

---

# Métodos de Programação Avançado

Padrões GOF

Dr<sup>a</sup>. Alana Morais

# Padrões GoF

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

## Padrões de Criação

- ✓ Se preocupa em como os objetos são criados e instanciados no projeto.
- ✓ Ajuda a tornar o sistema independente da forma como seus objetos são criados, compostos e representados.

# Padrões GoF – Padrões de Criação

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

- 
- “Definir uma interface para criar um objeto mas deixar que subclasses decidam que classe instanciar. *Factory Method* permite que uma classe delegue a responsabilidade de instanciamento às subclasses.” [Gof]

## Factory Method

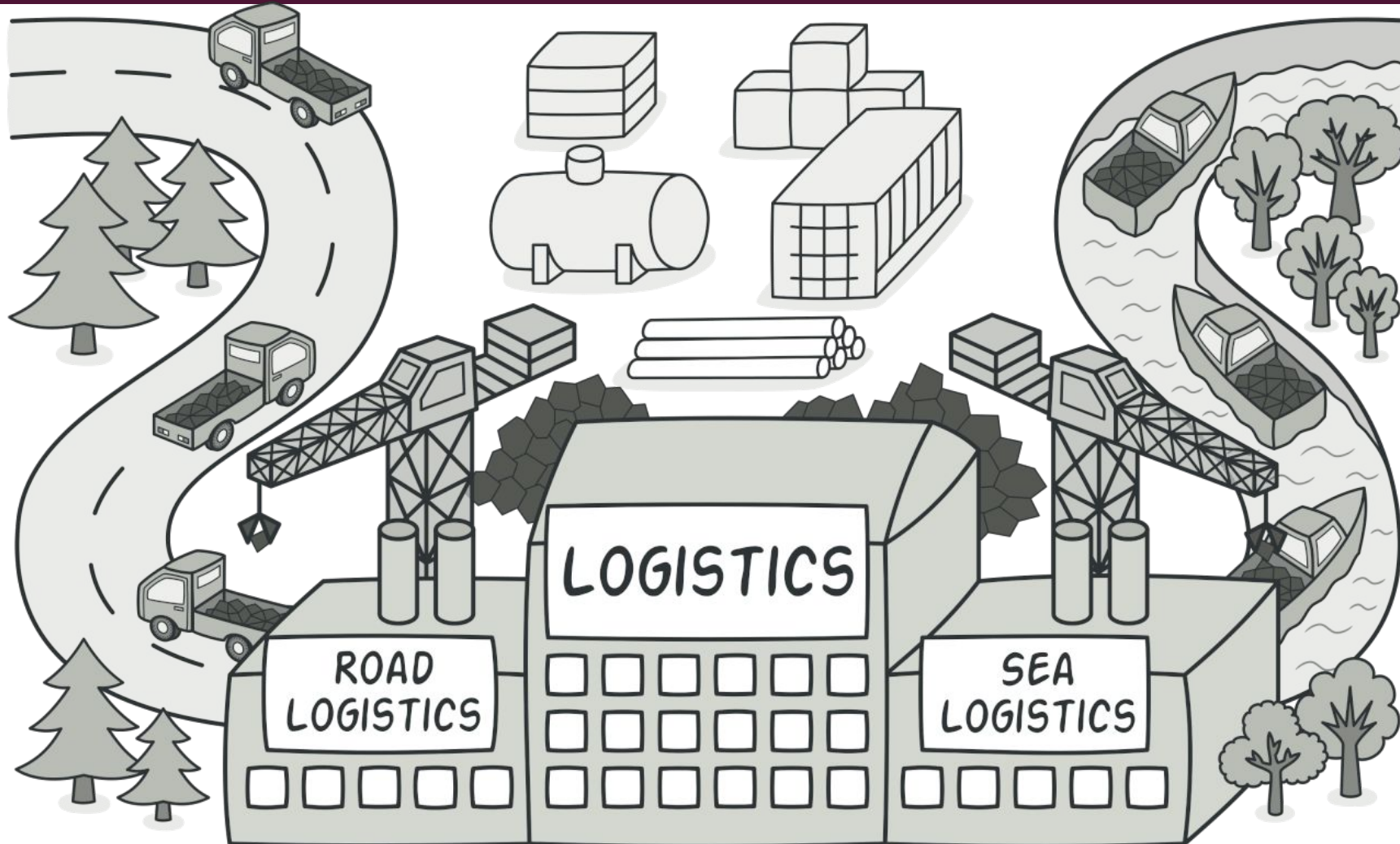


## Factory Method Problema

Uma classe não  
pode antecipar a  
classe de objetos  
que ela deve criar

