

1 Introdução a interfaces gráficas e persistência de dados

O pacote **javax.swing** e seus sub pacotes possuem classes úteis para a criação de interfaces gráficas para o usuário (GUI: *Graphical User Interface*). Até então, fizemos uso de uma delas a **JOptionPane**. Neste material estudaremos a criação de interfaces gráficas utilizando diferentes recursos, indo além da simples exibição de caixas de diálogo. Alguns IDEs facilitam bastante o desenvolvimento, principalmente quando a quantidade de código a ser gerada é extensa. Vamos utilizar de agora em diante o Netbeans.

2 Desenvolvimento

2.1 (Criando interfaces gráficas com o NetBeans: Um sistema acadêmico) Nesta seção passaremos a utilizar o plugin de arrastar e soltar componentes do NetBeans para implementar um sistema acadêmico. O sistema irá permitir operações básicas de acesso a uma base de dados contendo dados de alunos e cursos de uma instituição de ensino.

2.1.1 (Criando o projeto) Comece criando um novo projeto/programa com nome de acordo com o contexto.

2.1.2 (Criando a tela de login) A primeira tela do sistema permitirá que o usuário insira seus dados de acesso. Para criá-la, clique com o direito no pacote principal da aplicação e escolha **New >> JFrame Form**. Seu nome será **LoginTela**.

2.1.3 (Campo para login) O usuário irá digitar seu login em um **JTextField**. Note que há uma paleta de componentes à direita. Arraste um componente do tipo **TextField** para a tela. Faça os seguintes ajustes:

- **Largura:** 270
- **Altura:** 54
- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão.
- Clique com o direito, escolha **Change Variable Name** e digite **loginTextField**.

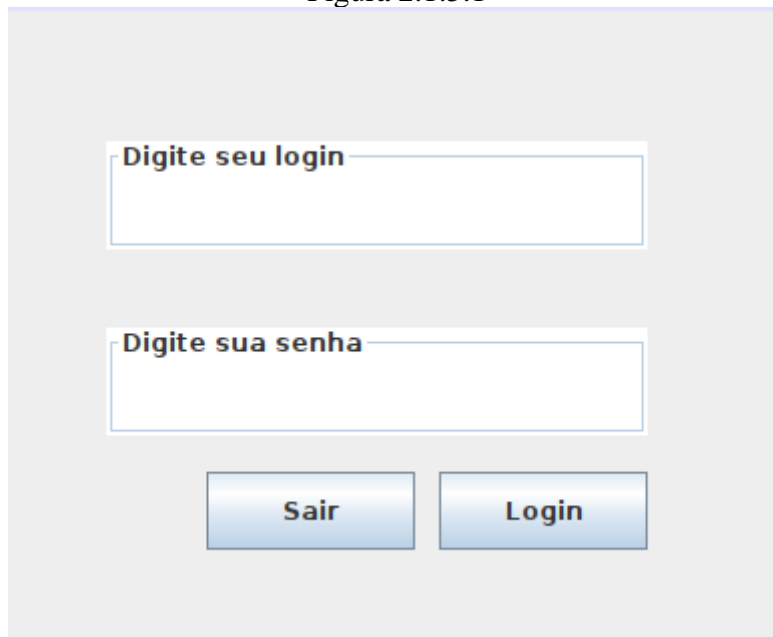
- Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite seu login** no campo **Title**.

2.1.4 (Campo para senha) Arraste e solte um componente do tipo **Password Field** logo **abaixo** do **loginTextField**. Faça os seguintes ajustes:

- Posição, largura e altura iguais aos do **loginTextField**.
- Clique com o direito, escolha **Edit Text** e apague o texto que ele exibe por padrão.
- Clique com o direito, escolha **Change Variable Name** e digite **senhaPasswordField**.
- Mantenha-o selecionado e veja suas propriedades na parte inferior direita da tela. Encontre a propriedade **border**. Escolha **Titled Border** e digite **Digite sua senha** no campo **Title**.

2.1.5 (Botões para sair e para fazer login) Arraste e solte dois componentes **Button** e faça ajustes para que o resultado seja parecido com o que exibe a Figura 2.1.5.1.

Figura 2.1.5.1



- Clique com o direito em cada botão, escolha **Change Variable Name** e altere seus nomes para **sairButton** e **loginButton**.

