Módulo IV

Uma Proposta Inicial de Projeto para Aplicações Java

Objetivo

Descrever como implementar aplicações Java a partir de produtos do processo de software

Créditos

Autor

Prof. Alessandro Cerqueira

(alessandro.cerqueira@hotmail.com)

Camadas da Aplicação

- Ao iniciarmos o projeto de uma aplicação, devemos visualizá-la em três camadas fundamentais:
  - Camada de Interface (ou de Apresentação)
    - É composta pelos objetos responsáveis pelo estabelecimento de uma interface gráfica com o usuário (GUI – Graphical User Interface).
  - Camada de Controle (ou de Negócios)
    - É composta pelos objetos responsáveis pela execução dos casos de uso do sistema.
  - Camada de Modelo (ou de Dados, ou de Domínio)
    - É composta pelos objetos pertencentes às classes geradas pelo processo de modelagem de dados; assim, estes descrevem os os dados que são manipulados durante a execução do sistema

Projeto da Camada de Interface

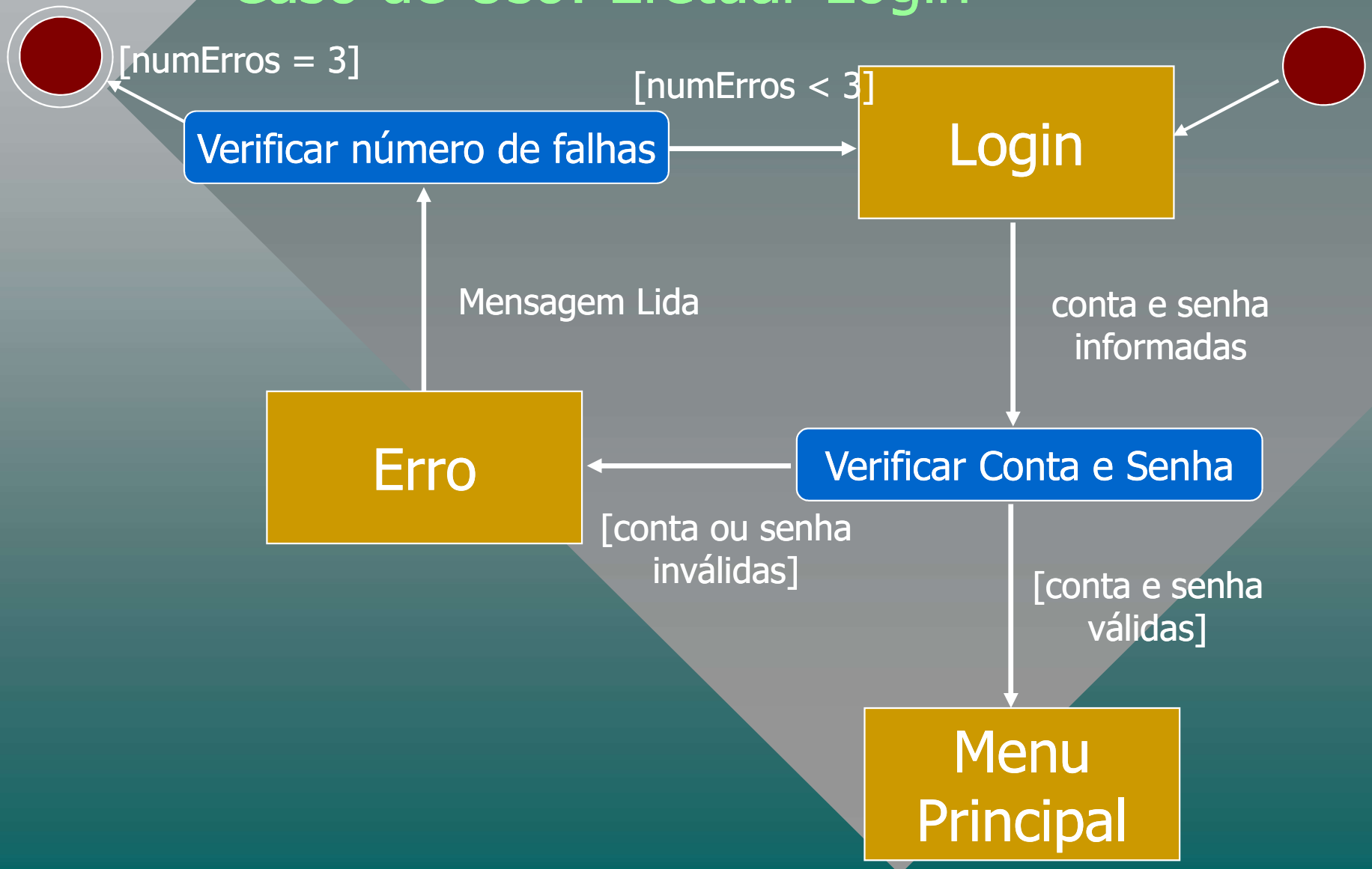
- Projeto de Interface
  - Deve ser realizado durante o *fluxo de trabalho* de Projeto.
  - Identificar quais elementos de interface que serão necessários para dar suporte às diversas ações que podem ser executadas durante os casos de uso do sistema.
  - Dividir o projeto de interface em duas fases:
    - Projeto de Interface Abstrato
      - Identifica os artefatos de interface que serão gerados, quais são os eventos aos quais estes são sensíveis, quais ações devem ser executadas e quais transições devem ser efetuadas.
      - Este projeto pode ser elaborado a partir de um diagrama de máquina de estados.
      - Regra: Para cada *caso de uso* devemos gerar um diagrama de *máquina de estados*. Cada *estado* deve representar um *artefato de interface* (tela ou janela) através do qual o usuário interage.
    - Projeto de Interface Concreto
      - Design visual dos elementos de interface a serem gerados.

