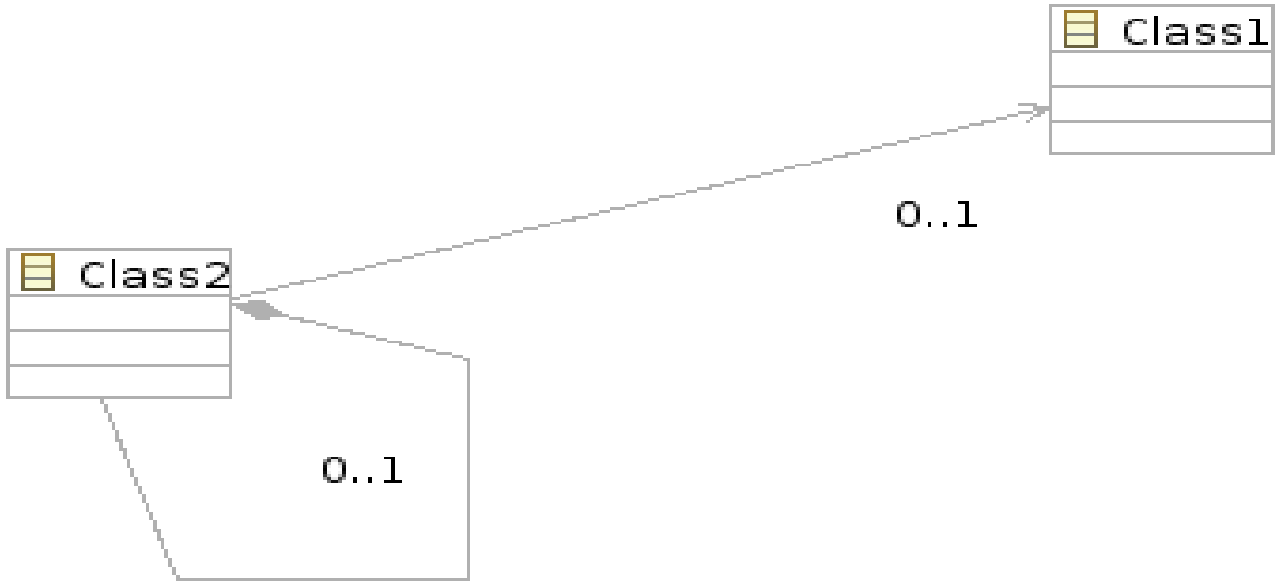


Editors @ Tues

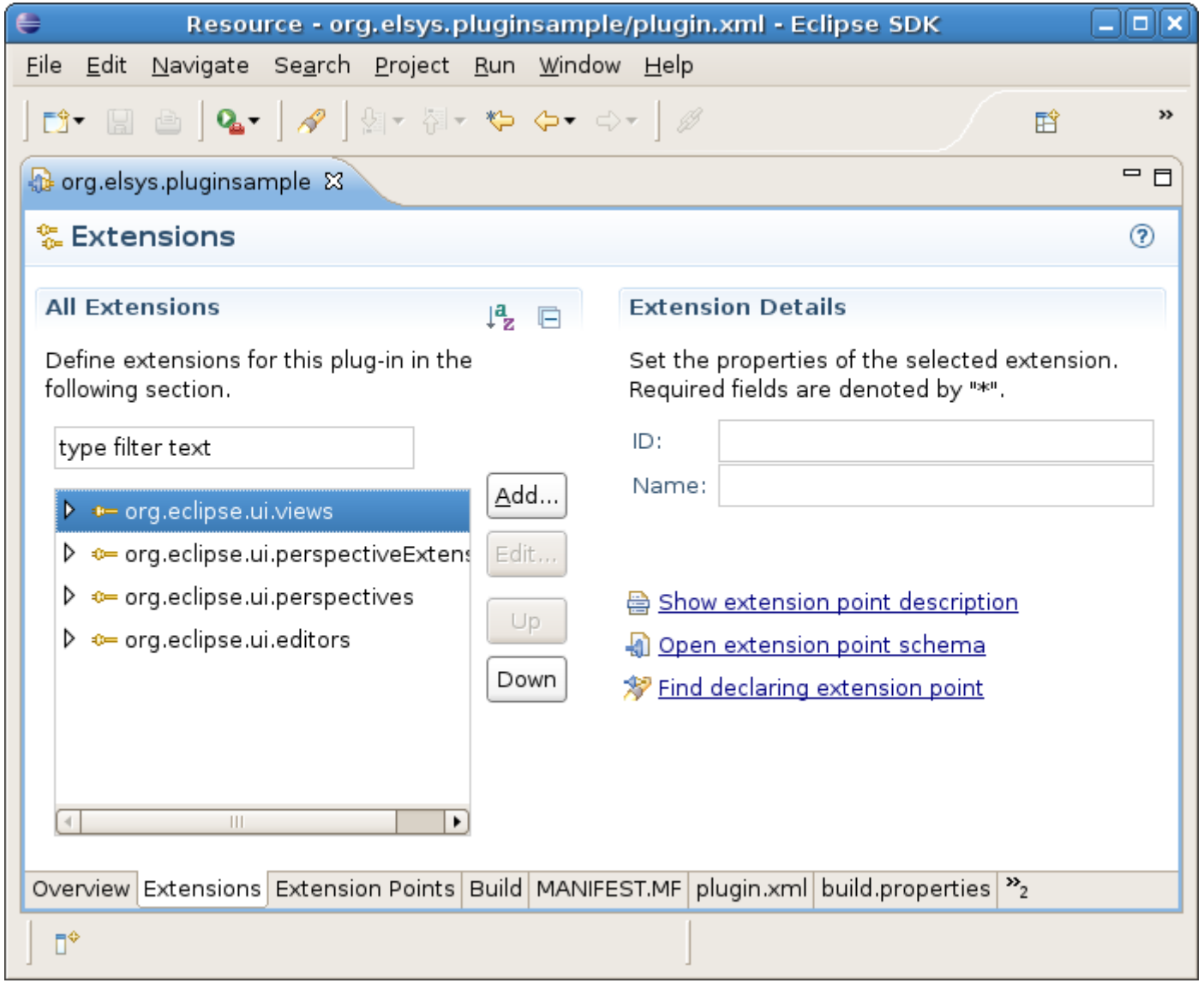
Editors

Основният начин за създаване и модифициране на ресурси в Eclipse са Редакторите. Съществуват значителен брой редактори от най-простите текстови редактори до по-сложни редактори включващи няколко страници.