2.1.6 (Tratando o evento “clique” dos botões) Há uma tarefa específica a ser executada quando cada um dos botões for clicado.

- No caso do botão **sairButton**, apenas desejamos encerrar a aplicação. Para isso, basta clicar duas vezes sobre ele e completar o corpo do método que irá aparecer, como mostra a Listagem 2.1.6.1. Note que o registro do observador já foi feito automaticamente.

Listagem 2.1.6.1

```
private void sairButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    this.dispose();  
}
```

- Para o botão **loginButton**, iremos especificar uma lógica simples, que ainda não acessa a base:

- Primeiro, pegamos o login que o usuário digitou.
- Depois, pegamos a senha. Note que a senha é entregue como um vetor de char. É preciso converter para String.
- Depois, verificamos se ambos são iguais a **admin** (isso vai ser alterado quando passarmos a usar uma base de dados). Em caso positivo, o sistema mostra uma mensagem de boas-vindas. Caso contrário, mostra uma mensagem de usuário inválido. Veja a Listagem 2.1.6.2.

Listagem 2.1.6.2

```
private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    //pega o login do usuário  
    String login = loginTextField.getText();  
    //pega a senha do usuário como char[] e converte para String  
    String senha = new String (senhaPasswordField.getPassword());  
    //verifica se o usuário é válido  
    if (login.equals("admin") && senha.equals("admin"))  
        JOptionPane.showMessageDialog (null, "Bem vindo!");  
    else  
        JOptionPane.showMessageDialog(null, "Usuário inválido");  
}
```

2.1.7 (Ajustes finos no código gerado pelo NetBeans) Para demonstrar a possibilidade de personalização no código gerado pelo NetBeans, vamos **centralizar** a tela e adicionar um **título** à moldura. Não podemos editar o código gerado por ele. Podemos, porém, adicionar código ao construtor.

- Encontre o construtor e adicione as linhas destacadas na Listagem 2.1.7.1.

Listagem 2.1.7.1

```
public LoginTela() {  
    super ("Sistema Acadêmico");  
    initComponents();  
    this.setLocationRelativeTo(null);  
}
```

2.2 (Implementando a funcionalidade de login) Os dados de usuários do sistema serão armazenados em uma base relacional gerenciada pelo MySQL. Como sabemos, a fim de obter esses dados, a aplicação Java precisa estabelecer uma conexão com o MySQL Server, o que pode ser feito utilizando a API JDBC.

- Começamos criando um database para o sistema. No Workbench, uma vez conectado com o MySQL Server, use

```
CREATE DATABASE nome_do_seu_db;  
USE nome_do_seu_db;
```

para criar o novo database e informar ao MySQL Server que as próximas instruções deverão ter impacto sobre ele.

- A seguir, crie uma tabela para armazenar os dados de usuários com

```
CREATE TABLE tb_usuario (id INT PRIMARY KEY AUTO_INCREMENT,  
nome VARCHAR(200), senha VARCHAR(200));
```

- Faça a inserção de um usuário com

```
INSERT INTO tb_usuario (nome, senha) VALUES ('admin', 'admin');
```

- Segundo o princípio conhecido como **alta coesão**, cada classe que criamos deve ter um único propósito, uma única razão de ser. Sendo assim, criaremos uma classe cuja única responsabilidade será a de gerenciar conexões com o banco. Veja a Listagem 2.2.1.

Listagem 2.2.1

```
import java.sql.Connection;  
import java.sql.DriverManager;  
public class ConexaoBD {  
    private static String host = "localhost";  
    private static String porta = "3306";  
    private static String db = "seubd";  
    private static String usuario = "seusuario";  
    private static String senha = "suasenha";  
  
    public static Connection obterConexao () throws Exception{  
        String url = String.format(  
            "jdbc:mysql://%s:%s/%s",  
            host,  
            porta,  
            db  
        );  
        return DriverManager.getConnection(url, usuario, senha);  
    }  
}
```

- Lembre-se de abrir o arquivo **pom.xml** e especificar que o driver do MySQL deve ser baixado pelo Maven. O ajuste a ser feito é exibido na Listagem 2.2.2.

