



Universidade Federal do Ceará – Campus de Quixadá

Disciplina: Sistemas Distribuídos - 2021.1

Prof. Dr. Antonio Rafael Braga

Tutorial *EJB – Java*

Escopo:

- 1) Downloads
- 2) Instalações
- 3) Configurações
- 4) Como criar interface EJB
- 5) Como chamar método de uma classe remota
- 6) Rodar o exemplo do tutorial

1- Downloads

- Para nosso trabalho vamos utilizar o Eclipse EE - O download pode ser feito no link <https://www.eclipse.org/downloads/packages/>
- Usaremos o server JBoss - Wildfly. O download [wildfly 10.1.0.Final](#)
- Utilizaremos o JDK Kit 8 - Download: [JDK-Kit-8](#)

2- Instalação

• 1º Passo - JDK:

Executando o .exe do jdk, a instalação é simples Next → Next → Próximo → Close. Ele vai instalar diretamente no “C:/Arquivos de Programas/”

• 2º Passo - Eclipse:

Descompactar o .zip vai gerar uma pasta, entrar e executar o “eclipse.exe”. O eclipse vai abrir e você escolhe o diretório para os projetos ou fica no padrão “C:/Users/(Usuario)/”.

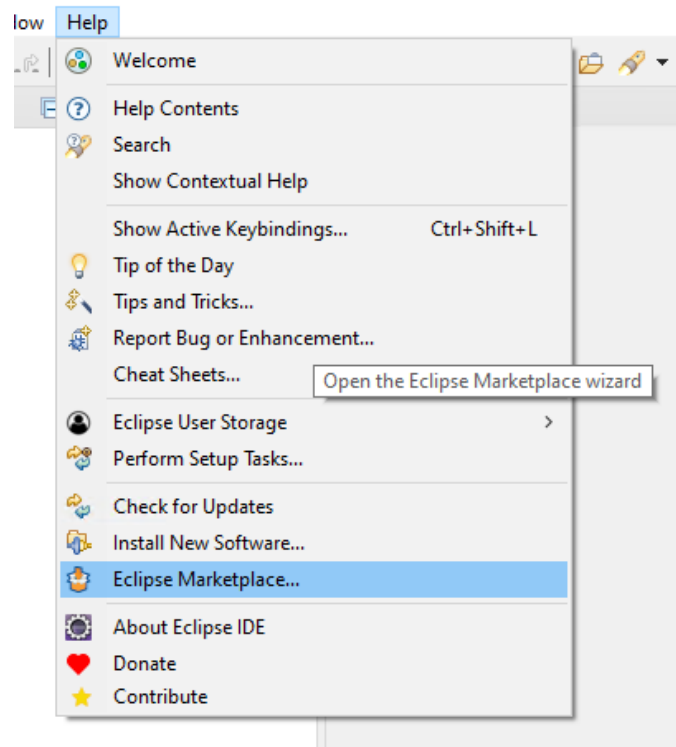
• 3º Passo - Wildfly:

Descompactar o .zip vai gerar uma pasta e depois iremos utilizá-la.

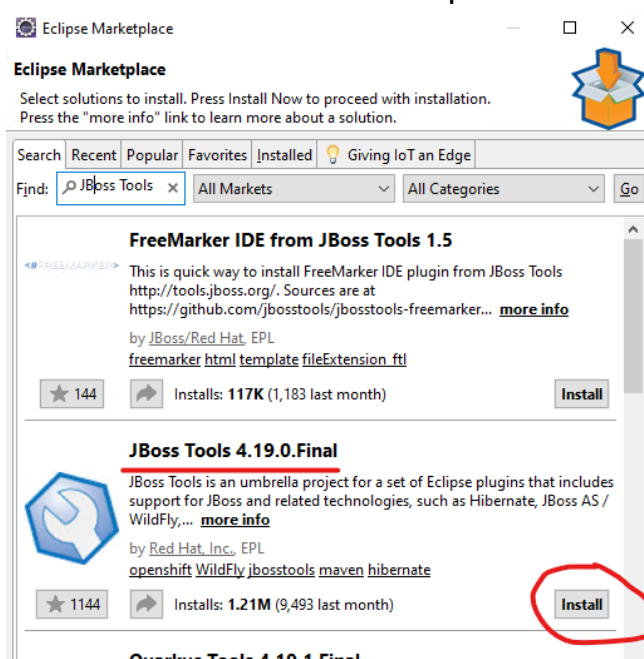
3- Configurações

- JBoss:

Ao abrir o eclipse abre o menu “Help → Eclipse Marketplace...”
digite “JBoss” na caixa de filtro e aperte “Enter”



Escolha “JBoss Tools 4.19.0.Final” e clique no “install”

