Arquitetura em Camadas: Uma abordagem para estruturação de aplicações

Benefícios

- Modularidade:
  - Dividir para conquistar
  - Separação de conceitos
    - Coesão
    - Melhor estruturação do sistema
    - Melhor entendimento
  - Reusabilidade
  - Extensibilidade
- Facilidade de manutenção:
  - Custos de manutenção representam em média 70% do custo total do software
  - Mudanças em uma camada não afetam as outras, desde que as interfaces sejam preservadas (plug and play)

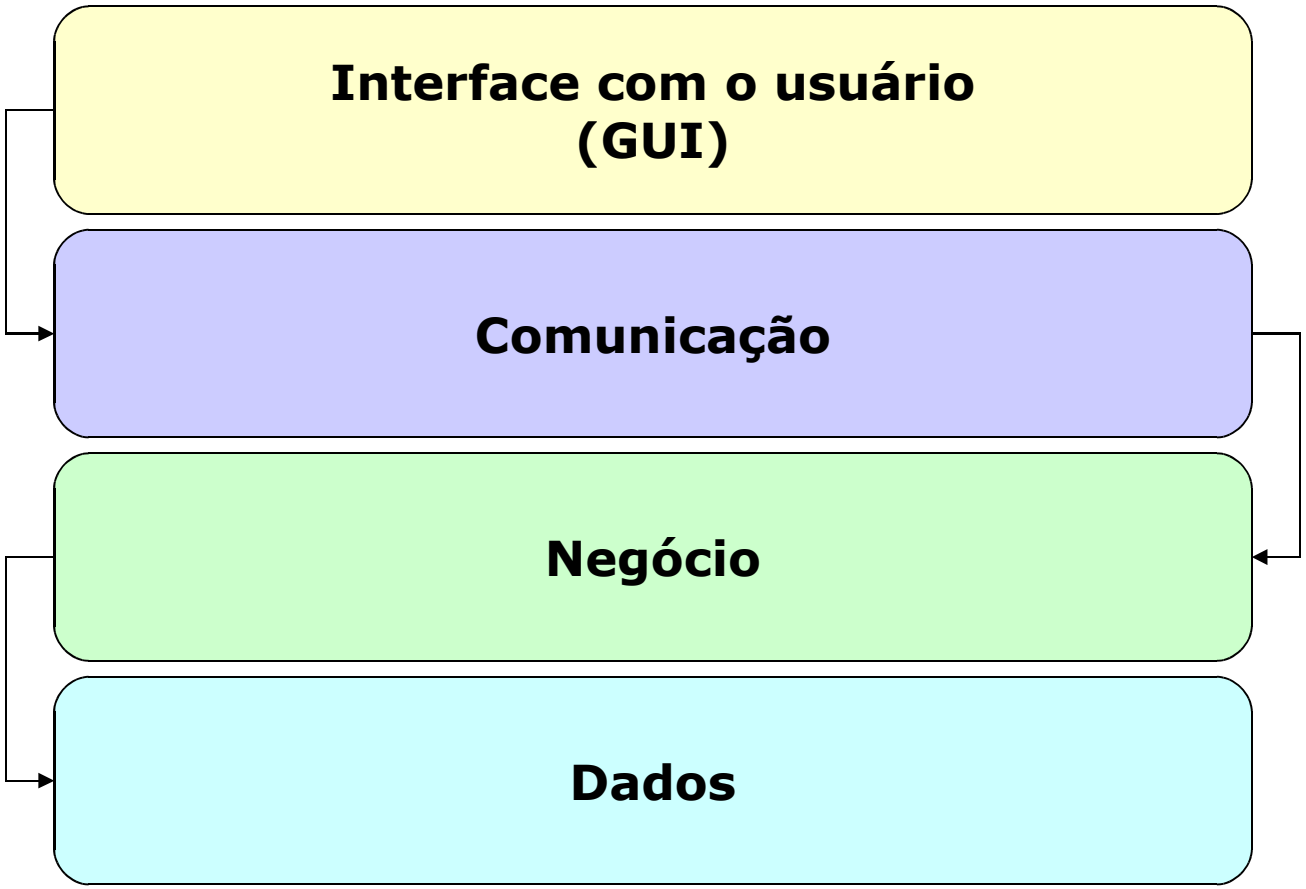
Benefícios

- 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

Características

- Propósito geral de cada camada:
  - Fornecer suporte para alguma camada superior
  - Abstrair as camadas superiores de detalhes específicos
- Propósito específico de cada camada:
  - Preocupar-se com os detalhes específicos que serão ‘escondidos’ das camadas superiores

