

# Roteiro

# Reutilização de Projeto e Software

-

•  
•  
•

# Reutilização de Projeto

- ❖ **Design Patterns** - padrões de projeto e análise
- ❖ **Frameworks** orientados a objetos

• • • • • • • • • •

•  
•  
•

# Padrões de Projeto e Análise

**Padrões de organização de hierarquias de classes, protocolos e distribuição de responsabilidades entre classes, que caracterizam construções elementares de projeto orientado a objetos.**

**Um padrão de projeto é uma estrutura que aparece repetidamente nos projetos orientados a objetos para resolver um determinado problema de forma flexível e adaptável dinamicamente.**

• • • • • • • • • •

•  
•  
•

# Padrões de Projeto e Análise

## *Tipos de padrões de projeto*

- ✓ De criação – **abstract factory**, builder, factory method, prototype, singleton
- ✓ Estruturais – adapter, bridge, **composite**, decorator, facade, flyweight, proxy
- ✓ Comportamentais – chain of responsibility, command, interpreter, iterator, mediator, memento, **observer**, state, **strategy**, template method, visitor

• • • • • • • • • •

•  
•  
•

# Padrões de Projeto e Análise

*Descrição padrão de um “design pattern”*

- ✓ objetivo - problema
- ✓ motivação - cenário
- ✓ aplicabilidade - situação
- ✓ estrutura – representação gráfica
- ✓ participantes – classes e/ou objetos
- ✓ colaborações – responsabilidades dos participantes
- ✓ consequências – compromissos e resultados
- ✓ implementação - técnicas
- ✓ exemplo de codificação

• • • • • • • • • •

•  
•  
•

# Padrões de Projeto

## *Composite* - Problema

Na aplicação de Controle de Projetos, um projeto pode ser de rede, de consultoria ou de desenvolvimento de sistemas.

Este último é na verdade uma composição de projetos que devem ser executados pela área de desenvolvimento e pelas outras especialidades mencionadas acima.

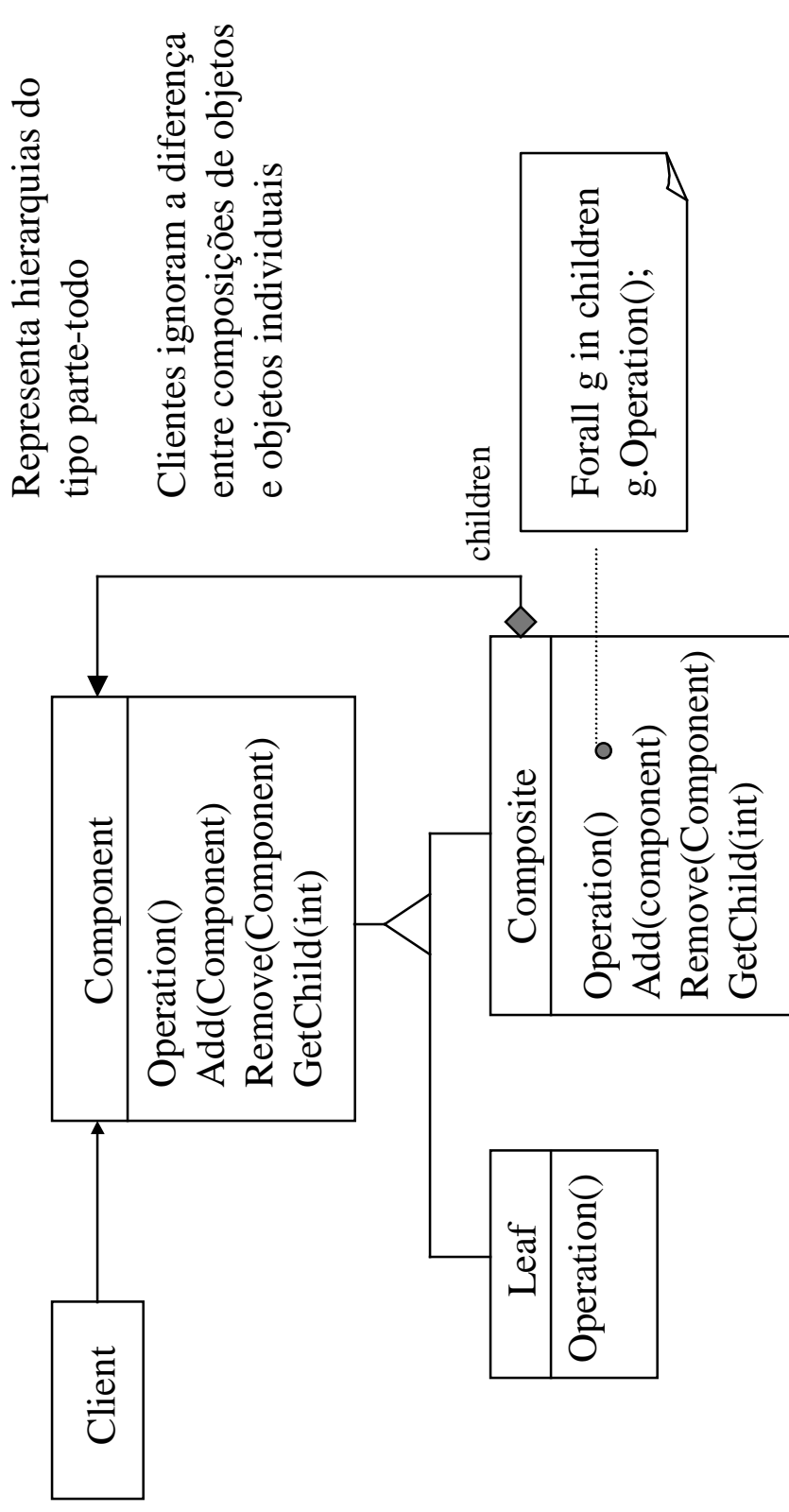
Quando um projeto-objeto receber a mensagem `descreve()`, não queremos nos preocupar com o tipo de projeto que está recebendo a mensagem.

