

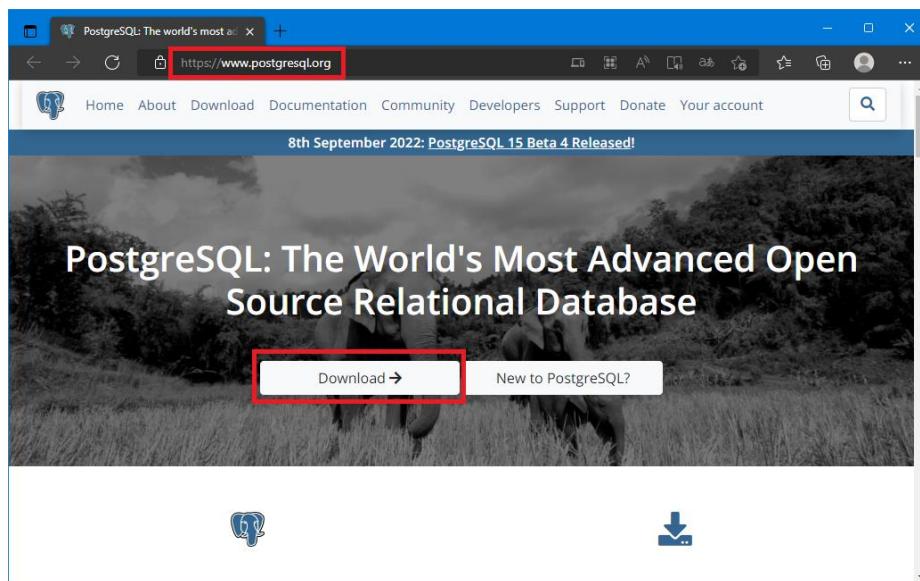
# Alterações Apostila de Java MOD II

## AULA 1

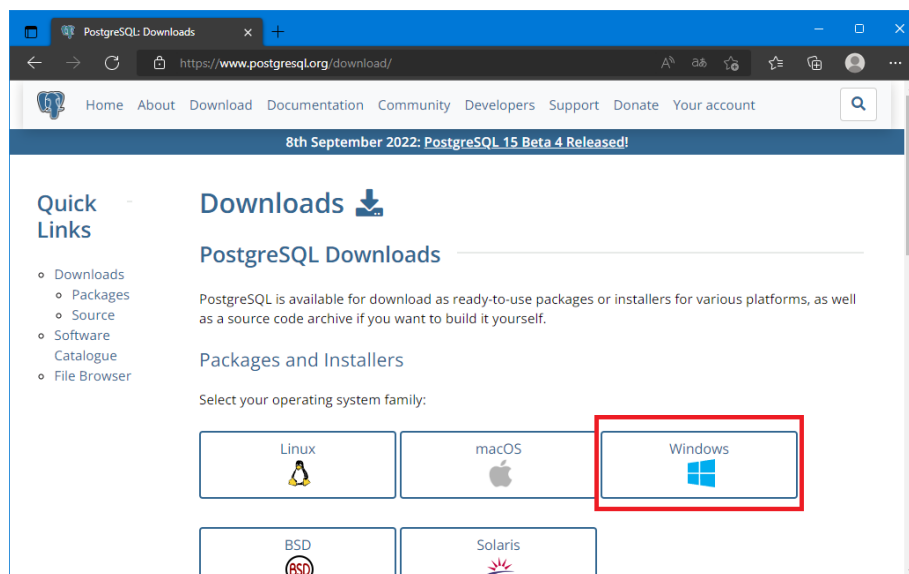
Páginas 9 a 18: Download e instalação do PostgreSQL e pgAdmin 4

Download:

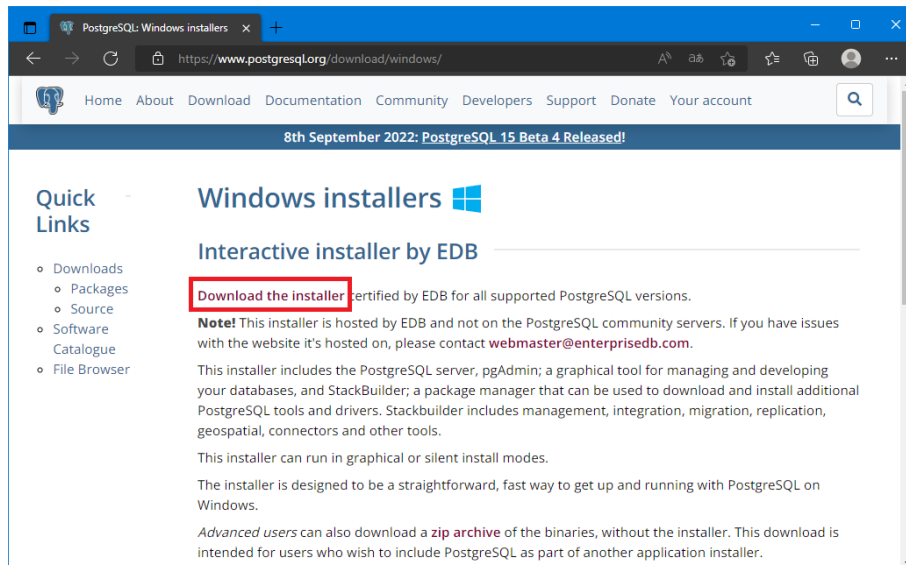
- Acesse o Site: <https://www.postgresql.org/> e em seguida clique na opção “DOWNLOAD”



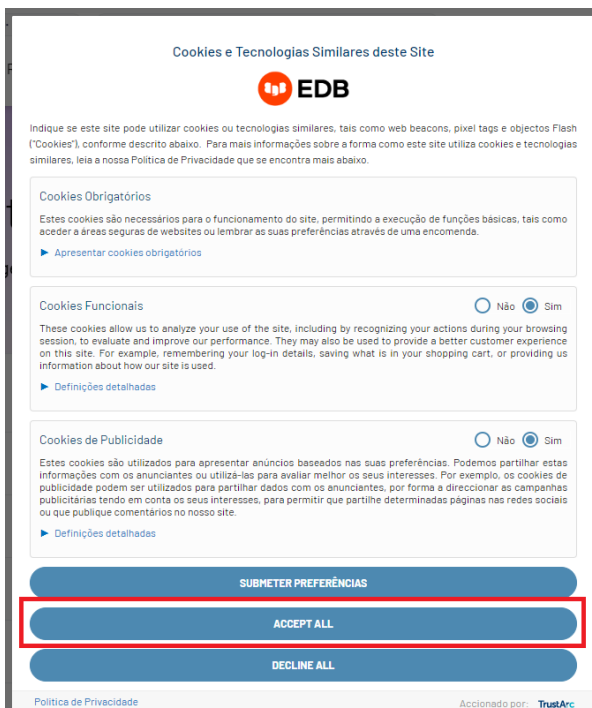
- Serão exibidas as opções de download para cada sistema operacional, no nosso caso clique na opção “Windows”



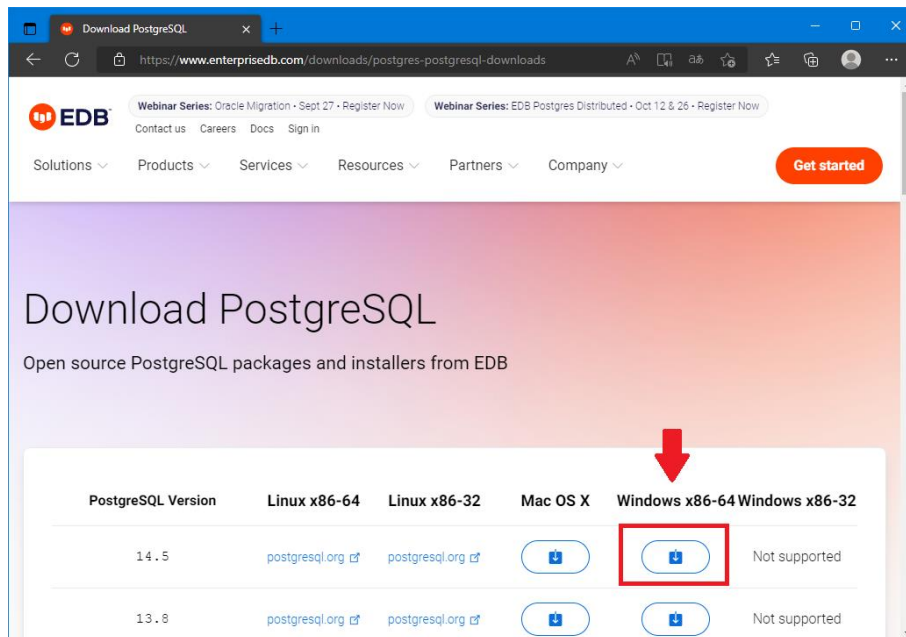
- Será exibido as formas de download para o Windows, clique na opção “Download the installer”



- Caso apareça uma janela de cookies apenas clique em “accept all” e em seguida em “fechar”



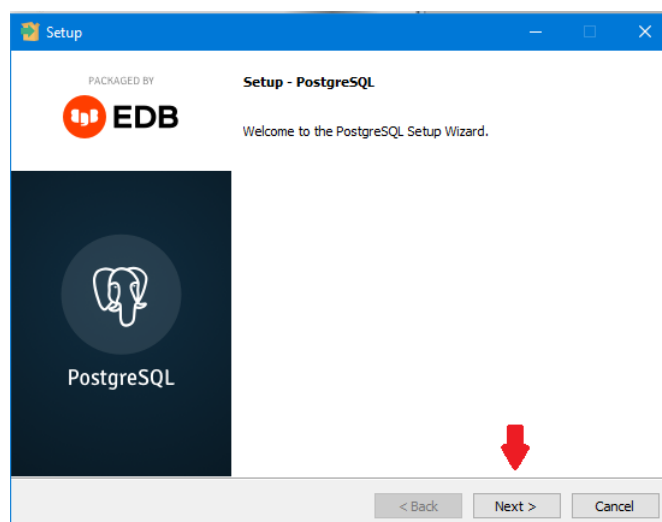
- Ao clicar na opção acima o site será redirecionado para o site da EDB, desenvolvedora do PostgreSQL, onde poderá selecionar o Download do instalador de acordo com a versão desejada. (Durante o curso usaremos a versão 14.5)



- Agora é só aguardar o download ser finalizado

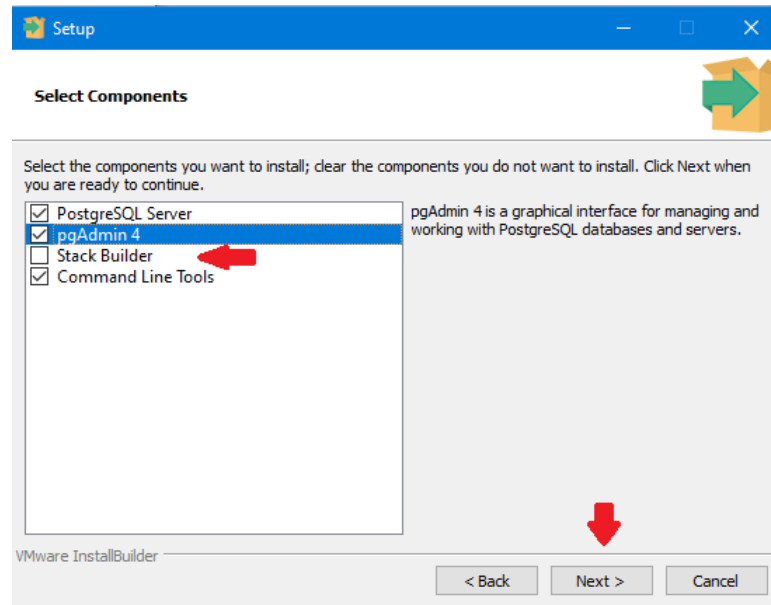
Instalação:

- Abra o arquivo postgresql-14.5-1-windows-x64.exe baixado no passo anterior e em seguida clique em "Next"

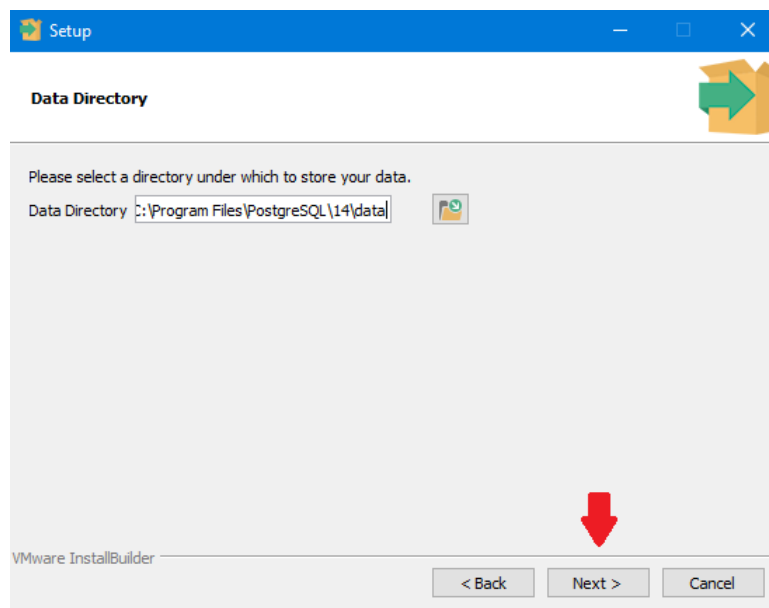


**Icaro Daflon**  
**JAVA MOD II**

- Irá aparecer a janela para selecionar os componentes a serem instalados. Desmarque a opção “Stack Builder” demonstrada na imagem e em seguida clique em “Next”

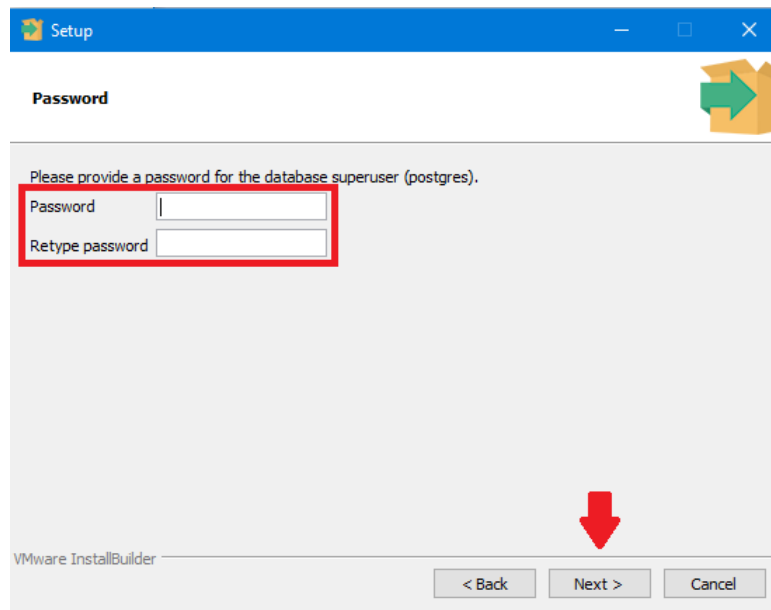


- A seguir será exibida a tela para selecionar o diretório onde serão salvos as informações dos bancos de dados criados, caso queira alterá-lo basta selecionar o diretório que preferir. Neste caso não iremos alterá-lo, portanto, clique novamente em “Next”

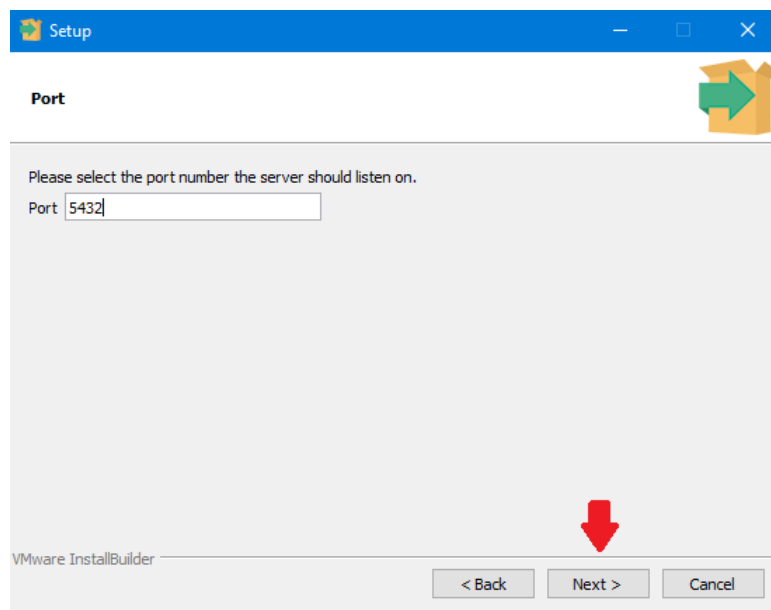


- A seguir será necessário definir uma senha para o usuário padrão “postgres” que é o administrador do pgAdmin. É importante definir uma senha segura para assegurar que somente o administrador do SGDB tenha acesso como um super user. Neste caso definiremos a senha como “123456” após clique em “Next”

## Icaro Daflon JAVA MOD II

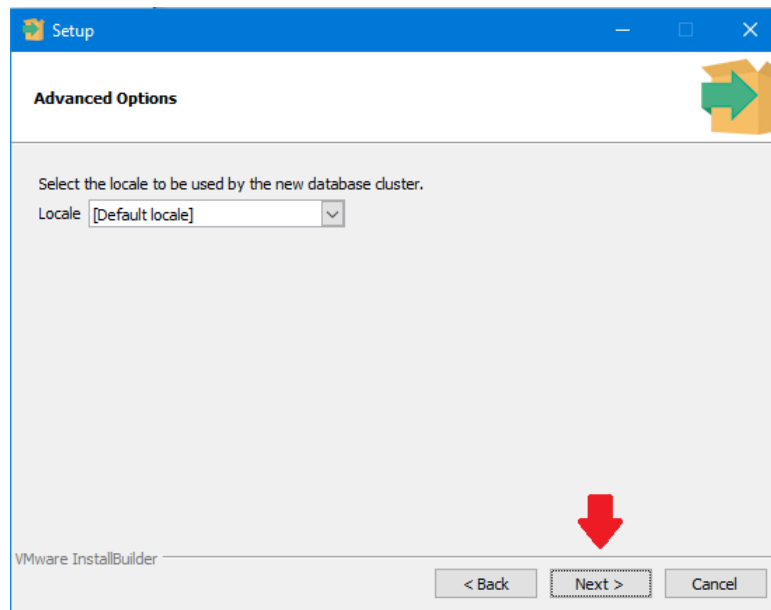


- Na próxima tela será configurada a porta para o acesso ao banco de dados. A porta padrão é a 5432 exibida na tela e é interessante não a alterar a menos que outro programa a esteja utilizando, assim clique em “Next”

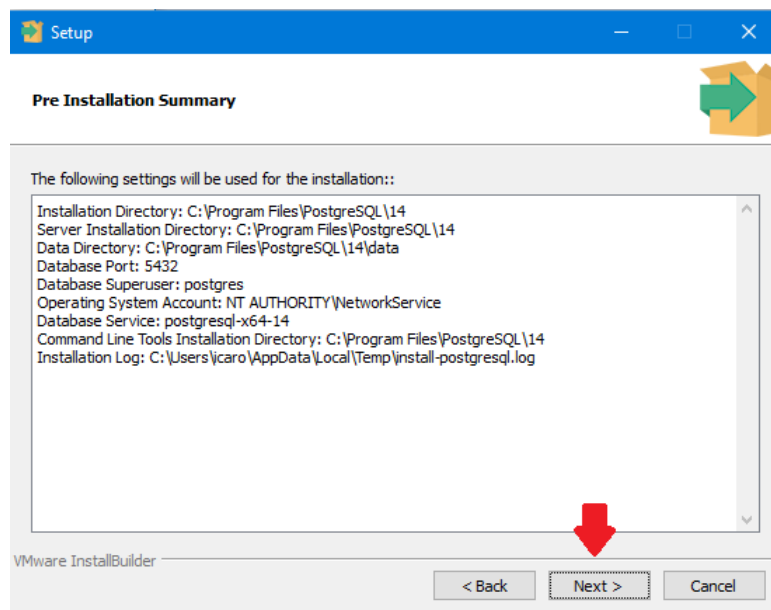


- A seguir será configurada o local para definição da linguagem utilizada pelo PostgreSQL. É recomendado deixá-la como padrão “Default Locale” para não haver conflito entre o sistema utilizado e o PostgreSQL. Portanto deixe selecionado a opção “Default Locale” e então clique novamente em “Next”

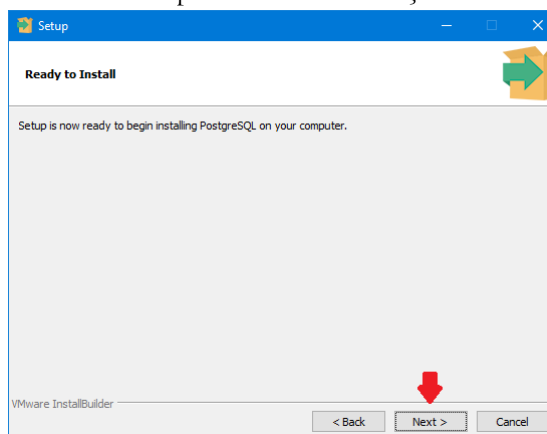
## Icaro Daflon JAVA MOD II



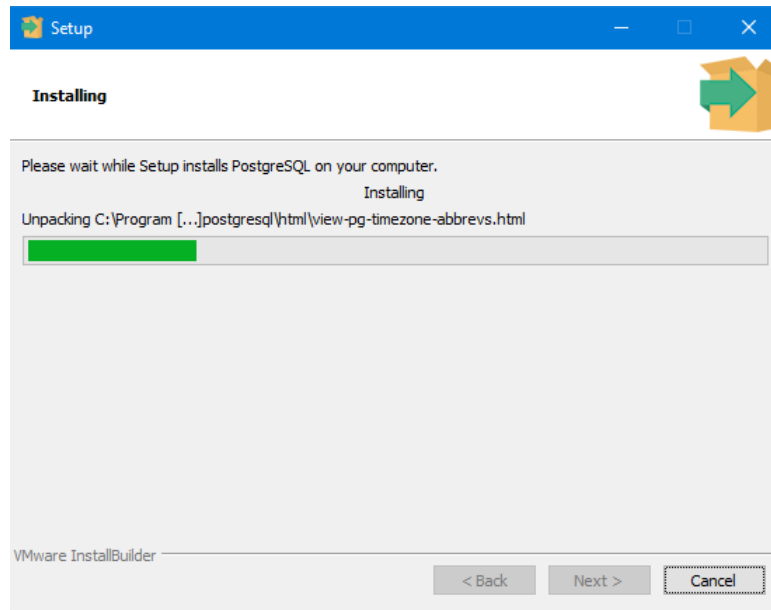
- Na próxima tela exibe um resumo do conteúdo a ser instalado clique em “Next”



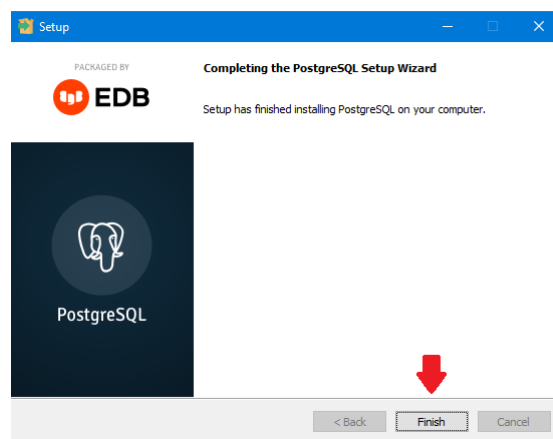
- Na próxima tela clique novamente em “Next” para iniciar a instalação



- Aguarde a instalação terminar



- Por fim clique em "Finish" para finalizar a instalação

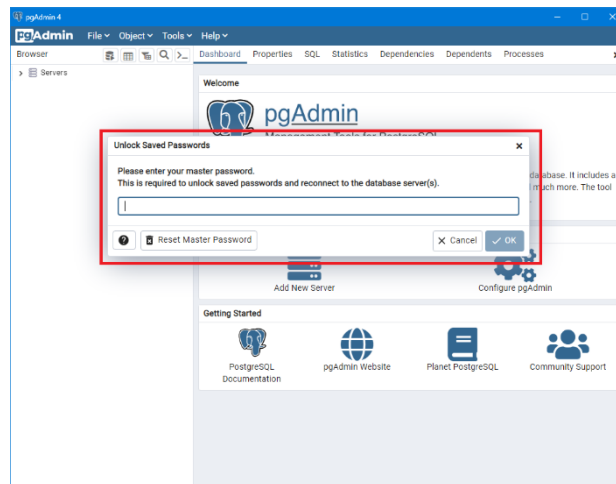


## AULA 2

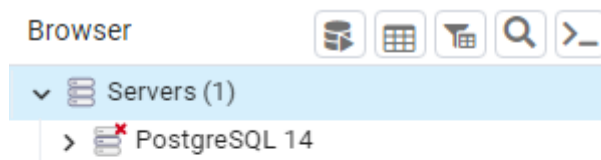
Abertura e configuração inicial do pgAdmin (Páginas 21 a 23):

- Inicie o pgAdmin, para isso clique no menu iniciar e digite pgAdmin 4
- Na primeira inicialização o pgAdmin irá pedir para digitar a "Master Password" que é a senha mestre do programa para poder ter acesso aos servidores. Assim como a senha configurada na instalação é recomendado utilizar de uma senha segura afim de proteger os dados, nesta aula iremos utilizar novamente a senha "123456"

## Icaro Daflon JAVA MOD II



Ao inserir a Master Password o pgAdmin está pronto para ser utilizado, é importante frisar no lado esquerdo tem-se o browser, nele se encontram os servidores que conterão os bancos de dados a serem criados durante o curso. Nele por enquanto há apenas uma opção de conexão “PostgreSQL 14” que foi o servidor localhost (Salvo no computador local) configurado durante a instalação anterior.



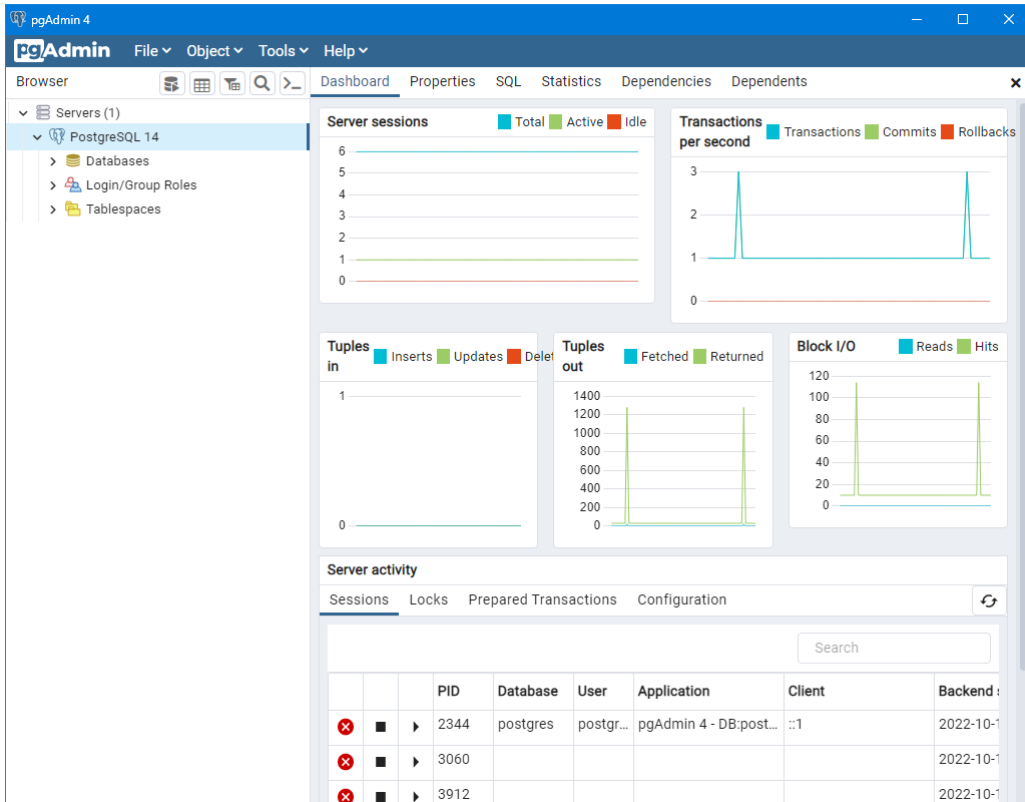
- Para conectar-se ao servidor dê 2 cliques no “PostgreSQL 14”. Aparecerá uma janela pedindo a senha do usuário administrador postgres para que haja a conexão. Está é a senha configurada durante a instalação “123456”. Basta digitar a senha e então teclar “ENTER” ou clicar em “OK”. (Caso a senha não seja essa verifique com seu instrutor)



A conexão então será realizada e a tela se encontrará da seguinte maneira:

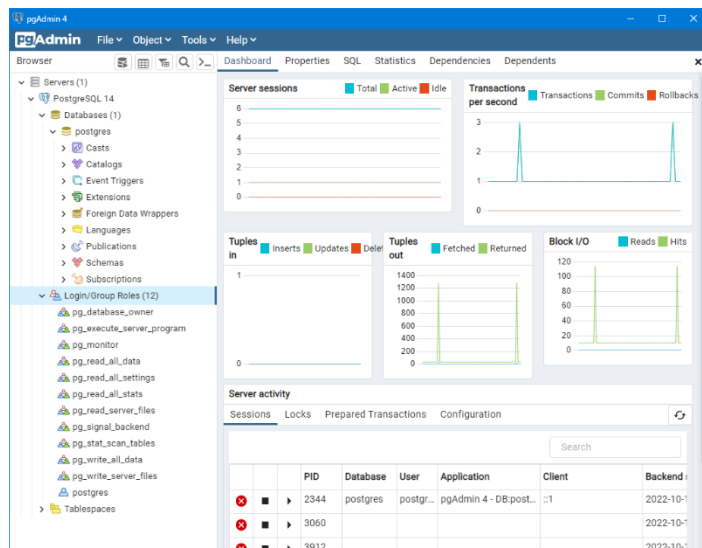


**Icaro Daflon**  
**JAVA MOD II**



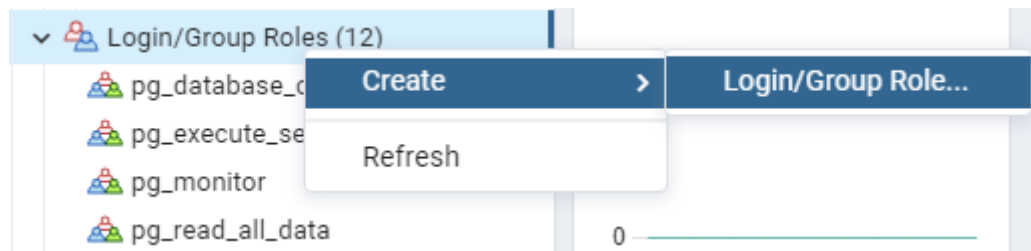
Criação do Banco de dados (Páginas 27 a 40):

- Dê um clique no botão (+) dos itens “Databases” e “Login/Group Roles”. A tela deverá ficar assim:

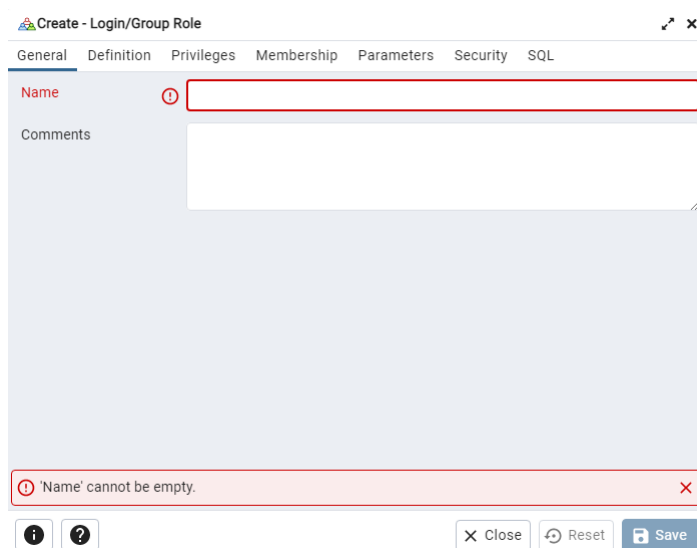


É possível observar que há um Banco de dados e um usuário de nome postgres além de diversos group roles padrões do sistema, criados no momento da instalação do postgres. Para termos a independência dos dados e não misturar os dados públicos do sistema, criaremos então um Login e Database pessoais.

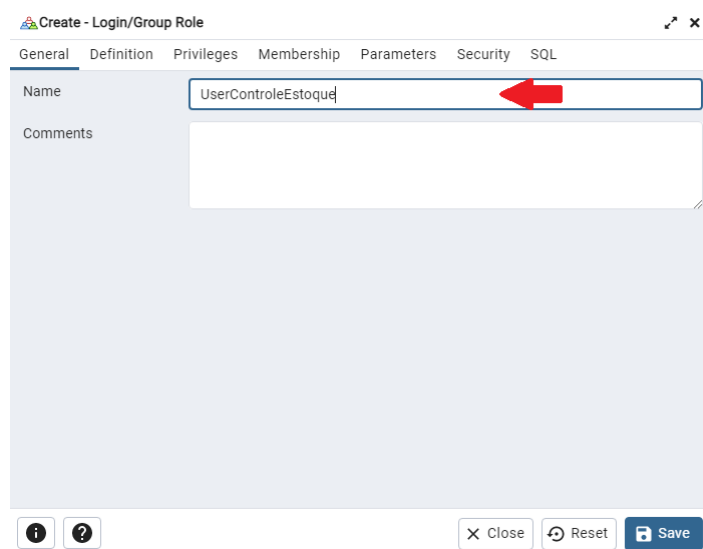
- Para criarmos o Login Role, clique com o botão direito na aba “Login/Group Roles”, posicione o curso sobre a opção “Create” e selecione a opção “Login/Group Role...”



Surgirá a seguinte janela:

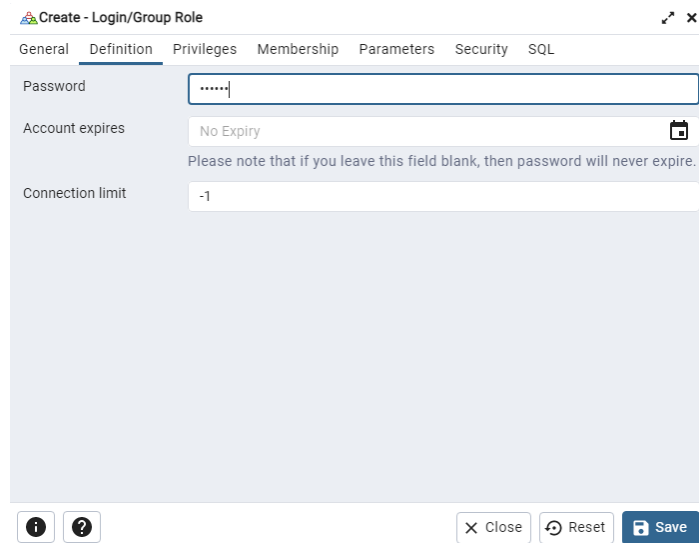


- Na aba “General” no campo “Name” você definirá o nome do usuário, para esta aula digite “UserControleEstoque”



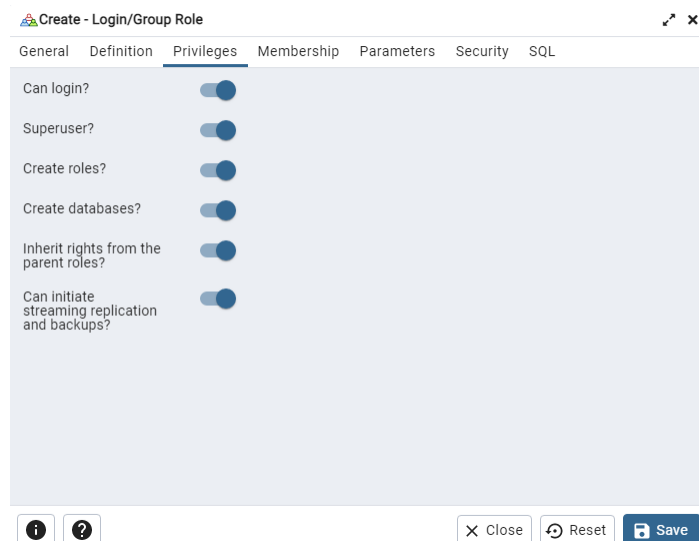
## Icaro Daflon JAVA MOD II

- Na aba “Definition” iremos definir uma senha para o nosso usuário, para esta aula utilize a senha “123456”, mas lembre-se de sempre utilizar uma senha forte para assegurar a segurança de seus dados.



