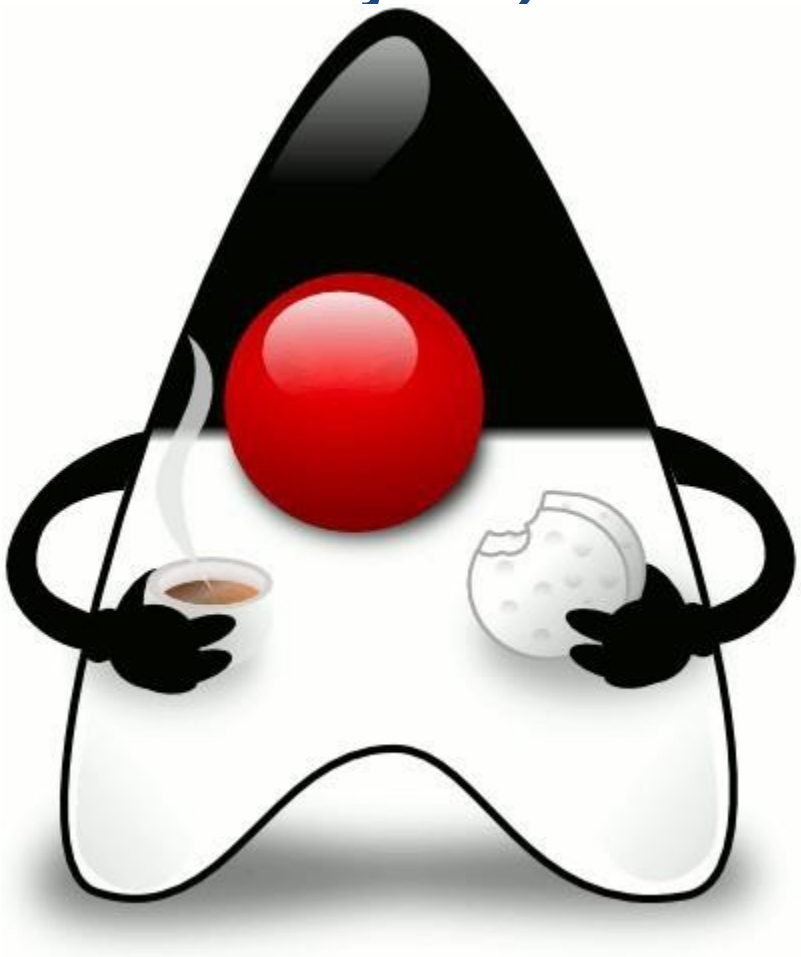
UNIP
UNIVERSIDADE PAULISTA

# Disciplina: Aplicação de Linguagem de Programação Orientada a Objetos (ALPOO)

Prof. Saulo –- saulo.silva@docente.unip.br

UNIP
UNIVERSIDADE PAULISTA

# Design Patterns – DAO (Data Access Object)



# Design Patterns

- A idéia de padrões foi apresentada por Christopher Alexander em 1977 no contexto de arquitetura de prédios e cidades:

  *“Cada padrão descreve um problema que ocorre repetidamente de novo e de novo em nosso ambiente, e então descreve a parte central da solução para aquele problema de uma forma que você possa usar essa solução um milhão de vezes, sem nunca implementá-las duas vezes da mesma forma”*

UNIP
UNIVERSIDADE PAULISTA

# Catálogo de soluções

- Um padrão encerra o conhecimento de uma pessoa muito experiente em um determinado assunto de forma que esse conhecimento pode ser transmitido para outras pessoas menos experientes

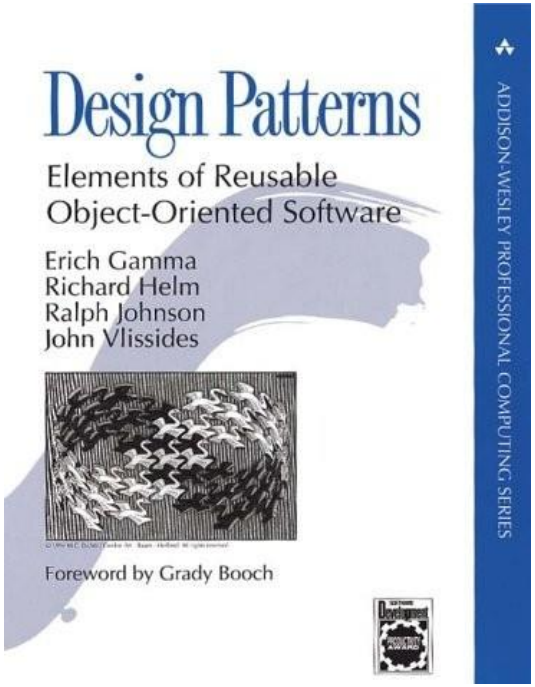
- Outras ciências (por exemplo Química e engenharia) possuem catálogos de soluções.

