



UNIFESSPA

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ

Universidade Federal do Sul e Sudeste do Pará

Sistemas Distribuídos

Prof.: Warley Junior

wmvj@unifesspa.edu.br

Agenda

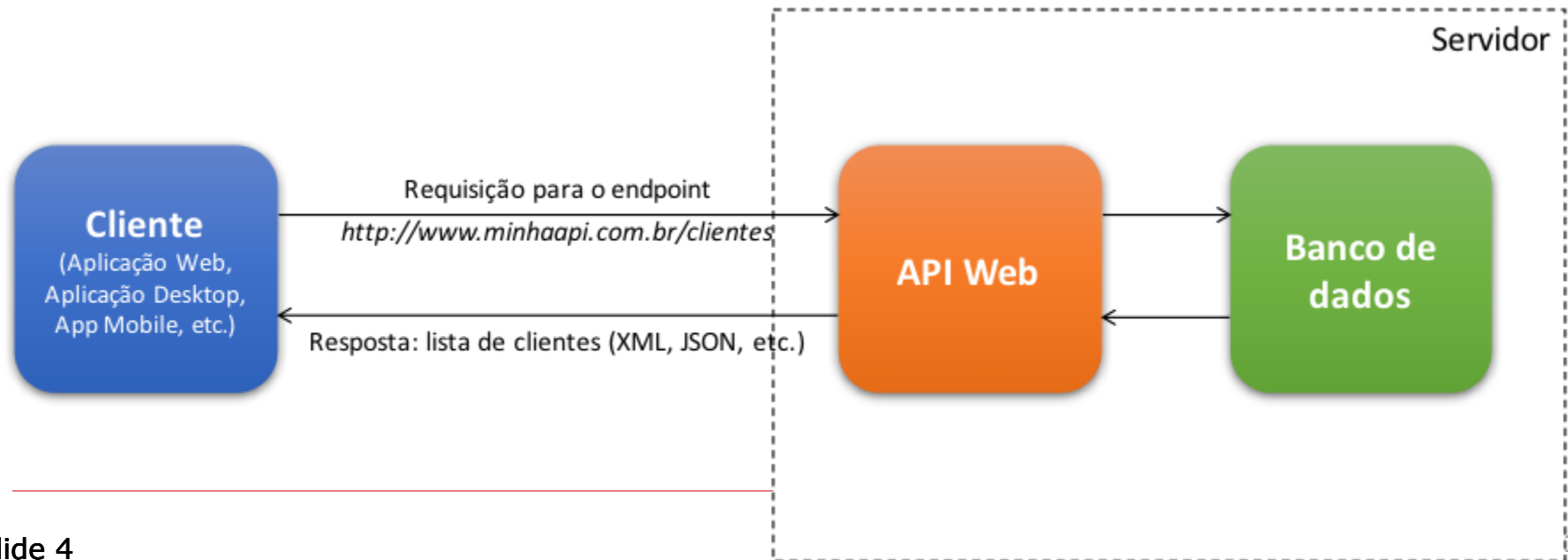
- ❑ AULA 11 (PARTE II):
- ❑ Middlewares
 - WebServices
 - JAX-RS

O que é REST?

- ❑ REST é um acrônimo para *REpresentational State of Transfer*, ou seja: Transferência Representacional de Estado.
- ❑ É um **estilo de arquitetura**
- ❑ Funciona em cima do protocolo **HTTP**.

Alguns termos importantes

- ❑ Para uma API Web disponibilizar informações e executar processos, ela disponibiliza os chamados **endpoints**.



Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Fazer o download do Jersey a partir do site: <https://eclipse-ee4j.github.io/jersey/>

JAX-RS 2.1 / Jersey 2.26+

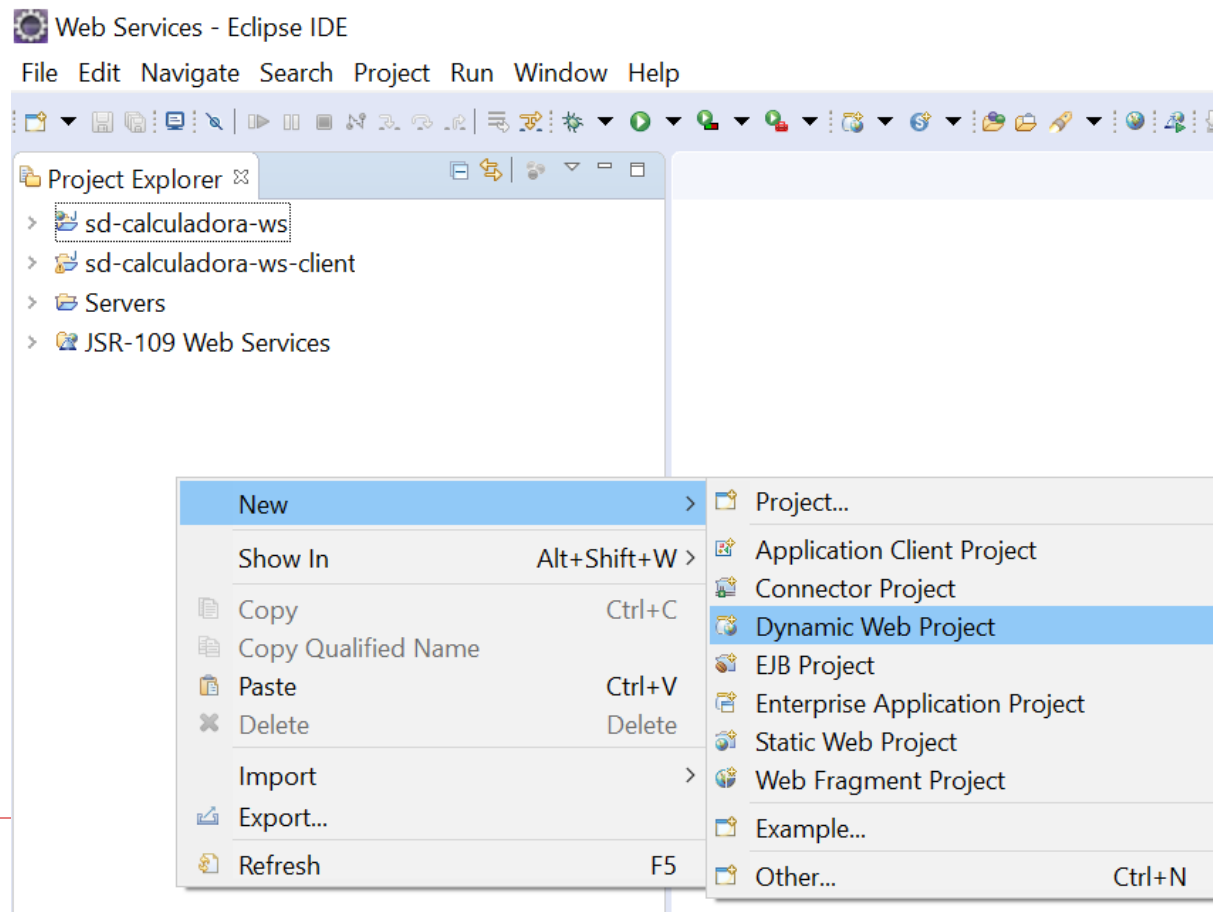
Jersey 2.29.1, that implements [JAX-RS 2.1 API](#) is the most recent release of Jersey. To see the details about all changes, bug fixed and updates, please check the [Jersey 2.29.1 Release Notes](#).

For the convenience of non-maven developers the following links are provided:

- [Jersey JAX-RS 2.1 RI bundle](#) bundle contains the JAX-RS 2.1 API jar, all the core Jersey module jars as well as all the required 3rd-party dependencies.
- [Jersey 2.29.1 Examples bundle](#) provides convenient access to the Jersey 2 examples for off-line browsing.

Criando um Dynamic Web Project e incorporando o Jersey

❑ Criar um novo Projeto Web Dinâmico



Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Defina um nome para seu projeto.
- ❑ Em target runtime define o servidor Tomcat.

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 v9.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☒ Add project to an EAR

EAR project name:

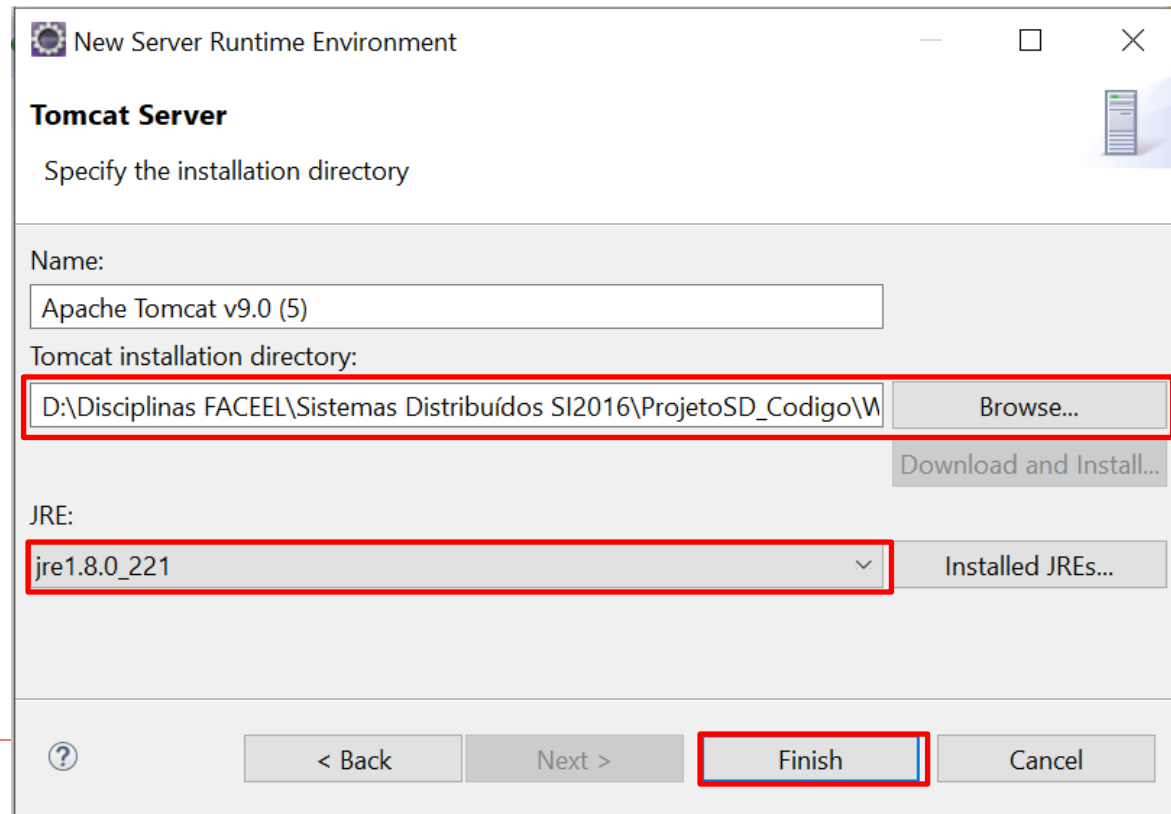
Working sets

☒ Add project to working sets

Working sets:

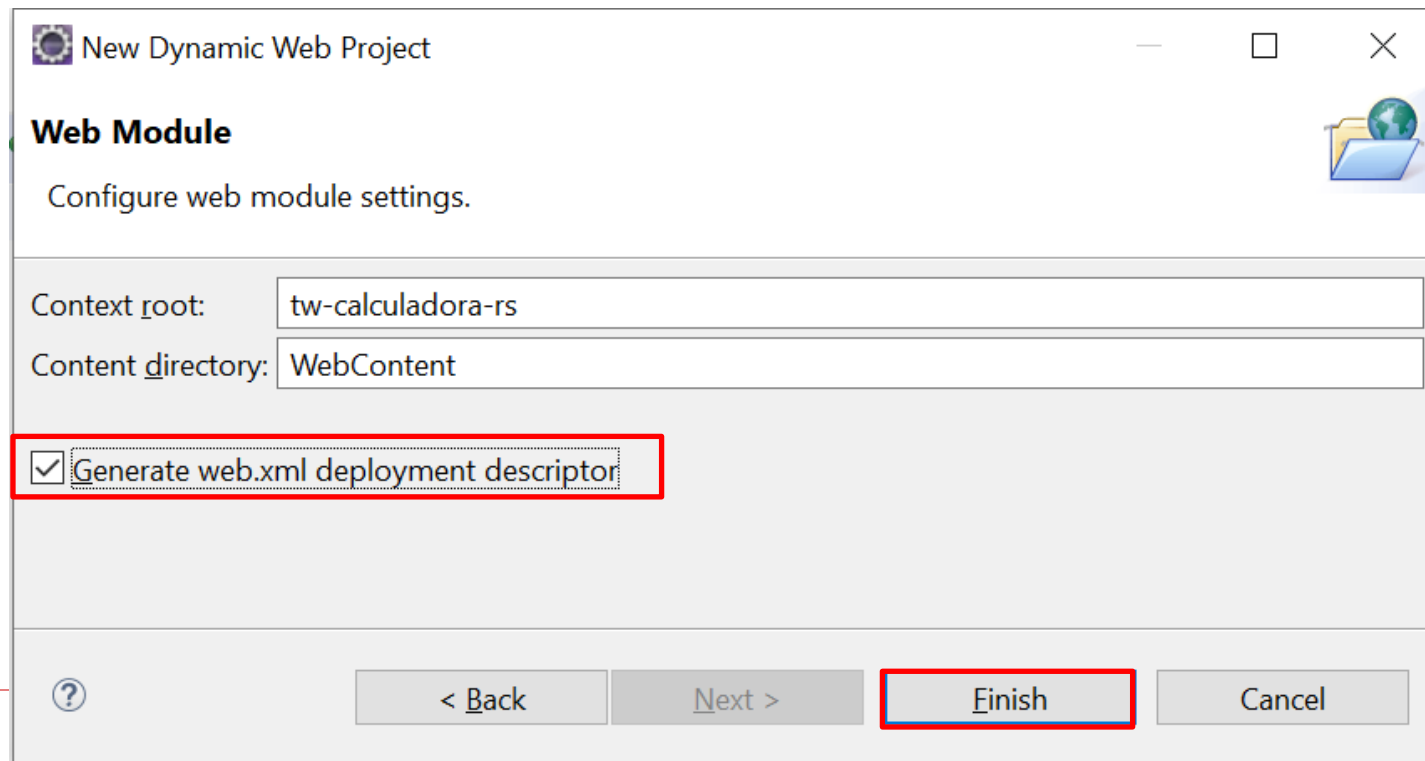
Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Selecione a pasta correspondente do TomCat.



Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Vamos precisar do web.xml para fazer com que o Jersey funcione corretamente.



New Dynamic Web Project

Web Module
Configure web module settings.

Context root: tw-calculadora-rs

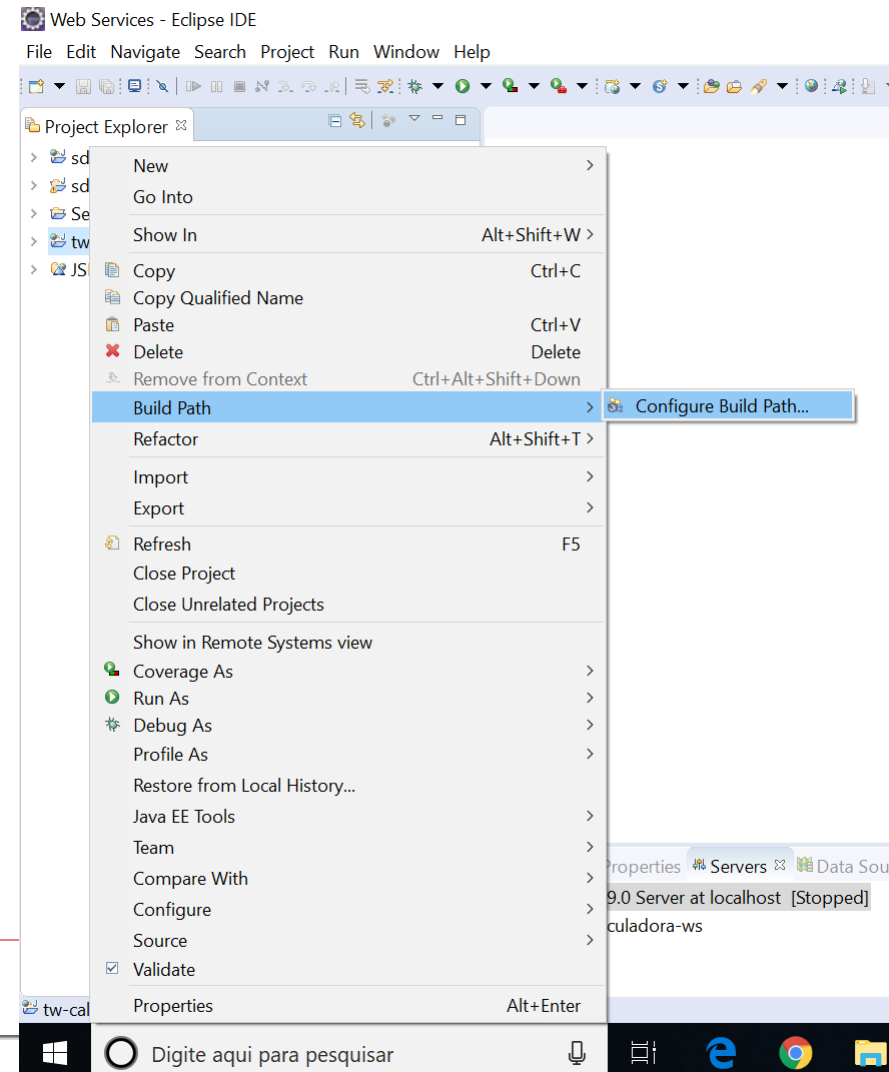
Content directory: WebContent

☒ Generate web.xml deployment descriptor

? < Back Next > Finish Cancel

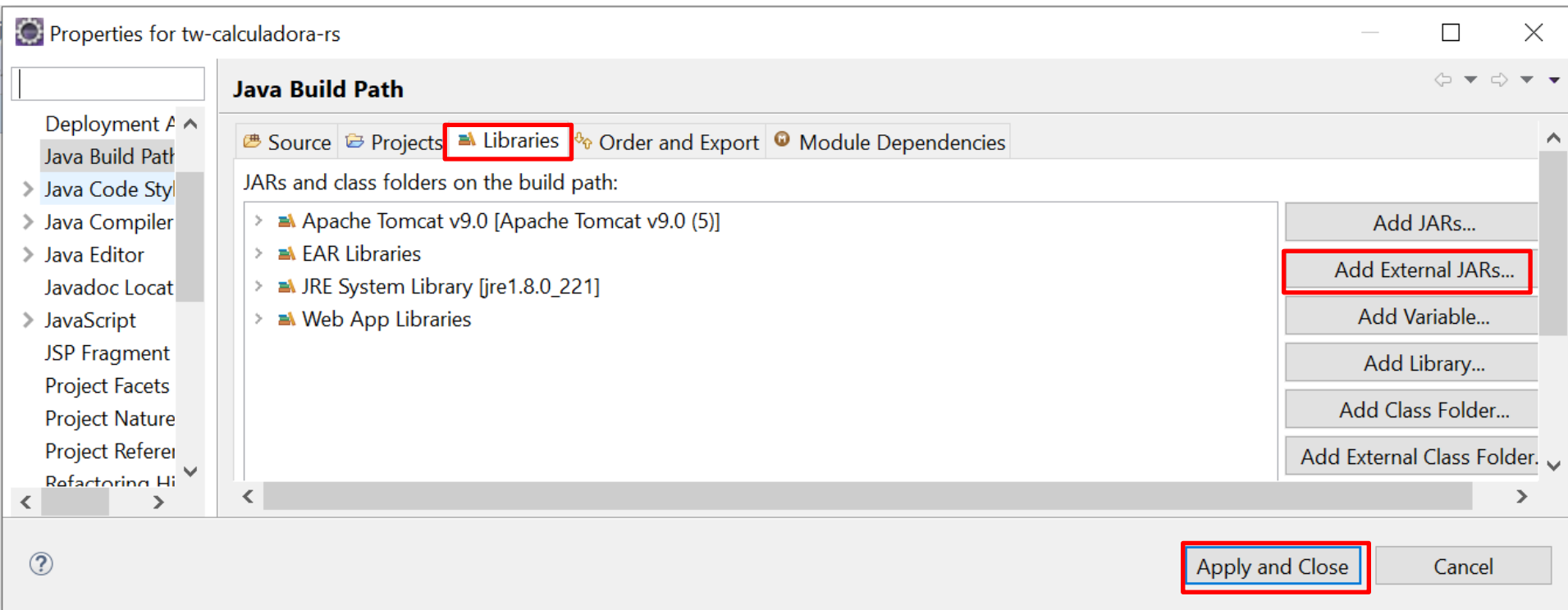
Criando um Dynamic Web Project e incorporando o Jersey

- ❑ O Tomcat não tem todas as implementações para que suporte o JAX-RS. Por conta disso, temos que adicionar todos os JARS no build path.



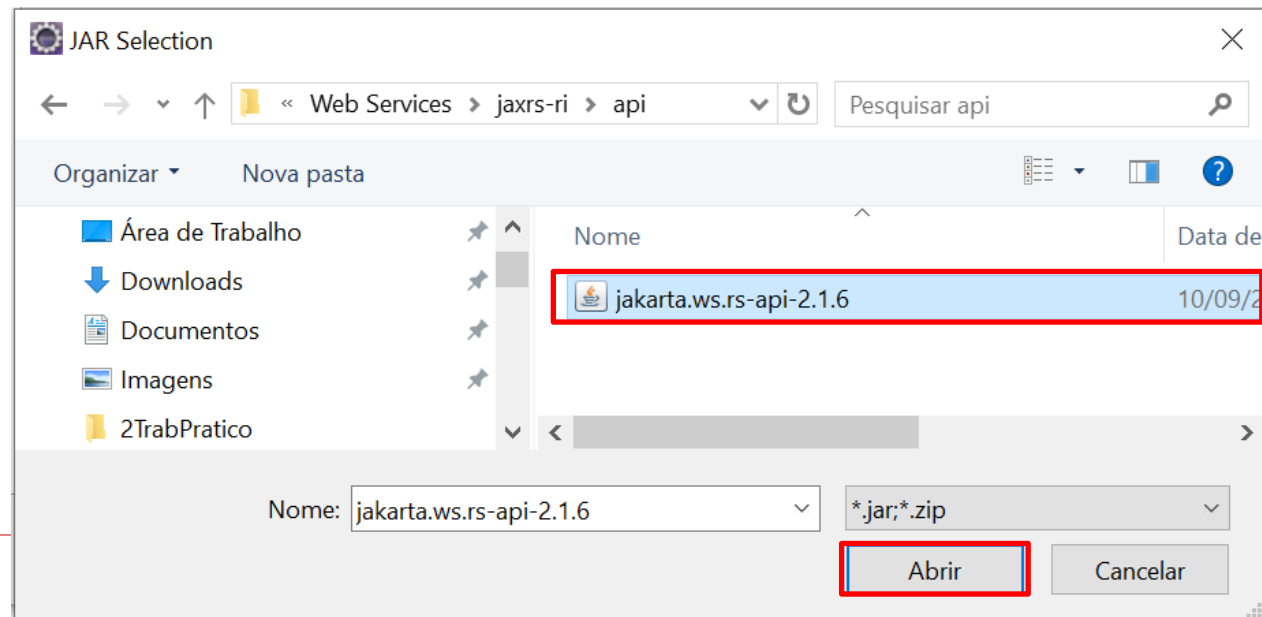
Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Adicione todos os JARs do Jersey para o projeto.



Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Dentro da pasta "jaxrs-ri" e para cada pasta "api", "ext", e "lib", incluir todos os arquivos JARs.



Criando um Dynamic Web Project e incorporando o Jersey

- Fazer agora com que todos os JARs estejam presentes no WAR para que o Tomcat tenha acesso a eles.

The screenshot shows the Eclipse IDE interface with the 'Properties for tw-calculadora-rs' dialog box open. The 'Deployment Assembly' tab is selected in the left-hand tree. The 'Web Deployment Assembly' tab is active, showing a table with 'Source' and 'Deploy Path' columns. The 'Add...' button is highlighted. A 'New Assembly Directive' dialog box is open, showing the 'Select Directive Type' list with 'Java Build Path Entries' selected. The 'Next >' button is highlighted.

Properties for tw-calculadora-rs

type filter text

- Builders
- Coverage
- Deployment Assembly**
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- JavaScript
- JSP Fragment
- Project Facets
- Project Natures
- Project References
- Run/Debug Settings
- Server
- Service Policies
- Targeted Runtimes
- Task Repository
- Task Tags

Web Deployment Assembly

Define packaging structure for this Java EE Web Application project.

Source	Deploy Path
/src	WEB-INF/classes
/WebContent	/

Add...

Edit...

Remove

New Assembly Directive

Select Directive Type

Add a new assembly directive.