The screenshot shows the 'Create - Login/Group Role' dialog box with the 'Definition' tab selected. The 'Password' field contains '123456'. The 'Account expires' field is set to 'No Expiry'. The 'Connection limit' field is set to '-1'. The dialog box has tabs for General, Definition, Privileges, Membership, Parameters, Security, and SQL. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

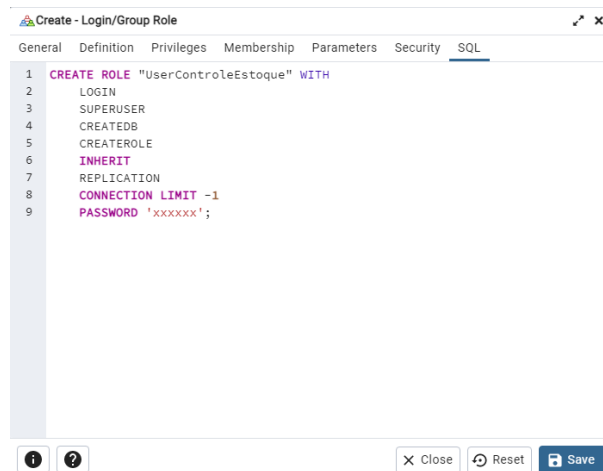
- Na janela Privileges, marque todas as opções disponíveis para que nosso usuário possa realizar todas as ações possíveis dentro do pgAdmin



The screenshot shows the 'Create - Login/Group Role' dialog box with the 'Privileges' tab selected. All privilege options are checked: 'Can login?', 'Superuser?', 'Create roles?', 'Create databases?', 'Inherit rights from the parent roles?', and 'Can initiate streaming replication and backups?'. The dialog box has tabs for General, Definition, Privileges, Membership, Parameters, Security, and SQL. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

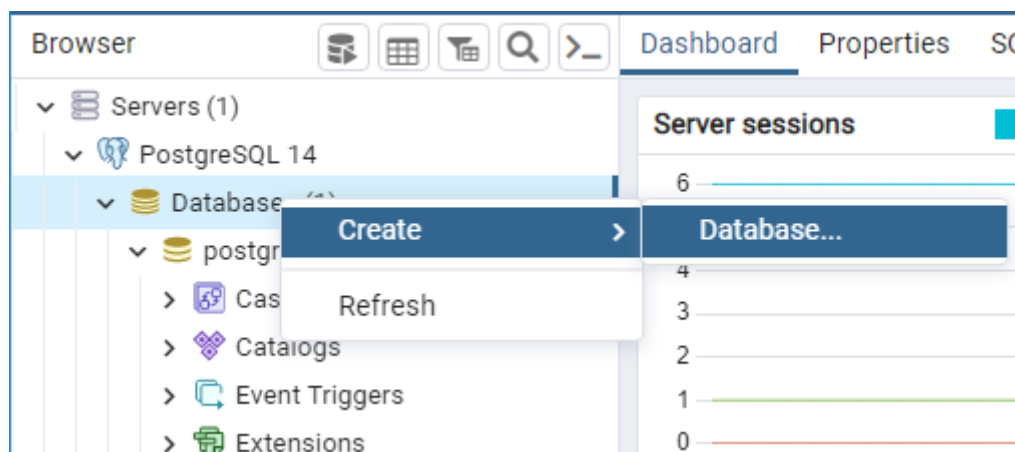
- Nosso usuário está completamente configurado, nas abas Membership, Parameters, Security e SQL não serão feitas nenhuma alteração. Porém vale ressaltar na aba “SQL”, ao clicar nessa aba é possível visualizar o código SQL gerado na criação do usuário. Desmarcando a opção “read only” é possível realizar alterações no código SQL caso seja necessário. Porém não é recomendado a menos que saiba exatamente as alterações que estará fazendo.

## Icaro Daflon JAVA MOD II



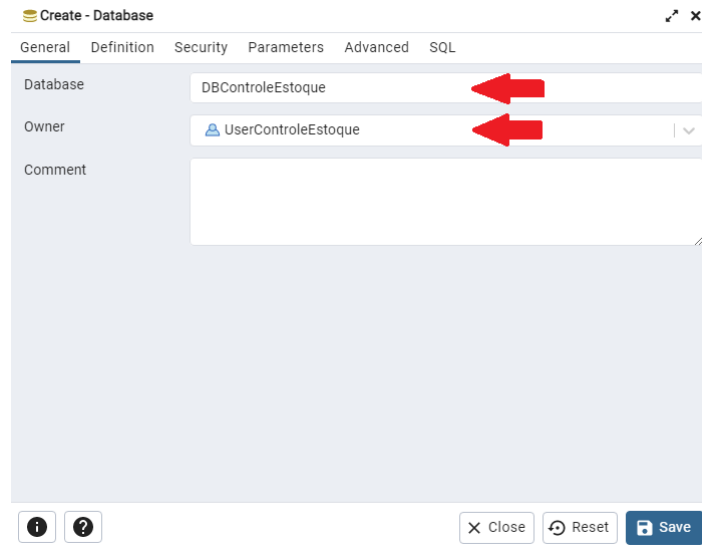
Como nosso código está de acordo com o esperado, clique em “Save” para finalizar a criação do nosso usuário. Observe que abaixo de “Login/Group Roles” no Browser apareceu o Usuário recém criado.

- Agora iremos prosseguir para a criação do Banco de Dados e associar ele ao usuário que acabamos de criar, para isso clique com o botão direito sobre “Databases” no Browser, posicione o cursor sobre a opção “Create...” e clique sobre a opção “Databases”

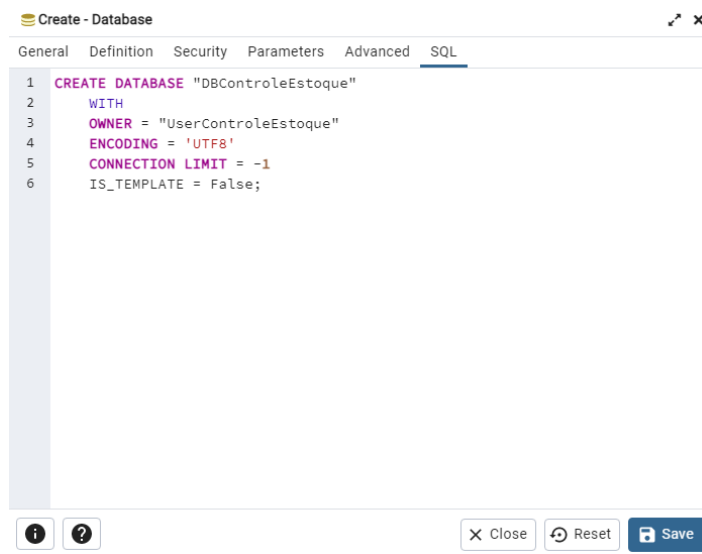


- Na janela que surge, na aba “General” digite no campo “Name” o nome do Banco de dados, neste caso “DBControleEstoque”, e no campo “Owner” selecione o Usuário responsável por este banco de dados, neste caso “UserControleEstoque que criamos anteriormente.

## Icaro Daflon JAVA MOD II

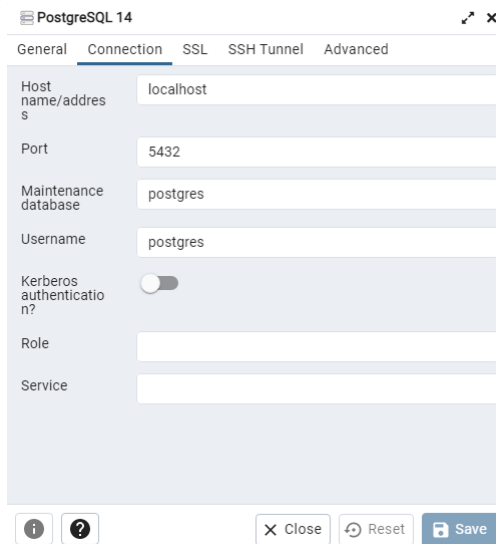


- As abas Definition, Security, Parameters, Advanced e SQL não precisarão ser alteradas já que possuem configurações avançadas que não serão necessárias neste Banco de Dados. Navegue pelas abas para visualizar as opções disponíveis, inclusive na aba SQL que assim como na janela de criação do Usuário apresentará o código SQL gerado na criação do Banco de Dados como é mostrado na imagem abaixo, ao final clique em “Save” para criar o banco de dados:

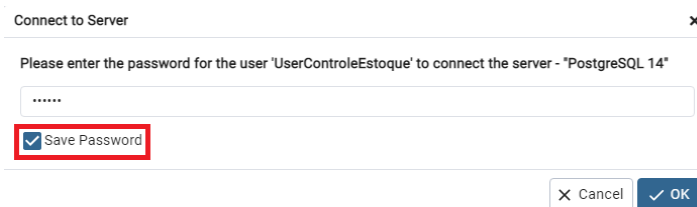


- Operações finalizadas, agora iremos configurar o servidor a fazer login utilizando o usuário “UserControleEstoque” que criamos, para isso iremos nos desconectar do servidor PostgreSQL 14 clicando com o botão direito sobre o servidor e selecionando a opção “Disconnect from Server”. Em seguida clique em “Yes” na janela que surge para desconectar do servidor.
- Com o servidor desconectado iremos configurar o login pelo usuário que criamos, para isso clique novamente sobre o servidor “PostgreSQL 14” com o botão direito e selecione a opção “Properties...” e na janela que surge abra a aba “Connection”

## Icaro Daflon JAVA MOD II



- No campo “Username” substitua a opção “postgres” pelo usuário que criamos “UserControleEstoque” e clique em “Save”
- Com isso basta efetuarmos novamente o login no servidor “PostgreSQL 14”, para isso dê um duplo clique sobre o servidor e agora será pedido a senha novamente, porém para o usuário “UserControleEstoque”
- Basta digitar a senha e é recomendado marcar a opção na caixa “Save Password” para que posteriormente não seja necessário digitar a senha novamente e depois clique em “OK”.



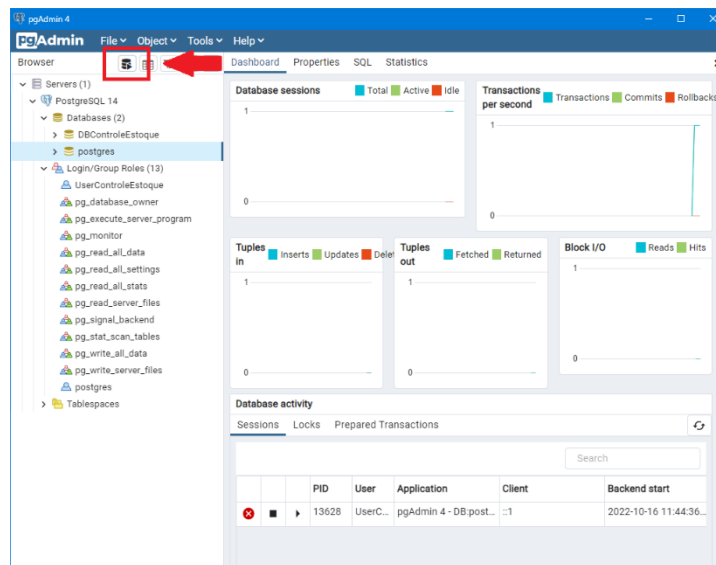
## AULA 3

Criando tabelas páginas 45 a 49

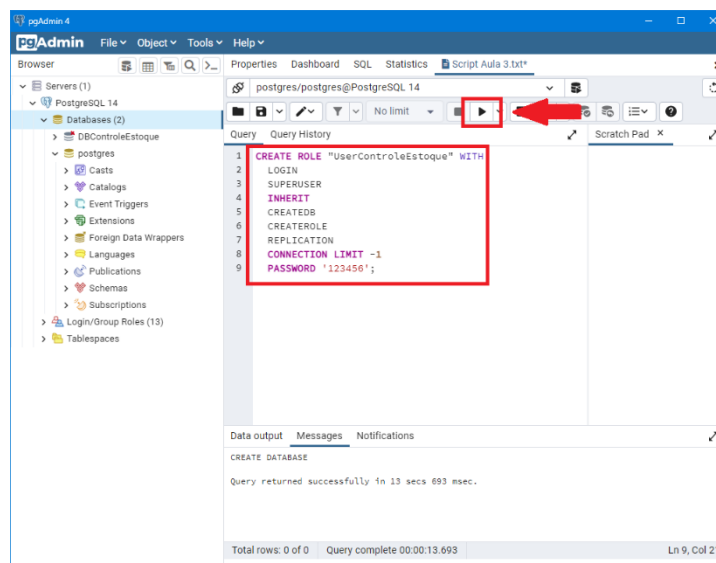
- Primeiramente abra o pgAdmin 4 e insira a “Master Password”, 123456, caso não seja essa solicite a seu instrutor.
- Conecte-se ao servidor, caso seja solicitada uma senha, digite “123456” ou peça ao seu instrutor.
- Certifique-se que o Banco de Dados “DBControleEstoque” e o Usuário “UserControleEstoque” estão criados, caso não estejam, solicite a seu instrutor o script para criação deles.

## Icaro Daflon JAVA MOD II

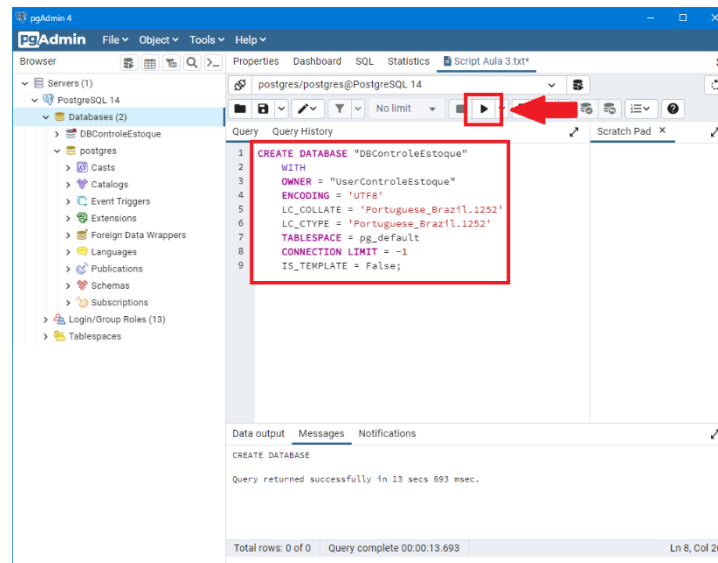
- Para executar o script basta abrir o “Query tool”, selecione o banco de dados “postgres” e clique no primeiro botão ao lado de “Browser”



- No painel de query, basta copiar e colar o código de cada script e separadamente executá-los clicando no botão “execute/refresh” ou apenas aperte F5 como mostra as imagens. (Lembre-se que primeiro deve ser executado o “Script Login Role” e depois o “Script Database”)



## Icaro Daflon JAVA MOD II



- Espere alguns segundos para que o banco de dados seja criado, as vezes é necessário clicar na seta de “Databases” para fechar a aba e em seguida clicar sobre ela de novo para abrir e então aparecer o banco criado.
- Agora iremos criar 2 tabelas utilizando SQL, portanto seguindo os passos anteriores, com o banco de dados “DBControleEstoque” selecionado clique sobre o botão “Query Tool” para abrir o painel onde iremos escrever os códigos em SQL.
- A primeira tabela a ser criada será a tabela usuário, portanto, no painel Query Tool digite o código SQL abaixo:

```
Query  Query History
1  CREATE TABLE usuario
2  (id INT PRIMARY KEY,
3  nome VARCHAR(256) NOT NULL,
4  idade INT);
```

- Agora clique em “execute/refresh” ou apenas aperte F5 para executar a query. Deve aparecer a seguinte mensagem:

```
Data output  Messages  Notifications
CREATE TABLE
Query returned successfully in 761 msec.
```

- Com a tabela usuário criada, crie mais uma tabela, agora com nome de veículo. Digite o código abaixo:



Query	Query History
1	CREATE TABLE veiculo
2	(id INT PRIMARY KEY,
3	nome VARCHAR(256) NOT NULL,
4	tipo VARCHAR(64) NOT NULL,
5	marca VARCHAR(64) NOT NULL,
6	modelo VARCHAR(64));

Inserindo dados, páginas 49 e 50

- Agora você irá inserir alguns dados, portanto no painel Query Tool, digite:

Query	Query History
1	INSERT INTO usuario
2	(id, nome, idade)
3	VALUES
4	(1, 'José', 28);

- Execute o código. Caso não haja nenhum erro o retorno será o seguinte:

Data output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 413 msec.		

- Agora, insira mais quatro registros, dois referentes à tabela usuário e dois à tabela veiculo, de acordo com as especificações abaixo:

Query	Query History
1	INSERT INTO usuario
2	(id, nome, idade)
3	VALUES
4	(2, 'João', 60);
5	
6	INSERT INTO usuario
7	(id, nome, idade)
8	VALUES
9	(3, 'Maria', 18);
10	
11	INSERT INTO veiculo
12	(id, nome, tipo, marca, modelo)
13	VALUES
14	(1, 'Stilo', 'Carro', 'Fiat', '2011');
15	
16	INSERT INTO veiculo
17	(id, nome, tipo, marca, modelo)
18	VALUES
19	(2, 'CB300R', 'Motocicleta', 'Honda', '2013');

- Execute esse código clicando no botão “execute/refresh” ou apertando o botão F5.

## Recuperando dados, Páginas 51 e 52

- Execute o SQL abaixo:

```
Query  Query History
1  SELECT * FROM usuario;
```

Após executar o comando, observe que na janela “Data Output” aparecem os três usuários que foram inseridos na tabela usuario, como abaixo:

Data output  Messages  Notifications			
	id [PK] integer	nome character varying (256)	idade integer
1	1	José	28
2	2	João	60
3	3	Maria	18

Agora iremos localizar somente o usuário que possuir o nome Maria.

- Para trazer os usuários que tem o nome Maria, execute o SQL abaixo:

```
Query  Query History
1  SELECT * FROM usuario WHERE nome='Maria';
```

Observe o resultado na janela Data Output:

Data output  Messages  Notifications			
	id [PK] integer	nome character varying (256)	idade integer
1	3	Maria	18

Agora iremos realizar mais uma consulta que somente buscará os usuários com idade maior ou igual a 21 anos.

- Para isso, execute o SQL abaixo para obter o resultado:

```
Query  Query History
1  SELECT * FROM usuario WHERE idade>=21;
```

O resultado esperado será esse:

Data output  Messages  Notifications			
	id [PK] integer	nome character varying (256)	idade integer
1	1	José	28
2	2	João	60

Atualizando dados Página 53:

- No query tool digite o seguinte código para atualizarmos nossa base de dados:

```
Query  Query History
1  UPDATE usuario SET nome='Benedito' WHERE id=1;
```

Observe que na janela “Messages” apenas aparece uma mensagem que a query foi executada com sucesso. Para verificar se realmente a atualização foi efetuada com sucesso, você deverá fazer uma busca de usuário pelo id.

- Para isso, execute o código abaixo:

```
Query  Query History
1  SELECT * FROM usuario WHERE id=1;
```

O retorno da instrução deverá ser esse:

Data output  Messages  Notifications			
	id [PK] integer	nome character varying (256)	idade integer
1	1	Benedito	28

deletando dados, páginas 53 e 54:

- Para deletar dados não utilizados da base de dados, digite e execute no Query Tool o seguinte código:

Query Query History

```
1 DELETE FROM usuario WHERE id=1;
```

- O retorno da execução será o mesmo da execução do UPDATE. Caso queira conferir se o dado foi realmente apagado do banco, execute o código abaixo:

Query Query History

```
1 SELECT * FROM usuario WHERE id=1;
```

- O retorno será vazio, como na figura abaixo:

Data output Messages Notifications

id [PK] integer	nome character varying (256)	idade integer
--------------------	---------------------------------	------------------

## Deletando tabelas, Página 54:

- Após aprender a manipular as tabelas iremos apaga-las para deixar o banco de dados limpo para as aplicações futuras. Para isso execute o código SQL abaixo:

Query Query History

```
1 DROP TABLE usuario;
```

- A tabela usuario foi apagada. Faça o mesmo com a tabela veiculo.
- Ao final, feche a janela Query e em seguida, feche o pgAdmin 4.

# AULA 4

## Criando a tabela produto, Páginas 58 e 59:

- Primeiro, abra o pgAdmin 4 e conecte-se ao servidor PostgreSQL 14.
- Verifique se os bancos de dados DBControleEstoque e o login Role UserControleEstoque já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.

- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBControleEstoque”.
- Após conectar-se iremos criar as tabelas para isso selecione o Banco de Dados “DBControleEstoque” e clique no botão “Query Tool” para abrir o editor.
- Na janela Query Tool agora execute o código abaixo:

```
Query  Query History
1  CREATE TABLE produtos
2  (
3      id INT PRIMARY KEY,
4      nome VARCHAR(256) NOT NULL,
5      descricao VARCHAR(256),
6      quantidade INT NOT NULL
7  )
```

Populando a tabela, Páginas 60 e 61:

- Com a tabela criada agora é necessário preenche-la Isso será feito diretamente no banco de dados pois o sistema somente irá controlar o estoque dos produtos e não inserir novos produtos
- Para isso, no editor SQL do pgAdmin, execute o seguinte código:

```
INSERT INTO produtos VALUES
(1, 'Geladeira', 'Utensílio eletrodoméstico
utilizado na conservação de alimento', 10);

INSERT INTO produtos VALUES
(2, 'Fogão', 'Utensílio culinário usado para
cozinhar, geralmente em panelas ou frigideiras,
e por meio de calor', 32);

INSERT INTO produtos VALUES
(3, 'Máquina de lavar roupas', 'Máquina
projetada para limpeza de roupas', 6);

INSERT INTO produtos VALUES
(4, 'Ar condicionado', 'Responsável pelo
tratamento do ar interior em espeaços fechados', 14);

INSERT INTO produtos VALUES
(5, 'Lava-louças', 'Aparelho eletrodoméstico cuja
finalidade é lavar os apetrechos utilizados
na cozinha', 27);

INSERT INTO produtos VALUES
(6, 'Micro-ondas', 'Aparelho eletrodoméstico que
possibilita a preparação rápida de alimentos
para o consumo humano ou de animais', 10);
```

## Icaro Daflon JAVA MOD II

- Para verificar se tudo ocorreu bem execute a query abaixo e veja o resultado:

Query

Query History









1

SELECT \* FROM produtos;

Data output

Messages

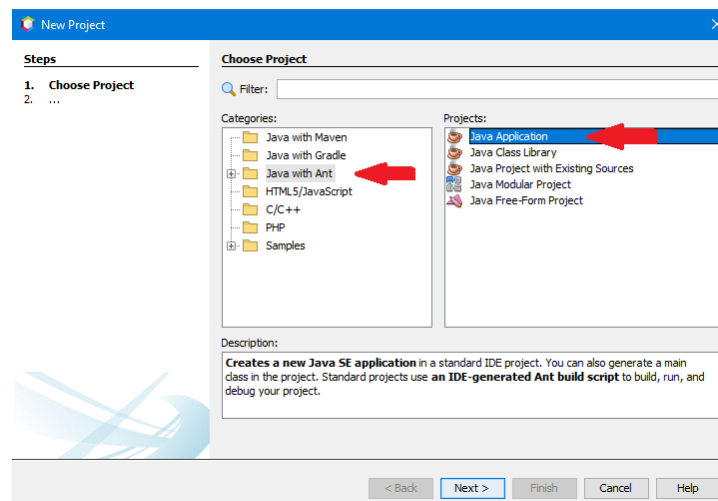
Notifications



	id [PK] integer	nome character varying (256)	descricao character varying (256)	quantidade integer
1	1	Geladeira	Utensílio eletrodomésti...	10
2	2	Fogão	Utenwsílio culinário us...	32
3	3	Máquina de lavar roupas	Máquina projetada par...	6
4	4	Ar condicionado	Responsável pelo trata...	14
5	5	Lava-louças	Aparelho eletrodomést...	27
6	6	Micro-ondas	Aparelho eletrodomést...	10

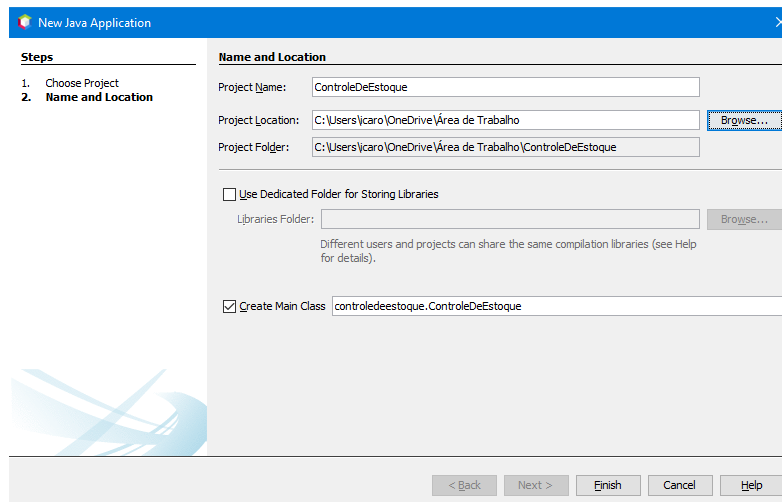
### Controle de Estoque – Projeto, Páginas 61 a 64

- No menu iniciar, digite NetBeans (Neste curso estamos utilizando a versão Apache Netbeans 15 e JDK 18) e com o programa selecionado, pressione a tecla Enter para abri-lo
- Com o programa inicializado, clique no menu “File” e após, em “New Project...”
- Devemos então criar um projeto Java. Para isso, clique sobre a Categoria “Java with Ant” e em seguida, escolha a opção “Java Application”



- Em seguida clique no botão “Next”
- Surgirá a tela para escolha do nome do projeto e do caminho onde o mesmo deve ser salvo. Insira “ControleDeEstoque” no campo “Project Name”, e em “Project Location”, clique no botão “Browse...” e escolha o seu local de gravação.

## Icaro Daflon JAVA MOD II



- Após informar todos os dados, clique em “Finish”

Será apresentado o ambiente de desenvolvimento da ferramenta NetBeans

O projeto criado será listado na aba “Projects”, e por default, possui uma classe principal que recebe o nome do seu projeto com a extensão .java. Essa classe será a a classe responsável por executar os métodos da classe que contém as instruções SQL.

Ela é criada dentro do pacote ControleDeEstoque que também é definido na criação do projeto.

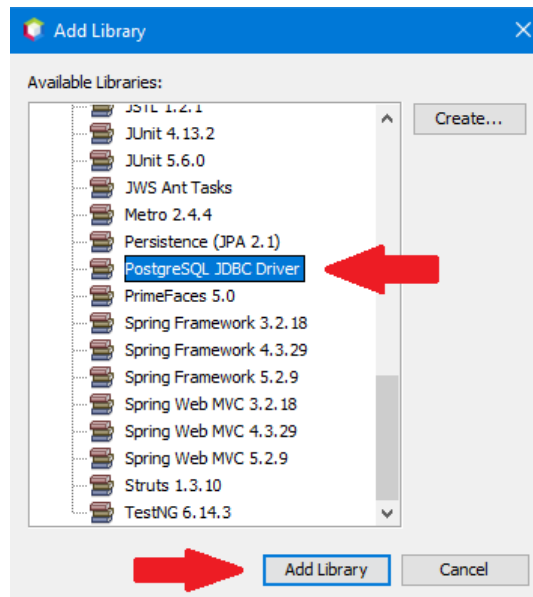
Agora, você irá adicionar a API responsável pela conexão entre o Java e o PostgreSQL.

- Clique com o botão direito na pasta “Libraries” e em seguida, selecione “Add Library...”



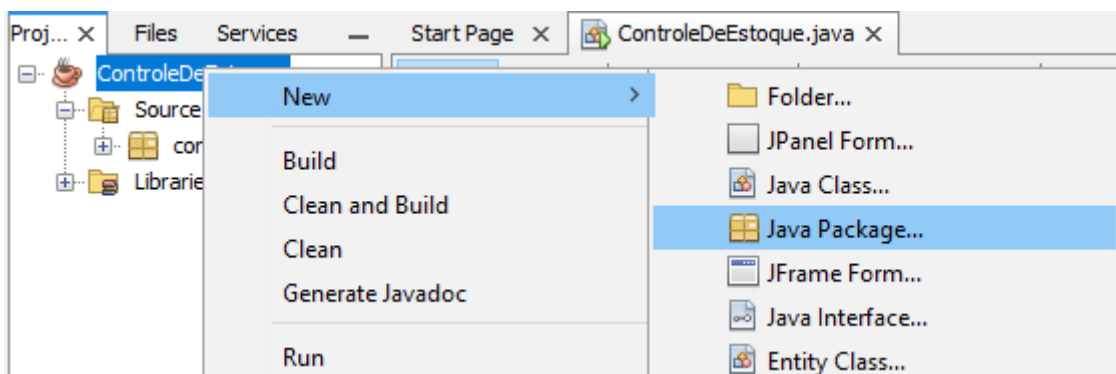
- Na tela que surge, seleciona a biblioteca PostgreSQL JDBC Driver e clique no botão “Add Library”, como abaixo:

Icaro Daflon  
JAVA MOD II



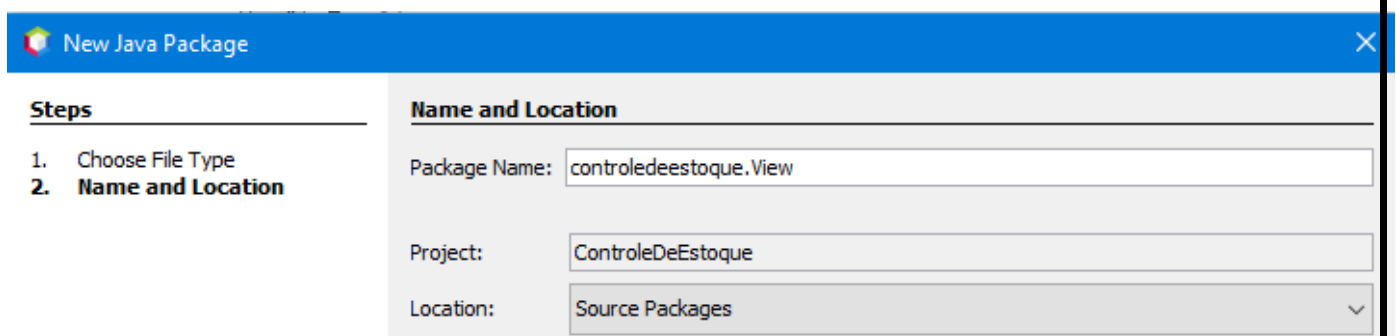
### Controle de Estoque – Pacotes, Páginas 65 a 67

- Clique com o botão direito em cima do projeto “ControleDeEstoque”, selecione “New” e após, escolha a opção “Java Package...”



Na tela que surge, é necessário definir o nome do pacote. Primeiro, crie o pacote que conterá as classes de exibição. Por padrão, o pacote tem o nome do pacote principal (controledeestoque) mais a inclusão da palavra View separados por um ponto (.). Assim, o nome do pacote será controledeestoque.View.

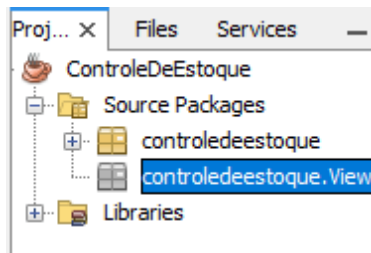
- Então digite esse nome exatamente como está escrito (tudo em minúsculo) no campo “Package Name:” e clique no botão “Finish”



A aba Projetos ficará da seguinte maneira:



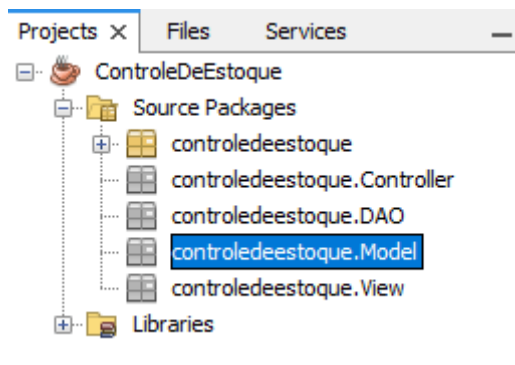
## Icaro Daflon JAVA MOD II



Após a criação do pacote View, você deve criar o restantes dos pacotes de acordo com o que foi explicado anteriormente.

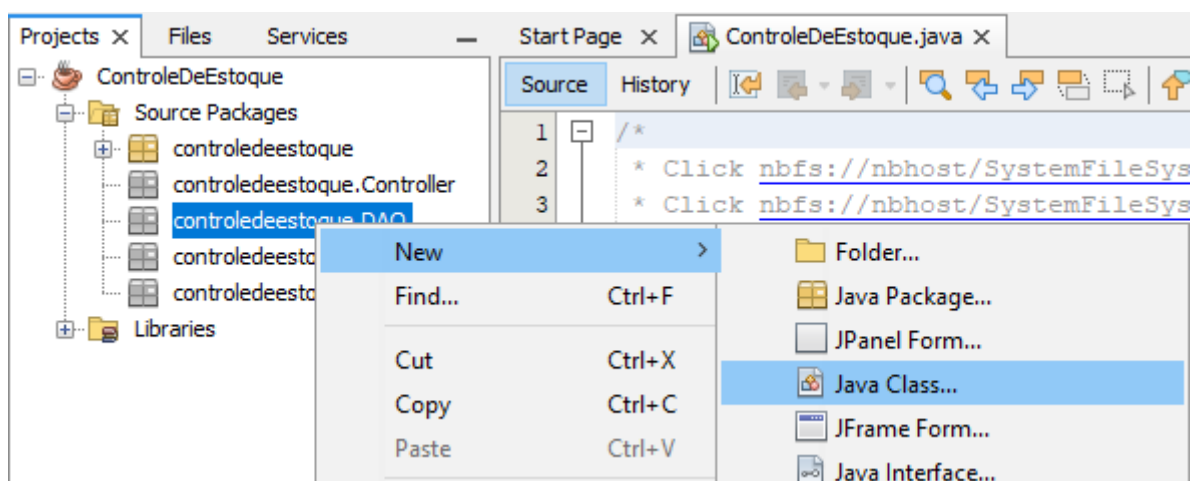
- Crie os pacotes Model, Controller e DAO. Seguindo o padrão, os pacotes terão os nomes controledeestoque.Model, controledeestoque.Controller e controledeestoque.DAO respectivamente

Após a criação dos pacotes, a aba projeto ficará da seguinte forma:



Manipulando o Banco em Java, Páginas 67 a 69:

- Clique com o botão direito do mouse no pacote controledeestoque.DAO, posicione o mouse sobre “New” e após, clique sobre a opção “Java Class...”



