ECLIPSE RCP COURSE

Exercises script

GMV AEROSPACE AND DEFENCE S.A.
Isaac Newton, 11; PTM Tres Cantos; Madrid 28760
Tel. +34 918072100; Fax. +34 918072199
www.gmv.com

1. INTRODUCTION

This document assumes the reader has already attended GMV's EclipseRCP course. It is intended to enable the reader to REPEAT the exercises from the course on his/her own.

To follow the exercises you need the course material, which includes:

- “Infrastructure” plugins the exercises are built on top of.
- Solutions to the exercises (to be used as cheatsheet).

The exercises material and solutions are located at repository:

- https://github.com/dmarina-esa/EclipseRCPTraining/

The source code for the exercises has been tested with java 11 and Eclipse IDE 2019-09 (4.13) for RCP and RAP Developers, running under Microsoft Windows 10.

- Link to Eclipse IDE 2019-09 for RCP and RAP Developers

Extra information sources:

- Eclipse RCP course slides
- Eclipse RCP book (ISBN13: 978-0321334619), available at GMV’s library.

If you notice inconsistencies, lack of information or any other issue, please report them to David Marina (dmarina@gmv-insyen.com).

2. EXERCISE 01: AN EMPTY PLUGIN

2.1. GOALS

- First glance at the elements that conforms a plugin.
- First glance at the Target Platform.

2.2. MATERIAL

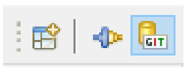
- Eclipse IDE for RCP and RAP Developers 4.13 (2019-09) or higher (Java 11 compatible)
- Target Platform folder for the course.

2.3. ABSTRACT

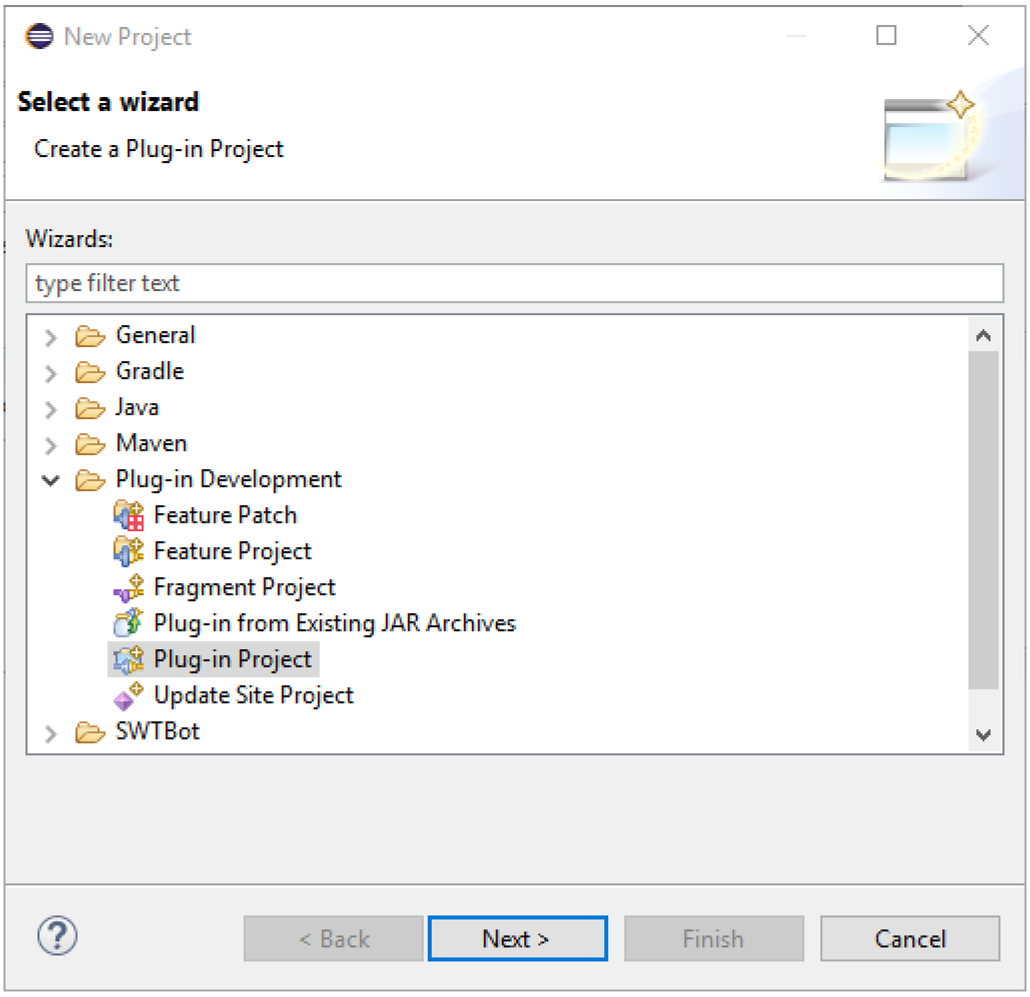
In this exercise, the student creates a new empty RCP Plugin to understand which the required files are. The plugin dependencies mechanisms are explored to understand the role of the Target Platform.

2.4. STEPS

1. Open Eclipse and create a new Workspace
2. Clone the course repository:
    a. Copy the project’s repository URL to your clipboard
    b. Open Eclipse’s Git perspective (on the right top corner)
    c. In Git Repoitories view, select “Clone a Git repository
        - The repository location is auto-completed from the clipboard



3. Create a new plug-in project
    a. Switch back to the Plug-in Development perspective
    b. In the main menu: File -> New -> Project…
    c. Select Plug-in Development -> Plug-in Project in the wizard).
    d. Click next, specify a project name: com.gmv.course.exercise01
    e. Click next, select “Generate an activator, […]”.
    f. Click Finish.



New Project

Select a wizard

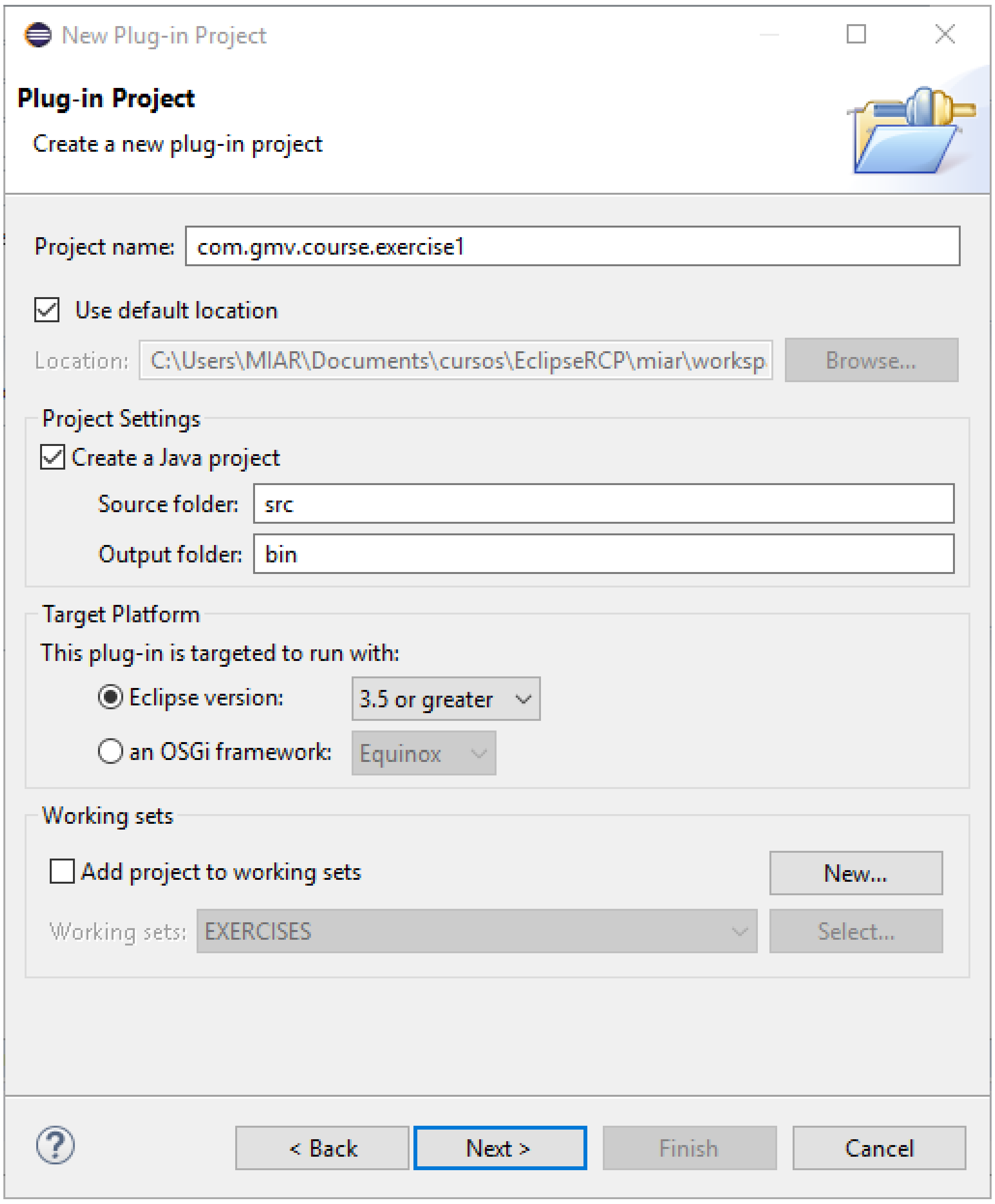
Create a Plug-in Project

Wizards:

type filter text

- General
- Gradle
- Java
- Maven
- Plug-in Development
  - Feature Patch
  - Feature Project
  - Fragment Project
  - Plug-in from Existing JAR Archives
  - Plug-in Project
  - Update Site Project
- SWTBot

