

IDU0080

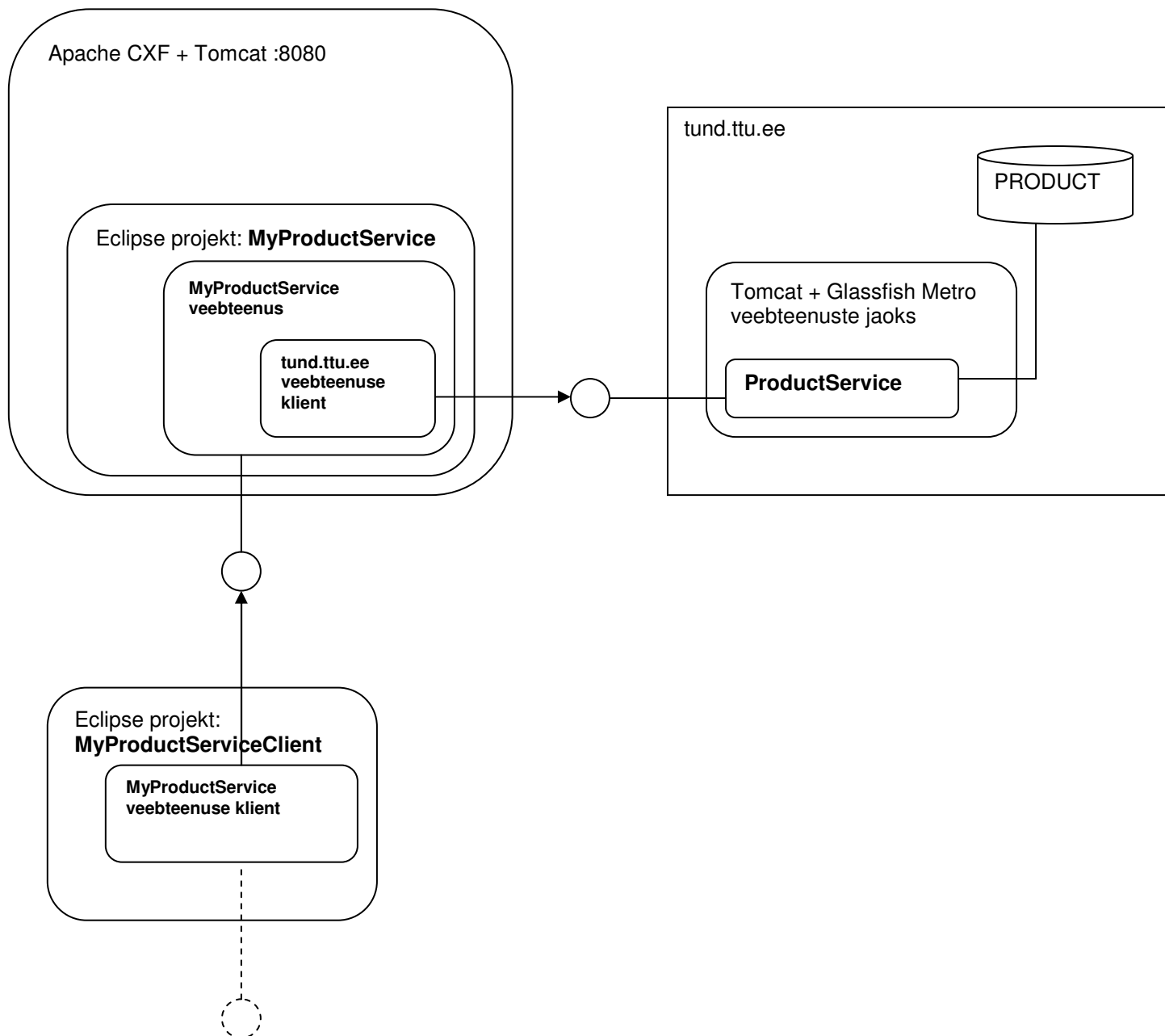
## 5. koduülesande abimaterjal.

Näide selle kohta kuidas teha veebteenust Eclipse/ApacheCXF-i/Tomcat-i keskkonnas mis kasutab ühte tund.ttu.ee serveri teenust ja sellele teenusele omakorda klienti.

Selle näite mõte on selles et kui te ei kujuta päris hästi ette kuidas koduülesannet nr. 5 alustada siis selles näites tehakse sellest koduülesandest põhimõtteliselt üks sarnane „nurk” ära (see osa mis tegeleb kullerfirmade päringuga) . Et asi ei läheks siiski liiga lihtsaks on siin kasutatud selliseid teenuseid ja andmeid mis teie koduülesande teemaga kokku ei lange.

<b>1. Näite arhitektuur. ....</b>	<b>2</b>
<b>2. tund.ttu.ee teenus ProductService .....</b>	<b>3</b>
<b>3. Teeme eraldi Eclipse projektina oma ProductService'i.....</b>	<b>3</b>
<b>4. Teeme eraldi Eclipse projektina oma ProductService kliendi.Kokkuvõte.....</b>	<b>22</b>

## 1. Näite arhitektuur.



**MyProductService** veebteenus on ühtlasi tund.ttu.ee veebteenuse **ProductService** kliendiks.

## 2. tund.ttu.ee teenus ProductService

Serveril tund.ttu.ee asub toote andmete teenus mida selle näite käigus tehtav MyProductService peab andmete serveerimisel kasutama – peab sealt küsima toodete andmed.

Teenuse wsdl asub aadressil

<http://tund.ttu.ee/orderservice/products?wsdl>

Kõigi selles ülesandes vajalike teenuse nimekiri asub aadressil

<http://tund.ttu.ee/orderservice/orders>

NB! Selles näites kasutataud ProductService'it ei ole koduülesandes vaja kasutada, selline teenus on üles pandud ainult selle näite tegemiseks.

ProductService kasutab PostgreSQL-i andmebaasi, andmebaasi loomise skriptid leiata aadressilt

<http://maurus.ttu.ee/idu0080/5/db/>

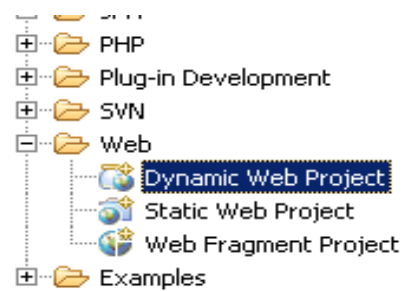
Sealt, failist INSERT\_DATA\_ORDER.txt saab siivi vaadata mis andmed on selle andmebaasi PRODUCT tabelis, sealt näeb ka mis andmed on teistest tabelites (näiteks palju on tellimusi tabelis ORDER ja mis on nende andmed )

## 3. Teeme eraldi Eclipse projektina oma ProductService'i.

Selles juhendis on eeldatud et juhendi lugeja oskab Eclipse seadistada Apache CXF-i ja Tomcat-i kasutama. Kui sellist oskust ei ole tuleks enne vaadata mõnest vastavast Eclipse kasutamise juhendist.

Selle punkti eesmärgiks on teha Eclipse „Dynamic Web Project” mis kasutab serverina Apache Tomcat 7-t ja veebteenuste tarkvarakihina Apache CXF-i.

1). Teeme Eclipse uue „Dynamic Web Project”-i.



