

O Essencial de JavaFX com Hibernate

Henrique Junqueira

Sumário

Introdução.	3
Configuração Hibernate.	4
Classe de Conexão com Banco de Dados.	12
Criando uma classe.	17
Mapeando a Classe Criada.	20
Classe Dao Método Salvar.	25
Classe Dao Método Excluir	28
Classe Dao Método Consultar	29
Criando uma Interface	31
Salvando registros.	35
Excluindo Registros	36
Setando dados no formulário	37
Atualizar tabela.	38
Pesquisando Registros	39
Criando Colunas na TableView.	40
Limpar Campos do Formulário	42
Adicionando Máscara nos componentes JavaFX.	43
Exemplo Máscara Telefone.	44
Exemplo Máscara Cep	45
Popular dados no ComboBox	46
Popular dados do banco no combobox.	47
Adicionando Tooltip	49
DatePicker.	52
Adicionando ícone no título da janela	54
Códigos Utilizados.	55
Arquivo hibernate.cfg	56
Arquivo Conexao.java.	57
Arquivo Classe Usuario.java	58
Arquivo UsuarioDao.java	60
Arquivo UsuarioController.java.	62
Arquivo da Interface Icadastro.java	67
Arquivo ClienteController.java.	68

Introdução

Desenvolvi esse pequeno guia para estudantes de JavaFX, não com o intuito de ter tudo que você precisa, mas apenas para auxiliá-lo com alguns comandos que podem ser difíceis de lembrar ou até mesmo seja difícil de encontrar respostas em fóruns, pois já passei por isso e sei como é.

Nesse guia irei adicionar o essencial para a utilização do hibernate, mapeamento de classes e como utilizar alguns componentes interessantes de se ter numa aplicação Javafx.

Aproveite o guia e lembre-se sempre: "A prática leva à perfeição".

Configuração Hibernate

Para aqueles que não conhecem a ferramenta de consulta e persistência de banco de dados Hibernate, vai conhecer como trabalhar com ela, pois ela facilita a vida de qualquer programador.

Uma das vantagens do hibernate é que não há mais a necessidade da criação de tabelas de forma manual como se fazia a um tempo atrás, e isso agiliza demais o processo, pois imagina só ter que criar tabela por tabela, campo por campo de forma manual adicionando o tamanho de permitido para cada campo, o tipo de dado e demais configurações. Para um sistema pequeno até que é fácil, mas quando se torna algo mais complexo o negócio é diferente.

Além dessa e de muitas outras vantagens, você verá como a classe de conexão já não é o mesmo monstro de sete cabeças como era com JDBC. Digo isso porque, quem chegou a trabalhar com a tal classe entende o que estou dizendo.

Mas vamos lá! Você entenderá melhor quando chegarmos na criação da tal classe.

Primeiramente é necessário colocar as dependências no arquivo pom.xml, pois é ele que irá baixar todas as dependências necessárias para realizar a utilização do hibernate, mysql e demais funcionalidades.

Abra o arquivo pom.xml que fica na pasta "Arquivos do projeto", em seguida copie e cole as dependências para Hibernate e MySQL que estão logo abaixo:

```
<dependencies>
    <!-- HIBERNATE -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.6.Final</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.1.Final</version>
    </dependency>

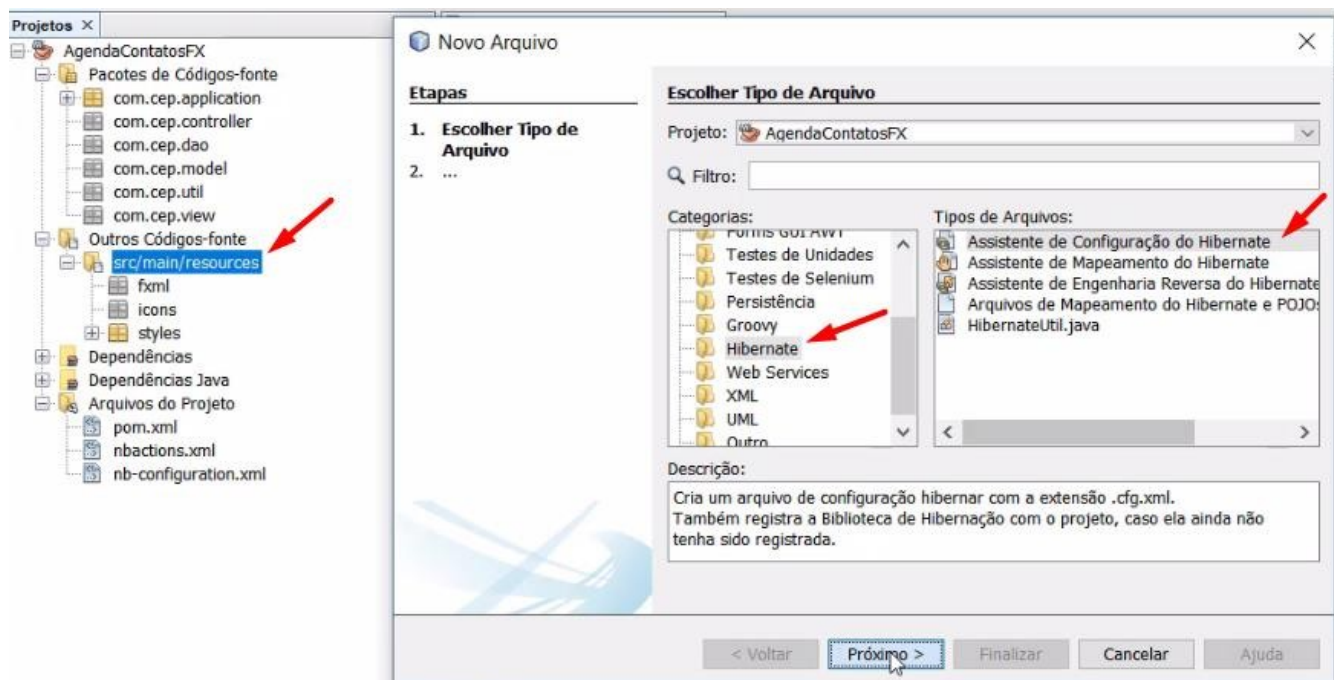
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
```

```
<version>4.3.1.Final</version>
</dependency>

<!-- MYSQL -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.41</version>
</dependency>
</dependencies>
```

Feito isso crie o arquivo de configurações para conexão com banco de dados – hibernate.cfg

Vá até outros códigos-fonte e clique com o botão direito sobre src/main/resources. Escolha a opção outros. Em Categorias clique em hibernate e em Tipos de arquivos escolha Assistente de configuração do hibernate. Veja a figura abaixo:



Mantenha o nome indicado e clique em próximo.

Etapas

1. Escolher Tipo de Arquivo
- 2. Nome e Localização**
3. Selecionar Código-Fonte de Dados

Nome e LocalizaçãoNome do Arquivo: Projeto: Pasta: Arquivo Criado:

Acesse um SGBD (WorkBench, MySQL-Front, PHPMyAdmin, etc) e crie um banco de dados com nome que desejar. Agora podemos continuar a configuração de conexão. Selecione a opção Conexão com Banco de Dados. Clique em Nova Conexão com Banco de Dados. Veja figura abaixo:

Etapas

1. Escolher Tipo de Arquivo
2. Nome e Localização
3. **Selecionar Código-Fonte de Dados**

Selecionar Código-Fonte de Dados

Conexão de Banco de Dados: jdbc:derby://localhost:1527/sample [app em APP] ▼

Dialeto do Banco de Dados: jdbc:derby://localhost:1527/sample [app em APP]
jdbc:mysql://localhost:3306/agecon?zeroDateTimeBehavior=conver
jdbc:mysql://localhost:3306/ageconfx?zeroDateTimeBehavior=conve
jdbc:mysql://localhost:3306/ageconnoite?zeroDateTimeBehavior=cc
Nova Conexão de Banco de Dados...

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Escolha um driver para o servidor de banco de dados e clique em próximo. No meu caso utilizarei o MySQL.

Localizar Driver

Driver:	Java DB (Embedded) ▾
Arquivo	Java DB (Embedded)
	Java DB (Network)
	MySQL (Connector/J driver)
	Oracle OCI
	Oracle Thin
	PostgreSQL
	Novo Driver...

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Determine as informações para conexão com seu banco de dados. Lembrando que o servidor de banco de dados deverá estar rodando (startado). Para essa conexão estamos utilizando o WampServer e o mesmo deverá estar com o seguinte ícone



Assistente de Nova Conexão

Personalizar Conexão

Nome do Driver:

MySQL (Driver Connector/J) em MySQL (Connector/J driver)

Host:

localhost

Porta:

3306

Banco de dados:

ageconfx

Nome do Usuário:

root

Senha:

☐ Lembrar senha

Propriedades da Conexão

Testar Conexão

JDBC URL:

jdbc:mysql://localhost:3306/ageconfx?zeroDateTimeBehavior=convertToNull

Conexão Bem-sucedida.

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Clique em Testar Conexão e verifique se tudo ocorreu bem. Se a conexão foi bem sucedida, clique em Finalizar / Finalizar.

Após a criação do arquivo vá até outros códigos-fonte em seguida src/main/resources, abra o arquivo hibernate.cfg e clique em código-fonte para visualizar o código:

Projetos

Arquivos

Serviços

AgeConFX2017

AgendaContatos

Pacotes de Códigos-fonte

com.cep.application

com.cep.controller

com.cep.dao

com.cep.model

com.cep.util

Outros Códigos-fonte

src/main/resources

<pacote default>

hibernate.cfg.xml

hibernate.cfg.xml

Design

Código-fonte

Histórico

Propriedades de JDBC

Fábrica de Sessão

Propriedades de JDBC

Nome	Valor
hibernate.connection.driver_class	com.mysql.jdbc.Dri
hibernate.connection.url	jdbc:mysql://localh
hibernate.connection.username	root

Adicionar...

Editar...

