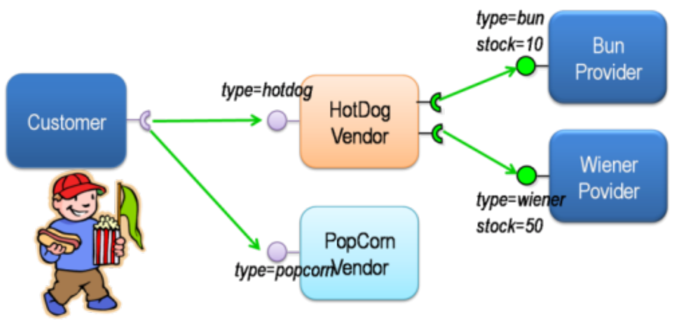
Tutorial – iPOJO

1. Context

This tutorial is based on a pretty simple application. This application 'simulates' a snack bar where products (hotdog, popcorn) are provided by vendors exposed as services. This simple application exhibits a lot of interesting use cases such:

- To sell hot dogs, a vendor needs both buns and wieners
- To sell pop-corn, only corn is required, salt and sugar are optional



The objectives of this tutorial is to illustrate how to use basic iPOJO features, as well as introduce less basic features and how iPOJO can be used in real world applications.

2. Preparation

Download the archive from:
http://people.apache.org/~clement/ipojo/tutorials/ipojo-training/ipojo-training.zip.

Unzip it.
This tutorial relies on OW2 Chameleon and Apache Felix. It provides a pretty simple way to launch the application and to manage it.

Before launching anything, edit the 'chameleon/chameleon.properties' files and set the HTTP Port to a valid port (property 'org.osgi.service.http.port') like 8080. You must choose a different HTTP Port (each group a different one).

The chosen port is denoted as 'port' in the remaining of the document.

2.1. Launching the Chameleon

A chameleon is a 'ready-to-go' OSGi distribution. The provided chameleon is shipped with the basic services used in this training session. To launch the chameleon, go to the chameleon directory and execute *chameleon-start.(sh|bat)*.

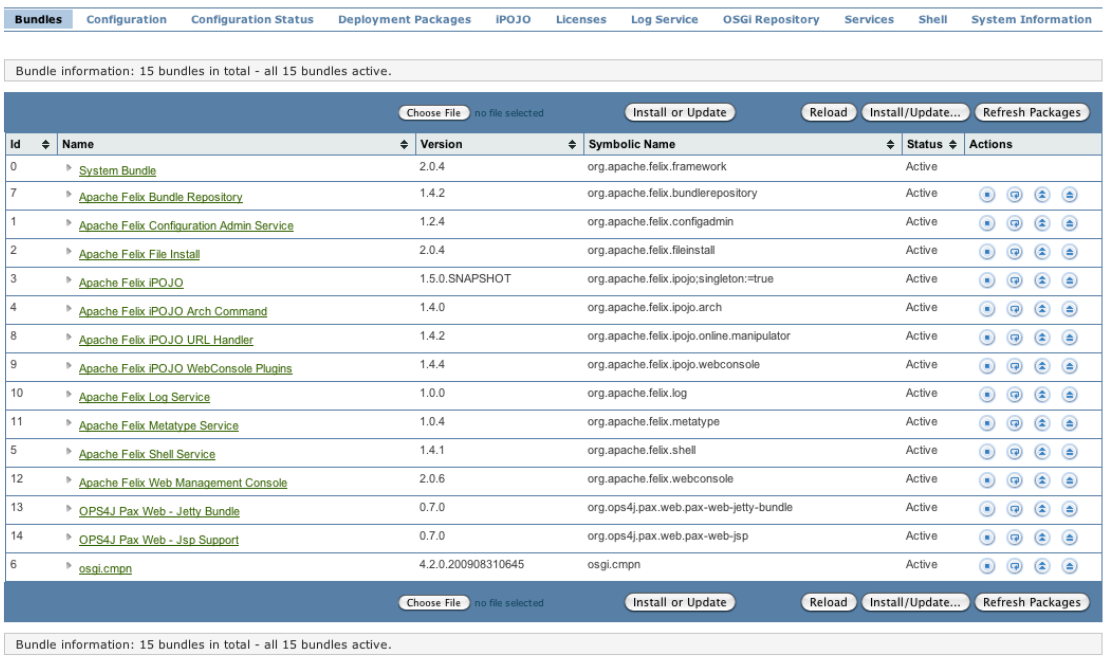
Note: For linux / unix / macos users, you must add the execution permission to the script with 'chmod 774 chameleon-start.sh'.

