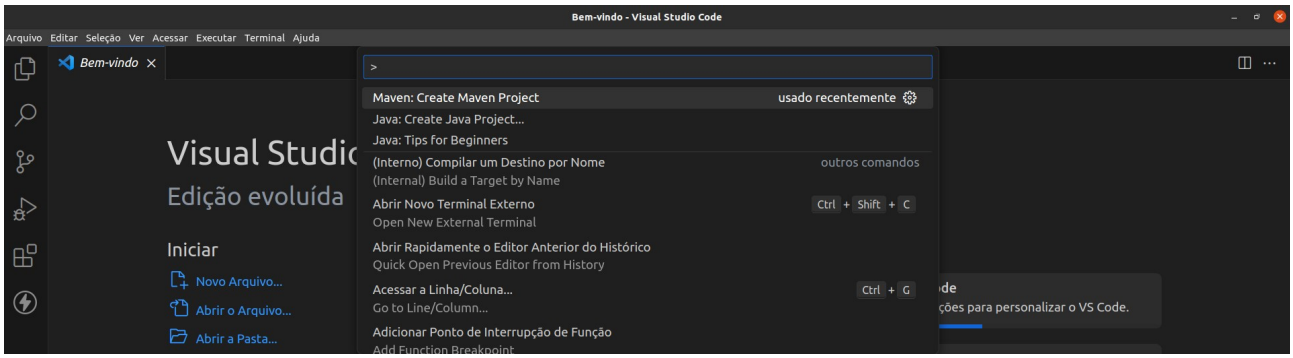
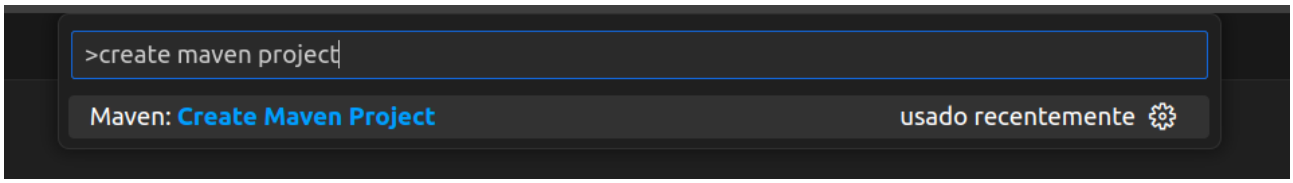
Configuração do JPA/Hibernate no VSCode.

Utilizaremos o gerenciador de projetos Maven, que simplifica e gerencia a construção de projetos Java. Maiores informações do Maven: https://maven.apache.org/

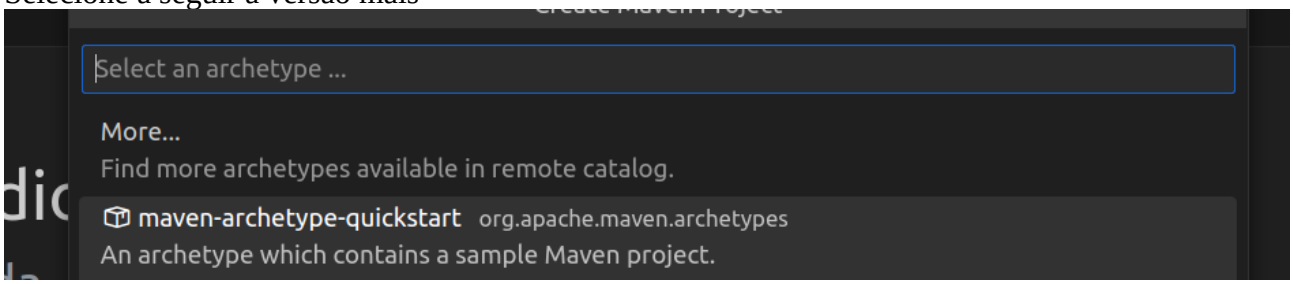
Para criar um projeto Maven, utilize a paleta de comandos (View -> command palette) e digite Create Maven Project:



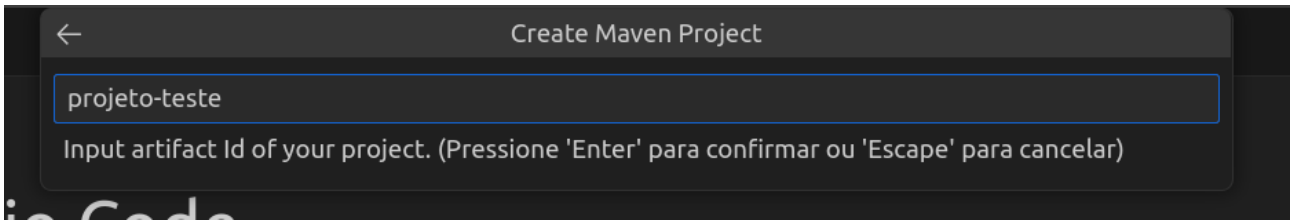


Será solicitado o arquétipo do projeto. Selecione maven-archetype-quickstart:
(arquétipos são tipos específicos de projetos que definem uma estrutura básica, configurações iniciais e dependências mais comuns)

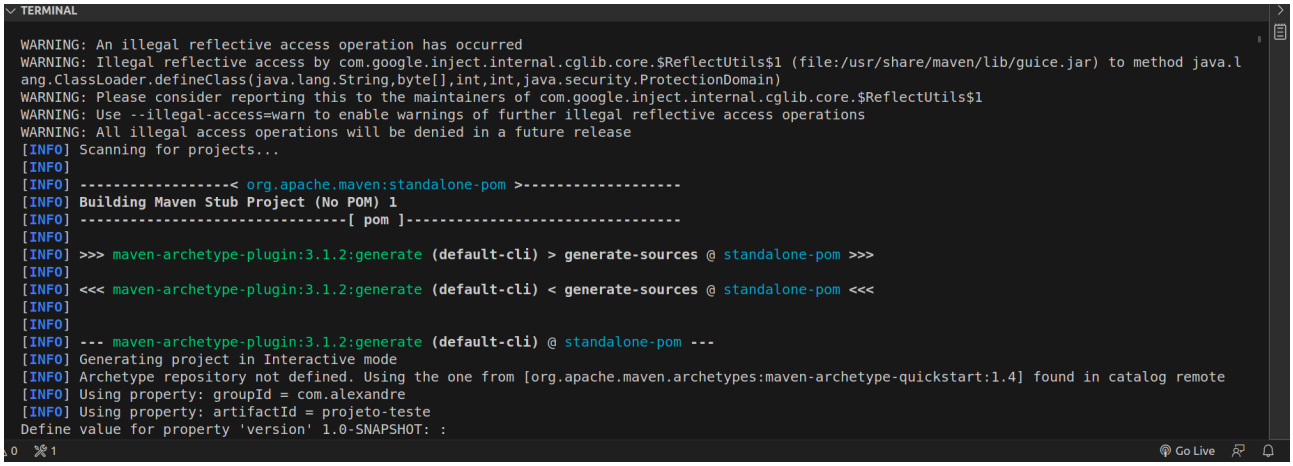
Selecione a seguir a versão mais



recente (1.4 provavelmente);
Em seguida digite o group-id: com.alexandre no meu caso.
E preencha o artifact-id:

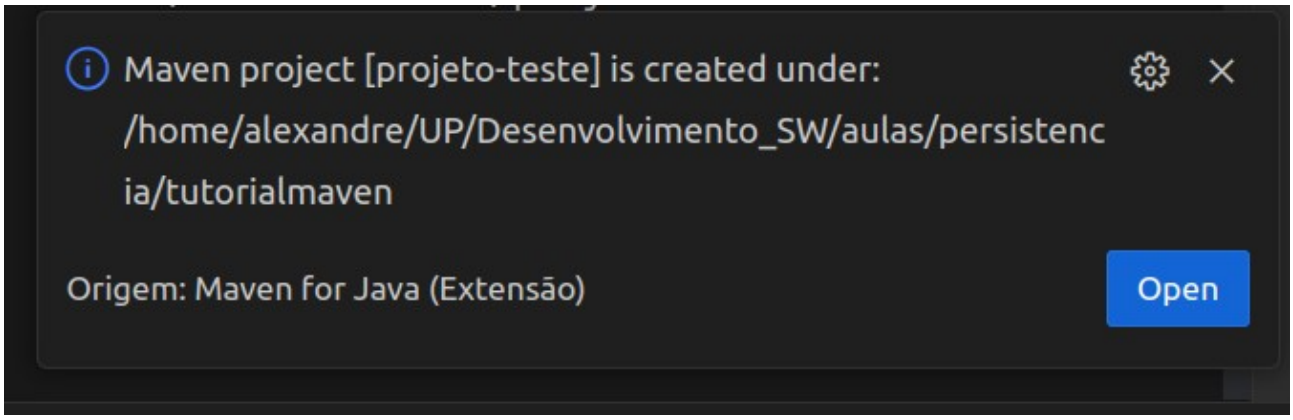


Em seguida, a criação do projeto será iniciada. Aperte enter para manter a versão padrão de desenvolvimento 1.0-SNAPSHOT. (convenção para indicar apenas que é uma versão em desenvolvimento):



Na sequência aperte enter para confirmar a criação.

Poderá ser aberta uma opção de abrir o projeto. Clique em open:



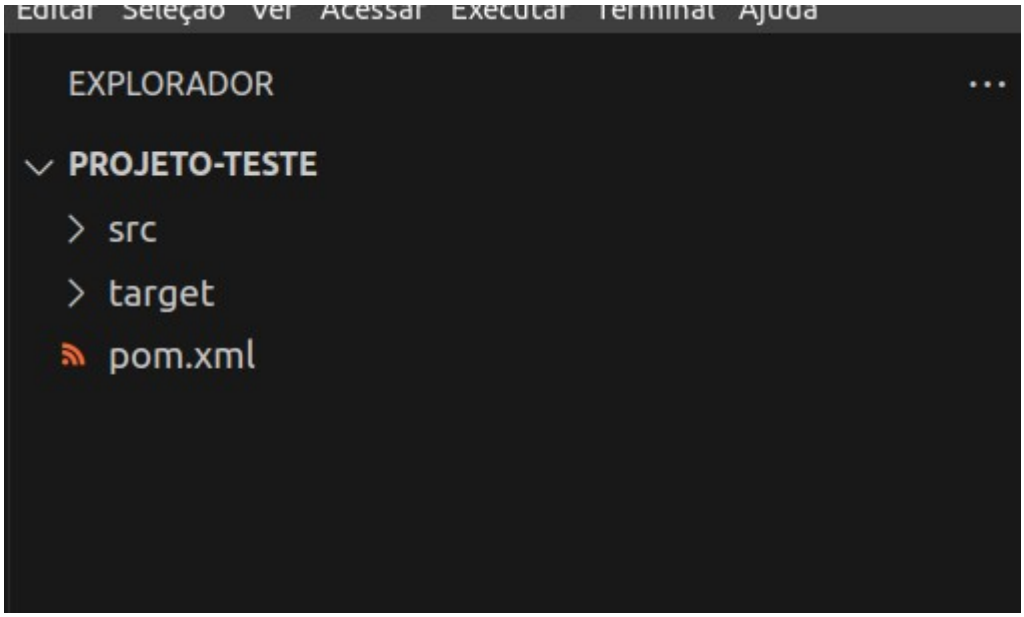
Deverá ser aberto o projeto novo com a seguinte estrutura de arquivos e pastas:
src:
-src é o diretório raiz onde você deve colocar todos os arquivos-fonte do seu projeto.
-Dentro do diretório src, há subdiretórios como src/main e src/test.
-src/main é o diretório que contém o código-fonte principal do projeto.
-src/test é o diretório que contém os arquivos de teste relacionados ao projeto.

target:
O diretório target é criado pelo Maven e é onde os resultados da compilação e construção do projeto são armazenados.