Remover

Apague completamente o código gerado e copie o código abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property
>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/sistemave
ndas?zeroDateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username">root</property>
    <!-- VISUALIZAR OS CÓDIGOS SQL NA COMPILAÇÃO -->
    <property name="hibernate.show_sql">true</property>
    <!-- FORMATAR A EXIBIÇÃO DOS DADOS ACIMA -->
    <property name="hibernate.format_sql">true</property>
    <!-- MODO DE CRIAÇÃO DAS TABELAS -->
    <property name="hbm2ddl.auto">update</property>
  </session-factory>
</hibernate-configuration>
```

Após colar o código em seu hibernate.cfg vá até a linha 7, onde diz:

```
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/agecong
x?zeroDateTimeBehavior=convertToNull</property>
```

Altere o nome do banco de acordo com o nome que você definiu no SGBD. Para isso, é só alterar, na mesma linha que destaquei, depois de `jdbc:mysql://localhost:3306/` para o nome do seu banco de dados.

Exemplo:

```
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/sistema
vendafx?zeroDateTimeBehavior=convertToNull</property>
```

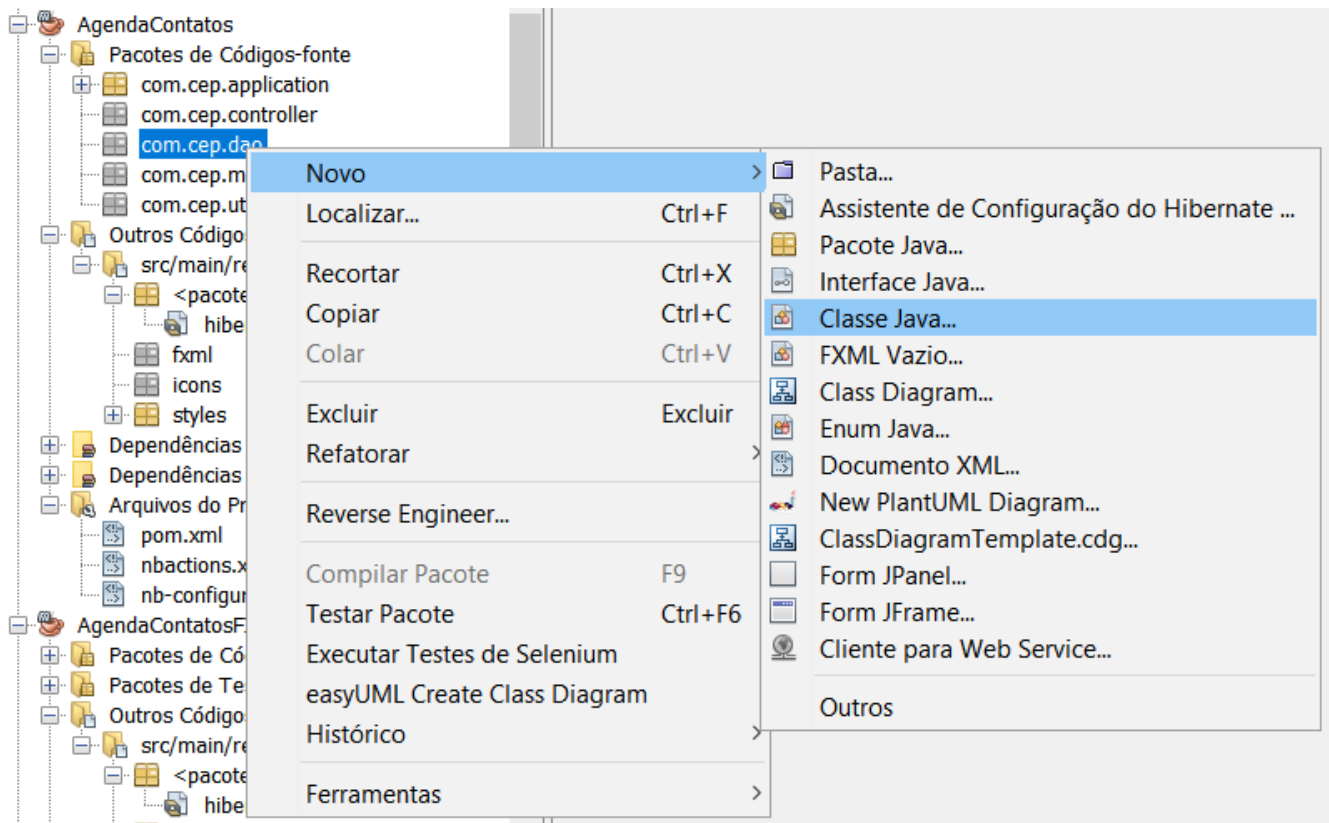
Feito isso seu hibernate está pronto para trabalhar com o banco de dados. Lembrando que com ele, não é necessário se preocupar, de uma forma mais ampla, com os processos que irão acontecer com o banco de dados, pois o próprio hibernate fará todos os processos de criação de tabelas, bastando apenas mapearmos nossas classes que ele entenderá e fará o resto.

Classe de Conexão com Banco de Dados

Como foi dito no início do capítulo anterior, a criação da classe de conexão em JDBC era complicada, pois era necessário criar um método complidado de conexão com o banco de dádos, método de salvar, excluir, atualizar, consultar no banco de dados, o que era necessário saber comandos em sql.

Porém, agora não é necessário ter um conhecimento tão aprofundado em sql, claro que pelo menos o básico é bom saber, mas a vida se tornou mais fácil, vamos lá.

Primeiramente, dentro do pacote dao, crie uma classe denominada ConexaoBanco ou o nome que preferir. Siga as imagens para entender melhor:



New Classe Java

Etapas

1. Escolher Tipo de Arquivo

2. Nome e Localização

Nome e Localização

Nome da Classe: ConexaoBanco

Projeto: AgendaContatos

Localização: Pacotes de Códigos-fonte

Pacote: com.cep.dao

Arquivo Criado: sFX\AgendaContatos\src\main\java\com\cep\dao\ConexaoBanco.java

< Voltar

Próximo >

Finalizar

Cancelar

Ajuda

Após finalizar, abra o arquivo de conexão, copie e cole o código abaixo:

```
private static SessionFactory conexao = null;
private static Configuration configuracao;
```

Essas são as variáveis que criarão a sessão de conexão e configuração da sessão.

conexao: armazena uma conexão com banco de dados

configuracao: armazena os dados do arquivo hibernate.cfg. Podemos também criar diversas configurações para acesso ao banco de dados.

Em seguida copie e cole o método que irá gerar uma conexão com banco de dados:

```
private static SessionFactory buildSessionFactory(){
}
```

Implementando o método `buildSessionFactory()` será criada a conexão com banco de dados.

Agora temos que instanciar uma nova configuração:

```
//----> Objeto que armazena as configurações de conexão  
configuracao = new Configuration().configure();
```

Em seguida iremos configurar o usuário e senha para acesso ao banco de dados, de forma que pessoas não autorizadas não tenham acesso a ele:

```
configuracao.setProperty("hibernate.connection.username", "root");  
configuracao.setProperty("hibernate.connection.password", "1234");
```

Em password adicione a senha do seu servidor, caso seu wampserver, xampp ou o servidor que estiver utilizando não tiver uma senha configurada, deixe dessa forma:

```
configuracao.setProperty("hibernate.connection.username", "root");  
configuracao.setProperty("hibernate.connection.password", "");
```

Agora chegou a hora de mapear as classes.

O hibernate irá criar todas as tabelas do banco de dados baseando-se nas classes do nosso pacote `model`. Para isso devemos mapeá-las. Como ainda não temos classes no pacote `model`, deixaremos o código comentado, copie o código abaixo, em seguida adicione `//` na frente da linha para comentar:

```
configuracao.addPackage("com.cep.model").addAnnotatedClass(Usuario.class);  
  
configuracao.addPackage("com.cep.model").addAnnotatedClass(Cliente.class);
```

Dentro de `addAnnotatedClass`, adicione a classe que você criar. Só é permitida uma classe em cada `addAnnotatedClass`, ou seja, para cada classe que você tiver que utilizar, é necessário repetir toda a linha novamente.

Após isso, iremos pegar as configurações e setar uma conexão, utilizando as linhas abaixo:

```
conexao = configuracao.buildSessionFactory();  
return conexao;
```

Concluimos a primeira parte. Agora iremos criar um pequeno método para selecionar uma conexão com banco de dados:

```
public static SessionFactory getSessionFactory(){
    if(conexao == null){
        conexao = buildSessionFactory();
    }
    return conexao;
}
```

Esse pequeno método verifica se a conexão é nula, ou seja, se não há banco de dados conectado. Se não houver, cria uma conexão com o banco de dados. Então retorna a conexão para quem chamou o método.

Para conferir se está tudo certo, adicionarei o código completo logo abaixo:

```
package com.cep.dao;

import com.cep.model.Cliente;
import com.cep.model.Login;
import com.cep.model.Pagamento;
import com.cep.model.Produto;
import com.cep.model.Usuario;
import com.cep.model.Venda;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

public class Conexao {
    private static SessionFactory conexao = null;
    private static Configuration configuracao;

    private static SessionFactory buildSessionFactory(){
        //----> Objeto que armazena as configurações de conexão
        configuracao = new Configuration().configure();

        //configuracao.setProperty("hibernate.connection.username",
"root");
        //configuracao.setProperty("hibernate.connection.password",
"");

        //configuracao.addPackage("com.cep.model").addAnnotatedClass(Usu
ario.class);
    }
}
```

```
//configuracao.addPackage("com.cep.model").addAnnotatedClass(Cli
ente.class);

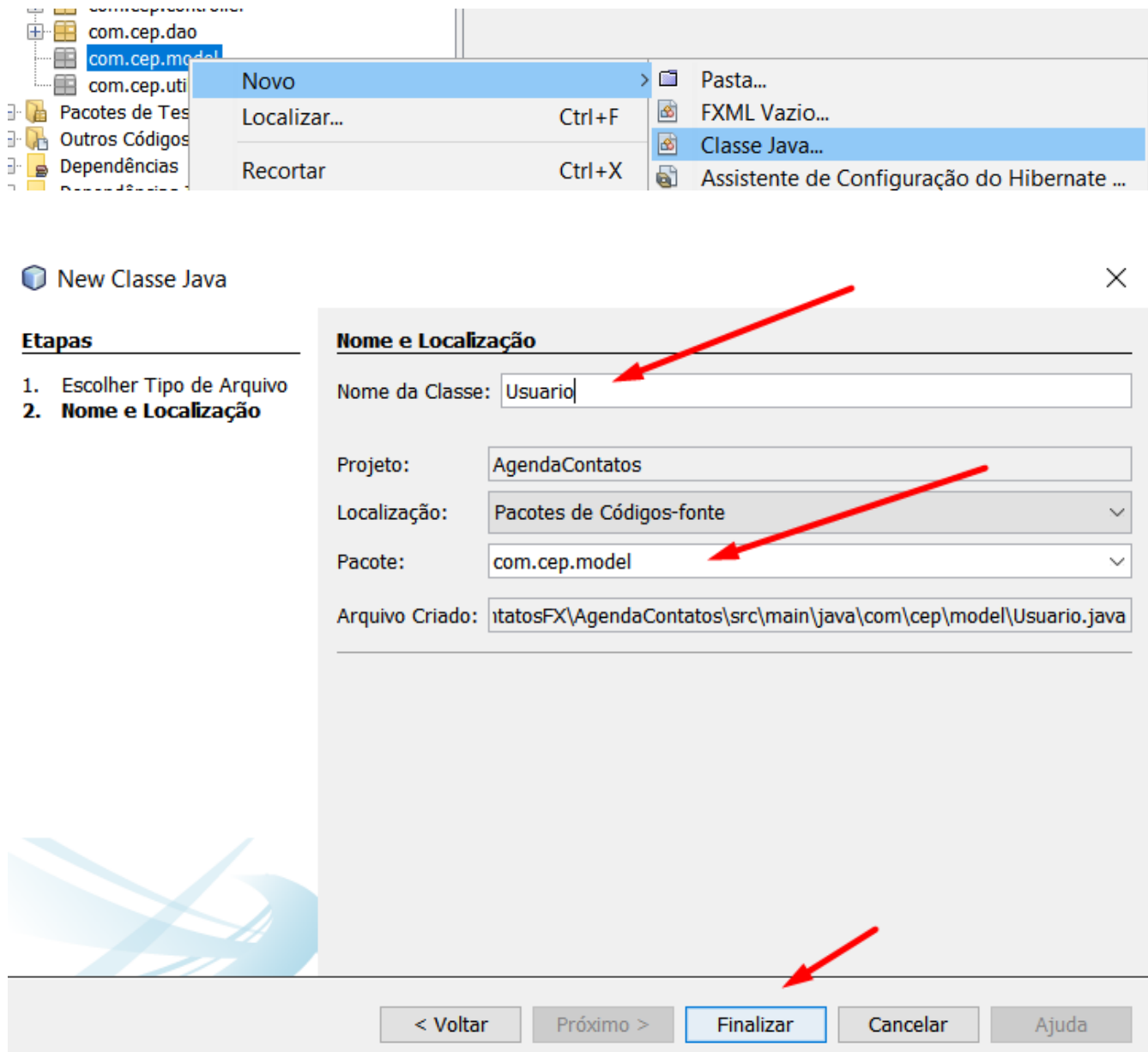
    conexao = configuracao.buildSessionFactory();
    return conexao;
}

public static SessionFactory getSessionFactory(){
    if(conexao == null){
        conexao = buildSessionFactory();
    }
    return conexao;
}
}
```

Caso esteja ocorrendo algum erro, verifique se as importações estão corretas. Há casos de importação de bibliotecas do awt, swing ou até mesmo bibliotecas que não tem as funcionalidades de que precisamos. Então verifique com atenção.

Criando uma classe

Irei criar aqui a classe usuário, porém se preferir, você pode criar uma classe de acordo com a sua necessidade.



A seguir foi gerada para mim a seguinte estrutura:

```
package com.cep.model;  
  
public class Usuario{  
  
}
```

Verifique que para você pode ser que a classe esteja com nome diferente:

```
package com.cep.model;

public class NomeDaSuaClasse{

}
```

Agora irei criar os atributos da minha classe, adicione os atributos de acordo com a sua necessidade:

```
package com.cep.model;

public class Usuario{
    private Long id;
    private String descricao;
    private String senha;
}
```

Após feito isso, insira os métodos getters e setters. Pressione Alt+Ins no seu teclado ou Alt+Insert. Selecione Getter e Setter. Marque Selecione tudo. Clique em Gerar. O resultado da minha classe foi o seguinte:

```
package com.cep.model;

public class Usuario{
    protected Long id;
    protected String descricao;
    protected String permissoes;
    protected String email;
    protected String login;
    protected String senha;
    private Boolean ativo;

    public Long getId() {
        return id;
    }

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

    public String getDescricao() {
        return descricao;
    }
}
```

```
public void setDescricao(String descricao) {
    this.descricao = descricao;
}

public String getPermissoes() {
    return permissoes;
}

public void setPermissoes(String permissoes) {
    this.permissoes = permissoes;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

public Boolean getAtivo() {
    return ativo;
}

public void setAtivo(Boolean ativo) {
    this.ativo = ativo;
}
}
```

Caso ocorra algum erro, verifique quais tipos de dados você utilizou para criar sua classe.

Mapeando a Classe Criada

Como estamos usando hibernate, devemos mapear a classe para que o mesmo entenda que esta classe deverá ser transformada em tabela no banco de dados. Utilizaremos @ para fazer as anotações necessárias, e assim o hibernate já cria tudo relacionado a banco de dados pra nós.

Primeiro iremos dizer que a classe é uma entidade, adicionando na nossa classe o seguinte:

```
package com.cep.model;

@Entity
public class Usuario{
    ...
}
```

Em seguida iremos dizer qual tabela deve ser criada. Adicione em name, o nome que deseja para sua tabela, mas não há necessidade caso não queira adicionar a linha:

```
package com.cep.model;

@Entity
@Table(name="usuario")
public class Usuario{
    ...
}
```

Depois de declarada, iremos dizer qual será a chave primária da nossa tabela:

```
package com.cep.model;

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    protected Long id;
    ...
}
```

Agora, para que não haja necessidade de colocarmos id manualmente nos nossos registros, iremos adicionar o auto-incremento. Ou seja, o próprio banco de dados fará o processo de adicionar id aos registros:

```

package com.cep.model;

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    protected Long id;
    ...
}

```

Vale dizer que GenerationType.IDENTITY, faz referência ao banco de dados MySQL, caso esteja utilizando outro banco de dados, é só alterar para GenerationType.AUTO ou outro tipo referente ao seu banco de dados.

Depois disso iremos nomear a coluna e dizer que esse campo não pode ser nulo:

```

package com.cep.model;

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    protected Long id;
    ...
}

```

Agora vamos para o próximo atributo, que no meu caso é a descrição. Para esse atributo, irei adicionar a quantidade de caracteres permitidos que o usuário pode digitar no campo de texto:

```

package com.cep.model;

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)

```

```

protected Long id;

@Column(name = "descricao", length = 100, nullable = false)
protected String descricao;
...
}

```

Para os outros atributos é só copiar e colar a mesma linha @Column, e alterar o nome do campos e se necessário a quantidade de caracteres que pode ser digitado.

No final a classe mapeada deverá ficar da seguinte forma:

```

package com.cep.model;

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

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    protected Long id;
    @Column(name = "descricao", length = 100, nullable = false)
    protected String descricao;
    @Column(name = "permissoes", length = 20, nullable = false)
    protected String permissoes;
    @Column(name = "email", length = 100, nullable = false)
    protected String email;
    @Column(name = "login", length = 100, nullable = false)
    protected String login;
    @Column(name = "senha", length = 8, nullable = false)
    protected String senha;
    @Column(name = "ativo", nullable = true)
    private Boolean ativo;

    public Long getId() {
        return id;
    }

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

```

```
public String getDescricao() {
    return descricao;
}

public void setDescricao(String descricao) {
    this.descricao = descricao;
}

public String getPermissoes() {
    return permissoes;
}

public void setPermissoes(String permissoes) {
    this.permissoes = permissoes;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

public Boolean getAtivo() {
    return ativo;
}

public void setAtivo(Boolean ativo) {
    this.ativo = ativo;
}
}
```

Verifique se as importações estão corretas.

Para finalizar devemos acionar o mapeamento na classe de conexão. Para isso, descomente a linha da anotação para `Usuario.class` ou a linha da classe que você estiver utilizando, retirando `//` da frente da linha. E por último corrija as importações:

```
configuracao.addPackage("com.cep.model").addAnnotatedClass(Usuario.  
class);
```


Classe Dao Método Salvar

Para esse capítulo iremos criar as classes que são necessárias para trabalhar com o banco de dados, nelas ficarão os métodos salvar, excluir e consultar.

Eu irei criar a classe UsuarioDao.java no pacote com.cep.dao, crie a classe dao com um nome de acordo com a sua necessidade.

Dentro da classe Dao que você criou, adicione o seguinte método:

```
public void salvar(Usuario usuario){
    try{
        Session session =
        Conexao.getSessionFactory().openSession();
        session.beginTransaction();
        session.merge(usuario);
        session.getTransaction().commit();
        session.close();
        System.out.println("Registro gravado com sucesso");
    }catch(Exception erro){
        System.out.println("Ocorreu o erro: " + erro);
    }
}
```

Se for necessário, altere o nome da classe na seguinte linha:

```
public void salvar(Usuario usuario){
```

Verifique que o método salvar necessita de um parâmetro, neste caso um Usuario.

O Controller recebe as informações da View, armazena no model Usuario e envia este model para o Dao que por sua vez, salva as informações no banco de dados.

Aqui utilizaremos a classe ConexaoBanco.java que criamos no capítulo anterior. Para isso basta invocar o método getSessionFactory() e em seguida solicitar a abertura da sessão, que é feita na seguinte linha:

```
Session session = Conexao.getSessionFactory().openSession();
```

Após isso, a linha seguinte indica o início da sessão e nesse caso salva no banco de dados.

```
session.beginTransaction();
```

A próxima linha solicita a inclusão ou alteração de um registro no banco de dados.

```
session.merge(usuario);
```

Caso exista um botão de alteração e outro de salvar, é só alterar de merge para save no método de salvar:

```
session.save(usuario);
```

E no método de atualização ficaria assim:

```
public void alterar(Usuario usuario){
    try{
        Session session =
        Conexao.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(usuario);
        session.getTransaction().commit();
        session.close();
        System.out.println("Registro gravado com sucesso");
    }catch(Exception erro){
        System.out.println("Ocorreu o erro: " + erro);
    }
}
```

Mudando de merge para update, e assim faria a atualização no banco de dados quando necessário.

A diferença entre os três é que update apenas atualiza, save apenas salva e merge faz as duas coisas, salva e atualiza. Além de não haver a necessidade de criar um botão a mais na sua aplicação, não é necessário criar um método a mais, por isso irei utilizar merge, mas use de acordo com a sua necessidade.

Outra parte importante no método é a que finaliza a transação e grava o registro no banco de dados. Essa parte é feita na seguinte linha:



```
session.getTransaction().commit();
```

E por último a sessão é fechada com a linha:

```
session.close();
```

Quando trabalhamos com banco de dados, podem ocorrer diversos tipos de erros como start de servidor, banco de dados não encontrado, tabela não encontrada, campo não encontrado, etc.

O Java possui um comando que nos ajuda nesse quesito que é o try catch:

```
try {  
     Tentar. Se tudo estiver certo ...  
} catch (Exception erro) {  
     Se alguma coisa der errado  
}  
    Gera uma exceção e grava o erro na variável  
}
```

Dessa forma vamos colocar todo nosso código dentro de um try. Será efetuada uma tentativa de gravação, se nada der errado o Usuario será gravado no banco de dados. Caso contrário, o comando será desviado para o catch, gerando uma exceção e o erro ficará armazenado na variável erro.

O código final do método é o seguinte:

```
public void salvar(Usuario usuario){  
    try{  
        Session session =  
        Conexao.getSessionFactory().openSession();  
        session.beginTransaction();  
        session.merge(usuario);  
        session.getTransaction().commit();  
        session.close();  
        System.out.println("Registro gravado com sucesso");  
    }catch(Exception erro){  
        System.out.println("Ocorreu o erro: " + erro);  
    }  
}
```

Classe Dao Método Excluir

O método excluir da classe Dao, geralmente iria deletar dados e ficaria assim:

```
public void excluir(Usuario usuario){
    try{
        Session session =
        Conexao.getSessionFactory().openSession();
        session.beginTransaction();
        session.delete(usuario);
        session.getTransaction().commit();
        session.close();
        System.out.println("Registro excluído com sucesso");
    }catch(Exception erro){
        System.out.println("Ocorreu o erro: " + erro);
    }
}
```

Porém, não é uma boa prática excluir dados de um banco de vez, porque isso poderia gerar até mesmo um processo dependendo dos casos, então para isso, faremos apenas a atualização dos dados e iremos escondê-los e se caso precisarmos recuperá-lo, será possível. Para isso iremos o método excluir deverá ficar assim:

```
public void excluir(Usuario usuario){
    try{
        Session session =
        Conexao.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(usuario);
        session.getTransaction().commit();
        session.close();
        System.out.println("Registro excluído com sucesso");
    }catch(Exception erro){
        System.out.println("Ocorreu o erro: " + erro);
    }
}
```

Observe que ao invés de utilizar o comando delete como no método anterior, foi utilizado o comando update, então ele irá apenas alterar o dado do registro.

Classe Dao Método Consultar

Agora iremos criar o método consultar, conforme o código abaixo. Esse será nosso último método da classe Dao:

```
public List<Usuario> consultar(String descricao){
    List<Usuario> lista = new ArrayList();
    Session session =
Conexao.getSessionFactory().openSession();
    session.beginTransaction();

    if(descricao.length() == 0){
        lista = session.createQuery(" from Usuario ").list();
    }else{
        lista = session.createQuery("from Usuario u where
u.descricao like"+"'" +descricao+"%'").list();
    }

    session.getTransaction().commit();
    session.close();

    return lista;
}
```

O método implementado necessita de um retorno que deve ser uma lista de usuários. Necessita receber como parâmetro a descrição da pesquisa. Se a no campo de pesquisa a descrição estiver vazia, então todos os registros serão exibidos na tabela da aplicação (tableView). Caso contrário, será efetuado um filtro com a descrição recebida.

Como já foi dito no tópico anterior, nosso método deverá retornar uma lista de usuários, a qual será buscada no banco de dados. E para declarar a lista devemos fazer o seguinte:

```
public List<Usuario> consultar(String descricao) {
    List<Usuario> lista = new ArrayList(); ←
    return lista; ← Retornando a Lista
}
```

Após ter feito isso iremos montar a estrutura de conexão com banco de dados:

```

public List<Usuario> consultar(String descricao) {
    List<Usuario> lista = new ArrayList();

    Session session = ConexaoBanco.getSessionFactory().openSession();
    session.beginTransaction();

    session.getTransaction().commit();
    session.close();

    return lista;
}

```

E para que a pesquisa no banco seja feita é necessária uma estrutura condicional, que é a seguinte linha de código:

```

if(descricao.length() == 0){
    lista = session.createQuery(" from Usuario ").list();
}else{
    lista = session.createQuery("from Usuario u where
u.descricao like"+" "+descricao+"%").list();
}

```

A condição `descricao.length() == 0`. Retorna o tamanho da descrição. Se for zero retorna todos os registros.

A linha:

```
lista = session.createQuery(" from Usuario ").list();
```

Retorna todos os registros. Observe que o comando `from` não está indicando a tabela e sim a classe `Usuario`.

E por último, a linha:

```
lista = session.createQuery("from Usuario u where u.descricao
like"+" "+descricao+"%").list();
```

Seleciona os registros da classe `Usuario`, apelidando a tabela de `u`, onde a descrição comece o texto digitado no campo de pesquisa.

Criando uma Interface

Antes de prosseguirmos, é necessário criarmos uma interface que irá conter alguns métodos necessários para efetuar algumas ações.

Os métodos que irei implementar podem ser modificados de acordo com a sua necessidade, assim como podem ser adicionados novos métodos ou retirados os que lhe forem desnecessários.

Se você seguiu todos os passos necessário, então seu código deverá estar mais ou menos assim:

```
package com.cep.controller;

import com.cep.dao.UsuarioDao;
import com.cep.model.Usuario;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ComboBox;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.Tooltip;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class UsuarioController implements Initializable{

    @FXML
    private Button btnSair;
    @FXML
    private Button btnCadastrar;
    @FXML
    private Button btnNovo;
    @FXML
```

```

private TextField tfDescricao;
@FXML
private TextField tfEmail;
@FXML
private ComboBox<String> cbPermissoes;
@FXML
private TextField tfLogin;
@FXML
private PasswordField pfSenha;
@FXML
private TableView<Usuario> tabelaUsuario;
@FXML
private TextField tfPesquisa;
@FXML
private Button btnExcluir;
@FXML
private TextField tfId;
@FXML
private CheckBox ckAtivo;

@Override
public void initialize(URL url, ResourceBundle rb) {
}

@FXML
private void fecharJanela(ActionEvent event) {
}

@FXML
private void limparCamposFormulario(ActionEvent event) {
}

@FXML
private void cadastrarRegistro(ActionEvent event) {
}

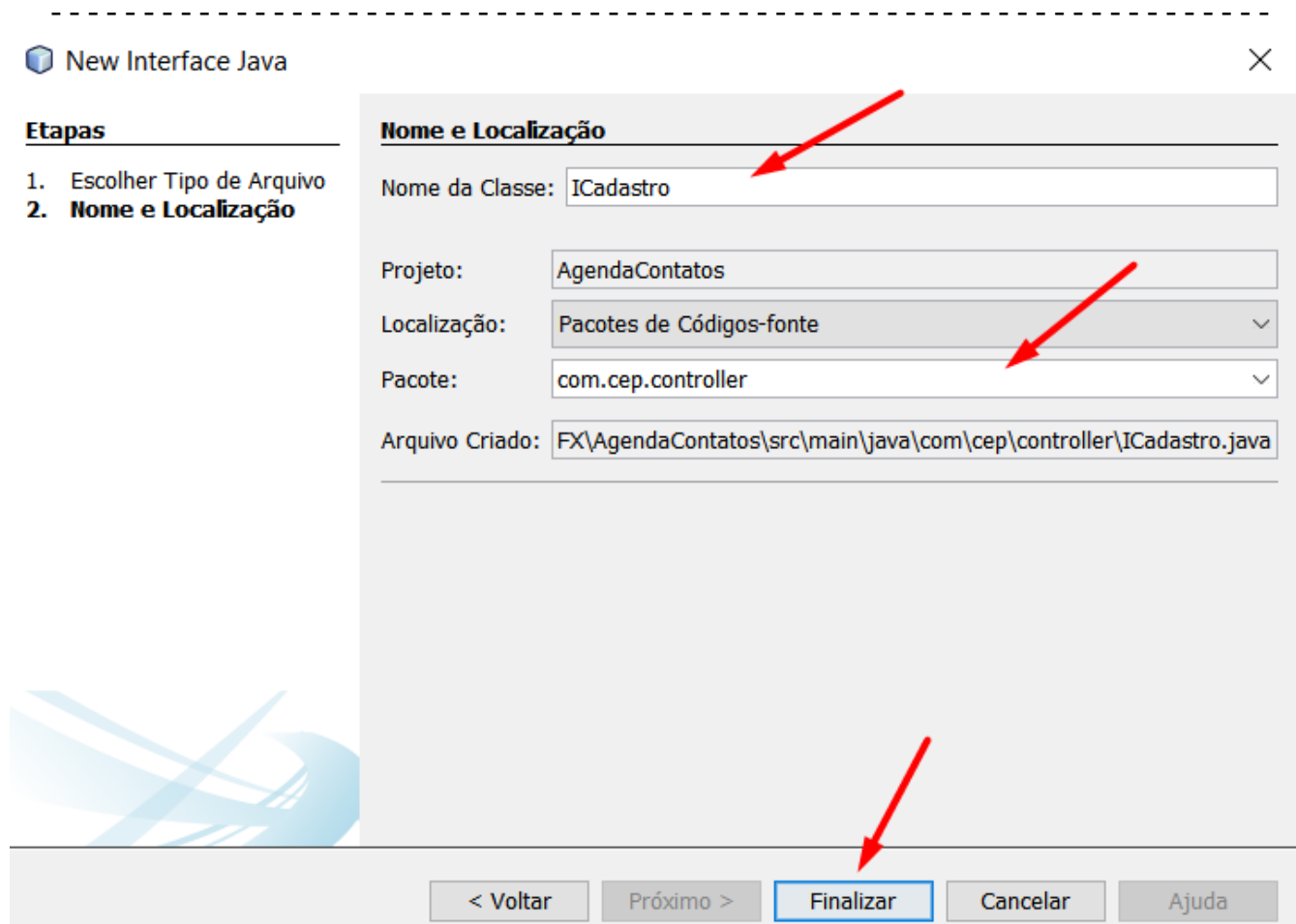
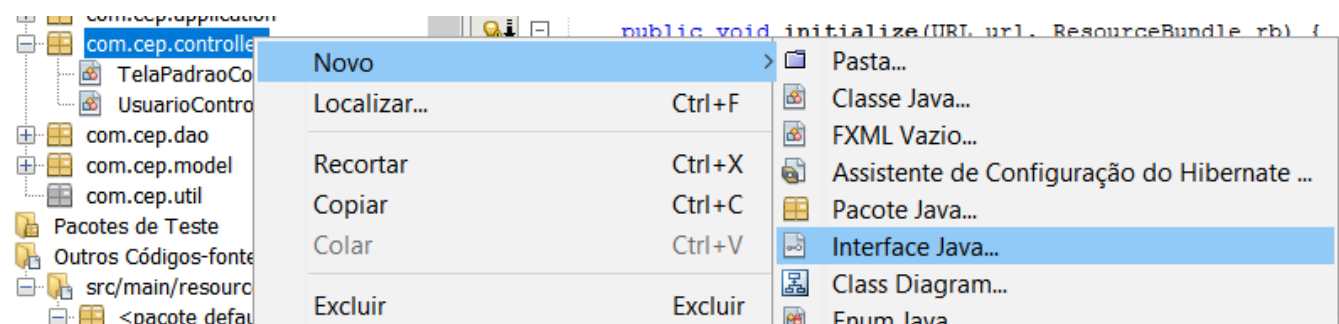
@FXML
private void excluirRegistro(ActionEvent event) {
}

@FXML
private void pesquisarRegistro(KeyEvent event) {
}

@FXML
private void clicarTabela(MouseEvent event) {
}
}

```


Agora vamos implementar a interface `ICadastro.java`, defina o nome que preferir:

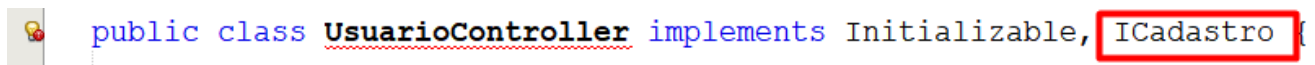


Após finalizar, iremos criar uma assinatura na interface ICadastro. Para isso adicionaremos o seguinte código:

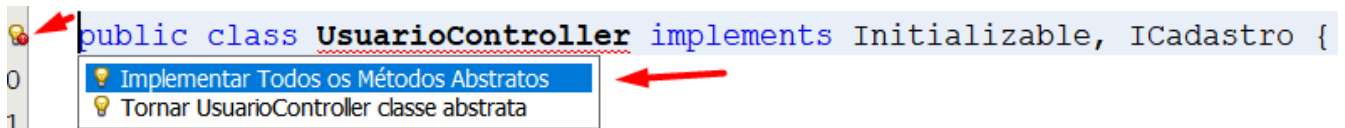
```
package com.cep.controller;

public interface ICadastro {
    public abstract void criarColunasTabela();
    public abstract void atualizarTabela();
    public abstract void setCamposFormulario();
    public abstract void limparCamposFormulario();
}
```

Retorne a classe UsuarioController e implemente a interface ICadastro. Da seguinte forma:

A screenshot of an IDE showing the class declaration: `public class UsuarioController implements Initializable, ICadastro {`. The `ICadastro` interface is highlighted with a red rectangle. A lightbulb icon is visible on the left margin.

Implemente todos os métodos abstratos:

A screenshot of an IDE showing the class declaration: `public class UsuarioController implements Initializable, ICadastro {`. A code completion menu is open, showing two options: "Implementar Todos os Métodos Abstratos" (highlighted in blue) and "Tornar UsuarioController classe abstrata". Red arrows point to the menu and the first option. Line numbers 0 and 1 are visible on the left margin.

Observe que todos os métodos foram adicionados no final da classe. Apague todas as linhas throw que estão dentro dos métodos criados.

Salvando registros

Para salvarmos registros precisaremos criar o botão salvar e depois precisaremos criar a ação que esse botão irá efetuar. E para que essa ação possa ser feita, usaremos listas. Para criarmos uma lista, faremos o seguinte. Devemos ir até o método initialize, e antes dele colar o seguinte código:

```
Long id;
private UsuarioDao dao = new UsuarioDao();
private ObservableList<Usuario> observableList =
FXCollections.observableArrayList();
private List<Usuario> listaUsuarios;
private Usuario usuarioSelecionado = new Usuario();
```

Feito isso, podemos criar o código do cadastro, que será feito após o fechamento do método initialize:

```
@FXML
private void cadastrarRegistro(ActionEvent event) {
    Usuario usuario = new Usuario();
    if(usuarioSelecionado.getId() != null){
        usuario.setId(usuarioSelecionado.getId());
    }
    usuario.setDescricao(tfDescricao.getText());
    usuario.setPermissoes(cbPermissoes.getSelectionModel().getSelectedItem());
    usuario.setEmail(tfEmail.getText());
    usuario.setLogin(tfLogin.getText());
    usuario.setSenha(pfSenha.getText());
    usuario.setAtivo(ckAtivo.isSelected());

    dao.salvar(usuario);