- Na janela que surge, atribua ConnectionFactory ao campo da opção “Class Name” e após, clique no botão “Finish”

Se tudo for feito corretamente, a classe `ConnectionFactory.java` será criada dentro do pacote `controldeestoque.DAO` e estará sendo exibida na tela principal do NetBeans, como mostra a imagem abaixo:



- Agora iremos criar as variáveis e métodos necessários para realizar a conexão com o PostgreSQL. Para isso antes iremos importar as bibliotecas necessárias para realizar a conexão com o Banco de Dados, portanto escreva o código como mostrado abaixo:

```
5 package controldeestoque.DAO;
6
7 import java.sql.*;
8
9 public class ConnectionFactory
```

- Crie os objetos String como variáveis globais de classe. Faça como na figura abaixo:

```
1 import java.sql.*;
2
3 public class ConnectionFactory {
4     private static final String URL = "jdbc:postgresql://localhost:5432/DBControleEstoque";
5     private static final String USER = "UserControleEstoque";
6     private static final String PASS = "123456";
7 }
8
9
10
11
12
13
14
```

- Agora, crie o método `getConnection` como na imagem abaixo:

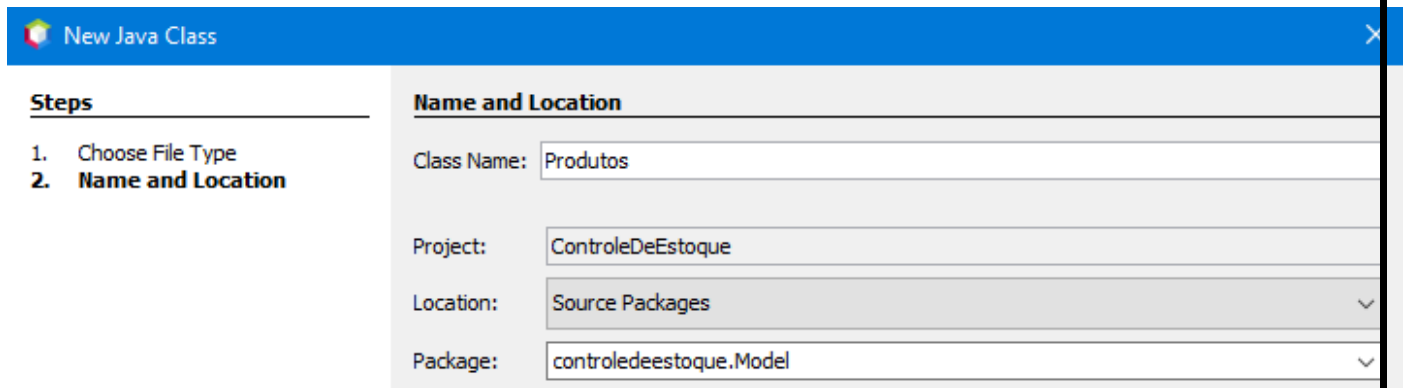
```
15      public static Connection getConnection()
16      {
17          try
18          {
19              System.out.print("Conectado ao banco de dados!");
20              return DriverManager.getConnection(URL, USER, PASS);
21          }
22          catch(SQLException e)
23          {
24              throw new RuntimeException(e);
25          }
26      }
27  }
```

- Nesse momento a classe deve estar da seguinte maneira:

```
5      package controledeestoque.DAO;
6
7      import java.sql.*;
8
9      public class ConnectionFactory
10     {
11         private static final String URL = "jdbc:postgresql://localhost:5432/DBControleEstoque";
12         private static String USER = "UserControleEstoque";
13         private static final String PASS = "123456";
14
15         public static Connection getConnection()
16         {
17             try
18             {
19                 System.out.print("Conectado ao banco de dados!");
20                 return DriverManager.getConnection(URL, USER, PASS);
21             }
22             catch(SQLException e)
23             {
24                 throw new RuntimeException(e);
25             }
26         }
27     }
```

## Controle de Estoque – Classe Model, Páginas 71 a 75

- Clique com o botão direito no pacote controledeestoque.Model, posicione o cursor sobre “New” e clique na opção “Java Class...”
- Na janela que surge, preencha o campo “Class Name” com “Produtos”



**New Java Class**

**Steps**

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

**Name and Location**

Class Name:

Project:

Location:

Package:

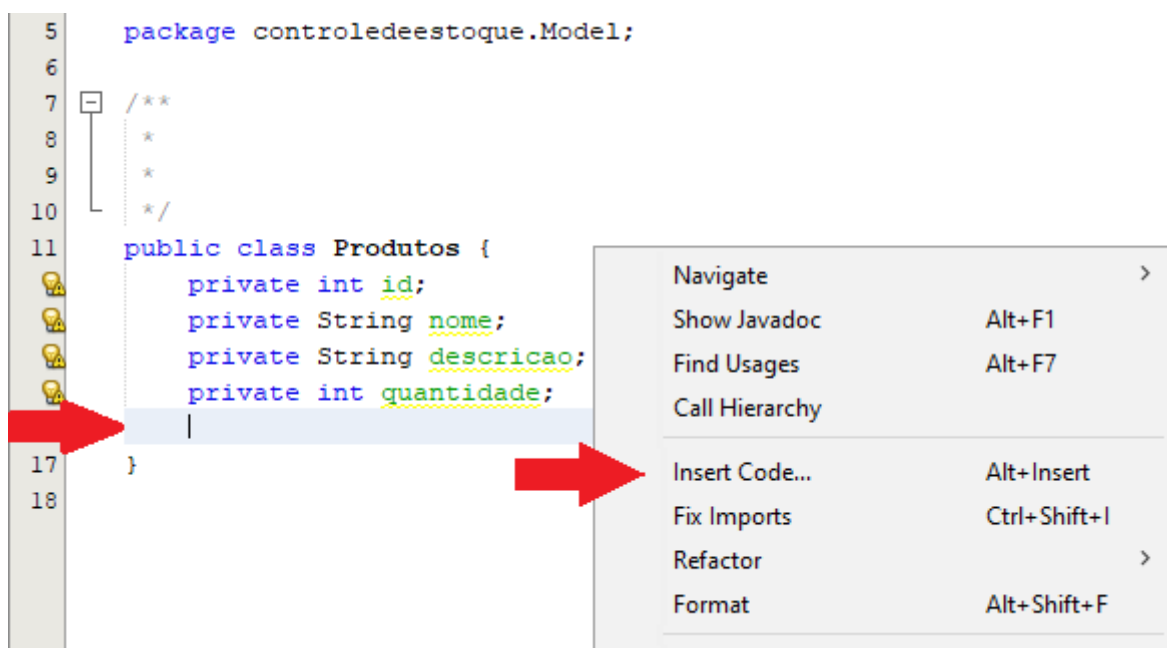
- Após, clique em “Finish”. Observe que a classe `Produtos.java` será criada dentro do pacote `controledestoque.Model`

Vamos agora preencher a classe `Produtos.java` com seus atributos. Os atributos serão: `id`, `nome`, `descricao` e `quantidade`, onde `id` e `quantidade`, são objetos numéricos e o restante são `Strings`.

- Sendo assim, defina os atributos na classe como na figura abaixo:

```
5      package controledestoque.Model;
6
7      /**
8       *
9       *
10     */
11     public class Produtos {
12         private int id;
13         private String nome;
14         private String descricao;
15         private int quantidade;
16     }
```

- Agora iremos criar os getters e setters da classe. Para isso basta clicar com o botão direito abaixo do ultimo objeto criado (`quantidade`) e escolha a opção “Insert Code...” assim como abaixo:



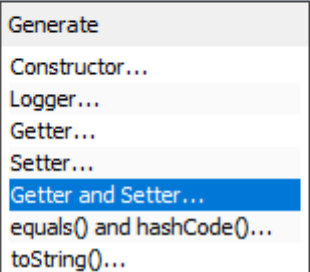
```
5      package controledestoque.Model;
6
7      /**
8       *
9       *
10     */
11     public class Produtos {
12         private int id;
13         private String nome;
14         private String descricao;
15         private int quantidade;
16     }
```

Context Menu:

- Navigate
- Show Javadoc Alt+F1
- Find Usages Alt+F7
- Call Hierarchy
- Insert Code...** Alt+Insert
- Fix Imports Ctrl+Shift+I
- Refactor
- Format Alt+Shift+F

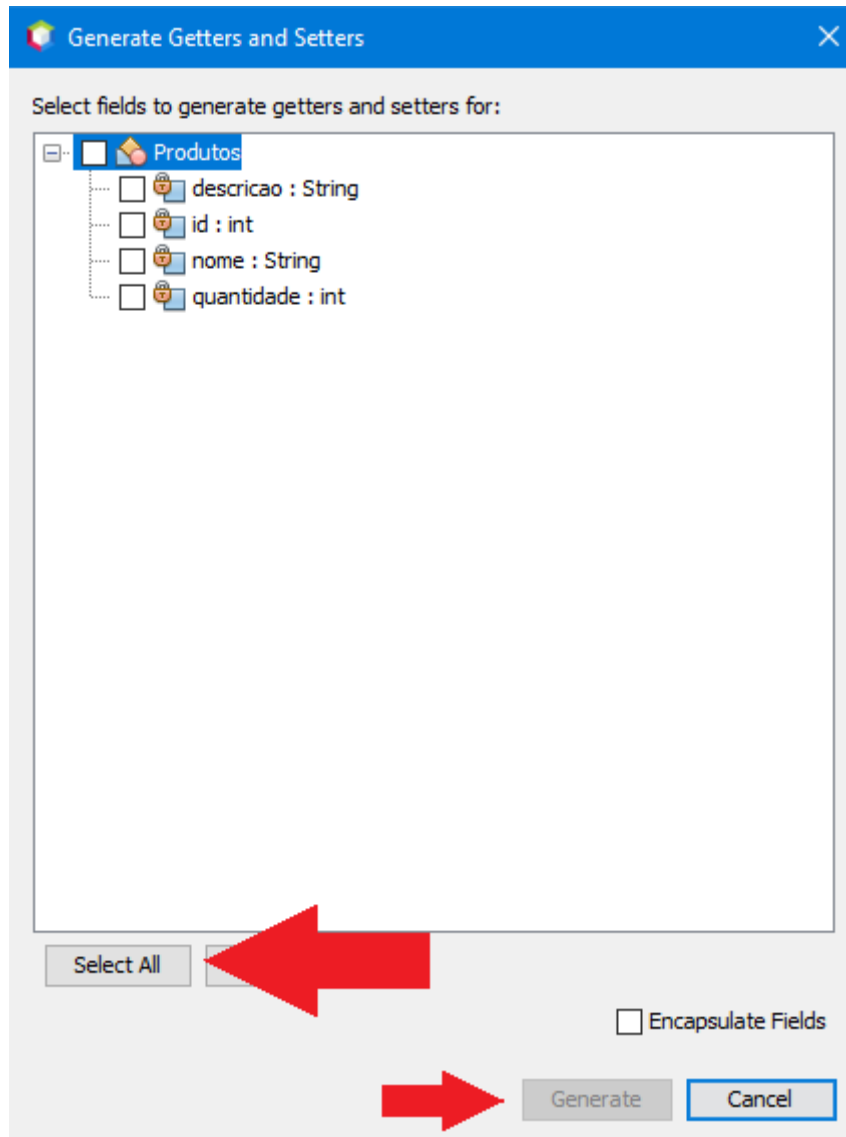
- Nas opções que surgem clique sobre “Getter e Setter...”

```
public class Produtos {  
    private int id;  
    private String nome;  
    private String descricao;  
    private int quantidade;  
}
```



A context menu is displayed over the code, listing several actions: Generate, Constructor..., Logger..., Getter..., Setter..., **Getter and Setter...** (highlighted in blue), equals() and hashCode()..., and toString()...

- Na janela que surge, é possível selecionar os objetos que deseja-se inserir os getters e setters automaticamente. Nesse caso, você deve gera-los para todos. Portanto, clique em “Select All” e clique no botão “Generate”.



Observe que os getters e setters foram criados automaticamente.

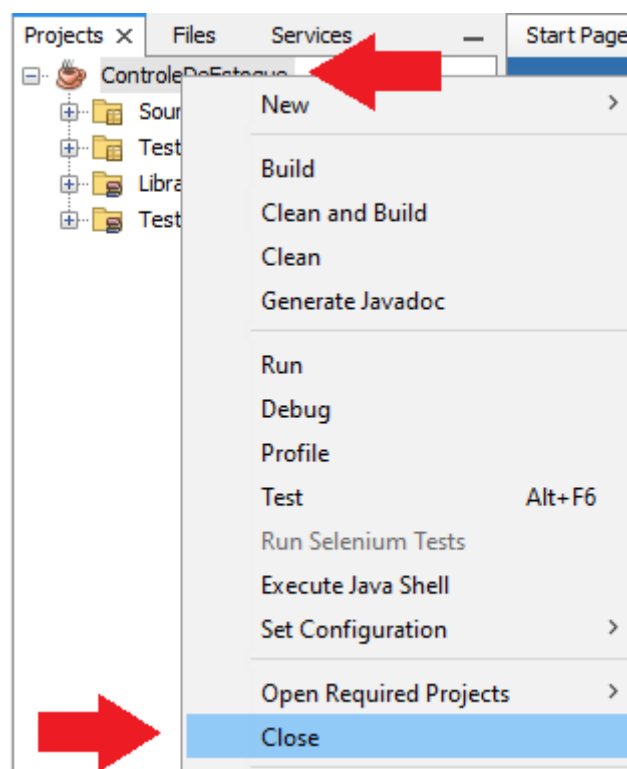
```
11 public class Produtos {
12     private int id;
13     private String nome;
14     private String descricao;
15     private int quantidade;
16
17     public int getId() {
18         return id;
19     }
20
21     public void setId(int id) {
22         this.id = id;
23     }
24
25     public String getNome() {
26         return nome;
27     }
28
29     public void setNome(String nome) {
30         this.nome = nome;
31     }
32
33     public String getDescricao() {
34         return descricao;
35     }
36
37     public void setDescricao(String descricao) {
38         this.descricao = descricao;
39     }
40
41     public int getQuantidade() {
42         return quantidade;
43     }
44
45     public void setQuantidade(int quantidade) {
46         this.quantidade = quantidade;
47     }
48
49 }
```

➤ Com isso finalizamos nossa aula. Salve o projeto e após, feche o NetBeans.

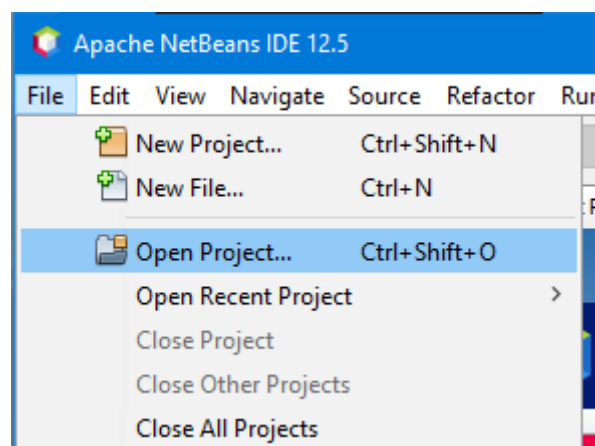
# AULA 5

## Controle de Estoque – Continuação, Páginas 78 a 80

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 4” e copie a pasta do projeto “ControleDeEstoque”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

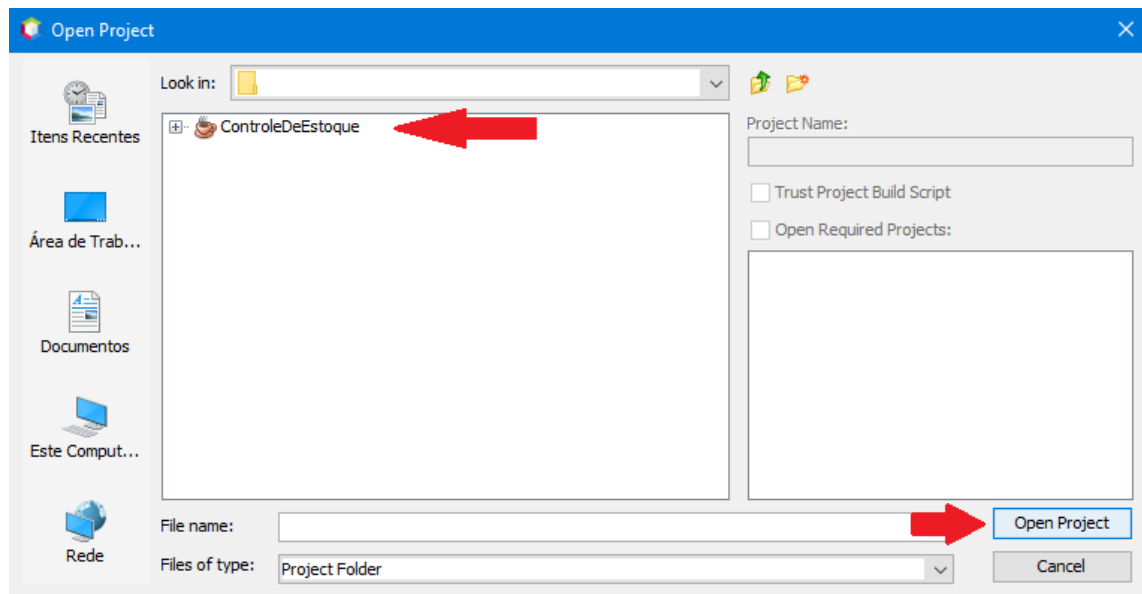


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”

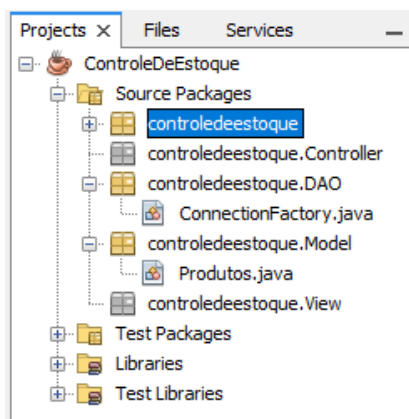


**Icaro Daflon**  
**JAVA MOD II**

- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



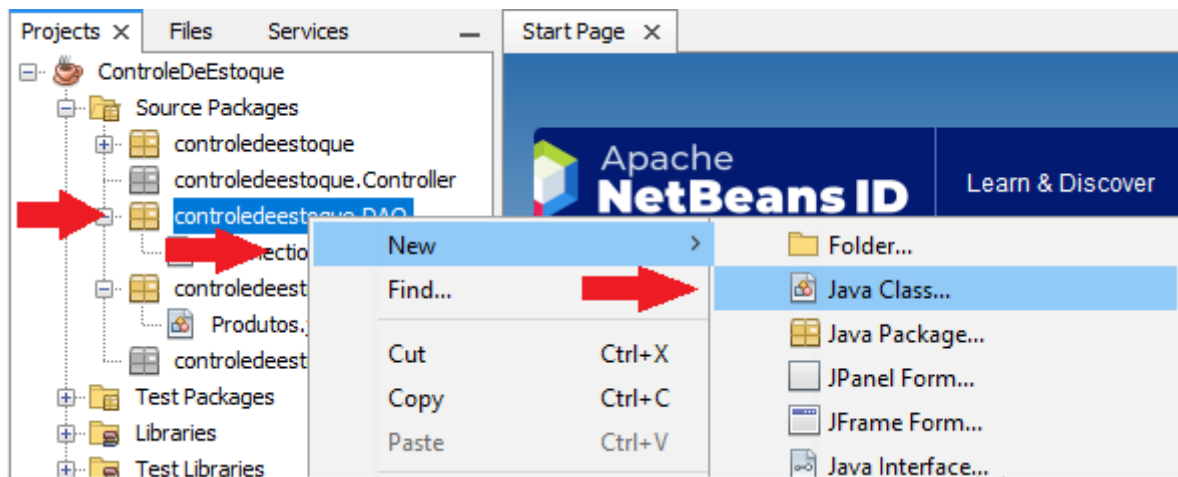
- Com o projeto aberto, expanda os itens da aba Projetos. Observe a imagem abaixo:



Classe ProdutosDAO, Páginas 81 a 90

- Clique com o botão direito do mouse no pacote controldeestoque.DAO, posicione o mouse sobre “New” e após, clique sobre a opção “Java Class...”





- Na janela que surge, atribua ProdutosDAO ao campo da opção “Class Name” e após, clique no botão “Finish”.
- Primeiramente, você irá criar a variável “conn” e o método ProdutosDAO. Deixe sua classe como na imagem abaixo (Lembre-se sempre de realizar os imports necessários):

```
5      package controledeestoque.DAO;
6
7      import java.sql.*;
8
9      public class ProdutosDAO {
10         Connection conn;
11
12         public ProdutosDAO() {
13             conn = ConnectionFactory.getConnection();
14         }
15     }
```

- Crie agora o método atualizar digitando o código abaixo:

```
16     public void atualizar (int id, int qtde) throws SQLException{
17         String sql = "UPDATE produtos SET quantidade = ? WHERE id = ?";
18         PreparedStatement pstmt = conn.prepareStatement(sql);
19         pstmt.setInt(1, qtde);
20         pstmt.setInt(2, id);
21         pstmt.execute();
22         pstmt.close();
23     }
```

Ao final sua classe deve estar desta forma:

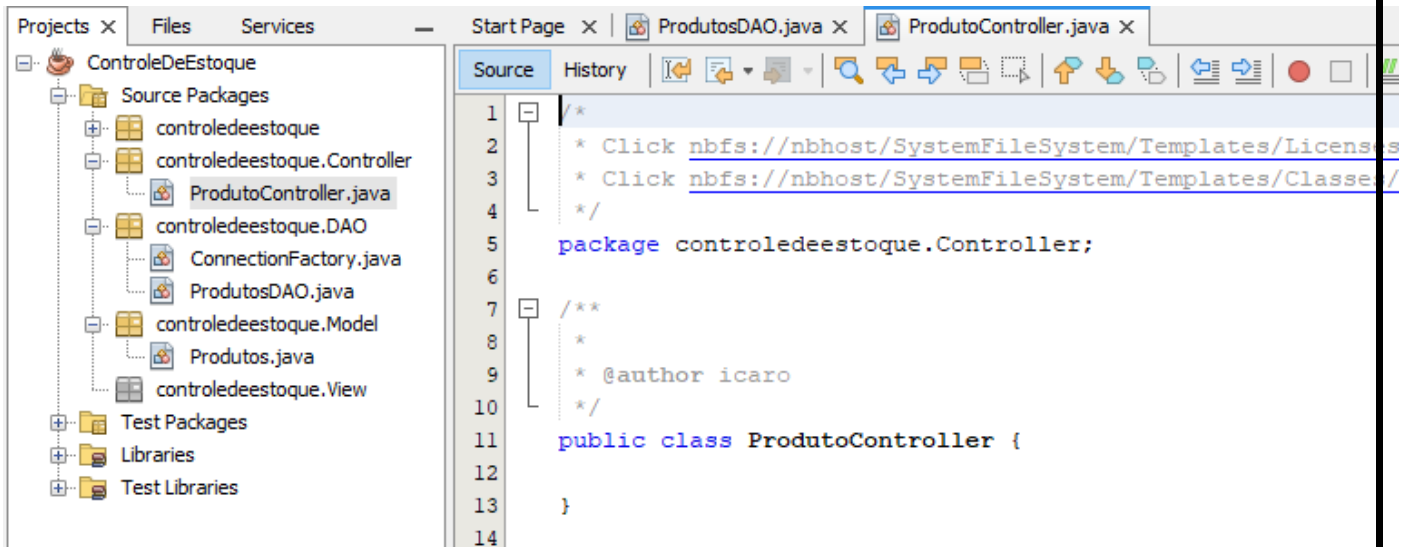
```
5 package controledeestoque.DAO;
6
7 import java.sql.*;
8
9 public class ProdutosDAO {
10     Connection conn;
11
12     public ProdutosDAO() {
13         conn = ConnectionFactory.getConnection();
14     }
15
16     public void atualizar (int id, int qtde) throws SQLException{
17         String sql = "UPDATE produtos SET quantidade = ? WHERE id = ?";
18         PreparedStatement pstmt = conn.prepareStatement(sql);
19         pstmt.setInt(1, qtde);
20         pstmt.setInt(2, id);
21         pstmt.execute();
22         pstmt.close();
23     }
24 }
```

- Crie os outros dois métodos restantes nessa classe de acordo com a imagem:

```
25 public Integer buscarQuantidadeDeEstoque (int id){
26     try{
27         String sql = "SELECT quantidade FROM produtos WHERE id = ?";
28         PreparedStatement pstmt = conn.prepareStatement(sql);
29         pstmt.setInt(1, id);
30         ResultSet rs = pstmt.executeQuery();
31         if(rs.next()){
32             return rs.getInt("quantidade");
33         }
34         pstmt.close();
35         return null;
36     } catch(SQLException e){
37         throw new RuntimeException(e);
38     }
39 }
40
41 public ResultSet buscarTodosProdutos () {
42     try{
43         String sql = "SELECT * FROM produtos";
44         PreparedStatement pstmt = conn.prepareStatement(sql);
45         ResultSet rs = pstmt.executeQuery();
46         return rs;
47     } catch(SQLException e){
48         throw new RuntimeException(e);
49     }
50 }
```

- Agora iremos criar a classe que conterá as regras de negócio do sistema, para isso crie uma nova classe java para o pacote "controledeestoque.Controller" e dê a ela o nome de "ProdutosController".

- O NetBeans deve estar da seguinte forma:

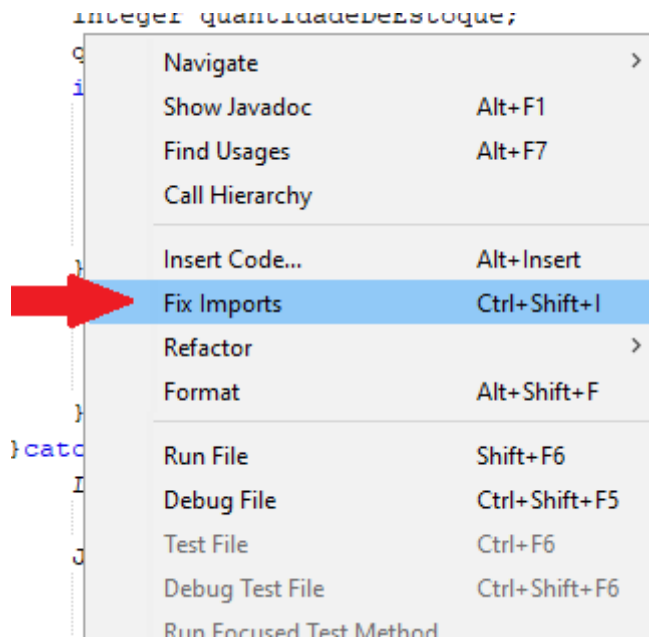


- Com a classe criada, primeiro crie o método Inserir, conforme a imagem abaixo:

```
20 public void Inserir(int id, int qtde){
21     try{
22         ProdutosDAO produtosDAO = new ProdutosDAO();
23         Integer quantidadeDeEstoque;
24         quantidadeDeEstoque = produtosDAO.buscarQuantidadeDeEstoque(id);
25         if(quantidadeDeEstoque != null){
26             produtosDAO.atualizar(id, qtde + quantidadeDeEstoque);
27             JOptionPane.showMessageDialog(null,
28                 "Estoque atualizado", "Operação Confirmada",
29                 JOptionPane.INFORMATION_MESSAGE);
30         }else{
31             JOptionPane.showMessageDialog(null,
32                 "Código Inválido", "Operação Incorreta",
33                 JOptionPane.ERROR_MESSAGE);
34         }
35     }catch (SQLException e){
36         System.getLogger(InserirProdutosView.class.getName())
37             .log(Level.ERROR, e);
38         JOptionPane.showMessageDialog(null,
39             "Código Inválido", "Operação Incorreta",
40             JOptionPane.ERROR_MESSAGE);
41     }
```

Para fazer os imports mais rapidamente desta vez iremos clicar com o botão direito sobre o nosso código e após, iremos clicar sobre a opção “fix imports” como mostra a imagem abaixo:

Icaro Daflon  
JAVA MOD II



- Porém para o caso do SQLException o NetBeans pode não reconhecer sozinho o import necessário portanto abaixo dos outros imports digite o código em destaque:

```
7 import controledeestoque.DAO.ProdutosDAO;
8 import java.lang.System.Logger;
9 import java.lang.System.Logger.Level;
10 import javax.swing.JOptionPane;
11 import java.sql.*;
```

- Agora digite o código abaixo, para que seja criado nosso segundo método:

```
44 public void Baixar (int id, int qtde){
45     try{
46         ProdutosDAO produtosDAO = new ProdutosDAO();
47         Integer quantidadeDeEstoque;
48         quantidadeDeEstoque = produtosDAO.buscarQuantidadeDeEstoque(id);
49         if(quantidadeDeEstoque != null){
50             produtosDAO.atualizar(id, quantidadeDeEstoque - qtde);
51             JOptionPane.showMessageDialog(null,
52                 "Estoque atualizado", "Sucesso",
53                 JOptionPane.INFORMATION_MESSAGE);
54         }else{
55             JOptionPane.showMessageDialog(null,
56                 "Código Inválido", "Erro",
57                 JOptionPane.ERROR_MESSAGE);
58         }
59     }catch (SQLException e){
60         System.getLogger(BaixarProdutosView.class.getName())
61             .log(Level.ERROR, e);
62         JOptionPane.showMessageDialog(null,
63             "Digite valores numéricos", "Erro",
64             JOptionPane.ERROR_MESSAGE);
65     }
66 }
```

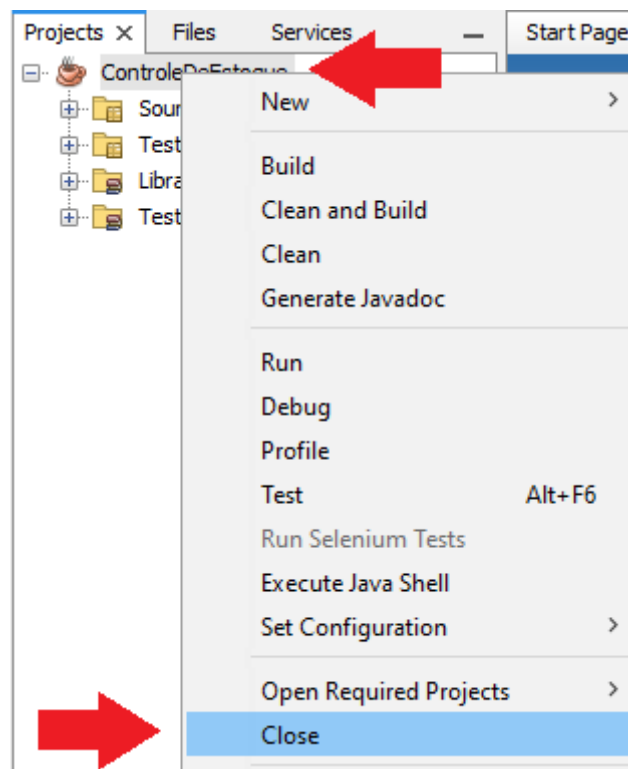
Perceba que mesmo após realizar as importações ainda ocorrem dois erros, um em cada método. Isso se deve ao fato de que as Views ainda não foram criadas. Com isso, não se preocupe com esses erros agora. Pois, na próxima aula, eles serão corrigidos.

- A classe foi finalizada. Salve o projeto e em seguida, feche o NetBeans.

## AULA 6

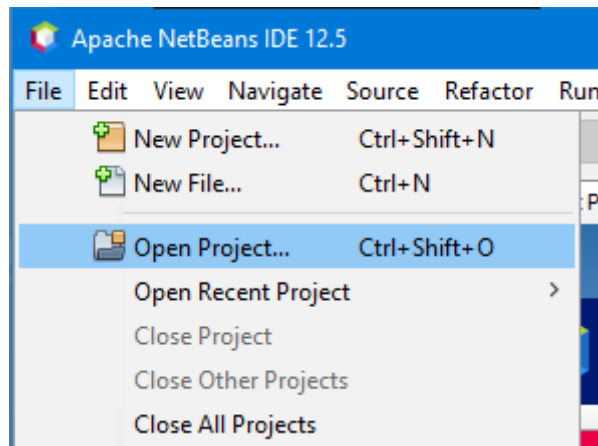
### Controle de Estoque – Continuação, Páginas 92 a 98

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 5” e copie a pasta do projeto “ControleDeEstoque”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

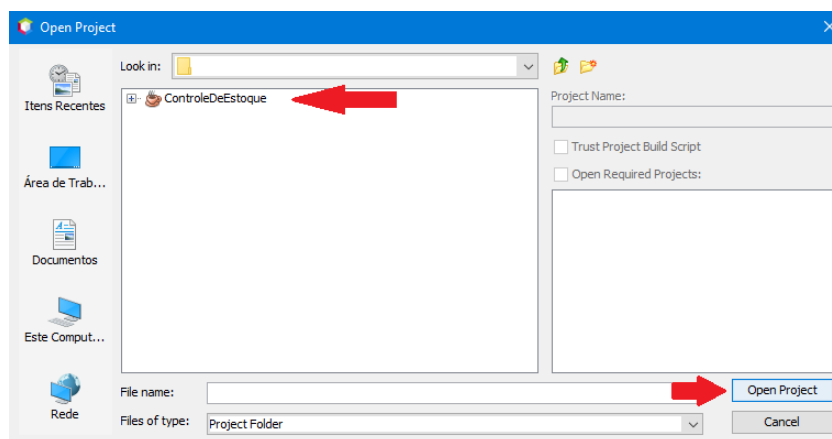


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”

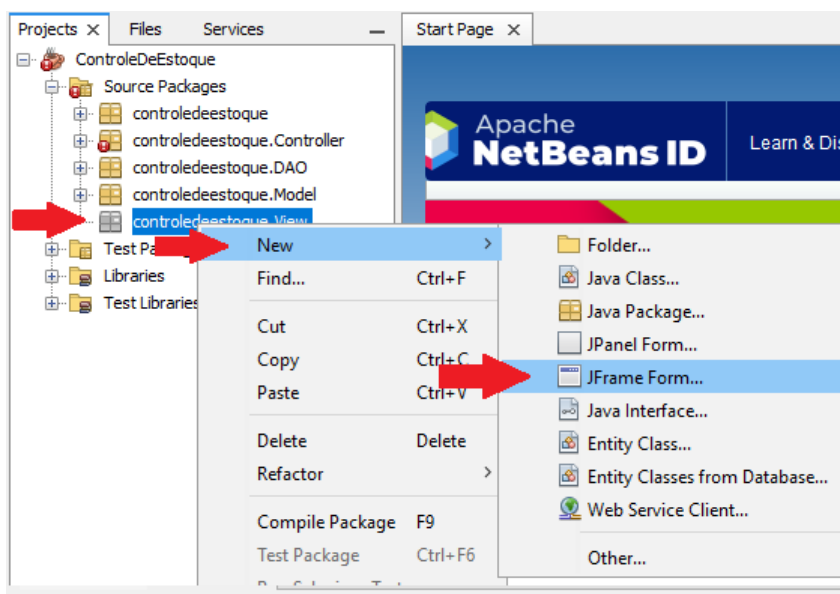
Icaro Daflon  
JAVA MOD II



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



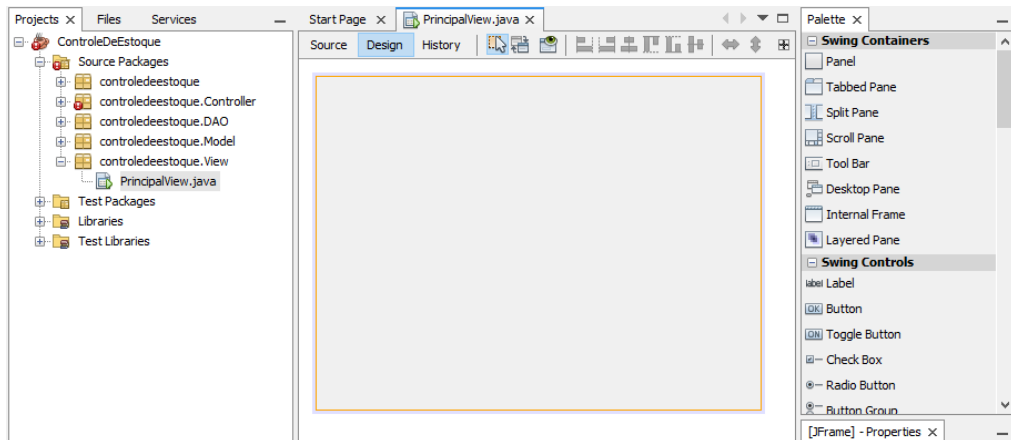
- Com o projeto inicializado corretamente, primeiro iremos criar a tela principal do sistema. Para isso, clique com o botão direito no pacote controledeestoque.View, posicione o cursor do mouse sobre “New” e em seguida clique sobre a opção “JFrame Form...”



## Icaro Daflon JAVA MOD II

- No campo “Class Name” digite “PrincipalView” e em seguida clique em “Finish”.

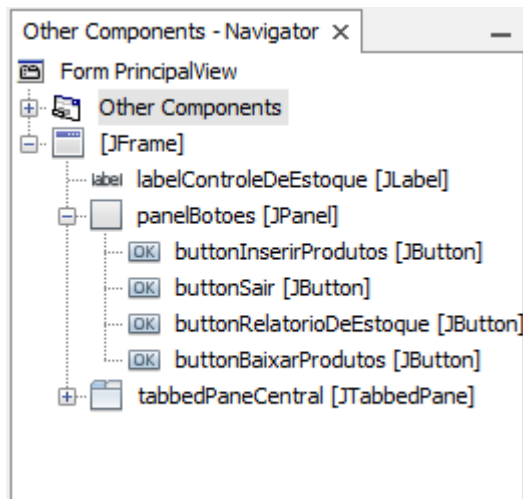
O NetBeans deve estar desta forma:



- Com o formulário criado deixe-o semelhante ao da imagem abaixo:



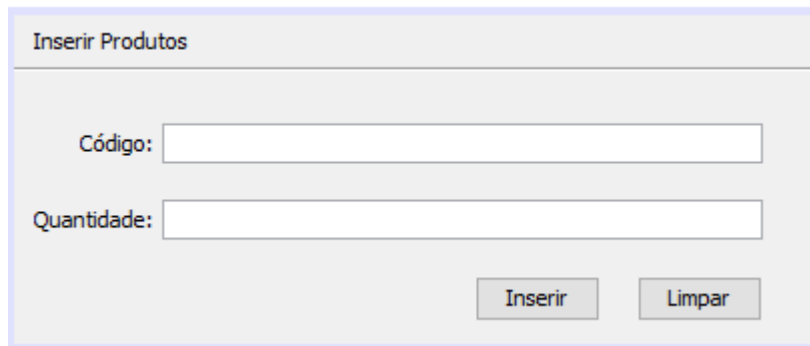
Os componentes usados para a criação do formulário e seus respectivos nomes são:



**Icaro Daflon**  
**JAVA MOD II**

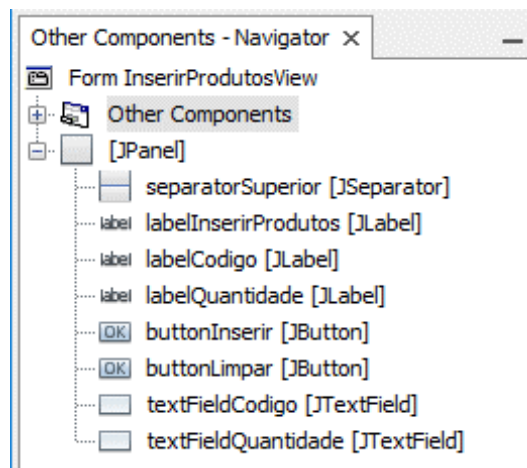
Agora, iremos criar outras duas classes, do tipo JPanel. Uma para fazermos a inserção do estoque e outro para baixa de estoque.

- Clique com o botão direito no pacote controledeestoque.View, no menu que surge posicione o cursor sobre “New” e após, clique sobre “JPanel Form...”
- Dê o nome de InserirProdutosView no campo “Class Name” e clique no botão “Finish”.
- Deixe o formulário como na imagem abaixo:

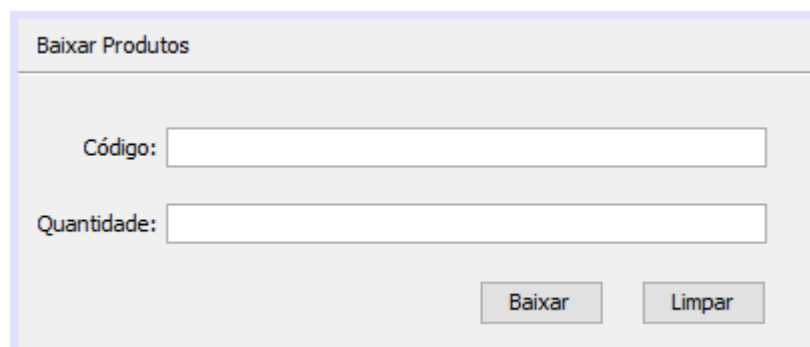


O formulário 'Inserir Produtos' possui um título 'Inserir Produtos' no topo. Abaixo dele, há dois campos de texto: 'Código:' e 'Quantidade:'. No canto inferior direito, há dois botões: 'Inserir' e 'Limpar'.

