Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
Hibatűrő Rendszerek Kutatócsoport
Modellalapú tervezés és kódgenerálás szakkör

Eclipse alapú fejlesztés
– Gyakorlat –

Darvas Dániel,
Horányi Gergő

2012. március

1 Bevezető

Ebben a dokumentumban lépésről lépésre leírjuk a szakköri alkalmon bemutatott gyakorlati példát. Reméljük, hogy e leírás segíteni fogja a házi feladat elkészítését is.

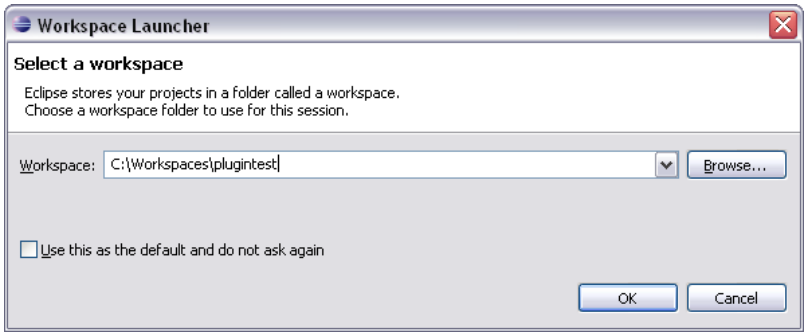
1.1 A feladat

A feladat a Sheldon nevű Eclipse-alapú alkalmazáshoz egy olyan plugin készítése, amely egy Eclipse Forms felületen képes megjeleníteni az egyes vágányszakaszok aktuális foglaltságát az alábbihoz hasonló módon:

2 A feladat megoldása lépésről lépésre

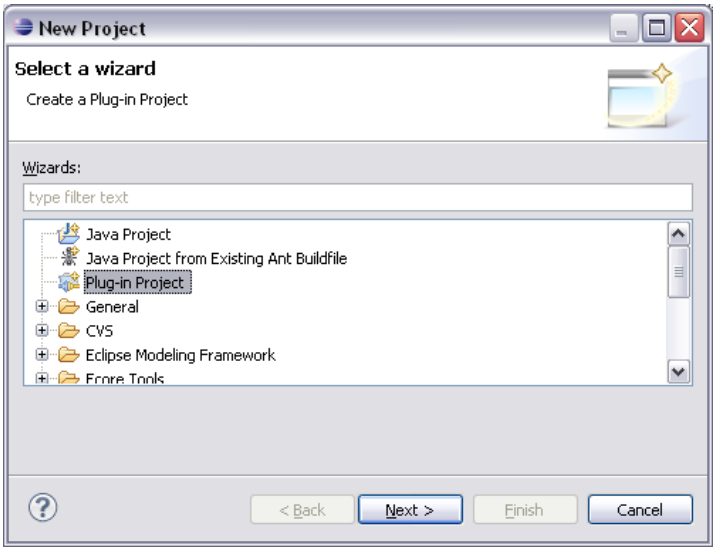
2.1 Új plugin létrehozása

1. Indítsuk el az Eclipse alkalmazást! (A kiadott virtuális gépen a helye: *C:\eclipse\eclipse.exe*.) Válasszunk egy új workspace-t, pl. *c:\workspaces\plugintest*.

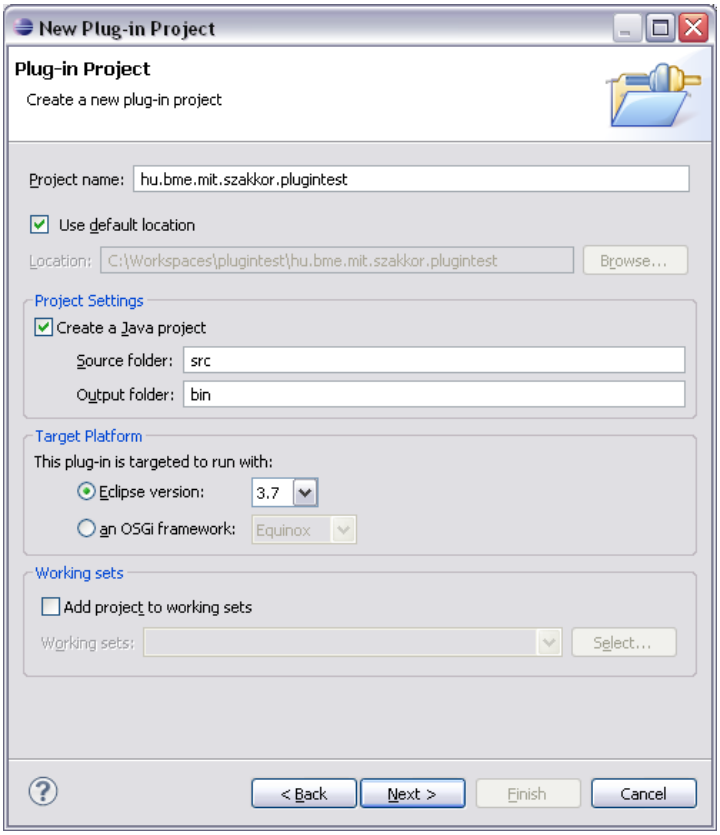


2. Ha megjelenik, zárjuk be a *Welcome* ablakot!
3. Hozzunk létre egy új projektet!
    a. Válasszuk a *File > New > Project…* menüpontot.

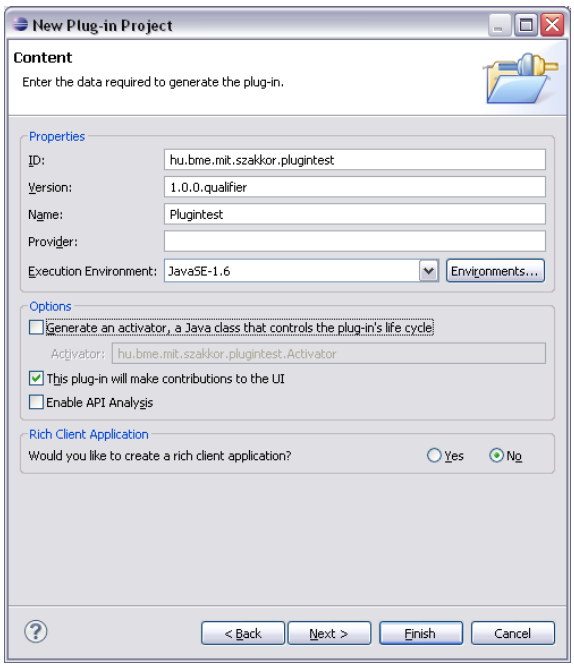
b. Itt válasszuk ki a *Plug-in Project*-et, hiszen egy új plug-int szeretnénk készíteni a Sheldon programhoz. Kattintsunk a *Next* gombra.



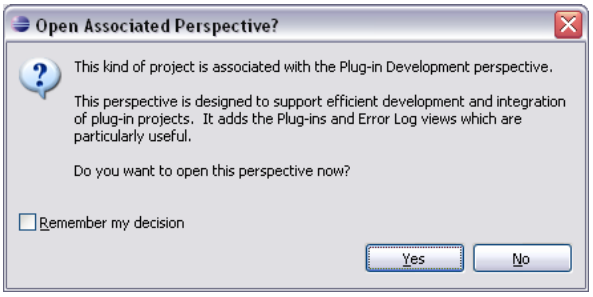
c. Írjuk be a projekt nevét: *hu.bme.mit.szakkor.plugintest*. A többi beállítást nem kell megváltoztatni. Kattintsunk a *Next* gombra.



d. A következő oldalon megadhatjuk a plug-inünk tulajdonságait, pl. nevét. Emellett egyéb opciókat is beállíthatunk. Vegyük ki a pipát a „*Generate an activator…*” opció elől, erre most nem lesz szükségünk. Kattintsunk a *Finish* gombra.

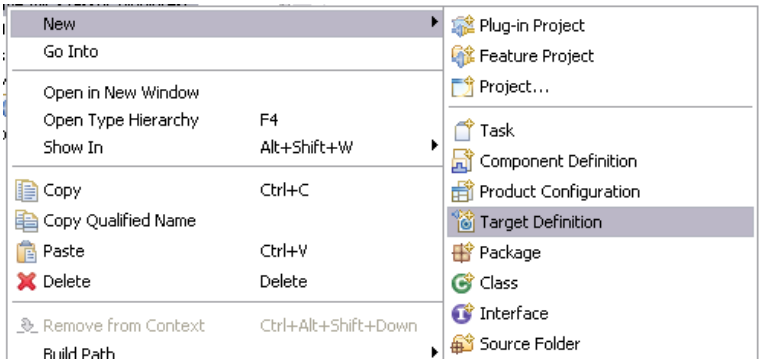


e. Ha az Eclipse rákérdez, hogy kívánjuk-e használni a *Plug-in Development* perspektívát, kattintsunk nyugodtan *Yes*-re.

