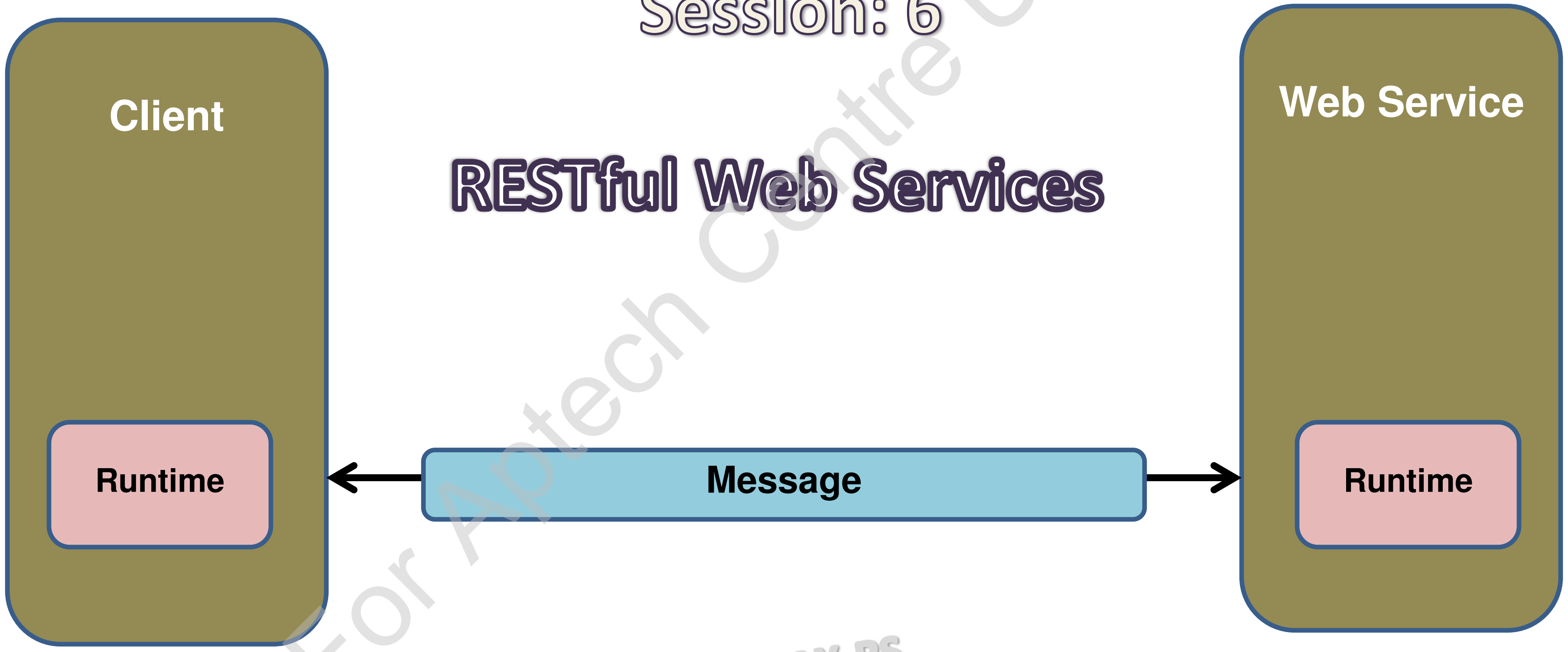
Developing Java Web Services



Session: 6

RESTful Web Services

Objectives

- Explain RESTful Web Services
- Describe JAX-RS API
- Explain how to build a RESTful Web Service with JAX-RS API

Introduction to RESTful Web Services

Web application development is based on HTTP protocol. HTTP is a network protocol used for data communication on the World Wide Web.

The other technologies that are used in Web application development are HTML, JavaScript, XML, and Asynchronous JavaScript and XML (AJAX).

Web services are Web applications based on XML standards and transport protocol, HTTP.

The development of Web services can be classified based on the technologies and the architectures available on the Web.

The two approaches used for designing the Web services are Standards-based approach and REST-based approach.

REST

REST is a set of guidelines or principles applied to the architectures available in a network system.

REST is an architecture-style on which systems are designed consisting of protocols, data components, hyperlinks, and clients.

The different constraints applied to the architectures based on the REST style are as follows:

- Client-Server
- Stateless
- Cache
- Layered components
- Uniform interface
- Code on demand

RESTful Web Services

RESTful Web services are Web services based on REST architecture and are accessed using HTTP protocol on the Web.

- RESTful Web services are similar to SOAP-based Web services.

The requirements for developing RESTful Web services based on the REST architecture style are as follows:

Resources	Representation of a Resource	URI	HTTP Methods

Resources

Resource can be defined as any kind of information exposed over the Web. Information can be a document, an image, or an entity such as a book or a flight detail from a database.

Examples of some RESTful Web services resources are as follows:

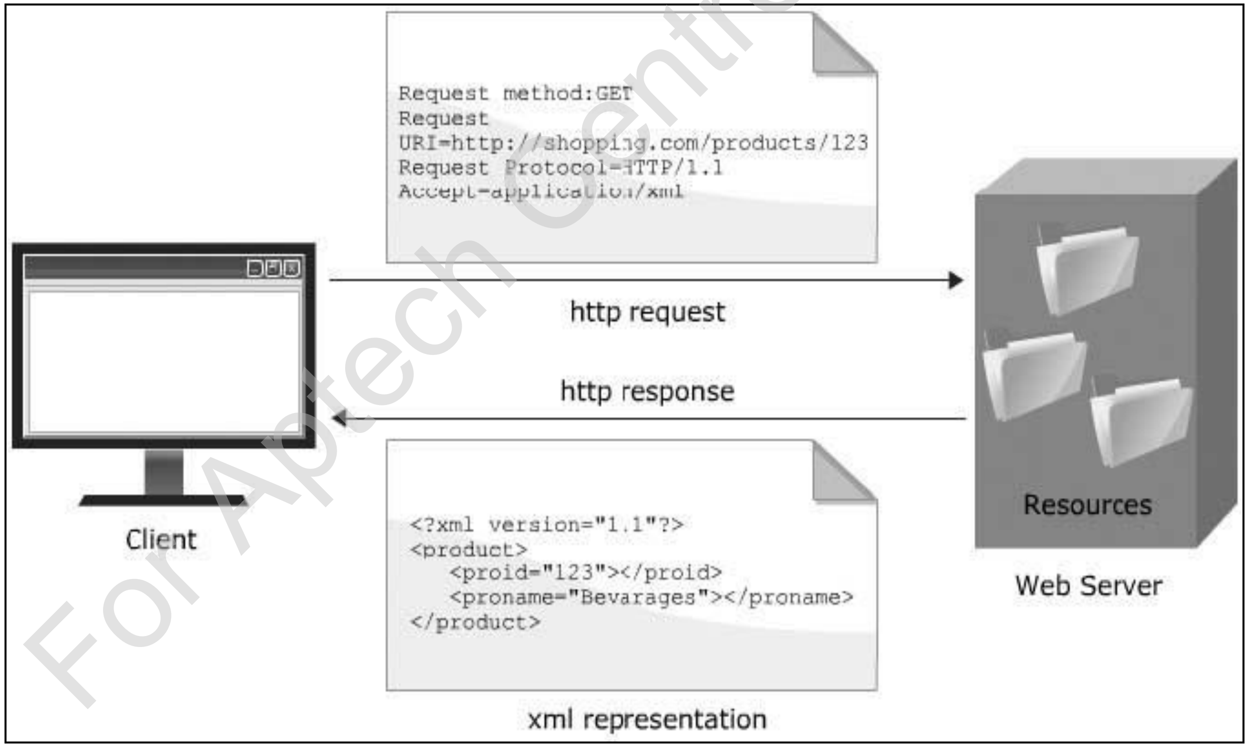
- Courses offered by a university
- A CNN news story board
- A search result for any article in the search engine
- Flight information stored in the flight database

Representation of a Resource

Representation can be defined as the format in which the resource is sent and received during client and server interaction on the Web.

The format of the resource representation, returned as a response to the client, represents the temporary state of the resource.

Following figure shows the handling of request and response of a RESTful Web service:



```
Request method:GET
Request
URI=http://shopping.com/products/123
Request Protocol=HTTP/1.1
Accept=application/xml
```

```
<?xml version="1.1"?>
<product>
   <proid="123"></proid>
   <proname="Bevarages"></proname>
</product>
```

URI

In the RESTful Web service, a resource is identified by a URI.

A URI is a combination of name and address of the resource, which helps the client and server to exchange data formats during interaction.

HTTP

HTTP is a stateless transport protocol used to exchange data over the Web. Web browsers support HTTP methods to send and receive data from the server.

Most commonly used HTTP methods for transferring data to the server in a request object are GET and POST. RESTful Web services support these HTTP methods for sending and receiving the data represented as a resource on the Web.

- HTTP methods are also known as verbs.
- The different types of actions are create, read, update, and delete, also known as CRUD actions.
- Following table describes the mapping of HTTP methods with their matching CRUD actions:

HTTP Methods	Action
POST	Create a new resource from the incoming data in the request
GET	Retrieve a resource
PUT	Update a resource from the incoming data in the request
DELETE	Delete a resource

JAX-RS API

JAX-RS specification is an official API on Java EE platform for creating Web services based on REST architecture. JAX-RS is a Java programming API, designed to simplify the development process of RESTful Web service using annotations.

JAX-RS API allows creating two types of resources, which are as follows:

- Root resource class
- Sub resources

JAX-RS API Annotations 1-7

JAX-RS API annotations simplify the development process of RESTful Web services. These annotations help to identify the resources and the actions to be performed on the identified resources.

Annotations available in the java.ws.rs package are categorized as follows:

- @Path annotation
- HTTP method annotations
- Annotations for injecting data from request URI
- Data representation type annotations

JAX-RS API Annotations 2-7

- **@Path Annotation**

The @Path annotation specifies the URI of the Web service class, hosted as a resource on the server. The @Path annotation has a single attribute which accepts a String value.

