sãojudas universidade	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT – 2022/1 – Programação de Soluções Computacionais

Professor Bossini

Aula: 10 – Programação Orientada a Objetos – Interface Gráfica (GUI) e JDBC

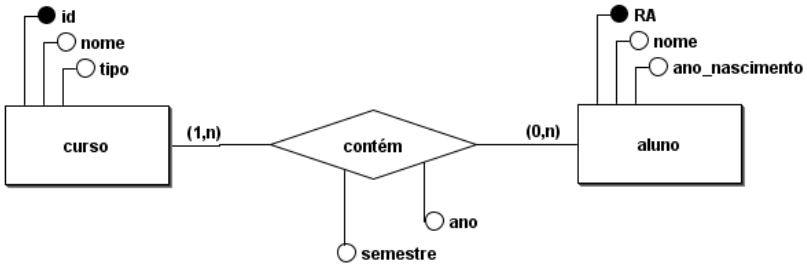
1 Introdução a Interfaces gráficas

Neste material ainda daremos continuidade ao desenvolvimento da aplicação da última aula. Vamos falar sobre o relacionamento entre alunos e cursos, desde o modelo conceitual dos dados, passando pelo mapeamento para o modelo relacional chegando à sua implementação no banco de dados. A aplicação receberá uma nova tela, que será chamada a partir da tela de cursos, que deverá exibir todos os alunos do curso, utilizando a JTable do pacote javax.swing.

2 O modelo de dados

2.1 (Modelagem Conceitual) Sabemos que um dos modelos conceituais de bancos de dados é o Modelo Entidade Relacionamento, que contém as entidades sobre as quais devemos armazenar informações e os relacionamentos entre elas. Essas informações são representadas como seus atributos, que podem ser tanto de entidades quanto de relacionamentos. A figura 2.1.1 mostra o modelo que vamos considerar para esta aula.

Figura 2.1.1



2.2 (Modelagem Lógico) A fim de implementarmos esse modelo, devemos fazer o seu mapeamento do modelo conceitual para o modelo lógico relacional, pois o SGBD que utilizamos implementa esse modelo. A listagem 2.2.1 mostra como fica esse mapeamento.

Listagem 2.2.1

```
aluno (ra, nome, ano_nascimento)
curso (id, nome, tipo)
aluno_curso (ra, id, semestre, ano)
    ra referencia aluno
    id referencia curso
```

Conforme ilustra a Listagem 2.2.1 os atributos identificadores das entidades do modelo ER foram mapeados para as chaves primárias das relações (tabelas) que representam essas entidades e o relacionamento (n para n) foi mapeado para uma tabela, cuja chave primária é composta pelas chaves estrangeiras que referenciam cada tabela que representa suas respectivas entidades.

2.3 (Modelo Físico) Finalmente, vamos implementar o modelo físico dos novos componentes, no nosso banco **db_sistema_academico**. A listagem 2.3.1 contém a criação das tabelas *aluno* e *aluno_curso*.