Пример за такъв редактор е редакторът за MANIFEST.MF и plugin.xml.
В този случай два различни файла се редактират едновременно с един и същи редактор.
Всеки plugin може да предостави редактор като използва същия extension point както всички други редактори.

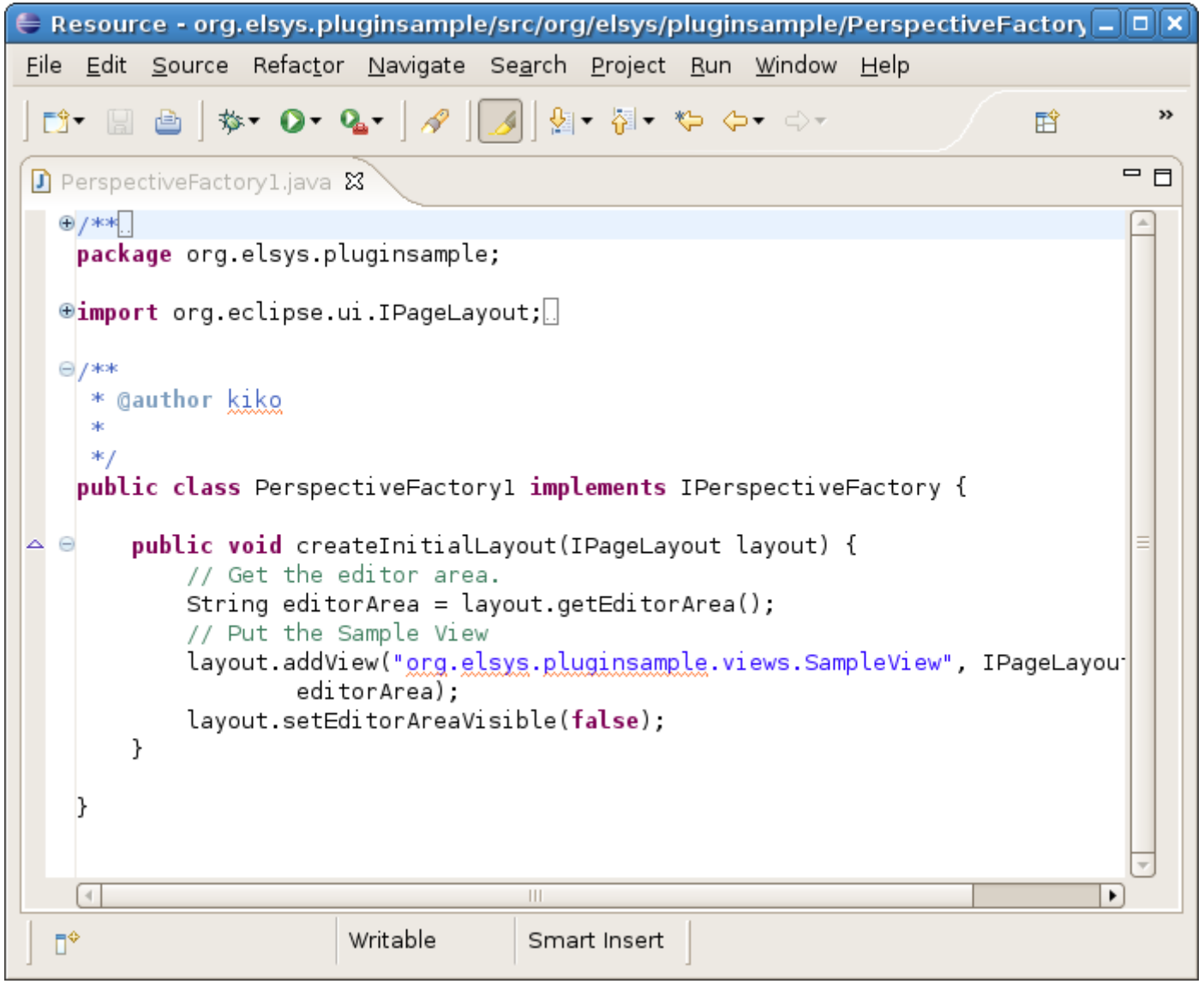
Редакторите следват следния цикъл на живот:

