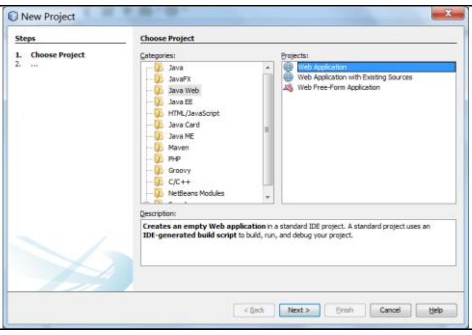
Sessão prática I – Criar JPA "entities" a partir de uma base de dados já existente

1 – Gerar JPA "entities", criar um EJB para aceder à Base de Dados e criar uma Servlet para testar o Bean.

a) Criar uma Web Application (JPAfirst) como descrito anteriormente (Lab 1 ex. 6). Executar a aplicação e observar que a página index.jsp é aberta no Browser definido por omissão. (Se no seu editor foi criada a página index.html, crie a página index.jsp e elimine a primeira.


New Project

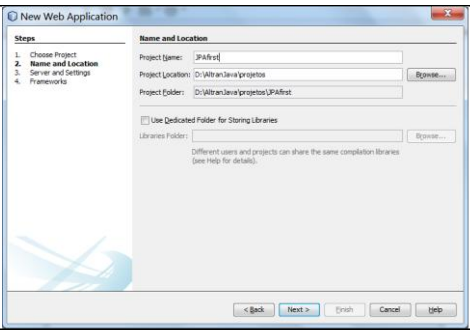
Steps
1. Choose Project
2. ...

Choose Project

Categories: Java, JavaFX, Java Web, Java EE, HTML/JavaScript, Java Card, Java ME, Maven, PHP, Groovy, C/C++, NetBeans Modules

Projects: Web Application, Web Application with Existing Sources, Web Free-Form Application

Description:
Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

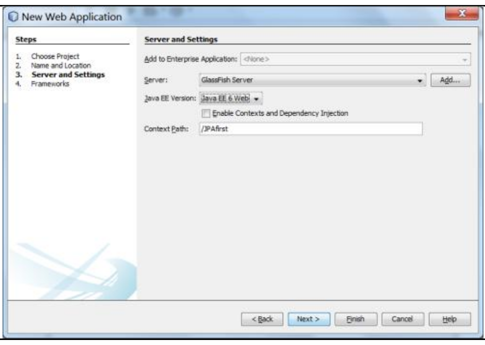

New Web Application

Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks

Name and Location

Project Name: JPAfirst
Project Location: D:\AltranJava\projetos
Project Folder: D:\AltranJava\projetos\JPAfirst

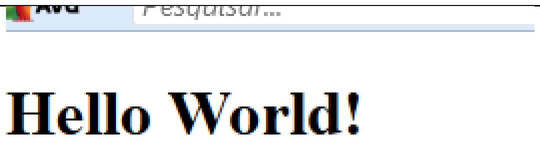
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).


New Web Application

Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks

Server and Settings

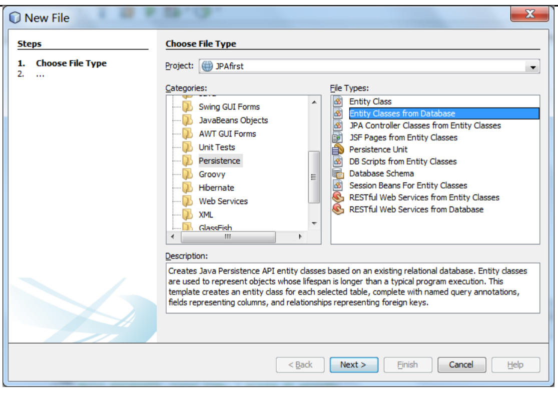
Add to Enterprise Application: <None>
Server: GlassFish Server
Java EE Version: Java EE 6 Web
☐ Enable Contexts and Dependency Injection
Context Path: /JPAfirst


Hello World!

b) Criar uma "entity" para aceder a uma base de dados já existente

- Fazer rigth-click no projeto e selecionar New / Other … / Persistence / Entity from database /Next

- Selecionar no campo Data Source o valor "jdbc/sample". As tabelas da base de dados exemplo aparecerão em "Available Tables". Selecionar a tabela CUSTOMER e fazer Add.


New File

Steps
1. Choose File Type
2. ...

