Injection in Eclipse 4

27 March 2013

Eclipse Con Boston 2013
Introduction

OPCoach

➢ Training: RCP, E4, Modeling, Build in french, english and ... spanish (2013)
➢ Consulting
➢ Recruitment service to link companies and job applicants
➢ Twitter: @OPCoach_Eclipse, @OPCoach_Job
Injection and annotations

Introduction / Principle

➢ The goal of injection is to delegate the field or parameters initializations to a framework.
➢ Injection works by using a context that contains the values.
➢ We use the annotation @Inject (javax.inject) to inject the values.
➢ It can be applied to a constructor, a method or a field.
➢ The framework introspects classes to manage the artifacts that own an @Inject annotation.

Image 2 Sample injected class

ContextInjectionFactory Class

A class with injection annotations:
➢ must be instantiated using the ContextInjectionFactory
➢ can not be instantiated with a call to new

ContextInjectionFactory instantiates objects and puts them in context.
This is a static class.
The injection context

- The context stores the values associated with keys or classes
- The context is hierarchical
- Branches in context are activated by the framework (UI focus, part activation, ...)
- Branch activation gives an access to the available values in this branch
- If a value is not found, the context will ask its parent (`getParent()`)
- The context is an implementation of `IEclipseContext`

Example of use (in current branch)

```java
package com.opcoach.training.e4.codeexamples;

import org.eclipse.e4.core.contexts.ContextInjectionFactory;
import org.eclipse.e4.core.contexts.IEclipseContext;
import org.eclipse.e4.core.contexts.IContextInjectionFactory;
import org.eclipse.e4.core.modeling.BundleActivator;

public class AnActivatorFillingTheContext { // is implements BundleActivator

    public void defineMyService(IEclipseContext context) {
        // Create a MyService instance and put it in the context
        ContextInjectionFactory.make(MyService.class, context);
    }

    public void start(BundleContext context) throws Exception {
    }

    public void stop(BundleContext context) throws Exception {
    }
}
```

Additional annotations for `@Inject`

3 additional annotations are used to manage object creation:

- `@PostConstruct`: is used to annotate methods that must be called at the end of initialization
- `@Optional`: is used to indicate that a field or a parameter could be null
@Named: is used to get an object by its name

Call order

The injection is processed in the following order:

- @Inject Constructor call
- @Inject Fields initialization
- @Inject Methods call
- @PostConstruct Methods call

```
package com.opcoach.training.e4.codesamples.inject;
import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;
import org.eclipse.e4.core.di.annotations.Optional;
import org.eclipse.e4.ui.workbench.modeling.RGBSelectionService;

public class CallOrder {
    public static final String MYPARAM = "myparam";
    // --> Call #2 field is initialized after call to constructor
    @Inject
    private ESelectionService selectionService;
    // --> Call #1 Constructor is called first
    public CallOrder() {
        @Inject
        // --> Call #3 (after field init)
        public void aMethod() {

            @Inject
            // --> Call #4
            public void anotherMethod() {

                @Inject
                // --> Call #5 and after when a value change
                public void methodWithInjectedParam(Optional @Named(MYPARAM) Object o) {

                    @PostConstruct
                    // --> Call #6 at the end
                    public void endOfInit() {
                }
            }
        }
    }
}
```

Constructor call (1)

- ContextInjectionFactory will choose a constructor with an @Inject annotation
- If several constructors are annotated, it chooses the constructor having the maximum parameters that can be injected
- Fields marked with @Inject have a null value in the constructor.
- Other fields can be used normally
- Parent constructors are called by the super() (implicit) or by super(...) (explicit)

Fields Injection (2)

- Each field preceded by a @Inject annotation is initialized after the constructor call
➢ The order of initialization is unspecified
➢ If a field can not be injected the injector generates an exception
➢ If the value changes after the injection, it will be re-injected automatically
➢ If a field can have no value, it must be annotated with @Optional
➢ The inherited @Inject fields are initialized before the others

@ Inject methods invocation (3)

➢ Methods annotated with @Inject are called:
  ➢ after the constructor call
  ➢ after the injected fields initialization
  ➢ if an injected parameter value has changed
➢ If several methods are annotated with @Inject, the call order is undefined
➢ The inherited @Inject methods are called before the others
➢ The inherited and overriden @Inject methods are not called

@ PostConstruct methods invocation (4)

➢ Methods annotated with @PostConstruct are called:
  ➢ after the calls to each @Inject methods
➢ If several methods are annotated with @PostConstruct, the call order is undefined
➢ The inherited @PostConstruct methods are called before the others
➢ The inherited and overriden @PostConstruct methods are not called
**Fundamental: Basic rule of injection**

If a value previously injected into the injector changes after injection:
- it will automatically be re-injected into the relevant `@Inject` fields
- `@Inject` methods that have received this value as a parameter are automatically re-called.

