FIAP ON

WEBSERVICES &
RESTFUL TECHNOLOGIES

JAX-RS E JAX-WS

EDUARDO GALEGO

3

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

3 JAX-RS E JAX-WS

JAX-RS e JAX-WS são especificações JEE dentro do universo Java.

Especificações JAVA são um conjunto de diretrizes que norteiam os desenvolvedores de software na construção de funcionalidades dentro do ambiente Java. Na prática, são interfaces sem uma implementação.

O desenvolvedor de Web Services utiliza, em seu código-fonte, referências a essa especificação, sem que, necessariamente, faça referência ao código implementador. Depois de o código ser compilado, empacotado e publicado (*deploy)*, acontece no servidor de aplicação a execução do código a partir de um implementador.

A principal vantagem na utilização dessa estratégia é evitar que o código-fonte fique amarrado a um framework específico. Podemos migrar o mesmo pacote para diferentes servidores de aplicação, e o resultado da execução será o mesmo.

Nos exemplos, será utilizado o servidor de aplicação Tomcat, que contém embarcadas as seguintes implementações: Jersey para JAX-RS e Axis2 para JAX-WS.

3.1 Warm UP

Vamos preparar nosso ambiente para desenvolver. Utilizaremos:

JDK versão 8.281: https://www.oracle.com/java/technologies/javase/8u281-relnotes.html;

Servidor de Aplicação Tomcat versão 9: <https://tomcat.apache.org/download-90.cgi>;

IDE Eclipse JEE: <https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-ide-enterprise-java-developers>.

Por padrão, o Eclipse já contém um JRE embarcado. Para alterar para o JDK 8 instalado, siga os seguintes passos:

1. Com o Eclipse aberto, abra o menu **Windows** >> **Preferences**.
2. Pesquise, na barra no canto superior esquerdo, pelo menu **Installed JREs**.
3. Clique sobre o botão Add..., conforme a figura “Configuração de uma nova JRE”:

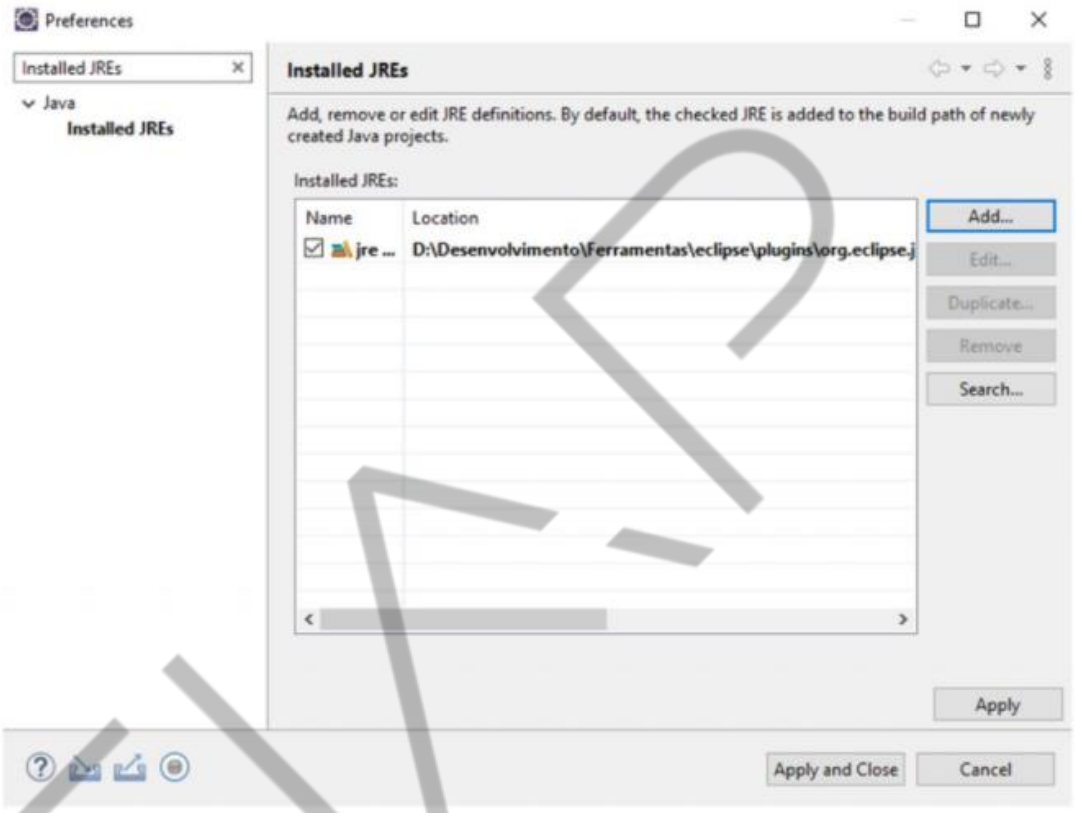

Figura 3.1 – Configuração de uma nova JRE
Fonte: Elaborado pelo autor (2021)

Escolha a opção **Standard VM** e clique em **Next**.

Dentro do campo **JRE home**, digite o diretório padrão da instalação do JDK ou pressione o botão **Directory...** para selecionar a pasta. Exemplo: Para o Windows, o endereço padrão da instalação é **C:\Program Files\Java\jdk1.8.0_281**.

Uma vez informado, o IDE irá se encarregar de preencher os campos restantes, conforme a figura “Configuração de nova JRE no Eclipse com os campos preenchidos”.

