**Travaux Pratiques**

1)      **Reconfiguration du serveur**

Les noms de services dans le fichier de configuration (*basic\_deployment\_descriptor*), ainsi que leur ordre, déterminent les services d’interception utilisés par le container.

Note : la construction «**//** » peut être utilisée pour commenter le nom d’un service – le container ignorera les noms de services commentés.

a)      Enlevez un service d’interception (Ex : Logging)

b)      Changez l’ordre des services dans la chaîne d’interception (Ex : Security, Logging, Monitoring)

Qu’elle est la différence, du côté serveur, lors de l’appel d’une méthode sur le GerantMagasin ? (Note : Pour que les modifications soient prises en compte, il faut redémarrer le Server)

2)      **Usage du service de sécurité**

a)      Enlevez l’authentification du client - dans la classe ClientMagasin :

i)        Soit : supprimez l’appel à la méthode qui permet au client de s’authentifier auprès du serveur - **//**obj.setSecurityCredentials("login");

ii)       Soit : passez null en paramètre de cette méthode obj.setSecurityCredentials(**null**);

Note : pour que les modifications soient prises en compte, il faut recompiler la classe Client : javac –d classes Client.java

b)      Modifiez le service de sécurité de façon à ce qu’il authentifie seulement les clients avec le login « toto »

Note : assurez-vous que le service de sécurité soit en place du côté serveur

Quelle est la différence, du côté serveur, lors de l’appel d’une méthode sur le GerantMagasin ?

Quelle est la réponse obtenue par le client ?

**3)      Implantation des services d’interception**

Etendez la fonctionnalité des services d’interception :

a)      Monitoring : doit calculer le temps d’exécution de chaque méthode et le mettre dans le Context courant du container

b)      Logging : pour chaque appel de méthode intercepté, doit écrire dans un fichier (ou à la ligne de commande) les informations suivantes : La date et l’heure de l’appel de méthode, le nom de la méthode, la valeur du paramètre et le temps d’exécution de la méthode

Notes :

-          La classe java.util.GregorianCalendar peut être utilisée pour toute activité liée à la date et/ou l’heure courante :

o   Pour obtenir une estampille, utilisez : GregorianCalendar calendar = new GregorianCalendar() - création d’un objet de type GregorianCalendar et son initialisation de façon à représenter la date et l’heure courante

o   Pour obtenir le nombre de millisecondes d’une date (nombre de millisecondes depuis le premier Janvier 1970 GMT), utilisez : calendar.getTimeInMillis();

o   Pour obtenir la valeur d’un champ donné d’un calendrier, utilisez : calendar.get(<nom du champ>);

Exemple : pour obtenir le jour d’un calendrier : calendar.get(Calendar.DAY\_OF\_MONTH); Autres champs possibles : Calendar.*MONTH*, *YEAR*, *HOUR\_OF\_DAY*, *MINUTE*, *SECOND*, …

o   Pour plus d’informations, consultez l’API de java : <https://docs.oracle.com/javase/7/docs/api/>

-          Ainsi, il est possible d’obtenir le temps d’exécution d’une méthode, en prenant une première estampille au moment de l’interception de l’appel de méthode (pre-invoke : methodCalled\_callback) et une deuxième estampille au moment de l’interception du retour de méthode (post-invoke : methodReturned\_callback).

-          Le temps d’exécution calculé par l’intercepteur de supervision (MonitoringInterceptor) peut être transmis à l’intercepteur de journalisation (LoggingInterceptor) via l’objet Context (transmis en paramètre à toutes les méthodes d’interception)

**4)      Extension de la chaîne d’interception**

Implantez un nouveau service d’interception (ex : Transaction) et insérez-le dans la chaîne d’interception.

a)      Implantez un nouvel intercepteur spécifique (ex : nouvelle classe – TransactionInterceptor correspondant au nouveau serviceType – Transaction\_Service) ; la nouvelle classe doit implanter l’interface InterceptorService

b)      Modifiez la classe InterceptorProxy de façon à ce qu’elle puisse reconnaître le nouveau type de service (Obs : il faut étendre le constructeur de la classe de façon à ce qu’il crée une instance du nouvel Intercepteur – TransactionInterceptor, quand il reçoit le nom du nouveau service en paramètre - Transaction\_Service)

c)       Ajoutez le nouveau service dans le fichier de configuration - Transaction\_Service

javac -d ../classes \*.java shop/\*.java container/\*.java

cp java.policy ../classes

cp basic\_deployment\_descriptor ../classes

cp Mag\* ../classes

java -Djava.security.policy=java.policy Server 53000 Mag1

java -Djava.security.policy=java.policy ClientMagasin 53000 sel C125-09

**1)**

original:

Configuring Container..

