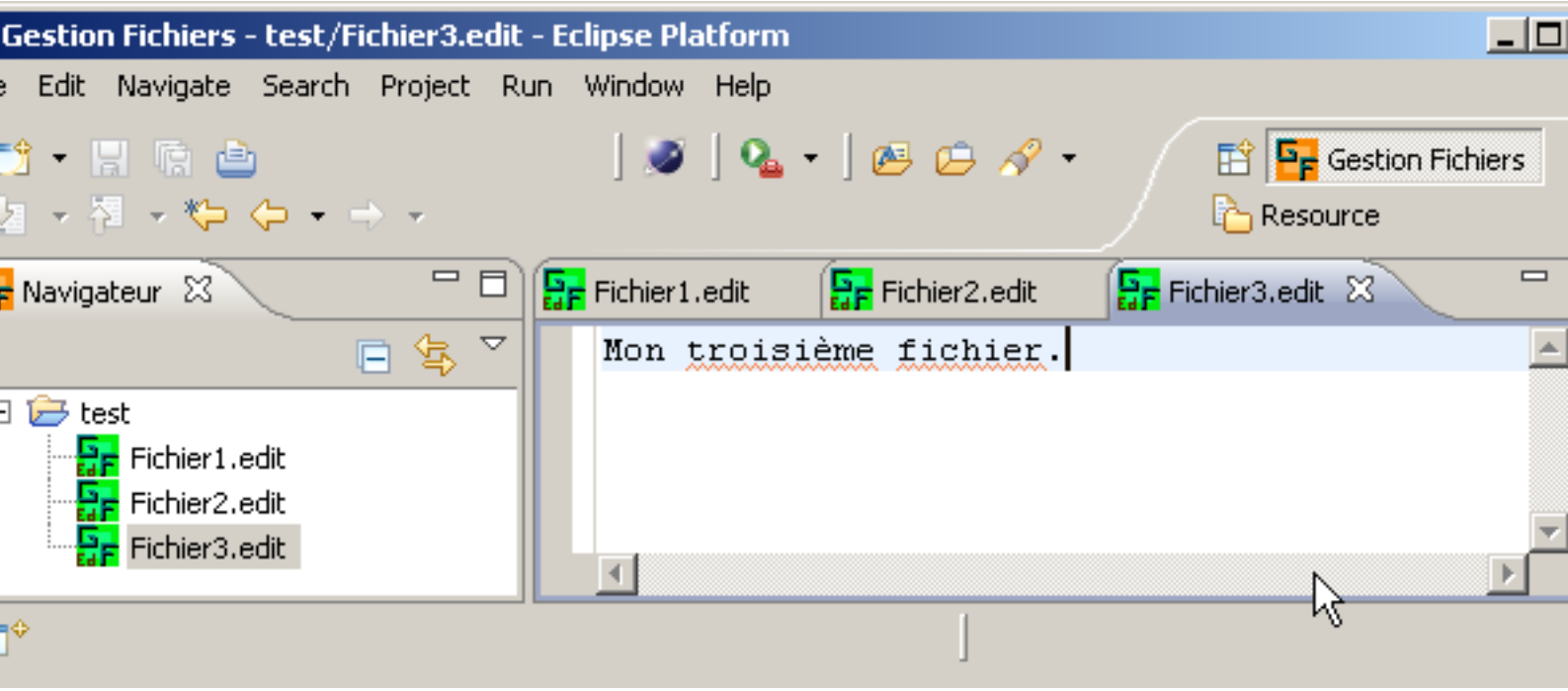
Nommer le fichier :



