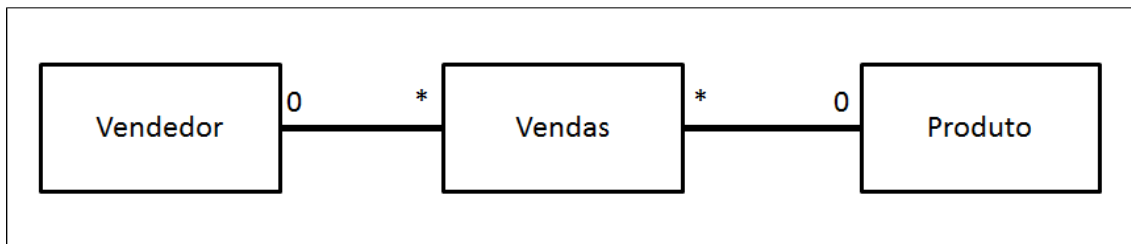


# Usando a API de Persistência Java

---

Este roteiro demonstra a implementação de um projeto simples, para apresentar o uso da API de Persistência Java, que será usada a partir de um aplicativo Java SE, iniciado pela linha de comando. Este projeto se baseia no uso da JPA para gerenciar todas as transações com a base de dados, usando o seguinte relacionamento entre um vendedor e as vendas que ele fez num determinado período de tempo.



*Figura 1 – Modelo da base de dados da aplicação*

O modelo que demonstra essa relação é apresentada na Figura 1, e de acordo com ele, um vendedor pode ter de nenhuma a várias vendas atribuídas a ele. De modo similar, um produto pode estar presente em nenhuma ou várias vendas, e as Listagens de 1 a 3 apresentam os comandos SQL que criam essas tabelas na base de dados.

```
CREATE TABLE vendedor
(
    id INT NOT NULL PRIMARY KEY GENERATED ALWAYS
        AS IDENTITY (START WITH 1, INCREMENT BY 1),
    nomecharacter(20)
);
```

*Listagem 1 – Script de criação da tabela Vendedor*

```
CREATE TABLE produto
(
    id INT NOT NULL PRIMARY KEY GENERATED ALWAYS
        AS IDENTITY (START WITH 1, INCREMENT BY 1),
    nomecharacter(20)
);
```

*Listagem2 – Script de criação da tabela Produto*

```

CREATE TABLE venda
(
    id INT NOT NULL PRIMARY KEY GENERATED ALWAYS
        AS IDENTITY (START WITH 1, INCREMENT BY 1),
    vendedor INT,
    produto INT,
    valorvenda numeric(10,2),

    CONSTRAINT vendedor_fkey FOREIGN KEY (vendedor) REFERENCES
vendedor (id),

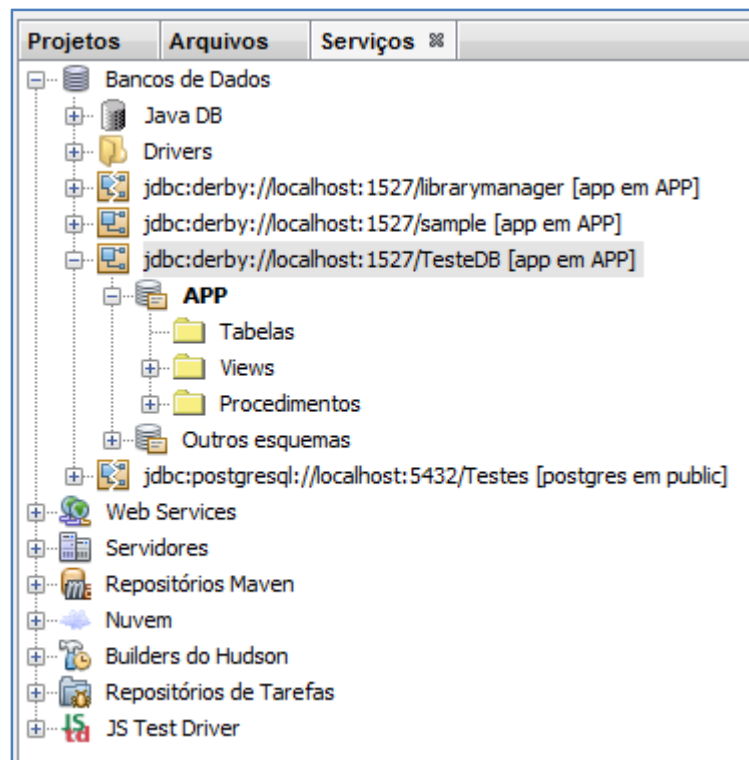
    CONSTRAINT produto_fkey FOREIGN KEY (produto) REFERENCES
produto (id)

    ON UPDATE NO ACTION ON DELETE NO ACTION
);

```

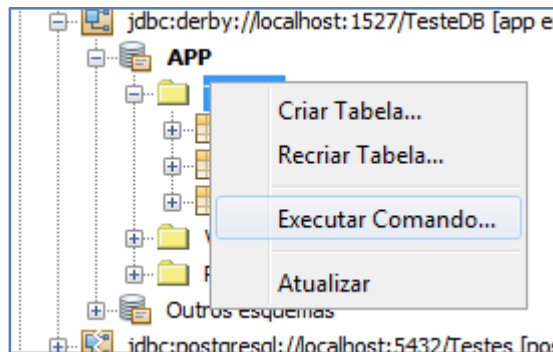
*Listagem3 – Script de criação da tabela Vendas*

Para usar um banco de dados, use o utilitário Databases do NetBeans, que pode ser encontrado na aba Serviços, como apresentado na Figura 2.



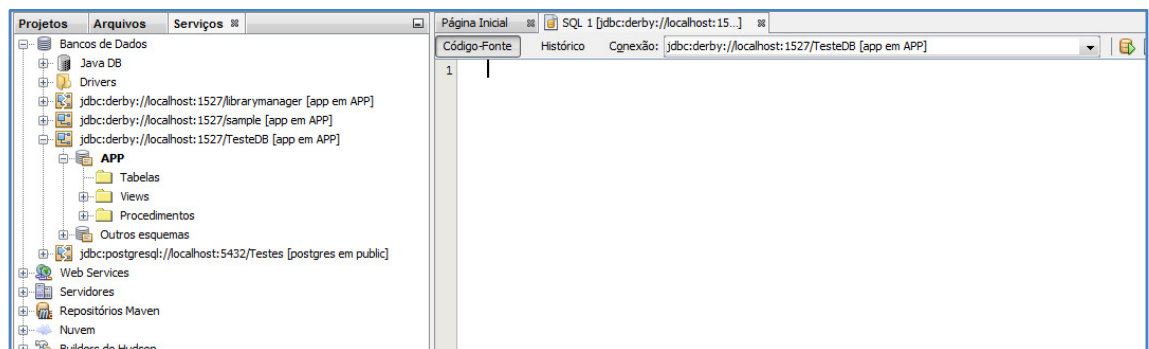
*Figura 2 – Localização de Databases na aba Serviços*

Neste momento, é possível usar um banco de dados já existente, ou criar um novo. No caso deste roteiro, usarei o banco de dados TesteDB. Como pode ser observado na Figura 2, não há tabelas nesse banco de dados, por isso, vou criá-las, usando os scripts fornecidos nas listagens de 1 a 3, nessa mesma ordem, por causa das chaves estrangeiras, que causam dependência na tabela Venda.



*Figura 3 – Seleção de execução de comandos SQL*

Para executar os scripts, clique com o botão direito do mouse em Tabelas, para abrir o menu apresentado na Figura 3. Nesse menu, selecione a opção Executar Comando, que abrirá uma tela semelhante à apresentada na Figura 4.



*Figura 4 – Área de execução de comandos SQL*

Nessa área que abriu, escreva (ou copie) o script SQL para criar a tabela que preciso, e clique no ícone “Executar SQL”. Ao término do processo, a tela deve ficar semelhante ao que é apresentado nas Figuras 5 e 6, e esses passos devem ser repetidos para cada um dos scripts SQL, até chegar ao resultado mostrado na Figura 7.

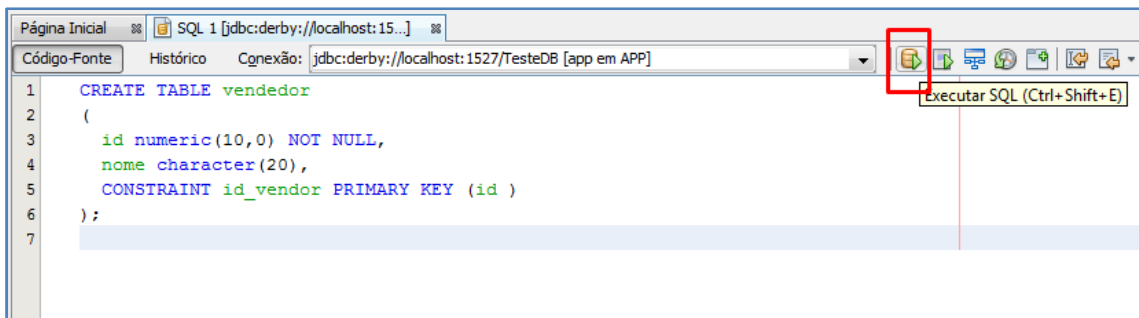


Figura 5 – Área de edição de Scripts SQL, com o ícone “Executar SQL” circulado em vermelho.

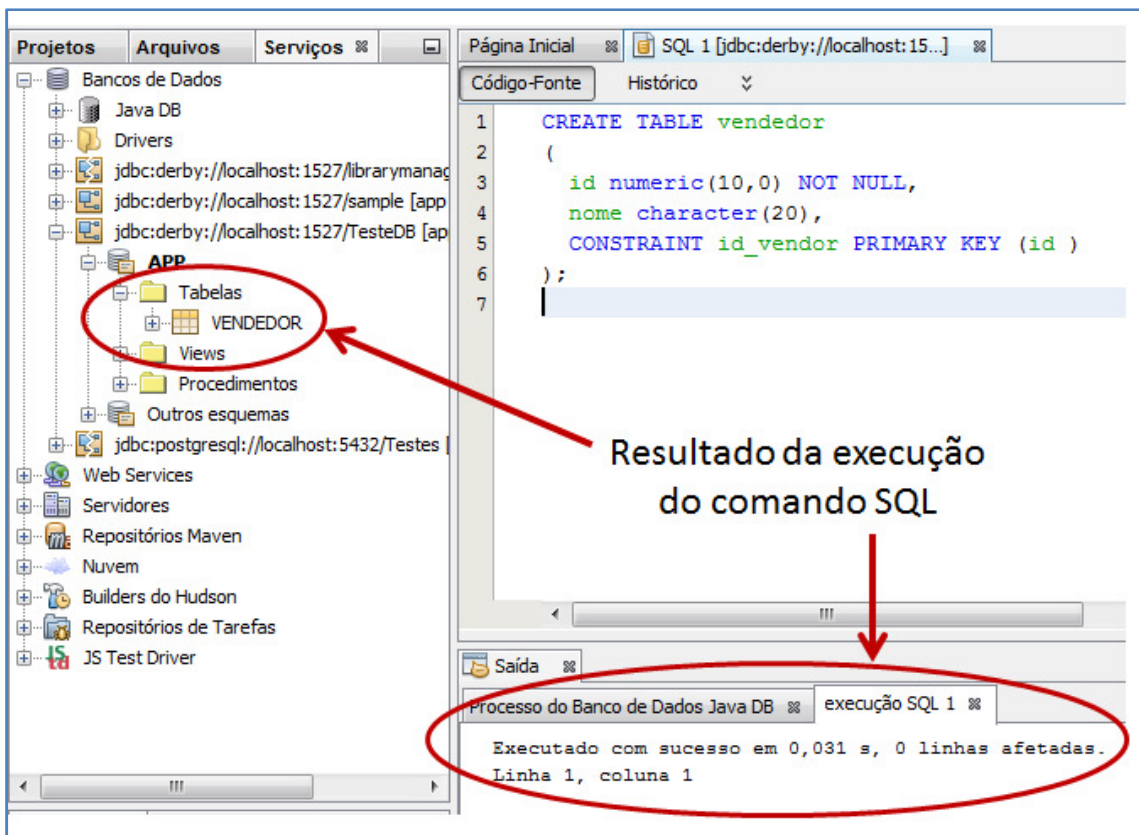


Figura 6 – Resultados da execução do script SQL: tabela criada no banco de dados, e janela de log mostrando que o comando SQL foi executado com sucesso.

