



Padrão Arquitetura em Camadas

Universidade Católica de Pernambuco
Ciência da Computação

Prof. Márcio Bueno
pooite@marciobueno.com

Fonte: Material da Prof^a Karina Oliveira

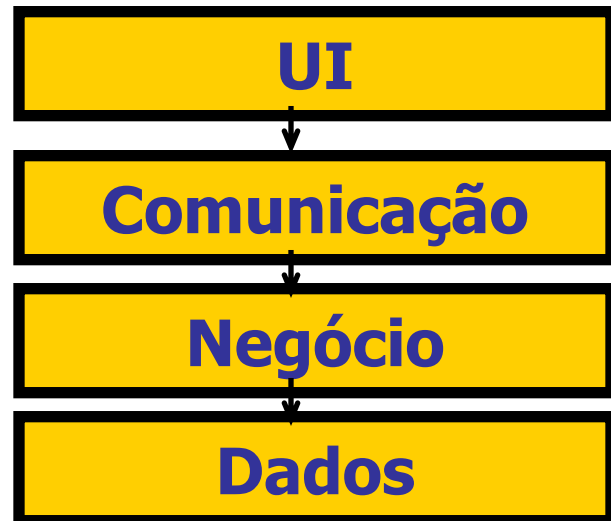
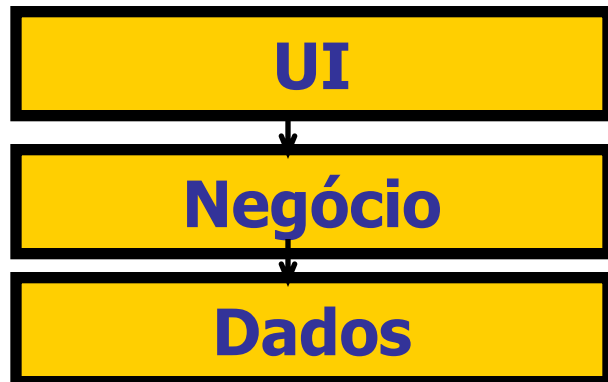


Definição

- Estimula a organização da arquitetura do sistema em um conjunto de camadas coesas com fraco acoplamento entre elas.
- Cada camada possui um propósito bem definido.
- A camada superior conhece apenas a camada imediatamente inferior (que fornece seus serviços através de uma interface).

Definição

- Cada camada é formada por um conjunto de classes com um determinado propósito.





Propósito de Cada Camada

- **UI:** agrega as classes do sistema com as quais os usuários interagem.
- **Negócio:** mantém as classes do sistema responsáveis pelos serviços e regras do negócio.
- **Dados:** camada responsável pelo armazenamento e recuperação dos dados persistentes do sistema.
- **Comunicação:** responsável pela distribuição do sistema em várias máquinas.