Отваряне на файл -> Редактиране -> Затваряне.

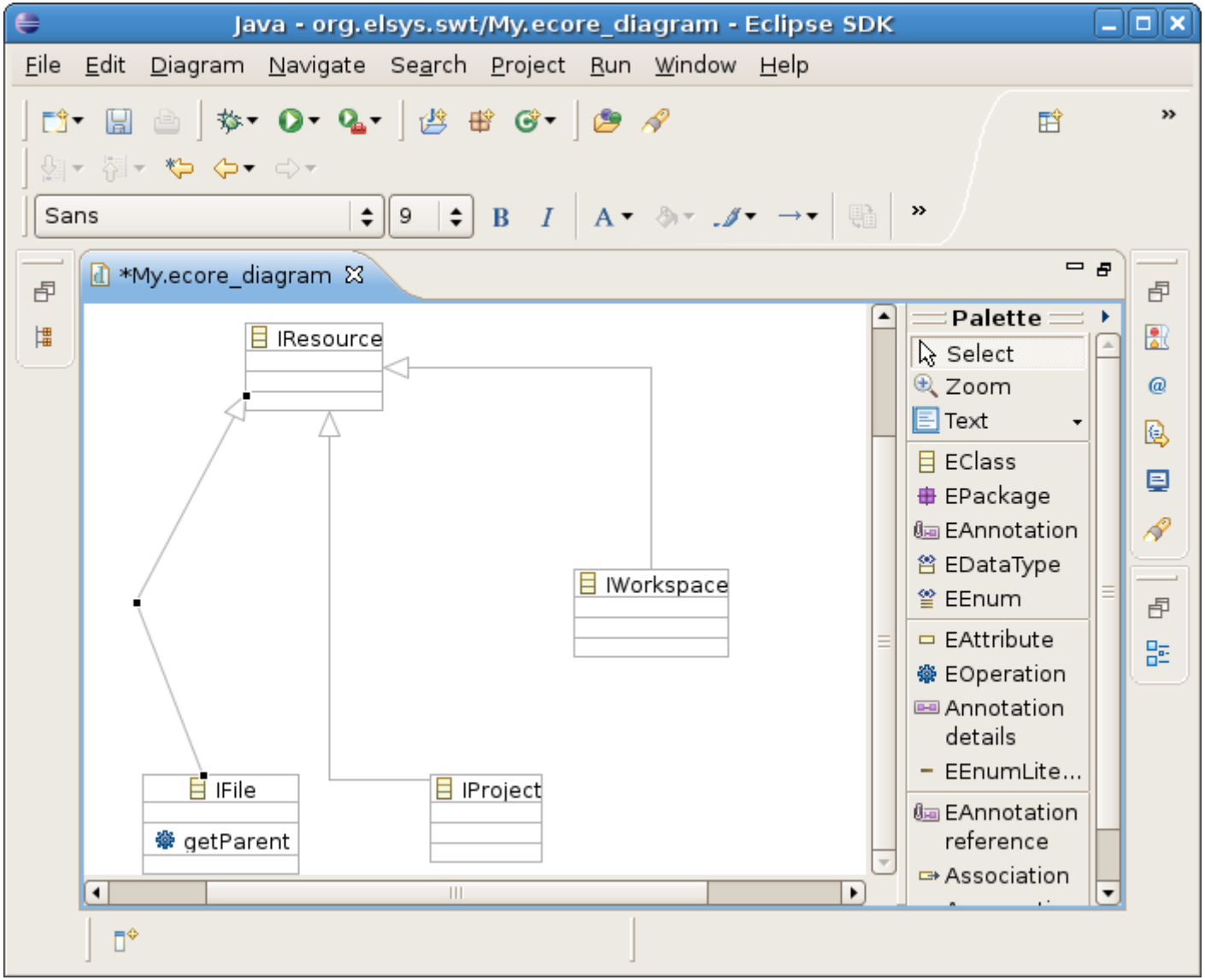
Редактори


Resource - org.elsys.pluginsample/plugin.xml - Eclipse SDK
File Edit Navigate Search Project Run Window Help
org.elsys.pluginsample
Extensions
All Extensions
Define extensions for this plug-in in the following section.
type filter text
org.eclipse.ui.views
org.eclipse.ui.perspectiveExtens
org.eclipse.ui.perspectives
org.eclipse.ui.editors
Add...
Edit...
Up
Down
Extension Details
Set the properties of the selected extension. Required fields are denoted by "*".
ID:
Name:
Show extension point description
Open extension point schema
Find declaring extension point
Overview
Extensions
Extension Points
Build
MANIFEST.MF
plugin.xml
build.properties