Container Service 0 : Logging\_Service

Container Service 1 : Security\_Service

Container Service 2 : Monitoring\_Service

GerantMagasin enregistre : //c124-02:53000/GerantMagasin

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: setSecurityCredentials

> name: method\_parameters; value: login=login

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Logging Service pre-invoked..

Security Service pre-invoked..

Successful security check on login: login

Monitoring Service pre-invoked..

Monitoring Service post-invoked..

Security Service post-invoked..

Logging Service post-invoked..

a) Le Service commenté a disparu.

Configuring Container..

Container Service 0 : Security\_Service

Container Service 1 : Monitoring\_Service

GerantMagasin enregistre : //c124-02:53000/GerantMagasin

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: setSecurityCredentials

> name: method\_parameters; value: login=login

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Security Service pre-invoked..

Successful security check on login: login

Monitoring Service pre-invoked..

Monitoring Service post-invoked..

Security Service post-invoked..

b) l'ordre de l'éxcution de serveur a changé aussi.

Configuring Container..

Container Service 0 : Security\_Service

Container Service 1 : Logging\_Service

Container Service 2 : Monitoring\_Service

GerantMagasin enregistre : //c124-02:53000/GerantMagasin

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: setSecurityCredentials

> name: method\_parameters; value: login=login

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Security Service pre-invoked..

Successful security check on login: login

Logging Service pre-invoked..

Monitoring Service pre-invoked..

Monitoring Service post-invoked..

Logging Service post-invoked..

Security Service post-invoked..

**2)**

a)

GerantMagasin enregistre : //c124-02:53000/GerantMagasin

Context parameters:

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Logging Service pre-invoked..

Security Service pre-invoked..

Context could not find a parameter named: security\_parameter; will return null..

!!WARNING: security check failed on login: null

Logging Service post-invoked..

<<C124-02 : Le prix minimum de sel est -1.0>> en fait ça dois etre 1.0

Ce -1.0 est dans le fichier GerantMagasin.java, function getIngredientPrice()

b)

Context parameters:

> name: security\_parameter; value: toto

> name: method\_name; value: setSecurityCredentials

> name: method\_parameters; value: login=toto

Context parameters:

> name: security\_parameter; value: toto

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Logging Service pre-invoked..

Security Service pre-invoked..

Successful security check on login: toto

Monitoring Service pre-invoked..

Monitoring Service post-invoked..

Security Service post-invoked..

Logging Service post-invoked..

<<C124-02 : Le prix minimum de sel est 1.0>> correcte

**3)**

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: setSecurityCredentials

> name: method\_parameters; value: login=login

Context parameters:

> name: security\_parameter; value: login

> name: method\_name; value: getIngredientPrice

> name: method\_parameters; value: ingredient=sel

Logging Service pre-invoked..

The method is called in 13h 26/10.

{security\_parameter=login, method\_name=getIngredientPrice, method\_parameters=ingredient=sel}

Security Service pre-invoked..

Successful security check on login: login

Monitoring Service pre-invoked..

Monitoring Service post-invoked..

Execution time of monitoring is 36451 ns.

Security Service post-invoked..

Logging Service post-invoked..

The execution time of logging is 262104 ns.

MonitoringInterceptor.java

package container;

public class MonitoringInterceptor implements InterceptorService {

private long startTime;

private long endTime;

@Override

public boolean methodCalled\_callback(Context ctx) {

System.out.println("Monitoring Service pre-invoked.. ");

startTime = System.nanoTime();

return true;

//TODO

}

@Override

public boolean methodReturned\_callback(Context ctx) {

System.out.println("Monitoring Service post-invoked.. ");

endTime = System.nanoTime();

System.out.println("Execution time of monitoring is "+(endTime-startTime)+" ns.");

return true;

//TODO

}

}

LoggingInterceptor.java

package container;

import java.util.GregorianCalendar;

public class LoggingInterceptor implements InterceptorService {

private long startTime;

private long endTime;

private GregorianCalendar calendar;

@Override

public boolean methodCalled\_callback(Context ctx) {

System.out.println("Logging Service pre-invoked.. ");

calendar = new GregorianCalendar();

System.out.println("The method is called in " + calendar.get(calendar.HOUR\_OF\_DAY)+"h "+ calendar.get(calendar.DAY\_OF\_MONTH)+"/"+calendar.get(calendar.MONTH)+"." );

System.out.println("The parameters of this method are " + ctx.getParams());

startTime = System.nanoTime();

return true;

//TODO

}

@Override

public boolean methodReturned\_callback(Context ctx) {

System.out.println("Logging Service post-invoked.. ");

endTime = System.nanoTime();

System.out.println("The execution time of logging is "+(endTime-startTime)+" ns.");

return true;

//TODO

}

}

**4)**

Logging Service pre-invoked..

The method is called in 14h 26/10.

The parameters of this method are {security\_parameter=login, method\_name=getIngredientPrice, method\_parameters=ingredient=sel}

Security Service pre-invoked..

