



# Formation Architecture Logicielle

## *Exercices*



Formateurs :

Frantz Degrigny, frantz.degrigny@keyconsulting.fr

Méric Garcia, meric.garcia@keyconsulting.fr

Téléphone : 02.30.96.02.75

<http://www.keyconsulting.fr>

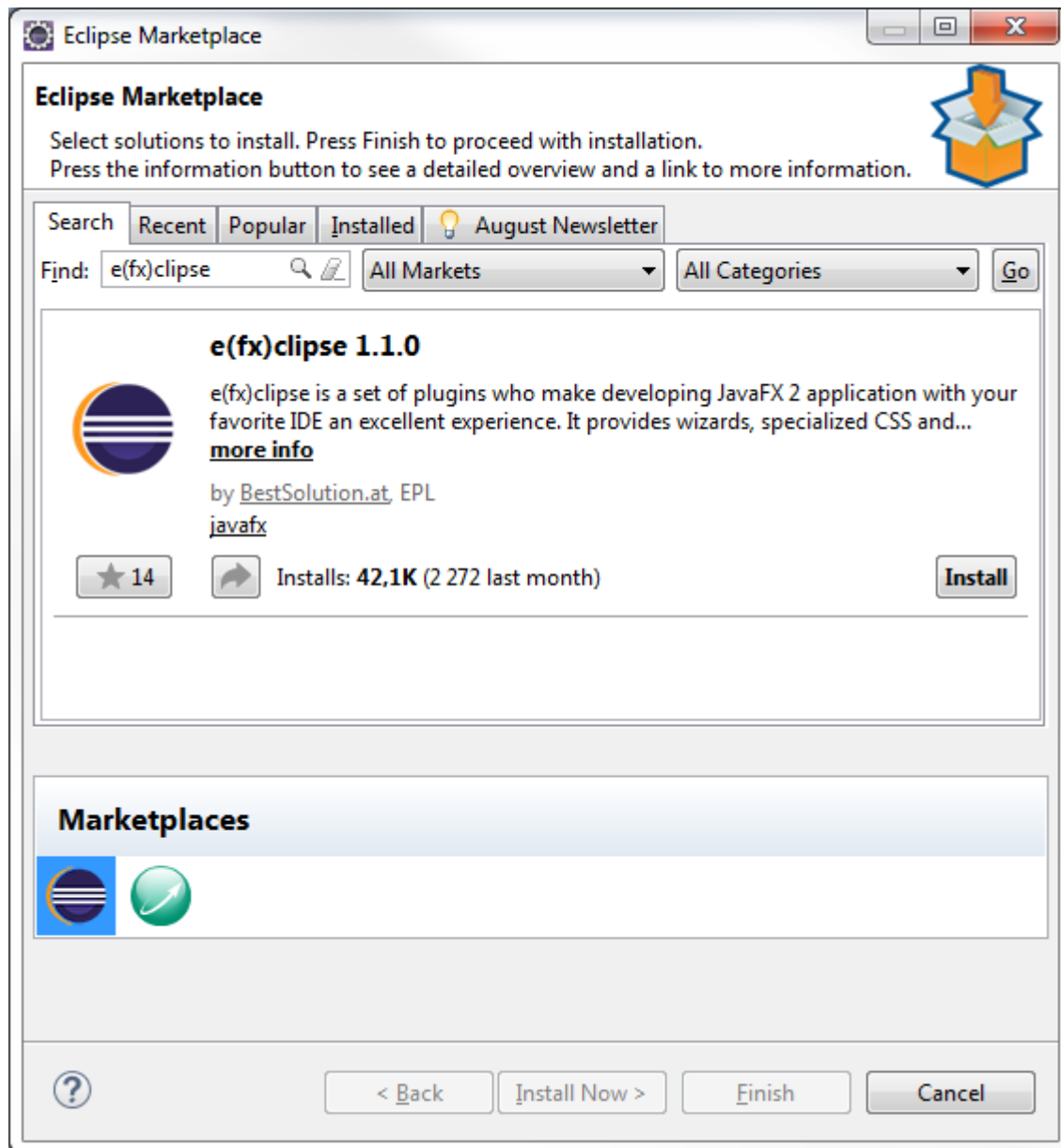
## Sommaire

<b>1</b>	<b>TECHNIQUES DE BASE .....</b>	<b>4</b>
1.1	INSTALLER E(FX)CLIPSE PLUGIN JAVA FX POUR ECLIPSE .....	4
1.2	ABSTRACT FACTORY .....	5
1.2.1	<i>Les fabriques.....</i>	5
1.2.2	<i>Le programme .....</i>	5
1.3	SINGLETON.....	6
1.3.1	<i>Un Properties manager : .....</i>	6
1.3.2	<i>Le programme .....</i>	6
1.4	ABSTRACT FACTORY + SINGLETON .....	7
1.4.1	<i>Une classe abstraite .....</i>	7
1.4.2	<i>Le programme .....</i>	7
1.5	SPRING .....	8
1.5.1	<i>Dépendances Spring .....</i>	8
1.5.2	<i>Les Java Beans .....</i>	9
1.5.3	<i>La conf XML .....</i>	10
1.5.4	<i>Le programme .....</i>	10
1.6	TIMEMANAGER AVEC JMX ET SPRING .....	10
1.6.1	<i>TimeManager.....</i>	10
1.6.2	<i>TimeManagerMBean .....</i>	10
1.6.3	<i>JConsole.....</i>	11
1.7	AOP (ASPECT ORIENTED PROGRAMMING).....	12
1.7.1	<i>Gestion des logs application - Partie 1 .....</i>	12
1.7.2	<i>Gestion des logs application - Partie 2 .....</i>	12
<b>2</b>	<b>DES COMPOSANTS ELEMENTAIRES .....</b>	<b>13</b>
2.1	EXERCICE 1 : PLUGIN DYNAMIQUE .....	13
2.1.1	<i>Interface : .....</i>	13
2.1.2	<i>Implémentations .....</i>	13
2.1.3	<i>Fichier de configuration.....</i>	14
2.2	EXERCICE 2 : JAVA PROXY .....	14
2.3	EXERCICE 3 : JAVA PROXY CGLIB .....	15
2.4	EXERCICE 4 : PLUGINCLASSLOADER.....	15
2.5	EXERCICE 5 : OSGI HELLOWORLD .....	16
2.5.1	<i>Créer le bundle : .....</i>	16
2.5.2	<i>Exécuter le bundle .....</i>	17
2.6	GERER LES DEPENDANCES.....	19
2.6.1	<i>Créer un service .....</i>	19
2.6.2	<i>Exposer l'interface.....</i>	20
2.6.3	<i>Importer la dépendance .....</i>	20
2.6.4	<i>Enregistrer le service auprès du conteneur .....</i>	20
2.6.5	<i>Obtenir le service auprès du conteneur .....</i>	21
2.6.6	<i>Exécuter.....</i>	21