Após a execução de um comando de construção, como mvn compile ou mvn package, o Maven coloca os arquivos gerados, como classes compiladas, arquivos JAR, WAR ou outros artefatos, no diretório target.
É no diretório target que você encontrará o resultado final do processo de construção do seu projeto.

pom.xml

O arquivo pom.xml (Project Object Model) é o arquivo de configuração principal do Maven.
Ele está localizado no diretório raiz do projeto e contém todas as informações necessárias para o Maven construir, testar e gerenciar seu projeto.
No arquivo pom.xml, você define as dependências do projeto, plugins, configurações de compilação, teste e construção, além de outras informações relevantes.
O Maven usa o arquivo pom.xml para realizar as tarefas de construção e gerenciamento do projeto.



No arquivo pom.xml devem ser inseridas as dependências do hibernate, jpa e mysql que utilizaremos. O arquivo pom.xml deve estar assim (ou com pequenas alterações a depender da versão desejada de cada dependência):

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.teste</groupId>
<artifactId>demo</artifactId>
<version>1.0-SNAPSHOT</version>
<name>demo</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>
```

```
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
</properties>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.11</version>
<scope>test</scope>
</dependency>
<!-- Dependências do Hibernate e JPA -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.4.32.Final</version>
</dependency>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-entitymanager</artifactId>
<version>5.4.32.Final</version>
</dependency>
<dependency>
<groupId>javax.persistence</groupId>
<artifactId>javax.persistence-api</artifactId>
<version>2.2</version>
</dependency>
<!-- Dependência do driver JDBC para o MySQL -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.27</version>
</dependency>
</dependencies>
<build>
<pluginManagement>
<!--
 lock down plugins versions to avoid using Maven defaults (may be moved to parent pom)
-->
<plugins>
<!--
 clean lifecycle, see
https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle
-->
<plugin>
<artifactId>maven-clean-plugin</artifactId>
<version>3.1.0</version>
</plugin>
<!--
 default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_jar_packaging
```

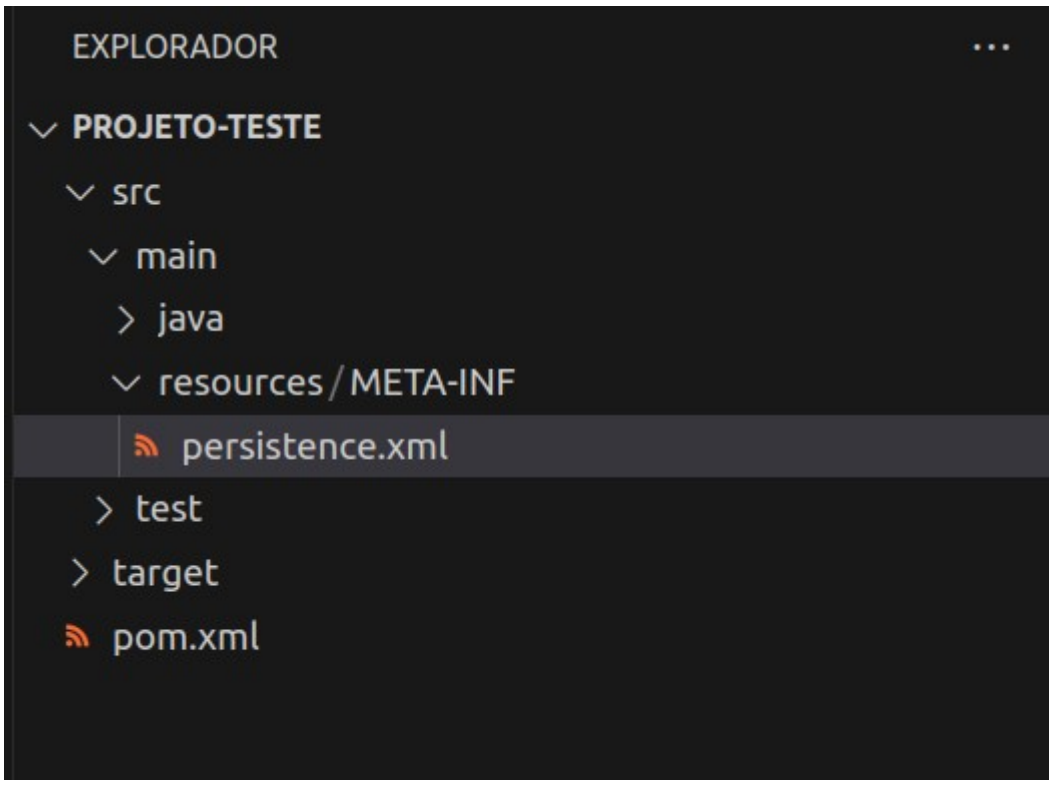
```
-->
<plugin>
<artifactId>maven-resources-plugin</artifactId>
<version>3.0.2</version>
</plugin>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
</plugin>
<plugin>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.22.1</version>
</plugin>
<plugin>
<artifactId>maven-jar-plugin</artifactId>
<version>3.0.2</version>
</plugin>
<plugin>
<artifactId>maven-install-plugin</artifactId>
<version>2.5.2</version>
</plugin>
<plugin>
<artifactId>maven-deploy-plugin</artifactId>
<version>2.8.2</version>
</plugin>
<!--
 site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle
-->
<plugin>
<artifactId>maven-site-plugin</artifactId>
<version>3.7.1</version>
</plugin>
<plugin>
<artifactId>maven-project-info-reports-plugin</artifactId>
<version>3.0.0</version>
</plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

