# Introdução a Padrões de Projeto

Prof. Marum Simão Filho

# Agenda

- Visão Geral
- Histórico
- Motivação
- Classificações de padrões
- Idiomas
- Anti-padrões
- Fundamentos de Projeto Orientado a Objetos
  - Herança, Agregação, Interfaces, Polimorfismo, Delegação, Encapsulamento, Funcionalidade.

# Existem várias maneiras para se construir uma aplicação

# Nos dias de hoje...

- Construir software é **complicado**
- **Lento** e **Caro**
- Ainda não somos capazes de gerar software **sem erro**
- Novas tecnologias e necessidades
  - Aumentam a **complexidade** dos sistemas
  - Diminuem o número de pessoas **capacitadas**
- Processo e metodologia nem sempre são adequados ou não existem

# Problemas

- Projetos **cancelados**
- Sistemas **não funcionam** como planejado
- **Alto custo** de desenvolvimento e manutenção
- **Baixa qualidade** e **produtividade**
- Os prazos muitas vezes são **estourados**
- É difícil **reutilizar**
- É difícil **gerenciar** a mudança

# uma luz no fim do túnel

Orientação a Objetos

# Fundamentos e Princípios OO

- Abstração
- Encapsulamento
- Polimorfismo
- Herança

Orientação a Objetos

# Conhecer OO != ser bom com OO

- Conhecer os **princípios de OO** não significa dizer que seremos bons na construção de sistemas flexíveis, reutilizáveis e de fácil manutenção

- Sistema OO
  - Possuem propriedades nem sempre óbvias

# Conhecer

- Requer
  - Experiência
  - Trocar experiência

  - Para quê?
  - Sistemas possuem problemas semelhantes e recorrentes

  => Soluções semelhantes

# Padrões de Projeto

uma **SOLUÇÃO**

para um **PROBLEMA**

dentro de um **CONTEXTO**

# Explicando

- **Contexto**
  - Situação à qual um padrão é aplicável
  - Recorrente
- **Problema**
  - Objetivo que está se tentando atingir
  - + Limitações que ocorram no contexto
- **Solução**
  - O que se pretende obter
  - Um design genérico que possa ser aplicado para qualquer pessoa contornar as limitações a atingir um objetivo

# Exemplo

- Problema:
  - **Como faço para não me atrasar para o trabalho?**
- Contexto:
  - **Tranquei as chaves dentro do carro**
- Solução
  - **Quebre a janela, entre no carro, ligue o motor e dirija para o trabalho**

# Definindo...

- Segundo o Aurélio:
  - **Padrão** sm.
    1. Modelo oficial.
    2. Qualquer objeto que serve de modelo à feitura de outro.
  - **Modelo** sm.
    1. Objeto para ser reproduzido por imitação.
    2. Molde.
    3. Coisa cuja imagem serve para estudo, etc.

# Definições Históricas

- "Cada padrão descreve um **problema** que ocorre outra e mais outra vez no nosso ambiente, e então descreve o âmago da **solução** deste problema, de forma que você possa usar esta solução um milhão de vezes, sem fazer o mesmo duas vezes." [Christopher Alexander]

- "Um padrão de projeto sistematicamente **nomeia, motiva e explica um projeto genérico**, que endereça um **problema de projeto recorrente** em sistemas orientados a objetos. Ele descreve o **problema**, a **solução**, quando é **aplicável** e quais as **consequências** de seu uso." [Gamma]

# Definições

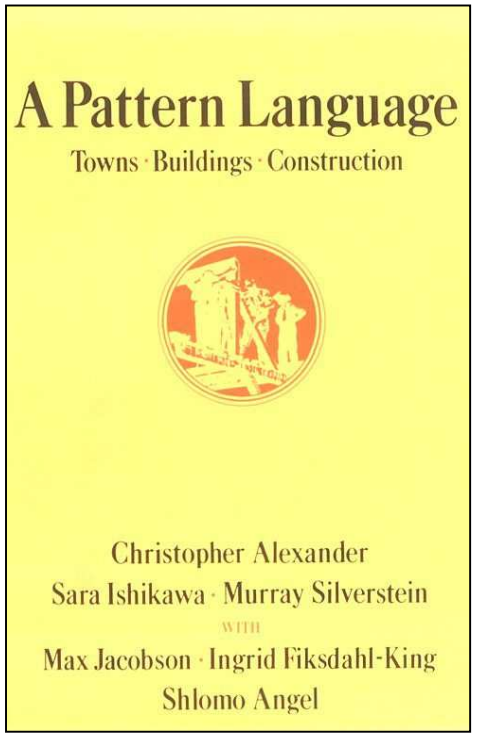
- ***Patterns* ou Padrões** são soluções genéricas e reutilizáveis, aplicáveis em classes de problemas bem conhecidos. Soluções que um dia funcionaram tornam-se receitas para situações similares (desde que estas soluções tenham sido projetadas com flexibilidade).
- **Padrão**
  - Deve ser aplicado a um problema recorrente
  - Não é uma solução que possa ser aplicada diversas vezes
  - Custo

# Breve histórico

- A idéia de armazenar informação sobre padrões observados em um contexto pode ser atribuída ao arquiteto **Christopher Alexander** e foi elaborada no contexto de arquitetura.
- Nos anos 70, a arquitetura era considerada uma disciplina que requeria muita experiência.
- O arquiteto Christopher Alexander e alguns colegas criaram alguns livros com o objetivo de fazer com que pessoas sem muita experiência em arquitetura pudessem criar seus projetos.

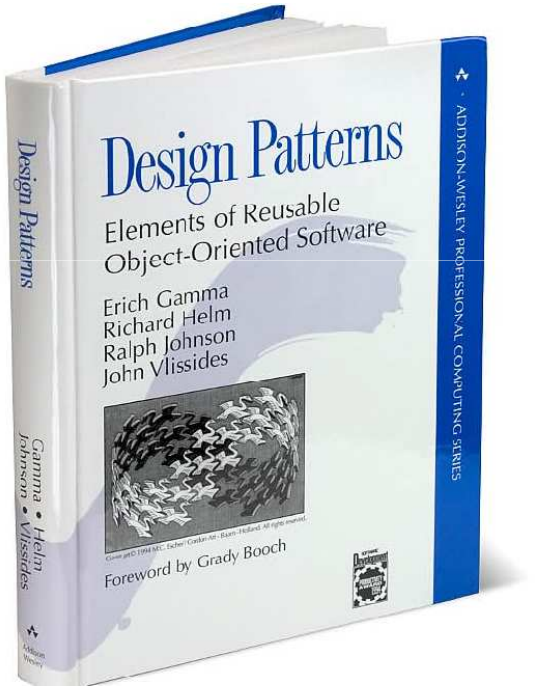
# O Início

- Os livros identificavam similaridades entre bons projetos e seus princípios comuns.
- Eles chamaram essas soluções de ***patterns***.



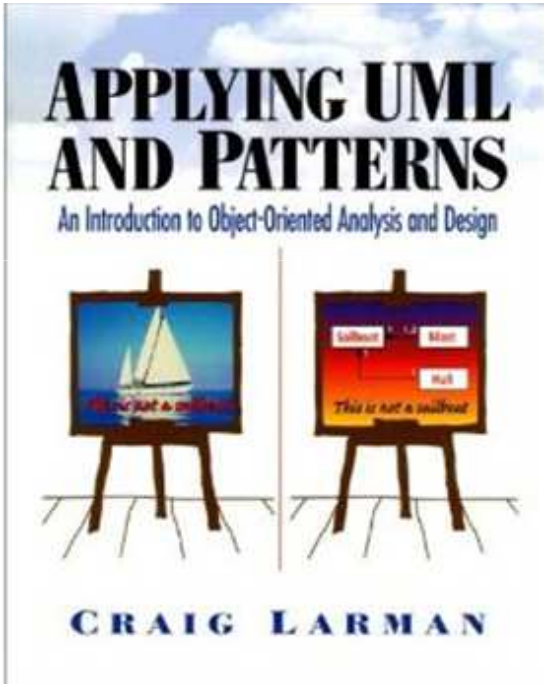
# Padrões GoF

- Em 1994, quatro autores – Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides – publicaram o primeiro catálogo de *Design Patterns* para programas orientado a objetos:
- "*Design Patterns* – Elements of Reusable Object-Oriented Software".
- Os quatro autores acima ficaram conhecidos como **GoF** (*Gang of Four*).