Редактори


Resource - org.elsys.pluginsample/src/org/elsys/pluginsample/PerspectiveFactory
File Edit Source Refactor Navigate Search Project Run Window Help
PerspectiveFactory1.java
/**
package org.elsys.pluginsample;
import org.eclipse.ui.IPageLayout;
/**
* @author kiko
*
*/
public class PerspectiveFactory1 implements IPerspectiveFactory {
public void createInitialLayout(IPageLayout layout) {
// Get the editor area.
String editorArea = layout.getEditorArea();
// Put the Sample View
layout.addView("org.elsys.pluginsample.views.SampleView", IPageLayou
editorArea);
layout.setEditorAreaVisible(false);
}
}
Writable
Smart Insert

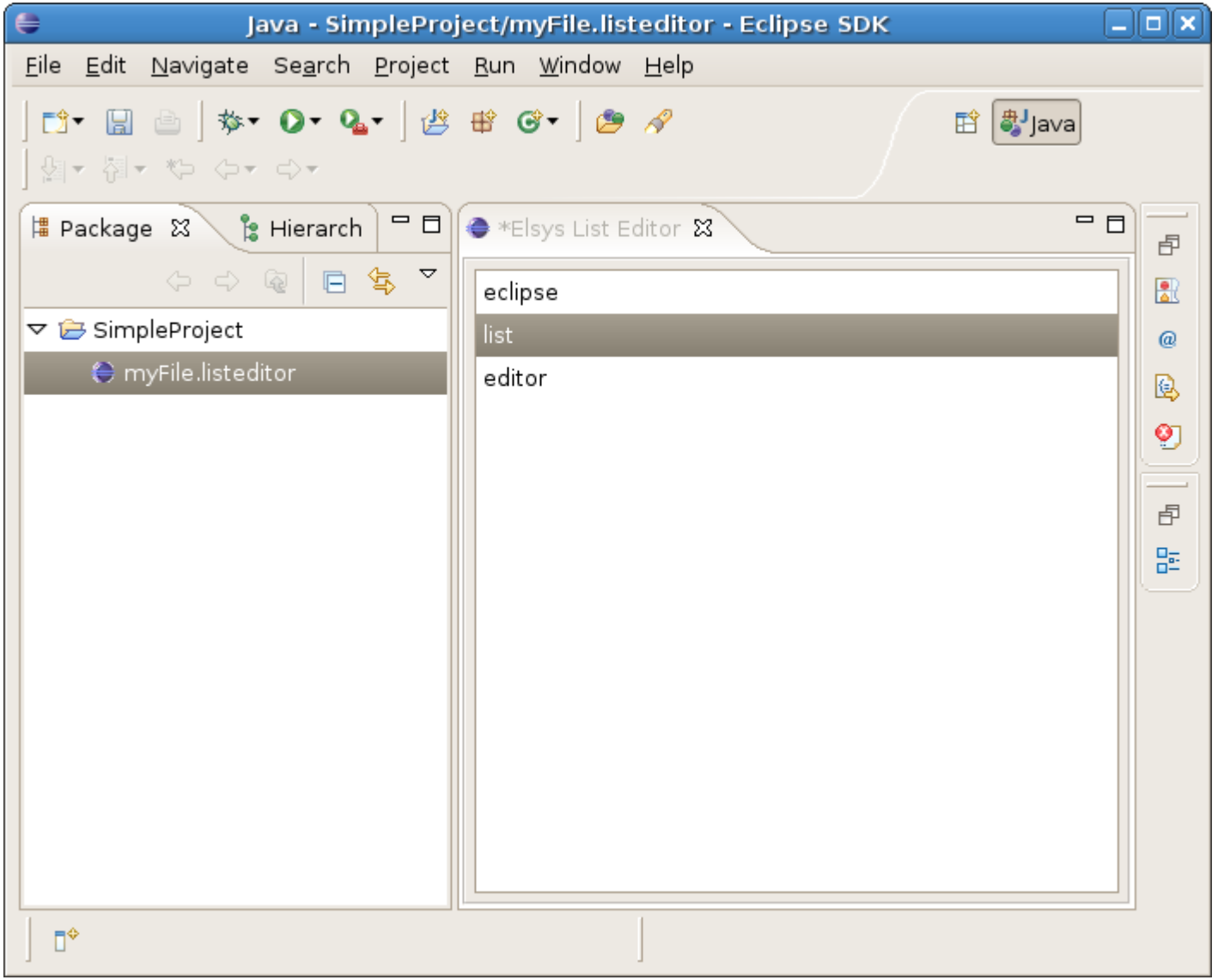
Редактори


Java - org.elsys.swt/My.ecore_diagram - Eclipse SDK
File Edit Diagram Navigate Search Project Run Window Help
Sans
9
*My.ecore_diagram
IResource
IWorkspace
IFile
getParent
IProject
Palette
Select
Zoom
Text
EClass
EPackage
EAnnotation
EDataType
EEnum
EAttribute
EOperation
Annotation details
EEnumLite...
EAnnotation reference
Association