- Desde 1995, o desenvolvimento de software passou a ter o seu primeiro catálogo de soluções para projetos de software: O livro GoF

# Gang of Four (GoF)

E. Gamma and R. Helm and R. Johnson and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

http://www.corej2eepatterns.com/DataAccessObject.htm



# Gang of Four (GoF)

- Criaram um vocabulário comum para conversar sobre projetos de software.
- Soluções que não tinham nome passaram a ter.
- Ao invés de discutir um sistema utilizando termos como pilha, arvore, fila, lista encadeada, passaram a se falar de coisas em mais alto nível como Fábricas (Factory), Fachadas (Facade), Observador (Observer), etc;
- Inicialmente o livro foi escrito voltado para a linguagem C++

UNIP
UNIVERSIDADE PAULISTA

# Exemplos de Design Patterns

- Observer
- Decorator
- Factory
- Singleton
- Command
- Facade
- **Data Access Object (DAO)**

# DAO

Data Access Object (DAO)

Motivação: Necessidade de armazenamento de dados. Todos os sistemas modernos utilizam alguma forma de acesso e manipulação de dados. Esses programas necessitam de uma forma de se comunicar com as bases de dados.

# DAO

XMLs, a grosso modo, são um conjunto de tags dispostas de modo hierárquico, contendo informações estruturadas. Arquivos texto são um conjunto de caracteres. Bancos de dados relacionais são um conjunto de tabelas, colunas, e linhas. Aplicações Java são um conjunto de objetos.

O DAO deve funcionar como um tradutor dos mundos. Suponha um banco relacional. O DAO deve saber buscar os dados do banco e converter em objetos para ser usado pela aplicação. Semelhantemente, deve saber como pegar os objetos, converter em instruções SQL e mandar para o banco de dados. É assim que um DAO trabalha.

# DAO

Devido à sua qualidade de tradutor, o DAO abstrai a origem e o modo de obtenção / gravação dos dados, de modo que o restante do sistema manipula os dados de forma transparente, sem se preocupar com o que acontece por trás dos panos. Isso ajuda muito em processos de migrações de fonte de dados e testes unitários.

# DAO

É muito comum em códigos de programadores iniciantes vermos a base de dados sendo acessada em diversos pontos da aplicação, de maneira extremamente explícita e repetitiva.
Isso vai contra os princípios da OO.

O DAO também nos ajuda a resolver este problema, provendo pontos unificados de acesso a dados.
Desse modo, a lógica de interação com a base de dados ficam em lugares específicos e especializados nisso, além de eliminar códigos redundantes, facilitando a manutenção e futuras migrações.

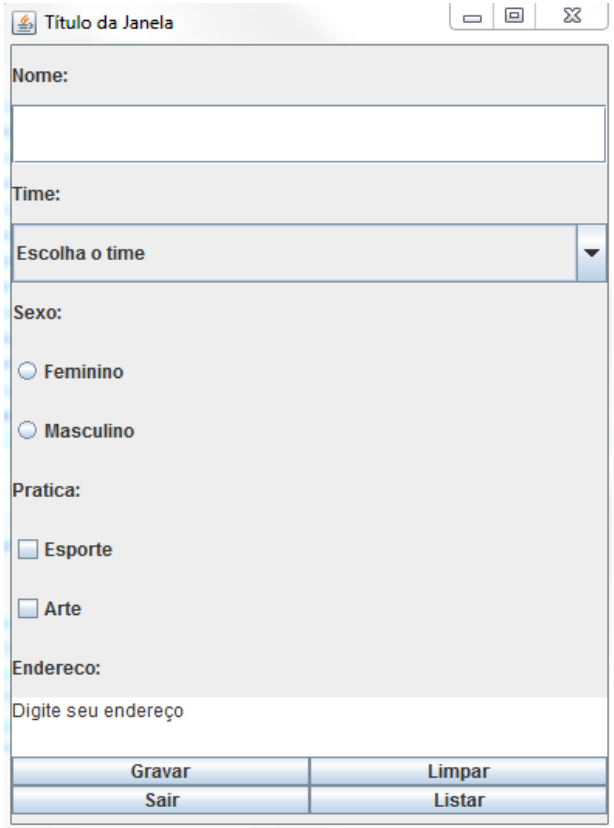
# DAO

Geralmente, temos um DAO para cada objeto do domínio do sistema (Produto, Cliente, Compra, etc.), ou então para cada módulo, ou conjunto de entidades fortemente relacionadas.