Vantagens e Desvantagem

- **Vantagens:**

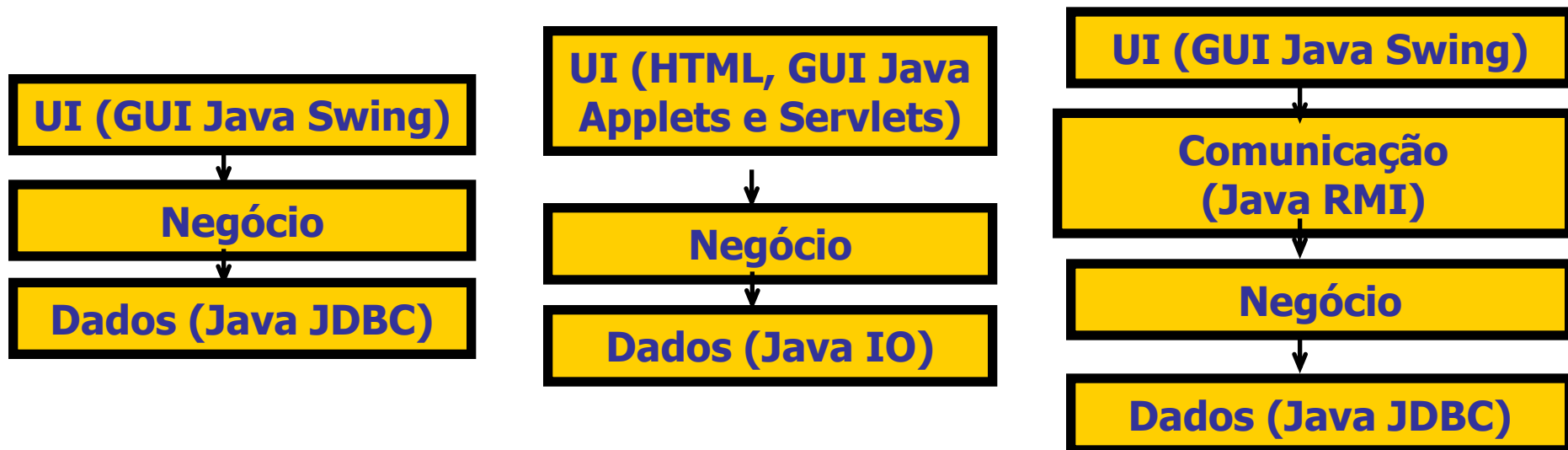
- **Separação de código** relativo a interface com o usuário (UI), comunicação, negócio e dados.
- **Permite a mudança de implementação de uma camada sem afetar a outra**, desde que a interface entre as mesmas seja mantida.
- **Possibilita que uma camada trabalhe com diferentes versões de outra camada.**

- **Desvantagem:**

- **Aumento no número de classes existentes no sistema.**

Padrão Arquitetura em Camadas

- Exemplos de diferentes configurações do padrão arquitetura em camadas usando tecnologias Java.





Padrão Arquitetura em Camadas

- Arquitetura em 3 camadas
 - Possui as camadas: UI, Regras de Negócio e Acesso a Dados
 - **A camada de UI:** agrega as classes de fronteira
 - Exemplo: GUIAluno
 - **A camada de Regras de Negócio:** agrega as classes de controle e entidade
 - Exemplos: ControladorAluno e Aluno
 - **A camada de Acesso a Dados:** agrega as classes de persistência dos dados
 - Exemplo: RepositorioAluno