- Following code snippet demonstrates the use of @Path annotation with the URI template applied at class level to access the Web service class deployed on the Web server:

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.GET;
 //Sets the path to base URL + /Welcome
@Path("/welcome")
public class Welcome{
// This method is called if TEXT_PLAIN is requested
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sayPlainTextWelcome(){
return "Welcome to the Learning Portal";
}
}
```

JAX-RS API Annotations 3-7

- The URI template containing the embedded variables specified with the @Path annotation can be written as follows:

```
@Path("resourcePath/{param1}/…/{paramN}")
```

- Following code snippet demonstrates the URI template with the embedded variables to access the sub resource from the Web Service class on the Web server:

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.GET;
 //Sets the path to base URL + /Welcome
 @Path("/welcome/{username}")
 public class Welcome{
// This method is called if TEXT_PLAIN is requested
@GET
@Produces(MediaType.TEXT_PLAIN)
 public String sayPlainTextWelcome(@PathParam("username") String userName){
return "Welcome to the Learning Portal " + userName;
}
```

JAX-RS API Annotations 4-7

```
 @GET
@Path("students/{id}")
@Produces({"application/xml", "application/json"})
public Student find(@PathParam("id") String id) {
return super.find(id);
}
 }
```

- **HTTP Method Annotations:** The annotations in the javax.ws.rs packages corresponding to the equivalent HTTP methods such as GET, POST, PUT, and DELETE are as follows:

@GET annotation maps to the HTTP GET method

@POST annotation maps to the HTTP POST method

@PUT annotation maps to the HTTP PUT method

@DELETE annotation maps to the HTTP DELETE method

JAX-RS API Annotations 5-7

- **Injecting Data from Request URI Annotations** –extract the data from the request URI into the resource class.
- Following code snippet demonstrates the use of the @PathParam annotation on the parameter of a method of the Web service class present on the Web server:

```
@PUT
    @Path("{id}")
    @Consumes({"application/xml", "application/json"})
    public void edit(@PathParam("id") String id, Student entity){
        super.edit(entity);
    }
```

- Following code snippet demonstrates the use of @QueryParam annotation with method parameters in the Web service class:

```
...
@Path("/mess")
public String getMessage(@QueryParam("str")String studentName){
return "Welcome " + studentName + " ....";
}
...
```

JAX-RS API Annotations 6-7

- **Data Representation Type Annotations** –specifies the data type to be produced or consumed by the Web Service class.
- Following code snippet demonstrates the use of @Produces annotation applied to a method of the Web service class:

```
@GET
    @Override
    @Produces({"application/xml", "application/json"})
    public List<Student> findAll() {
            return super.findAll();
    }
```

- Following code snippet demonstrates the use of @Consumes annotation applied at the method level of the Web service class:

```
@POST
    @Override
    @Consumes({"application/xml", "application/json"})
    public void create(Student entity){
          super.create(entity);
    }
```

JAX-RS API Annotations 7-7

- Following code snippet demonstrates the use of @Provider annotation applied at the method level of the Web service class:

```
//Use of @Provider annotation in MessageBodyReader
 @Consumes("application/x-www-form-urlencoded")
@Provider
public class FormReader implements MessageBodyReader<NameValuePair> {
.....
....
.....
}
//Use of @Provider annotation in MessageBodyWriter
@Produces("text/html")
@Provider
public class FormWriter implements
  MessageBodyWriter<Student<String studentName, String studentGrade>>{
....
....
}
```

Building RESTful Web Service with JAX-RS API

The steps to design and build a RESTful Web service using JAX-RS API are as follows:

1. Code the implementation class which will act as a root resource class
2. Define the methods within the main implementation class that act as sub resources
3. Annotate the class with JAX-RS annotations
4. Build, package, and deploy the files on the application server
5. Access the resource in the client browser application

RESTful Web Implementation Class 1-5

The RESTful implementation class is a resource which can be accessed by the client through a URI.

Criteria to be followed by a class to behave as a root resource class:

- The RESTful implementation class must be annotated with @Path, which specifies it as a base URI of the resource implemented by the service.
- The implementation class must have a public constructor so that it can be invoked by the JAX-RS runtime.
- The HTTP methods accepts different types of request such as GET, POST, PUT, or DELETE for handling various types of requests.

RESTful Web Implementation Class 2-5

- Following code snippet demonstrates implementation of the StudentFacadeREST Web service class:

```
package com.syskan.studentdb.entities.service;
import com.syskan.studentdb.entities.Student;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
@Stateless
@Path("com.syskan.studentdb.entities.student")
public class StudentFacadeREST extends AbstractFacade<Student>{
@PersistenceContext(unitName = "com.syskan_StudentDB_war_1.0-SNAPSHOTPU")
private EntityManager em;
```

RESTful Web Implementation Class 3-5

```
public StudentFacadeREST() {
super(Student.class);
}
@POST
@Override
@Consumes({"application/xml", "application/json"})
public void create(Student entity) {
super.create(entity);
}
@PUT
@Path("{id}")
@Consumes({"application/xml", "application/json"})
public void edit(@PathParam("id") Integer id, Student entity) {
super.edit(entity);
}
@DELETE
@Path("{id}")
public void remove(@PathParam("id") Integer id) {
super.remove(super.find(id));
}
```

RESTful Web Implementation Class 4-5

```
@GET
@Path("{id}")
@Produces({"application/xml", "application/json"})
public Student find(@PathParam("id") Integer id){
return super.find(id);
}


@GET
@Override
@Produces({"application/xml", "application/json"})
public List<Student> findAll() {
return super.findAll();
}
@GET
@Path("{from}/{to}")
@Produces({"application/xml", "application/json"})
public List<Student> findRange(@PathParam("from") Integer from,
@PathParam("to") Integer to) {
return super.findRange(new int[]{from, to});
}
```

RESTful Web Implementation Class 5-5

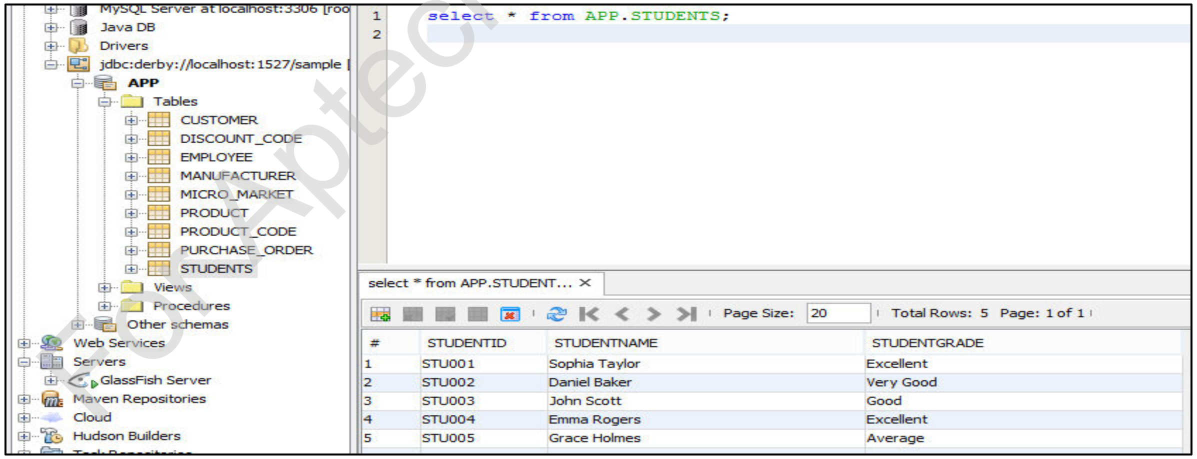
```
@GET
@Path("count")
@Produces("text/plain")
public String countREST() {
return String.valueOf(super.count());
}
@Override
protected EntityManager getEntityManager(){
return em;
 }
}
```

Build, Package, and Deploy the RESTful Web Service 1-8

To create a RESTful Web service based on a database table, perform the following steps:

1. In the NetBeans IDE, create a new Web application project. To do this, click File → New and then in the New Project wizard, select Java Web from the Categories list and Web Application from the Projects list.
2. Specify a name and location for the project.
3. Ensure that GlassFish Server is selected and the Context Path is set.
4. To create a RESTful Web service in the Web application project, right-click the project node and then click New → Other.
5. In the New File wizard, select Web Services from the Categories list and RESTful Web Services from Database from the File Types list.

- Following figure shows a STUDENTS table:



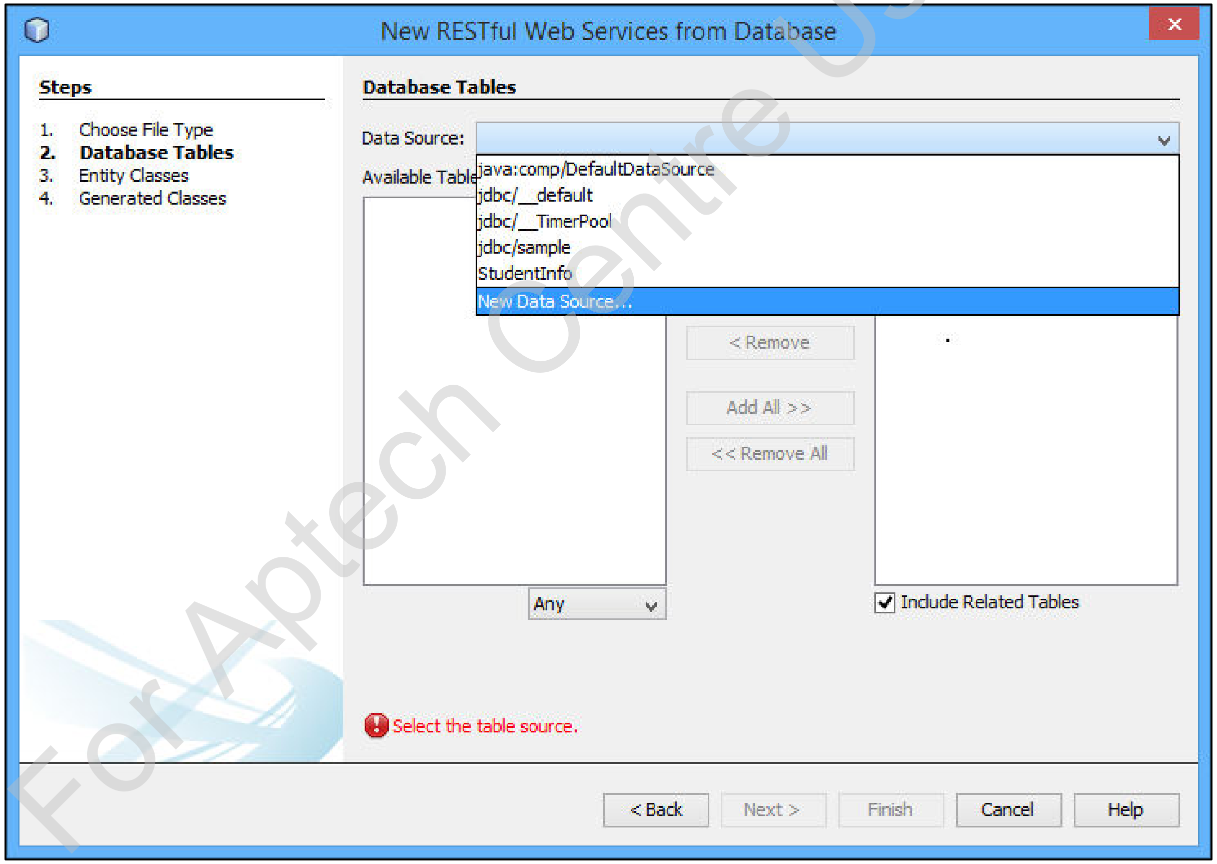
```
select * from APP.STUDENTS;
```

#	STUDENTID	STUDENTNAME	STUDENTGRADE
1	STU001	Sophia Taylor	Excellent
2	STU002	Daniel Baker	Very Good
3	STU003	John Scott	Good
4	STU004	Emma Rogers	Excellent
5	STU005	Grace Holmes	Average

