

→ Enterprise java Beans

→ Session beans

**A – Aplicação cliente que acede a um session bean**

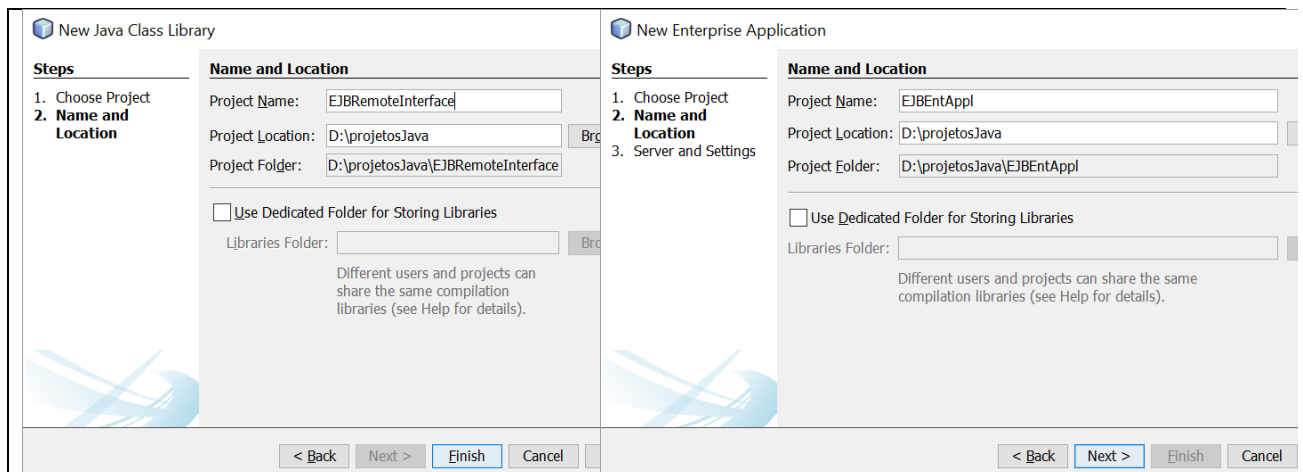
(Exemplo retirado de: <https://netbeans.org/kb/docs/javaee/entappclient.html>)

1 – Criar uma Java Class Library

Vamos construir uma biblioteca (java class library) que irá conter a interface remota para o EJB que irá ser criado. Qualquer cliente que pretenda aceder ao Bean, apenas necessita de adicionar a biblioteca ao seu projecto.

Em File, New Project, categoria Java, selecione Java Class Library. Dê ao projecto o nome EJBRemoteInterface e seleccione Finish.

Para criarmos o Bean precisamos de incluí-lo numa *Enterprise application*, para depois ser implantado (*fazer o deploy*) no servidor.



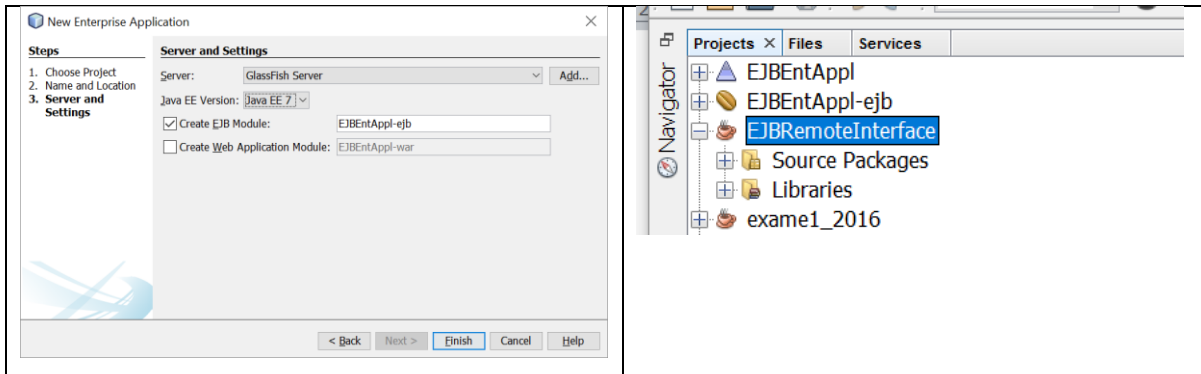
2 – Criar uma Enterprise Application que num módulo EJB irá conter um ou vários Beans.

