Desenvolvimento Rápido de Aplicações Padrões de Arquitetura de Software

Profa. Joyce Miranda

- Uma historinha...
 - ▶ João é um programador...
 - ▶ João está ansioso...
 - ▶ Seu chefe pediu pra ele desenvolver um "sisteminha"...



- Uma historinha...
 - No auge de sua empolgação, João consegue desenvolver o código-fonte de uma única vez, como um relâmpago.



- Uma historinha...
 - É chegada a hora da verdade: o primeiro teste!
 - ▶ Nessa hora tudo acontece, só o sistema que não funciona.
 - João desconfia de tudo: do hardware, do compilador, do Windows...

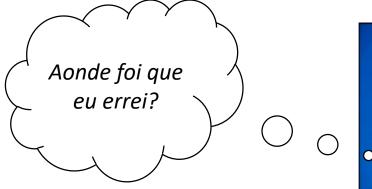


Uma historinha...



- Analisando a solução de João:
 - ▶ Ele não se preocupou com comentários, modularização e endentação;
 - Para agilizar o desenvolvimento, ele tomou a sábia decisão de usar exemplos da internet;
 - ▶ E para mostrar que é muito eficiente, antes mesmo dos testes, ele modificou as rotinas e as deixou mais "rebuscadas".

- Uma historinha...
 - Desesperado, João sai mexendo no código aleatoriamente a fim de encontrar e resolver o problema.
 - ▶ João vai gastar pelo menos duas vezes o tempo previsto para realizar a tarefa.
 - Conclusão
 - ▶ João tentou ser <u>eficiente</u> em tempo, mas não foi <u>eficaz</u>!

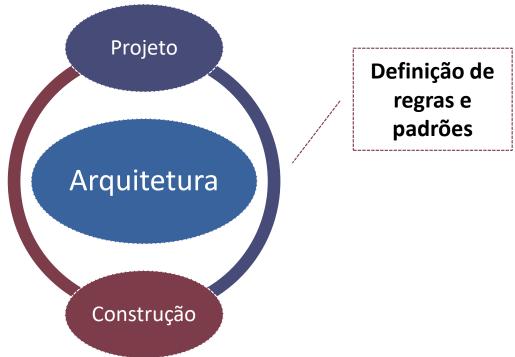




Desenvolver rápido não está relacionado a desenvolver um software no menor espaço de tempo, mas em desenvolver um software de qualidade em um tempo factível.

Escopo

Conjunto de decisões estratégicas relacionadas à <u>estrutura</u> e ao <u>comportamento</u> do software a fim de atender seus requisitos funcionais e não funcionais.

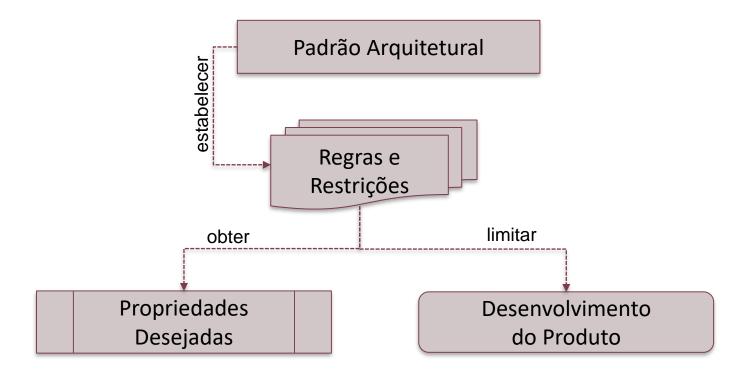


Definição formal

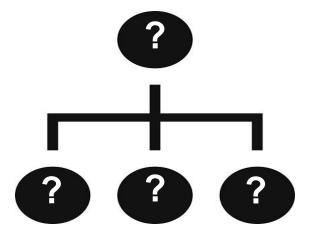
Organização fundamental de um sistema, expressa nos seus componentes, nos <u>relacionamentos</u> entre eles e com o ambiente, e nos <u>princípios</u> que governam seu projeto e sua evolução.

Padrão Arquitetural

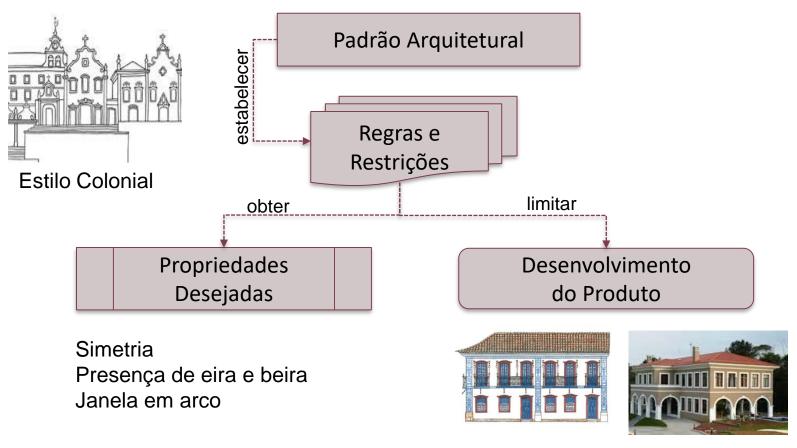
Solução estruturada de design, pronta para ser reutilizada para solucionar problemas recorrentes de arquitetura.



- Padrão Arquitetural
 - Deve ser abstrato.
 - É um template que precisa ser refinado
 - Identifica a estrutura geral da organização do software
 - Define elementos, relações e regras a serem seguidas que já tiveram sua utilidade avaliada em soluções de problemas passados.