Title

Пример:
Да се изгради редактор редактиращ списък от низове.
Редакторът трябва да работи с файлове с разширение .listeditor.
Във визуалната си част редакторът ще изобразява низовете като списък. Към списъка може да се добавят и изтриват низове.
Действията ще се извършват чрез избор от контекстно меню.

Title


Java - SimpleProject/myFile.listeditor - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Package
Hierarch
*Elsys List Editor
SimpleProject
myFile.listeditor
eclipse
list
editor

Title

Използваме следния extension point

```
   <extension
         point="org.eclipse.ui.editors">
      <editor

class="org.elsys.pluginsample.editor.ListEditor"
            extensions="listeditor"
            icon="icons/sample.gif"
            id="org.elsys.pluginsample.editor1"
            name="Elsys List Editor">
      </editor>
   </extension>
```

class – класът реализиращ редактора. Трябва да е наследник на `org.eclipse.ui.IEditorPart`

extension – разширението на файловете, които може да се отварят с този редактор

name – име подходящо за потребителя.

Зависимости

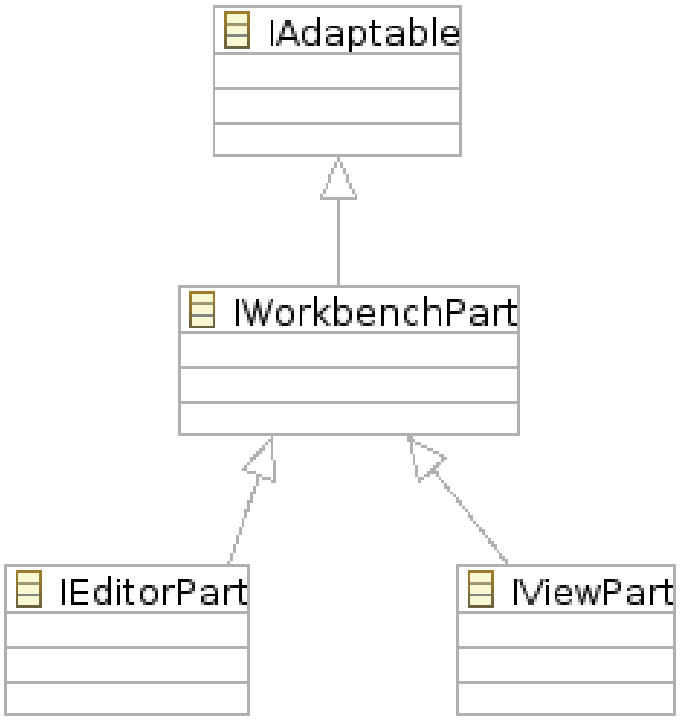
В реализирането на редактора ще използваме част от предоставените от Eclipse класове.

Необходимо е да добавиме следните зависимости към MANIFEST.MF

```
Require-Bundle : org.eclipse.core.runtime,
 org.eclipse.ui,
 org.eclipse.ui.ide,
 org.eclipse.core.resources
```

ListEditor

ListEditor
Логиката на редактора се реализира от класа `org.elsys.pluginsample.editor.ListEditor`.
Класът трябва да имплементира `org.eclipse.ui.IEditorPart`.
IEditorPart има подразбираща се изплементация, от която може да се възползваме – `org.eclipse.ui.part.EditorPart`



ListEditor

```
public  class ListEditor extends EditorPart {

   private List<String> fContent;

   private boolean fIsDirty;

   private ListViewer fViewer;

   private AddAction fAddAction;

   private DeleteAction fDeleteAction;

   private IFile fInputFile;

   /* ... code missed ... */
}
```

ListEditor

- **fContent – списъкът от низове, който искаме да редактираме**
- **fIsDirty – показва дали файлът е редактиран и може ли да се запамети.**
- **fViewer – променлива от тип `org.eclipse.jface.viewers.ListViewer` . Ще използваме този viewer, за да изобразим списъка fContent**
- **fAddAction, fDeleteAction – създадени от нас класове извършващи добавяне/изтриване на низ от списъка**
- **fInputFile – поле от тип `org.eclipse.core.resources.IFile` изобразяващо текущо редактирания файл. Посредством това поле можем да достъпим файла, неговото съдържание, родител, път до файла и др.**