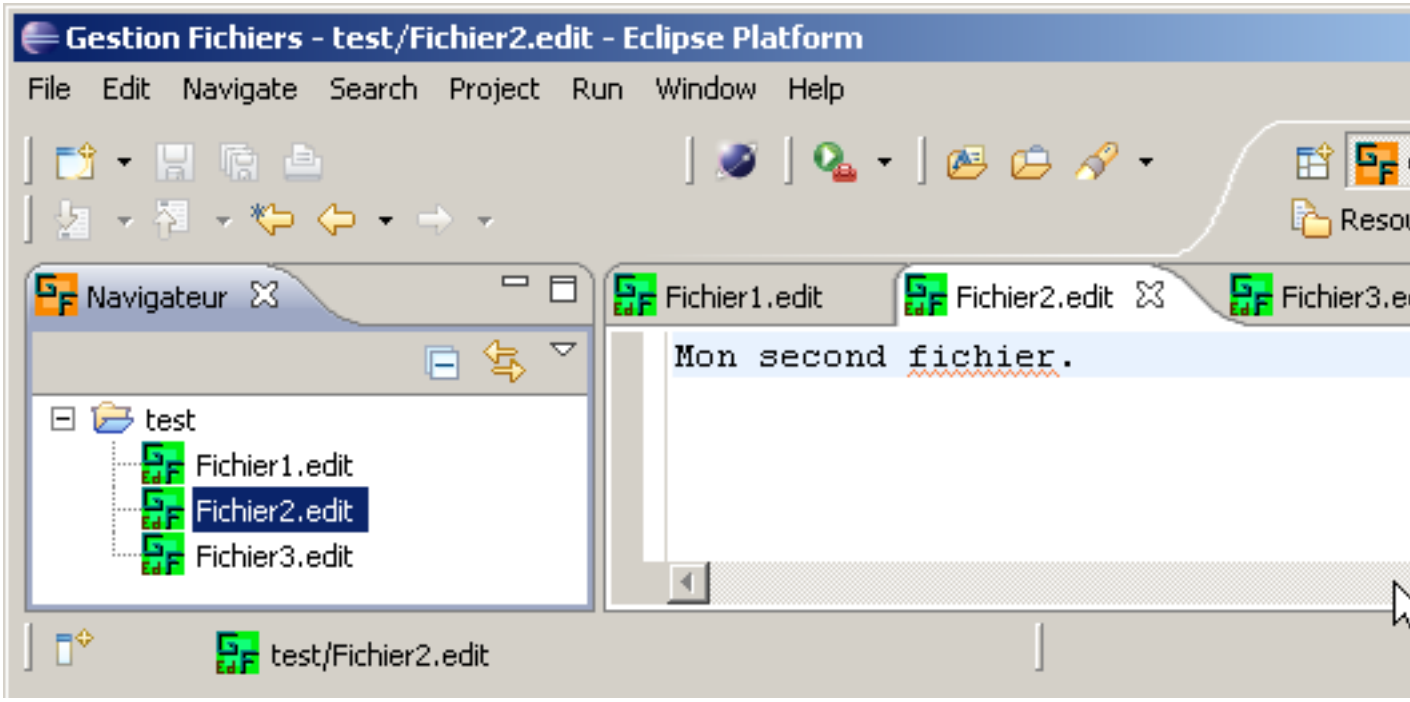
On obtient :



On peut créer d'autres fichiers :



et naviguer d'un fichier à l'autre :



5 - Accès aux fichiers

Nous allons ajouter dans la barre des menus, un menu contenant une commande, permettant de lire et d'afficher le fichier sélectionner dans le « Navigateur ».