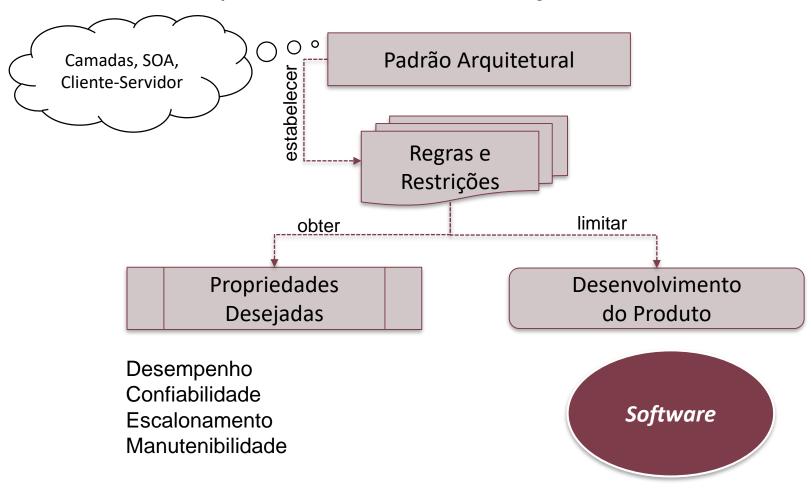


Padrão Arquitetural – Na construção civil



Arquiteturas distintas, mesmo padrão

▶ Padrão Arquitetural – Na construção de software



Padrão Arquitetural

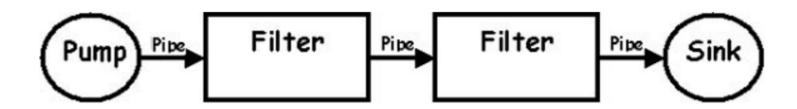
 Cada padrão propõe uma maneira de organizar o sistema e define características que indicam quando devemos utilizá-lo.



Os padrões <u>não</u> são mutuamente exclusivos.

Pipes & Filters

- Contexto
 - Divisão de uma tarefa de processamento em uma sequência de passos (Filters) que são conectados por canais (Pipes).
 - Sequência de transformações sobre uma fonte de dados.
- Características
 - Arquitetura linear
 - ▶ *Filter*: executa transformações sobre os dados de entrada.
 - ▶ *Pipe*: conector que passa dados de um filtro para outro.



- Pipes & Filters
 - Shell do Linux
 - A saída de um programa pode ser "linkada" como a entrada de outro programa

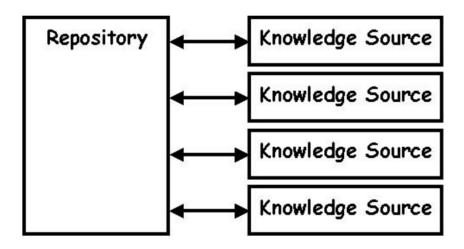
```
$ ls | grep b | sort -r | tee arquivo.out | wc -l
```

A saída será a quantidade de arquivos que contém a letra "b" e o nome desses arquivos salvos em "arquivo.out".

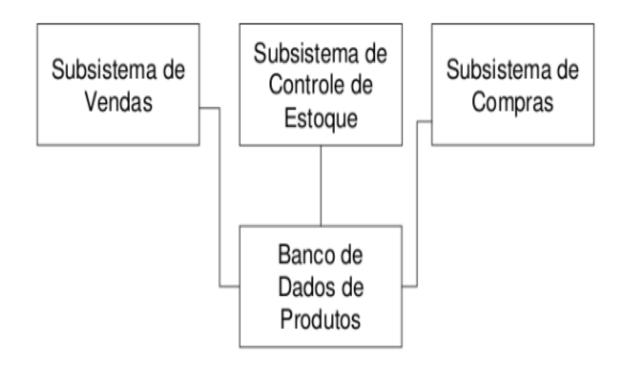
Exemplo de <u>componentes</u> (filtros) com independência computacional <u>que executam uma transformação nos dados de entrada</u> e <u>condutores</u> que <u>transmitem os dados de saída de um componente a outro</u>.

Repository

- Contexto
 - Útil quando subsistemas compartilham um mesmo repositório de dados.
- Características
 - Todos os subsistemas podem ler e escrever no repositório
 - A forma de acesso e a sincronização das interações são definidas pelo repositório.

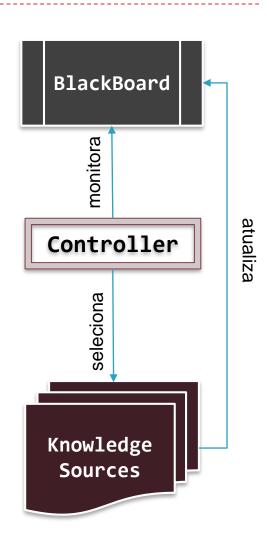


- Repository
 - Ex: Sistemas Gerenciadores de Banco de Dados (SGBD)