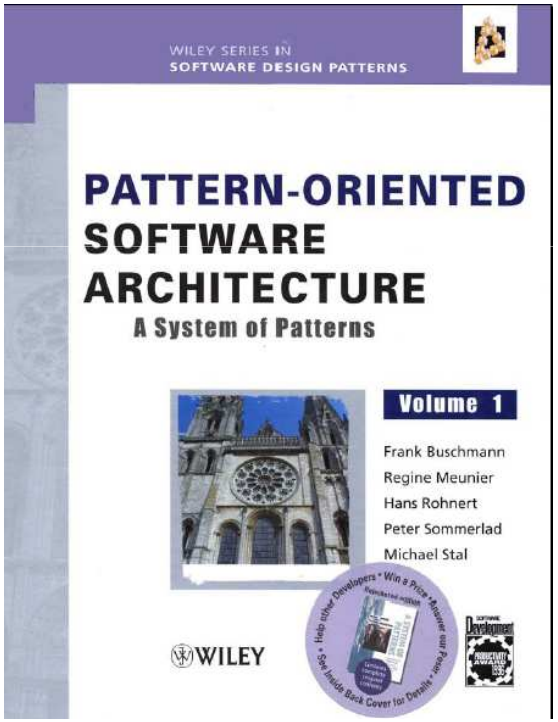
# Padrões GRASP

- **GRASP: G**eneral **R**esponsibility and **A**ssignment **S**oftware **P**atterns
  - Padrões de Software Gerais para Atribuição de Responsabilidade
- Introduzidos por Craig Larman em seu livro "Applying UML and Patterns".
- Descrevem os princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões.
- Ajudam à compreender melhor a utilização de vários do paradigma OO em projetos mais complexos.



# Padrões POSA

- Padrões **POSA** - **P**attern-**O**riented **S**oftware **A**rchitecture: A System of Patterns.
- Buschmann, Frank; Meunier Regine; Rohnert, Hans; Sommerlad, Peter; Stal, Michael.
- Classifica os padrões em 3 categorias:
  - Padrões Arquiteturais
  - Padrões de Projeto
  - Idiomas



# Padrões POSA

- Padrões de Arquiteturais
  - Expressam um esquema de organização estrutural para sistemas de software.
  - Oferecem um conjunto de subsistemas pré-definidos, especifica suas respectivas responsabilidades e inclui regras e diretrizes para organizar as relações entre eles.
  - Considerados padrões de alto nível.
  - Exemplos:
    - Camadas, Pipes e Filtros.

# Padrões POSA

- Padrões de Projeto
  - Oferece um esquema para refinar os subsistemas ou componentes de um sistema de software ou as relações entre eles.
  - São considerados padrões de média escala.
  - Exemplos:
    - Singleton
    - Observer
    - Adapter

# Padrões POSA

- Idiomas
  - Padrão de baixo nível específico de uma linguagem de programação.
  - Mostra como se pode implementar um dado componente/classe ou interação entre componentes/classes usando os recursos de uma linguagem de programação.
  - Exemplos:
    - Singleton em C++ ou em Java
    - Iterator em Java

# Padrões de Arquitetura de Aplicações Corporativas

- Livro de mesmo nome de autoria de **Martin Fowler**.
- Os padrões são voltados para:
  - O desenvolvimento de aplicações corporativas
  - Sobre plataformas multicamadas
  - Desenvolvidas em linguagens orientadas a objetos, tais como Java e .Net.

# Características de um Padrão

- Descrevem e justificam **soluções** para problemas concretos e bem definidos
- **Não são** estratégias de implementação
- Devem ser **comprovados**, isto é, devem ter sido previamente experimentados e testados
- Tratam problemas que ocorrem em **diferentes contextos**
- Descrevem **relações** entre conceitos, estruturas e mecanismos existentes nos sistemas

# Características de um Padrão

- Possuem **pontos fortes** e **pontos fracos**
- Capturam a **evolução** e **aprimoramento** das soluções
- Podem ser utilizados em **conjunto** com outros padrões, compondo linguagens de padrões, *frameworks* ou arquiteturas

# Benefícios

- Facilitam a **reutilização** de projetos e arquiteturas
- Ajudam a **documentar** a arquitetura de software
- Introduzem um **vocabulário comum**
- Propiciam compartilhamento de **experiência** entre profissionais
- Permitem que os desenvolvedores concentrem seus esforços nos **aspectos inéditos** do problema

# Benefícios

- Proporcionam uma **solução comprovada**
- Ajudam a **minimizar os erros**
  - Reduz riscos
- Possibilitam a **redução de prazos**

# Elementos essenciais de um padrão

- **Nome**
  - A identificação importante
  - Vocabulário
- **Problema**
  - Quando aplicar o padrão
  - Classe de problemas em que pode ser aplicado e seu contexto

# Elementos essenciais de um padrão

- **Solução**
  - Elementos que fazem parte do design
  - Relacionamento
  - Responsabilidades
  - Colaborações

- **Consequências**
  - Resultados
  - Efeitos

# Catálogo de Padrões

- Descreve
  - Padrões :P
  - Relacionamentos
- Sub-divisão em categorias
- Exibidos em um formato específico
- Design das classes
- Como implementar
- Exemplo

**Nome** – apresenta o padrão de forma sucinta

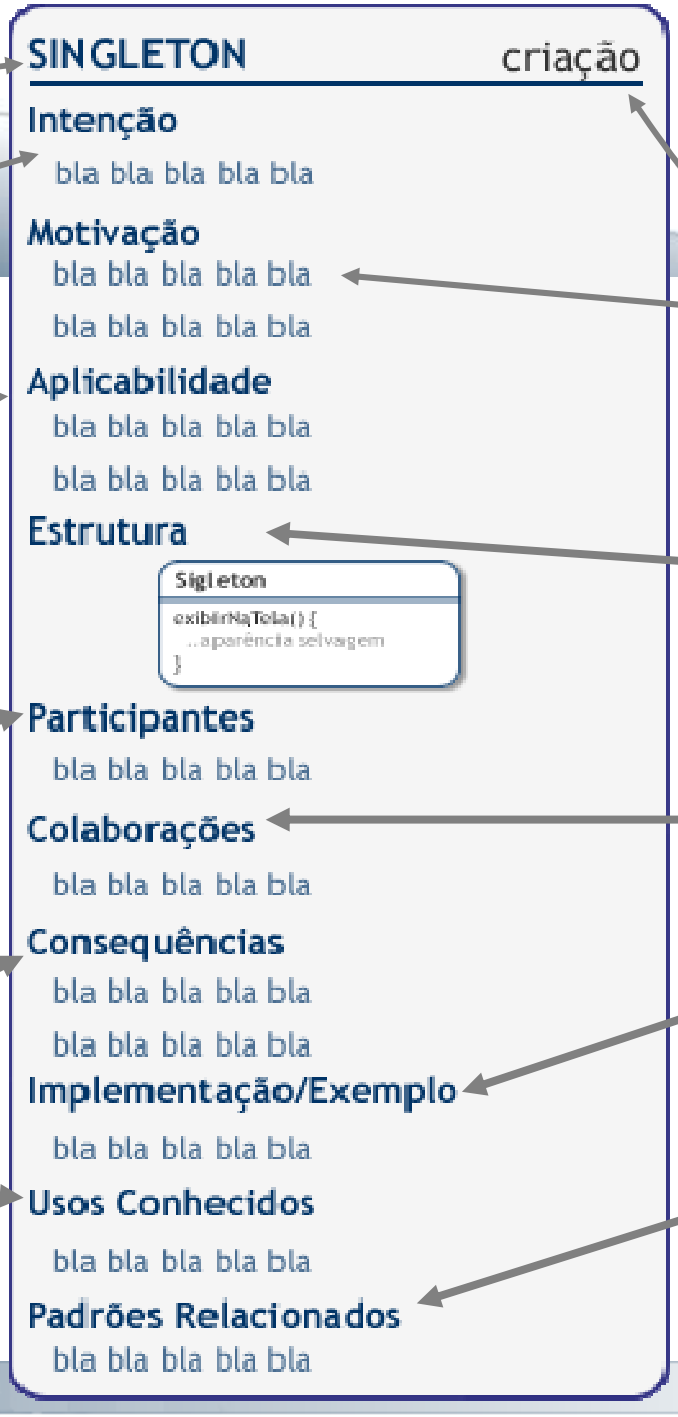
**Intenção** – O que o padrão faz. A descrição.