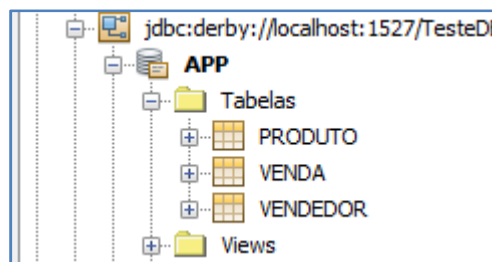


Figura 7 – Resultado final após criação de todas as tabelas.

Neste momento, estão disponíveis as três tabelas para manipular a base de dados. Agora, é necessário criar a camada de dados do aplicativo, que vai usar a JPA.

## Criando o Projeto de Teste

Para fazer os testes com a JPA, será usado um projeto simples, executado por linha de comando, que terá uma camada de dados, a ser desenvolvida usando os recursos fornecidos pela JPA. Para isso, use a opção Novo Projeto -> Java -> Aplicação Java, conforme mostrado na Figura 8.

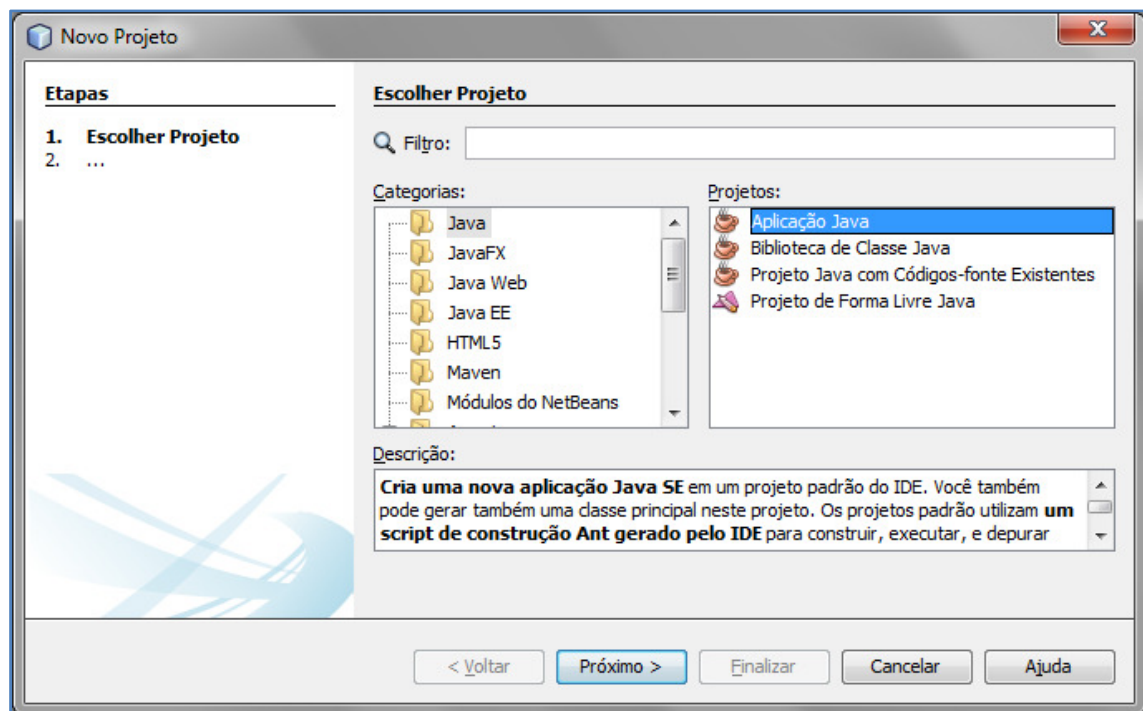


Figura 8 – Criação de novo projeto no NetBeans.

Depois de selecionado o projeto, clique em Próximo, e na tela que abrir, dê um nome ao projeto (neste exemplo, é usado o nome TesteJPA) e clique em Finalizar. Isso fará com que a área de trabalho do NetBeans seja similar à imagem mostrada na Figura 9.

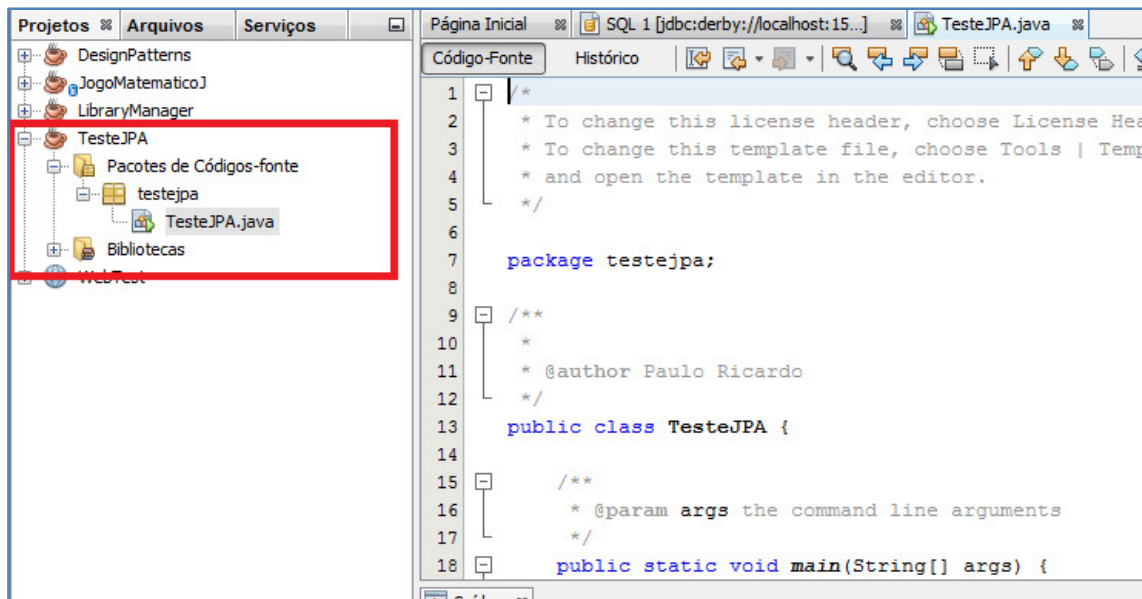


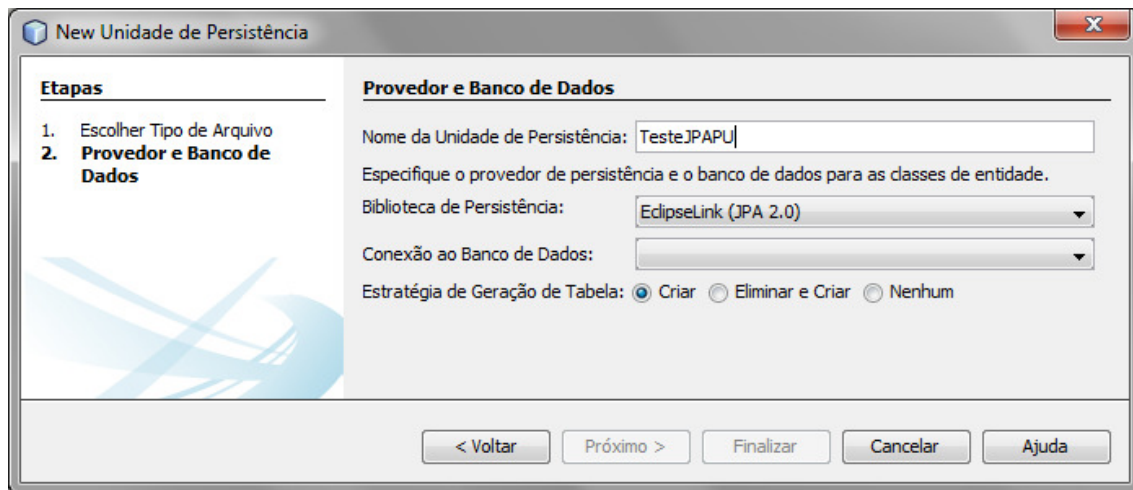
Figura 9 – Área de trabalho do NetBeans com o projeto criado, destacado em vermelho.

Após estes passos, o projeto já está pronto para ser desenvolvido, e podemos criar a camada de dados que ele vai usar.

## Criando a Camada de Dados

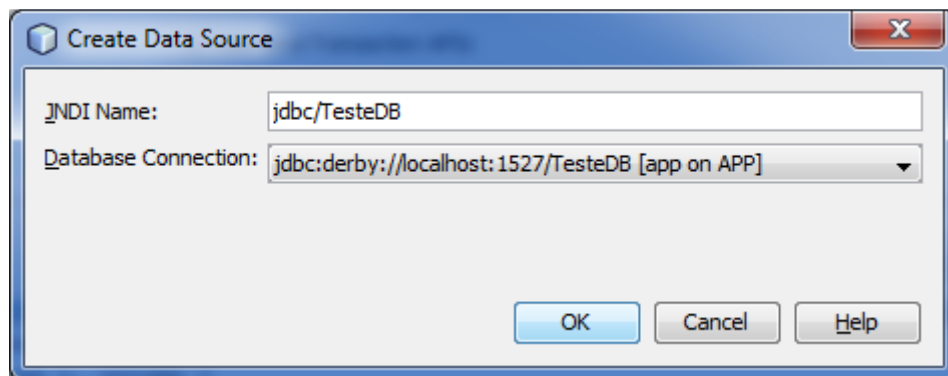
A camada de dados é a parte da aplicação onde são executadas todas as classes que mapeiam as tabelas do bando de dados. Nessa camada, há as classes de entidade, que são usadas para persistir os dados. No contexto de desenvolvimento de software, persistência significa o ato de armazenar automaticamente os dados contidos em objetos em bases de dados. Como a API de Persistência Java (JPA) é uma tecnologia de mapeamento objeto-relacional, ela necessita das classes de entidade para evitar a escrita explícita de código SQL para realizar essa tarefa para as operações de consulta, inserção, atualização e remoção dos dados.

Para adicionar as entidades ao projeto, é necessário ter uma unidade de persistência criada. Para isso, clique com o botão direito do mouse no nome do projeto (TesteJPA neste roteiro) e siga os seguintes passos a partir do menu que aparece: Novo -> Outro -> Persistence -> Unidade de Persistência; para que apareça a tela da Figura 10.