BackBoard

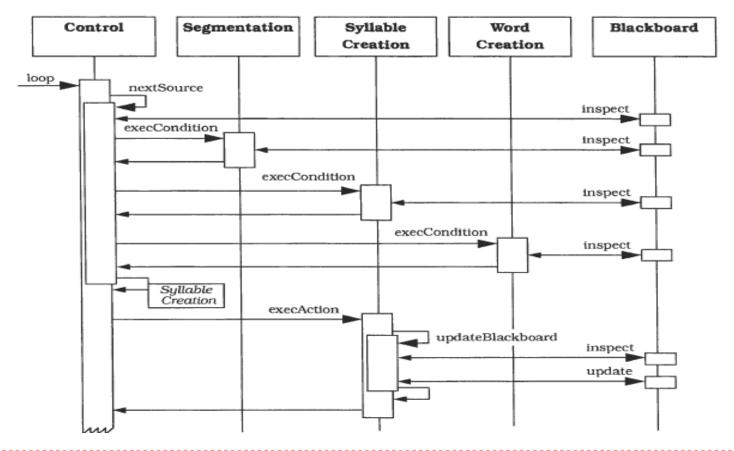
- Contexto
 - Solução para problemas não determinísticos.
 - Subsistemas reúnem seus conhecimentos para alcançar uma solução aproximada.
- Características
 - ▶ BlackBoard: elemento central de armazenamento
 - Knowledge Sources: subsistemas que resolvem aspectos específicos do problema
 - Controller: monitora mudanças e decide qual ação executar em seguida



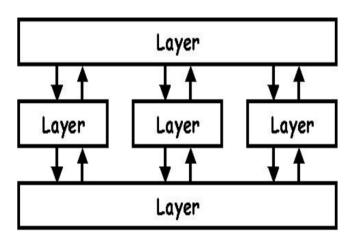
BlackBoard

cada fonte vai criar uma hipótese sobre a possível solução

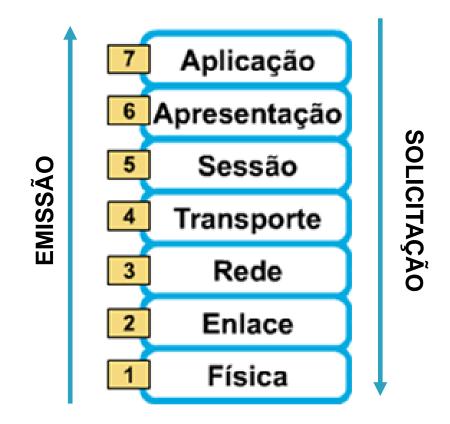
Programa de reconhecimento de voz.



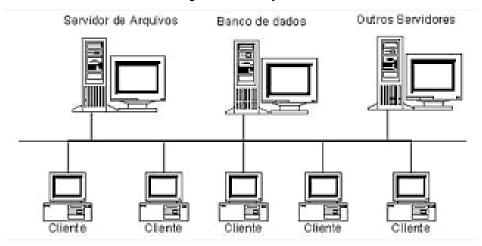
- Layers (Camadas)
 - Contexto
 - Decomposição do sistema em camadas, com alto grau de abstração e baixa dependência entre as camadas.
 - Características
 - Cada camada provê um conjunto de funcionalidades bem específicas.
 - ☐ Fornece serviços para a camada superior
 - □ Solicita serviços da camada inferior.
 - Comunicação apenas entre camadas vizinhas



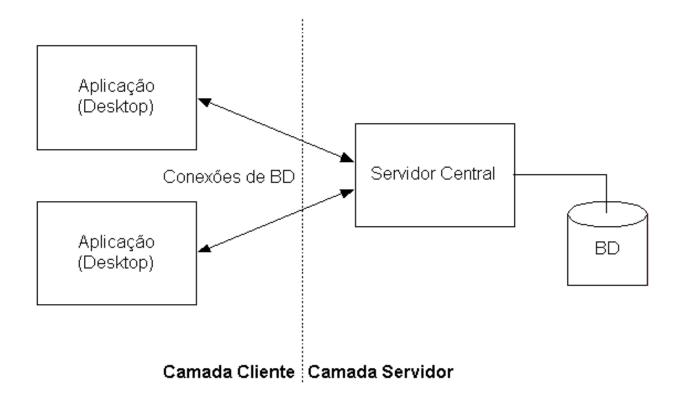
- Layers (Camadas)
 - Ex: Modelo de Referência OSI



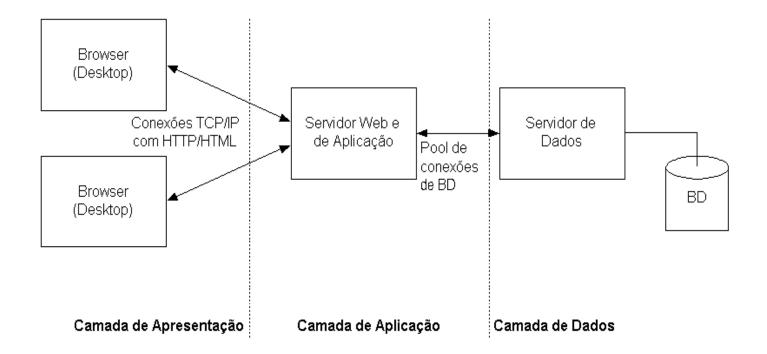
- Cliente-Servidor (2 Camadas)
 - Contexto
 - Sistemas distribuídos que seguem o modelo de comunicação requisição/resposta.
 - Características
 - Um cliente faz um pedido ao servidor e espera pela resposta.
 - ▶ O servidor executa o serviço e responde ao cliente.