Build, Package, and Deploy the RESTful Web Service 2-8

6. In the New RESTful Web Service from Database dialog box, from the Data Source drop-down list, select New Data Source.

- Following figure shows how to create a new data source:

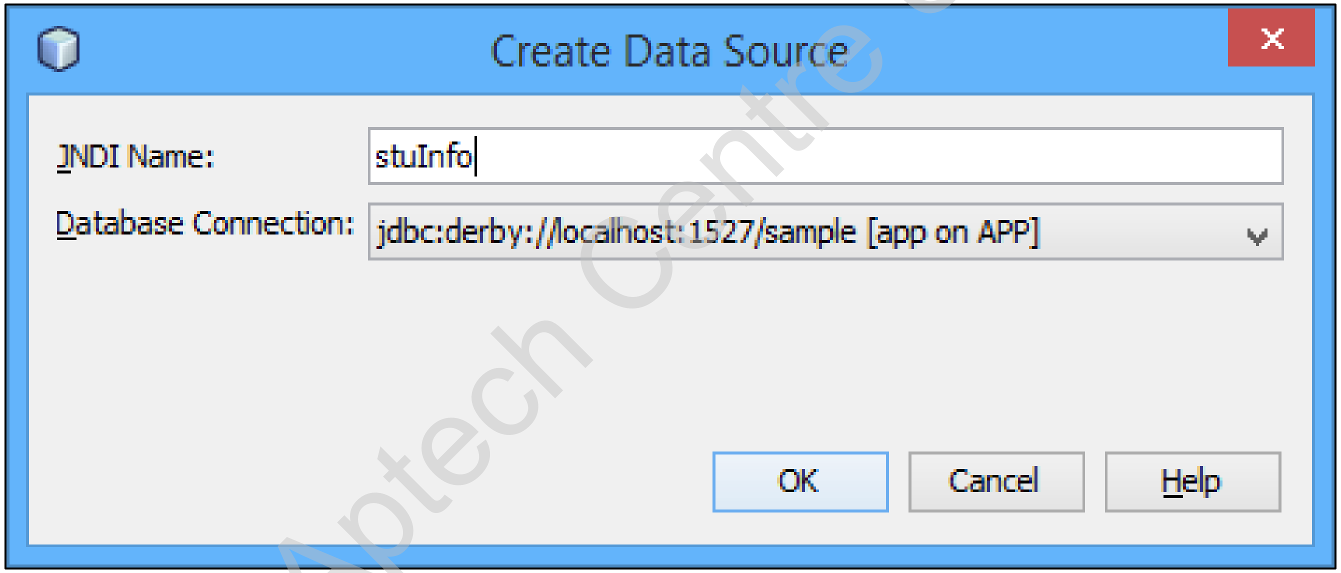


Select the table source.

Build, Package, and Deploy the RESTful Web Service 3-8

7. In the Create Data Source dialog box, in the Database Connection drop-down list, select jdbc:derby://localhost:1527/sample and specify a JNDI name for the data source.

- Following figure shows how to name the new data source:



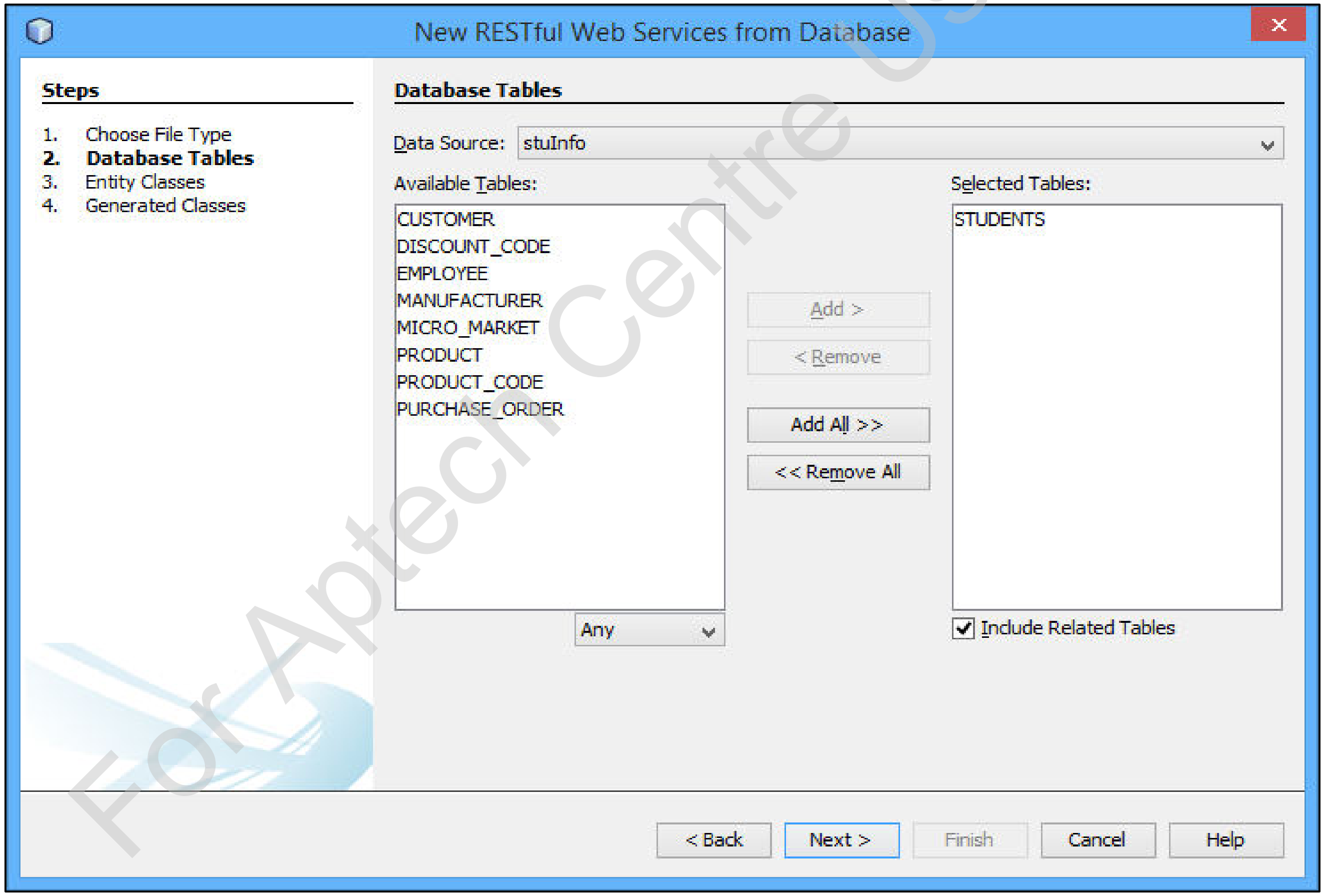
JNDI Name: stuInfo

Database Connection: jdbc:derby://localhost:1527/sample [app on APP]

Build, Package, and Deploy the RESTful Web Service 4-8

8. Select the Students database and click the Add button. The Students database is added to the Selected Tables list.

- Following figure shows how to select the database table:



New RESTful Web Services from Database

Steps

1. Choose File Type
2. **Database Tables**
3. Entity Classes
4. Generated Classes

Database Tables

Data Source: stuInfo

Available Tables:
CUSTOMER
DISCOUNT_CODE
EMPLOYEE
MANUFACTURER
MICRO_MARKET
PRODUCT
PRODUCT_CODE
PURCHASE_ORDER

Selected Tables:
STUDENTS

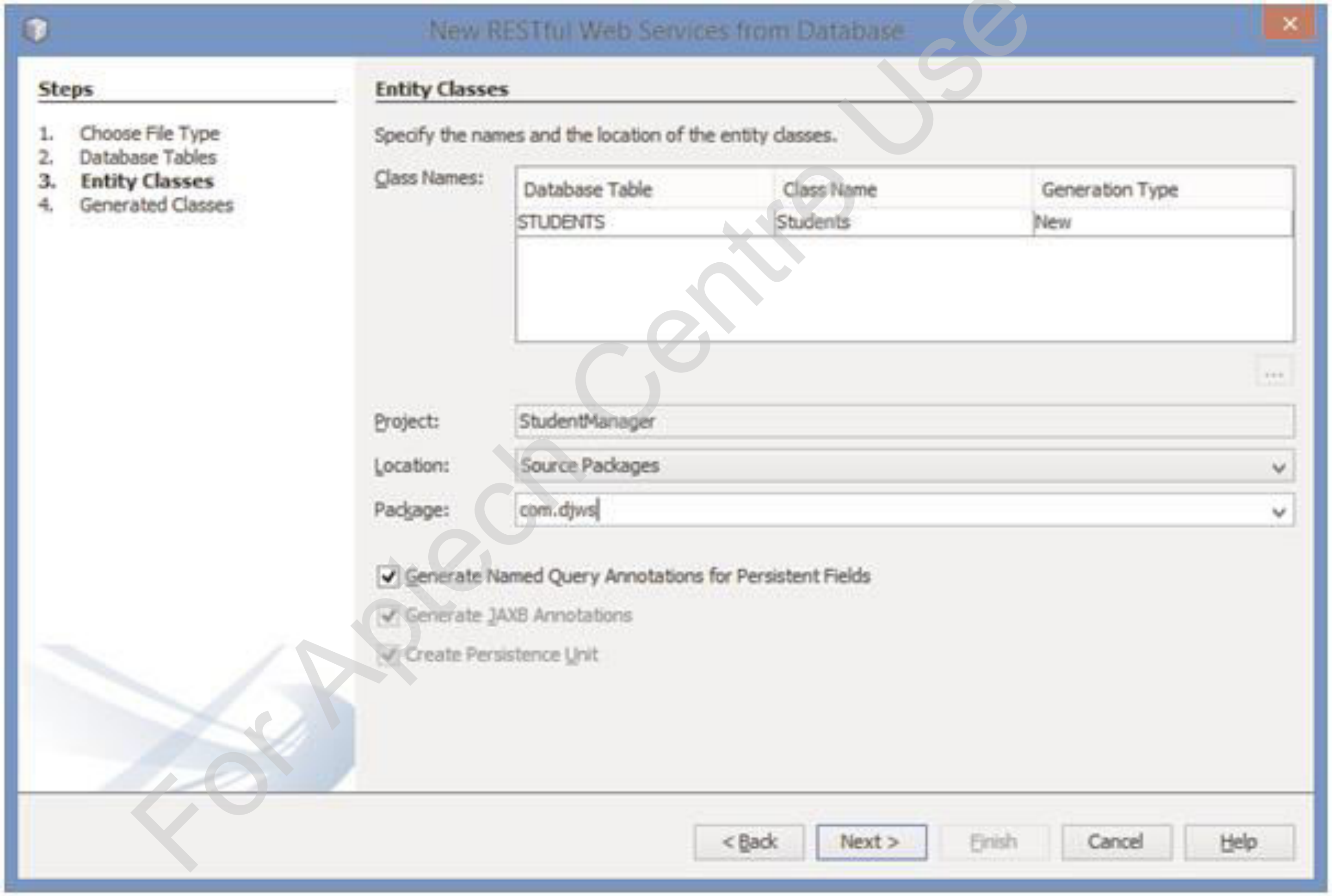
Any

[✓] Include Related Tables

Build, Package, and Deploy the RESTful Web Service 5-8

9. Specify a package name for the Web service.

- Following figure shows how to specify the package name:



Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
STUDENTS	Students	New

Project: StudentManager

Location: Source Packages

Package: com.djws

[✓] Generate Named Query Annotations for Persistent Fields

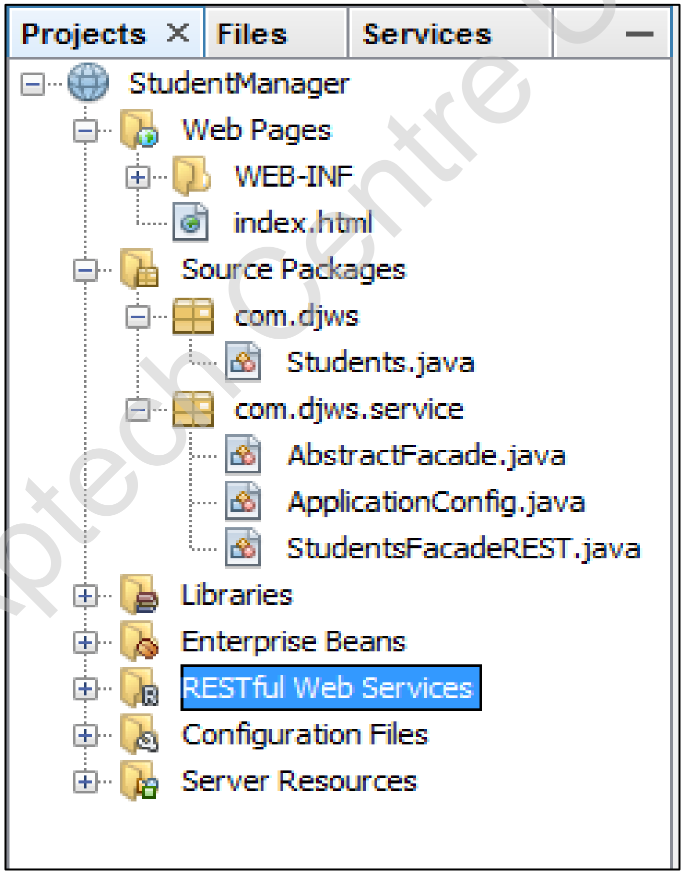
[✓] Generate JAXB Annotations

[✓] Create Persistence Unit

Build, Package, and Deploy the RESTful Web Service 6-8

10. On the next page of the wizard, accept the defaults and click Finish. The IDE generates the RESTful Web service. The generated entity classes are listed in the com.djws package and the services are listed in the com.djws.service package.

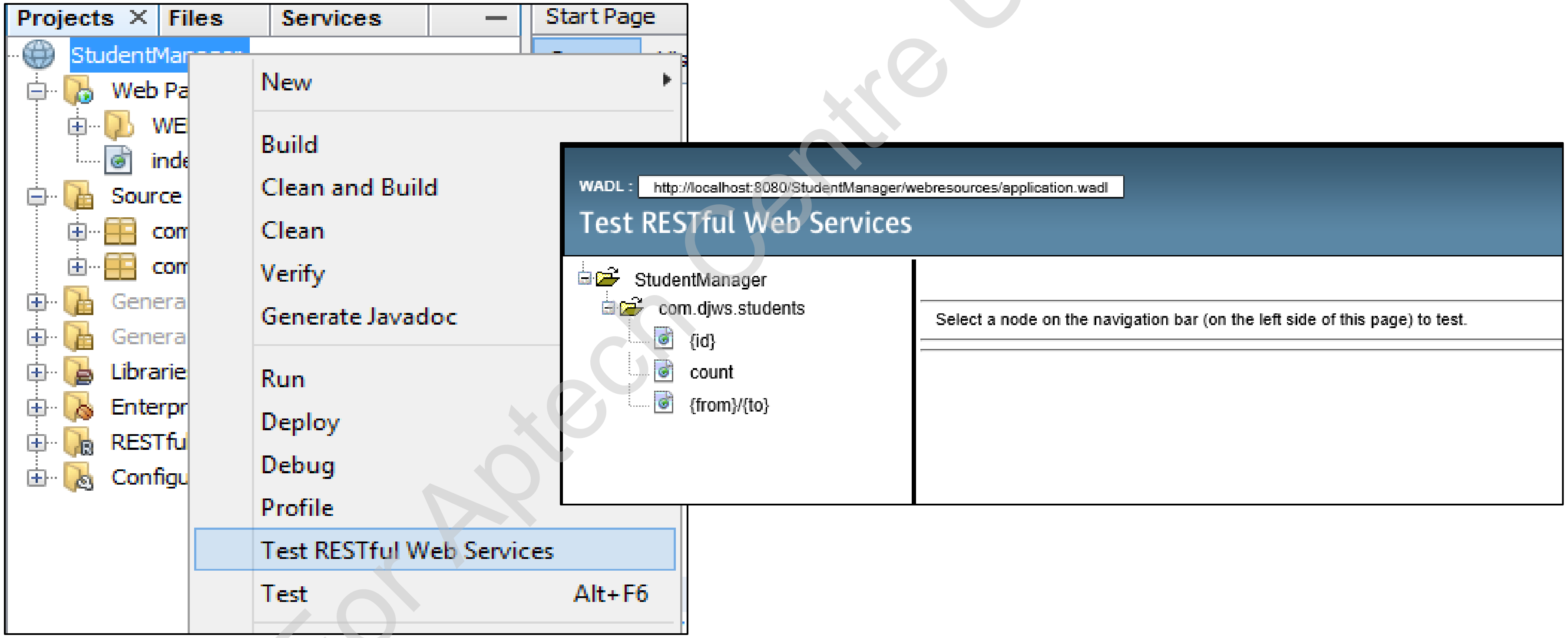
- Following figure shows entity classes and services:



Build, Package, and Deploy the RESTful Web Service 7-8

11. To test the Web service, right-click StudentManager project, and then click Test RESTful Web Services. The Web service opens in the browser.

- Following figure shows Test RESTful Web Services:

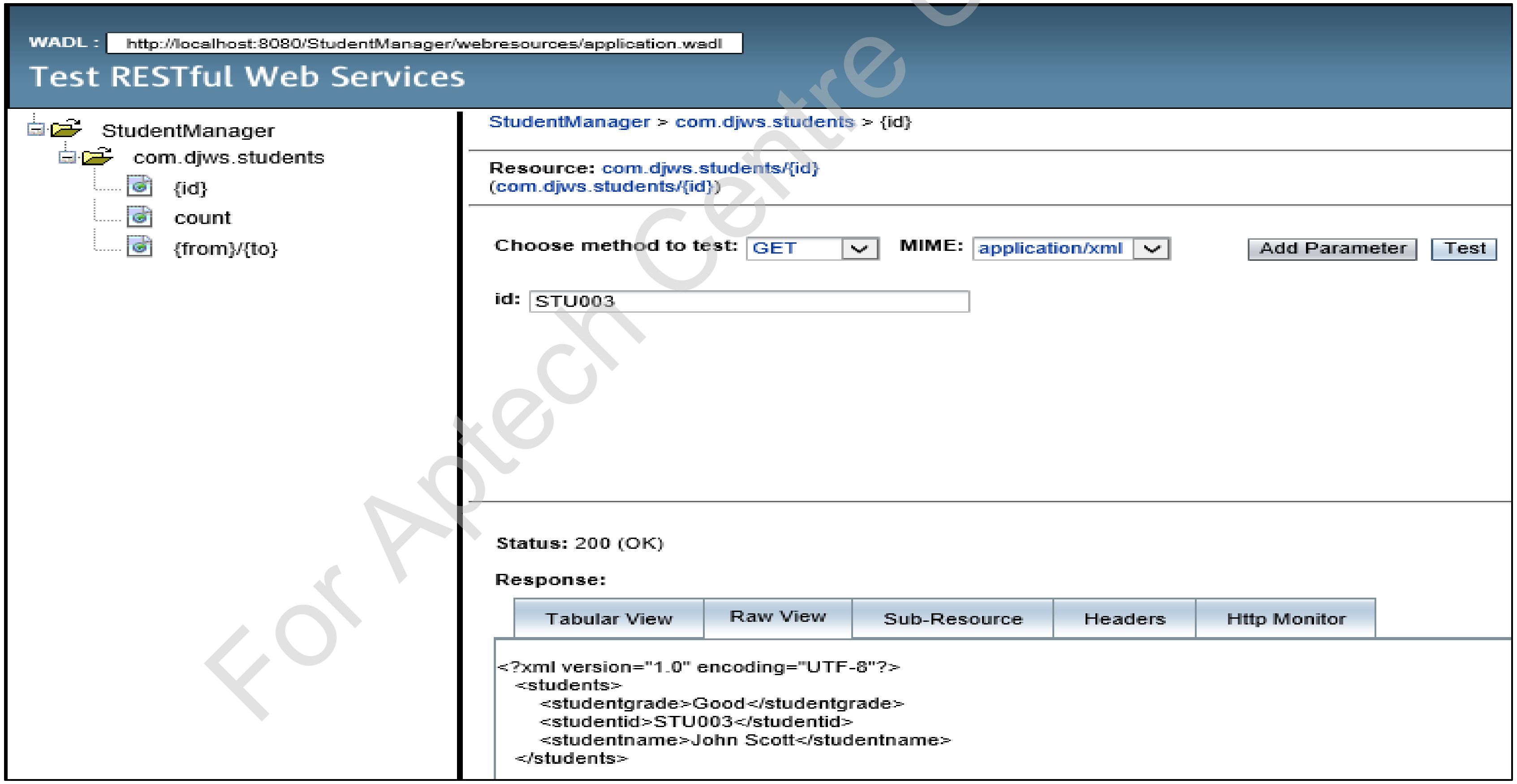


Select a node on the navigation bar (on the left side of this page) to test.

Build, Package, and Deploy the RESTful Web Service 8-8

12. To find the details of a specific student, in the left pane, click {id}.
13. In the right pane, in the id box, enter the ID of the student and then, click Test. The details of the student whose ID was specified is displayed.