Os componentes dessa tela e seus respectivos nomes são:



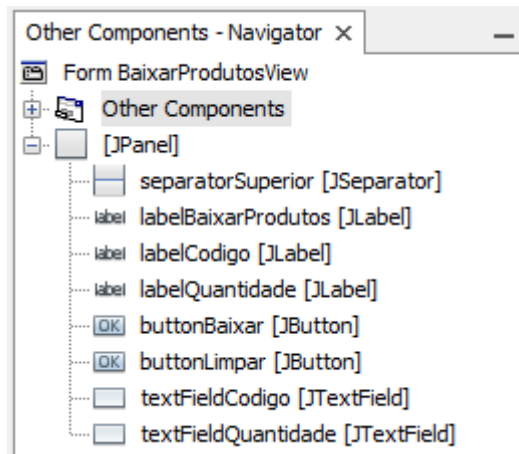
- Clique novamente com o botão direito no pacote controledeestoque.View, no menu que surge posicione o cursor sobre “New” e após, clique sobre “JPanel Form...”
- Dê o nome de BaixarProdutosView no campo “Class Name” e clique no botão “Finish”.
- Deixe o formulário como na imagem abaixo:



O formulário 'Baixar Produtos' possui um título 'Baixar Produtos' no topo. Abaixo dele, há dois campos de texto: 'Código:' e 'Quantidade:'. No canto inferior direito, há dois botões: 'Baixar' e 'Limpar'.



Os componentes dessa tela e seus respectivos nomes são:

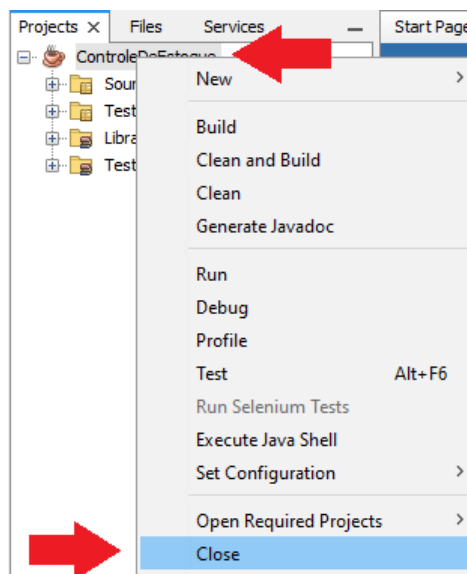


- Perceba a semelhança entre os formulários sendo que um é para Inserir produtos e o outro para Baixar produtos.
- Com isso finalizamos a aula, salve e feche o NetBeans.

## AULA 7

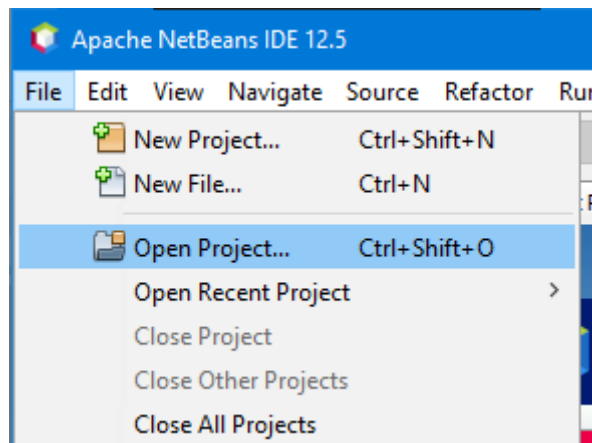
Controle de Estoque – Continuação, Páginas 100 a 110

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 6” e copie a pasta do projeto “ControleDeEstoque”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

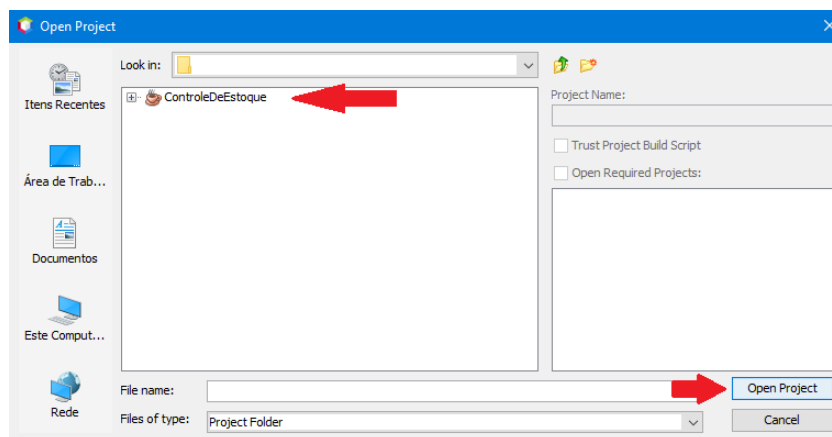


**Icaro Daflon**  
**JAVA MOD II**

- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



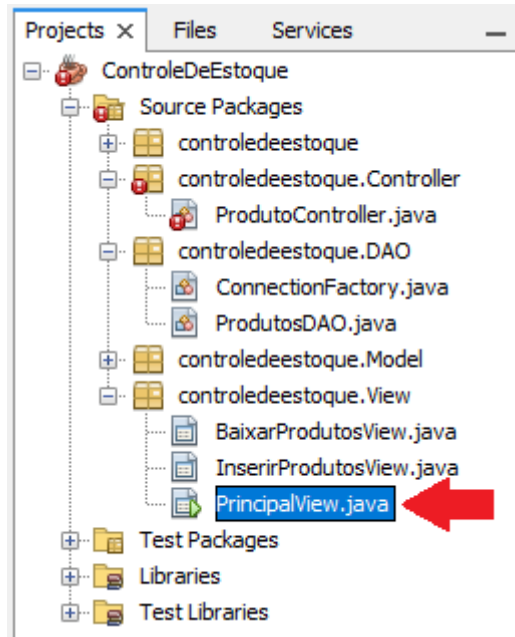
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL I4
- Verifique se os bancos de dados DBControleEstoque e o login Role UserControleEstoque já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBControleEstoque”.
- Agora devemos criar a tabela e popular ela, para isso, peça a seu instrutor o script “Tabela Produtos” e o execute no pgAdmin 4.

Com a tabela criada, o sistema e o banco de dados estão prontos para uso.

Vamos iniciar a programação dessa aula pelo botão Sair. Como o próprio nome diz, será responsável em sair do sistema.

- No NetBeans, abra o formulário PrincipalView.

## Icaro Daflon JAVA MOD II



- Agora, na guia Design, dê dois cliques no botão Sair: Com isso, será criado o método de ação do botão.
- Dentro desse método, inclua a ação para saída do sistema. Para isso digite o código abaixo:

```
if(JOptionPane.showConfirmDialog(this,  
    "Deseja mesmo fechar o sistema?", "Confirmação",  
    JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
    System.exit(0);  
}
```

- Provavelmente a palavra JOptionPane estará grifada indicando que um erro acontece, para corrigi-lo, clique com o botão direito sobre o código e em seguida, selecione “Fix Imports”, como feito nas aulas anteriores.
- O sistema já possui uma ação. A seguir, você deverá executar o projeto e testa-lo, mas antes, lembre-se que na classe ControleDeEstoque precisa ser instanciada a classe PrincipalView pois é a classe principal do nosso sistema.
- Abra a classe ControleDeEstoque e dentro do método “main”, digite o código:

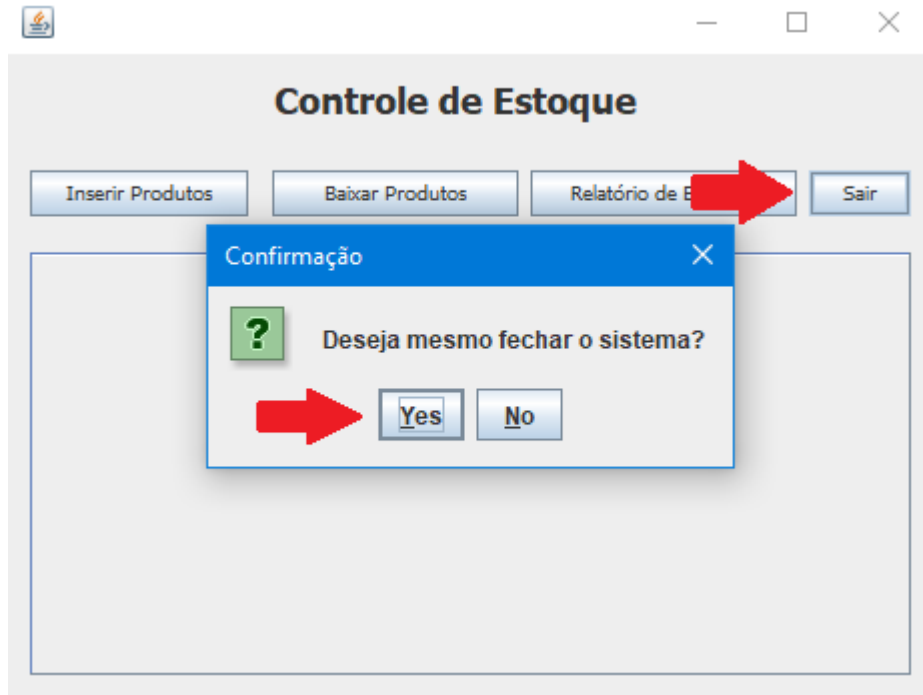
```
18  public static void main(String[] args) {  
19      PrincipalView principalView = new PrincipalView();  
20      principalView.validate();  
21      principalView.repaint();  
22      principalView.setVisible(true);  
23  }
```

Esse código irá criar uma instancia da classe PrincipalView e irá exibi-la na tela.

Agora, você irá corrigir o erro da classe ProdutoController.java

- Abra a classe ProdutoController.java
- Para corrigir os erros utilize o mesmo método utilizado anteriormente, clique com o botão direito do mouse sobre o código e selecione a opção “Fix Imports”

- Com isso feito, execute o projeto e verifique se o formulário PrincipalView está funcionando corretamente. Clique no botão Sair e após, clique em Sim para confirmar.



- Novamente na classe PrincipalView, selecione a guia “Design” e dê dois cliques sobre o botão “Inserir Produtos”.
- Digite o código abaixo dentro do método:

```
tabbedPaneCentral.removeAll();  
tabbedPaneCentral.add("Inserir Produtos",  
    new InserirProdutosView());  
tabbedPaneCentral.validate();  
tabbedPaneCentral.repaint();
```

- Insira agora também a ação do botão Baixar Produtos
- Para isso, volte para a guia “Design” e dê dois cliques sobre o botão “Baixar Produtos”.
- Digite o seguinte código dentro do método:

```
tabbedPaneCentral.removeAll();  
tabbedPaneCentral.add("Baixar Produtos",  
    new BaixarProdutosView());  
tabbedPaneCentral.validate();  
tabbedPaneCentral.repaint();
```

- Execute o programa para ver se está tudo funcionando corretamente:

The image displays two screenshots of a Java Swing application titled "Controle de Estoque".

The top screenshot shows the "Inserir Produtos" tab selected. A red arrow points to the "Inserir Produtos" button in the top navigation bar. The form contains two text boxes labeled "Código:" and "Quantidade:", and two buttons labeled "Inserir" and "Limpar".

The bottom screenshot shows the "Baixar Produtos" tab selected. A red arrow points to the "Baixar Produtos" button in the top navigation bar. The form contains two text boxes labeled "Código:" and "Quantidade:", and two buttons labeled "Inserir" and "Limpar".

Agora iremos programar as ações dos botões das classes `InserirProdutosView` e `BaixarProdutosView` respectivamente.

- Abra a classe `InserirProdutosView`, na guia "Design", dê dois cliques no botão "Limpar".
- Dentro do método digite o código abaixo:

```
textFieldCodigo.setText(null);  
textFieldQuantidade.setText(null);
```

- Faça o mesmo na classe `BaixarProdutosView`.

- Agora volte novamente para a classe InserirProdutosView e, na guia “Design”, dê dois cliques sobre o botão “Inserir”.
- Dentro do método digite o código abaixo:

```
ProdutoController produtoController = new ProdutoController();  
produtoController.Inserir(Integer.parseInt  
    (textFieldCodigo.getText()), Integer.parseInt  
    (textFieldQuantidade.getText()));
```

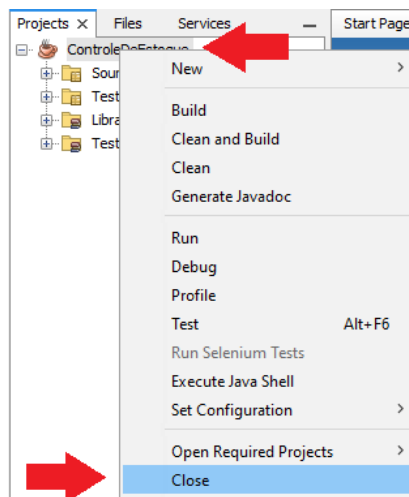
Observe que nesse código foi criada uma instância da classe ProdutoController e através dela, buscado a quantidade de estoque do código digitado. Após, foi atualizada a quantidade do produto somando a quantidade anterior à digitada no textFieldQuantidade. Caso o produto não exista no banco, surgirá uma mensagem de erro. E caso o usuário entre, por exemplo, com uma letra nos campos código e quantidade, novamente surgirá uma mensagem de erro.

- Faça agora a ação do botão “Baixar” da classe BaixarProdutosView, trocando o método utilizado pelo método Baixar criado nas aulas anteriores.
- Execute o projeto e verifique se tudo está de acordo com o esperado.
- Salve o projeto. Em seguida feche o NetBeans e o pgAdmin 4.

## AULA 8

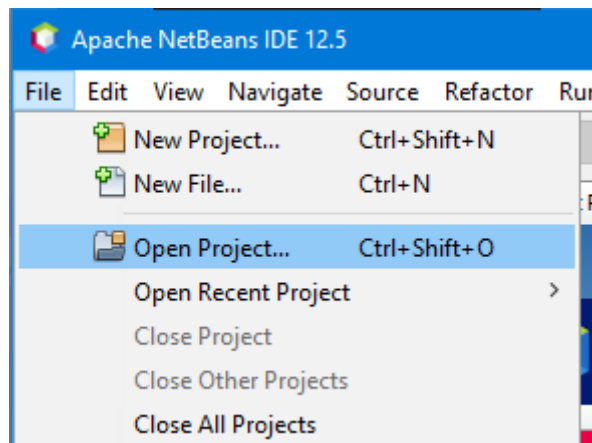
### Controle de Estoque – Relatório de Estoque, Páginas 113 a 124

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 6” e copie a pasta do projeto “ControleDeEstoque”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

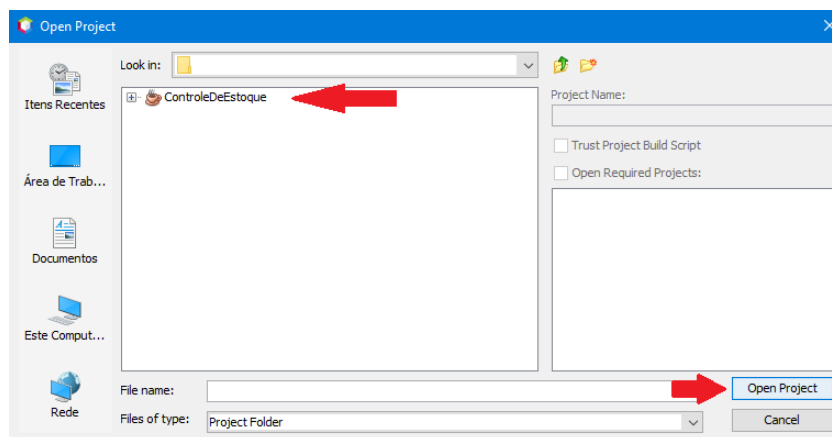


**Icaro Daflon**  
**JAVA MOD II**

- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



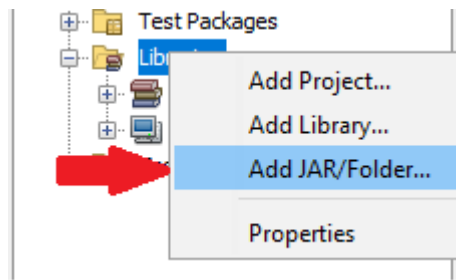
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL I4
- Verifique se os bancos de dados DBControleEstoque e o login Role UserControleEstoque já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBControleEstoque”.
- Agora devemos criar a tabela e popular ela, para isso, peça a seu instrutor o script “Tabela Produtos” e o execute no pgAdmin 4.

Com a tabela criada, o sistema e o banco de dados estão prontos para uso.

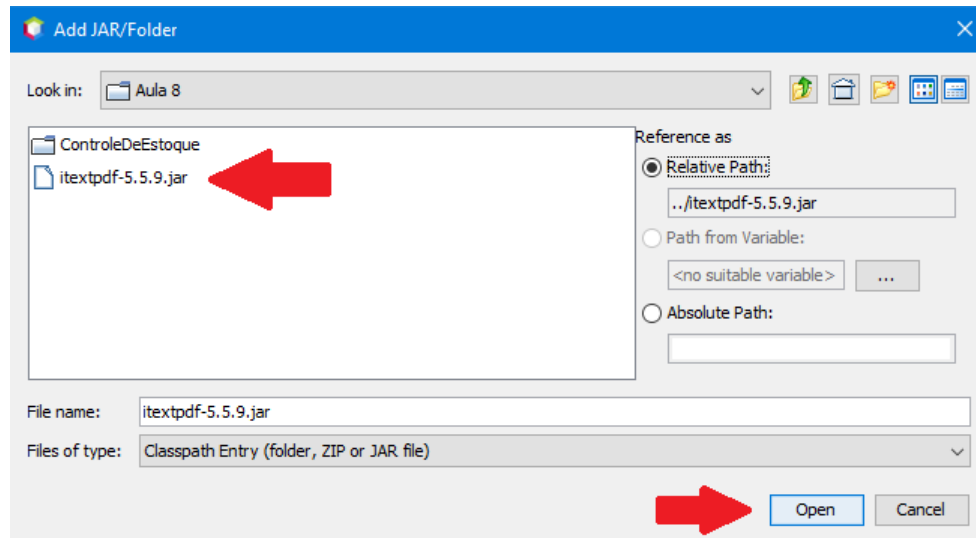
Agora iremos iniciar fazendo a importação da API

- Peça a seu instrutor o arquivo “itextpdf-5.5.9.jar” e ponha-o em seu local de gravação.
- Clique com o botão direito sobre a pasta “Libraries”, em seguida, clique sobre “Add JAR/Folder...”

## Icaro Daflon JAVA MOD II



- Na janela “Add JAR/Folder”, selecione o arquivo “itextpdf-5.5.9.jar” em seu local de gravação e após, clique em “Open”.



Com isso nós já podemos utilizar o recurso da API

- Crie uma nova “Java Class” dentro do pacote controledeestoque.Controller, com o nome “RelatorioDeEstoque”.
- Insira as variáveis abaixo, na classe criada, como variáveis globais.

```
13 public class RelatorioDeEstoque {  
    private String local;  
    private PdfPTable table;  
16 }
```

- Crie o construtor da classe como abaixo:

```
public RelatorioDeEstoque(String local) {  
    this.local = local;  
    this.table = new PdfPTable(3);  
}
```

Na instanciação da PdfPTable, foi definido que ela possuirá 3 colunas.

- Agora iremos criar dois métodos que serão fundamentais nessa class. O método para imprimir o titulo de cada produto e o segundo imprime os detalhes de cada produto.
- Crie um novo método com o nome imprimeTitulo e deixe-o como abaixo:



## Icaro Daflon JAVA MOD II

```
26 public void imprimeTitulo(String linha){
27     PdfPCell cellcodigo, cellestoque,
28     celldescricao, celltitulo, cellbranco;
29     cellbranco = new PdfPCell(new Paragraph(" "));
30     celltitulo = new PdfPCell(new Paragraph(linha,
31     new Font(Font.FontFamily.UNDEFINED, 14,
32     Font.BOLD)));
33     cellcodigo = new PdfPCell(new Paragraph("Código",
34     new Font(Font.FontFamily.UNDEFINED, 12,
35     Font.BOLD)));
36     cellestoque = new PdfPCell(new Paragraph("Estoque",
37     new Font(Font.FontFamily.UNDEFINED, 12,
38     Font.BOLD)));
39     celldescricao = new PdfPCell(new Paragraph("Descrição",
40     new Font(Font.FontFamily.UNDEFINED, 12,
41     Font.BOLD)));
42     celltitulo.setHorizontalAlignment(Element.ALIGN_CENTER);
43     celltitulo.setColspan(3);
44     celltitulo.setBorder(0);
45     cellcodigo.setHorizontalAlignment(Element.ALIGN_CENTER);
46     cellestoque.setHorizontalAlignment(Element.ALIGN_CENTER);
47     celldescricao.setHorizontalAlignment(Element.ALIGN_CENTER);
48     cellbranco.setColspan(3);
49     cellbranco.setBorder(0);
50     table.addCell(cellbranco);
51     table.addCell(cellbranco);
52     table.addCell(celltitulo);
53     table.addCell(cellcodigo);
54     table.addCell(cellestoque);
55     table.addCell(celldescricao);
56 }
```

- Crie agora o método com o nome `imprimeCorpo` e deixe-o como abaixo:

```
58 public void imprimeCorpo(int codigo, int quantidade, String descricao){
59     PdfPCell cellcodigo, cellestoque, celldescricao;
60     cellcodigo = new PdfPCell(new Paragraph(
61     Integer.toString(codigo)));
62     cellestoque = new PdfPCell(new Paragraph(
63     Integer.toString(quantidade)));
64     celldescricao = new PdfPCell(new Paragraph(
65     descricao));
66     cellcodigo.setHorizontalAlignment(Element.ALIGN_CENTER);
67     cellestoque.setHorizontalAlignment(Element.ALIGN_CENTER);
68     celldescricao.setHorizontalAlignment(Element.ALIGN_CENTER);
69     table.addCell(cellcodigo);
70     table.addCell(cellestoque);
71     table.addCell(celldescricao);
72 }
```

- Com os métodos de impressão criados, vamos criar agora o método para gerar o relatório. Para isso crie um novo método chamado `gerarRelatorio` que retorne um valor boolean para sabermos se o relatório foi gerado corretamente e deixe-o como o código abaixo:

## Icaro Daflon JAVA MOD II

```
83 public boolean gerarRelatorio() throws Exception{
84     Document document = new Document();
85     DateFormat df = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
86     try{
87         PdfWriter.getInstance(document,
88             new FileOutputStream(local));
89         document.open();
90         int headerwidths[] = {12, 12, 76};
91         table.setWidths(headerwidths);
92         table.setWidthPercentage(100);
93         table.getDefaultCell().setBorder(1);
94         table.getDefaultCell().setHorizontalAlignment(
95             Element.ALIGN_CENTER);
96         table.getDefaultCell().setVerticalAlignment(
97             Element.ALIGN_MIDDLE);
98         table.setHorizontalAlignment(Element.ALIGN_CENTER);
99         Paragraph t;
100         document.add(new Paragraph(" "));
101         document.addTitle("Relatório do estoque de produtos");
102         document.add(new Paragraph(" "));
103         t = new Paragraph("RELATÓRIO DE ESTOQUE",
104             new Font(Font.FontFamily.UNDEFINED, 16,
105                 Font.BOLD));
106         t.setAlignment(Element.ALIGN_CENTER);
107         document.add(t);
108         document.add(new Paragraph(
109             "
110             + "
111         t = new Paragraph(df.format(Calendar.getInstance(
112             new Locale("pt", "BR")).
113                 getTime()));
114         t.setAlignment(Element.ALIGN_RIGHT);
115         document.add(t);
116         ProdutosDAO produtosDAO = new ProdutosDAO();
117         ResultSet rs = produtosDAO.buscarTodosProdutos();
118
119         if(rs != null){
120             while(rs.next()){
121                 int codigo = rs.getInt("id");
122                 String nome = rs.getString("nome");
123                 String descricao = rs.getString("descricao");
124                 int quantidade = rs.getInt("quantidade");
125                 imprimeTitulo(nome);
126                 imprimeCorpo(codigo, quantidade, descricao);
127             }
128             document.add(table);
129             document.add(new Paragraph(
130                 "
131                 + "
132             document.add(t);
133             document.close();
134             return true;
135         }
136     }catch (Exception e){
137         System.err.println(e.getMessage());
138     }
139     document.close();
140     return false;
141 }
```

- Não se esqueça de realizar todos os imports necessários para o funcionamento da classe, utilizando o “Fix Imports” utilizado em aulas anteriores. Os imports devem estar da seguinte maneira:

```
7  import com.itextpdf.text.Document;
8  import com.itextpdf.text.Element;
9  import com.itextpdf.text.Font;
10 import com.itextpdf.text.Paragraph;
11 import com.itextpdf.text.pdf.PdfPCell;
12 import com.itextpdf.text.pdf.PdfPTable;
13 import com.itextpdf.text.pdf.PdfWriter;
14 import controledeestoque.DAO.ProdutosDAO;
15 import java.io.FileOutputStream;
16 import java.text.DateFormat;
17 import java.util.Calendar;
18 import java.util.Locale;
19 import java.sql.*;
20 import java.text.SimpleDateFormat;
```

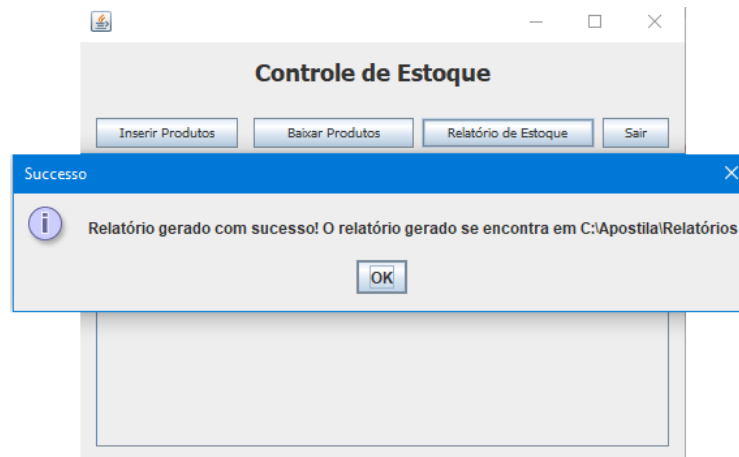
- Agora iremos criar a ação para o botão Relatório de Estoque, para isso, abra a classe PrincipalView
- Na guia Design, clique duas vezes sobre o botão “Relatório de Estoque”
- Digite o código abaixo dentro do método:

```
try{
    DateFormat df = new SimpleDateFormat("dd_MM_yyyy");
    String data = df.format(Calendar.getInstance(
        new Locale("pt", "BR")).getTime());
    RelatorioDeEstoque relatorioEstoque =
        new RelatorioDeEstoque("C:\\Program Files\\Apostila"
            + "RelatorioEstoque.pdf" + data + ".pdf");
    if(relatorioEstoque.gerarRelatorio()){
        JOptionPane.showMessageDialog(this,
            "Relatório gerado com sucesso! O relatório "
            + "gerado se encontra em C:\\Program Files\\Apostila",
            "Sucesso", JOptionPane.INFORMATION_MESSAGE);
    }else{
        JOptionPane.showMessageDialog(this,
            "Relatório não gerado!", "Erro", JOptionPane.ERROR_MESSAGE);
    }
}catch(Exception e){
    System.getLogger(PrincipalView.class
        .getName()).log(System.Logger.Level.ERROR, e);
}
```

Obs: O caminho do relatório neste código é fictício. Consulte o instrutor e veja a pasta que o mesmo pode ser gerado. Feito isto está tudo pronto para gerarmos nosso relatório.

- Execute o projeto e clique sobre o botão “Relatório de Estoque”

## Icaro Daflon JAVA MOD II



- Após a execução, vá até a pasta que você definiu para gravar e verifique se o relatório foi gravado corretamente.

RELATÓRIO DE ESTOQUE		
19/10/2022 03:45:17		
<b>Geladeira</b>		
Código	Estoque	Descrição
1	37	Utensílio eletrodoméstico utilizado na conservação de alimento
<b>Fogão</b>		
Código	Estoque	Descrição
2	32	Utensílio culinário usado para cozinhar, geralmente em panelas ou frigideiras, e por meio de calor
<b>Máquina de lavar roupas</b>		
Código	Estoque	Descrição
3	6	Máquina projetada para limpeza de roupas
<b>Ar condicionado</b>		
Código	Estoque	Descrição
4	14	Responsável pelo tratamento do ar interior em espaços fechados
<b>Lava-louças</b>		
Código	Estoque	Descrição
5	27	Aparelho eletrodoméstico cuja finalidade é lavar os apetrechos utilizados na cozinha
<b>Micro-ondas</b>		
Código	Estoque	Descrição
6	10	Aparelho eletrodoméstico que possibilita a preparação rápida de alimentos para o consumo humano ou de animais
19/10/2022 03:45:17		

- Faça testes, atualizando e baixando o estoque e criando novos relatórios para verificar se os valores estão todos corretos. Após feche o NetBeans.

# AULA 9

Criando as tabelas, Páginas 126 a 129

- Primeiro, abra o pgAdmin 4 e conecte-se ao servidor PostgreSQL 14.
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Após conectar-se iremos criar as tabelas para isso selecione o Banco de Dados “DBJHospital” e clique no botão “Query Tool” para abrir o editor.
- Na janela Query Tool agora execute o código abaixo, para criar a tabela paciente e a sequência paciente\_seq:

```
Query  Query History
1  CREATE TABLE paciente(
2      id INT PRIMARY KEY,
3      nome VARCHAR(256) NOT NULL,
4      email VARCHAR(64) NOT NULL,
5      telefone VARCHAR(16),
6      endereco VARCHAR(256) NOT NULL,
7      doenca VARCHAR(256) NOT NULL,
8      temPlanoDeSaude BOOLEAN NOT NULL,
9      diasDeInternacao INT,
10     numeroDoQuarto INT
11 );
12
13 CREATE SEQUENCE paciente_seq
14     START 0
15     MINVALUE 0;
```

- Para criar a tabela Visitante e a sequência visitante\_seq execute o seguinte código:

```
Query  Query History
1  CREATE TABLE visitante(
2      id INT PRIMARY KEY,
3      nome VARCHAR(256) NOT NULL,
4      email VARCHAR(64) NOT NULL,
5      telefone VARCHAR(16),
6      endereco VARCHAR(256) NOT NULL,
7      paciente VARCHAR(256) NOT NULL
8  );
9
10 CREATE SEQUENCE visitante_seq
11     START 0
12     MINVALUE 0;
```

- Para criar a tabela Médico e a sequência medico\_seq execute o código abaixo:

Query	Query History
1	CREATE TABLE medico(
2	id INT PRIMARY KEY,
3	nome VARCHAR(256) NOT NULL,
4	email VARCHAR(64) NOT NULL,
5	telefone VARCHAR(16),
6	endereco VARCHAR(256) NOT NULL,
7	especialidade VARCHAR(256) NOT NULL,
8	horasMensais INT,
9	valorDasHoras INT
10	);
11	
12	CREATE SEQUENCE medico_seq
13	START 0
14	MINVALUE 0;

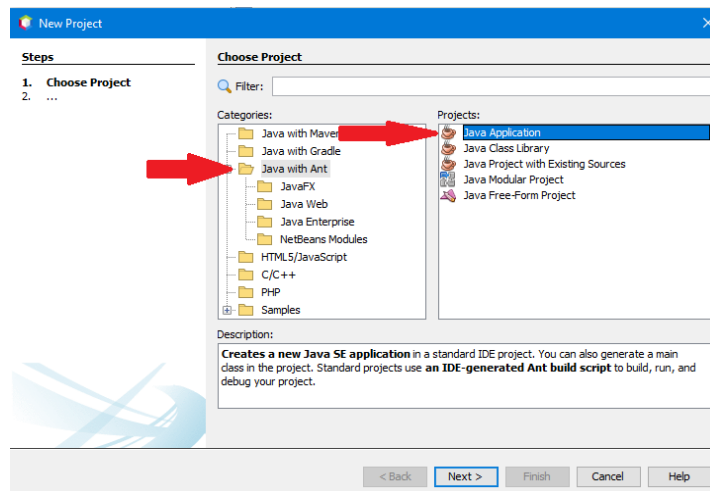
- E por último para criar a tabela Enfermeiro execute o seguinte código:

Query	Query History
1	CREATE TABLE enfermeiro(
2	id INT PRIMARY KEY,
3	nome VARCHAR(256) NOT NULL,
4	email VARCHAR(64) NOT NULL,
5	telefone VARCHAR(16),
6	endereco VARCHAR(256) NOT NULL,
7	horasMensais INT,
8	valorDasHoras INT
9	);
10	
11	CREATE SEQUENCE enfermeiro_seq
12	START 0
13	MINVALUE 0;

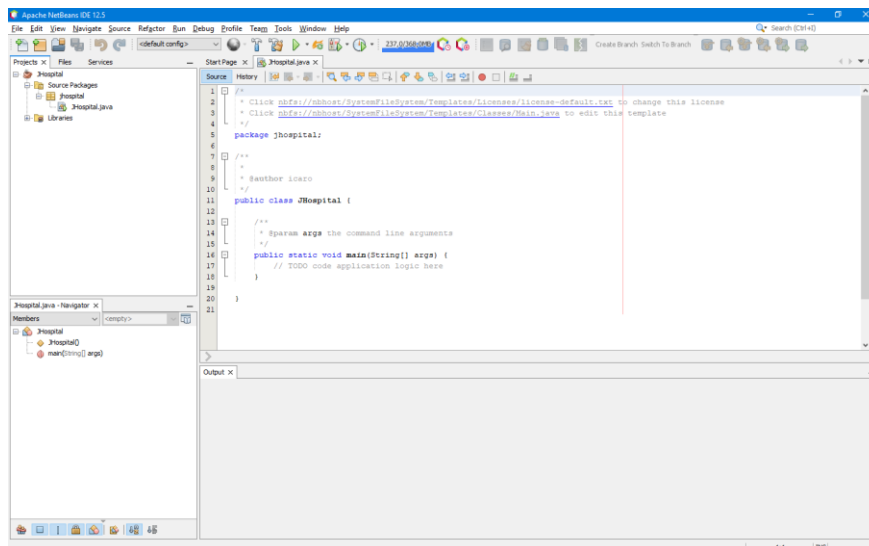
## JHospital – Projeto, Páginas 129 a 131

- Abra o NetBeans, caso haja um projeto aberto feche-o.
- Com o NetBeans aberto, crie um novo projeto acessando o menu File, e clicando no botão “New Project...”
- Na janela que surge, selecione a categoria “Java with Ant” e a opção “Java Application”

## Icaro Daflon JAVA MOD II

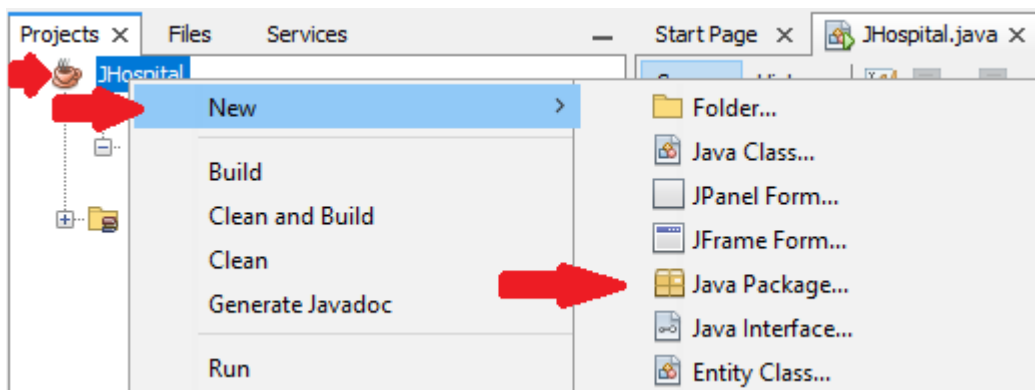


- Após clique no botão Próximo. Na janela que se abre, insira em “Project Name”, “JHospital”, e no campo “Project Location”, selecione o seu local de gravação. Após, clique em “Finish”.
- Sua tela deverá estar assim:



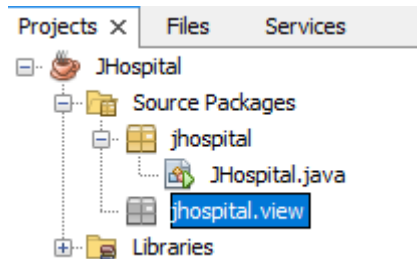
### JHospital – Pacotes, Páginas 132 e 133

- Clique com o botão direito sobre o projeto JHospital, selecione “New” e depois escolha a opção “Java Package”

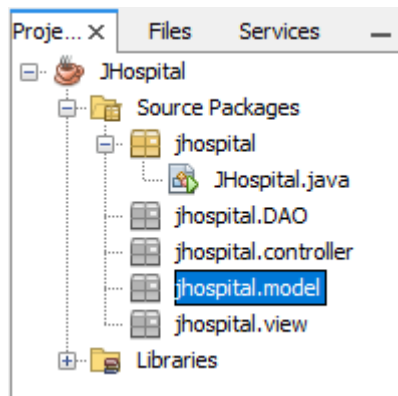


## Icaro Daflon JAVA MOD II

- Na janela “New Java Package”, é necessário definir o nome do pacote. Assim defina-o como `jhospital.view`, clique no botão “Finish”
- A aba Projects ficará desta forma:

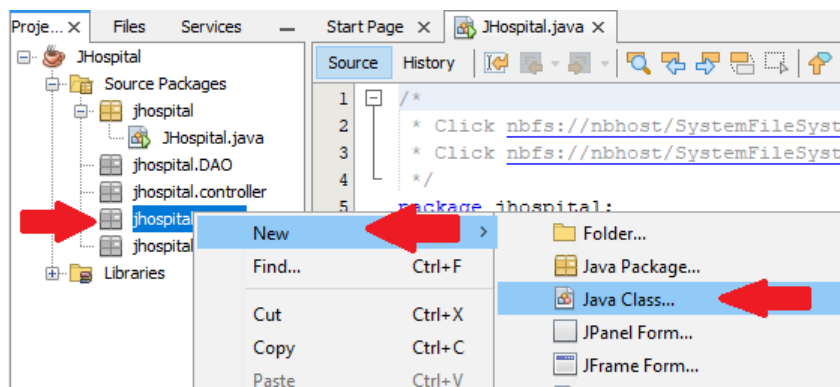


- Após a criação do view iremos criar o restante dos pacotes do MVC, portanto crie os pacotes Model, Controller e DAO. Eles terão respectivamente os nomes `jhospital.model`, `jhospital.controller` e `jhospital.DAO`.
- Sua aba Projects ficará assim:



### JHospital – Classes Model

- Como podemos observar nossas tabelas são baseadas em 4 tipos de pessoas, todas as 4 possuindo 4 atributos em comum, sendo eles: id, nome, email, telefone e endereço. Portanto para facilitar a criação de nossas classes iremos criar uma classe modelo para elas. Esta classe será chamada de Pessoa já que engloba características em comum das outras 4 classes. Posteriormente iremos criar as classes Paciente, Médico, Visitante e Enfermeiro cada um com suas especificações.
- Agora iremos criar nossa classe model para isso clique com o botão direito sobre o pacote `jhospital.model`, posicione o cursor sobre “New” e selecione a opção “Java Class...”