    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Cadastro Usuário");
    alert.setHeaderText("Informação");
    alert.setContentText("Usuário salvo com sucesso!");
    alert.showAndWait();

    limparCamposFormulario();
    atualizarTabela();
}
```

Observação, altere o nome dos campos, dos métodos e das ações de acordo com a sua necessidade.

Excluindo Registros

Agora veremos como excluir um registro, mas sem excluí-lo de vez do banco de dados como foi dito no capítulo em que criamos o método excluir na classe Dao.

O que faremos será apenas uma alteração do registro para que ele não apareça na tableView. Então mãos à obra!

Para fazermos a exclusão precisaremos criar o botão excluir e em seguida adicionar uma ação a ele, também é necessário que no formulário exista um checkbox para ativar e desativar o usuário. Feito isso copie o seguinte código e adicione-o após o fechamento da ação do botão salvar:

```
@FXML
private void excluirRegistro(ActionEvent event) {
    usuarioSelecionado.setAtivo(false);

    dao.excluir(usuarioSelecionado);

    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Exclusão de Usuário");
    alert.setHeaderText("Informação");
    alert.setContentText("Usuário excluído com sucesso!");
    alert.showAndWait();

    limparCamposFormulario();
    atualizarTabela();
}
```

A linha:

```
usuarioSelecionado.setAtivo(false);
```

Fará com que o usuário selecionado seja desativado através do checkbox. Você entenderá melhor no próximo capítulo, em que os dados são setados na tableView.

Setando dados no formulário

Para que seja possível setar os dados no formulário, é necessário adicionar uma ação à tableView. O tipo de ação será um On Mouse Clicked. Defina um nome para a ação de acordo com a sua necessidade. Feito isso, criaremos o seguinte método:

```
@FXML
private void clicarTabela(MouseEvent event) {
    setCamposFormulario();
}
```

Ele chama o método setCamposFormulario() quando algum registro é clicado na tableView. Logo após isso, iremos criar o método para setar os dados do registro no formulário:

```
@Override
public void setCamposFormulario() {
    usuarioSelecionado =
tabelaUsuario.getItems().get(tabelaUsuario.getSelectionModel().getSelectedIndex());
    tfId.setText(Long.toString(usuarioSelecionado.getId()));
    tfDescricao.setText(usuarioSelecionado.getDescricao());
    tfEmail.setText(usuarioSelecionado.getEmail());
    tfLogin.setText(usuarioSelecionado.getLogin());
    cbPermissoes.setValue(usuarioSelecionado.getPermissoes());
    pfSenha.setText(usuarioSelecionado.getSenha());
    ckAtivo.setSelected(usuarioSelecionado.getAtivo());
}
```

Verifique o nome dos campos do seu formulário e adicione de acordo com a sua necessidade.

Atualizar tabela

Para atualizarmos a tabela, foi criado o método abstrato `atualizarTabela`, que veio da interface `Icadastro`. Antes de continuar, verifique o nome da sua interface e identifique o método de atualização dentro dela, feito isso copie o seguinte código e cole após o fechamento do método `initialize`:

```
@Override
public void atualizarTabela() {
    observableList.clear();
    listaUsuarios = dao.consultar(tfPesquisa.getText());
    for(Usuario u : listaUsuarios){
        if(u.getAtivo() == true){
            observableList.add(u);
        }else{

        }
    }

    tabelaUsuario.getItems().setAll(observableList);
    tabelaUsuario.getSelectionModel().selectFirst();
}
```

A estrutura condicional:

```
if(u.getAtivo() == true){
    observableList.add(u);
}else{

}
```

Tem a função de verificar se o registro está ativado ou desativado. Se ele estiver ativado, então ele será setado na `tableView`, se não estiver, ele não aparecerá. Essa função é quase a mesma que o excluir registro.

Para que o método `atualizar` possa ser utilizado, é necessário adicioná-lo dentro do método `initialize`:

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    atualizarTabela();
}
```

Pesquisando Registros

Para que seja possível fazer pesquisas por registros na sua aplicação, é necessário que antes seja criada uma barra de pesquisa e adicionada nela a ação On Key Pressed com o nome de sua preferência.

Feito isso, é só adicionar o seguinte código após o método initialize:

```
@FXML
private void pesquisarRegistro(KeyEvent event) {
    atualizarTabela();
}
```

O método de pesquisa vai apenas utilizar o método de atualizar a tableView, e a cada vez que uma letra for digitada, será feita uma atualização na tabela aparecendo apenas os registros correspondentes ao que está sendo digitado.

Criando Colunas na TableView

Para esse capítulo, iremos criar as colunas da tabela, para isso primeiro precisamos verificar a qual classe pertence a tabela, vá até o escopo do seu controller, após as importações e verifique a seguinte linha:

```
@FXML
private TableView<Usuario> tabelaUsuario;
```

Dentro de <> deve ser adicionado o nome da classe que irá utilizar a tabela. No meu caso será a classe Usuário.

Depois de fazer esse processo, copie o seguinte código e cole após o fechamento do método initialize:

```
@Override
public void criarColunasTabela() {
    TableColumn<Usuario, Long> colunaId = new TableColumn<>("ID");
    TableColumn<Usuario, String> colunaDescricao = new
TableColumn<>("DESCRIÇÃO");
    TableColumn<Usuario, String> colunaPermissoes = new
TableColumn<>("PERMISSÕES");
    TableColumn<Usuario, String> colunaEmail = new
TableColumn<>("EMAIL");
    TableColumn<Usuario, String> colunaLogin = new
TableColumn<>("LOGIN");
    TableColumn<Usuario, String> colunaAtivo = new
TableColumn<>("STATUS");

    tabelaUsuario.getColumns().addAll(colunaId, colunaDescricao,
colunaPermissoes, colunaEmail, colunaLogin, colunaAtivo);

    colunaId.setCellValueFactory(new PropertyValueFactory("id"));
    colunaDescricao.setCellValueFactory(new
PropertyValueFactory("descricao"));
    colunaPermissoes.setCellValueFactory(new
PropertyValueFactory("permissoes"));
    colunaEmail.setCellValueFactory(new
PropertyValueFactory("email"));
    colunaLogin.setCellValueFactory(new
PropertyValueFactory("login"));
    colunaAtivo.setCellValueFactory(new
PropertyValueFactory("ativo"));
}
```

Dentro das aspas de:

```
TableColumn<>("ID");
```


Será adicionado o título da coluna, altere de acordo com a sua necessidade.

Na linha:

```
tabelaUsuario.getColumns().addAll(colunaId, colunaDescricao,  
colunaPermissoes, colunaEmail, colunaLogin, colunaAtivo);
```

É indicada a ordem que as colunas irão ficar. E dentro de:

```
PropertyValueFactory("id")
```

É indicado o nome do atributo da classe, não o nome da coluna da tabela do banco de dados.

Limpar Campos do Formulário

Para que os campos do formulário sejam limpos, foi criado na interface `Icadastro` o método `limparCamposFormulário`. Para ele foi feito um pequeno método:

```
@Override
public void limparCamposFormulario() {
    usuarioSelecionado.setId(null);
    tfId.clear();
    tfDescricao.clear();
    tfEmail.clear();
    tfLogin.clear();
    cbPermissoes.setValue("Usuário");
    ckAtivo.setSelected(false);
    pfSenha.clear();
}
```

Verifique os campos necessários e os nomes de acordo com sua necessidade.

Adicionando Máscara nos componentes JavaFX

Para utilizar a expressão lambda para máscara, é necessário modificar o source do plugin do apache para 1.8 no arquivo Pom.xml:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <compilerArguments>
      <bootclasspath>${sun.boot.class.path}${
{path.separator}${java.home}/lib/jfxrt.jar</bootclasspath>
    </compilerArguments>
  </configuration>
</plugin>
```

Após fazer a configuração necessária, copie e cole o código abaixo no método initialize do controler que precisar utilizar a máscara:

```
tfCPF.lengthProperty().addListener((ObservableValue extends Number&gt;
observableValue, Number number, Number number2) -&gt; {
    String mascara = "###.###.###-##";
    String alphaAndDigits =
tfCPF.getText().replaceAll("[\\-\\.]", "");
    StringBuilder resultado = new StringBuilder();
    int i = 0;
    int quant = 0;

    if(number2.intValue() &gt; number.intValue()) {
        if(tfCPF.getText().length() &lt;= mascara.length()) {
            while (i&lt;mascara.length()) {
                if (quant &lt; alphaAndDigits.length()) {
                    if ("#".equals(mascara.substring(i,i+1)))
{
                        resultado.append(alphaAndDigits.substring(quant,quant+1));
                        quant++;
                    } else {
                        resultado.append(mascara.substring(i,i+1));
                    }
                }
                i++;
            }
        }
    }
}</pre
```

```

    }
    tfCPF.setText(resultado.toString());
}
if (tfCPF.getText().length() > mascara.length()) {
    tfCPF.setText(tfCPF.getText(0, mascara.length()));
}
}
});

```

Observação:

Para alterar o tipo de máscara, basta alterar o nome do componente, o formato da máscara e os caracteres nas variáveis:

```

String mascara = "###.###.###-##";
String alphaAndDigits = tfCPF.getText().replaceAll("[\\-\\.]", "");

```

Exemplo Máscara Telefone

```

tfTelefone.lengthProperty().addListener((ObservableValue<? extends
Number> observableValue, Number number, Number number2) -> {
    String mascara = "(##)####-####";
    String alphaAndDigits =
tfTelefone.getText().replaceAll("[\\(\\)\\-\\.]", "");
    StringBuilder resultado = new StringBuilder();
    int i = 0;
    int quant = 0;

    if (number2.intValue() > number.intValue()) {
        if (tfTelefone.getText().length() <=
mascara.length()) {
            while (i < mascara.length()) {
                if (quant < alphaAndDigits.length()) {
                    if
("#".equals(mascara.substring(i, i+1))) {
resultado.append(alphaAndDigits.substring(quant, quant+1));
                        quant++;
                    } else {
resultado.append(mascara.substring(i, i+1));
                    }
                }
                i++;
            }
        }
    }
}

```

```

        tfTelefone.setText(resultado.toString());
    }
    if (tfTelefone.getText().length() >
mascara.length()) {
tfTelefone.setText(tfTelefone.getText(0,mascara.length()));
    }
});

```

Exemplo Máscara Cep

```

tfCep.lengthProperty().addListener((ObservableValue<? extends
Number> observableValue, Number number, Number number2) -> {
    String mascara = "#####-###";
    String alphaAndDigits =
tfCep.getText().replaceAll("[\\-\\.]", "");
    StringBuilder resultado = new StringBuilder();
    int i = 0;
    int quant = 0;

    if (number2.intValue() > number.intValue()) {
        if (tfCep.getText().length() <= mascara.length()) {
            while (i<mascara.length()) {
                if (quant < alphaAndDigits.length()) {
                    if
("#".equals(mascara.substring(i,i+1))) {
resultado.append(alphaAndDigits.substring(quant,quant+1));
                        quant++;
                    } else {
resultado.append(mascara.substring(i,i+1));
                        }
                    }
                i++;
            }
            tfCep.setText(resultado.toString());
        }
        if (tfCep.getText().length() > mascara.length()) {
tfCep.setText(tfCep.getText(0,mascara.length()));
        }
    }
});

```

Popular dados no ComboBox

Nesse capítulo irei explicar como setar dados no controle combobox, porém primeiro de forma manual. No próximo capítulo explicarei como setar dados do banco de dados no combobox.

Para setar dados no combobox teremos que primeiro verificar qual tipo de dado será setado, para isso devemos verificar, logo após as importações a seguinte linha:

```
@FXML  
private ComboBox<String> cbPermissoes;
```

No meu caso o tipo de dado que irei utilizar é String, porém, é possível alterar dentro de <> para Integer, Float, Double ou o tipo que necessitar.

Feito isso, iremos criar um observableList antes do método initialize utilizando o seguinte comando:

```
private ObservableList<String> listaPermissoes =  
FXCollections.observableArrayList("Administrador", "Usuário");
```

Agora devemos iniciar o observableList no método initialize:

```
cbPermissoes.setItems(listaPermissoes);
```

Por último, podemos indicar qual valor será setado quando a aplicação for startada:

```
cbPermissoes.setValue("Usuário");
```

Popular dados do banco no combobox

Para podermos setar os dados do banco de dados no combobox, primeiramente precisamos ir até a classe que iremos setar e colocar a seguinte linha de código:

```
@Override
public String toString(){
    return getDescricao();
}
```

Feito isso precisamos ir até a classe que irá puxar os dados, que no meu caso é uma classe de venda e mapear o atributo que será puxado da seguinte forma:

```
@OneToOne
private Usuario usuario;
```

O usuario será do tipo Usuario mesmo, porque assim será possível pegar todos os atributos da classe Usuario.

Após fazer essas configurações importantes, devemos ir até o controller que irá utilizar o tal combobox e verificar logo abaixo das importações pela seguinte linha de código:

```
@FXML
private ComboBox<Usuario> cbUsuario;
```

Lembre-se que dentro de <> deverá ser colocado o tipo de dado, que no nosso caso é Usuario. Em seguida iremos criar uma observableList de usuários antes do método initialize:

```
private ObservableList<Usuario> obsListUsuario =
FXCollections.observableArrayList();
```

Dentro de ObservableList<> será adicionada a classe que o combobox fará referência, que no meu caso é a classe Usuario.

Agora iremos criar o método que irá popular dados no combobox utilizando o seguinte código:

```
private void popularComboBoxUsuario(){
    List<Usuario> list = new ArrayList<>();
    Session session = Conexao.getSessionFactory().openSession();
    session.beginTransaction();
    list = session.createQuery(" from Usuario").getResultList();
}
```

```
session.getTransaction().commit();
session.close();

for(Usuario usuario : list){
    if(usuario.getAtivo() == true){
        obsListUsuario.add(usuario);
    }
}
cbUsuario.setItems(obsListUsuario);
}
```

Por último devemos iniciar o método
popularComboBoxUsuario() no método initialize:

```
popularComboBoxUsuario();
```


Adicionando Tooltip

Agora iremos trabalhar com um componente que é muito interessante. É o tooltip. Esse componente exibe um pequeno texto quando passamos o mouse sobre uma caixa de texto, botão, combobox ou o componente java que desejar:



Como você pode ver, utilizei um tooltip no campo descrição, e quando passei o mouse por cima do campo, ele exibiu a mensagem "Digite o nome do usuário...".

Esse componente facilita muito quando o usuário do nosso programa não sabe o que fazer ou qual tipo de dado inserir em um campo.

Mas vamos lá! O processo é bem simples. Primeiramente precisamos instanciar um objeto da classe tooltip da seguinte forma:

```
Tooltip ttDescricao = new Tooltip("Digite o nome do usuário...");
```

Dentro das aspas ficará a mensagem que será exibida. Em seguida, podemos definir o tipo de fonte que será utilizada:

```
ttDescricao.setFont(new Font("Arial", 14));
```

Note que chamei o objeto ttDescricao e setei nele um tipo de fonte, caso deseje utilizar outro tipo de fonte é só alterar dentro das aspas em:

```
Font("Arial", 14)
```

Caso queira alterar o tamanho é só modificar a numeração para o valor que deseje.

Por último, e não menos importante, iremos setar o objeto ttDescricao no componente que desejamos utilizar o tooltip:

```
tfDescricao.setToolTipText(ttDescricao);
```

No meu caso utilizei num textfield com o nome tfDescricao, mas você pode utilizar no componente que desejar.

O método tooltip logo abaixo veio da interface Icadastro. Copie e cole após o fechamento do método initialize e inicie ele dentro do método initialize. Verifique o capítulo que adicionei todos os códigos para entender melhor:

```
@Override
public void adicionarTooltip() {
    Tooltip ttId = new Tooltip("Campo Id não pode ser preenchido!");
    ttId.setFont(new Font("Arial", 14));
    tfId.setToolTipText(ttId);

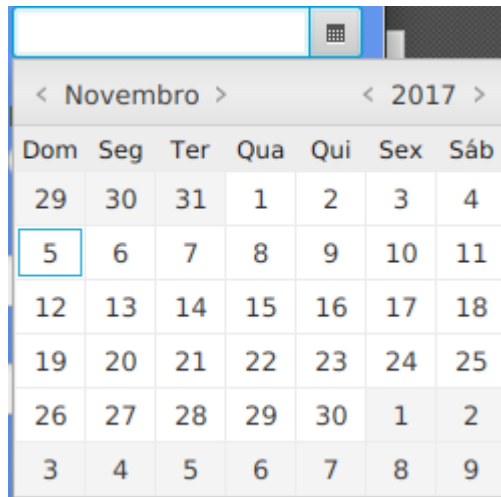
    Tooltip ttDescricao = new Tooltip("Digite o nome do
usuário...");
    ttDescricao.setFont(new Font("Arial", 14));
    tfDescricao.setToolTipText(ttDescricao);

    Tooltip ttPermissoes = new Tooltip("Selecione o nível de
permissão do usuário...");
    ttPermissoes.setFont(new Font("Arial", 14));
    cbPermissoes.setToolTipText(ttPermissoes);
}
```

```
        Tooltip ttEmail = new Tooltip("Email para recuperação de  
senha...");  
        ttEmail.setFont(new Font("Arial", 14));  
        tfEmail.setToolTipText(ttEmail);  
  
        Tooltip ttLogin = new Tooltip("Login para acessar o  
sistema...");  
        ttLogin.setFont(new Font("Arial", 14));  
        tfLogin.setToolTipText(ttLogin);  
  
        Tooltip ttSenha = new Tooltip("Senha para acessar o  
sistema...");  
        ttSenha.setFont(new Font("Arial", 14));  
        pfSenha.setToolTipText(ttSenha);  
  
        Tooltip ttAtivo = new Tooltip("Necessário marcar para ativar o  
usuário!");  
        ttAtivo.setFont(new Font("Arial", 14));  
        ckAtivo.setToolTipText(ttAtivo);  
    }
```

DatePicker

Um componente que também é interessante de trabalhar é o datepicker:



Ele nos permite selecionar a data através de um calendário sem a necessidade de ficar digitando e nem mesmo criar máscara para data.

Podemos até mesmo pegar a data atual e exibi-la no datepicker. Irei criar um datepicker para data de nascimento, mas crie de acordo com a sua necessidade. Então mãos a obra!

Após criar o componente, preciso definir o tipo de dado dele como LocalDate:

```
@Column(name = "nascimento", nullable = false)
private LocalDate nascimento;
```

Em seguida, eu posso ir até o método initialize e setar a data atual com o seguinte comando:

```
dpNascimento.setValue(LocalDate.now());
```

Simple, não é mesmo?

Agora irei mostrar como salvar a data, para isso irei utilizar o método cadastrar:

```
Cliente cliente = new Cliente();
cliente.setNascimento(dpNascimento.getValue());
```

O atributo `getValue` irá pegar a data digitada no datepicker e setar no objeto cliente e assim salvar no banco de dados.

Para exibir a data que foi salva no banco de dados novamente no datepicker, podemos utilizar o método `setCamposFormulario`:

```
dpNascimento.setValue(clienteSelecionado.getNascimento());
```

Para setar o valor do banco de dados no datepicker utilizei um `observableList` de cliente selecionado. Adicionarei o código completo no capítulo Arquivo `ClienteController.java`.

Adicionando ícone no título da janela

Uma coisa que deixa o programa com uma cara melhor são os ícones. Eles fazem com que o usuário tenha mais facilidade para utilizar o software. Uma imagem vale mais que mil palavras, não é mesmo?

Neste capítulo irei mostrar como adicionar as tais ícones na barra de título da sua aplicação. Para isso, crie na pasta Outros Códigos-Fonte, e dentro de src/main/resources, crie um pacote com o nome icons, ou o nome que preferir. Assim sua aplicação ficará mais organizada e mais fácil de fazer uma manutenção futura se for preciso.

Após criar o pacote para ícones, baixe os ícones que desejar, copie e cole dentro do pacote.

Agora que está tudo pronto vamos finalmente ao código. Abra o arquivo MainApp.java que fica dentro de Pacotes de Códigos-Fonte.

O código que iremos usar é bem pequeno. Copie e cole dentro do método start:

```
Image applicationIcon = new  
Image(getClass().getResourceAsStream("/icons/Shopping cart.png"));  
stage.getIcons().add(applicationIcon);
```

Dentro de:

```
getResourceAsStream("/icons/Shopping cart.png")
```

Indique o caminho do pacote de ícones que você criou e indique o nome do ícone com a extensão que ele utiliza. Recomendo utilizar sempre extensões do tipo .png, pois elas não tem fundo branco igual jpg, jpeg e outras.

Códigos Utilizados

Logo abaixo irei disponibilizar os códigos por completo, altere e utilize de acordo com a sua necessidade. Para melhor organização, sugiro a criação dos pacotes de MVC e Dao dentro de Pacotes de Códigos-Fonte. Pois isso facilitará o entendimento e a manutenção do seu código com mais facilidade.

Se preferir, poderá utilizar o gerenciador de pacotes Maven para criar suas aplicações. Se estiver tudo certo pra você, é só prosseguir.

Arquivo hibernate.cfg

Esse arquivo deve ficar dentro de Outros Códigos-Fonte, na pasta src/main/resources. Se quiser, crie um pacote e adicione o hibernate.cfg dentro dele:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property
>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/sistemave
ndas?zeroDateTimeBehavior=convertToNull</property>
    <property name="hibernate.connection.username">root</property>
    <!-- VISUALIZAR OS CÓDIGOS SQL NA COMPILAÇÃO -->
    <property name="hibernate.show_sql">true</property>
    <!-- FORMATAR A EXIBIÇÃO DOS DADOS ACIMA -->
    <property name="hibernate.format_sql">true</property>
    <!-- MODO DE CRIAÇÃO DAS TABELAS -->
    <property name="hbm2ddl.auto">update</property>
  </session-factory>
</hibernate-configuration>
```


Arquivo Conexao.java

Esse arquivo deve ser criado em Pacotes de Código-Fonte, Dentro do pacote controller adicione a classe de conexão:

```
package com.cep.dao;

import com.cep.model.Cliente;
import com.cep.model.Login;
import com.cep.model.Pagamento;
import com.cep.model.Produto;
import com.cep.model.Usuario;
import com.cep.model.Venda;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

public class Conexao {
    private static SessionFactory conexao = null;
    private static Configuration configuracao;

    private static SessionFactory buildSessionFactory(){
        //----> Objeto que armazena as configurações de conexão
        configuracao = new Configuration().configure();

        configuracao.setProperty("hibernate.connection.username",
"root");
        configuracao.setProperty("hibernate.connection.password",
"");
        configuracao.addPackage("com.cep.model").addAnnotatedClass(Usuario.
class);
        configuracao.addPackage("com.cep.model").addAnnotatedClass(Clientes.
class);

        conexao = configuracao.buildSessionFactory();
        return conexao;
    }

    public static SessionFactory getSessionFactory(){
        if(conexao == null){
            conexao = buildSessionFactory();
        }
        return conexao;
    }
}
```