**New Dynamic Web Project**

**Dynamic Web Project**  
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location  
☒ Use default location  
Location:

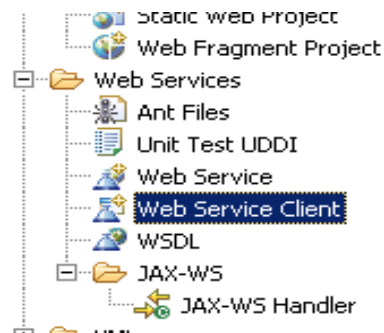
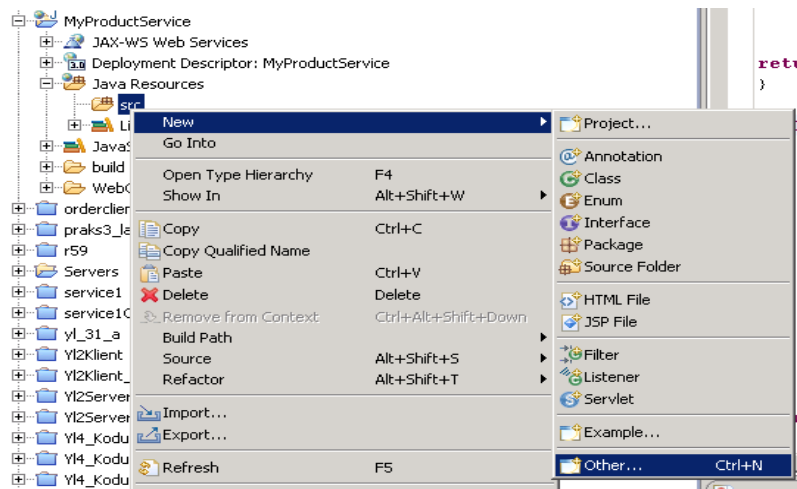
Target runtime

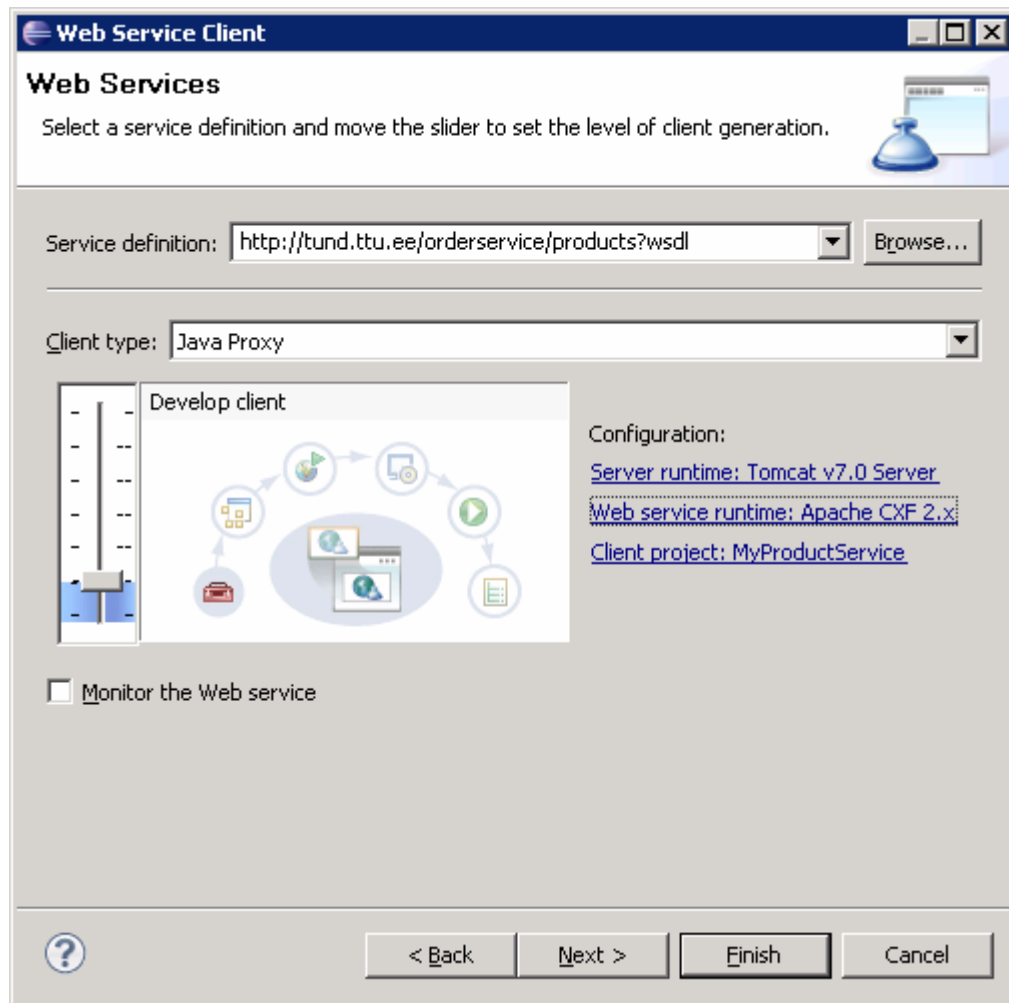
Dynamic web module version

Configuration  
   
A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership  
☐ Add project to an EAR  
EAR project name:

2) Teeme Eclipse uue veebteenuste kliendi (genereerime Eclipsega ).





Teenuse wsdl-i aadressiks on <http://tund.ttu.ee/orderservice/products?wsdl>  
Oleme kindlad et „Web service runtime” on Apache CXF (tegelikult sobib Axis2 ka aga jääme praegu meile juba tuntud CXF-i juurde.

NB! Kui „Server runtime” alt pole võimalik Tomcat-i valida siis tuleb see seadistada Eclipse üldiste seadistuste alt (“Preferences”)

**Web Service Client**

**Apache CXF 2.3.3 Web Service WSDL2Java Client Configuration**

Customise your Web Service generation by selecting options on this page

**Output Directory:** \\MyProductService\\src

**Package Name:** ttu.idu0080.order.server

☐ Specify WSDL Namespace to Package Name Mappings

**Binding Files:**

Add...

Remove...

Specify zero, or more, JAXWS or JAXB

Vaikimisi tahavad kõik tund.ttü.ee teenused paketi nimeks võtta ttu.idu0080.order.server

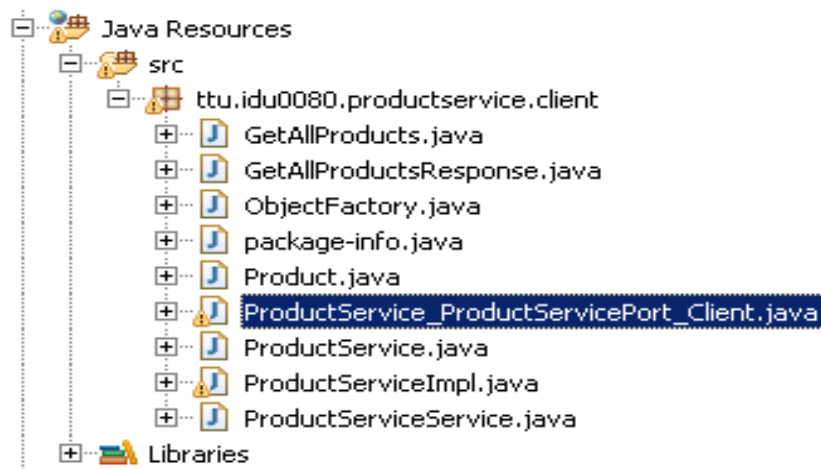
Võib ka. Kui aga arvestada et näiteks koduülesandes on vaja sellelt serverilt kasutada mitmeid teenused ja et erinevate teenuste kliendiosad ühe projekti sees segamini ei läheks – muudan paketi nime ära, selgemaks. Hiljem on siis aru saada mis teenuse kliendiga selles paketis tegu on.

**Output Directory:** \\MyProductService\\src

**Package Name:** ttu.idu0080.productservice.client

☐ Specify WSDL Namespace to Package Name Mappings

3) Muudame genereeritud veebteenuste kliendi koodi.  
Vaatame mis meile nüüd projekti lähtekoodi alla tekkis:



Pildil väljavalitud klassi lähtekood sisaldab nüüd teenuse väljakutsimist – seda võib redigeerida (ja ümber nimetada või kopeerida sealt teenusega ühendamise koodi).

Selles näites siin redigeerime koodi. Võtame  
`ProductService_ProductServicePort_Client.java`

lahti ja muudame mõnevõrra koodi.

Redigeerimise mõte on main-meetod asendada mõne teise nimega meetodiga mida on hea teisest klassist välja kutsuda, klassi konstruktor muuta „public”-uks ja meetodi sees täiendada koodi nii et meetod tagastaks meile sobivad (teenuselt laekunud) andmed.