- Following figure shows the output of the RESTful Web Service:


WADL : http://localhost:8080/StudentManager/webresources/application.wadl
Test RESTful Web Services
StudentManager
com.djws.students
{id}
count
{from}/{to}
StudentManager > com.djws.students > {id}
Add Parameter
Test
Tabular View
Raw View
Sub-Resource
Headers
Http Monitor

Resource: com.djws.students/{id}
(com.djws.students/{id})

Choose method to test: GET

MIME: application/xml

id: STU003

Status: 200 (OK)

Response:

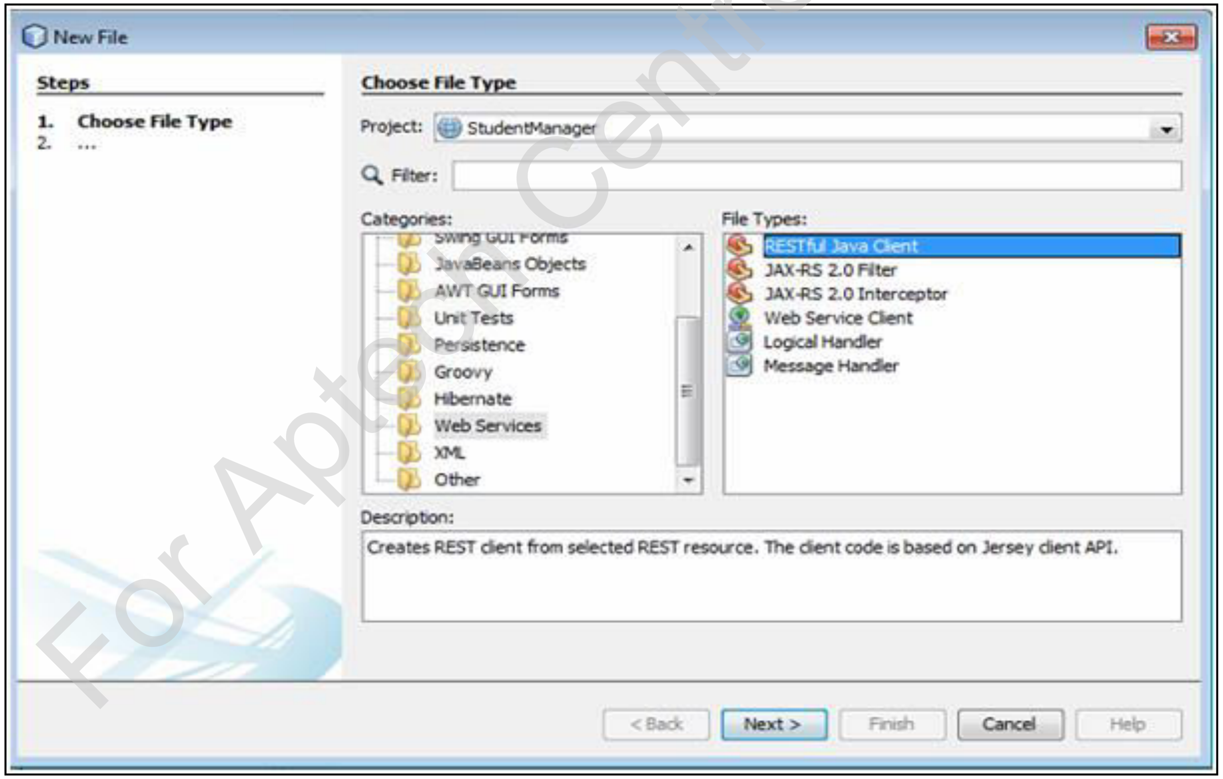
```
<?xml version="1.0" encoding="UTF-8"?>
  <students>
     <studentgrade>Good</studentgrade>
     <studentid>STU003</studentid>
     <studentname>John Scott</studentname>
  </students>
```

Implementing a Web Client 1-14

To create a Web client for the RESTful Web service, perform the following steps:

1. To create a Java client for the RESTful Web service in the Java application project, right-click the project node and then, click New → Other. In the New File wizard, select Web Services from the Categories list and RESTful Java Client from the File Types list.

- Following figure shows how to choose a RESTful Java Client:



New File

Steps
1. Choose File Type
2. ...

Choose File Type

Project: StudentManager

Filter:

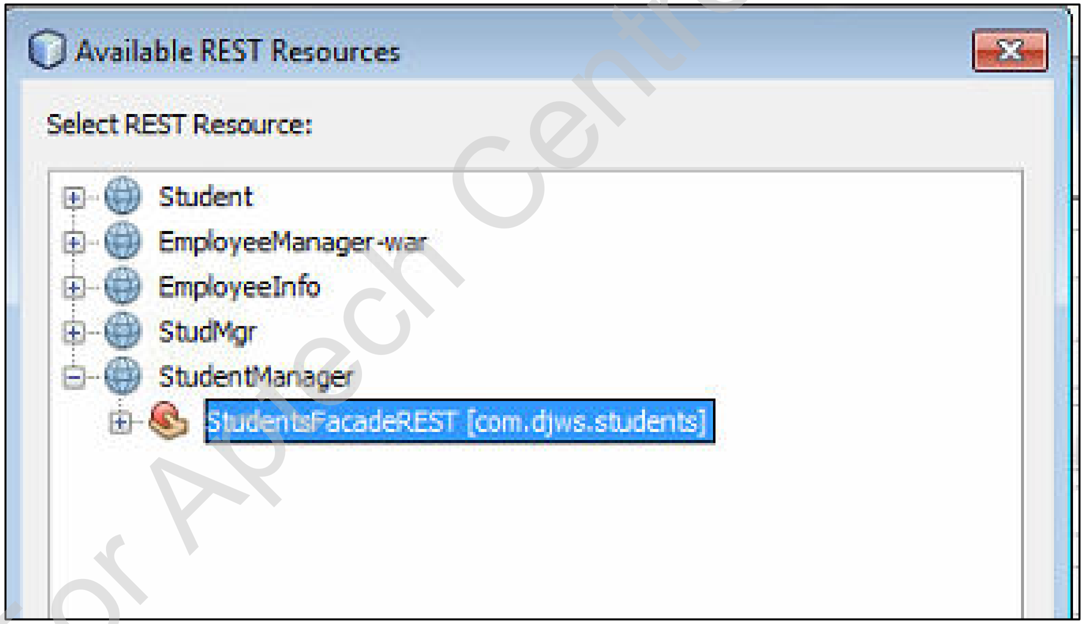
Categories: Swing GUI Forms, JavaBeans Objects, AWT GUI Forms, Unit Tests, Persistence, Groovy, Hibernate, Web Services, XML, Other

File Types: RESTful Java Client, JAX-RS 2.0 Filter, JAX-RS 2.0 Interceptor, Web Service Client, Logical Handler, Message Handler

Description:
Creates REST client from selected REST resource. The client code is based on Jersey client API.

Implementing a Web Client 2-14

2. Specify a name for the client.
3. To select the REST resource to be used for the client, under Select REST Resource, ensure that the From Project option is selected and then, click Browse.
4. In the Available REST Resources dialog box, select the appropriate REST Web service as shown in the following figure:


Available REST Resources

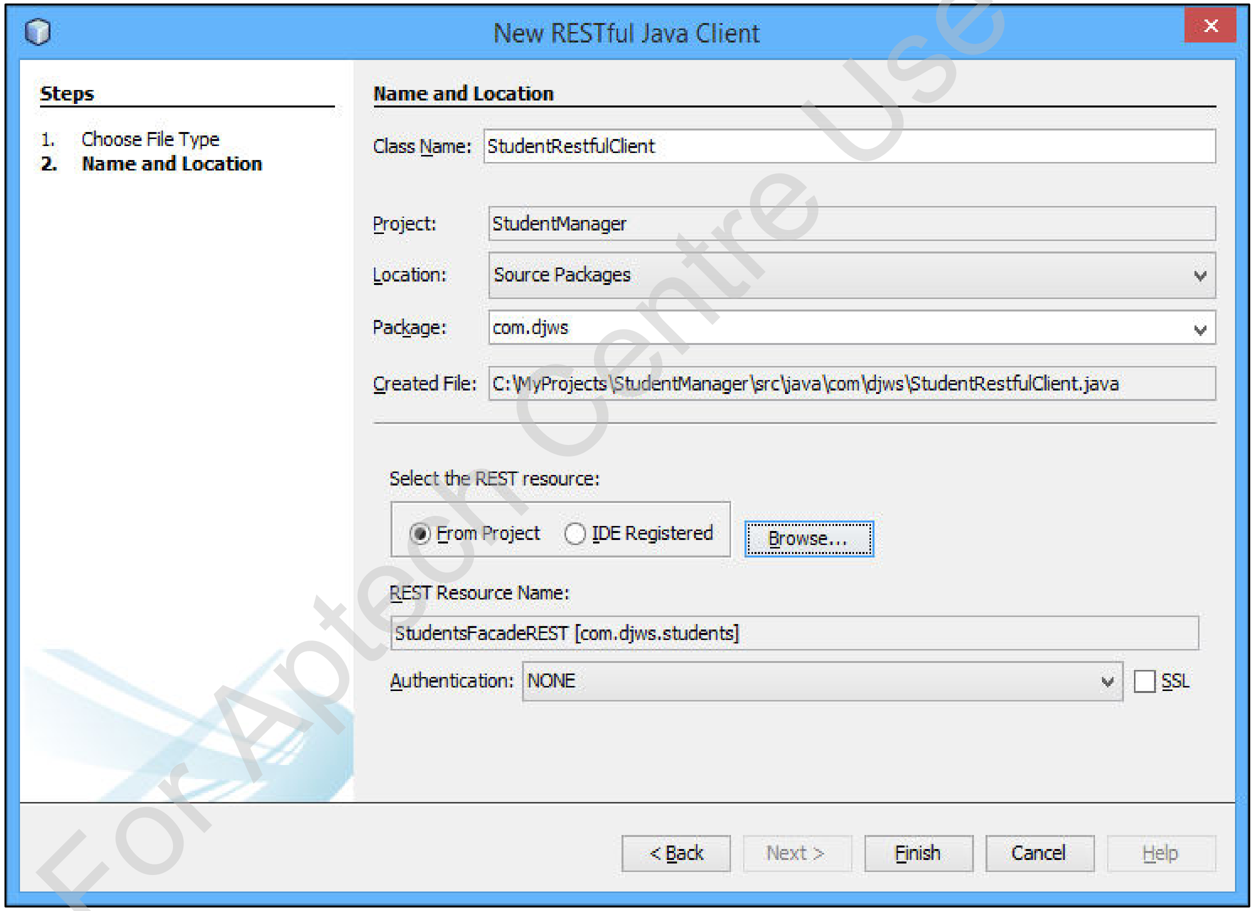
Select REST Resource:

- Student
- EmployeeManager-war
- EmployeeInfo
- StudMgr
- StudentManager
  - StudentsFacadeREST [com.djws.students]

Implementing a Web Client 3-14

5. Select the package for the client.

- Following figure shows how to specify package for Java client:



Class Name: StudentRestfulClient

Project: StudentManager

Location: Source Packages

Package: com.djws

Created File: C:\MyProjects\StudentManager\src\java\com\djws\StudentRestfulClient.java