ListEditor

```
public  class ListEditor extends EditorPart {
   public void doSave(IProgressMonitor monitor) {  }
   private IWorkspaceRunnable createFileSaveRunnable() { }
   public void doSaveAs() {}
   public ListViewer getViewer() {  }
   public List<String> getContent() {  }
   public void init(IEditorSite site, IEditorInput input)
         throws PartInitException { }
   public boolean isDirty() {}
   public boolean isSaveAsAllowed() {  }
   public void setDirty() {}
   public void createPartControl(Composite parent) {  }
   private void createActions() {}
   private void createContextMenu() {  }
   private void fillContextMenu(IMenuManager menuMgr) {  }
   protected void fillBody(Composite parent) {  }
   public void setFocus() {}
}
```

ListEditor

- `public void doSaveAs() {}` – Методът се използва ако изскаме да дадем възможност на редактора да запаметява съдържанието под нов файл.

```
public boolean isSaveAsAllowed() {
    return false;
```

} -Редакторът намя да поддържа Save As

-
```
public ListViewer getViewer() {
return fViewer;
}
public List<String> getContent() {
    return fContent;
}
```

Методи за достъп да съответните полета от класа

- `public void setFocus() {}`- Извиква се от средата когато редакторът получи фокуса на мишката и клавиетурата.

ListEditor

```
public boolean isDirty() {
    return fIsDirty;
}
public  void setDirty() {
    fIsDirty = true;
    firePropertyChange(IEditorPart.PROP_DIRTY);
}
```

- Файлът е редактиран. Необходимо е неактивният Save бутон в лентата с инструменти да стане активен.
- Необходимо е менюто File->Save да стане активно.
- Необходимо е в прозореца на редактора да се изобрази „ * “.
- Добавяме метода `setDirty()`, който уведомява „всички интересуващи се“ за това, че състоянието на редактора се е променило.
- Методът isDirty() ще бъде извикан, когато трябва да се определи състоянието на Save менюто. Ако методът върне true то менюто ще е активно.

ListEditor

```
public  void init(IEditorSite site, IEditorInput input)
      throws PartInitException {
   fInputFile = (IFile) input.getAdapter(IFile.class);
   if (fInputFile == null)
      throw new IllegalArgumentException(
            "The input must be adaptable to IFile");
   try {
      InputStream in = fInputFile.getContents();
      ObjectInputStream objectStream = new
ObjectInputStream(in);
      fContent = (List<String>) objectStream.readObject();
   } catch (Exception e) {
      e.printStackTrace();
      fContent = new ArrayList<String>();
   }
   fIsDirty = false;
   setSite(site);
   setInput(input);
}
```

ListEditor

Методът init е първият, който ще се извика в работата на редактора. Има за цел да инициализира редактора и да зададе входните му данни на база получения обект input. Тъй като нашият редактор работи с файлове проверяваме дали полученият обект може да се адаптира към обект от тип IFile.

```
fInputFile = (IFile) input.getAdapter(IFile.class);
```

При отварянето си редакторът трябва да е „чист“ - не може да се извърши запаметяване.

```
fIsDirty = false;
setSite(site);
setInput(input);
```

Базовият клас изисква да извикаме и методите setSite() и setInput().

ListEditor

Следващият метод, който ще се извика след init() е createPartControl()

```
public  void createPartControl(Composite parent) {
      Composite composite = new Group(parent, SWT.NONE);
      composite.setLayout(new GridLayout(1, false));
      composite.setLayoutData(new
GridData(GridData.FILL));
      fillBody(composite);
      createActions();
      createContextMenu();
}
```

Методът има за цел да създаде визуалната част на редактора.

Методите fillBody(), createActions(), createContextMenu() ще бъдат допълнени в процеса на разработка.

ListEditor

```
protected void fillBody(Composite parent) {
   ListViewer viewer = new ListViewer(parent, SWT.BORDER |
SWT.SINGLE);
   GridData data = new GridData(GridData.FILL_BOTH);
   viewer.getList().setLayoutData(data);
   viewer.setContentProvider(new ListContentProvider());
   viewer.setLabelProvider(new LabelProvider());
   viewer.setInput(fContent);
   fViewer = viewer;
}
```

Методът `fillBody()` създава нов viewer от тип ListViewer.

За ContentProvider - ще създадем клас ListContentProvider.

За LabelProvider - Използваме `org.eclipse.jface.viewers.LabelProvider`, който по подразбиране се обръща към метода `toString()` на подадените му обекти. Тази функционалност ни е достатъчна.

ListEditor

```
private static class ListContentProvider implements
        IStructuredContentProvider {

      public Object[] getElements(Object inputElement) {
         if (inputElement instanceof List) {
            List list = (List) inputElement;
            return list.toArray(new Object[list.size()]);
         }
         throw new IllegalArgumentException("Unsupported
input element");
      }

   /* ... code missed ... */
}
```

ListEditor

```
private  void createActions() {
      fAddAction = new AddAction(this, fContent, fViewer);
      fDeleteAction = new DeleteAction(this, fContent,
fViewer);

}
```

Логиката за добавяне и изтриване на низ от списъка ще бъде капсулирана в отделни класове.