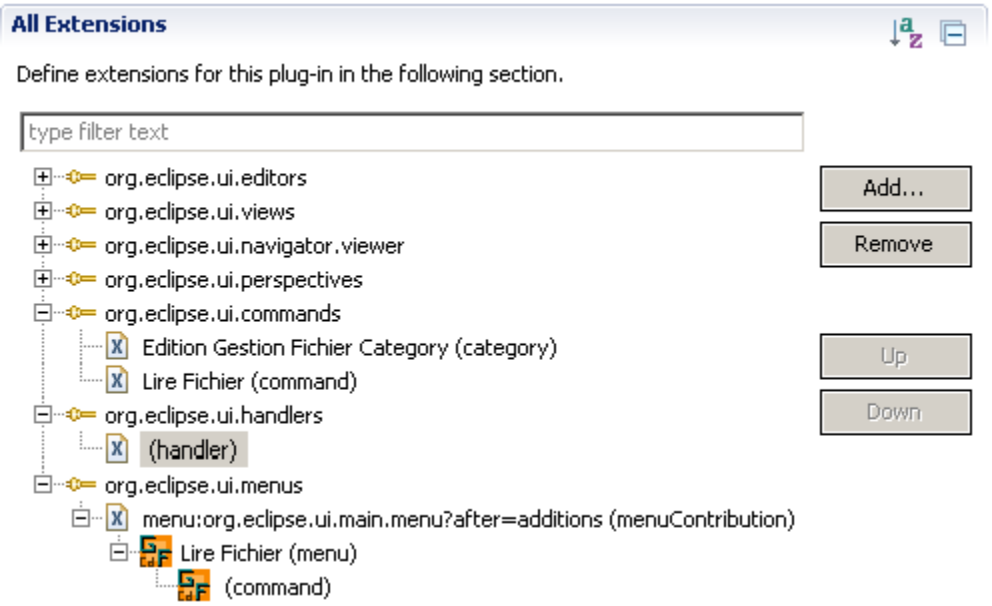
Il existe plusieurs document où la mise en place d'un menu est décrite pas à pas. Le menu est créé par édition du fichier « plugin.xml ». Le menu est défini par les extensions :

```
<extension
         point="org.eclipse.ui.commands">
      <category
            id="gestionFichiers.category"
            name="Edition Gestion Fichier Category">
      </category>
      <command
            categoryId="gestionFichiers.category"
            id="gestionFichiers.lireFichier"
            name="Lire Fichier">
      </command>
   </extension>
   <extension
         point="org.eclipse.ui.handlers">
      <handler
            class="monCode.LireFichier"
            commandId="gestionFichiers.lireFichier">
      </handler>
   </extension>
   <extension
         point="org.eclipse.ui.menus">
      <menuContribution
            locationURI="menu:org.eclipse.ui.main.menu?after=additions">
         <menu
               icon="icons/LaasCd.gif"
               id="LireFichierMenu"
               label="Lire Fichier">
            <command
                  commandId="gestionFichiers.lireFichier"
                  icon="icons/LaasCd.gif"
                  style="push">
            </command>
```

```
            </menu>
         </menuContribution>
      </extension>
```

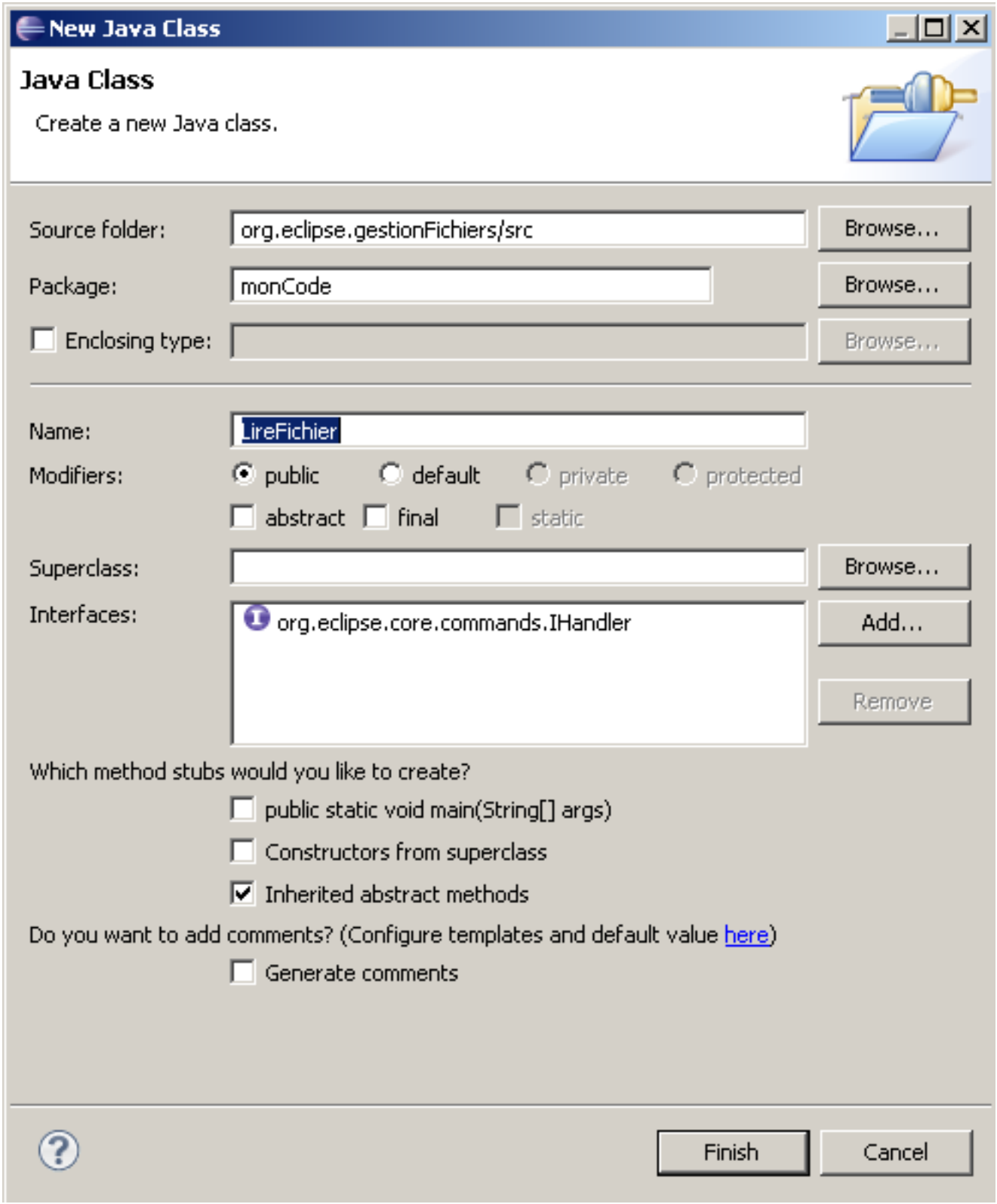
après édition le fichiers « plugin.xml » devient :