- Archive via Path Variable
- Archives from File System
- Archives from Workspace
- Folder
- Java Build Path Entries**
- Project

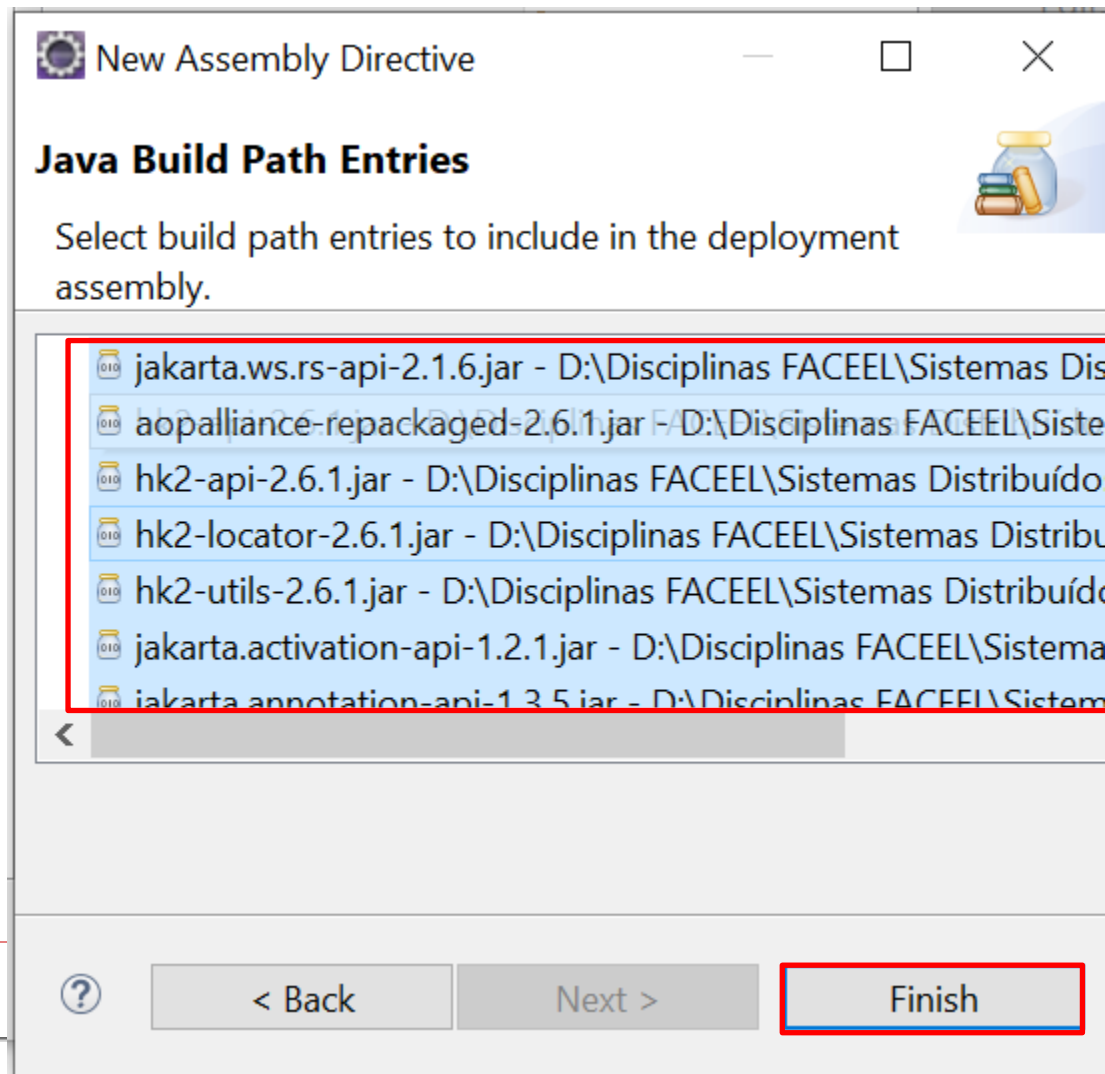
Next >

< Back

Finish

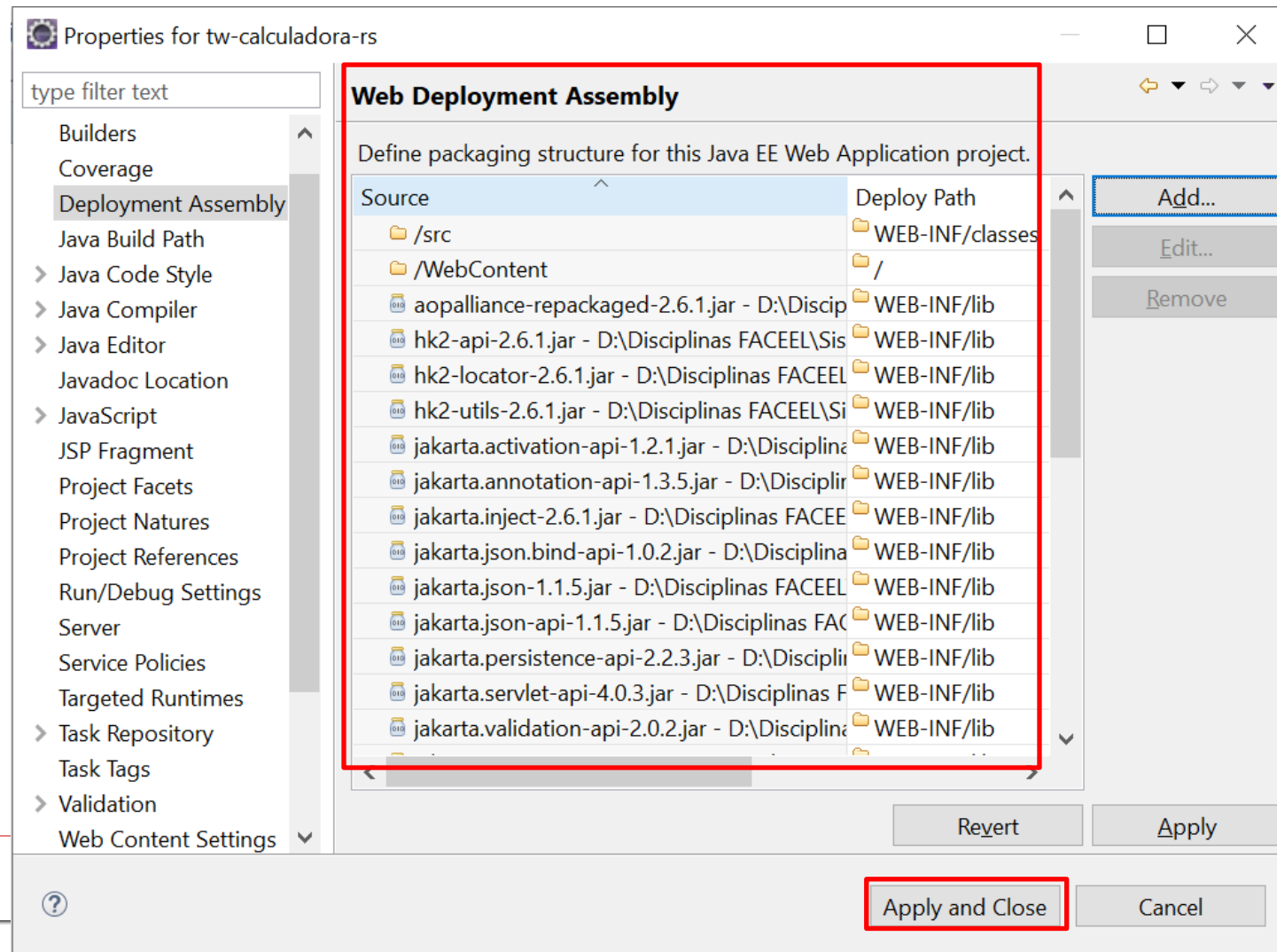
Criando um Dynamic Web Project e incorporando o Jersey

- ❑ Selecionar todos os "JARs".
- ❑ Em seguida clicar em "Finish".



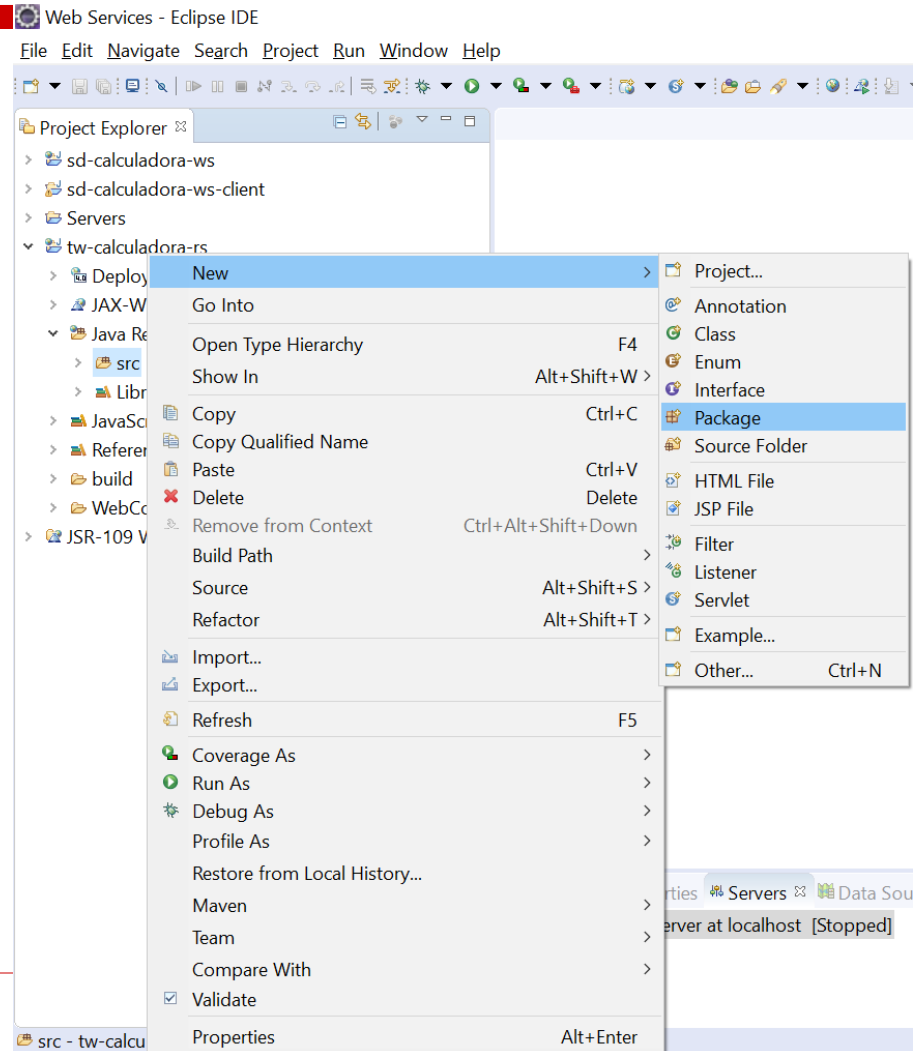
Criando um Dynamic Web Project e incorporando o Jersey

□ Clique em “Apply and Close”.



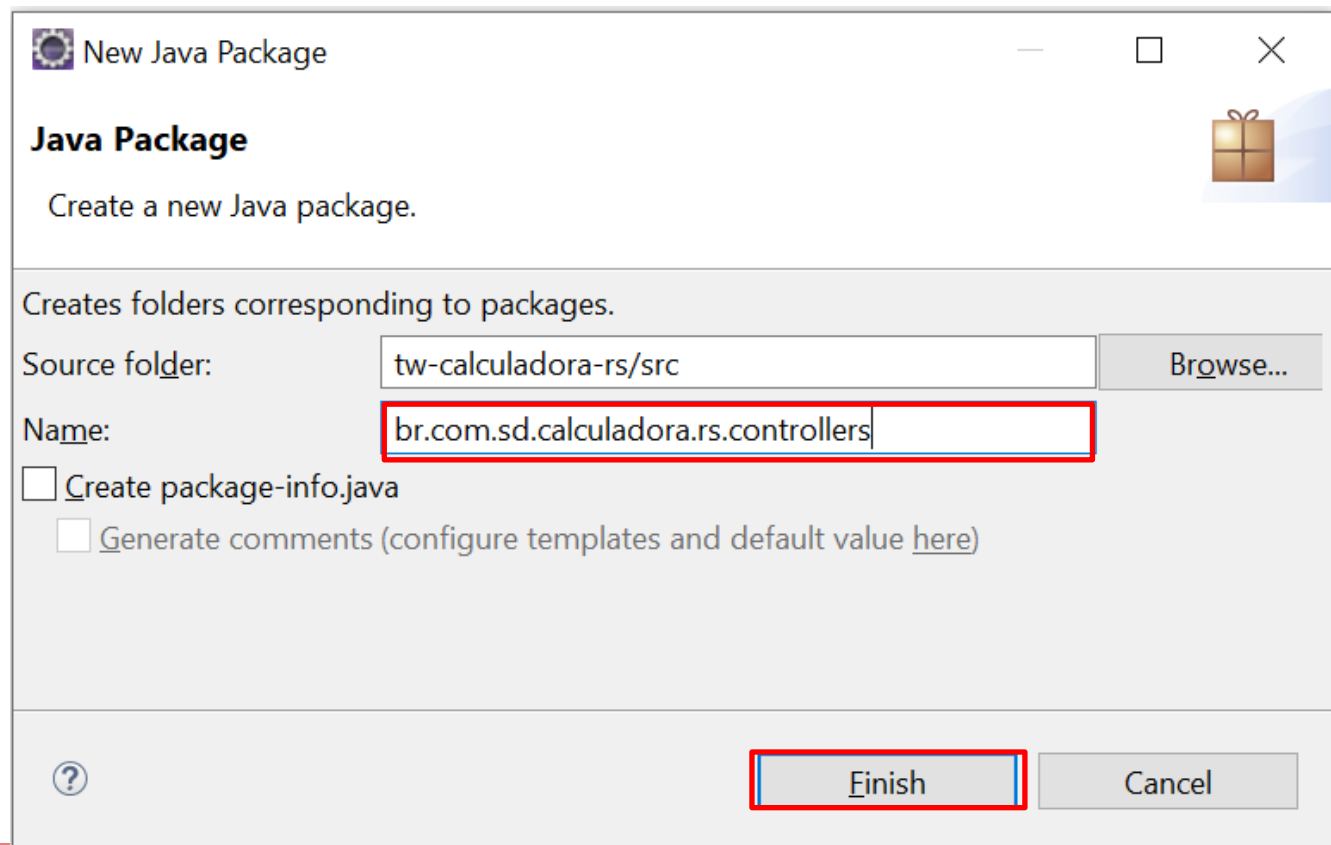
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

□ Expor os recursos que a API REST irá permitir a manipulação.