Listagem 2.2.2

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.bossini</groupId>

  <artifactId>peessoal_sistema_academico_com_netbeans_para_montar_pdf</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>14</maven.compiler.source>
    <maven.compiler.target>14</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.20</version>
    </dependency>
  </dependencies>
</project>
```

- Um sistema desenvolvido com linguagem que tem suporte à orientação a objetos é uma representação simplificada do mundo real (lembra do **mini-mundo**?). Assim, vamos criar uma classe para representar o que é um usuário do sistema. No momento, usuários possuem apenas duas coisas de interesse: login e senha. Veja a definição da classe que descreve o que é um usuário na Listagem 2.2.3.

Listagem 2.2.3

```
public class Usuario {  
  
    private String nome;  
    private String senha;  
  
    public Usuario(String nome, String senha) {  
        this.nome = nome;  
        this.senha = senha;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getSenha() {  
        return senha;  
    }  
  
    public void setSenha(String senha) {  
        this.senha = senha;  
    }  
}
```

- A seguir, vamos criar uma classe que será responsável por todas as operações de persistência de dados da aplicação. Embora não seja a melhor prática possível, fazê-lo nesse momento tende a dar origem a código de mais fácil compreensão. No futuro (em semestres mais avançados) aprenderemos a escrever códigos melhor organizados, que usam padrões de projeto e que tendem a ter maior nível de reusabilidade e tendem a ser mais fácil de se manter. Neste momento, o que nos é mais importante é a simplicidade. Assim, crie a classe da Listagem 2.2.4. No momento, o único método que ela possui se encarrega de verificar se um determinado usuário existe ou não na base de dados.

Nota: DAO é um acrônimo para **Data Access Object**, ou seja, Objeto de Acesso aos Dados. Trata-se de um dos padrões de desenvolvimento de software mais antigos. Uma classe DAO tem a finalidade de encapsular código de acesso a bases de dados. É comum que um projeto possua muitas classes DAO, cada qual apropriada para a manipulação de diferentes tipos de objetos. Por simplicidade, como mencionado, teremos uma única classe DAO.

Listagem 2.2.4

```
public class DAO {
    public boolean existe (Usuario usuario) throws Exception{
        String sql = "SELECT * FROM tb_usuario WHERE nome = ? AND senha = ?";
        try (Connection conn = ConexaoBD.obterConexao();
            PreparedStatement ps = conn.prepareStatement(sql)){
            ps.setString(1, usuario.getNome());
            ps.setString(2, usuario.getSenha());
            try (ResultSet rs = ps.executeQuery()){
                return rs.next();
            }
        }
    }
}
```

- O método acionado quando o botão de login é clicado será cliente do método existe. Assim ele passa a validar os dados do usuário em função do que existe realmente na base. Veja a Listagem 2.2.5.

Listagem 2.2.5

```
private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {
    //pega o login do usuário
    String login = loginTextField.getText();
    //pega a senha do usuário como char[] e converte para String
    String senha = new String (senhaPasswordField.getPassword());

