# **How to create services in Java**

A blog by Alberto G (Alberto Gutierrez)

<http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/>

Creating services is quite an abstract subject, everyone has his own ideas and preconceptions, so is worth clarifying what I mean by services in this article. A service is a component that holds some business logic which can be easily reused anywhere else in the application no matter where the service is located, or what the communication mechanism is used between the client and the service. The following code samples detail my own personal approach on how to implement services in Java.

First thing is to design the service itself through its interface

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **interface** ServiceA {
2. **public** **void** doThis();
3. **public** **void** doThat();
4. }

public interface ServiceA {

public void doThis();

public void doThat();

}

Next step is writing the implementation of the service. Is also important to remind that the implementation should just read business logic, it shouldn’t be aware of if is going to be called through a web service, jms…

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** ServiceAImp **implements** ServiceA{
3. **protected** ServiceAImp (){};
5. @Override
6. **public** **void** doThat() {
7. System.out.println("do that");
8. }
10. @Override
11. **public** **void** doThis() {
12. System.out.println("do this");
13. }
15. }

public class ServiceAImp implements ServiceA{

protected ServiceAImp (){};

@Override

public void doThat() {

System.out.println("do that");

}

@Override

public void doThis() {

System.out.println("do this");

}

}

Now it comes the tricky part, I like to make the services look as if they were local to whoever is using them, it makes them easier to use and because the communication code is hidden and doesn’t need to be written every time a service is invoked they are also less prone to bugs. For this purpose I like to use [the proxy pattern](http://en.wikipedia.org/wiki/Proxy_pattern).

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** ServiceAProxy **implements** ServiceA {
2. ServiceA implementation;
4. **protected** ServiceAProxy(ServiceA imp) {
5. **this**.implementation = imp;
6. }
8. **public** **void** doThat() {
9. implementation.doThat();
10. }
12. **public** **void** doThis() {
13. implementation.doThis();
14. }
15. }

public class ServiceAProxy implements ServiceA {

ServiceA implementation;

protected ServiceAProxy(ServiceA imp) {

this.implementation = imp;

}

public void doThat() {

implementation.doThat();

}

public void doThis() {

implementation.doThis();

}

}

Next thing is to deal with the communication issue, now we have a proxy that makes the service look as local to the client, but there isn’t an implementation of a communication mechanism between the client and the service, to do so, we create as many different “stubs” as communication mechanisms we want to implement. For this example, we are going to support a remote WebService call, and a remote JMS call.

The WebService stub.

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** ServiceAWebServiceClient **implements** ServiceA{
2. ServiceA webService;
3. **public** ServiceAWebServiceClient() {
4. //webService = code to get the reference to Service A from the webservice call
5. }
7. @Override
8. **public** **void** doThat() {
9. webService.doThat();
10. }
12. @Override
13. **public** **void** doThis() {
14. webService.doThis();
15. }
16. }

public class ServiceAWebServiceClient implements ServiceA{

ServiceA webService;

public ServiceAWebServiceClient() {

//webService = code to get the reference to Service A from the webservice call

}

@Override

public void doThat() {

webService.doThat();

}

@Override

public void doThis() {

webService.doThis();

}

}

The WebService implementation could be something like:

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. @WebService
2. **public** **class** ServiceAWebService **implements** ServiceA{
3. ServiceA imp = **new** ServiceAImp();
5. @Override
6. **public** **void** doThat() {
7. imp.doThat();
8. }
10. @Override
11. **public** **void** doThis() {
12. imp.doThis();
13. }
14. }

@WebService

public class ServiceAWebService implements ServiceA{

ServiceA imp = new ServiceAImp();

@Override

public void doThat() {

imp.doThat();

}

@Override

public void doThis() {

imp.doThis();

}

}

The JMS stub

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** ServiceAJmsClient **implements** ServiceA{
2. ServiceA jmsObject;
4. **public** ServiceAJmsClient() {
5. //jmsObject = code to get the reference to Service A from the jms call
6. }
8. @Override
9. **public** **void** doThat() {
10. jmsObject.doThat();
11. }
13. @Override
14. **public** **void** doThis() {
15. jmsObject.doThis();
16. }
17. }

public class ServiceAJmsClient implements ServiceA{

ServiceA jmsObject;

public ServiceAJmsClient() {

//jmsObject = code to get the reference to Service A from the jms call

}

@Override

public void doThat() {

jmsObject.doThat();

}

@Override

public void doThis() {

jmsObject.doThis();

}

}

Now the service is almost ready, everything is kept separated so changes to the business logic are only done in the ServiceAImp, and changes to the actual transportation mechanism are implemented in their own stub, also we provide with a proxy so that whoever is using the service will use it transparently, the only problem left we have, which is a big one, is that this is quite a collection of classes we are asking our clients to understand in order to use our service, to fix this, I like to use a factory.