Estrutura de Camadas



Camada de negócios

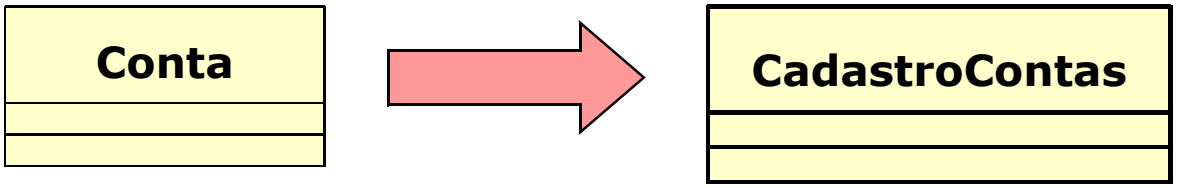
- Responsável por implementar a lógica do negócio
- Classes do domínio da aplicação
  - classes básicas do negócio
  - coleções de negócio
  - fachada do sistema

Classes básicas do negócio

- Representam conceitos básicos do domínio da aplicação

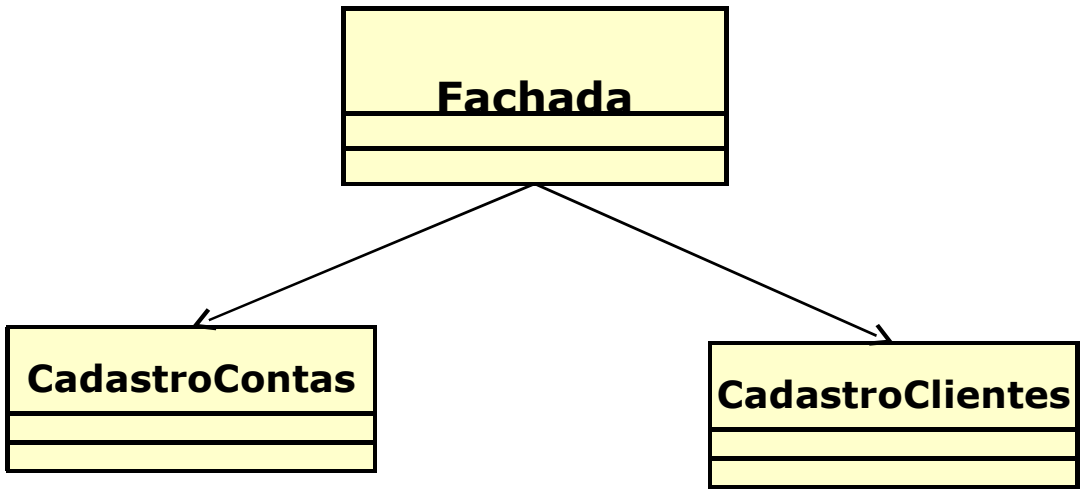
Coleções de negócio

- Representam conjuntos de objetos
- Responsáveis pela inclusão, remoção, atualização e consultas a instâncias das classes básicas
- Encapsulam as verificações e validações relativas ao negócio
- Utilizam coleções de dados como suporte para o armazenamento de objetos