Padrão Arquitetura em Camadas

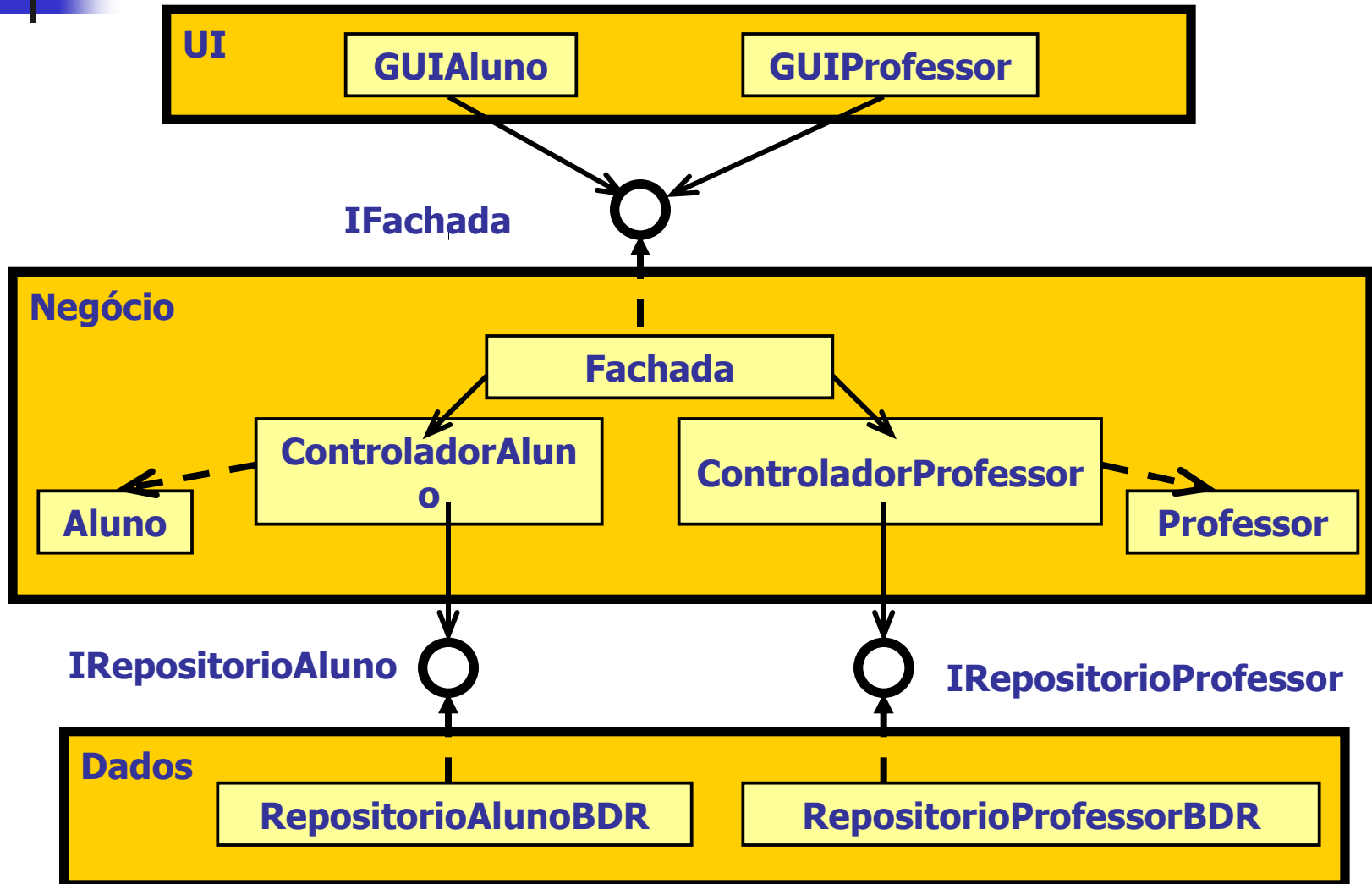
- Arquitetura em 3 camadas
 - Entre as camadas UI e Negócio haverá sempre uma **interface Java** que uma classe Fachada do sistema implementará.
 - A classe **Fachada** é utilizada para oferecer um caminho único para acesso aos serviços da camada de regras de negócio.
 - **As classes da UI, portanto, comunicam-se apenas com a classe Fachada**, que por sua vez colabora com as outras classes internas da camada de regras de negócio para oferecer os serviços.



