Orientação a Objetos e Java

Sérgio Soares
sergio@dsc.upe.br

Recursão

- Capacidade de um procedimento, método ou função ser definido em termos de (ou "chamar" a) si próprio
- Muitos algoritmos são inerentemente recursivos e só com dificuldade podem ser programados de forma iterativa

Em Java...

- Métodos podem ser recursivos
- Recursão é útil para implementar definições indutivas, em geral
  - Exemplo: fatorial, fibonacci, ordenação, etc.
- É útil para manipular estruturas de dados recursivas
  - Exemplo: listas ligadas, pilhas, árvores, etc.

Exemplo: Série de Fibonacci

```
public class FibonacciRecursivo {

    public static int fib(int m) {
        if (m == 0 || m == 1) {
            return 1;
        } else {
            return fib(m - 1) + fib(m - 2);
        }
    }

    public static void main(String[] args) {
        int n,f;
        System.out.println(“Entre com um número”);
        n = Util.readInt();
        f = fib(n);
        System.out.println(”fib(“ + n + ”) = “ + f);
    }
}
```

Exercícios

- Faça um programa recursivo que computa o produto de dois números inteiros arbitrários usando o operador de soma
- Faça um programa recursivo que computa o fatorial de um número n arbitrário

Iteração e recursão

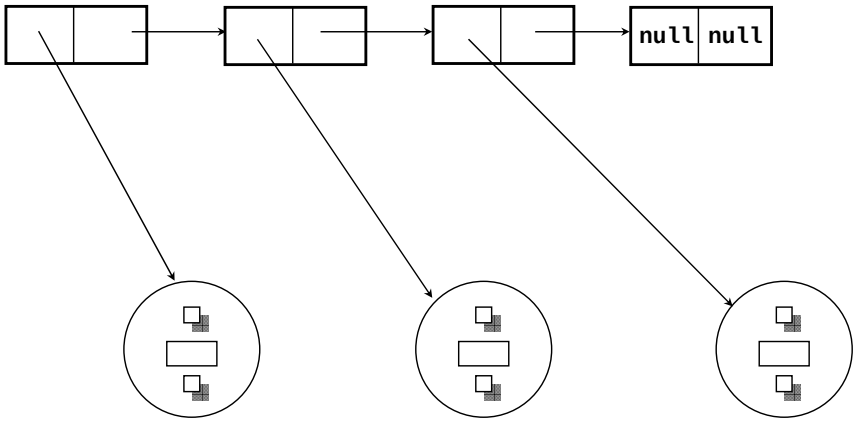
- Iteração é um caso particular de recursão. O comando

```
while(b) { p();}
```

pode ser implementado por um método recursivo `m_rec` da seguinte forma:

```
public static void m_rec() {
    if(b) {
        p();
        m_rec();
    }
}
```

Lista Encadeada de Contas



Listas de Contas: Assinatura

```
public class ListaContas {
  public void inserir(Conta conta) {}
  public void remover(Conta c) {}
  public Conta procurar(String numero) {}
  ...
}
```

Listas de Contas: Descrição

```
public class ListaContas {
  private Conta conta;
  private ListaContas prox;

  public void inserir (Conta conta) {
    if (this.conta == null) {
      this.conta = conta;
      this.prox = new ListaContas();
    } else {
      this.prox.inserir(conta);
    }
  }
```

```
public void remover(Conta c) {
  if (this.conta != null) {
    if (this.conta.equals(c)) {
      this.conta = this.prox.conta;
      this.prox = this.prox.prox;
    } else {
      this.prox.remover(c);
    }
  }
}
```

```
public Conta procurar (String numero) {
  Conta result = null;
  if (this.conta != null) {
    if (this.conta.getNumero().equals(numero)) {
      result = this.conta;
    } else {
      result = this.prox.procurar(numero);
    }
  } else {
    result = null;
  }
  return result;
}
```

CadastroContas: Descrição Modular

```
public class CadastroContas {
  private ListaContas contas;

  public CadastroContas() {
    contas = new ListaContas();
  }
  public void cadastrar(Conta c) {
    contas.inserir(c);
  }
```

CadastroContas: Descrição Modular

```
public void debitar(String num, double val) {
  Conta c;
  c = contas.procurar(num);
  if (c != null) {
    c.debito(val);
  } else {
    System.out.println("Conta "+
                       "inexistente!");
  }
}
```

Lembrem-se que as mensagens com o usuário devem ser dadas nas classes que fazem parte da camada de interface com o usuário, não como acima.