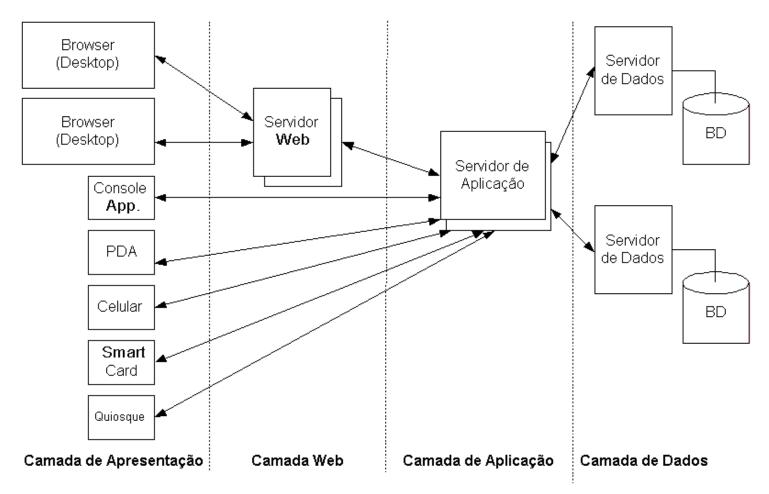
- Cliente-Servidor (2 Camadas)
 - Camada cliente trata da lógica de negócio e da interface
 - Camada servidor trata dos dados



- Cliente-Servidor (3 Camadas)
 - Camada de apresentação (interface)
 - Camada de aplicação (lógica de negócio)
 - Camada de dados



Cliente-Servidor (N Camadas)



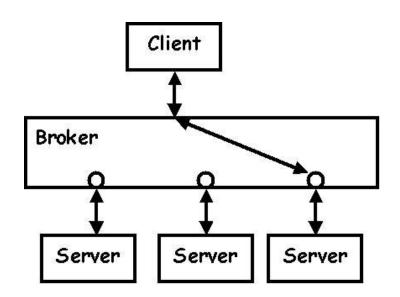
Broker/SOA

Contexto

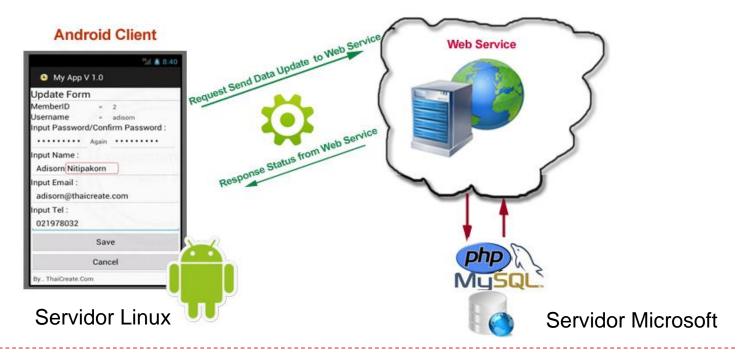
- Clientes e servidores interagem por meio de um intermediador (broker).
- Arquitetura Orientada a Serviços
 - Comunicação estabelecida por meio de chamadas remotas a serviços.

Características

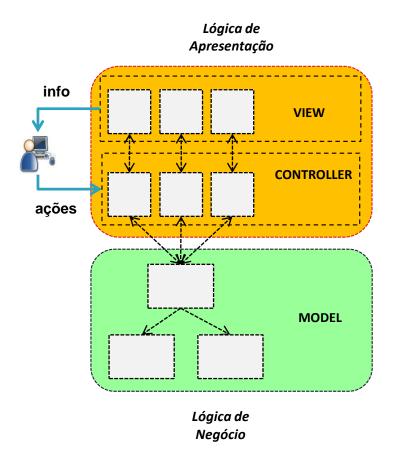
- □ Os servidores se registram junto ao broker e tornam seus serviços disponíveis aos clientes.
- Clientes acessam a funcionalidade dos servidores enviando requisições através do broker.



- Broker/SOA
 - Exemplo
 - WebService
 - □ Proporcionar interoperabilidade entre sistemas distribuídos, independente da plataforma e da linguagem de programação.

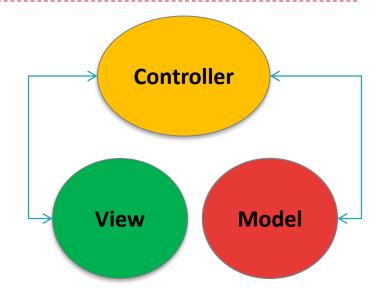


- ▶ MVC Model View Controller
 - Contexto
 - Separação entre <u>lógica de negócio</u> e <u>lógica de apresentação</u>.
 - Características
 - Camadas
 - □ Model
 - □ Regras de negócio e acesso a dados
 - □ View
 - Interface com o usuário
 - □ Controller
 - ☐ Intermedia a comunicação entre Model e View.



MVC

- Fundamentos
 - Padrão Arquitetural de Software
 - □ Não é um padrão de projeto
 - Dividir a aplicação em camadas com responsabilidades específicas



- Vantagens
 - Legibilidade
 - Facilidade de manutenção
 - Independência maior entre as camadas