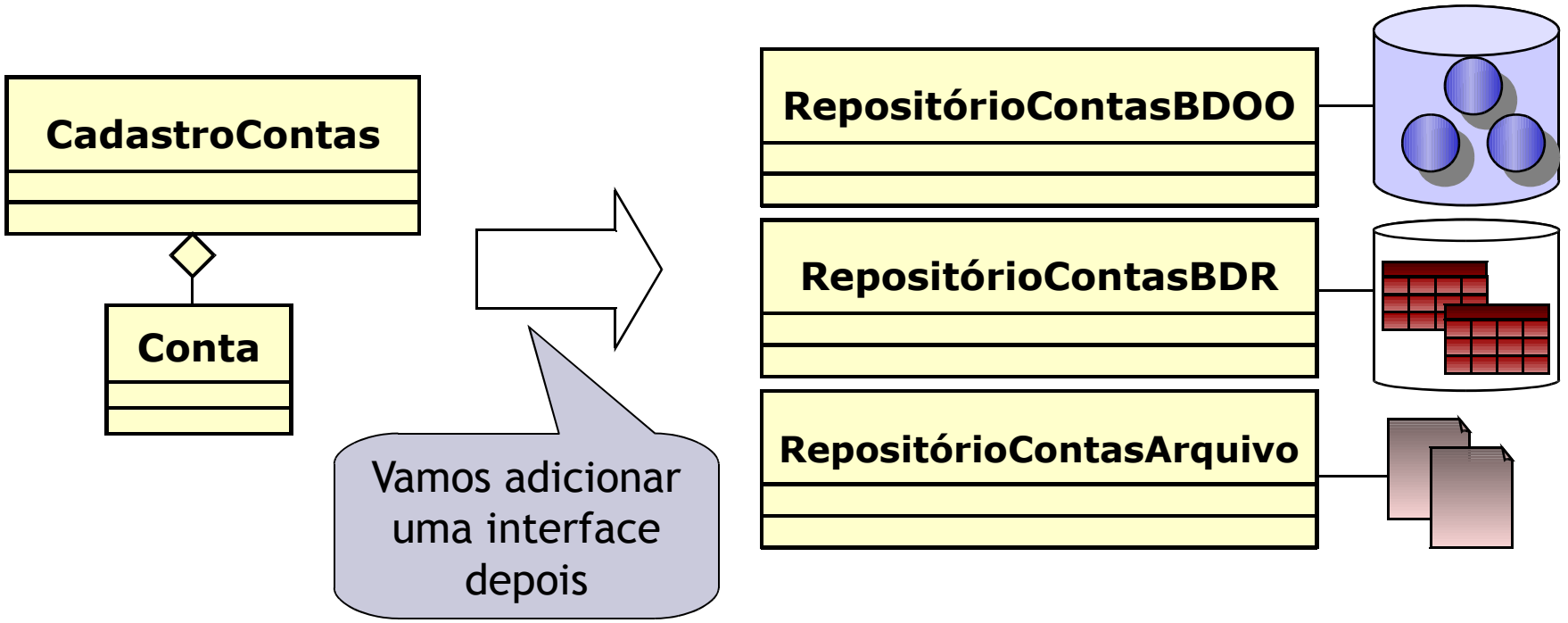
Fachada do sistema

- Segue o padrão de projeto *Facade*
- Representa os **serviços** oferecidos pelo sistema
- Centraliza as instâncias das coleções de negócio
  - Realiza críticas de restrição de integridade
- Gerencia as **transações** do sistema