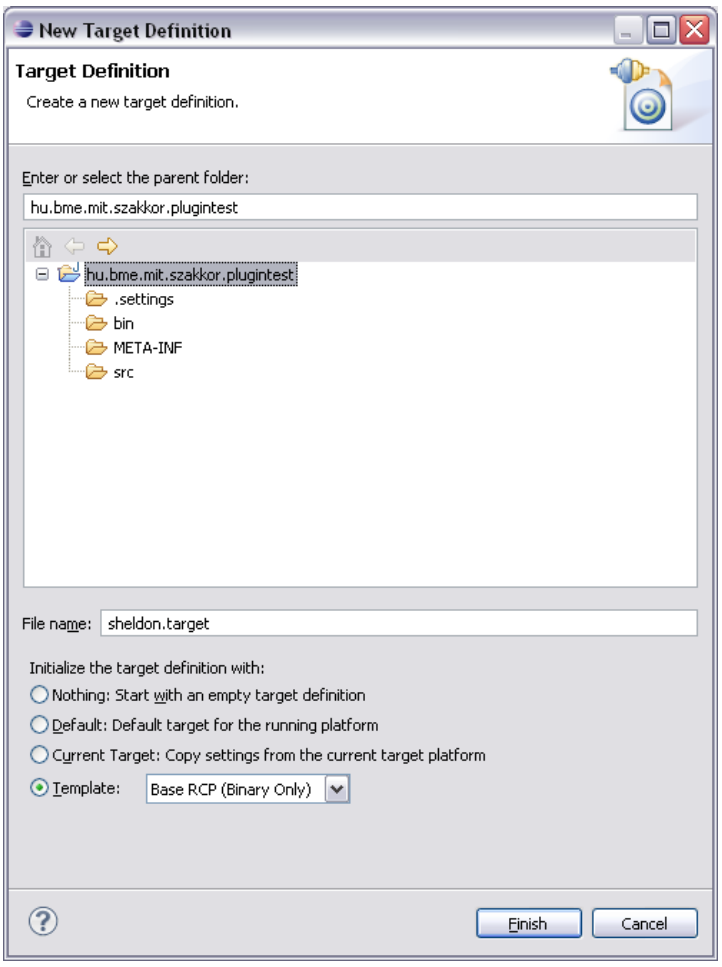


f. Máris elkészült a plug-inünk projektje. Azonban egyelőre még semmilyen kódot nem tartalmaz, mindössze a plug-in tulajdonságait és viselkedését leíró *manifest.mf* és a plug-in fordítását leíró *build.properties* fájlt.

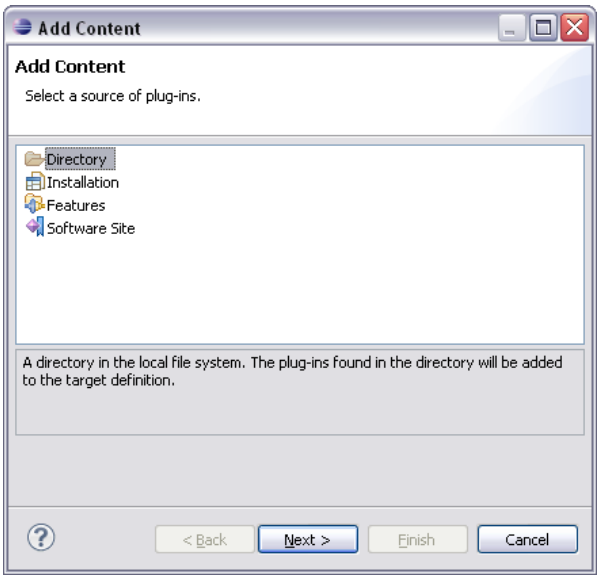
4. Mi most nem egy általános Eclipse plug-int szeretnénk fejleszteni, hanem a Sheldon programot szeretnénk kibővíteni. Ehhez egy új *Target Definition* felvételére lesz szükség.
   a. A *Package Explorer* nézetben kattintsunk jobb gombbal a projekt nevére. Válasszuk a *New > Target Definition* lehetőséget.



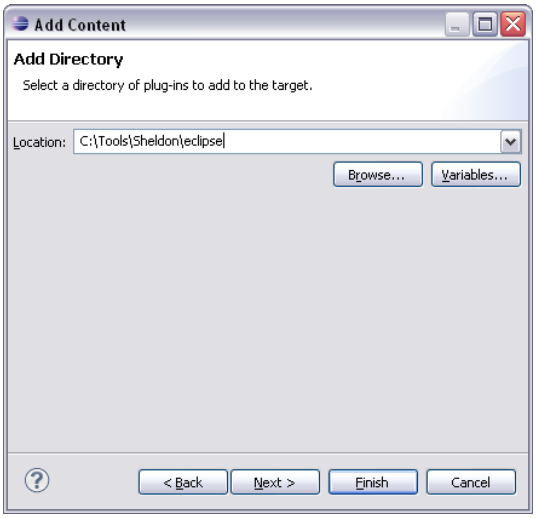
b. Adjunk egy nevet a Target Definitionnek: *sheldon.target*. Válasszuk ki, hogy egy *Template* alapján inicializálja a targetet, méghozzá a *Base RCP (Binary Only)* template-et használva.



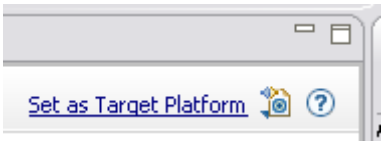
c. Adjuk hozzá a Target Definitionhöz a Sheldon mappáját! Ehhez a *Locations* rész *Add...* gombjára kattintsunk.
d. Válasszuk ki a *Directory* opciót, majd kattintsunk a *Next*-re.



e. Adjuk meg a Sheldon program mappáját, ami a virtuális gépen a következő: *C:\Tools\Sheldon\eclipse*. Kattintsunk a *Finish* gombra.



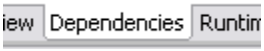
f. Az ablak tetején kattintsunk a *Set as Target platform* linkre.



2.2 Kontribúció egy nem eclipse-es kiterjesztési ponthoz

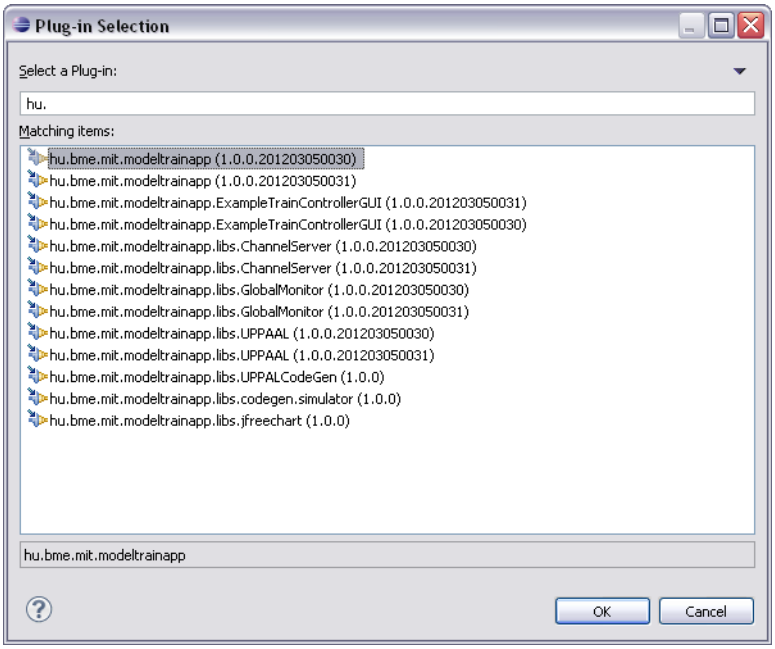
Ebben a fejezetben egy, a Sheldonban definiált kiterjesztési ponthoz készítünk egy egyszerű kiterjesztést. Ennek segítségével a plug-inünkben a későbbiekben felhasználhatjuk a Sheldonban lévő vasúti objektumokat.