Como estruturar a classe Projeto de forma a permitir essa facilidade?

• • • • • • • • • •

# Padrões de Projeto

## *Composite*



- 
- 
- 

# Padrões de Projeto

## *Observer* - Problema

Voltemos à aplicação de Controle de Projetos da nossa empresa onde projetos-objetos devem ser sempre atualizados a respeito da alteração do departamento a que pertence o funcionário-objeto responsável pela sua coordenação.

Como modelar a estrutura de dados e os métodos das classes Projeto e Funcionário para atender esse requisito?

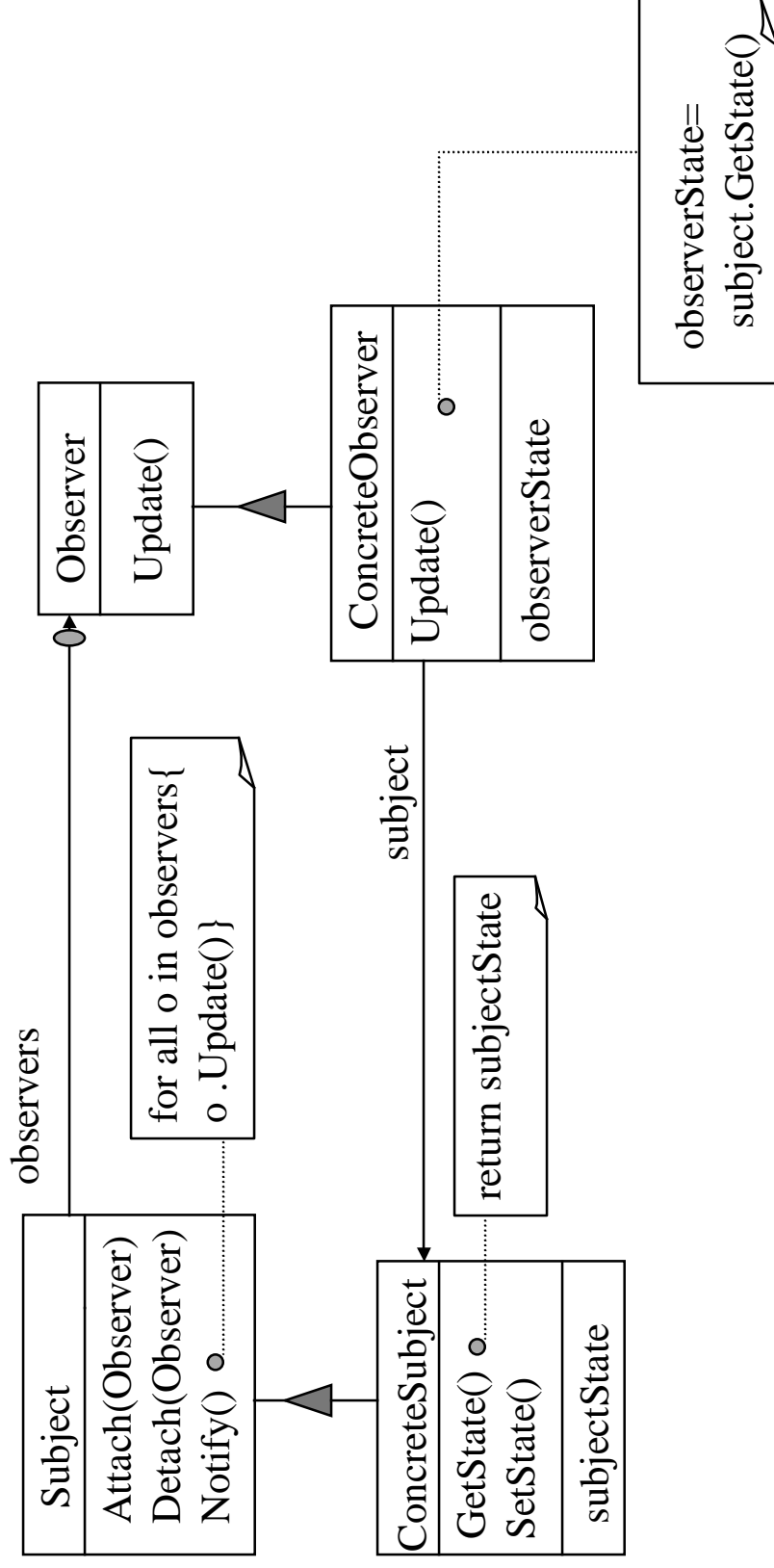
---

- 
- 
- 
- 
- 
- 
-



# Padrões de Projeto

**Observer:** define uma dependência de um para muitos entre objetos de tal forma que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente



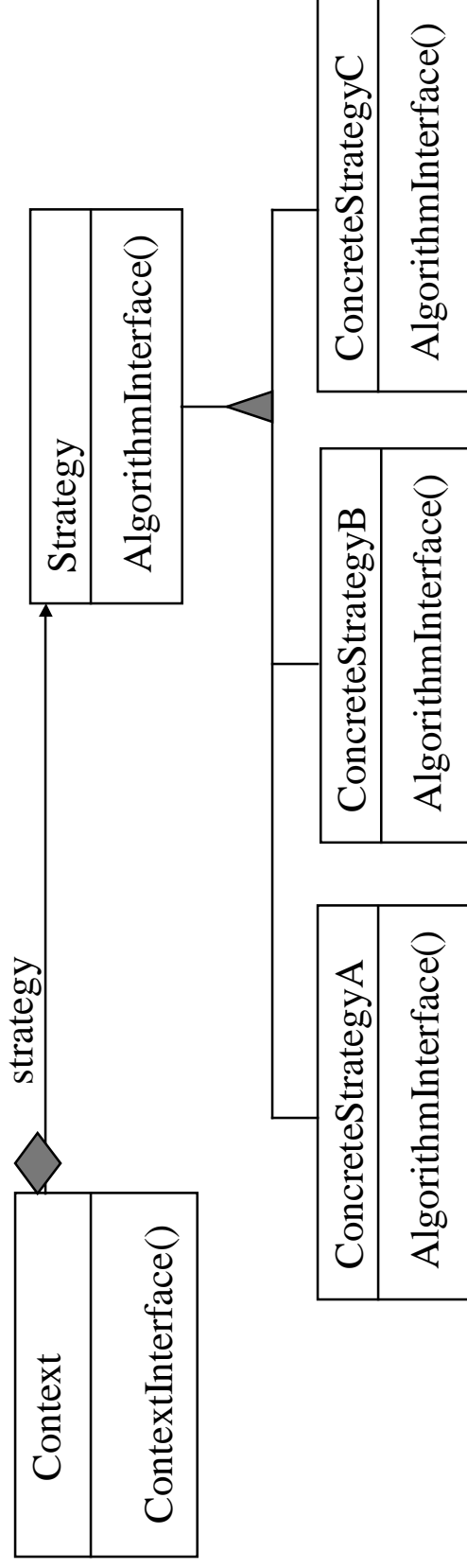
- # Padrões de Projeto

Na aplicação de Controle de Projetos, o custo de um projeto depende do tipo do projeto: desenvolvimento de sistemas, rede, consultoria.

Como modelar a estrutura de dados e os métodos da classe Projeto de tal forma que quando um projeto-objeto recebe mensagem calculaCusto(), não precisamos nos preocupar com o tipo de projeto que está recebendo a mensagem?

# Padrões de Projeto

**Strategy:** define uma família de algoritmos, encapsula cada um deles e os faz intercambiáveis. Permite que o algoritmo varie independentes do cliente.



# Padrões de Projeto

## *Abstract Factory* - Problema

Suponhamos uma empresa de prestação de serviços na área de desenvolvimento de Sistemas de Informação, projetos de rede de comunicação e consultoria em Tecnologia de Informação.

A aplicação de Controle de Projetos deve ser projetada de tal forma a respeitar as políticas e normas de cada uma das filiais.