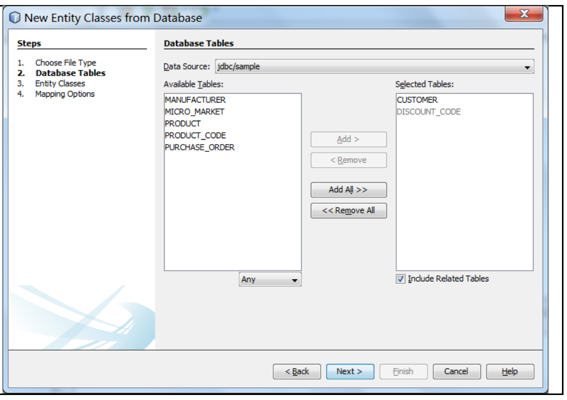
Choose File Type

Project: JPAfirst

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

File Types: Entity Class, Entity Classes from Database, JPA Controller Classes from Entity Classes, JSF Pages from Entity Classes, Persistence Unit, DB Scripts from Entity Classes, Database Schema, Session Beans For Entity Classes, RESTful Web Services from Entity Classes, RESTful Web Services from Database

Description:
Creates Java Persistence API entity classes based on an existing relational database. Entity classes are used to represent objects whose lifespan is longer than a typical program execution. This template creates an entity class for each selected table, complete with named query annotations, fields representing columns, and relationships representing foreign keys.


New Entity Classes from Database

Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options

Database Tables

Data Source: jdbc/sample

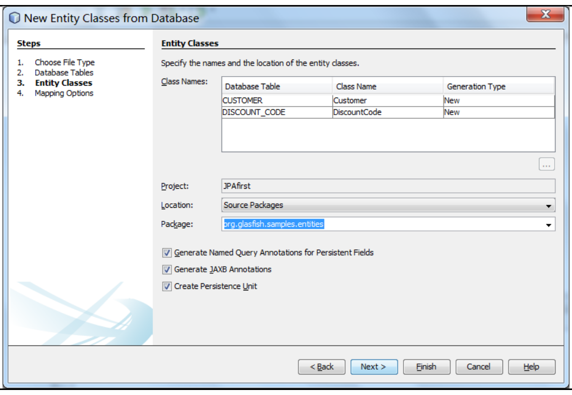
Available Tables: MANUFACTURER, MICRO_MARKET, PRODUCT, PRODUCT_CODE, PURCHASE_ORDER

Selected Tables: CUSTOMER, DISCOUNT_CODE

Any

☑ Include Related Tables

- Após Next introduzir "org.glassfish.samples.entities" para o nome do package. Na coluna Class names aparecem as classes mapeadas da base de dados. Após Next, observar as opções de mapeamento usadas por omissão.



Entity Classes

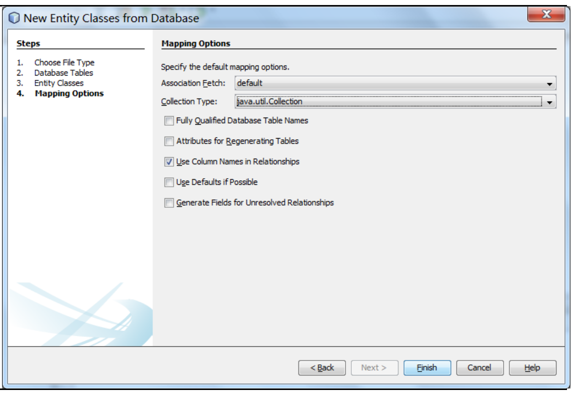
Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
CUSTOMER	Customer	New
DISCOUNT_CODE	DiscountCode	New

Project: JPAfirst
Location: Source Packages
Package: org.glasfish.samples.entities

[✓] Generate Named Query Annotations for Persistent Fields
[✓] Generate JAXB Annotations
[✓] Create Persistence Unit



Mapping Options

Specify the default mapping options.

Association Fetch: default
Collection Type: java.util.Collection

[ ] Fully Qualified Database Table Names
[ ] Attributes for Regenerating Tables
[✓] Use Column Names in Relationships
[ ] Use Defaults if Possible
[ ] Generate Fields for Unresolved Relationships

- Observar as classes que foram criadas em "Source Packages" "org.glassfisf.samples.entities". Para cada tabela mapeada foi criada uma classe.

As classes geradas usam JPQL (Java Persistence Query Language) para definir queries à base de Dados. Para cada tabela, foram gerados queries para consultar a tabela por cada um dos seus campos e um query para consultar todas as linhas da tabela. Podem ser adicionados novos queries.