Projeto da Camada de Interface

- Projeto de Interface (continuação)
  - Em aplicações tradicionais, realizamos o design das janelas do sistema. Em aplicações web, realizamos o design das páginas HTML com seus formulários.
  - Identificar quais são os eventos aos quais os elementos de interface são sensíveis.
  - Com isto, estamos preparando o caminho para que a aplicação adote o estilo de Programação Orientada a Eventos.

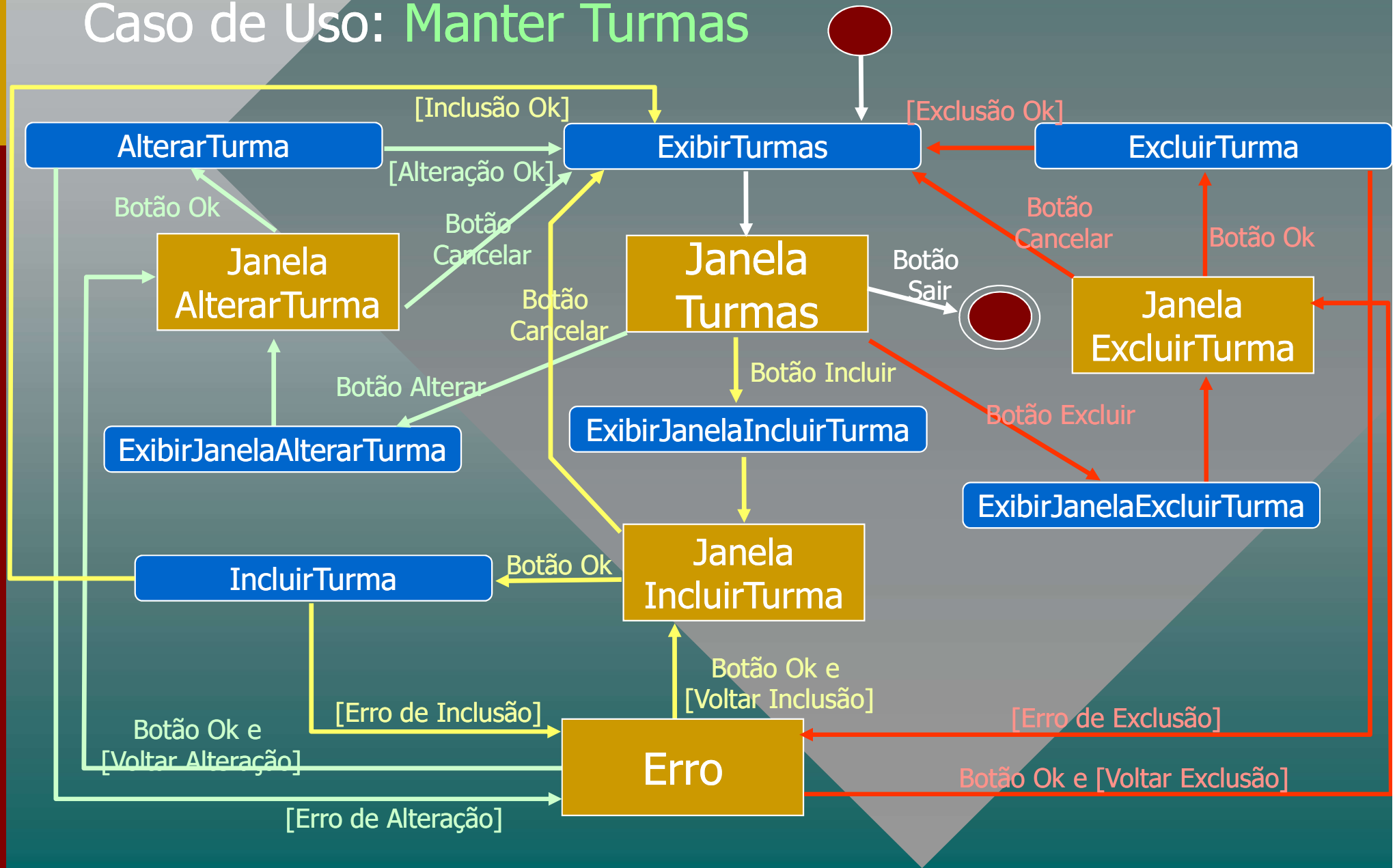
Projeto de Interface Abstrato
(Diagrama de Máquina de Estados)