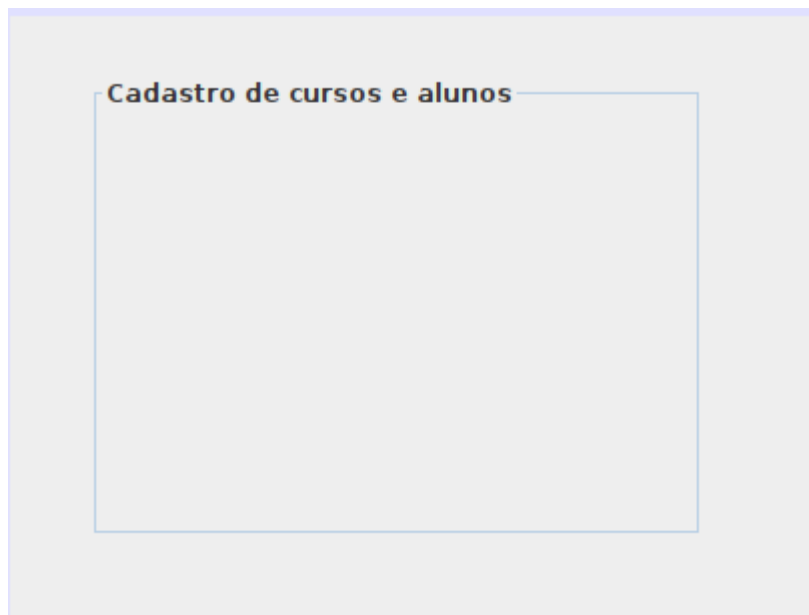
    try{
        //verifica se o usuário é válido
        Usuario usuario = new Usuario (login, senha);
        DAO dao = new DAO();
        if (dao.existe(usuario)){
            JOptionPane.showMessageDialog (null, "Bem vindo, " +
usuario.getNome() + "!");
        }
        else{
            JOptionPane.showMessageDialog(null, "Usuário inválido");
        }
    }
    catch (Exception e){
        JOptionPane.showMessageDialog (null, "Problemas técnicos. Tente
novamente mais tarde");
        e.printStackTrace();
    }
}
```

2.3 (Implementando a tela principal) Uma vez feito o login, a aplicação irá mostrar para o usuário uma espécie de Dashboard que ele pode utilizar para escolher quais funcionalidades deseja utilizar. Ela permitirá o acesso ao cadastro de cursos e ao cadastro de alunos.

- Para criar a nova tela, clique com o direito no pacote principal da aplicação e escolha **New >> JFrame Form**. Escolha o nome DashboardTela.

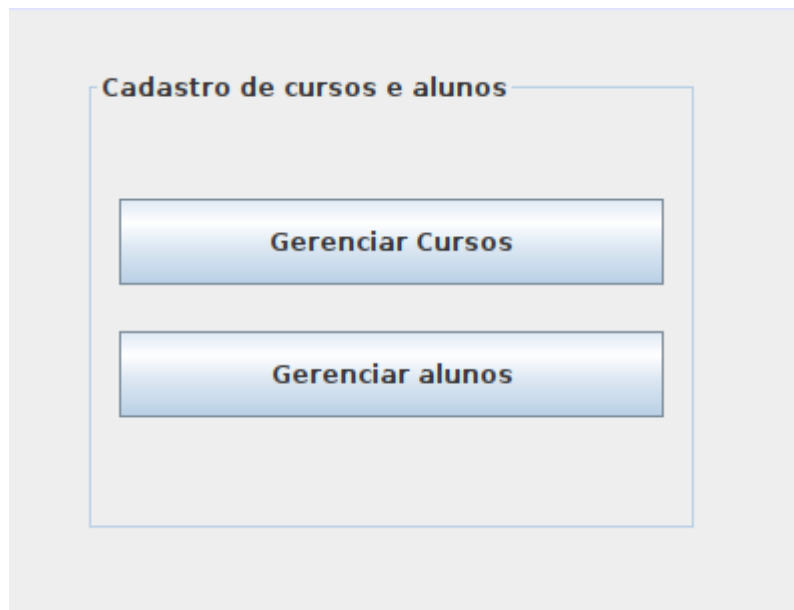
- Arraste e solte um **Panel**. Ajuste suas propriedades para que ele tenha o aspecto exibido na Figura 2.3.1.

Figura 2.3.1



- Arraste e solte dois botões. O primeiro será usado para a manipulação de dados de cursos. O segundo, para manipulação de dados de alunos. Altere seus nomes para `gerenciarCursosButton` e `gerenciarAlunosButton`, respectivamente. Para isso, basta clicar com o direito sobre cada um deles e escolher a opção **Change Variable Name**. O resultado visual é exibido na Figura 2.3.2.

Figura 2.3.2



- Como fizemos com a tela de login, vamos centralizar a tela que exibe o dashboard e configurar seu título. Encontre seu construtor padrão e faça os ajustes destacados na Listagem 2.3.1.

Listagem 2.3.1

```
public DashboardTela() {  
    super ("Cadastro de cursos e alunos");  
    initComponents();  
    setLocationRelativeTo(null);  
}
```

- No momento, quando o usuário loga com sucesso na aplicação, ela somente exibe uma mensagem de boas-vindas. Desejamos que ela abra a tela que exibe o dashboard. Para isso, uma vez feito o login, basta fazer o seguinte:

- Instanciar a classe DashboardTela
- Tornar a tela de dashboard visível com setVisible(true)
- Fechar a tela de login

Abra o método **loginButtonActionPerformed** e ajuste-o como mostra a Listagem 2.3.2.

Listagem 2.3.2