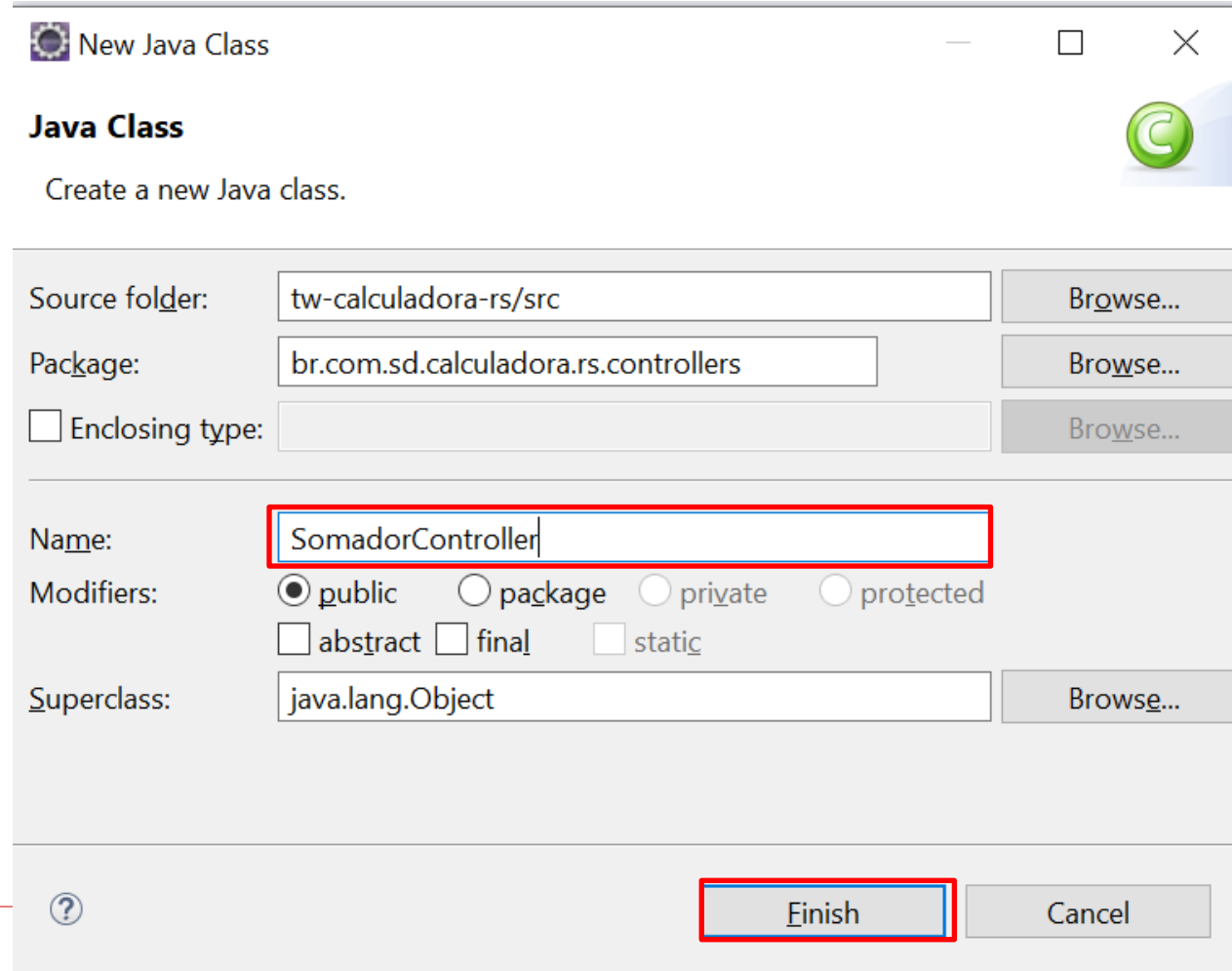
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Defina um nome para seu pacote.



Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- Defina um nome para sua classe



New Java Class

Java Class
Create a new Java class.

Source folder: [Browse...](#)

Package: [Browse...](#)

☐ Enclosing type: [Browse...](#)

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: [Browse...](#)

[?](#) [Finish](#) [Cancel](#)

Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Vamos expor esta classe como um controller ou um recurso REST.

```
package br.com.sd.calculadora.rs.controllers;

import javax.ws.rs.Path;

@Path("/somador")
public class SomadorController {

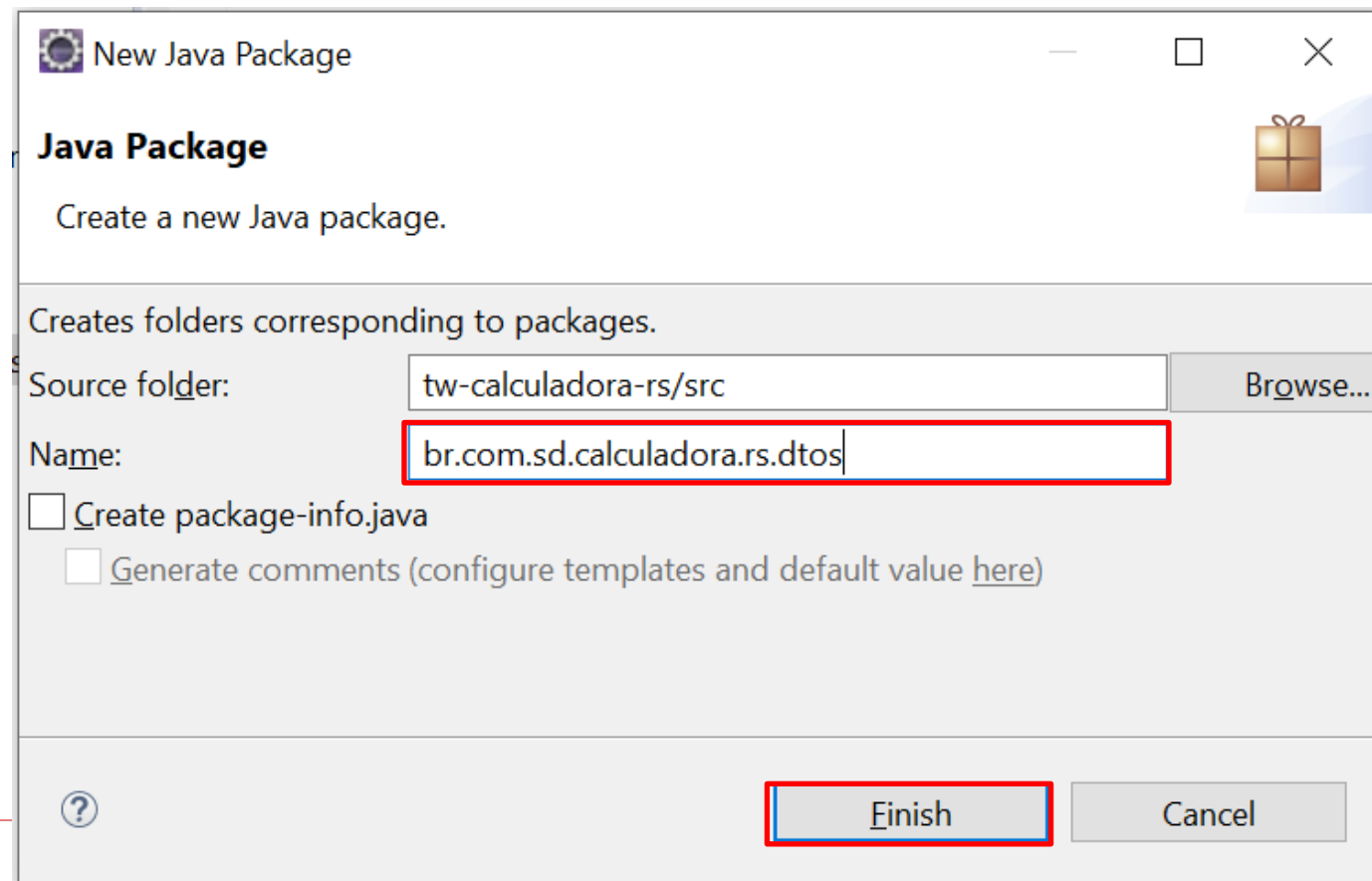
    public ? somar(int n1, int n2){

    }

}
```

Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Defina um nome para este pacote.



Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- Sobre o pacote recém criado, deve criar uma nova classe.

New Java Class

Java Class

Create a new Java class.

Source folder: tw-calculadora-rs/src Browse...

Package: br.com.sd.calculadora.rs.dtos Browse...

☐ Enclosing type: Browse...

Name: ResultadoDTO

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

? Finish Cancel

Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- A função desta classe é representar recursos que serão trafegadas do meu “controller” para meus clientes.

```
package br.com.sd.calculadora.rs.dtos;

public class ResultadoDTO {

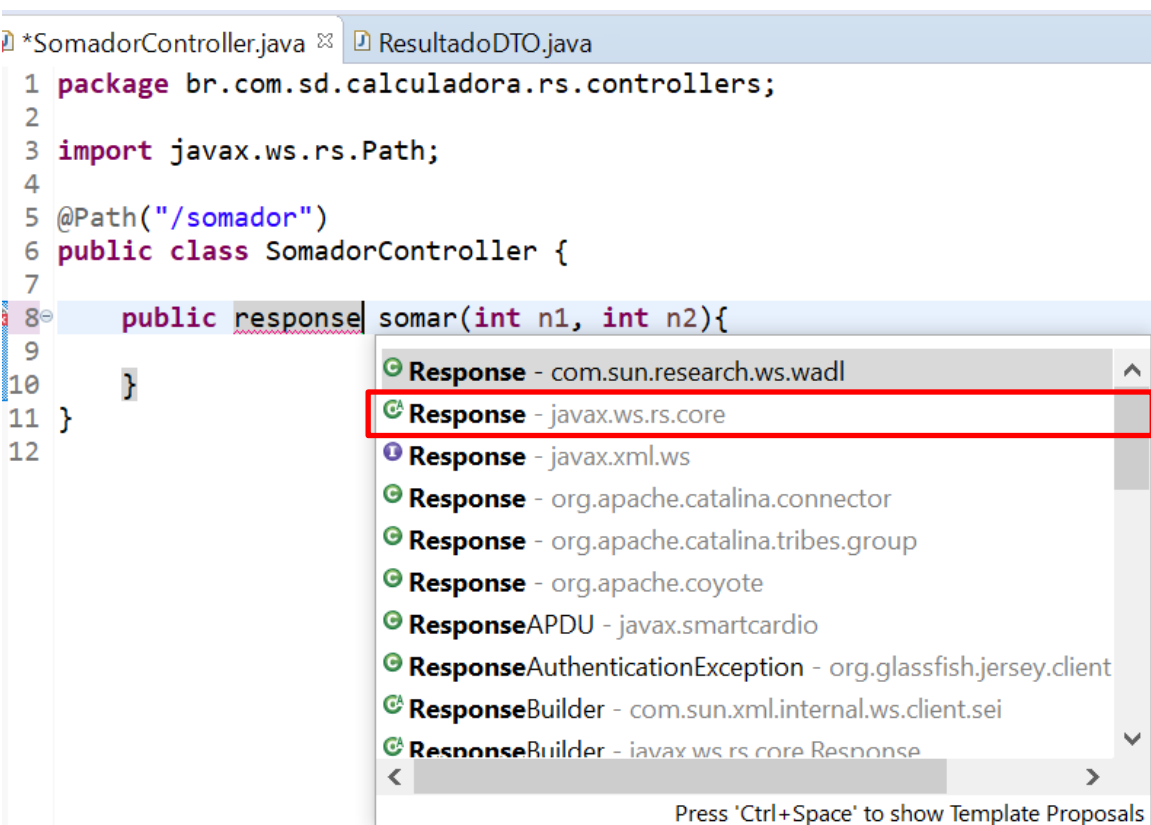
    private int resultado;

    public int getResultado() {
        return resultado;
    }

    public void setResultado(int resultado) {
        this.resultado = resultado;
    }

}
```

Compreendendo a estrutura do Jersey e implementando um serviço RESTful



```
*SomadorController.java  ResultadoDTO.java
1 package br.com.sd.calculadora.rs.controllers;
2
3 import javax.ws.rs.Path;
4
5 @Path("/somador")
6 public class SomadorController {
7
8     public response somar(int n1, int n2){
9
10    }
11 }
12
```

Response - com.sun.research.ws.wadl
Response - javax.ws.rs.core
Response - javax.xml.ws
Response - org.apache.catalina.connector
Response - org.apache.catalina.tribes.group
Response - org.apache.coyote
ResponseAPDU - javax.smartcardio
ResponseAuthenticationException - org.glassfish.jersey.client
ResponseBuilder - com.sun.xml.internal.ws.client.sei
ResponseBuilder - javax.ws.rs.core.Response