*Figura 10 – Tela inicial para configuração da Unidade de Persistência*

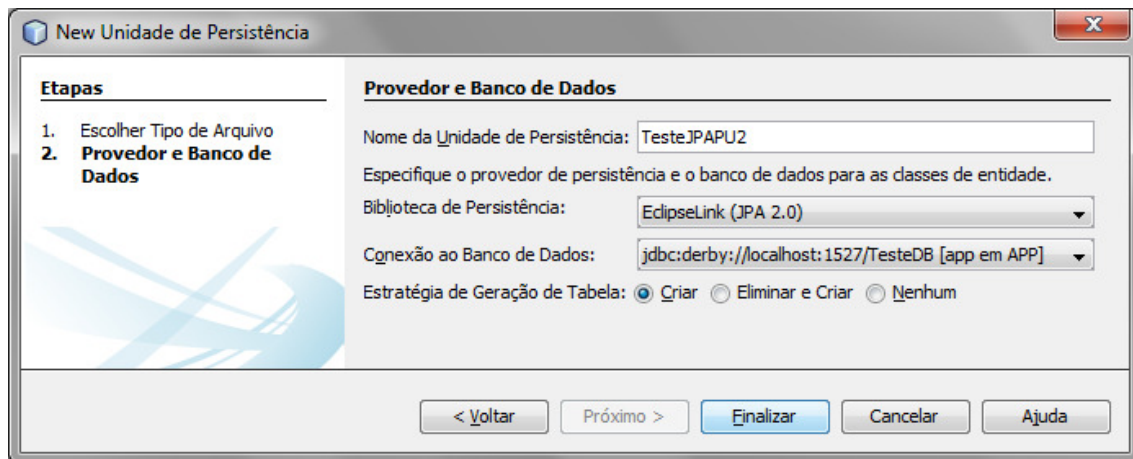
No combo “Fonte de Dados”, selecione a opção “Conexão ao Banco de Dasos” para abrir a tela apresentada na Figura 11. Defina um nome JNDI de modo análogo ao da Figura 11, para conectar ao banco de dados usado, e clique no botão Ok.



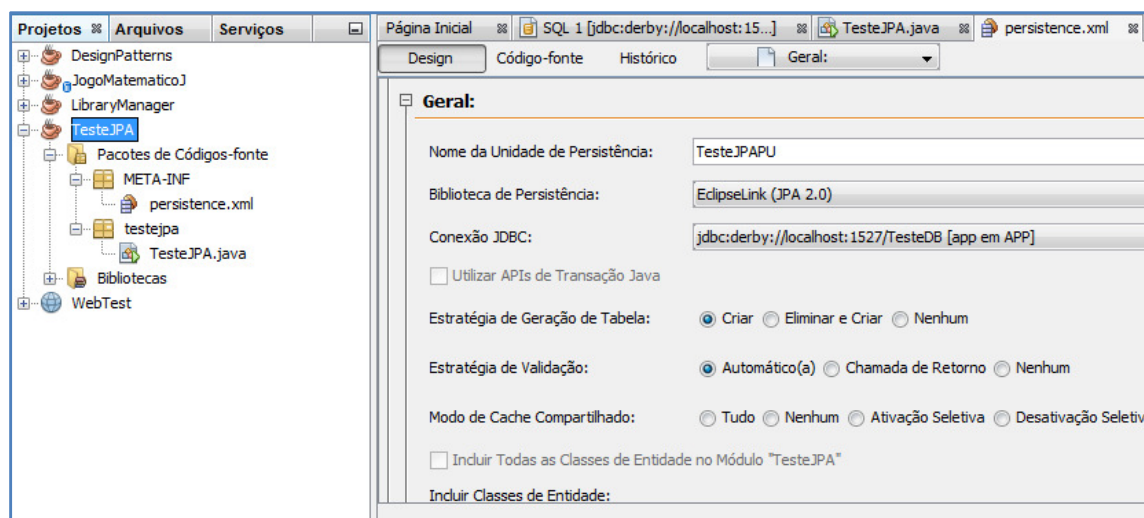
*Figura 11 – Tela com a Nova Fonte de Dados já configurada*

A Figura 12 apresenta a tela com a configuração pretendida pela unidade de persistência. Se a tela estiver semelhante à da Figura 12, clique no botão Finalizar. O resultado disso é apresentado na Figura 13.





*Figura 12 – Configuração final da Unidade de Persistência*



*Figura 13 – Ambiente do NetBeans pós criação da Unidade de Persistência (arquivo persistence.xml)*

Como pode ser observado na Figura 13, foi criado no projeto TesteJPA o arquivo persistence.xml. Esse arquivo contém a configuração de acesso à base de dados, e será usado pela JPA no momento em que ela precisar acessar o banco para persistir os dados. A partir deste momento, é possível adicionar as classes de entidade ao projeto.

Para isso, clique com o botão direito do mouse no nome do projeto e siga as opções: Novo -> Classes de Entidade a Partir do Banco de Dados para abrir a tela apresentada na Figura 14.



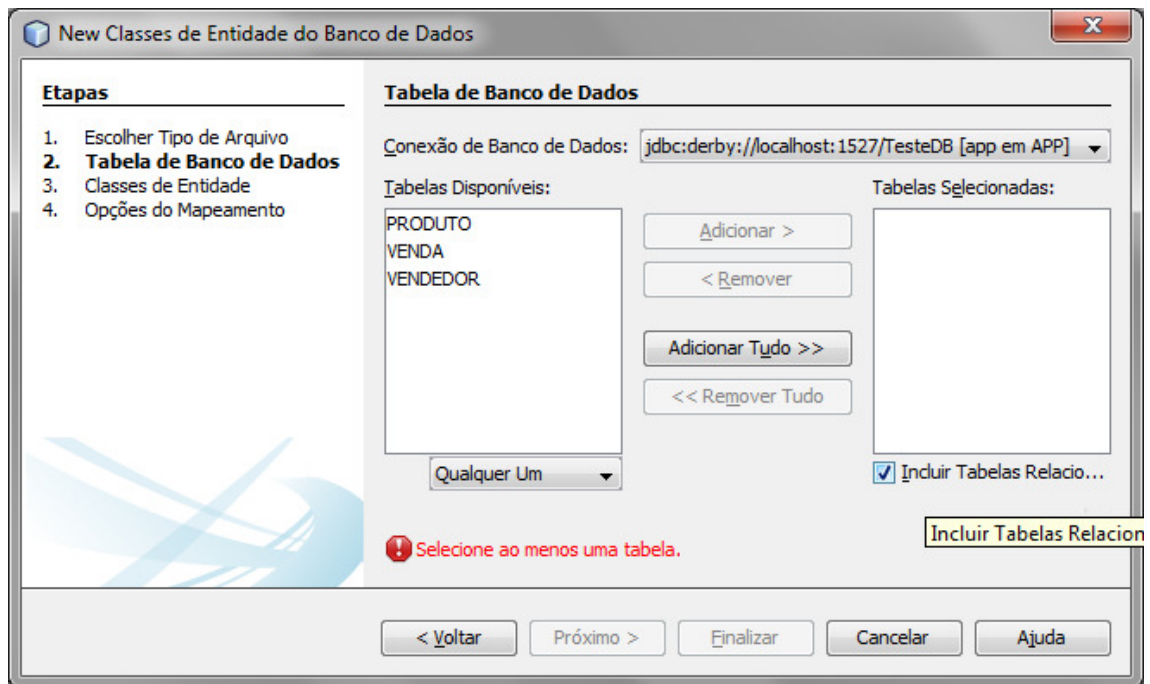


Figura 14 – Seleção das classes de entidade

Como a unidade de persistência já existe, este utilitário do NetBeans já aparece com todas as tabelas existentes no banco de dados, e com o combo “Conexão de Banco de Dados” já preenchido. Neste momento, clique no botão “Adicionar Tudo” e depois, selecione o botão “Próximo” para a ir para a tela apresentada na Figura 15.

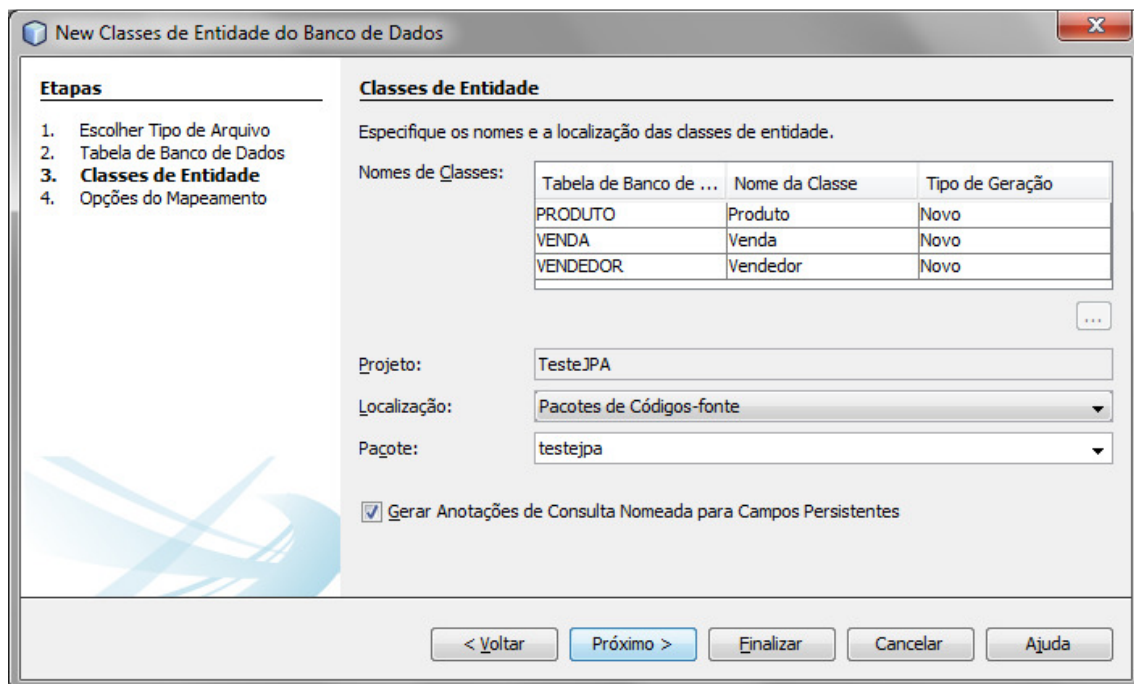


Figura 15 – Tela para configuração da das classes de entidades

Devido a uma prática de programação em Java, é convencionado armazenar as classes em pacotes, sendo que cada pacote deve conter as classes associadas à mesma função. Como eu quero criar uma camada de dados, eu crio um pacote persistência, como pode ser observado na Figura 16. Depois que atribuir o nome do pacote, clique no botão Finalizar, isso criará a tela apresentada pela Figura 17, onde é possível notar as classes de entidade já criadas e adicionadas ao projeto.