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** ServiceAFactory {
2. **private** ServiceAFactory() {}
4. **public** **static** ServiceA getLocalService(){
5. **return** **new** ServiceAImp ();
6. }
8. **public** **static** ServiceA getRemoteServiceUsingJms(){
9. **return** **new** ServiceAProxy (**new** ServiceAJmsClient());
10. }
12. **public** **static** ServiceA getRemoteServiceUsingWebService(){
13. **return** **new** ServiceAProxy (**new** ServiceAWebServiceClient());
14. }
15. }

public class ServiceAFactory {

private ServiceAFactory() {}

public static ServiceA getLocalService(){

return new ServiceAImp ();

}

public static ServiceA getRemoteServiceUsingJms(){

return new ServiceAProxy (new ServiceAJmsClient());

}

public static ServiceA getRemoteServiceUsingWebService(){

return new ServiceAProxy (new ServiceAWebServiceClient());

}

}

The previous factory is the only thing our customers need to know about us! To put all the pieces together, let’s look at how a client code using our service will look like.

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. **public** **class** MainClass {
3. /\*\*
4. \* @param args
5. \*/
6. **public** **static** **void** main(String[] args) {
7. //Calling doThis with a local instance of the ServiceA, very useful for testing!!!!
8. ServiceA serviceA1 = ServiceAFactory.getLocalService();
9. serviceA1.doThis();
10. //Calling doThis with a remote instance of the ServiceA using Web services.
11. ServiceA serviceA2 = ServiceAFactory.getRemoteServiceUsingWebService();
12. serviceA2.doThis();
13. //Calling doThis with a remote instance of the ServiceA using Web services.
14. ServiceA serviceA3 = ServiceAFactory.getRemoteServiceUsingJms();
15. serviceA3.doThis();
16. }
18. }

public class MainClass {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

//Calling doThis with a local instance of the ServiceA, very useful for testing!!!!

ServiceA serviceA1 = ServiceAFactory.getLocalService();

serviceA1.doThis();

//Calling doThis with a remote instance of the ServiceA using Web services.

ServiceA serviceA2 = ServiceAFactory.getRemoteServiceUsingWebService();

serviceA2.doThis();

//Calling doThis with a remote instance of the ServiceA using Web services.

ServiceA serviceA3 = ServiceAFactory.getRemoteServiceUsingJms();

serviceA3.doThis();

}

}

[EDIT] Now you can make the client even less aware of the service location, you can use a dependency injection framework as Spring to set the implementation of your service.

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. `
2. **public** **class** MainClassUsingSpring {
3. **public** **static** **void** main(String[] args) {
4. ServiceA serviceA = loadFromSpring();
5. serviceA.doThis();
6. }
8. **private** **static** ServiceA loadFromSpring() {
9. ApplicationContext context = **new** ClassPathXmlApplicationContext ("beans.xml");
10. **return** (ServiceA) context.getBean("serviceAProxy");
11. }
13. }

`

public class MainClassUsingSpring {

public static void main(String[] args) {

ServiceA serviceA = loadFromSpring();

serviceA.doThis();

}

private static ServiceA loadFromSpring() {

ApplicationContext context = new ClassPathXmlApplicationContext ("beans.xml");

return (ServiceA) context.getBean("serviceAProxy");

}

}

Then you can have a spring file configuration similar to this.

[view plaincopy to clipboardprint?](http://www.makinggoodsoftware.com/2009/11/17/how-to-create-services-in-java/)

1. <bean class="ServiceAProxy" id="serviceAProxy">
2. <constructor-arg>
3. <bean class="ServiceAWebServiceClient" id="serviceAImp">
4. </bean>
5. </constructor-arg>
6. </bean>

By changing your spring configuration file now you are able to change the implementation of your service and its location without even changing your source code!