

COMPOSIÇÃO E AGREGAÇÃO DE CLASSES EM JAVA E VISIO

Em orientação a objetos é comum que objetos de uma certa classe possuam referências a um ou mais objetos de outras classes. Um exemplo muito simples é um objeto de uma classe Pedido que possui referência a uma lista de Itens, de forma que cada objeto desse tipo possua referência a vários objetos de outro tipo. Definimos composição quando um dos objetos não existe independente do outro e agregação é quando os dois podem existir independentemente. Vejamos de forma mais aprofundada:

Agregação

É quando um objeto possui outros objetos, ele não depende desses objetos para existir.

Exemplo:



Uma Gaveta pode conter Meias, mas a Gaveta não é feita de Meias. Ou seja, mesmo sem Meias a Gaveta ainda existirá.

Composição

É quando um objeto é formado por outros objetos. Ou seja, suas partes o compõem, sem elas o objeto não existe.

Exemplo:

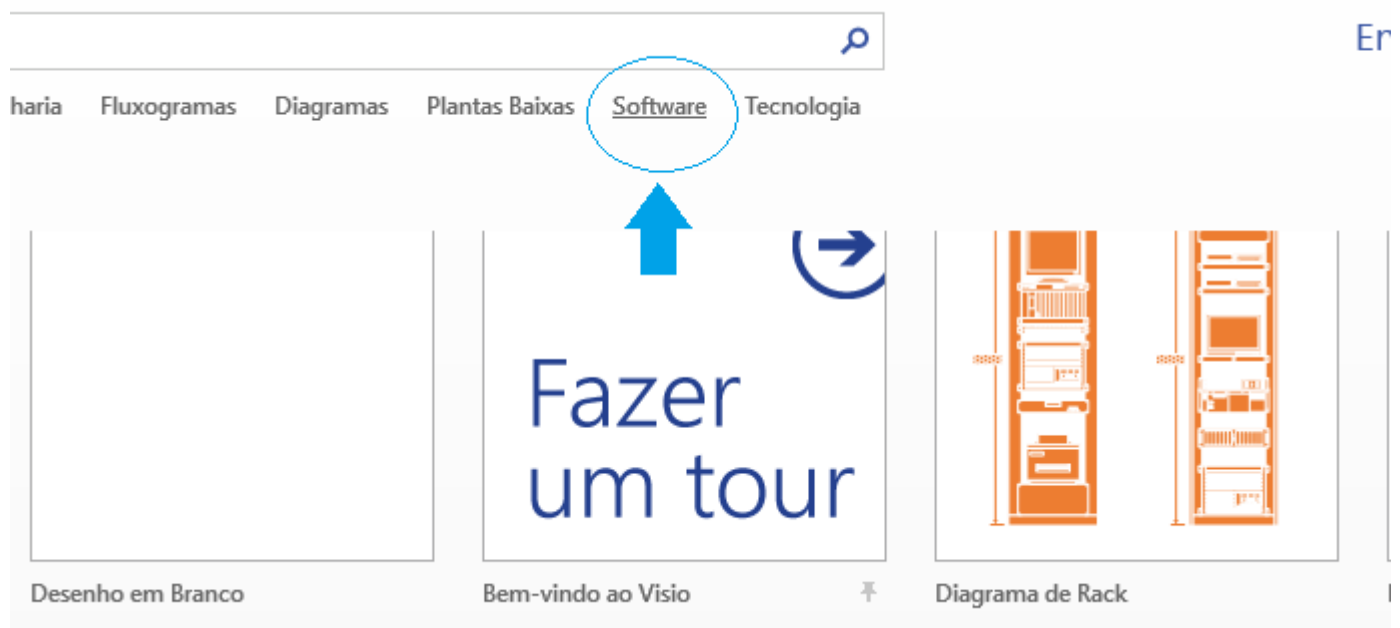


Um Computador é formado por seus componentes, como por exemplo placa mãe, gabinete, disco, memória, placa de vídeo, etc. Sem todas essas peças não existe nosso Computador de acordo com a representação do diagrama. Logo, no nosso diagrama o Computador é um conceito, pois concretamente ele é composto por um conjunto diferentes componentes.

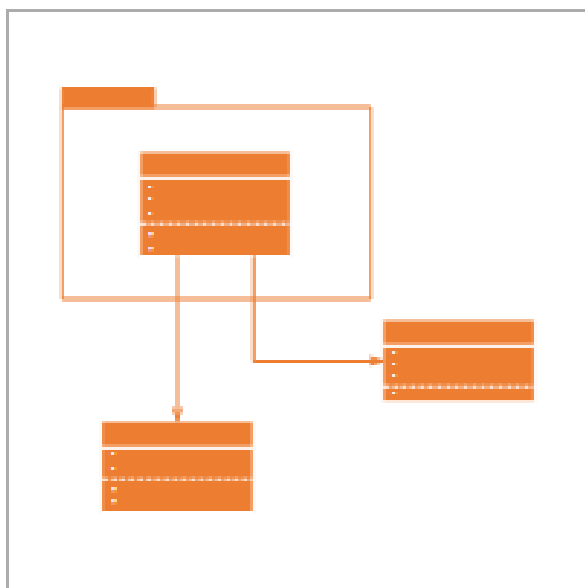
A diferença é apenas conceitual. Quem bater o olho no seu diagrama de classes entenderá com facilidade o que seu sistema está modelando, e essa é a intenção dos diagramas UML, uma representação visual do seu sistema.

MODELANDO UM DIAGRAMA UML DE COMPOSIÇÃO/AGREGAÇÃO NO MICROSOFT VISIO

Vamos exemplificar a modelagem de um projeto de classes em UML usando o VISIO antes de codificar a classe em Java. Nosso projeto criará um sistema de controle de salas de aula para uma escola. Abra o Visio e escolha na tela principal digramas de software:



Em seguida selecione estrutura estática UML e pressione o botão criar:



Estrutura Estática UML

Você verá uma barra direita para a criação dos nossos componentes, nela podemos desenhar as nossas classes e em seguida estabelecer o relacionamento entre elas, em nosso caso estaremos usando a composição e a agregação.

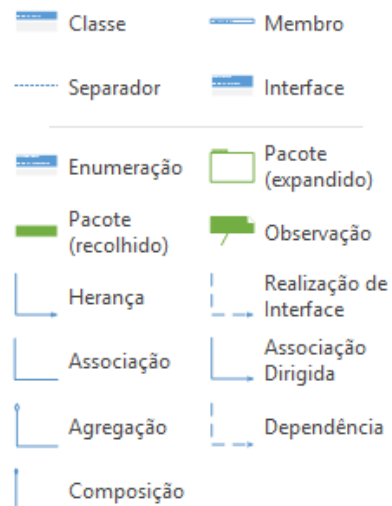
Formas

ESTÊNCIAS | PESQUISA

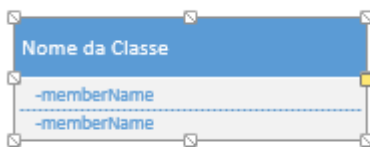
Mais Formas ▶

Formas Rápidas

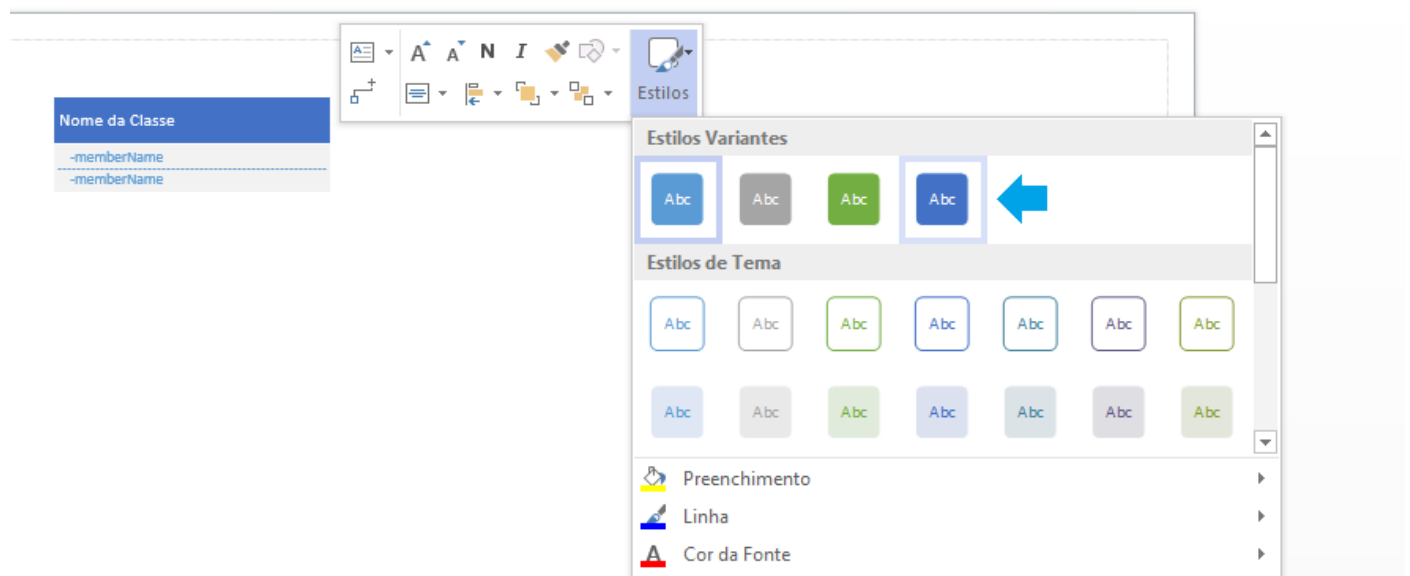
Classe de UML



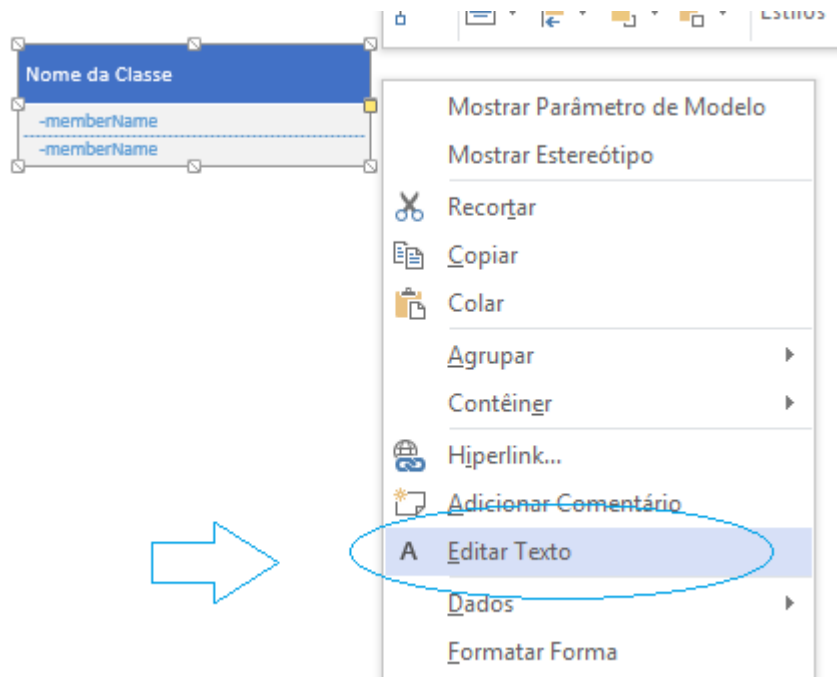
Arraste a classe para a área de desenho:



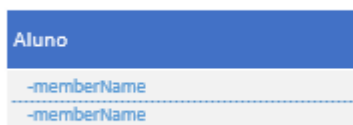
Selecione o seguinte estilo para a nossa classe:



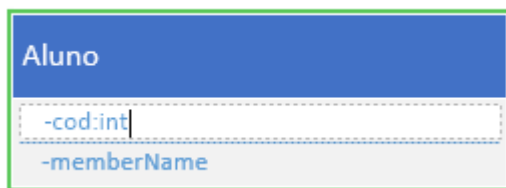
Edite o nome da classe para aluno:



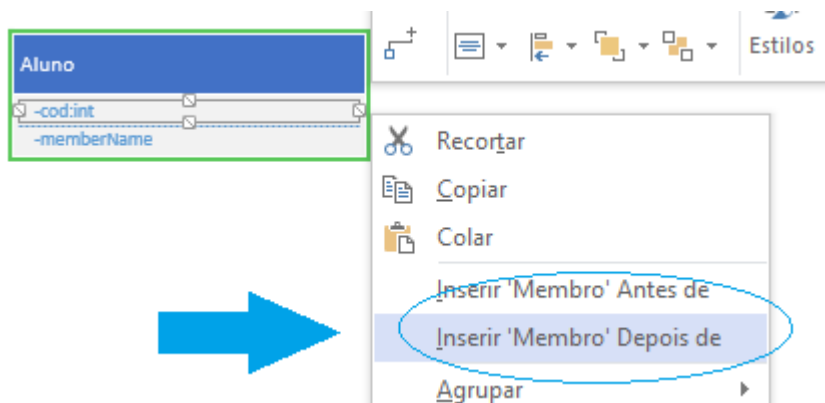
A classe ficará da seguinte forma:



Vamos agora acrescentar as propriedades da classe aluno. Selecione o primeiro **memberName** e dê dois cliques no mesmo para editá-lo:



Selecione o membro editado e em seguida, com o botão direito do mouse escolha inserir novo membro após o selecionado:



Edite esse membro conforme abaixo:

Aluno
- cod : int
- nome : String
-memberName

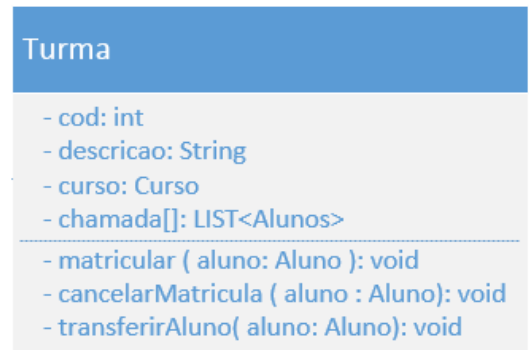
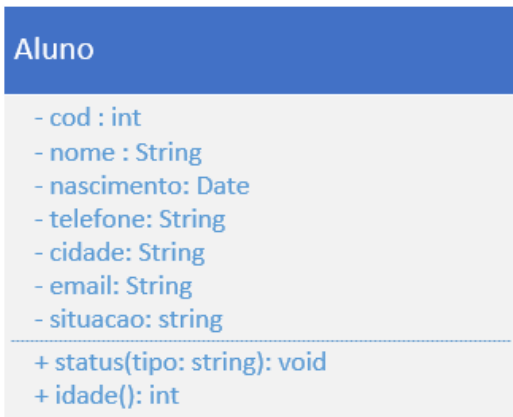
Repita o processo para que tenhamos a seguinte estrutura:

Aluno
- cod : int
- nome : String
- nascimento: Date
- telefone: String
- cidade: String
- email: String
- situacao: string
- memberName

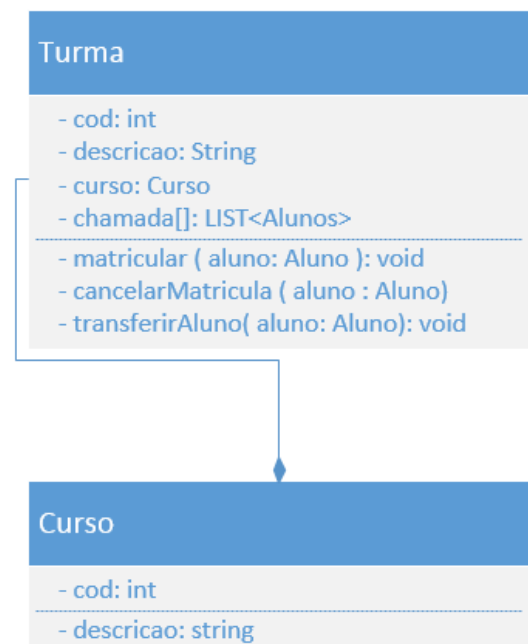
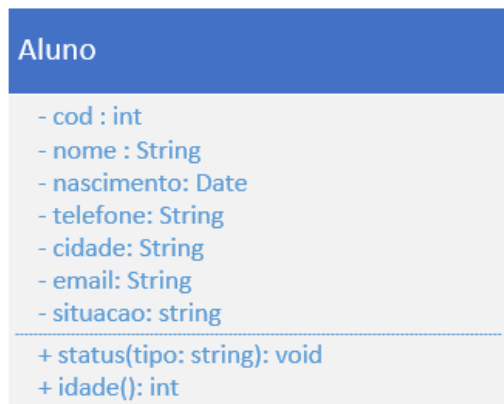
A barra tracejada delimita em nosso caso os métodos. Em nosso caso iremos definir os seguinte métodos, lembrando que o construtor e getter e setters não são necessários se forem as definições padrões:

Aluno
- cod : int
- nome : String
- nascimento: Date
- telefone: String
- cidade: String
- email: String
- situacao: string
+ status(tipo: string): void
+ idade(): int

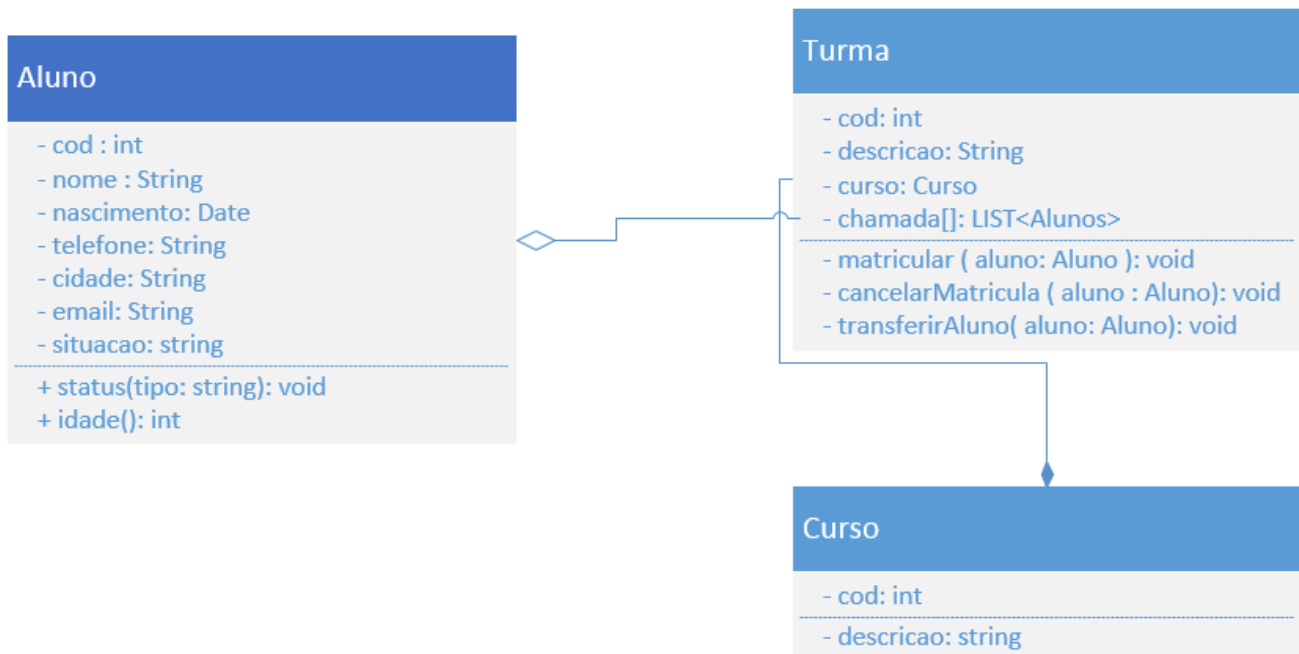
Criamos um método de status para saber a situação do nosso aluno (ativo ou inativo, transferido, evadido etc.) e um método para calcular a idade. Em seguida iremos criar uma nova classe que dever conter as definições da nossa sala de aula, no caso do curso técnico, chamado de turmas e a lista de cursos de nosso sistema:



Vamos agora ajustar as relações a serem feitas. É possível perceber que uma turma não pode existir sem um curso, portanto temos ai uma relação de composição do tipo 1 para 1. Para isso arraste o conector de composição e o posicione para que fique da seguinte forma:



A relação entre aluno e turma e do tipo de agregação, já que podemos ter uma turma sem alunos no momento da criação da mesma, ou seja, o todo pode existir sem a parte. Use o conector de agregação e posicione para que fique com o seguinte aspecto:



Agora temos a nossa estrutura criada. Salve o arquivo para que possamos modifica-lo quando necessário. Agora podemos realizar o projeto a partir dos dados do digrama UML:

Novo Aplicação Java

Etapas

- Escolher Projeto
- Nome e Localização**

Nome e Localização

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

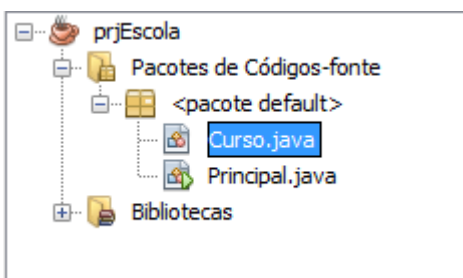
☐ Usar Pasta Dedicada para Armazenar Bibliotecas

Pasta Bibliotecas:

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Criar Classe Principal

Vamos começar com a codificação da classe curso, que é mais simples:



Estrutura da classe Curso:

```

public class Curso {
    private int cod;
    private String descricao;

    //getters e setters
    public int getCod() {
        return cod;
    }

    public void setCod(int cod) {
        this.cod = cod;
    }

    public String getDescricao() {
        return descricao;
    }

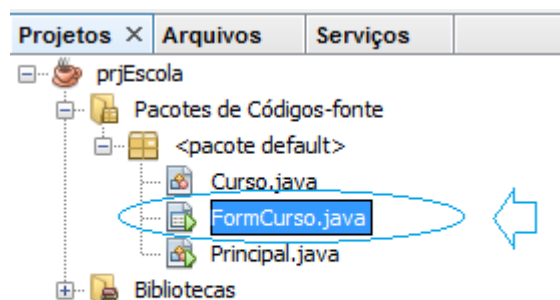
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    //construtor completo
    public Curso(int cod, String descricao) {
        this.cod = cod;
        this.descricao = descricao;
    }

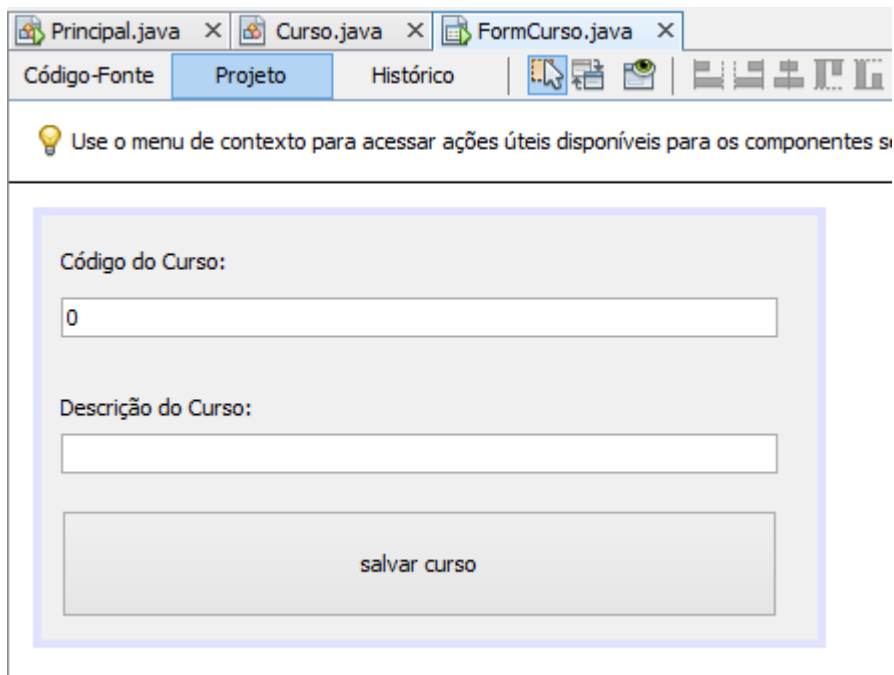
    //construtor vazio
    public Curso() {
        this(0, "");
    }
}

```

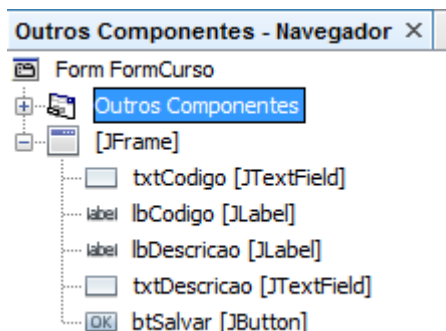
Codificar a classe Principal para realizar o cadastro de um Curso. Criar um formulário para realizar o cadastro de Cursos:



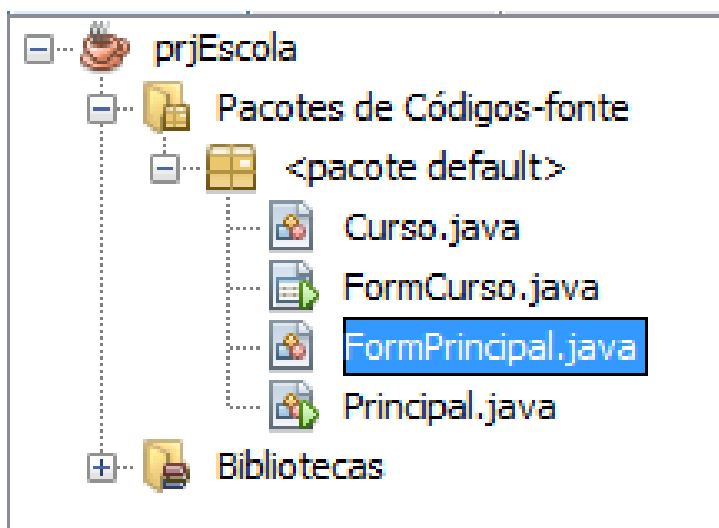
Desenhe o formulário conforme o design seguinte:



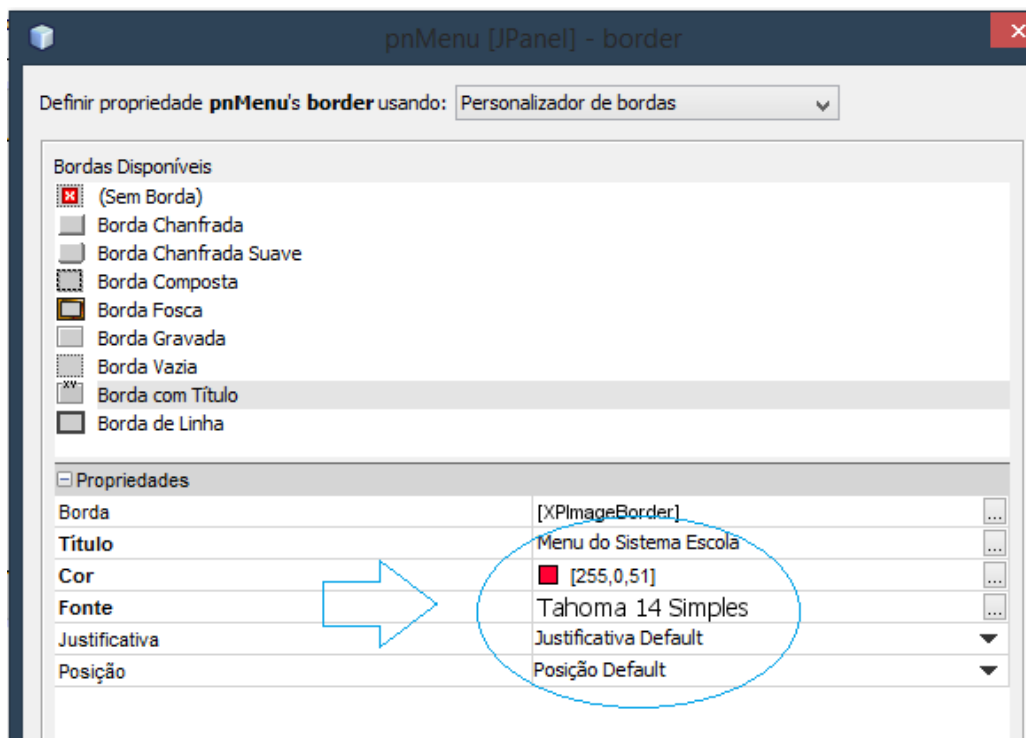
Nome das variáveis:



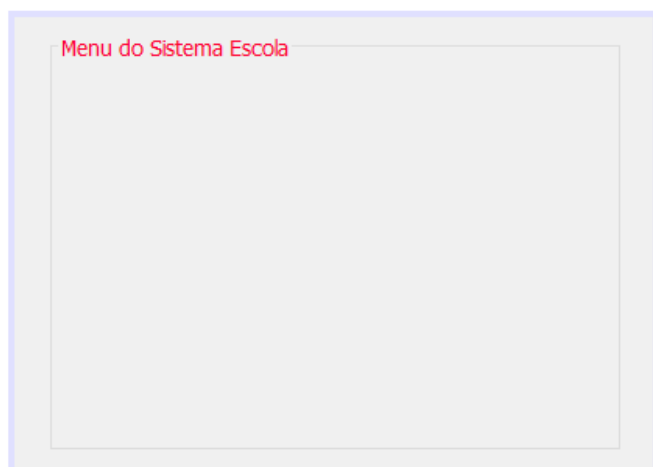
Criar um formulário com 4 botões que definirá qual cadastro deverá ser realizado:



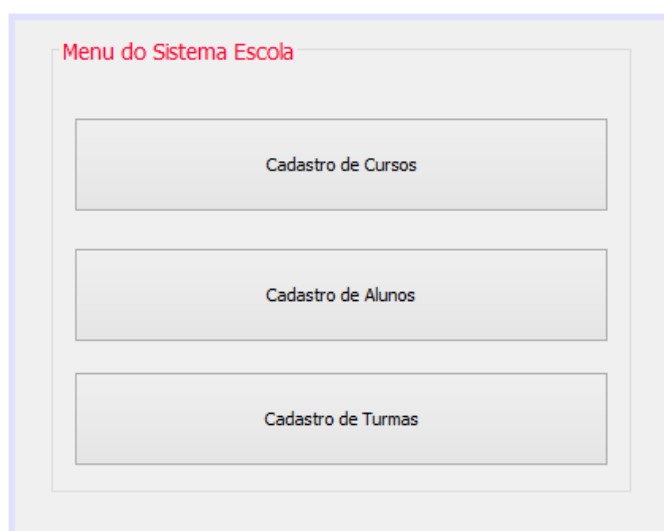
Design do formulário principal. Crie um painel no formulário e mude a propriedade **border** para os seguintes valores:



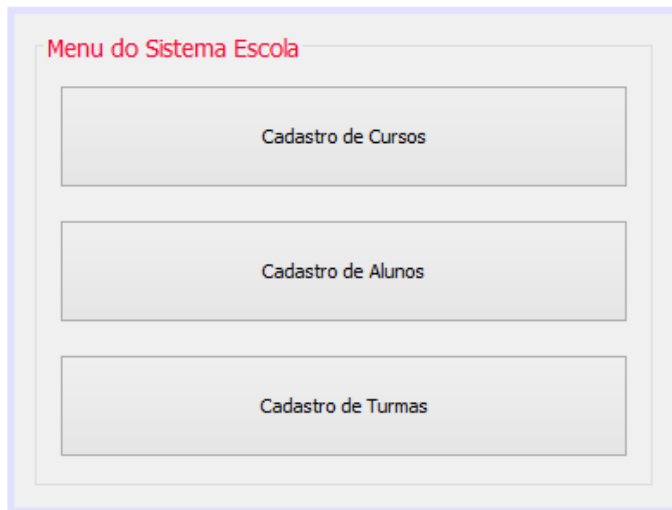
A aparência será a seguinte:



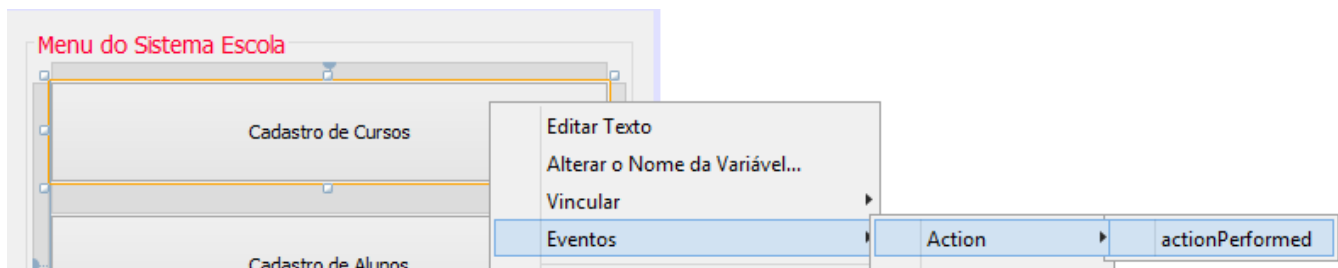
Acrescente os botões no painel:



Nome dos componentes:



O próximo passo é definir um evento que ao clicar o botão cursos, abra o respectivo formulário:



Código do evento:

```
private void btCursosActionPerformed(java.awt.event.ActionEvent evt) {  
    // cria a variável de Formulário  
    FormCurso fr = new FormCurso();  
    // mostra o formulário e seta para fechar apenas o formulário  
    fr.setDefaultCloseOperation(DISPOSE_ON_CLOSE);  
    fr.setVisible(true);  
}
```


Mudei a propriedade do JFrame para dispose, senão ao fechar o formulário ele acaba fechando o programa junto. Agora na classe Principal basta chamar o formulário principal:

```
public class Principal {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        FormPrincipal fr = new FormPrincipal();  
        fr.setVisible(true);  
    }  
}
```

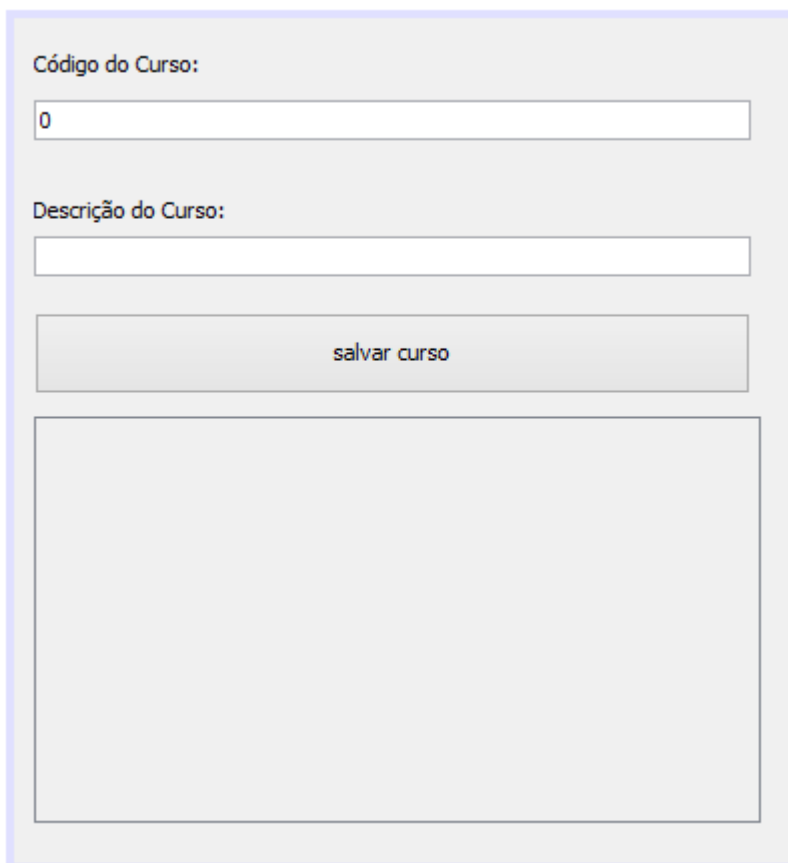
Teste para ver se formulário está funcionando corretamente.

A etapa seguinte consiste em cadastrar os cursos. Para fazer isso vamos criar uma lista de cursos para o formulário de curso e definir um modelo para o Jtable:

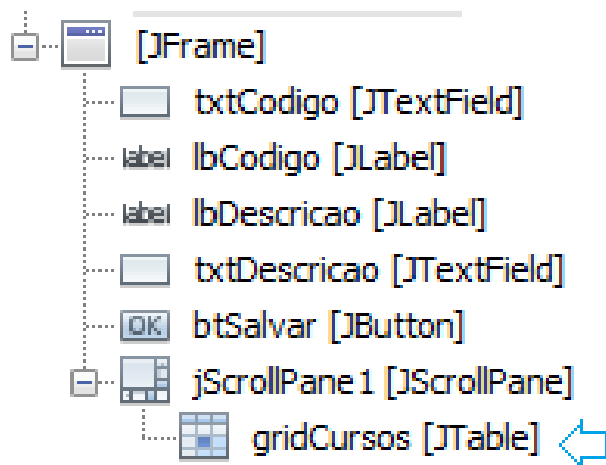
```
public class FormCurso extends javax.swing.JFrame {  
  
    /**  
     * Creates new form FormCurso  
     */  
    // lista de cursos a serem cadastrados  
    public ArrayList<Curso> cursos = new ArrayList<>();  
    //modelo para o Jtable  
    DefaultTableModel modelo = new DefaultTableModel();  
  
    public FormCurso() {  
        initComponents();  
    }  
}
```



Onde exibiremos os cursos cadastrados? Para isto será necessário desenhar um grid para exibir os cursos:



Nome dos componentes:



No evento de abertura do formulário ajuste as colunas do grid que serão exibidas:

```

private void formWindowOpened(java.awt.event.WindowEvent evt) {
    modelo.addColumn("Código");
    modelo.addColumn("Descrição");
    gridCursos.setModel(modelo);
}

```

Programar o evento do botão salvar para gravar o curso no grid:

```

private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        //seta o curso
        Curso curso = new Curso(
            Integer.parseInt(txtCodigo.getText()),
            txtDescricao.getText());
        //adiciona o curso na lista de cursos
        cursos.add(curso);

        JOptionPane.showMessageDialog(null, "Curso cadastrado");

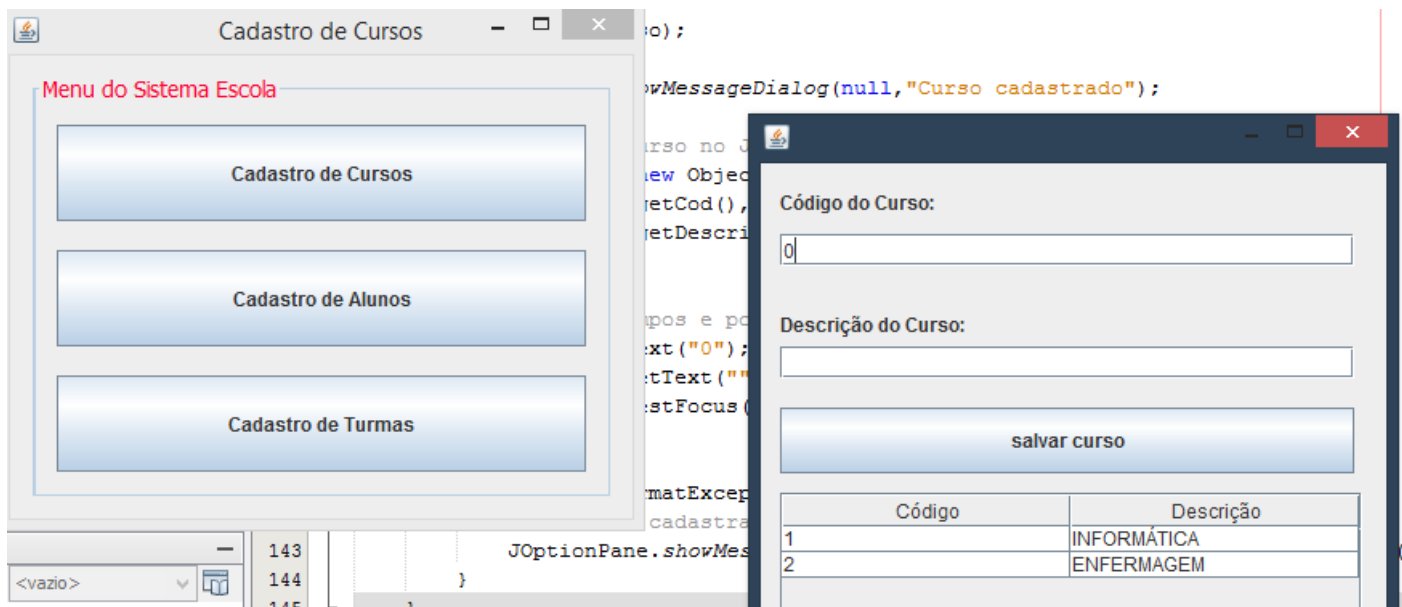
        //adiciona o curso no JTable
        modelo.addRow(new Object[] {
            curso.getCod(),
            curso.getDescricao()
        });

        // limpa os campos e posiciona o mouse no código
        txtCodigo.setText("0");
        txtDescricao.setText("");
        txtCodigo.requestFocus();
    }

    catch (NumberFormatException e){
        // erro ao cadastrar
        JOptionPane.showMessageDialog(null, "Erro ao cadastrar:" + e.getMessage());
    }
}