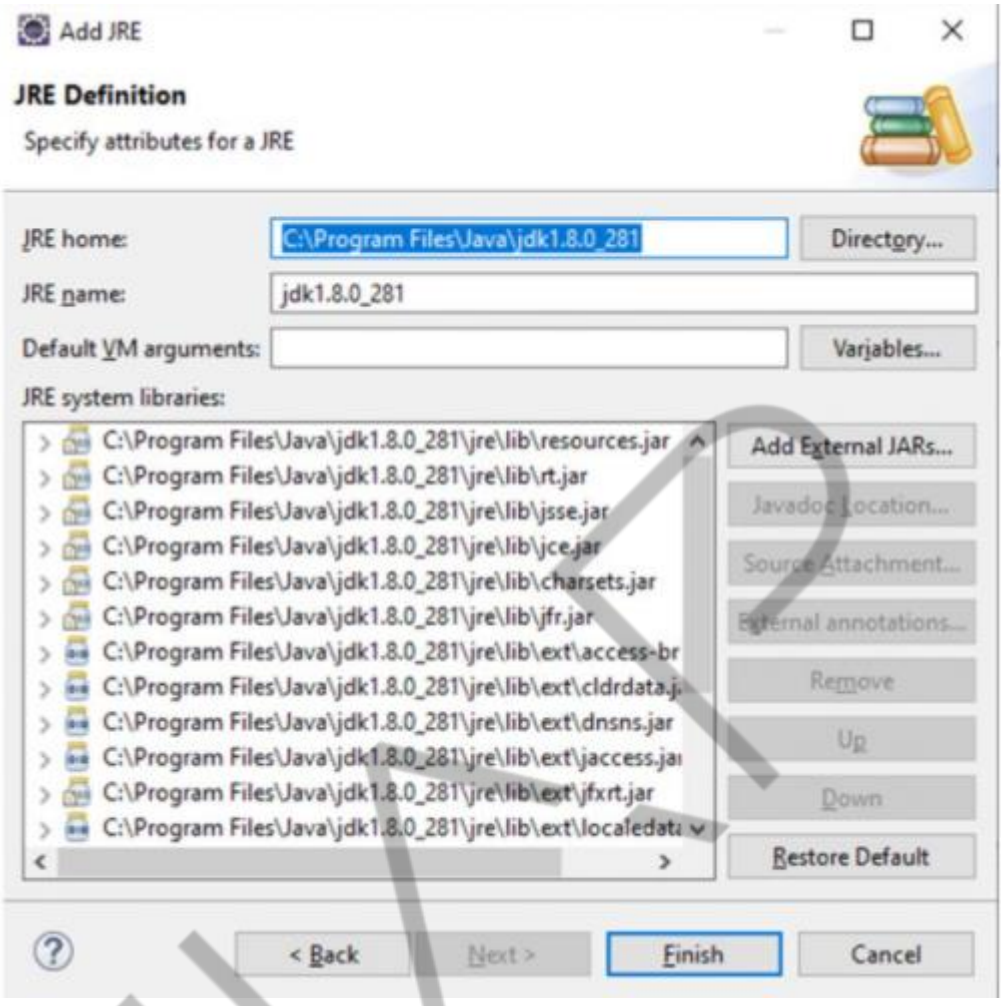

Figura 3.2 – Configuração de nova JRE no Eclipse com os campos preenchidos
Fonte: Elaborado pelo autor (2021)

Pressione o botão **Finish**.

Um novo item deve ser adicionado à lista de JRE instalados. Marque o item recém-criado e clique em **Apply and Close**.

3.2 JAX-RS

Vamos iniciar a apresentação de JAX-WS fazendo um exercício prático que compreende a criação de um Web Service REST utilizando JAX-RS. Sigamos o passo a passo para completar a tarefa.

1. Crie um novo projeto, pressionando o menu **File** > **New** > **Dynamic Web Project**.

2. Informe no campo **Project Name** o valor **FiapOnJaxrsServer** e certifique-se de que **Target runtime** seja **Apache Tomcat v9.0**, conforme a figura “Criação de novo projeto Web Dinâmico (Dynamic Web Project)”.