Caso de Uso: Efetuar Login



Projeto de Interface Abstrato
(Diagrama de Máquina de Estados)

Caso de Uso: Manter Turmas



Projeto da Camada de Interface

- Para cada caso de uso gerar um diagrama de máquina de estados
  - Cada estado representa uma janela ou página a ser implementada
  - Os eventos produzidos pelos usuários indicam a realização de uma ação do sistema na execução do caso de uso
  - De acordo com as condições de término da ação, é realizada uma transição para outro artefato de interface.

Projeto da Camada de Interface

- Regras Básicas para Criação
  - Para aplicações convencionais:
    - Para cada tipo de janela identificada no Projeto, iremos criar uma classe em Java.
    - Para cada elemento de interface na janela, deve ser adicionado um atributo desta classe.
    - Para cada evento ao qual esta janela é sensível deve ser um método desta classe. A missão deste método é entrar em contato com o controlador do caso de uso e notificá-lo da ocorrência do evento.
  - Para aplicações web:
    - Para cada tipo de página identificada no Projeto, iremos criar uma página JSP.
    - Cada elemento de interface é inserido na página utilizando componentes visuais de interface (ex. em JSF → Text).
    - Cada evento ao qual esta janela é sensível deve ter um código que efetue uma requisição ao servidor.

Projeto da Camada de Interface

- Regras Básicas para Criação
  - Caso observemos que um conjunto de elementos esteja se repetindo em várias janelas, podemos criar uma classe para este conjunto
    - **Estímulo ao reuso**
  - Os métodos da classe de interface não devem implementar nem fazer referência a conceitos do domínio. Simplesmente deverão avisar ao controlador responsável que tal evento ocorreu.

Projeto da Camada de Controle

- Modelagem de Casos de Uso
  - Deve ser realizado durante o fluxo de trabalho de Levantamento de Requisitos.
- Um sistema é composto de vários casos de uso. Cada caso de uso descreve um conjunto de ações que são executadas até que um bem tangível seja dado ao usuário ao final de sua execução.
  - Um caso de uso indica uma funcionalidade do sistema.
- Regra Geral:
  - Para cada caso de uso iremos codificar uma classe que implementará um controlador para o caso de uso.
  - Cada ação tratada pelo caso de uso (identificada no diagrama de gráficos de estado) irá se transformar em um método.
  - A execução do método de ação será disparada a partir do envio de uma mensagem de um objeto da camada de interface para o controlador.