A full sample with inheritance

```
Parent
- Parent()
- parentMethod1()
- parentMethod2()
- parentMethodToBeOverriden()
- parentMethodWithChangeableParam(E.javaObject)
- parentPostConstruct()

Child
- Child()
- Child(ESectionService)
- childMethod1()
- childMethod2()
- childMethodWithChangeableParam(E.javaObject)
- childMethodToBeOverriden()
- childMethodPostConstruct1()
- childMethodPostConstruct2()
```

All methods and constructors have a `@Inject` and `*Postconstruct?()` have a `@Postconstruct`
**Child instanciation**

```java
public class Parent {
    public static final String PARAM_IN_PARENT="parentParam";

    public Parent() {
        @Inject public Parent(Adapter adapter) {
        }
        @Inject public Parent(EmailService service) {
        }
    }

    @Inject public void parentMethod() {
    }

    @Inject public void parentMethodWithParam(String param) {
    }

    @PostConstruct public void parentMethodPostConstruct() {
    }
}

public class Child extends Parent {
    public static final String PARAM_IN_CHILD="childParam";

    @Inject private ESelectionService selectionService;

    public Child() {
        @Inject public Child(EmailService ss) {
        }
    }

    @Inject public void childMethod() {
    }

    @Inject public void childMethod2() {
    }

    @Inject public void childMethodWithChangeableParam(String param) {
    }

    @Override @Inject public void parentMethodOverride() {
    }

    @PostConstruct public void childMethodPostConstruct() {
    }
    @PostConstruct public void childMethodPostConstruct2() {
    }
}
```

**Modifying a parameter injected in parent**

```java
public class Parent {
    public static final String PARAM_IN_PARENT="parentParam";

    public Parent() {
        @Inject public Parent(Adapter adapter) {
        }
        @Inject public Parent(EmailService service) {
        }
    }

    @Inject public void parentMethod() {
    }

    @Inject public void parentMethodWithParam(String param) {
    }

    @PostConstruct public void parentMethodPostConstruct() {
    }
    @Call
}

public class Child extends Parent {
    public static final String PARAM_IN_CHILD="childParam";

    @Inject private ESelectionService selectionService;

    public Child() {
        @Inject public Child(EmailService ss) {
        }
    }

    @Inject public void childMethod() {
    }

    @Inject public void childMethod2() {
    }

    @Inject public void childMethodWithChangeableParam(String param) {
    }

    @Override @Inject public void parentMethodOverride() {
    }

    @PostConstruct public void childMethodPostConstruct() {
    }
    @PostConstruct public void childMethodPostConstruct2() {
    }
}
```

--- Step 1: create the ChildCallOrder
--- Parent Call parentMethod
--- Parent Call parentMethodWithParam with param: null
--- Child Call: childMethod
--- Child Call: childMethod2
--- Child Call: childMethodWithChangeableParam
--- Child Call: parentMethodWithOverride
--- Parent Call: parentMethodPostConstruct
--- Child Call: childMethodPostConstruct
--- Child Call: childMethodPostConstruct2

--- Step 2: change the parameter used by parent
--- Parent Call parentMethodWithChangeableParam with param: new value for parent parameter
Modifying a parameter injected in child

```java
public class Parent {
    public static final String PARAM_IN_PARENT = "parentParent";
    public Parent() {
        @Inject public Parent(Adapter adapter) {
            System.out.println("Step 1: change the parameter used by child");
        } @Inject public Parent(toString service) {
            System.out.println("Step 2: change the parameter used by child");
            ctx.set(Child.PARAM_IN_PARENT, "new value for child parameter");
            }
    }
}

public class Child extends Parent {
    public static final String PARAM_IN_CHILD = "ypopar";
    public Child() {
        @Inject public Child(IChildService service) {
            System.out.println("Step 3: change the parameter used by child");
            ctx.set(Child.PARAM_IN_CHILD, "new value for child parameter");
        } @Inject public Child() {
            System.out.println("Step 4: change the parameter used by child");
            }
    }
}
```

Image 7 Change param in child

Annotations managed by Eclipse 4

- **@ Named**: To inject an object using a name
- **@ Singleton**: To indicate that a class should be instantiated once
- **@ Preference**: to inject the value stored in the preferences
- **@ Creatable**: to inject an object, and if it does not exist, allows its instanciation (this is a class annotation)
- **@ PreDestroy**: to annotate a method called before the object is destroyed

Sample with `@Named`

```java
@Named @Optional
public void setSelection(Object service, ActiveSelection selection) {
    Adapter adapter = (Adapter) service;
    adapter.setSelection(selection);
}
```

Image 8 `@Inject @Named @Optional`

Predefined names

The `IServiceConstants` class defines constants for common objects:

- ACTIVE_SELECTION
- ACTIVE_CONTEXTS
- ACTIVE_PART
- ACTIVE_SHELL
Sample with @Preference

The @Preference retrieves the value of a preference.