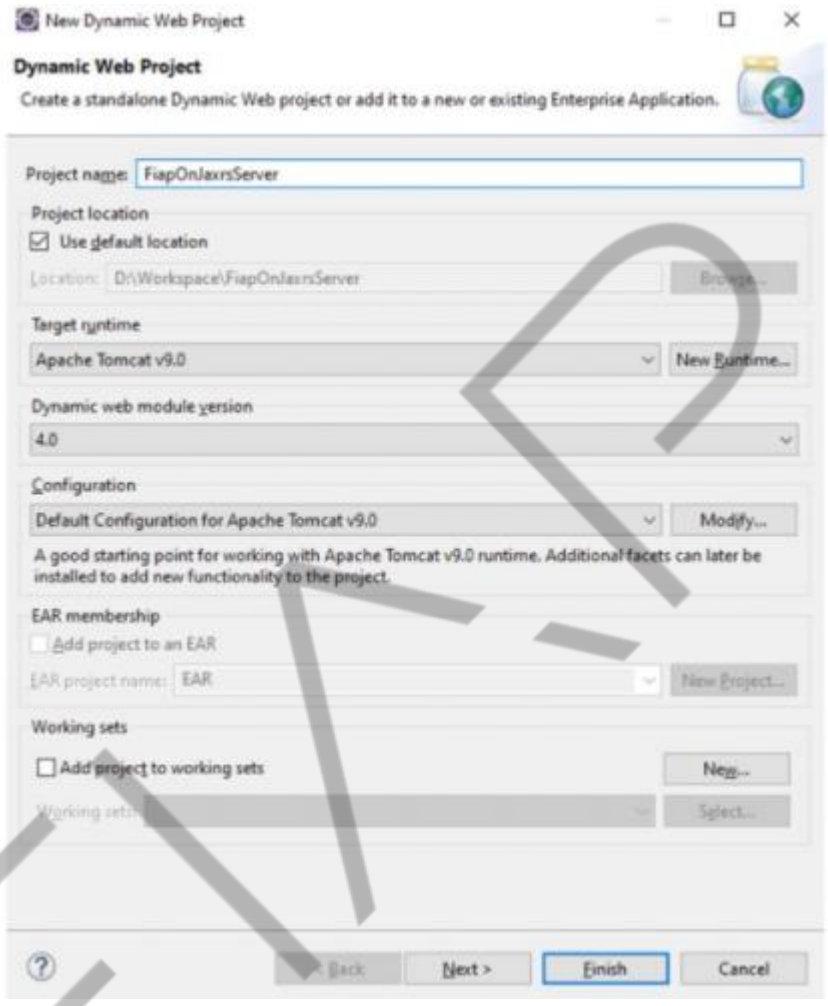

Figura 3.3 – Criação de novo projeto Web Dinâmico-REST
Fonte: Elaborado pelo autor (2021)

3. Clique em **Finish**. Um novo projeto deverá ser exibido em **Project Explorer**.
4. Converta esse projeto em um projeto Maven, pressionando o botão direito do mouse sobre o nome do projeto e, depois, **Configure** > **Convert to Maven Project.**
5. Preencha os campos, conforme a figura “Conversão para projeto Maven – Criação POM”.

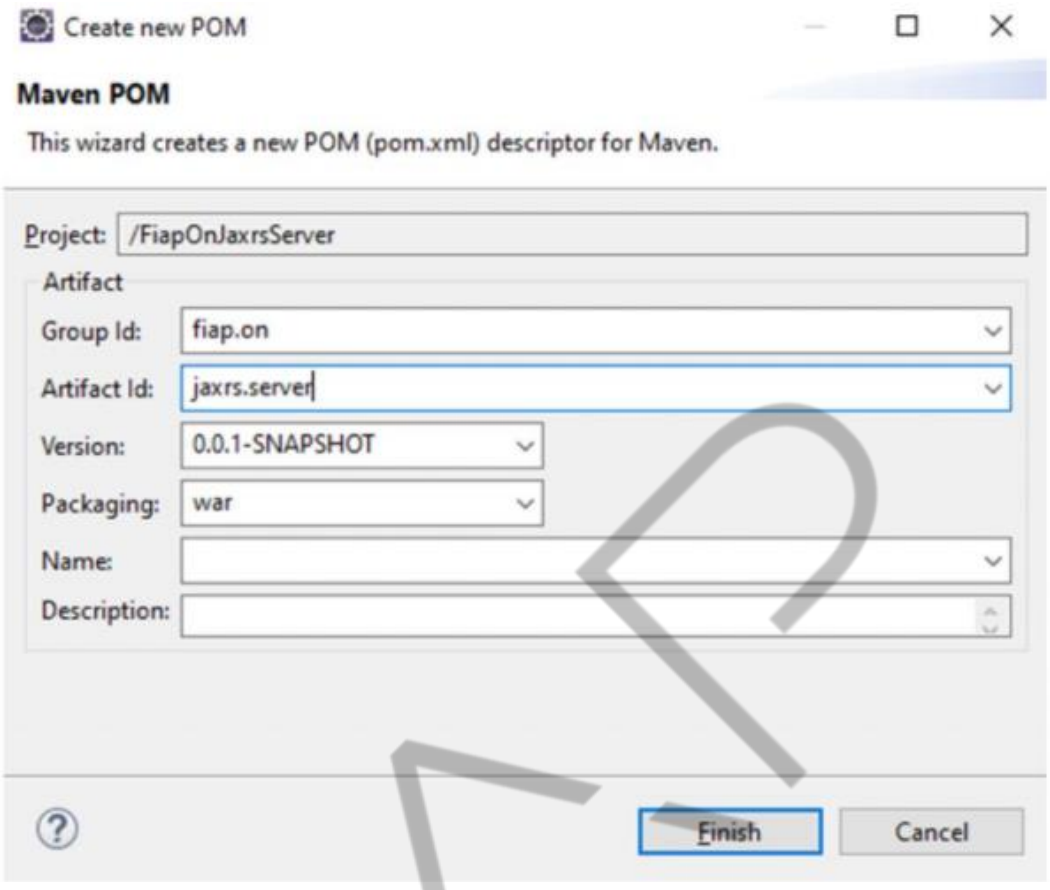

Figura 3.4 – Conversão para projeto Maven – Criação POM (REST)
Fonte: Elaborado pelo autor (2021)

6. Um novo arquivo chamado **pom.xml** deve ser criado na raiz do projeto. Clique nele e adicione as seguintes dependências:

```
<dependency>
<groupId>com.sun.jersey</groupId>
<artifactId>jersey-bundle</artifactId>
<version>1.19.4</version>
</dependency>
<dependency>
<groupId>com.sun.jersey</groupId>
<artifactId>jersey-server</artifactId>
<version>1.19.4</version>
</dependency>
<dependency>
<groupId>com.sun.jersey</groupId>
<artifactId>jersey-core</artifactId>
<version>1.19.4</version>
</dependency>
```

```
<dependency>
<groupId>com.sun.jersey</groupId>
<artifactId>jersey-json</artifactId>
<version>1.19.4</version>
</dependency>
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.6.3</version>
</dependency>
```

Código-fonte 3.1 – Dependências para incluir no arquivo pom.xml
Fonte: Elaborado pelo autor (2021)

7. Salve o arquivo. Automaticamente o IDE deverá começar a baixar as dependências da Web. Aguarde a conclusão dessa tarefa.

