

Melhorando o CRUD de Usuários

Prof. Leonardo Vianna do Nascimento
Disciplina de Desenvolvimento de Sistemas I

Problemas no Projeto

- A classe *Usuario* não segue o princípio de separação modelo – apresentação proposto pelo padrão MVC
 - Os métodos *salvar* e *remover* possuem código de apresentação, mas são métodos de uma classe de modelo
- Segundo o MVC, deveríamos poder trocar a apresentação sem precisar alterar o modelo

Problemas no Projeto

Uso de classes da API JSF (dependente da apresentação)

```
public void salvar() {  
    FacesContext context = FacesContext.getCurrentInstance();  
    Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
    Transaction t = sessao.beginTransaction();  
    sessao.saveOrUpdate(this);  
    t.commit();  
    context.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "Usuár  
}
```

Problemas no Projeto

- Classe *Usuario* possui ***Acoplamento Alto***
- Acoplamento
 - Nível de dependência da classe com outras classes externas
- A ideia é tentar manter o ***Acoplamento Baixo***

GRASP

- *General Responsibility Assignment Software Patterns*
- Padrões Gerais de Software para Definição de Responsabilidades
- Definem princípios de projeto com o objetivo de tornar o projeto mais elegante, eficiente e reusável

Acoplamento Baixo

- Princípio de projeto GRASP
- A ideia aqui é manter a classe dependente o mínimo possível de outras classes
- Isso faz com que uma modificação em uma classe impacte o mínimo possível em outras classes
- Aumentamos também o potencial de reuso de uma classe

Dependências de Usuário

- Ver *imports* presentes no arquivo
- A classe *Usuario* depende de:

```
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import org.hibernate.Session;
import org.hibernate.Transaction;
import persistencia.HibernateUtil;
```

Coesão

- Outro problema associado com acoplamento alto é a **Baixa Coesão**
 - Coesão tem relação com as funções desempenhadas por uma classe
 - Quanto mais funções diferentes uma classe desempenhar, menor sua coesão
- A classe *Usuario* desempenha funções de modelo, apresentação e persistência

Coesão Alta

- Outro padrão GRASP que nos orienta a manter a coesão de uma classe alta
 - Uma classe deve ser voltada a desempenhar uma única função
- Por exemplo, a classe *Usuario* é uma classe de modelo para usuários da biblioteca e não deveria desempenhar funções de apresentação e persistência

Resolvendo Problemas

- Devemos:
 - Diminuir o acoplamento da classe *Usuario*
 - Aumentar a coesão da classe *Usuario*
 - Tornar a classe *Usuario* coerente com o princípio de separação modelo-apresentação
- Como fazer isso?

Indireção

- Padrão GRASP que permite diminuir o acoplamento e aumentar a coesão de uma classe criando uma classe intermediária que execute funções que a classe original não deveria realizar
- Utilizaremos indireção para criar uma classe intermediária que funcionará como *bean* JSF e servirá como mediadora entre as páginas da apresentação e a classe *Usuario*

Atividade 1

- Criaremos uma classe chamada *CadastroUsuariosBean*
 - Essa classe funcionará como *bean* JSF com o nome de *usuarioBean*
 - Essa classe será criada em um novo pacote chamado de *beans*
 - Acrescentaremos uma atributo chamado *usuario* do tipo *Usuario* para ligar a classe intermediária ao modelo
 - Incluiremos implementações dos métodos *salvar*, *carregar* e *remover*

Classe CadastroUsuariosBean

```
package beans;

import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;
import modelo.Usuario;

@ManagedBean(name="usuarioBean")
@RequestScoped
public class CadastroUsuariosBean {
    private Usuario usuario = new Usuario();

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }
}
```

Classe CadastroUsuariosBean

```
public void salvar() {
    FacesContext context = FacesContext.getCurrentInstance();
    usuario.salvar();
    context.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Usuário cadastrado com sucesso", ""));
}

public void carregar() {
    usuario.carregar();
}

public void remover() {
    FacesContext context = FacesContext.getCurrentInstance();
    usuario.remover();
    context.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Usuário removido com sucesso", ""));
}
}
```