```
private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    //pega o login do usuário  
    String login = loginTextField.getText();  
    //pega a senha do usuário como char[] e converte para String  
    String senha = new String (senhaPasswordField.getPassword());  
  
    try{  
        //verifica se o usuário é válido  
        Usuario usuario = new Usuario (login, senha);  
        DAO dao = new DAO();  
        if (dao.existe(usuario)){  
            JOptionPane.showMessageDialog (null, "Bem vindo, " +  
usuario.getNome() + "!");  
            DashboardTela dt = new DashboardTela();  
            dt.setVisible(true);  
            this.dispose();  
        }  
        else{  
            JOptionPane.showMessageDialog(null, "Usuário inválido");  
        }  
    }  
    catch (Exception e){  
        JOptionPane.showMessageDialog (null, "Problemas técnicos. Tente  
novamente mais tarde");  
        e.printStackTrace();  
    }  
}
```

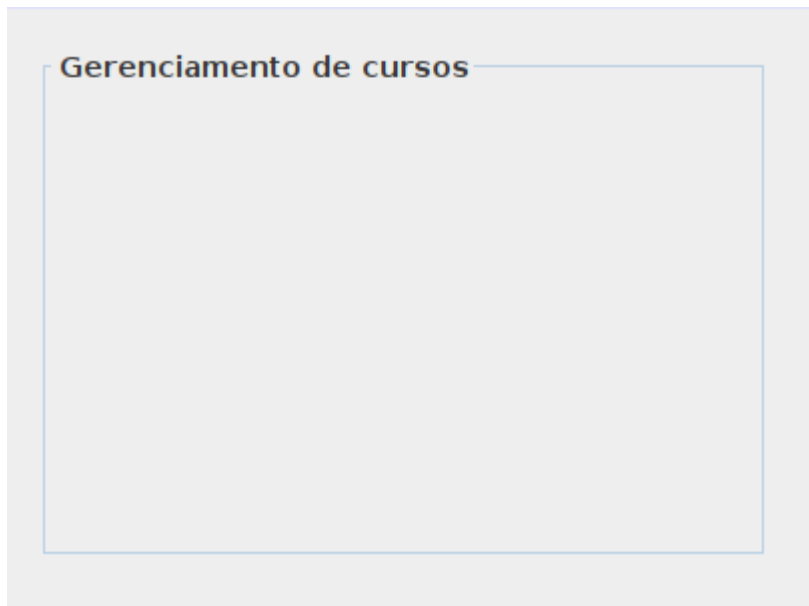
- Execute a aplicação (começando pela tela de login) e verifique se tudo está funcionando corretamente.

2.4 (Implementando a tela para gerenciamento de cursos) A tela de gerenciamento de cursos irá exibir a lista de cursos existentes no banco e também irá permitir a realização de operações junto ao banco, como o cadastro e remoção de cursos. A lista de cursos será exibida em um objeto do tipo **JComboBox**. Trata-se de um componente visual que permite a exibição de uma lista de dados em um menu e a seleção de um ou mais deles.

- Comece criando uma nova tela. Para isso, clique com o direito no pacote principal da aplicação e escolha **New >> JFrame Form**. Seu nome será **CursosTela**.

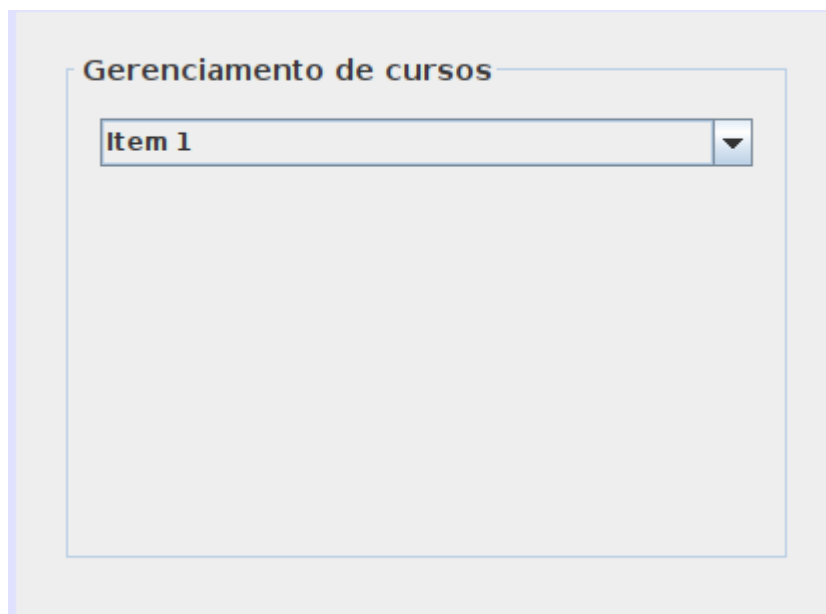
- Arraste um **Panel** e adicione a ela uma borda com título, mantendo o padrão usado até então. Veja o resultado esperado na Figura 2.4.1.

Figura 2.4.1



- Arraste um **Combo Box** para a tela, como mostra a Figura 2.4.2. Troque seu nome para **cursosComboBox**. Para isso, clique com o direito no componente que acabou de arrastar e escolha **change Variable Name**.

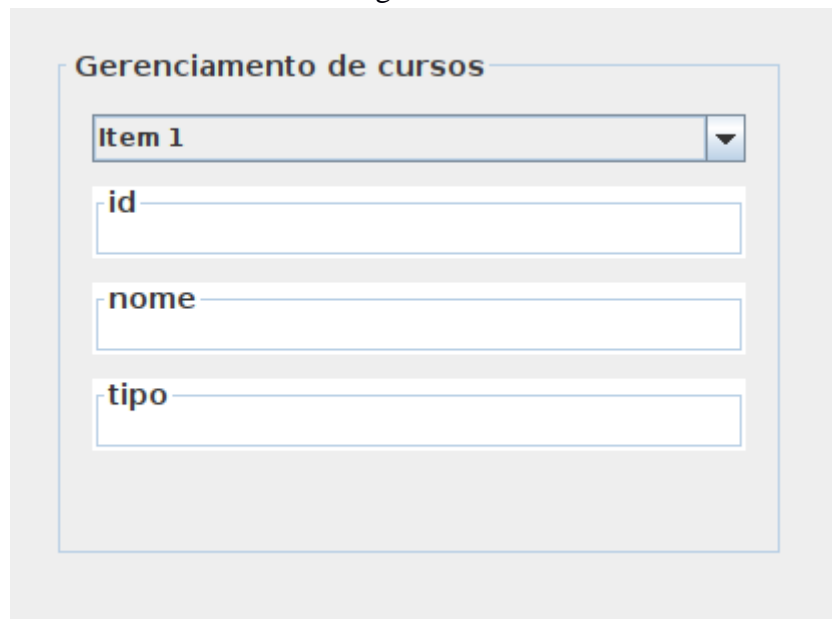
Figura 2.4.2