<b>3</b>	<b>DECOUPLEZ !</b>	<b>23</b>
3.1	EXERCICE 1 : REQUETE POUR WEBSERVICE SOAP AVEC SOAPUI	23
3.2	EXERCICE 2 : CREER UN MOCK DE SERVICE AVEC SOAPUI	24
3.3	EXERCICE 3 : PASSONS AU JAVA - LE SERVEUR	26
3.3.1	<i>Configuration</i>	26
3.3.2	<i>Implémentation</i>	26
3.3.3	<i>Vérification :</i>	27
3.4	EXERCICE 4 : PASSONS AU JAVA - COTE CLIENT	28
3.4.1	<i>Configuration</i>	28
3.4.2	<i>Implémentation</i>	28
3.4.3	<i>Vérification</i>	28
3.4.4	<i>Evolution</i>	29

# 1 Techniques de base

## 1.1 Installer e(fx)clipse plugin Java FX pour Eclipse



## 1.2 Abstract Factory

Ouvrez le projet Java « patterns »

### 1.2.1 Les fabriques

Créez le package : `fr.kc.formation.patterns.afactory.factories`

Dedans, créez l'interface suivante :

```
package fr.kc.formation.patterns.afactory.factories;

import fr.kc.formation.patterns.afactory.model.IFlyingAnimal;
import fr.kc.formation.patterns.afactory.model.IRunningAnimal;
import fr.kc.formation.patterns.afactory.model.ISwimmingAnimal;

public interface IAnimalFactory {

    public IFlyingAnimal createFlyingAnimal();

    public IRunningAnimal createRunningAnimal();

    public ISwimmingAnimal createSwimmingAnimal();

}
```

Et ses 2 implémentations suivantes :

- `DinosaursFactory`
- `MammalsFactory`

### 1.2.2 Le programme

Modifiez le main pour utiliser la fabrique qui sera initialisée au démarrage du programme :

```
public static void main(String... args) {
    Main program = new Main(GeologicalPeriod.QUATERNARY);
    program.start();
}

private GeologicalPeriod period;
private IAnimalFactory factory;

private Main(GeologicalPeriod period) {
    super();
    this.period = period;
}

private void start() {
    this.factory = initAnimalsFactory(period);
    IFlyingAnimal flying = factory.createFlyingAnimal();
    IRunningAnimal running = factory.createRunningAnimal();
    ISwimmingAnimal swimming = factory.createSwimmingAnimal();

    System.out.println("Period: " + period);
    System.out.println();
    display("flying: ", flying);
    display("running: ", running);
    display("swimming: ", swimming);
    System.out.println("Ok.");
}
```

## 1.3 Singleton

### 1.3.1 Un Properties manager :

Créez le package : `fr.kc.formation.patterns.singleton`

Dedans, créez la classe suivante :

```
package fr.kc.formation.patterns.singleton;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class PropertiesManager {

    private static PropertiesManager instance = new PropertiesManager();

    public static PropertiesManager getInstance() {
        return instance;
    }

    private Properties properties;

    private PropertiesManager() {
        properties = new Properties();

        try(InputStream is = getClass().getResourceAsStream("/application.properties")) {
            properties.load(is);
        } catch (IOException e) {
            throw new RuntimeException("Cannot load app properties", e);
        }
    }

    public String getProperty(String key) {
        return properties.getProperty(key);
    }
}
```

### 1.3.2 Le programme

Modifiez le Main pour charger l'époque, à partir du fichier de configuration, en utilisant ce **PropertiesManager** :

```
public static void main(String... args) {
    Main program = new Main();
    program.start();
}

private static GeologicalPeriod loadPeriod() {
    String periodStr = PropertiesManager.getInstance().getProperty("geological.period");
    return GeologicalPeriod.valueOf(periodStr);
}

private Main() {
    this(loadPeriod());
}
```

## 1.4 Abstract Factory + Singleton

### 1.4.1 Une classe abstraite

Créez la classe suivante :

```
package fr.kc.formation.patterns.afactory.factories;

import fr.kc.formation.patterns.afactory.model.GeologicalPeriod;

public abstract class AFactory implements IAnimalFactory {

    private static AFactory instance;

    public static void init(GeologicalPeriod period) {
        if(GeologicalPeriod.QUATERNARY.equals(period)) {
            instance = new MammalsFactory();
        } else {
            instance = new DinosaursFactory();
        }
    }

    public static IAnimalFactory getInstance() {
        if( instance == null ) {
            throw new IllegalStateException("Not initialized! (call init method first)");
        }
        return instance;
    }
}
```

Puis modifier les fabriques de la façon suivante :

```
public class DinosaursFactory extends AFactory {
    //...
```

```
public class MammalsFactory extends AFactory {
    //...
```

### 1.4.2 Le programme

Modifier le main pour utiliser ce nouveau singleton :

```
private void start() {
    AFactory.init(this.period);

    IFlyingAnimal flying = AFactory.getInstance().createFlyingAnimal();
    IRunningAnimal running = AFactory.getInstance().createRunningAnimal();
    ISwimmingAnimal swimming = AFactory.getInstance().createSwimmingAnimal();

    //...
}
```

## 1.5 Spring

### 1.5.1 Dépendances Spring

Ouvrez le projet IOC

Ajoutez ces dépendances dans le « pom.xml »

```
<!-- Shared version number properties -->
<properties>
  <org.springframework.version>3.0.5.RELEASE</org.springframework.version>
</properties>

<dependencies>
  <!-- Core utilities used by other modules. Define this if you use Spring
  Utility APIs (org.springframework.core.* / org.springframework.util.*) -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>

  <!-- Expression Language (depends on spring-core) Define this if you use
  Spring Expression APIs (org.springframework.expression.*) -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>

  <!-- Bean Factory and JavaBeans utilities (depends on spring-core) Define
  this if you use Spring Bean APIs (org.springframework.beans.*) -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>

  <!-- Aspect Oriented Programming (AOP) Framework (depends on spring-core,
  spring-beans) Define this if you use Spring AOP APIs (org.springframework.aop.*) -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>

  <!-- Application Context (depends on spring-core, spring-expression, spring-aop,
  spring-beans) This is the central artifact for Spring's Dependency Injection
  Container and is generally always defined -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>

  <!-- Various Application Context utilities, including EhCache, JavaMail,
  Quartz, and Freemarker integration Define this if you need any of these integrations -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${org.springframework.version}</version>
  </dependency>
</dependencies>
```



### 1.5.2 Les Java Beans

Dans le package *fr.kc.ioc*, créez les 2 POJOs (Plain Ordinary Java Objects) suivants :

```
public class Child {  
  
    private int age;  
  
    public Child(int age) {  
        super();  
        this.age = age;  
    }  
  
    public Child() {  
        this(0);  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return "Child [age=" + age + "]";  
    }  
}
```

```
public class Father {  
  
    private Child child;  
  
    public Father(Child child) {  
        super();  
        this.child = child;  
    }  
  
    public Father() {  
        this(null);  
    }  
  
    public Child getChild() {  
        return child;  
    }  
  
    public void setChild(Child child) {  
        this.child = child;  
    }  
  
    @Override  
    public String toString() {  
        return "Father [child=" + child + "]";  
    }  
}
```

### 1.5.3 La conf XML

src/main/resources/spring/application-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="father" class="fr.kc.ioc.Father" scope="prototype">
        <property name="child" ref="child"></property>
    </bean>