Press 'Ctrl+Space' to show Template Proposals

- Esse DTO estará dentro de um "response". É a resposta que a minha API irá fornecer para quem solicitar este recurso.

```
package br.com.sd.calculadora.rs.controllers;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

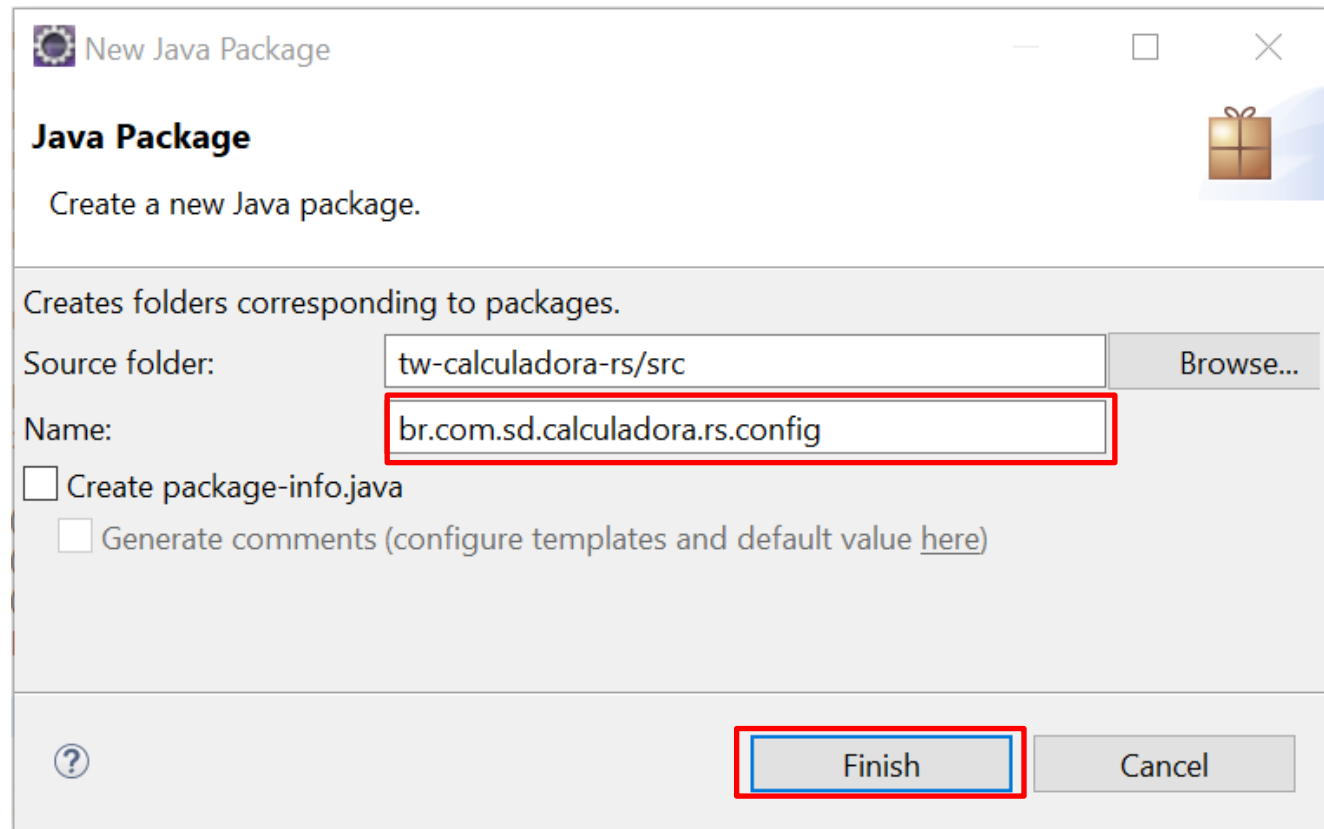
import br.com.sd.calculadora.rs.dtos.ResultadoDTO;

@Path("/somador")
public class SomadorController {

    @GET
    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
    @Path("/{n1}/{n2}")
    public Response somar(@PathParam("n1") int n1, @PathParam("n2") int n2){
        ResultadoDTO resultado = new ResultadoDTO();
        resultado.setResultado(n1 + n2);
        return Response.status(200).entity(resultado).build();
    }
}
```


Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Criar um novo pacote.



New Java Package

Java Package
Create a new Java package.

Creates folders corresponding to packages.

Source folder:

Name:

☐ Create package-info.java
☐ Generate comments (configure templates and default value [here](#))

Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ A partir do pacote recém criado, deve criar uma nova classe.

New Java Class

Java Class
Create a new Java class.

Source folder: [Browse...](#)

Package: [Browse...](#)

☐ Enclosing type: [Browse...](#)

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: [Browse...](#)

[?](#) [Finish](#) [Cancel](#)

Compreendendo a estrutura do Jersey e implementando um serviço RESTful

```
package br.com.sd.calculadora.rs.config;

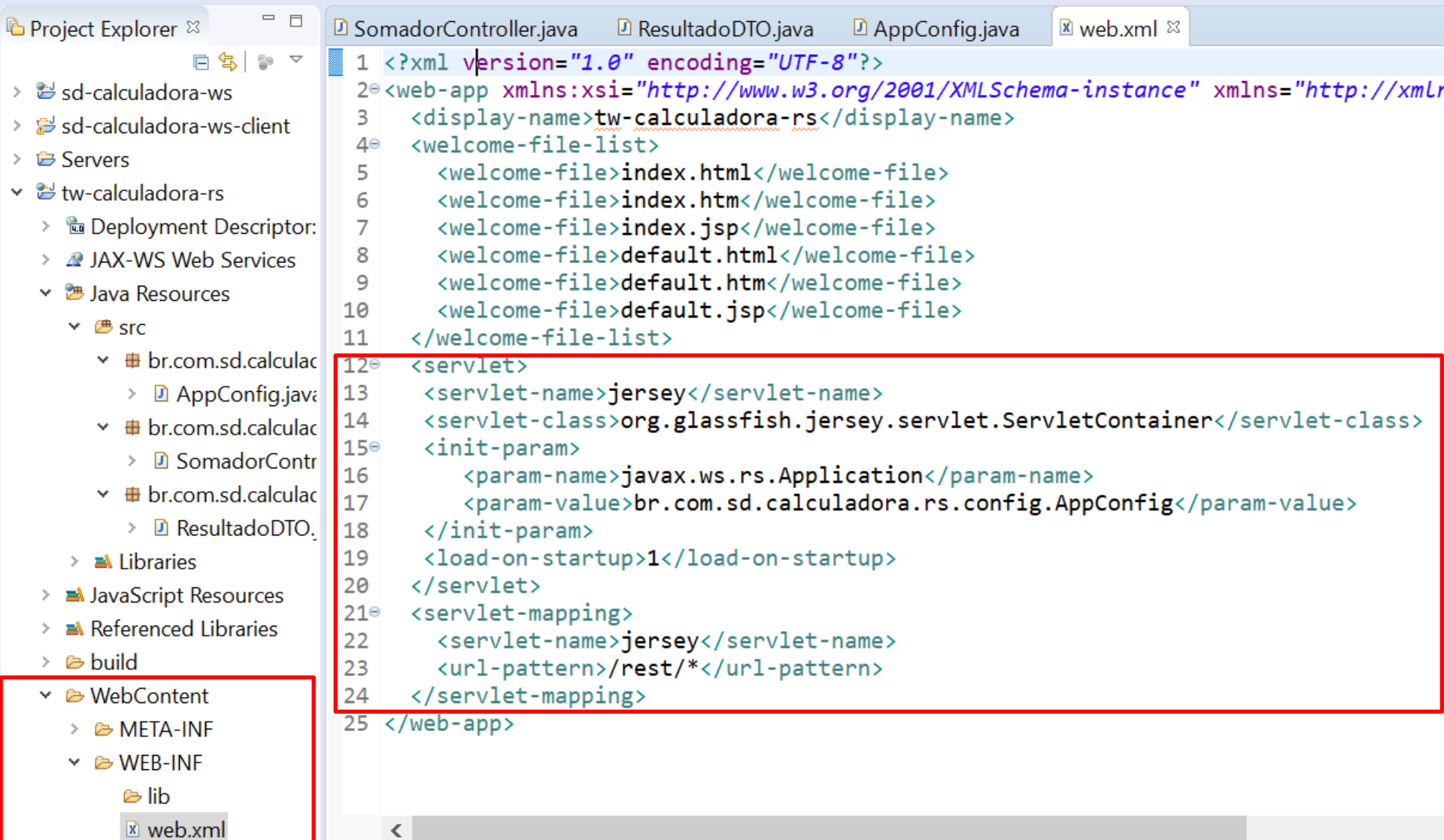
import org.glassfish.jersey.server.ResourceConfig;

import br.com.sd.calculadora.rs.controllers.SomadorController;

public class AppConfig extends ResourceConfig {

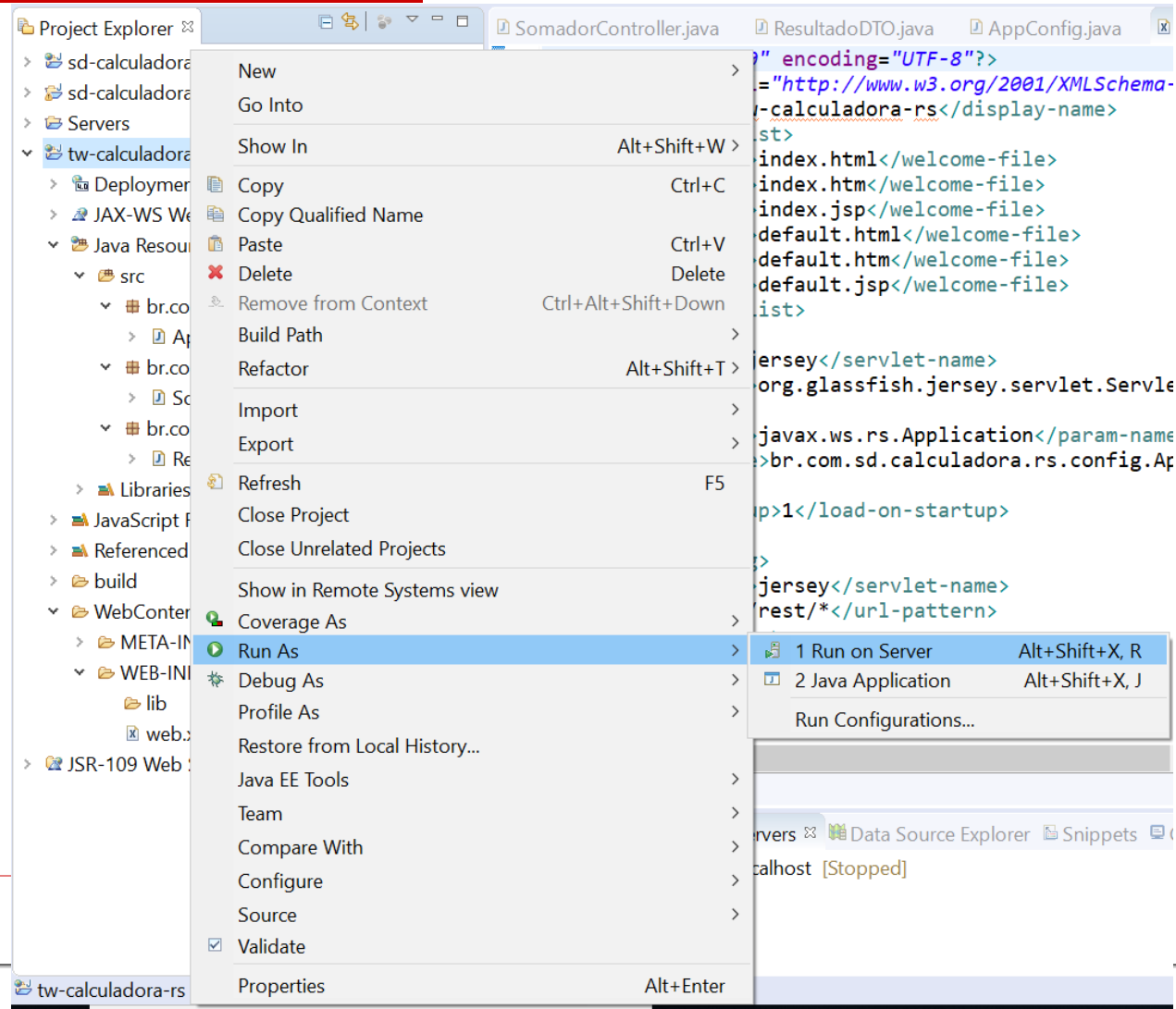
    public AppConfig(){
        register(SomadorController.class);
    }
}
```