- A tela também terá campos que permitirão a inserção, exibição, remoção e atualização de dados de cursos. Cursos terão os atributos **id**, **nome** e **tipo**. Quando um curso for selecionado na caixa, desejamos que seus dados sejam exibidos na tela apropriadamente. Veja como a tela deve ficar no Figura 2.4.2. Cada componente textual é um **JTextField**. Troque os nomes dos componentes para **idCursoTextField**, **nomeCursoTextField** e **tipoCursoTextField**. Por padrão, eles serão **desabilitados**

para edição. Isso também é uma propriedade que você pode editar na mesma região em que edita as demais propriedades.

Figura 2.4.2



- Teremos também botões para as operações de acesso à base:
- O botão de **novo curso**, quando clicado, irá habilitar os campos textuais. Além disso, seu texto será alterado para **Confirmar**. Isso quer dizer que, para adicionar um novo curso, será necessário clicar uma vez no botão para habilitar os campos, digitar os valores e clicar novamente para confirmar.
- O botão de **atualização de curso** opera de maneira similar. Ele é desabilitado por padrão e somente é habilitado quando um curso é selecionado no menu. Quando clicado, ele habilita os campos textuais e seu texto é alterado para confirmar. Quando clicado novamente, a atualização dos dados acontece, ele volta a ficar desabilitado e os campos textuais também são desabilitados e limpos.
- O botão de remover deve ser clicado duas vezes para que o curso selecionado seja removido. Seu funcionamento é análogo aos demais.
- O botão de cancelar deve ser usado quando o usuário clica em algum dos outros e se arrepende. Quando clicado, ele desabilitará os campos textuais e os limpará. Os demais botões também serão desabilitados por ele.

Os nomes dos botões serão **adicionarCursoButton**, **atualizarCursoButton**, **removerCursoButton** e **cancelarCursoButton**.

Veja o resultado esperado na Figura 2.4.3.

Figura 2.4.3

The image shows a graphical user interface for course management. It features a title bar 'Gerenciamento de cursos'. Below the title bar, there is a dropdown menu currently showing 'Item 1'. Underneath the dropdown are three text input fields labeled 'id', 'nome', and 'tipo'. At the bottom of the form, there are four buttons arranged in a 2x2 grid: 'Novo' (highlighted in blue), 'Atualizar', 'Remover', and 'Cancelar'.

- Evidentemente, os cursos serão armazenados na base de dados. Por isso, acesse a base com o Workbench e crie uma tabela apropriada para o armazenamento de cursos com