    <bean id="child" class="fr.kc.ioc.Child">
        <property name="age" value="3"></property>
    </bean>

</beans>
```

### 1.5.4 Le programme

```
public class Main {

    public static void main(String[] args) {

        ApplicationContext applicationContext
            = new ClassPathXmlApplicationContext("/spring/application-context.xml");

        Child child = applicationContext.getBean("child", Child.class);
        System.out.println(child);

        Father father = applicationContext.getBean("father", Father.class);
        System.out.println(father);

    }
}
```

## 1.6 TimeManager avec JMX et Spring

### 1.6.1 TimeManager

Dans le Model, lire le code du singleton TimeManager. Que fait ce code ?

Dans Calcul.java, Remplacer les appels à LocalDateTime.now() par  
TimeManager.getInstance().getCurrentDate();

### 1.6.2 TimeManagerMBean

Dans le package : fr.keyconsulting.formation.control.time, créez un MBean chargé d'exposer les opérations du TimeManager en JMX :

```
public class TimeManagerMBean {

    private DateTimeFormatter dateFormatter;

    public TimeManagerMBean() {
        super();
        dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    }

}
```

```
public LocalDateTime getCurrentDate() {
    return TimeManager.getInstance().getCurrentDate();
}

public LocalDateTime getReferenceDate() {
    return TimeManager.getInstance().getReferenceDate();
}

public void setReferenceDate(String referenceDateStr) {
    LocalDateTime referenceDate;
    if( referenceDateStr == null || referenceDateStr.isEmpty() ||
    "null".equalsIgnoreCase(referenceDateStr)) {
        referenceDate = null;
    } else {
        referenceDate = LocalDateTime.parse(referenceDateStr, dateFormatter);
    }
    TimeManager.getInstance().setReferenceDate(referenceDate);
}

public LocalDateTime getStartDate() {
    return TimeManager.getInstance().getStartDate();
}
}
```

Ensuite ajouter la conf dans Spring :

```
<bean id="timeManagerMBean"
class="fr.keyconsulting.formation.control.time.TimeManagerMBean" />

<bean id="exporter" class="org.springframework.jmx.export.MBeanExporter" lazy-
init="false">
    <property name="beans">
        <map>
            <entry key="beans:name=calc.TimeManager" value-ref="timeManagerMBean"/>
        </map>
    </property>
</bean>
```

### 1.6.3 JConsole

Lancez l'application, il doit y avoir les lignes suivantes dans la console :

```
...
août 31, 2015 5:55:06 PM org.springframework.jmx.export.MBeanExporter afterPropertiesSet
INFOS: Registering beans for JMX exposure on startup
août 31, 2015 5:55:06 PM org.springframework.jmx.export.MBeanExporter registerBeanInstance
INFOS: Located managed bean 'beans:name=calc.TimeManager': registering with JMX server as MBean
[beans:name=calc.TimeManager]
```

et connectez vous avec l'utilitaire : java/bin/jconsole.exe

## 1.7 AOP (Aspect Oriented Programming)

### 1.7.1 Gestion des logs application - Partie 1

Dans le projet Maven **java-aop**,

Dans la classe, Service ajouter (à la main) des logs au début et à la fin de chaque méthode (avec l'heure).

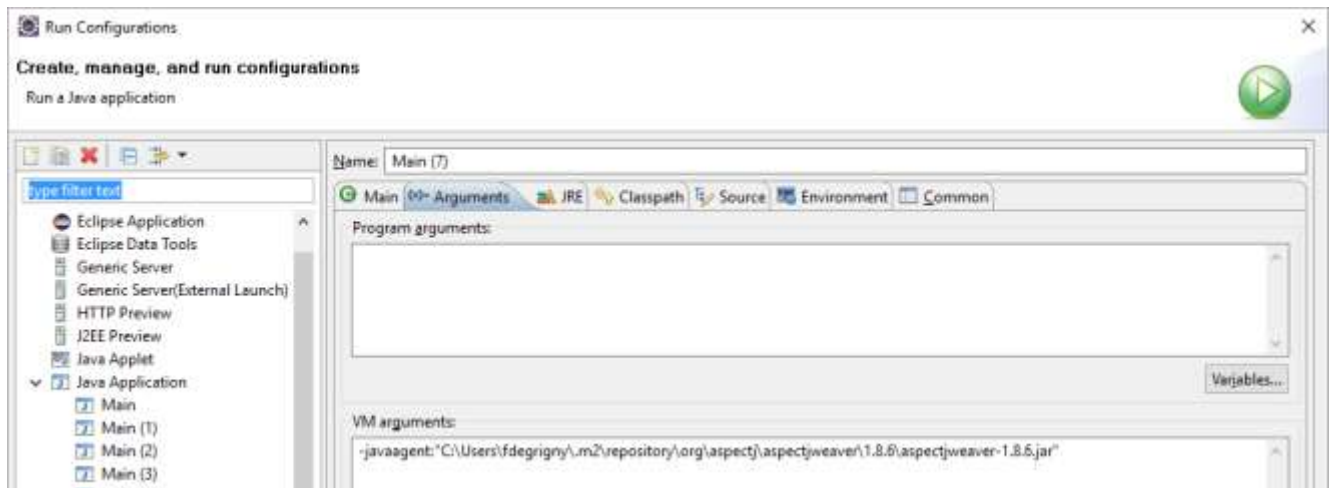
### 1.7.2 Gestion des logs application - Partie 2

Nous allons effectuer la tâche précédente en utilisant l'AOP.

Regarder la classe le LogAspect. Que fait-elle (*Indications: chercher **aspectj-cheat-sheet** sur google*) ?

Lancer le Main en ajoutant, le tisseur AjaspectJ dans les arguments de la JVM :

```
-javaagent:"<path_to_maven_respository>\org\aspectj\aspectjweaver\1.8.6\aspectjweaver-1.8.6.jar"
```



## 2 Des composants élémentaires

### 2.1 Exercice 1 : Plugin Dynamique

Importer le projet Maven : java-dynamic

#### 2.1.1 Interface :

Créez l'interface suivante :

```
package fr.keyconsulting.formation.plugins;

public interface IPlugin {

    public default String getName() {
        return getClass().getSimpleName();
    }

    public default void doSomething() {
        System.out.println("I'm the " + getName());
    }
}
```

#### 2.1.2 Implémentations

Et 2 implémentations triviales dans le package : fr.keyconsulting.formation.plugins.impl

```
package fr.keyconsulting.formation.plugins.impl;

import fr.keyconsulting.formation.plugins.IPlugin;

public class PluginOne implements IPlugin {

}
```

```
package fr.keyconsulting.formation.plugins.impl;

import fr.keyconsulting.formation.plugins.IPlugin;

public class PluginTwo implements IPlugin {

}
```

Créez le Main qui va avec. Remarque : utilisez Class.forName(...) et Class.newInstance()

```
public static void main(String... args) throws Exception {

    Class<?> clazz = Class.forName("fr.keyconsulting.formation.plugins.impl.PluginTwo");
    IPlugin plugin = (IPlugin) clazz.newInstance();

    System.out.println(plugin.getName() + ": ");
    plugin.doSomething();
    System.out.println("Ok.");
}
```

### 2.1.3 Fichier de configuration

Spécifiez le nom de l'implémentation à utiliser dans un fichier de properties.