-----ProductService\_ProductServicePort\_Client.java—(algne versioon)-----

```
package ttu.idu0080.productservice.client;

/**
 * Please modify this class to meet your needs
 * This class is not complete
 */

/**
 * This class was generated by Apache CXF 2.3.3
 * 2011-04-23T08:37:23.952+03:00
 * Generated source version: 2.3.3
 */
public final class ProductService_ProductServicePort_Client {

    private static final QName SERVICE_NAME = new QName("http://server.order.idu0080.ttu/", "ProductServiceService");

    private ProductService_ProductServicePort_Client() {

    }

    public static void main(String args[]) throws Exception {
        URL wsdlURL = ProductServiceService.WSDL_LOCATION;
        if (args.length > 0) {
            File wsdlFile = new File(args[0]);
            try {
                if (wsdlFile.exists()) {
                    wsdlURL = wsdlFile.toURI().toURL();
                } else {
                    wsdlURL = new URL(args[0]);
                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
        }

        ProductServiceService ss = new ProductServiceService(wsdlURL, SERVICE_NAME);
        ProductService port = ss.getProductServicePort();

        {
            System.out.println("Invoking getAllProducts...");
            java.util.List<ttu.idu0080.productservice.client.Product> _getAllProducts__return = port.getAllProducts();
            System.out.println("getAllProducts.result=" + _getAllProducts__return);
        }

        System.exit(0);
    }
}
```

See on tegelikult see  
mida tahame  
(toodete nimekiri  
tund.ttu.ee teenuselt)

Peale muudatust (import-laused ja kommentaardi on maha võetud siit):

---

```
package ttu.idu0080.productservice.client;

public final class ProductService_ProductServicePort_Client {

    private static final QName SERVICE_NAME = new QName("http://server.order.idu0080.ttu/",
"ProductServiceService");

    public ProductService_ProductServicePort_Client() {

    }

    public java.util.List<ttu.idu0080.productservice.client.Product> getProducts() throws Exception {
        URL wsdlURL = ProductServiceService.WSDL_LOCATION;
        String args[]={""};
        java.util.List<ttu.idu0080.productservice.client.Product> __getAllProducts__return = null ;
        if (args.length > 0) {
            File wsdlFile = new File(args[0]);
            try {
                if (wsdlFile.exists()) {
                    wsdlURL = wsdlFile.toURI().toURL();
                } else {
                    // wsdlURL = new URL(args[0]);
                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
        }

        ProductServiceService ss = new ProductServiceService(wsdlURL, SERVICE_NAME);
        ProductService port = ss.getProductServicePort();

        {
            System.out.println("Invoking getAllProducts...");
            // java.util.List<ttu.idu0080.productservice.client.Product> __getAllProducts__return =
port.getAllProducts();
            __getAllProducts__return = port.getAllProducts();
            System.out.println("getAllProducts.result=" + __getAllProducts__return);

        }

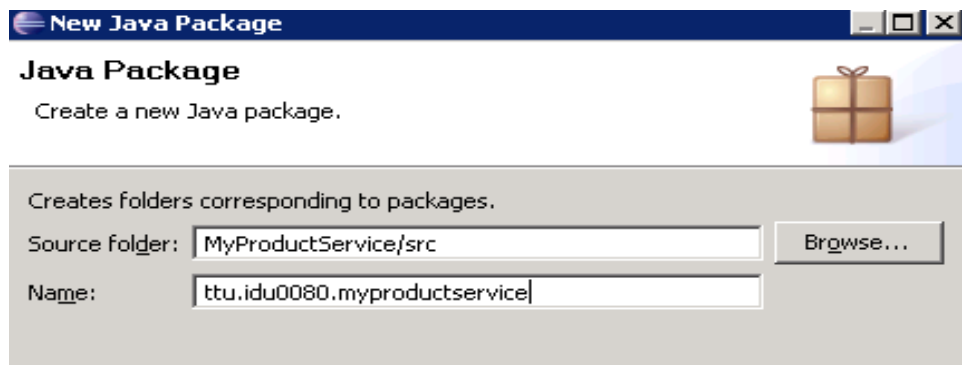
        return __getAllProducts__return ;
    }
    // System.exit(0);
}

}
```

---

4) Kirjutame oma klassi mis seda eelmises punktis kirjutatud kliendi koodi käivitab.

Sellest klassist on plaan teha veebteenust **MyProductService** realiseeriv klass, seepärast võiks ta panna teise nimega paketti.



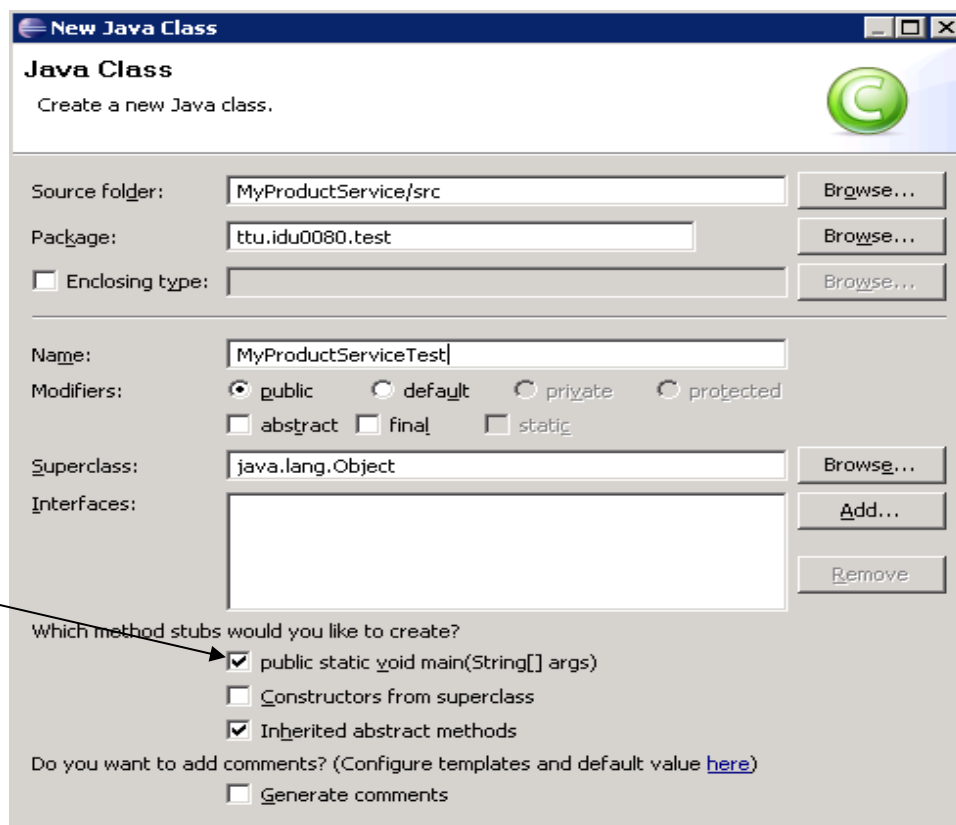
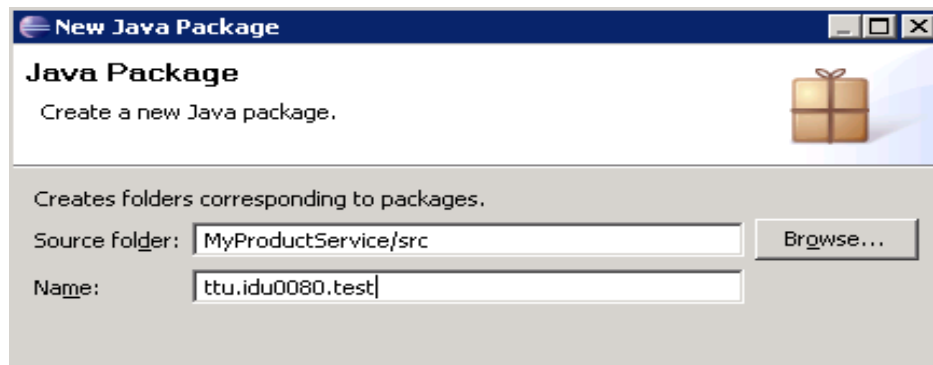
Klassi **MyProductServiceImpl** sisu on lihtne, ta pöördub meie (muudetud koodiga) veebteenuse kliendi poole:

```
-----  
package ttu.idu0080.productservice;  
  
import java.util.*;  
  
public class MyProductServiceImpl {  
  
    public List<ttu.idu0080.productservice.client.Product>  
getProductList()  
    {  
        ttu.idu0080.productservice.client.ProductService_ProductServicePort_C  
lient TundTtuEeProductServiceClient = new  
ttu.idu0080.productservice.client.ProductService_ProductServicePort_Client(  
    );  
        List<ttu.idu0080.productservice.client.Product> ProductList = null;  
        try {  
            ProductList = TundTtuEeProductServiceClient.getProducts()    ;  
        } catch (Exception e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
        return ProductList ;  
    }  
  
}
```

```
-----
```

5) Käivitame ja testime.