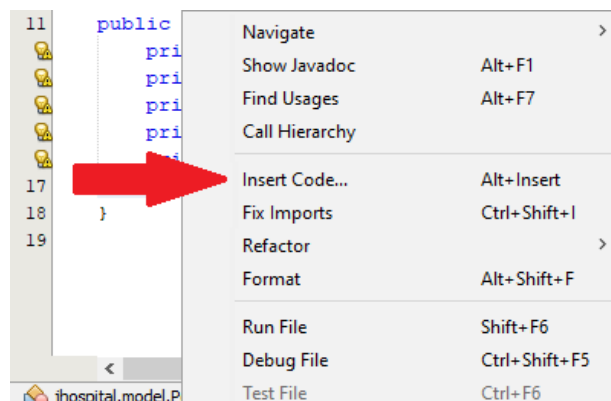


**Icaro Daflon**  
**JAVA MOD II**

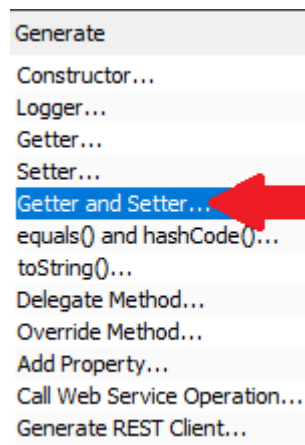
- No campo “Class Name”, defina o nome como Pessoa e após, clique sobre o “Finish”.
- Agora iremos preencher a classe com os atributos que ela terá. Como falado anteriormente esta classe possuirá apenas os atributos que as outras classes possuem em comum, neste caso: id, nome, email, telefone e endereço, onde id é um objeto numérico e os restantes serão Strings. Portanto digite o código como abaixo:

```
5 package jhospital.model;
6
7 /**
8  *
9  * @author icaro
10 */
11 public class Pessoa {
12     private long id;
13     private String nome;
14     private String email;
15     private String telefone;
16     private String endereco;
17
18 }
```

- Após digitar o código iremos definir os getters e setters da classe. Para isso, clique com o botão direito dentro do código, abaixo do ultimo objeto criado (endereco) e escolha a opção “Insert code...”

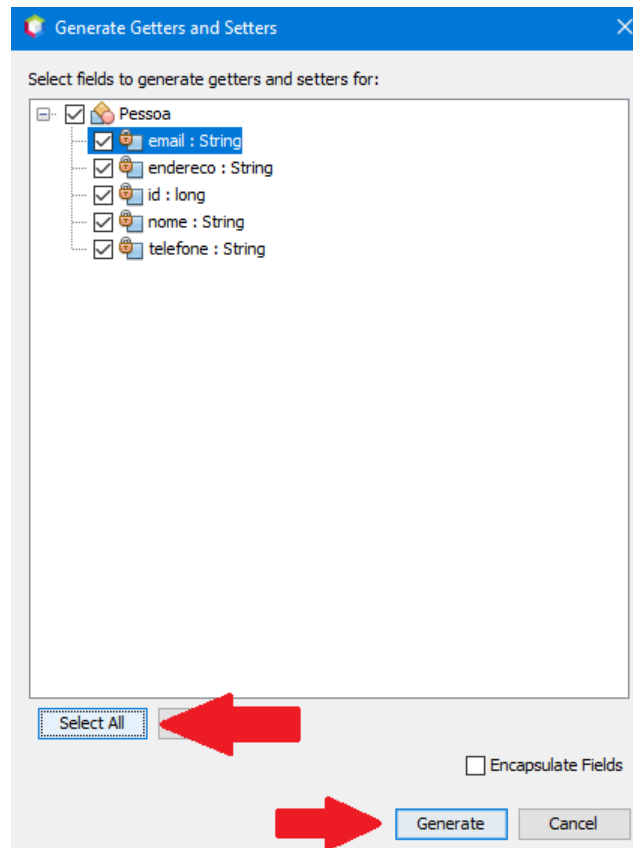


- Na caixa de opções que surge, selecione a opção “Getter and Setter”



- A seguir, é possível selecionar os objetos que iremos criar os Getters e Setters automaticamente. Nesse caso, serão gerados para todos. Portanto, clique no botão “Select All” e após em “Generate”.

## Icaro Daflon JAVA MOD II



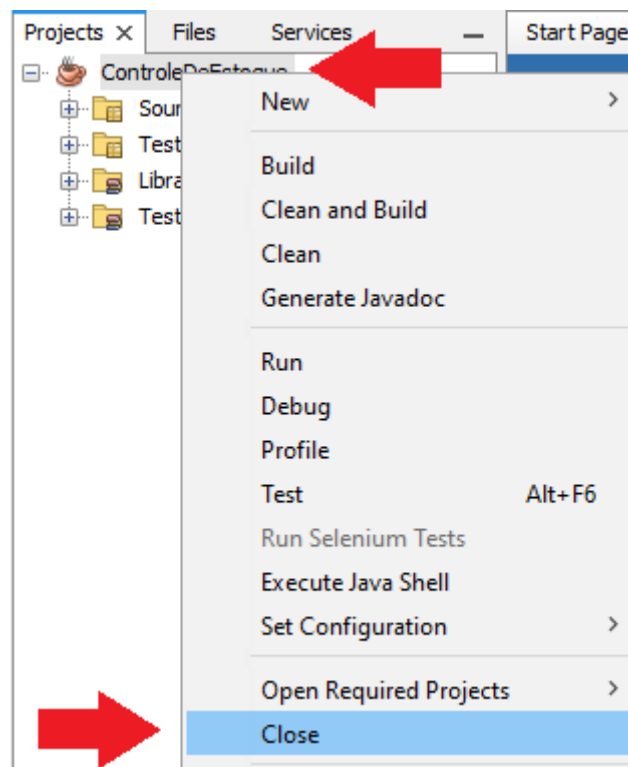
- Os métodos foram gerados e sua classe Pessoa deve estar da seguinte forma:

```
11 public class Pessoa {  
12     private long id;  
13     private String nome;  
14     private String email;  
15     private String telefone;  
16     private String endereco;  
17  
18     public long getId() {  
19         return id;  
20     }  
21  
22     public void setId(long id) {  
23         this.id = id;  
24     }  
25  
26     public String getNome() {  
27         return nome;  
28     }  
29  
30     public void setNome(String nome) {  
31         this.nome = nome;  
32     }  
33  
34     public String getEmail() {  
35         return email;  
36     }  
37  
38     public void setEmail(String email) {  
39         this.email = email;  
40     }  
41  
42     public String getTelefone() {  
43         return telefone;  
44     }  
45 }
```

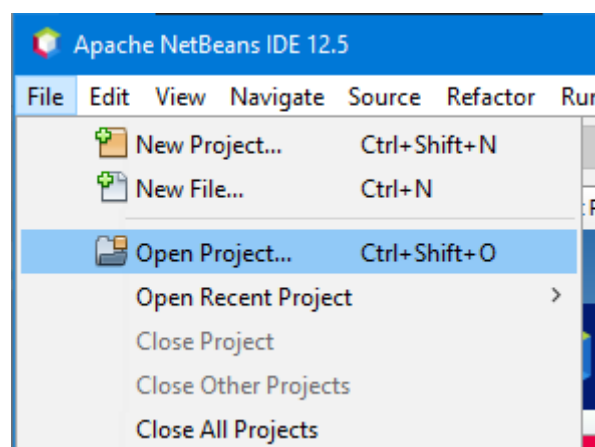
- Com isso finalizamos esta aula. Salve todas as criações e após feche todos os programas utilizados.

# AULA 10

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 9” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

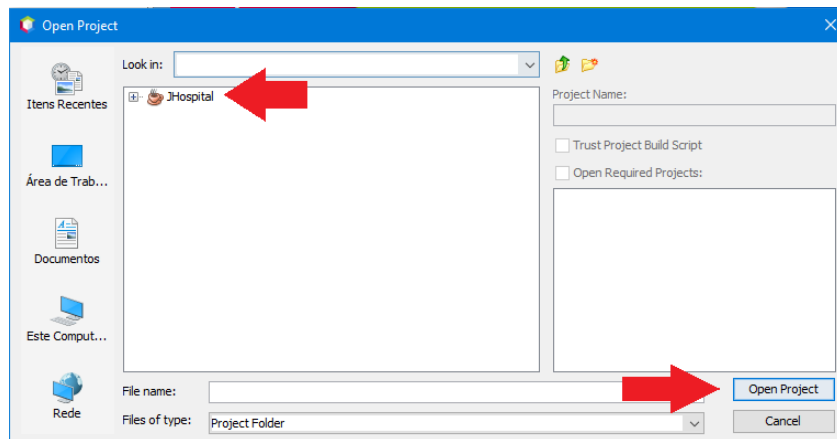


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

## Icaro Daflon JAVA MOD II



- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL I4
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

### JHospital Model – Continuação, Páginas I42 a I45

- Com tudo pronto volte para o NetBeans. Clique com o botão direito sobre o pacote jhospital.model, vá em “New” e clique na opção “Java Class...”
- Defina o nome como paciente, no campo “Class Name” e clique no botão “Finish”
- Agora faremos a classe paciente herdar a classe pessoa. Para isso, na linha em que a classe é definida, acrescente “extends Pessoa”, como na figura abaixo:

```
5      package jhospital.model;
6
7      /**
8       *
9       * @author icaro
10     */
11     public class Paciente extends Pessoa{
12
13     }
14
```

- Pronto, a classe Paciente já tem a classe Pessoa como pai. Agora, iremos definir os objetos da classe Paciente, além dos que já foram criados na classe Pessoa. Defina os objetos e crie os métodos getters e setters na classe Paciente como feito na classe Pessoa. A classe terá os seguintes objetos:

```
private String doenca;  
private boolean temPlanoDeSaude;  
private int diasDeInternacao;  
private int numeroDoQuarto;
```

- No final sua classe deve estar da seguinte forma:

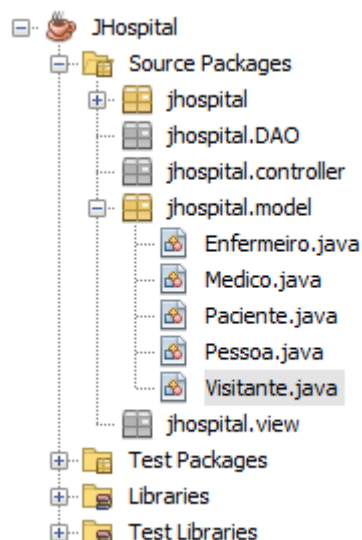
```
11 public class Paciente extends Pessoa{  
12     private String doenca;  
13     private boolean temPlanoDeSaude;  
14     private int diasDeInternacao;  
15     private int numeroDoQuarto;  
16  
17     public String getDoenca() {  
18         return doenca;  
19     }  
20  
21     public void setDoenca(String doenca) {  
22         this.doenca = doenca;  
23     }  
24  
25     public boolean isTemPlanoDeSaude() {  
26         return temPlanoDeSaude;  
27     }  
28  
29     public void setTemPlanoDeSaude(boolean temPlanoDeSaude) {  
30         this.temPlanoDeSaude = temPlanoDeSaude;  
31     }  
32  
33     public int getDiasDeInternacao() {  
34         return diasDeInternacao;  
35     }  
36  
37     public void setDiasDeInternacao(int diasDeInternacao) {  
38         this.diasDeInternacao = diasDeInternacao;  
39     }  
40  
41     public int getNumeroDoQuarto() {  
42         return numeroDoQuarto;  
43     }  
44  
45     public void setNumeroDoQuarto(int numeroDoQuarto) {  
46         this.numeroDoQuarto = numeroDoQuarto;  
47     }  
48 }
```

- Agora, crie as classes para Médico, Visitante e Enfermeiro, lembrando que elas também herdarão da classe Pessoa. A tabela abaixo demonstra quais objetos/variáveis que cada classe conterá:

Medico		
Acesso	Tipo	Nome da Variável
private	String	especialidade
private	int	horasMensais
private	int	valorDasHoras
Visitante		
Acesso	Tipo	Nome da Variável
private	String	paciente
Enfermeiro		
Acesso	Tipo	Nome da Variável
private	int	horasMensais
private	int	valorDasHoras

É interessante salientar que as classes Medico e Enfermeiro possuem dois objetos iguais. Portanto seria possível criar uma classe abstrata com os getters e setters destes dois objetos e as classes Medico e Enfermeiro apenas estenderiam esta classe. Porém por se tratar de apenas dois objetos, neste caso, é interessante mantê-los separados.

Após a criação das classes, sua aba Projetos deve estar da seguinte forma:



## JHospital – Classes Controller, Páginas 145 a 151

Inicialmente iremos criar apenas as classes, e os métodos principais de cada uma, sem nenhuma ação. Os métodos serão preenchidos com suas ações após a criação das Views e dos DAOs.

- Primeiramente, crie a classe de controle do Paciente. Para isso clique com o botão direito no pacote `jhospital.controller`, posicione o cursor sobre “New” e após clique em “Java Class...”
- Defina o nome `PacienteController` no campo “Class Name” e após clique em “Finish”.
- Agora, na classe `PacienteController`, crie o método `inserir` como na figura abaixo:

```
12 public void inserir(String nome, String email, String endereco,  
13 String telefone, String quarto, String doenca,  
14 String diasInternado, String temPlanoDeSaude) throws Exception{  
15  
16 }
```

- Os argumentos do método `inserir` são de acordo com a classe `Paciente` do pacote `jhospital.model`, é possível observar que o paciente terá 8 informações que serão captadas nas classes de visualização “View”, sendo 4 vindos da classe `Pessoa` e outras 4 vindos da própria classe `Paciente` em questão.
- Agora crie o método `alterar` seguindo a figura abaixo:

```
18 public void alterar(Integer id, String nome, String email,  
19 String endereco, String telefone, String quarto, String doenca,  
20 String diasInternado, String temPlanoDeSaude) throws Exception{  
21  
22 }
```

- Os parâmetros são os mesmos do método `inserir`, com acréscimo da variável `id`, pois como está sendo feita uma alteração no cadastro do paciente, é necessário saber qual a identificação que ele possui no banco de dados.

- Agora, crie o método `excluir` como na figura abaixo:

```
24 public void excluir(Integer id) throws Exception{  
25  
26 }
```

- Aqui, perceba que é passado somente o `id`, pois precisa-se apenas dele para que a exclusão seja realizada.
- Agora crie o método `buscar` seguindo a figura abaixo:

```
31 public List<Paciente> buscar(String nome, String email) throws Exception{  
32 return null;  
33 }  
34 }
```

OBS: Lembre-se de corrigir as importações

- Neste método, foi adicionado como retorno uma `List<Paciente>` pois esse será o retorno do método futuramente, ou seja, uma lista de pacientes. Inicialmente, o retorno do método é `null`, pois você ainda não irá preenchê-lo.

- Sua classe deve estar da seguinte forma:

```
5 package jhospital.controller;
6
7 import java.util.List;
8 import jhospital.model.Paciente;
9
10 /**
11  *
12  * @author icaro
13  */
14 public class PacienteController {
15     public void inserir(String nome, String email, String endereco,
16         String telefone, String quarto, String doenca,
17         String diasInternado, String temPlanoDeSaude) throws Exception{
18     }
19
20     public void alterar(Integer id, String nome, String email,
21         String endereco, String telefone, String quarto, String doenca,
22         String diasInternado, String temPlanoDeSaude) throws Exception{
23     }
24
25     public void excluir(Integer id) throws Exception{
26     }
27
28     public List<Paciente> buscar(String nome, String email) throws Exception{
29         return null;
30     }
31 }
32
33
34
35
```

- Agora crie as outras classes controllers do sistema.

Crie as classes EnfermeiroController, MedicoController e VisitanteController e seus respectivos métodos de acordo com a tabela abaixo:

EnfermeiroController		
Retorno	Métodos	Parâmetros
void	inserir	String nome String email String endereço String telefone String horasMensais String valorDasHoras
void	alterar	Integer id String nome String email String endereço String telefone String horasMensais String valorDasHoras
void	excluir	Integer id
List<Enfermeiro>	buscar	String nome

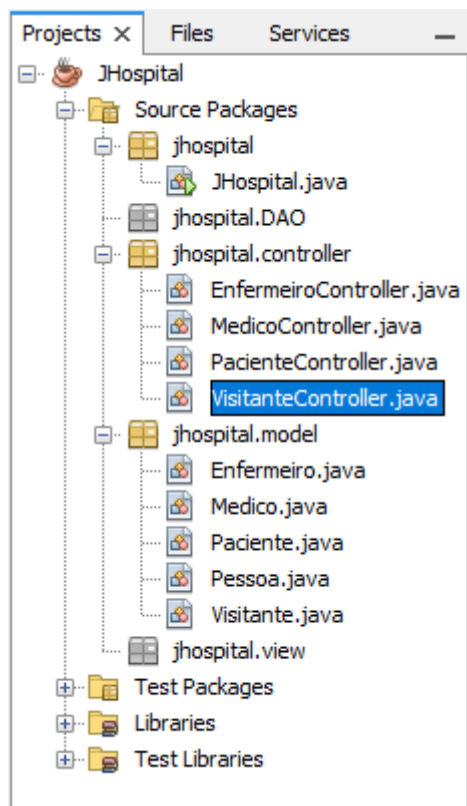


MedicoController		
Retorno	Métodos	Parâmetros
void	inserir	String nome String email String endereço String telefone String especialidade String horasMensais String valorDasHoras
void	alterar	Integer id String nome String email String endereço String telefone String especialidade String horasMensais String valorDasHoras
void	excluir	Integer id
List<Medico>	buscar	String nome String especialidade
VisitanteController		
Retorno	Métodos	Parâmetros
void	inserir	String nome String email String endereço String telefone String paciente

**Icaro Daflon**  
**JAVA MOD II**

void	alterar	Integer id String nome String email String endereço String telefone String paciente
void	excluir	Integer id
List<Visitante>	buscar	String nome String paciente

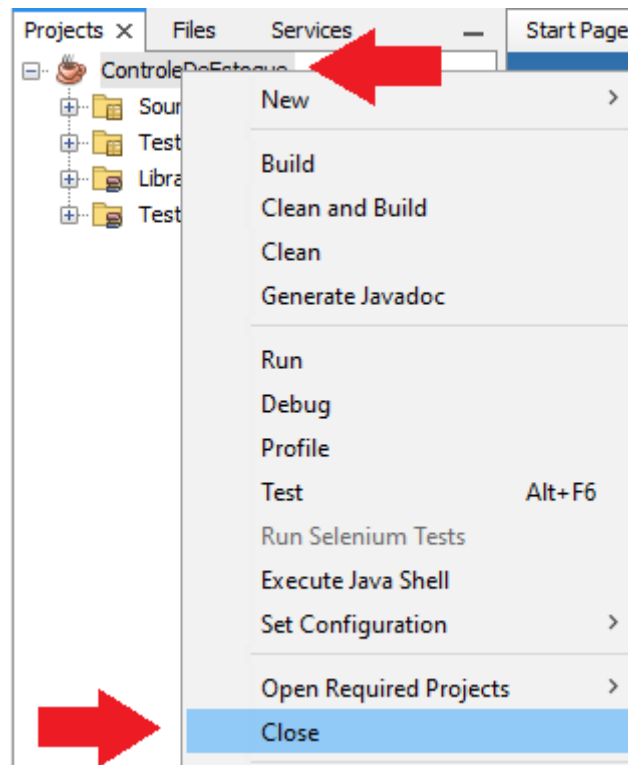
Ao término da criação das classes, a aba Projetos estará da seguinte forma:



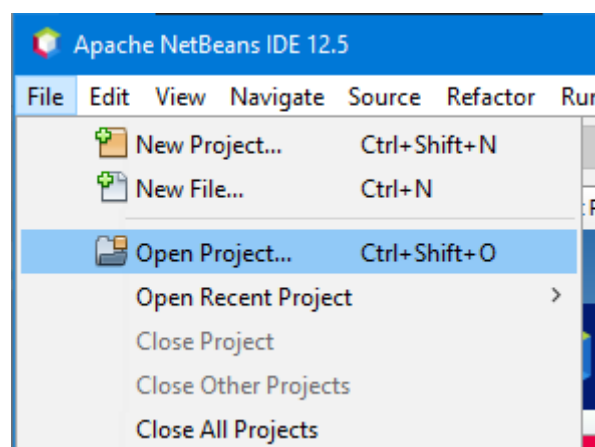
- Salve todas as alterações feitas e feche os programas abertos.

# AULA 11

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 10” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

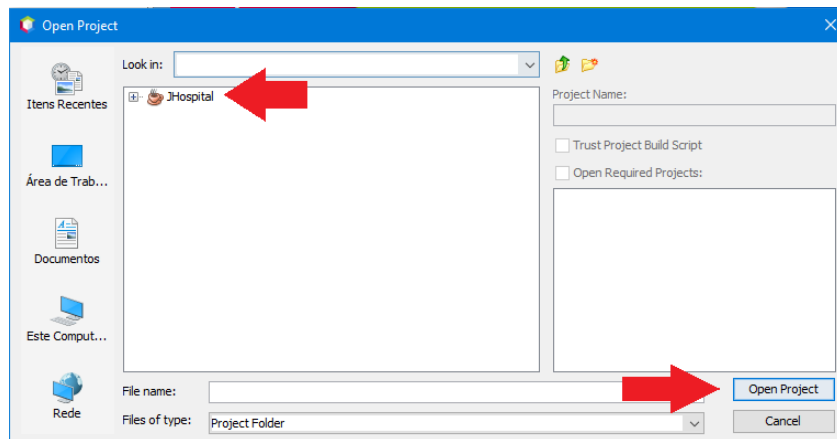


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



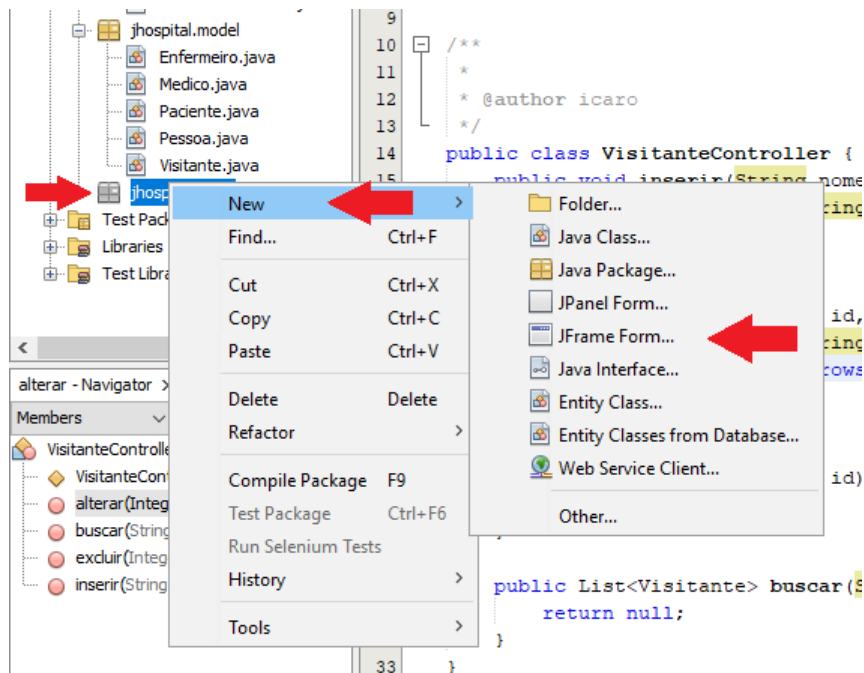
- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

## Icaro Daflon JAVA MOD II

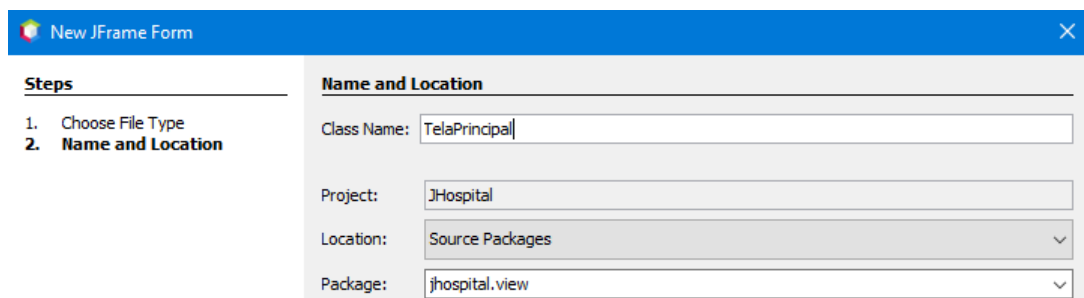


### JHospital – View, Páginas I54 a I66

- Clique com o botão direito sobre o pacote `jhospital.view`, vá em “New” e clique na opção “JFrame Form...”

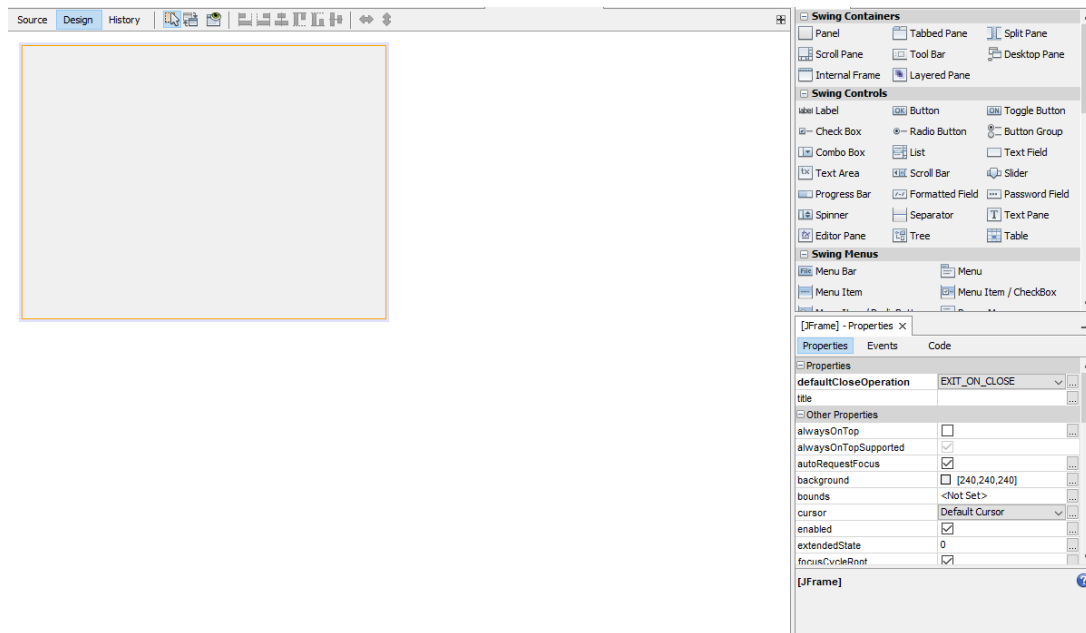


- Em “Class Name” configure com o nome “TelaPrincipal” e após, clique em “Finish”.

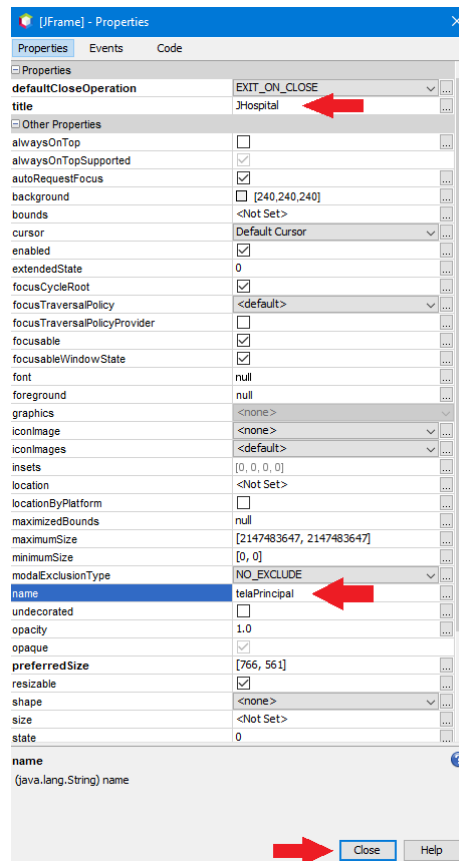


- Será aberta a tela de Edição do JFrame como abaixo:

## Icaro Daflon JAVA MOD II

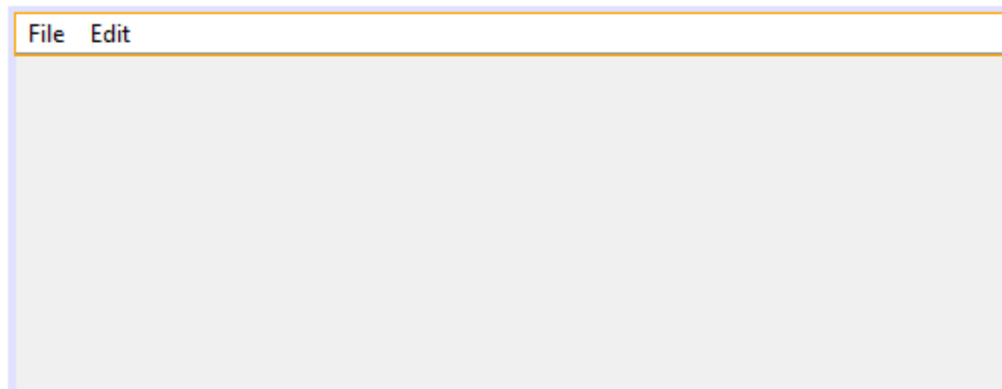
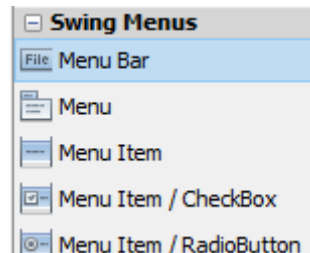


- Aumente o tamanho do formulário que foi aberto para que fique melhor a visualização.
- Agora iremos definir o nome do JFrame e também o que irá aparecer na barra de título. Isso é importante para uma melhor manipulação e exibição dos objetos.
- Para isso, clique com o botão direito em uma área livre do formulário e clique em “Properties”
- Na tela que surge, na aba Properties, atribua JHospital à opção title e na opção name, atribua telaPrincipal. Após clique em “Close”



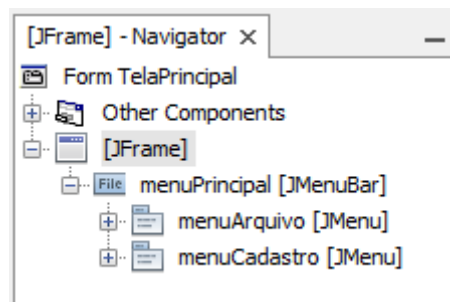
**Icaro Daflon**  
**JAVA MOD II**

- Esse formulário vai consistir em uma Barra de Menu contendo as opções desejadas no sistema e um JTabbedPane logo abaixo, onde serão abertas as abas para cada opção do menu.
- Na aba Palette, encontre a opção “Menu Bar”, e coloque essa barra no canto superior esquerdo como abaixo:



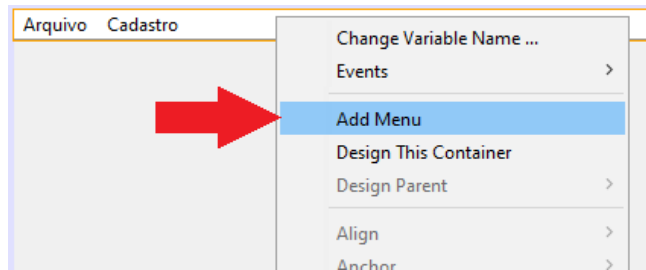
- Feito isso, agora iremos editar a barra de menu para que fique do jeito necessário. O sistema possuirá 4 menus: Arquivo, que conterà as opções referentes ao sistema em si; Cadastro que conterà os cadastros do sistema; Consulta que conterà as informações de consulta do sistema; e Ajuda que conterà as informações de ajuda do sistema. O NetBeans cria automaticamente duas opções por padrão: File e Edit. Assim, você irá editá-las e criar mais duas.
- Primeiro, mude os nomes tanto da barra de menu como dos menus criados. Renomeie a opção “Variable Name” da barra de menu de JMenuBarI para menuPrincipal.
- Altere o texto das duas opções do menu de File e Edit para Arquivo e Cadastro, respectivamente.
- Renomeie a propriedade “Variable Name” das opções do menu. A opção “Arquivo” renomeie para menuArquivo e a opção “Cadastro” renomeie para menuCadastro.

O [JFrame] – Navigator deve estar da seguinte maneira:

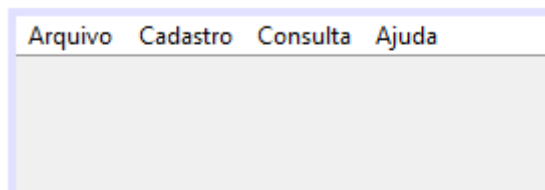


- Agora clique com o botão direito em cima do menu e escolha a opção “Add Menu”

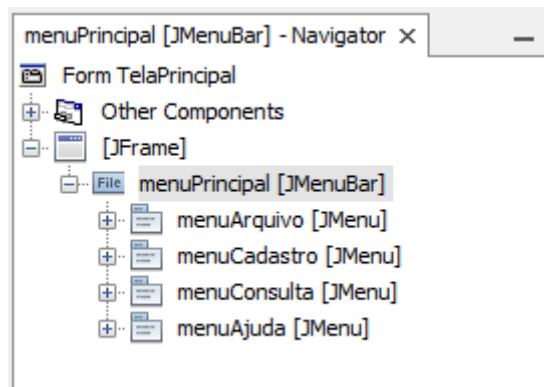
**Icaro Daflon**  
**JAVA MOD II**



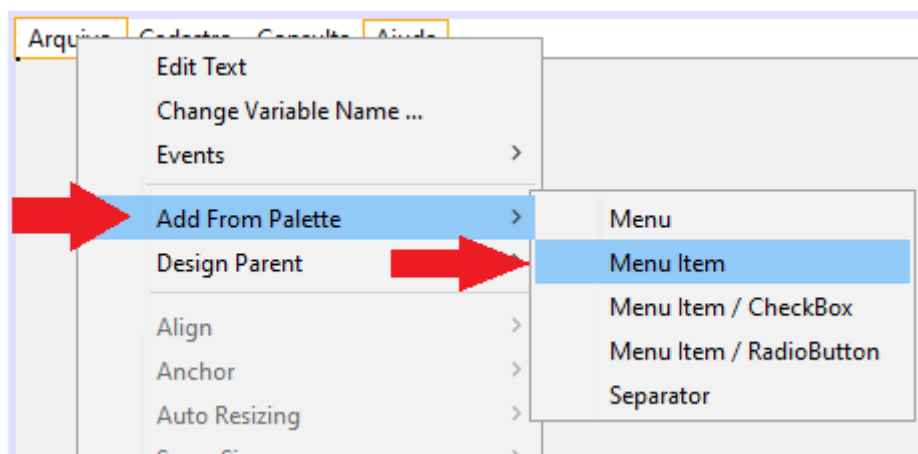
- Será adicionado um novo menu à barra de menu.
- Repita a operação para criar um novo menu, e edite o texto desses menus conforme a imagem abaixo:



- Após isso, altere o nome dos objetos, em “Variable Name”, para que fique exatamente como abaixo:



- O menu está criado. Agora iremos criar os itens de cada menu. Primeiro iremos criar o item do menu Arquivo. Por enquanto, ele possuirá somente 1 item: Sair.
- Para criá-lo, clique com o botão direito em cima do menu Arquivo, no menu que surge, escolha “Add From Palette” e em seguida, clique em “Menu Item”

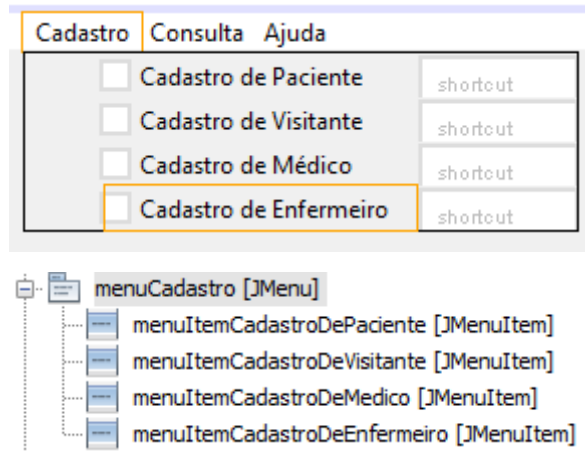


- Altere o texto do item criado para Sair e altere a propriedade “Variable Name” para menuItemSair.