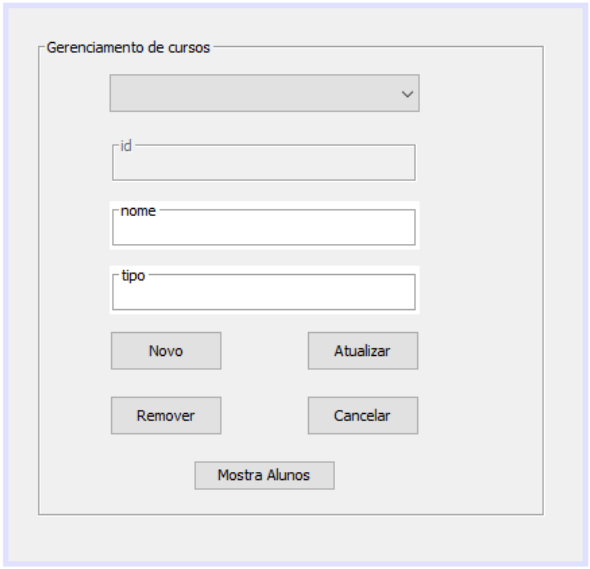
Listagem 2.3.1

```
CREATE TABLE `db_sistema_academico`.`tb_aluno` (
    `RA` INT(11) NOT NULL,
    `nome` VARCHAR(80) NOT NULL,
    `ano_nascimento` INT NULL,
    PRIMARY KEY (`RA`)
);
CREATE TABLE `db_sistema_academico`.`tb_aluno_curso` (
    `id_curso` INT(11) NOT NULL,
    `RA_aluno` INT(11) NOT NULL,
    `ano` INT NULL,
    `semestre` INT NULL,
    PRIMARY KEY (`id_curso`, `RA_aluno`),
    INDEX `fk_aluno_idx` (`RA_aluno` ASC),
    CONSTRAINT `fk_aluno`
      FOREIGN KEY (`RA_aluno`)
      REFERENCES `db_sistema_academico`.`tb_aluno` (`RA`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
    CONSTRAINT `fk_curso`
      FOREIGN KEY (`id_curso`)
      REFERENCES `db_sistema_academico`.`tb_curso` (`id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```

As chaves estrangeiras permitem a verificação da integridade referencial, isto é, não é possível criar um registro na tabela aluno_curso sem que o aluno exista na tabela aluno e o curso exista na tabela curso. Mais adiante, vamos inserir alguns dados no banco para preparamos os elementos para a nova tela.

2.4 (Criando a nova tela) A tela do nosso dashboard não será alterada, você pode fazer como exercício o gerenciamento dos alunos, com base no gerenciamento de cursos lá no item 2.6 você verá que serão acrescentados alguns detalhes importantes quanto à integridade dos dados. O que vamos implementar é uma funcionalidade na nossa tela de cursos que nos permite visualizar os alunos de um curso escolhido. Para isso, vamos alterar a nossa classe CursosTela, acrescentando um botão que abra a tela que deve conter os alunos do curso escolhido. A Figura 2.4.1 mostra como deve ficar essa tela. Não se esqueça de atualizar o nome da variável para o novo botão, seu texto e utilizar os 2 clicks sobre ele para que ele acrescente ao código o método private void mostraAlunosButtonActionPerformed(java.awt.event.ActionEvent evt).

Figura 2.4.1



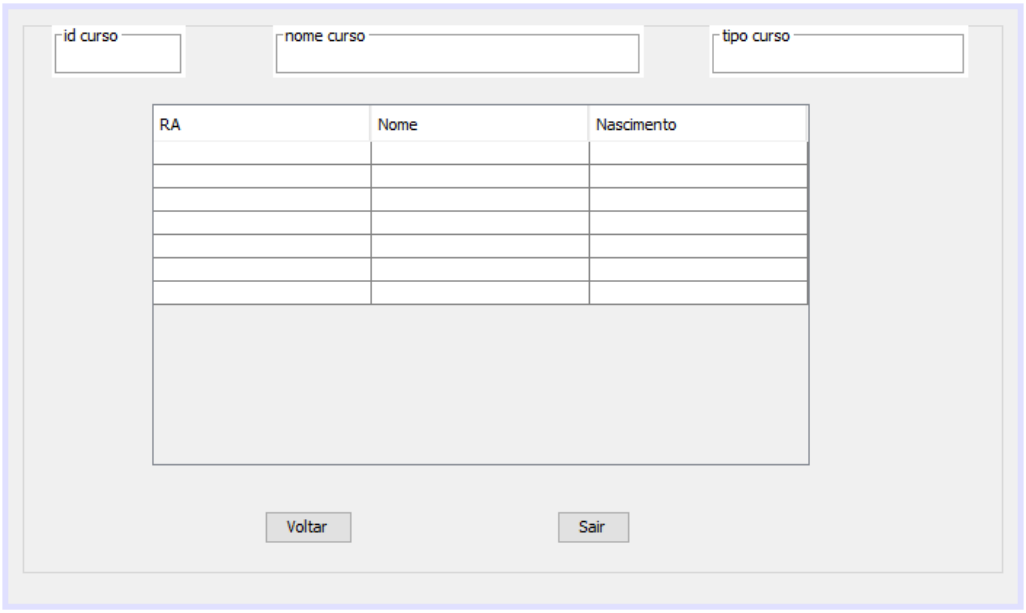
A ideia é que, ao clicarmos no botão mostraAlunosButton, seja exibida a tela mostraAlunosCursoTela ilustrada na Figura 2.4.2. Mais adiante, vamos acrescentar novas funcionalidades, mas por enquanto, vamos exibir os alunos de um determinado curso e voltar a tela anterior ou simplesmente sair da aplicação.

Antes da tela, vamos acrescentar a ação ao novo botão. A listagem 2.4.1 mostra o código que devemos acrescentar ao método gerado quando clicamos 2 vezes no botão, durante a criação da tela.

Listagem 2.4.1

```
  private void mostraAlunosButtonActionPerformed(java.awt.event.ActionEvent evt) {
    mostraAlunosCursoTela ac = new mostraAlunosCursoTela();
    ac.setVisible(true);
    this.dispose();
  }
```

Figura 2.4.2



As caixas de texto (jTextfiels) já são conhecidas, vamos começar com elas. Não se esqueçam de nome apropriados para as variáveis. Os botões também já ficaram fácil, vamos alterar os textos exibidos, os nomes das variáveis e as ações que devem ser executadas quando clicados (2 clicks em cada um para gerar o esqueleto dos métodos ButtonActionPerformed. A listagem 2.4.2 mostra o construtor com os ajustes necessários e os códigos associados a cada um dos botões.

Listagem 2.4.2

```
public mostraAlunosCursoTela() {
    super("Alunos por Curso");
    initComponents();
    setLocationRelativeTo(null);
}

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

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

Note que a tela para a qual desejamos voltar é a tela de cursos, pois é possível que o usuário queira escolher outro curso para exibir seus alunos ou de lá, ele pode voltar para o Dashboard. É importante termos esse mapa de navegação, ele está de acordo com o diagrama de sequência que descreve a sequência das ações no seu sistema.