Teeme näiteks paketi „test” ja kirjutame sinan paketti ühe väikese „main” meetodiga klassi mida käivitades saame nüüd oma süsteemi käima tõmmata.



siia „lind”

Selle test-klassi sisse kirjutame väikese toodete massiivi väljatrükkimise meetodi:

```
public static void printProducts(List<ttu.idu0080.productservice.client.Product> products)
{
    if (products != null)
    {
        for (ttu.idu0080.productservice.client.Product p: products) {
            System.out.println("product id:" + p.getProduct() + " name:" + p.getName() );
        }
    }
}
```

Klass tervikuna:

```

package ttu.idu0080.test;
import java.util.*;
import ttu.idu0080.myproductservice.*;

public class MyProductServiceTest {

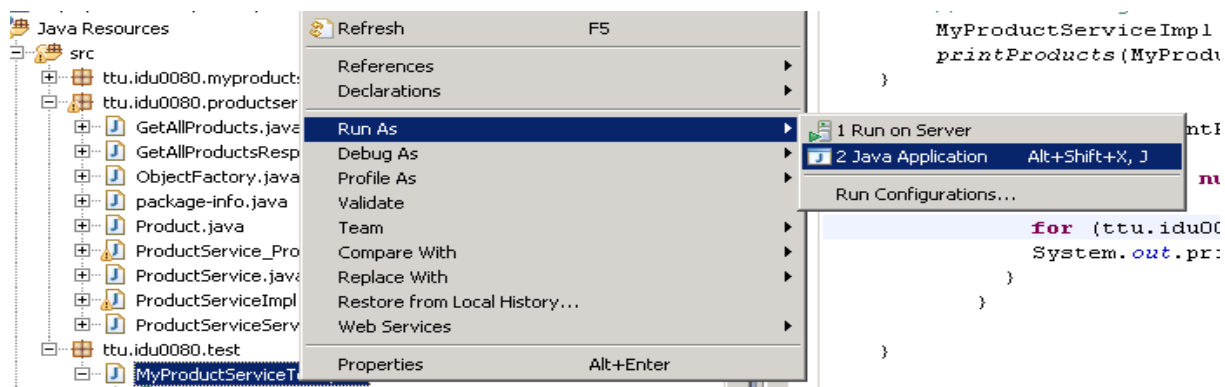
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MyProductServiceImpl MyProductService = new MyProductServiceImpl();
        printProducts(MyProductService.getProductList());
    }

    public static void printProducts(List<ttu.idu0080.productservice.client.Product> products)
    {
        if (products != null)
        {
            for (ttu.idu0080.productservice.client.Product p: products) {
                System.out.println("product id:" + p.getProduct() + " name:" + p.getName() );
            }
        }
    }

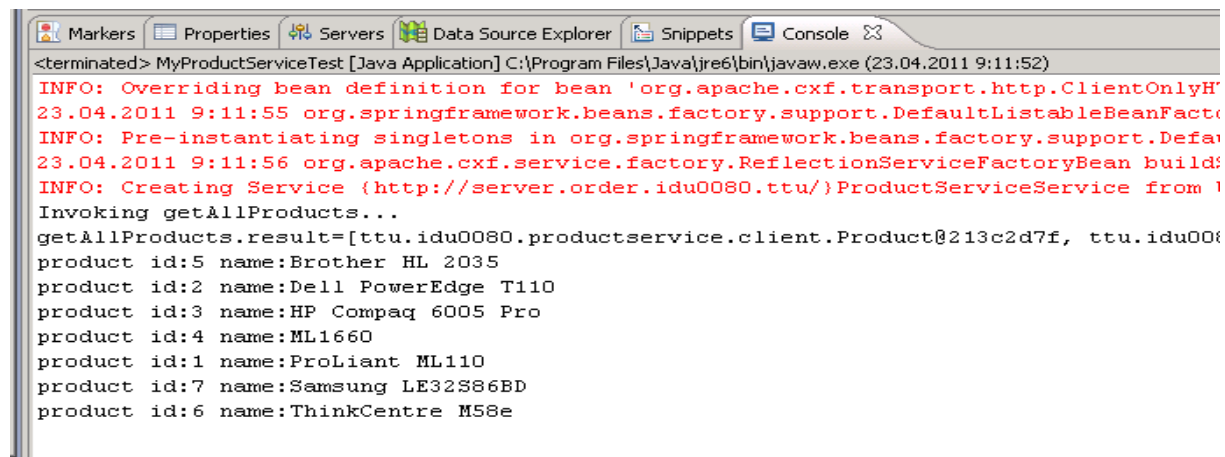
}

```

Käivitame **MyProductServiceTest**-i : „Run As Java Application”.

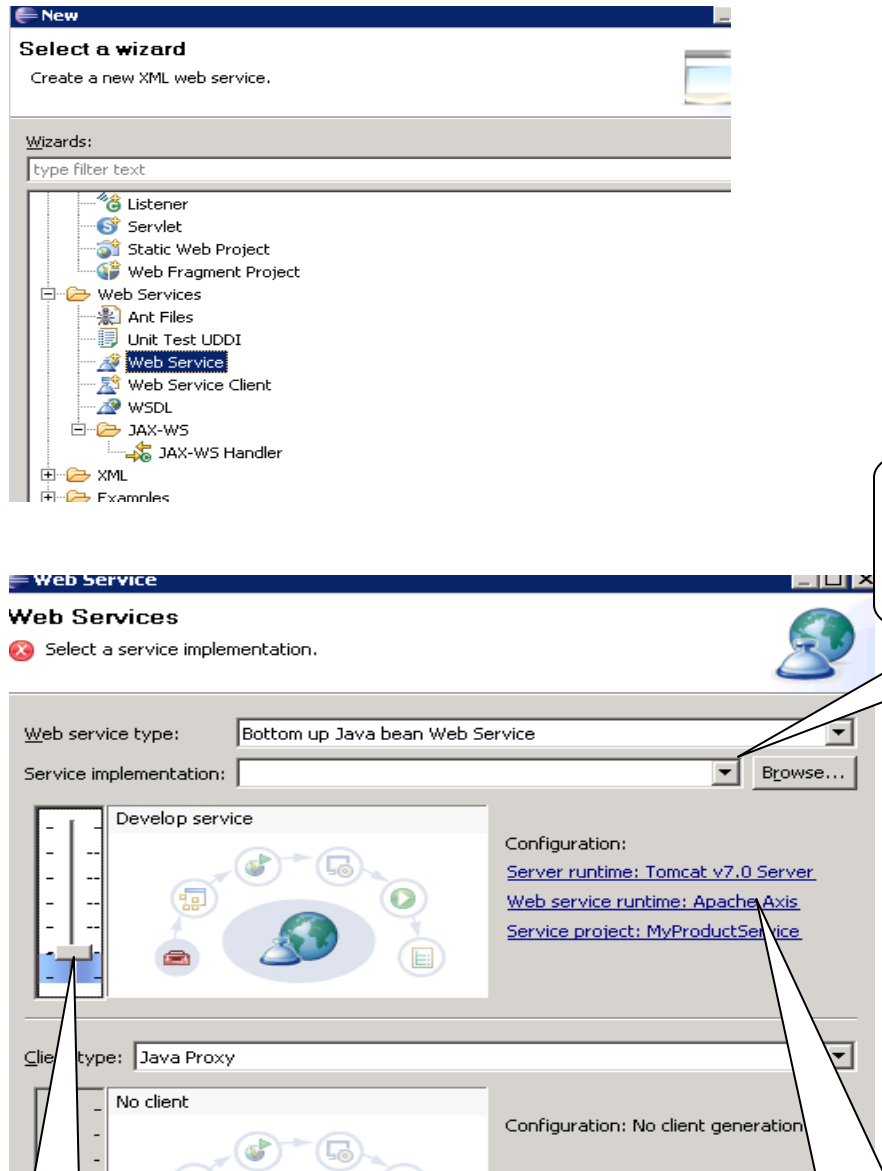


Käivitamise tulemusena saame konsoolile toodete nimekirja mis tulnud meie juurde serverist **tund.ttu.ee** üle SOAP teenuse.



Tundub et MyProductServiceImpl töötab, järgmises punktis teeme sellest veebteenuse.

6) Teeme eelmises punktis tehtud klassist veebteenuse (sellest klassist saab meie veebteenuse realiseerimise klassi).



Siia valime oma klasside hulgast veebteenuse realiseerimise (MyProductServiceImpl praegu) NB! Kui siin muuta siis muudetakse mõnikord automaatselt tagasi Axis-e peale „Web service runtime”

selle võime alla tõmmata, „develop” peale.