Uma classe quer  
que suas subclasses  
especifiquem os  
objetos que ela cria

# Factory Method Problema

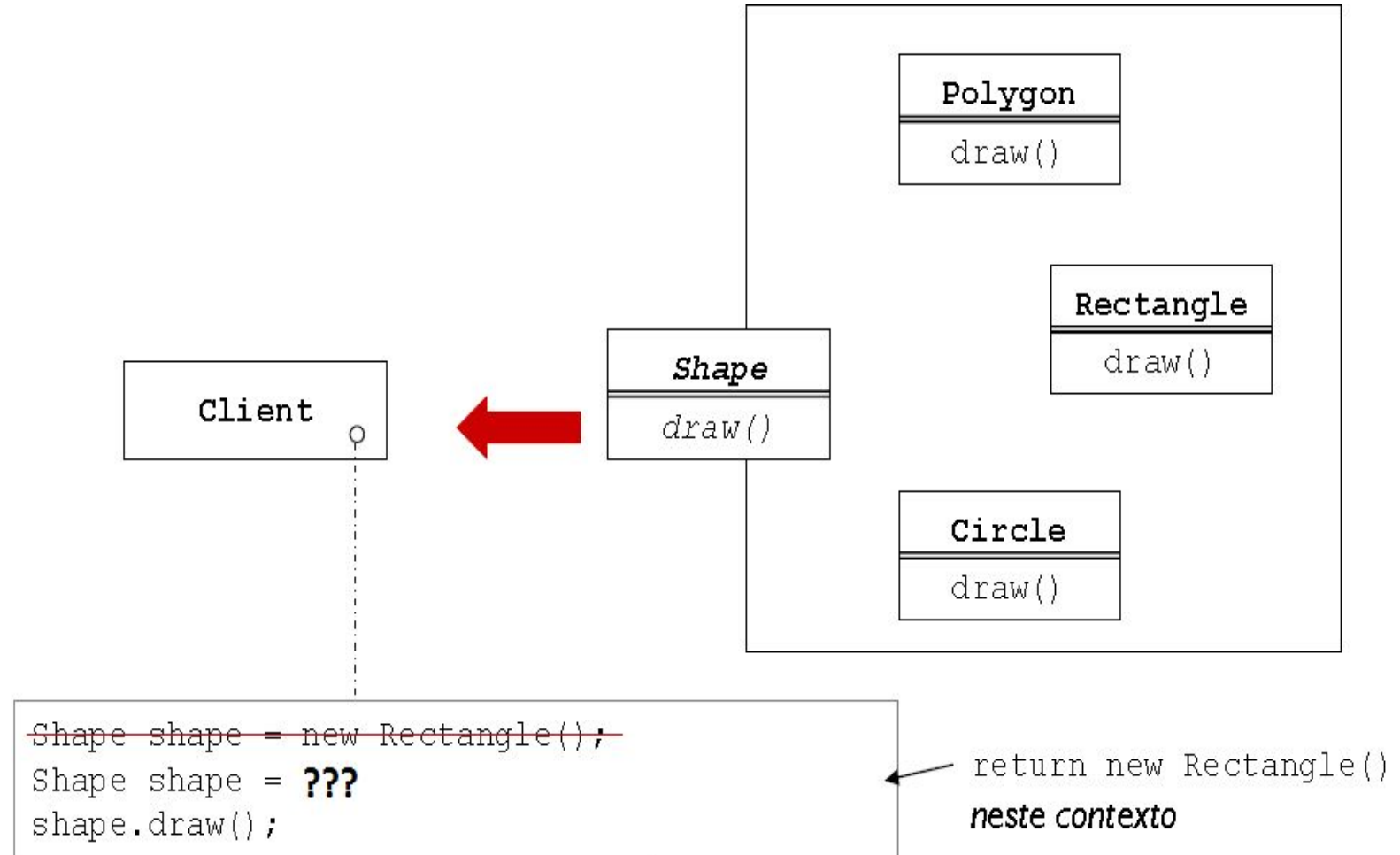




# Factory Method

## Exemplo

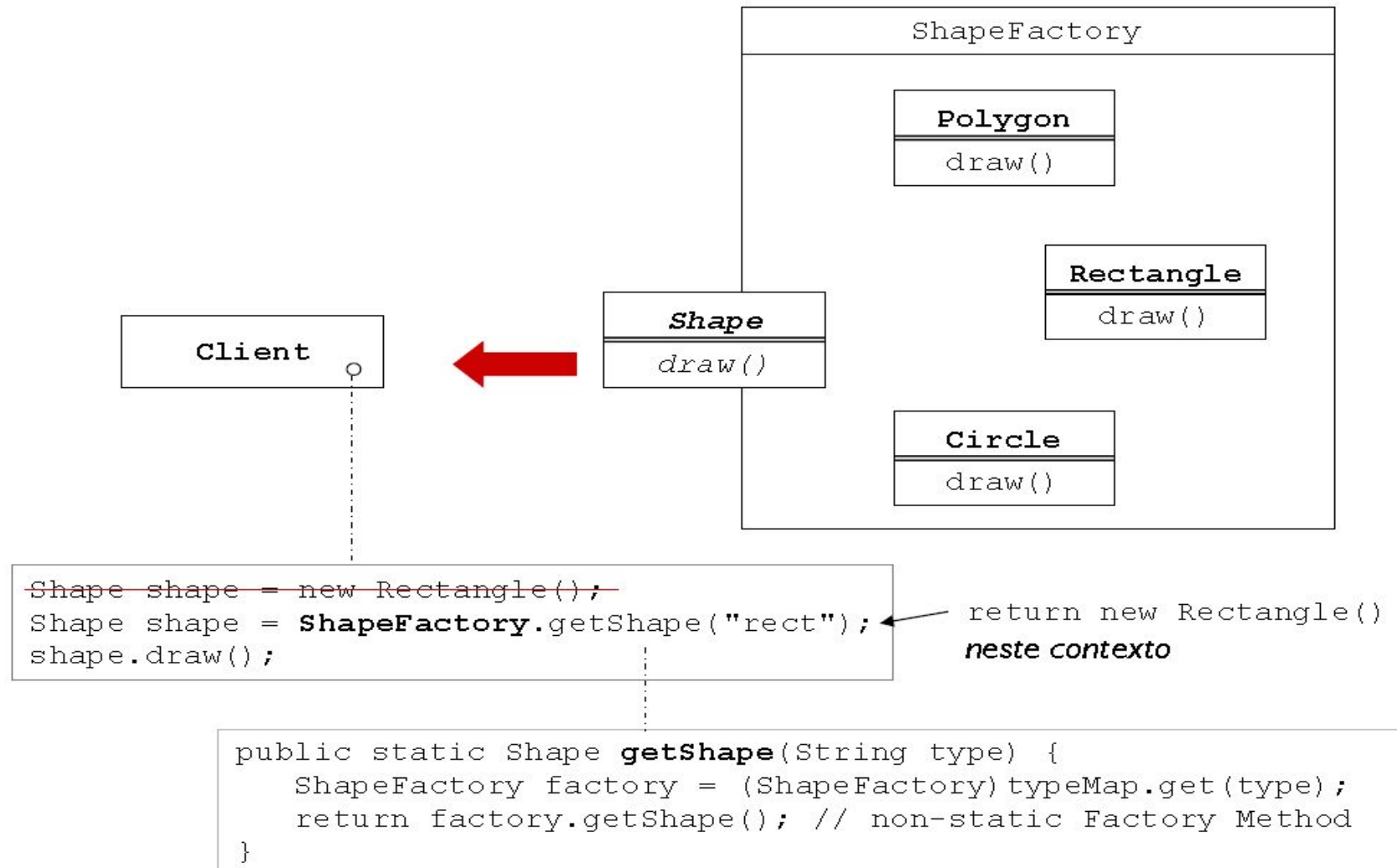
- O acesso a um objeto concreto será por meio da interface conhecida por meio de sua superclasse.
- O cliente não quer (ou não pode) saber qual implementação concreta está usando.