**Aplicabilidade** – Situações em que o padrão pode ser utilizado

**Participantes** – Classes e objetos existentes no design. Descreve responsabilidades e papéis que desempenham no padrão.

**Conseqüências** – Efeitos – positivos e negativos – que o uso do padrão pode gerar.

**Usos conhecidos** – Exemplos do uso em sistemas reais




**Classificação** – agrupa o padrão de acordo com algumas categorias

**Motivação** – cenário concreto, descrevendo o problema e como ele foi equacionado pela solução

**Estrutura** – Fornece um diagrama que ilustra a relação entre as classes participantes do padrão

**Colaborações** – Informa como os participantes interagem no padrão

**Exemplo** – Trecho ou bloco de código que podem ser úteis na implementação

**Relacionados** – Relacionamento deste com outros

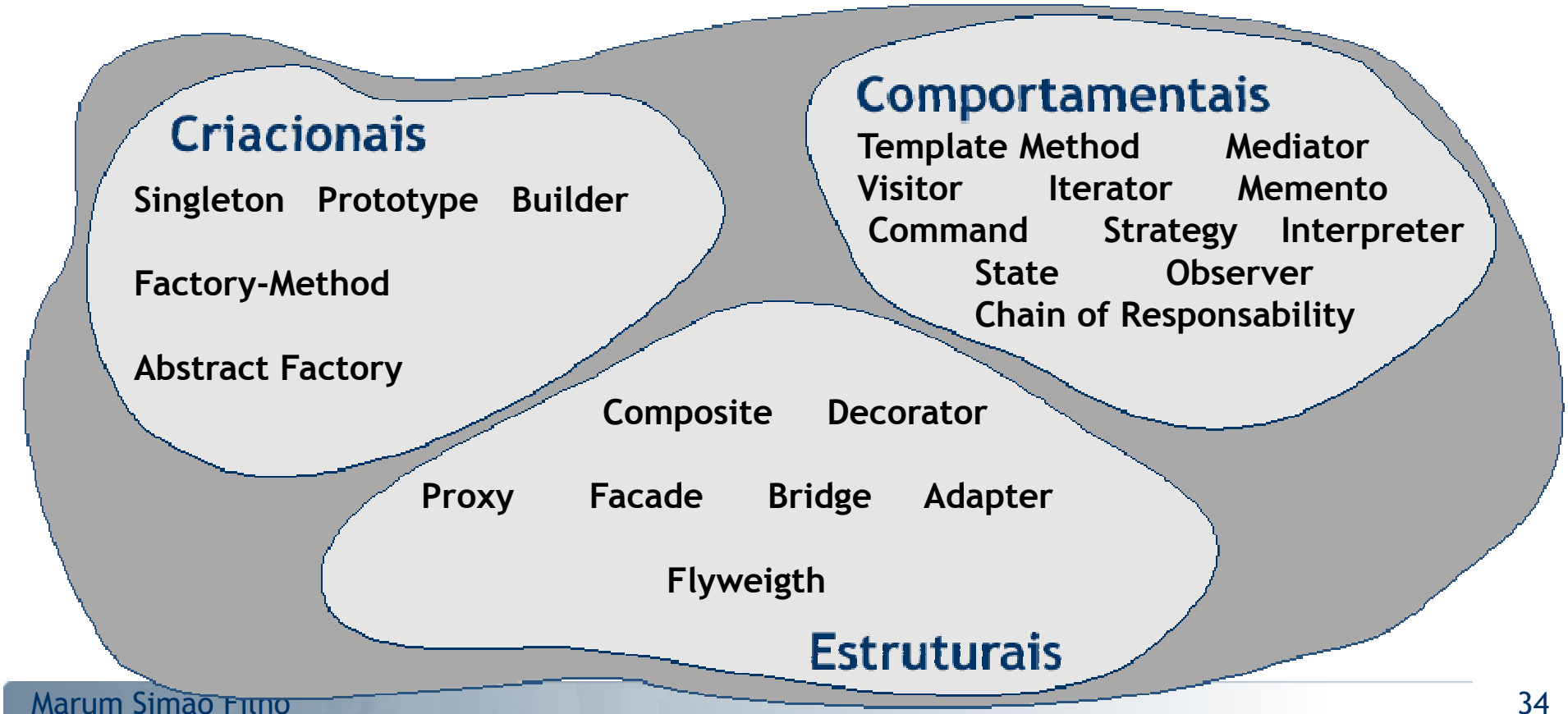
# Organizando Padrões

- Classificação em categorias para facilitar a organização

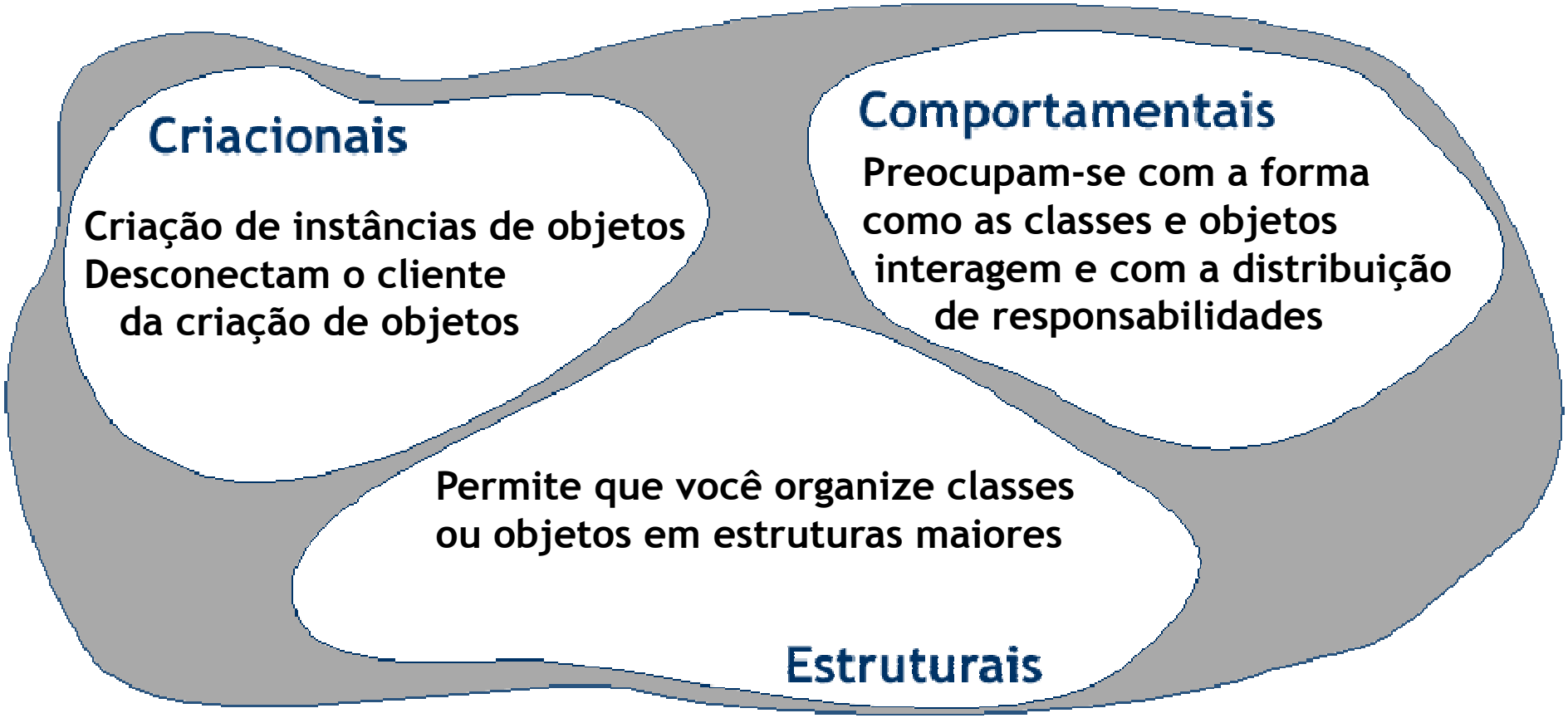
- Classificações mais conhecidas:
  - Finalidade
  - Escopo