Escolher File > New Project e na categoria Java EE seleccionar Enterprise Application. Next.

Dê ao projecto o nome EJBEntAppl e seleccione Next.

Selecione o servidor GlassFish, confirme a criação do módulo “EJB Module” e des-selecione a opção Create Web Application Module. Finish.

Observe que foi criada uma Enterprise application com um módulo EJB. (figura abaixo à direita)



### 3 – Criar um Session Bean

- Vamos criar um Bean no módulo EJB da Enterprise Application e simultaneamente gerar a interface remota do Bean na Java Class Library criada inicialmente.

No módulo da enterprise application fazer Right-click e escolher New > Session Bean.

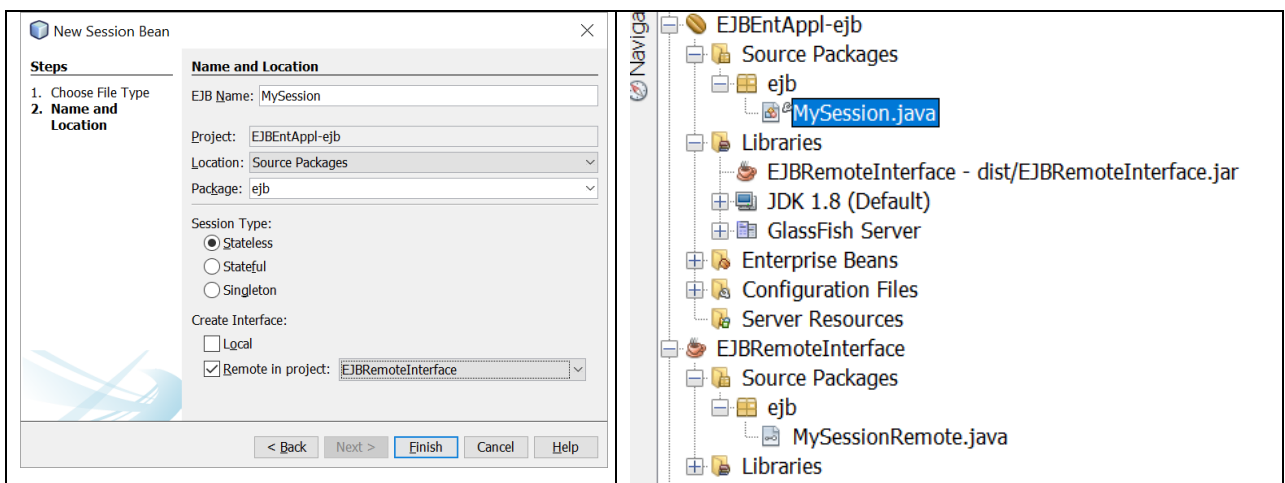
Escreva **MySession** em EJB Name.

Escreve ejb para o package.

Selecione Stateless para Session Type.

Selecione Remote para a opção Create Interface.

Select o projeto **EJBRemoteInterface** da lista de projectos. Significa que a interface remota será inserida na Java Class Library que criou no início. Finish.



Observe que foram criados os templates para o Bean (no módulo EJB) e para a interface remota na library.