Note: For windows users, the chameleon is started in another command windows.

This command starts and provisioned bundles contained in 'core', 'runtime', 'application' and 'deploy'. The 'deploy' folder is a dynamic folder. Bundles dropped inside this directory will be automatically installed and started. Removing a bundle from this directory is equivalent to uninstalling it. This also works with configurations.

2.2. Launching the web console

This chameleon starts a bundle allowing administering the OSGi platform via a web interface. Once the chameleon is launch, open: http://localhost:port/system/console/bundles (login/password: admin/admin)



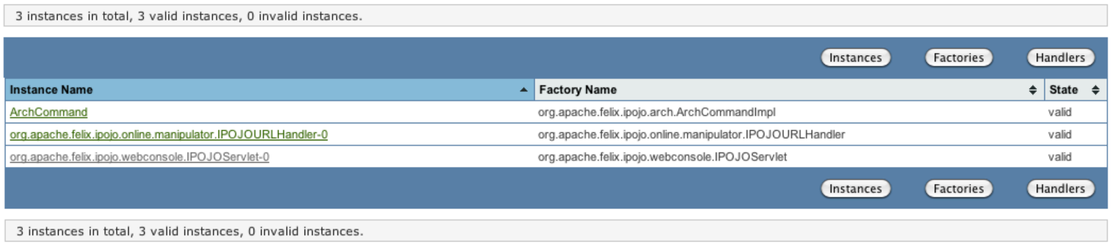
Bundle information: 15 bundles in total - all 15 bundles active.

Id	Name	Version	Symbolic Name	Status	Actions
0	System Bundle	2.0.4	org.apache.felix.framework	Active	
7	Apache Felix Bundle Repository	1.4.2	org.apache.felix.bundlerepository	Active	
1	Apache Felix Configuration Admin Service	1.2.4	org.apache.felix.configadmin	Active	
2	Apache Felix File Install	2.0.4	org.apache.felix.fileinstall	Active	
3	Apache Felix iPOJO	1.5.0.SNAPSHOT	org.apache.felix.ipojo;singleton:=true	Active	
4	Apache Felix iPOJO Arch Command	1.4.0	org.apache.felix.ipojo.arch	Active	
8	Apache Felix iPOJO URL Handler	1.4.2	org.apache.felix.ipojo.online.manipulator	Active	
9	Apache Felix iPOJO WebConsole Plugins	1.4.4	org.apache.felix.ipojo.webconsole	Active	
10	Apache Felix Log Service	1.0.0	org.apache.felix.log	Active	
11	Apache Felix Metatype Service	1.0.4	org.apache.felix.metatype	Active	
5	Apache Felix Shell Service	1.4.1	org.apache.felix.shell	Active	
12	Apache Felix Web Management Console	2.0.6	org.apache.felix.webconsole	Active	
13	OPS4J Pax Web - Jetty Bundle	0.7.0	org.ops4j.pax.web.pax-web-jetty-bundle	Active	
14	OPS4J Pax Web - Jsp Support	0.7.0	org.ops4j.pax.web.pax-web-jsp	Active	
6	osgi.cmpn	4.2.0.200908310645	osgi.cmpn	Active	

Bundle information: 15 bundles in total - all 15 bundles active.

The 'action' buttons allow starting, stopping, updating, and uninstalling a bundle.

The iPOJO view (click on iPOJO) lists the instances, the factories and handlers. It provides a navigable view were you can see who provides used services…



3 instances in total, 3 valid instances, 0 invalid instances.

Instance Name	Factory Name	State
ArchCommand	org.apache.felix.ipojo.arch.ArchCommandImpl	valid
org.apache.felix.ipojo.online.manipulator.IPOJOURLHandler-0	org.apache.felix.ipojo.online.manipulator.IPOJOURLHandler	valid
org.apache.felix.ipojo.webconsole.IPOJOServlet-0	org.apache.felix.ipojo.webconsole.IPOJOServlet	valid

3 instances in total, 3 valid instances, 0 invalid instances.