Ce qui correspond aux extensions :



Cliquer « class: »

Ne pas modifier « New Java Class - Create a new Java class »



Cliquer « Finish ».

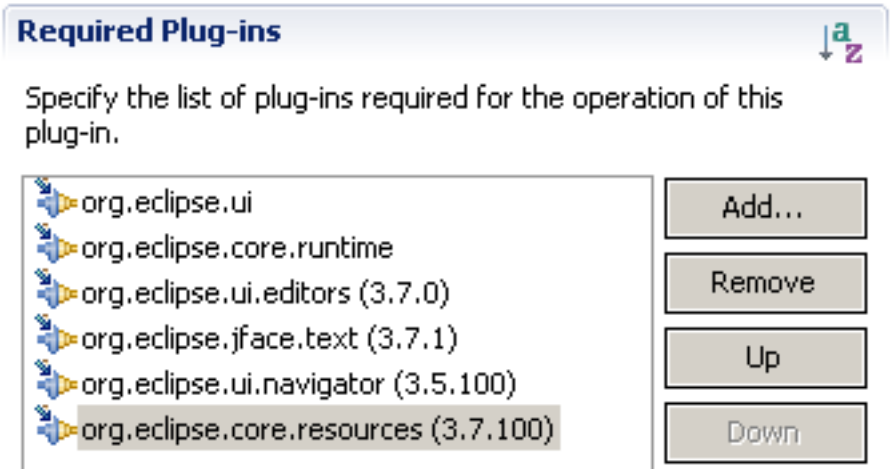
Éditer la classe « LireFichier » :

```
package monCode;

import java.io.IOException;
```

```
import java.io.InputStream;

import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.commands.IHandlerListener;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IWorkspaceRoot;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.ui.IWorkbenchWindow;

publicclass LireFichier implements IHandler {

    private String[] nomCompose;

    public IWorkbenchWindow[] windows;

    @Override
    publicvoid addHandlerListener(IHandlerListener handlerListener) {
        // TODO Auto-generated method stub

    }

    @Override
    publicvoid dispose() {
        // TODO Auto-generated method stub

    }

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        windows =
 org.eclipse.gestionfichiers.Activator.getDefault().getWorkbench().getWorkbenchWindows();
        nomCompose = MonCommonNavigator.selectionFichier;

        IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();

        // ouvertureduprojet
        //    nomCompose[0] non utilisé
        //    nomCompose[1] nomduprojet
        IProject myProject = root.getProject(nomCompose[1]);
        if (myProject.exists() && !myProject.isOpen()){
            try {
                myProject.open(null);
            } catch (CoreException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        int i = 2;
        int imax = nomCompose.length -1;
        IFolder folderCourant = (IFolder) null;
        IFile fich = (IFile) null;
        while (i <= imax) {
            if (i == imax) {
                // fichier
                if (folderCourant == null){
                    // precedent: projet
                    fich =myProject.getFile(nomCompose[i]);
                } else {
                    // precedent: folder
                    fich =folderCourant.getFile(nomCompose[i]);
                }
            } else {
                // folder
                if (folderCourant == null){
                    // precedent: projet
                    folderCourant =myProject.getFolder(nomCompose[i]);
```

```
                } else {
                    // precedent: folder
                    folderCourant =folderCourant.getFolder(nomCompose[i]);
                }
            }
            i++;
        }

        //Lecture Fichier
        InputStream is;
        try {
            is = fich.getContents();

        } catch (CoreException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            returnnull;
        }

        String contenuFichier = lecture(is);

        // affichagecontenudufichier
        MessageDialog.openInformation(windows[0].getShell(), "Contenu fichier sélectionné",
                contenuFichier);

        returnnull;
    }

    private String lecture (InputStream is) {
        String result = new String();
        int c =0;
        boolean stop = false;
        while (!stop) {
            try {
                c = is.read();
                if (c == -1) {
                    stop = true;
                } else {
                    result += (char)c;
                }
            }
            catch (IOException e) {
                stop = true;
            }
        }
        return result;

    }

    @Override
    publicboolean isEnabled() {
        // TODO Auto-generated method stub
        returntrue;
    }

    @Override
    publicboolean isHandled() {
        // TODO Auto-generated method stub
        returntrue;
    }

    @Override
    publicvoid removeHandlerListener(IHandlerListener handlerListener) {
        // TODO Auto-generated method stub

    }

}
```