# Organizando Padrões finalidade

- Finalidade
  - Refletem o que eles fazem

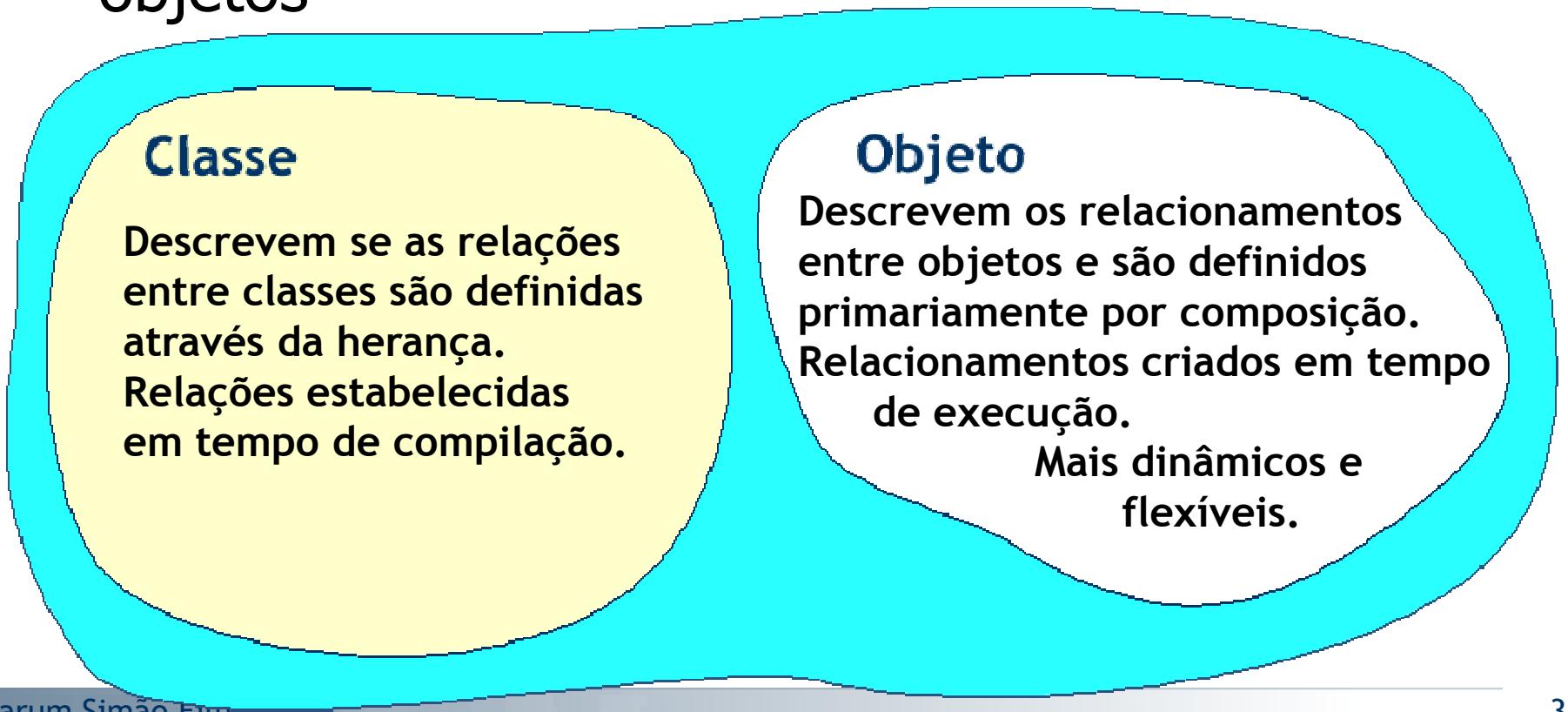


# Organizando Padrões finalidade

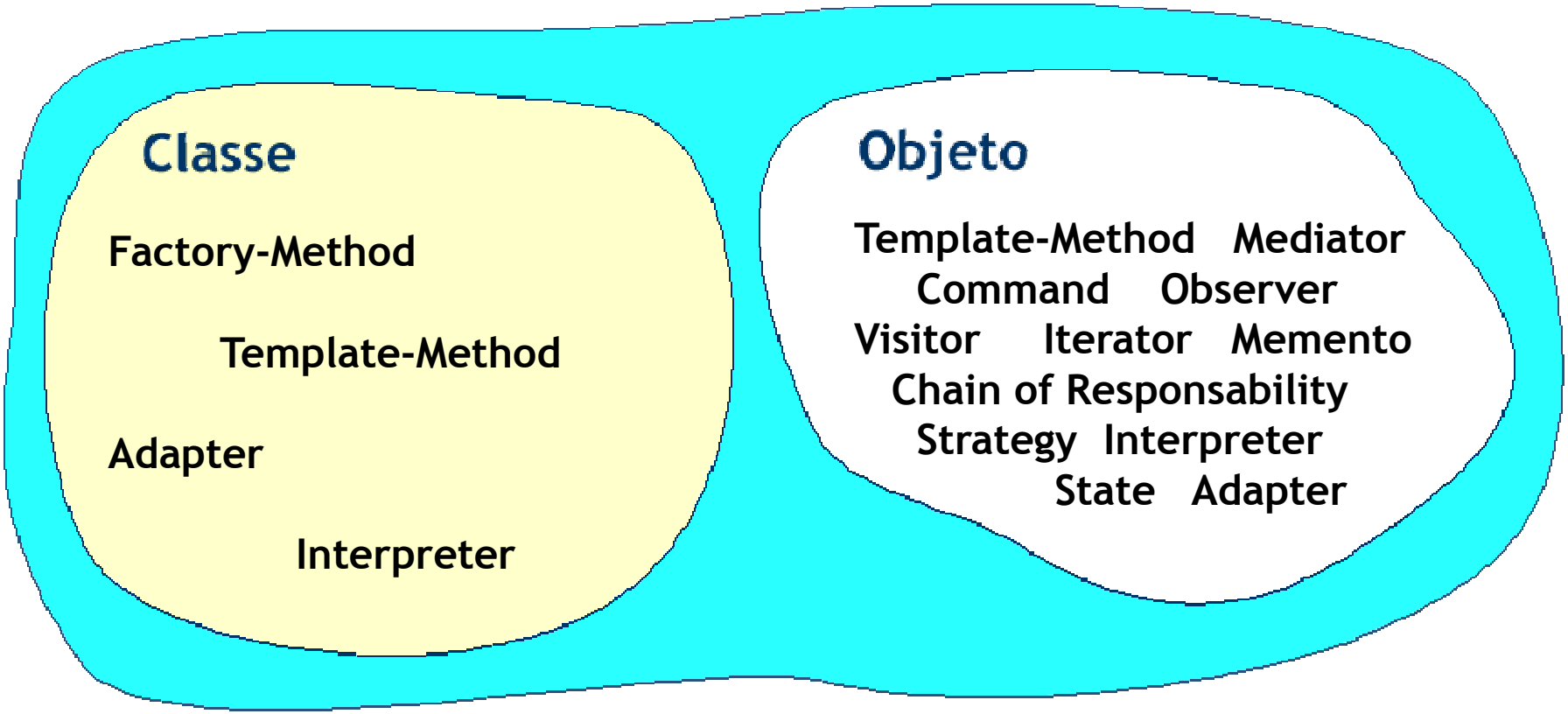


# Organizando Padrões
## escopo

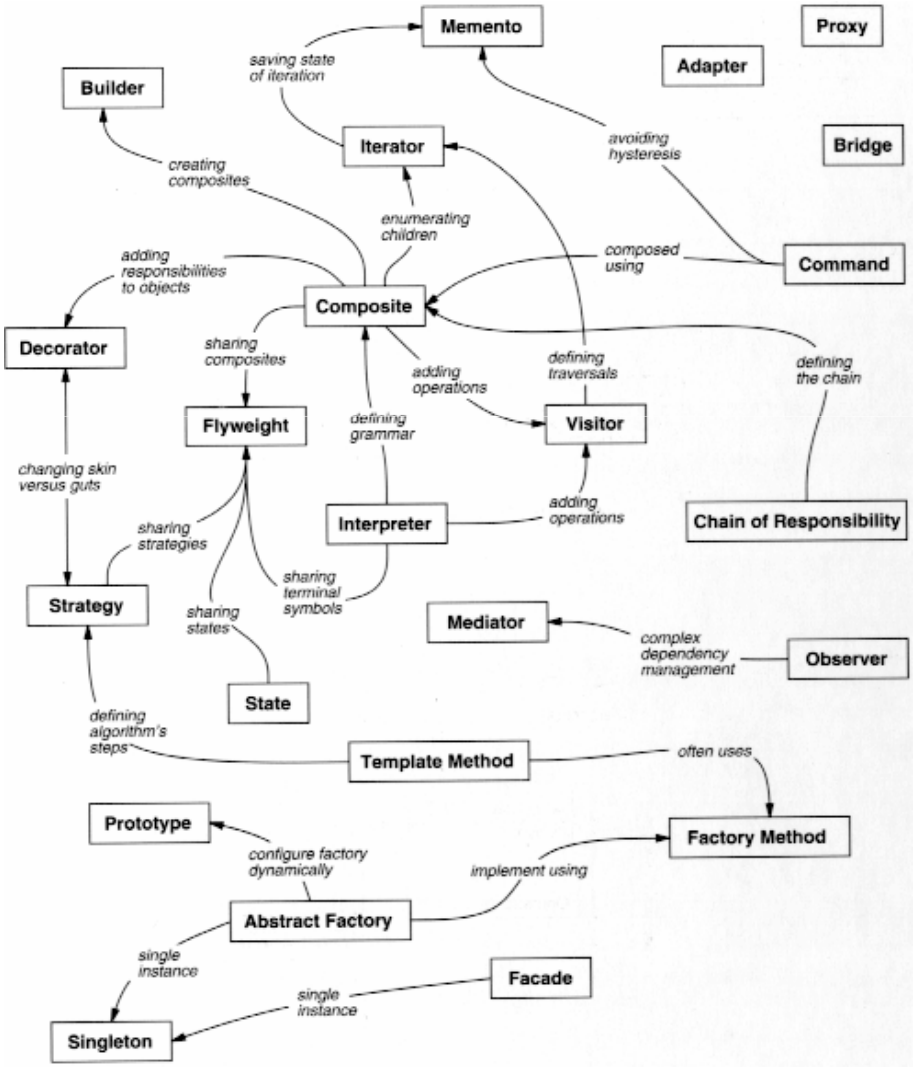
- Escopo
  - Se o padrão se aplica primariamente a classes ou objetos



# Organizando Padrões
## escopo



# Relacionamentos entre os 23 padrões "clássicos"

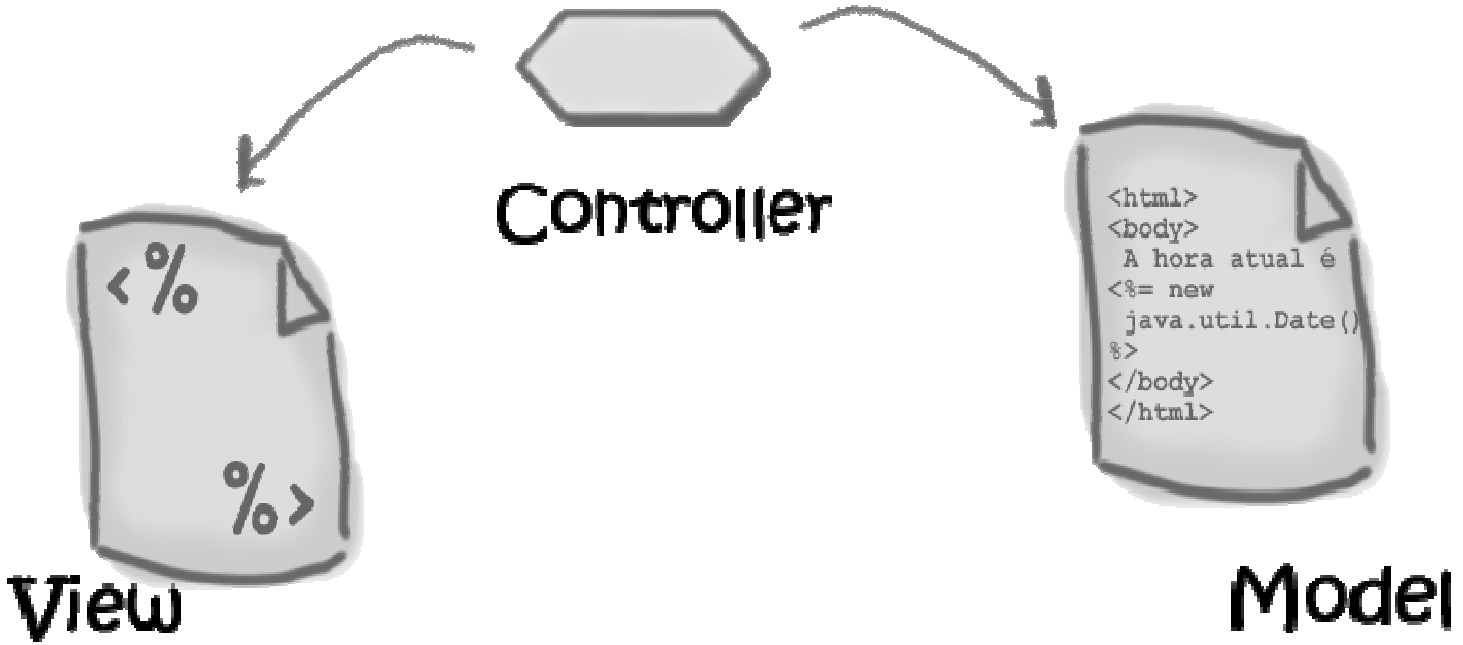


# Padrões Arquiteturais

- Arquitetura - Estilo geral de um sistema
  - Particionamento Físico
  - Considerações sobre a Infra-estrutura
    - Software
- Idéia **fundamental** do esquema da **organização** do software
- Soluções **eficientes** e **elegantes** para problemas comuns de projeto de software

# Padrões Arquiteturais

- Mais conhecidos
  - MVC (Model-View-Controller)



# Padrões Arquiteturais

- Mais conhecidos
  - Camadas
  - Pipes e Filtros
  - Broker

# Mais um pouco...

- **Linguagem de Padrões**
  - Coleção estruturada de padrões
  - Construídos a partir de outros padrões
  - Transformam necessidades e restrições em uma arquitetura
- **Sistema de Padrões**
  - Conjunto coesivo de padrões relacionados
  - Trabalham em conjunto para auxiliar na construção e evolução de arquiteturas

# Mais um pouco...

- **Padrões de Análise**
  - Listam os problemas comuns encontrados em um domínio de negócio particular e maneiras pelas quais esses problemas podem ser modelados
- **Idiomas**
  - Padrões de baixo nível
  - Específicos para uma linguagem
  - Como implementar aspectos particulares de cada componente