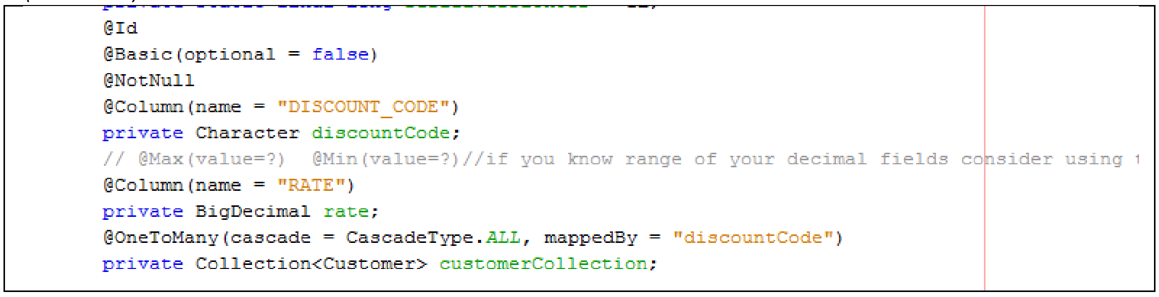
São criadas regras de validação "BeanValidation Constraints" para cada campo, baseadas na definição do esquema relacional. Essas restrições serão usadas sempre que uma instância da entity é gravada, atualizada ou removida da base de dados. Restrições baseadas em expressões regulares podem ser usadas para validar os valores. Ver por exemplo o campo phone em Customer.java.

Observando os campos destas duas entidades, DISCOUNT_CODE e CUSTOMER, pode ver-se que existe uma associação (um para vários) em que um desconto está associado a vários clientes e um cliente está associado a um desconto. Esta associação está representada com as anotações @ManyToOne (em Customer) e @OneToMany (em DiscountCode). O campo discountCode permite fazer a junção das tabelas.

Em Customer.java

```java
    @JoinColumn(name = "DISCOUNT_CODE", referencedColumnName = "DISCOUNT_CODE")
    @ManyToOne(optional = false)
    private DiscountCode discountCode;
```

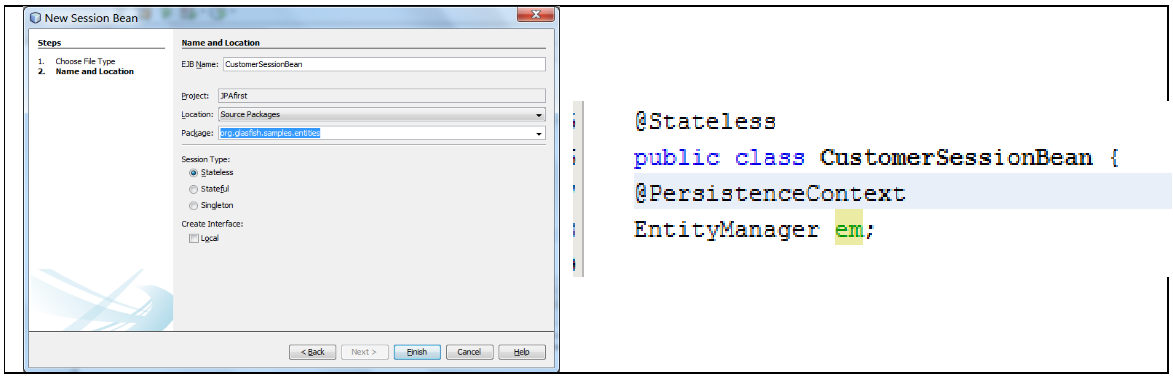
Em DiscountCode.java



```
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "DISCOUNT_CODE")
    private Character discountCode;
    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consider using t
    @Column(name = "RATE")
    private BigDecimal rate;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "discountCode")
    private Collection<Customer> customerCollection;
```

c) Criar um Bean para fazer queries à base de dados.

- Fazer Right-click em org.glassfish.samples package e criar um Stateless EJB com o nome CustomerSessionBean. Na classe criada, injetar uma instância de EntityManager como ilustrado na figura abaixo. Fazer “Fix Imports”.



New Session Bean

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

Name and Location

EJB Name: CustomerSessionBean
Project: JPAfirst
Location: Source Packages
Package: org.glasfish.samples.entities

Session Type:
[✓ Stateless]
Stateful
Singleton

Create Interface:
Local

```
@Stateless
public class CustomerSessionBean {
@PersistenceContext
EntityManager em;
```

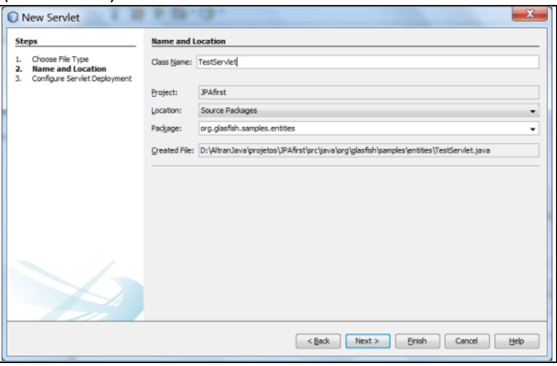
Adicionar à classe CustomerSessionBean o seguinte método:

```
public List<Customer> getCustomer () {
   return (List<Customer>) em.createNamedQuery("Customer.findAll").getResultList();
  }
```

(Atenção: o tipo List pertence ao package java.util)

Este método está a usar o query gerado anteriormente para obter todos os clientes (customers) da base de dados e devolvê-los num objeto do tipo lista de objetos do tipo Customer.

d) Criar uma servlet (de nome TestServlet) para testar o Bean. Observar na Servlet criada a anotação @WebServlet. Para invocar métodos do Bean criado, injetar a classe CustomerSessionBean na Servlet.