Il y a des erreurs. Il faut ajouter une dépendance. Aller dans le « Plug-in Manifest Editor » onglet « Dependencies ». Dans « Required Plug-ins » faire « Add... » pour ajouter le plugin: « org.eclipse.core.resources». On a les dépendances:



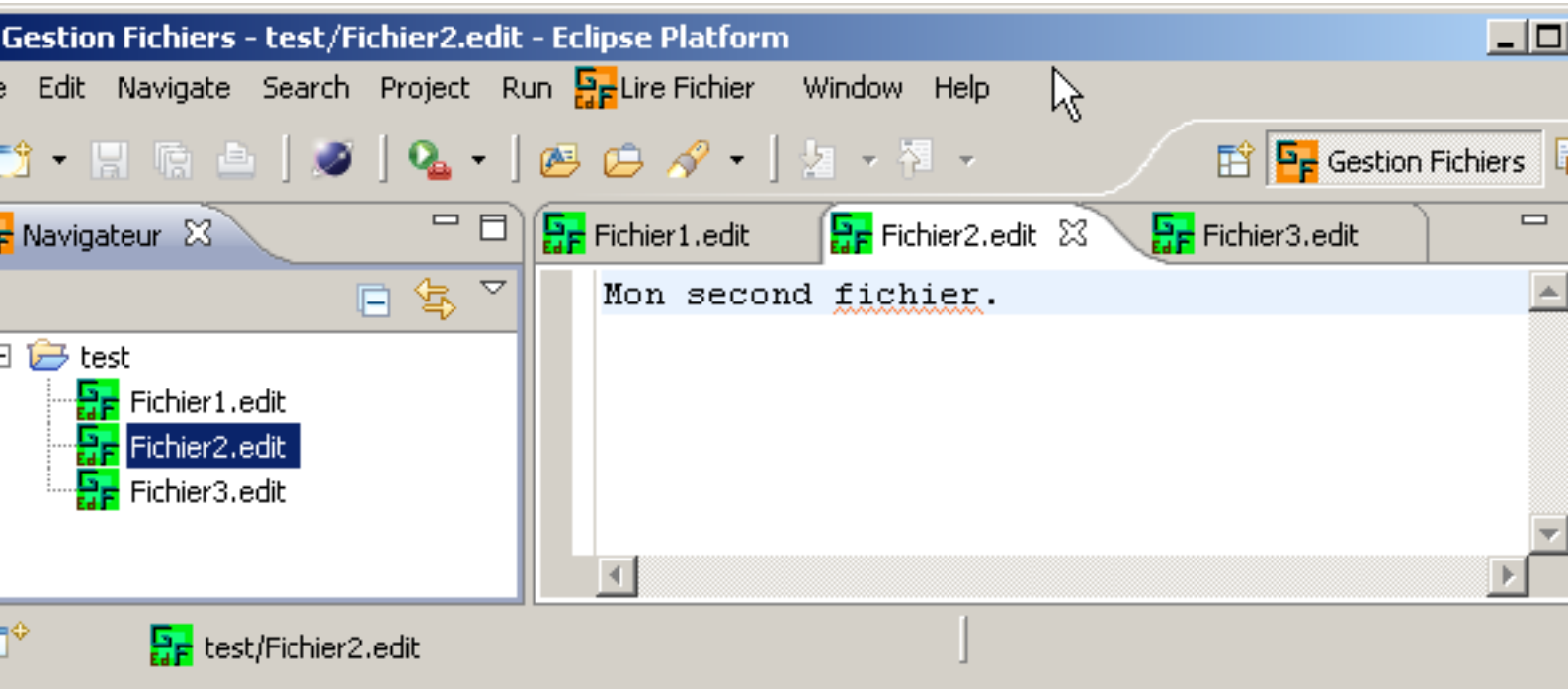
Faire une sauvegarde les erreurs disparaissent.