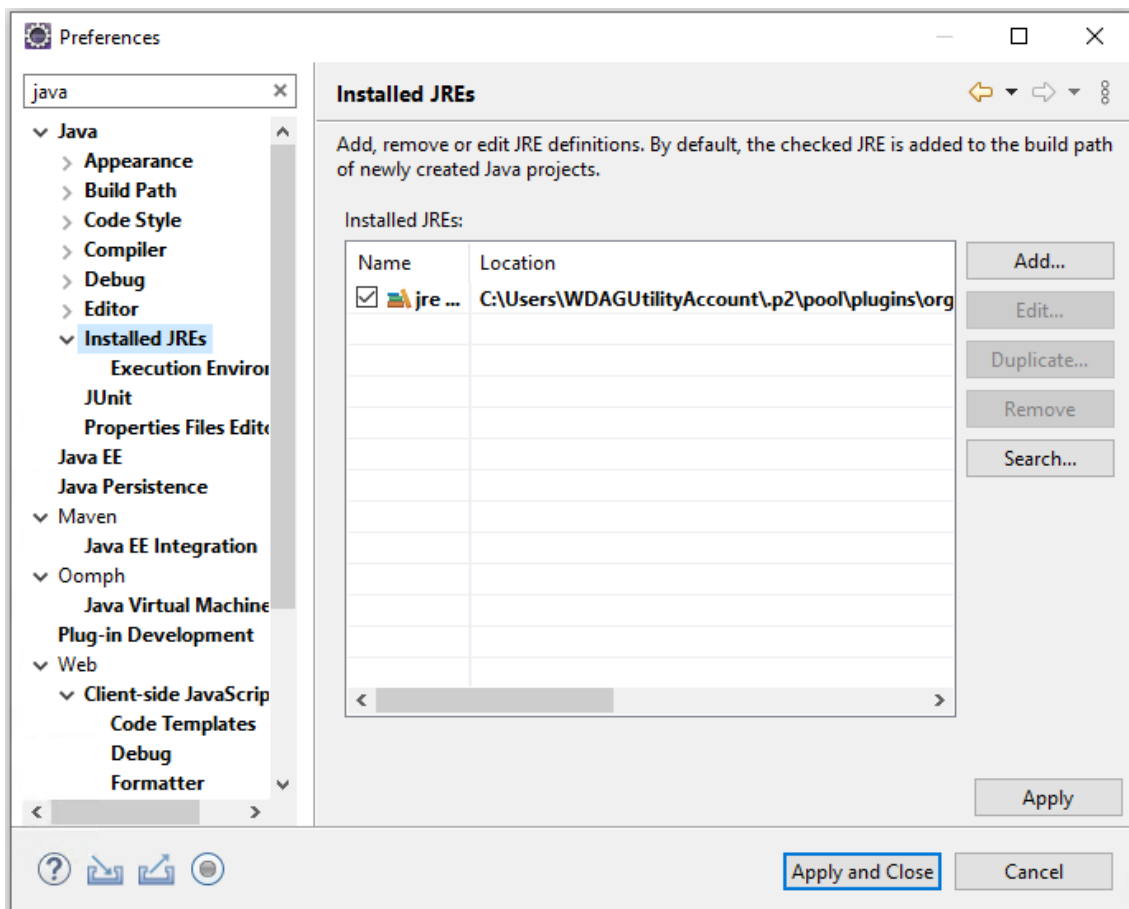


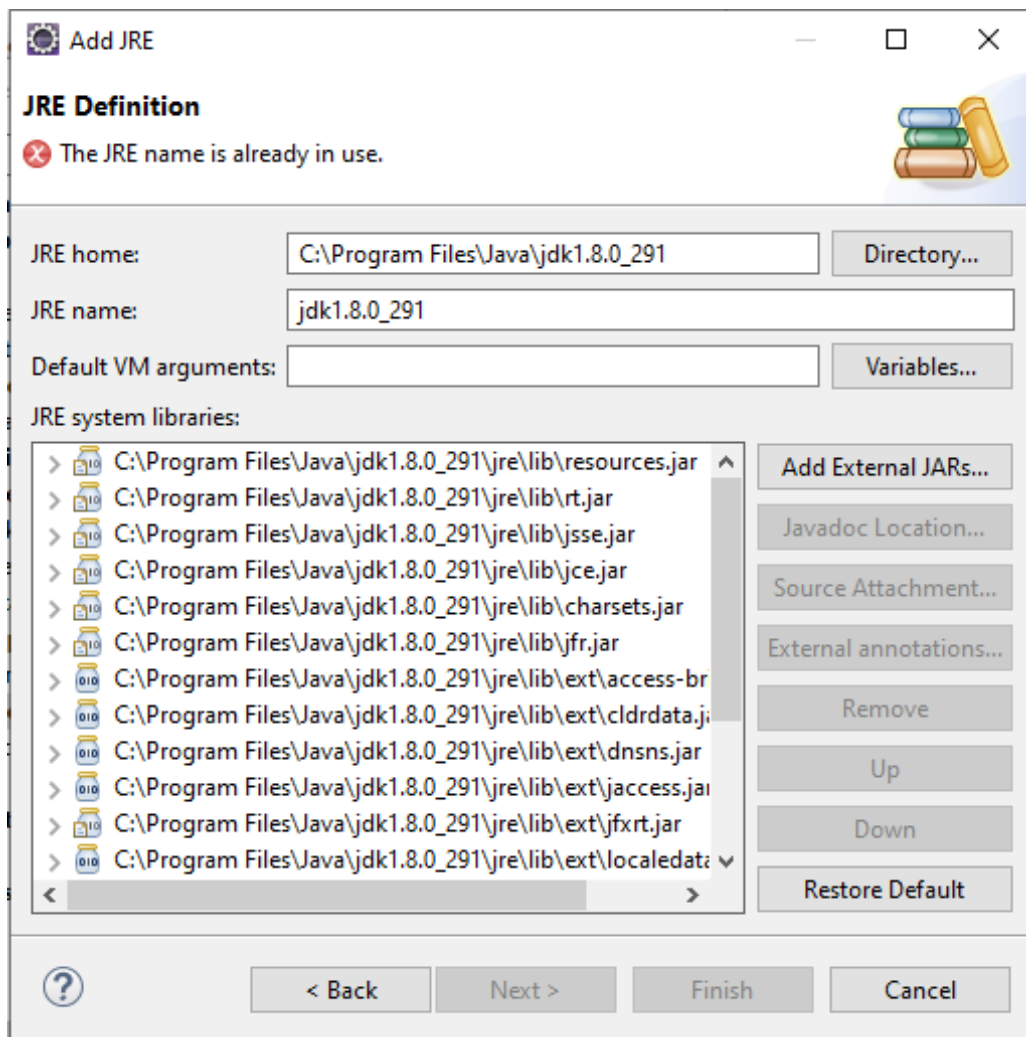
Confirme, aceite a licença e “Finish”