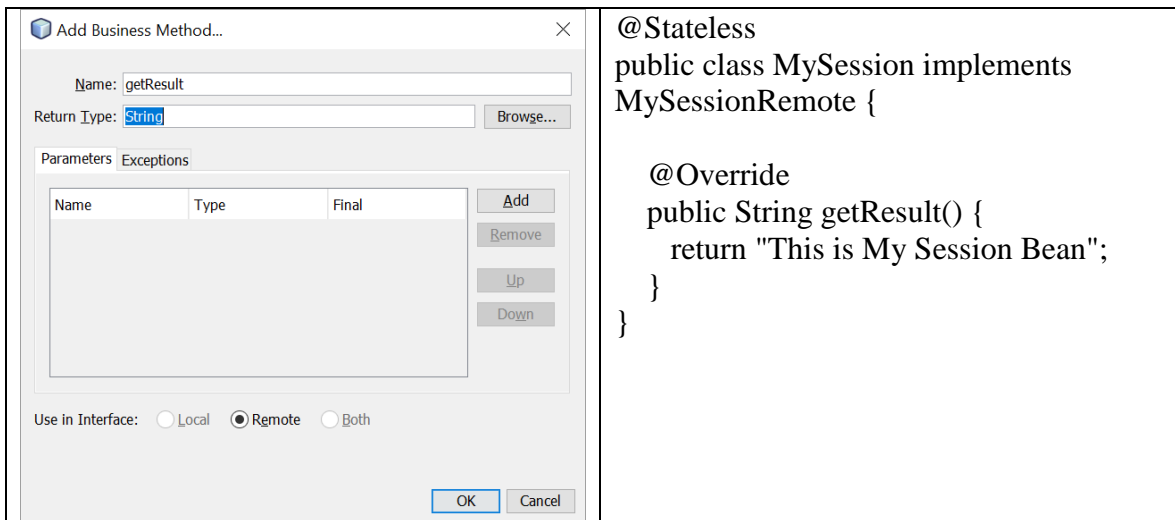
Observe também que o MySession bean implementa a interface MySessionRemote e que o JAR EJBRemoteInterface foi adicionado como uma library no módulo EJB.

#### 4 – Adicionar um Business Method

Abrindo no editor a classe MySession faça Right-click, escolha “Insert Code” and selecione “Add Business Method”.

Escreva **getResult** para o nome do método e String for the tipo do resultado. OK.

Modifique o corpo do método para que devolve uma String à sua escolha. O Bean ficará:



The image shows two parts: a dialog box on the left and a code editor on the right. The dialog box, titled "Add Business Method...", has a "Name" field containing "getResult" and a "Return Type" field containing "String". Below these are "Parameters" and "Exceptions" sections, and "Use in Interface" radio buttons for "Local", "Remote" (selected), and "Both". The code editor on the right shows the following Java code:

```
@Stateless
public class MySession implements
MySessionRemote {

    @Override
    public String getResult() {
        return "This is My Session Bean";
    }
}
```

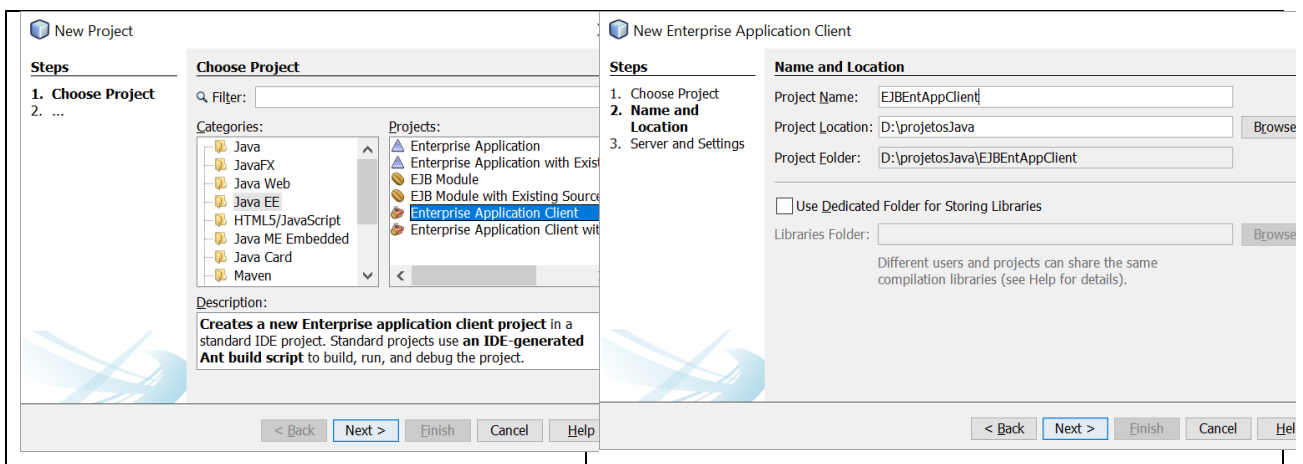
#### 5 – Implantar (Deploy) o Bean:

Faça Right-click na enterprise application EJBEntAppl e escolha Deploy.

#### 6 – Criar a aplicação cliente

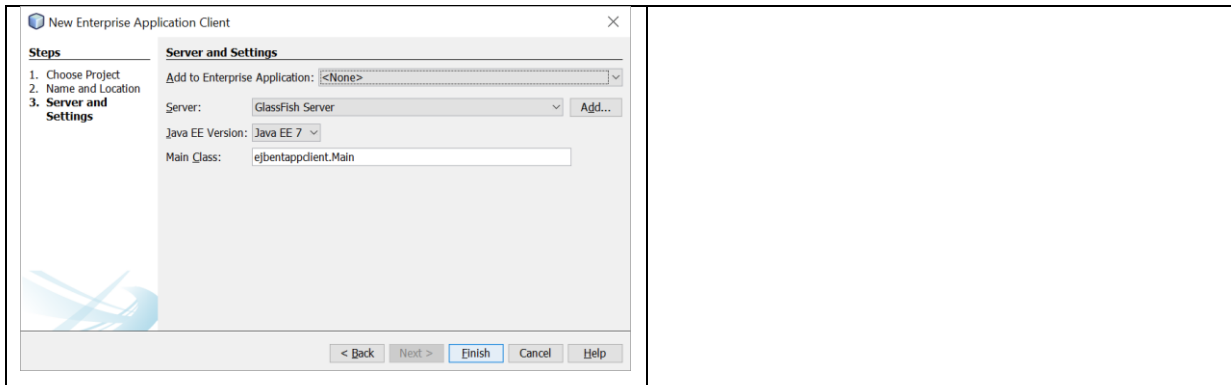
Escolher File > New Project, na categoria Java selecionar Enterprise Application Client. Next.

Para o nome do projeto escreva EJBEntAppClient. Next.



The image shows two dialog boxes side-by-side. The left dialog is titled "New Project" and shows the "Choose Project" step. The "Enterprise Application Client" option is selected in the "Projects" list. The right dialog is titled "New Enterprise Application Client" and shows the "Name and Location" step. The "Project Name" is "EJBEntAppClient", "Project Location" is "D:\projetosJava", and "Project Folder" is "D:\projetosJava\EJBEntAppClient".

Selecione o servidor GlassFish. Finish.