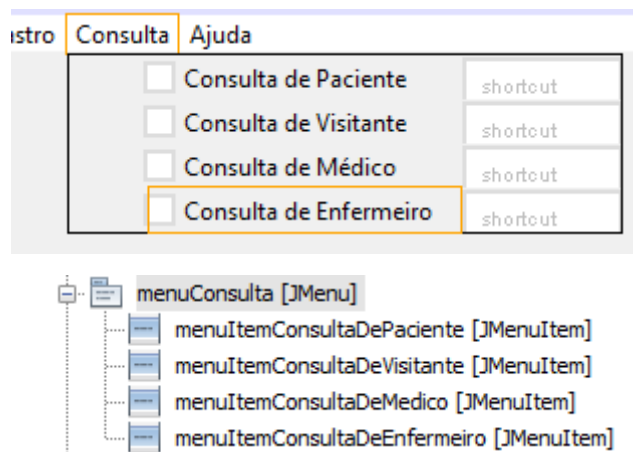
**Icaro Daflon**  
**JAVA MOD II**

- Com o item do menu Arquivo criado e configurado, agora iremos criar os itens dos outros menus. Para isso, siga as orientações abaixo:

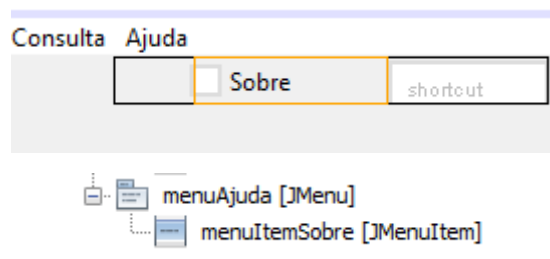
- Menu Cadastro:



- Menu Consulta:



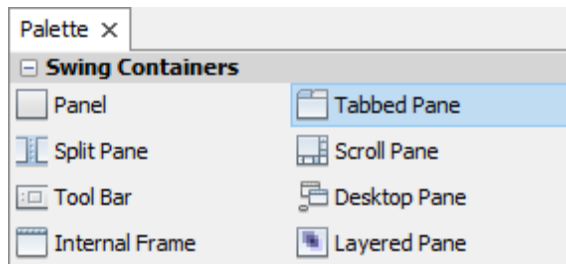
- Menu Ajuda:



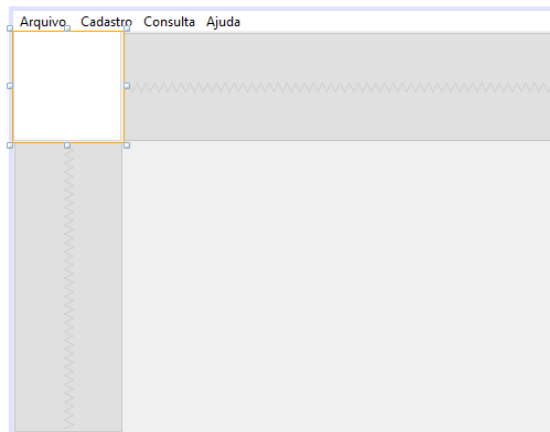
- Agora, iremos incluir um TabbedPane. Nesse painel, serão exibidas em abas, as telas de cadastro, consulta e edição.
- Na aba Palette, selecione a opção TabbedPane:



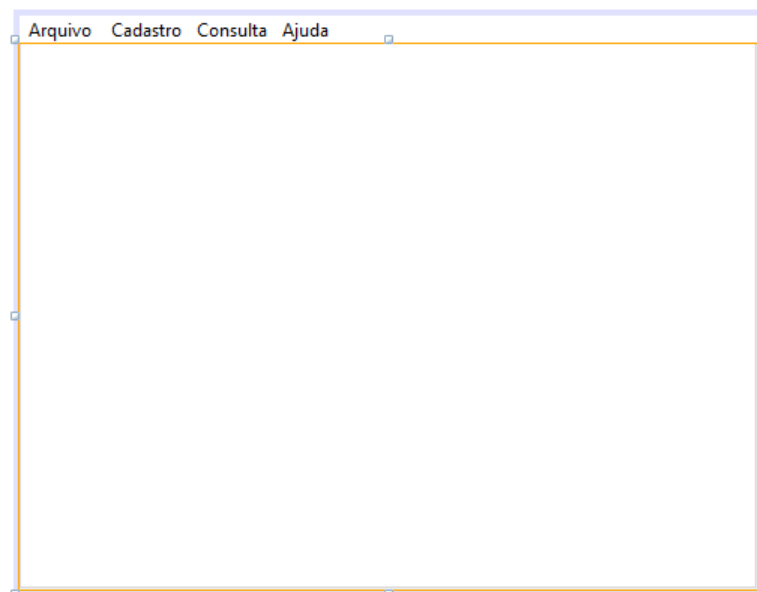
## Icaro Daflon JAVA MOD II



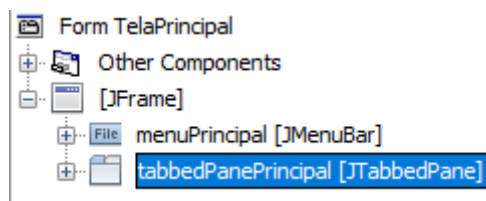
- Posicione-o como na figura abaixo:



- Aumente o formulário para que ocupe toda a área do JForm:



- Altere o “Variable Name” para tabbedPanePrincipal.



- Agora iremos programar para que a TelaPrincipal seja chamada quando o sistema for iniciado. Para isso abra a classe JHospital do pacote jhospital.

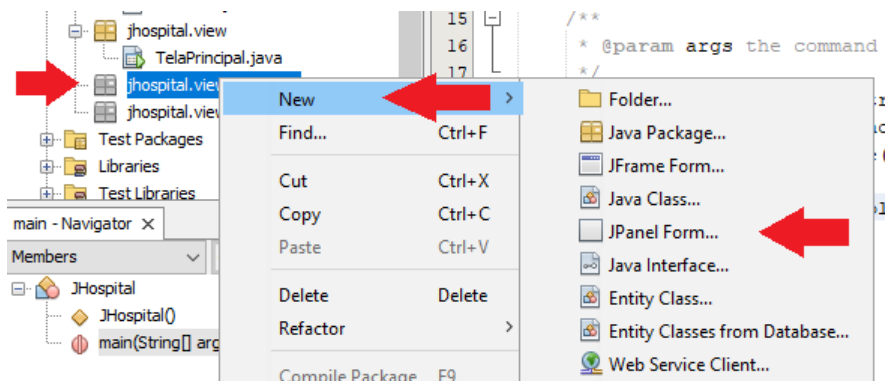
- Dentro do método main, insira o código abaixo:

```
public static void main(String[] args) {  
    TelaPrincipal telaPrincipal = new TelaPrincipal();  
    telaPrincipal.validate();  
    telaPrincipal.pack();  
    telaPrincipal.setVisible(true);  
}
```

- Execute o programa, verifique todos os menus que foram criados e após feche a janela de execução.
- Agora iremos criar as telas de cadastro e de consulta. Para isso, iremos criar um pacote para conter estas telas.
- Crie um pacote com o nome jhospital.view.cadastro
- Crie outro pacote com o nome jhospital.view.consulta

### JHospital – Cadastros View, Páginas 166 a 174.

- Primeiro iremos criar a janela de cadastro de Pacientes.
- Para isso, clique com o botão direito no pacote jhospital.view.cadastro, no menu que surge, selecione “New” e em seguida “JPanel Form...”



- Entre com o nome “CadastroDePacienteView” e clique no botão “Finish”.
- Edite o seu formulário para que fique semelhante à figura abaixo:

**Cadastro de Paciente**

Nome Completo:

Endereço:

E-mail:  Telefone:

Quarto:  Doença:

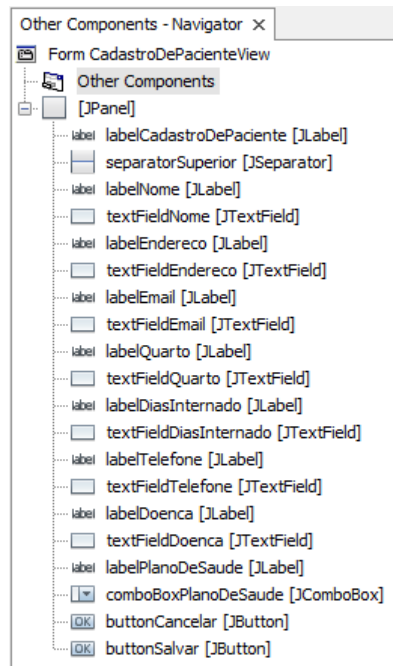
Dias Internado:  Plano de Saúde:

Salvar Cancelar

## Icaro Daflon JAVA MOD II

OBS: O combo Plano de Saúde conterá somente duas opções: “Sim” e “Não”.

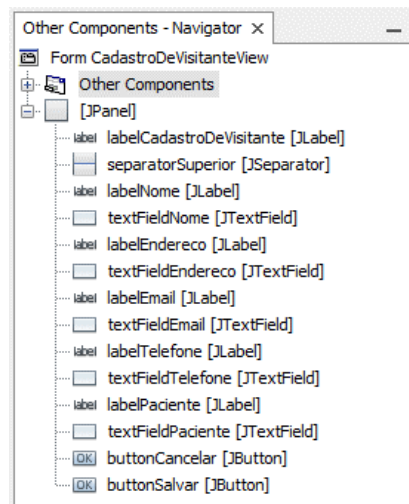
- Os componentes do formulário e seus respectivos nomes são os seguintes:



- Agora iremos criar o cadastro de Visitantes, portanto crie um “JPanel Form...” de nome CadastroDeVisitanteView no pacote jhospital.view.cadastro.
- Edite o formulário para que fique semelhante à imagem abaixo:

- A seguir, estão os componentes do formulário e seus respectivos nomes:

## Icaro Daflon JAVA MOD II



- Agora crie outro JPanel Form..., entre com o nome CadastroDeMedicoView e após clique no botão “Finish”.
- Edite o formulário para que fique como na imagem abaixo:

**Cadastro de Médico**

Nome Completo:

Endereço:

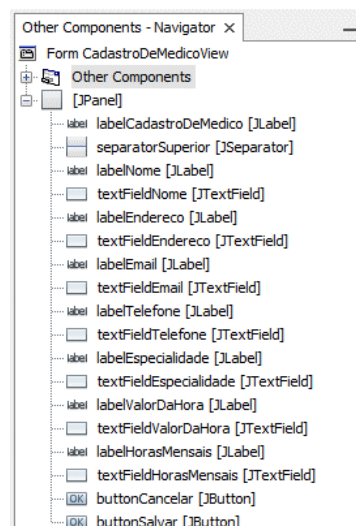
E-mail:  Telefone:

Especialidade:  Horas Mensais:

Valor da Hora:

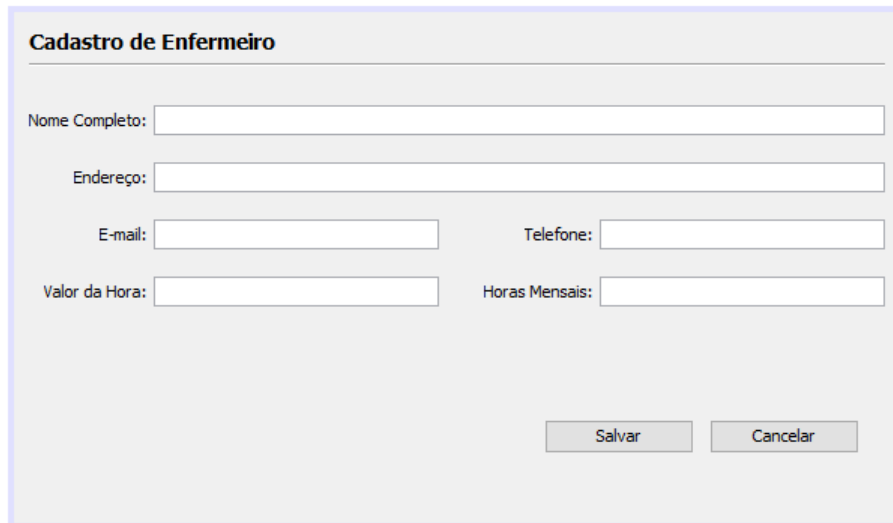
Salvar Cancelar

- Os componentes do formulário e seus respectivos nomes estão listados abaixo:

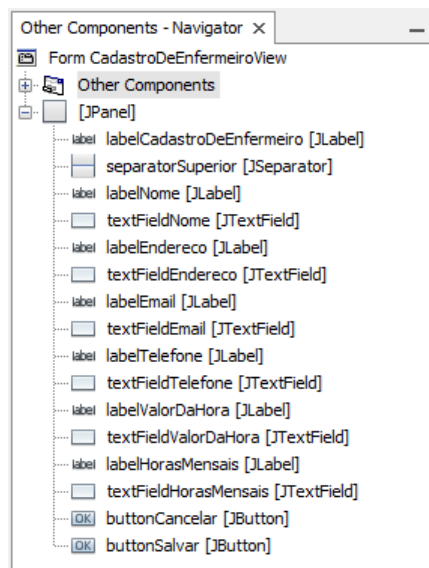


**Icaro Daflon**  
**JAVA MOD II**

- Agora crie outro JPanel Form..., entre com o nome CadastroDeEnfermeiroView e após clique no botão “Finish”.
- Edite o formulário para que fique como na imagem abaixo:



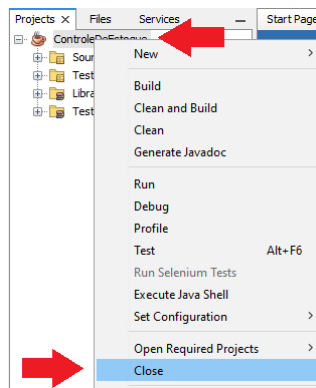
- Os componentes do formulário e seus respectivos nomes estão listados abaixo:



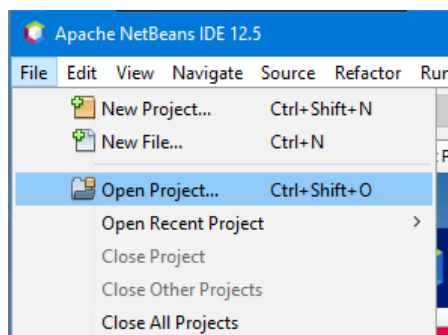
- Com isso todas nossas telas de cadastro estão prontas. Visualize o Design de cada uma para verificar como elas ficaram
- Salve o projeto e feche todos os programas que estão em execução.

# AULA 12

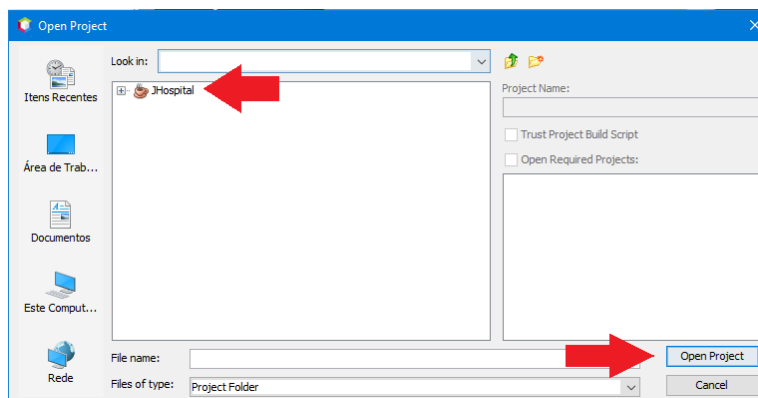
- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula II” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.



- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

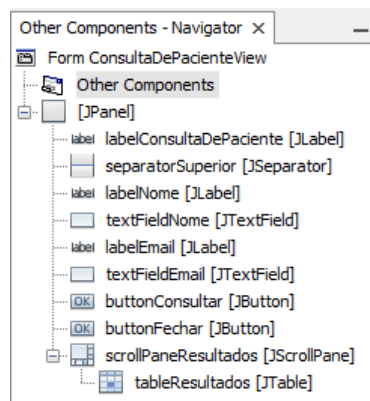


## JHospital – Consultas View, Páginas 176 a 191

- Primeiro iremos criar o painel para Consulta de Pacientes, para isso, clique com o botão direito sobre o pacote `jhospital.view.consulta` e crie o `JPanel Form...` de nome `ConsultaDePacienteView`.
- Edite o formulário para que fique como na imagem abaixo:

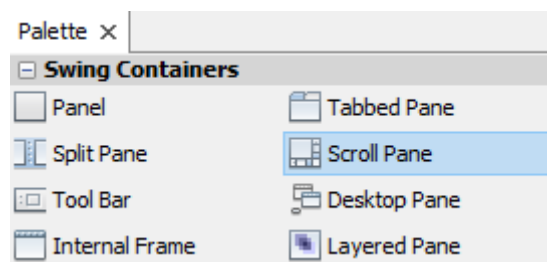
A imagem mostra uma interface gráfica de usuário para a consulta de pacientes. No topo, há um título "Consulta de Paciente". Abaixo dele, há dois campos de texto: "Nome:" e "E-mail". À direita dos campos, há dois botões: "Consultar" e "Fechar". Abaixo dos campos, há uma tabela com quatro colunas: "Nome", "Endereço", "E-mail" e "Telefone". A tabela está vazia.

- Os componentes do formulário e seus respectivos nomes estão listados abaixo:



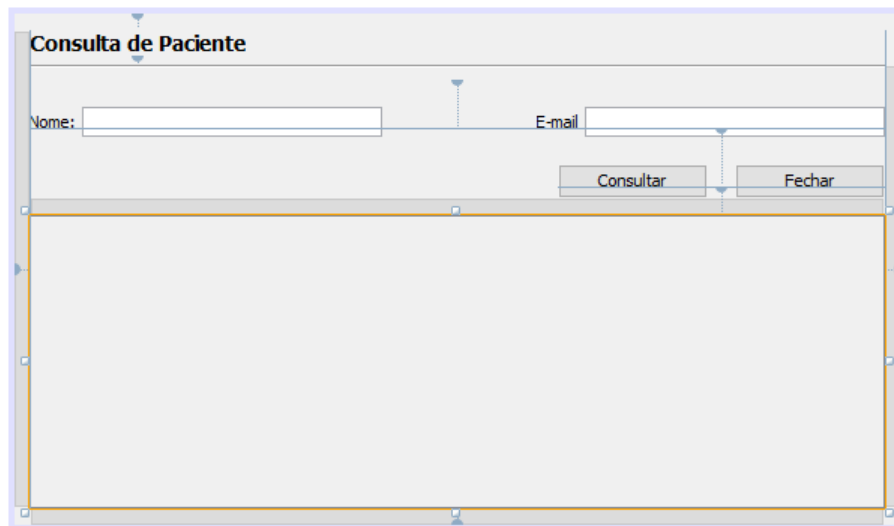
OBS: Todas as telas de consulta, possuem uma tabela para apresentação dos dados recuperados do banco de dados. Além disso, essa tabela está dentro de um Scroll Pane. Dessa maneira, é importante explicar a melhor forma de organizar uma Tabela em conjunto com um Scroll Pane de uma maneira simples e rápida.

- Para criar uma tabela, selecione na aba Palette dentro de Swing Containers a opção Scroll Pane.

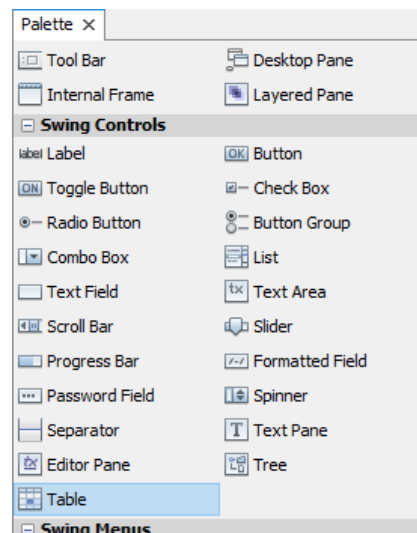


- Insira o painel de rolagem como na figura abaixo:

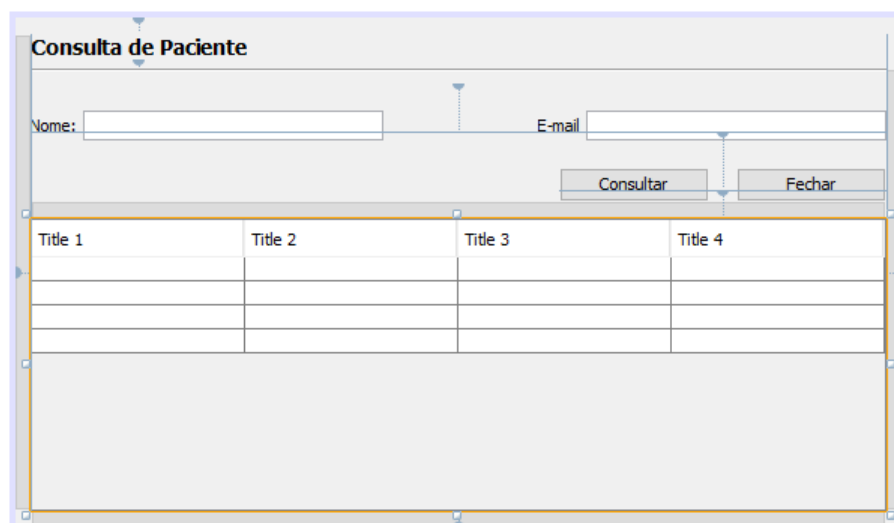
Icaro Daflon  
JAVA MOD II



- Com o painel de rolagem criado, agora você deve inserir dentro dele a tabela.
- Para criar uma tabela, selecione na aba Palette dentro do Swing Controls a opção Table.



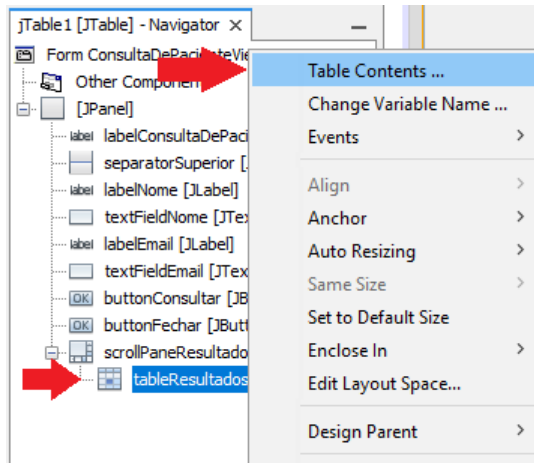
- Posicione a Tabela dentro do Scroll Pane que foi criado.



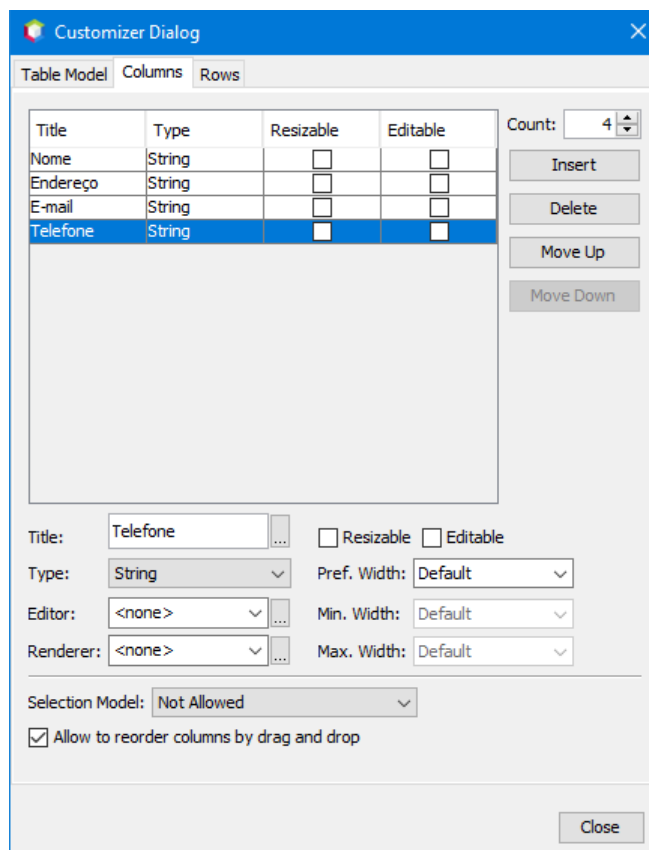


**Icaro Daflon**  
**JAVA MOD II**

- A seguir, iremos editar a tabela que foi criada.
- Na propriedade “Variable Name” do painel de rolagem defina para scrollPaneResultados.
- Na propriedade “Variable Name” da Tabela defina para tableResultados.
- Agora, clique com o botão direito sobre a tabela e no menu que surge, selecione Table Contents...

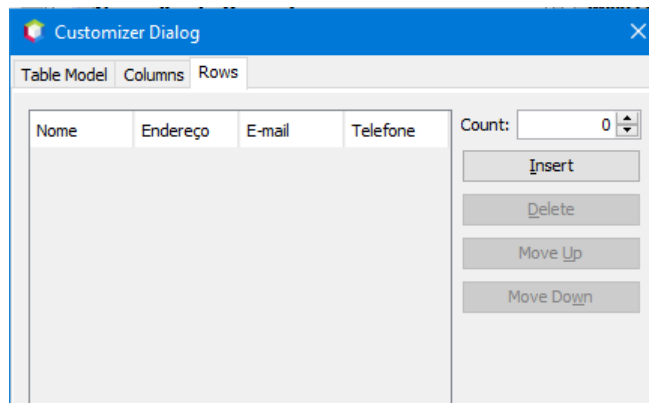


- Na janela que surge, clique na aba “Columns”.
- Edite as colunas da tabela deixando como a figura abaixo:



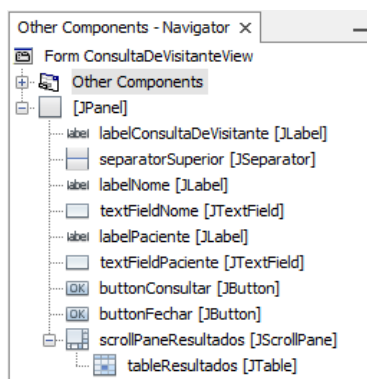
- Agora, clique sobre a aba “Rows”
- Apague todas as linhas que existem, assim como na figura abaixo e clique no botão “Close”.

## Icaro Daflon JAVA MOD II



- Pronto. Agora podemos dar continuidade ao desenvolvimento do restante das telas.
- Agora iremos criar o painel para Consulta de Visitante, para isso, clique com o botão direito sobre o pacote `jhospital.view.consulta` e crie o `JPanel Form...` de nome `ConsultaDeVisitanteView`.
- Edite o formulário para que fique como na imagem abaixo:

- Para criar uma tabela, selecione na aba Palette dentro do Swing Controls a opção Table.



- Agora iremos criar o painel para Consulta de Médico, para isso, clique com o botão direito sobre o pacote `jhospital.view.consulta` e crie o `JPanel Form...` de nome `ConsultaDeMedicoView`.
- Edite o formulário para que fique como na imagem abaixo:

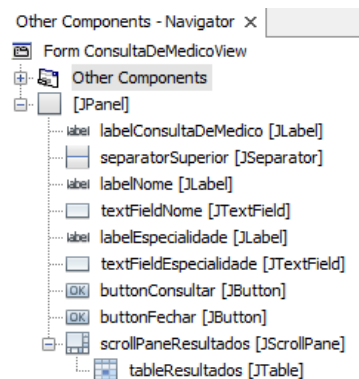
**Icaro Daflon**  
**JAVA MOD II**

**Consulta de Médico**

Nome:  Especialidade:

Nome	Especialidade	E-mail	Telefone
------	---------------	--------	----------

- Para criar uma tabela, selecione na aba Palette dentro do Swing Controls a opção Table.



- Agora iremos criar o painel para Consulta de Enfermeiro, para isso, clique com o botão direito sobre o pacote `jhospital.view.consulta` e crie o `JPanel Form...` de nome `ConsultaDeEnfermeiroView`.
- Edite o formulário para que fique como na imagem abaixo:

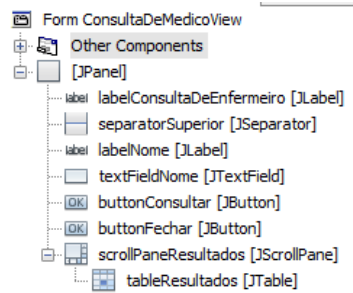
**Consulta de Enfermeiro**

Nome:

Nome	E-mail	Telefone
------	--------	----------

- Para criar uma tabela, selecione na aba Palette dentro do Swing Controls a opção Table.

## Icaro Daflon JAVA MOD II



Após criarmos todas as telas de Consulta, é preciso exibir cada uma delas dentro do TabbedPane de acordo com a opção de menu que foram escolhidas.

Agora iremos colocar as ações no itens de menu para justamente carregar no TabbedPane, cada tela de acordo com sua opção.

Antes de darmos início a criação dessas ações, vamos fazer uma pequena modificação nas nossas classes de visualização, tanto de cadastro quanto de consulta.

- Abra a classe CadastroDePacienteView, entre com o código abaixo dentro da classe:

```
13 public class CadastroDePacienteView extends javax.swing.JPanel {
14
15     private JTabbedPane tabbedPane;
16 }
```

- Iremos também modificar o construtor da classe. Atualmente ele estará da seguinte forma:

```
public CadastroDePacienteView() {
    initComponents();
}
```

- Altere o construtor para que fique dessa maneira:

```
public CadastroDePacienteView(JTabbedPane tabbedPane) {
    this.tabbedPane = tabbedPane;
    initComponents();
}
```

- Agora faça o mesmo para todas as outras classes (as 3 restantes de Cadastro e as 4 de Consulta).
- Feito isso, agora iremos trabalhar com a classe principal. Portanto abra a classe TelaPrincipal.
- Clique na opção Cadastro do menu e depois dê dois clique no item menuItemCadastroDePaciente.
- Com o código fonte da ação aberto, digite o código abaixo:

```
155 private void menuItemCadastroDePacienteActionPerformed(java.awt.event.ActionEvent evt) {
156
157     tabbedPanePrincipal.add("Cadastro de Paciente",
158         new CadastroDePacienteView(tabbedPanePrincipal));
159     tabbedPanePrincipal.revalidate();
160     tabbedPanePrincipal.repaint();
161     tabbedPanePrincipal.setSelectedIndex(
162         tabbedPanePrincipal.getSelectedIndex()+1);
163 }
```

- Esse código diz que todas as vezes que o item for acionado no menu será adicionada uma nova aba no painel tabulado da tela principal.

- Execute o programa para verificar se a tela de cadastro de pacientes será exibida quando acionado o item Cadastro de Paciente.

OBS: Caso a tela de Cadastro esteja sendo encoberta pela tela principal, vá até o formulário TelaPrincipal e aumente seu tamanho até deixá-lo no tamanho ideal.

- Agora, faça o mesmo para as outras opções do menu. Lembre-se de incluir o código que abre a view de acordo com o formulário que você deseja abrir.
- Para finalizar os itens do menu, iremos colocar as ações para a opção Sair, localizada em Arquivo e para a opção Sobre, localizada em Ajuda.
- Para isso, dê dois cliques sobre o menu menuItemSair e digite o código abaixo:

```
233 private void menuItemSairActionPerformed(java.awt.event.ActionEvent evt) {  
235     if (JOptionPane.showConfirmDialog(this,  
236         "Deseja mesmo fechar o sistema?", "Confirmação",  
237         JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
238         System.exit(0);  
239     }  
240 }
```

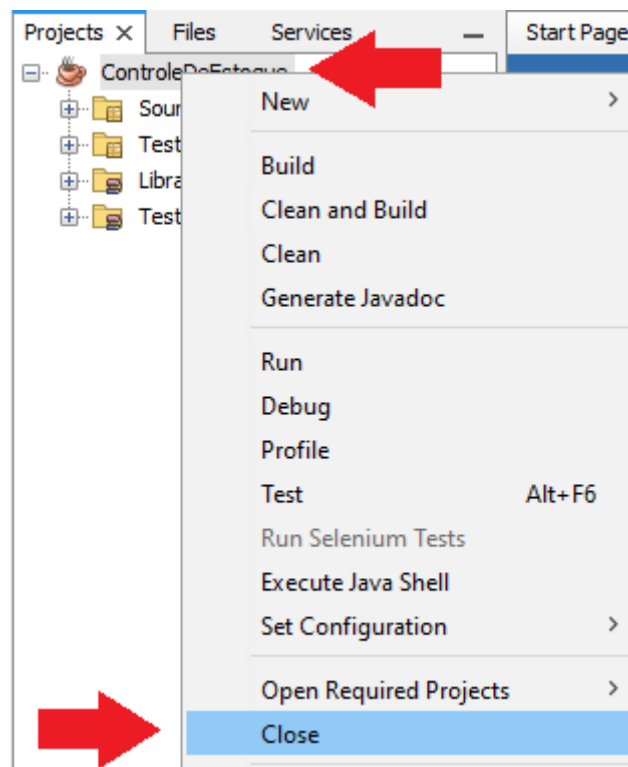
- Este código faz o sistema perguntar para o usuário se ele realmente deseja sair do sistema e, caso seja escolhida a opção Sim, o sistema será fechado.
- Agora, dê dois cliques no menu menuItemSobre e digite o código abaixo:

```
246 private void menuItemSobreActionPerformed(java.awt.event.ActionEvent evt) {  
248     JOptionPane.showMessageDialog(this,  
249         "JHospital\n\nVersão 1.0\nCopyright: Icaro",  
250         "Sobre", JOptionPane.INFORMATION_MESSAGE);  
251 }
```

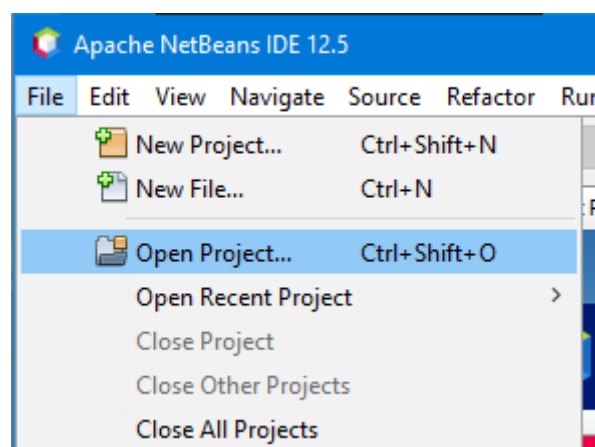
- O código acima exibe uma mensagem na tela com as informações do sistema.
- A aula foi finalizada. Salve e feche todos os programas abertos.