Compreendendo a estrutura do Jersey e implementando um serviço RESTful



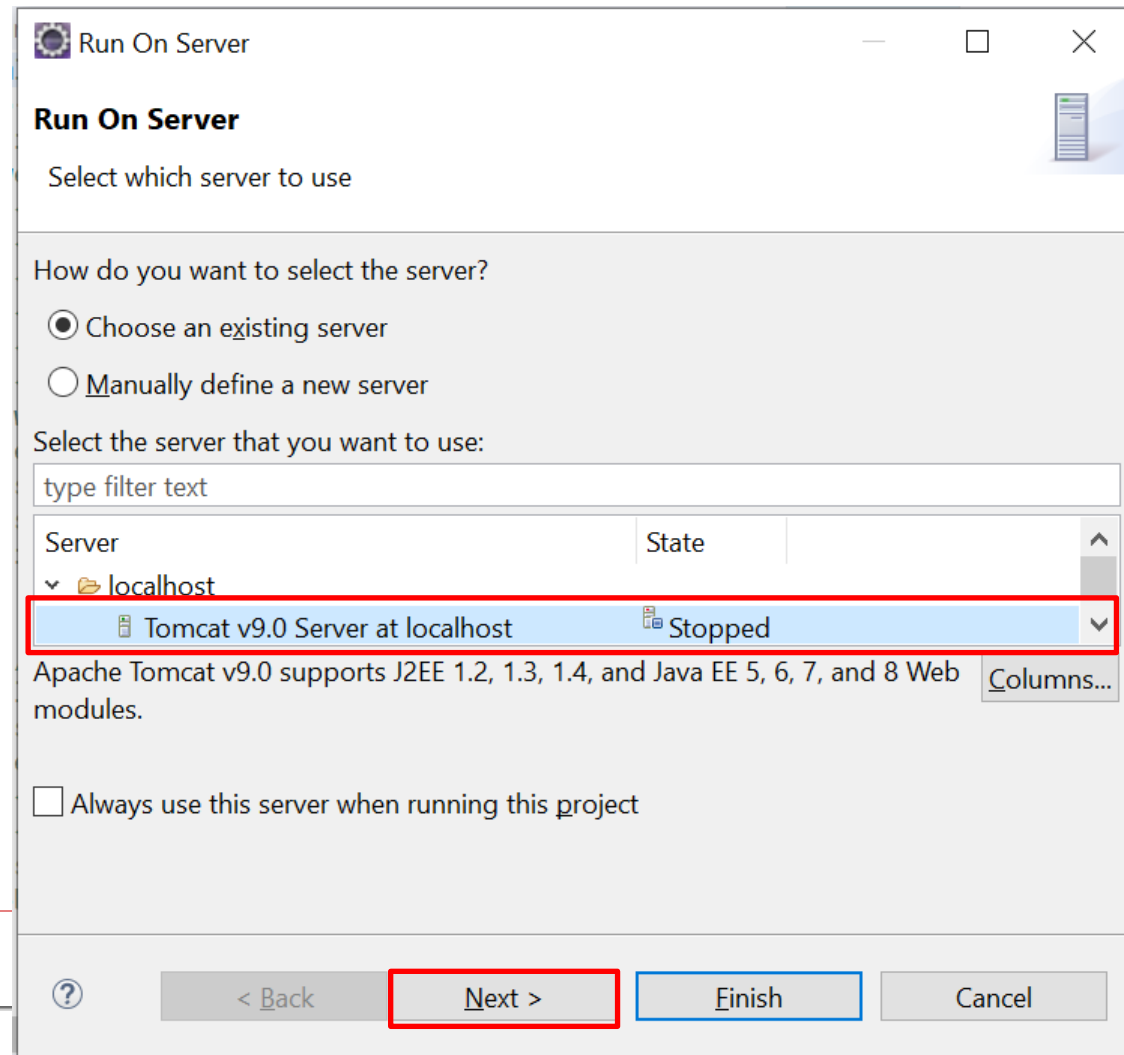
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Testar o web service REST.



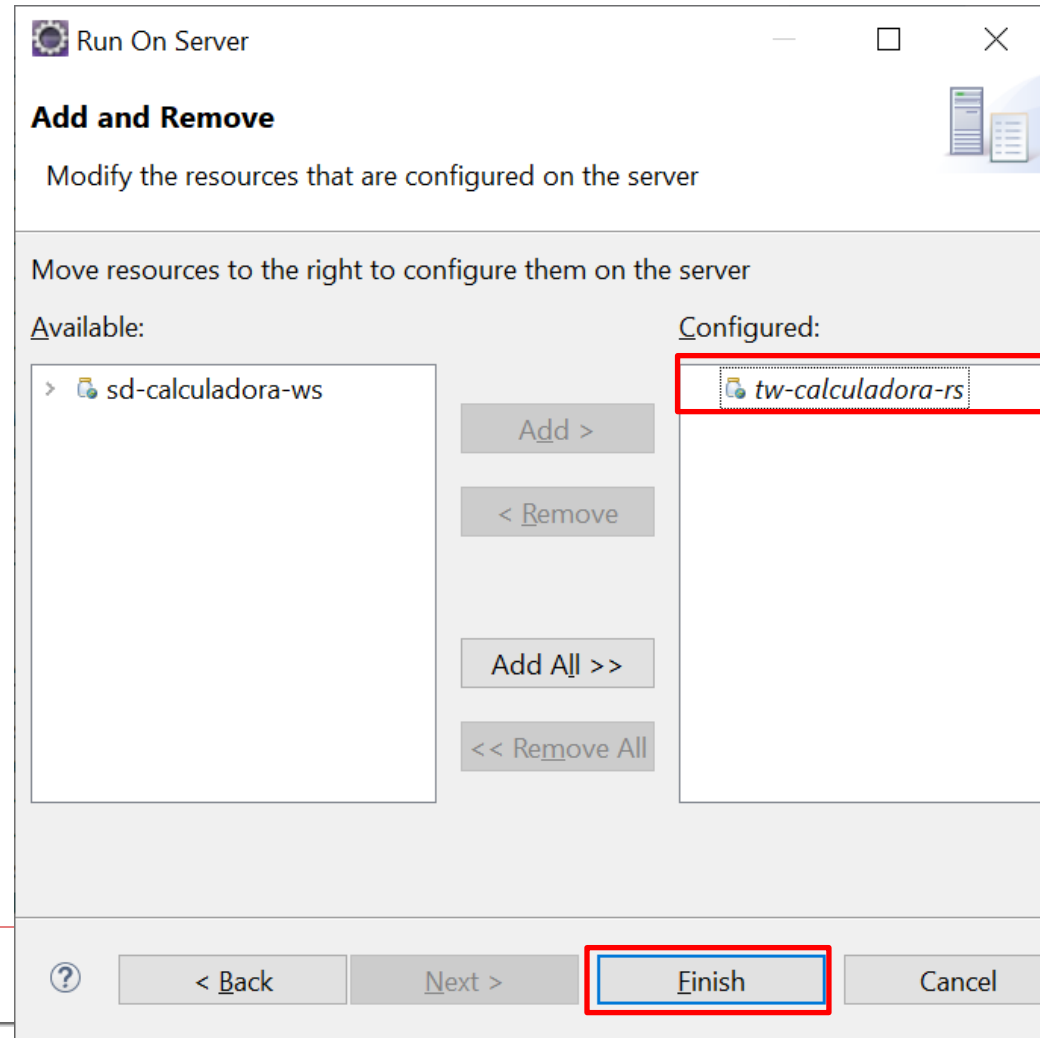
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Selecionar o servidor TomCat.



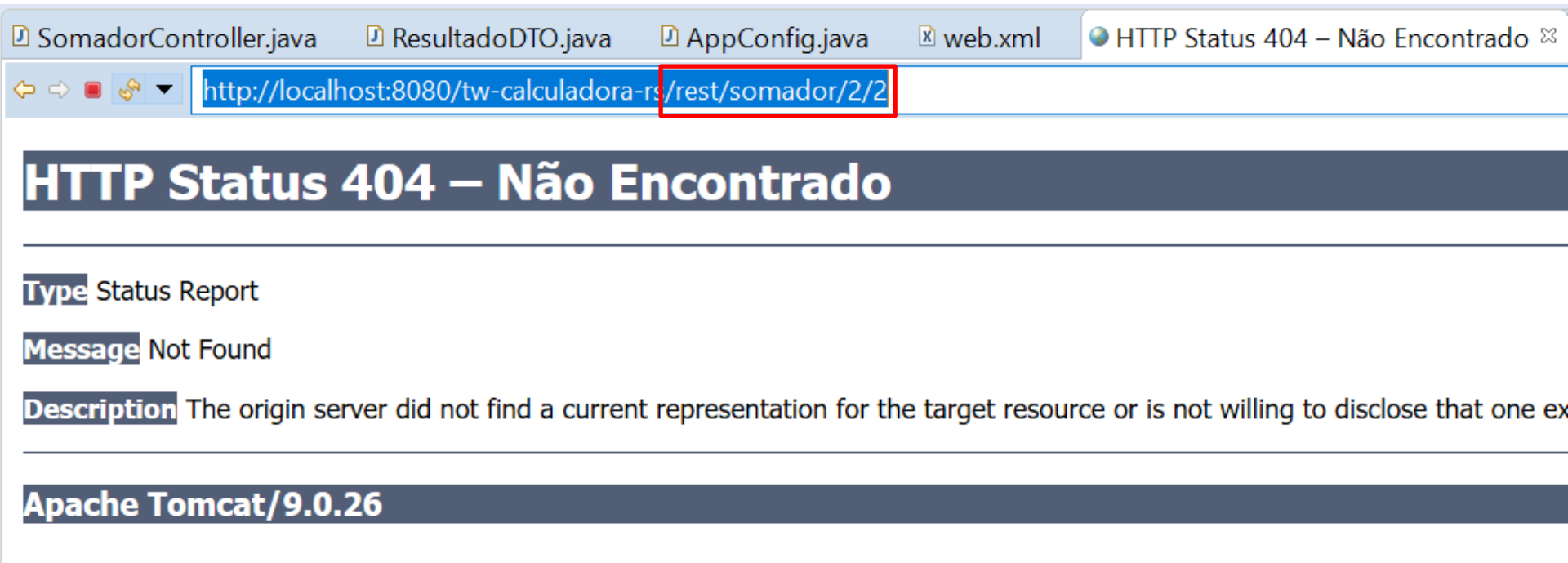
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Adicionar o projeto no servidor.



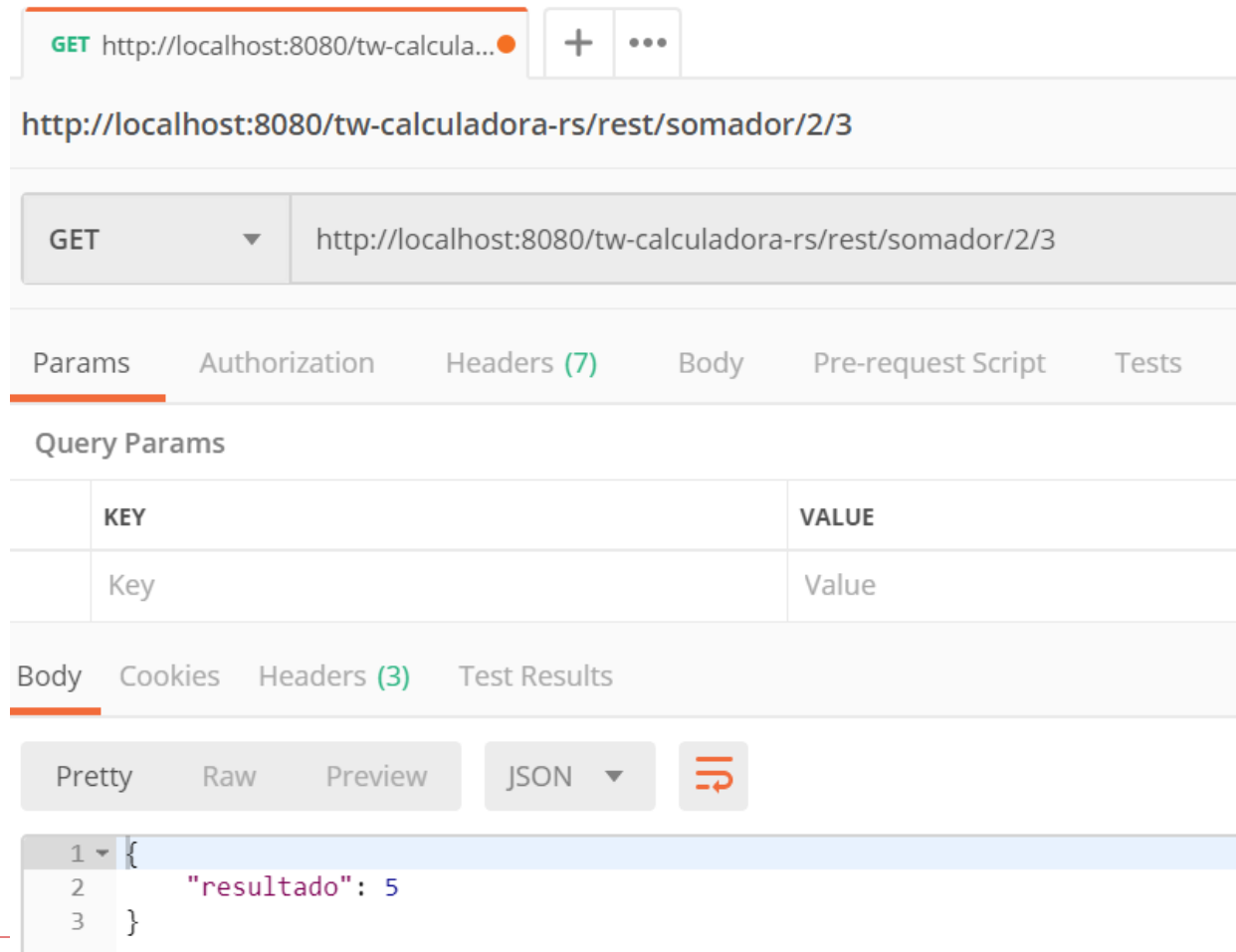
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Acessar o recurso no caminho:
rest/somador/2/2