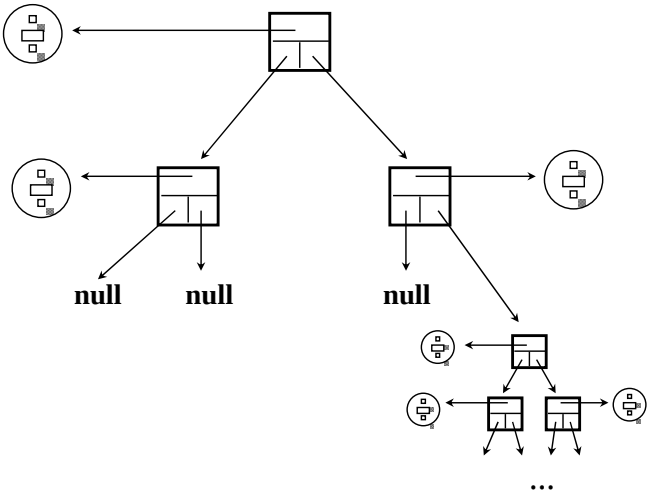
Exercício 1

- Completar a implementação da classe **`CadastroContas`** com os métodos **`transferir`** e **`getSaldo`**.

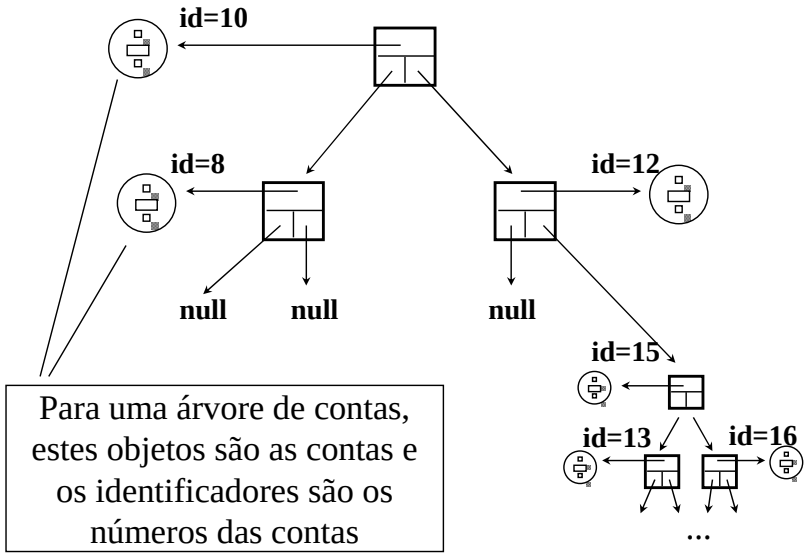
Exercício 2

- Desenvolva um sistema simples para controle de estoque, contendo pelo menos as classes Produto e Estoque, e as seguintes operações: alterar as propriedades dos produtos (nome, preço, quantidade em estoque), retirar um produto do estoque, e verificar que produtos precisam ser repostos.

Árvore Binária



Árvore de Busca Binária



Para uma árvore de contas, estes objetos são as contas e os identificadores são os números das contas

Árvore de Contas: Assinatura

```
public class ArvoreContas {
  public void incluir(Conta c) {
  public void remover(Conta c) {}
  public Conta procurar(String num) {}
}
```

Idêntica a de lista de contas!