```
public static void main2(String... args) throws Exception {
    Properties appProps = new Properties();
    try( InputStream is = Main.class.getResourceAsStream("/application.properties") ) {
        appProps.load(is);
    }

    String implClassName = appProps.getProperty("plugin.impl.class");

    Class<?> clazz = Class.forName(implClassName);
    IPlugin plugin = (IPlugin) clazz.newInstance();

    //...
}
```

src/main/resources/application.properties :

```
# application.properties

plugin.impl.class=fr.keyconsulting.formation.plugins.impl.PluginTwo
```

## 2.2 Exercice 2 : Java Proxy

Créer une classe : `fr.keyconsulting.formation.plugins.proxy.LogInvocationHandler`

```
public class LogInvocationHandler implements InvocationHandler {

    private IPlugin plugin;

    public LogInvocationHandler(IPlugin plugin) {
        super();
        this.plugin = plugin;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        final String className = plugin.getClass().getSimpleName();
        System.out.println("> method '" + method.getName() + "' called on '" + className
+ "' with args: " + args);
        return method.invoke(plugin, args);
    }
}
```

Modifier le main :

```
public static void main(String... args) throws Exception {
    //...
    IPlugin plugin = (IPlugin) clazz.newInstance();
    IPlugin proxy = getProxy(plugin);

    //...
}
```

```
private static IPlugin getProxy(IPlugin plugin) {
    Class<?>[] interfaces = new Class<?>[] {IPlugin.class};
    return (IPlugin) Proxy.newProxyInstance(IPlugin.class.getClassLoader(), interfaces, new
    LogInvocationHandler(plugin));
}
```

## 2.3 Exercice 3 : Java Proxy CGLIB

Créez la class : `fr.keyconsulting.formation.plugins.proxy.CglibInterceptor`

```
public class CglibInterceptor implements MethodInterceptor {

    private Object realObject;

    public CglibInterceptor(Object realObject) {
        super();
        this.realObject = realObject;
    }

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy
    methodProxy) throws Throwable {
        final String className = realObject.getClass().getSimpleName();
        System.out.println("> method '" + method.getName() + "' called on '" + className
        + "' with args: " + args);
        return method.invoke(realObject, args);
    }
}
```

Et modifiez le main :

```
public static void main(String... args) throws Exception {
    //...
    IPlugin plugin = (IPlugin) clazz.newInstance();