# Mais um pouco...

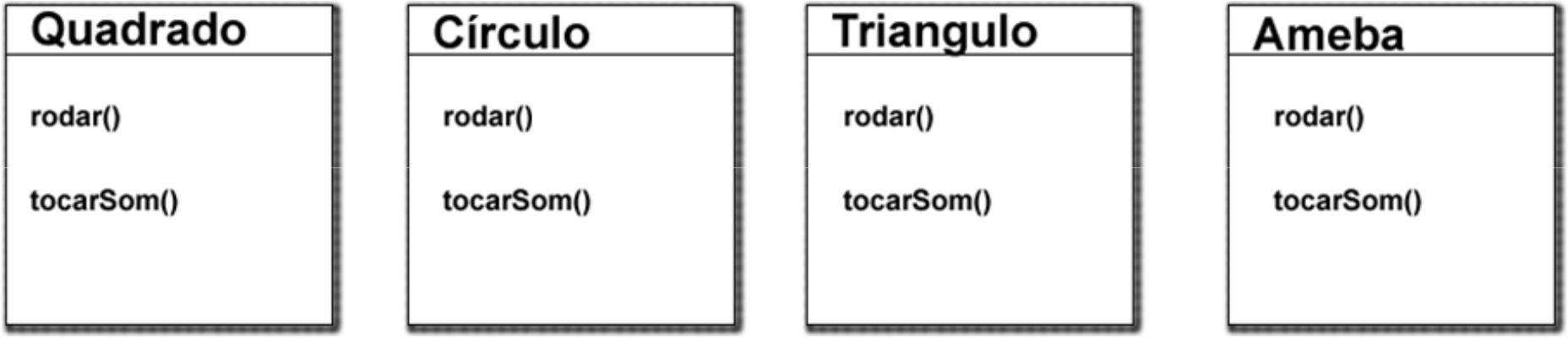
- **Anti-padrões**
  - Soluções PÉSSIMAS adotadas em projetos
  - Documentação de más práticas
  - Indicações de como NÃO solucionar problemas
  - Diversas categorias

  - :D → Pode ser uma solução atraente
  - Explica porque é uma solução ruim a longo prazo
  - Sugere outros padrões aplicáveis

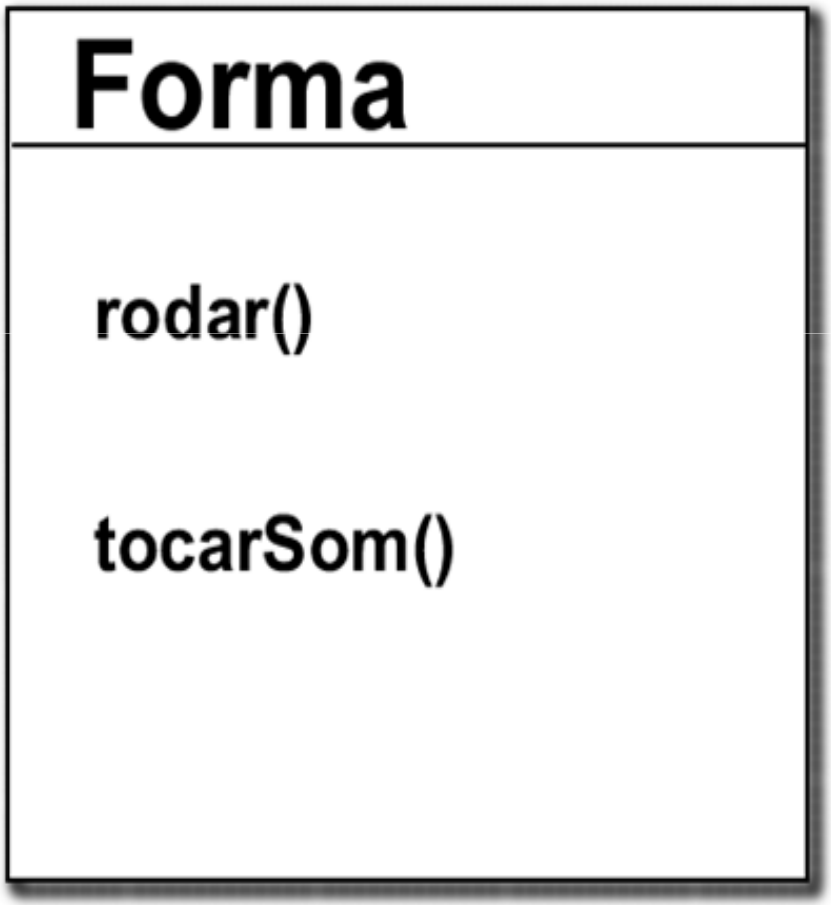
# Preparando

- Herança
- Interfaces
- Agregação
- Polifomorfismo

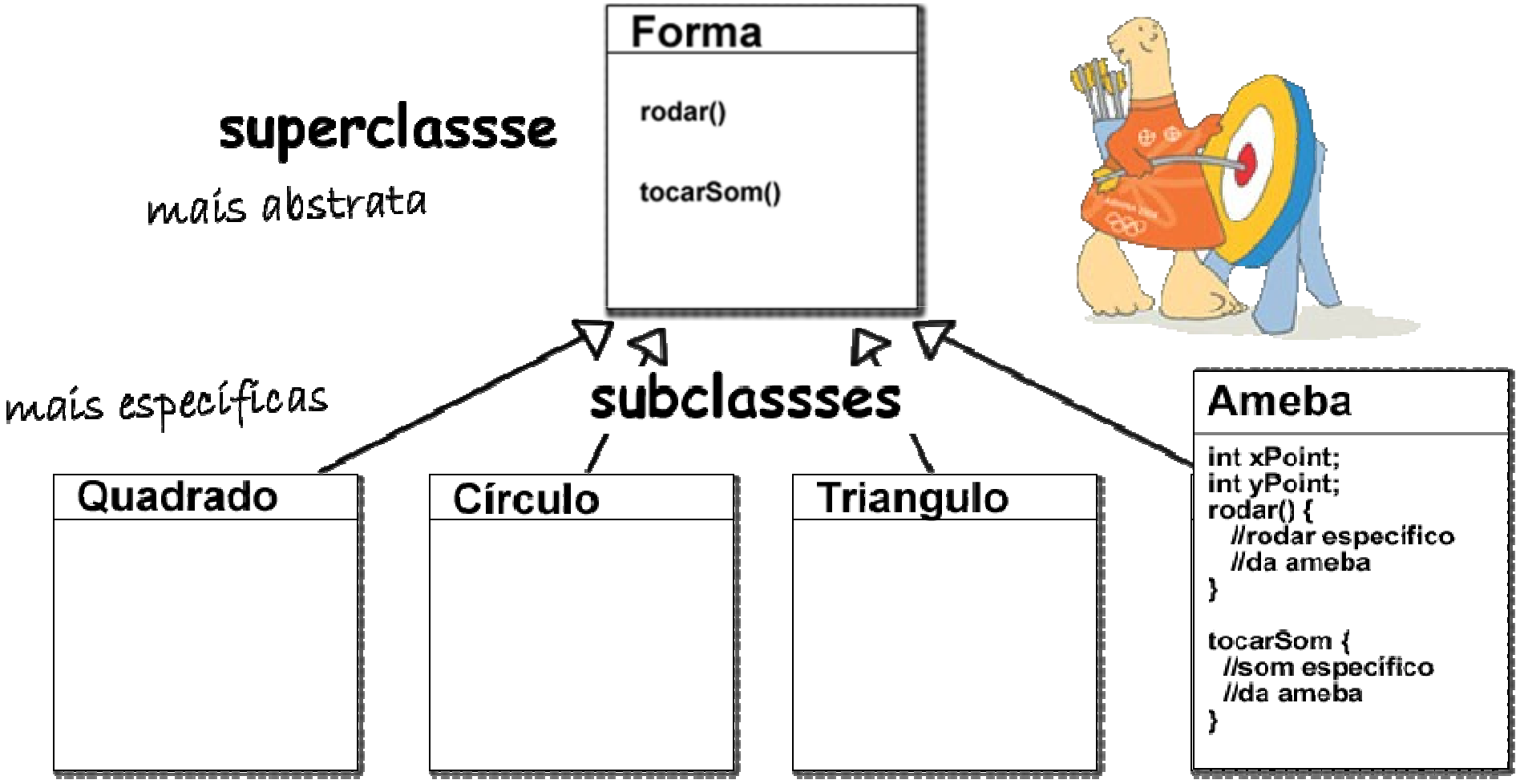

# Certas classes possuem **propriedades comuns**



# **Abstraio** as características comuns e as ponha numa nova classe Forma



# 3 Então eu ligo as outras 4 classes à classe Forma, criando a herança



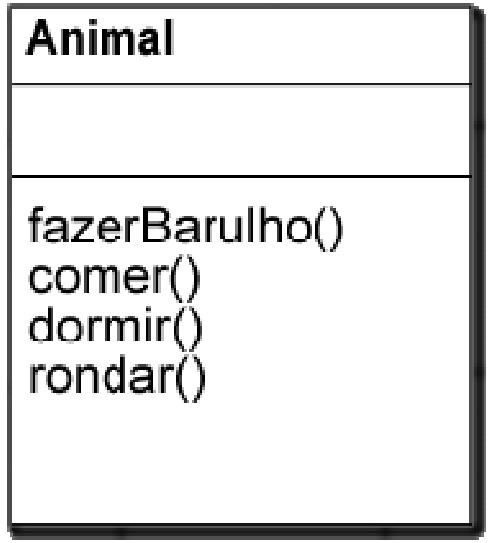
# O que toda essa herança faz por você???