Select the REST resource: [✓ From Project] IDE Registered

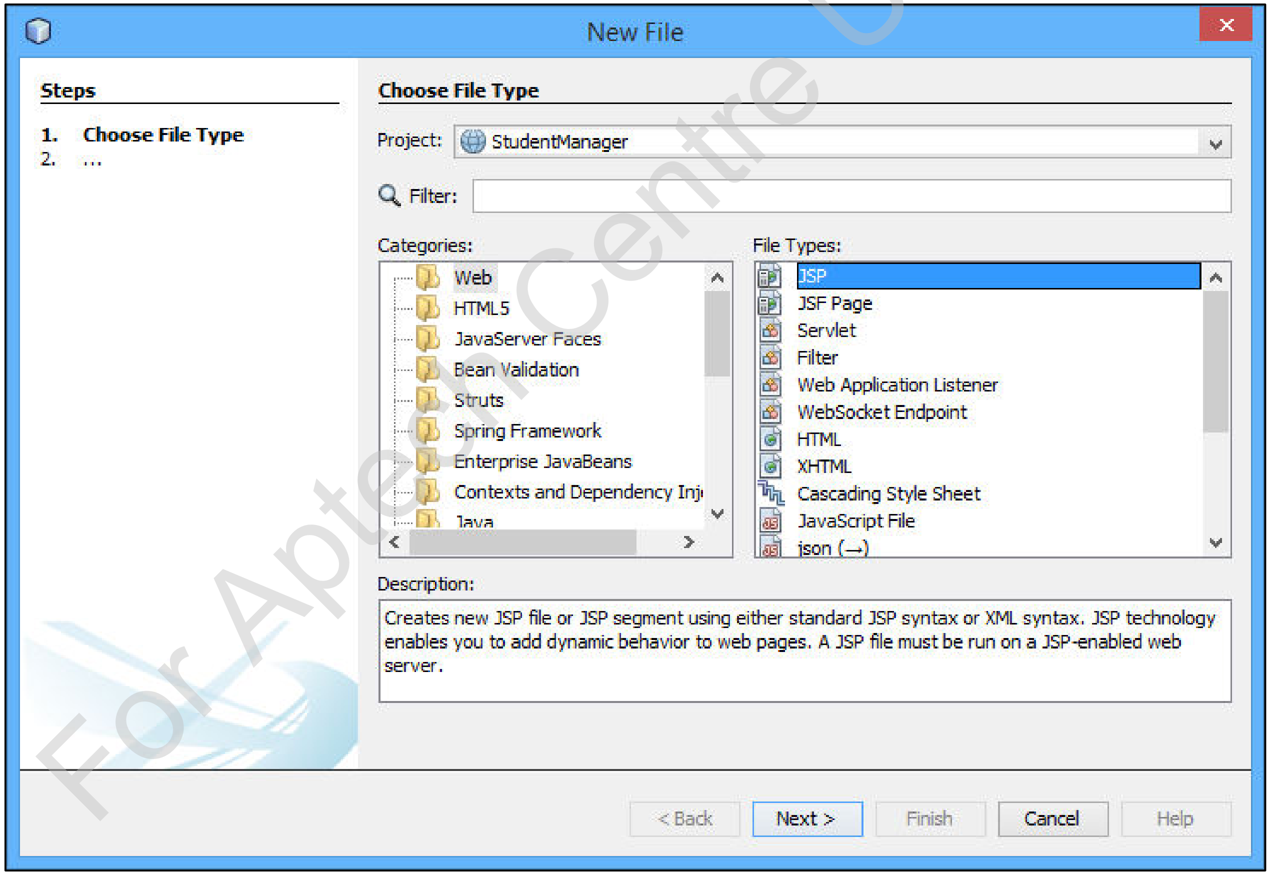
REST Resource Name: StudentsFacadeREST [com.djws.students]

Authentication: NONE SSL

Implementing a Web Client 4-14

6. To create a JSP file for the client, right-click the StudentManager project and then, click New → Other. Then, in the New File wizard, select Web from the Categories list and JSP from the File Types list.

- Following figure shows how to create a JSP file:


New File

Steps
1. Choose File Type
2. ...

Choose File Type

Project: StudentManager

Filter:

Categories:
- Web
- HTML5
- JavaServer Faces
- Bean Validation
- Struts
- Spring Framework
- Enterprise JavaBeans
- Contexts and Dependency Inj
- Java

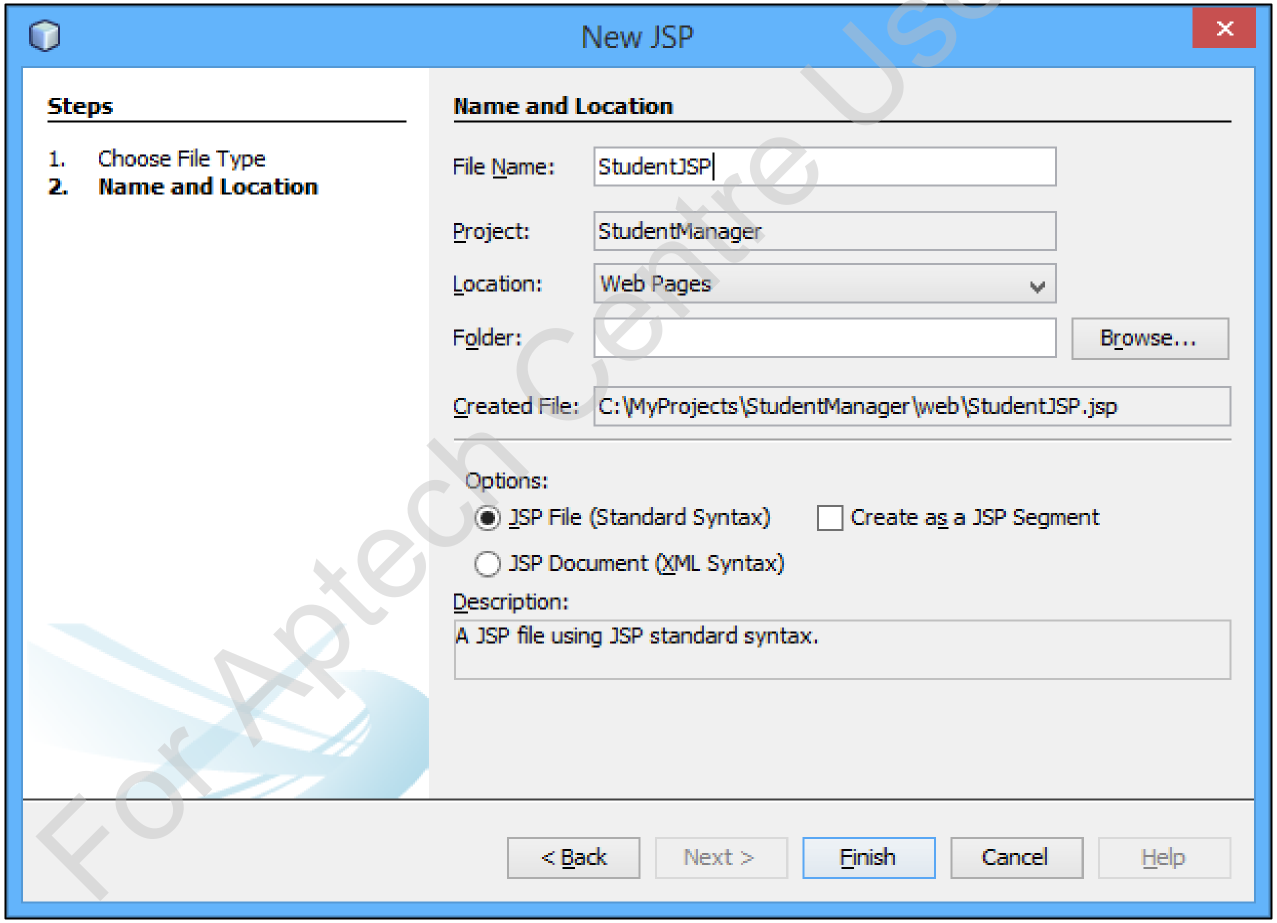
File Types:
- JSP
- JSF Page
- Servlet
- Filter
- Web Application Listener
- WebSocket Endpoint
- HTML
- XHTML
- Cascading Style Sheet
- JavaScript File
- json (→)

Description:
Creates new JSP file or JSP segment using either standard JSP syntax or XML syntax. JSP technology enables you to add dynamic behavior to web pages. A JSP file must be run on a JSP-enabled web server.

Implementing a Web Client 5-14

7. Specify a name for the JSP file.

- Following figure shows how to specify a name for the JSP file:



New JSP

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

File Name:	StudentJSP
Project:	StudentManager
Location:	Web Pages
Folder:	
Created File:	C:\MyProjects\StudentManager\web\StudentJSP.jsp

Options:

[✓ JSP File (Standard Syntax)] Create as a JSP Segment

JSP Document (XML Syntax)

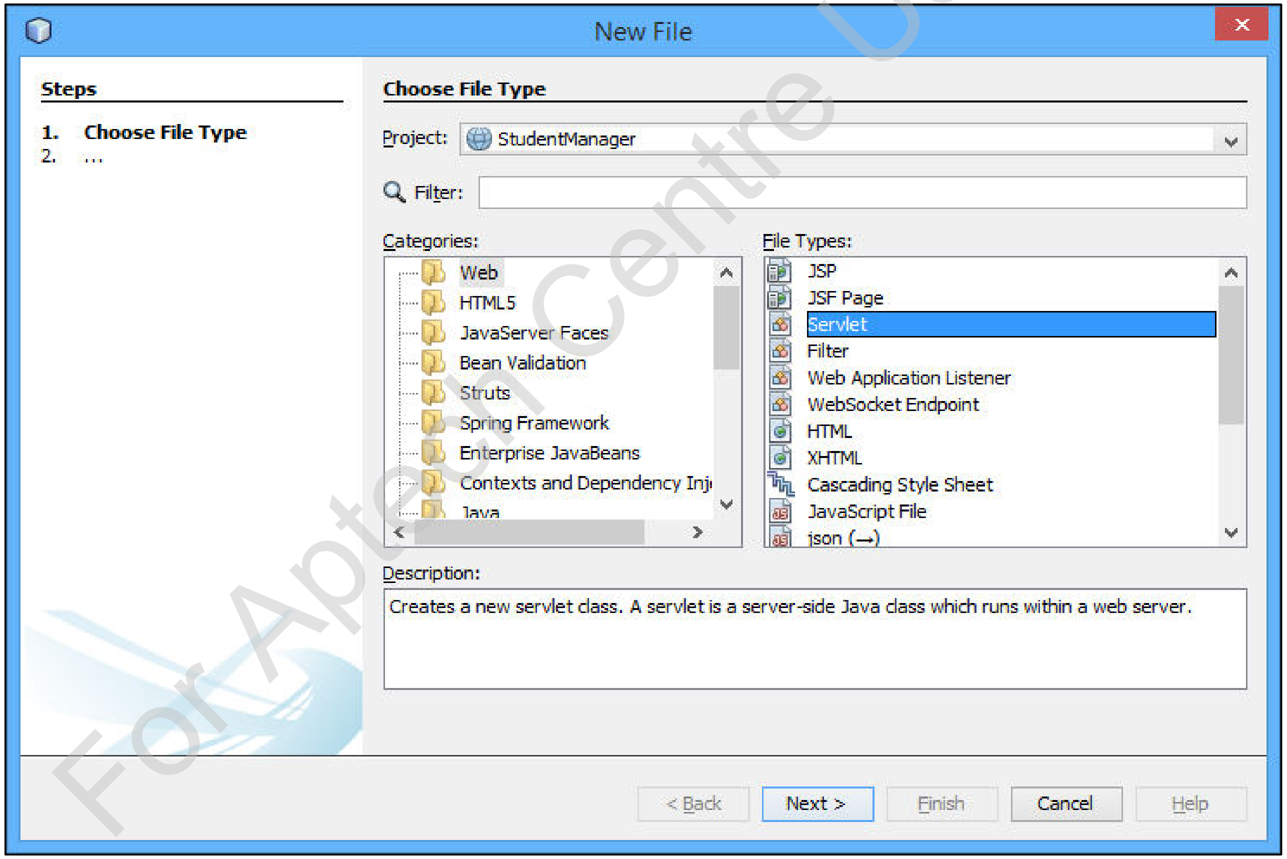
Description:

A JSP file using JSP standard syntax.