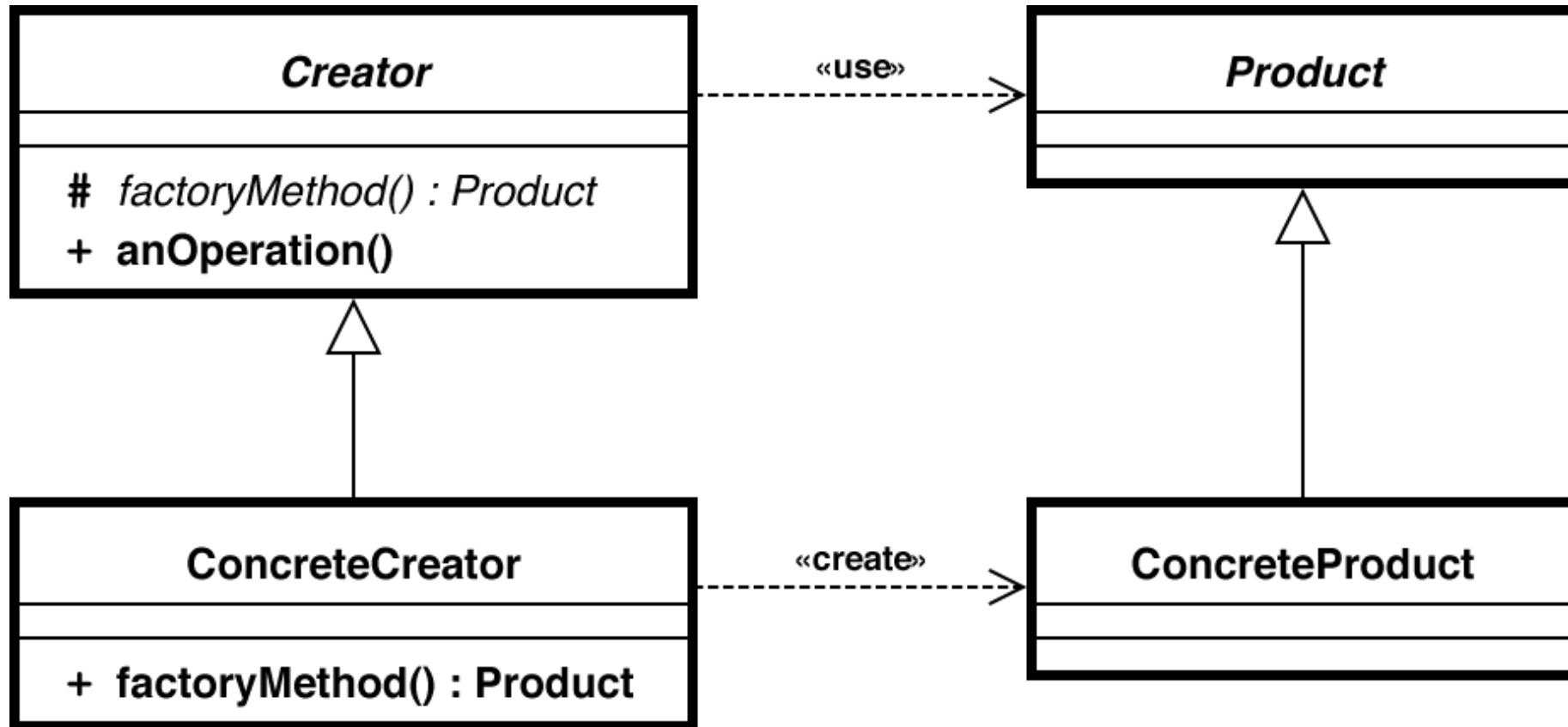
# Factory Method Exemplo

- O acesso a um objeto concreto será por meio da interface conhecida por meio de sua superclasse.
- O cliente não quer (ou não pode) saber qual implementação concreta está usando.



# Factory Method

## Estrutura



# Factory Method

## Solução

- É possível criar um objeto sem ter conhecimento algum de sua classe concreta?
  - Esse conhecimento deve estar em alguma parte do sistema, mas não precisa estar no cliente
  - **Factory Method** define uma interface comum para criar objetos
  - O objeto específico é determinado nas diferentes implementações dessa interface
  - O cliente do **Factory Method** precisa saber sobre implementações concretas do objeto criador do produto desejado

## Factory Method

### Consequências

- *Factory Methods* eliminam a necessidade de colocar classes específicas da aplicação no código
- Provê ganchos para subclasses
  - Criar objetos dentro de uma classe com um *Factory Method* é sempre mais flexível do que criar objetos diretamente
  - O *Factory Method* provê um gancho para que subclasses forneçam uma versão estendida de um objeto

# Factory Method

## TED – Sistema de saudações

- Uma aplicação precisa definir saudações diferentes para homens, mulheres e para pessoas que não queira identificar seu gênero.
  - Bem-vinda Sr<sup>a</sup>. X.
  - Bem-vindo Sr. Y.
  - Bem-vindo X.
- Sabe-se que cada usuário tem um nome e gênero. Como solucionar este cenário utilizando o padrão Factory Method?