< Back | Next > | Finish | Cancel



New Plug-in Project

Plug-in Project

Create a new plug-in project

Project name: com.gmv.course.exercise1

[✓] Use default location

Location: C:\Users\MIAR\Documents\cursos\EclipseRCP\miar\worksp

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

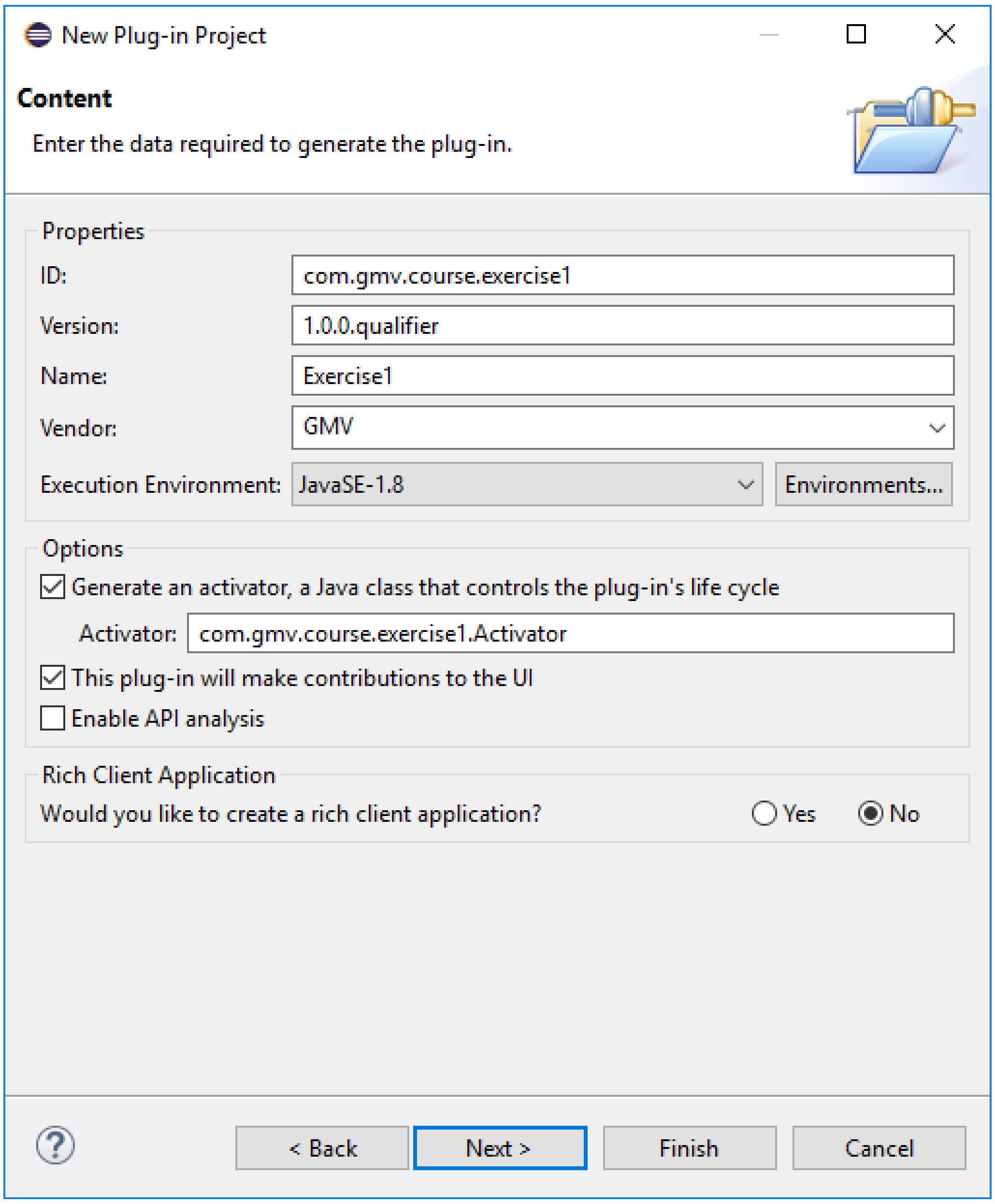
● Eclipse version: 3.5 or greater

○ an OSGi framework: Equinox

Working sets

[ ] Add project to working sets

Working sets: EXERCISES



New Plug-in Project

Content

Enter the data required to generate the plug-in.

Properties

ID:	com.gmv.course.exercise1
Version:	1.0.0.qualifier
Name:	Exercise1
Vendor:	GMV
Execution Environment:	JavaSE-1.8

Options

- [✓] Generate an activator, a Java class that controls the plug-in's life cycle
  - Activator: com.gmv.course.exercise1.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]

4. Open the META-INF/MANIFEST.MF file (it should be open automatically)
5. In the "Dependencies" tab, remove "org.eclipse.core.runtime" and save.
6. Open the Activator class, check that there are errors because the imports cannot be resolved.
7. In the MANIFEST.MF editor, in the "Dependencies" tab, click "Add..." and select "org.eclipse.core.runtime" again. Save.
8. Check that the errors are gone from the Activator class.
9. Open the Target Platform configuration
    a. In the main menu: Window -> Preferences.
    b. In the Preferences dialog, open Plug-in Development -> Target Platform.
10. Create a new target platform
    a. Click "Add..." and create a target platform from the current one "Current Target: Copy settings from the current target platform".
    b. Enter a name and click "Add...".
    c. In the wizard, select "Directory" and click "Next".

d. Select the folder with the additional Target Platform plugins for the course and click "Finish":
    - The folder named "target_plugins" at the master branch of the cloned repository.
    - It is recommended to copy this folder to a new location; otherwise, you will lose the Target Platform files when switching branches.

e. Click "Finish".

f. **Select** (tick) the new target platform and click "OK".

11. Add a dependency to "com.gmv.exercise.utils".

3. EXERCISE 02: CONTRIBUTE A VIEW TO THE IDE

3.1. GOALS

- Provide an extension for an extension point.
- Contributing a view.
- Executing a plugin with Eclipse.

3.2. MATERIAL

- Target platform folder for the course.
- Icons folder.

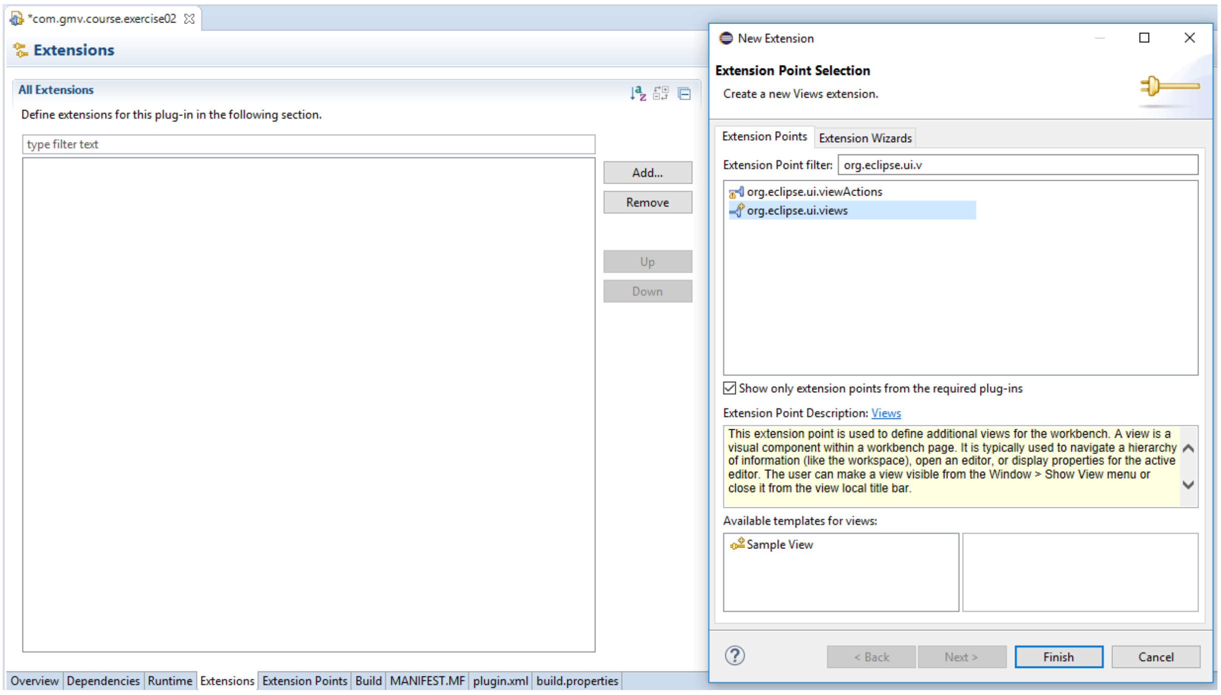
3.3. ABSTRACT

In this exercise, the student creates a new plugin that provides an extension to "org.eclipse.ui.views" extension point to contribute a view to the Eclipse IDE.

In the final steps, the student launches an application containing the Eclipse IDE, but extended with the functionality from the newly created plugin.

