

Introdução à Programação Orientada a Objetos com Java

Estruturação de Sistemas em Camadas

Paulo Borba
Centro de Informática
Universidade Federal de Pernambuco

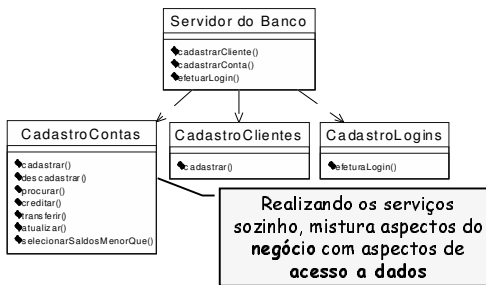
Depois de aprender a implementar aplicações...

Precisamos aprender a
estruturar melhor
estas aplicações

O objetivo desta aula é mostrar
como as aplicações devem ser
estruturadas em camadas independentes



Até agora temos a mistura de aspectos diferentes...



Aspectos de acesso a dados

```
class CadastroContas {  
    private Conta[] contas;  
    private int proxima;  
    private int maximo;  
  
    CadastroContas(int tamanho) {  
        contas = new Conta[tamanho];  
        proxima = 0;  
        maximo = tamanho;  
    }  
  
    CadastroContas () {this (100); }  
}
```

Aspectos relacionados com a forma de armazenamento e acesso a dados.

Seriam diferentes caso a forma de armazenamento fosse diferente

Os aspectos de negócio...

São inerentes à aplicação, independem da forma de armazenamento, comunicação, interface com o usuário, etc.

```
void cadastrar(Conta conta) {  
    if (conta != null &&  
        conta.valida() &&  
        !this.existe(conta.getNumero())) {  
        if (proxima <= maximo-1) {  
            contas[proxima] = conta;  
            proxima = proxima + 1;  
        } else {...Espaço insuficiente...}  
    } else {...Conta inválida...}  
}
```

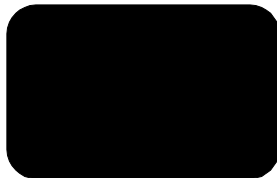
Código de acesso a dados

Os aspectos de interface com o usuário

```
void cadastrar(Conta conta) {  
    if (conta != null &&  
        conta.valida() &&  
        !this.existe(conta.getNumero())) {  
        if (proxima <= maximo-1) {  
            contas[proxima] = conta;  
            proxima = proxima + 1;  
        } else {...Espaço insuficiente...}  
    } else {...Conta inválida...}  
}
```

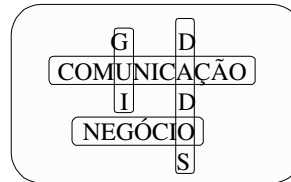
Código de interface com o usuário, depende da tecnologia utilizada: HTML, Swing, Console, etc.

Tudo bem se o código fosse visto como uma caixa preta...



Funciona mas não sabemos como, nem o que tem dentro dela

Mas normalmente precisamos analisar e modificar o código...



Não queremos encontrar um jogo de palavras cruzadas!

É melhor encontrar um bolo, com várias camadas!



Arquitetura em camadas

Os vários tipos de código devem ser escritos separadamente, em classes e camadas diferentes

- Interface com o Usuário
 - código para a apresentação da aplicação (entrada e saída de dados)

Mais arquitetura em camadas

- Comunicação
 - código para permitir acesso remoto aos serviços da aplicação
- Negócio
 - código **inerente** à aplicação
- Dados
 - código para acesso e manipulação de dados

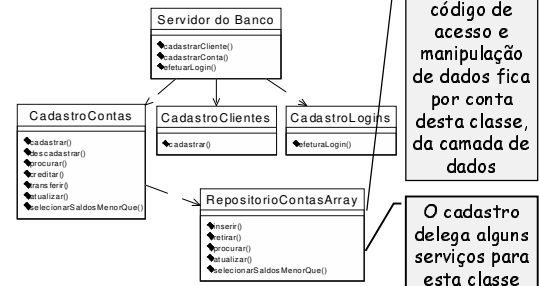
Benefícios da arquitetura em camadas

- Modularidade e seus benefícios:
 - Dividir para conquistar
 - Separação de conceitos
 - Reusabilidade e extensibilidade
- Mudanças em uma camada não afetam as outras, desde que as interfaces sejam preservadas
 - funcionalidade *plug-and-play*

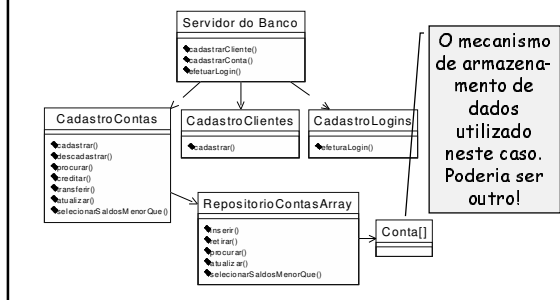
Mais benefícios da arquitetura em camadas

- Uma mesma versão de uma camada trabalhando com diferentes versões de outra camada:
 - várias GUIs para a mesma aplicação
 - vários mecanismos de persistência suportados pela mesma aplicação
 - várias plataformas de distribuição para acesso a uma mesma aplicação

Reverendo CadastroContas e seus serviços



RepositorioContasArray realiza seus serviços...