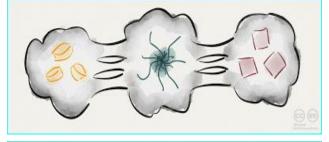
MVC

Fundamentos

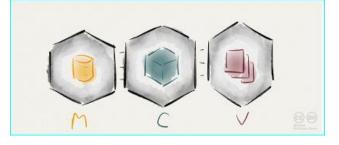
10



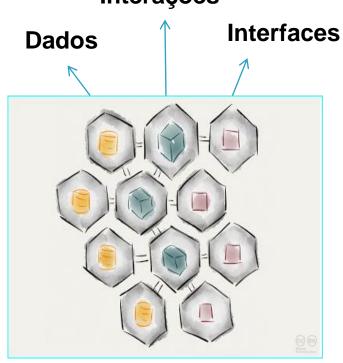
2º



30



Interações



N camadas M, V, C

* existindo regras de interação entre elas

MVC

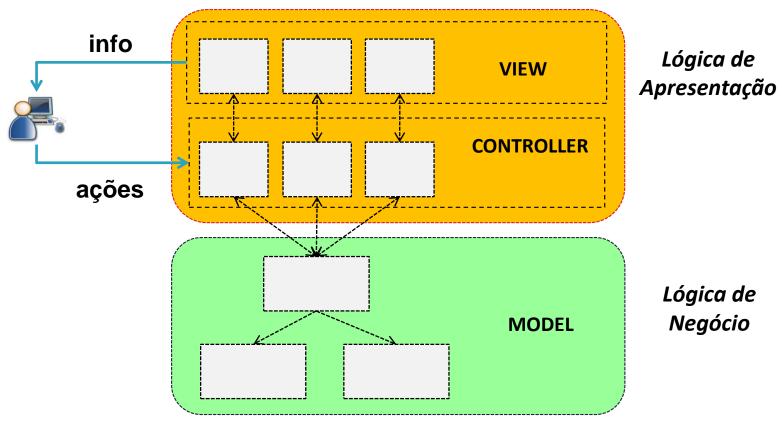
Fundamentos

- Regras
 - □ A comunicação entre camadas deve ser sempre intermediada pela camada Controller
 - □ Controladores não devem se comunicar entre si.



MVC

► Fundamentos – Separação de Responsabilidades

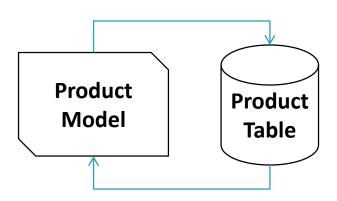


- MVC
 - Fundamentos
 - **▶ MODEL**
 - □ Composta por classes que representam o domínio da aplicação

Regras de Negócio

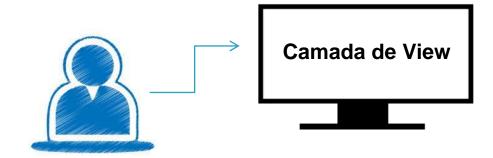
Representação dos Dados

Camada de Acesso a Dados

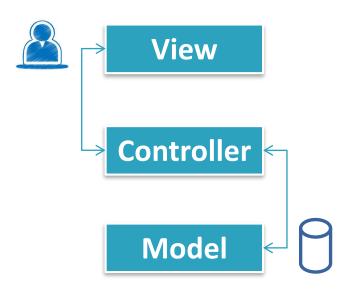


MVC

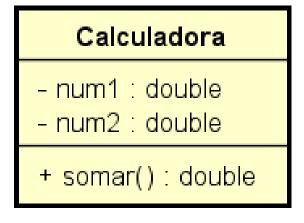
- Fundamentos
 - VIEW
 - □ Representa a camada de interface com o usuário
 - ☐ Invoca métodos do *Model* por meio do *Controller*
 - □ Apresenta o resultado dado como resposta a uma requisição
 - ☐ Monitora mudanças do *Model* e o apresenta atualizado



- MVC
 - Fundamentos
 - **▶ CONTROLLER**
 - □ Processa ações do usuário (capturadas pela View)
 - □ Apresenta novas 'Views'
 - Atualiza modelo



- MVC na prática
 - Estudo de caso: Calculadora

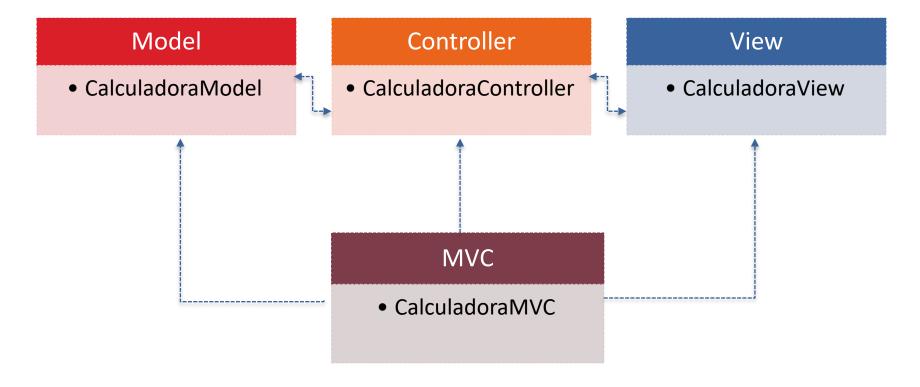




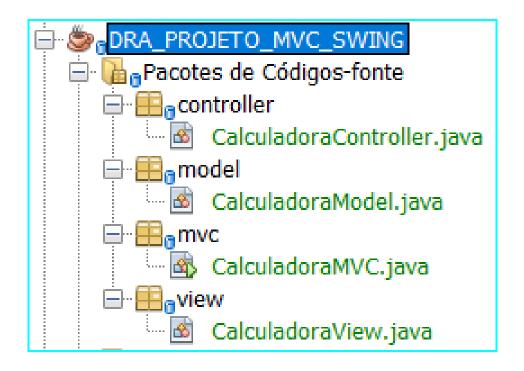
Código disponível em:

https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_PROJETO_MVC_SWING/src/calculadora

- MVC na prática
 - Estudo de caso: Calculadora
 - Estrutura de Classes



- MVC na prática
 - Estudo de caso: Calculadora
 - Estrutura de Pacotes



- MVC na prática
 - Estudo de caso: Calculadora
 - Modelo

Calculadora

- num1 : double
- num2 : double
- + somar(): double

Implementando elementos do modelo

```
public class CalculadoraModel {
   private double num1;
   private double num2;
   public double getNum1() {
        return this.num1;
    public double getNum2() {
        return this.num2;
    public void setNum1 (double num1) {
        this.num1 = num1;
    public void setNum2(double num2) {
        this.num2 = num2;
   public double somar() {
       return num1 + num2;
```