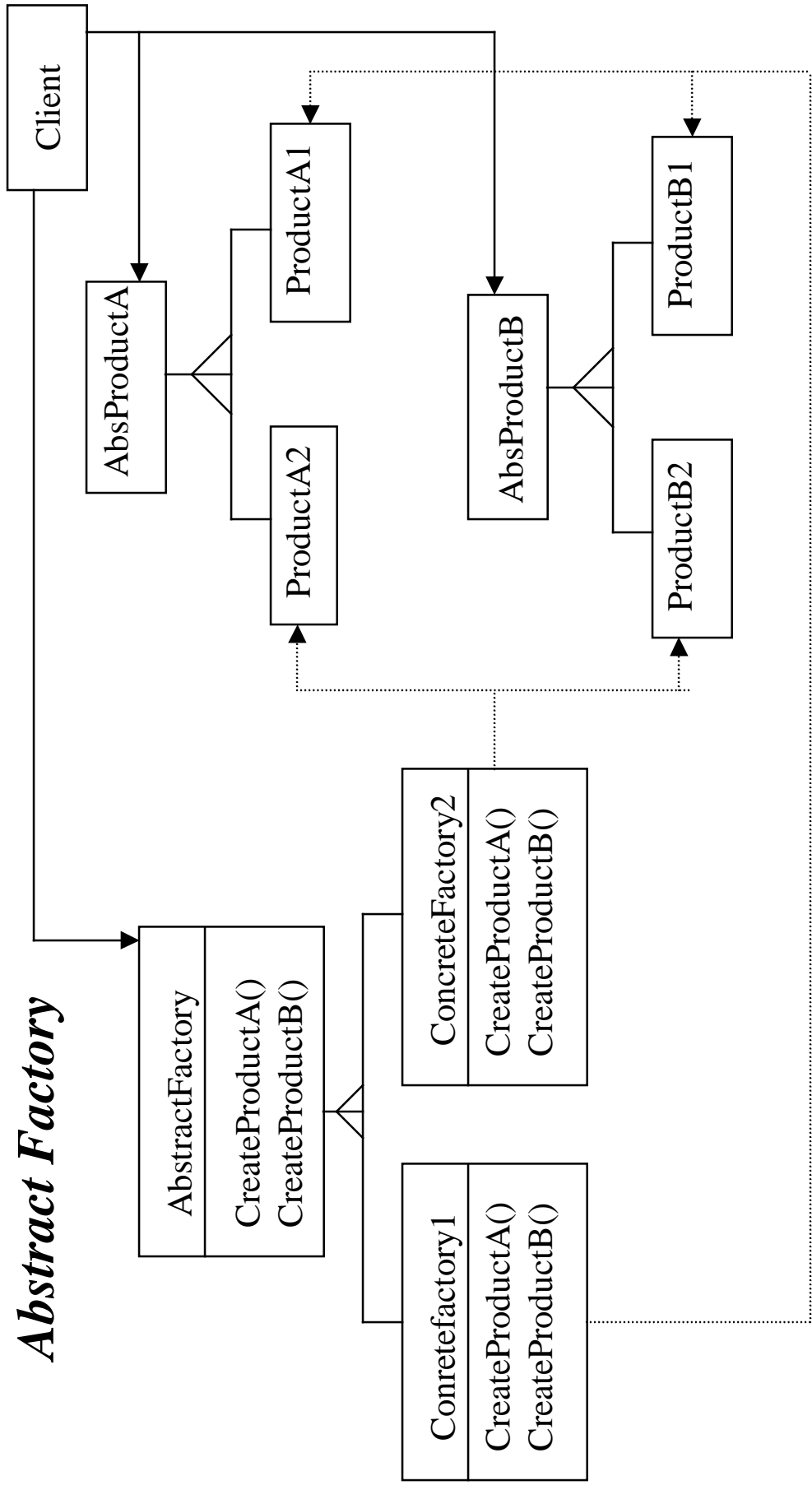
Não se quer fazer uma aplicação para cada uma destas filiais. Cada produto deve ser manipulado (instanciado e gerenciado) de acordo com as especificidades da filial correspondente.

Como estruturar a classe Produto de projetos de tal forma a permitir essa “independência” de qual filial o está tratando?

•  
•  
•

# Padrões de Projeto

## *Abstract Factory*



• • • • • • • • • •

- 
- 
- 

# Padrões de Projeto

## Como usar padrões de forma eficaz

- ✓ Estude os padrões aos quais você já tenha acesso
- ✓ Procure por conceitos básicos similares
- ✓ Aplique o padrão
- ✓ Não espere que tudo possa ser resolvido pelos padrões

## Como descobrir novos padrões

- ✓ O padrão pode ser utilizado em outros lugares?
- ✓ Utilize o padrão várias vezes
- ✓ Espere fazer mudanças em seu diagrama de classes

- # Padrões de Projeto

- ✓ Aumentam a produtividade dos desenvolvedores
- ✓ Aumentam a consistência entre as aplicações
- ✓ São potencialmente melhores do que os códigos reutilizáveis
- ✓ Podem ser utilizados em combinação para resolverem problemas difíceis
- ✓ Existem novos padrões sendo desenvolvidos todos os dias

- ✓ Precisamos aprender um grande número de padrões
- ✓ Alguns desenvolvedores não estão abertos a aceitar o trabalho dos outros
- ✓ Os padrões não são códigos

# Frameworks

Classes abstratas funcionam como um molde para as suas subclasses.

Da mesma forma, um projeto constituído por classes abstratas funciona como um molde para aplicações.