1. Keressük ki a *Package Explorerben* a *META-INF\mainfest.mf* fájlt és kattintsunk rá duplán.
2. A szerkesztő alján válasszuk ki az *Dependencies* fület.



3. Itt kell megadnunk a plug-inünk működéséhez szükséges további plug-ineket. Mivel a kiterjesztési pont definíciója a *hu.bme.mit.modeltrainapp* plug-inben szerepel, ezért ezt mindenképp hozzá kell adnunk.
   a. Kattintsunk a *Required Plug-ins* szekcióban az *Add…* gombra.
   b. Válasszuk ki a *hu.bme.mit.modeltrainapp* plug-int. Kattintsunk az *OK* gombra.
      i. Ehhez a plug-in nevének első pár betűjét be kell gépelnünk.

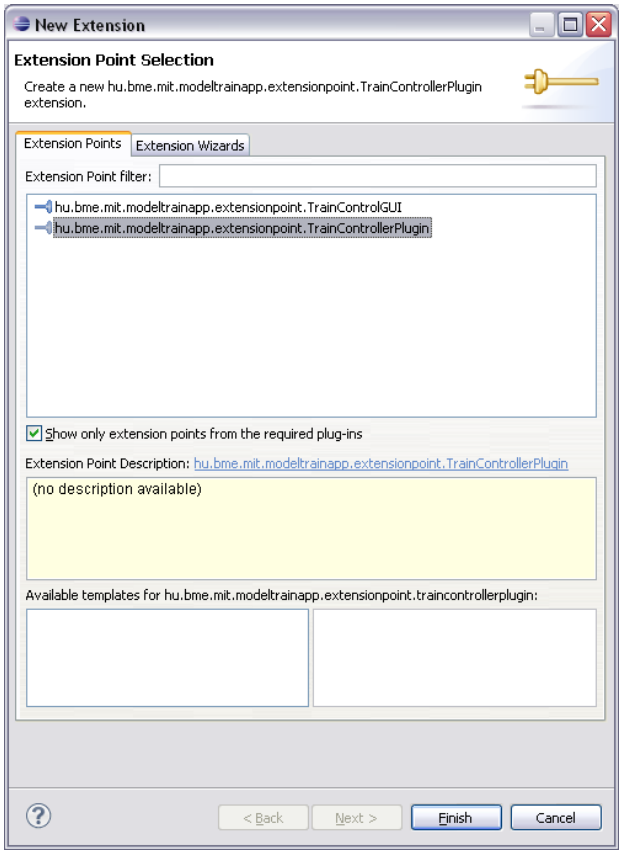
ii. Ha több, azonos nevű, eltérő verziójú plug-int látunk, közülük válasszuk a legfrissebbet (pl. r1003).



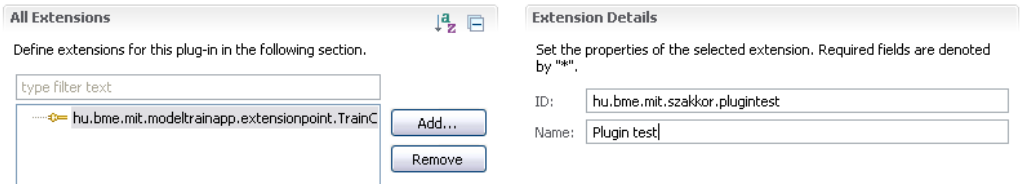
4. A szerkesztő alján válasszuk ki az *Extensions* fület.

5. Itt tudjuk majd megadni, hogy melyik kiterjesztési pontokhoz szeretnénk kiterjesztést készíteni. Először a *hu.bme.mit.modeltrainapp.extensionpoint.TrainControllerPlugin* kiterjesztési ponthoz készítünk kiterjesztést.
    a. Kattintsunk az *All Extensions* szekcióban az *Add…* gombra.

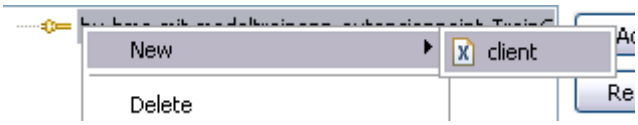
b. Válasszuk ki a *hu.bme.mit.modeltrainapp.extensionpoint.TrainControllerPlugin* kiterjesztési pontot, majd kattintsunk a *Finish* gombra.



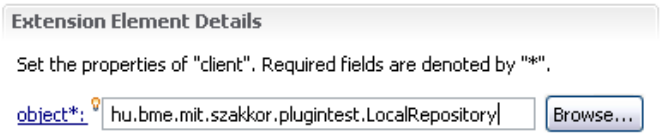
c. Adjuk meg az újonnan felvett kiterjesztés adatait:
ID: *hu.bme.mit.szakkor.plugintest*
Name: *Plugin test*



d. Emellett ez a konkrét kiterjesztési pont elvár még egy „klienst” is, egy osztályt, amely megvalósít egy megadott interfészt. Ezt a kiterjesztési pont nevén jobb gombbal kattintva megjelenő menüből a *New > client* kiválasztásával vehetjük fel.



e. Adjuk meg a kliensosztályt. Ehhez az *object* mezőbe kell beírnunk a megfelelő osztály teljes nevét. Ilyet még nem hoztunk létre, de erre képes az Eclipse is automatikusan. Adjuk meg a következő nevet az *object* mezőben: *hu.bme.mit.szakkor.plugintest.LocalRepository*.



f. Kattintsunk az *object** linkre, majd a *Finish* gombra.
g. Ez után megnyílik a frissen elkészült *LocalRepository.java* fájl. Ez fogja tartalmazni a vasúti objektumaink tárolóit, melyet a Sheldon program majd az *initialize* függvény meghívásával inicializál. Készítsük el az osztály alábbi, nagyon egyszerű implementációját:

```
package hu.bme.mit.szakkor.plugintest;

import hu.bme.mit.modeltrainapp.extensions.TrainControllerPlugin;
import hu.bme.mit.modeltrainapp.model.TrainObjectRepository;

public class LocalRepository implements TrainControllerPlugin {
	private static TrainObjectRepository repo = null;

	public LocalRepository() {
	}

	@Override
	public void initialize(TrainObjectRepository _repo) {
		repo = _repo;
	}

	public static TrainObjectRepository getRepo() {
		return repo;
	}
}
```

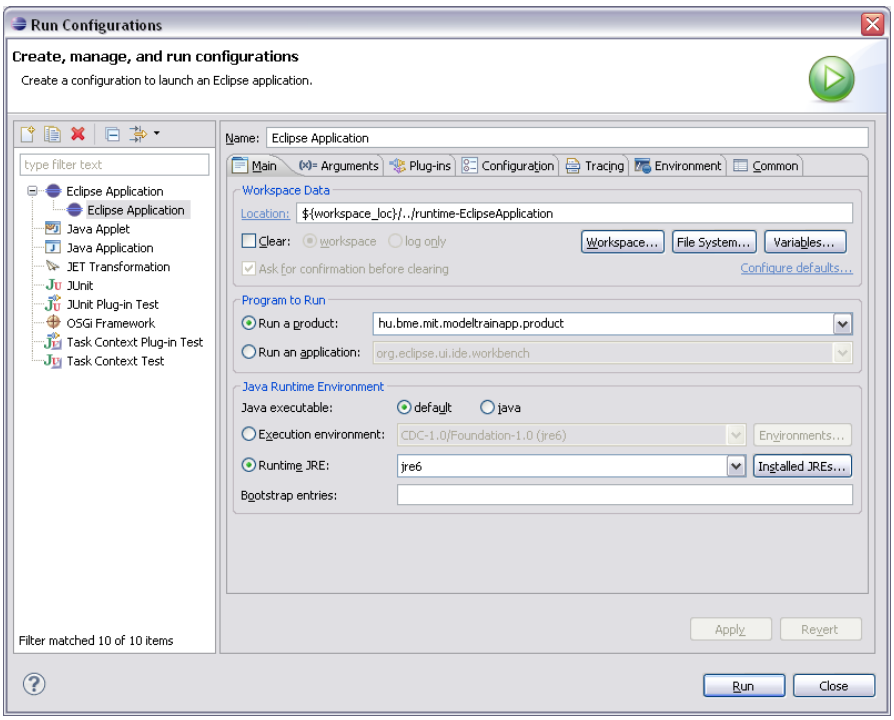
h. Már alig várjuk, hogy teszteljük első Eclipse plug-inünket. Ezért egészítsük ki az *initialize* függvényt a következő sorral:

```
System.out.println("Loaded signals: " + repo.getSignals().size());
```