Projeto da Camada de Controle

- Variante para a Regra
  - Crie uma classe abstrata chamada Comando que contenha um método abstrato chamado *executar()*
  - Cada ação do caso de uso poderá se transformar em uma classe que seja especialização da classe Comando (ou seja, que codifique o método executar())
    - O controlador de casos de uso será uma composição de Comandos que ficam registrados em um *HashMap* (tópico futuro)
    - A partir da ordem recebida pela interface, recuperar o objeto Comando e execute o método executar()
    - O grupamento de classes de ação deve ficar em um pacote próprio ou como classes internas à classe do controlador.

Projeto da Camada do Modelo

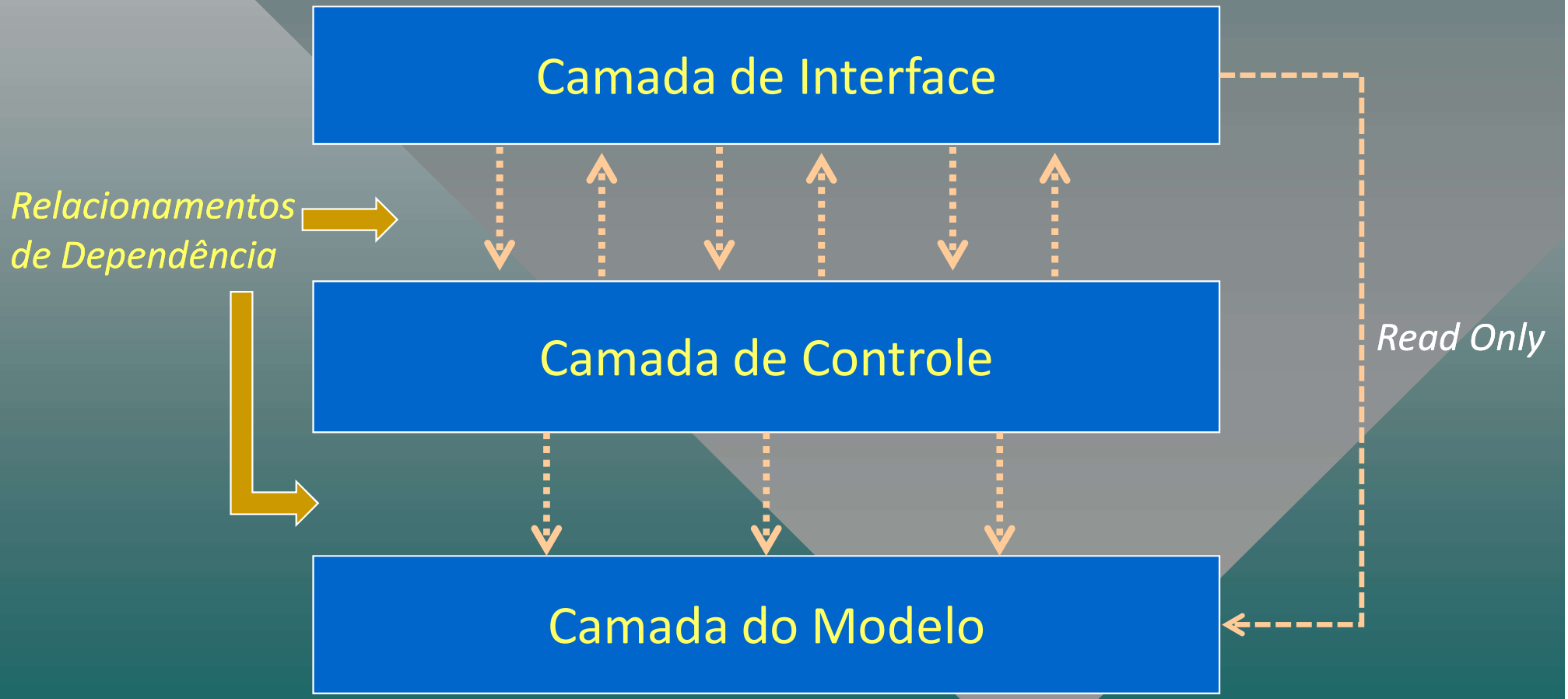
- Modelagem de Dados
  - Deve ser realizado durante o fluxo de trabalho de Análise.
- As classes do modelo são aquelas que identificamos na modelagem de dados nas atividades de análise.
  - Pela UML, documentamos a modelagem de dados através de um diagrama de classes
- Regra
  - Cada classe identificada na modelagem de dados deverá ser implementada sob o padrão JavaBeans ou PoJo (módulo 06).
    - Resumo
      - *Pelo menos ter o construtor com assinatura vazia*
      - *Métodos **get** e **set** vinculados a cada atributo*
  - Vinculada a cada classe JavaBean iremos também construir também um DAO (Data Access Object)