Методът `createActions()` е добавен от нас и има за цел да инициализира обектите fAddAction и fDeleteAction.

ListEditor

```
private void createContextMenu() {
    MenuManager menuMgr = new MenuManager("#PopupMenu");
    menuMgr.setRemoveAllWhenShown(true);
    menuMgr.addMenuListener(new IMenuListener() {
        public void menuAboutToShow(IMenuManager m) {
            ListEditor.this.fillContextMenu(m);
        }
    });
    org.eclipse.swt.widgets.List list = fViewer.getList();
    Menu menu = menuMgr.createContextMenu(list);
    list.setMenu(menu);
    getSite().registerContextMenu(menuMgr, fViewer);
}
```

Редакцията ще се извършва посредством действия от контексното меню. Методът `createContextMenu()` има за цел да създаде това меню и да го регистрира в „сайта“ на редактора. Всеки редактор се намира в съответния контейтер предоставен от средата. Този контейнер се нарича „сайт – site“ и може да бъде достъпен чрез `getSite()`.

ListEditor

Преди менюто да бъде показано то трябва да бъде запълнено със съответните елементи.

Това се извършва от метода `fillContextMenu()`. Този метод се вика преди всяко показване на менюто.

Това ни позвалява да променяме характеристиките и броя на действията показани в менюто на база на някакво условие.

```
private  void fillContextMenu(IMenuManager menuMgr) {
      menuMgr.add(fAddAction);
      menuMgr.add(fDeleteAction);
      menuMgr.add(new
Separator(IWorkbenchActionConstants.MB_ADDITIONS));
}
```

ListEditor

Методът doSave() ще се извика при запаметяване на файла.

```
public  void doSave(IProgressMonitor monitor) {
      IWorkspaceRunnable runnable =
createFileSaveRunnable();
   try {
      ResourcesPlugin.getWorkspace().run(runnable,
monitor);
      fIsDirty = false;
      firePropertyChange(IEditorPart.PROP_DIRTY);
   } catch (CoreException e) {
      e.printStackTrace();
   }
}
```

ListEditor

Eclipse е отворена платформа. Броят на plugin-ите е неограничен. Неограничена е и тяхната функционалност и действия. Всички plugin-и са равнопоставени.

Всички разработвани проекти стоят в така наречения „workspace“. Всяка инстанция на платформата разполага само с един workspace. Следователно възможно е извършване на асинхронна промяна на едни и същи данни от страна на различни plugin-и, което ще доведе до загуба на информация.

Един от подходите за решаване на този проблем е докато един plugin променя workspace-а всики останали да чакат.
Организира се опашка от заявки за промяна на workspace-а. Такава заявка трябва да е от тип `org.eclipse.core.resources.IWorkspaceRunnable`. Достъпът до workspace-а се извършва чрез `ResourcesPlugin.getWorkspace()`.

Методът `ResourcesPlugin.getWorkspace().run()` и синхронен за извикващата нишка и я блокира до приключването на неговата работа.

ListEditor

След като заявката приключи е необходимо да се съобщи на „всички интересуващи се“ ще състоянието е променено

```
fIsDirty = false;
firePropertyChange(IEditorPart.PROP_DIRTY);
```

Добавяме метода `createFileSaveRunnable()`, който ще използваме да създаваме заявки за запаметяване на файла.

Отделянето кода за изпълнение на заявката и кода за създаване на заявката позволяват по-лесна промяна на функционалността и по-добро капсулиране на данните.

ListEditor

```
private  IWorkspaceRunnable createFileSaveRunnable() {
    IWorkspaceRunnable runnable = new IWorkspaceRunnable() {
        public void run(IProgressMonitor monitor) throws CoreException {
            try {
                monitor.beginTask("Saving file...", 10);
                monitor.worked(2);
                if (monitor.isCanceled())
                    throw new OperationCanceledException();
                ByteArrayOutputStream os = new ByteArrayOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(os);
                oos.writeObject(fContent);
                ByteArrayInputStream is = new
ByteArrayInputStream(os.toByteArray());
                oos.close();
                os.close();
                fInputFile.setContents(is, IResource.FORCE, new
SubProgressMonitor(monitor, 9));
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                monitor.done();
            }
        }
    };
    return runnable;
}
```

ListEditor

Съдържанието на списъка fContent трябва да се запамети във файла. За целта използваме метода IFile.setContents().