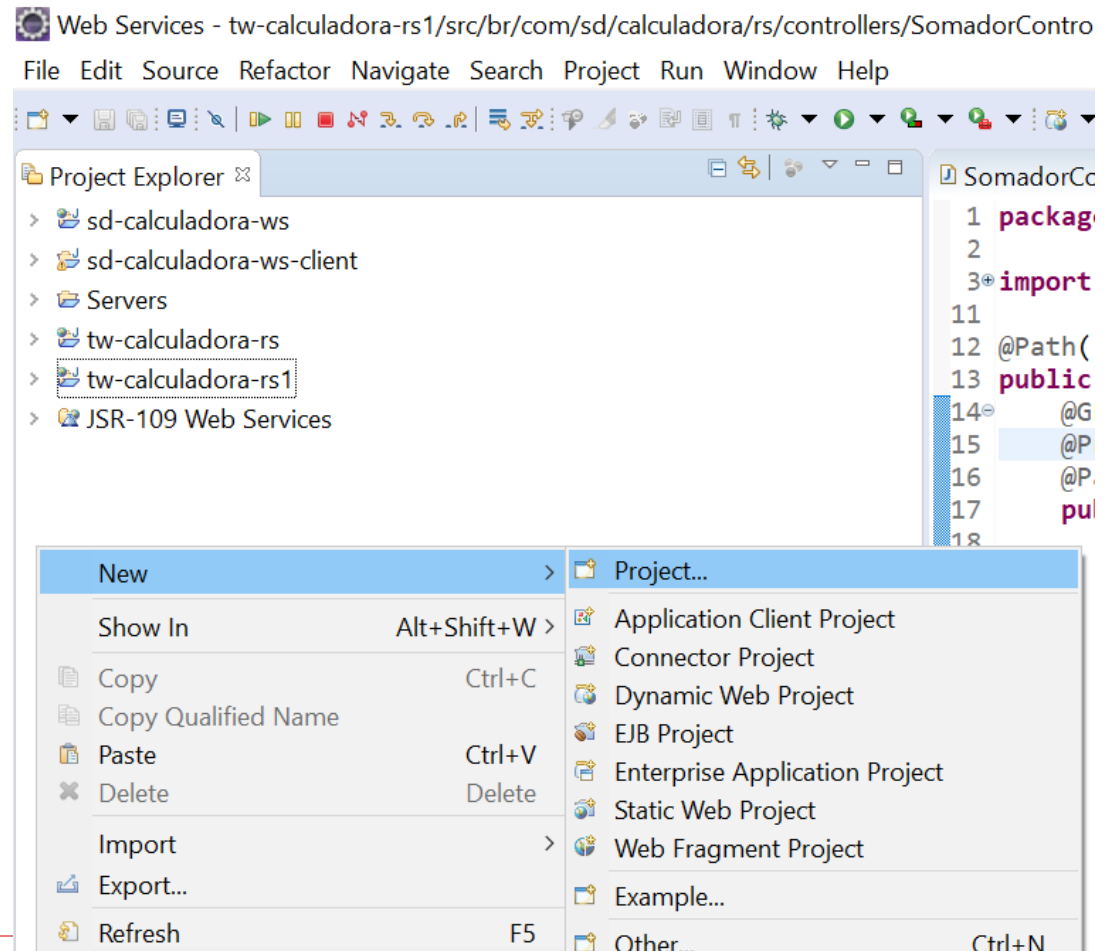
Compreendendo a estrutura do Jersey e implementando um serviço RESTful

- ❑ Resultado utilizando o Postman.

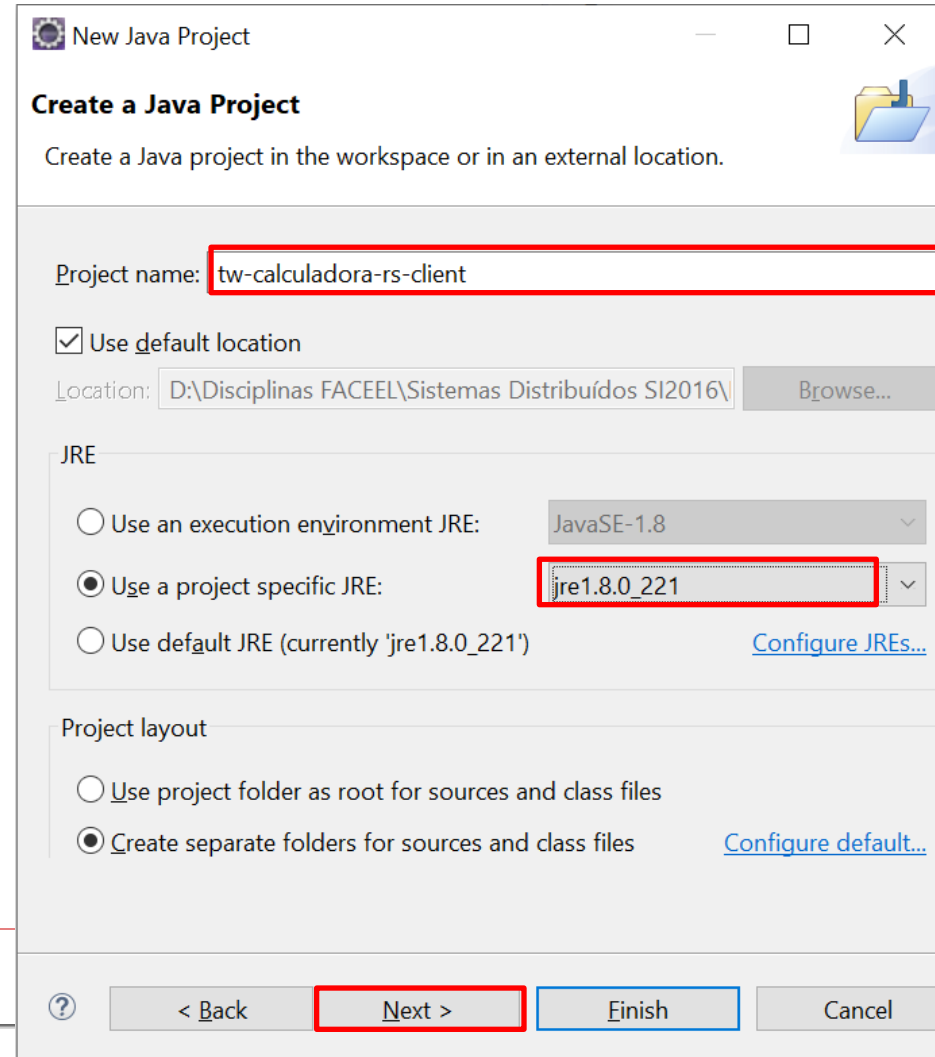
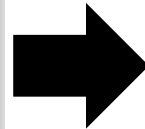
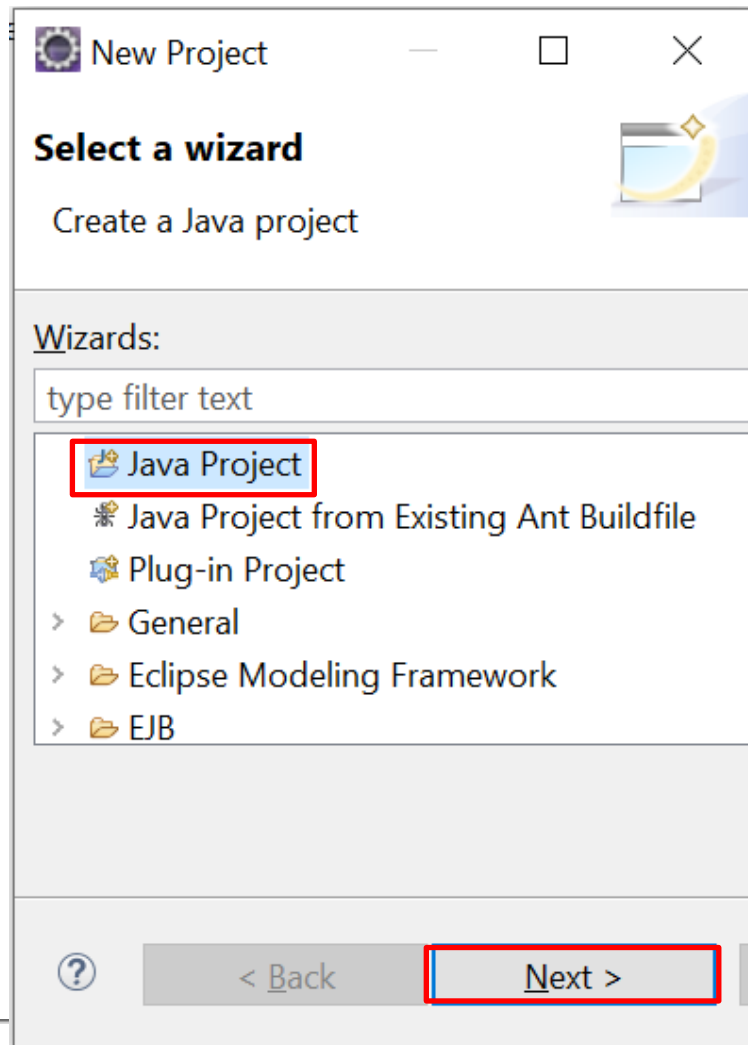


Implementando um cliente para consumo de um serviço RESTful com Jersey

- ❑ Vamos gerar um cliente para consumir a API REST. Para isso vamos criar um novo projeto.