- MVC na prática
 - Estudo de caso: Calculadora
 - View

Definindo elemento de interface

```
public class CalculadoraView extends JFrame {
    JPanel painel = new JPanel();
    JTextField txtNum1 = new JTextField(5);
    JTextField txtNum2 = new JTextField(5);
    JTextField txtResultado = new JTextField(5);
    JButton btnSomar = new JButton("Somar");
    public CalculadoraView() {
        this.setSize(400, 200);
        painel.add(txtNum1);
        painel.add(txtNum2);
        painel.add(btnSomar);
        painel.add(txtResultado);
        this.add(painel);
```

- MVC na prática
 - Estudo de caso: Calculadora
 - View (cont.)

Gets para campos de entrada
Sets para campos de saída

```
public String getTxtNum1() {
    return txtNum1.getText();
}

public String getTxtNum2() {
    return txtNum2.getText();
}

public void setResultado(String resultado) {
    txtResultado.setText(resultado);
}
```

- MVC na prática
 - Estudo de caso: Calculadora
 - View (cont.)

Adicionando ação aos elementos da tela

```
public void addBtnSomarListener (ActionListener listenForBtnSomar) {
   btnSomar.addActionListener(listenForBtnSomar);
}

public void displayMensagemDeErro(String msg) {
   JOptionPane.showMessageDialog(this, msg);
}
```

- MVC na prática
 - Estudo de caso: Calculadora
 - Controller

Linkando ação do botão

```
public class CalculadoraController {
    private CalculadoraView theView;
    public CalculadoraController(CalculadoraView theView) {
        this.theView = theView;
        this.theView.addBtnSomarListener(new SomarListener());
        this.theView.setVisible(true);
```

- MVC na prática
 - Controller (cont.)

```
□class SomarListener implements ActionListener{
         @Override
         public void actionPerformed(ActionEvent e) {
             try{
                 double num1 = Double.parseDouble(theView.getTxtNum1());
                 double num2 = Double.parseDouble(theView.getTxtNum2());
                 CalculadoraModel theModel = new CalculadoraModel();
                 theModel.setNum1(num1);
                 theModel.setNum2(num2);
                 double resultado = theModel.somar();
                 theView.setResultado(Double.toString(resultado));
             }catch (NumberFormatException ex) {
                 theView.displayMensagemDeErro(
                         "Entre com dos valores numéricos!!");
```

- MVC na prática
 - Estudo de caso: Calculadora
 - Classe MVC Sincronizadora

```
public class CalculadoraMVC {
   public static void main(String args[]) {
      CalculadoraView theView = new CalculadoraView();
      CalculadoraController controller = new CalculadoraController(theView);
   }
}
```

- Praticando...
 - Utilizando o padrão MVC
 - Crie uma interface responsável por receber a data de nascimento de uma pessoa e exibir sua idade.

Pessoa - dataNascimento : LocalDate + calcIdade(anoAtual : int) : int

Data de Nascimento	Calcular Idade	Idade	

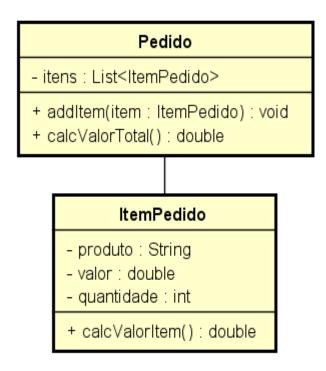
- Praticando...
 - Utilizando o padrão MVC
 - Crie uma interface responsável por receber o número de uma conta e fornecer as funcionalidades de depósito e saque.
 - □ Regra de Negócio: O saldo só poderá ser modificado mediante as operações de saque e depósito.

Conta
- numero : int
- saldo : double
+ sacarValor(valor : double) : boolean
+ depositarValor(valor : double) : boolean



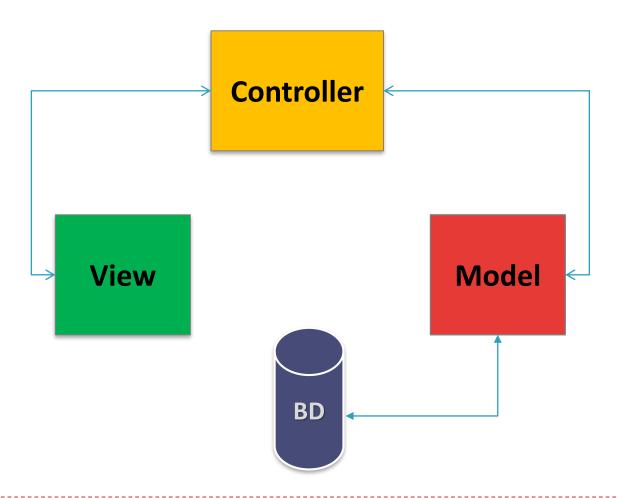


- Tarefa de Implementação
 - Utilizando o padrão MVC
 - Crie uma interface responsável por registrar os itens de um pedido e atualizar o valor total do pedido.