3.4. STEPS

1. Add the course target platform folder to the default target platform. (skip this step if you finished Exercise 01)
2. Create a new plug-in project as in Exercise 01 (optionally, the plugin created in Exercise 01 can be reused).
3. Notice that initially the plugin does not have a file called "plugin.xml". It is created automatically when you specify an extension or extension point using the manifest editor.
4. Add a dependency on "com.gmv.course.exercise.utils". (skip this step if you finished Exercise 01 and you are reusing the created plugin)
5. Open the MANIFEST.MF and in the "Overview" tab click on "Extensions".
6. In the "Extensions" tab, click "Add…" and select "org.eclipse.ui.views". Press "Finish". (note that file plugin.xml is automatically created)

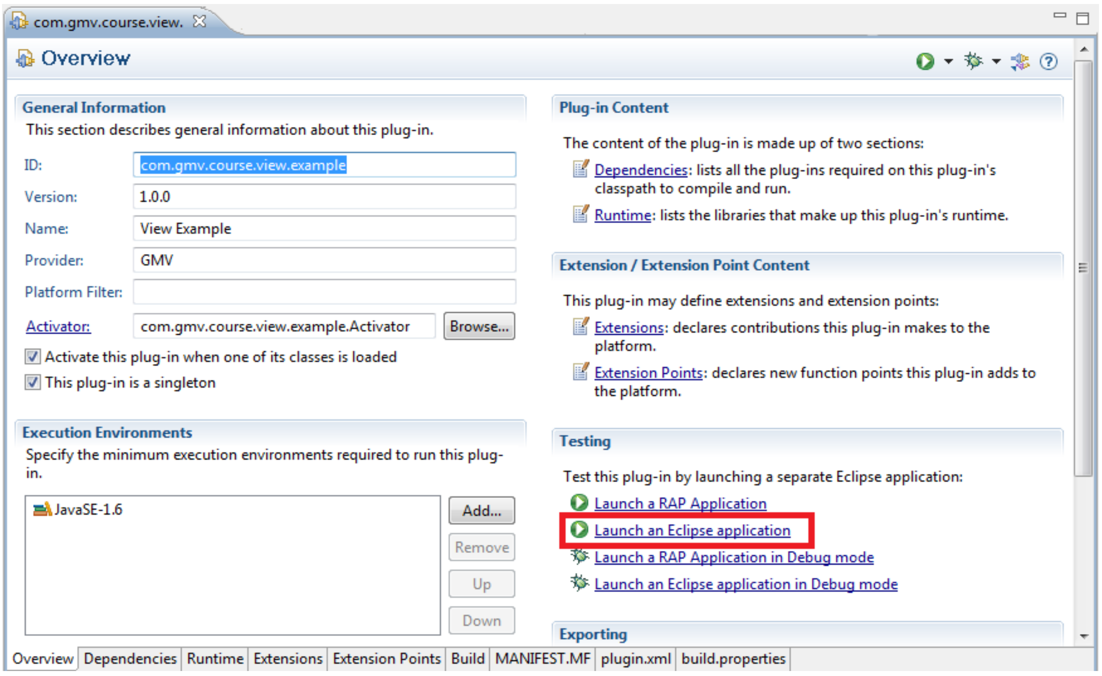

*com.gmv.course.exercise02
Extensions
All Extensions
Define extensions for this plug-in in the following section.
type filter text
Add...
Remove
Up
Down
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
plugin.xml
build.properties
New Extension
Extension Point Selection
Create a new Views extension.
Extension Points
Extension Wizards
Extension Point filter: org.eclipse.ui.v
org.eclipse.ui.viewActions
org.eclipse.ui.views
Show only extension points from the required plug-ins
Extension Point Description: Views
Available templates for views:
Sample View
< Back
Next >
Finish
Cancel

This extension point is used to define additional views for the workbench. A view is a visual component within a workbench page. It is typically used to navigate a hierarchy of information (like the workspace), open an editor, or display properties for the active editor. The user can make a view visible from the Window > Show View menu or close it from the view local title bar.

7. Right-click on the just created element "org.eclipse.ui.views" and select New -> view.
8. In Extension Element Details, fill the mandatory data marked with "*".

- Id = "com.gmv.course.exercise02.view"
- Name = "Exercise02"
- For the class, click on **"<u>class*:</u>"** and fill the class name and package in the wizard
    - Package = "com.gmv.course.exercise02.views"
        - It is a good practice that the plugin id is used as the root for all the package names, so you may want to call it "com.gmv.course.exercise01.views" if you reused the plugin from exercise01
    - Name = "Exercise02View"
- Press "Finish"

9. In the view editor, go to createPartControl() method and add: Exercise02Utils.createViewContents(parent);
10. Save all the files.
11. Double click in the Manifest file. In the "Overview" tab of the manifest editor, click "Launch an Eclipse application".


com.gmv.course.view.

Overview

General Information
This section describes general information about this plug-in.

ID:	com.gmv.course.view.example
Version:	1.0.0
Name:	View Example
Provider:	GMV
Platform Filter:	
Activator:	com.gmv.course.view.example.Activator

☑ Activate this plug-in when one of its classes is loaded
☑ This plug-in is a singleton

Execution Environments
Specify the minimum execution environments required to run this plug-in.

JavaSE-1.6

Plug-in Content
The content of the plug-in is made up of two sections:
- Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.
- Runtime: lists the libraries that make up this plug-in's runtime.

Extension / Extension Point Content
This plug-in may define extensions and extension points:
- Extensions: declares contributions this plug-in makes to the platform.
- Extension Points: declares new function points this plug-in adds to the platform.

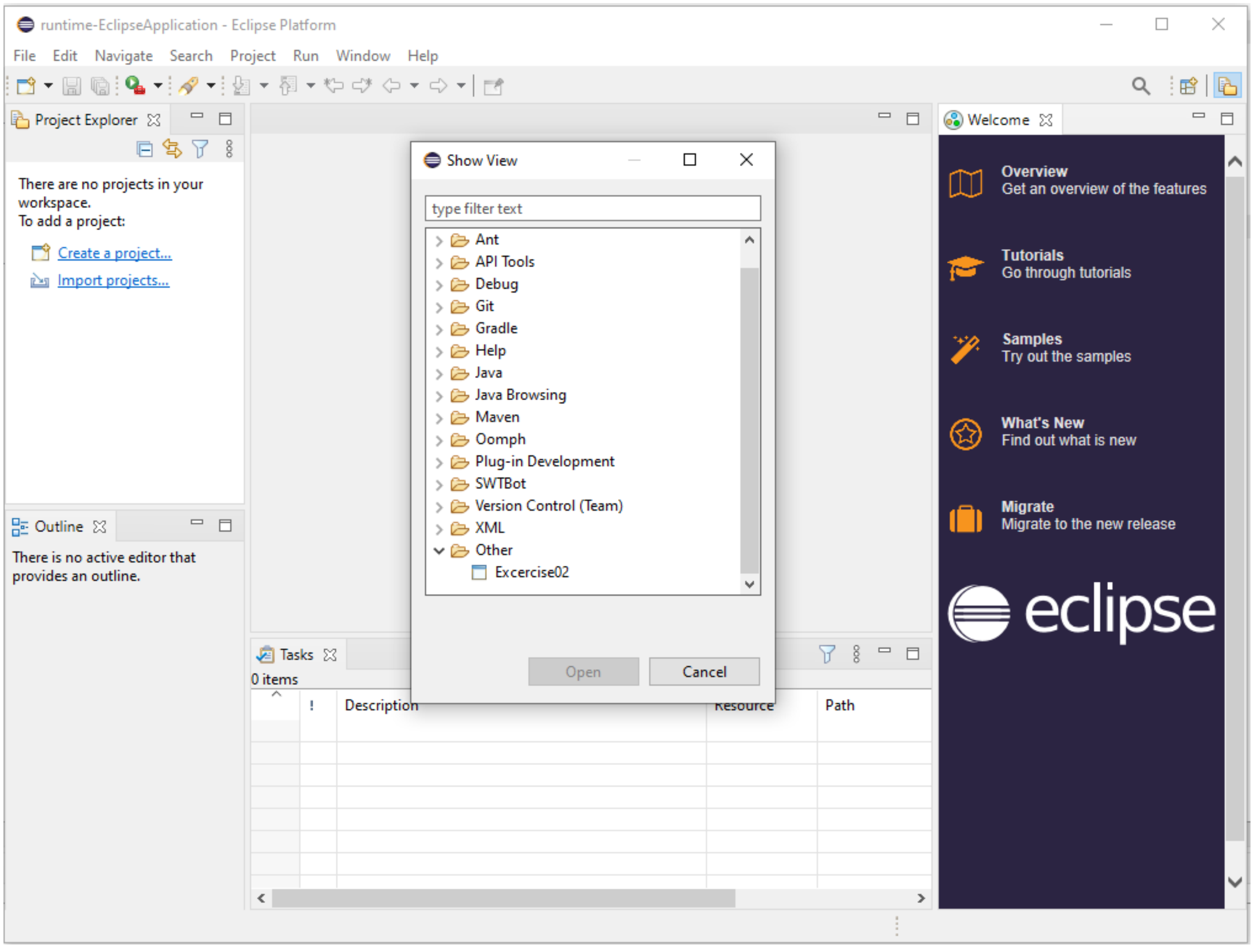
Testing
Test this plug-in by launching a separate Eclipse application:
- Launch a RAP Application
- Launch an Eclipse application
- Launch a RAP Application in Debug mode
- Launch an Eclipse application in Debug mode

Exporting

Browse...
Add...
Remove
Up
Down
Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

- *Some versions of Eclipse may show some errors, ignore them for the moment*

12. Wait for a new Eclipse application to start and close the Overview page if it is displayed.
    - *Depending on the version this may take long: we are launching a full Eclipse!!*
13. In the new Eclipse, select Window -> Show view -> other -> other -> Exercise 02.


runtime-EclipseApplication - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Project Explorer
There are no projects in your workspace.
To add a project:
Create a project...
Import projects...
Outline
There is no active editor that provides an outline.
Show View
type filter text
Ant
API Tools
Debug
Git
Gradle
Help
Java
Java Browsing
Maven
Oomph
Plug-in Development
SWTBot
Version Control (Team)
XML
Other
Excercise02
Open
Cancel
Tasks
0 items
Description
Resource
Path
Welcome
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Migrate
Migrate to the new release
eclipse

4. EXERCISE 03: SPORT SIMULATOR APPLICATION

4.1. GOALS

1. Plugin project creation
2. Extending an extension point.
3. Creating a view.
4. First steps with SWT and JFace viewers.
5. Creating an RCP application.