Arquivo Classe Usuario.java

Esse arquivo deve ficar dentro de Pacotes de Códigos-Fonte no pacote model:

```
package com.cep.model;

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

@Entity
@Table(name="usuario")
public class Usuario{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    protected Long id;
    @Column(name = "descricao", length = 100, nullable = false)
    protected String descricao;
    @Column(name = "permissoes", length = 20, nullable = false)
    protected String permissoes;
    @Column(name = "email", length = 100, nullable = false)
    protected String email;
    @Column(name = "login", length = 100, nullable = false)
    protected String login;
    @Column(name = "senha", length = 8, nullable = false)
    protected String senha;
    @Column(name = "ativo", nullable = true)
    private Boolean ativo;

    public Long getId() {
        return id;
    }

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

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }
}
```

```
public String getPermissoes() {
    return permissoes;
}

public void setPermissoes(String permissoes) {
    this.permissoes = permissoes;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

public Boolean getAtivo() {
    return ativo;
}

public void setAtivo(Boolean ativo) {
    this.ativo = ativo;
}

@Override
public String toString(){
    return getDescricao();
}
}
```

Arquivo UsuarioDao.java

Esse arquivo deve ficar dentro de Pacotes de Código-Fonte, no pacote Dao:

```
package com.cep.dao;

import com.cep.model.Usuario;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;

public class UsuarioDao {
    public void salvar(Usuario usuario){
        try{
            Session session =
Conexao.getSessionFactory().openSession();
            session.beginTransaction();
            session.merge(usuario);
            session.getTransaction().commit();
            session.close();
            System.out.println("Registro gravado com sucesso");
        }catch(Exception erro){
            System.out.println("Ocorreu o erro: " + erro);
        }
    }

    public void excluir(Usuario usuario){
        try{
            Session session =
Conexao.getSessionFactory().openSession();
            session.beginTransaction();
            session.update(usuario);
            session.getTransaction().commit();
            session.close();
            System.out.println("Registro excluído com sucesso");
        }catch(Exception erro){
            System.out.println("Ocorreu o erro: " + erro);
        }
    }

    public List<Usuario> consultar(String descricao){
        List<Usuario> lista = new ArrayList();
        Session session =
Conexao.getSessionFactory().openSession();
        session.beginTransaction();

        if(descricao.length() == 0){
            lista = session.createQuery(" from Usuario ").list();
        }else{
```

```
        lista = session.createQuery("from Usuario u where  
u.descricao like"+" "+descricao+"%").list();  
    }  
  
    session.getTransaction().commit();  
    session.close();  
  
    return lista;  
}  
}
```

Arquivo UsuarioController.java

Esse arquivo deve ficar dentro de Pacotes de Códigos-Fonte, no pacote controller:

```
package com.cep.controller;

import com.cep.dao.UsuarioDao;
import com.cep.model.Usuario;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ComboBox;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.Tooltip;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class UsuarioController implements Initializable, ICadastro
{
    @FXML
    private Button btnSair;
    @FXML
    private Button btnCadastrar;
    @FXML
    private Button btnNovo;
    @FXML
    private TextField tfDescricao;
    @FXML
    private TextField tfEmail;
    @FXML
    private ComboBox<String> cbPermissoes;
    @FXML
    private TextField tfLogin;
```

```

@FXML
private PasswordField pfSenha;
@FXML
private TableView<Usuario> tabelaUsuario;
@FXML
private TextField tfPesquisa;
@FXML
private Button btnExcluir;
@FXML
private TextField tfId;
@FXML
private CheckBox ckAtivo;

Long id;
private UsuarioDao dao = new UsuarioDao();
private ObservableList<Usuario> observableList =
FXCollections.observableArrayList();
private List<Usuario> listaUsuarios;
private Usuario usuarioSelecionado = new Usuario();
private ObservableList<String> listaPermissoes =
FXCollections.observableArrayList("Administrador", "Usuário");

@Override
public void initialize(URL url, ResourceBundle rb) {
    cbPermissoes.setItems(listaPermissoes);
    cbPermissoes.setValue("Usuário");
    ckAtivo.setSelected(false);
    criarColunasTabela();
    atualizarTabela();
    adicionarTooltip();
}

@FXML
private void cadastrarRegistro(ActionEvent event) {
    Usuario usuario = new Usuario();
    if(usuarioSelecionado.getId() != null){
        usuario.setId(usuarioSelecionado.getId());
    }
    usuario.setDescricao(tfDescricao.getText());

usuario.setPermissoes(cbPermissoes.getSelectionModel().getSelectedItem());

    usuario.setEmail(tfEmail.getText());
    usuario.setLogin(tfLogin.getText());
    usuario.setSenha(pfSenha.getText());
    usuario.setAtivo(ckAtivo.isSelected());
    dao.salvar(usuario);
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Cadastro Usuário");

```

```

        alert.setHeaderText("Informação");
        alert.setContentText("Usuário salvo com sucesso!");
        alert.showAndWait();
        limparCamposFormulario();
        atualizarTabela();
    }

    @FXML
    private void pesquisarRegistro(KeyEvent event) {
        atualizarTabela();
    }

    @FXML
    private void excluirRegistro(ActionEvent event) {
        usuarioSelecionado.setAtivo(false); desativar o usuário
        dao.excluir(usuarioSelecionado);
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Exclusão de Usuário");
        alert.setHeaderText("Informação");
        alert.setContentText("Usuário excluído com sucesso!");
        alert.showAndWait();

        limparCamposFormulario();
        atualizarTabela();
    }

    @FXML
    private void limparCamposFormulario(ActionEvent event) {
        limparCamposFormulario();
    }

    @FXML
    private void fecharJanela(ActionEvent event) {
        Stage stage = (Stage) btnSair.getScene().getWindow();
        stage.close();
    }

    @FXML
    private void clicarTabela(MouseEvent event) {
        setCamposFormulario();
    }

    @Override
    public void criarColunasTabela() {
        TableColumn<Usuario, Long> colunaId = new
        TableColumn<>("ID");
        TableColumn<Usuario, String> colunaDescricao = new
        TableColumn<>("DESCRIÇÃO");
    }

```



```

        TableColumn<Usuario, String> colunaPermissoes = new
TableColumn<>("PERMISSÕES");
        TableColumn<Usuario, String> colunaEmail = new
TableColumn<>("EMAIL");
        TableColumn<Usuario, String> colunaLogin = new
TableColumn<>("LOGIN");
        TableColumn<Usuario, String> colunaAtivo = new
TableColumn<>("STATUS");

        tabelaUsuario.setColumns().addAll(colunaId,
colunaDescricao, colunaPermissoes, colunaEmail, colunaLogin,
colunaAtivo);

        colunaId.setCellValueFactory(new
PropertyValueFactory("id"));
        colunaDescricao.setCellValueFactory(new
PropertyValueFactory("descricao"));
        colunaPermissoes.setCellValueFactory(new
PropertyValueFactory("permissoes"));
        colunaEmail.setCellValueFactory(new
PropertyValueFactory("email"));
        colunaLogin.setCellValueFactory(new
PropertyValueFactory("login"));
        colunaAtivo.setCellValueFactory(new
PropertyValueFactory("ativo"));
    }

    @Override
    public void atualizarTabela() {
        observableList.clear();
        listaUsuarios = dao.consultar(tfPesquisa.getText());
        for(Usuario u : listaUsuarios){
            if(u.getAtivo() == true){
                observableList.add(u);
            }else{

            }
        }
        tabelaUsuario.getItems().setAll(observableList);
        tabelaUsuario.getSelectionModel().selectFirst();
    }

    @Override
    public void setCamposFormulario() {
        usuarioSelecionado =
tabelaUsuario.getItems().get(tabelaUsuario.getSelectionModel().getSel
ectedIndex());
        tfId.setText(Long.toString(usuarioSelecionado.getId()));
    }

```

```
tfDescricao.setText(usuarioSelecioneado.getDescricao());
tfEmail.setText(usuarioSelecioneado.getEmail());
tfLogin.setText(usuarioSelecioneado.getLogin());

cbPermissoes.setValue(usuarioSelecioneado.getPermissoes());
pfSenha.setText(usuarioSelecioneado.getSenha());
ckAtivo.setSelected(usuarioSelecioneado.getAtivo());
    }

    @Override
    public void limparCamposFormulario() {
        usuarioSelecioneado.setId(null);
        tfId.clear();
        tfDescricao.clear();
        tfEmail.clear();
        tfLogin.clear();
        cbPermissoes.setValue("Usuário");
        ckAtivo.setSelected(false);
        pfSenha.clear();
    }
}
```

Arquivo da Interface Icadastro.java

Esse arquivo também deve ficar dentro de Pacotes de Códigos-Fonte, no pacote controller:

```
package com.cep.controller;

public interface ICadastro {
    public abstract void criarColunasTabela();
    public abstract void atualizarTabela();
    public abstract void setCamposFormulario();
    public abstract void limparCamposFormulario();
    public abstract void adicionarTooltip();
}
```

Arquivo ClienteController.java

```
package com.cep.controller;

import com.cep.dao.ClienteDao;
import com.cep.model.Cliente;
import java.net.URL;
import java.time.LocalDate;
import java.util.List;
import java.util.ResourceBundle;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ComboBox;
import javafx.scene.control.DatePicker;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.Tooltip;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.KeyEvent;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ClienteController implements Initializable, ICadastro{

    @FXML
    private Button btnSair;
    @FXML
    private TextField tfId;
    @FXML
    private Button btnCadastrar;
    @FXML
    private Button btnNovo;
    @FXML
    private TextField tfDescricao;
    @FXML
    private DatePicker dpNascimento;
    @FXML
    private TextField tfEndereco;
    @FXML
    private TextField tfComplemento;
```

```

@FXML
private TextField tfNumero;
@FXML
private TextField tfBairro;
@FXML
private TextField tfCidade;
@FXML
private TextField tfCep;
@FXML
private ComboBox<String> cbUf;
@FXML
private TextField tfTelefone;
@FXML
private TextField tfPesquisa;
@FXML
private Button btnExcluir;
@FXML
private TableView<Cliente> tabelaCliente;
@FXML
private CheckBox ckClienteAtivo;

Long id;
private ClienteDao dao = new ClienteDao();
private ObservableList<Cliente> observableList =
FXCollections.observableArrayList();
private List<Cliente> listaClientes;
private Cliente clienteSelecionado = new Cliente();
private ObservableList<String> listaUf =
FXCollections.observableArrayList("AC", "AL", "AP", "AM", "BA", "CE",
"DF", "ES", "GO", "MA", "MT", "MS", "MG", "PA", "PB", "PR", "PE",
"PI", "RJ", "RN", "RS", "RO", "RR", "SC", "SP", "SE", "TO");
@Override
public void initialize(URL url, ResourceBundle rb) {
    cbUf.setItems(listaUf);
    cbUf.setValue("MG");
    dpNascimento.setValue(LocalDate.now());
    criarColunasTabela();
    atualizarTabela();
    adicionarTooltip();
    mascaraControles();
}

@FXML
private void cadastrar(ActionEvent event) {
    Cliente cliente = new Cliente();
    String descricao = tfDescricao.getText();
    String endereco = tfEndereco.getText();
    String numero = tfNumero.getText();
    String bairro = tfBairro.getText();