OBS: vai demorar um pouco e ao terminar vai pedir para reiniciar o eclipse. Ao voltar já estará instalado o JBoss.

- **JDK:**

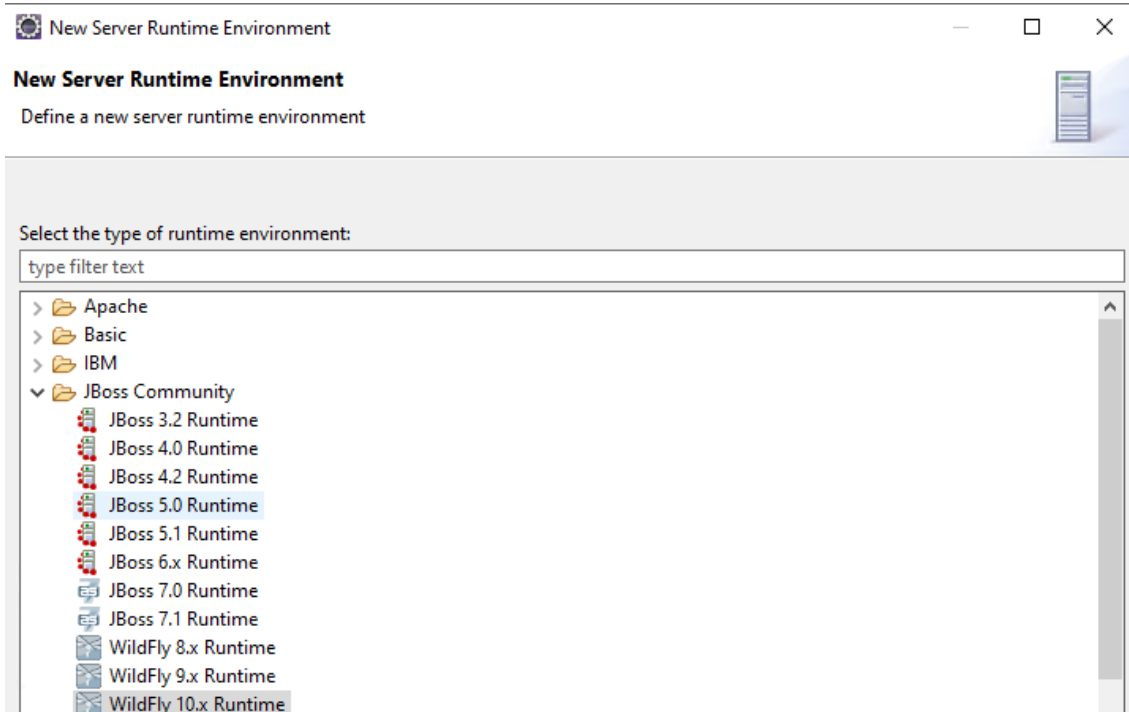
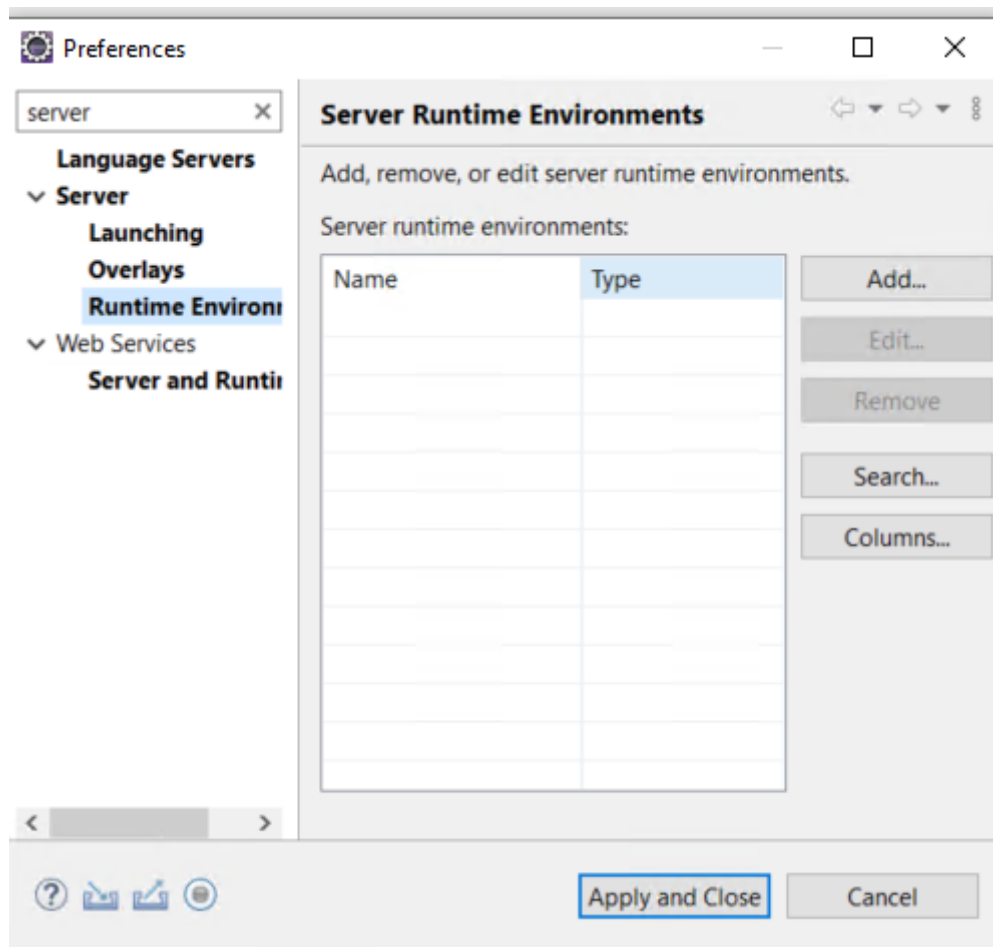
Para adicionar o JDK que instalamos precisamos ir no menu “*Window → Preferences*” digite Java e procure “*Installed JREs*”, abrirá uma janela. “Add” escolha “*Standard VM → Next*” Busque o diretório que instalamos o JDK “*C:\Program Files\Java\jdk1.8.0_291*” → “Finish”. Marque o checkbox do novo JDK 1.8.0_291 “Apply”