4.2. MATERIAL

6. Target platform folder for the course.
    - Eclipse plugin exporting sport service API (called "com.gmv.sportsimulator.api")
    - Eclipse plugin exporting tennis simulator service (called "com.gmv.sportsimulator.tennis")
7. Icons folder.

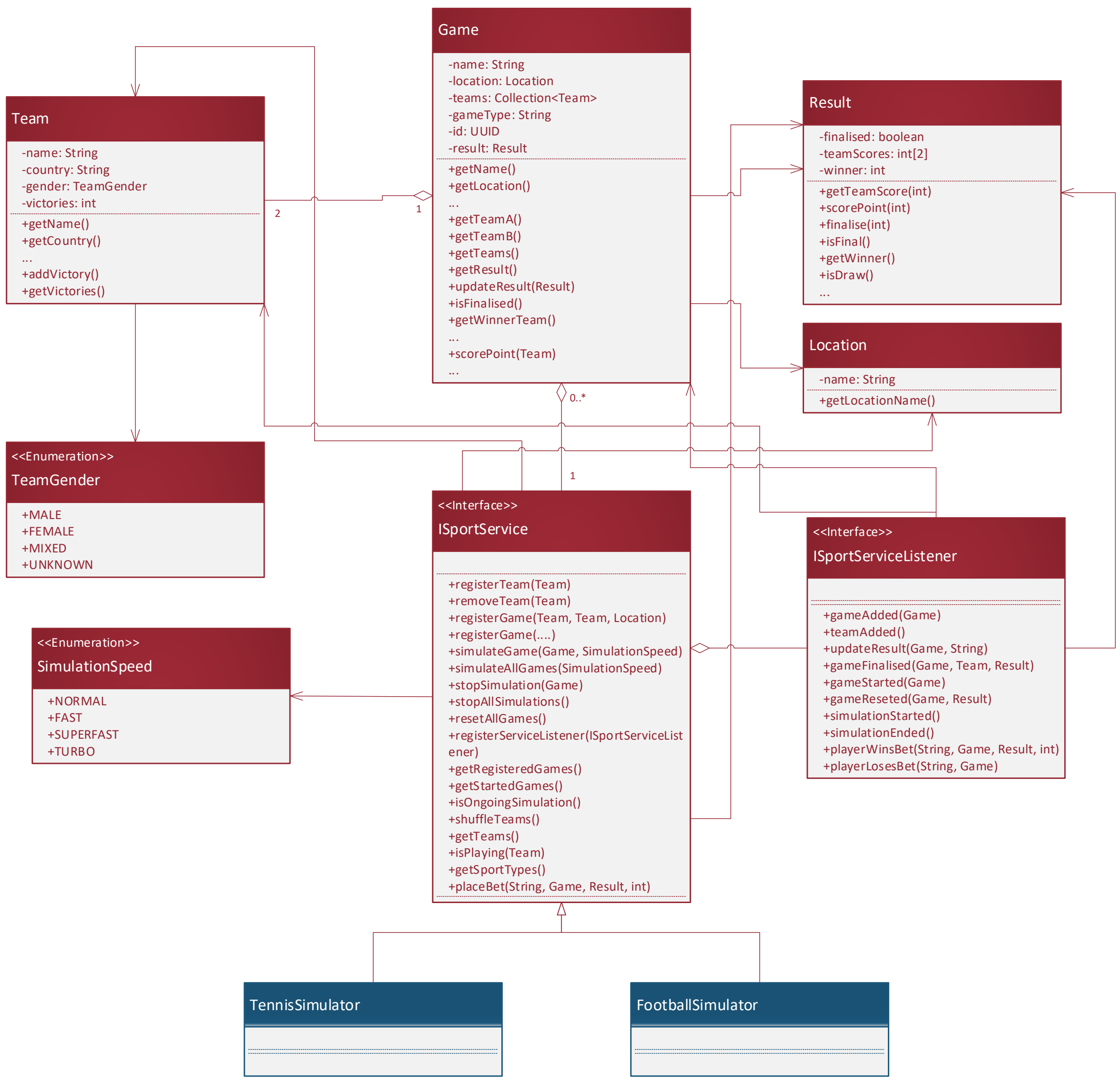
4.3. ABSTRACT

In this exercise, the student creates a new plugin that extends org.eclipse.ui.views extension point and codes a ViewPart showing a TreeViewer displaying game/players hierarchy. For the domain model to be filled with data, the student hardcodes the creation of Teams and Games in the body of the view constructor.

The student will use the Eclipse wizard to create an empty RCP application and will modify it to display the created views as a separate application out of the Eclipse IDE.

4.3.1. THE SPORT SIMULATOR SERVICE

The plugins included as the addition to the Target Platform compose a sport simulator service. The following is the class diagram describing the Simulator Service:



Plugin *com.gmv.sportsimulator.api* provides a generic service interface that shall serve as the contact point for the service clients, regardless of the type of sport that the actual implementation(s) provides. Classes provided by this plugin are displayed in **red** color in the diagram.

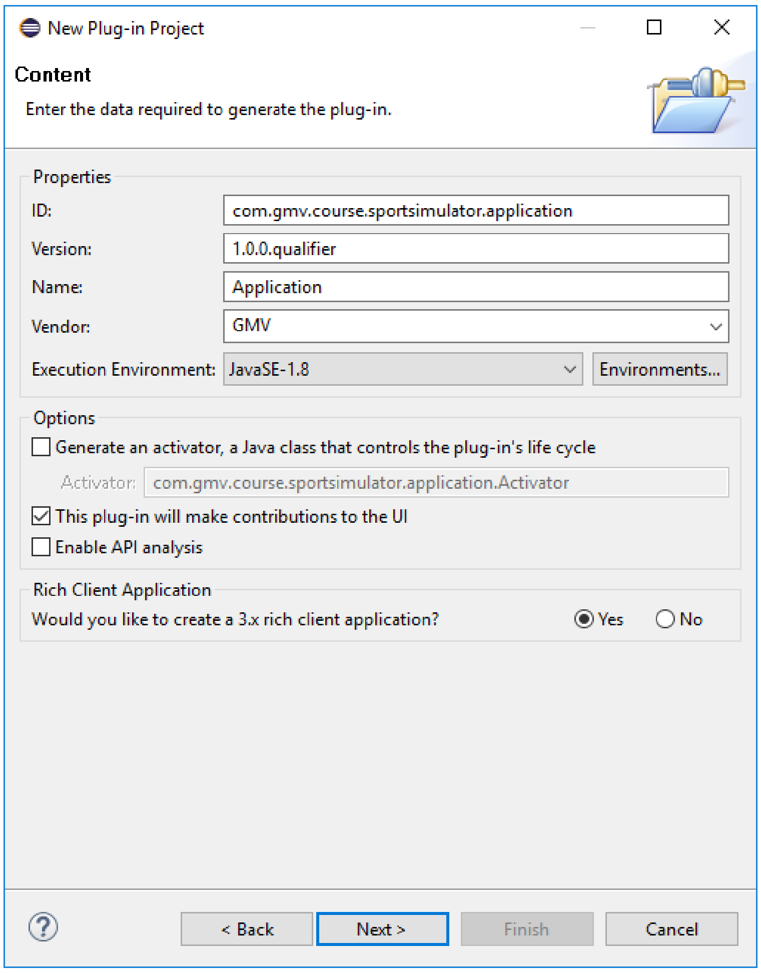
In general, a sport service is composed of Games that can be simulated to produce some game Result. A **Game** is played by two **Teams**, it takes place in a **Location** and it has some **Result** (either ongoing or final if the game is simulated). Each Team has a name, and same basic information, like the Country, the TeamGender or the number of accumulated victories.

The **ISportService** provides the tools to manage the set of Game that compose the simulation, as well as the Teams that are registered. Clients can register Team and Games manually, start the simulation of some Games, place a bet on a Game… It also allows to register **ISportServiceListeners** that will be notified about changes in the list of games, teams, simulation status, bet status…

The classes displayed in **blue** in the diagram represent the specific ISportService implementations. The provided target platform includes one implementation for a Tennis Simulator. Students can decide to implement other simulator types that will be exchangeable in the application by simply replacing the plugin with one or another implementation.

4.4. STEPS

1. Launch eclipse and specify a new workspace. (You can continue with the workspace from previous exercises)
2. Include the course's target platform folder in the configured target platform. (Omit this step if you completed the previous exercises)
3. Create a new plugin project:
   - File -> New -> Project.
   - Select "Plug-in development -> Plug-in project" and press "Next".
   - Give a name to the project (i.e. com.gmv.course.sportsimulator.application) and press "Next".
   - Uncheck the option for generating an "Activator".
   - Check the option "This plug-in will make contributions to the UI".
   - Answer "Yes" to the question on whether you would like to create a rich client application, and press "Next".


New Plug-in Project

Content

Enter the data required to generate the plug-in.