It can be used for a class field or for a parameter method

```java
private final static String PLUGIN_ID = RentalUIActivator.PLUGIN_ID;

@Inject
public void refreshTree(@Preference(nodePath=PLUGIN_ID, value=CUSTOMER_KEY) String custCol,
                        @Preference(nodePath=PLUGIN_ID, value=RENTAL_KEY) String rk,
                        @Preference(nodePath=PLUGIN_ID, value=RENTAL_OBJECT_KEY) String rol)
{
    if (agencyViewer != null)
        {
            labelProvider.initPalette();
            agencyViewer.refresh();
        }
}
```

Image 9 @Preference

Injection: advantages / disadvantages

Benefits:
- Significant reduction in the coupling (inheritance with the legacy framework is no longer required)
- Source code depends only on the high-level interfaces
- Easy to find objects: usually if you need it, you inject it

Disadvantages:
- When debugging an injected method, the caller code is introspection code

Question

But what are the values that can be injected?
Injection Contexts

Eclipse allows you to define hierarchical injection contexts. The default Eclipse injection context contains:

- the application model
- OSGi services
- all objects explicitly added to the context

An object is sought in the current context, and then in the parent context if not found.

Local contexts

Objects that implement the interface `MContext` have their own local context. These objects are connected to the parent context. For instance, these objects extends the `MContext`:

- `MApplication`
- `MWindow`
- `MPerspective`
- `MPart`
- `MPopupMenu`

To get the context of a `MContext` just call the `getContext()` method.