- **Wildfly Server**


Para adicionar o Server Wildfly que baixamos precisamos ir no menu “Window → Preferences” digite Server procure “Runtime Environments” botão “Add” busque na pasta “JBoss Community → Wildfly 10x Runtime → Next” aperte “Browser” e procure a pasta que baixamos e descompactamos do wildfly. Marque “Alternate JRE:” escolha “jdk1.8.0_291” e “Finish”



New Server Runtime Environment

JBoss Runtime

WildFly Application Server 10.x



A JBoss Server runtime references a JBoss installation directory. It can be used to set up classpaths for projects which depend on this runtime, as well as by a "server" which will be able to start and stop instances of JBoss.

Name

WildFly 10.x Runtime

Home Directory [Download and install runtime...](#)

C:\Users\Delano\Desktop\EJB\wildfly-10.1.0.Final Browse...

Runtime JRE

☐ Execution Environment: JavaSE-1.8 Environments...

☒ Alternate JRE: jdk1.8.0_291 Installed JREs...

Server base directory: standalone Browse...

Configuration file: standalone.xml Browse...

?< BackNext >FinishCancel

4- Como criar interface EJB

- Criar um Projeto EJB dá um nome, em nosso exemplo temos “ServerEJB” e “Finish”

New EJB Project

Create an EJB Project and add it to a new or existing Enterprise Application.

Project name:

Project location

☒ Use default location

Location:

Target runtime

EJB module version

Configuration

A good starting point for working with WildFly 10.x Runtime runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