Desta maneira, o Maven irá configurar as dependências conforme descritas no arquivo.

O próximo passo é configurar a comunicação do JPA/Hibernate com o banco de dados.
Para isso, primeiro crie uma database no MySQL com o nome testehibernate, por exemplo.

Para configurar é preciso criar o arquivo persistence.xml que deve estar dentro de um diretório nomeado META-INF. (aqui é preciso cuidado para digitar corretamente).

Para organizar a estrutura, podemos criar os diretórios *resource/META-INF* dentro do diretório *main* e criar o arquivo persistence.xml*:*



Como no VSCode não há uma maneira de gerar automaticamente o arquivo persistence.xml (a não ser com o uso de extensões), segue um modelo do arquivo:
***Importante:** as linhas em cinza devem ser alteradas de acordo com a configuração do seu banco de dados.
Observe também a linha grifada em amarelo. Aqui, a cada nova classe que se deseja persistir no banco de dados, deve ser inserida uma nova linha correspondente. Se for inserida a classe Usuario, por exemplo, deve-se inserir uma nova linha **<class>**com.alexandre.Usuario**</class>**

```
<persistence xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" version="2.0">
<persistence-unit name="my-persistence-unit" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>com.alexandre.Pessoa</class>
<properties>
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/testehibernate"/>
<property name="javax.persistence.jdbc.user" value="root"/>
<property name="javax.persistence.jdbc.password" value="positivo"/>
<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
<!-- Configurações do Hibernate -->
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
<property name="hibernate.hbm2ddl.auto" value="update"/>
<!-- Opções: create, create-drop, update -->
<property name="hibernate.show_sql" value="true"/>
<!-- Mostrar SQL gerado pelo Hibernate -->
</properties>
</persistence-unit>
</persistence>
```

Com a configuração completa, passamos à criação da classe Pessoa:

```java
package com.alexandre;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int idade;

    private String nome;

    public Pessoa(int idade, String nome) {
        this.idade = idade;
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }


}
```

Explicação das anotações JPA utilizadas:

Linha 8-@**Entity**: Essa anotação é usada para indicar que a classe Pessoa é uma entidade que será mapeada para uma tabela no banco de dados. É uma anotação obrigatória para todas as entidades JPA.