Implementando CadastroContas

```
class CadastroContas {
    private RepositorioContasArray contas;

    CadastroContas() {
        contas = new RepositorioContasArray();
    }
}
```

Se encarrega de todo o trabalho (serviços) para armazenar e acessar as contas do cadastro

Implementando o método cadastrar

Delega os serviços de acesso a dados

Implementa o código de negócio

```
void cadastrar(Conta conta) {
    if (conta != null &&
        conta.valida() &&
        !this.existe(conta.getNumero())) {
        contas.inserir(c);
    } else {...Conta inválida...}
}
```

Depois nos livraremos do código de interface com o usuário...

Implementando os métodos existe e procurar

```
boolean existe(String n) {
    return contas.existe(n);
}

Conta procurar(String n) {
    return contas.procurar(n);
}
```

Simplesmente delegam os serviços de acesso a dados, já que não possuem aspectos de negócio

Implementando o método descadastrar

```
void descadastrar(String n) {
    Conta c = conts.procurar(numero);
    if (c != null) {
        if (c.getSaldo() == 0) conts.remove(n);
        else {...Conta ainda ativa...}
    } else {...Conta inexistente...}
}
```

Delega os serviços de acesso a dados

Implementa o código de negócio

Implementando o método creditar

```
void creditar(String numero, double valor) {
    Conta c = conts.procurar(numero);
    if (c != null) c.creditar(valor);
    else {...Conta inexistente...}
}
```

Delega os serviços de acesso a dados

Implementa o código de negócio

debitar seria bem parecido

Implementando RepositorioContasArray

```
class RepositorioContasArray {
    private Conta[] conts;
    private int proxima;
    private int maximo;
    RepositorioContasArray(int tamanho) {
        conts = new Conta[tamanho];
        proxima = 0; maximo = tamanho;
    }
    RepositorioContasArray() {this (100);}
}
```

Uma classe só com aspectos de acesso a dados, para servir à classe com aspectos de negócio

Implementando o método inserir

```
void inserir(Conta conta) {
    if (conta != null) {
        if (proxima <= maximo-1) {
            conts[proxima] = conta;
            proxima = proxima + 1;
        } else {...Espaço insuficiente...}
    } else {...Conta nula...}
}
```

É importante para não perder espaço armazenando null

Só código de acesso a dados

Depois nos livraremos do código de interface com o usuário...

Implementando os métodos existe e procurar

```
boolean existe(String n) {
    int indice = this.procurarIndice(n);
    return (indice != proxima);
}
Conta procurar(String n) {
    int indice = this.procurarIndice(n);
    Conta resultado;
    if (indice != proxima)
        resultado = conts[indice];
    else resultado = null;
    return resultado;
}
```

Só código de acesso a dados

Implementando o método remove

```
void remove(String n) {
    int indice = this.procurarIndice(n);
    if (indice != proxima) {
        conts[indice] = conts[proxima-1];
        conts[proxima-1] = null;
        proxima = proxima - 1;
    } else {...Conta inexistente...}
}
```

Só código de acesso a dados, invocado pelo método descadastrar

Implementando o método procurarIndice

```
private int procurarIndice(String n) {
    int i = 0;
    if (n != null) {
        boolean achou = false;
        while ((!achou) && (i < proxima)) {
            if (n.equals(contas[i].getNumero()))
                achou = true;
            else i = i + 1;
        }
    } else i = proxima;
    return i;
}
```

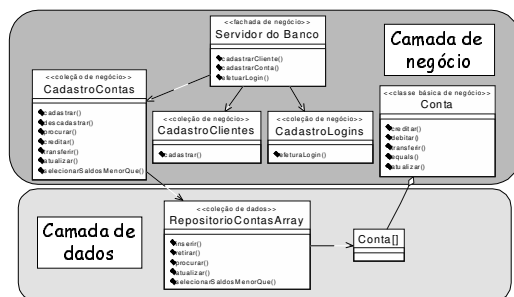
É importante para garantir a corretude do serviço

Só código de acesso a dados

Código de negócio versus código de dados

- Operações de acesso a dados são implementadas na camada de dados, mas são utilizadas na camada de negócio também
- Operações auxiliares de acesso a dados, como `procurarIndice`, só pertencem à camada de dados
- Operações de negócio, como `creditar`, não aparecem na camada de dados

Camadas de negócio e dados



Classes das camadas de negócio e de dados

- Camada de negócio
 - classes básicas (Conta, Endereço)
 - coleções de negócio (CadastroContas)
 - fachadas (ServidorBanco, SistemaX)
- Camada de dados
 - coleções de dados (RepositorioContasArray, ConjuntoClientesBDR)

Resumindo...

- Tipos de código (negócio, dados, interface com o usuário, comunicação)
- Organização do sistema em camadas independentes
- Classes básicas, coleções de dados, coleções de negócio, fachada