    IPlugin proxy = getCglibProxy(plugin);
    //...
}

private static IPlugin getCglibProxy(Object obj) {
    Enhancer e = new Enhancer();
    e.setSuperclass(IPlugin.class);
    e.setCallback(new CglibInterceptor(obj));
    return (IPlugin) e.create();
}
```

## 2.4 Exercice 4 : PluginClassLoader

Compléter la classe : `fr.keyconsulting.formation.plugins.dynamic.PluginClassLoader` en utilisant la méthode `loadClassFromFile`.

Exécutez le main.

## 2.5 Exercice 5 : OSGi HelloWorld

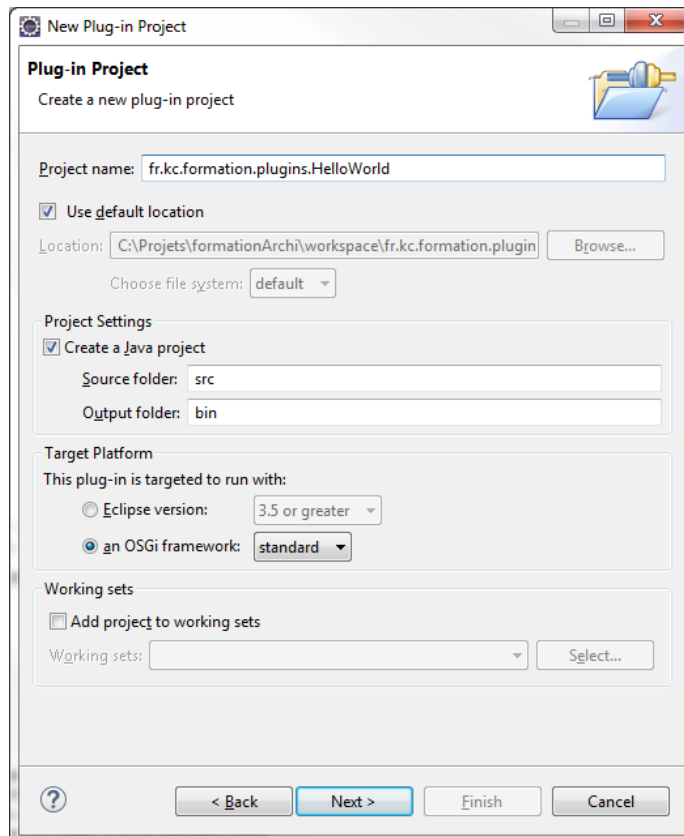
### 2.5.1 Créer le bundle :

Dans Eclipse, faire **File > New > Project**, Sélectionnez Projet Type = **Plug-in Project**

Entrez les valeurs suivantes :

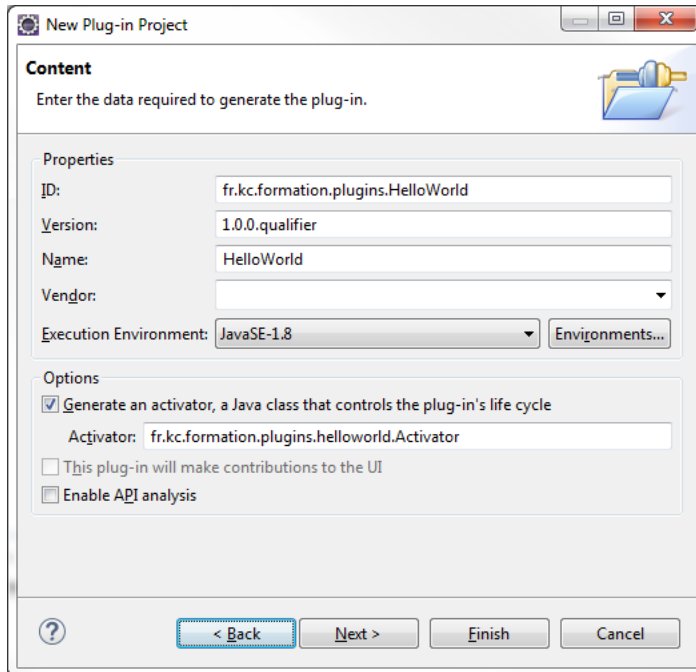
Project Name: fr.kc.formation.plugins.HelloWorld

Target Platform: OSGi framework => Standard

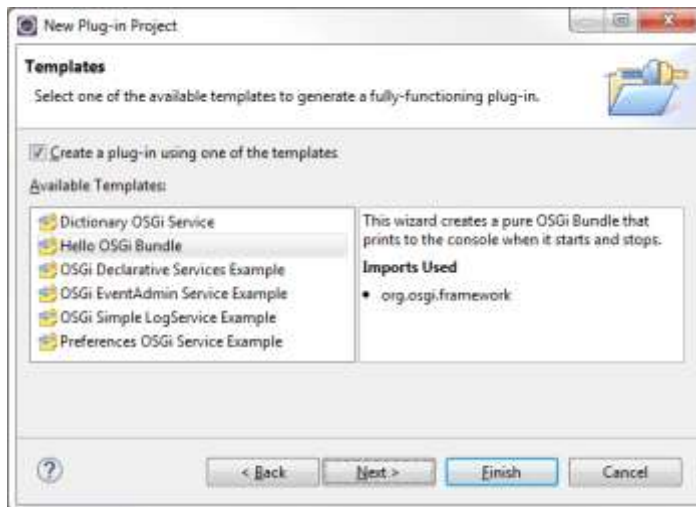


Faire : **Next**





Next encore



Sélectionnez le template « **Hello OSGi Bundle** » et **Finish**

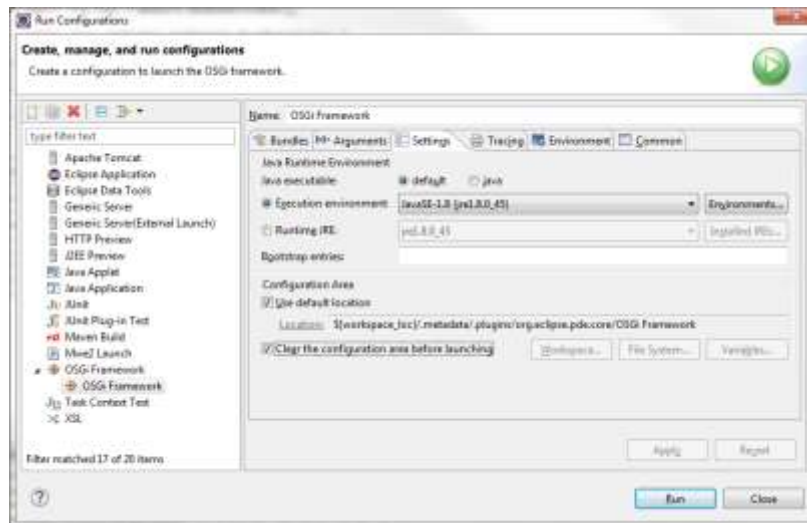
Eclipse va générer : Activator.java et MANIFEST.MF, inspectez les.

### 2.5.2 Exécuter le bundle

Faire : **Run > Run Configurations...**

Créez une nouvelle configuration de type **OSGi Framework**

Dans l'onglet **Settings**, cochez « Clear the configuration area before launching »

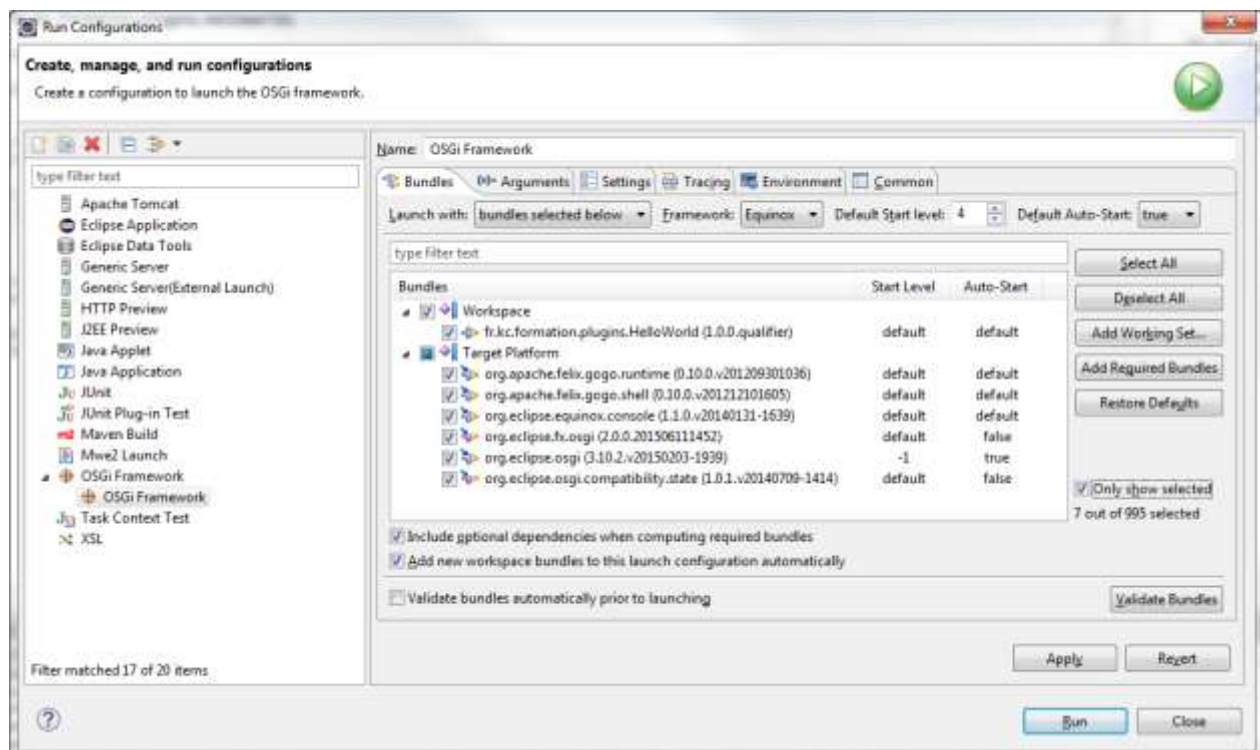


Dans l'onglet « **Bundles** », Faire **Deselect All**, Cochez **Workspace**, puis Faire **Add required Bundles**

Les modules suivants doivent être présents, ajoutez-les si nécessaires :

- org.eclipse.osgi
- org.eclipse.equinox.console
- org.apache.felix.gogo.runtime
- org.apache.felix.gogo.shell

Cochez **Only show selected** pour verifier.



Faire **Run**

Eclipse lance la console OSGi

```

<terminated> OSGi Framework [OSGi Framework] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (21 août 2015 16:25:41)
Hello World!!
osgi> ss
"Framework is launched."

id      State      Bundle
0       ACTIVE      org.eclipse.osgi_3.10.2.v20150203-1939
          Fragments=3, 5
1       ACTIVE      fr.kc.formation.plugins.HelloWorld_1.0.0.qualifier
2       ACTIVE      org.apache.felix.gogo.runtime_0.10.0.v201209301036
3       RESOLVED     org.eclipse.fx.osgi_2.0.0.201506111452
          Master=0
4       ACTIVE      org.eclipse.equinox.console_1.1.0.v20140131-1639
5       RESOLVED     org.eclipse.osgi.compatibility.state_1.0.1.v20140709-1414
          Master=0
6       ACTIVE      org.apache.felix.gogo.shell_0.10.0.v201212101605
osgi> stop 1
Goodbye World!!
osgi> start 1
Hello World!!
osgi> exit
Really want to stop Equinox? (y/n; default=y) y
  
```

Les principales commandes sont les suivantes :

ss	affiche la liste de bundles avec leur état
start <bundle_id>	démarre un bundle
stop <bundle_id>	arrête un bundle
update <bundle_id>	met à jour un bundle à partir d'un nouveau JAR
install <bundle_url>	installe un nouveau bundle dans le conteneur OSGi
uninstall <bundle_id>	désinstalle un bundle du conteneur OSGi

## 2.6 Gérer les dépendances

### 2.6.1 Créer un service

On veut créer un nouveau bundle **fr.kc.formation.plugins.HelloService** qui va exposer une interface que l'on va utiliser dans notre premier bundle **fr.kc.formation.plugins.HelloWorld**.

- 1) Créer ce nouveau bundle de la même façon que le premier.
- 2) Créer dans ses sources l'interface suivante :