8. Crie um arquivo descritor para Web, pressionando o botão direito do mouse sobre o projeto, Java EE Tools > Generate Deployment Descriptor Stub. Um arquivo chamado web.xml deve ser adicionado dentro de WebContent > WEB-INF. Adicione o seguinte trecho ao final do arquivo:

```
<servlet>
<servlet-name>Jersey Web Application</servlet-name>
<servlet-
class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
<init-param>
<param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
<param-value>true</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>Jersey Web Application</servlet-name>
<url-pattern>/*</url-pattern>
</servlet-mapping>
```

Código-fonte 3.2 – Trecho para adicionar no arquivo web.xml
Fonte: Elaborado pelo autor (2021)

9. Salve o arquivo.

10. Crie uma nova classe no nosso projeto, pressionando o botão direito do mouse sobre o projeto, New > Class. Preencha os campos conforme a figura “Criação da classe TasksService”.

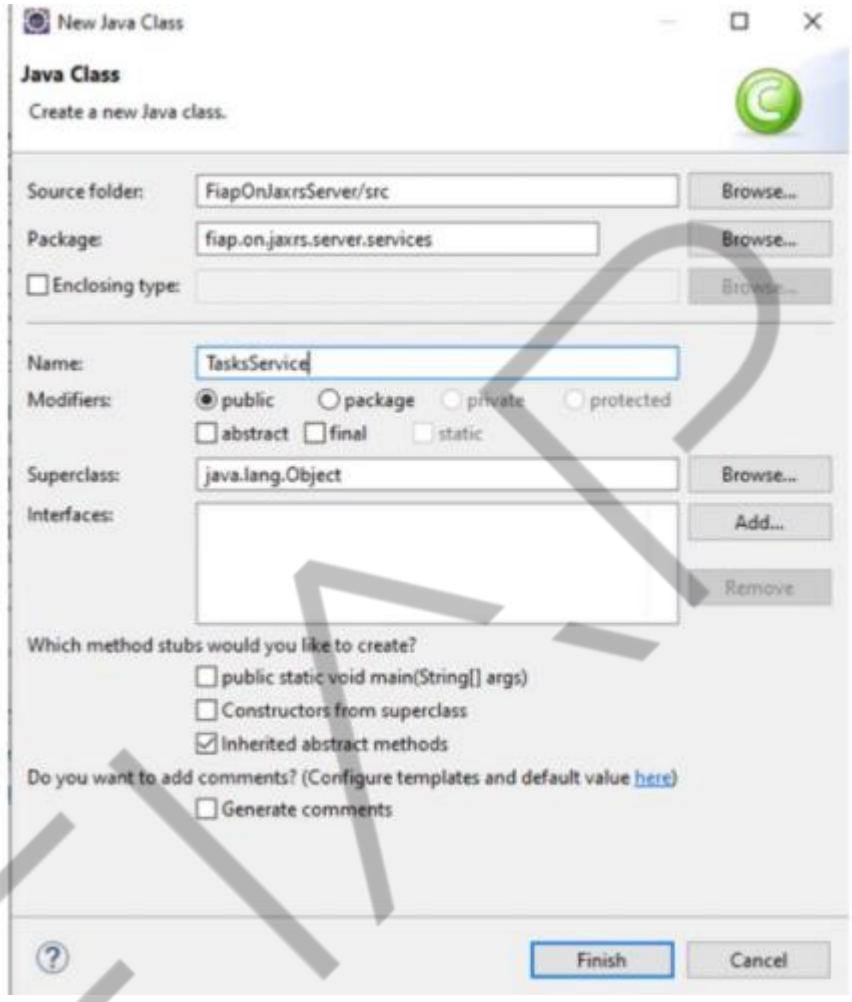

Figura 3.5 – Criação da classe TasksService
Fonte: Elaborado pelo autor (2021)

11. Clique em Finish. Preencha a classe conforme código abaixo:

```
package fiap.on.jaxrs.sample.services;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
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 javax.ws.rs.core.Response.Status;
```

```
import fiap.on.jaxrs.sample.models.Task;
import fiap.on.jaxrs.sample.models.TaskList;

@Path("/tasks")
public class TasksService {

private static final List<Task> REPO = new ArrayList<Task>();

@GET
@Path("/echo")
public String success() {
return "success";
}

@GET
@Produces(MediaType.APPLICATION_JSON)
public Response listAll() {
return Response.status(Status.OK).entity(new TaskList(REPO)).build();
}

@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public Response findById(@PathParam("id") final Integer id) {
final Optional<Task> opt = REPO.stream().filter(t ->
id.equals(t.getId())).findFirst();
if (!opt.isPresent()) {
return Response.status(Status.NOT_FOUND).build();
}
return Response.status(Status.OK).entity(opt.get()).build();
}

@POST
@Consumes(MediaType.APPLICATION_JSON)
public Response create(final Task task) {
REPO.add(task);
return Response.status(Status.CREATED).build();
}
}
```