1. **Você evita duplicar código**
   - Ponha o código comum em apenas em um lugar e faça as subclasses herdarem aquele código da superclasse. Quando você quiser mudar tal comportamento, você tem que mudar apenas em um lugar
2. **Você define um protocolo comum para um grupo de classes**

# estabelecendo O CONTRATO

# Estabelecendo um Contrato

- **A herança faz você garantir que todas as classes agrupadas sob certo supertipo têm todos os métodos que aquele supertipo tem.***
- Você diz que qualquer animal pode fazer estas 4 coisas



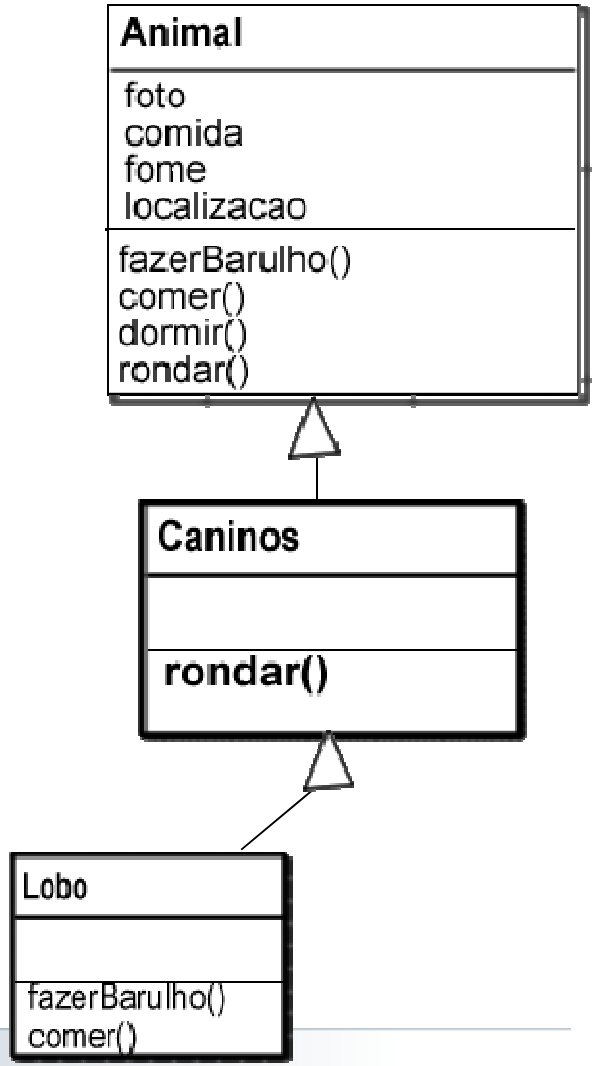
# Usando "É UM" e "TEM UM"

- Quando uma classe herda de outra, dizemos que ela estende a superclasse
- Use o teste "É UM?" para saber se uma classe deve estender de outra
- Triângulo É UMA Forma
- Gato É UM Felino
- Cirurgião É UM médico

# O teste deve funcionar na hierarquia toda

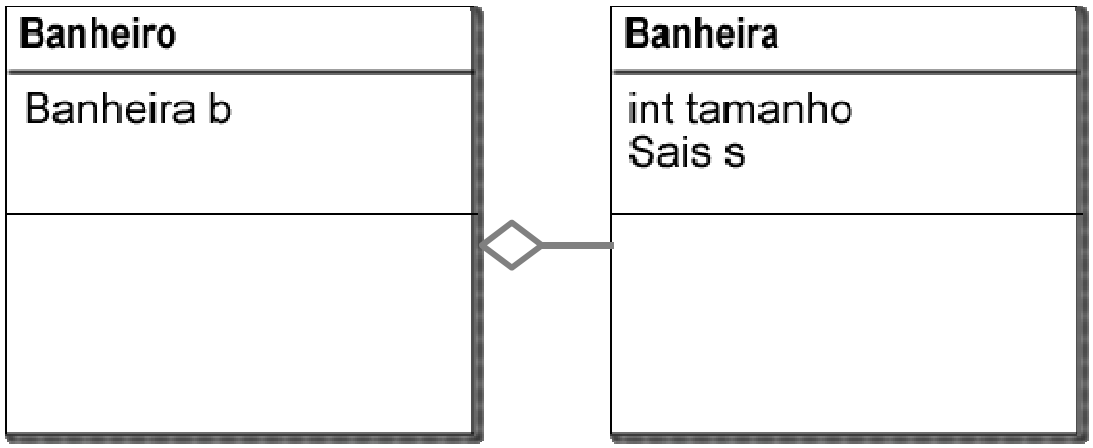
- Canino estende de Animal
- Lobo estende de Canino
- Lobo estende de Animal

- Canino É UM Animal
- Lobo É UM Canino
- Lobo É UM Animal



# Usando "É UM" e "TEM UM"

- Banheira estende de Banheiro?
- Aplique o teste "É UM"
- E o contrário?
- Banheiro TEM UMA Banheira
  - Agregação




# Polimorfismo

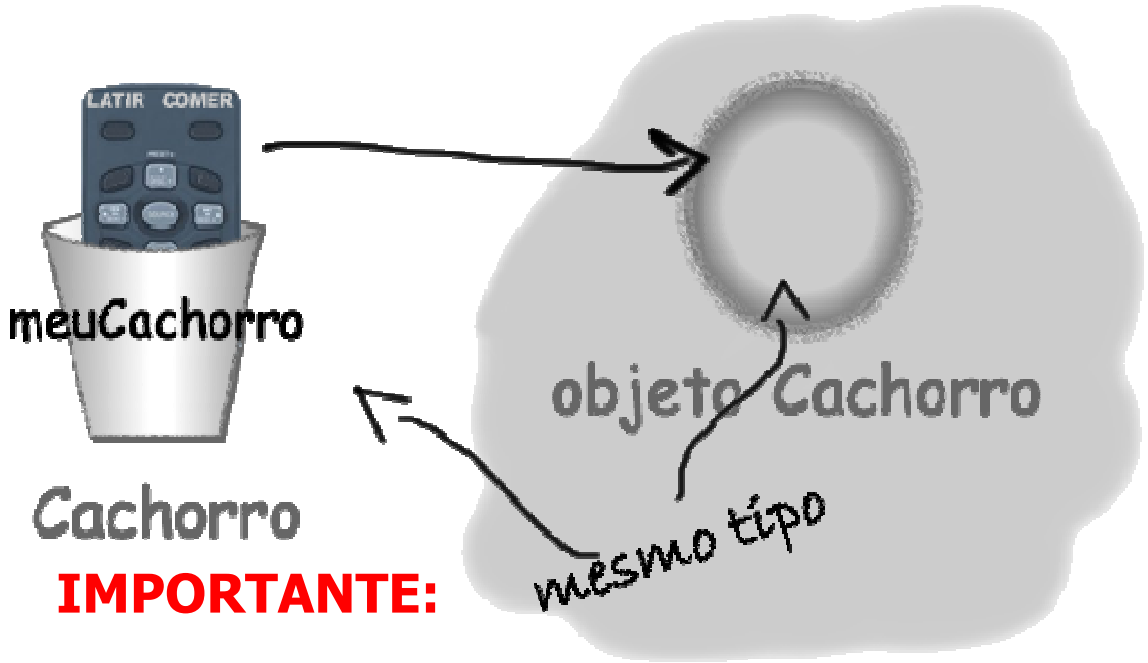
- Quando você define um supertipo para um grupo de classes, **qualquer** subclasse daquele supertipo pode substituí-la onde o supertipo é esperado
- Flexibilidade
- Código mais limpo
- Mais fácil de desenvolver estender