Classe Usuario

- Os métodos *salvar* e *remover* não fazem mais uso de classes da API JSF

```
public void salvar() {  
    Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
    Transaction t = sessao.beginTransaction();  
    sessao.saveOrUpdate(this);  
    t.commit();  
}
```

```
public void carregar() {  
    Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
    Transaction t = sessao.beginTransaction();  
    sessao.load(this, id);  
    t.commit();  
}
```

```
public void remover() {  
    Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
    Transaction t = sessao.beginTransaction();  
    sessao.delete(this);  
    t.commit();  
}
```

Arquivos XHTML

- Foi necessário alterar as referências para propriedades
 - Por exemplo, trocar *usuarioBean.nome* por *usuarioBean.usuario.nome*
 - OBSERVACAO
 - É possível acessar uma propriedade dentro de outra propriedade
 - A expressão `#{usuarioBean.usuario.nome}` geraria a sequência de chamadas *usuarioBean.getUsuario().getNome()*

Arquivo *inclusaoUsuarios.xhtml*

```
<h:body>
  <h1>Biblioteca - Cadastro de Usuários</h1>
  <h:messages/>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="txtNome" value="Nome: "/>
      <h:inputText id="txtNome" label="Nome" maxlength="100" size="60"
        value="#{usuarioBean.usuario.nome}" required="true"/>
      <h:outputLabel for="txtCPF" value="CPF: "/>
      <h:inputText id="txtCPF" label="CPF" maxlength="11" size="12"
        value="#{usuarioBean.usuario.cpf}" required="true"/>
      <h:outputLabel for="txtTelefone" value="Telefone: "/>
      <h:inputText id="txtTelefone" label="Telefone" maxlength="20" size="25"
        value="#{usuarioBean.usuario.telefone}" required="true"/>
      <h:outputLabel for="txtEmail" value="E-mail: "/>
      <h:inputText id="txtEmail" label="E-mail" maxlength="255" size="60"
        value="#{usuarioBean.usuario.email}" required="true"/>
    </h:panelGrid>
    <h:commandButton value="Salvar" actionListener="#{usuarioBean.salvar()}" />
    <h:button value="Voltar" outcome="index"/>
  </h:form>
</h:body>
```

Arquivo *consultaUsuarios1.xhtml*

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.or
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Biblioteca - Consulta de Usuários</title>
  </h:head>
  <h:body>
    <h1>Biblioteca - Consulta de Usuários</h1>
    <h:messages/>
    <h:form>
      <h:outputLabel for="txtID" value="ID do Usuário: "/>
      <h:inputText id="txtID" value="#{usuarioBean.usuario.id}"/>
      <h:commandButton value="Consultar" actionListener="#{usuarioBean.car
      <h:commandButton value="Remover" actionListener="#{usuarioBean.remov
      <h:button value="Voltar" outcome="index"/>
    </h:form>
  </h:body>
</html>
```

Arquivo *consultaUsuarios2.xhtml*

```
<h:body>
  <h1>Biblioteca - Consulta de Usuários</h1>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="txtNome" value="Nome: "/>
      <h:outputText id="txtNome" value="#{usuarioBean.usuario.nome}"/>
      <h:outputLabel for="txtCPF" value="CPF: "/>
      <h:outputText id="txtCPF" value="#{usuarioBean.usuario.cpf}"/>
      <h:outputLabel for="txtTelefone" value="Telefone: "/>
      <h:outputText id="txtTelefone" value="#{usuarioBean.usuario.telefone}"/>
      <h:outputLabel for="txtEmail" value="E-mail: "/>
      <h:outputText id="txtEmail" value="#{usuarioBean.usuario.email}"/>
    </h:panelGrid>
    <h:button value="Voltar" outcome="consultaUsuarios1"/>
  </h:form>
</h:body>
```

Padrão DAO

- *Data Access Object*
- Permite implementar indireção para o serviço de persistência
 - Iremos retirar as ações de persistência da classe *Usuario* e transferi-las para uma classe intermediária chamada *UsuarioDAO*

Classe Usuario

- Removemos os métodos *salvar*, *carregar* e *remover*
 - A classe Usuario não terá mais estas responsabilidades (aumentamos a coesão)
 - A classe *UsuarioDAO* terá a única responsabilidade de lidar com a persistência de objetos da classe Usuario

Classe *UsuarioDAO*