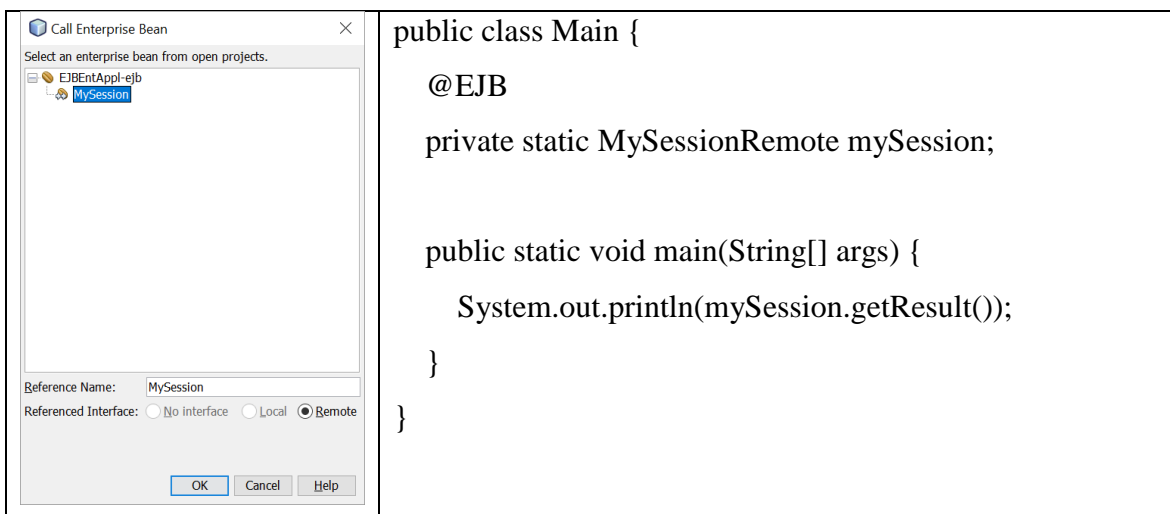
Uma classe Main é criada neste projeto. Nesta classe poderá invocar os métodos do Bean.

7 – Adicionar à aplicação cliente a Java class Library que contém a interface remota do Bean.

- Verifique que o projecto EJBRemoteInterface (java class library) está aberto.
- Na aplicação cliente EJBEntAppClient expanda o nó Source Packages e abra a classe Main.java no editor.
- No corpo do main escolha Insert Code e selecione Call Enterprise Bean.
- Expanda o nó do projeto EJBEntApp e selcione MySession. OK.

Observe que na classe Main foi injectado o Bean mySession.

Modifique o método main para invocar o método getResult(). Por exemplo:



8 – Executar a aplicação cliente.

Execute o projeto EJBEntAppClient.

9 – Adicione um outro método ao mySession bean e invoque-o na aplicação cliente.

10 – Por analogia com o exemplo anterior criar um **novo projeto** em que uma aplicação cliente é usada para aceder a um Stateful Session Bean com funcionalidades à escolha.

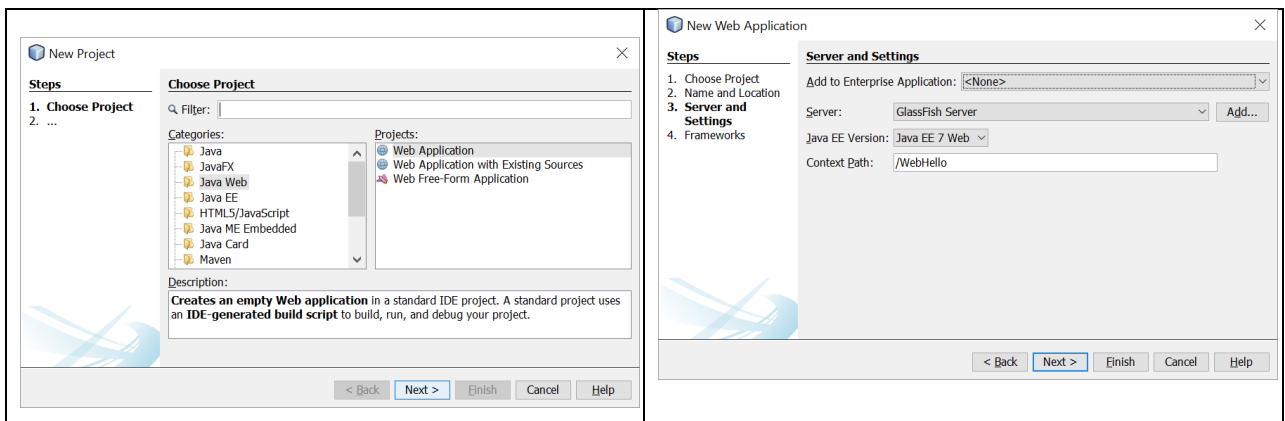
- a) No mínimo criar um método de inicialização, os getters e setters e testar o Bean.
- b) Verifique que o estado do bean permanece entre invocações diferentes feitas pelo mesmo cliente.