6 - Test du plugin

6-A - Test commande « Lire Fichier »

Relancer le test à partir de la configuration déjà créée :

Sélectionner un fichier :



Faire :

le message si dessous s'affiche :

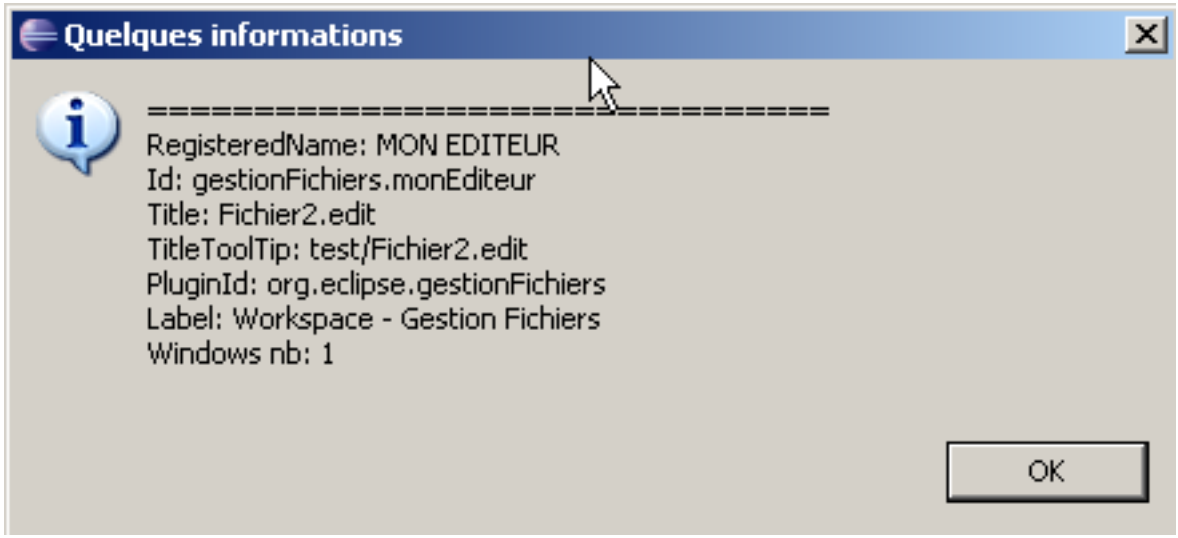
6-B - Affichage informations sur l'éditeur

Éditer la classe « monCode.MonEditeur »: supprimer dans la méthode « doSave » les commentaires

afin d'afficher quelques informations.