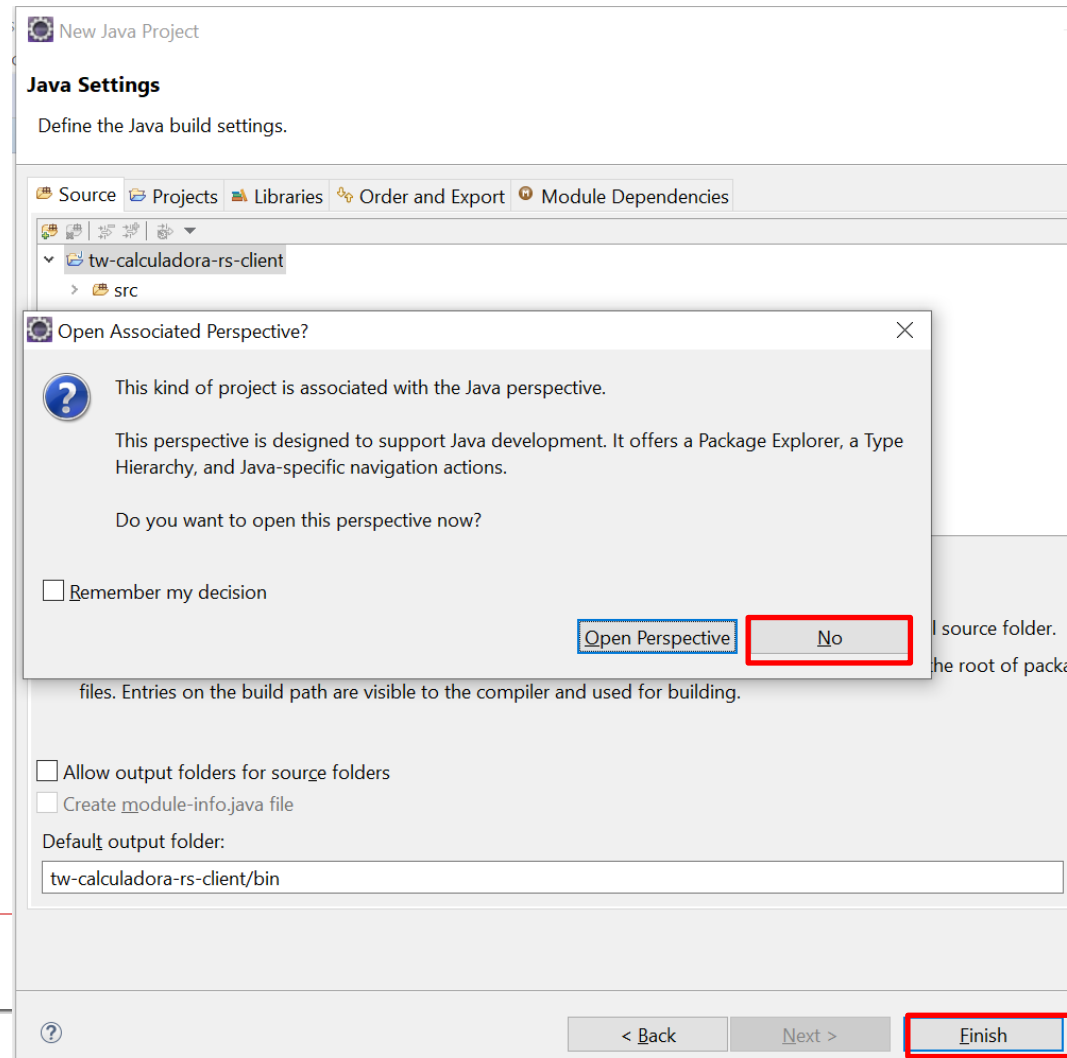


Implementando um cliente para consumo de um serviço RESTful com Jersey



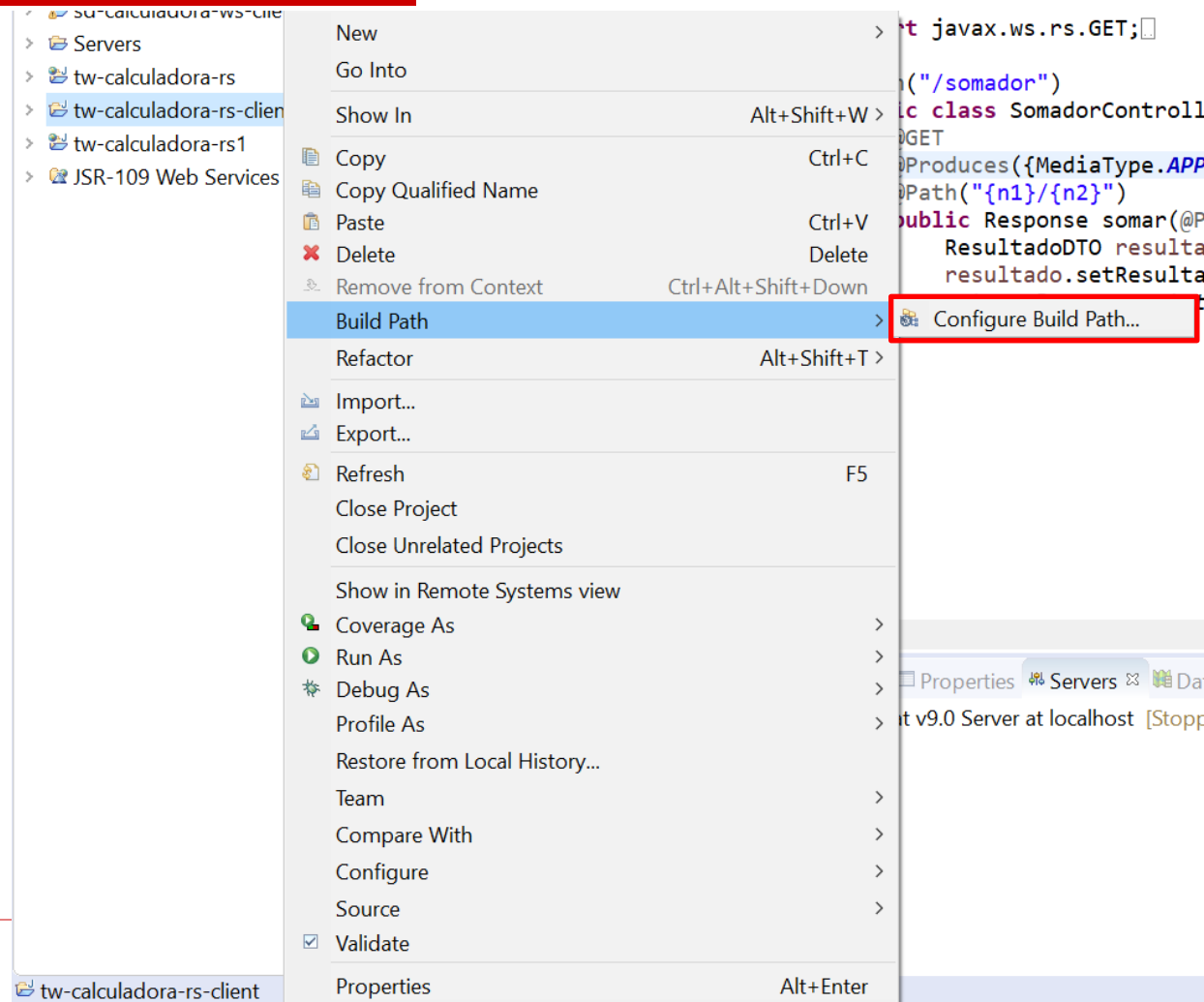
Implementando um cliente para consumo de um serviço RESTful com Jersey

- ❑ Manter a perspectiva java.



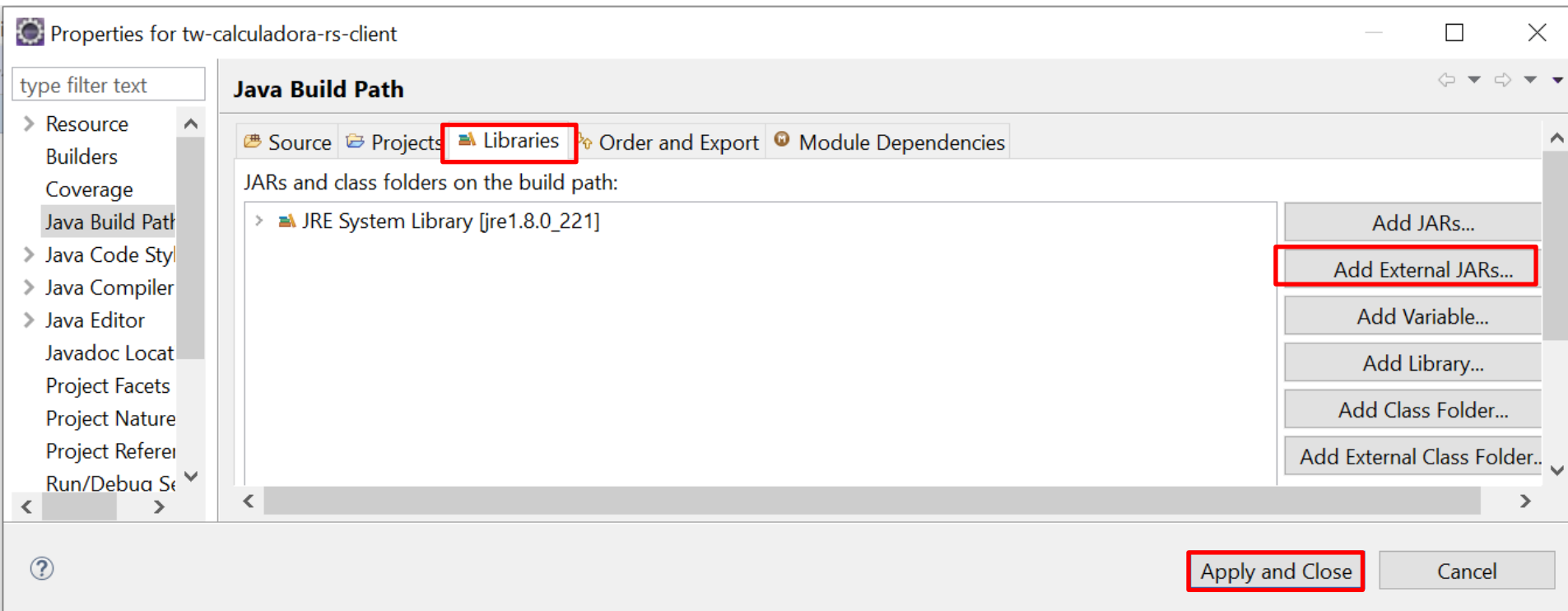
Implementando um cliente para consumo de um serviço RESTful com Jersey

- Vamos agora configurar o build path pra conter todas as bibliotecas do JERSEY



Implementando um cliente para consumo de um serviço RESTful com Jersey

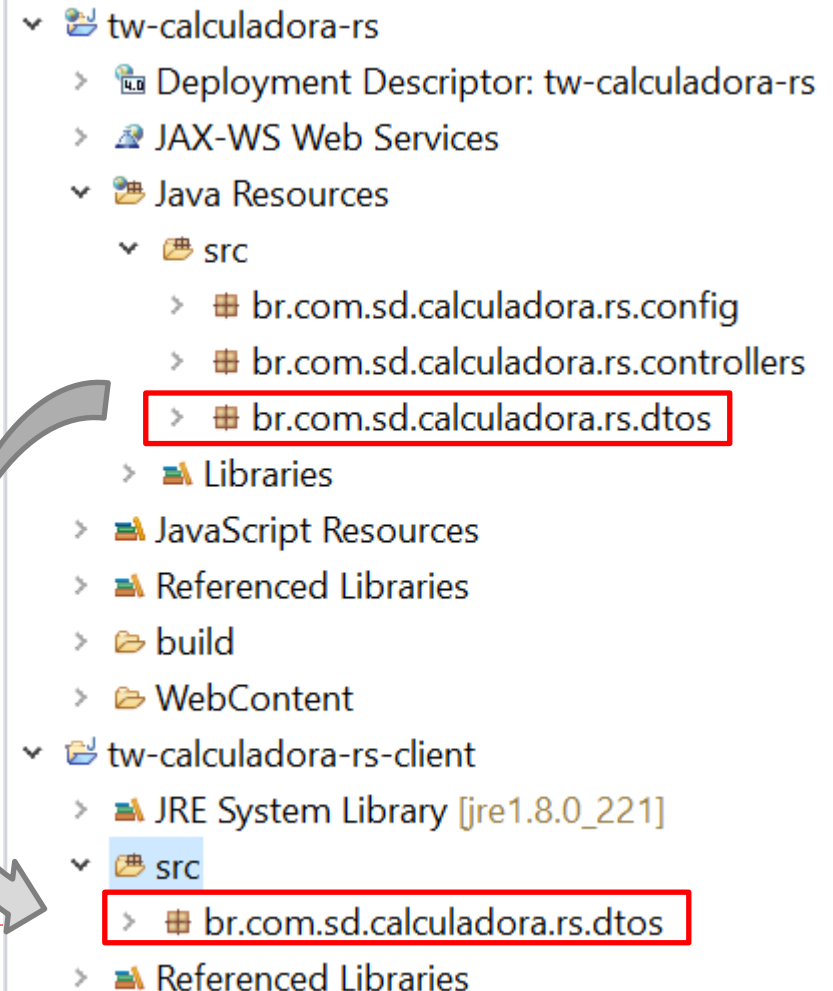
- ❑ Adicionar todos os JARs de cada pasta do JERSEY: "api", "ext", e "lib".



Implementando um cliente para consumo de um serviço RESTful com Jersey

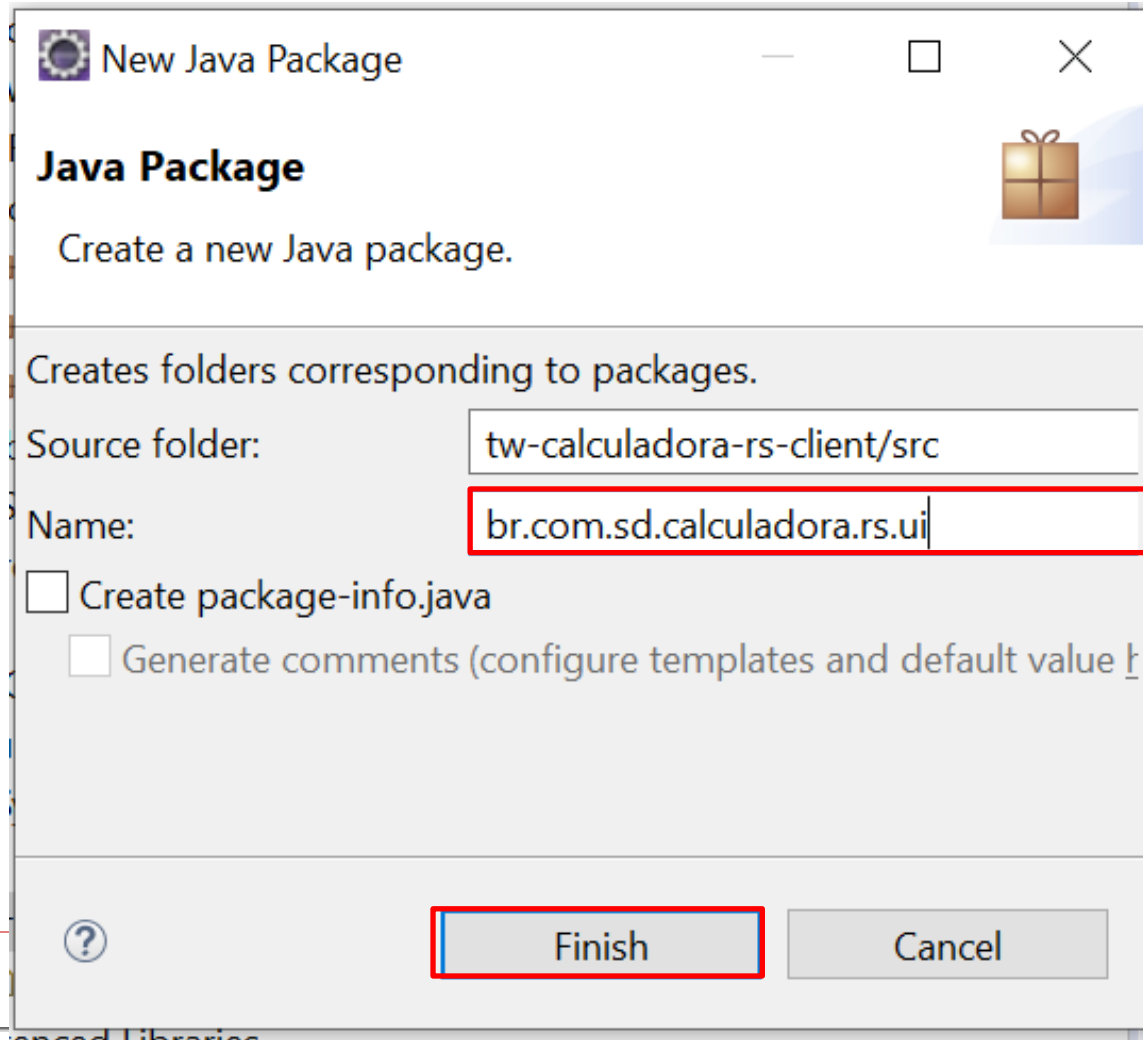
- ❑ Vamos agora fazer uma chamada para o somador. O somador vai devolver uma entidade (que é o DTO).

Copiar o pacote de um projeto para o outro



Implementando um cliente para consumo de um serviço RESTful com Jersey

- ❑ Criar um novo pacote em cima do pacote existente.



Implementando um cliente para consumo de um serviço RESTful com Jersey

- ❑ Em cima do pacote criado, deve criar uma nova classe.

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Implementando um cliente para consumo de um serviço RESTful com Jersey

```
package br.com.sd.calculadora.rs.ui;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Invocation;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;

import br.com.sd.calculadora.rs.dtos.ResultadoDTO;

public class Main {

    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target =
            client.target("http://localhost:8080/tw-calculadora-rs/rest").path("somador/2/3");
        Invocation.Builder invocador = target.request(MediaType.APPLICATION_JSON);
        ResultadoDTO resultado = invocador.get(ResultadoDTO.class);
        System.out.println(resultado.getResultado());
    }
}
```

Implementando um cliente para consumo de um serviço RESTful com Jersey