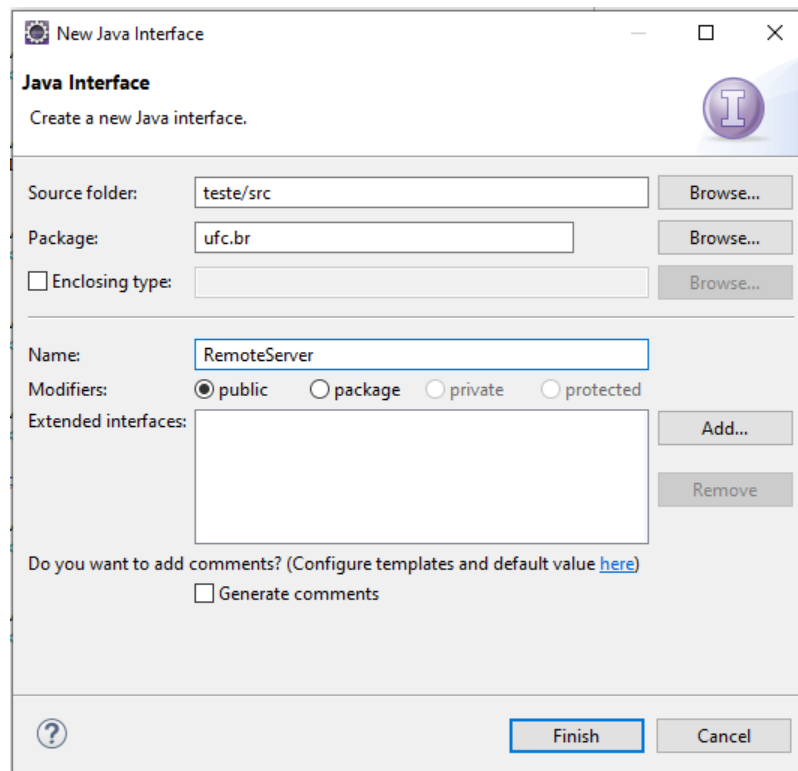
EAR project name:

Working sets

☐ Add project to working sets

Working sets:

- Adicione um novo pacote no “*ejbModule*” que fica dentro do projeto gerado. Usaremos “*ufc.br*”. Crie uma nova interface “*RemoteServer*”. Também uma nova Classe “*RemoteServerImpl*”



The "New Java Interface" dialog box is shown. It has a title bar with a question mark icon and standard window controls. The main title is "Java Interface" with a sub-header "Create a new Java interface." and a purple 'I' icon. The "Source folder:" field contains "teste/src" with a "Browse..." button. The "Package:" field contains "ufc.br" with a "Browse..." button. There is an unchecked checkbox for "Enclosing type:" with a "Browse..." button. The "Name:" field contains "RemoteServer". The "Modifiers:" section has radio buttons for "public" (selected), "package", "private", and "protected". The "Extended interfaces:" list is empty, with "Add..." and "Remove" buttons. A checkbox for "Generate comments" is unchecked, with a link to "here" for configuration. At the bottom are "Finish" and "Cancel" buttons.

New Java Interface

Create a new Java interface.

Source folder: teste/src Browse...

Package: ufc.br Browse...

☐ Enclosing type: Browse...

Name: RemoteServer

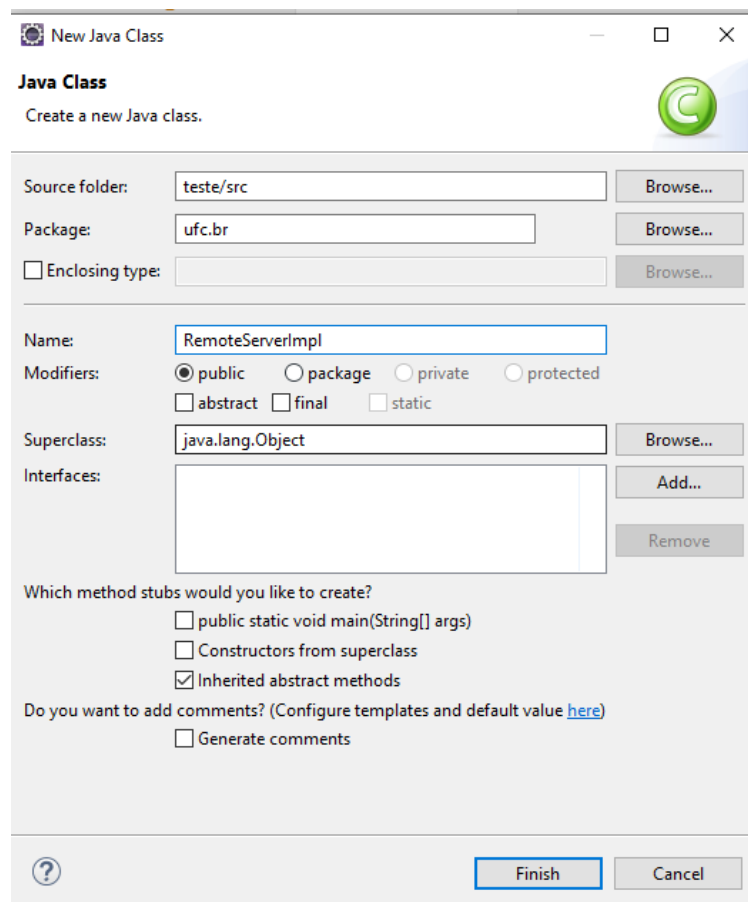
Modifiers: ☒ public ☐ package ☐ private ☐ protected