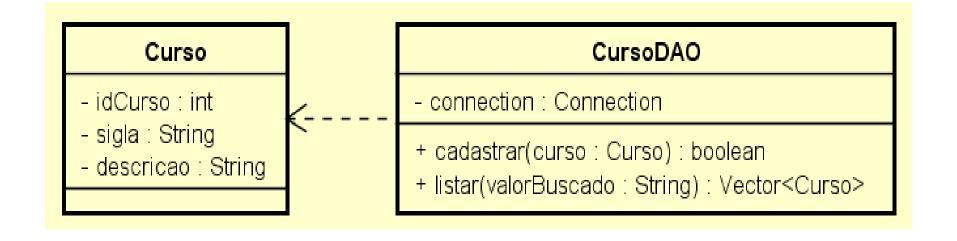




MVC com acesso ao BD



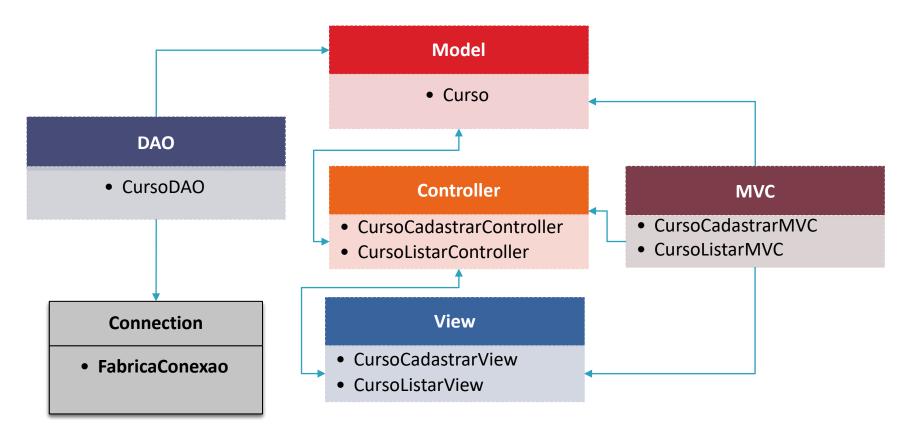
- MVC na prática
 - Estudo de caso: Curso



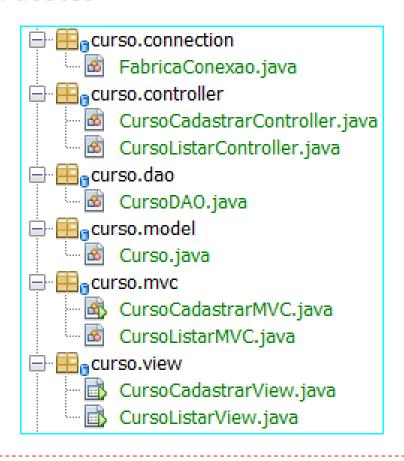
Código disponível em:

https://github.com/joyceMiranda/codigosDeExemplo/tree/master/DRA_PROJETO_MVC_SWING/src/curso

- MVC na prática
 - Estrutura de Classes



- MVC na prática
 - Estrutura de Pacotes



- MVC na prática
 - Base de Dados
 - MySQL
 - Workbench

Curso

- idCurso : int
- sigla : String
- descricao : String

```
-- Database: 'sysControleAcademico'
CREATE DATABASE sysControleAcademico;
use sysControleAcademico;
 -- Table structure for table 'curso'
□CREATE TABLE "curso" (
   "idcurso" int(11) NOT NULL AUTO INCREMENT,
   "sigla" varchar(15) NOT NULL,
   "descricao" varchar(200) NOT NULL,
   PRIMARY KEY ("idcurso")
   ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- MVC na prática
 - FabricaConexao Conexão com JDBC
 - Necessário incluir o driver de conexão ao projeto.

```
public class FabricaConexao {
   public static Connection getConnection() {
       try{
           String host = "jdbc:mysql://localhost/sysControleAcademico";
           String user = "root";
           String password = "";
           return DriverManager.getConnection(
                   host, user, password);
       }catch(SQLException e) {
           throw new RuntimeException(e);
```

MVC na prática

Curso (Model)

Curso

- idCurso : int
- sigla : String
- descricao : String

```
public class Curso {
   private int idCurso;
   private String sigla;
   private String descricao;
   public Curso() {...2 linhas }
   public Curso(int idCurso, String sigla, String descricao) {...5 linhas }
   public int getIdCurso() {...3 linhas }
   public void setIdCurso(int idCurso) {...3 linhas }
   public String getSigla() {...3 linhas }
   public void setSigla(String sigla) {...3 linhas }
   public String getDescricao() {...3 linhas }
   public void setDescricao(String descricao) {...3 linhas }
```

MVC na prática

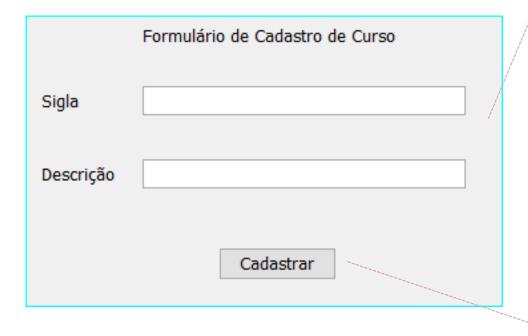
CursoDAO

- connection : Connection
- + cadastrar(curso : Curso) : boolean
- + listar(valorBuscado : String) : Vector<Curso>

CursoDAO (Model) – Trecho de Código