```
CREATE TABLE tb_curso (id INT PRIMARY KEY AUTO_INCREMENT,  
nome VARCHAR (200) NOT NULL, tipo VARCHAR (200) NOT NULL);
```

- A seguir, faça a inserção de um curso para que tenhamos um primeiro dado de teste. Para isso, use

```
INSERT INTO tb_curso (nome, tipo) VALUES ('Ciência da Computação',  
'Bacharelado');
```

- Crie a classe **Curso** com os campos, métodos e construtores apropriados, como na Listagem 2.4.1.

Listagem 2.4.1

```
public class Curso {
    private int id;
    private String nome;
    private String tipo;

    public Curso(int id, String nome, String tipo) {
        this.id = id;
        this.nome = nome;
        this.tipo = tipo;
    }

    public int getId() {
        return id;
    }

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

    public String getNome() {
        return nome;
    }

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

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }
}
```

- Agora vamos implementar o método que acessa a base e traz para a memória principal a lista de cursos cadastrados. Ele faz parte da classe DAO. Veja a sua implementação na Listagem 2.4.2. O método precisa devolver uma coleção de itens. Utilizaremos um vetor temporariamente. No futuro, aprenderemos formas muito mais sofisticadas para a representação de coleções de objetos.

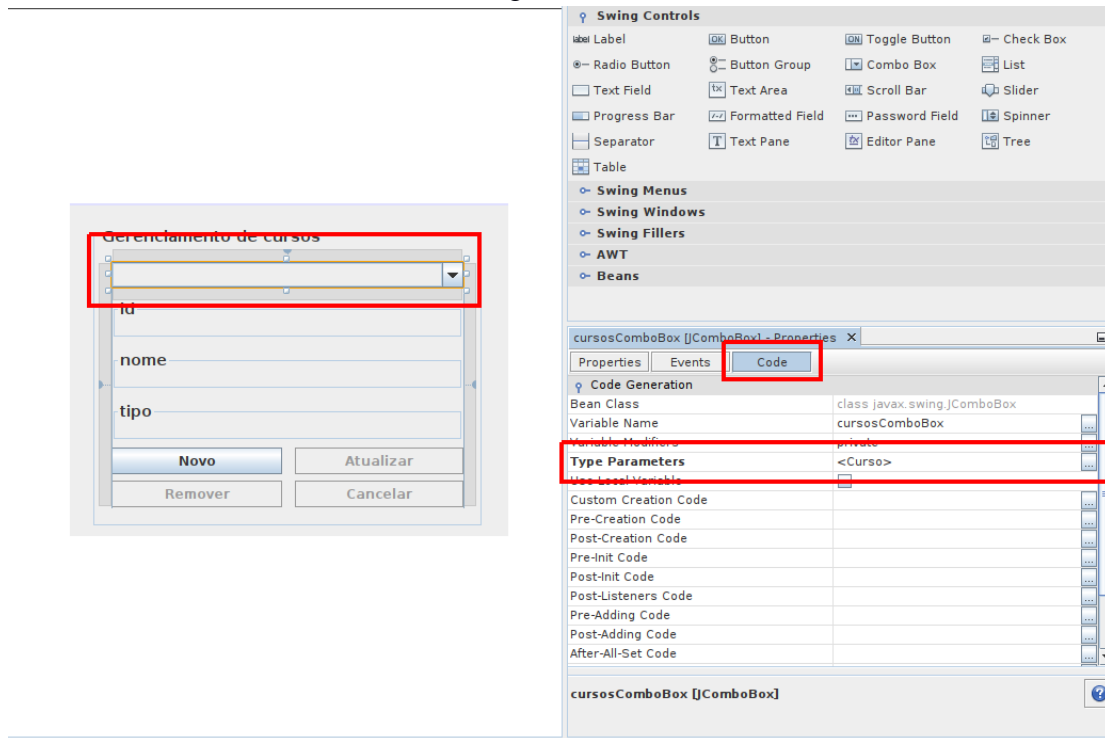
Listagem 2.4.2