2.3. Compilation and Deployment

The project uses Apache Ant to compile and package bundles. On each project, three main goals are available:

- package: compile and create the bundle in the 'target' directory

- install: copy the create bundle to the 'live' folder of the chameleon (*deploy*)
- clean: all is in the goal name…

Launch them by navigating in the project folder, and launch 'ant clean package install'. This command will compile, package and deploy the bundle. Launch them in directories containing a 'build.xml' file.

At the end of the session, you can create a distribution of your work with: 'ant assembly' A zip file will be created in the 'dist' directory.

3. The service interfaces

The service interfaces used in this tutorial are mostly (expect one for demonstration purpose) packaged in their own bundle named 'service-interfaces'.

Open the source files from the 'de.akquinet.gomobile.ipojo.training.service' package (in the service-interfaces folder) to have a look to the service interfaces:

- VendorService is implemented by vendors selling Products
- ResellerService is implemented by resellers providing the ingredients to vendors

1. <u>Task:</u>
Read the service interfaces and explain why they are packaged in a separated bundle. Also explain why service properties are specified in the service interfaces.

Then, package and deploy the service interfaces by launching 'ant clean package install'.

4. The Shop Servlet

Open the Shop-Servlet project and read the ShopServlet.java class.

2. <u>Task:</u>
Explain the behavior of this component. Are the synchronization policies valid for the start and stop method?

Once done, package and deploy the Shop-Servlet bundle. Open http://localhost:port/shop with your browser.

3. <u>Task:</u>
Is the displayed page what you expected?

5. Providing a Service: The Corn Service

In this section, you will implement a CornVendor selling Corn. For demonstration purpose, this implementation does not implement the ResellerService but embeds its own service interface.

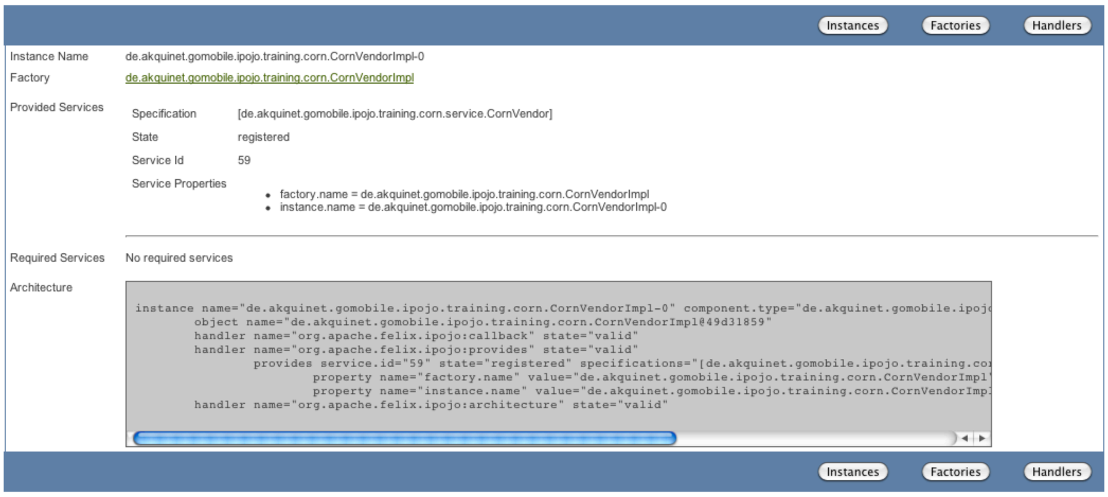
4. *Task:*
   What do you think about this choice?

Open the 'de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl' java class.

5. *Task:*
   Edit the CornVendorImpl class to provide the CornVendor service (don't forget to add the @Instantiate annotation). Moreover, imports must be added for the annotation classes - e.g. import org.apache.felix.ipojo.annotations.Component; ….