Contexts for an application

Image 10 Contexts in application
Context Explorer

Available on github: https://github.com/opcoach/contextExplorer

It is used to display all the contexts and their contents
### Workbench context

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeChildContext</td>
<td>TrimmedWindowimplContext</td>
</tr>
<tr>
<td>applicationCSS</td>
<td>platform:/plugin/com.opencoh.4.contextexplorer/css/default.css</td>
</tr>
<tr>
<td>applicationCSSResources</td>
<td>null</td>
</tr>
<tr>
<td>applicationXMI</td>
<td>com.opencoh.4.contextexplorer/Application.xmi</td>
</tr>
<tr>
<td>bindingTable.org.eclipse.uicontextsdialog</td>
<td>org.eclipse.e4.ui.bindings.internal.BindingTable22f82a14</td>
</tr>
<tr>
<td>bindingTable.org.eclipse.uicontextsdialogAndWindow</td>
<td>org.eclipse.e4.ui.bindings.internal.BindingTable111edeb</td>
</tr>
<tr>
<td>bindingTable.org.eclipse.uicontexts.window</td>
<td>org.eclipse.e4.ui.bindings.internal.BindingTable82bca8cc</td>
</tr>
<tr>
<td>clearPersistedState</td>
<td>true</td>
</tr>
<tr>
<td>com.opencoh.4.contextexplorer.addonSampleAddon</td>
<td>com.opencoh.4.contextexplorer.addonSampleAddon.cs5624</td>
</tr>
<tr>
<td>com.opencoh.4.contextexplorer.lifecycle.PostContextinitializercssTheme.null</td>
<td>com.opencoh.4.contextexplorer.lifecycle.PostContextinitializer89236cd5</td>
</tr>
<tr>
<td>debugString</td>
<td>WorkbenchContext</td>
</tr>
<tr>
<td>deltaRestore</td>
<td>true</td>
</tr>
<tr>
<td>handler.org.eclipse.ui.cocoa.arrangewindowsinfront</td>
<td>org.eclipse.e4.ui.workbench.renderers.swt.cocoa.ArrangeWindowsHandler...</td>
</tr>
<tr>
<td>handler.org.eclipse.ui.cocoa.fullscreenwindow</td>
<td>org.eclipse.e4.ui.workbench.renderers.swt.cocoa.FullscreenWindowHandler...</td>
</tr>
<tr>
<td>handler.org.eclipse.ui.cocoa.minimizewindow</td>
<td>org.eclipse.e4.ui.workbench.renderers.swt.cocoa.MinimizeWindowHandler...</td>
</tr>
<tr>
<td>handler.org.eclipse.ui.file.exit</td>
<td>com.opencoh.4.contextexplorer.handlers.QuitHandler139241b1</td>
</tr>
<tr>
<td>handler.org.eclipse.ui.help.aboutAction</td>
<td>com.opencoh.4.contextexplorer.handlers.AboutHandler211bd3e2</td>
</tr>
<tr>
<td>initialWorkbenchModelURI</td>
<td>platform:/plugin/com.opencoh.4.contextexplorer/Application.xmi</td>
</tr>
<tr>
<td>instanceLocation</td>
<td>org.eclipse.core.runtime.internal.adapter.BasicLocation@c705d65</td>
</tr>
<tr>
<td>limbo</td>
<td>Shell ()</td>
</tr>
<tr>
<td>localContexts</td>
<td>[org.eclipse.ui.contexts.dialogAndWindow, org.eclipse.core.commands.CommandManager@4e9722c9]</td>
</tr>
<tr>
<td>org.eclipse.core.commands.CommandManager</td>
<td>org.eclipse.core.commands.CommandManager@4e9722c9</td>
</tr>
<tr>
<td>org.eclipse.core.commands.ContextManager</td>
<td>org.eclipse.core.commands.ContextManager@7c39b3a66</td>
</tr>
<tr>
<td>org.eclipse.core.databinding.observable.Realm</td>
<td>org.eclipse.jface.databinding.swt.IWDatabindingImpl@7779d9c0d</td>
</tr>
<tr>
<td>org.eclipse.core.commands.ECommandService</td>
<td>org.eclipse.core.commands.internal.CommandServiceImpl@2c6691da</td>
</tr>
<tr>
<td>org.eclipse.core.internal.contexts.ContextObjectSupplier</td>
<td>org.eclipse.core.internal.contexts.ContextObjectSupplier@5097d026</td>
</tr>
<tr>
<td>org.eclipse.core.services.log.Logger</td>
<td>org.eclipse.ui.internal.workbench.WorkbenchLogger@1e2e3920</td>
</tr>
<tr>
<td>org.eclipse.core.bindings.keys.KeyBindingDispatcher</td>
<td>org.eclipse.core.bindings.keys.KeyBindingDispatcher@350d4376</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.di.UISynchronize</td>
<td>org.eclipse.ui.internal.workbench.swt.EApplication102354c054</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.internal.workbench.swt.CSSRenderingUtils</td>
<td>org.eclipse.ui.internal.workbench.swt.CSSRenderingUtils@2ab565ce</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.model.application.IMApplication</td>
<td>org.eclipse.ui.model.application.IMApplication@8930a4c16e161e20</td>
</tr>
</tbody>
</table>
Trimmed Window context

Image 12 Trimmer Window Context

Perspective context

Image 13 Perspective context
Part context

Handling context
Annotations extracts objects from context automatically
You can use the `IEclipseContext` class to add objects in context
To put objects in a context:

- use the annotation `@PostContextCreate` on a method
- add a `lifeCycleURI` property in the product extension

```java
/**
 * The activator class controls the plug-in life cycle
 * This class must be registered as a lifeCycleURI to fill the context
 */

public class RentalUIActivator extends AbstractUIPlugin implements RentalUIConstants {
    [PostContextCreate]
    private void initAgency(IEclipseContext context) {
        context.set(RentalAgency.class, RentalAgencyGenerator.createSampleAgency());
        // fill the context here
    }
```
define a lifecycleURI property with a bundleclass reference to the class containing the @Postcontextcreate annotation.

Warning: use the bundleclass:// annotation to refer to the class.

Extension Element Details
Set the properties of "property". Required fields are denoted by "*".

name*: lifecycleURI
value*: bundleclass://com.opcoach.training.e4.rental.ui/com.opcoach.trac

Updating context using an addon
The application model allows you to add Addon
The addon is used to fill the context (received with injection)
Addon code sample:

```java
/*
  public class CommandServiceAddon {
    @PostConstruct
    public void init(IEclipseContext context) {
        // global command service. There can be only one ... per application :-)
        CommandManager manager = context.get(CommandManager.class);
        if (manager == null) {
            manager = new CommandManager();
            setCommandFireEvents(manager, false);
            context.set(CommandManager.class, manager);
        }

        CommandServiceImpl service = ContextInjectionFactory
            .make(CommandServiceImpl.class, context);
        context.set(ECommandService.class, service);

        // handler service - a mediator service
        context.set(EHandlerService.class.getName(), new HandlerServiceCreationFunction());
    }
  }
*/
```

Image 19 Addon