Cada DAO deve possuir uma interface, que especifica os métodos de manipulação de dados.

# DAO - Exemplo

Vamos usar como exemplo a aplicação criada nas últimas aulas, de cadastro de usuário:



# DAO - Exemplo

Os dados dessa tela de cadastro podem ser armazenados em um novo tipo de objeto chamado Pessoa.

```
public class Pessoa {
    private String nome;
    private String time;
    private String sexo;
    private boolean esporte;
    private boolean arte;
    private String endereco;

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getTime() {
        return time;
    }
    public void setTime(String time) {
        this.time = time;
    }
    public String getSexo() {
        return sexo;
    }
    }
```

Essa classe possui um atributo para cada campo correspondente na tela, e o método get e set correspondente

# DAO - Exemplo

Agora criaremos uma interface DAO para a nossa classe Pessoa.

```java
package br.unip.laboratorio;

import java.util.List;

public interface PessoaDAO {

    public void salvar(Pessoa p);

    public void deletar(Pessoa p);

    public List<Pessoa> listar();

    public Pessoa procurar(String nome);

}
```

São definidos métodos para salvar e deletar um objeto do tipo pessoa, um método para listar todas as pessoas cadastradas, e um método para procurar uma pessoa a partir de um nome.

# DAO - Exemplo

Agora criaremos uma classe que implementa a interface PessoaDAO

```
package br.unip.laboratorio;

import java.util.List;

public class PessoaDAOImpl implements PessoaDAO {

    @Override
    public void salvar(Pessoa p) {
    }

    @Override
    public void deletar(Pessoa p) {
    }

    @Override
    public List<Pessoa> listar() {
        return null;
    }

    @Override
    public Pessoa procurar(String nome) {
        return null;
    }

}
```

Ao implementar a interface somos obrigados a implementar todos os métodos definidos na interface. Nessa classe vai o código específico de comunicação com o banco de dados.

As outras classes do sistema utilizaram o DAO sempre que precisarem de alguma informação de pessoa a partir do banco de dados.

# DAO - Exemplo

Adicionando os dados de conexão com o banco na classe PessoaDAOImpl

```
public class PessoaDAOImpl implements PessoaDAO {

    // URL de conexao com o banco
    String url  = "jdbc:mysql://localhost:3306/java";
    // Usuario de acesso ao banco
    String login = "root";
    // Senha de acesso ao banco
    String senha = "password";

    // Construtor padrao da classe
    public PessoaDAOImpl() {
        // Especifica o driver JDBC usado para a conexao
        // Nesse exemplo eh MySQL
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Falha ao carregar o driver JDBC!", "Erro", JOptionPane.ERROR_MESSAGE);
        }
    }
```

# DAO - Exemplo

Criando um método utilitário para obter uma conexão com o banco de dados na classe PessoaDAOImpl

```
/**
 * Metodo responsavel por abrir uma conexao com o banco de dados
 * @return
 * @throws SQLException
 */
public Connection obtemConexao() throws SQLException {
    Connection conn = DriverManager.getConnection(url, login, senha);

    return conn;
}
```

# DAO - Exemplo

Implementando os métodos específicos da classe PessoaDAOImpl

```
/**
 * Metodo responsavel por salvar um objeto do tipo Pessoa na tabela
 * correspondente no banco de dados.
 */
@Override
public void salvar(Pessoa p) {

    try {

    Connection conn = obtemConexao();

    String sql = "INSERT INTO cadastro (nome, time, sexo, pratica_esporte, pratica_arte, endereco) ";
           sql += "VALUES (?, ?, ?, ?, ?, ?) ";

    PreparedStatement s = conn.prepareStatement(sql);

    s.setString(1, p.getNome());
    s.setString(2, p.getTime());
    s.setString(3, p.getSexo());
    s.setBoolean(4, p.isEsporte());
    s.setBoolean(5, p.isArte());
    s.setString(6, p.getEndereco());

    s.execute();

    conn.close();

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Falha ao salvar a pessoa no banco!", "Erro", JOptionPane.ERROR_MESSAGE);
    }

}
```

