Criando um webservice e um client pelo Eclipse

Passo-a-passo para criação de um web service simples, bem como um cliente para consumir esse web service, ou algum outro serviço já existente.

Web Service:

- No Eclipse, vá em File->New->Project e crie um "Dynamic Web Project".

Nesse projeto web ficará o webservice. Caso um servidor de aplicação (Tomcat, JBoss, ...) ainda não tenha sido configurado e/ou integrado ao Eclipse, ele será necessário.

- Dê um nome ao projeto (por ex, *MeuProjeto*), as outras opções podem ser padrão.

IMPORTANTE: Não clique diretamente em Finish, vá clicando em Next, pois na última tela uma mudança será necessária.

- Na tela "Web Module", selecione a opção "Generate web.xml deployment descriptor".

Outra alternativa é lembrar de criar o arquivo web.xml manualmente.

- Vá agora em File->New->Other... e crie um "Simple Web Service".
- Configurações:

Technology: mantenha "JAX-WS(WSDL-based)"
Dynamic web project: escolha o nome do projeto criado nos passos anteriores.
service name: complemento do link que será usando em sua URL. Exemplo: http://localhost:8080/MeuProjeto/servicename
Update web.xml: deixe selecionado
Package e Class: escolha de acordo com sua conveniência, exemplo: br.com.empresa.pacote. O nome do pacote será o namespace utilizado pelo serviço (de trás para frente). Ele poderá ser alterado, na classe, utilizando as opções da Annotation @WebService.
Essas opções farão o Eclipse criar uma classe padrão, com um exemplo simples de web-service "Hello World", com Annotations (versão do Eclipse 3.6) e editar o web.xml do projeto, criando um mapeamento para o webservice.

- A classe criada pelo Eclipse possuirá um único método chamado *sayHello*, que recebe uma String como parâmetro. Esse método será o responsável por processar a chamada do seu serviço, portanto é interessante alterar o seu nome para algo que condiga mais com o webservice. Além disso, por default, o nome do parâmetro recebido pelo método não é utilizado pelo webservice em si, a não ser que se utilize a Annotation @WebParam para definir esse nome. Por exemplo:

```
public String metodo(@WebParam(name="nome") String esteNomeNaoImportaParaOWebService, @WebParam(name="objeto") MeuObjeto obj)
```

Um único webservice pode ter vários métodos, e um único servidor pode conter vários webservices.

- Dê um Start no servidor onde o projeto foi criado e acesse o link http://localhost:8080/MeuProjeto/servicename?wsdl

Isso deverá retornar um XML com o conteúdo do WSDL do serviço.
IMPORTANTE: Suas classes personalizadas (ou seja, POJOs como o MeuObjeto, no exemplo) devem conter todos os getters e setters necessários, para que o WSDL seja gerado corretamente.

Cliente:

- Crie um projeto Java no Eclipse.
- Clique com o botão direito no projeto, escolha New->Other... e crie um "Web Service Client".
- Em "Service definition", copie e cole o seu link wsdl (no exemplo acima, http://localhost:8080/MeuProjeto/servicename?wsdl). As outras opções podem ser padrão.

IMPORTANTE: O servidor de aplicação deve estar funcionando para que essa etapa funcione, pois o Eclipse irá obter o WSDL em tempo real. Caso o link utilize SSL (link seguro do tipo HTTPS) provavelmente o Eclipse dará a mensagem "The service definition selected is invalid". Nesse caso, será necessário importar o certificado de acesso no Eclipse.
Os passos feitos até agora fazem o Eclipse utilizar automaticamente a ferramenta WSDL2Java (da biblioteca Apache Axis) para gerar diversas classes no projeto Java, dentro de um pacote específico definido pelo namespace do serviço. As classes criadas são todos os POJOs usados no webservice, juntamente com 5 classes (considerando a versão do Eclipse 3.6), com o nome iniciando pelo mesmo nome da classe do serviço. O wizard também colocará no Build Path do projeto as libs necessárias.

- Crie uma nova classe no projeto e utilize a seguinte linha para conectar o serviço.

```
System.out.println(new ClasseProxy().metodo("meu nome", new MeuObjeto()));
```

- Basta agora executá-la como uma aplicação Java comum. Logicamente o webservice deve estar no ar para que o cliente possa consumí-lo.

Esse wizard tem a desvantagem de recriar, no novo projeto, toda a estrutura de classes POJO do webservice. Uma opção é, após a criação do cliente, apagar as classes POJO recriadas e, através do Build Path (aba Projects), adicionar o outro projeto onde estão as suas classes POJO usadas pelo webservice. Assim, qualquer alteração feita lá será refletida no projeto cliente.