- 
- “Prover uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.” [GOF]

Abstract Factory



# Abstract Factory

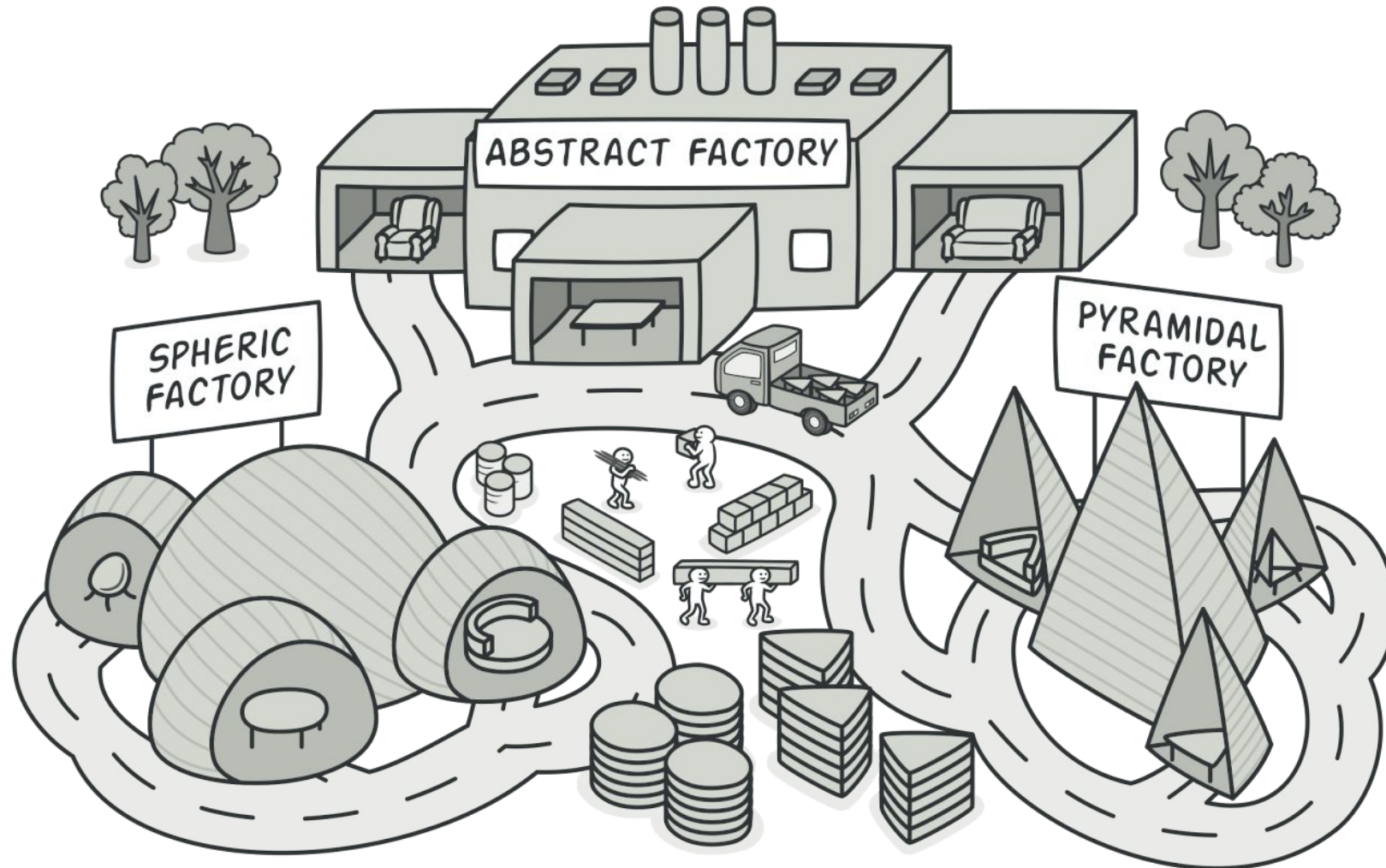
## Problema

○ sistema deve ser configurado com uma de múltiplas famílias de produtos.