Siia Tomcat ja CXF

Ekraanivorm pärast muudatuste tegemist:

**Web Service**

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Bottom up Java bean Web Service

Service implementation: ttu.idu0080.myproductservice.MyProductServiceImpl [Browse...](#)

Develop service

Configuration:

- [Server runtime: Tomcat v7.0 Server](#)
- [Web service runtime: Apache CXF 2.x](#)
- [Service project: MyProductService](#)

Client type: Java Proxy

No client

Configuration: No client generation.

**Web Service**

**Apache CXF 2.3.3 Web Service Java Class Starting Point Cor**

Select an existing Service Endpoint Interface (SEI) or create an SEI by extracting an interface from the implementation class.

☒ Use a Service Endpoint Interface:

☐ Select an SEI: [Browse...](#)

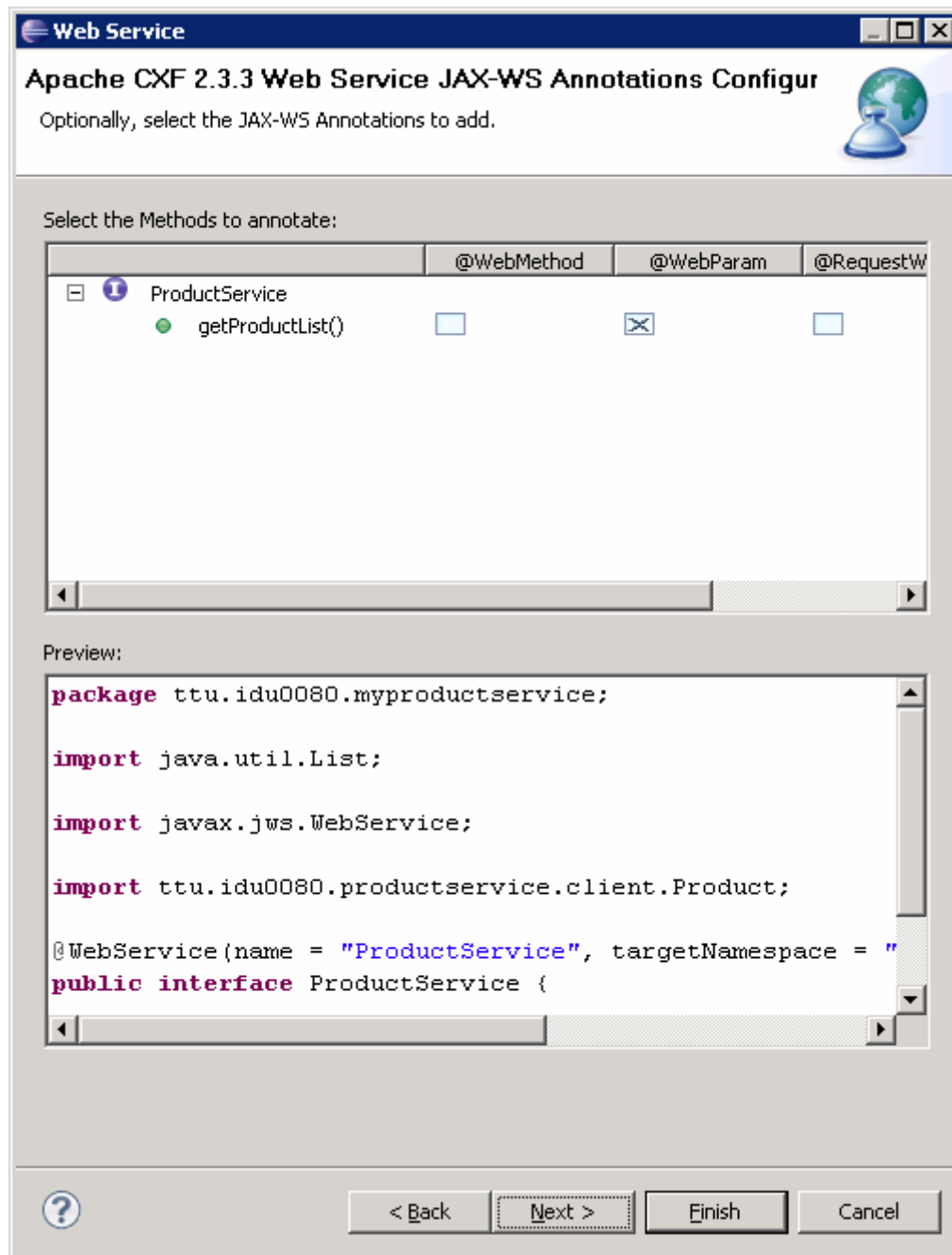
☒ Create an SEI: ProductService

Members to declare in the extracted SEI:

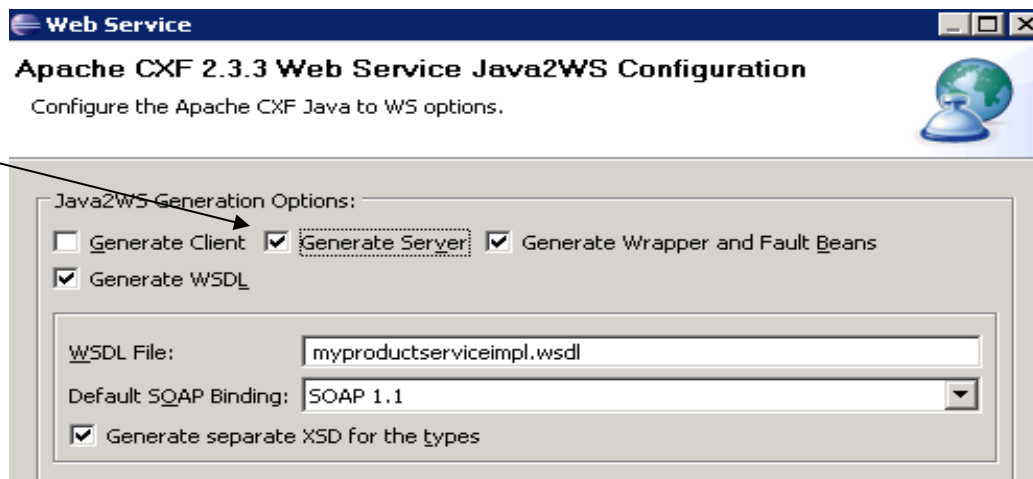
- ☒ getProductList()

[Select All...](#)

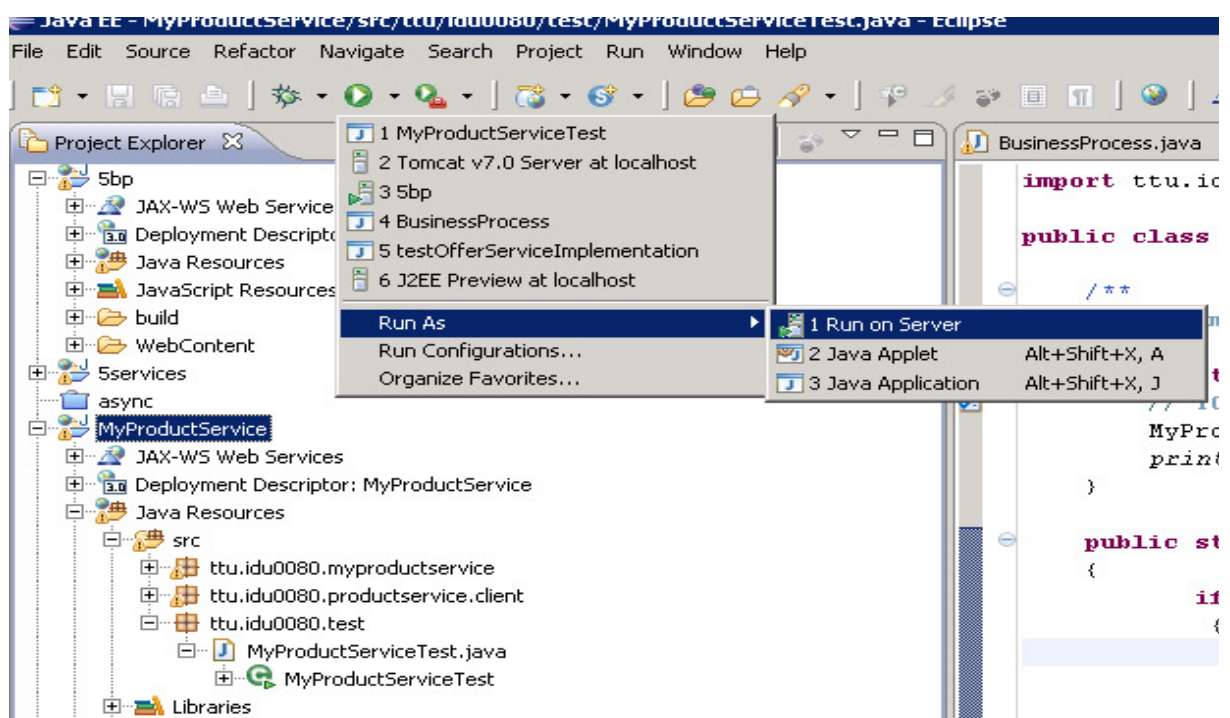
[Deselect All...](#)