```

```

        String cidade = tfCidade.getText();
        String cep = tfCep.getText();
        if(descricao.length() == 0 || endereco.length() == 0 ||
numero.length() == 0 || bairro.length() == 0 || cidade.length() == 0
|| cep.length() == 0){
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Cadastro Venda");
            alert.setHeaderText("Atenção");
            alert.setContentText(String.format("Campo obrigatório
está vazio!"));
            alert.showAndWait();
        }else{
            if(clienteSelecionado.getId() != null){
                cliente.setId(clienteSelecionado.getId());
            }
            cliente.setDescricao(tfDescricao.getText());
            cliente.setNascimento(dpNascimento.getValue());
            cliente.setEndereco(tfEndereco.getText());
            cliente.setNumero(Integer.parseInt(tfNumero.getText()));
            cliente.setComplemento(tfComplemento.getText());
            cliente.setBairro(tfBairro.getText());
            cliente.setCidade(tfCidade.getText());
            cliente.setCep(tfCep.getText());

cliente.setUf(cbUf.getSelectionModel().getSelectedItem());
cliente.setTelefone(tfTelefone.getText());
cliente.setAtivo(ckClienteAtivo.isSelected());

        dao.salvar(cliente);

        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Cadastro Cliente");
        alert.setHeaderText("Informação");
        alert.setContentText("Cliente salvo com sucesso!");
        alert.showAndWait();

        limparCamposFormulario();
        atualizarTabela();
    }
}

@FXML
private void limparCamposFormulario(ActionEvent event) {
    limparCamposFormulario();
}

@FXML
private void pesquisarRegistro(KeyEvent event) {
    atualizarTabela();
}

```

```
}
```

```
@FXML
```

```
private void excluir(ActionEvent event) {  
    clienteSelecionado.setAtivo(false);  
    dao.excluir(clienteSelecionado);  
  
    Alert alert = new Alert(Alert.AlertType.INFORMATION);  
    alert.setTitle("Exclusão de Cliente");  
    alert.setHeaderText("Informação");  
    alert.setContentText("Cliente excluído com sucesso!");  
    alert.showAndWait();  
  
    limparCamposFormulario();  
    atualizarTabela();  
}
```

```
@FXML
```

```
private void fecharJanela(ActionEvent event) {  
    Stage stage = (Stage) btnSair.getScene().getWindow();  
    stage.close();  
}
```

```
@FXML
```

```
private void clicarTabela(MouseEvent event) {  
    setCamposFormulario();  
}
```

```
@Override
```

```
public void criarColunasTabela() {  
    TableColumn<Cliente, Long> colunaId = new  
TableColumn<>("ID");  
    TableColumn<Cliente, String> colunaDescricao = new  
TableColumn<>("DESCRIÇÃO");  
    TableColumn<Cliente, LocalDate> colunaNascimento = new  
TableColumn<>("NASCIMENTO");  
    TableColumn<Cliente, String> colunaEndereco = new  
TableColumn<>("ENDERECO");  
    TableColumn<Cliente, Integer> colunaNumero = new  
TableColumn<>("NUMERO");  
    TableColumn<Cliente, String> colunaComplemento = new  
TableColumn<>("COMPLEMENTO");  
    TableColumn<Cliente, String> colunaBairro = new  
TableColumn<>("BAIRRO");  
    TableColumn<Cliente, String> colunaCidade = new  
TableColumn<>("CIDADE");  
    TableColumn<Cliente, Long> colunaCep = new  
TableColumn<>("CEP");
```

```

        TableColumn<Cliente, String> colunaUf = new
TableColumn<>("UF");
        TableColumn<Cliente, Long> colunaTelefone = new
TableColumn<>("TELEFONE");
        TableColumn<Cliente, Boolean> colunaClienteAtivo = new
TableColumn<>("STATUS");

        tabelaCliente.setColumns().addAll(colunaId, colunaDescricao,
colunaNascimento, colunaEndereco, colunaNumero, colunaComplemento,
colunaBairro, colunaCidade, colunaCep, colunaUf, colunaTelefone,
colunaClienteAtivo);

        colunaId.setCellValueFactory(new PropertyValueFactory("id"));
        colunaDescricao.setCellValueFactory(new
PropertyValueFactory("descricao"));
        colunaNascimento.setCellValueFactory(new
PropertyValueFactory("nascimento"));
        colunaEndereco.setCellValueFactory(new
PropertyValueFactory("endereco"));
        colunaNumero.setCellValueFactory(new
PropertyValueFactory("numero"));
        colunaComplemento.setCellValueFactory(new
PropertyValueFactory("complemento"));
        colunaBairro.setCellValueFactory(new
PropertyValueFactory("bairro"));
        colunaCidade.setCellValueFactory(new
PropertyValueFactory("cidade"));
        colunaCep.setCellValueFactory(new
PropertyValueFactory("cep"));
        colunaUf.setCellValueFactory(new PropertyValueFactory("uf"));
        colunaTelefone.setCellValueFactory(new
PropertyValueFactory("telefone"));
        colunaClienteAtivo.setCellValueFactory(new
PropertyValueFactory("ativo"));
    }

    @Override
    public void atualizarTabela() {
        observableList.clear();
        listaClientes = dao.consultar(tfPesquisa.getText());

        for(Cliente c : listaClientes){
            if(c.getAtivo() == true){
                observableList.add(c);
            }
        }

        tabelaCliente.getItems().setAll(observableList);
    }

```



```

        tabelaCliente.getSelectionModel().selectFirst();
    }

    @Override
    public void setCamposFormulario() {
        clienteSelecionado =
tabelaCliente.getItems().get(tabelaCliente.getSelectionModel().getSel
ectedIndex());
        tfId.setText(Long.toString(clienteSelecionado.getId()));
        tfDescricao.setText(clienteSelecionado.getDescricao());
        dpNascimento.setValue(clienteSelecionado.getNascimento());
        tfEndereco.setText(clienteSelecionado.getEndereco());

        tfNumero.setText(Integer.toString(clienteSelecionado.getNumero()));
        tfComplemento.setText(clienteSelecionado.getComplemento());
        tfBairro.setText(clienteSelecionado.getBairro());
        tfCidade.setText(clienteSelecionado.getCidade());
        tfCep.setText(clienteSelecionado.getCep());
        cbUf.setValue(clienteSelecionado.getUf());
        tfTelefone.setText(clienteSelecionado.getTelefone());
        ckClienteAtivo.setSelected(clienteSelecionado.getAtivo());
    }

    @Override
    public void limparCamposFormulario() {
        clienteSelecionado.setId(null);
        tfId.clear();
        tfDescricao.clear();
        dpNascimento.setValue(LocalDate.now());
        tfEndereco.clear();
        tfNumero.clear();
        tfComplemento.clear();
        tfBairro.clear();
        tfCidade.clear();
        tfCep.clear();
        cbUf.setValue("MG");
        tfTelefone.clear();
        ckClienteAtivo.setSelected(false);
    }

    @Override
    public void adicionarTooltip() {
        Tooltip ttId = new Tooltip("Campo Id não pode ser
preenchido!");
        ttId.setFont(new Font("Arial", 14));
        tfId.setTooltip(ttId);
    }

```

```
Tooltip ttDescricao = new Tooltip("Digite o nome do cliente.  
Campo obrigatório!");  
ttDescricao.setFont(new Font("Arial", 14));  
tfDescricao.setTooltip(ttDescricao);  
  
Tooltip ttNascimento = new Tooltip("Data de nascimento do  
cliente...");  
ttNascimento.setFont(new Font("Arial", 14));  
dpNascimento.setTooltip(ttNascimento);  
  
Tooltip ttEndereco = new Tooltip("Digite o endereço do  
cliente. Campo obrigatório!");  
ttEndereco.setFont(new Font("Arial", 14));  
tfEndereco.setTooltip(ttEndereco);  
  
Tooltip ttNumero = new Tooltip("Número de residência do  
cliente. Campo obrigatório!");  
ttNumero.setFont(new Font("Arial", 14));  
tfNumero.setTooltip(ttNumero);  
  
Tooltip ttComplemento = new Tooltip("Complemento do  
cliente...");  
ttComplemento.setFont(new Font("Arial", 14));  
tfComplemento.setTooltip(ttComplemento);  
  
Tooltip ttBairro = new Tooltip("Digite o bairro do cliente.  
Campo obrigatório!");  
ttBairro.setFont(new Font("Arial", 14));  
tfBairro.setTooltip(ttBairro);  
  
Tooltip ttCidade = new Tooltip("Digite a cidade do cliente.  
Campo obrigatório!");  
ttCidade.setFont(new Font("Arial", 14));  
tfCidade.setTooltip(ttCidade);  
  
Tooltip ttCep = new Tooltip("Digite o cep. Campo  
obrigatório!");  
ttCep.setFont(new Font("Arial", 14));  
tfCep.setTooltip(ttCep);  
  
Tooltip ttUf = new Tooltip("Selecione o Estado. Campo  
obrigatório!");  
ttUf.setFont(new Font("Arial", 14));  
cbUf.setTooltip(ttUf);  
  
Tooltip ttTelefone = new Tooltip("Telefone do cliente...");  
ttTelefone.setFont(new Font("Arial", 14));  
tfTelefone.setTooltip(ttTelefone);
```

```

        Tooltip ttAtivo = new Tooltip("Necessário marcar para ativar
o cliente!");
        ttAtivo.setFont(new Font("Arial", 14));
        ckClienteAtivo.setTooltip(ttAtivo);
    }

    public void mascaraControles(){
        //-----Máscara telefone-----//
        tfTelefone.lengthProperty().addListener((ObservableValue<?
extends Number> observableValue, Number number, Number number2) -> {
            String mascara = "(##)####-####";
            String alphaAndDigits =
tfTelefone.getText().replaceAll("[\\(\\)\\-\\.]", "");
            StringBuilder resultado = new StringBuilder();
            int i = 0;
            int quant = 0;

            if (number2.intValue() > number.intValue()) {
                if (tfTelefone.getText().length() <=
mascara.length()) {
                    while (i<mascara.length()) {
                        if (quant < alphaAndDigits.length()) {
                            if ("#".equals(mascara.substring(i,i+1)))
{
                                resultado.append(alphaAndDigits.substring(quant,quant+1));
                                quant++;
                            } else {
                                resultado.append(mascara.substring(i,i+1));
                            }
                        }
                        i++;
                    }
                    tfTelefone.setText(resultado.toString());
                }
                if (tfTelefone.getText().length() > mascara.length())
{
                    tfTelefone.setText(tfTelefone.getText(0,mascara.length()));
                }
            }
        });
        //-----Máscara cep-----//
        tfCep.lengthProperty().addListener((ObservableValue<? extends
Number> observableValue, Number number, Number number2) -> {
            String mascara = "#####-###";
            String alphaAndDigits =
tfCep.getText().replaceAll("[\\-\\.]", "");

```

```

        StringBuilder resultado = new StringBuilder();
        int i = 0;
        int quant = 0;

        if (number2.intValue() > number.intValue()) {
            if (tfCep.getText().length() <= mascara.length()) {
                while (i < mascara.length()) {
                    if (quant < alphaAndDigits.length()) {
                        if ("#".equals(mascara.substring(i, i+1)))
                    {
                        resultado.append(alphaAndDigits.substring(quant, quant+1));
                        quant++;
                    } else {
                        resultado.append(mascara.substring(i, i+1));
                    }
                    i++;
                }
                tfCep.setText(resultado.toString());
            }
            if (tfCep.getText().length() > mascara.length()) {
                tfCep.setText(tfCep.getText(0, mascara.length()));
            }
        }
    });
}
}

```