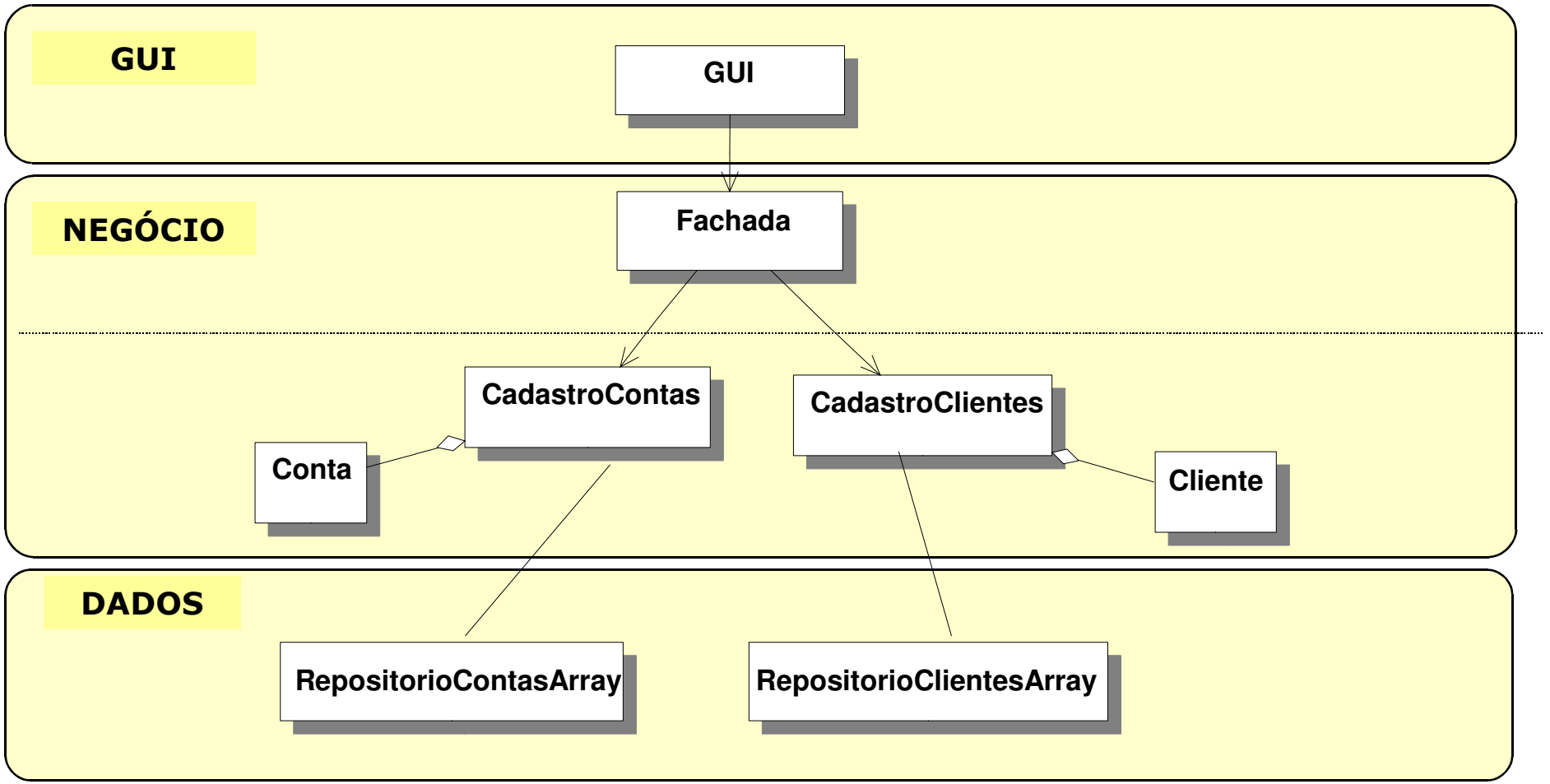


Camada de dados

- Responsável pela manipulação da estrutura de armazenamento dos dados
- Isola o resto do sistema do mecanismo de persistência utilizado
- Classes de coleções de dados
- Executam inclusões, remoções, atualizações e consultas no meio de armazenamento usado



Visão geral da arquitetura



Exemplos de código (sem interface)

Coleção de Negócio

```
public class CadastroContas {
  private RepositorioContasArray contas;
  public CadastroContas(RepositorioContasArray r) {
    this.contas = r;
  }
  public void atualizar(Conta c) {
    contas.atualizar(c);
  }
  public void cadastrar(Conta c){
    if (!contas.existe(c.getNumero())) {
      contas.inserir(c);
    } else {
       System.out.println("Conta já cadastrada");
    }
  }
```

Coleção de Negócio

```
public void creditar(String n, double v) {
  Conta c = contas.procurar(n);
  c.creditar(v);
}
public void debitar(String n, double v) {
   Conta c = contas.procurar(n);
   c.debitar(v);
}
public void descadastrar(String n) {
   contas.remover(n);
}
```

Coleção de Negócio

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

  public void transferir(String origem,
                         String destino,
                         double val) {

      Conta o = contas.procurar(origem);
      Conta d = contas.procurar(destino);
      o.transferir(d, val);
  }
}
```

Fachada

```
public class Fachada {
  private static Fachada instancia;
  private CadastroContas contas;
  private CadastroClientes clientes;

 public static Fachada obterInstancia() {
    if (instancia == null) {
         instancia = new Fachada();
    }
    return instancia;
 }
  private Fachada() {
    initCadastros();
  }
```

Fachada

```
private void initCadastros() {
    RepositorioContasArray rep =
                            new RepositorioContasArray();
    contas = new CadastroContas(rep);

    RepositorioClientesArray repClientes =
                            new RepositorioClientesArray();
    clientes = new CadastroClientes(repClientes);
}
```

Fachada

```
//metodos para manipular clientes
 public void atualizar (Cliente c) {
   clientes.atualizar(c);
 }
 public Cliente procurarCliente(String cpf) {
   return clientes.procurar(cpf);
 }
 public void cadastrar(Cliente c) {
   clientes.cadastrar(c);
 }
 public void descadastrarCliente(String cpf) {
   clientes.remover(cpf);
 }
```

Fachada

```
//metodos para manipular contas
 public void atualizar (Conta c) {
   contas.atualizar(c);
 }
 public Conta procurarConta(String n) {
   return contas.procurar(n);
 }
 public void cadastrar(Conta c) {
   Cliente cli = c.getCliente();
   if (cli != null) {
     clientes.procurar(cli.getCpf());
     contas.cadastrar(c);
   } else {
      System.out.println("cliente nulo");
   }
 }
```

Fachada

```
  public void removerConta(String n) {
    contas.remover(n);
  }
  public void creditar(String n, double v) {
    contas.creditar(n, v);
  }
  public void debitar(String n, double v) {
    contas.debitar(n, v);
  }
  public void transferir(String origem,
      String destino, double val) {
     contas.transferir(origem, destino, val);
  }
}
```