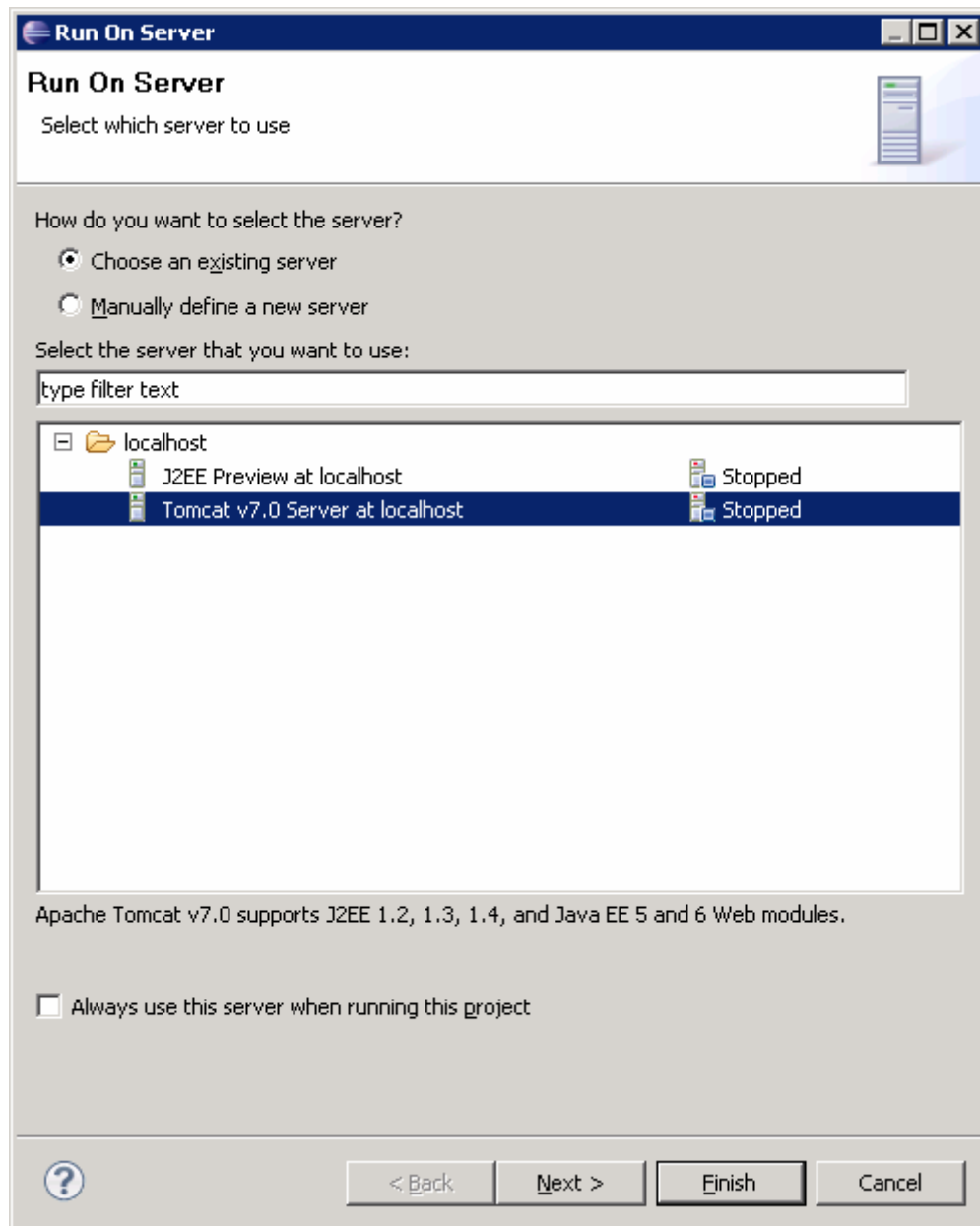


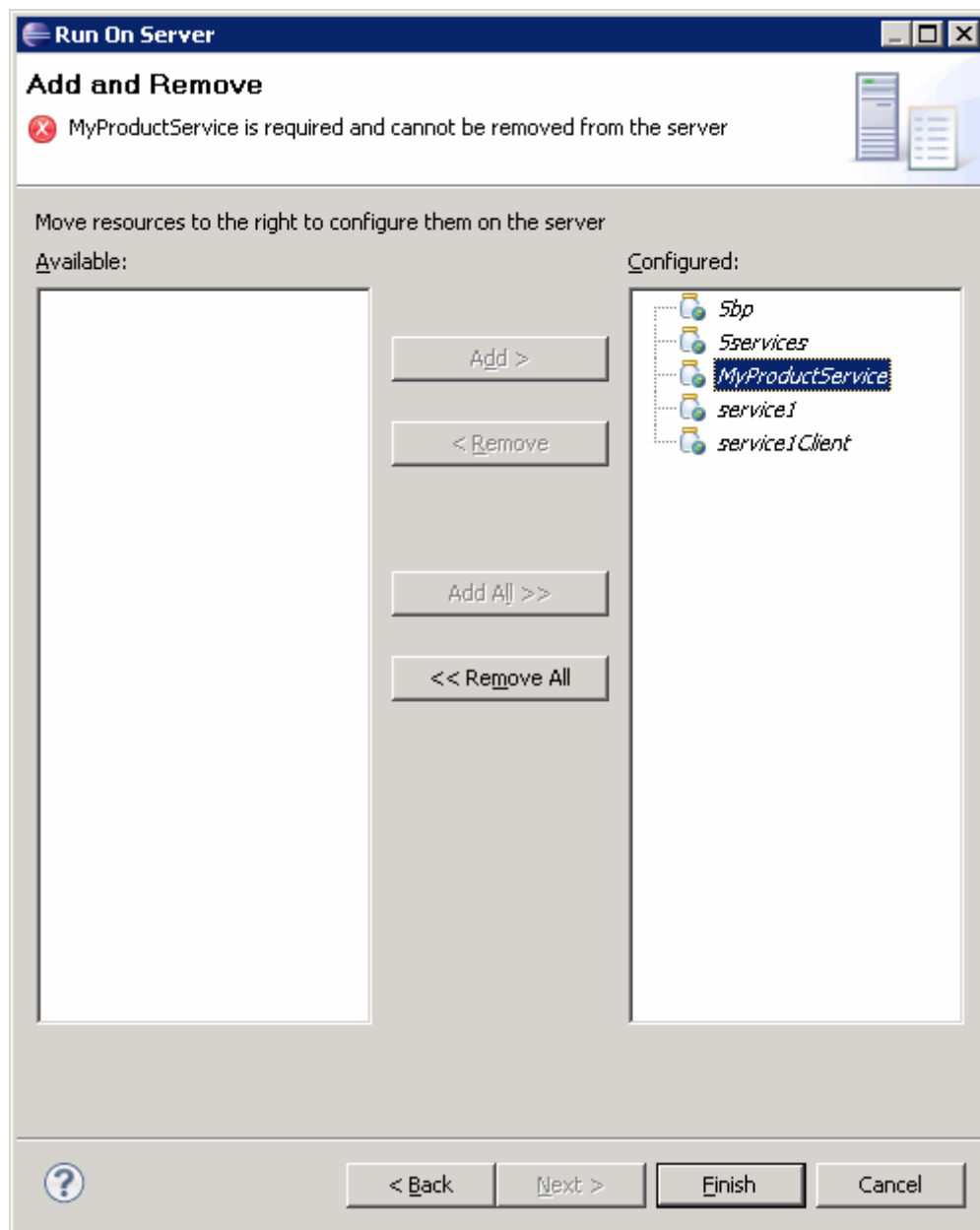




Paneme nüüd oma projekti rakendusena (ehk SOAP-teenusena) serverile üles.