Código-fonte 3.3 – Classe TasksService
Fonte: Elaborado pelo autor (2021)

12. Crie outras duas classes, representando o modelo de dados. Segue o código:

```
package fiap.on.jaxrs.sample.models;

import java.util.Date;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Task {

private Integer id;
private String description;
private Date date;
private Boolean active;

      // Getters e Setters omitidos!
}
```

Código-fonte 3.4 – Classe Task
Fonte: Elaborado pelo autor (2021)

```
package fiap.on.jaxrs.sample.models;

import java.util.List;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class TaskList {

private List<Task> result;

public TaskList() {
super();
}

public TaskList(List<Task> result) {
this();
this.result = result;
}

      // Getters e Setters omitidos!
}
```

Código-fonte 3.5 – Classe TaskList
Fonte: Elaborado pelo autor (2021)

13. Faça o *deploy* deste projeto clicando o botão direito do mouse sobre o projeto, **Run As** > **Run on Server**.
14. Na tela que se abre, escolha a opção **Apache Tomcat v9.0** e clique em **Next**.
15. Clique no botão **Browse...** e escolha a pasta raiz da instalação do Tomcat.
16. Uma vez preenchido, o IDE se encarrega de preencher os campos restantes. Clique em **Next >**.
17. Na tela seguinte, certifique-se de que o projeto está dentro da lista de **Configured**. Clique em **Finish**. Para os usuários Windows, talvez seja necessário dar permissão para acesso à porta 8080.
18. Após o *deploy*, um navegador será aberto dentro da própria IDE. Para certificar que houve sucesso, digite na barra de endereço: http://localhost:8080/jaxrs.server/tasks/echo. Deverá aparecer a palavra "*success*" na resposta.

A partir deste momento, o Web Service REST já está no ar. Para fazer alguns testes, utilize um software Rest Client como Postman, SOAP UI ou Insomnia. Tente incluir algumas tarefas e depois listá-las.

O código-fonte completo pode ser obtido em: https://github.com/prof-eduardo-galego/fiapon-jaxrs-server.

3.3 JAX-WS

Vamos falar de JAX-WS, fazendo um exercício prático que compreende a criação de um Web Service SOAP utilizando JAX-WS. Usando o Eclipse, siga o passo a passo para completar a tarefa.

1. Crie um novo projeto, pressionando o menu **File** > **New** > **Dynamic Web Project.**
2. Informe, no campo **Project Name,** o valor **FiapOnJaxwsServer**, conforme a figura “Criação de novo projeto Web Dinâmico-SOAP”:

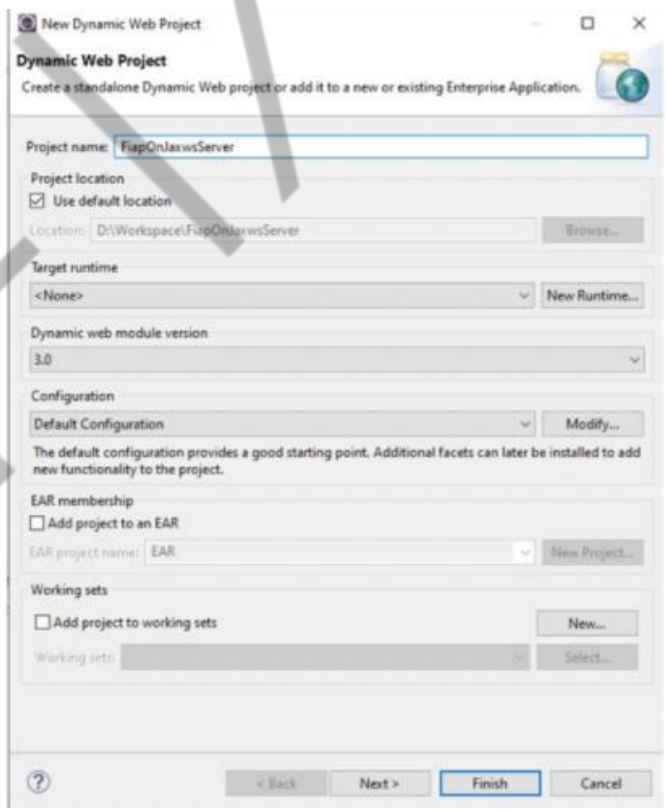

Figura 3.6 – Criação de novo projeto Web Dinâmico-SOAP
Fonte: Elaborado pelo autor (2021)

3. Clique em **Finish**. Um novo projeto deverá ser exibido em **Project Explorer**.
4. Converta esse projeto em um projeto Maven, pressionando o botão direito do mouse sobre o projeto no Project, depois **Configure** > **Convert to Maven Project**.
5. Preencha os campos, conforme a figura “Conversão para projeto Maven – Criação POM (SOAP)”:

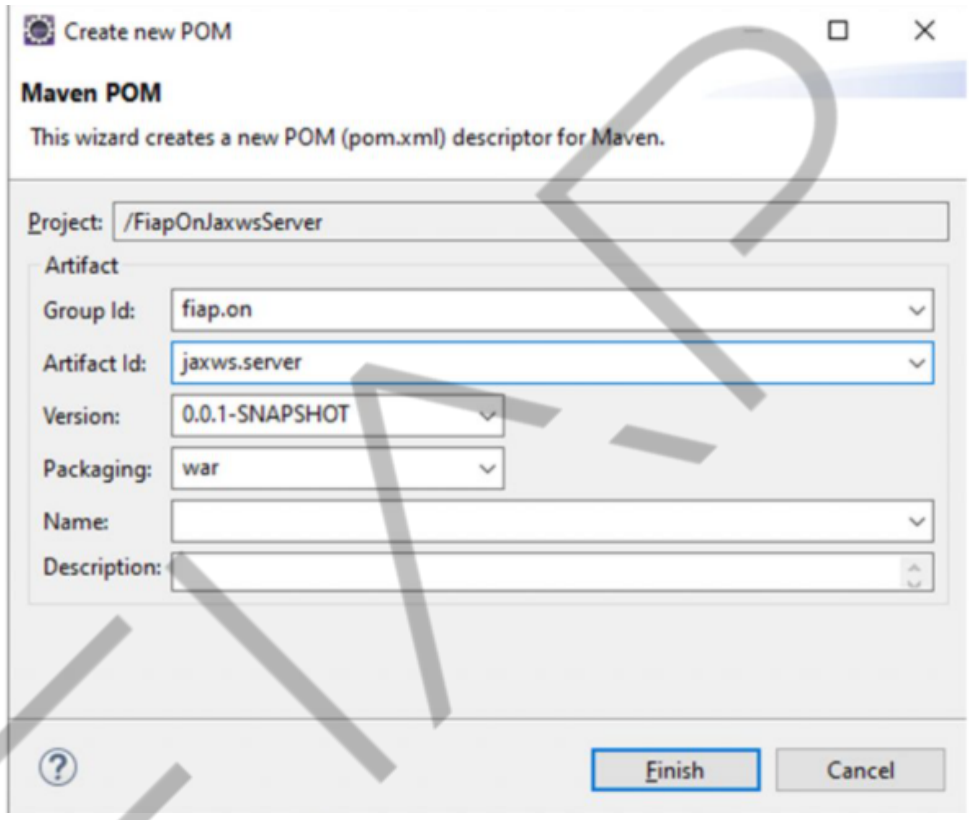

Figura 3.7 – Conversão para projeto Maven – Criação POM (SOAP)
Fonte: Elaborado pelo autor (2021)

6. Um novo arquivo chamado **pom.xml** deve ser criado na raiz do projeto. Clique nele e adicione a seguinte dependência:

```
<dependency>
<groupId>com.sun.xml.ws</groupId>
<artifactId>jaxws-rt</artifactId>
<version>2.3.2</version>
</dependency>
```

Código-fonte 3.6 – Inclusão da dependência jaxws-rt no arquivo pom.xml
Fonte: Elaborado pelo autor (2021)

7. Salve o arquivo. Automaticamente o IDE deverá começar a baixar a dependência da Web. Aguarde a conclusão dessa tarefa.

8. Crie um arquivo descritor para Web, pressionando o botão direito do mouse sobre o projeto, **Java EE Tools** > **Generate Deployment Descriptor Stub**. Um arquivo chamado **web.xml** deve ser adicionado dentro de **WebContent** > **WEB-INF**. Adicione o seguinte trecho ao final do arquivo:

```
<listener>
<listener-class>
com.sun.xml.ws.transport.http.servlet.WSServletContextListener
</listener-class>
</listener>
<servlet>
<servlet-name>WSServlet</servlet-name>
<servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-
class>
</servlet>
<servlet-mapping>
<servlet-name>WSServlet</servlet-name>
<url-pattern>/*</url-pattern>
</servlet-mapping>
```

Código-fonte 3.7 – Descritor web.xml (SOAP)
Fonte: Elaborado pelo autor (2021)