Extended interfaces: Add... Remove

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Finish Cancel



The "New Java Class" dialog box is shown. It has a title bar with a question mark icon and standard window controls. The main title is "Java Class" with a sub-header "Create a new Java class." and a green 'C' icon. The "Source folder:" field contains "teste/src" with a "Browse..." button. The "Package:" field contains "ufc.br" with a "Browse..." button. There is an unchecked checkbox for "Enclosing type:" with a "Browse..." button. The "Name:" field contains "RemoteServerImpl". The "Modifiers:" section has radio buttons for "public" (selected), "package", "private", and "protected", and checkboxes for "abstract", "final", and "static". The "Superclass:" field contains "java.lang.Object" with a "Browse..." button. The "Interfaces:" list is empty, with "Add..." and "Remove" buttons. The "Which method stubs would you like to create?" section has checkboxes for "public static void main(String[] args)", "Constructors from superclass", and "Inherited abstract methods" (checked). A checkbox for "Generate comments" is unchecked, with a link to "here" for configuration. At the bottom are "Finish" and "Cancel" buttons.

New Java Class

Create a new Java class.

Source folder: teste/src Browse...

Package: ufc.br Browse...

☐ Enclosing type: Browse...

Name: RemoteServerImpl

Modifiers: ☒ public ☐ package ☐ private ☐ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Finish Cancel

Dentro da Interface temos as assinaturas dos métodos.

```
package ufc.br;

import javax.ejb.Remote;

@Remote
public interface RemoteServer {

    public String getString(String nome);

}
```

Dentro da Classe temos as implementações

```
package ufc.br;

import javax.ejb.Stateless;

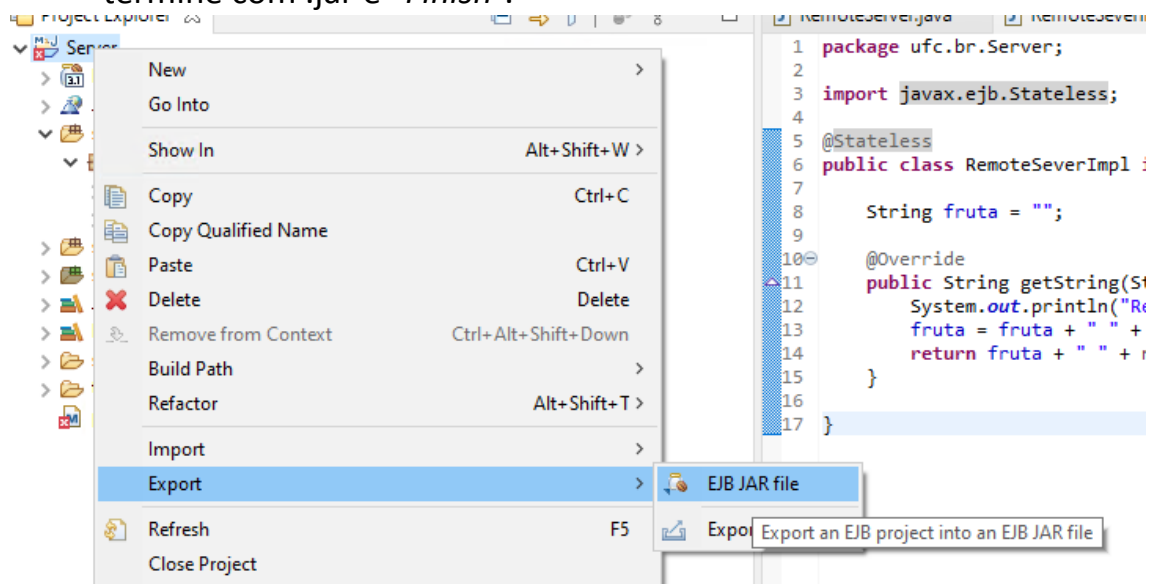
@Stateless
public class RemoteServerImpl implements RemoteServer {

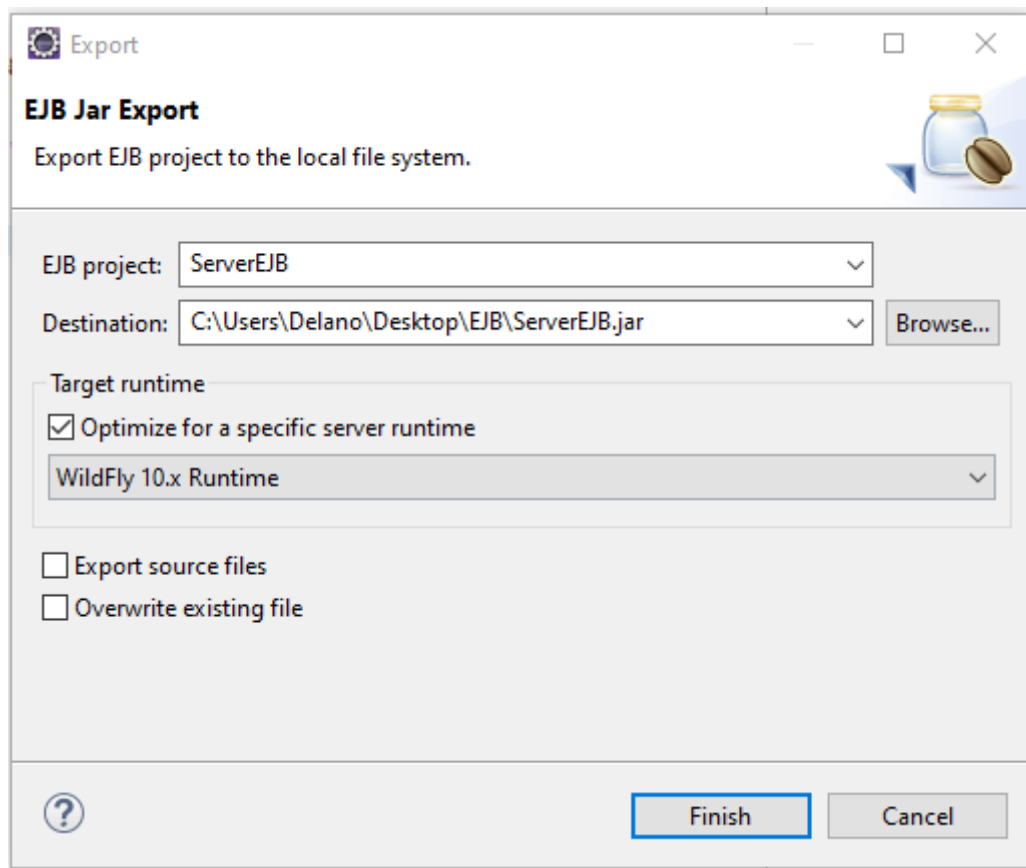
    @Override
    public String getString(String nome) {
        System.out.println("Recebido: " + nome); //mostrado no
servidor

        return "Hello " + " " + nome; // string vai para o cliente
    }

}
```