Um projeto constituído por classes abstratas é denominado ***framework* de aplicações orientado a objetos.**

Um *framework* pode ser considerado como uma infra-estrutura de classes que provêem o comportamento necessário para implementar aplicações dentro de um domínio através dos mecanismos de especialização e composição de objetos, típicos das linguagens orientadas a objetos





•  
•  
•

# *Frameworks - Exemplo*

## **MVC - *Model/View/Controller***

**Modelo:** contém os dados da aplicação junto com a lógica dos negócios que define como alterar e acessar os dados; o modelo pode ser compartilhado entre várias visões e controladores.

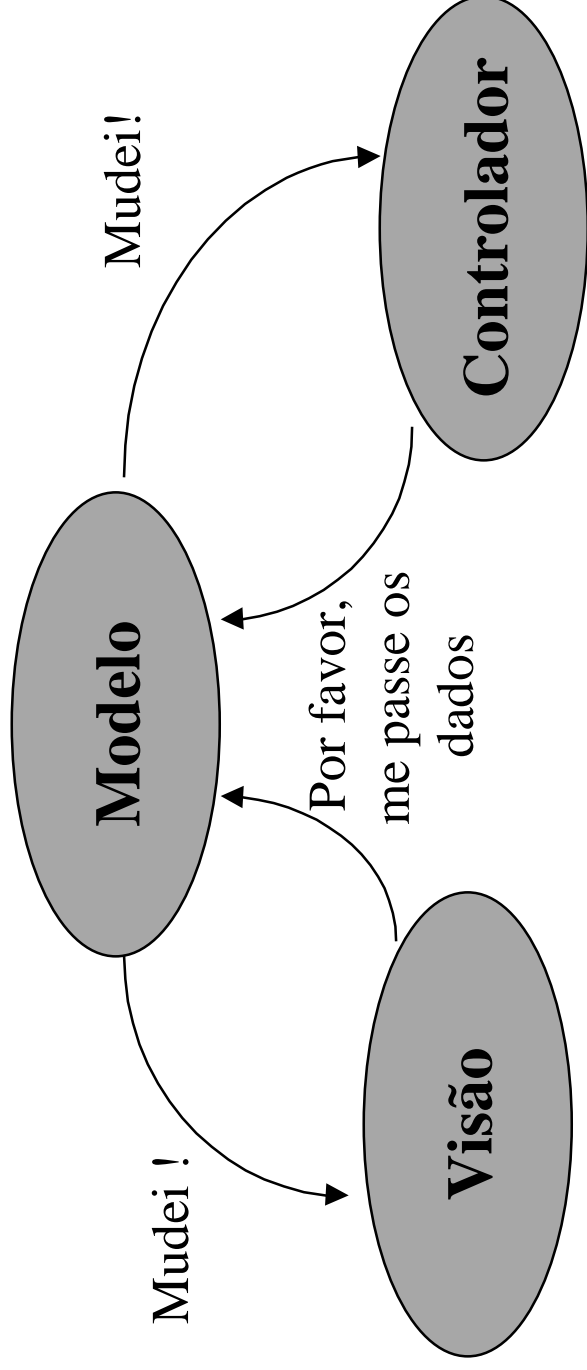
**Visão:** é a forma de apresentação dos dados do modelo para o mundo externo; pode ser na forma de GUI, fala, som, listagens ou mesmo em uma saída não orientada a usuários, como ligar um ar condicionado.

**Controlador:** é a forma de tratar a entrada do usuário ou outro meio e dar *feedback* para o modelo, normalmente alterando alguns de seus dados.