Implementing a Web Client 6-14

8. To create a Servlet for the client, right-click the StudentManager project and then, click New → Other. Then, in the New File wizard, select Web from the Categories list and Servlet from the File Types list.

- Following figure shows how to create a servlet:



Project: StudentManager

Filter:

Categories: Web, HTML5, JavaServer Faces, Bean Validation, Struts, Spring Framework, Enterprise JavaBeans, Contexts and Dependency Inj, Java

File Types: JSP, JSF Page, Servlet, Filter, Web Application Listener, WebSocket Endpoint, HTML, XHTML, Cascading Style Sheet, JavaScript File, json (→)

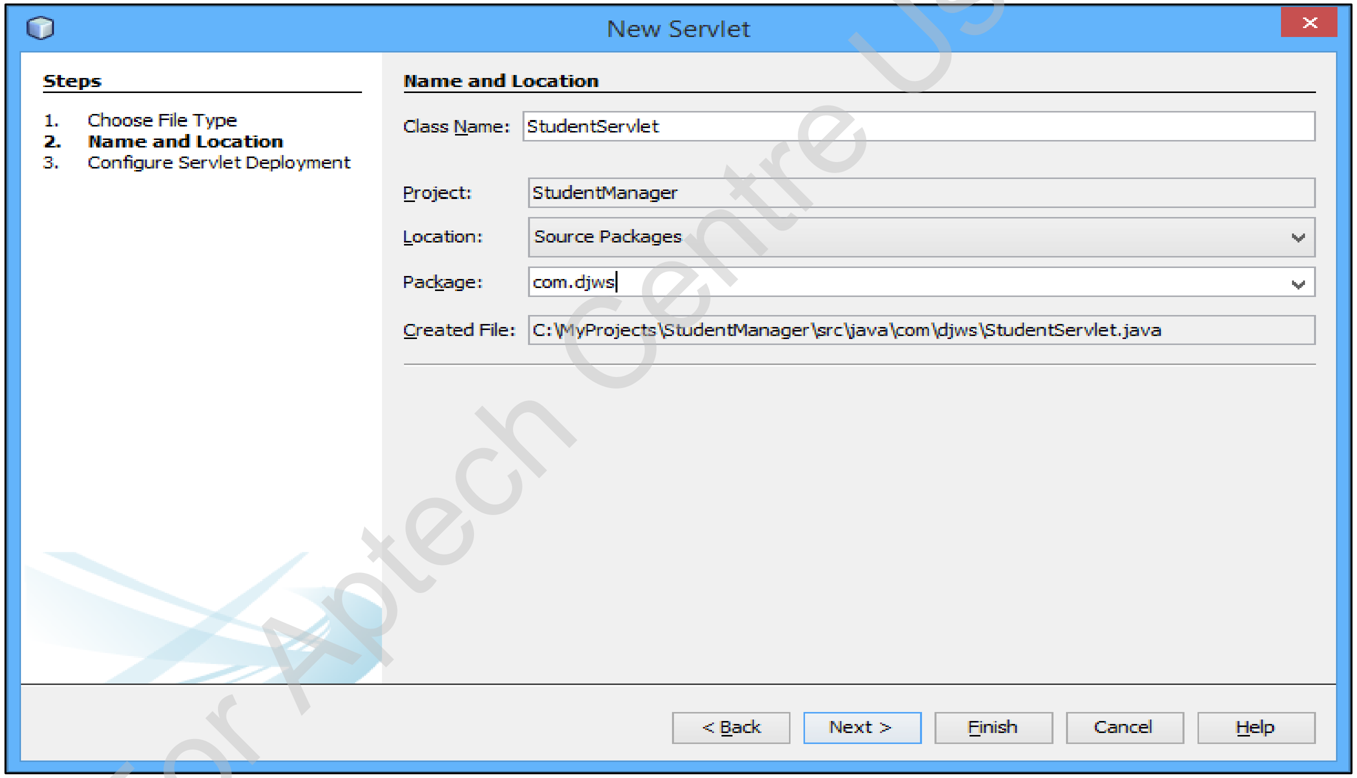
Description:

Creates a new servlet class. A servlet is a server-side Java class which runs within a web server.

Implementing a Web Client 7-14

9. Specify a name and package for the servlet.

- Following figure shows how to specify a name for servlet:



Steps

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

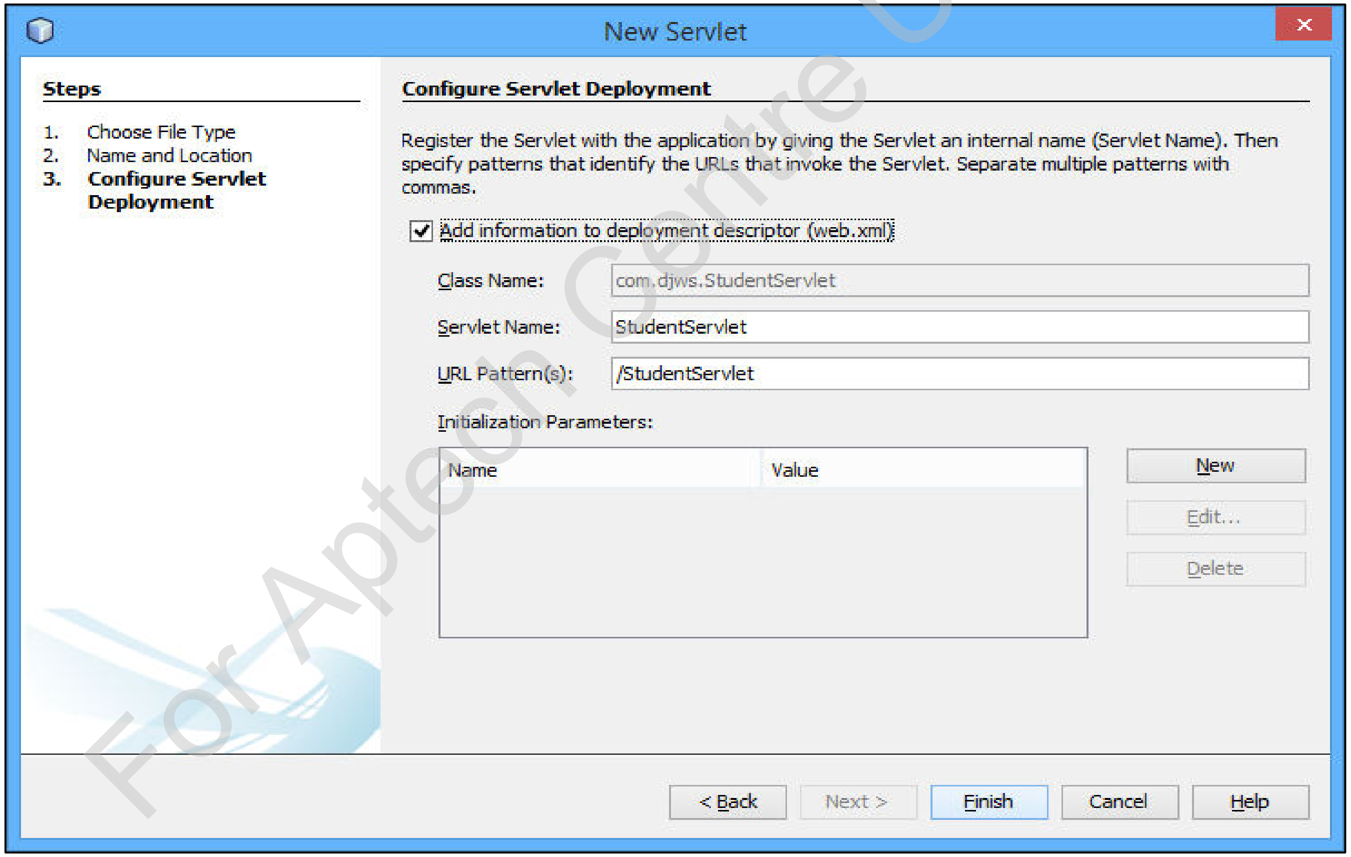
Name and Location

Class Name:	StudentServlet
Project:	StudentManager
Location:	Source Packages
Package:	com.djws
Created File:	C:\MyProjects\StudentManager\src\java\com\djws\StudentServlet.java

Implementing a Web Client 8-14

10. To add the servlet information to the web.xml page, select the Add information to deployment descriptor (web.xml) check box.

- Following figure shows how to add servlet information to web.xml:



Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

[✓] Add information to deployment descriptor (web.xml)

Class Name: com.djws.StudentServlet

Servlet Name: StudentServlet

URL Pattern(s): /StudentServlet

Initialization Parameters:

Name	Value

Implementing a Web Client 9-14

11. To add a text box and a Submit button to the JSP file, update the code as shown in the following code snippet:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <form action ="StudentServlet">