Properties

ID:	com.gmv.course.sportsimulator.application
Version:	1.0.0.qualifier
Name:	Application
Vendor:	GMV
Execution Environment:	JavaSE-1.8

Options

☐ Generate an activator, a Java class that controls the plug-in's life cycle

Activator: com.gmv.course.sportsimulator.application.Activator

☑ This plug-in will make contributions to the UI

☐ Enable API analysis

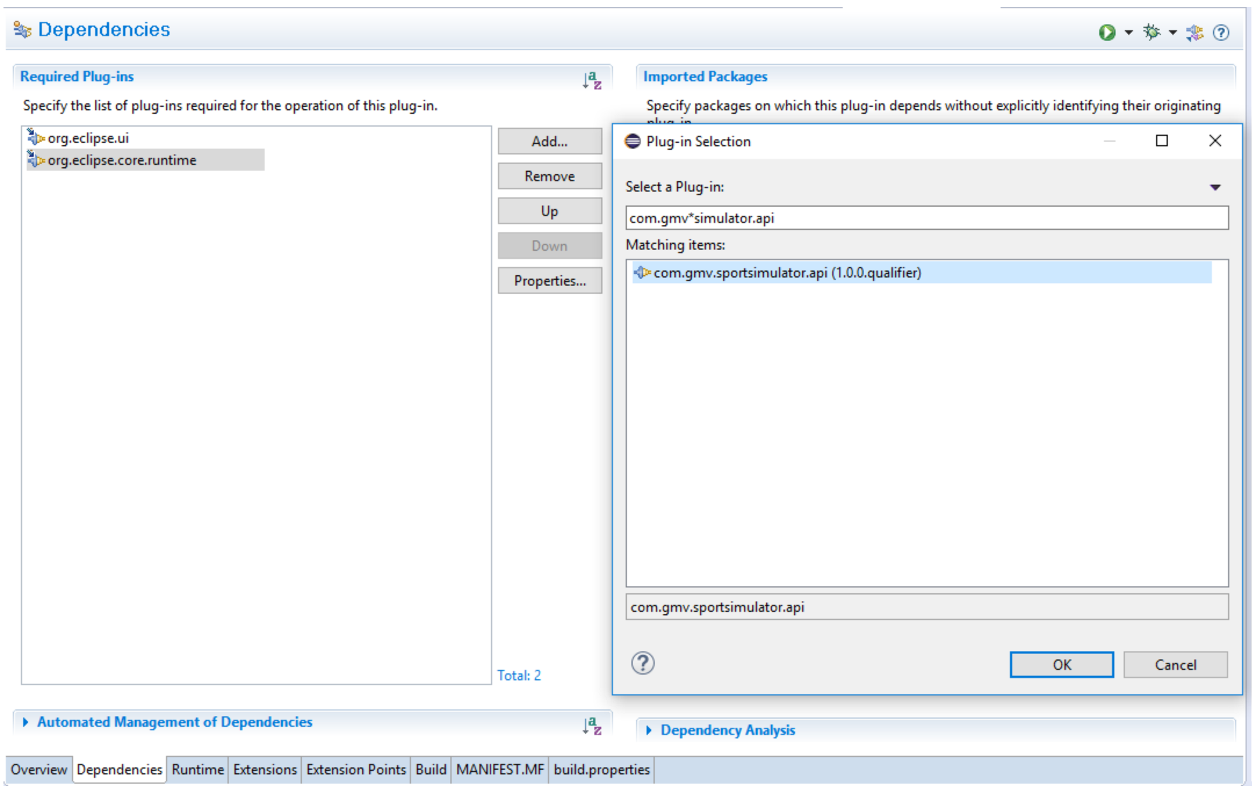
Rich Client Application

Would you like to create a 3.x rich client application? [✓ Yes] No

   - Create the plug-in using the template "RCP 3.x application (minimal)
     - In some versions this same template is named "Hello RCP".
   - Explore generated classes (Application, ApplicationActionBarAdvisor, ApplicationWorkbenchAdvisor, ApplicationWorkbenchWindowAdvisor and Perspective).
4. Explore generated classes (Application, ApplicationActionBarAdvisor, ApplicationWorkbenchAdvisor, ApplicationWorkbenchWindowAdvisor and Perspective)
5. Perform right click on the plug-in and select "Run as"-> "Eclipse Application"
   - Depending on the content of your workspace, some errors may be displayed. You can ignore them for now.

6. A new empty application opens.
   - Notice that, simultaneously, a ***Run Configuration*** has been generated by Eclipse. You can check this configuration navigating to the menu: Run -> Run Configurations…
   - You can find the new run configuration on the "Eclipse Application" section on the tree on the left side. Click on it to open its settings.
7. In tab "Main" mark option "Clear" and unmark "Ask for confirmation before clearing"
8. Verify that section "Program to Run" option displays "Run an application" and the application id is *com.gmv.sportsimulator.application.application*
9. Navigate to section Plug-ins. You will notice that only a "small" amount of plug-ins are selected and **not all** the available plugins. Close the dialog.
10. Create a new plugin project that will provide a view for the application:
    - File -> New -> Project.
    - Select "Plug-in development -> Plug-in project" and press "Next".
    - Give a name to the project (i.e. com.gmv.course.sportsimulator.display) and press "Next".
    - Mark the options for generating an "Activator" and for stating the plugin will make contributions to the UI.
    - Answer "No" to the question on whether you would like to create a rich client application or not, and press "Finish".
      - *This may sound confusing: we are developing a rich client application!!!... However, the option here simply means that the plugin we are creating will contain an IApplication and add it to the registry (entry point to the app). Our entry point was already created in step 3. ☺*

11. Copy the folder called "icons" (included with the course material) in the just created project.
12. Add plugin com.gmv.sportsimulator.api as dependency of your plugin.
    - Open file "META-INF/MANIFEST.MF".
    - Select tab labeled "Dependencies".
    - Add there the dependency. Click on the "Add..." button and select the "com.gmv.sportsimulator.api" plug-in.


Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

- org.eclipse.ui
- org.eclipse.core.runtime

Total: 2

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in

Plug-in Selection

Select a Plug-in:

com.gmv*simulator.api

Matching items:

com.gmv.sportsimulator.api (1.0.0.qualifier)

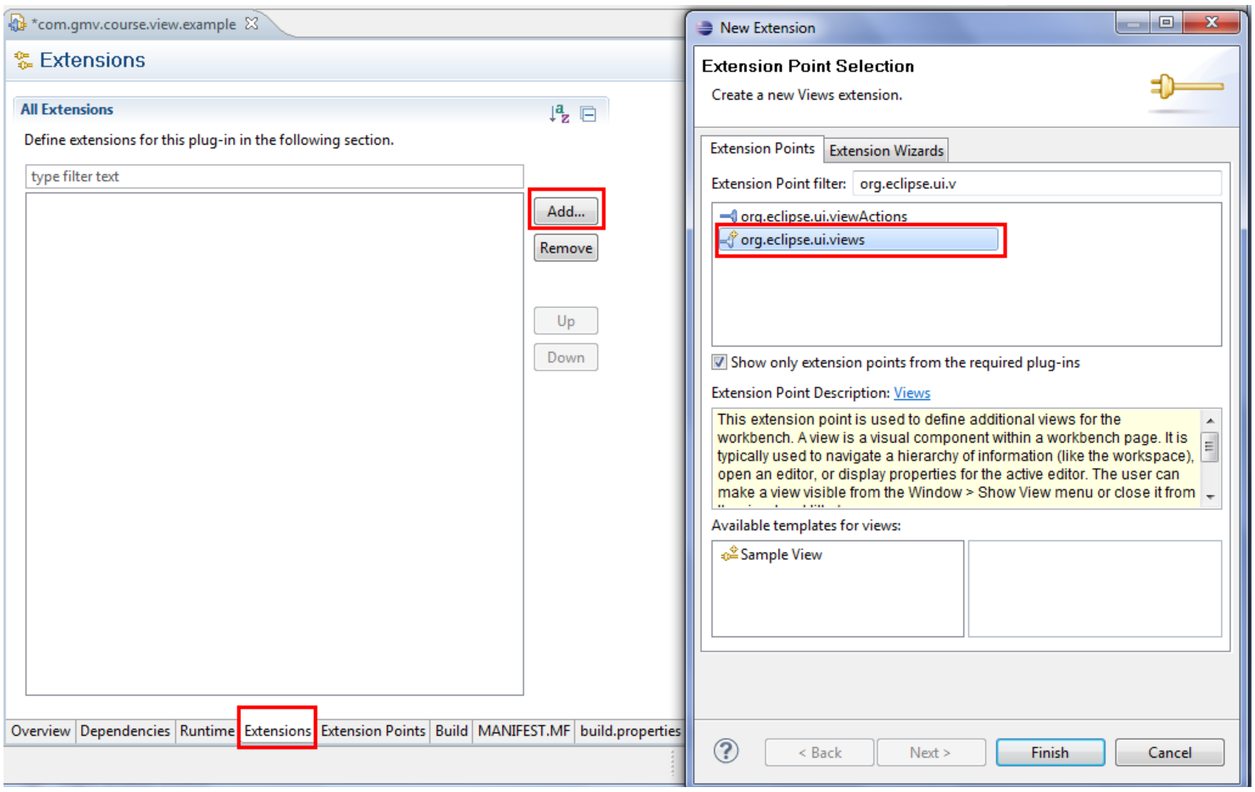
com.gmv.sportsimulator.api

Automated Management of Dependencies

Dependency Analysis

Figure 1: Add dependencies to a plug-in

13. In the plugin.xml file, state your plug-in has an eclipse view:
    - Notice your project does not have a file called "plugin.xml". It is created automatically when you specify an extension or extension point when editing the manifest.
    - Open the file "META-INF/MANIFEST.MF", which is opened by default in a multi-tab editor. This editor provides a "unified view" of three files: MANIFEST.MF, plugin.xml and build.properties.
    - Select "Extensions" tab. It should look quite empty.
    - Press "Add…". The extension point selection dialog should be showed. (see Figure 2: Add the views' extension point).
    - Select the extension point "org.eclipse.ui.views" and press "Finish".
    - Check that "plugin.xml" file has been added to your project.