• • • • • • • • • •

•  
•  
•

# *Framework MVC*



•

•

•

•

•

•

•

•

•

•

•

•

•

•

•

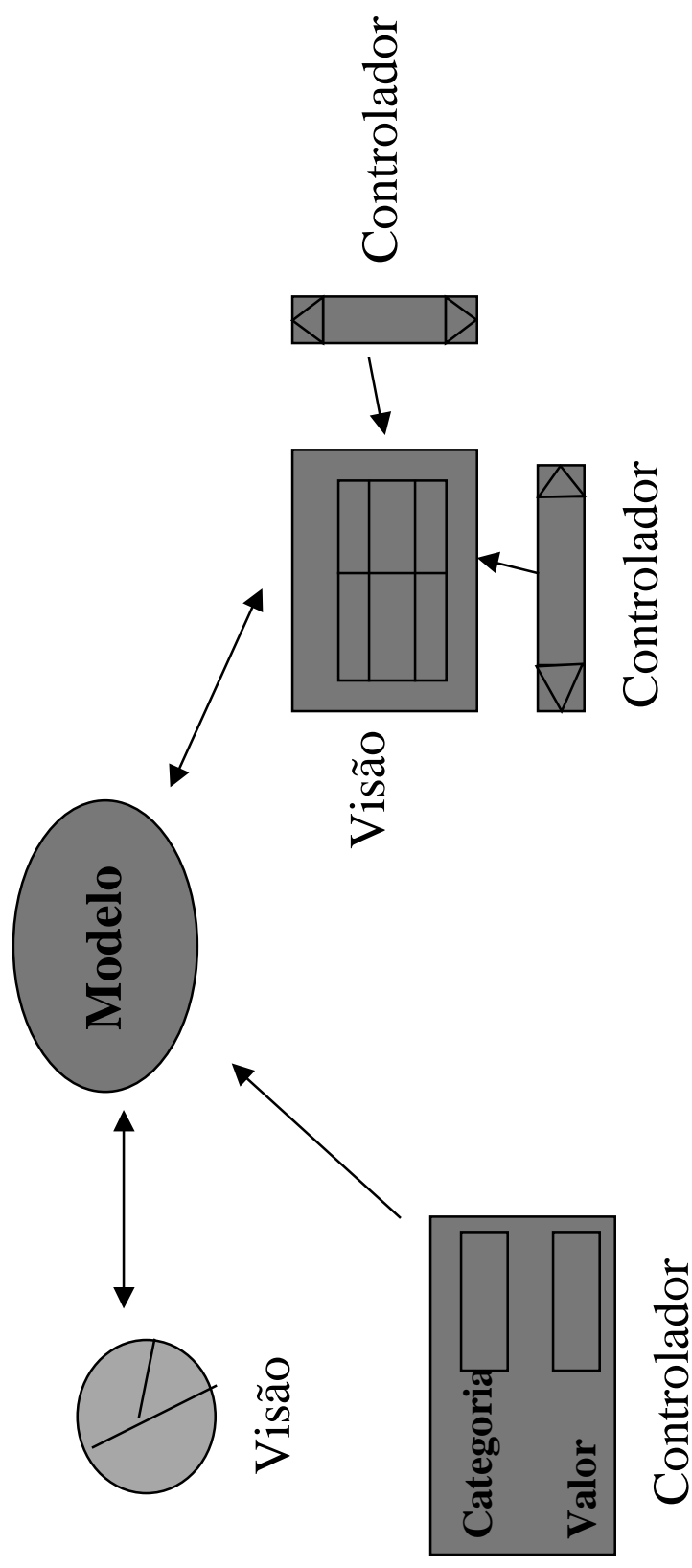
# Framework MVC



•  
•  
•

# MVC - Exemplo

MVC - *Model/View/Controller*



• • • • • • • • • •

•  
•  
•

# Padrões e Frameworks

## *Diferenças*

- ❖ Padrões são mais abstratos que *frameworks*
- ❖ Padrões são elementos de arquitetura menores que *frameworks*
- ❖ Padrões são menos especializados que *frameworks*

• • • • • • • • • •

# Reutilização de Software

# Reutilização de Software

## ◆ Pacotes (Packages)

# ◆ Componentes

**RVI**

•  
•  
•

# Packages

**Packages** são conjuntos de classes relacionadas entre si de forma que ofereçam facilidades uma às outras.

Java traz uma grande quantidade de classes agrupadas em pacotes que estão prontas para serem utilizadas pelo programador.

• • • • • • • • • •



- 
- 
- 

# Packages

## Alguns exemplos de Packages em Java

**java.lang** - base da linguagem Java

(Boolean, Character, Double, Float, Integer, Long, math, Object, String,...)

**java.io** - pacote que permite manipulação de streams lendo ou gravando em arquivos  
(DataInputStream, FileInputStream, FileOutputStream, PrintScreen)

**java.util** - pacote que provê uma miscelânea de classes úteis incluindo estrutura de dados, time, date, geração de números randômicos, etc..

**java.awt** - pacote que provê um conjunto de manipulações de interface para o usuário tais como windows, caixas de diálogo, botões, cores, checkboxes..

**java.applet** - pacote que habilita a criação de applets através da classe Applet e também provê recursos de áudio

•  
•  
•

# Componentes

Um componente é um pedaço de código encapsulado e acessível apenas a partir de sua interface

Um componente possui:

- seu **comportamento externo** (o que ele faz) que é definido na sua especificação
- suas **operações internas** (como ele faz o que se propõe a fazer) que está escondido do mundo externo e pode ser entendido apenas se examinarmos seu código fonte
- seu **executável** (runtime binário .exe .dll)

• • • • • • • • • • •

- # Componentes

# Componentes

## Componentes de negócio são essencialmente objetos.

Cada componente implementa a **lógica de negócio** e as propriedades relativas a uma entidade do mundo real.

O que os distingue dos objetos tradicionais é a capacidade de ser utilizados por aplicações produzidas em **diferentes linguagens e tecnologias**, rodando sobre diferentes sistemas operacionais.

A tecnologia de componentes altera radicalmente a forma como os sistemas de informação são desenvolvidos.

Os componentes podem ser considerados como blocos básicos de construção.