```

package fr.kc.formation.plugins.service;

public interface MessageService {

    public String sayHello();
}
  
```

- 1) Ainsi que son implémentation :

```
package fr.kc.formation.plugins.service.impl;

import fr.kc.formation.plugins.service.HelloService;

public class MessageServiceImpl implements HelloService {

    @Override
    public String sayHello() {
        System.out.println("Inside MessageServiceImpl.sayHello()");
        return "Say Hello";
    }
}
```

### 2.6.2 Exposer l'interface

Ouvrir le fichier **MANIFEST.MF**, dans l'onglet « **Runtime** », ajouter **fr.kc.formation.plugins.service** dans **Exported Packages**.

Dans le fichier Manifest, on doit avoir la ligne suivante :

```
Export-Package: fr.kc.formation.plugins.service
```

### 2.6.3 Importer la dépendance

Dans l'autre bundle HelloWorld, importer le ce package :

Ouvrir le fichier **MANIFEST.MF**, dans l'onglet « **Dependancies** », ajouter **fr.kc.formation.plugins.service** dans **Imported Packages**.

Dans le fichier Manifest, on doit avoir la ligne suivante :

```
Import-Package: fr.kc.formation.plugins.service, org.osgi.framework;version="1.3.0"
```

### 2.6.4 Enregistrer le service auprès du conteneur

Maintenant, on peut déclarer l'interface « **MessageService** » dans le bundle **HelloWorld**, mais on ne peut pas en faire grand-chose, car l'implémentation est invisible à l'extérieur du bundle de service.

Il faut déclarer ce service auprès du contexte OSGi.

Dans le bundle de Service, créez l'Activator suivant :

```
package fr.kc.formation.plugins.helloworldservice;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;

import fr.kc.formation.plugins.service.MessageService;
```

```
import fr.kc.formation.plugins.service.impl.HelloServiceImpl;

public class Activator implements BundleActivator {

    ServiceRegistration<?> serviceRegistration;

    public void start(BundleContext context) throws Exception {
        System.out.println("Hello Service Bundle starts");
        MessageService helloService = new HelloServiceImpl();
        serviceRegistration = context.registerService(MessageService.class.getName(),
helloService, null);
    }

    public void stop(BundleContext context) throws Exception {
        serviceRegistration.unregister();
        System.out.println("Hello Service Bundle stopped.");
    }
}
```

### 2.6.5 Obtenir le service auprès du conteneur

Dans le bundle HelloWorld, modifier l'Activator de la façon suivante :

```
public class Activator implements BundleActivator {

    ServiceReference<MessageService> helloServiceRef;

    public void start(BundleContext context) throws Exception {
        System.out.println("Hello World!!");
        helloServiceRef = context.getServiceReference(MessageService.class);
        MessageService helloService = context.getService(helloServiceRef);
        System.out.println("Message: " + helloService.sayHello());
    }

    public void stop(BundleContext context) throws Exception {
        context.ungetService(helloServiceRef);
        System.out.println("Goodbye World!!");
    }
}
```

### 2.6.6 Exécuter

Bien veiller à ce que les 2 plugins soient cochés dans la configuration.