# DAO - Exemplo

Implementando os métodos específicos da classe PessoaDAOImpl

```
/**
 * Metodo responsavel por deletar uma pessoa do banco de dados. O campo nome
 * da pessoa eh usado na query de delete
 */
@Override
public void deletar(Pessoa p) {

    try {

        Connection conn = obtemConexao();

        String sql = "DELETE FROM cadastro WHERE nome = ?";

        PreparedStatement s = conn.prepareStatement(sql);

        s.setString(1, p.getNome());

        int qtdLinhas = s.executeUpdate();

        if(qtdLinhas == 0) {
            JOptionPane.showMessageDialog(null, "Pessoa nao encontrada no banco de dados", "Info", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(null, "Pessoa deletada com sucesso", "Info", JOptionPane.INFORMATION_MESSAGE);
        }

        conn.close();

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Falha ao salvar a pessoa no banco!", "Erro", JOptionPane.ERROR_MESSAGE);
        }


}
```

# DAO - Exemplo

Implementando os métodos específicos da classe PessoaDAOImpl

```
@Override
public List<Pessoa> listar() {
    try {

        Connection conn = obtemConexao();

        String sql = "SELECT * FROM cadastro";

        Statement s = conn.createStatement();

        ResultSet rs = s.executeQuery(sql);

        Pessoa p;
        ArrayList<Pessoa> listaPessoas = new ArrayList<Pessoa>();

        while (rs.next()) {
            p = new Pessoa();
            p.setNome(rs.getString("nome"));
            p.setTime(rs.getString("time"));
            p.setSexo(rs.getString("sexo"));
            p.setEsporte(rs.getBoolean("pratica_esporte"));
            p.setArte(rs.getBoolean("pratica_arte"));
            p.setEndereco(rs.getString("endereco"));

            listaPessoas.add(p);
        }

        conn.close();

        return listaPessoas;

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Falha ao salvar a pessoa no banco!", "Erro", JOptionPane.ERROR_MESSAGE);
            return null;
        }

}
```

# DAO - Exemplo

Implementando os métodos específicos da classe PessoaDAOImpl

```
@Override
public Pessoa procurar(String nome) {

    Pessoa p = null;

    try {
    Connection conn = obtemConexao();

    String sql = "SELECT * FROM cadastro where nome = ?";

    PreparedStatement s = conn.prepareStatement(sql);

    s.setString(1, nome);

    ResultSet rs = s.executeQuery();

    while (rs.next()) {
        p = new Pessoa();
        p.setNome(rs.getString("nome"));
        p.setTime(rs.getString("time"));
        p.setSexo(rs.getString("sexo"));
        p.setEsporte(rs.getBoolean("pratica_esporte"));
        p.setArte(rs.getBoolean("pratica_arte"));
        p.setEndereco(rs.getString("endereco"));
    }

    if (p == null) {
        JOptionPane.showMessageDialog(null, "Pessoa com nome: " + nome + " não encontrada no banco!" , "Info", JOptionPane.INFORMATION_MESSAGE);
    }

    conn.close();

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Falha ao salvar a pessoa no banco!", "Erro", JOptionPane.ERROR_MESSAGE);
    }

    return p;
}
```

# DAO - Exemplo

Depois de implementado todos os métodos da classe DAO, criaremos uma classe de Fábrica, que possui um método estático para obter o DAO de pessoa.

Isso é feito para que qualquer código que necessite usar o DAO de pessoa precise apenas chamar um método da fábrica para obtê-lo.

Além disso, se resolvermos trocar a nossa implementação da classe que faz acesso ao banco de dados de JDBC para Hibernate por exemplo, basta alterar o código na classe Fabrica, que todos os outros lugares continuam funcionando normalmente.

# DAO - Exemplo

Criando a classe FabricaDAO

```
package br.unip.laboratorio;

public class FabricaDAO {

    public static PessoaDAO obtemPessoaDAO() {
        return new PessoaDAOImpl();
    }
}
```

# DAO - Exemplo

Alterando os métodos da classe janela para utilizarem os métodos da classe DAO para acessar o banco de dados. A classe Janela não vai ter mais nenhuma informação com relação ao banco de dados.