*com.gmv.course.view.example

Extensions

All Extensions

Define extensions for this plug-in in the following section.

type filter text

Add... Remove Up Down

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | build.properties

New Extension

Extension Point Selection

Create a new Views extension.

Extension Points | Extension Wizards

Extension Point filter: org.eclipse.ui.v

org.eclipse.ui.viewActions
org.eclipse.ui.views

☑ Show only extension points from the required plug-ins

Extension Point Description: Views

This extension point is used to define additional views for the workbench. A view is a visual component within a workbench page. It is typically used to navigate a hierarchy of information (like the workspace), open an editor, or display properties for the active editor. The user can make a view visible from the Window > Show View menu or close it from

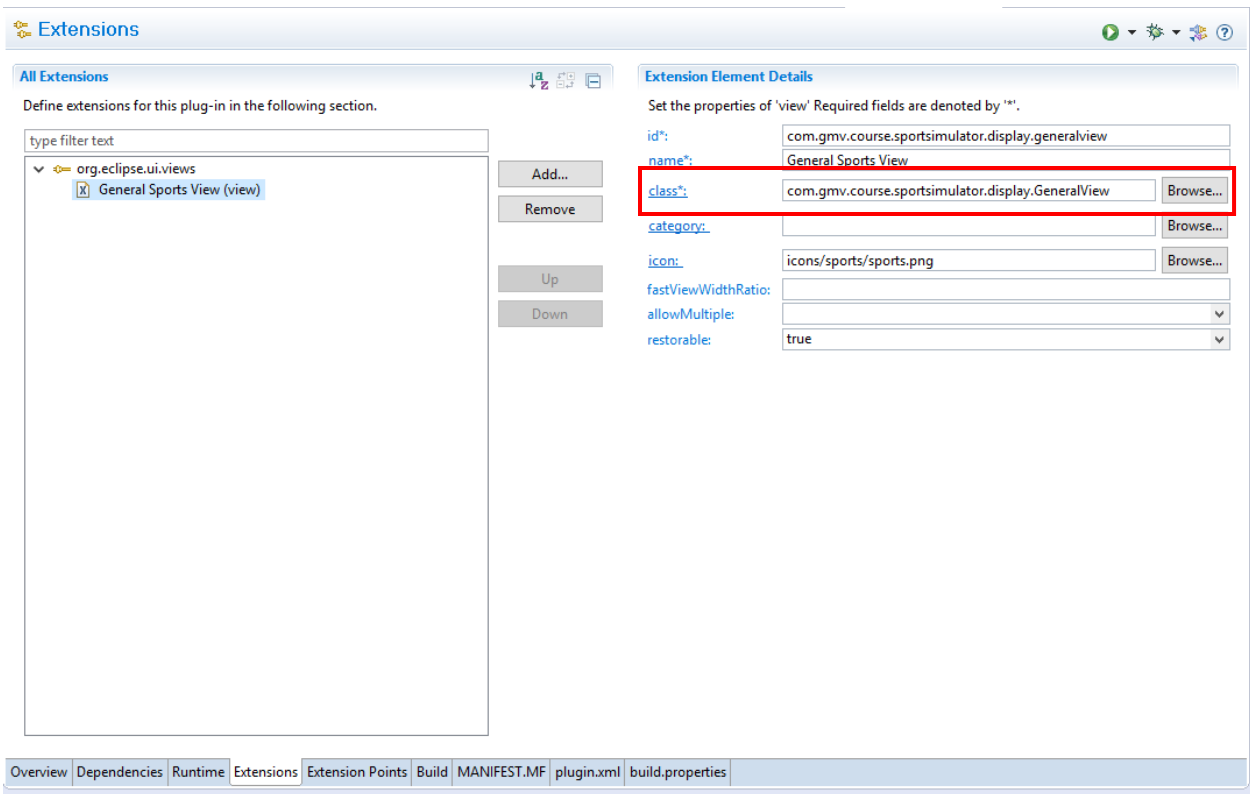
Available templates for views:

Sample View

< Back | Next > | Finish | Cancel

Figure 2: Add the views' extension point

14. Right-click on the just created extension element "org.eclipse.ui.views" and select New -> View.
15. Select the view element you just created and fill its "extension element details:"
    - id: com.gmv.course.sportsimulator.display.generalview
    - name: General Sports View
    - class: com.gmv.course.sportsimulator.display.GeneralView (this class will be the View implementation)
    - icons/sports/sports.png



Define extensions for this plug-in in the following section.

Extension Element Details

Set the properties of 'view' Required fields are denoted by '*'.

id*: com.gmv.course.sportsimulator.display.generalview
name*: General Sports View
class*: com.gmv.course.sportsimulator.display.GeneralView
category:
icon: icons/sports/sports.png
fastViewWidthRatio:
allowMultiple:
restorable: true

Figure 3: View's properties

16. Click on the hyperlink labeled "class". The class creation dialog should appear, click "Finish" button. The View class is created automatically. (If the view class exists when you click on “class”, the existing class will be opened).

17. Fill method "createPartControl" with proper SWT/JFace stuff:
    - For convenience, the target platform plugin *com.gmv.course.exercise.utils* will help us creating the display → Add it as a dependency
    - Then, let’s give some content to GeneralView:
        - Get the service reference:
            - `Exercise03Utils.getSportService()`
        - Populate the service with teams and games. Remember: games are composed at least of the name, two teams, and the location.
    - Create a JFace TreeViewer to show the hierarchical structure of games/teams. ***Note:*** *you need a Content Provider, and a Label Provider.*
        - *Use Content Provider to feed items to the Label Provider*
        - *Use Label Provider to select the information to display in the tree; typically text and icons*

      Optionally, you can use Exercise03Utils class from com.gmv.course.exercise.utils and let it create a simple tree by invoking:

        - `Exercise03Utils.createViewContents(parent);`

      It is recommended to use it as a reference for the first implementation. Later, you should try to create your own tree with its providers. You can also look into the repository solutions to see a more advanced implementation.

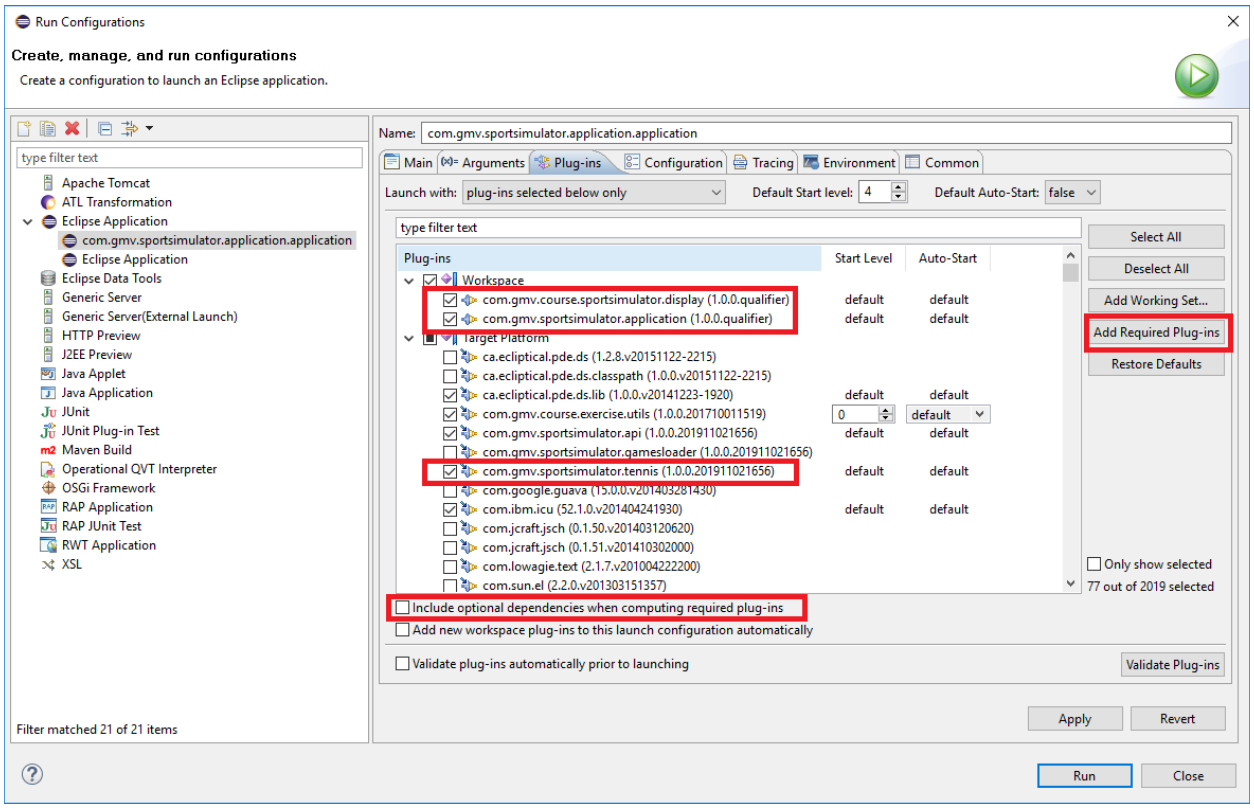
18. Back at plug-in *com.gmv.sportsimulator.application* open class *ApplicationWorkbenchWindowAdvisor.java*