## B – Aplicação web acede a um session bean

(Exemplo retirado de: <https://netbeans.org/kb/docs/web/quickstart-webapps.html>)

### 1 – Criar a aplicação web

- Escolha File > New Project, Na categoria Java Web, seleccionar Web Application. Next.
- Atribua o nome WebHello.
- Escolha o servidor Glassfish a versão do JavaEE.
- Observe que foi criada uma página de boas vindas index.jsp.

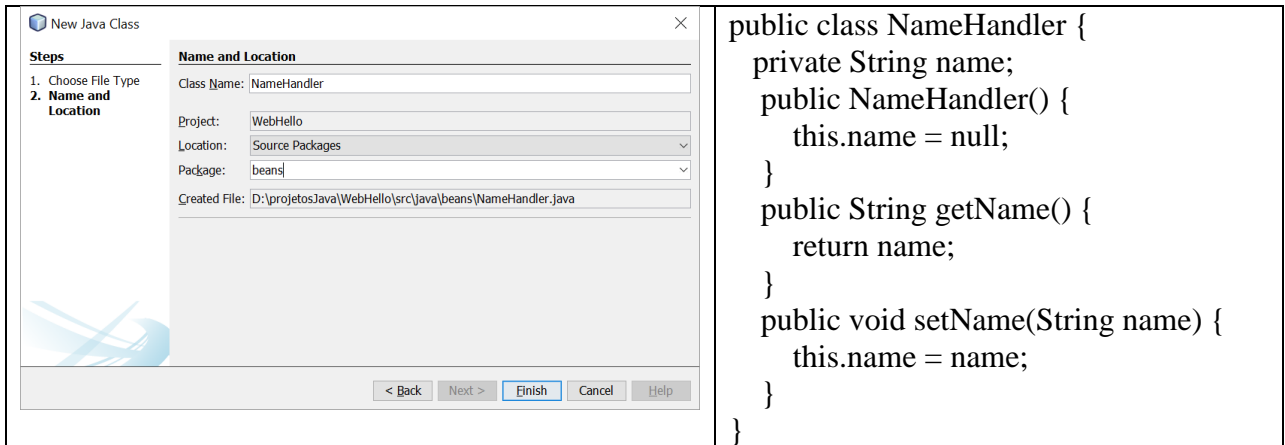


*Nota: se ao criar o projeto do tipo web application na pasta Web Pages tiver um ficheiro index.html em vez de um index.jsp, apenas terá de criar um ficheiro index.jsp e eliminar o existente. Para criar o ficheiro index.jsp, deve clicar com o botão direito na pasta do projeto e seleccionar New / JSP.*

## 2 – Criar uma classe java

No nó “Source Packages” crie uma classe NameHandler.java, num package de nome beans.

Construa a classe como na figura abaixo:



```
public class NameHandler {  
    private String name;  
    public NameHandler() {  
        this.name = null;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

## 3 – Página index.jsp

Vamos agora editar a página index.jsp, para pedir o nome ao utilizador.

- Abra a paleta do editor (em Windows, IDE Tools, Palette) e nesta depois de expandir o HTML forms, arraste o item Form para um ponto após as tags <h1>.

- Na caixa de diálogo que aparece insira os valores:

**Action:** response.jsp

**Method:** GET

**Name:** Name Input Form

- Arraste o item Text Input para o ponto antes da tag </form> e especifique os valores:

**Name:** name

**Type:** text

- Arraste o item Button para o ponto antes da tag </form> e especifique os valores:

**Label:** OK

**Type:** submit

Escreva Enter your name: antes da primeira tag <input> e troque o texto Hello World! para Entry Form ou outro texto à sua escolha.

- O texto final ficará:

```
<body>  
  <h1> Entry Form </h1>  
  <form name="name input form" action="response.jsp">  
    <input type="text" name="name" />  
    <input type="submit" value="Ok" />  
  </form>  
</body>
```

#### 4 – Criar a página response.jsp

- Crie uma página response.jsp junto à página
- Da palette, depois de expandir o jsp, arraste o item Use Bean para um ponto abaixo da tag <body>.
- Na caixa de diálogo correspondente insira os valores:

**ID:** mybean

**Class:** beans.NameHandler

**Scope:** session

- Arraste o item Set Bean Property para o ponto antes da tag <h1>. Após ok apague o atributo value para que o valor recebido da página index.jsp não seja perdido.

- Modifique o texto <h1>Hello World!</h1>, para <h1>Hello, </h1> e para o ponto após a virgula arraste o item

Get Bean Property e especifique os valores:

**Bean Name:** mybean

**Property Name:** name

- O texto final ficará:

```
<body><jsp:useBean id="mybean" scope="session" class="beans.NameHandler" />
  <jsp:setProperty name="mybean" property="name" />
  <h1>Hello, <jsp:getProperty name="mybean" property="name" /> !</h1>
</body>
```

#### 5 – Executar a aplicação.

Execute o projeto WebHello e teste a aplicação.

6 – Ao exemplo anterior pretende-se adicionar um Singleton Session Bean que conte o número de acessos à página web.

a) Construir o Session Bean usando o código abaixo

```
@Singleton
public class CounterBean {
    private int hits = 1;
    // Increment and return the number of hits
    public int getHits() {
        return hits++;
    }
}
```

b) Modificar a página response.jsp para invocar o método getHits. Para isso terá que indicar qual o bean que vai usar (item “use bean”) a aceder à propriedade *hits* (item “Get Bean Property”).

c) Testar o Bean fazendo *reload* da página e executando novamente a aplicação. Quando é que o contador volta a “1”? No item “use bean”, explore os diferentes tipos de *scope* possíveis.

## C – Timers

### 6 – Timer Programado

Considere a aplicação do exercício 1, aplicação cliente que acede a um session stateless ejb. Para adicionar à aplicação um Timer programado adicione ao stateless bean da aplicação, após o cabeçalho da classe, o seguinte código:

```
@Resource  
TimerService timerService;
```

No interior da classe (Bean) definir um método **local** anotado com @Timeout. Por exemplo,

```
@Timeout  
public void metodoTimeout(Timer timer) {  
    System.out.println ("Ocorreu um timeout" + new Date());  
}
```

- Para testar o método criar um novo método remoto do Bean, durationTimer, onde irá criar um objeto do tipo Timer. Quando esse Timer expirar será executado o método anotado com @Timeout. O método remoto durationTimer (ou outro nome à sua escolha) deve receber um parâmetro do tipo long (e.g., intervalDuration) que corresponde ao intervalo de tempo que irá decorrer até o Timer expirar.

No método durationTimer instancie o timer com a instrução:

```
Timer timer = timerService.createTimer(intervalDuration, "Created new timer");
```

Finalmente, deve modificar a aplicação cliente de forma a invocar o método que cria o timer.

- Testar o exemplo pedindo ao utilizador vários valores para a duração do timeout.

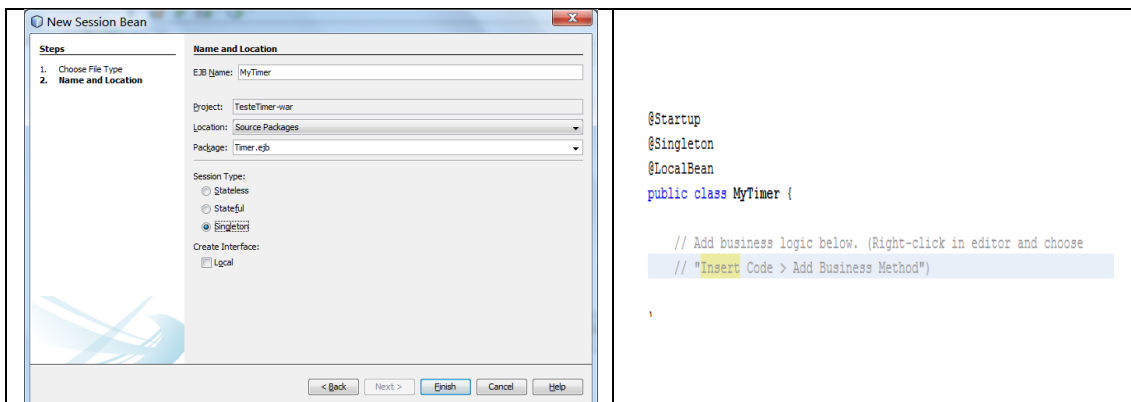
*Nota: É possível ver o output do método "metodoTimeout" na consola do servidor GlassFish.*



## 7 – Timer Automático

a) Criar uma “enterprise application” (nome = TesteTimer) com um module “web Application”. Isto é, criar um novo projecto, na categoria javaEE seleccionar EnterpriseApplication, dar o nome ao projecto, após Next, seleccionar o servidor e a versão do javaEE e seleccionar a opção “Create Web Application Module”.

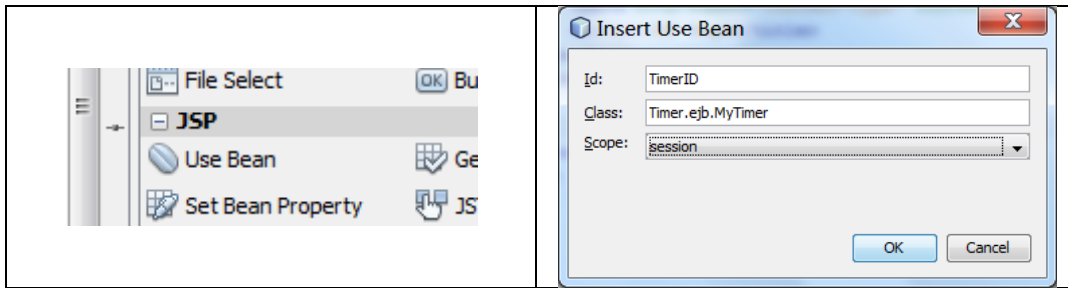
b) No módulo TesteTimer-war, Source Packages, criar um Session Singleton Bean com o nome MyTimer. Na classe do Bean, inserir a anotação @Startup e o código do método automaticTimeout() listado abaixo.



```
@Schedule(minute = "*/1", hour = ".*")
public void automaticTimeout() {
    System.out.println ("Automatic timeout occurred " + new Date());
}
```

c) Após corrigir os erros, editar o ficheiro index.jsp em TesteTimer-war, Web Pages. Se o ficheiro não existe, comece por criá-lo. Para editar o ficheiro index.jsp, no menu Window | IDE Tools selecione a Palette caso não tenha essa janela aberta.

- Na linha a seguir a “<h1>Hello World!</h1>” inserir o item UseBean da secção JSP da Palette. Dar um nome ao item, indicar o nome do Bean que construiu e escolher o scope “session”.



- d) Executar a aplicação TesteTimer e observar o que acontece no Browser e na consola do servidor GlassFish.
- e) Pode terminar o Timer: - Aceder a “Services” “Server” GlassFish Server” “Applications” em TesteTimer fazer undeploy.
- f) Testar outros calendários (Schedules).
- g) Na página index.jsp, comentar a linha “use Bean” e perceber o que acontece!