Projeto da Camada do Modelo

- Data Access Object (DAO)
  - São objetos que interagem com os controladores com o objetivo de fazer a busca, recuperação e armazenamento dos objetos do modelo.
  - Para cada classe presente no diagrama de classes iremos criar uma classe DAO
    - Ex: Classe Turma → Classe DaoTurma
      Classe Aluno → Classe DaoAluno
  - Se um controlador precisar recuperar um ou mais objetos de uma classe do modelo, ele o fará junto ao DAO da classe necessária. Se um objeto precisar ser persistido (armazenado), quem o fará será o DAO
  - A estratégia de persistência fica encapsulada nas classes DAO.

Camadas da Aplicação

- A comunicação entre as camadas deve seguir a seguinte estratégia:



Relacionamento de Dependência

- Dependência é um relacionamento previsto na UML que indica que objetos de uma determinada classe enviam mensagens para objetos de outra classe.
  - A troca de mensagens indica que há um processo de **colaboração** entre objetos.
- Não precisamos expressar dependência quando especificamos uma associação ou agregação.
  - Quando temos associação ou agregação, os objetos naturalmente irão colaborar (trocar mensagens)
  - Só precisamos especificar dependência quando não temos associação ou agregação.
- **Se** um objeto A tem dependência com um objeto B, **Então** A tem uma referência (ponteiro) para B.
- De forma geral, a maneira mais comum de guardar esta referência é *colocar na classe A um atributo que aponte para o objeto B* com o qual se tem dependência.

Artefatos do Processo de Desenvolvimento

- Todos os artefatos produzidos no Processo de Software influenciam direta ou indiretamente a implementação; entretanto destacamos alguns dos mais importantes:
- Diagrama de Casos de Uso
  - Para geração da camada de controle
- Modelagem de Dados (diagrama de classes)
  - Para geração da camada do modelo
  - Para geração de DAOs (Data Access Objects)
- Projeto de Interface Abstrato realizado com o Diagrama de Gráficos de Estado
  - Para identificação dos elementos de interface
- Projeto de BD
  - Para a geração do mapeamento OO → Relacional

Padrões de Software

- Um padrão é a descrição de uma estratégia que pode-se adotar para solucionar um determinado tipo de problema que pode ocorrer de formas diferentes mas que, na essência, apresenta a mesma estrutura.
- Paródia
  - Um time de futebol chamado "*Fiasco na Grama*" e que é treinado pelo técnico Sr. Treineiro está perdendo suas partidas há 20 jogos. O presidente do clube pergunta a seus beneméritos patrícios o que fazer. Solução: demitir o técnico Sr. Treineiro.
  - Na realidade, esta solução não se aplica somente ao Fiasco da Grama!
    - poderia ser aplicada a qualquer time de futebol,
    - Mas ainda, poderia ser aplicada a qualquer equipe esportiva.
    - Mais ainda, pode ser aplicada a qualquer equipe de trabalho que não está produzindo resultados.
  - Assim, vemos que o problema do *Fiasco na Grama* se enquadra em um problema genérico ao qual temos uma solução genérica que chamamos de Padrão de Projeto.
- Padrão "Detonator"
  - Problema genérico: Uma equipe de trabalho não está produzindo resultados.
  - Solução genérica: Detonar seu coordenador.