Padrão Arquitetura em Camadas

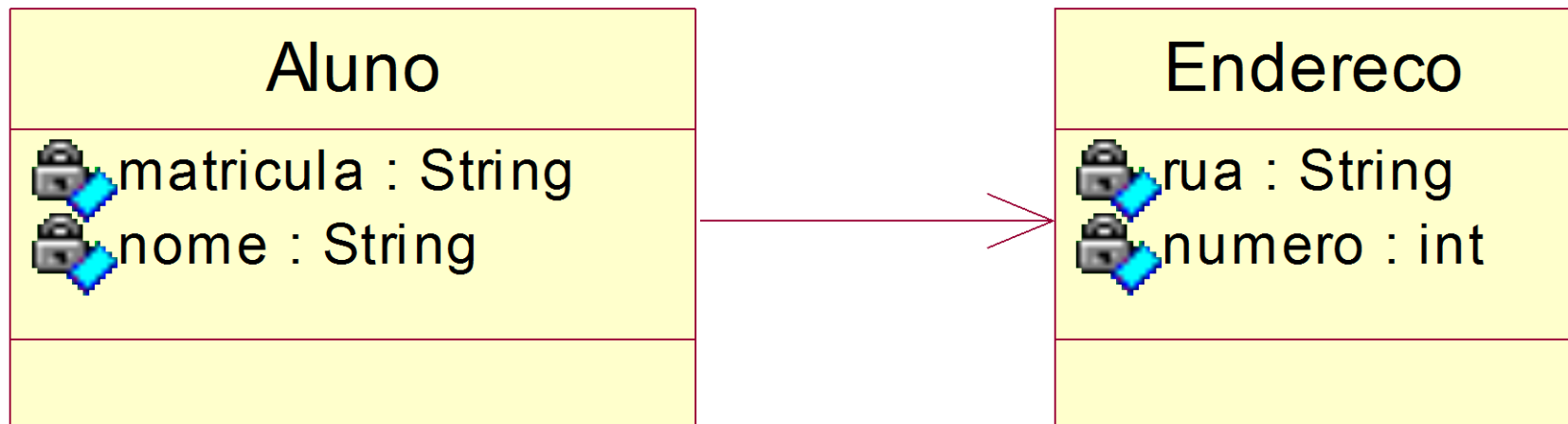
- Arquitetura em 3 camadas
 - O **Controlador** pode conter regras de controle do sistema e delega ações da fachada para a camada de acesso a dados.
 - Entre as camadas Negócio e Dados haverá sempre uma **interface Java** que uma classe Repositório implementará.
 - O **Repositório** armazena os objetos persistentes do sistema em algum meio de armazenamento físico (banco de dados, arquivo, etc.).

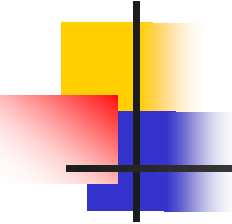
Visão Arquitetural



Implementação da Arquitetura em 3 Camadas

- Exemplo: Implementação de um Sistema para cadastro de alunos
 - Classes Básicas de Negócio (Entidades):
Aluno, Endereco