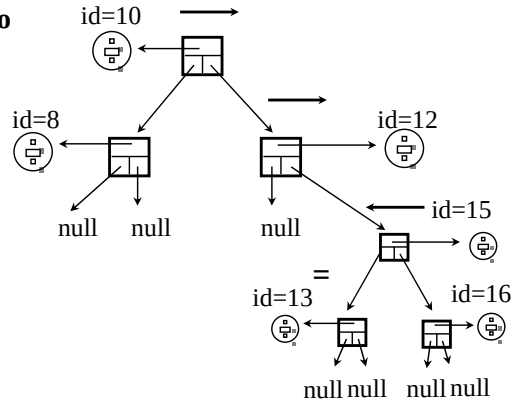
Árvore de contas: Descrição

```
public class ArvoreContas {
  private Conta conta;
  private ArvoreContas esquerda;
  private ArvoreContas direita;
  ...
}
```

Consultando uma conta

- **Compara um identificador com o identificador do objeto conta.**
  - **Se os valores forem iguais, o objeto foi encontrado**
  - **Se o valor do parâmetro for maior, a busca é feita na árvore direita**
  - **Se o valor do parâmetro for menor, a busca é feita na árvore esquerda**

- **Exemplo: consultar por um objeto com identificador = 13**



```
public Conta procurar(String identificador) {
  Conta conta = new Conta(identificador);
  Conta result = null;
  if (this.conta == null) {
    result = null;
  } else if (conta.equals(this.conta)) {
    result = this.conta;
  } else if (conta.ehMaiorQ(this.conta)) {
    if (this.direita != null) {
      result = this.direita.consultar(identificador);
    }
  } else if (this.esquerda != null) {
    result = this.esquerda.consultar(identificador);
  }
  return result;
}
```

```
public void incluir(Conta conta) {
  if (conta != null) {
    if (this.conta == null || this.conta.equals(conta)) {
      this.conta = conta;
    } else  if (conta.ehMaiorQ(this.conta)) {
      if (this.direita == null) {
        this.direita = new ArvoreContas(conta);
      } else {
         this.direita.incluir(conta);
      }
    } else {
      if (this.esquerda == null) {
        this.esquerda = new ArvoreDeContas(conta);
      } else {
        this.esquerda.incluir(conta);
      }
    }
  }
}
```

Orientação a Objetos e Java

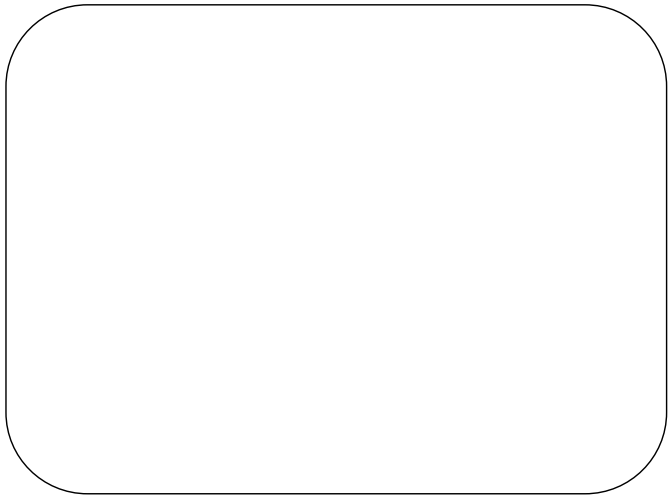
Sérgio Soares
sergio@dei.unicap.br
http://www.dei.unicap.br/~sergio

Estruturação do Código em Camadas

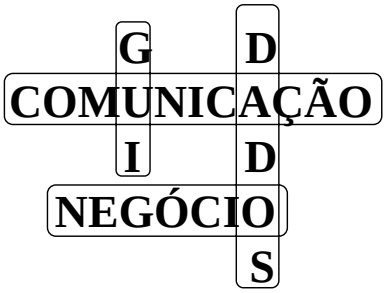
Objetivo

Estruturar o código em camadas, de forma a obter melhor reuso e extensibilidade.