Ennek segítségével az inicializálásról tudomást szerzünk, ha a konzolt figyeljük.
i. Mentsünk minden fájlt! (Ctrl+Shift+S vagy a menüsoron: )

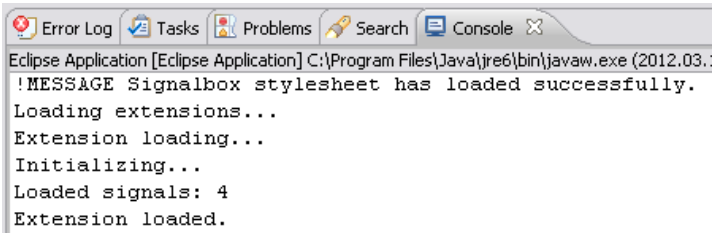
6. Még egy beállítást meg kell tennünk, hogy futtatni tudjuk a plug-inünket: be kell állítanunk, hogy mit kell futtatni.
   a. Kattintsunk a *Run > Run configurations…* menüpontra!

b. Válasszuk a fastruktúrában az *Eclipse Applicationt*, majd a *Main* fülön a *Program to run* szekcióban jelöljük be a *Run a product* lehetőséget, a listából pedig válasszuk ki a *hu.bme.mit.modeltrainapp.product* elemet.



c. Kattintsunk a *Close* gombra.

7. Most már futtathatjuk az alkalmazásunkat. Ehhez kattintsunk a *Run* > *Run* menüpontra vagy a menüsoron a gombra.
   a. Ha az Eclipse rákérdez, milyen konfigurációt futtassunk, válasszuk az *Eclipse Application* lehetőséget (hiszen ezt állítottuk be az előbb a céljainknak megfelelően).
8. Ha mindent jól csináltunk, elindul a Sheldon alkalmazás, illetve a konzolablakban láthatjuk a diagnosztikai kimenetünket a plug-inek betöltése során.



9. Zárjuk be a futó Sheldon alkalmazást.

2.3 Kontribúció a grafikus felhasználói felülethez

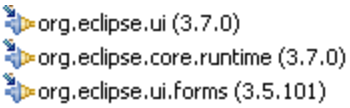
Most megpróbálunk egy új nézetet hozzáadni a Sheldon program felületéhez.

Valójában egy nézet hozzáadásához több kiterjesztési ponthoz is kell majd implementációt készítenünk: egyet a nézet definiálására, egyet a nézet megjelenítésére, egyet pedig a nézet megjelenítésére szolgáló eszköztár-elemhez.

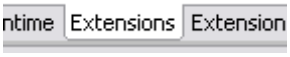
1. Vegyük fel a nézet megjelenítéséhez szükséges függőségeket!
   a. Keressük ki a *Package Explorerben* a *plugin.xml* fájlt és kattintsunk rá duplán.

b. A szerkesztő alján válasszuk ki az *Dependencies* fület.

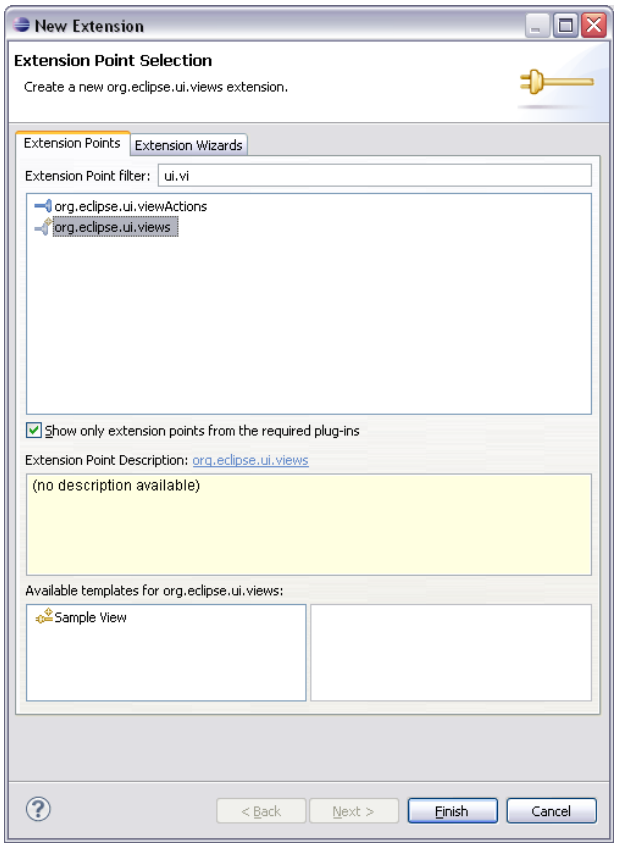
c. Kattintsunk a *Required Plug-ins* szekcióban az *Add…* gombra.
d. Vegyük fel egymás után az alábbi plug-ineket:
   i. *org.eclipse.ui*
   ii. *org.eclipse.core.runtime*
   iii. *org.eclipse.ui.forms*



2. Készítsünk egy új View kiterjesztést!
   a. A szerkesztő alján válasszuk ki az *Extensions* fület.



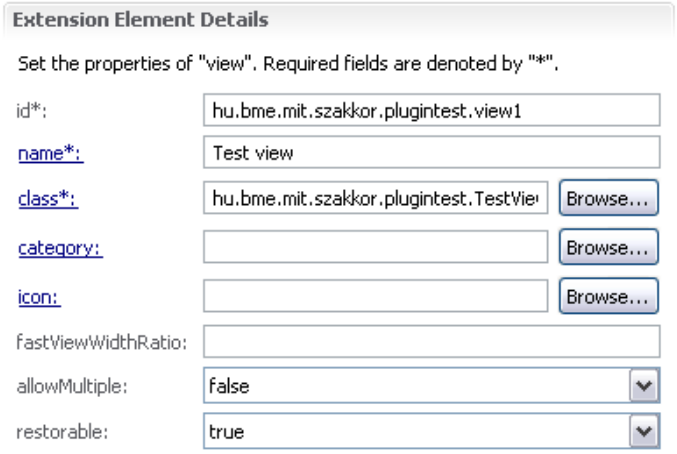
   b. Kattintsunk az *All Extensions* szekcióban az *Add…* gombra.
   c. Válasszuk ki az *org.eclipse.ui.views* kiterjesztési pontot és kattintsunk a *Finish* gombra.



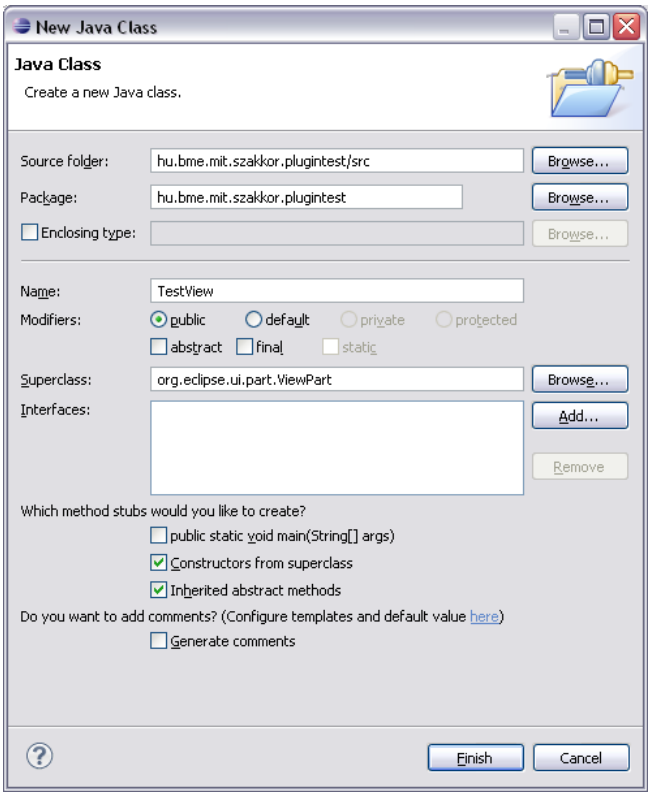
   d. Itt nem kell a nézethez semmilyen adatot megadnunk.

e. Kattintsunk jobb gombbal az *org.eclipse.ui.views* elemen és válasszuk a *New > category* menüpontot.