**Para criar um novo sistema, os desenvolvedores apenas combinam componentes.**

•  
•  
•

# *Java Beans* - Componentes Java

Arquitetura Java de **componentes reutilizáveis**  
independente de plataforma.

Um *JavaBean* é um componente de software reutilizável  
que pode ser manipulado visualmente em uma  
ferramenta de construção  
(construtor de páginas Web, construtor visual de aplicações,  
construtor de GUIs)

• • • • • • • • • •

•  
•  
•

# Objetos Distribuídos em Java

O sistema RMI (Remote Method Invocation) permite que um objeto rodando em uma máquina virtual Java (JVM) chame métodos de um objeto que esteja rodando em outra JVM.

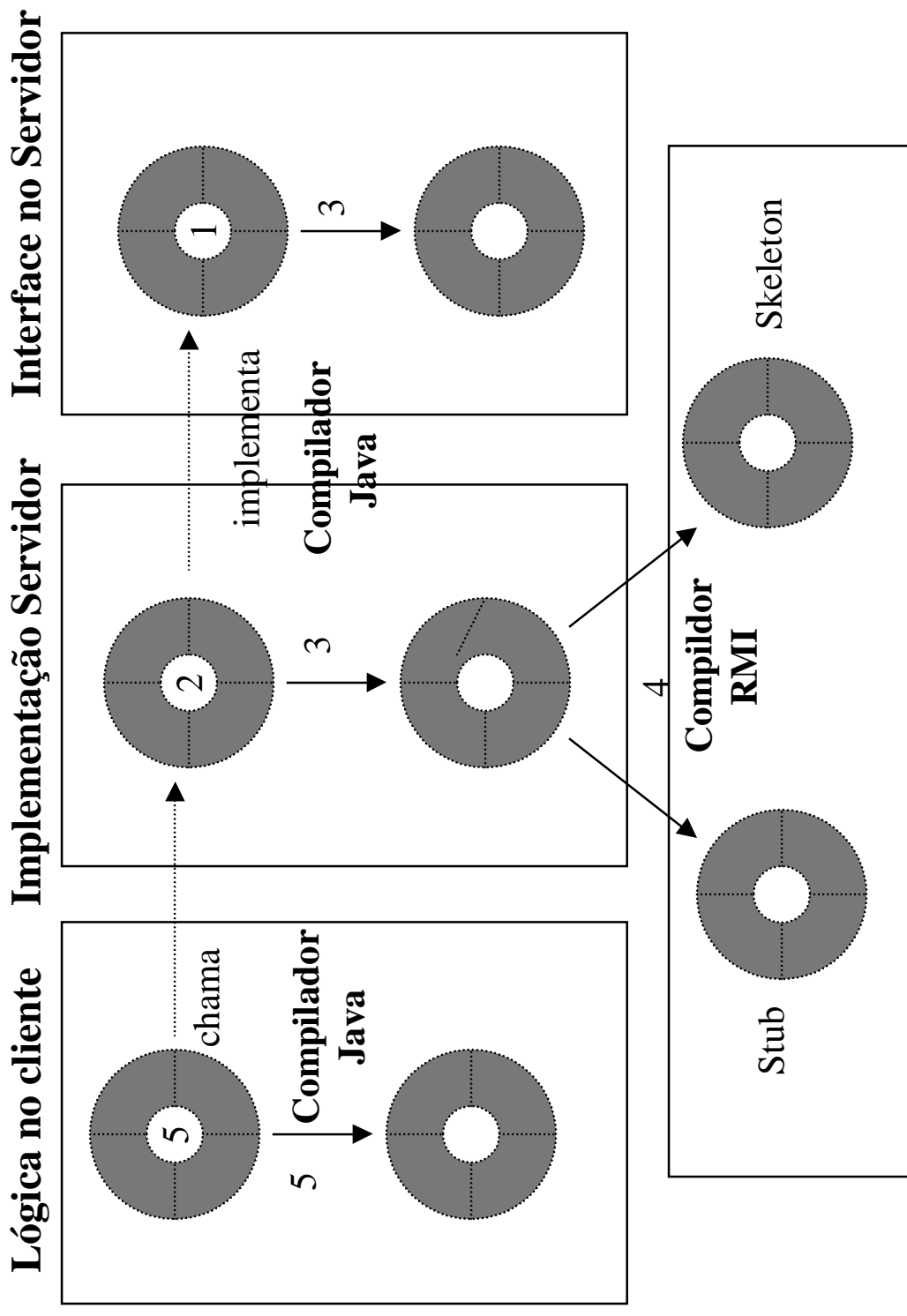
RMI permite a comunicação remota entre programas escritos em Java.