9. Salve o arquivo.
10. Crie um novo arquivo chamado **sun-jaxws.xml,** dentro de **WebContent** > **WEB-INF,** com o seguinte conteúdo:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
<endpoint name="TasksService"
implementation="fiap.on.jaxws.server.services.TasksServiceImpl" url-
pattern="/taskservice" />
</endpoints>
```

Código-fonte 3.8 – Conteúdo do arquivo sun-jaxws.xml
Fonte: Elaborado pelo autor (2021)

11. Salve o arquivo.
12. Crie uma classe Java representando o nosso modelo de dados. Para isso, pressione o botão direito do mouse sobre o projeto **New** > **Java Class**. Preencha os campos conforme a figura “Criação de classe Java para testar a aplicação”:

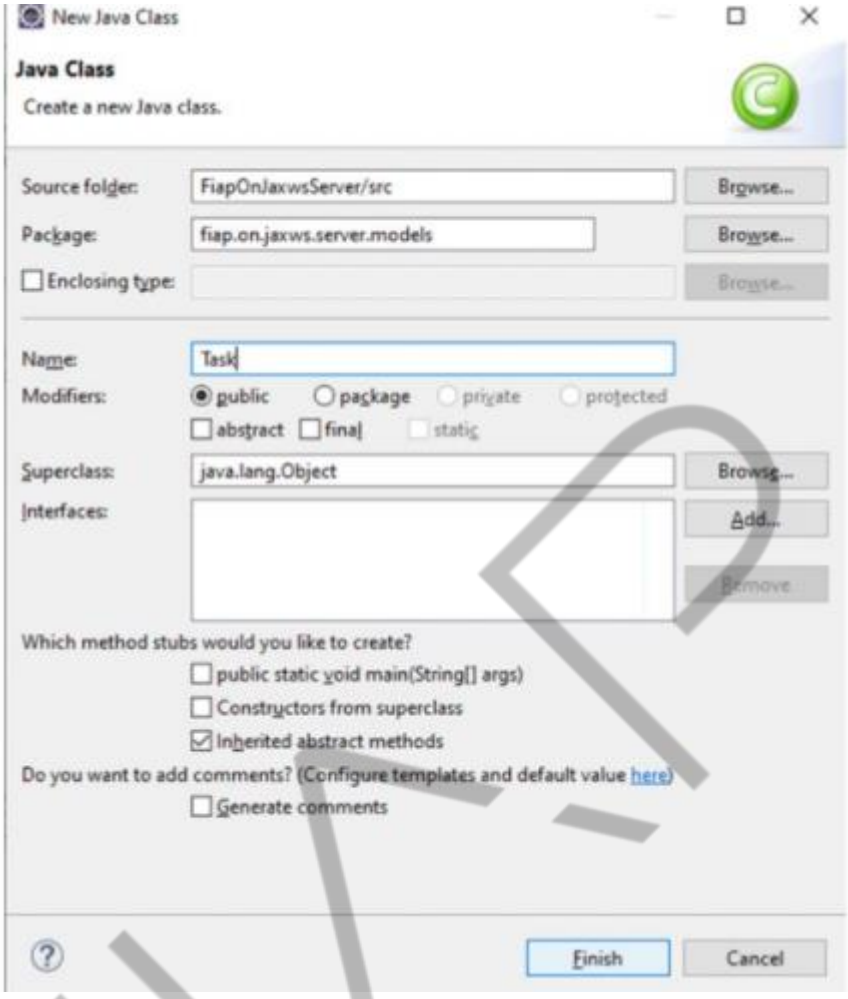

Figura 3.8 – Criação de classe Java para testar a aplicação
Fonte: Elaborado pelo autor (2021)

13. Preencha a classe gerada com o seguinte conteúdo:

```
package fiap.on.jaxws.server.models;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Task {

private Integer id;
private String description;

// Omitindo Getters e Setters
}
```

Código-fonte 3.9 – Conteúdo da classe Task
Fonte: Elaborado pelo autor (2021)

14. Crie uma nova interface no nosso projeto, pressionando o botão direito do mouse sobre o projeto **New** > **Interface**. Preencha os campos conforme a figura "Criação da interface TasksService":

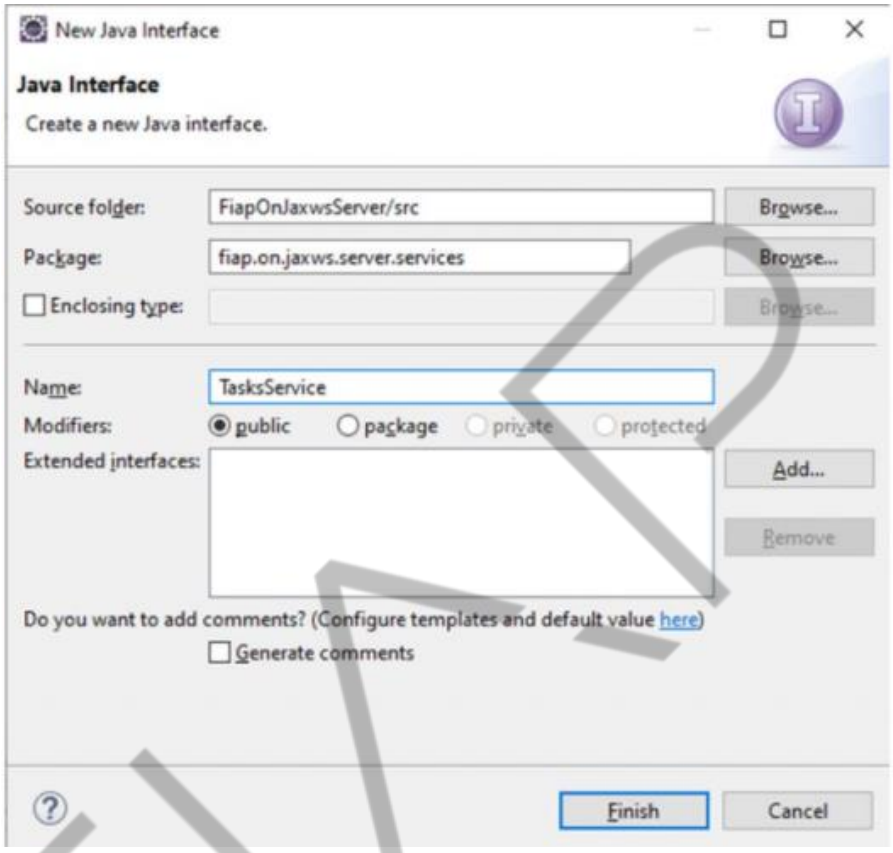

Figura 3.9 – Criação da interface TasksService
Fonte: Elaborado pelo autor (2021)

15. Clique em **Finish**. Preencha a classe conforme o código-fonte "Conteúdo da interface TasksService":

```
package fiap.on.jaxws.server.services;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

import fiap.on.jaxws.server.models.Task;

@WebService
public interface TasksService {

@WebMethod
void createTask(Task task);