```
public class UsuarioDAO {  
    public void salvar(Usuario u) {  
        Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        Transaction t = sessao.beginTransaction();  
        sessao.saveOrUpdate(u);  
        t.commit();  
    }  
  
    public Usuario carregar(int id) {  
        Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        Transaction t = sessao.beginTransaction();  
        Usuario ret = (Usuario) sessao.load(Usuario.class, id);  
        t.commit();  
        return ret;  
    }  
  
    public void remover(Usuario u) {  
        Session sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        Transaction t = sessao.beginTransaction();  
        sessao.delete(u);  
        t.commit();  
    }  
}
```

Novo Método *load*

- No método *carregar* utilizamos uma variante do método *load* que recebe a classe do objeto que se deseja obter e o id do mesmo
- O método retorna o objeto
 - A referência retornada é do tipo *Object*
 - Por isso foi necessária uma conversão de tipo

Melhorias na Classe DAO

- Os métodos *salvar*, *carregar* e *remover* são quase idênticos
 - Em todos obtemos a sessão do Hibernate, iniciamos uma transação e encerramos realizando um *commit* na transação
 - Isso é código duplicado!


DRY

- *Don't Repeat Yourself* (Não Repita a Si Mesmo)
 - Princípio de projeto contra duplicação de código
- A duplicação de código é o problema mais comum na lista conhecida como *bad smells* (mau cheiros de projeto)
 - Lista de más práticas que acontecem em desenvolvimento de software
- Código duplicado pode facilitar o desenvolvimento mas complica a manutenção

Como Resolver?

- Vamos incluir a obtenção da sessão e o início da transação no construtor da classe

```
public class UsuarioDAO {  
    private Session sessao;  
  
    public UsuarioDAO() {  
        sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        sessao.beginTransaction();  
    }  
}
```



Referência à sessão
armazenada como atributo

- Criamos um método *encerrar* contendo a finalização da transação

```
public void encerrar() {  
    sessao.getTransaction().commit();  
}
```

Classe CadastroUsuariosBean

- Devemos modificar os métodos da classe para utilizarem os métodos *salvar*, *carregar* e *remover* da classe *UsuarioDAO*
- Um objeto *UsuarioDAO* será criado ao se criar o objeto
 - Sua referência será armazenada em um atributo
- Adicionaremos também um método *encerrar* com a anotação *@PreDestroy*
 - Será executado automaticamente antes do *bean* ser destruído

Classe CadastroUsuariosBean

```
public class CadastroUsuariosBean {  
    private Usuario usuario = new Usuario();  
    private final UsuarioDAO dao = new UsuarioDAO();  
  
    public Usuario getUsuario() { ...3 linhas }  
  
    public void setUsuario(Usuario usuario) { ...3 linhas }  
  
    public void salvar() {  
        FacesContext context = FacesContext.getCurrentInstance();  
        dao.salvar(usuario);  
        context.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO,  
            "Usuário cadastrado com sucesso", ""));  
    }  
  
    public void carregar() {  
        usuario = dao.carregar(usuario.getId());  
    }  
  
    public void remover() {  
        FacesContext context = FacesContext.getCurrentInstance();  
        dao.remover(usuario);  
    }  
}
```


Atributo dao

Utilizamos o atributo dao para persistir usuários

Classe CadastroUsuariosBean

```
@PreDestroy  
public void encerrar() {  
    dao.encerrar();  
}
```

Anotação que indica que o método deve ser executado antes da destruição do *bean* (classe `javax.annotation.PreDestroy`)



Encerramos a transação ao finalizar o bean