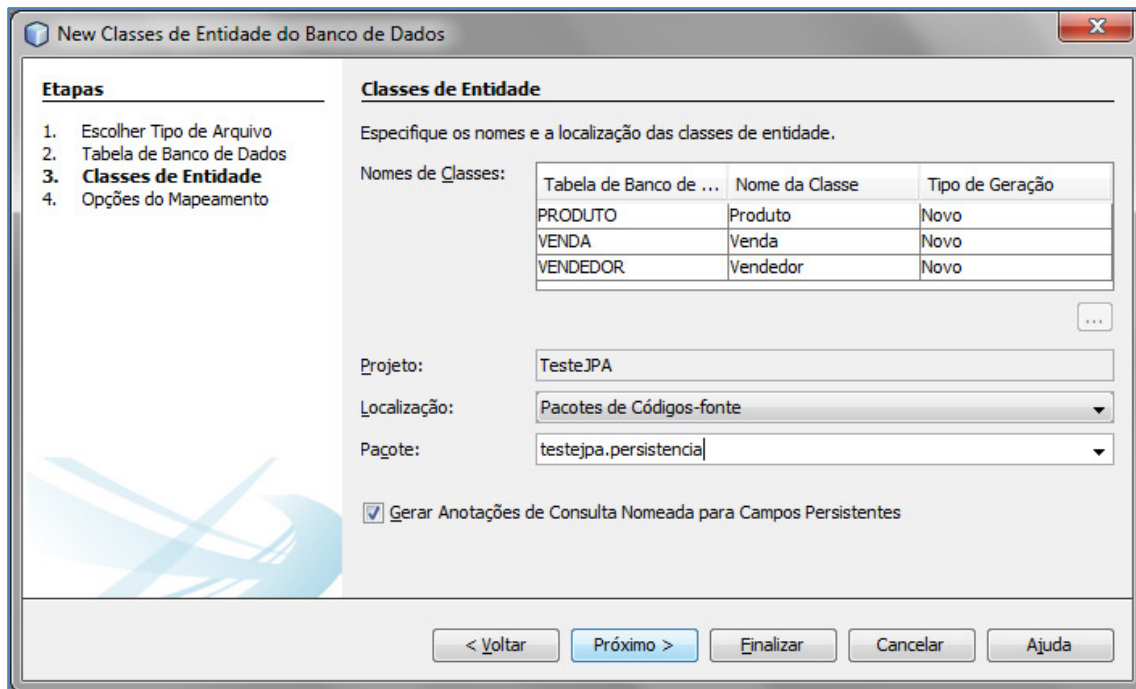


Figura 16 – Declaração do nome do pacote de persistência.

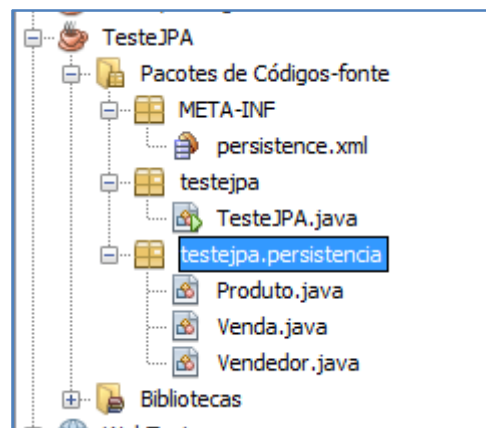


Figura 17 – Aparência do projeto após adicionar a camada de persistência.

Ao clicar duas vezes no nome de um dos arquivos java, o código-fonte da classe é aberto para edição e pode-se ver o resultado gerado pelo uso da JPA. Um ponto que vale destacar são as anotações `@NamedQuery`, que já montam os comandos SQL para seleção de dados nas tabelas relacionadas no banco de dados. O próximo passo é criar o controlador de

JPA, que conectará o aplicativo às classes de entidade. Para isso, clique com o botão direito no nome do projeto (TesteJPA) e use as opções mostradas na Figura 18.

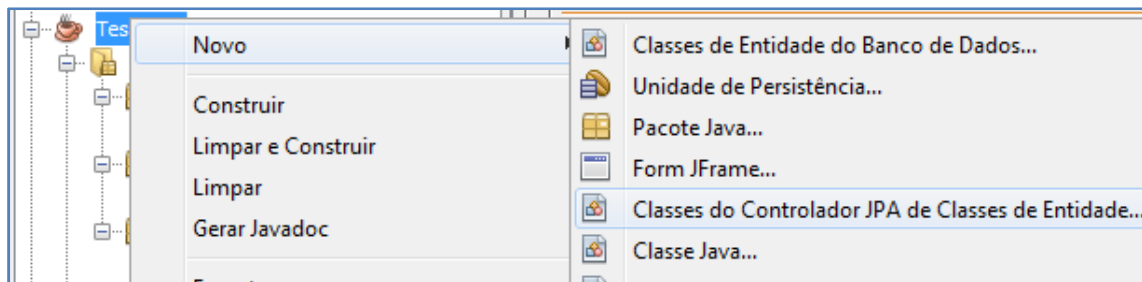


Figura 18 – Configuração do Controlador de JPA.

Isso fará com que apareça uma tela similar à da Figura 19, onde preciso indicar quais são as classes que precisam do controlador. Como serão usadas todas, use a opção “Adicionar Tudo”, e depois, clique no botão Próximo.

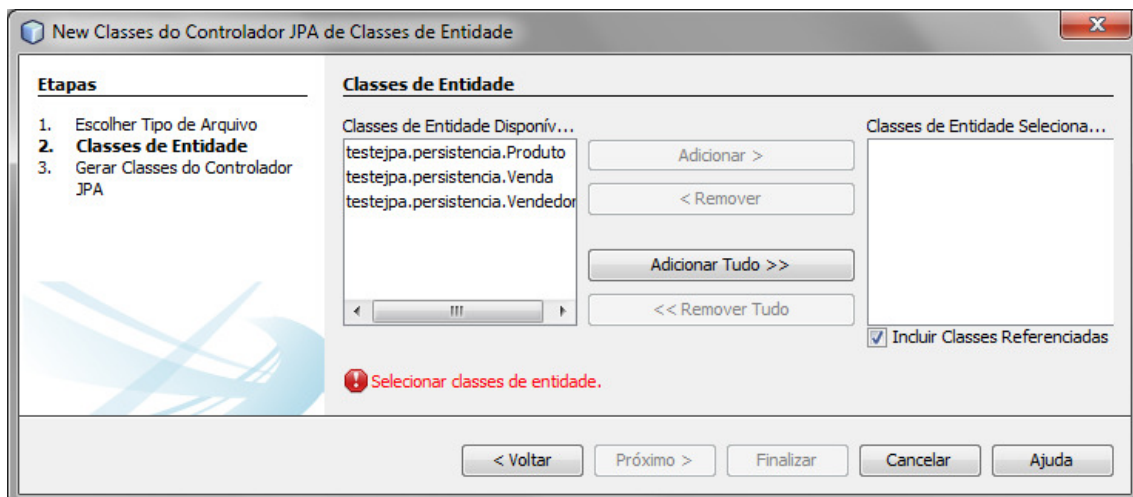


Figura 19 – Seleção das Classes de Entidade para criar o controlador de JPA.

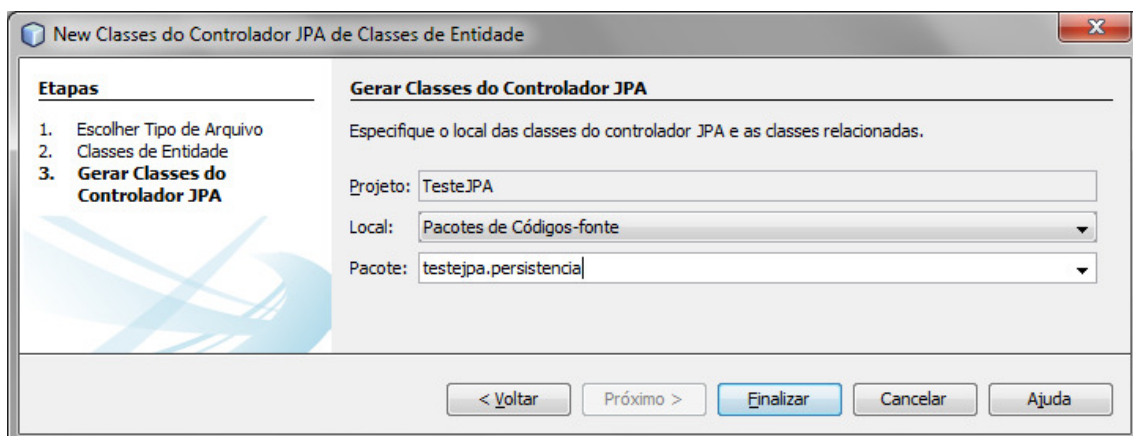
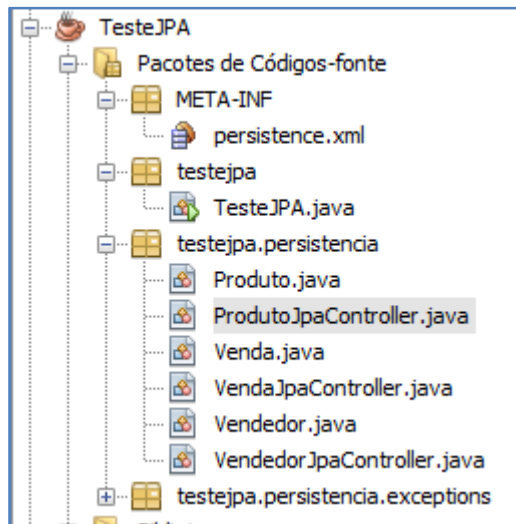


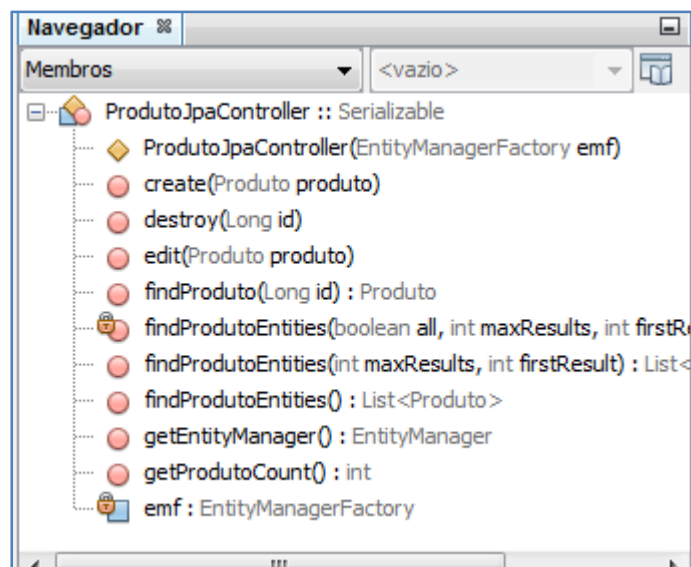
Figura 20 – Definição do pacote onde ficará a unidade de persistência.

Depois de clicar em Próximo, surgirá uma tela para configurar onde as classes de controlador ficarão armazenadas. Na opção Pacote dessa nova tela, selecione a opção que contém persistência, conforme a Figura 20, e depois, clique no botão Finalizar, fazendo com que o projeto do NetBeans seja similar ao da Figura 21.



*Figura 21 – Aparência do projeto após a adição das classes de controle JPA.*

Agora, é possível afirmar que a camada de dados está pronta, e podemos proceder aos próximos passos. Na Figura 22, aparecem todos os métodos que estão disponíveis para manipular os dados. São esses métodos que serão usados para interagir com a base de dados. Agora, já é possível desenvolver a parte da aplicação que servirá de interface para o aplicativo coletar e manipular os dados fornecidos pelo usuário para.