Os produtos relacionados são projetados para serem utilizados juntos, e você quer garantir essa restrição.

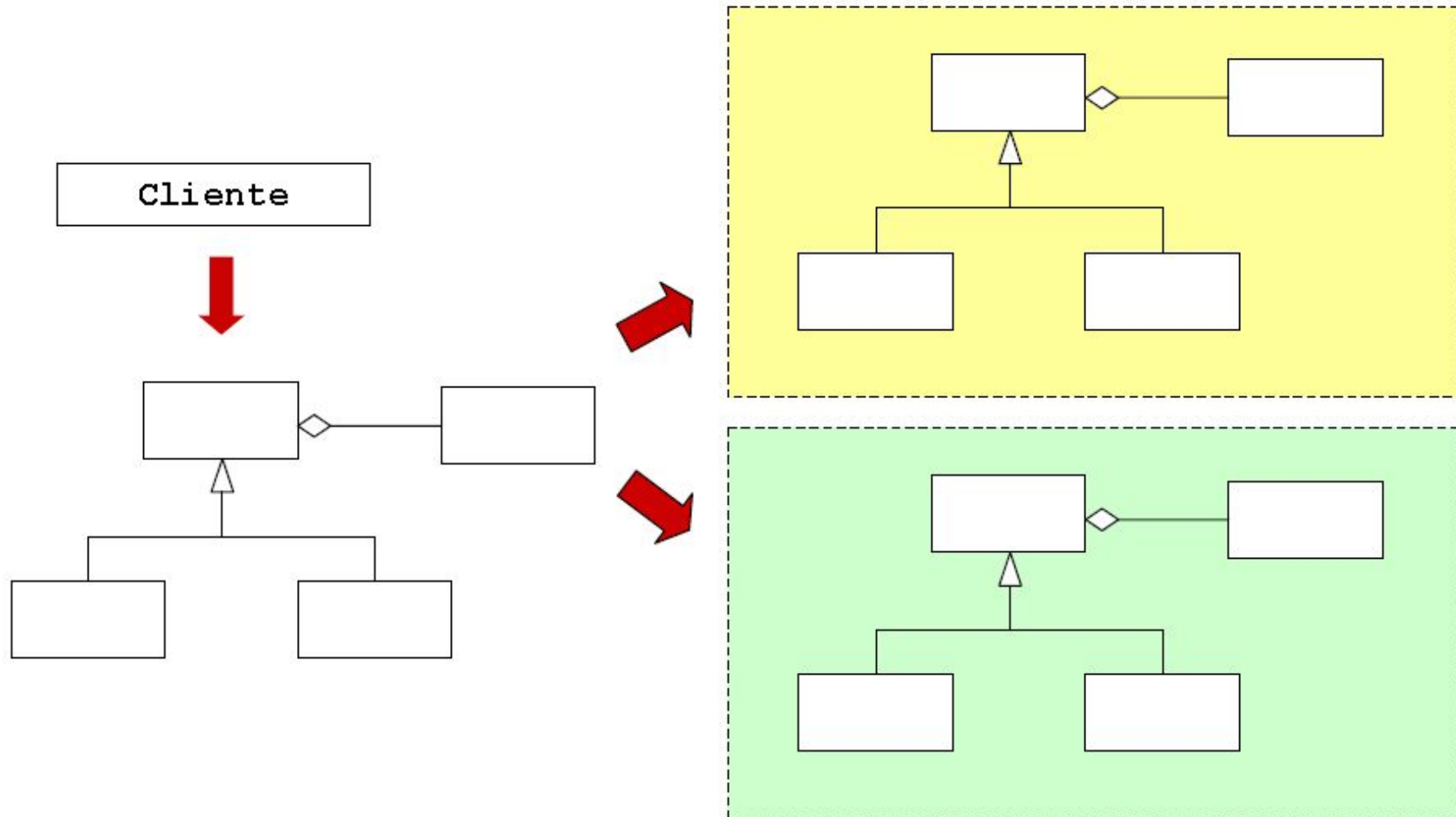


# Abstract Factory Problema












# Abstract Factory

## Exemplo



# Abstract Factory Problema

	Chair	Sofa	Coffee Table
Art Deco			
Victorian			
Modern			

# Abstract Factory

## Exemplo

- Implemente um sistema de venda de carros de luxo e carros simples utilizando o padrão Abstract Factory.