f. Adjuk meg az új nézet adatait:
   i. id: *hu.bme.mit.szakkor.plugintest.view1*
   ii. name: *Test view*
   iii. class: *hu.bme.mit.szakkor.plugintest.TestView*
   iv. allowMultiple: *false*

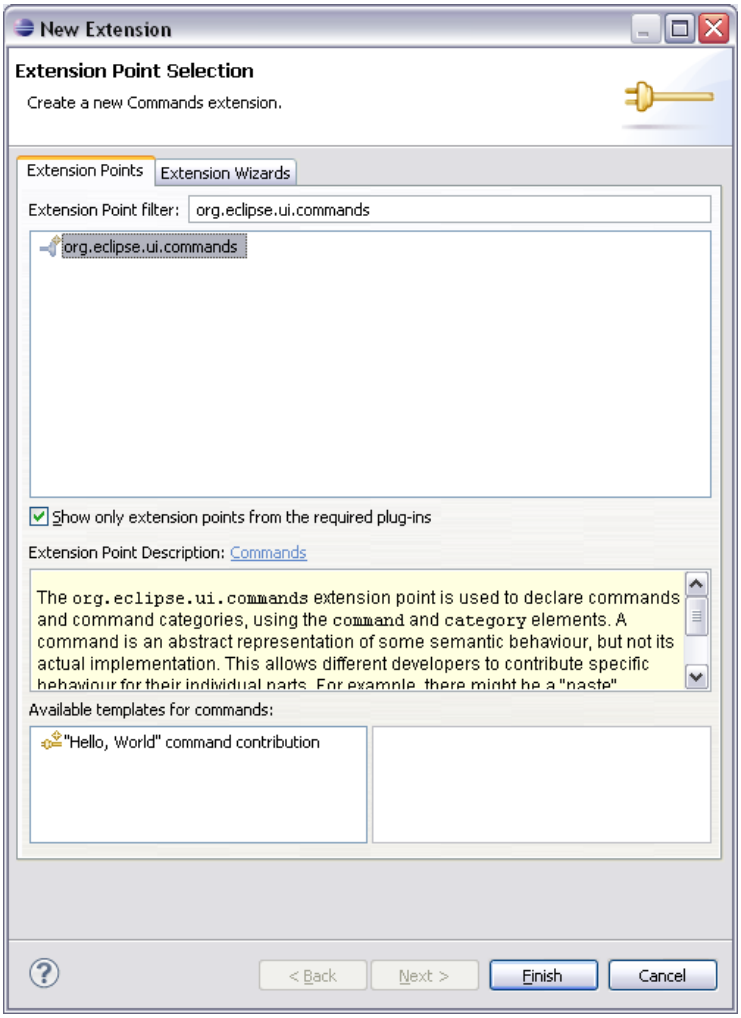


g. A *class* mezőben most is egy még nem létező osztály nevét adtuk meg. Kattintsunk a *class** hivatkozásra, a megjelenő ablakban pedig a *Finish* gombra.

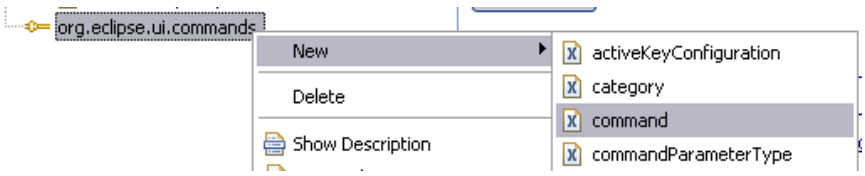


h. A *TestView* osztály implementálását halasszuk későbbre.

3. Készítsünk egy új Command kiterjesztést, amely képes megjeleníteni a *TestView* nézetet!
   a. Térjünk vissza a *plugin.xml* fájlhoz és válasszuk ki az *Extensions* fület.
   b. A korábbiakhoz hasonlóan vegyünk fel egy *org.eclipse.ui.commands* kiterjesztést.

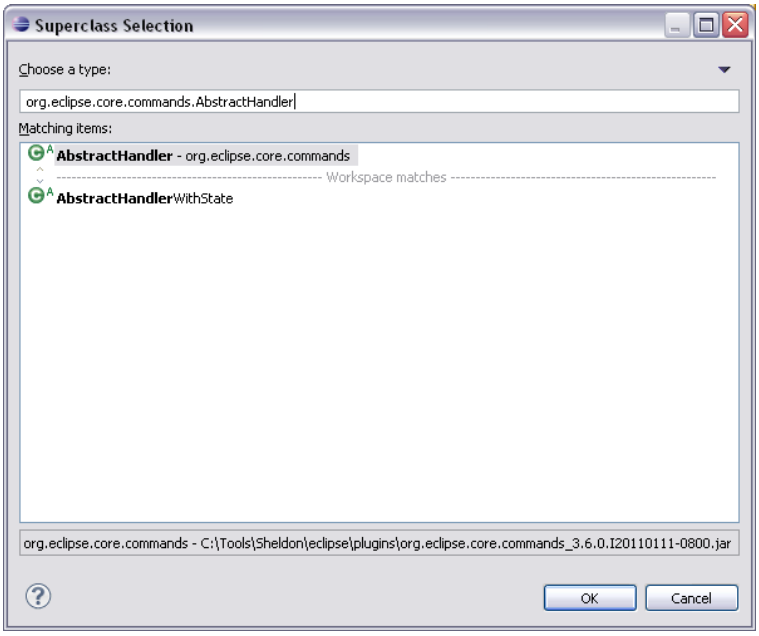


   c. Vegyünk fel egy új parancsot az *org.eclipse.ui.commands* elemre jobb gombbal kattintva, a *New > command* menüpontot választva.

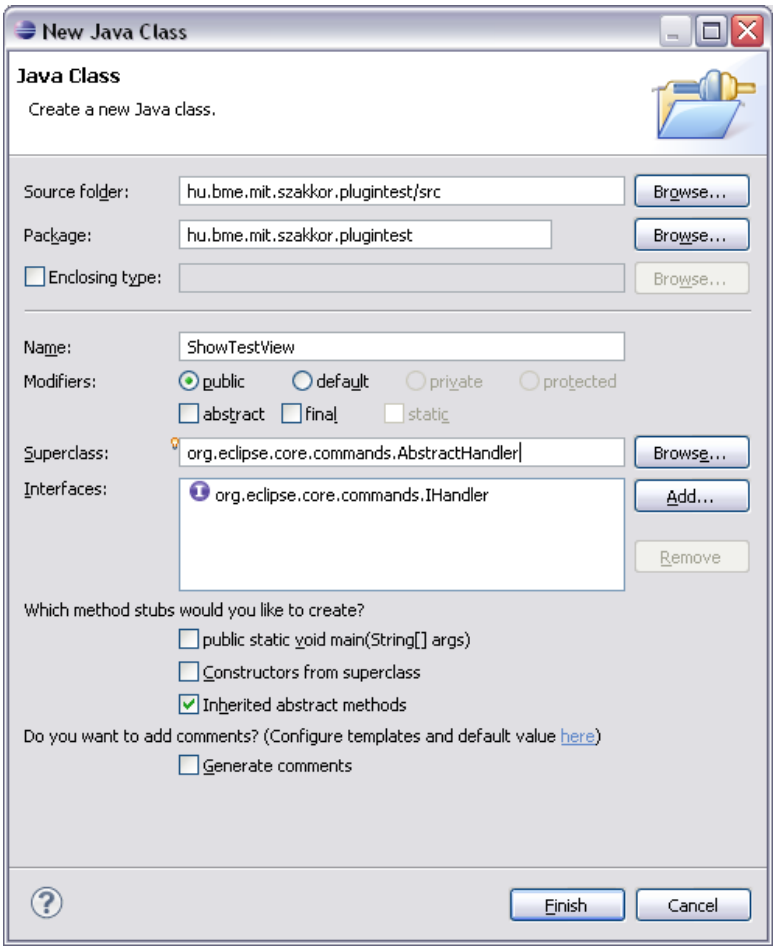


   d. Adjuk meg a *command* adatait:
      i. id: *hu.bme.mit.szakkor.plugintest.ShowTestView*
      ii. name: *Show Test View*
      iii. defaultHandler: *hu.bme.mit.szakkor.plugintest.ShowTestView*
   e. Létre kell hoznunk a *defaultHandler*-nél megadott osztályt. Kattintsunk a *defaultHandler* linkre! Most nem hagyunk minden beállítást alapértelmezetten.
      i. Kattintsunk a *Superclass* mező melletti *Browse* gombra.

ii. Válasszuk ki az *org.eclipse.core.commands.AbstractHandler* osztályt és kattintsunk az *OK*-ra.



iii. Kattintsunk a *Finish* gombra.