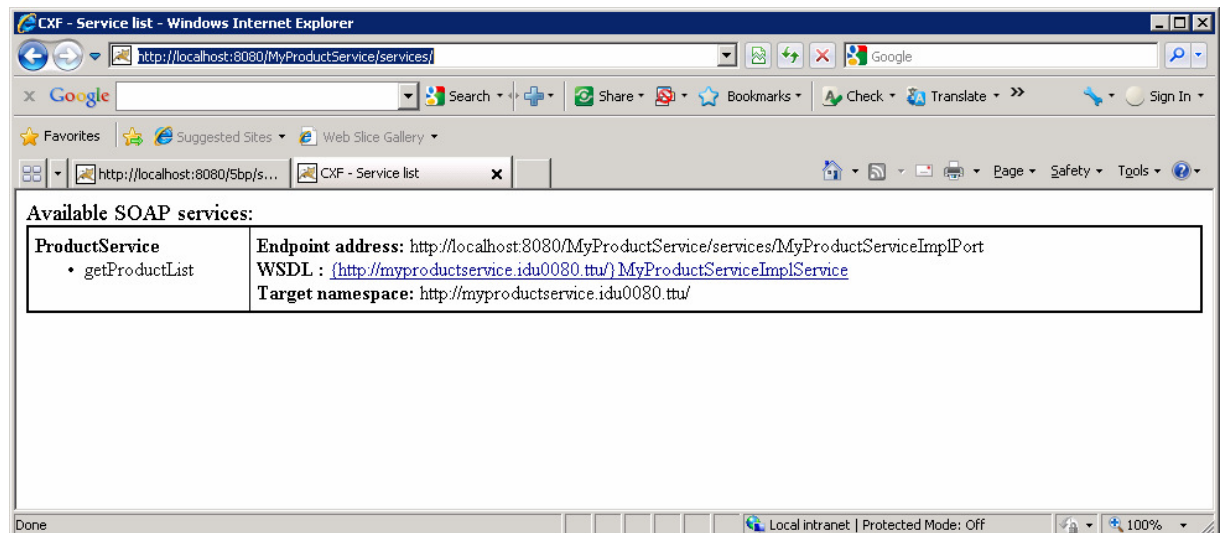




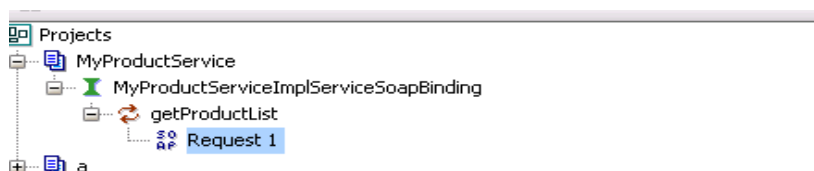
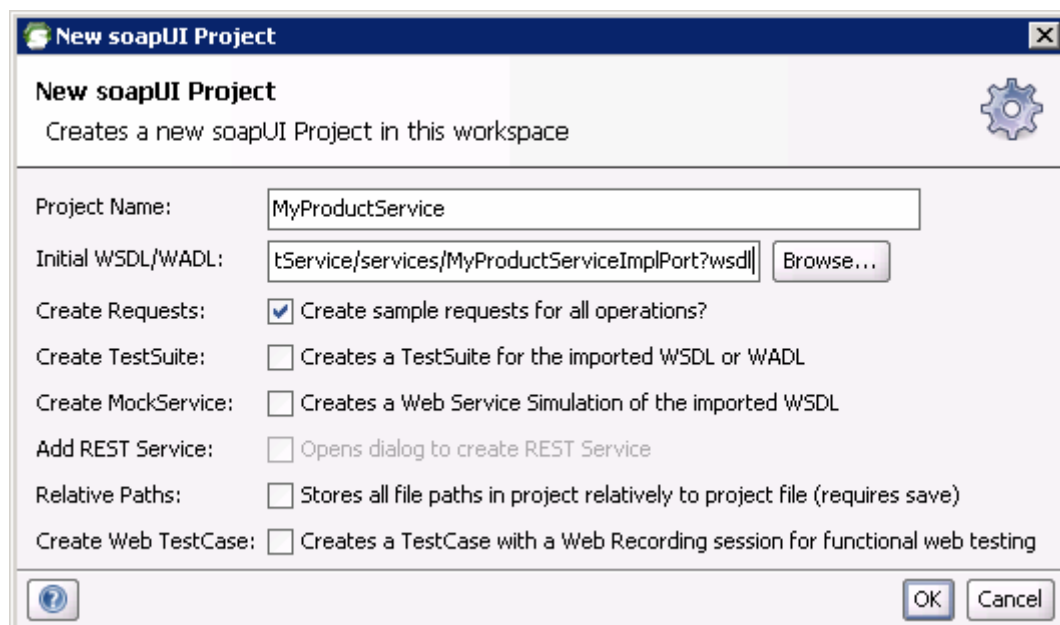


Vaatame aadressile

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



## 7) Testtime SoapUI-ga.



Request 1

http://localhost:8080/MyProductService/services/MyProductServiceImplPort

Raw XML

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <myp:getProductList/>
  </soapenv:Body>
</soapenv:Envelope>
```

Raw XML

```
<?xml version='1.0' encoding='UTF-8'>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:getProductListResponse xmlns:ns2="http://myproduct.com/ns2">
      <return>
        <description>Brother HL 2035 - printer - B&W - laser</description>
        <eshop_price>97.0</eshop_price>
        <name>Brother HL 2035</name>
        <product>5</product>
        <product_code>HL2035ZVM1</product_code>
      </return>
      <return>
        <description>Dell PowerEdge T110 - Core i3 540 3.06 GHz</description>
        <eshop_price>450.0</eshop_price>
        <name>Dell PowerEdge T110</name>
        <product>2</product>
        <product_code>S11T1102102E</product_code>
      </return>
      <return>
        <description>HP Compaq 6005 Pro - Phenom II X2 B55</description>
        <eshop_price>466.0</eshop_price>
        <name>HP Compaq 6005 Pro</name>
        <product>3</product>
        <product_code>AX357AWW#ABB</product_code>
      </return>
      <return>
        <description>Samsung ML 1660 printer B&W laser</description>
        <eshop_price>58.0</eshop_price>
        <name>ML1660</name>
      </return>
    </ns2:getProductListResponse>
  </soap:Body>
</soap:Envelope>
```

... Headers (5) Attachments (0) SSL Info WSS (0) JMS (0)

response time: 725ms (1641 bytes)

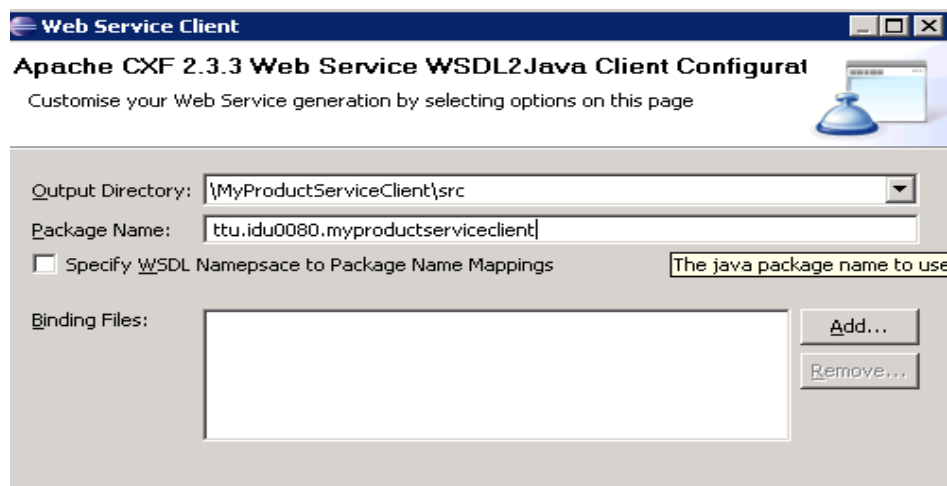
1 : 1

#### 4. Teeme eraldi Eclipse projektina oma ProductService kliendi.Kokkuvõte.

- 1) Teeme uue Eclipse „Dynamic Web Project”-i, näiteks nimega **MyProductServiceClient**
- 2) Teeme uue veebteenuste kliendi (genereerime Eclipse wizard-iga), enne veendume et meie punktis 3. tehtud veebteenus on meie serveril üleval ja server käivitatud.

Veebteenuse aadressiks on nüüd juba meie enda teenus definitsiooniga aadressil:

<http://localhost:8080/MyProductService/services/MyProductServiceImplPort?wsdl>



Pakettide nimed on mõistlik enne koduülesande tegemist läbi mõelda sest kui ühe Eclipse projekti all tegeletakse 6-7 veebteenuse kliendiga ja mitme serveriga võib juba üsna kergelt segamini minna kus midagi asub.

- 3) Muudame vajadusel genereeritud kliendi koodi.

Genereeritud klassi kood peale muudatusi:

```
-----  
package ttu.idu0080.myproductserviceclient;  
  
/**  
 * Please modify this class to meet your needs  
 * This class is not complete  
 */  
  
import java.io.File;  
import java.net.MalformedURLException;  
import java.net.URL;  
import javax.xml.namespace.QName;  
import javax.jws.WebMethod;  
import javax.jws.WebResult;  
import javax.jws.WebService;  
import javax.xml.bind.annotation.XmlSeeAlso;
```

```

import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

/**
 * This class was generated by Apache CXF 2.3.3
 * 2011-04-23T09:44:27.650+03:00
 * Generated source version: 2.3.3
 */
public final class ProductService_MyProductServiceImplPort_Client {

    private static final QName SERVICE_NAME = new QName("http://myproductservice.idu0080.ttu/",
"MyProductServiceImplService");

    public ProductService_MyProductServiceImplPort_Client() {

    }

    public java.util.List<ttu.idu0080.order.server.Product> getProductsFromMyService() throws Exception {
        URL wsdlURL = MyProductServiceImplService.WSDL_LOCATION;
        String args[] = {""};
        java.util.List<ttu.idu0080.order.server.Product> _getProductList__return = null;
        if (args.length > 0) {
            File wsdlFile = new File(args[0]);
            try {
                if (wsdlFile.exists()) {
                    wsdlURL = wsdlFile.toURI().toURL();
                } else {
                    // wsdlURL = new URL(args[0]);
                }
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
        }

        MyProductServiceImplService ss = new MyProductServiceImplService(wsdlURL, SERVICE_NAME);
        ProductService port = ss.getMyProductServiceImplPort();

        {
            System.out.println("Invoking getProductList...");
            _getProductList__return = port.getProductList();
            System.out.println("getProductList.result=" + _getProductList__return);

        }

        return _getProductList__return ;
    }
}

```

---

#### 4) Kirjutame klassi mis kutsub genereeritud kliendi koodi välja.

```

-----
package ttu.idu0080.test;

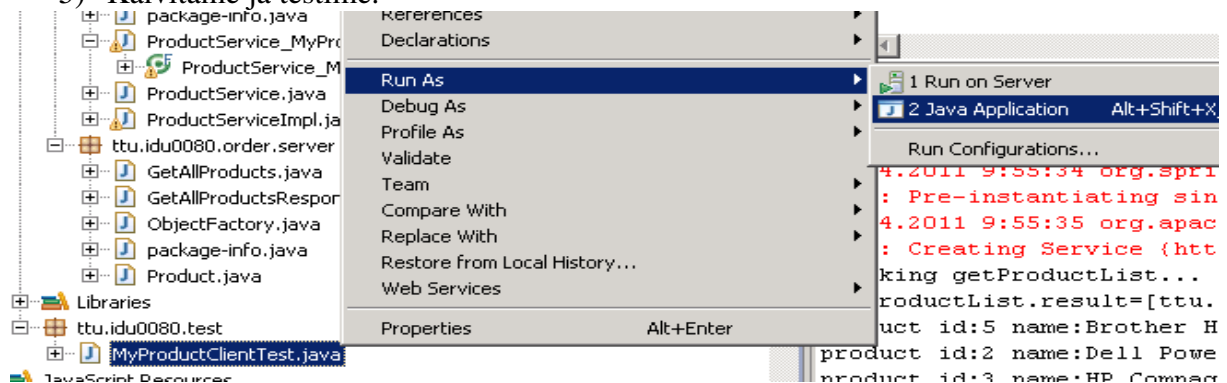
import java.util.List;
import ttu.idu0080.myproductserviceclient.*;
public class MyProductClientTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        ProductService_MyProductServiceImplPort_Client Client = new
ProductService_MyProductServiceImplPort_Client ();
        try {
            printProducts(Client.getProductsFromMyService());
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        public static void printProducts(List<ttu.idu0080.order.server.Product> products)
        {
            if (products != null)
            {
                for (ttu.idu0080.order.server.Product p: products) {
                    System.out.println("product id:" + p.getProduct() + " name:" + p.getName() );
                }
            }
        }
    }
}
-----

```

#### 5) Käivitame ja testime.

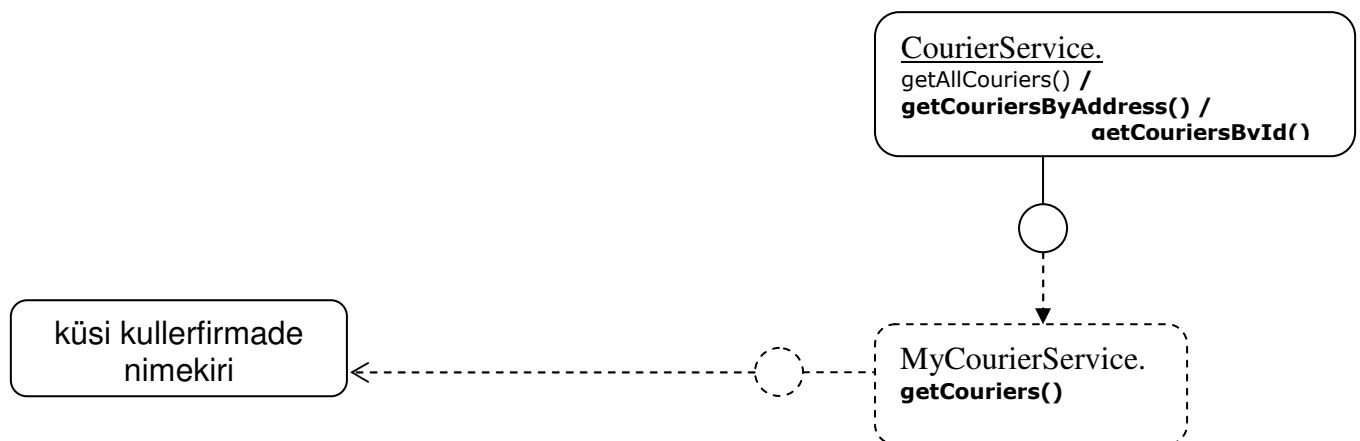




```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> MyProductClientTest [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
INFO: Overriding bean definition for bean 'org.apache.cxf.
23.04.2011 9:55:34 org.springframework.beans.factory.support
INFO: Pre-instantiating singletons in org.springframework.
23.04.2011 9:55:35 org.apache.cxf.service.factory.Reflecti
INFO: Creating Service {http://myproductservice.idu0080.tt
Invoking getProductList...
getProductList.result=[ttu.idu0080.order.server.Product@5e
product id:5 name:Brother HL 2035
product id:2 name:Dell PowerEdge T110
product id:3 name:HP Compaq 6005 Pro
product id:4 name:ML1660
product id:1 name:ProLiant ML110
product id:7 name:Samsung LE32S86BD
product id:6 name:ThinkCentre M58e
```

**MyProductServiceClient** on midagi sarnast sellele äriprotsessile kui veebteenuste kliendile mida peate tegema 5. koduülesandes. Koduülesandes peate selle äriprotsessi enda ka tegema kättesaadavaks veebteenusena , kuidas seda teha (ehk kuidas veebteenuse kliendist teha omakorda veebteenus) pole punkti 3 lugedes ilmselt enam keeruline aru saada.

Nii et selles näites sai põhimõtteliselt tehtud süsteem mis sarnaneb praktikaülesande alltoodud osaga:



- \* üks olemasolev teenus tund.ttü.ee peal
- \* üks ise-tehtud teenus oma arvutis mis on ise kliendiks tund.ttü.ee teenusele
- \* üks ise-tehtud teenusega ühendatud klient teise Eclipse projektina.