Linhas 11 e 12:
@**Id**: Essa anotação é usada para indicar que o campo id é a chave primária da entidade Pessoa. A anotação @**GeneratedValue** é usada em conjunto com @Id para especificar a estratégia de geração automática de valores para a chave primária. No nosso caso, a estratégia utilizada é **GenerationType**.**IDENTITY**, que indica que o valor do ID será gerado automaticamente pelo banco de dados.

Descrição de algumas estratégicas de geração de IDs:

- GenerationType.AUTO: Essa estratégia delega a geração do valor da chave primária para o hibernate. O hibernate escolherá a estratégia apropriada com base no banco de dados subjacente.
- GenerationType.SEQUENCE: Essa estratégia usa uma sequência de banco de dados para gerar os valores da chave primária. O hibernate obtém valores da sequência e os associa aos objetos persistidos.
- GenerationType.TABLE: Essa estratégia usa uma tabela no banco de dados para gerar os valores da chave primária. O hibernate cria uma tabela dedicada para registrar os valores gerados e os associa aos objetos persistidos.
- GenerationType.IDENTITY: Essa estratégia usa um recurso específico do banco de dados para gerar os valores da chave primária automaticamente. Geralmente, isso é implementado usando colunas autoincrementáveis ou recursos semelhantes fornecidos pelo banco de dados.

Desta forma a classe Pessoa está preparada para ser persistida no banco de dados.

Para operacionalizar este processo iremos utilizar a classe DAO correspondente, PessoaDAO, com implementação sugerida a seguir, conforme vimos em sala:

```
1  //ajuste o pacote correto
2
3
4  import javax.persistence.EntityManager;
5  import javax.persistence.EntityManagerFactory;
6  import javax.persistence.Persistence;
7  import javax.persistence.Query;
8
9  import java.util.List;
10
11 public class PessoaDAO {
12     private EntityManagerFactory emf;
13
14     public PessoaDAO() {
```

```
15          emf = Persistence.createEntityManagerFactory("my-persistence-unit");
16      }
17
18      public void salvarPessoa(Pessoa pessoa) {
19          EntityManager em = emf.createEntityManager();
20          em.getTransaction().begin();
21          em.persist(pessoa);
22          em.getTransaction().commit();
23          em.close();
24      }
25
26      public Pessoa buscarPessoaPorId(Long id) {
27          EntityManager em = emf.createEntityManager();
28          Pessoa pessoa = em.find(Pessoa.class, id);
29          em.close();
30          return pessoa;
31      }
32
33      public List<Pessoa> buscarTodasPessoas() {
34          EntityManager em = emf.createEntityManager();
35          Query query = em.createQuery("SELECT p FROM Pessoa p");
36          List<Pessoa> pessoas = query.getResultList();
37          em.close();
38          return pessoas;
39      }
40
41      public void atualizarPessoa(Pessoa pessoa) {
42          EntityManager em = emf.createEntityManager();
43          em.getTransaction().begin();
44          em.merge(pessoa);
45          em.getTransaction().commit();
46          em.close();
47      }
48
49      public void excluirPessoa(Pessoa pessoa) {
50          EntityManager em = emf.createEntityManager();
51          em.getTransaction().begin();
52          pessoa = em.merge(pessoa);
53          em.remove(pessoa);
54          em.getTransaction().commit();
55          em.close();
56      }
57  }
```

Com isso, utilizaremos um objeto DAO para gerenciar os acessos aos dados. Para testar, crie uma instância de PessoaDAO, uma instância de Pessoa e salve no banco de dados, conforme código a seguir. Na sequência, verifique se o objeto foi salvo na sua base de dados. Deve ter sido criada automaticamente a tabela *pessoa:*

```
1
2  package com.alexandre;
3
4  public class App
5  {
6      public static void main( String[] args )
7      {
8          PessoaDAO pessoaDAO = new PessoaDAO();
9
10         Pessoa p1 = new Pessoa(20, "alex");
11
12
13         pessoaDAO.salvarPessoa(p1);
14     }
15 }
```