*Figura 22 – Métodos internos da classe de controle JPA, que podem ser usados pelo aplicativo para interagir com a base de dados.*

## Criando a Interface de Usuário

Como dito anteriormente, esta interface será desenvolvida através de linha de comando, para facilitar esta tarefa.

Observando a Figura 22, percebe-se que há um pacote chamado `testeJpa`, que contém uma classe chamada `TesteJpa.java`. É nesse pacote que serão armazenadas as classes para a interface de usuário, sendo que as primeiras a serem feitas serão usadas para manipular os dados de Produto e Vendedor.

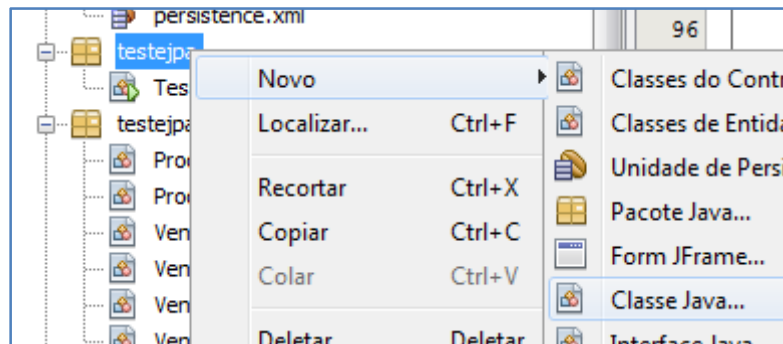


Figura 23 –Adicionando uma Classe Java ao pacote testeJpa.

Para isso, clique com o botão direito do mouse no nome do pacote, e use as opções mostradas na Figura 23. Ao clicar nessa opção, será aberta uma tela para que o desenvolvedor dê um nome à classe, e depois que der o nome à classe, clique no botão Finalizar, e é apenas isso que deve ser feito, tanto para Produto quanto Vendedor. Isso fará com que o projeto do NetBeans fique com a aparência da Figura 24.

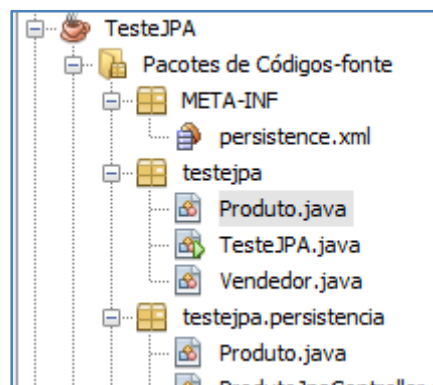
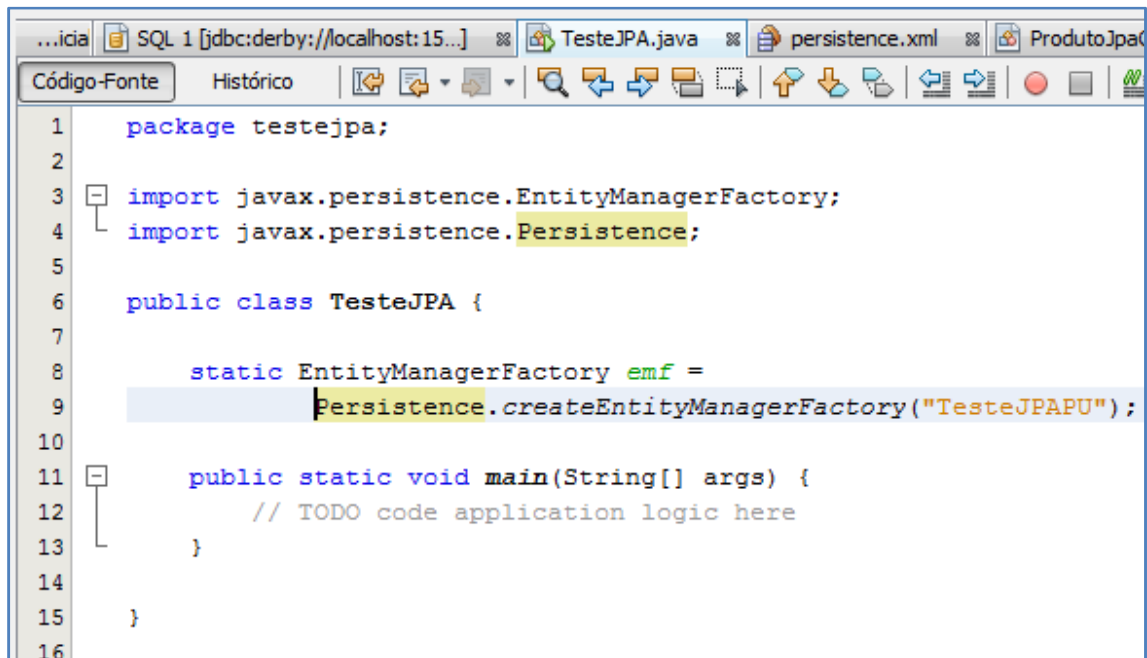


Figura 24 – Adicionando uma Classe Java ao pacote testeJpa.

A primeira coisa a fazer neste projeto, é criar um gerenciador de persistência, pois é um aplicativo desktop, executado fora do ambiente de um servidor de aplicações. Normalmente, quem gerencia a persistência é o servidor de aplicações, mas neste exemplo, como ele não está sendo usado, isso precisa ser simulada.

Essa ação é feita pela instrução `EntityManagerFactory emf = Persistence.createEntityManagerFactory("TesteJPAPU");`, sendo que o nome TesteJPAPU, é o

nome da unidade de persistência que está definido no arquivo *persistence.xml*. Por uma questão de conveniência, esse comando deve ficar no arquivo “*TesteJPA.java*”, que é a classe principal deste projeto (o arquivo que contém o método *main()*). Por isso, esse arquivo deve ser modificado para ficar semelhante ao que aparece na Figura 25.



```
1 package teste.jpa;
2
3 import javax.persistence.EntityManagerFactory;
4 import javax.persistence.Persistence;
5
6 public class TesteJPA {
7
8     static EntityManagerFactory emf =
9         Persistence.createEntityManagerFactory("TesteJPAPU");
10
11     public static void main(String[] args) {
12         // TODO code application logic here
13     }
14
15 }
16
```

Figura 25 –Classe *TesteJPA* após adicionar o gerenciador de persistência.

O primeiro teste é inserir um novo produto na base de dados, e depois, listar todo o conteúdo que está no banco, após inserir um novo item no mesmo. Por isso é necessário criar na classe produto do pacote teste.jpa dois métodos para isso, além de um método construtor que recebe o gerenciador de persistência.

Assim, a estrutura da classe Produto, no pacote teste.jpa, deve ficar semelhante ao que mostra a Figura 26.

Um método construtor que recebe o gerenciador de persistência é obrigatório, pois ele será usado como referência em todas as transações de banco de dados efetuadas pela JPA, e ele obriga que essa informação seja fornecida sempre que a classe for instanciada.

```

1  package testeipa;
2
3  import javax.persistence.EntityManagerFactory;
4  import testeipa.persistencia.ProdutoJpaController;
5
6  public class Produto {
7
8      private ProdutoJpaController prodJC = null;
9      private testeipa.persistencia.Produto p = null;
10
11     public Produto(EntityManagerFactory emf) {
12         prodJC = new ProdutoJpaController(emf);
13     }
14
15     public void criarNovoProduto() {
16         |
17     }
18
19     public void listaTudo() {
20
21     }
22
23 }
24

```

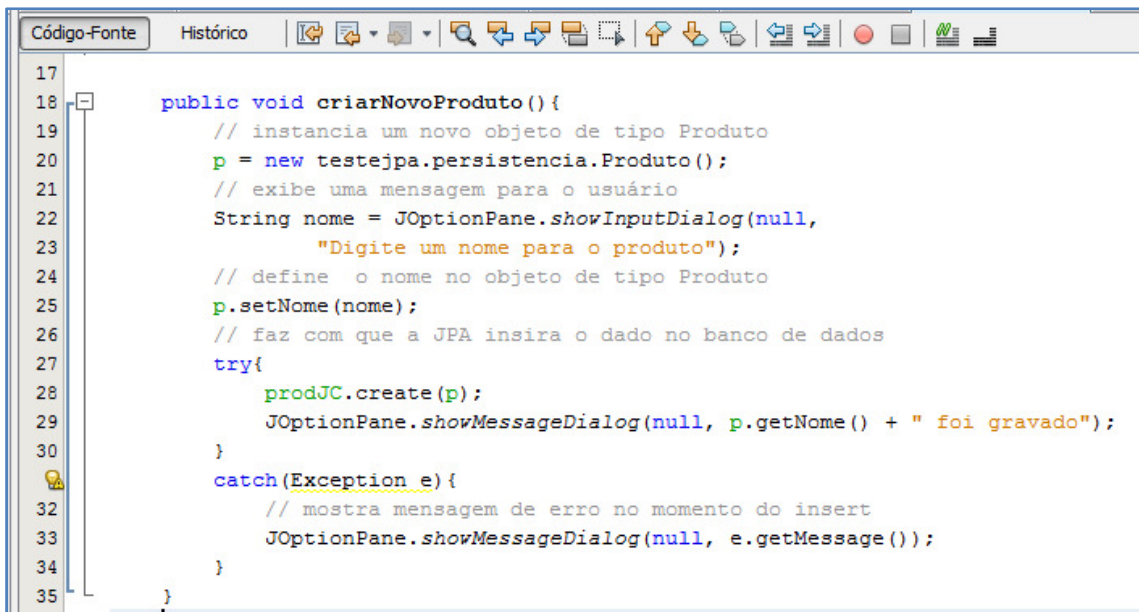
Figura 26 –Estrutura da classe Produto após as alterações iniciais.

Neste momento, trabalharemos apenas com o método *criarNovoProtudo()*. O que ele precisa fazer, resumidamente, é perguntar ao usuário o nome que ele quer que o produto a ser cadastrado no banco de dados tenha.

O funcionamento básico dele será exibir uma caixa de mensagem solicitando que o usuário selecione um produto para ser cadastrado. O valor fornecido pelo usuário será armazenado num objeto de tipo Produto, que é a classe de entidade no pacote persistência. Depois, o programa deve chamar o método *create()* da classe controladora de JPA para gravar o objeto no banco de dados. Se a operação for bem sucedida, o programa deve apresentar uma mensagem indicado que o objeto foi persistido no banco de dados, caso contrário, deve ser apresentada uma mensagem indicando o erro que ocorreu no momento em que o objeto tentou ser persistido no banco de dados.

Assim, a programação do método *criarNovoProtudo()* deve ser o que é apresentado na Figura 27.