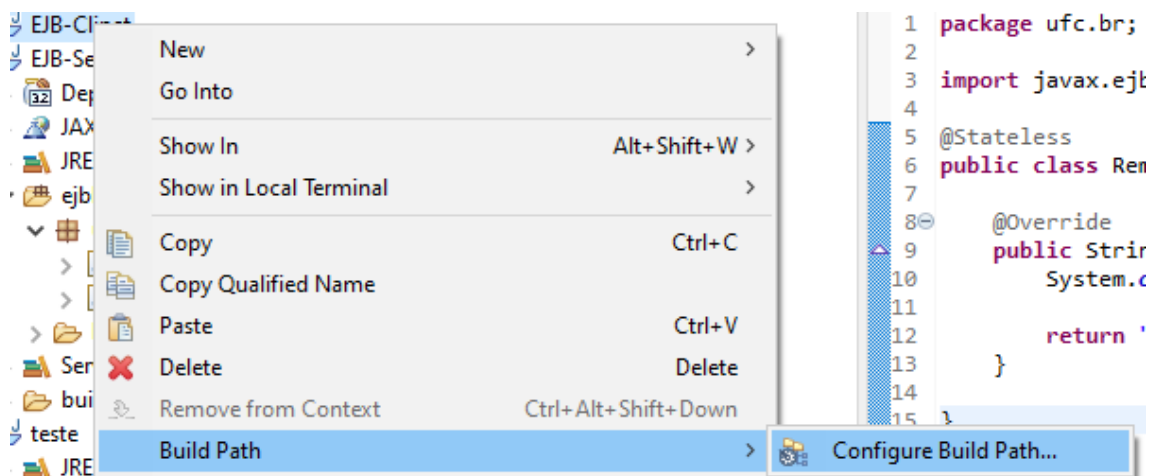
- Exportar o .jar do nosso projeto “Botão direito no projeto → Export → EJB JAR file” escolhe um diretório para salvar com um nome que termine com .jar e “Finish”.