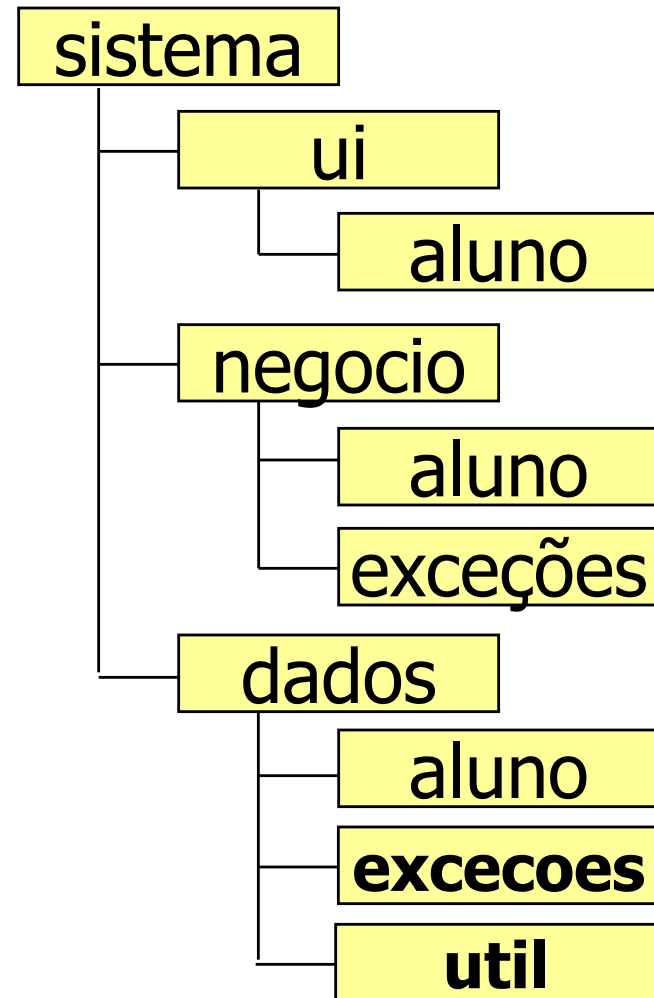


Implementação da Arquitetura em 3 Camadas

- **Divisão em Pacotes**
 - Um **pacote** é um conjunto de classes.
 - Agrupar em um pacote classes fortemente relacionadas.
 - O pacote deve ser visto como um elemento altamente coeso:
 - Classes de um mesmo pacote devem ter acoplamento e colaboração relativamente altos.
 - Por outro lado, o acoplamento e colaboração entre classes agrupadas em diferentes pacotes devem ser relativamente baixos.

Implementação da Arquitetura em 3 Camadas

- Divisão em Pacotes
 - **Exemplo:**



Implementação da Arquitetura em 3 Camadas

■ Divisão em Pacotes

- Para incluir uma classe em um pacote, utiliza-se a palavra reservada **package**.

Sintaxe: **package nomePacote;**

- Obrigatoriamente, é o primeiro comando do código-fonte.
- Pacotes estão diretamente associados a diretórios no sistema de arquivos.
- Exemplos:
 - **package sistema.ui.aluno;**
 - **package sistema.negocio.aluno;**

Implementação da Arquitetura em 3 Camadas

- **Divisão em Pacotes**

- **Importando pacotes**

- Utilizar a palavra reservada **import**.
- Pode-se importar uma classe específica ou todas as classes de um pacote.

- **Sintaxe:**

```
import nomePacote.*;  
import nomePacote.NomeClasse;
```

- **Exemplos:**

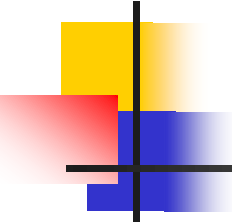
```
import sistema.negocio.aluno.Aluno;  
import sistema.negocio.aluno.*;
```

Implementação da Arquitetura em 3 Camadas

- Entidades

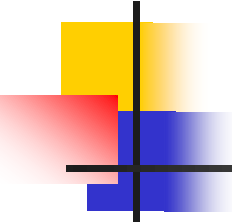
```
public class Aluno {  
    ...  
}
```

```
public class Endereco {  
    ...  
}
```

Implementação da Arquitetura em 3 Camadas

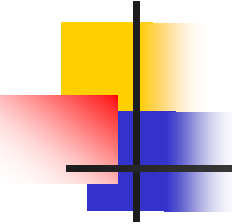
- **Interface da Fachada**
 - Irá oferecer todos os serviços do sistema.
 - Exemplo: Serviços básicos de manutenção de um cadastro tais como inserir, alterar, remover e buscar.



Implementação da Arquitetura em 3 Camadas

■ Interface da Fachada

```
public interface IFachada {  
public void inserirAluno(Aluno aluno) throws  
    ExcecaoElementoJaExistente, ExcecaoRepositorio;  
public void alterarAluno(Aluno aluno) throws  
    ExcecaoElementoInexistente, ExcecaoRepositorio;  
public void removerAluno(String mat) throws  
    ExcecaoElementoInexistente, ExcecaoRepositorio;  
public Aluno buscarAluno(String mat) throws  
    ExcecaoElementoInexistente, ExcecaoRepositorio;  
}
```



Implementação da Arquitetura em 3 Camadas

■ Fachada

- Obrigatoriamente, deve implementar todos os métodos da interface da fachada.
- Tem um ou mais atributos do tipo **Controlador**.
- Delega para os controladores as chamadas de métodos.