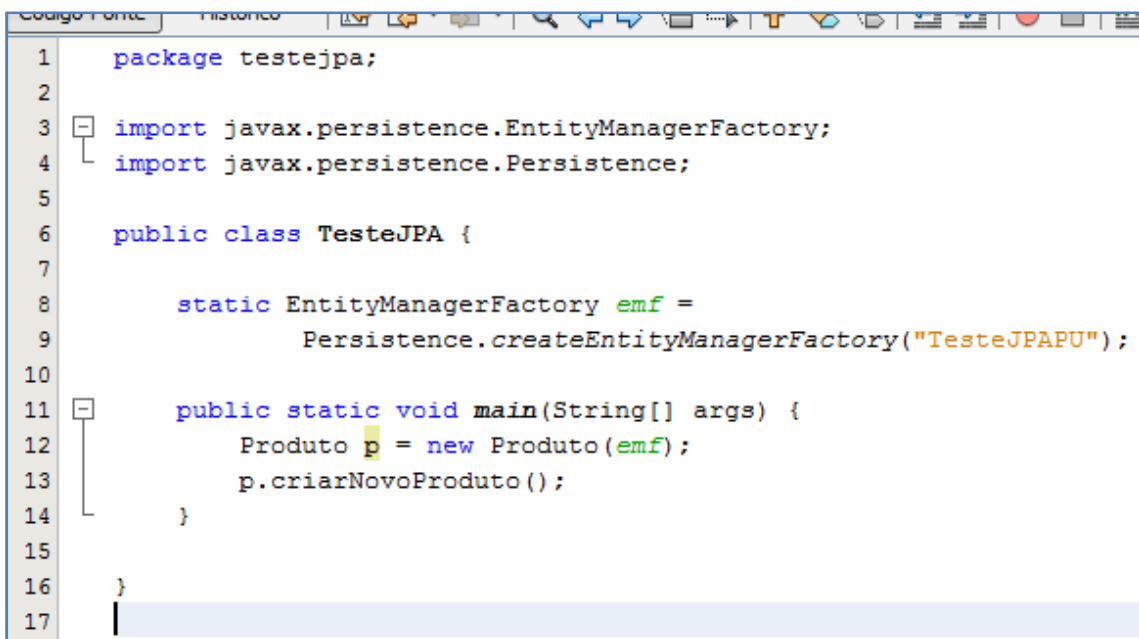




```
17
18 public void criarNovoProduto() {
19     // instancia um novo objeto de tipo Produto
20     p = new testejpa.persistencia.Produto();
21     // exibe uma mensagem para o usuário
22     String nome = JOptionPane.showInputDialog(null,
23         "Digite um nome para o produto");
24     // define o nome no objeto de tipo Produto
25     p.setNome(nome);
26     // faz com que a JPA insira o dado no banco de dados
27     try {
28         prodJC.create(p);
29         JOptionPane.showMessageDialog(null, p.getNome() + " foi gravado");
30     }
31     catch (Exception e) {
32         // mostra mensagem de erro no momento do insert
33         JOptionPane.showMessageDialog(null, e.getMessage());
34     }
35 }
```

Figura 27 –Programação do método criarNovoProtudo().

Para que esse método possa ser testado, a classe TesteJPA deve ser modificado para o código apresentado na Figura 28.



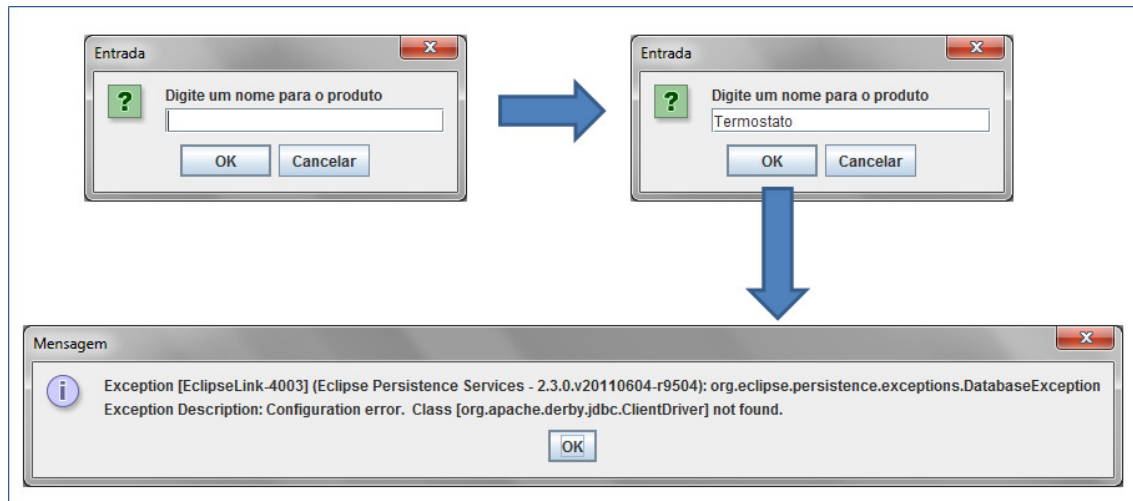
```
1 package testejpa;
2
3 import javax.persistence.EntityManagerFactory;
4 import javax.persistence.Persistence;
5
6 public class TesteJPA {
7
8     static EntityManagerFactory emf =
9         Persistence.createEntityManagerFactory("TesteJPAPU");
10
11     public static void main(String[] args) {
12         Produto p = new Produto(emf);
13         p.criarNovoProduto();
14     }
15
16 }
17
```

Figura 28 –Atualização na classe TesteJPA.

A atualização dessa classe consiste em mudar o código do método *main()* para que ele use o método *ciarNovoProduto()* da classe *Produto*, que fará a interação com o usuário. Então, ele precisa da instrução “*Produto p = new Produto(emf);*” para criar um objeto de tipo *Produto*, que usará um gerenciador de persistência definido pelo atributo *emf*. Depois que esse objeto

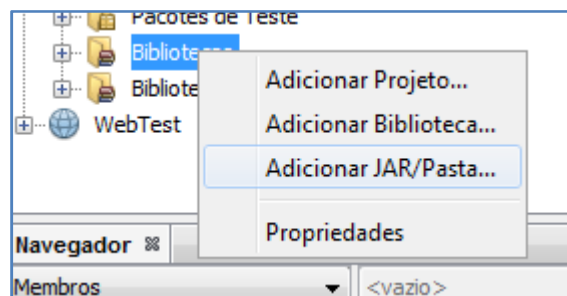
for criado, a instrução “*p.criarNovoProduto();*” executará as instruções para gravar a informação na base de dados.

Assim, ao ser executado, o aplicativo deve apresentar a sequência de telas mostrada na Figura 29. Como pode ser observado, é um erro causado pela falta do driver cliente de acesso à base de dados.



*Figura 29—Sequência de telas no primeiro teste.*

Isso demonstra o erro interceptado pela instrução “*catch(Exception e)*” existente no arquivo *Produto.java*, da pasta *testeJpa*. Para corrigir esse problema, clique com o botão direito do mouse em *Bibliotecas* no projeto *TesteJPA*, e use a opção *Adicionar JAR/Pasta*, conforme mostrado na Figura 30.



*Figura 30—Adicionando bibliotecas ao projeto.*

Isso fará com que seja aberta a tela da Figura 31, onde deve ser selecionado o arquivo “*derbyclient.jar*”, que é o driver de conexão com a base de dados usada no projeto. Cada banco de dados possui seu próprio driver de conexão, e esta opção deve ser usada de acordo com o banco de dados usado no projeto. A localização desses arquivos também depende do seu ambiente de testes e/ou desenvolvimento.

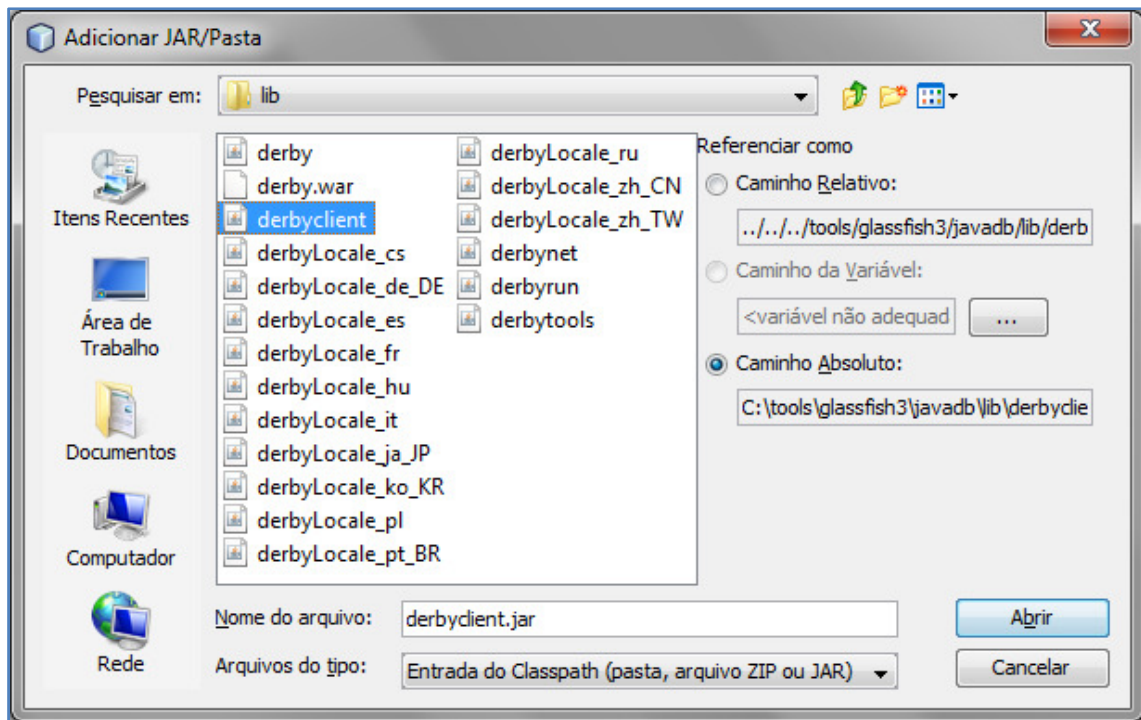


Figura 31–Adicionando a biblioteca derbyclient.jar ao projeto.

Depois de adicionar esse arquivo, teste novamente o projeto, e a sequência de telas deve ser a mesma da Figura 32. Agora, posso inserir vários valores para lidar com uma consulta ao banco de dados.