Na classe Janela, alterar o metodo gravar para:

```
public void gravar() {
    Pessoa p = new Pessoa();

    p.setNome(nome.getText());
    p.setTime((String) time.getSelectedItem());
    p.setSexo(chkboxMasc.isSelected()?"Masculino":"Feminino");
    p.setEsporte(esporte.isSelected());
    p.setArte(arte.isSelected());
    p.setEndereco(endereco.getText());

    PessoaDAO pessoaDAO = FabricaDAO.obtemPessoaDAO();

    pessoaDAO.salvar(p);
}
```

# DAO - Exemplo

Na classe Janela, alterar o metodo imprimeRespostas para:

```
public void imprimeRespostas() {

    PessoaDAO pessoaDAO = FabricaDAO.obtemPessoaDAO();

    List<Pessoa> listaPessoas = pessoaDAO.listar();

    JTable table = new JTable(montaModeloTabela(listaPessoas));

    JOptionPane.showMessageDialog(null, new JScrollPane(table), "Listagem", JOptionPane.NO_OPTION);
}
```

# DAO - Exemplo

Na classe Janela, alterar o metodo imprimeRespostas para:

```
public void imprimeRespostas() {

    PessoaDAO pessoaDAO = FabricaDAO.obtemPessoaDAO();

    List<Pessoa> listaPessoas = pessoaDAO.listar();

    JTable table = new JTable(montaModeloTabela(listaPessoas));

    JOptionPane.showMessageDialog(null, new JScrollPane(table), "Listagem", JOptionPane.NO_OPTION);
}
```

# DAO - Exemplo

Na classe Janela, alterar o metodo montaModeloTabela para:

```
private TableModel montaModeloTabela(List<Pessoa> listaPessoas) {
    Vector<String> nomeColunas = new Vector<String>();
    nomeColunas.add("Nome");
    nomeColunas.add("Time");
    nomeColunas.add("Sexo");
    nomeColunas.add("Pratica Esporte");
    nomeColunas.add("Pratica Arte");
    nomeColunas.add("Endereco");

    Vector<Vector<Object>> data = new Vector<Vector<Object>>();
    for(Pessoa p : listaPessoas) {
        Vector<Object> vector = new Vector<Object>();
        vector.add(p.getNome());
        vector.add(p.getTime());
        vector.add(p.getSexo());
        vector.add(p.isEsporte());
        vector.add(p.isArte());
        vector.add(p.getEndereco());

        data.add(vector);
    }

    DefaultTableModel model = new DefaultTableModel(data, nomeColunas) {
          // Além de instanciar uma instancia de DefaultTableModel, para a nossa instância
          // estamos sobrescrevendo o método isCellEditable de forma que nenhuma célula da tabela
          // seja editável. Por padrão todas as células da JTable são editáveis.
          @Override
          public boolean isCellEditable ( int row, int column )
          {
              return false;
          }
    };
    return model;
}
```

# DAO - Exemplo

Adicionando um novo botão para Deletar

```
private Label labelEndereco;
private TextArea endereco;
private Button botaoGravar;
private Button botaoLimpar;
private Button botaoSair;
private Button botaoListar;
private Button botaoDeletar;
```

```
// Instancia os quatro botoes: Gravar, Limpar, Sair e Listar
botaoGravar = new Button("Gravar");
botaoGravar.setName("botaoGravar");
botaoLimpar = new Button("Limpar");
botaoLimpar.setName("botaoLimpar");
botaoSair = new Button("Sair");
botaoSair.setName("botaoSair");
botaoListar = new Button("Listar");
botaoListar.setName("botaoListar");
botaoDeletar = new Button("Deletar");
botaoDeletar.setName("botaoDeletar");
```