Successful security check on login: login

Monitoring Service pre-invoked..

Transaction Service pre-invoked..

Transaction Service post-invoked..

Monitoring Service post-invoked..

Execution time of monitoring is 55884 ns.

Security Service post-invoked..

Logging Service post-invoked..

The execution time of logging is 198584 ns.

InterceptorService.java

package container;

public interface InterceptorService {

public static final String LOGGING\_SERVICE = "Logging\_Service";

public static final String SECURITY\_SERVICE = "Security\_Service";

public static final String MONITORING\_SERVICE = "Monitoring\_Service";

public static final String TRANSACTION\_SERVICE = "Transaction\_Service";

public boolean methodCalled\_callback(Context ctx);

public boolean methodReturned\_callback(Context ctx);

}

TransactionInterceptor.java

package container;

public class TransactionInterceptor implements InterceptorService {

@Override

public boolean methodCalled\_callback(Context ctx) {

System.out.println("Transaction Service pre-invoked.. ");

return true;

//TODO

}

@Override

public boolean methodReturned\_callback(Context ctx) {

System.out.println("Transaction Service post-invoked.. ");

return true;

//TODO

}

}

InterceptorProxy.java

package container;

import java.rmi.RemoteException;

import shop.Magasin;

/\*\*

\* @author ada

\* A Proxy for a Magasin object

\* Represents a link in the intercepting chain

\* Intercepts Magasin method calls:

\* - notifies an associated Interceptor service

\* - forwards the method call to the Magasin instance / next Proxy in the chain

\* Intercepts Magasin method call responses:

\* - notifies an associated Interceptor service

\* - returns the result (to the initial method caller)

\*/

public class InterceptorProxy implements Magasin{

/\*\*

\* The container that manages the Magasin instance and the associated intercepting chain

\*/

private Container container = null;

/\*\*

\* Indicates the type of concrete Interceptor associated to this Proxy

\* (e.g. Security, Logging or Monitoring)

\*/

private String serviceType = null;

/\*\*

\* the next Proxy in the intercepting chain

\* this Proxy will forward incoming Magasin method calls to the next Proxy

\* (if this Proxy is the last one in the intercepting chain, then the nextProxy will be the Magasin object)

\*/

private Magasin nextProxy = null;

/\*\*

\* The Interceptor service associated to this Proxy

\* The Proxy notifies this Interceptor service whenever it receives a Magasin method call or response

\*/

private InterceptorService myInterceptor = null;

/\*\*

\* constructor

\* @param container: the container that this Proxy will be part of

\* @param serviceType: the type of Interceptor service associated to this Proxy

\*/

public InterceptorProxy(Container container, String serviceType){

//set container

this.container = container;

//instantiate specific Interceptor service, depending on the given serviceType

if(InterceptorService.SECURITY\_SERVICE.equals(serviceType)){

this.myInterceptor = new SecurityInterceptor();

}

else if(InterceptorService.LOGGING\_SERVICE.equals(serviceType)){

this.myInterceptor = new LoggingInterceptor();

}

else if(InterceptorService.MONITORING\_SERVICE.equals(serviceType)){

this.myInterceptor = new MonitoringInterceptor();

}

else if(InterceptorService.TRANSACTION\_SERVICE.equals(serviceType)){

this.myInterceptor = new TransactionInterceptor();

}

else{

System.out.println("WARNING!! unknown service type: " + serviceType );

}

//set the serviceType

this.serviceType = serviceType;

}

/////////////////////////////

//Magasin Interface methods//

/////////////////////////////

public void setSecurityCredentials( String login ){}

public float getIngredientPrice(String ingredient) throws RemoteException{

float result = -1;

//determines whether or not this Proxy will forward the incoming method call down the intercepting chain

boolean doContinue;

//get the context object from the container

Context context = this.container.getContext();

//preinvoke the associated Interceptor service

if( null!=this.myInterceptor ){

doContinue = this.myInterceptor.methodCalled\_callback(context);

}

else{

System.out.println("WARNING!! NULL Interceptor Service for Proxy: " + serviceType );

doContinue = false;

}

//determine the pre-invocation outcome; return if unsuccessful

if( !doContinue ){

return result;

}

//else (true == doContinue)

//invoke

result = this.nextProxy.getIngredientPrice(ingredient);

//post-invoke the associated Interceptor service

if( null!=this.myInterceptor ){

this.myInterceptor.methodReturned\_callback(context);

}

else{

System.out.println("WARNING!! NULL Interceptor Service for Proxy: " + serviceType );

}

//return the invocation result

return result;

}

public void setMyInterceptor(InterceptorService myInterceptor) {

this.myInterceptor = myInterceptor;

}

public void setNextProxy(Magasin nextProxy){

this.nextProxy = nextProxy;

}

}