Once done, package and deploy the bundle (ant clean package install). To check that your component is correctly implemented, open the web console, go to the iPOJO view, and select the 'de.akquinet.gomobile.ipojo.training.CornVendorImpl-0' instance.

You should see something like:



Instance Name	de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl-0
Factory	de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl

Provided Services

Specification	[de.akquinet.gomobile.ipojo.training.corn.service.CornVendor]
State	registered
Service Id	59
Service Properties	• factory.name = de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl • instance.name = de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl-0

Required Services No required services

Architecture

```
instance name="de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl-0" component.type="de.akquinet.gomobile.ipoj
        object name="de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl@49d31859"
        handler name="org.apache.felix.ipojo:callback" state="valid"
        handler name="org.apache.felix.ipojo:provides" state="valid"
                provides service.id="59" state="registered" specifications="[de.akquinet.gomobile.ipojo.training.co
                        property name="factory.name" value="de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl
                        property name="instance.name" value="de.akquinet.gomobile.ipojo.training.corn.CornVendorImp
        handler name="org.apache.felix.ipojo:architecture" state="valid"
```

6. *Task:*
   Explain the factory value, and the provided services section.

6. Requiring a Mandatory Service: The PopCornVendor

In this section, you will implement a VendorService selling PopCorn and relying on the CornVendor service defined and implemented previously.

Open the 'de.akquinet.gomobile.ipojo.training.popcorn.PopCornVendor' class from the popcorn-vendor project.

7. *Task:*
   Edit the class to declare a component type providing the VendorService and create an instance of it.

8. *Task:*
   Configure the m_corn field to receive a CornVendor service object. The service dependency must be scalar and mandatory.

9. *Task:*

In the buy method call the CornVendor to get corn before creating the PopCorn product.

10. Task:
Explain the two service properties.

Once done, package and deploy the bundle (ant clean package install). Then, reload the shop. If everything is correct, you should see the popcorn vendor, and you can buy pop-corn.

11. Task:
In the web console, go to the bundle view, and stop the 'corn-vendor' bundle (with the stop button). Go back to the shop, and refresh the page. Explain the result.

Restart the corn vendor bundle from the web console.

7. Exposing service properties: The Wiener Reseller

Now that you have created your first vendor, you will create a slightly more complex vendor (selling hotdogs). This vendor requires two resellers: a wiener reseller and a bun reseller. In this section, you will implement wiener reseller exposing service properties.

12. Task:
Open the wiener-provider project and fill the de.akquinet.gomobile.ipojo.training.wiener.WienerReseller class. This component must be instantiated, and implement the Reseller Service specification. The provided type is 'wiener'.

Once done, package and deploy the bundle (ant clean package install). Then, check that the Reseller Service providing Wiener is correctly exposed.

8. Properties and Lifecycle: The Bun Reseller

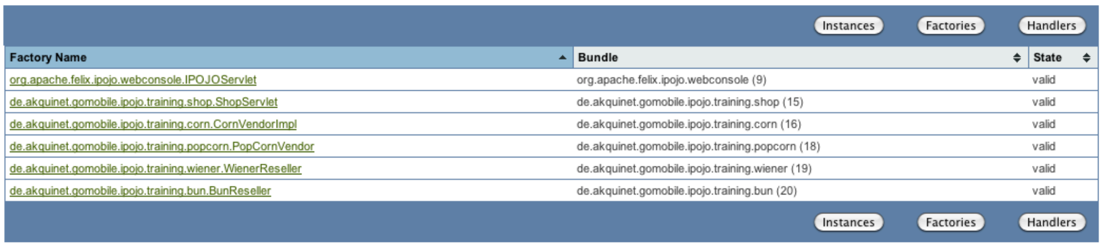
The bun reseller is more complex. Indeed, there are potential starvations… In this section, you will implement the Bun Reseller as a configurable instance (stock). If the stock reaches 0, the service is withdrawn from the service registry.

The Bun Reseller will not be instantiated by default (so no @Instantiate annotation). Instead, a configured instance will be creating with the initial stock. This property is declared with the @Property annotation.

To control the service exposition, the @ServiceController annotation is used. When set to false, the services are withdrawn from the service registry. When set to true, the services are published.