```
public class CursoDAO {
    private Connection connection;
    public CursoDAO() {
        this.connection = FabricaConexao.getConnection();
    public boolean cadastrar(Curso curso) {
        String sql = "INSERT INTO CURSO VALUES (0, ?, ?)";
        try {
            PreparedStatement ps = connection.prepareStatement(sql,
                    Statement.RETURN GENERATED KEYS);
            ps.setString(1, curso.getSigla());
            ps.setString(2, curso.getDescricao());
            ps.execute();
            //recuperando id gerado pelo banco
            final ResultSet rs = ps.getGeneratedKeys();
            rs.next();
            curso.setIdCurso(rs.getInt(1));
            connection.close():
            return true;
        } catch (SQLException e) {
            throw new RuntimeException(e);
```

- MVC na prática
 - CursoCadastrarView



Implementar gets
para campos de
entrada e métodos
para adicionar ação
na interface

Adicionar ação aos elementos da tela

```
public class CursoCadastrarController {
                                              CursoCadastrarController
    CursoCadastrarView theView;
    public CursoCadastrarController (CursoCadastrarView theView) {
        this.theView = theView;
        theView.addBtnCadastrarEventListener(new CadastrarCursoListener());
        theView.setVisible(true);
    class CadastrarCursoListener implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            String sigla = theView.getTxtSigla();
            String descricao = theView.getTxtDescricao();
                                                                  Ação do
            //camada modelo
            Curso curso = new Curso();
                                                                   botão
            curso.setSigla(sigla);
                                                                 cadastrar
            curso.setDescricao(descricao);
            CursoDAO dao = new CursoDAO();
            boolean cadastrou = dao.cadastrar(curso);
            if(cadastrou) {
                theView.showMessage ("Cadastro realizado com sucesso");
            }else{
                theView.showMessage("Cadastro não realizado!!");
```

- MVC na prática
 - CursoMVC Trecho de Código

- connection : Connection

- + cadastrar(curso : Curso) : boolean
- + listar(valorBuscado : String) : Vector<Curso>

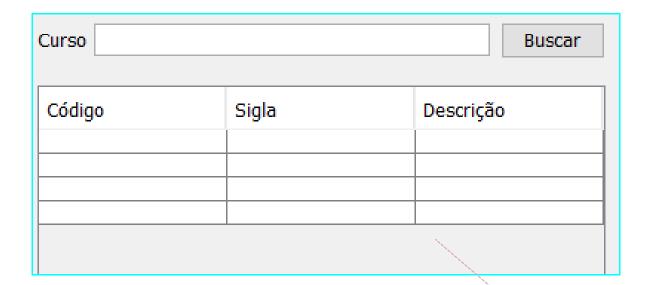
CursoDAO

MVC na prática

CursoDAO – Trecho de Código (Listar)

```
public Vector<Curso> listar(String valorBuscado) {
    String sql = "SELECT * FROM CURSO c WHERE c.descricao LIKE ? ";
    try{
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setString(1, '%' + valorBuscado + '%');
        ResultSet rs = ps.executeQuery();
        Vector<Curso> listaCursos = new Vector();
        while (rs.next()) {
            int codigo = rs.getInt("idCurso"); /** nome do campo no BD **/
            String sigla = rs.getString("sigla");
            String descricao = rs.getString("descricao");
            Curso curso = new Curso(codigo, sigla, descricao);
            listaCursos.add(curso);
        ps.close();
        connection.close();
        return listaCursos;
    }catch (SQLException e) {
        throw new RuntimeException(e);
```

- MVC na prática
 - CursoListarView



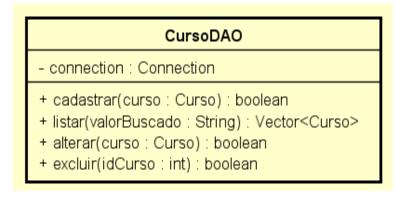
Implementar gets e sets para campos de entrada e métodos para adicionar ação na interface

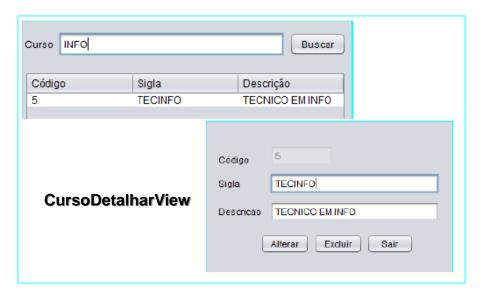
- MVC na prática
 - CursoListarController Trecho de código

```
class ListarCursoListener implements ActionListener{
    @Override
   public void actionPerformed(ActionEvent e) \ {
       String valorBuscado = theView.getTxtValorBuscado();
       CursoDAO dao = new CursoDAO();
       listaCursos = dao.listar(valorBuscado);
       Vector conjuntoLinhas = new Vector();
                                                              Ação do botão
                                                                  buscar
        for(Curso curso: listaCursos){
           Vector linha = new Vector();
           linha.add(curso.getIdCurso());
           linha.add(curso.getSigla());
            linha.add(curso.getDescricao());
            conjuntoLinhas.add(linha);
```

- MVC na prática
 - CursoListarController Trecho de código (cont.)

- Praticando...
 - Utilizando o padrão MVC
 - Crie uma interface responsável por alterar e excluir cursos de acordo com as especificações abaixo.





- MVC na prática
 - CursoListarController Trecho de código

Ação do Click da Tabela

```
class DetalharCursoListener implements MouseListener{
    @Override
    public void mouseClicked (MouseEvent e) {
        int indice = theView.getTblCursos();
        Curso curso = listaCursos.get(indice);
        new CursoDetalharView(curso).setVisible(true);
}
```

CursoListarView – Trecho de código

```
public int getTblCursos() {
    return this.tblCursos.getSelectedRow();
}
```

PROGRAMAÊ!

- Tarefa de Implementação
 - Utilizando o padrão MVC
 - Com base no modelo abaixo,
 - ☐ Crie uma interface gráfica para manipular os objetos.