```
Hello World!!
Hello Service Bundle starts
Inside MessageServiceImpl.sayHello()
Message: Say Hello
osgi> ss
```

"Framework is launched."

id	State	Bundle
0	ACTIVE	org.eclipse.osgi_3.10.2.v20150203-1939 Fragments=6, 3
1	ACTIVE	fr.kc.formation.plugins.HelloWorld_1.0.0.qualifier
2	ACTIVE	org.apache.felix.gogo.runtime_0.10.0.v201209301036
3	RESOLVED	org.eclipse.fx.osgi_2.0.0.201506111452 Master=0
4	ACTIVE	org.eclipse.equinox.console_1.1.0.v20140131-1639
5	ACTIVE	fr.kc.formation.plugins.HelloWorldService_1.0.0.qualifier
6	RESOLVED	org.eclipse.osgi.compatibility.state_1.0.1.v20140709-1414 Master=0
7	ACTIVE	org.apache.felix.gogo.shell_0.10.0.v201212101605

osgi>

## 3 Découpez !

### 3.1 Exercice 1 : Requête pour Webservice Soap avec SoapUI

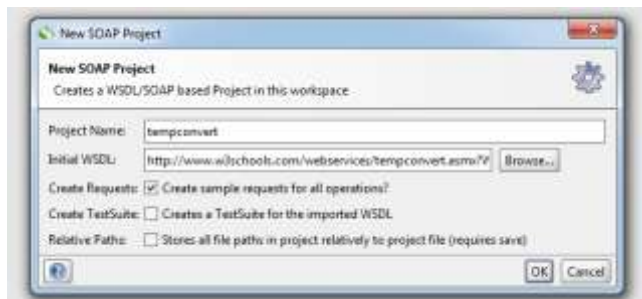
Télécharger et installer SoapUI : <http://www.soapui.org/>

Créer un nouveau projet Soap.

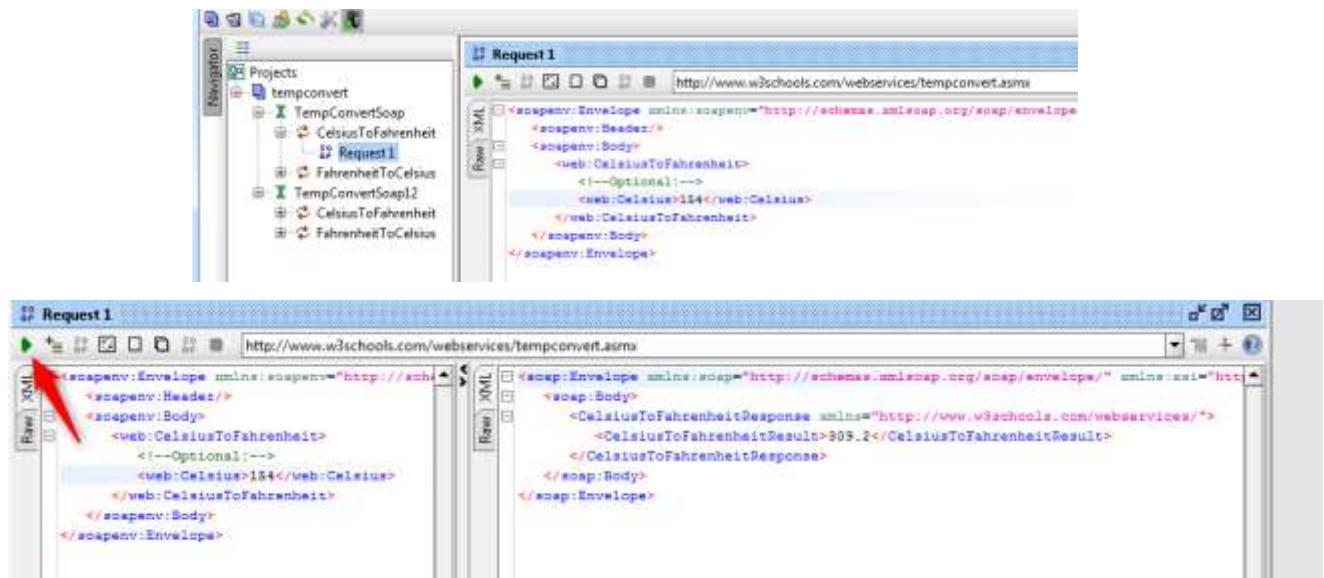


Importer la description d'un Webservice présent online :

<http://www.webservices.net/globalweather.asmx?wsdl>



Effectuer une requête et vérifier le résultat obtenu :



## 3.2 Exercice 2 : Créer un mock de service avec SoapUI

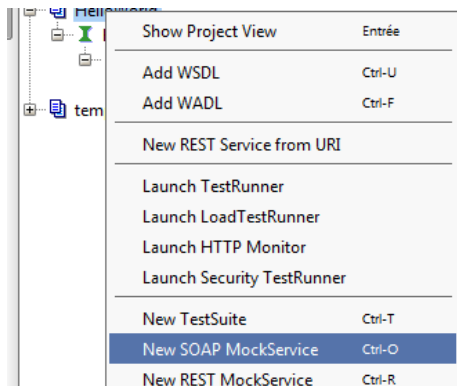
Créer un nouveau projet Soap.

Importer la WSDL : HelloWorld.wsdl présente sur le github dans Tools.

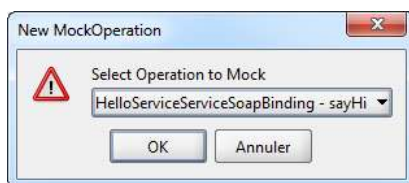
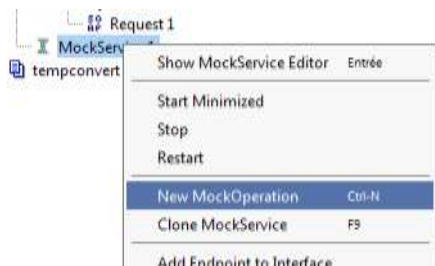
Nous allons créer un bouchon (mock) de ce service.

Pour cela créer un projet SoapUI en important cette WSDL :

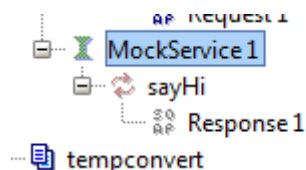
⇒ Créer un nouveau MockService



⇒ Ajouter une mock opération :

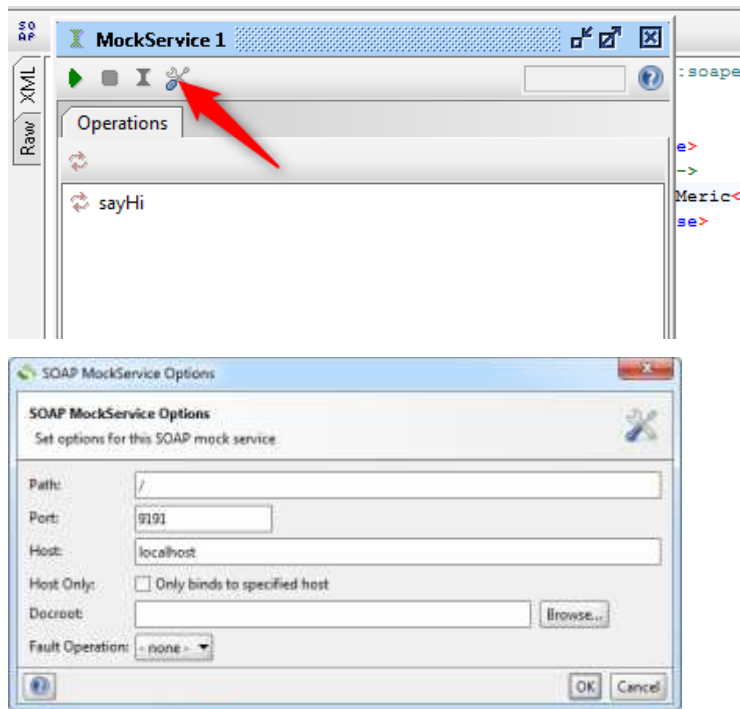


⇒ Double cliquer sur MockService1



⇒ Modifier les propriétés sur service





- ⇒ Le lancer (flèche verte).
- ⇒ Effectuer une requête pour vérifier que le mock est UP :  
<http://localhost:9191/webService/services/HelloWorld>
- ⇒ Vérification : effectuer une requête avec le même soapUI sur ce web service.

### 3.3 Exercice 3 : Passons au Java - le serveur

Nous allons créer un serveur web proposant le service HelloWorld. Pour cela nous utiliserons la librairie apache CXF avec le framework Spring. Dans un premier temps, nous allons implémenter le serveur. Le projet java-ws permet de démarrer un serveur web (tomcat) configuré pour utiliser CXF et spring. Le plugin cargo de maven afin de démarrer un tomcat embarqué dans notre projet.

#### 3.3.1 Configuration

Le fichier *Spring-context.xml* décrit les différents composants gérés par spring.

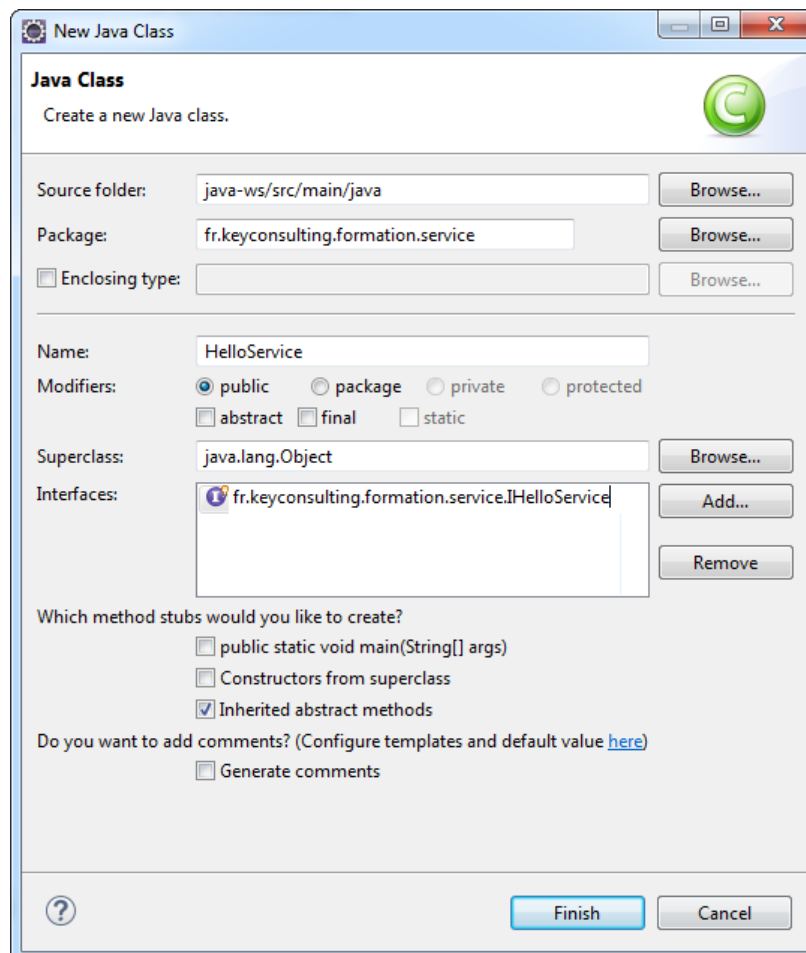
⇒ Ajouter la ligne suivante dans ce fichier :