13. Task:
Fill the de.akquinet.gomobile.ipojo.training.bun.BunReseller class to follow the description.

Deploy your bundle (ant clean package install). In the web console, in the iPOJO view, no instances are created. But, if you click on 'Factories', you should see the component type:



Factory Name	Bundle	State
org.apache.felix.ipojo.webconsole.IPOJOServlet	org.apache.felix.ipojo.webconsole (9)	valid
de.akquinet.gomobile.ipojo.training.shop.ShopServlet	de.akquinet.gomobile.ipojo.training.shop (15)	valid
de.akquinet.gomobile.ipojo.training.corn.CornVendorImpl	de.akquinet.gomobile.ipojo.training.corn (16)	valid
de.akquinet.gomobile.ipojo.training.popcorn.PopCornVendor	de.akquinet.gomobile.ipojo.training.popcorn (18)	valid
de.akquinet.gomobile.ipojo.training.wiener.WienerReseller	de.akquinet.gomobile.ipojo.training.wiener (19)	valid
de.akquinet.gomobile.ipojo.training.bun.BunReseller	de.akquinet.gomobile.ipojo.training.bun (20)	valid

9. Instantiating a Bun Reseller

In this section, you will create an instance of the Bun Reseller. There is a couple of way to do that:

- Using an XML file
- Using the cfg files and the configuration admin
- Using the iPOJO Factory service

You will use the second approach. It consists in creating a .cfg files containing the properties. This file will be analyzed and push to the configuration admin (an OSGi service managing configurations). This service will send the configuration to the component type, and the instance will be created.

14. <u>Task:</u>
Create a file named 'de.akquinet.gomobile.ipojo.training.bun.BunReseller-bun1.cfg' in the chameleon/deploy folder. This folder is a live folder. So, the file will be discovered, analyzed and the configuration will be pushed to the config admin. The file name is important, and must follow the following name rule: component_type_name-instance_name.cfg. In your case, the component type name is the class name. In the file, write:

```
stock: 10
```

When the file is saved, reload the iPOJO view of the web console. A bun reseller instance should be here.

10. Using Requirement Filters: the hotdog vendor

You now have the two resellers selling buns and wieners. It's time to implement the hotdog vendor relying on those resellers.

Open the hotdog-vendor project to fill the 'de.akquinet.gomobile.ipojo.training.hotdog.HotDogVendor' class.

15. <u>Task:</u>

Fill the HotDog vendor class. Before returning the product you must get a bun and a wiener. The @Requires annotation has a filter attribute receiving an OSGi LDAP filter.

Once implemented, package and deploy the bundle (ant clean package install). The hotdog vendor should appear in the shop.

16. <u>Task:</u>
Try to buy hotdogs several times (10 times). What is happening? Open the web console and look at the state of the HotDog vendor instance. Could you explain why the hotdog vendor is invalid?

11. Dynamic reconfiguration: stock management

17. <u>Task:</u>
Edit the cfg file from the 'deploy' folder and save it again. Then, refresh the shop. Explain what is happening when the file is saved.

18. <u>Task:</u>
Clone the cfg file and rename it (just the last part of the name). Open the web console and check the iPOJO instance. What do you remark?

19. <u>Task:</u>
Now try to buy hotdogs from the shop. What happens when the first bun provider 'closes'?

12. Optional Requirement: The log service

In all the components, the log uses SLF4J. However, OSGi defines a log service too.

20. <u>Task:</u>
*Adapt one of your components to replace the log by an **optional** service requirement on LogService (org.osgi.service.log.LogService).*

21. <u>Task:</u>
What happens if the log service is not available? To check this behavior, stop the bundle named 'Apache Felix Log Service' from the web console.

13. Default Implementation: strategy pattern

iPOJO also supports default-implementation to replace a missing service. iPOJO creates an instance of this class when a service is not available. The Default implementation class **must** implement the service interface.

22. <u>Task:</u>
Provide a default implementation of the log service delegating to a SLF4J logger. Check the behavior when the log service is available and when the log service is not available.

14. Aggregate Dependencies: The combo vendor

23. <u>Task:</u>

Implement from scratch (create the complete project) a vendor service that aggregates all the others vendors to provide a 'combo'.