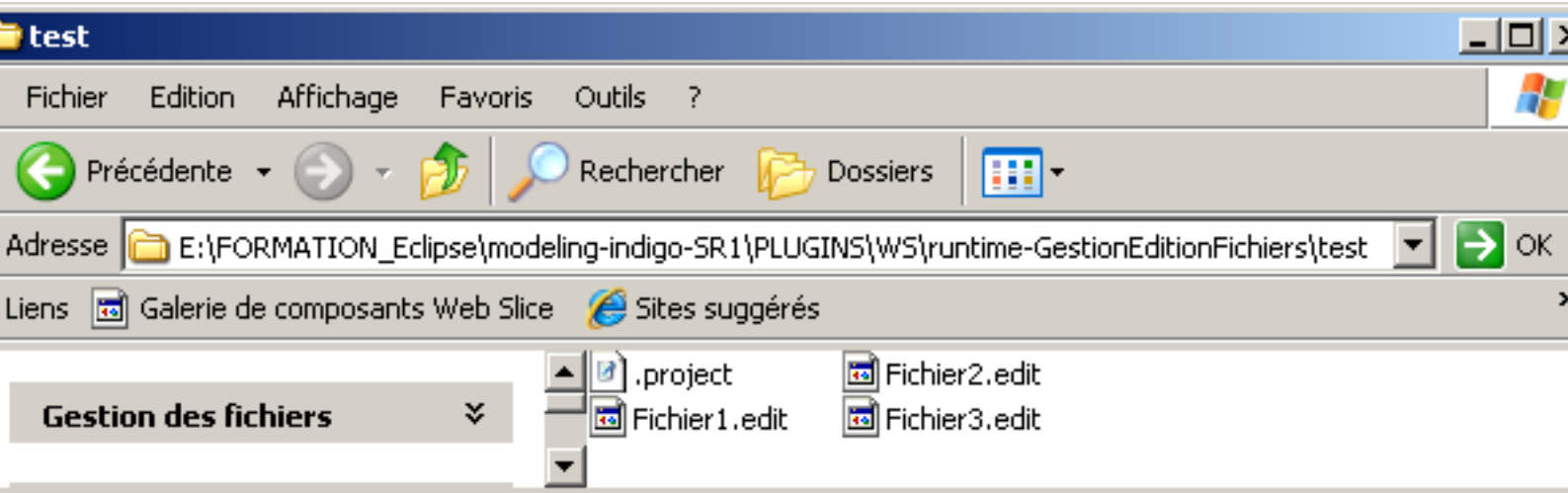
```
public void doSave(IProgressMonitor monitor) {
        super.doSave(monitor);

        // Quelques information obtenues lors de la sauvegarde
        String message = new String("================================");
        message = message +"\nRegisteredName: "+ this.getSite().getRegisteredName();
        message = message +"\nId: "+ this.getSite().getId();
        message = message +"\nTitle: "+ this.getSite().getPart().getTitle();
        message = message +"\nTitleToolTip: "+ this.getSite().getPart().getTitleToolTip();
        message = message +"\nPluginId: "+ this.getSite().getPluginId();
        message = message +"\nLabel: "+ this.getSite().getPage().getLabel();
        message = message +"\nWindows nb: "+ windows.length;
        MessageDialog.openInformation(windows[0].getShell(),
                "Quelques informations ", message);


    }
```

Relancer le test du plugin et lors de la sauvegarde du fichier en édition le message suivant s'affiche:



6-C - Emplacement des fichiers édités

Les fichiers édité sont sauvegardés dans le « workspace » de la plate-forme d'exécution :



7 - Conclusions

La classe « LireFichier » montre comment accéder au contenu du fichier édité.

Pour plus de précisions sur « Common Navigator Framework » afficher la documentation en ligne avec: « Help > HelpContents » et aller à la rubrique:

- Platform Plug-in Developer Guide > Programmer's Guide > Common Navigator Framework

8 - Licence

La licence « créative commons » :

- **http://creativecommons.org/licenses/by-nc-nd/2.0/fr/**

s'applique à ce document.