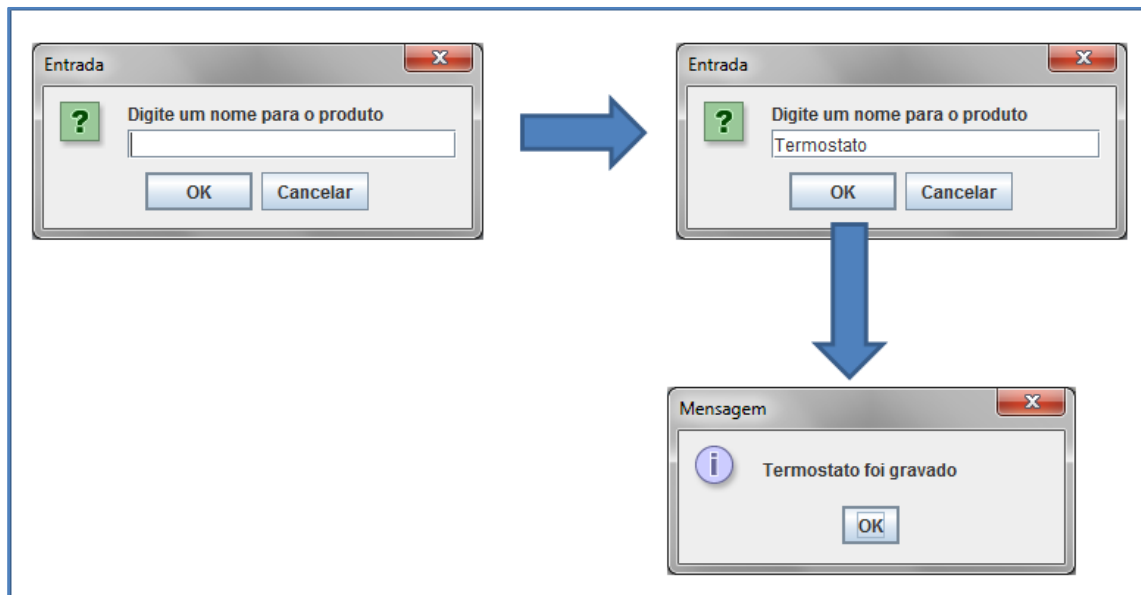


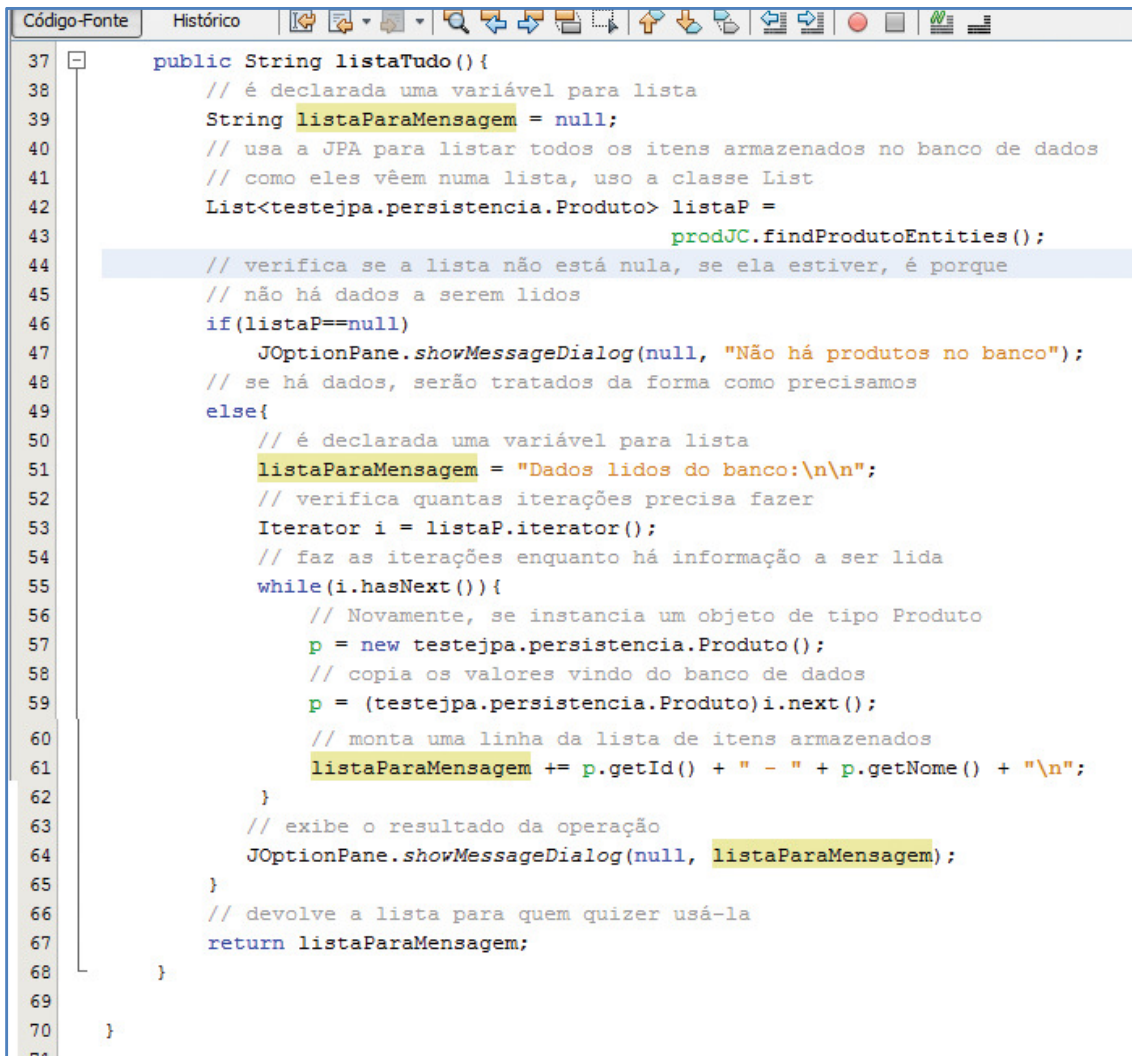
Figura 32–Execução bem sucedida do projeto.

## Consultando Dados no Banco de Dados

Para consultar os dados do banco de dados, será usado o método “public void listaTudo()” da classe Produto. Mas, por uma conveniência do projeto, ela terá a

declaração alterada para “*publicStringlistaTudo()*”, pois mais tarde, essa lista será usada para alterar e excluir dados.

O que esse método (código-fonte na Figura 33) precisa fazer é listar tudo o que está disponível na base de dados, e apresentar essas informações ao usuário. Por isso, ele usa o método “*findProdutoEntities()*” do controlador de JPA. Esse método devolve um array na forma de uma Lista de objetos do tipo Produto, caso tenha algum registro na base de dados, ou null, quando não há nenhum registro na base de dados.



```
37 public String listaTudo() {
38     // é declarada uma variável para lista
39     String listaParaMensagem = null;
40     // usa a JPA para listar todos os itens armazenados no banco de dados
41     // como eles vêm numa lista, uso a classe List
42     List<teste.jpa.persistencia.Produto> listaP =
43         prodJC.findProdutoEntities();
44     // verifica se a lista não está nula, se ela estiver, é porque
45     // não há dados a serem lidos
46     if(listaP==null)
47         JOptionPane.showMessageDialog(null, "Não há produtos no banco");
48     // se há dados, serão tratados da forma como precisamos
49     else{
50         // é declarada uma variável para lista
51         listaParaMensagem = "Dados lidos do banco:\n\n";
52         // verifica quantas iterações precisa fazer
53         Iterator i = listaP.iterator();
54         // faz as iterações enquanto há informação a ser lida
55         while(i.hasNext()) {
56             // Novamente, se instancia um objeto de tipo Produto
57             p = new teste.jpa.persistencia.Produto();
58             // copia os valores vindo do banco de dados
59             p = (teste.jpa.persistencia.Produto)i.next();
60             // monta uma linha da lista de itens armazenados
61             listaParaMensagem += p.getId() + " - " + p.getNome() + "\n";
62         }
63         // exibe o resultado da operação
64         JOptionPane.showMessageDialog(null, listaParaMensagem);
65     }
66     // devolve a lista para quem quiser usá-la
67     return listaParaMensagem;
68 }
69
70 }
```

Figura 33 – Programação do método *publicStringlistaTudo()*.

Depois que esses registros forem lidos da base de dados, eles são tratados através de um iterador, que informará o número de registros lidos na base de dados, e permitirá lê-los registro por registro. São respectivamente as instruções “*Iterator i = listaP.iterator();*” e “*p = (teste.jpa.persistencia.Produto)i.next();*”. Isso pode ser útil quando se quer excluir algum registro que não se deseja mostrar ao usuário, mas não é este o caso.

Neste exemplo, a cada iteração, é adicionada uma linha formatada à lista que será mostrada ao usuário através da instrução “*listaParaMensagem += p.getId() + " - " + p.getNome() + "\n";*”. Depois que forem lidos todos os registros do banco de dados, o

resultado é exibido através da instrução `"JOptionPane.showMessageDialog(null, listaParaMensagem);"`.

Por último, o método `main()` da classe `TesteJPA.java` será alterado para mostrar uma lista de opções para o usuário, e o código-fonte dessa classe deve ser igual ao da Figura 34.

```
12 public static void main(String[] args) {  
13     int opcao = 0;  
14     String listaOpcoes = "1 - Cadastrar Novo Produto\n";  
15     listaOpcoes += "2 - Listar Todos os Produtos\n";  
16     listaOpcoes += "9 - Encerrar\n";  
17     Produto p = new Produto(emf);  
18  
19     while (1==1) {  
20         opcao = Integer.parseInt(JOptionPane.showInputDialog(listaOpcoes));  
21         switch (opcao) {  
22             case 1:  
23                 p.criarNovoProduto();  
24                 break;  
25             case 2:  
26                 p.listaTudo();  
27                 break;  
28             case 9:  
29                 return;  
30             }  
31     }
```

Figura 34 – Atualização na classe `TesteJPA`.

Após essas alterações, o resultado do teste deve ter a sequência de telas apresentada na Figura 35.

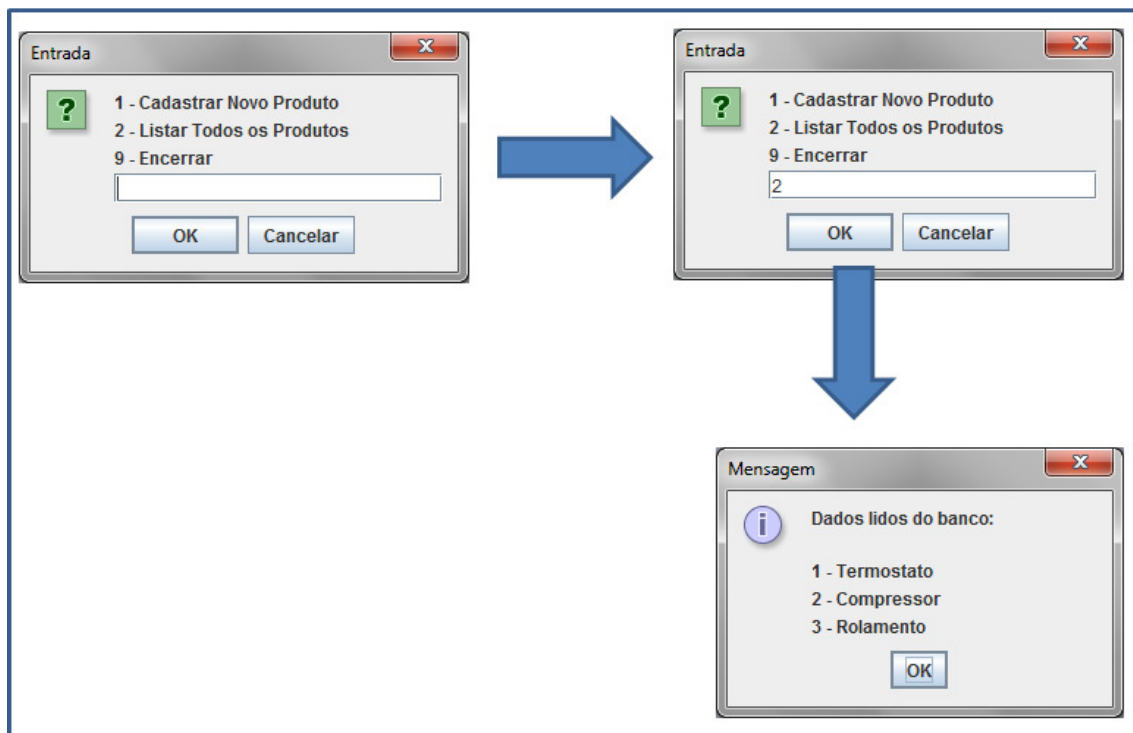


Figura 35–Execução bem sucedida do projeto.

## Editando e Apagando Registros do Banco de Dados

As próximas ações são a exclusão e edição de registros na base de dados. Para isso, também serão usados os métodos de JPA.