Необходимо е преди това да представиме съдържанието, което искаме да запишем като обект от тип InputStream.

```
ByteArrayOutputStream os = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(os);
oos.writeObject(fContent);
ByteArrayInputStream is = new ByteArrayInputStream(os.toByteArray());
oos.close();
os.close();
fInputFile.setContents(is, IResource.FORCE, new
                                        SubProgressMonitor(monitor, 9));
```

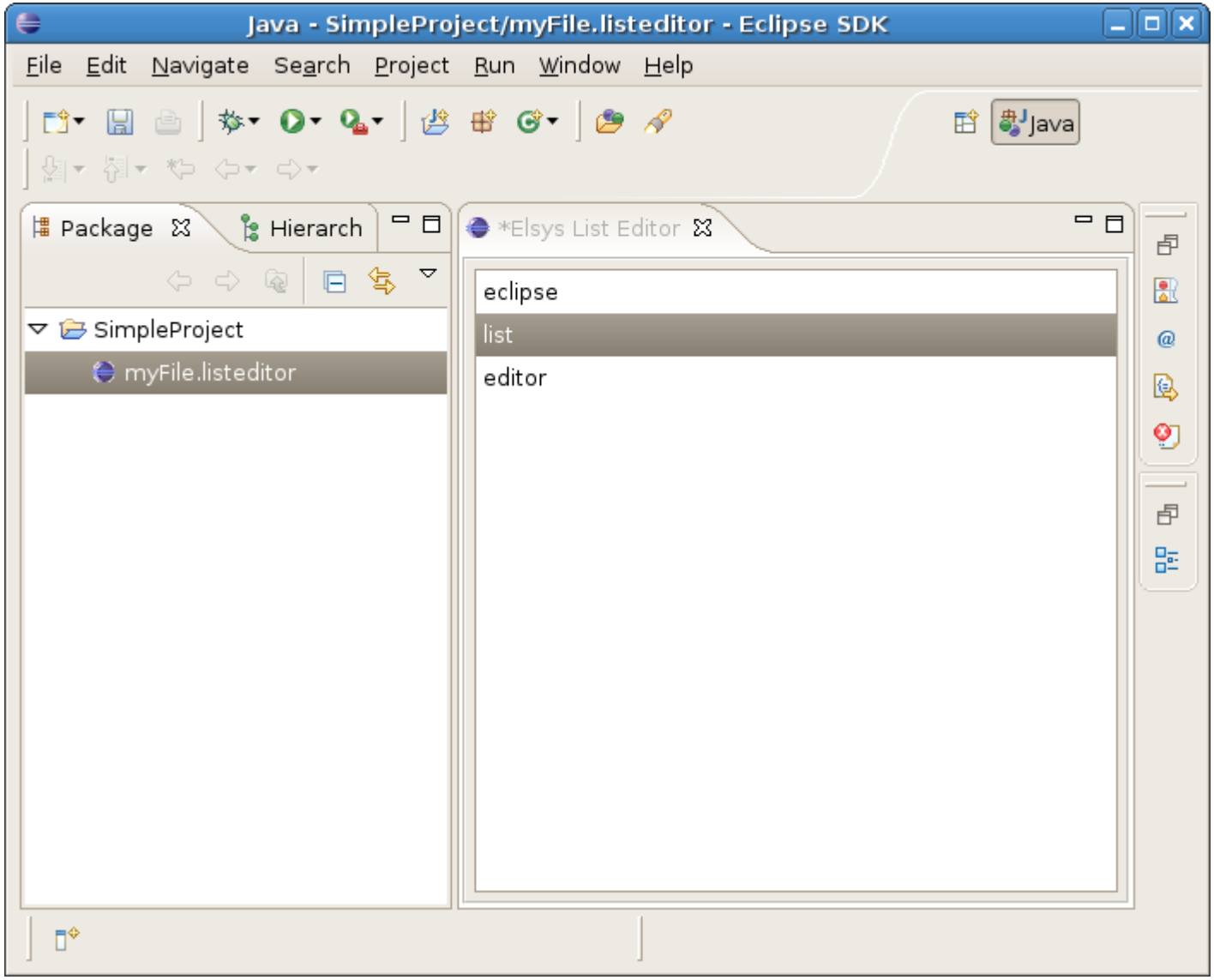
ListEditor

След като стартираме plugin-а трябва да създадем нов проект. File->New->Project. Даваме име на проекта – примерно SampleProject.

След това създаваме нов файл в този проект използвайки File->New->File. Важното е файлът да е с разширение „listeditor“.

Създаденият файл може да се отвори с няколко редактора – включително и с обикновен текстов редактор. Може да определим желания редактор чрез менюто Open With от контексното меню за дадения файл.

ListEditor


Java - SimpleProject/myFile.listeditor - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Java
Package
Hierarch
*Elsys List Editor
SimpleProject
myFile.listeditor
eclipse
list
editor

Упражнение

Използване на готовия код:

Проект съдържащ класовете ListEditor, AddAction, DeleteAction.

Необходимо е да добавите:

- **зависимостите в MANIFEST.MF**
- **необходимия extension point в plugin.xml**
- **липсващите методи и полета в ListEditor**

Copyright