Tipos de Padrões de Software

- Existem vários catálogos de padrões a serem adotados no desenvolvimento de software
  - Padrões de Projeto Clássicos
    - Também chamados de Padrões de Projeto GoF (Gang of Four)
    - Foi o primeiro catálogo relevante a ser publicado
  - Padrões GRASP (Responsabilidade)
  - Padrões JavaEE
  - Padrões Arquiteturais

Padrões de Projeto

- Padrão
  - Maneira testada ou documentada de alcançar um objetivo qualquer
- *Design Patterns* ou Padrões de Projeto
  - Padrões para alcançar objetivos na engenharia de software usando classes e métodos em linguagens orientadas a objeto
- "Design Patterns: Elements of Reusable Object-Oriented Software" (1994)
  - Livro de Erich Gamma, John Vlissides, Ralph Johnson e Richard Helm (Conhecidos como "The Gang of Four", ou GoF)
  - Catálogo que descreve 23 padrões de projeto úteis.
  - Inspirado em "A Pattern Language" de Christopher Alexander sobre padrões de arquitetura de cidades, casas e prédios.
- *"Os padrões de projeto são descrições de objetos que se comunicam e classes que são customizadas para resolver um problema genérico de design em um contexto específico"* (GoF)

Padrões de Projeto Clássicos

Classificação

- Quanto ao Escopo:
  - **Classe**: Padrões que tratam do relacionamento entre classes e subclasses.
  - **Objeto**: Padrões tratam relacionamentos entre objetos e por isso podem ser alterados em tempo de execução.
- Propósito
  - **Criacional**: Trata do processo de criação de um objeto;
  - **Estrutural**: Trata da composição de objetos e classes
  - **Comportamental**: Trata do modo como classes e objetos interagem e compartilham responsabilidades

Padrões de Projeto Clássicos

		Propósito		
		Criacional	Estrutural	Comportamental
Escopo	Classe	Factory Method	Adapter (classe)	Interpreter
				Template Method
	Objeto	Abstract Factory	Adapter (objeto)	Chain of Responsability
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Façade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

Padrão Model/Viewer/Controller

- Vemos que a implementação de um sistema (em qualquer linguagem) é um problema genérico, para ela temos uma solução genérica que é projetar a aplicação seguindo o padrão MVC (Model-Viewer-Controller)
  - Model → Classes do Modelo
  - Viewer → Classes da Interface
  - Controller → Classes de Controle
- A grande vantagem desta estruturação é a obtenção da Independência de Diálogo
  - A troca no projeto de interface não implica em alterações nos métodos ligados ao controle ou ao domínio.

MVC
Padrão Arquitetural

- O Model-Viewer-Controller não é um dos padrões de projeto clássicos.
  - Na realidade é um composto de padrões de projeto clássicos
  - MVC é um Padrão Arquitetural

Camadas da Aplicação

- A princípio, vamos estruturar nossas aplicações da seguinte forma:
  - As classes da camada de interface farão parte do pacote "viewer"
  - As classes da camada de controle farão parte do pacote "controller"
  - As classes da camada do modelo farão parte do pacote "model"

Estratégia para a Implementação

Sugestão

- Em cada classe *viewer* coloque um atributo que aponte para o *controller* ao qual a janela precise interagir.
- Acrescente um parâmetro no construtor da classe *viewer* para receber a referência para o *controller*. Faça a atribuição do atributo indicado acima utilizando este parâmetro.
- Acrescente no *controller* um atributo para cada objeto *viewer* com o qual ele precise interagir. Deixe a cargo do controlador a tarefa de instanciar o objeto *viewer*.
- Acrescente no *controller* um atributo para cada *DAO* ao qual ele precise interagir. Deixe a cargo do *controller* a tarefa de instanciar o *DAO*.

Estratégia para a Implementação

Sugestão

- Para cada aplicação iremos crie um *Controlador do Programa* (*CtrlPrograma*). Coloque nesta classe o método *main*.
- A tarefa do método *main* será instanciar o objeto *CtrlPrograma*.
- Crie um objeto *viewer* para ser a "*JanelaPrograma*" da aplicação. Estabeleça a ligação do *CtrlPrograma* com este objeto *viewer*.
- Coloque na "*JanelaPrograma*" elementos de interface que enviem para o *CtrlPrograma* mensagens de solicitação para se iniciar um caso de uso (instanciação do objeto *controller* do caso de uso).