# AULA 13

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 12” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

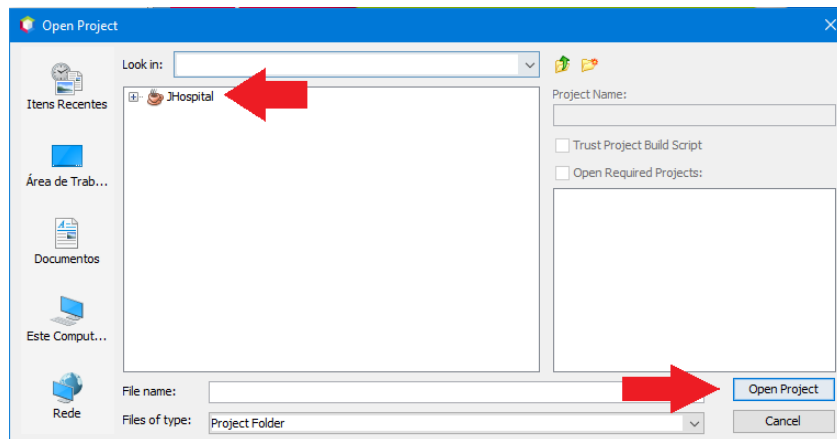


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

## Icaro Daflon JAVA MOD II



- Com o projeto aberto, adicione a biblioteca do PostgreSQL (PostgreSQL JDBC Driver)
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL 14
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

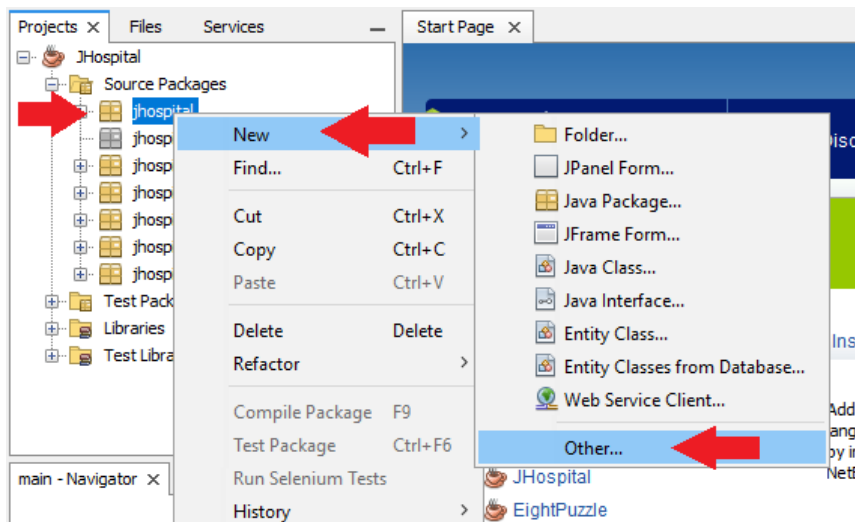
OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

Hibernate + JPA será trocado pelo EclipseLink nativo do NetBeans 15 a configuração mantém-se a mesma.

EclipseLink + JPA no JHospital, Páginas 196 a 202

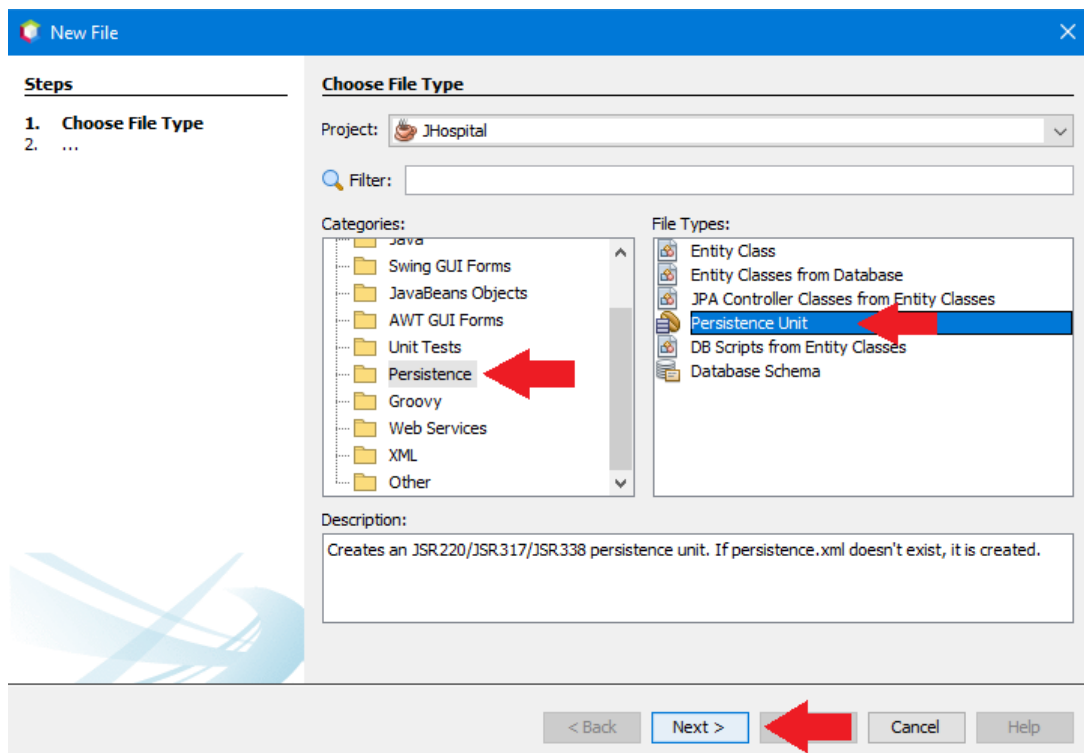
- Clique com o botão direito no projeto e no menu que surge selecione “New”. Agora verifique se a opção “Persistence Unite...” está disponível, caso não esteja, selecione a opção “Other...”

## Icaro Daflon JAVA MOD II



OBS: Caso a opção “Persistence Unit...” já esteja disponível nesse menu, acesse-o diretamente.

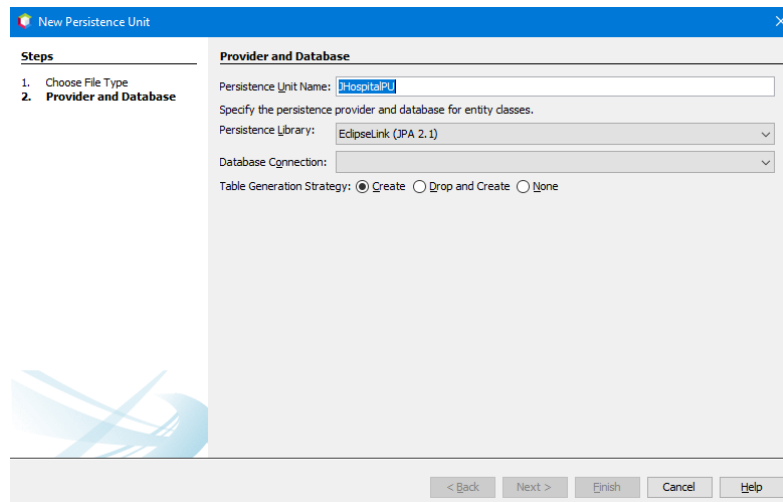
- Com a janela “New File” aberta, selecione “Persistence”, em seguida, selecione “Persistence Unit” e após, clique em “Next”.



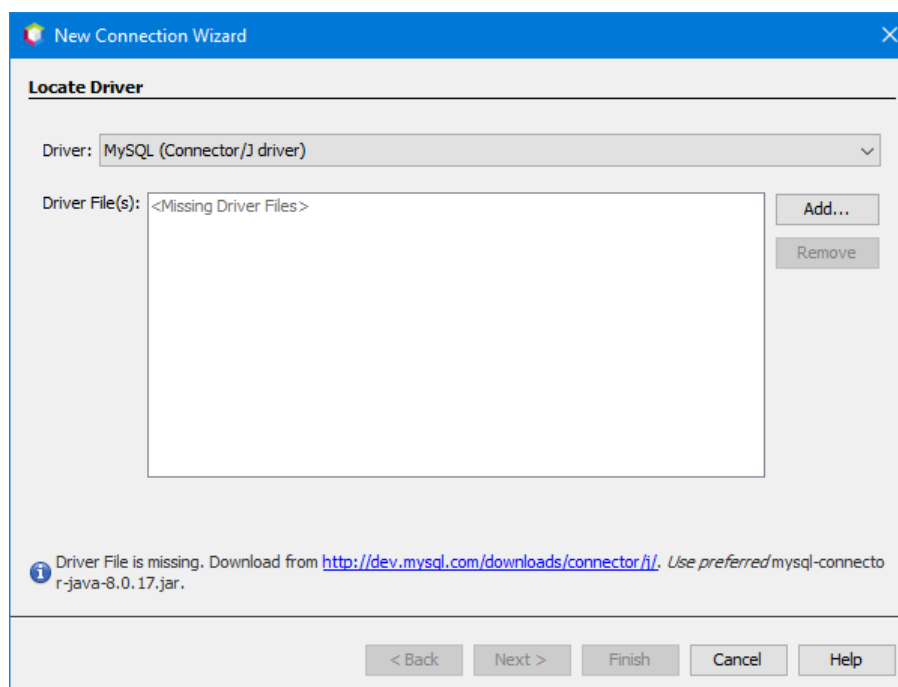
- A janela “New Persistence Unit” passa a ser exibida.



## Icaro Daflon JAVA MOD II



- O nome não será alterado, mas lembre-se dele pois será utilizado para recuperar a conexão posteriormente.
- No combo “Persistence Library” mantenha selecionado a opção “EclipseLink (JPA 2.1)”
- No combo “Database Connection” é necessário identificar o banco de dados que será utilizado e qual usuário e senha que fará a conexão. É a mesma coisa que fizemos na nossa classe Conexão.java, do projeto anterior. Porém, agora, de uma maneira mais simples. Portanto nesse combo selecione a opção “New Database Connection...”
- Surgirá a janela a seguir:



- Aqui, iremos selecionar o Driver do banco de dados. Portanto, no combro “Driver” selecione a opção PostgreSQL e em seguida clique em “Next”.
- Preencha a janela como na imagem abaixo:

Icaro Daflon  
JAVA MOD II

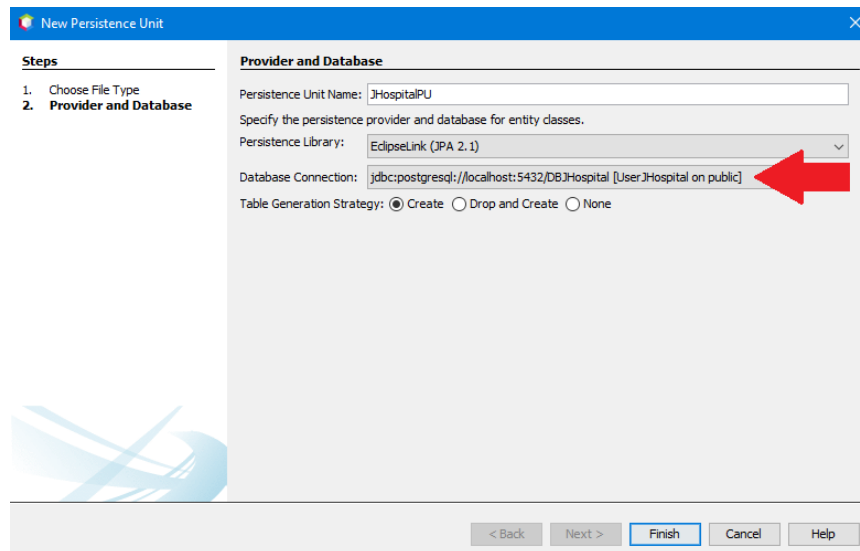
The screenshot shows the 'New Connection Wizard' dialog box, specifically the 'Customize Connection' step. The 'Driver Name' is set to 'PostgreSQL'. The 'Host' is 'localhost' and the 'Port' is '5432'. The 'Database' is 'DBJHospital'. The 'User Name' is 'UserJHospital' and the 'Password' is masked with dots. A red box highlights the password field with the text '123465'. The 'Remember password' checkbox is checked. The 'JDBC URL' is 'jdbc:postgresql://localhost:5432/DBJHospital'. The 'Next >' button is highlighted with a red arrow.

- Agora, clique no botão “Test Connection” e verifique se a conexão foi bem-sucedida, após clique no botão “Next”.
- Na janela que surge, selecione o esquema de banco de dados “public”. A última tela da janela “New Connection Wizard” não será alterada pois apenas possibilita alteração do código para realizar a conexão. Portanto, em seguida clique em “Finish”.

The screenshot shows the 'New Connection Wizard' dialog box, specifically the 'Choose Database Schema' step. The text reads: 'For each database connection, the Services window only displays objects from one database schema. Select the schema of the tables to be displayed.' The 'Select schema' dropdown menu is set to 'public'. The 'Finish' button is highlighted with a red arrow.

- Veja que ao finalizar a criação da nova conexão, ela ficará selecionada no combo “Database Connection”.

## Icaro Daflon JAVA MOD II

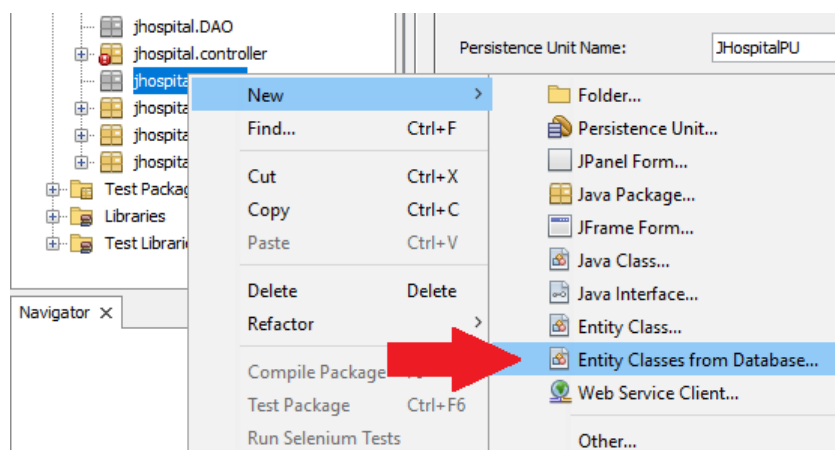


- Com tudo devidamente configurado, clique no botão “Finish”.

Note que foi criado o pacote com nome “META-INF” e dentro desse pacote foi criado um arquivo com nome “persistence.xml”. Esse arquivo é responsável pela conexão e pelo mapeamento de todas as entidades.

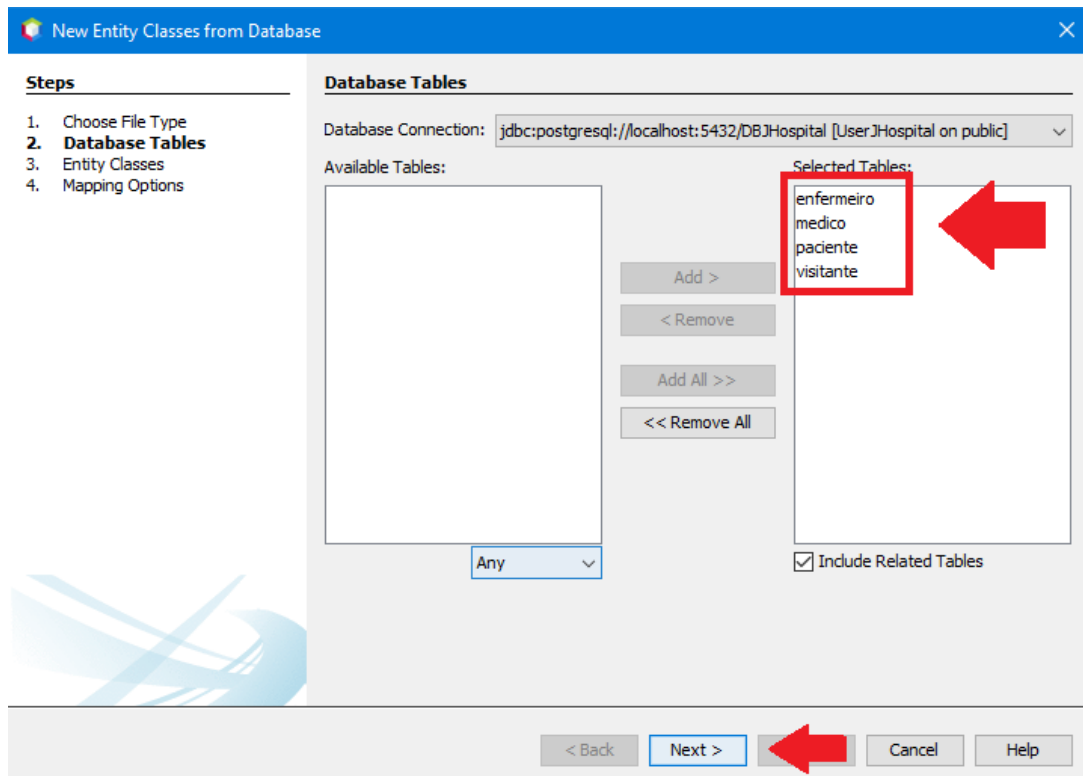
### JHospital – Classe Model (Entity), Páginas 202 a 205

- Agora iremos criar as classes de entidade. Essas classes serão representações do banco de dados no sistema e por isso iremos criá-las no pacote `jhospital.model`, mantendo o padrão MVC e substituindo as classes anteriormente criadas. Isso porque o NetBeans cria automaticamente os métodos mais importantes da classe assim como suas anotações necessários.
- Com isso, delete as classes criadas no pacote `jhospital.model`
- Em seguida, clique com o botão direito no pacote, selecione a opção “New” e após, clique na opção “Entity Class from Database...”

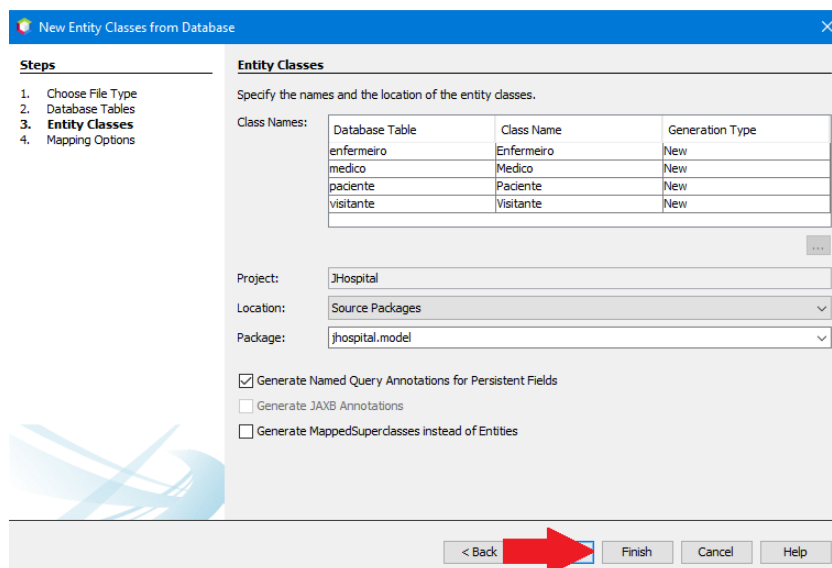


- Na janela que surge, devemos selecionar as tabelas que fazem parte do sistema, ou seja, as tabelas “enfermeiro”, “medico”, “paciente” e “visitante”.

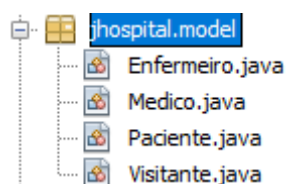
## Icaro Daflon JAVA MOD II



- Observe que a janela surge, selecione todas as tabelas, e após clique em “Next”
- Agora para finalizar clique em “Finish” para confirmar os dados.

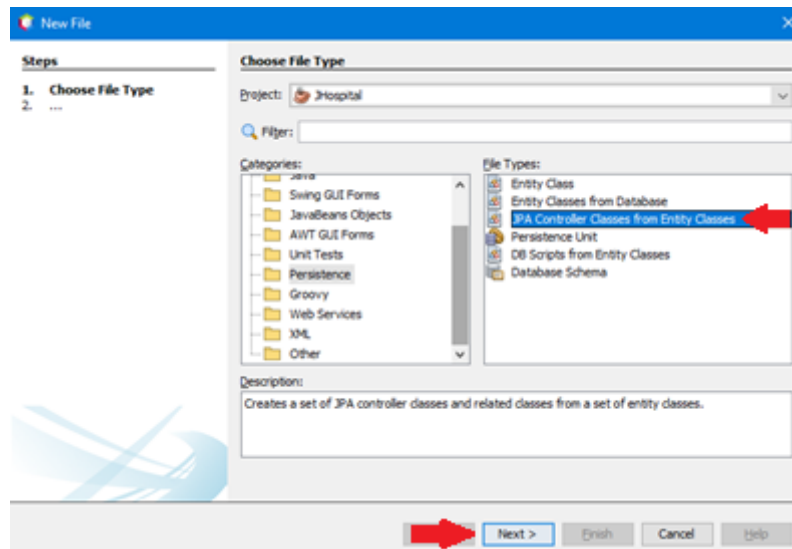


- As entidades foram criadas e agora você já pode trabalhar diretamente com os objetos. O pacote jhospital.model deve estar da seguinte maneira:

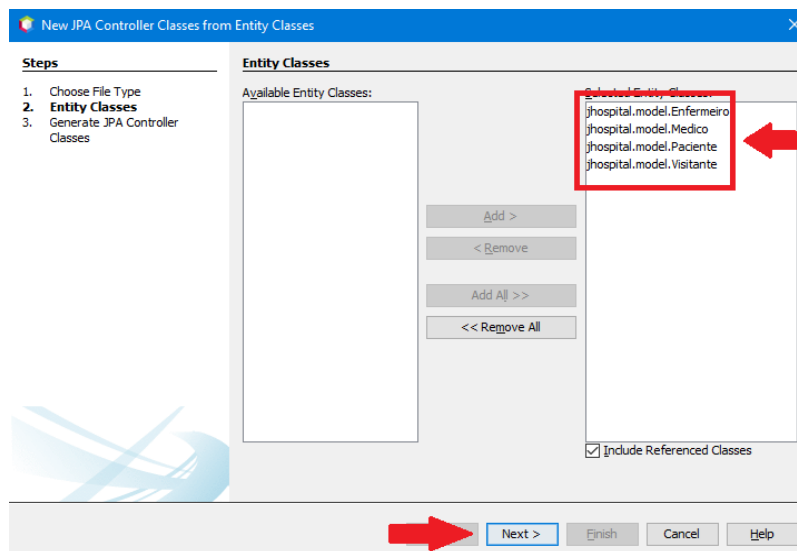


## JHospital – Classes DAO, Páginas 205 a 210

- Com as entidades criadas, agora vamos criar as classes DAO para manipular esses objetos. Como dito anteriormente, as DAO serão criadas automaticamente pelo NetBeans também, a partir das Entidades.
- Clique com o botão direito sobre o pacote `jhospital.DAO`, no menu que surge selecione “New” e verifique se a opção “JPA Controller Classes from Entity Classes” está disponível. Caso não esteja, clique em “Other...”, Selecione “Persistence” e então selecione a classe “JPA Controller Classes from Entity Classes” assim como feito anteriormente para criar a “Persistence Unit”, e Após clique em “Next”.

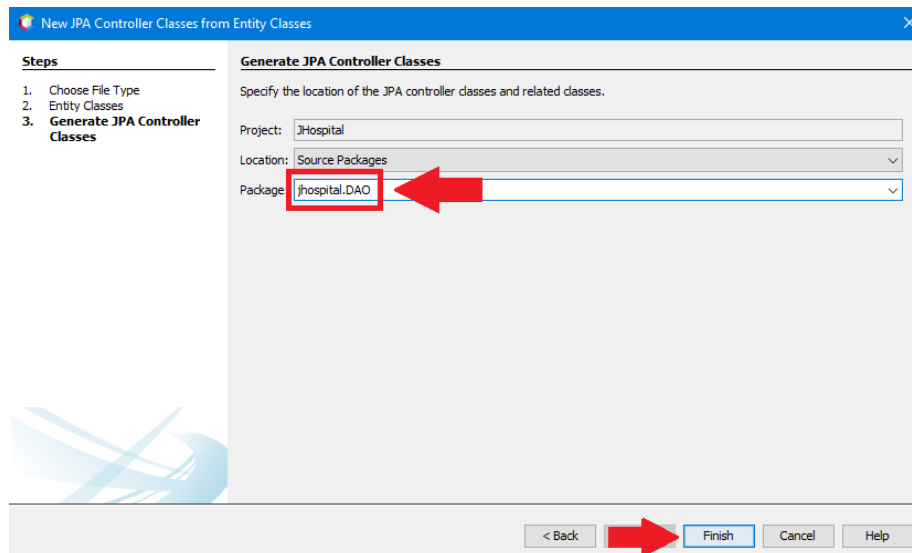


- Na janela que surge, selecione todas as classes e clique no botão “Next”.

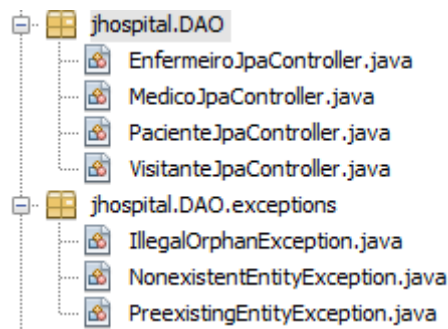


- Agora, no campo “Package” escreva `jhospital.DAO`, e após clique em “Finish”.

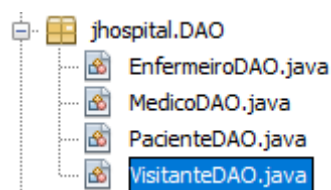
## Icaro Daflon JAVA MOD II



- Perceba que foram criadas as classes que farão a manipulação do banco de dados por meio das entidades (jhospital.model) e um pacote contendo as Exceptions que podem ser executadas em tempo de execução.



- Todas as classes foram criadas com o nome da tabela, mais o texto “JpaController” no final. Não há problema nenhum em utilizar este nome, mas para manter o padrão, iremos alterar o nome de todas para o nome da tabela, mais o texto DAO no final, como na figura abaixo:



OBS: Não esqueça de refatorar todas as classes após renomeá-las (Botão Refactor)

- Com as classes renomeadas e refatoradas, já está quase tudo pronto. Faltava apenas criar uma classe que irá efetivamente pegar a conexão feita pelo JPA e retornar para as classes DAO, para que ela faça todo o trabalho. E também fazer uma pequena alteração na classe DAO, para deixar mais simples a manipulação da mesma.
- Clique com o botão direito no pacote jhospital.DAO, posicione o cursor sobre “New” e clique em “Java Class...”
- Dê o nome da classe de ConexaoJPA.
- Essa classe conterá somente um método estático que retornará o EntityManager. Crie o método conforme a imagem abaixo:

```
public static EntityManager getEntityManager() {  
    EntityManagerFactory emf = Persistence.  
        createEntityManagerFactory("JHospitalPU");  
    EntityManager em = emf.createEntityManager();  
    return em;  
}
```

- Esse método vai criar uma conexão com a unidade de persistência criada e irá retornar o EntityManager para manipulação.
- Agora iremos fazer as modificações nas classes DAO. Para isso abra a classe PacienteDAO.

```
public PacienteDAO(EntityManagerFactory emf) {  
    this.emf = emf;  
}  
private EntityManagerFactory emf = null;  
  
public EntityManager getEntityManager() {  
    return emf.createEntityManager();  
}
```

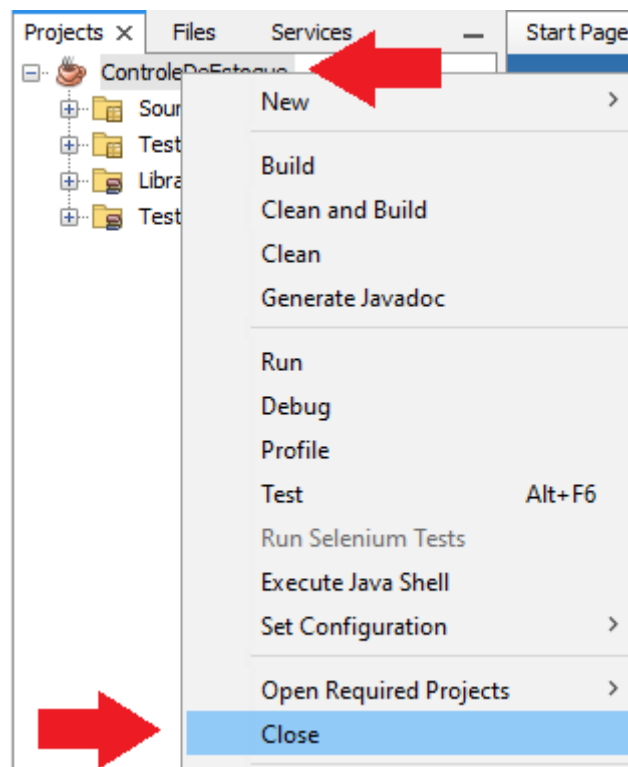
- No código acima, apague o construtor da classe e o objeto global emf, após modifique o código do método getEntityManager para que fique como na figura abaixo:

```
public EntityManager getEntityManager() {  
    return ConexaoJPA.getEntityManager();  
}
```

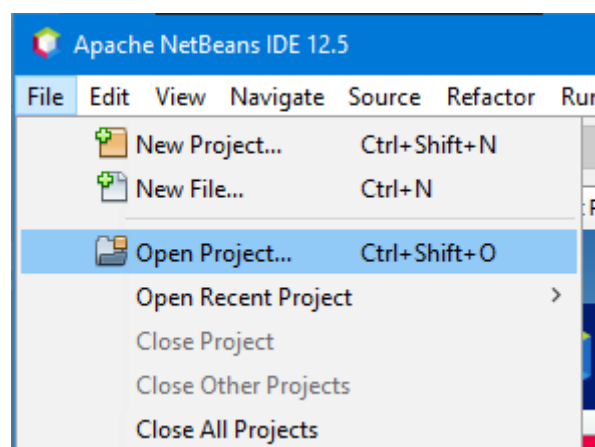
- Faça o mesmo para todas as outras classes DAO.
- Esta aula foi finalizada, salve e feche todos os programas abertos.

# AULA 14

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula I3” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.



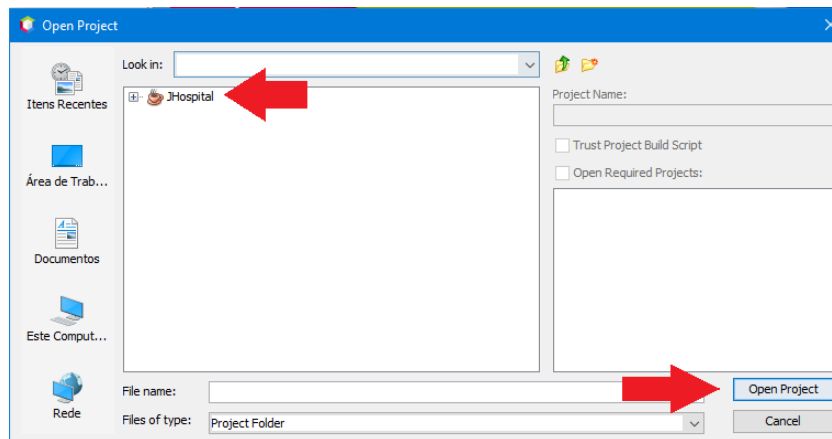
- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



## Icaro Daflon JAVA MOD II



- Com o projeto aberto, adicione a biblioteca do PostgreSQL (PostgreSQL JDBC Driver)
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL 14
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

### JHospital – Regras de Negócio – Cadastro, Páginas 213 a 219

- Após abrir o projeto e configurar o banco de dados, volte para o NetBeans e abra a classe CadastroDePacienteView.
- Abra a guia “Design” caso não esteja aberta e dê dois cliques no botão “Cancelar”.
- Digite o código abaixo para fechar a aba quando o botão cancelar for acionado:

```
if (JOptionPane.showConfirmDialog(this,  
    "Deseja mesmo fechar o cadastro de cliente sem salvar?",  
    "Confirmação", JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {  
    tabbedPane.remove(this);  
    tabbedPane.revalidate();  
    tabbedPane.repaint();  
}
```

- Faça o mesmo para os outros três formulários de cadastro. Lembre-se de modificar o texto.
- Agora, abra novamente a classe CadastroDePacienteView e dentro do código fonte da classe crie o método:

```
private void limparDados() {  
    textFieldNome.setText(null);  
    textFieldEmail.setText(null);  
    textFieldEndereco.setText(null);  
    textFieldTelefone.setText(null);  
    textFieldQuarto.setText(null);  
    textFieldDoenca.setText(null);  
    textFieldDiasInternado.setText(null);  
    comboBoxPlanoDeSaude.setSelectedIndex(0);  
}
```

- Volte para o projeto e dê dois cliques no botão Salvar. Dentro do método da ação digite o seguinte código:

```
if (textFieldNome != null &&  
    !textFieldNome.getText().equals("") &&  
    textFieldEmail != null &&  
    !textFieldEmail.getText().equals("") &&  
    textFieldEndereco != null &&  
    !textFieldEndereco.getText().equals("") &&  
    textFieldDoenca != null &&  
    !textFieldDoenca.getText().equals("")) {  
    PacienteController pc = new PacienteController();  
    try {  
        pc.inserir(textFieldNome.getText(),  
            textFieldEmail.getText(),  
            textFieldEndereco.getText(),  
            textFieldTelefone.getText(),  
            textFieldQuarto.getText(),  
            textFieldDoenca.getText(),  
            textFieldDiasInternado.getText(),  
            (String) comboBoxPlanoDeSaude.getSelectedItem());  
        JOptionPane.showMessageDialog(this,  
            "Paciente salvo com sucesso!",  
            "Sucesso", JOptionPane.INFORMATION_MESSAGE);  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(this,  
            "Não foi possível salvar o contato\n"  
            + e.getLocalizedMessage(),  
            "Erro", JOptionPane.ERROR_MESSAGE);  
    }  
} else {  
    JOptionPane.showMessageDialog(this,  
        "O nome, email, endereço e doença do "  
        + "paciente são campos obrigatórios!",  
        "Erro", JOptionPane.ERROR_MESSAGE);  
}  
limparDados();
```

- Repare que a classe PacienteController foi instanciada e foi chamado o método inserir, passando os parâmetros para que esse método grave as informações no banco. Mas, recordando, somente foi definido os métodos dentro das classes de controle. Ou seja, se você executar o sistema nesse momento e clicar no botão “Salvar” da tela de cadastro de paciente, nada irá acontecer. É necessário fazer com que as classes de controle conversem com as DAO. Inicie pela classe de Cadastro de Paciente.
- Para isso, abra a classe PacienteController
- Preencha o método inserir para que a gravação dos dados possa ser feita. Faça com que a tela de visualização converse com a classe de controle. Dentro do método, entre com o código abaixo:

## Icaro Daflon JAVA MOD II

```
Paciente paciente = new Paciente();
paciente.setNome(nome);
paciente.setEmail(email);
paciente.setTelefone(telefone);
paciente.setEndereco(endereco);
paciente.setNumerodoquarto(Integer.parseInt(quarto));
paciente.setDoenca(doenca);
paciente.setDiasdeinternacao(Integer.parseInt(diasInternado));
if(temPlanoDeSaude.equalsIgnoreCase("sim")){
    paciente.setTemplanodesaude(true);
}else{
    paciente.setTemplanodesaude(false);
}
new PacienteDAO().create(paciente);
```

- A classe de controle está realizando sua função quando o assunto é inserir um dado. Ela está montando o objeto que será inserido no banco e chamando a classe DAO referente para fazer essa inserção.

Mas o código de cadastro de paciente ainda não está funcional. Na primeira aula sobre JHospital, foram criadas as tabelas do banco de dados e também as sequências referentes a cada tabela. Essas sequências servem justamente para gerar o id das tabelas automaticamente. Para isso iremos fazer uma pequena alteração nas classes modelo (entidades) para que ela possa usar essa sequência para gerar o id automaticamente.

- Para isso, abra a classe Paciente dentro do pacote jhospital.model
- Entre com o código abaixo imediatamente acima da declaração da classe:

```
@SequenceGenerator(name = "SEQ_PACIENTE",
    sequenceName = "paciente_seq", allocationSize = 20)
```

- Faça o mesmo para as outras três classes (Enfermeiro, Medico e Visitante) e não esqueça de alterar o campo “sequenceName” respectivamente para a sequência de cada uma das classes.
- Agora, na variável id, adicione em frente ao @Id o código:

```
@Id @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_PACIENTE")
```

- A classe ficará da seguinte maneira:

```
36 @SequenceGenerator(name = "SEQ_PACIENTE",
37     sequenceName = "paciente_seq", allocationSize = 20)
38 public class Paciente implements Serializable {
39
40     private static final long serialVersionUID = 1L;
41     @Id @GeneratedValue(strategy = GenerationType.AUTO, generator = "SEQ_PACIENTE")
42     @Basic(optional = false)
43     @Column(name = "id")
```

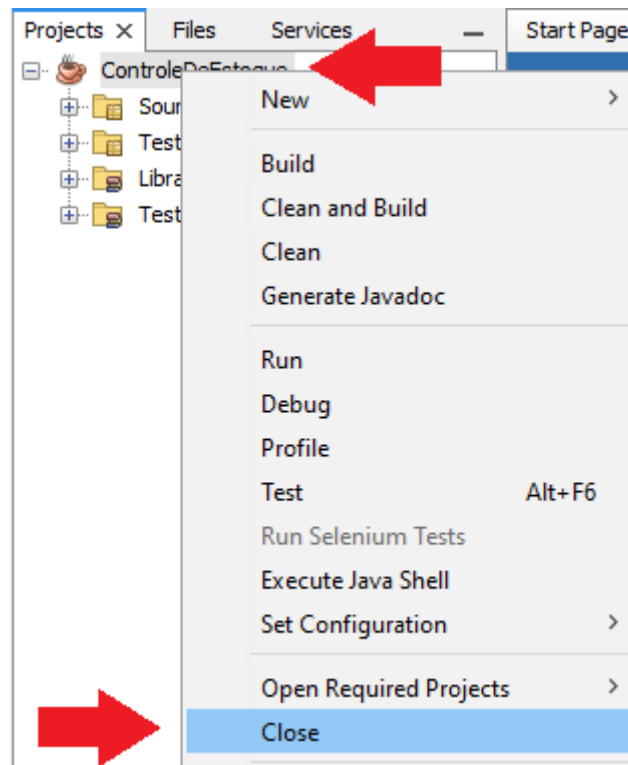
- Novamente, faça o mesmo para as outras três classes.

Com isso, o sistema já consegue cadastrar um cliente.