# DAO - Exemplo

## Adicionando um novo botão para Deletar

```
// Instancia um objeto da classe TratamentoEventosMouse e adiciona esse objeto
// como listener dos botoes
TratamentoEventosMouse eventosMouse = new TratamentoEventosMouse(this);

botaoGravar.addMouseListener(eventosMouse);
botaoLimpar.addMouseListener(eventosMouse);
botaoSair.addMouseListener(eventosMouse);
botaoListar.addMouseListener(eventosMouse);
botaoDeletar.addMouseListener(eventosMouse);


// Cria um grid layout de 3 linha e 2 colunas para adicionar os botoes
GridLayout gridBotoes = new GridLayout(3, 2);
// Cria um painel para adicionar os botoes. Esse painel usara o grid criado acima
Panel painelBotoes = new Panel();
painelBotoes.setLayout(gridBotoes);

// Adiciona os botoes no painel criado
painelBotoes.add(botaoGravar);
painelBotoes.add(botaoLimpar);
painelBotoes.add(botaoSair);
painelBotoes.add(botaoListar);
painelBotoes.add(botaoDeletar);

// Finalmente adiciona o painel na janela
this.add(painelBotoes);
```

# DAO - Exemplo

Na classe TratamentoEventosMouse, adicionar um if para o botão deletar

```
@Override
public void mouseClicked(MouseEvent e) {
    Button botao = (Button) e.getSource();

    if(botao.getName().equals("botaoGravar")) {
        // Aqui entra o codigo de tratamento do botao gravar
        janela.gravar();
    } else if(botao.getName().equals("botaoSair")) {
        // Aqui entra o codigo de tratamento do botao sair
        System.exit(1);
    } else if(botao.getName().equals("botaoLimpar")) {
        // Aqui entra o codigo de tratamento do botao limpar
        janela.limpar();
    } else if(botao.getName().equals("botaoListar")) {
        // Aqui entra o codigo de tratamento do botao limpar
        janela.imprimeRespostas();
    }  else if(botao.getName().equals("botaoDeletar")) {
        // Aqui entra o codigo de tratamento do botao deletar
        janela.deletar();
    }
}
```

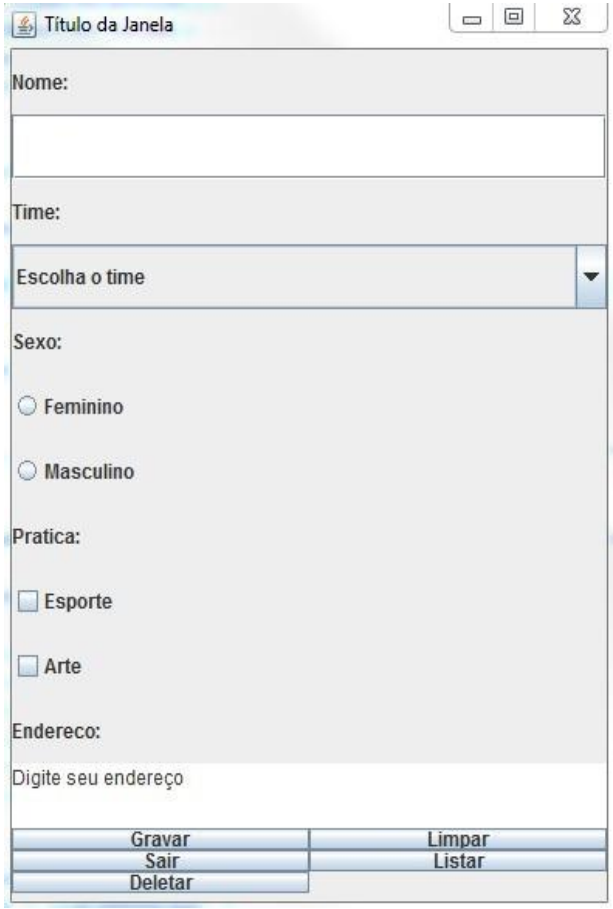
# DAO - Exemplo

Criar o método deletar na classe Janela

```
public void deletar() {
        Pessoa p = new Pessoa();

        p.setNome(nome.getText());

        PessoaDAO pessoaDAO = FabricaDAO.obtemPessoaDAO();

        pessoaDAO.deletar(p);
}
```

# DAO - Exemplo

Resultado Final:



# DAO - Conclusão

Com essas alterações conseguimos separar o código de acesso ao banco de dados na classe PessoaDAOImpl

A classe Janela não possui mais nenhuma informação sobre dados para conexão do banco de dados, ou como o objeto Pessoa está armazenado no banco de dados ( Qual tabela, quais colunas possui, etc).

Se quisermos mudar de JDBC para Hibernate por exemplo, bastaria criar uma nova classe que implemente a nossa interface PessoaDAO e alterar a classe FabricaDAO para usar essa nova classe com a implementação do Hibernate.