```
<jaxws:endpoint id="helloWorld" implementor="fr.keyconsulting.formation.service.HelloService"
address="/HelloWorld"/>
```

#### 3.3.2 Implémentation

Nous allons donc devoir créer l'implémentation du service : HelloService.

⇒ Implémenter l'interface IHelloService disponible dans le projet java-ws-interface.



Le WebService condient une méthode sayHi qui doit renvoyer « Hello » + l'argument reçu + « ! ».

⇒ Ecrire l'implémentation de la classe HelloService.

```
• package fr.keyconsulting.formation.service;
•
• public class HelloService implements IHelloService {
•
•     @Override
•     public String sayHi(String name) {
•         //implémenter la méthode
•     }
•
• }
```

### 3.3.3 Vérification :

Lancer le serveur. Pour cela compiler le projet :

⇒ mvn clean install

puis, se placer dans le projet java-ws, et lancer via le plugin cargo :

⇒ cd java-ws

⇒ mvn cargo:run

Une fois le serveur démarré, la liste des services qu'il propose est disponible ici :

<http://localhost:8080/webservice/services>

Vérifier le bon fonctionnement du WebService avec SoapUI en important la WSDL générée à l'url :

<http://localhost:8080/webservice/services/HelloWorld?wsdl>.

### 3.4 Exercice 4 : Passons au Java - côté client

Le projet java-calculatrice-client-ws est préconfiguré pour pouvoir créer un client WebService. Le fichier de configuration de Spring-ws-client.xml décrit les clients WebService qui seront rendu disponible grâce à Spring.

#### 3.4.1 Configuration

⇒ Ajouter la ligne suivante au fichier Spring-ws-client.xml :

```
<jaxws:client id="helloClient" serviceClass="fr.keyconsulting.formation.service.IHelloService"
address="http://localhost:8080/webService/services/HelloWorld" />
```

#### 3.4.2 Implémentation

Dans la classe Controller nous allons alors pouvoir récupérer le service.

⇒ Ajouter cette ligne au début de l'initialisation (méthode initialize):

```
service = (IHelloService) context.getBean("helloClient");
```

⇒ Et l'utiliser pour renseigner le texte en haut de la vue de la calculatrice (fin de la méthode run) :

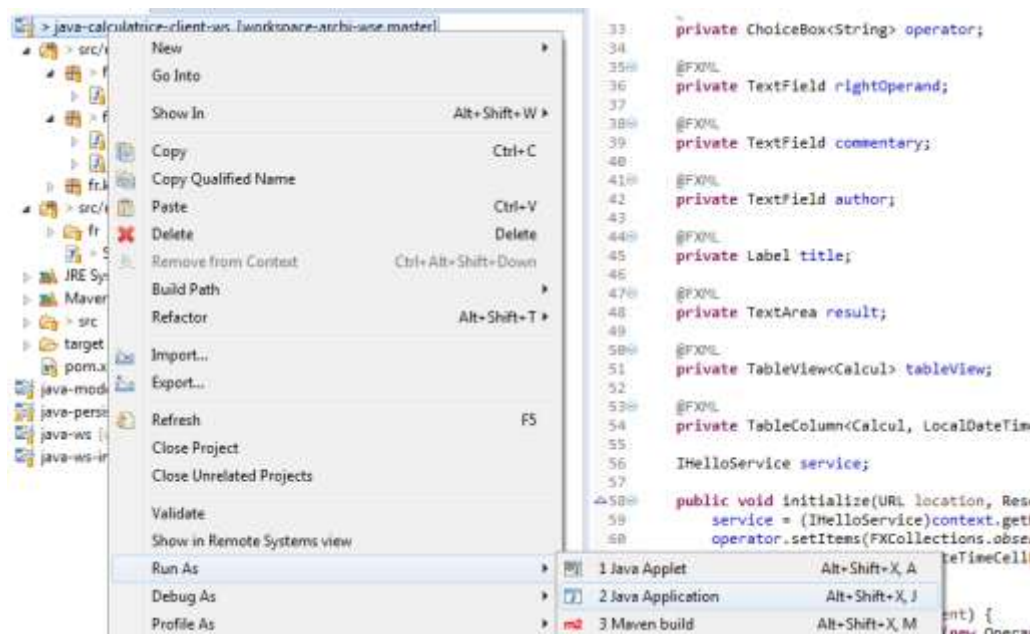
```
title.setText(service.sayHi("inconnu"));
```

#### 3.4.3 Vérification

Relancer le webservice (serveur) s'il a été arrêté :

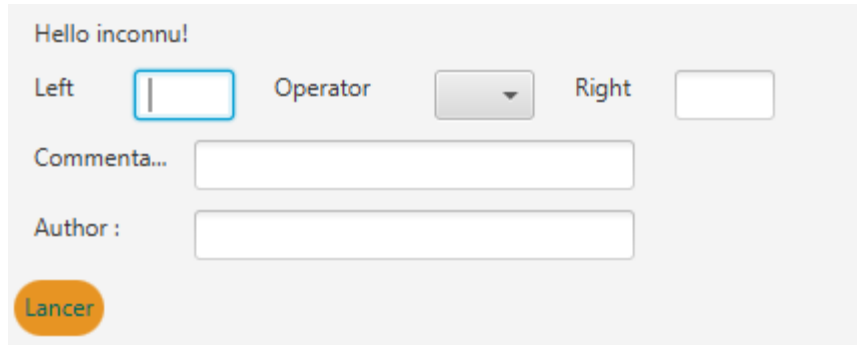
⇒ mvn cargo:run dans le répertoire java-ws.

Lancer l'application et vérifier le fonctionnement.



Si une classe main est demandée, elle se trouve ici : `fr.keyconsulting.formation.Main`

Résultat :



### 3.4.4 Evolution

Modifier la classe Controller afin de changer le message « title » lorsque un author est renseigné, il doit alors afficher l'auteur utilisé lors du dernier calcul effectué. Puis recompiler et relancer.

Résultat :