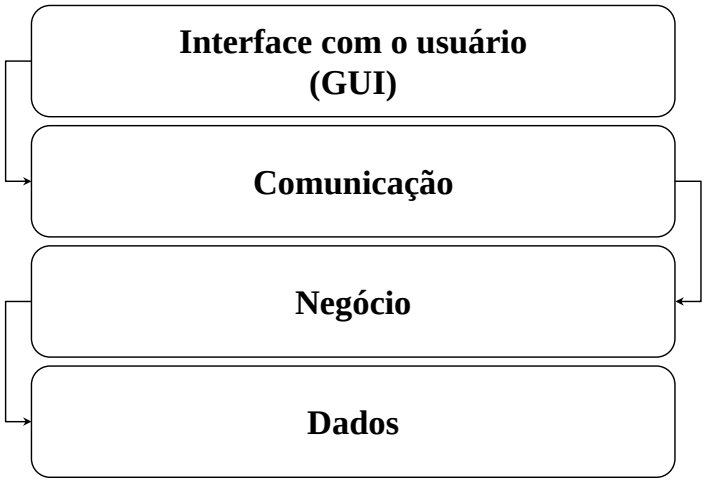
Vendo o código como uma caixa preta...



Vendo o código como palavras cruzadas...



Vendo o código como um bolo... com várias camadas!



Arquitetura em Camadas

- Interface com o Usuário
  - código para a apresentação da aplicação
- Comunicação
  - código de acesso remoto a aplicação
- Negócio
  - código inerente à aplicação sendo desenvolvida
- Dados
  - código para acesso e manipulação de dados

Benefícios da Arquitetura em Camadas

- Modularidade e seus benefícios:
  - dividir para conquistar
  - separação de preocupações (*separation of concerns*)
  - reusabilidade
  - extensibilidade
- Mudanças em uma camada não afetam as outras
  - *plug-and-play*

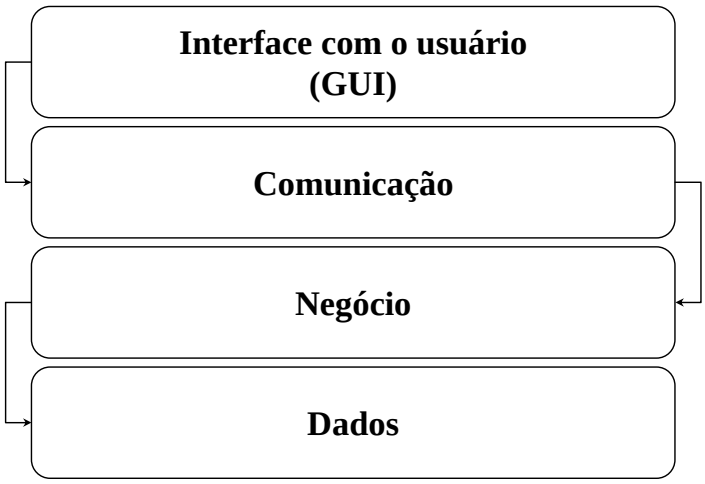
Benefícios da Arquitetura em Camadas

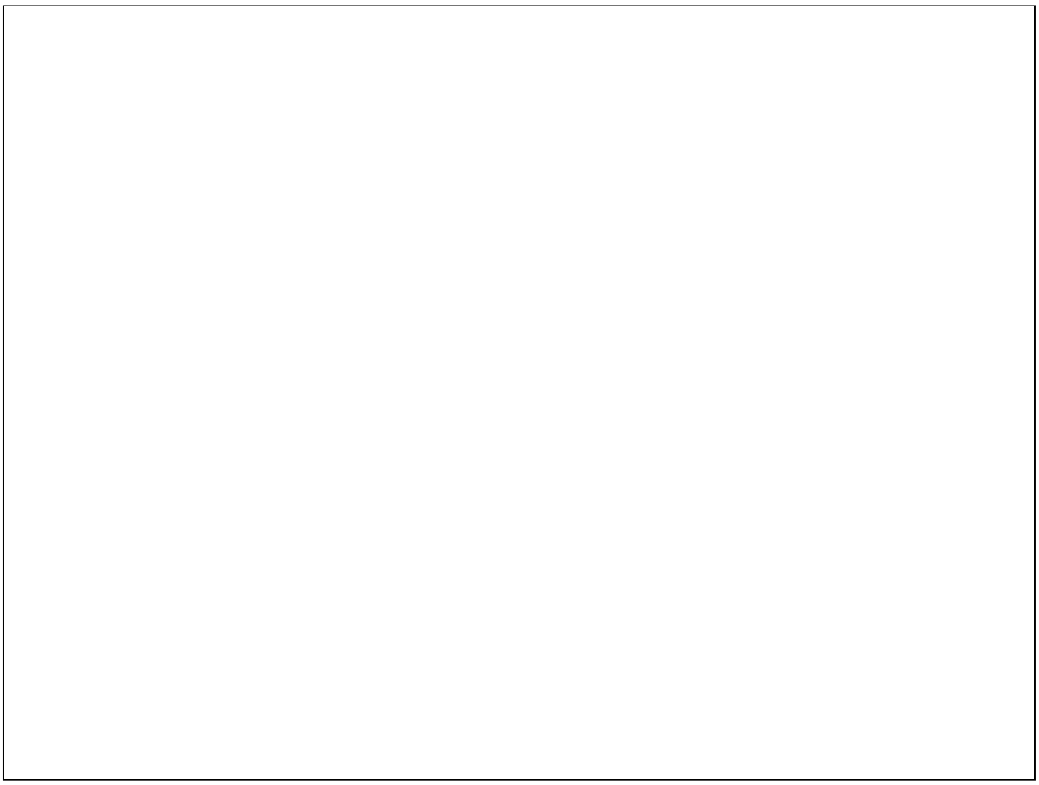
- Uma mesma versão de uma camada trabalhando com diferentes versões de outra camada:
  - – várias GUIs para a mesma aplicação
  - – vários mecanismos de persistência suportados pela mesma aplicação
  - – várias plataformas de distribuição para acesso a uma mesma aplicação