RMI provê um mecanismo através do qual o servidor e o cliente se comunicam e passam informações. Estas aplicações são chamadas de aplicações de **objetos distribuídos**

• • • • • • • • • •

•  
•  
•

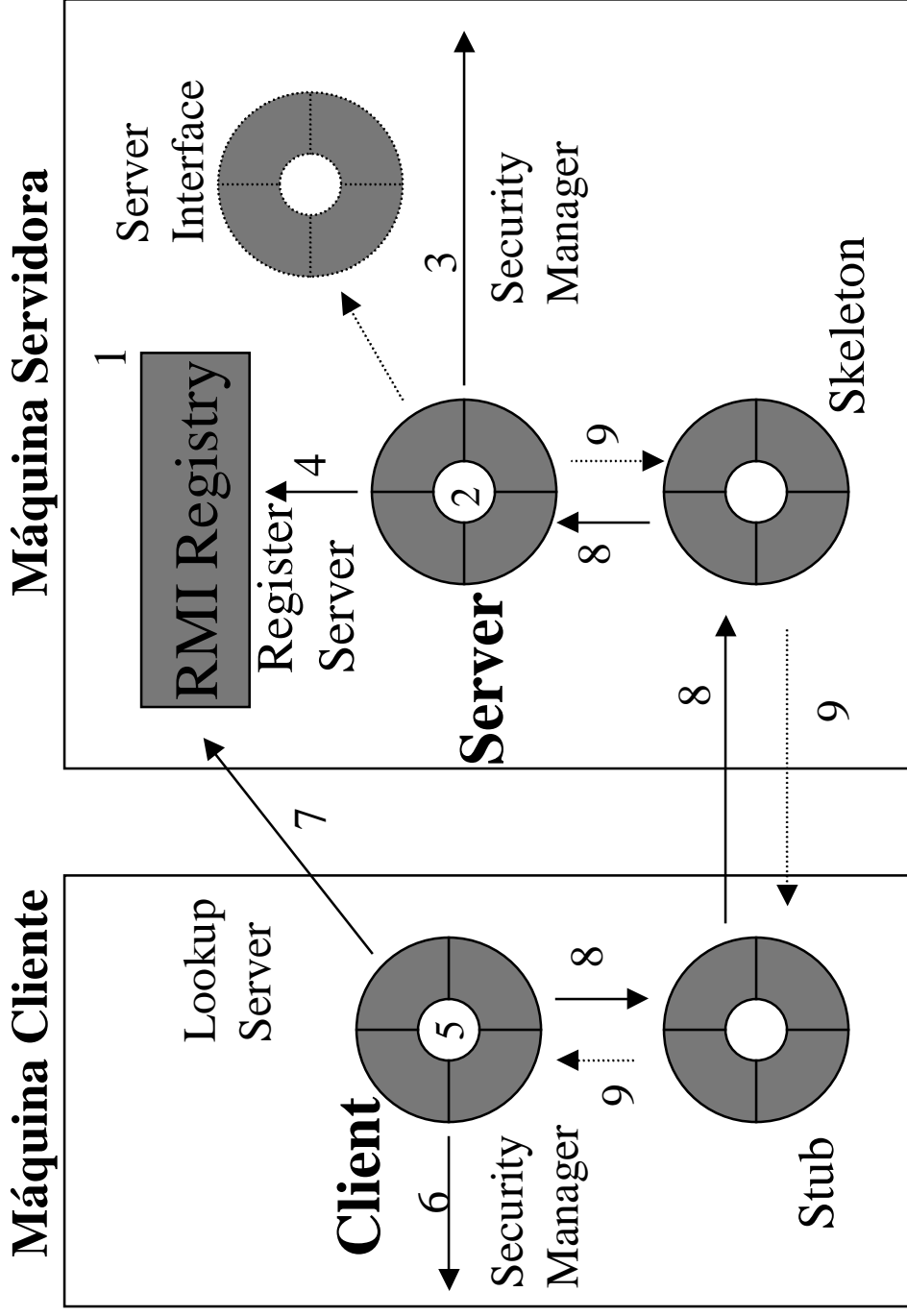
# RMI - Ambiente de Desenvolvimento



• • • • •

•  
•  
•

# RMI - Ambiente de Execução



• • • • •

# Referências

- (1) Análise e Projeto Orientados a Objetos  
Scott W. Ambler (IBPI Press)
- (2) Modelagem e Projetos Baseados em Objetos  
J. Rumbaugh et. al (Campus)
- (3) Princípios de Análise e Projeto Baseados em Objetos  
James Martin (Campus)
- (4) Design Patterns - Elements os Reusable Object-Oriented Software  
Erich Gamma et. al (Addison-Wesley)
- (5) Manual do Visual Age for Java
- (6) Software Engineering - A Practitioner's Approach  
Roger S. Pressman
- (7) Modelagem de Objetos através da UML  
José Davi Furlan (Makron Books)
- (8) Bancos de Dados Orientados a Objetos  
Engênio<sup>a</sup> Nassu, et. Al (Edgard Blücher Ltda)
- (9) Object-Oriented Programming with Java  
Timothy Budd (Editora Addison-Wesley)