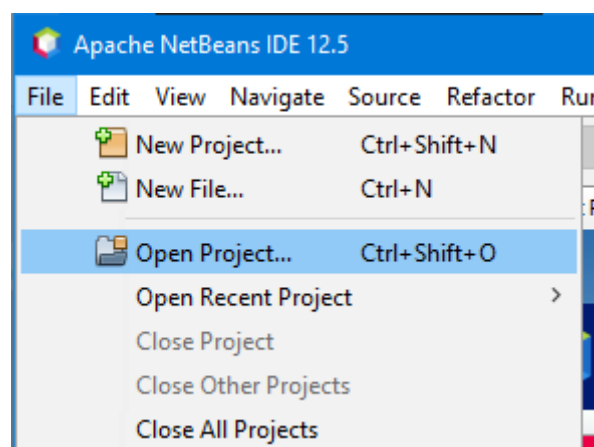
- Execute o programa e teste os botões Salvar e Cancelar do menu Cadastro de Paciente.
- Agora devemos fazer tudo que foi feito no cadastro de cliente para os outros cadastros
- Lembre-se que cada classe possui sua classe de controle específica e parâmetros e atributos diferentes que devem ser modificados na inserção do código.
- Após tudo pronto, salve as alterações e feche o programa.

# AULA 15

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 14” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

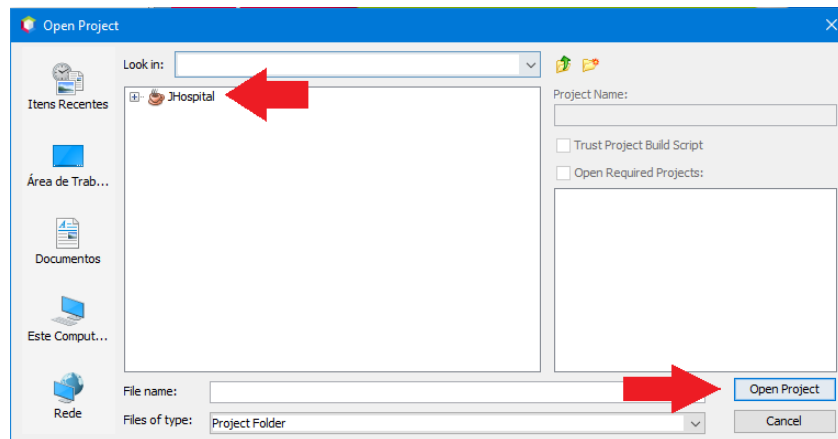


- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

## Icaro Daflon JAVA MOD II



- Com o projeto aberto, adicione a biblioteca do PostgreSQL (PostgreSQL JDBC Driver)
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL 14
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

### JHospital – Regras de Negócio – Consulta, Páginas 221 a 229

- Com o projeto aberto e o banco de dados configurado, volte para o NetBeans e abra a classe ConsultaDePacienteView.
- Na aba “Design” dê um duplo clique no botão fechar e após, digite o código abaixo:

```
tabbedPane.remove(this);  
tabbedPane.validate();  
tabbedPane.repaint();
```

- Já é possível fechar a consulta de paciente clicando no botão Fechar. Volte para a aba “Design” e dê um duplo clique no botão Consultar.
- Digite o código a seguir:

## Icaro Daflon JAVA MOD II

```
PacienteController pc = new PacienteController();
try{
    List<Paciente> listaDePacientes = pc.buscar(
        textFieldNome.getText(),
        textFieldEmail.getText());
    DefaultTableModel model =
        (DefaultTableModel) tableResultados.getModel();
    for (int i = model.getRowCount() - 1; i >= 0; i--){
        model.removeRow(i);
    }
    if (listaDePacientes != null){
        for (int i = 0; i < listaDePacientes.size(); i++){
            Paciente paciente = listaDePacientes.get(i);
            String[] p = new String[]{
                paciente.getNome(),
                paciente.getEndereco(),
                paciente.getEmail(),
                paciente.getTelefone(),
                String.valueOf(paciente.getNumerodoquarto())};
            model.insertRow(i, p);
        }
    }
    tableResultados.setModel(model);
}catch(Exception e){
    JOptionPane.showMessageDialog(this,
        "Não foi possível consultar os pacientes!\n\n"
        + e.getLocalizedMessage(), "Erro",
        JOptionPane.ERROR_MESSAGE);
}
```

- Como podemos ver no código acima, instanciamos a classe PacienteController assim como feito na view de Cadastro, porém, foi chamado o método buscar para trazer todos os pacientes encontrados com o nome/email especificado pelo usuário. Esse método buscar também não está preenchido. Iremos inserir dentro desse método a chamada de um método dentro da PacienteDAO que retorne uma lista de pacientes a partir do nome/email. Porém como os métodos criados nas classes DAO são métodos básicos de: inserção, alteração, exclusão e busca de todos os pacientes, nós iremos cria-lo.
- Portanto abra a classe PacienteDAO e crie o método abaixo dentro da classe:

```
public List<Paciente> getListaDePacientes(String nome,
    String email) throws Exception{
    EntityManager em = getEntityManager();
    try{
        String hql = "";
        if (nome != null && !nome.equals("")){
            hql += " WHERE p.nome LIKE UPPER(:nome)";
        }
        if (email != null && !email.equals("")){
            if (!hql.equals("")){
                hql += " AND p.email LIKE LOWER(:email)";
            }else{
                hql += " WHERE p.email LIKE LOWER(:email)";
            }
        }
        hql = "FROM Paciente p" + hql;
        Query q = em.createQuery(hql);
        if (nome != null && !nome.equals("")){
            q.setParameter("nome", "%" +
                nome.toUpperCase() + "%");
        }
        if (email != null && !email.equals("")){
            q.setParameter("email", "%" +
                email.toLowerCase() + "%");
        }
        return q.getResultList();
    }finally{
        em.close();
    }
}
```

- Observe que foi feita uma query comum no banco e serão adicionados os parâmetros nesse query somente se ela existir.
- Agora que já temos o método que busca, acesse-o pela classe PacienteController. Portanto abra a classe PacienteController e deixe o método buscar como abaixo:

```
public List<Paciente> buscar(String nome, String email) throws Exception{  
    return new PacienteDAO().  
        getListDePacientes(nome, email);  
}
```

- Antes de executar o projeto, você deve realizar algumas alterações no formulário CadastroDePacienteView. Portanto abra o formulário CadastroDePacienteView.
- Com o formulário aberto, dê um duplo clique sobre o botão Salvar e faça as alterações como indicado na imagem abaixo:

```
private void buttonSalvarActionPerformed(java.awt.event.ActionEvent evt) {  
    if (textFieldNome != null &&  
        !textFieldNome.getText().equals("") &&  
        textFieldEmail != null &&  
        !textFieldEmail.getText().equals("") &&  
        textFieldEndereco != null &&  
        !textFieldEndereco.getText().equals("") &&  
        textFieldDoenca != null &&  
        !textFieldDoenca.getText().equals("")) {  
        PacienteController pc = new PacienteController();  
        try{  
            pc.inserir(textFieldNome.getText().toUpperCase(),  
                textFieldEmail.getText().toLowerCase(),  
                textFieldEndereco.getText().toUpperCase(),  
                textFieldTelefone.getText(),  
                textFieldQuarto.getText(),  
                textFieldDoenca.getText().toUpperCase(),  
                textFieldDiasInternado.getText(),  
                (String)comboBoxPlanoDeSaude.getSelectedItem());  
            JOptionPane.showMessageDialog(this,  
                "Paciente salvo com sucesso!",  
                "Sucesso", JOptionPane.INFORMATION_MESSAGE);  
        } catch (Exception e) {  
            //  
        }  
    }  
}
```

toUpperCase: Inserindo esse comando, você determina que o registro deve ser salvo todo em letras maiúsculas no banco de dados.

toLowerCase: É o contrario do comando toUpperCase, ou seja, define que todos os registros sejam salvos em letra minúsculas no banco de dados.

- Faça o mesmo para os outros formulários de cadastro, como nas imagens abaixo:

## CadastroDeEnfermeiro

```
try{  
    ec.inserir(textFieldNome.getText().toUpperCase(),  
        textFieldEmail.getText().toLowerCase(),  
        textFieldEndereco.getText().toUpperCase(),  
        textFieldTelefone.getText(),  
        textFieldHorasMensais.getText(),  
        textFieldValorDaHora.getText());  
}
```

## CadastroDeMedico

```
try{
    mc.inserir(textFieldNome.getText().toUpperCase(),
               textFieldEmail.getText().toLowerCase(),
               textFieldEndereco.getText().toUpperCase(),
               textFieldTelefone.getText(),
               textFieldEspecialidade.getText().toUpperCase(),
               textFieldHorasMensais.getText(),
               textFieldValorDaHora.getText());
}
```

## CadastroDeVisitante

```
try{
    vc.inserir(textFieldNome.getText().toUpperCase(),
               textFieldEmail.getText().toLowerCase(),
               textFieldEndereco.getText().toUpperCase(),
               textFieldTelefone.getText(),
               textFieldPaciente.getText().toUpperCase());
}
```

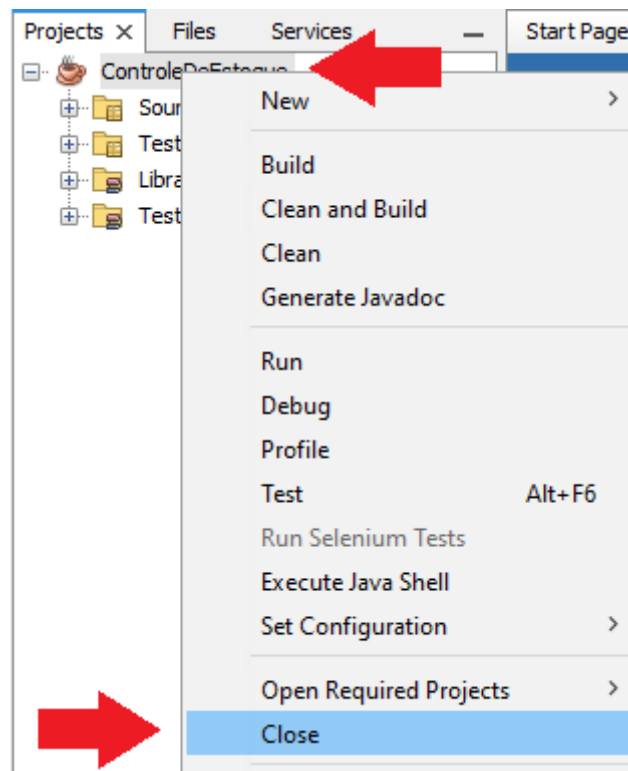
- Agora é hora de verificar se realmente as alterações tiveram efeito
- Execute o programa, faça alguns cadastros de pacientes e após, abra a janela de consulta e teste o botão Consultar.
- Enfim, foi modificado apenas a consulta do Paciente, agora faça as alterações para o restante das consultas.
- Salve as alterações e feche o programa.

# AULA 16

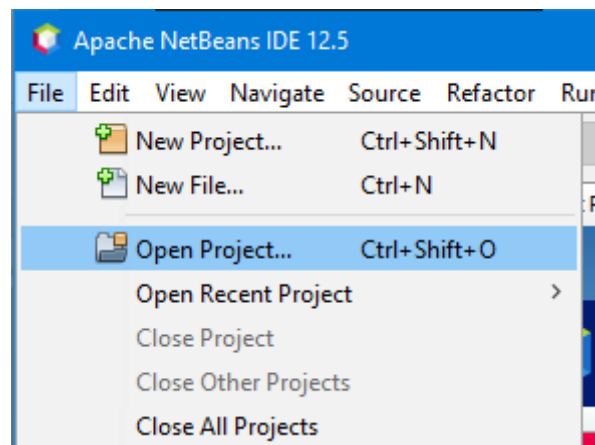
- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 15” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.



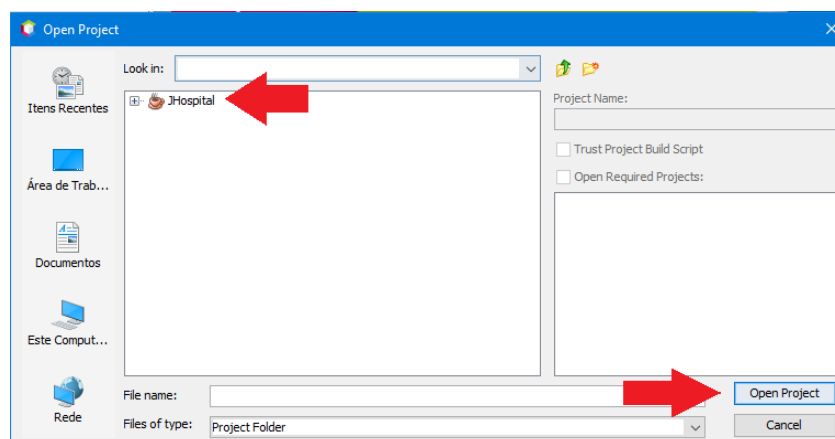
Icaro Daflon  
JAVA MOD II



- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”



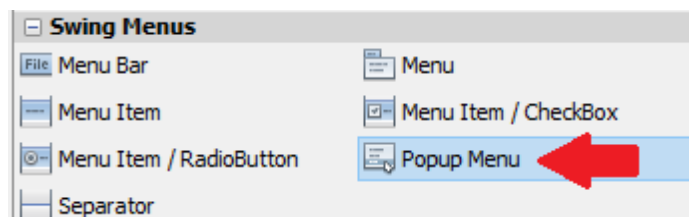
- Com o projeto aberto, adicione a biblioteca do PostgreSQL (PostgreSQL JDBC Driver)

- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL 14
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

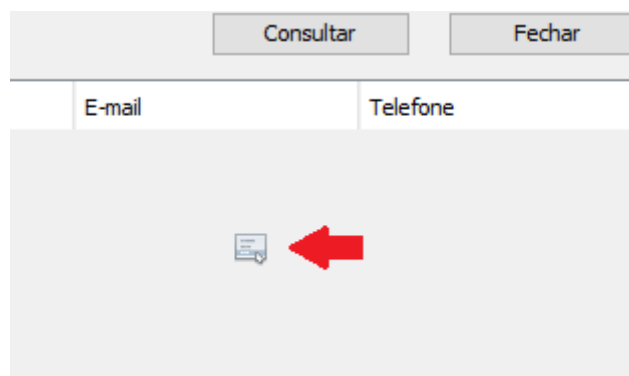
OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

### JHospital – Apagando Dados, Páginas 231 a 242

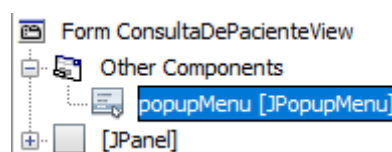
- Com o Banco de Dados inicializado, volte para o NetBeans e abra a classe ConsultaDePacienteView.
- Na guia “Design” adicione no formulário a propriedade “Popup Menu”.



- Coloque-o posicionado como na figura:

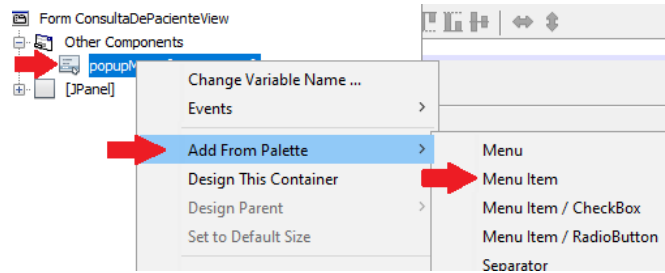


- Altere a propriedade “Variable Name” para “popupMenu”.

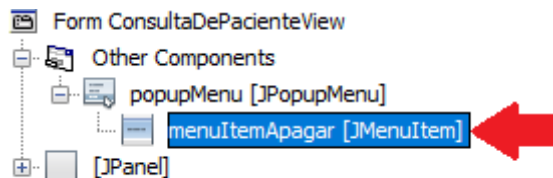


- O menu pop-up foi criado. Agora você deve criar os itens desse menu. Nesse primeiro momento, ele só irá possuir 1 item: Apagar.
- Clique com o botão direito em cima do popupMenu → Add from Palette → Menu Item

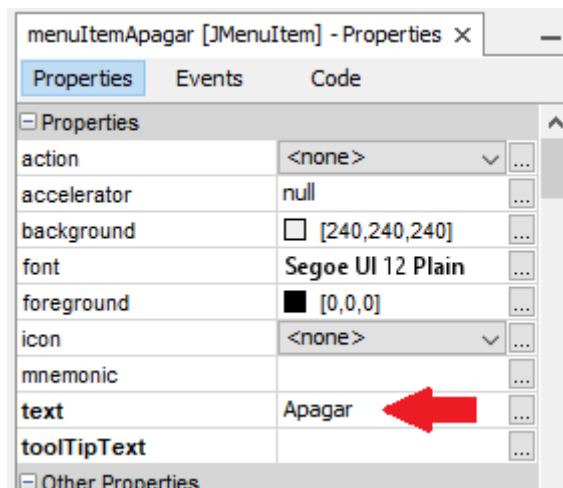
## Icaro Daflon JAVA MOD II



- Altere a propriedade “Variable Name” para menuItemApagar.

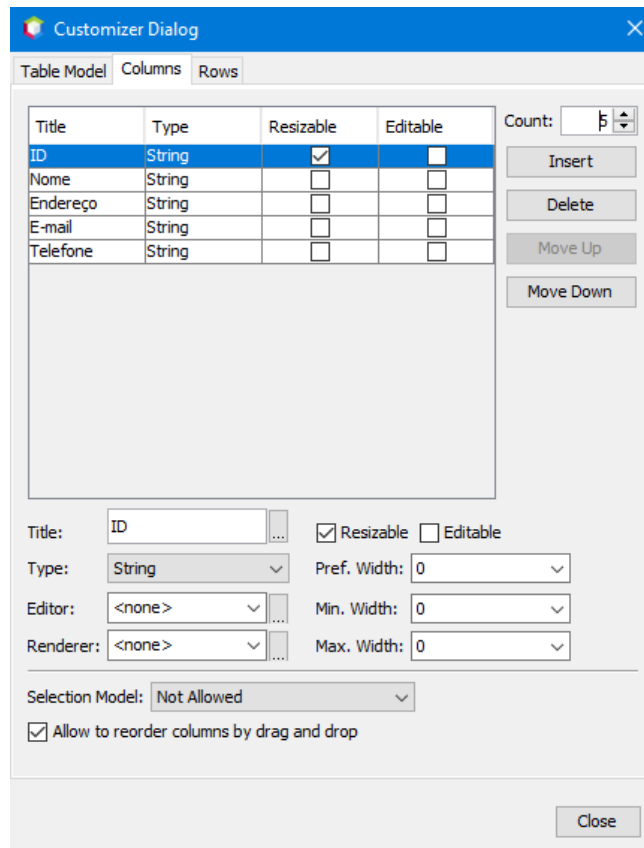


- Em properties do menuItemApagar, modifique o campo “text” para Apagar. Assim o texto exibido será Apagar.

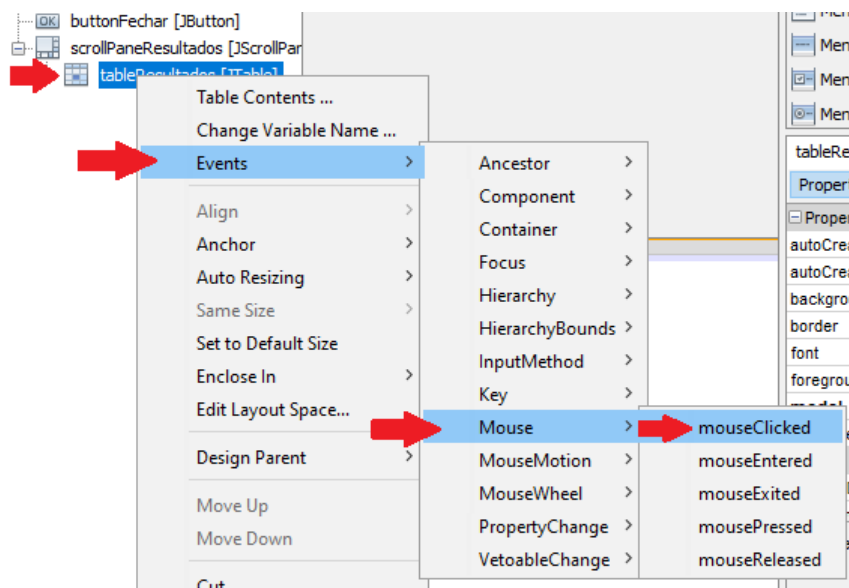


- Feito isso, é necessário modificar a tabela. Iremos adicionar uma coluna para guardar o id do paciente. Precisa-se adicionar essa coluna para que o método Apagar funcione corretamente. Porém essa coluna ficará invisível, pois não é interessante que o usuário veja o id do paciente.
- Clique com o botão direito na tableResultados → Table Contents...
- Acione a aba “Columns”.
- Insira uma nova coluna com o nome ID, e deixe-a como na tela abaixo:

## Icaro Daflon JAVA MOD II



- Veja que a coluna ID será redimensionável. Mas a largura dela (Width), tanto preferencial, como mínima e máxima serão 0, ou seja, ela não vai aparecer para o usuário.
- Feche a janela e veja que a coluna já não aparece no form de consulta de paciente.
- Agora é necessário determinar quando esse popupMenu será aberto. Nesse caso, o popupMenu será exibido quando o usuário clicar na tabela com o botão direito com o desejo de apagar alguma paciente. Assim será adicionar um MouseListener para a tabela.
- Clique com o botão direito em “tableResultados” → Events → Mouse → mouseClicked



- Esse evento será disparado toda vez que o usuário clicar com o mouse na tabela.
- Entre com o código abaixo dentro do método:

```
try{
    if (evt.getButton() == MouseEvent.BUTTON3){
        Point p = evt.getPoint();
        int row = tableResultados.rowAtPoint(p);
        if(row >= 0 && row < tableResultados.getRowCount()){
            tableResultados.setRowSelectionInterval(row, row);
            popupMenu.show(evt.getComponent(), evt.getX(), evt.getY());
        }
    }
}catch (Exception e){
    JOptionPane.showMessageDialog(this,
        "Não foi possível abrir o popup!\n\n"
        + e.getLocalizedMessage(),
        "Erro", JOptionPane.ERROR_MESSAGE);
}
```

- É interessante observar o primeiro if comparando o botão clicado no evento e o MouseEvent.BUTTON3. É esse if que justamente compara se o usuário clicou com o botão direito, pois o MouseEvent.BUTTON3 refere-se ao botão direito. O restante do código apenas permite que o popupMenu seja aberto no local onde foi clicado e apenas se for uma linha válida da tabela.
- Agora vamos programar o botão Apagar do popup, portanto abra a guia “Design” novamente, e na barra “Navigator” dê um duplo clique sobre a opção menuItemApagar.
- Dentro do método digite o código abaixo:

```
try{
    int row = tableResultados.getSelectedRow();
    PacienteController pc = new PacienteController();
    DefaultTableModel model =
        (DefaultTableModel) tableResultados.getModel();
    if (JOptionPane.showConfirmDialog(this,
        "Deseja mesmo apagar o paciente "
        + model.getValueAt(row, 1).toString(),
        "Confirmação", JOptionPane.YES_NO_OPTION)
        == JOptionPane.YES_OPTION){
        pc.excluir(Integer.parseInt(model.
           .getValueAt(row, 0).toString()));
        ((DefaultTableModel) tableResultados.
            getModel()).removeRow(row);
    }
}catch (Exception e){
    JOptionPane.showMessageDialog(this,
        "Não foi possível apagar o paciente!\n\n"
        + e.getLocalizedMessage(),
        "Erro", JOptionPane.ERROR_MESSAGE);
}
```

- Repare que neste método foi, instanciado novamente o PacienteController, porém agora, foi utilizado o método excluir para fazer a exclusão dessa pessoa. Repare também, que foi passado o valor da célula que está na linha clicada e primeira coluna (getValueAt(row, 0)). Ou seja, foi enviado para o método excluir o id do paciente. Porém devemos ainda alterar o método consultar para que o ID seja inserido na tabela.
- Portanto abra a guia “Design” e dê um duplo clique sobre o botão Consultar.

- Organize o array de String p adicionando o id na primeira casa e deixando-o como abaixo:

```
String[] p = new String[]{  
    String.valueOf(paciente.getId()),  
    paciente.getNome(),  
    paciente.getEndereco(),  
    paciente.getEmail(),  
    paciente.getTelefone(),  
    String.valueOf(paciente.getNumerodoquarto())};
```

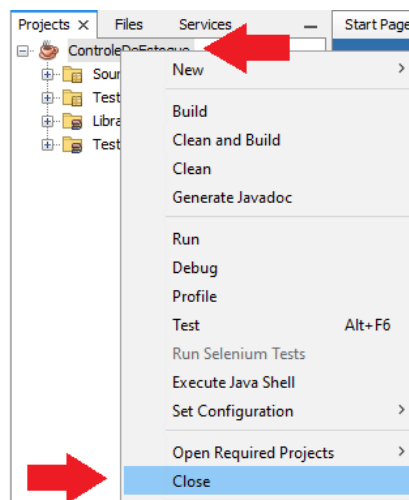
- Feito isso, agora basta modificar o método excluir da classe PacienteController para que a exclusão seja realmente feita.
- Abra a classe PacienteController e modifique o método excluir deixando-o como abaixo:

```
public void excluir(Integer id) throws Exception{  
    new PacienteDAO().destroy(id);  
}
```

- Com isso já é possível apagarmos um paciente. Execute o programa e verifique se todas as alterações foram feitas corretamente.
- Após verificar que tudo está funcionando corretamente, faça essas alterações para todas as classes restantes (Enfermeiro, Medico e Visitante).
- Salve as alterações feitas e Feche o NetBeans.

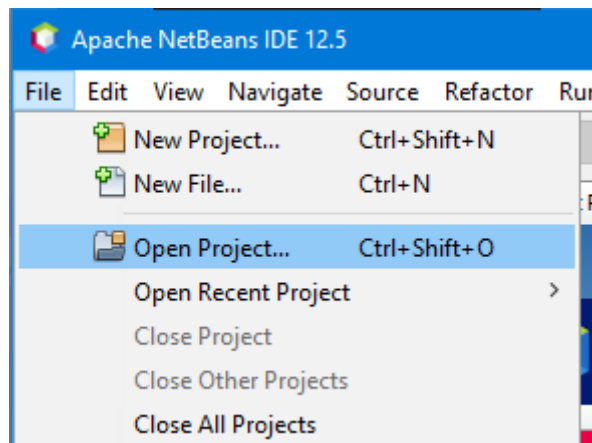
## AULA 17

- Primeiramente deveremos preparar o sistema para que possamos continuar do ponto que paramos na aula passada, para isso peça a seu instrutor o projeto da “Aula 16” e copie a pasta do projeto “JHospital”.
- Em seguida, cole a pasta copiada em seu local de gravação. Caso surja alguma mensagem, clique em Substituir os arquivos no destino.
- Em seguida abra o NetBeans. Caso já exista algum projeto aberto, clique com o botão direito do mouse sobre ele e em seguida, clique em “Close”.

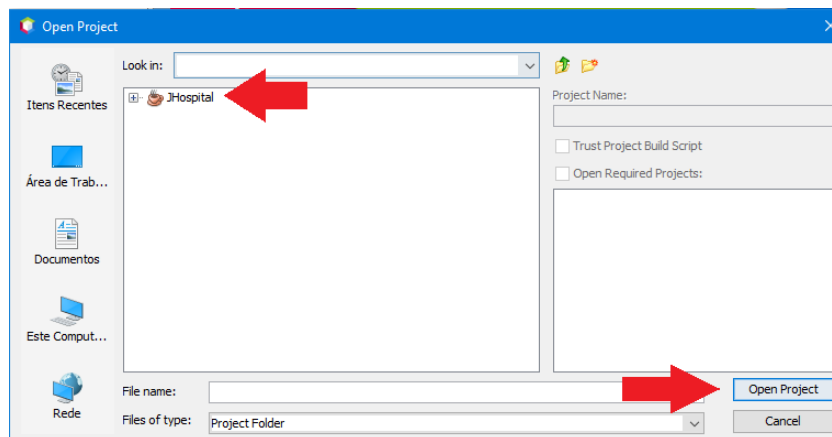


**Icaro Daflon**  
**JAVA MOD II**

- Com o NetBeans preparado, clique sobre o menu “File” e após, na opção “Open Project...”



- Na janela que surge, selecione o projeto salvo no seu local de gravação e após, clique em “Open Project”

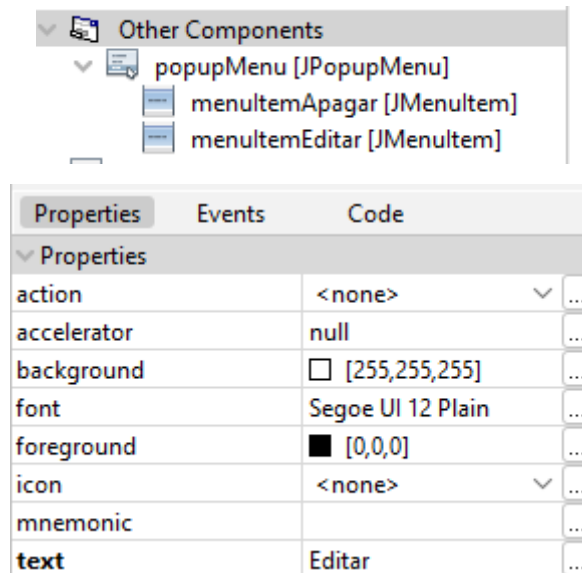


- Com o projeto aberto, adicione a biblioteca do PostgreSQL (PostgreSQL JDBC Driver)
- Após preparar o sistema, minimize o NetBeans e abra o pgAdmin 4
- Conecte-se ao servidor PostgreSQL 14
- Verifique se os bancos de dados DBJHospital e o login Role UserJHospital já estão criados. Caso o Banco de Dados e Usuario já estejam criados, abra o Query Tool pelo banco de dados postgres e execute os scripts “Script Drop Database” e o “Script Database” nesta mesma ordem.
- Caso o Banco de Dados e o Login Role não estejam criados execute então os scripts “Script Login Role” e o “Script Database” também nesta mesma ordem.
- Conecte-se ao Banco de Dados “DBJHospital”.
- Agora devemos criar as tabelas e as sequências, para isso, peça a seu instrutor o script “Tabelas JHospital” e o execute no pgAdmin 4.

OBS: Mesmo não se preocupe caso o script dê erro pois terá funcionado da mesma forma

## JHospital – Editando Dados, Páginas 246 a 254

- Com tudo configurado corretamente, abra a classe ConsultaDePacienteView.
- Na guia “Design” clique com o botão direito sobre popupMenu → Add from Palette → Menu Item.
- Altere a propriedade Variable Name para menuItemEditar e a propriedade text que será exibido para Editar.



- Feito isso, agora iremos fazer algumas mudanças para que se torne possível editar um paciente.
- A primeira mudança será justamente na classe CadastroDePacienteView. Iremos modifica-la, pois ela será utilizada tanto para cadastro como também para edição de um paciente. Se você criasse uma nova tela de editar para cada um, seria demorado e desnecessário, já que bastam algumas modificações para que a classe CadastroDePacienteView fique pronta para editar.
- Abra a classe CadastroDePacienteView e na aba “Source” crie um objeto global Paciente, como abaixo:  

```
private Paciente paciente;
```
- Crie um novo método com o nome preencherCampos que será responsável por preencher os campos exibidos na tela com as informações do paciente a ser editado. Deixe-o como na imagem abaixo:



```
private void preencherCampos (Integer id) throws Exception{
    PacienteController pc = new PacienteController();
    this.paciente = pc.buscarPeloId(id);
    textFieldNome.setText(paciente.getNome());
    textFieldEmail.setText(paciente.getEmail());
    textFieldEndereco.setText(paciente.getEndereco());
    textFieldTelefone.setText(paciente.getTelefone());
    textFieldQuarto.setText(
        String.valueOf(paciente.getNumerodoquarto()));
    textFieldDoenca.setText(paciente.getDoenca());
    textFieldDiasInternado.setText(
        String.valueOf(paciente.getDiasdeinternacao()));
    if(paciente.getTemplanodesaude()){
        comboBoxPlanoDeSaude.setSelectedIndex(0);
    }else{
        comboBoxPlanoDeSaude.setSelectedIndex(1);
    }
}
```

- Repare que esse método recebe o id do Paciente, faz uma busca pelo id e retorna o paciente referente ao id escolhido, e preenche os dados a partir disso. Provavelmente ocorrerá um erro no código, já que ainda não foi implementado o método buscarPeloId dentro da classe PacienteController. Iremos fazer essa modificação mais adiante portanto fique tranquilo.
- Agora crie um novo construtor na classe CadastroDePacienteView seguindo a figura abaixo:

```
public CadastroDePacienteView(JTabbedPane tabbedPane,
    Integer id) throws Exception{
    this.tabbedPane = tabbedPane;
    initComponents();
    preencherCampos(id);
}
```

- Agora iremos modificar o método do botão Salvar para que ele realize a edição do paciente caso a variável global paciente e seu id não forem nulos e caso contrário faça uma inserção.
- Selecione a aba “Design” e dê um clique duplo no botão Salvar.
- No método onde se encontra o seguinte código:

```
pc.inserir(textFieldNome.getText().toUpperCase(),
    textFieldEmail.getText().toLowerCase(),
    textFieldEndereco.getText().toUpperCase(),
    textFieldTelefone.getText(),
    textFieldQuarto.getText(),
    textFieldDoenca.getText().toUpperCase(),
    textFieldDiasInternado.getText(),
    (String) comboBoxPlanoDeSaude.getSelectedItem());
JOptionPane.showMessageDialog(this,
    "Paciente salvo com sucesso!",
    "Sucesso", JOptionPane.INFORMATION_MESSAGE);
```

- Modifique-o para que fique como na imagem abaixo:

```
if(paciente != null && paciente.getId() != null){
    pc.alterar(paciente.getId(),
        textFieldNome.getText(),
        textFieldEmail.getText().toLowerCase(),
        textFieldEndereco.getText().toUpperCase(),
        textFieldTelefone.getText(),
        textFieldQuarto.getText(),
        textFieldDoenca.getText().toUpperCase(),
        textFieldDiasInternado.getText(),
        (String) comboBoxPlanoDeSaude.getSelectedItem());
    JOptionPane.showMessageDialog(this,
        "Paciente editado com sucesso!",
        "Sucesso", JOptionPane.INFORMATION_MESSAGE);
    tabbedPane.remove(this);
    tabbedPane.validate();
    tabbedPane.repaint();
}else{
    pc.inserir(textFieldNome.getText().toUpperCase(),
        textFieldEmail.getText().toLowerCase(),
        textFieldEndereco.getText().toUpperCase(),
        textFieldTelefone.getText(),
        textFieldQuarto.getText(),
        textFieldDoenca.getText().toUpperCase(),
        textFieldDiasInternado.getText(),
        (String) comboBoxPlanoDeSaude.getSelectedItem());
    JOptionPane.showMessageDialog(this,
        "Paciente salvo com sucesso!",
        "Sucesso", JOptionPane.INFORMATION_MESSAGE);
}
```

- Repare que foi feito somente um if para verificar se o objeto paciente está ou não preenchido.
- Agora abra a classe PacienteController e crie o método buscarPeloId como a figura abaixo:

```
public Paciente buscarPeloId (Integer id) throws Exception{
    return new PacienteDAO().findPaciente(id);
}
```

- Ainda na classe PacienteController, modifique o método alterar de acordo com o código abaixo:

```
public void alterar(Integer id, String nome, String email,
    String endereco, String telefone, String quarto, String doenca,
    String diasInternado, String temPlanoDeSaude) throws Exception{
    Paciente paciente = new Paciente();
    paciente.setId(id);
    paciente.setNome(nome);
    paciente.setEmail(email);
    paciente.setTelefone(telefone);
    paciente.setEndereco(endereco);
    paciente.setNumerodoquarto(Integer.parseInt(quarto));
    paciente.setDoenca(doenca);
    paciente.setDiasdeinternacao(Integer.parseInt(diasInternado));
    if(temPlanoDeSaude.equalsIgnoreCase("sim")){
        paciente.setTemplanodesaude(true);
    }else{
        paciente.setTemplanodesaude(false);
    }
    new PacienteDAO().edit(paciente);
}
```

- Após essas mudanças, agora basta colocarmos esta ação no menuItemEditar
- Abra a classe ConsultaDePacienteView e na aba “Design” dê um duplo clique no menuItemEditar.
- No método aberto, escreva o seguinte código:

```
try{
    int row = tableResultados.getSelectedRow();
    DefaultTableModel model =
        (DefaultTableModel) tableResultados.getModel();
    CadastroDePacienteView cadastroDePacienteView
        = new CadastroDePacienteView(tabbedPane,
        Integer.parseInt(model.getValueAt(row, 0).toString()));
    tabbedPane.add("Edição de Paciente", cadastroDePacienteView);
    tabbedPane.setSelectedComponent(cadastroDePacienteView);
    tabbedPane.revalidate();
    tabbedPane.repaint();
}catch (Exception e){
    JOptionPane.showMessageDialog(this,
        "Não é possível editar o paciente!\n\n"
        + e.getLocalizedMessage(),
        "Erro", JOptionPane.ERROR_MESSAGE);
}
```

- E com isso finalizamos nosso projeto. Faça todas as alterações que fizemos nesta aula para as outras classes (Medico, Enfermeiro e Visitante).
- Teste as alterações feitas e o programa como um todo.
- Salve as alterações e feche o NetBeans.