# funcionamento do POLIMORFISMO

- **Os 3 passos para declaração e criação do objeto**

```
Cachorro meuCachorro = new Cachorro();
```

1 2 3

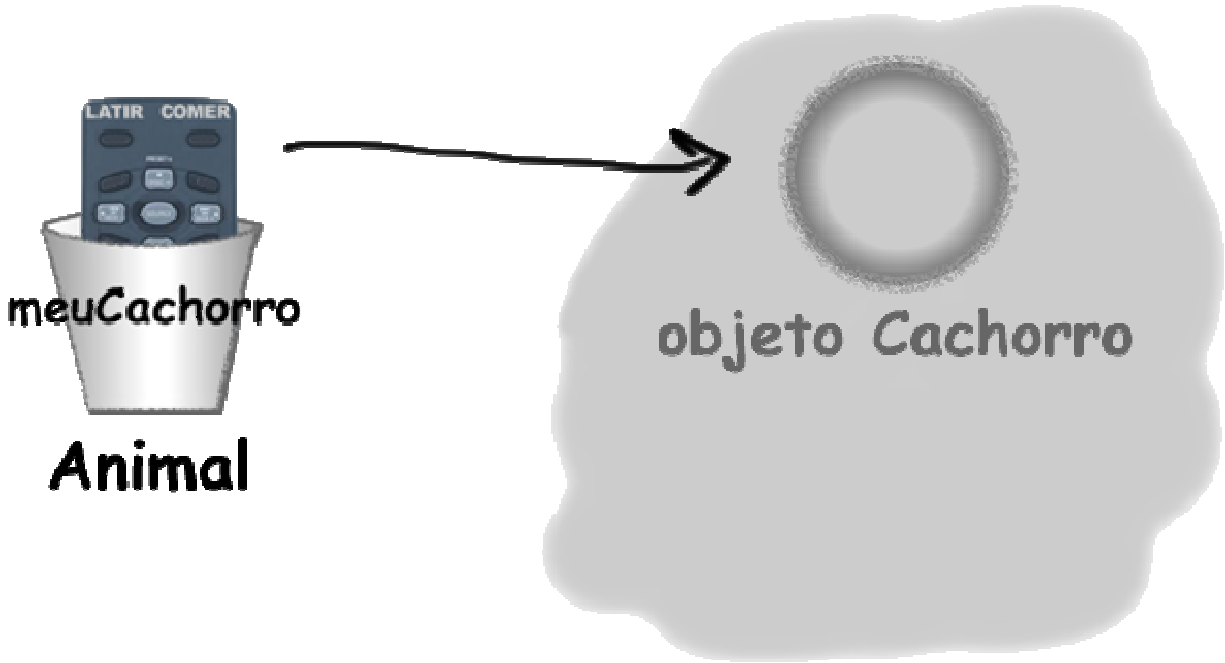



**IMPORTANTE:**

# funcionamento do POLIMORFISMO

- Com polimorfismo, a referência e o objeto podem ser diferentes

```
Animal meuCachorro = new Cachorro();
```



# funcionamento do POLIMORFISMO

- **Com polimorfismo, a referência do objeto pode ser uma superclasse do tipo do objeto atual**
  - TESTE É-UM
  - Qualquer coisa que *extends* o tipo da variável de referência declarada pode ser atribuída à variável

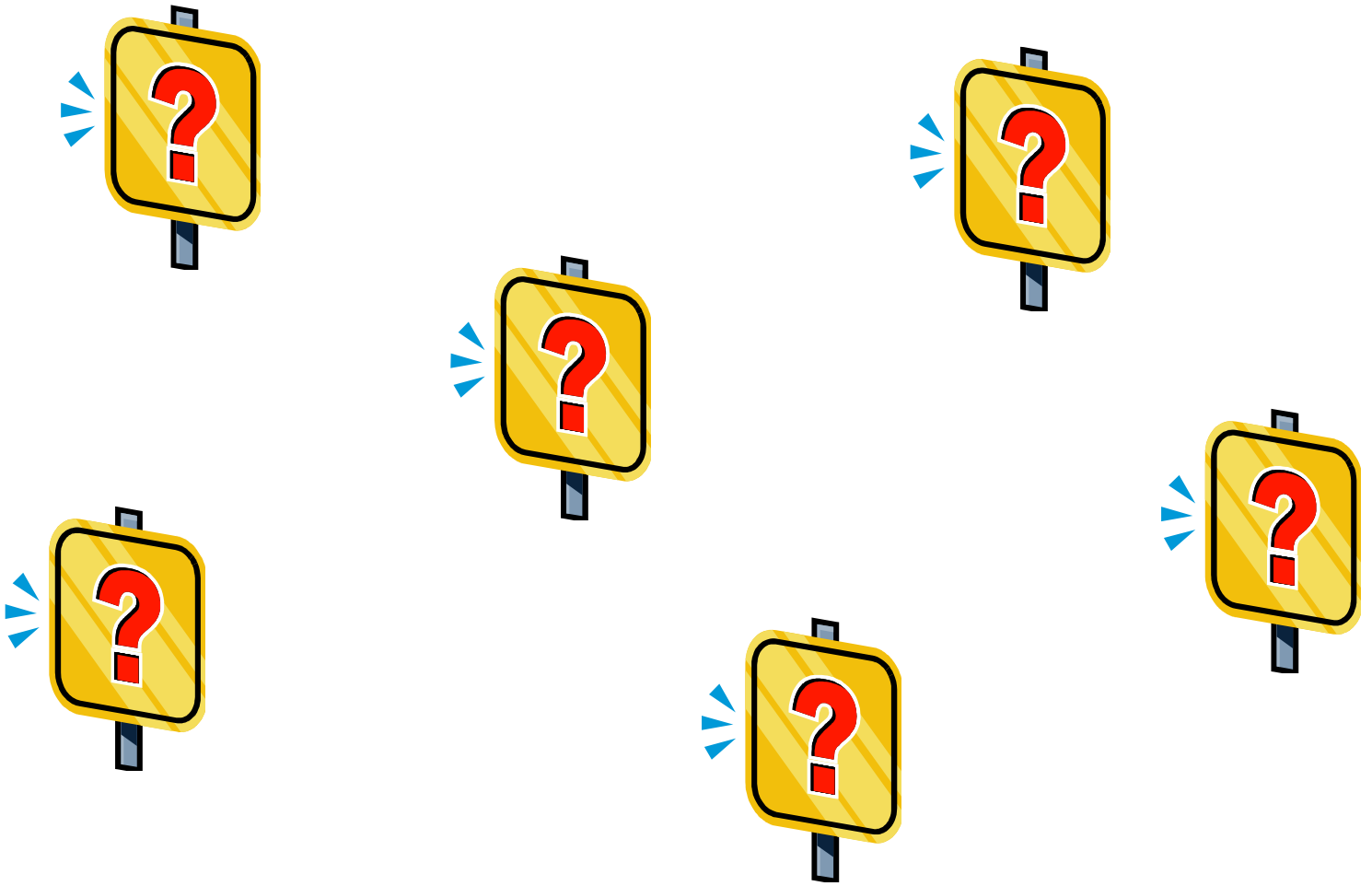
# Polimorfismo

- **Com Polimorfismo, você pode escrever código que não precisa mudar quando você introduz uma nova subclasse no programa**

# Interfaces

- **Interfaces**
  - Não é uma tela
  - Os benefícios do polimorfismo
  - Todos os métodos são abstratos
  - Subclasses devem implementar todos os métodos

# padrões



# *Obrigado!!!*

**Agradecimentos:**
Prof. Eduardo Mendes
Prof. Régis Simão

**Faculdade 7 de Setembro**