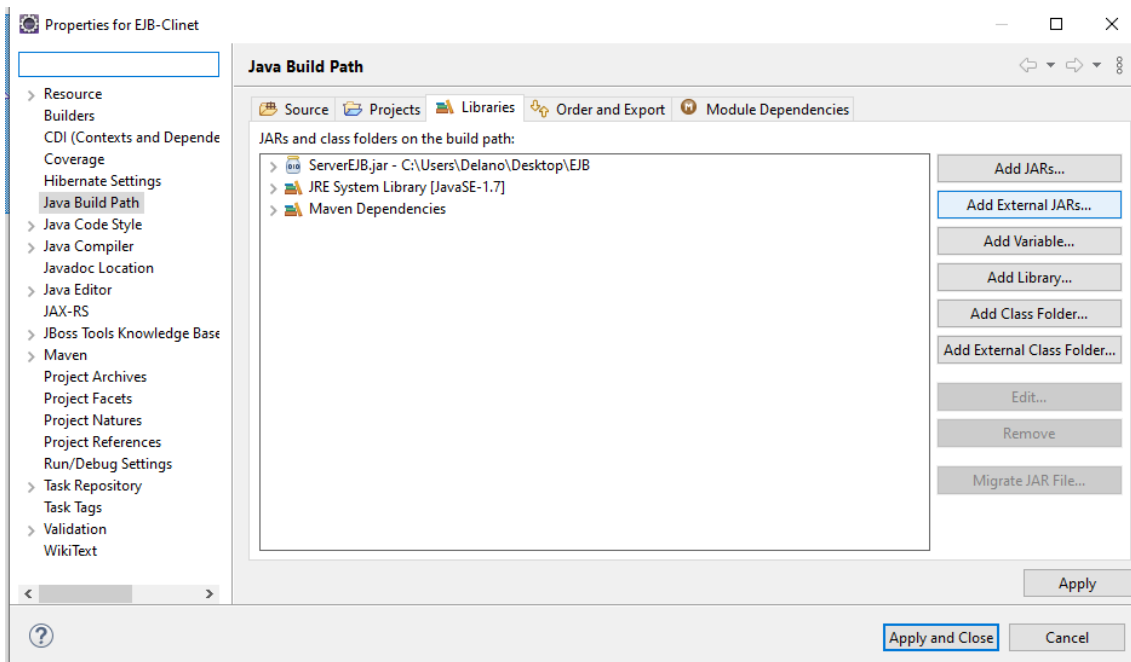




5- Como chamar método de uma classe remota

- Cria um projeto Maven, escolhe um *"maven-archetype-quickstart"* coloca um nome para ele. *"Finish"* espera um pouco e em seguida adiciona o JAR que exportamos do projeto anterior. *"Botão direito no Projeto → Build Path → Configure Build Path..."* Abre uma nova janela, na aba *"Libraries → Add External JARs..."*. Procura o jar que geramos e *"Apply and Close"*





- Modifique a Classe App.java para:

```
package ufc.br;

import static javax.naming.Context.INITIAL_CONTEXT_FACTORY;
import static javax.naming.Context.PROVIDER_URL;
import static javax.naming.Context.URL_PKG_PREFIXES;

import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import ufc.br.RemoteServer;

public class App
{
    public static void main(String[] args) {

        String host = "localhost";
        String port = "8080";

        Context remotingContext;
        try {
            remotingContext = createRemoteEjbContext(host, port);
        } catch (NamingException e) {
            System.err.println("Error setting up remoting
context");
            e.printStackTrace();
            return;
        }

        String ejbUrl =
"ejb:/EJB-Server/RemoteServerImpl!ufc.br.RemoteServer";

        RemoteServer service;
        try {
            service = createEjbProxy(remotingContext, ejbUrl,
RemoteServer.class);
        } catch (NamingException e) {
```

```

        System.out.println("Error resolving bean");
        e.printStackTrace();
        return;
    } catch (ClassCastException e) {
        System.out.println("Resolved EJB is of wrong type");
        e.printStackTrace();
        return;
    }

    // Call remote method with parameter

    String toGreet = "World";

    System.out.println("Fazendo chamada...");
    String exampleResult;
    try {
        exampleResult = service.getString(toGreet);
        System.out.println(exampleResult);
    } catch (Exception e) {
        System.out.println("Error accessing remote bean");
        e.printStackTrace();
        return;
    }

    System.out.println("Example result: " + exampleResult);
}

private static Context createRemoteEjbContext(String host, String
port) throws NamingException {

    Hashtable<Object, Object> props = new Hashtable<>();

    props.put(INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
    props.put(URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");

    props.put("jboss.naming.client.ejb.context", false);
    props.put("org.jboss.ejb.client.scoped.context", true);

    props.put("endpoint.name", "client-endpoint");

    props.put("remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABL
ED", false);
    props.put("remote.connections", "default");

    props.put("remote.connection.default.connect.options.org.xnio.Options.SASL_POL
ICY_NOANONYMOUS", false);

    props.put(PROVIDER_URL, "http-remoting://" + host + ":" +
port);
    props.put("remote.connection.default.host", host);
    props.put("remote.connection.default.port", port);

    return new InitialContext(props);
}

@SuppressWarnings("unchecked")
private static <T> T createEjbProxy(Context remotingContext, String
ejbUrl, Class<T> ejbInterfaceClass)
    throws NamingException, ClassCastException {
    Object resolvedproxy = remotingContext.lookup(ejbUrl);
    return (T) resolvedproxy;
}
}

```

- Adicione as dependências a seguir no pom.xml:

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-api</artifactId>
  <version>7.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.wildfly</groupId>
  <artifactId>wildfly-ejb-client-bom</artifactId>
  <version>10.1.0.Final</version>
  <type>pom</type>
</dependency>
```

6- Rodar o exemplo do tutorial

- **Server**

Botão direito no “*Projeto EJB → Run As → Run on Server*” escolha nosso wildfly 10 e “*Finish*” vai aparecer no log do Console a linha:

“java:app/ServerEjb/RemoteSeverImpl!ufc.br.RemoteServer”

Confira se na Classe App.java temos:

```
String ejbUrl =
"ejb:/ServerEjb/RemoteServerImpl!ufc.br.RemoteServer";
```

Se não coincidirem, faça a alteração.

- **Cliente**

Botão direito na classe “*App.java → Run As → Java Application*” assim o log mostrará nosso “Hello World”