            <h1>Student Details</h1>
            Enter Student ID: <input type="text" name="txtID" value="" />
            <input type="submit" value="Submit" name="Submit" />
        </form>
    </body>
</html>
```

Implementing a Web Client 10-14

12. To use the Web service to retrieve the student information from the database and display it on the Web page, update the code in the StudentServlet file as shown in the following code snippet:

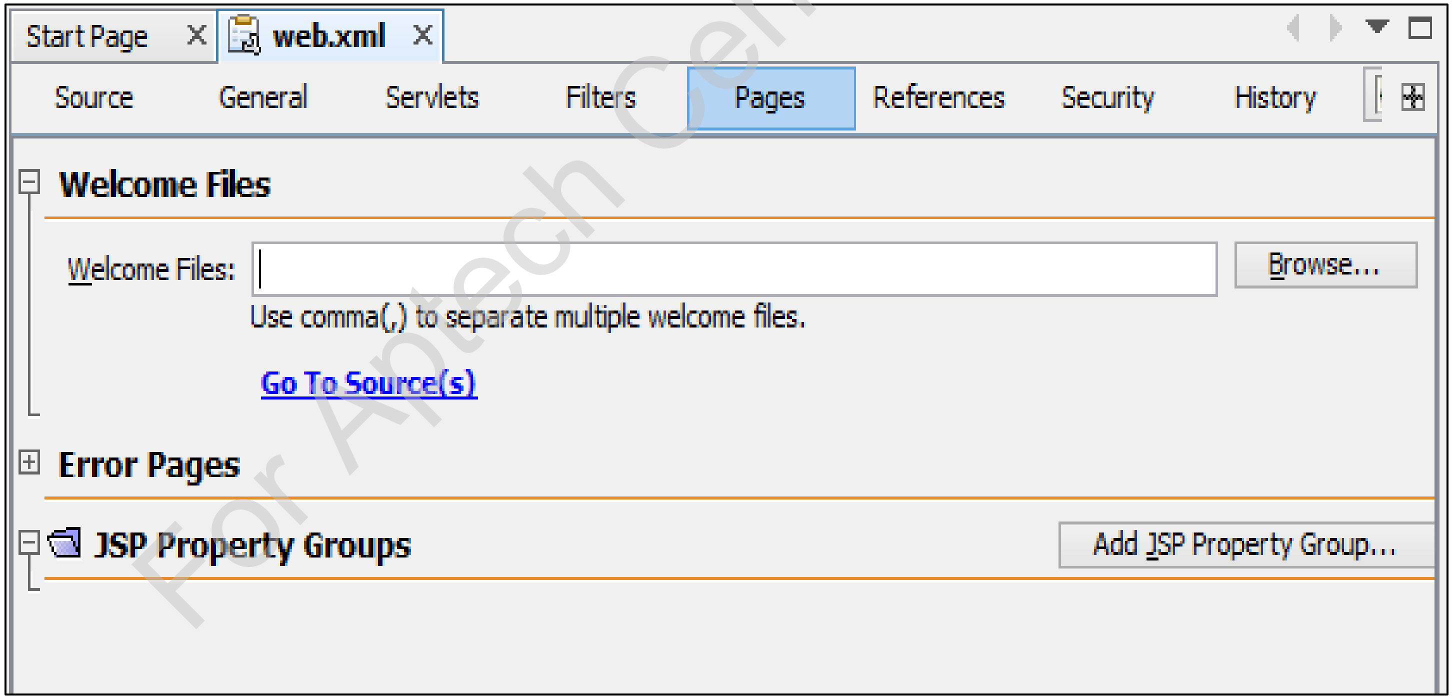
```
...
protected void processRequest(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet StudentServlet</title>");
            out.println("</head>");
            out.println("<body>");
            String id = request.getParameter("txtID");
            StudentRestfulClient stuInfo = new StudentRestfulClient();
Students stu = stuInfo.find_XML(Students.class, id);
out.println("Student ID: " + stu.getStudentid()+ "<br/>");
out.println("Student Name: " + stu.getStudentname()+ "<br/>");
out.println("Student Grade: " + stu.getStudentgrade());
```

Implementing a Web Client 11-14

```
stuInfo.close();
out.println("</body>");
out.println("</html>");
}
}
```

13. To specify the startup file for the project, open web.xml and then, click the Pages tab.

- Following figure shows Pages tab of web.xml:



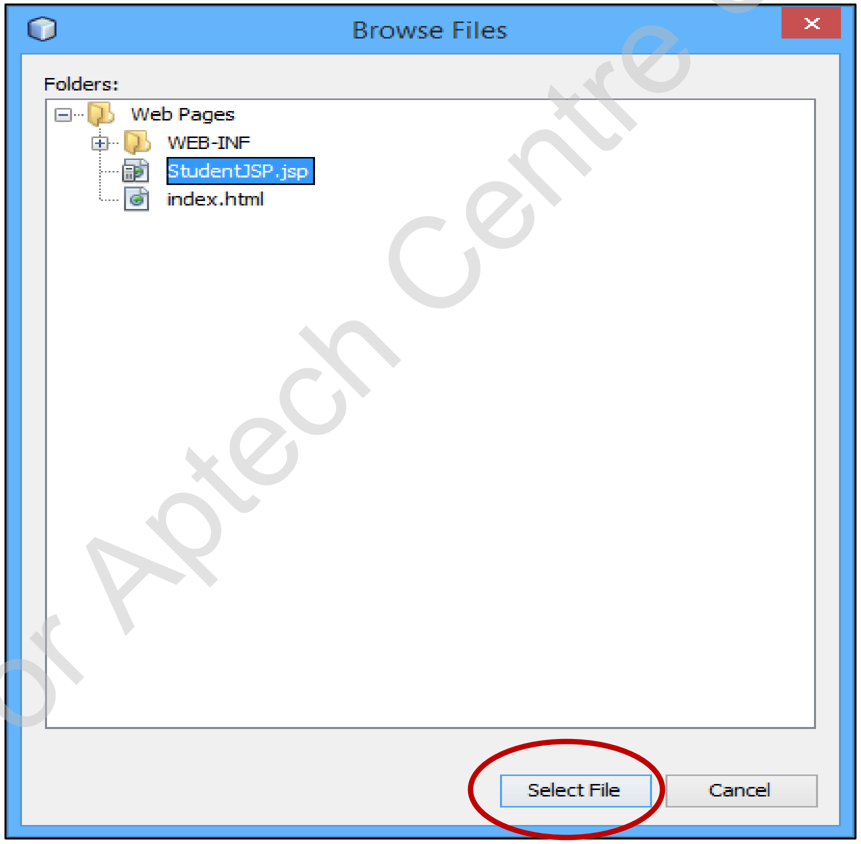
Use comma(,) to separate multiple welcome files.

Implementing a Web Client 12-14

14. Click the Browse button. In the Browse Files dialog box, select the StudentJSP file.

15. Click Select File button.

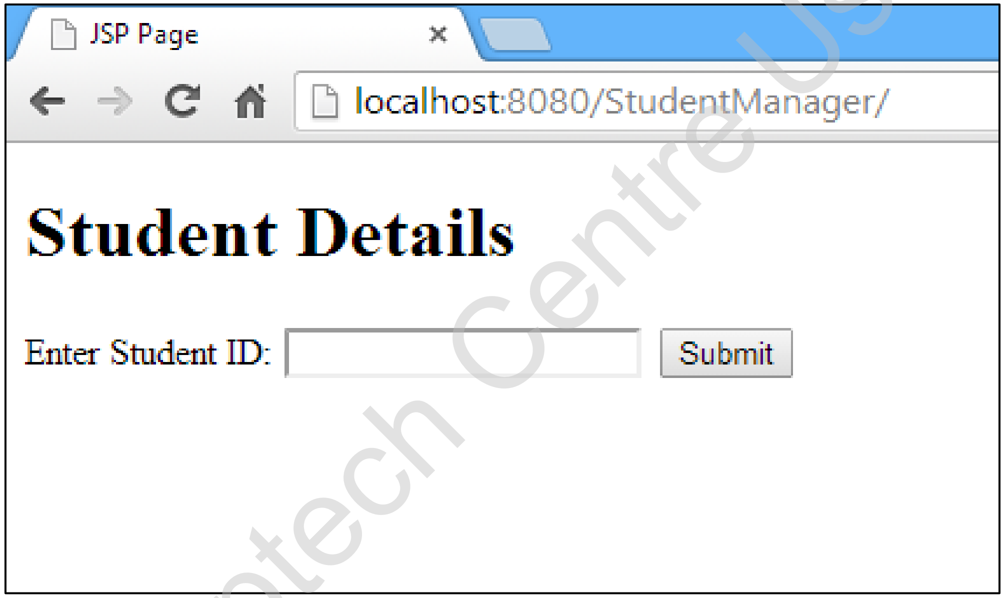
- Following figure shows how to select the JSP file:



Implementing a Web Client 13-14

16. Save, build, and run the application. The Web page opens displaying a text box to enter the student ID and a Submit button.

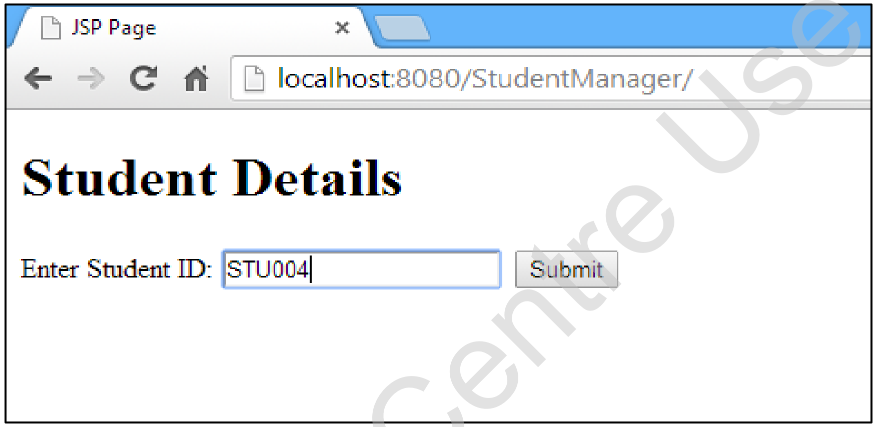
- Following figure shows the JSP page for entering Student ID:



Implementing a Web Client 14-14

17. Enter the Student ID.

- Following figure shows how to enter the Student ID:

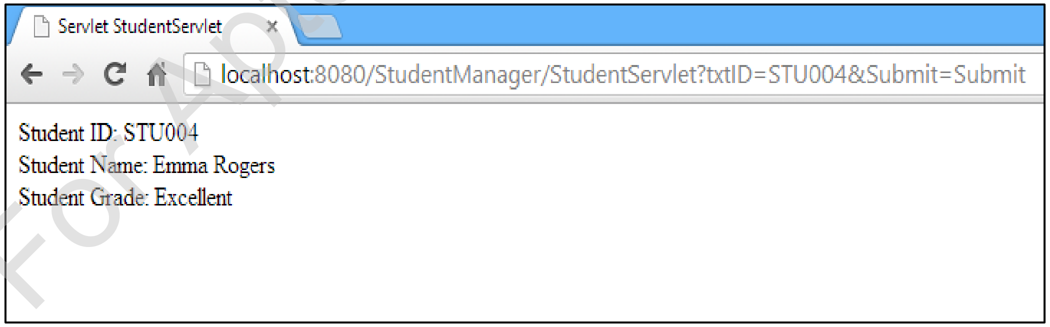


Student Details

Enter Student ID: STU004

18. Click the Submit button. The student's details are displayed.

- Following figure shows the output of Student Details:


Servlet StudentServlet
localhost:8080/StudentManager/StudentServlet?txtID=STU004&Submit=Submit

Student ID: STU004
Student Name: Emma Rogers
Student Grade: Excellent

Summary

- Web services are Web applications based on XML standards and transport protocol, HTTP.
- Web services can be developed either using SOAP based approach or REST based approach.
- REST is an architecture style used for developing Web services. It is a set of guidelines or principles for developing a network-system.
- RESTful Web services are Web services based on REST principles and accessed over HTTP protocol on the Web.
- The requirements for developing RESTful Web services based on the REST architecture style are resources, data representation, URI, and HTTP methods.
- JAX-RS is a Java programming API designed to simplify the development process of RESTful Web service.
- JAX-RS API is based on annotations such as @Path, @GET, @POST, @PUT, @DELETE, @Produce, @Consumes, and so on which are present in javax.ws.rs package.