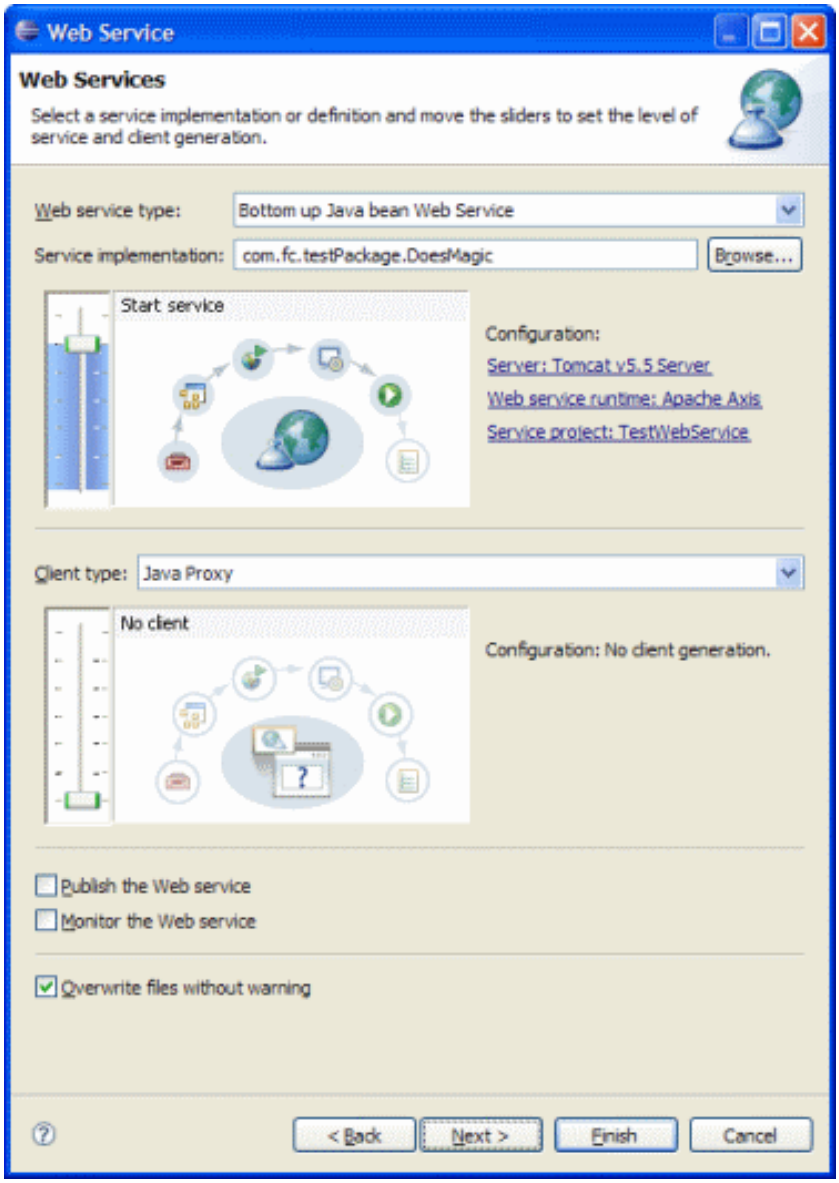
Exemplo 2

CREATE BASIC WEB SERVICE

- Create new Dynamic Web Project.
- Create new Java class (call it "DoesMagic").
- Add method like this (perhaps):

```
1       public class DoesMagic{
2           public String doBasicStuff(String message)
3           {
4               return "This is your message: " + message;
5           }
6       }
```

4. Right click on class you just made, New -> Other -> Web Services -> Web Service, you should get a form like the one below:



This will create all the necessary files to be able to publish your class as a web service. If you want you can also create a client automatically by moving the slider – but what it generates may be hard to understand at first glance, so I have written a simple example of client code (see later on…).

5. In order to run the web service – just, Run as -> Run on Server. Once the web service is running and alive on Tomcat – then we should be able to use it via some client code (see next bit).

CREATE BASIC WEB SERVICE CLIENT

Must create a project that has all the axis.jar type stuff in the WebContent folder – this can usually be made by generating a client application via the above wizard – you don't have to use all the auto generated classes to access your web service, just do the following:

1. Create a new Java Class – call it TestClient.
2. Make it a main class, and enter the following code:

```
import javax.xml.namespace.QName;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class TestClient {

public static void main(String [] args)
{
    try{
       String endpoint = "http://localhost:8080/TestWebService/services/DoesMagic";
       Service service = new Service();
       Call call = (Call) service.createCall();
       call.setTargetEndpointAddress( new java.net.URL(endpoint) );
       call.setOperationName( new QName("http://testPackage.fc.com", "doBasicStuff") );

       String ret = (String) call.invoke( new Object[] {"Hello World"} );
       System.out.println(ret);
       }
       catch(Exception e){
           System.err.println(e.toString());
       }
}
}
```

3. Go to your web service project you made before and look at the source code for the wsdl file that has been created. It's in the folder WebContent -> wsdl. Inside there you will find a wsdl that is the same name as your class.

4. You need to look at the wsdl and find the endpoint – which looks similar to:
"http://localhost:8080/TestWebService/services/DoesMagic"
and you need to get the namespace and the method name you want to invoke (as shown above, should be in a very similar format).

5. In this case, when we get to the point that the call.invoke command is being issued it is casting the result to a String and we are sending in a String called "Hello World" – this should create a message in the console like "This is your message: Hello World".

6. To test the class, just run the project as a Java Application and you should see the result in the Console printed out.