Projeto

- Como a arquitetura em camadas será usada nos projetos?
  - cada integrante desenvolverá uma parte do software de modo a implementar módulos de todas as camadas, com exceção da camada de comunicação
- As equipes já estão formadas?
  - já escolheram o sistema a ser implementado?
- Atenção para a forma de entrega dos projetos

Vendo o código como um bolo... com várias camadas!





Classes Básicas de Negócio

```
public class Conta {
  private double saldo;
  private String numero;
  private Cliente correntista;
  ...
  public void creditar(double valor) {
    saldo = saldo + valor;
  }
}
```

Cliente, Livro, Animal, Veiculo

Classes Coleção de Dados - assinatura

```
public class RepositorioContasArray {
  public void inserir(Conta conta) {}
  public void atualizar(Conta conta){}
  public void remover(String numero){}
  public Conta procurar(String numero) {}
  public boolean existe(String numero) {}
  public RepositorioContasArray procurar(Conta c) {}
  public Conta[] getIterator() {}
}
```

RepositorioContasArquivo, RepositorioContasLista
RepositorioContasBDR, RepositorioContasBDOO

Classes Coleção de Dados

```
public class RepositorioContasArray {
  private Conta[] contas;
  private int indice;
  public RepositorioContasArray(int tam) {
    contas = new Conta[tam]; ...
  }
  public void inserir(Conta conta) {
    contas[indice] = conta;
    indice = indice + 1;
  } ...
}
```

Classes Coleção de Negócio

```
public class CadastroContas {
  private RepositorioContasArray contas;
  public CadastroContas(RepositorioContasArray rep) {
    contas = rep;
  }
  public void cadastrar(Conta conta) {
    if (!contas.existe(conta.getNumero())) {
      contas.inserir(conta);
    } else ...
  } ...
}
```

CadastroClientes, CadastroLivros, CadastroAnimais, CadastroVeiculos

Classe Fachada

```
public class Banco {
  private CadastroContas contas;
  private CadastroClientes clientes;
  ...
  public void cadastrar(Conta conta) {
    Cliente c = conta.getCorrentista();
    if (clientes.existe(c.getCodigo()) {
      contas.cadastrar(conta);
    } else ...
  }
}
```

Livraria, Zoo, Locadora

Método `match`

```
public class Conta {
  ...
  public boolean match(Conta conta) {
    Correntista corr = conta.getCorrentista();
    boolean resp = false;
    if (numero != null && correntista != null)
      resp = numero.equals(conta.getNumero()) &&
             correntista.match(corr);
    else if (numero != null)
      resp = numero.equals(conta.getNumero());
    else  if (correntista != null)
      resp = correntista.match(corr);
    return resp;
  }
```