New Servlet

Steps

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

Name and Location

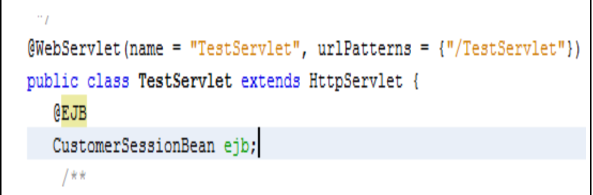
Class Name: TestServlet

Project: JPAfirst

Location: Source Packages

Package: org.glasfish.samples.entities

Created File: D:\AltranJava\projetos\JPAfirst\src\java\org\glasfish\samples\entities\TestServlet.java

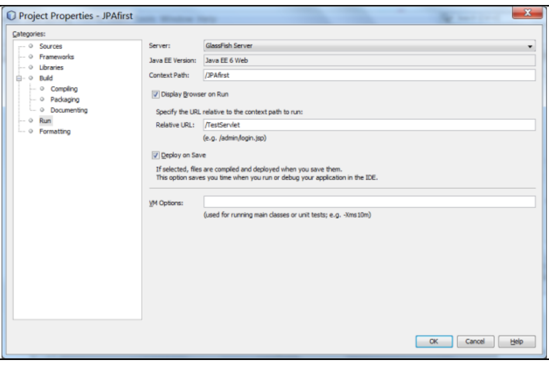


```
 */
@WebServlet(name = "TestServlet", urlPatterns = {"/TestServlet"})
public class TestServlet extends HttpServlet {
   @EJB
   CustomerSessionBean ejb;
    /**
```

No Body da Servlet introduzir a chamada do método getCustomer (). Instrução a bold na listagem abaixo.

```
out.println("<h1>Servlet TestServlet at " + request.getContextPath() + "</h1>");
out.println (ejb.getCustomer());
out.println("</body>");
```

Testar a servlet com rigth-click em TestServlet.java e selecionar Run File. Ou configurar o projeto para iniciar na servlet: Selecionar projeto, em Proprieties / run / relative URL inserir /TestServlet. Ao executar o projeto o Browser mostra a seguinte lista de identificadores:



Project Properties - JPAfirst

Categories:
- Sources
- Frameworks
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Formatting

Server: GlassFish Server

Java EE Version: Java EE 6 Web

Context Path: /JPAfirst

[✓] Display Browser on Run

Specify the URL relative to the context path to run:

Relative URL: /TestServlet

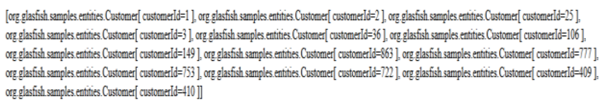
(e.g. /admin/login.jsp)

[✓] Deploy on Save

If selected, files are compiled and deployed when you save them.
This option saves you time when you run or debug your application in the IDE.

VM Options:

(used for running main classes or unit tests; e.g. -Xms10m)



```
[org.glasfish.samples.entities.Customer[ customerId=1 ], org.glasfish.samples.entities.Customer[ customerId=2 ], org.glasfish.samples.entities.Customer[ customerId=25 ],
org.glasfish.samples.entities.Customer[ customerId=3 ], org.glasfish.samples.entities.Customer[ customerId=36 ], org.glasfish.samples.entities.Customer[ customerId=106 ],
org.glasfish.samples.entities.Customer[ customerId=149 ], org.glasfish.samples.entities.Customer[ customerId=863 ], org.glasfish.samples.entities.Customer[ customerId=777 ],
org.glasfish.samples.entities.Customer[ customerId=753 ], org.glasfish.samples.entities.Customer[ customerId=722 ], org.glasfish.samples.entities.Customer[ customerId=409 ],
org.glasfish.samples.entities.Customer[ customerId=410 ]]
```

2 - Modificar a aplicação para que mostre outros atributos de cada Customer (porque só aparece o ID?).

3 – Modificar a aplicação para executar outros queries.

4 – Mostrar numa servlet os valores da tabela Product.

5 – Explorar como aceder a uma base de dados MySQL.

6 – Crie uma base de dados MySQL exemplo, e repita o processo anterior para essa base de dados.