You have to do almost the same set of tasks as you did for Contract First Approach, but instead of creating a WSDL file, you create a Java file like the one below. (Look at the package I have used, it is better to use a package as they will be associated with a namespace automatically).

```
package www.example.com;

public class StockPrice {

public float getStockValue(String tickerName)
{
if(null!=tickerName && !(tickerName.isBlank()) &&
tickerName.equalsIgnoreCase("ORCL"))
{
return 82.40f;
}
return 0.0f;
}
}
```

You select this java file and right click->Web Services->Create Web Service and choose to do the same as you did for the Create Java Web Bean Service in the Contract First Approach. The rest is similar.

For this java code, the WSDL file that will be generated will look like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://com.example.www"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://com.example.www" xmlns:intf="http://com.example.www"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://com.example.www"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="getStockValue">
    <complexType>
     <sequence>
      <element name="tickerName" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="getStockValueResponse">
    <complexType>
     <sequence>
      <element name="getStockValueReturn" type="xsd:float"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>

   <wsdl:message name="getStockValueRequest">

      <wsdl:part element="impl:getStockValue" name="parameters">

      </wsdl:part>
```

```
   </wsdl:message>

   <wsdl:message name="getStockValueResponse">

      <wsdl:part element="impl:getStockValueResponse" name="parameters">

      </wsdl:part>

   </wsdl:message>

   <wsdl:portType name="StockPrice">

      <wsdl:operation name="getStockValue">

         <wsdl:input message="impl:getStockValueRequest"
name="getStockValueRequest">

       </wsdl:input>

         <wsdl:output message="impl:getStockValueResponse"
name="getStockValueResponse">

       </wsdl:output>

      </wsdl:operation>

   </wsdl:portType>

   <wsdl:binding name="StockPriceSoapBinding" type="impl:StockPrice">

      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="getStockValue">

         <wsdlsoap:operation soapAction=""/>

         <wsdl:input name="getStockValueRequest">

            <wsdlsoap:body use="literal"/>

         </wsdl:input>

         <wsdl:output name="getStockValueResponse">

            <wsdlsoap:body use="literal"/>

         </wsdl:output>

      </wsdl:operation>

   </wsdl:binding>

   <wsdl:service name="StockPriceService">

      <wsdl:port binding="impl:StockPriceSoapBinding" name="StockPrice">
```

```
        <wsdlsoap:address
location="http://localhost:8080/CodeFirstApproach/services/StockPrice"/>

      </wsdl:port>

   </wsdl:service>

</wsdl:definitions>
```

In order to test it in Postman, your URL will change as per the WSDL's files endpoint definition and your input SOAP request message will look like the one below.

```
<soapenv:Envelope
      xmlns:mns="http://com.example.www"
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Body>
                  <mns:getStockValue>
                        <tickerName>ORCL</tickerName>
                       </mns:getStockValue>
      </soapenv:Body>
</soapenv:Envelope>
```

Which yields a response like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:x
sd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <soapenv:Body>
        <getStockValueResponse xmlns="http://com.example.www">
            <getStockValueReturn>82.4</getStockValueReturn>
        </getStockValueResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Question for you:

You do not see any other java code other than the one you have written to define your service. Where does the server know what to look for and how to invoke your java code as a service? Look through the files that the IDE generates in your project and find out any additional files that are being created and what it contains.