@WebMethod
List<Task> listAll();
```

```
}
```

Código-fonte 3.10 – Conteúdo da interface TasksService
Fonte: Elaborado pelo autor (2021)

16. Crie a implementação dessa classe. Para isso, pressione o botão direito do mouse sobre o projeto **New** > **Java Class**. Preencha os campos conforme a figura “Criação da classe TasksServiceImpl”:

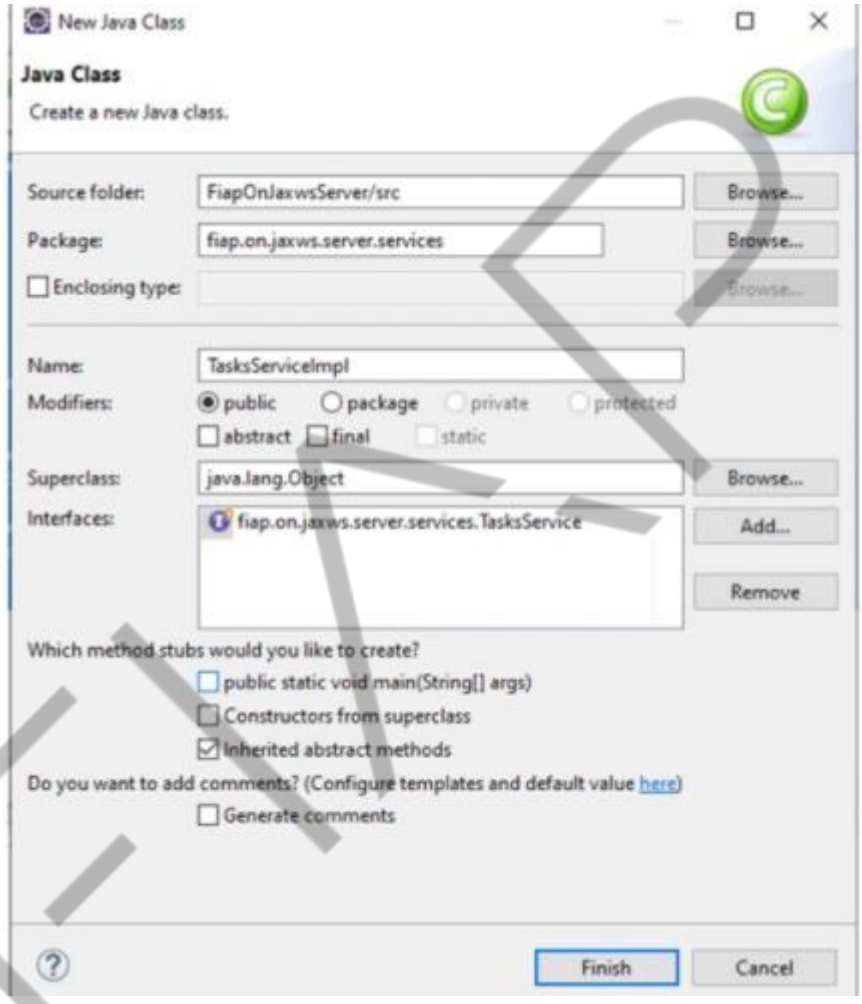

Figura 3.10 – Criação da classe TasksServiceImpl
Fonte: Elaborado pelo autor (2021)

17. Preencha a classe criada com o código-fonte “Conteúdo da interface TasksServiceImpl”.

```
package fiap.on.jaxws.server.services;

import java.util.ArrayList;
import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebService;

import fiap.on.jaxws.server.models.Task;
```

```
@WebService(endpointInterface =
"fiap.on.jaxws.server.services.TasksService")
public class TasksServiceImpl implements TasksService {

private static final List<Task> REPO = new ArrayList<Task>();

@WebMethod
@Override
public void createTask(Task task) {
REPO.add(task);
}

@WebMethod
@Override
public List<Task> listAll() {
return REPO;
}
}
```

Código-fonte 3.11 – Conteúdo da interface TasksServiceImpl
Fonte: Elaborado pelo autor (2021)

18. Faça o *deploy* do projeto, clicando o botão direito do mouse sobre ele e depois, **Run As** > **Run on Server**.

Abra o navegador e verifique se está disponível o WSDL da aplicação no seguinte endereço: http://localhost:8080/jaxws.server/taskservice?wsdl.

Utilizando um software no estilo *SOAP Client* (por exemplo: SOAP UI), crie um projeto informando o endereço do WSDL e faça chamadas de testes.

O código-fonte completo pode ser consultado em: https://github.com/prof-eduardo-galego/fiapon-jaxws-server.

Conclusão

Neste capítulo criamos dois projetos de Web Services em Java. Um REST utilizando a especificação JAX-RS e outro SOAP utilizando a especificação JAX-WS. Bons estudos e até a próxima!

REFERÊNCIAS

JCP. **JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services.** 2014. Disponível em: <https://jcp.org/en/jsr/detail?id=339>. Acesso em 09 fev. 2021.

JCP. **JSR 224: JavaTM API for XML-Based Web Services (JAX-WS) 2.0.** 2017. Disponível em: <https://jcp.org/en/jsr/detail?id=224>. Acesso em 09 fev. 2021.

GLOSSÁRIO

JAX-WS	Java API for XML Web Services
JAX-RS	Java API for RESTful Web Services
IDE	Integrated Development Environment
JEE	Java Platform, Enterprise Edition
JDK	Java Development Kit
JRE	Java Runtime Environment