19. Override method *postWindowOpen()* with the following code in order to open the new view on startup:

```
@Override
public void postWindowOpen()
```

```
{
  try
  {
   PlatformUI.getWorkbench().getActiveWorkbenchWindow()
          .getActivePage().showView("com.gmv.course.sportsimulator.display.generalview");
  }
  catch (PartInitException e)
  {
     e.printStackTrace();
  }
}
```

20. Navigate again to Run -> Run Configurations... and select the previously created application configuration.
21. **Unmark** option "Include optional dependencies when computing required plug-ins"
22. Select your display plugin *com.gmv.course.sportsimulator.display*
23. We will also need to have one simulator implementation. Select tennis simulator plugin: *com.gmv.sportsimulator.tennis*



Run Configurations

Create, manage, and run configurations

Create a configuration to launch an Eclipse application.

type filter text
- Apache Tomcat
- ATL Transformation
- Eclipse Application
  - com.gmv.sportsimulator.application.application
  - Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(External Launch)
- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Maven Build
- Operational QVT Interpreter
- OSGi Framework
- RAP Application
- RAP JUnit Test
- RWT Application
- XSL

Filter matched 21 of 21 items

Name: com.gmv.sportsimulator.application.application

Main | Arguments | Plug-ins | Configuration | Tracing | Environment | Common

Launch with: plug-ins selected below only | Default Start level: 4 | Default Auto-Start: false

type filter text

Plug-ins	Start Level	Auto-Start
☑ Workspace		
☑ com.gmv.course.sportsimulator.display (1.0.0.qualifier)	default	default
☑ com.gmv.sportsimulator.application (1.0.0.qualifier)	default	default
▪ Target Platform		
☐ ca.ecliptical.pde.ds (1.2.8.v20151122-2215)		
☐ ca.ecliptical.pde.ds.classpath (1.0.0.v20151122-2215)		
☑ ca.ecliptical.pde.ds.lib (1.0.0.v20141223-1920)	default	default
☑ com.gmv.course.exercise.utils (1.0.0.201710011519)	0	default
☑ com.gmv.sportsimulator.api (1.0.0.201911021656)	default	default
☐ com.gmv.sportsimulator.gamesloader (1.0.0.201911021656)		
☑ com.gmv.sportsimulator.tennis (1.0.0.201911021656)	default	default
☐ com.google.guava (15.0.0.v201403281430)		
☑ com.ibm.icu (52.1.0.v201404241930)	default	default
☐ com.jcraft.jsch (0.1.50.v201403120620)		
☐ com.jcraft.jsch (0.1.51.v201410302000)		
☐ com.lowagie.text (2.1.7.v201004222200)		
☐ com.sun.el (2.2.0.v201303151357)		

Select All | Deselect All | Add Working Set... | Add Required Plug-ins | Restore Defaults

☐ Only show selected
77 out of 2019 selected

☐ Include optional dependencies when computing required plug-ins
☐ Add new workspace plug-ins to this launch configuration automatically
☐ Validate plug-ins automatically prior to launching

Validate Plug-ins | Apply | Revert | Run | Close

Figure 4: Run Configuration for launching an application

24. Press button "Add Required Plug-ins". Notice that the number of plug-ins selected has increased
25. Press "Run" to start the application

5. EXERCISE 04: DETAILED VIEWS

5.1. GOALS:

- Understand the selection service mechanism and more generally, how parts collaborate.
- Gain expertise on View parts development.
- Gain expertise on SWT and SWT layout managers.

5.2. MATERIAL:

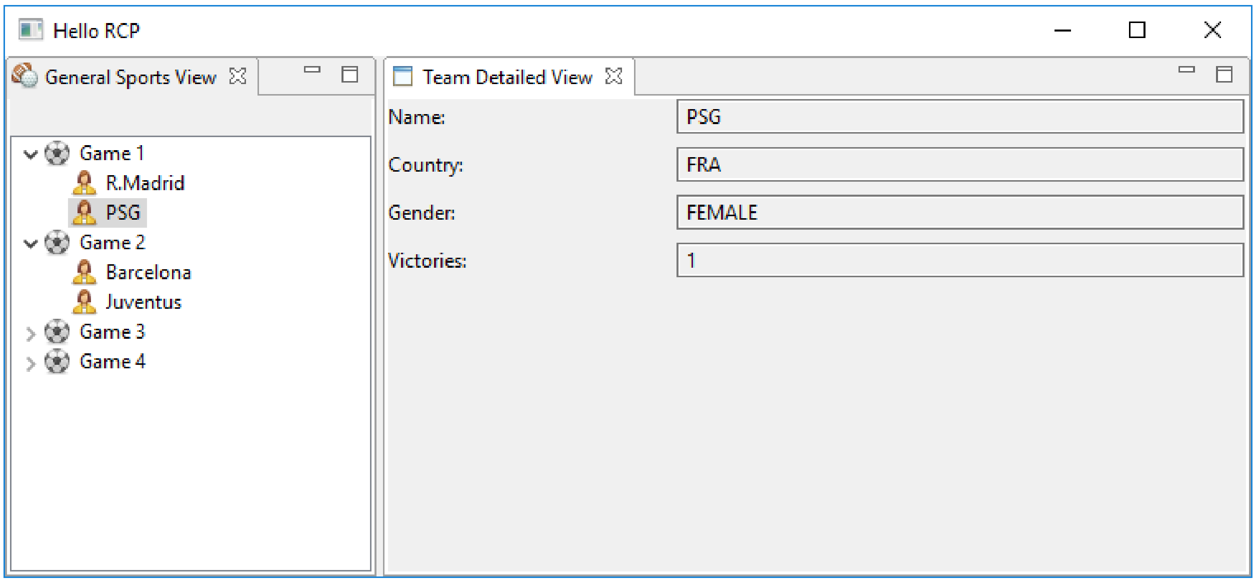
- Previous exercises.

5.3. ABSTRACT:

In this exercise, the student creates a plugin that contributes two views, one to display the Team properties and the other to display the Game and its status. The views shall subscribe to the selection service and, upon selection of the proper object (Team/Game), show its details.

5.4. STEPS:

1. Run eclipse, specifying the same workspace you created in previous exercises.
2. Select the display plugin created in Exercise 03 (com.gmv.course.sportsimulator.display)
3. Add a new View (see previous exercises). The view name could be “Team detailed view”. Create the following content (4 Labels, 4 Texts, and GridLayout):



Name:	PSG
Country:	FRA
Gender:	FEMALE
Victories:	1

Figure 5: Detailed view content

4. Modify the view you created in exercise 3 to be selection provider. Then, add the following line:

```
super.getSite().setSelectionProvider(viewer);
```

where `viewer` is the TreeViewer in charge of displaying the Games/Teams hierarchical structure

5. Register your just created view as selection listener and make it display the details of a Team, when one is selected. Use the Selection Service to register your view:

```
ISelectionService selectionService =
getSite().getWorkbenchWindow().getSelectionService();
```

Your view has to implement the listener `org.eclipse.ui.ISelectionListener`

Note, your view will be notified about all the selection changes. Then you have to check if the current selection is IStructuredSelection and if its first element is a Team object.

6. Create another view that performs the same operation, but instead shows the details of the selected Game.
7. (Optional) Register the Game details view to the sport service as a *ISportServiceListener*. Request the service to start all simulations. The listener will be notified about the game results. Fill the view with the updated information by creating a table with the games.

6. EXERCISE 05: EXTENSION POINT

6.1. GOALS:

- Understand the mechanism of extension points.

6.2. MATERIAL:

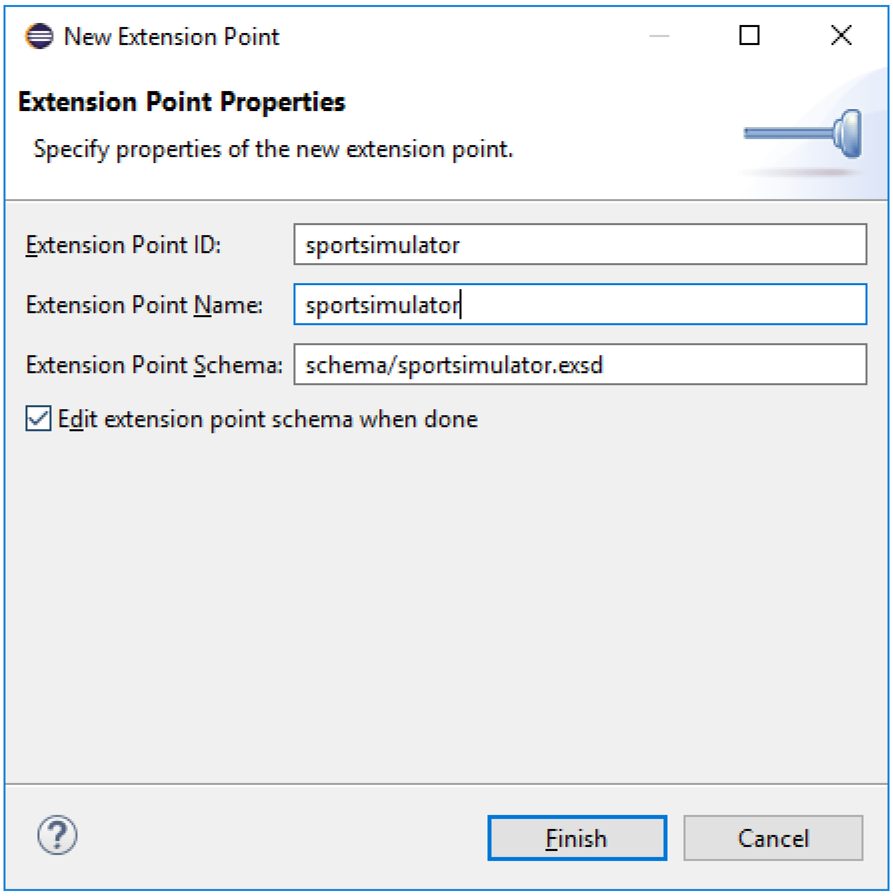
- Previous exercises.