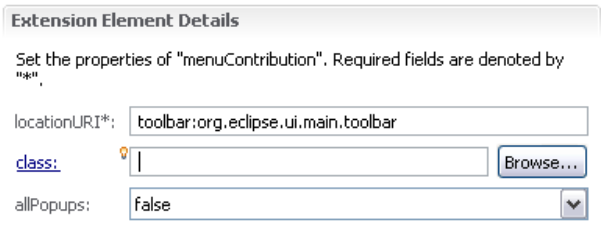


4. A létrejövő *ShowTestView* osztály *execute* metódusában kell implementálnunk a nézet megjelenítését. Ehhez a fájlban az alábbi kód szerepeljen:

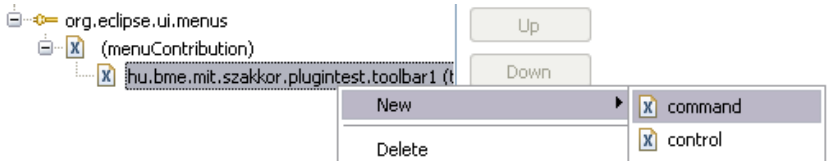
```
package hu.bme.mit.szakkor.plugintest;

import hu.bme.mit.modeltrainapp.rcp.Logger;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;

public class ShowTestView extends AbstractHandler implements IHandler {

	@Override
	public Object execute(ExecutionEvent event) throws ExecutionException {
		try {
			PlatformUI
					.getWorkbench()
					.getActiveWorkbenchWindow()
					.getActivePage()
					.showView("hu.bme.mit.szakkor.plugintest.view1", null,
							IWorkbenchPage.VIEW_ACTIVATE);
		} catch (PartInitException e) {
			Logger.log(IStatus.ERROR,
					"Error occured during the initialization of view.",
					e);
			Logger.showPopup(IStatus.ERROR, "Error",
					"Error occured during the initialization of view.");
		}

		return null;
	}
}
```

   a. A "hu.bme.mit.szakkor.plugintest.view1" helyén mindig az aktuális nézet pontos ID-jét kell megadni.
   b. A Logger osztály a Sheldon naplózó és hibajelző osztálya, hiba esetén ez fog értesíteni minket a grafikus felületen.
   c. Mentsük is a munkánkat!
5. Készítsünk egy új menükiterjesztést, amely képes megjeleníteni a nézetünk megjelenítésére szolgáló gombot!
   a. Térjünk vissza a *plugin.xml* fájlhoz és válasszuk ki az *Extensions* fület.
   b. A korábbiakhoz hasonlóan vegyünk fel egy *org.eclipse.ui.menus* kiterjesztést.
   c. Vegyünk fel ez alá egy *menuContribution* elemet a korábbiakhoz hasonlóan.

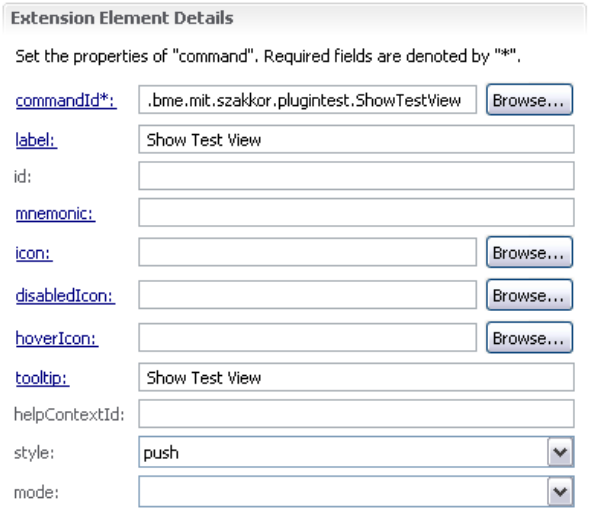
d. A *menuContribution* elem *locationURI* mezőjébe írjuk be a következőt: *toolbar:org.eclipse.ui.main.toolbar*. Ennek az a jelentése, hogy a definiált kontribúció az ablak fő eszköztárához járul hozzá.



e. A *menuContribution* gyerekeként vegyünk fel egy új *toolbar* elemet.
f. A *toolbar* elem gyerekeként vegyünk fel egy új *command* elemet.

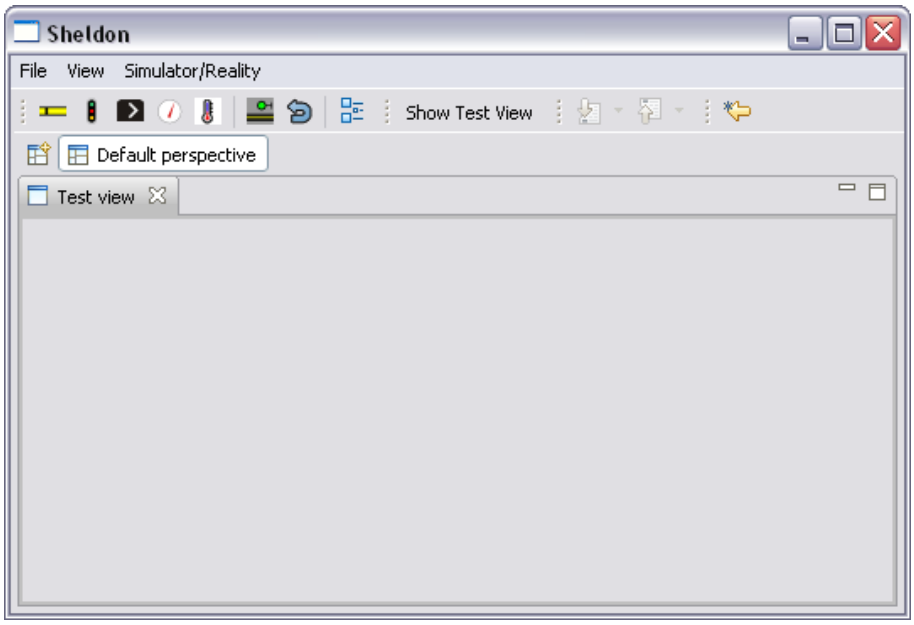


g. A *command* elem beállításai a következők legyenek:
   i. commandId: *hu.bme.mit.szakkor.plugintest.ShowTestView*
   ii. label: *Show Test View*
   iii. tooltip: *Show Test View*



h. Mentsünk minden módosítást. ( )

6. Végre eljutottunk odáig, hogy kipróbálhatjuk, mit alkottunk. Ha futtatjuk a programot ( ) és mindent jól csináltunk, megjelenik az eszköztáron egy *Show Test View* gomb, amire ha rákattintunk, megnyílik az új (és egyelőre töküres) nézetünk.



2.4 Nézet elkészítése Eclipse Forms API-val

Ebben a fejezetben egy egyszerű nézetet készítünk az Eclipse Forms API segítségével.

1. Keressük ki a *TestView.java* fájlt a *Package Explorerben* és kattintsunk rá duplán! Végre megtöltjük tartalommal.[1]
2. Vegyünk fel két változót az osztályba:

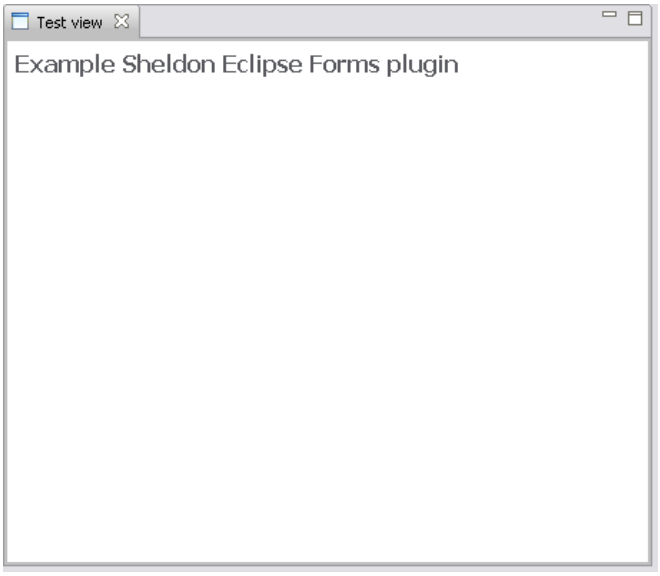
```
private FormToolkit toolkit;
private Form form;
```

   a. A *form* változó fogja az űrlapunkat reprezentálni, míg a *toolkit* változó az elemek példányosítását fogja segíteni.
3. A *createPartControl* metódusban inicializáljuk az űrlapot az alábbiak szerint:

```
// Toolkit példányosítása
toolkit = new FormToolkit(parent.getDisplay());

// új Form példány készítése
form = toolkit.createForm(parent);

// az űrlap elrendezésének beállítása
form.getBody().setLayout(new ColumnLayout());

// az űrlap címének beállítása
form.setText("Example Sheldon Eclipse Forms plugin");
```

[1] Számos osztály neve nem egyértelmű és mind a java.awt, mind az org.eclipse.swt csomagban megtalálható. A java.awt-ben lévő osztályokra NEM lesz szükségünk most, sehol se válasszuk ezt!