2.5 (Trazendo as informações da tela anterior) Note que a tela de alunos por curso traz as informações do curso, que foi escolhido na classe CursosTela. Parece natural, então que a classe mostraAlunosCursoTela seja iniciada com essas informações, podemos fazer isso, passando a instância de curso como parâmetro em seu construtor, veja Listagem 2.5.1.

Listagem 2.5.1

```
public mostraAlunosCursoTela() {
    super("Alunos por Curso");
    initComponents();
    idCursoTextField.setText(Integer.toString(curso.getId()));
    nomeCursoTextfield.setText(curso.getNome());
    tipoCursoTextfield.setText(curso.getTipo());
    setLocationRelativeTo(null);
}
```

Como sugerido anteriormente, podemos deixar esses campos não editáveis, pois eles vêm da tela anterior. Podemos fazer essa alteração desmarcando a propriedade *editable* de cada um deles.

2.6 (Voltando aos dados) Nós deixamos o modelo do banco preparado para receber alunos, que serão matriculados em um curso. Lembre-se, para cadastrar um aluno num curso, é necessário que o aluno e o curso estejam cadastrados. Vamos fazer alguns testes. A listagem 2.6.1 contém algumas inserções para realizarmos alguns testes. Vejam, é possível realizar múltiplas inserções.

Listagem 2.6.1

```
INSERT INTO tb_aluno (RA, nome, ano_nascimento) VALUES
    (123456, "João Henrique", 2000),
    (345678, "Ana Maria", 2001),
    (875321, "Paulo Rogerio", 2000),
    (456789, "Maria Regina", 1999)
;
```

Em aula, vamos tentar cadastrar alguns alunos em cursos, para ver como funciona.

2.7 (A tabela) Vamos inserir uma JTable na tela mostraAlunosCursoTela, conforme ilustra a Figura 2.4.2. Percebam que existem vários parâmetros que devem ser ajustados. Observando o código gerado automaticamente pelo IDE, vemos que o componente JTable possui vários elementos (Listagem 2.7.1):

Listagem 2.7.1

```
alunosTable.setModel(new javax.swing.table.DefaultTableModel
   (
      new Object [][] {
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null},
            {null, null, null}
      },
      new String [] {
          "RA", "Nome", "Nascimento"
      }
   )
   {
       Class[] types = new Class [] {
            java.lang.String.class, java.lang.String.class, java.lang.String.class
       };

       public Class getColumnClass(int columnIndex) {
            return types [columnIndex];
       }
   }
);
jScrollPane2.setViewportView(alunosTable);
```

Assim como para o componente JComboBox temos o setModel, para a JTable, também é preciso defini-lo, pois é a partir desse modelo que a tabela vai exibir seus dados adequadamente. Os dados da tabela são obtidos a partir de uma matriz, que quando não dizemos nada, é uma matriz de Objects. Seu cabeçalho é definido a partir de um vetor de Strings. Em seguida, temos o vetor de tipos, um para cada coluna e por último, temos um método que nos devolve o tipo de cada coluna.

2.8 (Um pouco sobre matrizes) Vimos que vetores são estruturas que nos permitem armazenar uma lista de valores e acessá-lo utilizando índices, o que é muito eficiente. A listagem 2.8.1 mostra alguns exemplos, para lembrarmos que podemos declarar e iniciar um vetor ou ainda instanciá-lo e preenchê-lo conforme necessidade da aplicação.

Listagem 2.8.1

```
int[] primos = { 2,3,5,7,11,13,17,19 };
char[] dd = { 'd','s','t','q','q','s','s'};
String[] meses = {"jan","fev","mar","abr" };
int[] fibonacci = new int[20];
fibonacci[0] = 1;
fibonacci[1] = 1;
for (int i=2; i<20; i++) {
    fibonacci[i] = fibonacci[i-2] + fibonacci[i-1];
}
```

Uma matriz é basicamente um vetor onde cada elemento é por sua vez um vetor. No exemplo mostrado na Listagem 2.8.2, mat é uma matriz com 10 linhas e 9 colunas.

Listagem 2.8.2

```
int[][] mat = new int[10][9];
for(int i = 0; i < 10; i++) {
   for(int j = 0; j < 9; j++) {
      mat[i][j] = i*j;
   }
}
```

Agora vamos ajustar a tabela, de acordo com os dados do banco.

Referências

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

ORACLE Java Tutorials. **Java Tutorials Learning Paths**. acesso em: <https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html>.