6.3. ABSTRACT:

In this exercise, the student adds an extension point for defining a **service factory** that provides the access to one sport service. Then, the student creates a new plugin that extends the just created extension point, providing a sport service factory that allows the access to the service.

6.4. STEPS:

1. Run eclipse, specifying the same workspace you created in previous exercises.
2. Create a new plug-in com.gmv.sportsimulator.api.service. This plugin will be the intermediate that will allow the access to the specific service implementation. Other plugins will depend on it instead of depending on the specific sport service.
3. Create an interface **ISportServiceFactory** for the service factory that declares method *getSportService()*
4. Open manifest file of the new plugin.
5. Create a new extension point to provide the information for a sport service factory. Extensions to this extension point shall provide the sport type and the name of a class implementing the interface defined by *ISportServiceFactory*.
    a. Open the plugin.xml and select tab labeled "Extension Points".
    b. Click on the "Add…" button to add a new extension point definition.


New Extension Point

Extension Point Properties

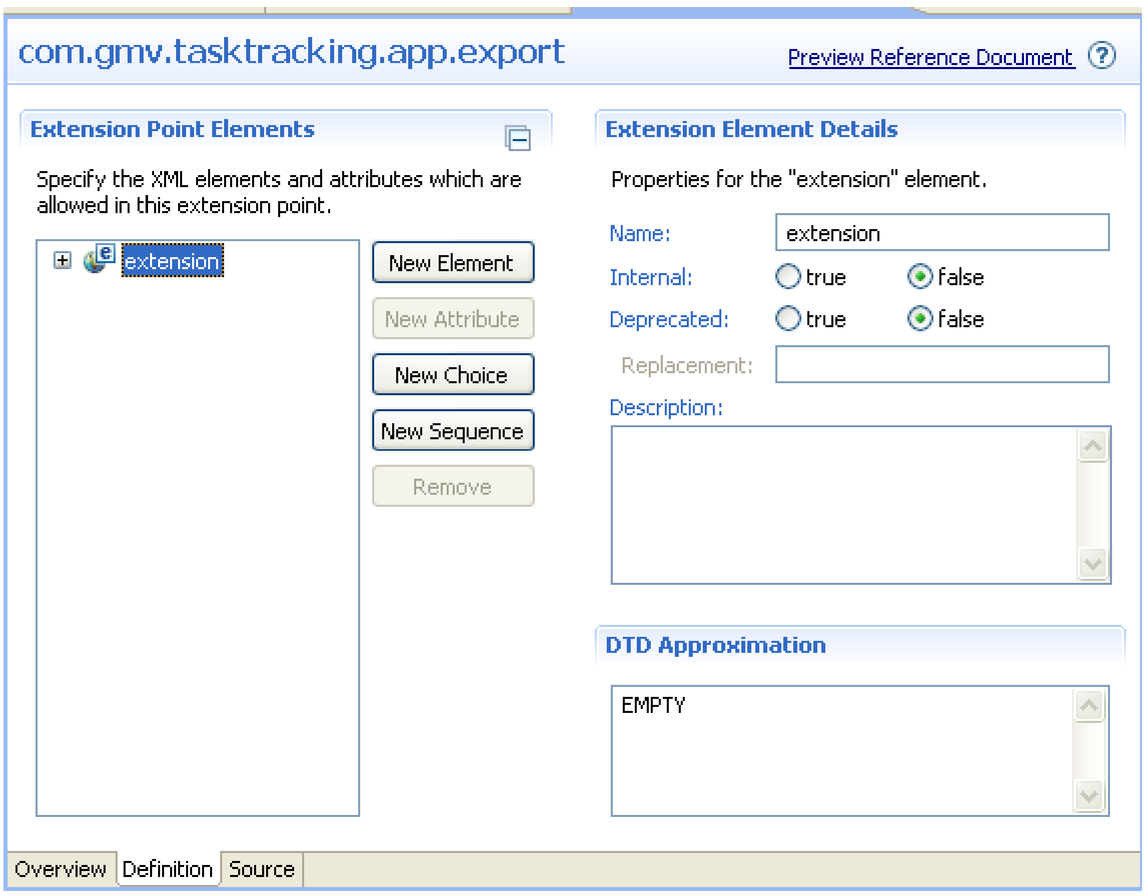
Specify properties of the new extension point.

Extension Point ID:	sportsimulator
Extension Point Name:	sportsimulator
Extension Point Schema:	schema/sportsimulator.exsd

☑ Edit extension point schema when done

Figure 6: Extension point configuration

c. The extension point is configured in a exsd file (it’s similar to a XSD file but you create it using an Eclipse editor):


com.gmv.tasktracking.app.export

Specify the XML elements and attributes which are allowed in this extension point.

Properties for the "extension" element.

Name: extension
Internal: true [✓ false]
Deprecated: true [✓ false]
Replacement:
Description:

EMPTY

Figure 7: EXSD Editor

d. Create an element that consists of three attributes, the first one of string type to give one id, the second of type string to give the sport type and the third one of type java to refer to the class implementing the factory.

e. Select the “extension” node and add a sequence of your element with 1 max. occurrence.

6. Create a new plugin for a soccer service (or the sport you choose) and name it *com.gmv.sportsimulator.soccer*
7. Create SoccerService class extending BaseSportService (from com.gmv.sportsimulator.api).
   a. Implement method getSportTypes by returning one array with String “Soccer”.
   b. Implement *createGameInstance* creating and returning a game using the parameters.
   c. Implement *createAndStartSimulation* by creating a thread implementing ISimulationThread. It will start by invoking *Game::resetGame()* method. Then, it perform changes in the game results, it will call *notifyResultChanged(Game)* from base class, and eventually it will invoke *Game::finalise()* and *notifyGameEnded(Game)* from base class.
8. Contribute to the sportsimulator extension point by registering a service factory class that creates an instance to the SoccerService.
9. Implement the factory class. Implementation should also populate the sport simulator data when the service is created. You may move here the set of sports and teams that you created in exercise 3.
10. In *com.gmv.course.sportsimulator.display* create a class that will read the extension contribution, instantiate the factory and return always the same instance of the soccer service. You can get the service factory like this:

```
IConfigurationElement[]  configElements  = Platform.getExtensionRegistry()
        .getConfigurationElementsFor("com.gmv.sportsimulator.servicelocator.sportsimulator");
serviceFactory = (ISportServiceFactory)  configElements[0].createExecutableExtension("class");
```

7. EXERCISE 06: CREATE COMMAND/HANDLER

7.1. GOALS:

- Understand what commands and handlers are.

7.2. MATERIAL:

- Previous exercises.

7.3. ABSTRACT:

In this exercise, the student creates a new command that will start the simulation of games. All the generated items from previous exercises are necessary. The application shall implement a perspective that shows both the view created in exercise 3 and the new detailed views.

7.4. STEPS:

1. Run eclipse, specifying the same workspace you created in previous exercises.
2. In plugin com.gmv.course.sportsimulator.display:
   a. Contribute to extension point *org.eclipse.ui.commands* and define a new command.
   b. Contribute to extension point *org.eclipse.ui.menus*:
      i. New -> menuContribution
      ii. locationURI: popup:com.gmv.sportsimulator.generalviewmenu
      iii. Select the menuContribution -> New -> Command
      iv. Reference to the command ID of the command defined previously (you can use the Browse… button)
      v. Provide a label and/or an icon for the menu item
   c. Contribute to extension point *org.eclipse.ui.handlers* and define a new command handler that references to the command and create a handler class
   d. Make your new handler class extend *org.eclipse.core.commands.AbstractHandler* and implement method *execute*. The execution should use the service reference and the selection service to invoke method *ISportSimulator::simulateGame(Game,SimulationSpeed)*
   e. Add a menu to the sports view from exercise 3:

   In *createPartControl* add the following code snippet and populate using the menu id as defined in step ***b***:

```
MenuManager mm = new MenuManager("com.gmv.sportsimulator.generalviewmenu");
((IMenuService) getSite().getService(IMenuService.class)).populateContributionManager(mm,
 "popup:com.gmv.sportsimulator.generalviewmenu");
Menu menu = mm.createContextMenu(viewer.getTree());
viewer.getTree().setMenu(menu);
```

8. EXERCISE 07 (OPTIONAL): EXPORT RCP APPLICATION

8.1. GOALS:

- Understand what an eclipse RCP application and a product configuration are.

8.2. MATERIAL:

- Previous exercises.

8.3. ABSTRACT:

In this exercise, the student exports the application created in the previous exercises.

8.4. STEPS:

1. Create a product configuration:
   a. File -> New... -> Product Configuration.
   b. Fill product name, product id and application id in Overview tab.
   c. Add the plugins your application is composed of in Configuration tab (hint: add the main plugin and then press "Add required plugins").
   d. Add all the platform required plugins:
      - Bundle org.eclipse.equinox.ds
        - Or *org.apache.felix.scr* instead (depending on the platform baseline)
      - Bundle org.eclipse.equinox.event
      - Bundle org.eclipse.equinox.simpleconfigurator
2. Press Export to export the product