4. Ha most kipróbálnánk a plug-inünket, ezt látnánk:

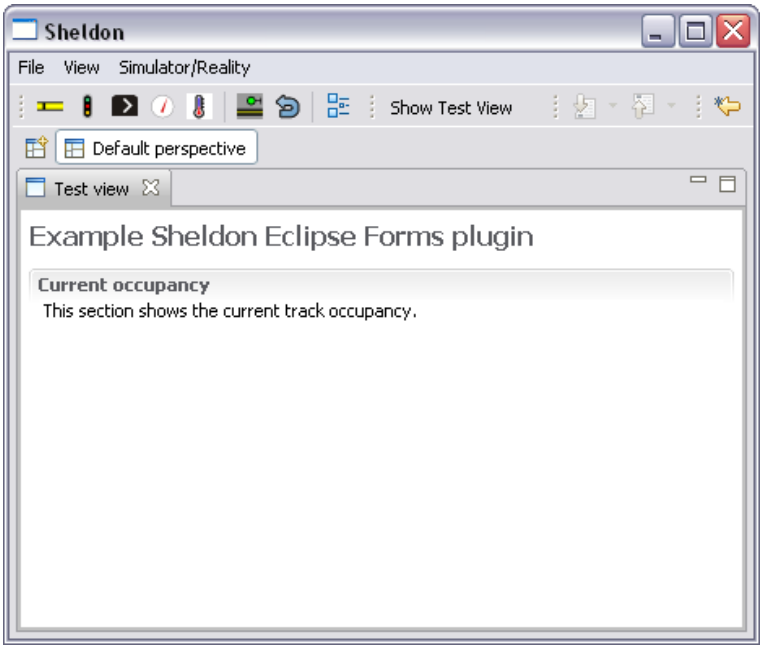


5. Hozzunk létre egy szekciót az űrlapon!

```
// szekció létrehozása (leírássa, címsorral, kibontva)
Section occup = toolkit.createSection(form.getBody(),
      Section.DESCRIPTION | Section.TITLE_BAR | Section.EXPANDED);

// szekció címének beállítása
occup.setText("Current occupancy");
// szekció leírásának beállítása
occup.setDescription("This section shows the current track occupancy.");

// szekció belsejéhez elemeket tároló objektum létrehozása
Composite occupBody = toolkit.createComposite(occup);

// szekció belső elrendezésének beállítása[2]
ColumnLayout cl = new ColumnLayout();
cl.maxNumColumns = 1;
occupBody.setLayout(cl);

// szekciót kezelő belső objektum beállítása
occup.setClient(occupBody);
```

[2] A ColumnLayout használata javítja a gyakorlaton előkerült problémát, miszerint a Label-ek nem idomulnak a bennük szereplő szöveg szélességének megváltozásához.

6. Ha most kipróbálnánk a plug-inünket, ezt látnánk:



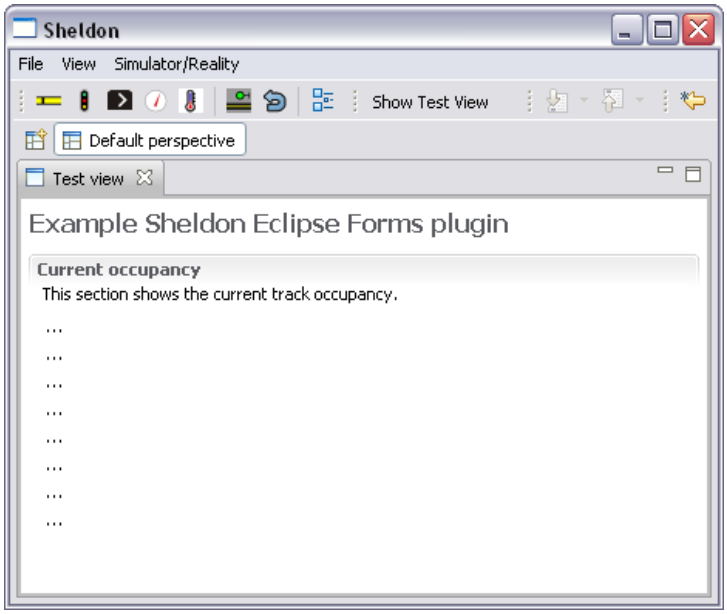
7. Vegyünk fel minden vágányszakaszhoz egy-egy címkét (legalábbis azokhoz, amelyekre a *getGraphicalRepresentation()* nem null értéket ad vissza – ezek a GUI-n tényleg megjelenítendő vágányszakaszok).
    a. Ehhez először vegyünk fel egy az osztályba egy hash-táblát, ami a vágányszakaszok azonosítóit és a címkéket összerendeli.

```
private Map<Integer, Label> trackLabels = new HashMap<Integer, Label>();
```

    b. Kérdezzük le a LocalRepository-ból az összes vágányszakaszt (TrackElement objektumot), majd mindegyikhez készítsünk 1-1 címkét.

```
for (TrackElement te : LocalRepository.getRepo().getTrackElements()) {
      if (te.getGraphicalRepresentation() != null) {
            // a Label objektum létrehozása (dummy szöveggel)
            trackLabels.put(te.getId(),
                        toolkit.createLabel(occupBody, "..."));
      }
}
```

8. Ha most kipróbálnánk a plug-inünket, ezt látnánk:

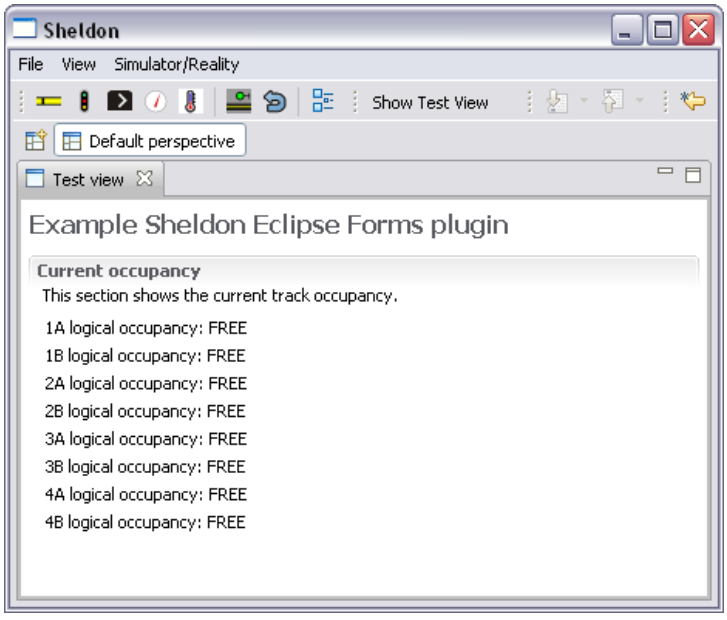


9. Készítsünk egy új metódust, ami minden vágányelemhez kitölti a címke szövegét az aktuális foglaltság alapján!

```
private void refreshOccupancy() {
	for (TrackElement te : LocalRepository.getRepo()
			.getTrackElements()) {
		if (trackLabels.containsKey(te.getId()) == false)
			continue;
		Label label = trackLabels.get(te.getId());
		String occ = te.currentOccupancy() ? "OCCUPIED" : "FREE";
		label.setText(te.getName() + " occupancy: " + occ);
	}
}
```

   a. Hívjuk meg a *refreshOccupancy* metódust a *createPartControl* metódus végén!