Implementação da Arquitetura em 3 Camadas

■ Fachada

```
public class Fachada implements IFachada {
    private ControladorAluno controladorAluno;

    public Fachada() {
        this.controladorAluno = new ControladorAluno();
    }

    public void inserirAluno(Aluno aluno) throws
        ExcecaoElementoJaExistente, ExcecaoRepositorio {
        this.controladorAluno.inserirAluno(aluno);
    }

    /* Continua aqui a implementação dos demais métodos! */
}
```

Implementação da Arquitetura em 3 Camadas

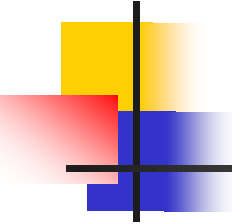
■ Controlador

- Um controlador para cada entidade persistente.
- Contém os métodos que serão chamados pela fachada.
 - Exemplo: Serviços básicos de manutenção de um cadastro tais como inserir, alterar, remover e buscar.
- Tem um atributo do tipo **IRepositorio** para acessar os métodos da camada de acesso a dados.

Implementação da Arquitetura em 3 Camadas

■ Controlador

```
public class ControladorAluno {
    private IRepositorioAluno repAlunos;
    public ControladorAluno( ) {
        this.repAlunos = new RepositorioAlunos( );
    }
    public void inserirAluno(Aluno aluno) throws
        ExcecaoElementoJaExistente, ExcecaoRepositorio {
        this.repAlunos.inserirAluno(aluno);
    }
    /* Continua aqui a implementação dos demais métodos! */
}
```



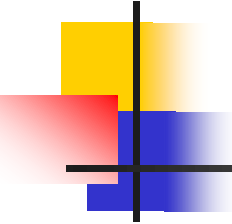
Implementação da Arquitetura em 3 Camadas

- **Interface do Repositório**
 - Irá oferecer os serviços de persistência de dados
 - Uma interface do repositório para cada entidade persistente.
 - Exemplo: Serviços básicos de persistência tais como inserir, alterar, remover, buscar e verificar se um determinado objeto existe.

Implementação da Arquitetura em 3 Camadas

■ Interface do Repositório

```
public interface IRepositorioAluno {  
    public void inserirAluno(Aluno aluno) throws  
        ExcecaoElementoJaExistente, ExcecaoRepositorio;  
    public void alterarAluno(Aluno aluno) throws  
        ExcecaoElementoInexistente, ExcecaoRepositorio;  
    public void removerAluno(String mat) throws  
        ExcecaoElementoInexistente, ExcecaoRepositorio;  
    public Aluno buscarAluno(String mat) throws  
        ExcecaoElementoInexistente, ExcecaoRepositorio;  
    public boolean verificarExistenciaAluno(String matricula);  
}
```

Implementação da Arquitetura em 3 Camadas

■ Repositório

- Implementa a persistência dos dados.
- Obrigatoriamente, deve implementar todos os métodos da interface do repositório.
- Contém os métodos que serão chamados pelo controlador.
 - Exemplo: Serviços básicos de persistência tais como inserir, alterar, remover, buscar e verificar se um determinado objeto existe.

Implementação da Arquitetura em 3 Camadas

■ Repositório (versão em array)

```
public class RepositorioAlunoArray implements IRepositorioAlunos {
    private Aluno[ ] alunos;
    private int quantAlunos;
    public RepositorioAlunoArray() {
        this.alunos = new Aluno[100];
        this.quantAlunos = 0;
    }
    public void inserirAluno(Aluno aluno) throws ExcecaoElementoJaExistente,
                                                ExcecaoRepositorio {
        if (this.verificarExistenciaAluno(aluno.getMatricula()) == false) {
            this.alunos[quantAlunos++] = aluno;
        } else {
            throw new ExcecaoElementoJaExistente("Aluno Já Cadastrado!");
        }
    }
    /* Continua aqui a implementação dos demais métodos! */
}
```