```

Teste o Código:



Para evitar que o sistema cadastre um curso que já existe, ou seja, um código repetido acrescente o seguinte código:

```
try{
    //seta o curso
    Curso curso = new Curso(
        Integer.parseInt(txtCodigo.getText()),
        txtDescricao.getText());

    // checar se código já cadastrado:
    if(verificaCurso(curso)) {
        JOptionPane.showMessageDialog(null, "Curso já cadastrado");
        return;
    }

    //adiciona o curso na lista de cursos
    cursos.add(curso);
}
```

Codificar a verificação do Curso:

```
private boolean verificaCurso(Curso c) {
    //checa se algum código igual existe na matriz de cursos
    // retorna verdadeiro se achar e falso se não encontrar
    return cursos.stream().anyMatch((item) -> (item.getCod() == c.getCod()));
}
```

Teste o código:

Código do Curso:

1

Descrição do Curso:

INFORMÁTICA

salvar curso

Código	Descrição
1	INFORMATICA

Mensagem

Curso já cadastrado

OK

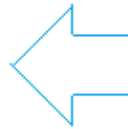
Vamos criar a rotina de edição, para isso vamos primeiro codificar o grid para devolver nos campos de texto código e descrição os dados da linha selecionada:

```
private void gridCursosMouseClicked(java.awt.event.MouseEvent evt) {  
    // procura o curso  
    Curso c = procurar(gridCursos.getValueAt(gridCursos.getSelectedRow(),  
        0).toString());  
    //achou?  
    if(c!=null){  
        // preenche as caixas de texto  
        txtCodigo.setText(String.valueOf(c.getCod()));  
        txtDescricao.setText(c.getDescricao());  
    }  
}
```

A rotina procurar deve percorrer a lista e devolver na variável “c” os dados do curso, que usaremos para preencher as caixas de código e descrição:

```
private boolean verificaCurso(Curso c) {
    //checa se algum código igual existe na matriz de cursos
    // retorna verdadeiro se achar e falso se não encontrar
    return cursos.stream().anyMatch((item) -> (item.getCod() == c.getCod()));
}
```

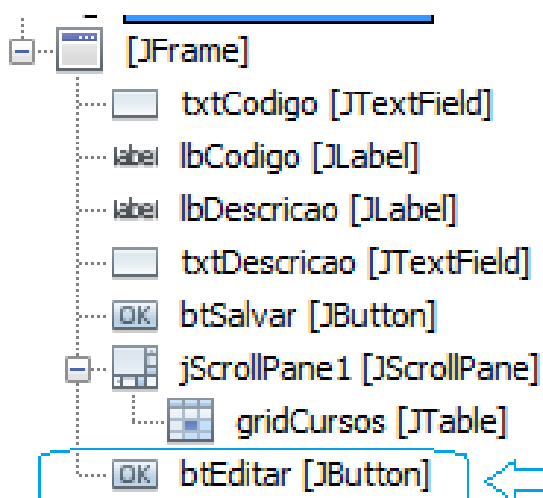
```
private Curso procurar(String cod) {
    for(Curso item: cursos){
        int num = Integer.parseInt(cod);
        if(item.getCod() == num)
            return item;
    }
    return null;
}
```



Teste o código.

O próximo passo é criar o botão editar, note que esse botão permite apenas a mudança da descrição já que o código não pode ser alterado.

Nome do componente:



Codificar o evento clique deste botão:


```

private void btEditarActionPerformed(java.awt.event.ActionEvent evt) {
    // procura o curso
    Curso c = procurar(gridCursos.getValueAt(gridCursos.getSelectedRow(),
        0).toString());
    // achou?
    if(c!=null){
        // altera apenas a descrição
        c.setDescricao(txtDescricao.getText());
        // acerta o texto da descrição no grid
        gridCursos.setValueAt(c.getDescricao(),gridCursos.getSelectedRow(),1);
        JOptionPane.showMessageDialog(null,"Curso Editado com Sucesso");
    }
}

```

Teste o código.

Crie agora um botão de remoção:

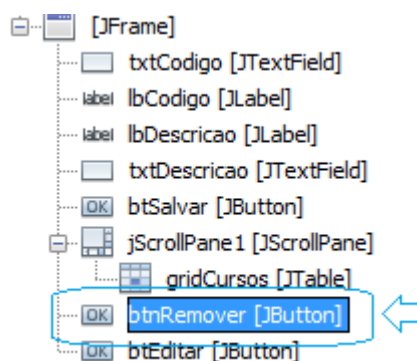
Código do Curso:

0

Descrição do Curso:

salvar curso editar curso remover curso

Nome do componente:



Codificar agora a rotina de remoção:

```

private void btnRemoverActionPerformed(java.awt.event.ActionEvent evt) {
    // procura o curso
    Curso c = procurar(gridCursos.getValueAt(gridCursos.getSelectedRow(),
        0).toString());
    //achou?
    if(c!=null){
        // remove o curso
        cursos.remove(c);
        // remove a linha
        modelo.removeRow(gridCursos.getSelectedRow());
        JOptionPane.showMessageDialog(null, "Curso removido com Sucesso");
        // limpa os campos e posiciona o mouse no código
        txtCodigo.setText("0");
        txtDescricao.setText("");
        txtCodigo.requestFocus();
    }
}

```

Teste o código.

Se você não selecionar nenhuma linha no grid fará com que os botões de editar e remover voltem uma mensagem de erro de índice fora de faixa. Para evitar este problema adicione o seguinte código tanto no botão remover quanto editar:

```

private void btEditarActionPerformed(java.awt.event.ActionEvent evt) {
    // verificar se uma linha foi selecionada
    if(gridCursos.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(null, "Nenhum Curso selecionado");
        return;
    }
    // procura o curso
    Curso c = procurar(gridCursos.getValueAt(gridCursos.getSelectedRow(),
        0).toString());
    //achou?
    if(c!=null){
        // altera apenas a descrição
        c.setDescricao(txtDescricao.getText());
        // acerta o texto da descrição no grid
        gridCursos.setValueAt(c.getDescricao(), gridCursos.getSelectedRow(), 1);
        JOptionPane.showMessageDialog(null, "Curso Editado com Sucesso");
    }
}

```

Módulo ALUNO

Criando as propriedades da classe Aluno:

```

private int cod;
private String nome;
private Calendar nascimento;
private String telefone;
private String cidade;
private String email;
private String situacao;

```

Note que substituímos o Date pela classe Calendário, pois o mesmo se tornou obsoleto nas novas versões de Java. Crie o construtor:

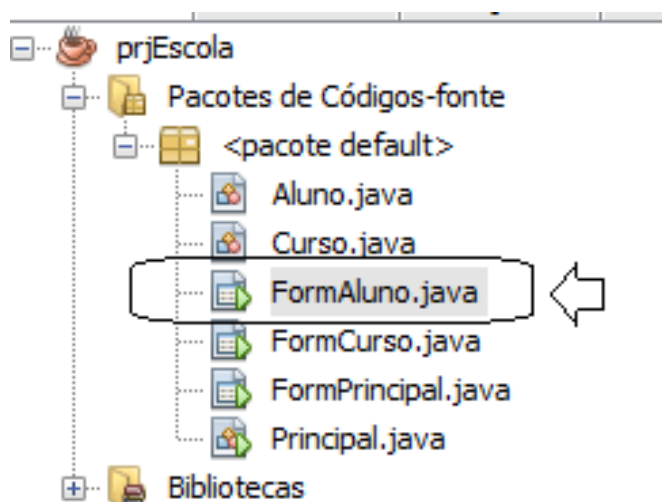
```
public Aluno(int cod, String nome, Calendar nascimento,
            String telefone, String cidade,
            String email, String situacao) {
    this.cod = cod;
    this.nome = nome;
    this.nascimento = nascimento;
    this.telefone = telefone;
    this.cidade = cidade;
    this.email = email;
    this.situacao = situacao;
}
```

Gere na sequencia os getters e setters e na sequencia codifique os métodos abaixo:

```
// fim dos getters e setters
// inicio dos métodos da classe Aluno
public void status(String tipo ) {
    if(!tipo.equals("")) this.setSituacao(tipo);
}

public int idade() {
    //le a data de hoje
    Calendar c = Calendar.getInstance();
    //le o ano atual
    int hoje = c.get(Calendar.YEAR);
    int anoNascimento = this.getNascimento().get(Calendar.YEAR);
    int idade = hoje - anoNascimento;
    return idade;
}
```

Desenhar o formulário **FormAluno**:



código: Nome do aluno:

0

Telefone: Cidade:

Email:

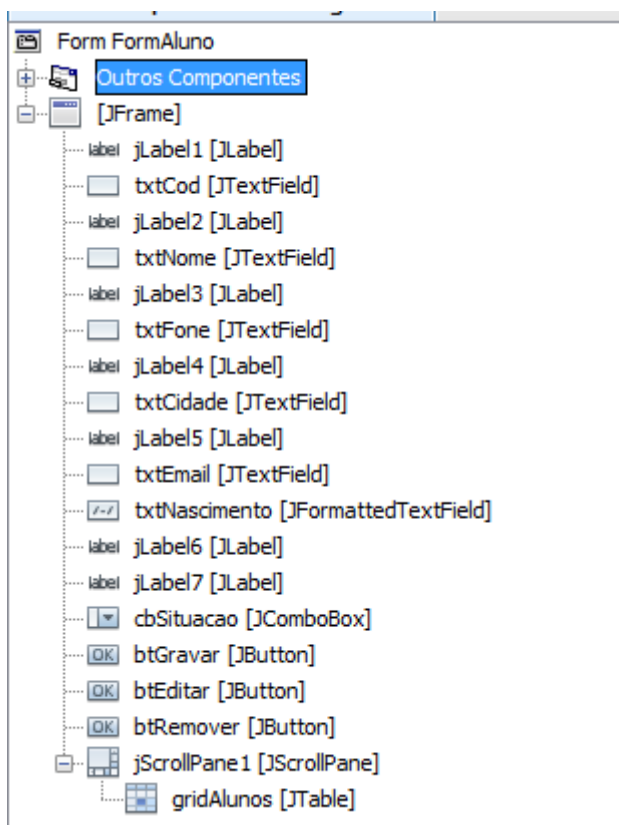
Nascimento: Situação:

 ATIVO ▾

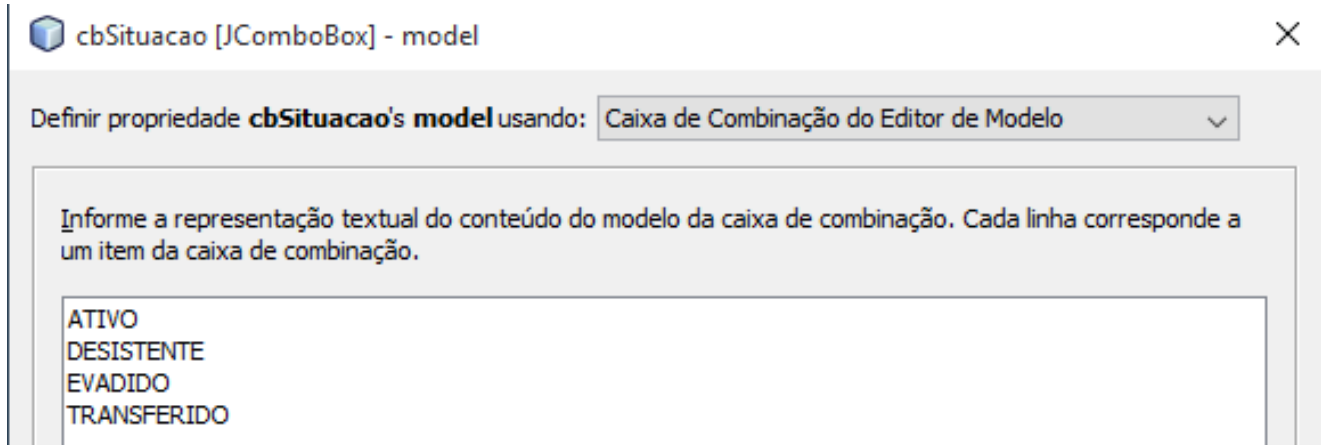
gravar editar remover

Title 1	Title 2	Title 3	Title 4

Nome dos campos:

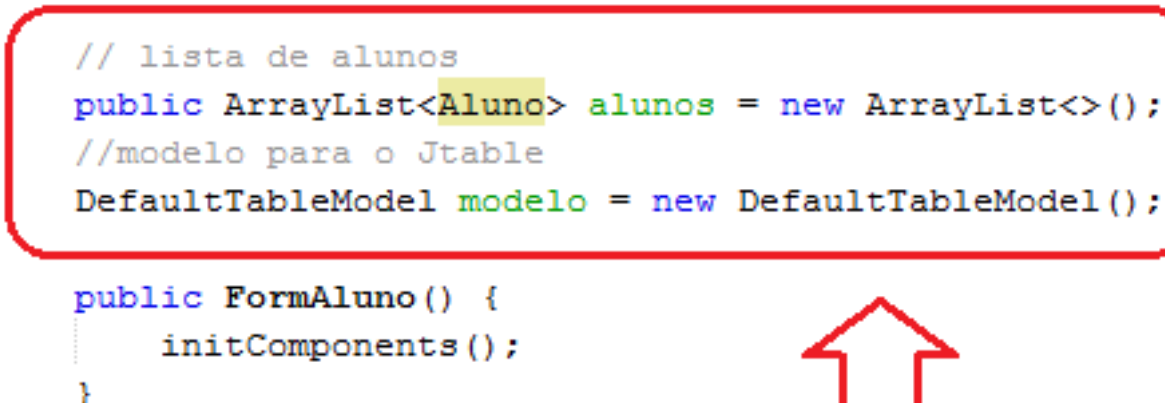


Em cbSituação, na propriedade model, preenche os seguintes valores:



Codificar no início da classe os seguintes valores:

```
public class FormAluno extends javax.swing.JFrame {  
  
    /**  
     * Creates new form FormAluno  
     */  
  
    // lista de alunos  
    public ArrayList<Aluno> alunos = new ArrayList<>();  
    //modelo para o Jtable  
    DefaultTableModel modelo = new DefaultTableModel();  
  
    public FormAluno() {  
        initComponents();  
    }  
}
```



Codificar as seguintes linhas no evento WINDOWS_OPENED do formulário:

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {  
  
    modelo.addColumn("Código");  
    modelo.addColumn("Nome");  
    modelo.addColumn("Situação");  
    gridAlunos.setModel(modelo);  
  
}
```

Observação: O grid conterà menos itens a serem exibidos que a classe para não ficar exageradamente grande. Ajuste conforme a sua necessidade.

A etapa seguinte é codificar os botões, começando pelo gravar:

```

private void btGravarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        Calendar nascimento = Calendar.getInstance();
        DateFormat formatter ;
        Date data ;
        formatter = new SimpleDateFormat("dd/MM/yyyy");
        data = (Date)formatter.parse(txtNascimento.getText());
        JOptionPane.showMessageDialog(null, data.toString());
        nascimento.setTime(data);
        Aluno aluno = new Aluno(
            Integer.parseInt(txtCod.getText()),
            txtNome.getText(),
            nascimento, txtFone.getText(), txtCidade.getText(), txtEmail.getText(),
            cbSituacao.getModel().getSelectedItem().toString()
        );

        if(verificaAluno(aluno)){
            JOptionPane.showMessageDialog(null, "aluno já cadastrado");
            return;
        }
        alunos.add(aluno);

        JOptionPane.showMessageDialog(null, "aluno cadastrado");

        //adiciona o aluno no JTable
        modelo.addRow(new Object[] {
            aluno.getCod(),
            aluno.getNome(),
            aluno.getSituacao()
        });

        // limpa os campos e posiciona o mouse no código
        txtCod.setText("0");
        txtNome.setText("");
        txtCidade.setText("");
        txtFone.setText("");
        txtEmail.setText("");
        txtNascimento.setText("");
        cbSituacao.setSelectedIndex(0);
        txtCod.requestFocus();
    }

    catch (NumberFormatException e){
        // erro ao cadastrar
        JOptionPane.showMessageDialog(null, "Erro ao cadastrar:" + e.getMessage());
    }
    catch (ParseException e){
    }
}
}

```

Rotina de verificar aluno:

```
private boolean verificaAluno(Aluno a) {
    //checa se algum código igual existe na matriz de alunos
    // retorna verdadeiro se achar e falso se não encontrar
    return alunos.stream().anyMatch((item) -> (item.getCod() == a.getCod()));
}
```

Chame o formulário no botão “alunos” de FormPrincipal:

```
private void btAlunosActionPerformed(java.awt.event.ActionEvent evt) {
    // cria a variável de Formulário
    FormAluno fr = new FormAluno();
    // mostra o formulário e seta para fechar apenas o formulário
    fr.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    fr.setVisible(true);
}
```

Teste a rotina. A próxima etapa é codificar o botão remover:

```
private void btRemoverActionPerformed(java.awt.event.ActionEvent evt) {
    // verificar se uma linha foi selecionada
    if(gridAlunos.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(null, "Nenhum aluno selecionado");
        return;
    }
    // procura o aluno
    Aluno a = procurar(gridAlunos.getValueAt(gridAlunos.getSelectedRow(),
        0).toString());
    //achou?
    if(a != null) {
        // remove o aluno
        alunos.remove(a);
        // remove a linha
        modelo.removeRow(gridAlunos.getSelectedRow());
        JOptionPane.showMessageDialog(null, "Aluno removido com Sucesso");
        // limpa os campos e posiciona o mouse no código
        txtCod.setText("0");
        txtNome.setText("");
        txtCidade.setText("");
        txtFone.setText("");
        txtEmail.setText("");
        txtNascimento.setText("");
        cbSituacao.setSelectedIndex(0);
        txtCod.requestFocus();
    }
}
```

Rotina de procurar:

```

private Aluno procurar(String cod) {
    for(Aluno item: alunos){
        int num = Integer.parseInt(cod);
        if(item.getCod() == num)
            return item;
    }
    return null;
}

```

Teste o código. Agora codifique o clique da tabela para preencher as caixas de texto:

```

private void gridAlunosMouseClicked(java.awt.event.MouseEvent evt) {
    // procura o aluno
    Aluno e = procurar(gridAlunos.getValueAt(gridAlunos.getSelectedRow(),
        0).toString());
    //achou?
    if(e!=null){
        // preenche as caixas de texto
        txtCod.setText(String.valueOf(e.getCod()));
        txtNome.setText(e.getNome());
        txtCidade.setText(e.getCidade());
        txtFone.setText(e.getTelefone());
        txtEmail.setText(e.getEmail());
        //formata a data na caixa de texto
        SimpleDateFormat s = new SimpleDateFormat("dd/MM/yyyy");
        txtNascimento.setText(s.format(e.getNascimento().getTime()));

        cbSituacao.setSelectedItem(e.getSituacao());
    }
}

```

Para finalizar codificar o evento clique do botão editar:


```

private void btEditarActionPerformed(java.awt.event.ActionEvent evt) {
    // verificar se uma linha foi selecionada
    if(gridAlunos.getSelectedRow() == -1) {
        JOptionPane.showMessageDialog(null, "Nenhum Aluno selecionado");
        return;
    }
    // procura o curso
    Aluno a = procurar(gridAlunos.getValueAt(gridAlunos.getSelectedRow(),
        0).toString());
    //achou?
    if(a != null) {
        a.setNome(txtNome.getText());
        a.setEmail(txtEmail.getText());
        a.setCidade(txtCidade.getText());
        a.setTelefone(txtFone.getText());
        a.setSituacao(cbSituacao.getSelectedItem().toString());
        Calendar nascimento = Calendar.getInstance();
        try{
            DateFormat formatter ;
            Date data ;
            formatter = new SimpleDateFormat("dd/MM/yyyy");
            data = (Date)formatter.parse(txtNascimento.getText());
            nascimento.setTime(data);
        }
        catch (ParseException e) {
        }
        // acerta o texto da descrição no grid
        gridAlunos.setValueAt(a.getNome(), gridAlunos.getSelectedRow(), 1);
        gridAlunos.setValueAt(a.getSituacao(), gridAlunos.getSelectedRow(), 2);
        JOptionPane.showMessageDialog(null, "Aluno Editado com Sucesso");
    }
}

```

Teste o código finalizado.

SALVANDO OS DADOS DE CURSO EM ARQUIVO TEXTO

Para não termos que ficar digitando os dados de curso a todo momento, vamos acrescentar uma rotina que lê os dados de curso a partir de um arquivo texto. Crie no formulário curso o seguinte método:

```

private void lerCursos() {

    try {

        FileReader arquivo = new FileReader("e://cursos.txt");

        BufferedReader buffer = new BufferedReader(arquivo);

        String linha = buffer.readLine();

        while (linha != null) {

            String[] campos;

            campos = linha.split(";");

```

```

        Curso curso = new Curso(

            Integer.parseInt(campos[0]),

            campos[1]);

        cursos.add(curso);

//adiciona o curso no JTable

linha = buffer.readLine(); // lê da segunda até a última linha

    }

} catch (FileNotFoundException ex) {

    JOptionPane.showMessageDialog(null, "erro ao ler arquivo");

} catch (IOException ex) {

    JOptionPane.showMessageDialog(null, "erro ao abrir arquivo");

}

}


```

Acrescente a chamada ao mesmo no evento window_opened:

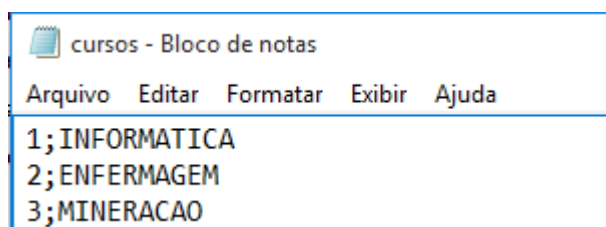
```

private void formWindowOpened(java.awt.event.WindowEvent evt) {
    modelo.addColumn("Código");
    modelo.addColumn("Descrição");
    gridCursos.setModel(modelo);
    // le os cursos que estão no arquivo cursos.txt:
    lerCursos();
}

```



Crie o arquivo no caminho especificado:



Criaremos agora a rotina para salvar os dados em cursos.txt quando o usuário fechar o formulário de cursos. Acrescente no formulário seguinte método:

```

private void salvarCursos() {

    try {

        FileWriter arquivo = new FileWriter("e://cursos.txt");

```

```

        BufferedWriter buffer = new BufferedWriter(arquivo);

        for(Curso item: cursos){

            String linha = String.valueOf(item.getCod()) + ";" +

                item.getDescricao();

            buffer.write(linha);

            buffer.newLine();

        }

        buffer.close();

    } catch (FileNotFoundException ex) {

        JOptionPane.showMessageDialog(null, "erro ao ler arquivo");

    } catch (IOException ex) {

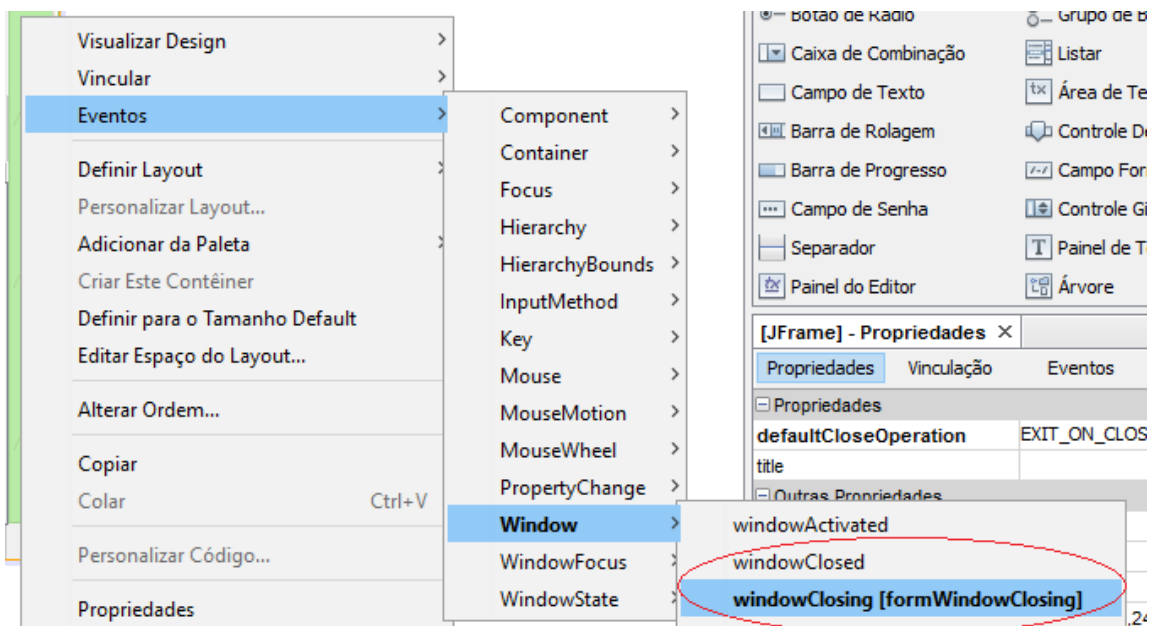
        JOptionPane.showMessageDialog(null, "erro ao abrir arquivo");

    }

}

```

Chame o mesmo no evento WINDOW_CLOSING do formulário:



```

private void formWindowClosing(java.awt.event.WindowEvent evt) {
    salvarCursos();
}

```

Exercício

1. Implemente essa mesma rotina para um arquivo de leitura e escrita de alunos usando como modelo o procedimento do arquivo de cursos.

MÓDULO TURMA.

Modelando a classe Turma, primeiro codificando as propriedades:

```
import java.util.ArrayList;

/*
 * To change this license header, choose
 * To change this template file, choose
 * and open the template in the editor.
 */

/**
 *
 * @author aluno
 */
public class Turma {

    private int cod;
    private String descricao;
    private Curso curso;
    private ArrayList<Aluno> chamada;

}
```

Crie o construtor:

```
public Turma(int cod, String descricao, Curso curso,
    ArrayList<Aluno> chamada) {
    this.cod = cod;
    this.descricao = descricao;
    this.curso = curso;
    this.chamada = chamada;
}
```

Acrescente os getters e setters (não serão mostrados aqui nesta apostila). Codificar os métodos públicos da classe:

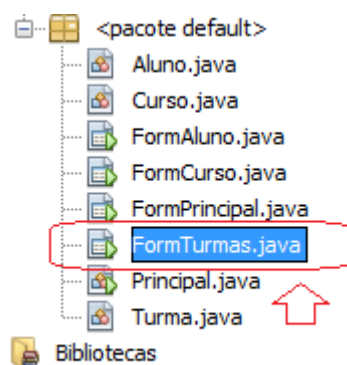
```
public void matricular(Aluno aluno) {
    // adiciona o aluno na lista de chamada
    this.getChamada().add(aluno);
}
```

```

public void cancelarMatricula(Aluno aluno){
    // adiciona o aluno na lista de chamada
    this.getChamada().remove(aluno);
}
public void trasnferirAluno(Aluno aluno){
    // procura pelo aluno na lista. Se encontrar muda su status
    // para transferido
    chamada.stream().forEach((item) -> {
        int num = aluno.getCod();
        if (item.getCod() == num) {
            item.setSituacao("TRANSFERIDO");
        }
    });
}
}

```

Começaremos agora a produzir o nosso formulário de cadastro de turmas:

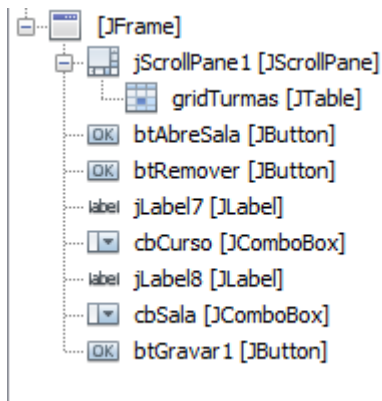


Inicie o desenho do formulário:

The image shows a Java Swing window titled 'FormTurmas.java'. The window contains a form with the following elements:

- Two dropdown menus at the top: 'Selecionar o curso:' and 'Sala:'.
- A button labeled 'criar Sala' located below the dropdown menus.
- A large, empty rectangular area in the center of the window, likely intended for displaying a list of classes or students.
- Two buttons at the bottom: 'adicionar aluno' and 'remover aluno'.

Nome das variáveis:



Iniciaremos preparando os dados a serem exibidos no COMBOBOX e NO GRID

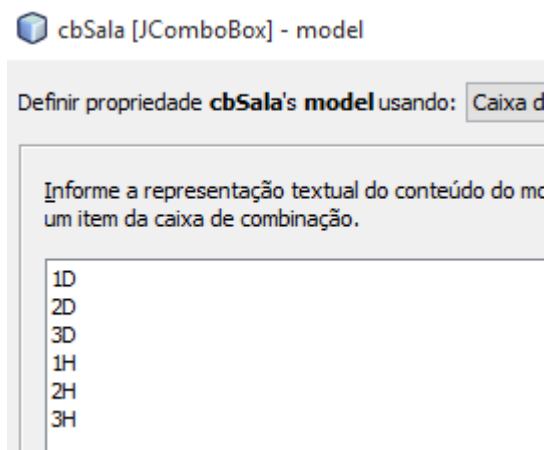
```
private void formWindowOpened(java.awt.event.WindowEvent evt) {
    // prepara a lista de cursos
    lerCursos();
    // monta o grid para turmas
    modelo.addColumn("Sala");
    modelo.addColumn("Aluno");
    modelo.addColumn("Situação");
    gridTurmas.setModel(modelo);

    for (Curso curso : cursos) {
        cbCurso.addItem(curso.getDescricao());
    }
}
```

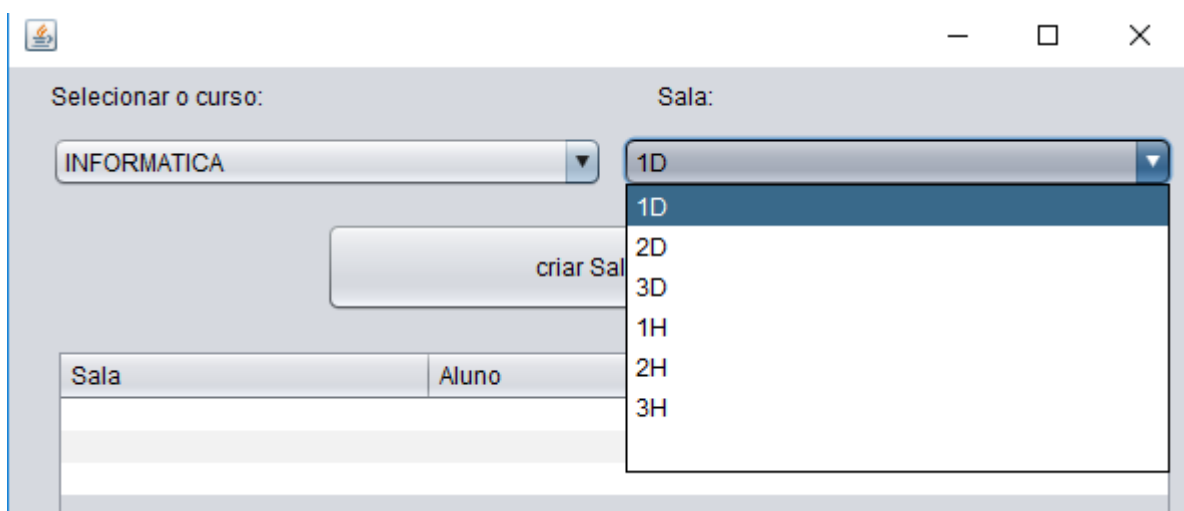
A rotina lerCursos é a mesa codificada anteriormente, copie ela para o formulário de turmas:

Se você executar apenas o formulário terá o seguinte resultado visual:

Deixe também algumas salas preenchidas no JComboBox sala para podermos cria-las:



O resultado visual será:



Passaremos a próxima etapa que será codificar o nosso botão de criar sala. Ao criar uma turma nós devemos selecionar um curso e uma sala e ele vai criar a lista de alunos em branco:

```
private void btAbreSalaActionPerformed(java.awt.event.ActionEvent evt) {

    // o código da turma será aleatório, um número entre 1 e 65536
    Random r = new Random();
    int codigo = r.nextInt(65536);
    Turma t = new Turma(
        codigo,
        cbSala.getSelectedItem().toString(),
        cursos.get(cbCurso.getSelectedIndex()),
        new ArrayList<Aluno>());


    JOptionPane.showMessageDialog(null,
        "Classe " + t.getDescricao() + " no curso " +
        t.getCurso().getDescricao() +
        " criada. Adicione alunos na sala");

}
```

Acrescentar ao formulário um ComboBox para escolher um aluno do banco de dados:



Nome do componente:

 cbAlunos [JComboBox]

Preencher o box de alunos lendo a partir do arquivo texto usando para isso o método lerAlunos(), antes porém crie a referência a lista de alunos:

```
public class FormTurmas extends javax.swing.JFrame {
    /**
     * Creates new form FormTurmas
     */

    public ArrayList<Curso> cursos = new ArrayList<>();
    // referencia a nossa turma que será cadastrada


    public ArrayList<Aluno> alunos = new ArrayList<>();

    public Turma turma = new Turma(0, "", null, new ArrayList<Aluno>());

    //modelo para o Jtable para as turmas
    DefaultTableModel modelo = new DefaultTableModel();

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

acrescente a linha
circulada



Implemente o método ler alunos:

```
private void lerAlunos() {
    try {
        FileReader arquivo = new FileReader("d://alunos.txt");
        BufferedReader buffer = new BufferedReader(arquivo);
        String linha = buffer.readLine();
        while (linha != null) {
            try {
                String[] campos;
                campos = linha.split(";");
            }
        }
    }
}
```



```

        Calendar nascimento = Calendar.getInstance();
        DateFormat formatter ;
        Date data ;
        formatter = new SimpleDateFormat("dd/MM/yyyy");
        data = (Date)formatter.parse(campos[2]);
        nascimento.setTime(data);
        Aluno aluno = new Aluno(
            Integer.parseInt(campos[0]),
            campos[1], nascimento,
            campos[3], campos[4],
            campos[5], campos[6]
        );
        alunos.add(aluno);
    } catch (ParseException err) {
        Logger.getLogger(FormTurmas.class.getName()).log(Level.SEVERE, null, err);
    }
    linha = buffer.readLine();
}

} catch (FileNotFoundException ex) {
    JOptionPane.showMessageDialog(null, "erro ao ler arquivo");
} catch (IOException ex) {
    JOptionPane.showMessageDialog(null, "erro ao abrir arquivo");
}
}
}

```

Chame a rotina no carregamento do formulário:

```

private void formWindowOpened(java.awt.event.WindowEvent evt) {
    // prepara a lista de cursos
    lerCursos();
    // monta o grid para turmas
    modelo.addColumn("Sala");
    modelo.addColumn("Aluno");
    modelo.addColumn("Situação");
    gridTurmas.setModel(modelo);

    for (Curso curso : cursos) {
        cbCurso.addItem(curso.getDescricao());
    }

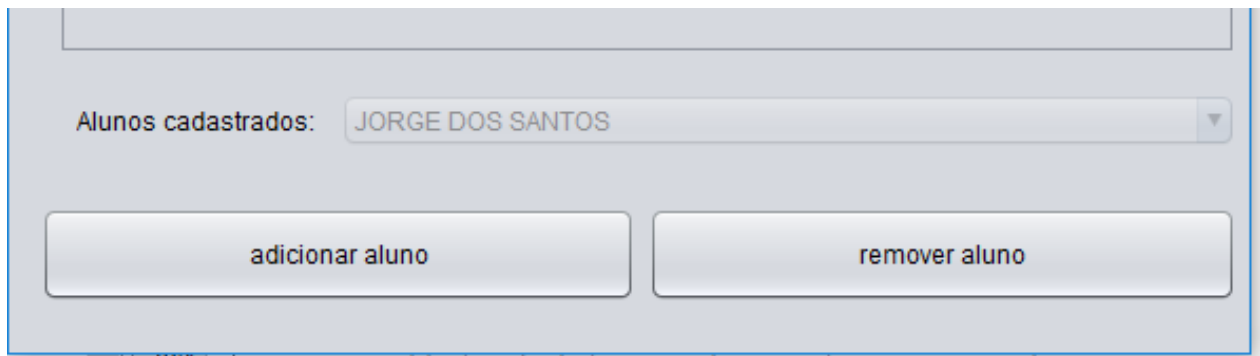
    lerAlunos();
    for (Aluno aluno : alunos) {
        cbAlunos.addItem(aluno.getNome());
    }

    // desabilita a seleção até cria a sala
    cbAlunos.setEnabled(false);
}

```



Execute e veja o resultado visual:



Note que a caixa de alunos será liberada apenas se você abrir a criação de salas. Habilite esse recurso:

```
private void btAbreSalaActionPerformed(java.awt.event.ActionEvent evt) {  
  
    // o código da turma será aleatório, um número entre 1 e 65536  
    Random r = new Random();  
    int codigo = r.nextInt(65536);  
    Turma t = new Turma(  
        codigo,  
        cbSala.getSelectedItem().toString(),  
        cursos.get(cbCurso.getSelectedIndex()),  
        new ArrayList<Aluno>());  
  
    JOptionPane.showMessageDialog(null,  
        "Classe " + t.getDescricao() + " no curso " +  
        t.getCurso().getDescricao() +  
        " criada. Adicione alunos na sala");  
  
    // habilita lista de alunos  
    cbAlunos.setEnabled(true);  
}
```

Programar o botão adicionar aluno:

```
private void btGravarActionPerformed(java.awt.event.ActionEvent evt) {  
    Aluno novo = alunos.get(cbAlunos.getSelectedIndex());  
    //adiciona um novo aluno  
    turma.getChamada().add(novo);  
    modelo.addRow(new Object[] {  
        turma.getDescricao(),  
        novo.getNome(),  
        novo.getSituacao()  
    });  
}
```

Modifique a seguinte linha para usarmos a referência de turma para todo o nosso formulário:

```
private void btAbreSalaActionPerformed(java.awt.event.ActionEvent evt) {  
  
    // o código da turma será aleatório, um número entre 1 e 65536  
    Random r = new Random();  
    int codigo = r.nextInt(65536);  
    Turma t = new Turma(  
        codigo,  
        cbSala.getSelectedItem().toString(),  
        cursos.get(cbCurso.getSelectedIndex()),  
        new ArrayList<Aluno>());  
  
    JOptionPane.showMessageDialog(null,  
        "Classe " + t.getDescricao() + " no curso " +  
        t.getCurso().getDescricao() +  
        " criada. Adicione alunos na sala");  
  
    // habilita lista de alunos  
    cbAlunos.setEnabled(true);  
    turma = t;  
}
```

← acrescente a seguinte linha

Execute o código e veja o resultado:

The screenshot shows a Java Swing application window with a light gray background. At the top, there are two labels: "Selecionar o curso:" and "Sala:". Below "Selecionar o curso:" is a dropdown menu with "ENFERMAGEM" selected. Below "Sala:" is a dropdown menu with "1H" selected. In the center, there is a button labeled "criar Sala". Below the button is a table with three columns: "Sala", "Aluno", and "Situação". The table has two rows: the first row has "1H", "JOAO DA SILVA", and "ATIVO"; the second row has "1H", "JOANA DA MATA", and "ATIVO". Below the table is a label "Alunos cadastrados:" followed by a dropdown menu with "JOANA DA MATA" selected. At the bottom, there are two buttons: "adicionar aluno" and "remover aluno".

Sala	Aluno	Situação
1H	JOAO DA SILVA	ATIVO
1H	JOANA DA MATA	ATIVO

Codificar o botão remover aluno:

```
private void btRemoverActionPerformed(java.awt.event.ActionEvent evt) {
    int pos = gridTurmas.getSelectedRow();
    // remove o aluno
    turma.getChamada().remove(pos);
    // remove a linha do grid
    modelo.removeRow(pos);
}
```

Agora é possível adicionar e remover alunos na turma. Para evitar que um aluno seja cadastrado duas vezes na turma, crie o seguinte método:

```
private boolean procurar(String cod) {
    for(Aluno item: turma.getChamada()){
        int num = Integer.parseInt(cod);
        if(item.getCod() == num)
            return true;
    }
    return false;
}
```

E modifique o botão adicionar:

```
private void btGravarActionPerformed(java.awt.event.ActionEvent evt) {
    Aluno novo = alunos.get(cbAlunos.getSelectedIndex());

    boolean achou = procurar(String.valueOf(novo.getCod()));
    //não achou?
    if(!achou){

        turma.getChamada().add(novo);
        modelo.addRow(new Object[] {
            turma.getDescricao(),
            novo.getNome(),
            novo.getSituacao()
        });
    }
    else
    {
        JOptionPane.showMessageDialog(null, "Aluno já cadastrado na turma.");
    }
}
```

Nosso projeto se encontra finalizado.