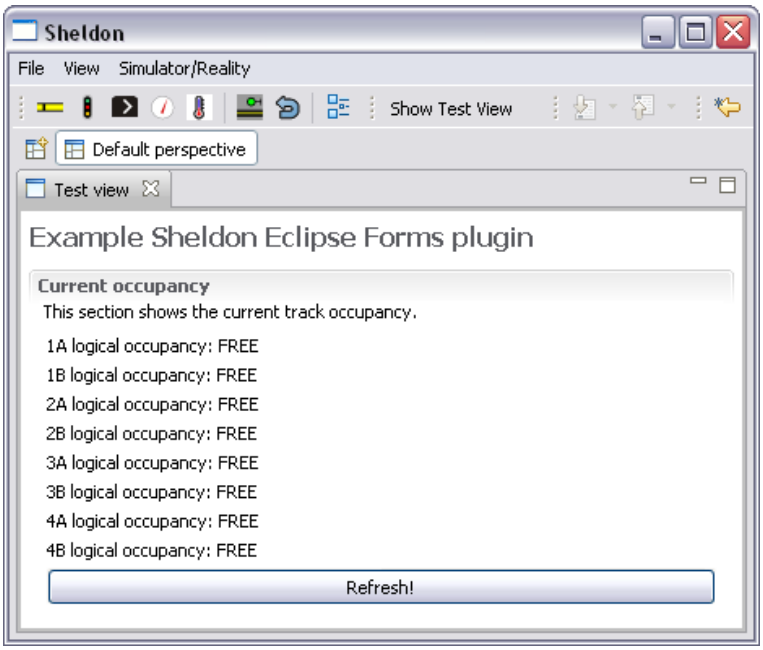
10. Ha most kipróbálnánk a plug-inünket, ezt látnánk:



11. Készítsünk egy gombot, amivel frissíteni tudjuk a foglaltságértékeket:

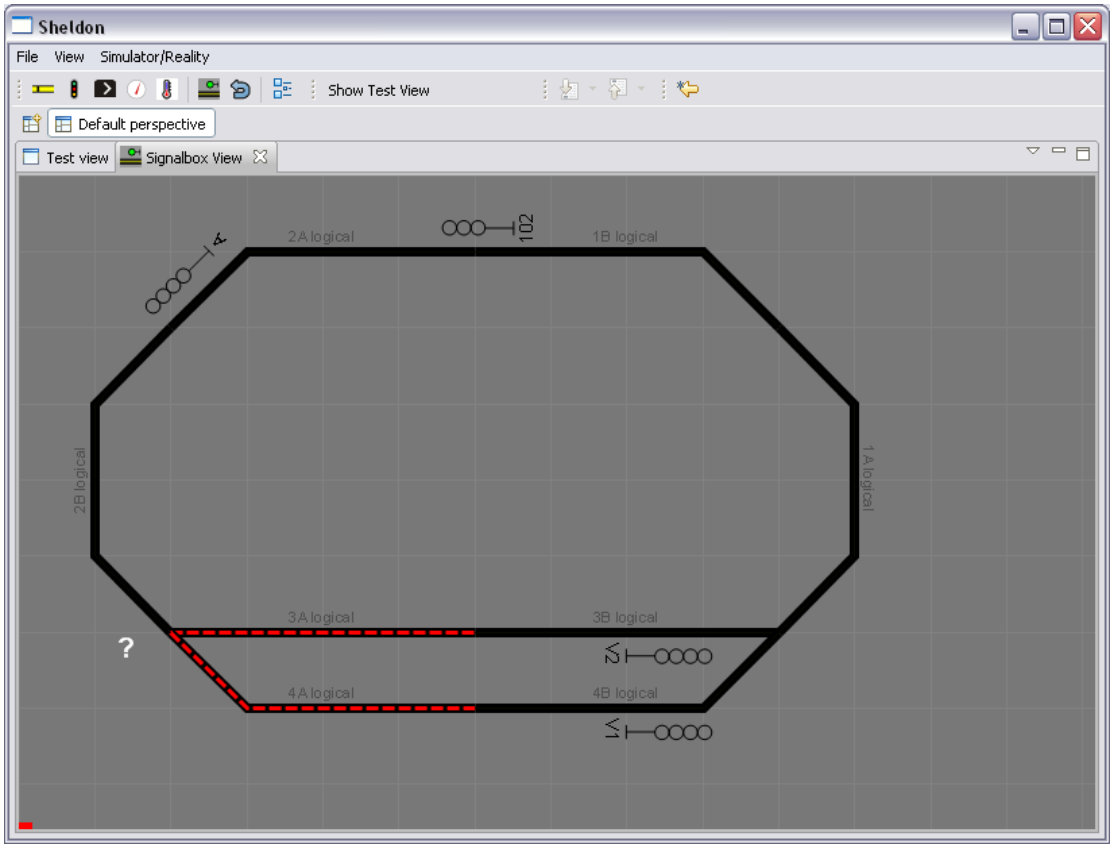
```
// gomb létrehozása
Button refresh = toolkit.createButton(occupBody, "Refresh!", SWT.NONE);

// eseménykezelő beállítása
refresh.addListener(SWT.Selection, new Listener() {
	@Override
	public void handleEvent(Event event) {
		// a gomb lenyomásakor végrehajtandó metódus
		refreshOccupancy();
	}
});
```

12. Indítsuk el a programot! Nyissuk meg a nézetünket, majd kapcsoljunk szimulátor módba a *Simulator* > *Simulator with 2 train* menüponttal. Ha most rákattintunk a *Refresh!* gombra, az alábbi képet láthatjuk:



a. Látható, hogy a 3A és 4A szakaszon állnak vonatok.

b. Válasszuk ki a *View > Signalbox* menüpontot! Itt grafikusan is láthatjuk a foglaltságokat. Látható, hogy valóban a 3A és 4A szakaszok foglaltak aktuálisan.



Megjegyzés: lehetőség van arra is, hogy a kijelzést automatikusan frissítsük. Minden *ModelTrainObject*-ből leszármazott objektum, így az egyes vágányszakaszok is képesek értesítést küldeni, ha módosult az állapotuk.

Ehhez a `track.addModificationListener(listener);` metódushívást kell végrehajtanunk, ahol a listener egy `IModelTrainObjectModificationListener` interfészű objektum. Amennyiben a példában a `track` objektum változik, úgy meghívja a `listener` objektum `onTrainObjectModification` metódusát.

A gyakorlatba átültetve az alábbi feladataink lennének, ha automatikusan frissíteni szeretnénk a kijelzést:

- A `TestView` osztályt kiegészítjük, hogy implementálja az `IModelTrainObject-ModificationListener` interfészt.

```
public class TestView implements IModelTrainObjectModificationListener { …
```

- Implementáljuk a módosulást kezelő függvényt.

```
public void onTrainObjectModification(ModelTrainObject sender){
      // a kijelzés frissítése ide kerül
}
```

- Feliratkozunk az inicializálás során az összes vágányszakasz eseménykezelőjére.

```
for (TrackElement te : LocalRepository.getRepository().getTrackElements()) {
      te.addModificationListener(this);
}
```

2.5 Kommunikáció a vezérlők felé

Egy egyszerű felületen keresztül lehetőségünk van arra is, hogy a Sheldonból (egy egy bájtos) üzenetet juttassunk el egy vezérlőhöz. Ehhez a vezérlőt reprezentáló `MbedController` osztály `sendByte(byte message)` metódusát használhatjuk.

A vezérlő számára ez az üzenetküldés egy speciális szinkronizációként látszik. E szinkronizáció csatornájának neve kötelezően `sheldon`, típusa `broadcast chan`, a küldött adat pedig az `int` típusú `sheldon_content` változóból olvasható ki.

Így tudunk például az 1-es ID-jű vezérlőnek egy fix tartalmú üzenetet eljuttatni egy gombra kattintva:

```
Button sendMessage = toolkit.createButton(occupBody, "Send!", SWT.NONE);
sendMessage.addListener(SWT.Selection, new Listener() {
	@Override
	public void handleEvent(Event event) {
		LocalRepository.getMbedControllerByID(1).
			sendByte((byte)42);
	}
});
```

Ha ezt arra szeretnénk felhasználni az 1-es vezérlőben, hogy addig ne induljon el a vezérlő futása, amíg nem kattintunk egy gombra rá a Sheldonban, azt például így tehetjük meg:

Opcionálisan őrfeltételt is írhatunk az élre, pl. `sheldon_content == 42`. Természetesen ez nem kötelező.

Figyelem! Ezen a csatornán keresztül nem lehet a Sheldon program felé üzenetet küldeni, azaz `sheldon!` szinkronizációt nem lehet csinálni.