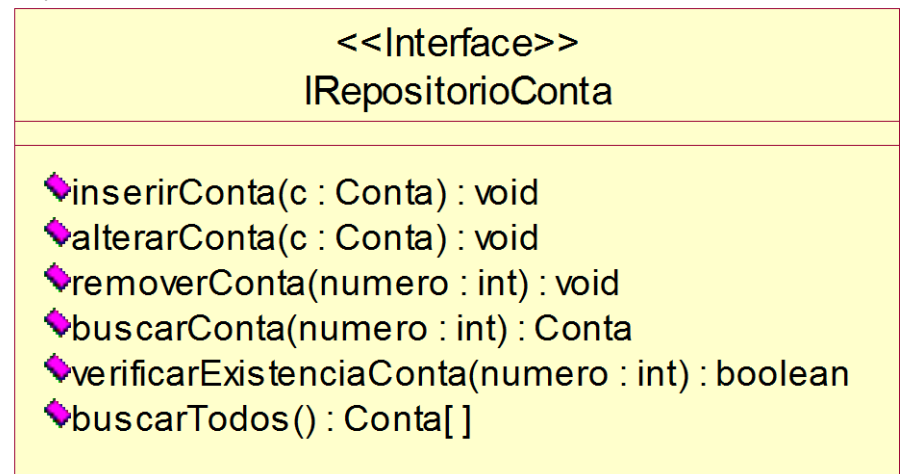
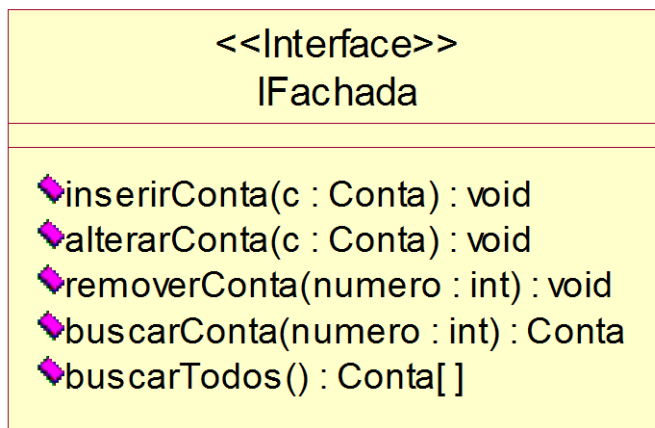
Implementação da Arquitetura em 3 Camadas

- Aplicação (Classe de interação com o usuário representando a camada de UI da arquitetura)

```
public class Aplicacao {  
    private static IFachada fachada = new Fachada( );  
    public static void main(String[ ] args) {  
        try {  
            Endereco end = new Endereco("Masc. Moraes", 111);  
            Aluno al = new Aluno("123", "João", end);  
            fachada.inserirAluno(al);  
        } catch (ExcecaoDadoInvalido e) {  
            System.out.println(e.getMessage( ));  
        } catch (ExcecaoElementoJaExistente e) {  
            System.out.println(e.getMessage( ));  
        } catch (ExcecaoRepositorio e) {  
            System.out.println(e.getMessage( ));  
        }  
    }  
}
```

Exercício

- Implementar a arquitetura em 3 camadas no cadastro de contas (inserir, buscar, remover, alterar, verificar existência e imprimir relatório com todas as contas cadastradas).
 - OBS 1: Utilize *array* no repositório.
 - OBS 2: Imprimir dados na tela é função da camada de interface de usuário.





Referências

- Leitura Recomendada:
 - Artigo: *Integrando Java com Bancos de Dados Relacionais*. Autores: Euricélia Viana e Paulo Borba (<http://www.cin.ufpe.br/~phmb/publications.htm>)
 - Artigo: *PDC: The persistent data collections pattern*. Autores: Tiago Massoni, Vander Alves, Sérgio Soares e Paulo Borba (<http://www.cin.ufpe.br/~phmb/publications.htm>)
 - Livro: *Design Patterns: Elements of Reusable Object-Oriented Software*. Autores: Erich Gamma et al. Editora Addison-Wesley