O método `edit()` que existe no controlador de JPA é o método que atualiza um objeto. Neste exemplo, ele será usado em conjunto com o método “`findProduto(Integer id)`” que lê um registro no banco de dados, a partir de um determinado ID passado como parâmetro. Para realizar esse teste, é criado o método “`publicvoideditaProduto()`” na classe `Produto` (código-fonte na Figura 36).

```
71 public void editaProduto() {
72     // declara variável que recupera a lista de produtos
73     String lista = listaTudo();
74     // declara uma variável para armazenar o id selecionado pelo usuário
75     Integer id = 0;
76     id = Integer.parseInt(JOptionPane.showInputDialog(
77         "Selecione um id para editar:\n\n" + lista));
78     // lê as informações daquela id do banco de dados
79     p = new testejpa.persistencia.Produto();
80     p = prodJC.findProduto(id);
81     // lê o novo nome do produto
82     String novoNome = JOptionPane.showInputDialog(
83         "Insira um novo nome para o produto" + p.getNome());
84     // atualiza o nome do produto no objeto
85     p.setNome(novoNome);
86     // atualiza o objeto na base de dados
87     try{
88         prodJC.edit(p);
89         // se a atualização foi bem sucedida, informa usuário
90         JOptionPane.showMessageDialog(null, p.getId() + " atualizado");
91     }
92     catch (NonexistentEntityException nee) {
93         // se a entidade não foi encontrada, informa o erro
94         JOptionPane.showMessageDialog(null, nee.getMessage());
95     }
96     catch (Exception e) {
97         JOptionPane.showMessageDialog(null, e.getMessage());
98     }
99 }
100
```

Figura 36 –Programação do método `publicvoideditaProduto()`.

Para realizar esse teste, o método `main()` da classe `TesteJPA.java`, também é modificado, de acordo com a Figura 37.



```

12 public static void main(String[] args) {
13     int opcao = 0;
14     String listaOpcoes = "1 - Cadastrar Novo Produto\n";
15     listaOpcoes += "2 - Listar Todos os Produtos\n";
16     listaOpcoes += "3 - Editar Um Produto\n";
17     listaOpcoes += "9 - Encerrar\n";
18     Produto p = new Produto(emf);
19
20     while (1==1) {
21         opcao = Integer.parseInt(JOptionPane.showInputDialog(listaOpcoes));
22         switch (opcao) {
23             case 1:
24                 p.criarNovoProduto();
25                 break;
26             case 2:
27                 p.listaTudo();
28                 break;
29             case 3:
30                 p.editaProduto();
31                 break;
32             case 9:
33                 return;
34         }

```

Figura 37 – Atualização na classe TesteJPA.

Após essas alterações, o resultado do teste deve ter a sequência de telas apresentada na Figura 38.

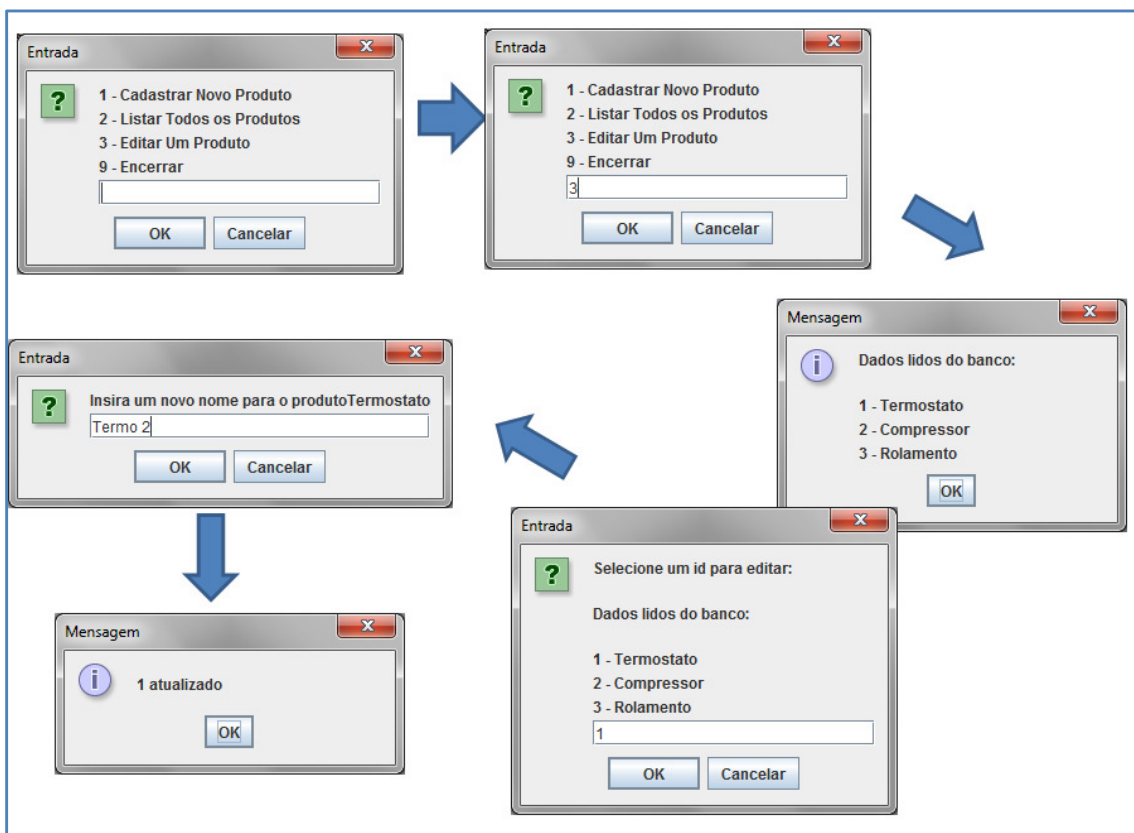


Figura 38–Execução bem sucedida do projeto.



Como pode ser observado na Figura 39, excluir um item pela JPA é bem simples. Basicamente, é a mesma lógica do método “*editaProduto()*”, sem as operações de atualização. Para testar esse método, note as atualizações na classe *TesteJPA.java* (Figura 40).

```

102 public void excluirProduto() {
103     // declara variável que recupera a lista de produtos
104     String lista = listaTudo();
105     // declara uma variável para armazenar o id selecionado pelo usuário
106     Integer id = 0;
107     id = Integer.parseInt(JOptionPane.showInputDialog(
108         "Selecione um id para excluir:\n\n" + lista));
109     // exclui o objeto na base de dados
110     try {
111         prodJC.destroy(id);
112         // se a exclusão foi bem sucedida, informa usuário
113         JOptionPane.showMessageDialog(null, id + " excluído");
114     } catch (NonexistentEntityException nee) {
115         // se a entidade não foi encontrada, informa o erro
116         JOptionPane.showMessageDialog(null, nee.getMessage());
117     } catch (Exception e) {
118         JOptionPane.showMessageDialog(null, e.getMessage());
119     }
120 }
121

```

Figura 39 – Programação do método *public void excluirProduto()*.

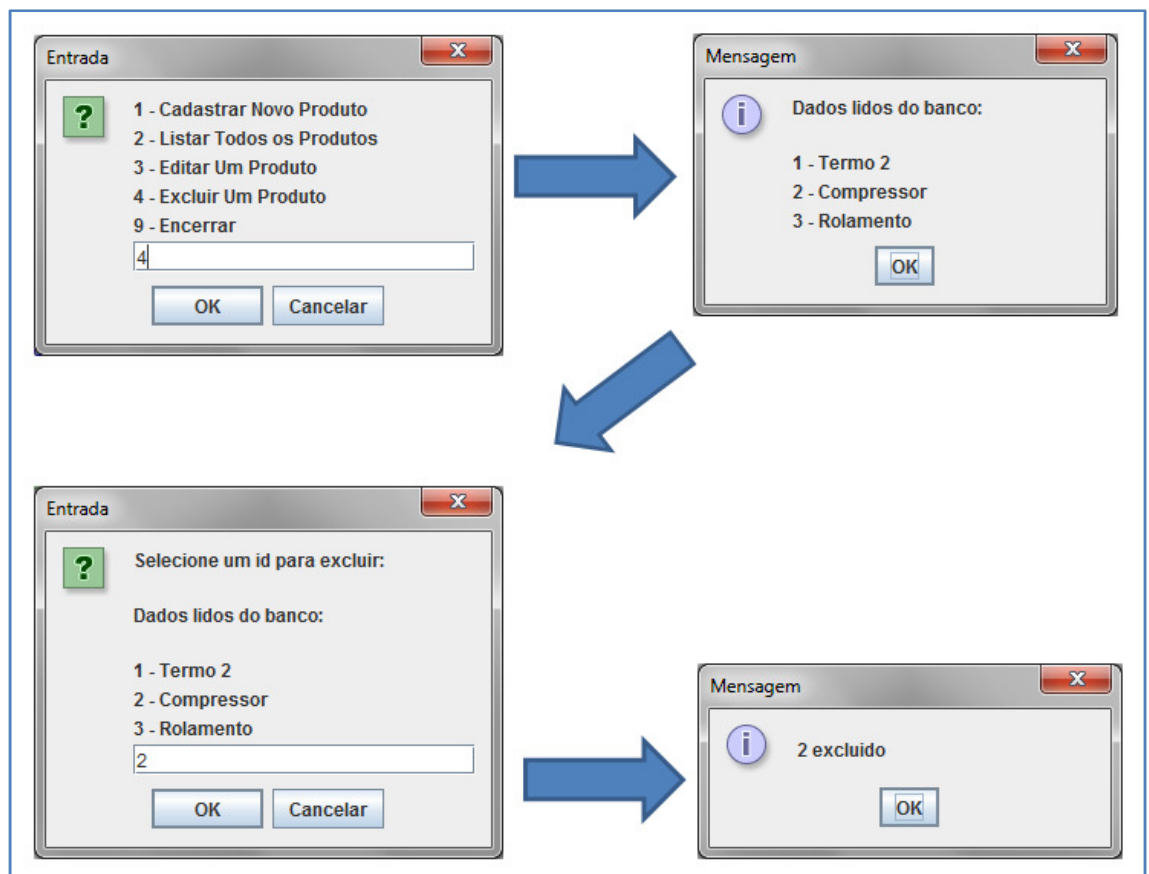
```

12 public static void main(String[] args) {
13     int opcao = 0;
14     String listaOpcoes = "1 - Cadastrar Novo Produto\n";
15     listaOpcoes += "2 - Listar Todos os Produtos\n";
16     listaOpcoes += "3 - Editar Um Produto\n";
17     listaOpcoes += "4 - Excluir Um Produto\n";
18     listaOpcoes += "9 - Encerrar\n";
19     Produto p = new Produto(emf);
20
21     while (1==1) {
22         opcao = Integer.parseInt(JOptionPane.showInputDialog(listaOpcoes));
23         switch (opcao) {
24             case 1:
25                 p.criarNovoProduto();
26                 break;
27             case 2:
28                 p.listaTudo();
29                 break;
30             case 3:
31                 p.editaProduto();
32                 break;
33             case 4:
34                 p.excluirProduto();
35                 break;
36             case 9:
37                 return;

```

Figura 40 – Atualização na classe *TesteJPA*.

Após essas alterações, o resultado do teste deve ter a sequência de telas apresentada na Figura 41.



*Figura 41—Execução bem sucedida do projeto.*