```
public Curso [] obterCursos () throws Exception{
    String sql = "SELECT * FROM tb_curso";
    try (Connection conn = ConexaoBD.obterConexao();
        PreparedStatement ps =
            conn.prepareStatement(sql,
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        ResultSet rs = ps.executeQuery()){

        int totalDeCursos = rs.last () ? rs.getRow() : 0;
        Curso [] cursos = new Curso[totalDeCursos];
        rs.beforeFirst();
        int contador = 0;
        while (rs.next()){
            int id = rs.getInt("id");
            String nome = rs.getString("nome");
            String tipo = rs.getString ("tipo");
            cursos[contador++] = new Curso (id, nome, tipo);
        }
        return cursos;
    }
}
```

- Quando o ComboBox foi arrastado para a tela, seu código foi gerado automaticamente pelo NetBeans. Um ComboBox é um componente capaz de lidar com uma coleção de itens cujo tipo precisa ser definido. Por padrão, O NetBeans gera um ComboBox que é capaz de lidar com objetos do tipo String. Porém, a coleção que temos em mãos armazena objetos do tipo curso. Assim, é preciso alterar o tipo de dado armazenado pelo ComboBox de String para Curso. Isso pode ser feito selecionando o ComboBox, por meio de sua propriedade **Type Parameters**. Veja a Figura 2.4.4.

Figura 2.4.4



- Na classe CursosTela, defina o método da Listagem 2.4.3. Ele busca os dados de cursos na base (usando o método da classe DAO) e coloca em um novo modelo de dados que alimenta o JComboBox.

Listagem 2.4.3

```
private void buscarCursos (){
    try{
        DAO dao = new DAO();
        Curso [] cursos = dao.obterCursos();
        cursosComboBox.setModel(new DefaultComboBoxModel<>(cursos));
    }
    catch (Exception e){
        JOptionPane.showMessageDialog(null, "Cursos indisponíveis, tente novamente mais tarde.");
        e.printStackTrace();
    }
}
```

- O construtor da classe CursosTela será cliente do método buscarCursos. Faça os ajustes da Listagem 2.4.4.

Listagem 2.4.4

```
public CursosTela() {  
    super ("Cursos");  
    initComponents();  
    buscarCursos();  
    setLocationRelativeTo(null);  
}
```

- Na classe DashboardTela, é precisa viabilizar a navegação até a classe CursosTela por meio do clique no botão de gerenciamento de cursos. Isso pode ser feito como mostra a Listagem 2.4.5. Para acessar esse método (que é criado automaticamente pelo NetBeans), basta clicar duas vezes sobre o botão.

Listagem 2.4.5

```
private void gerenciarCursosButtonActionPerformed(java.awt.event.ActionEvent  
evt) {  
    CursosTela ct = new CursosTela();  
    ct.setVisible(true);  
    this.dispose();  
}
```

- Execute a aplicação e veja que o ComboBox exibe um valor sem muito significado para o usuário final. Ocorre que ele é um componente capaz de exibir texto e entregamos para ele uma coleção de Cursos. O que ele faz é obter a representação textual de cada curso da coleção e exibi-la. Ele o faz por meio do uso do método **toString**, que é definido pela classe **Object** (da qual todas as demais herdam, inclusive Curso). A implementação padrão de toString devolve nomeCompletamenteQualificadoDaClasse@HashCode. Para personalizar isso, basta sobrescrever o método toString na classe Curso. Ele poderia, por exemplo, devolver somente o nome do curso, com na Listagem 2.4.6.

Listagem 2.4.6

```
@Override  
public String toString() {  
    return this.nome;  
}
```

- Execute novamente e veja o resultado.

Referências

DEITEL, P. e DEITEL, H. **Java Como Programar**. 8ª Edição. São Paulo, SP: Pearson, 2010.