# Abstract Factory

Com o que já conhecemos,  
como poderíamos lidar com  
isso?



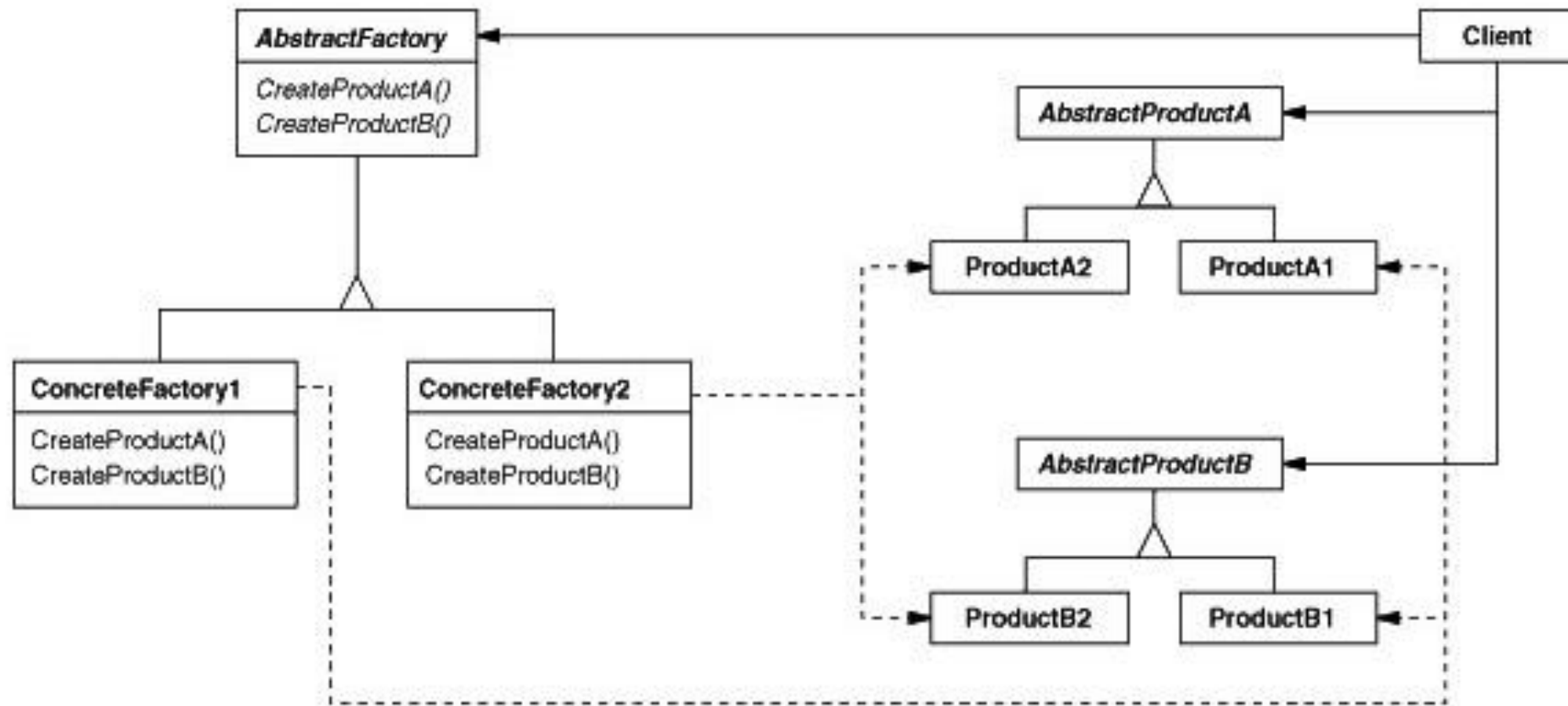
# Abstract Factory

## Solução

- Proporciona uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas
- Se o cliente possuir uma referência a uma *Abstract Factory* diferente, toda a criação será diferente
- ✓ Note que a estrutura do objeto composto é a mesma, o que muda são os componentes concretos em sua criação

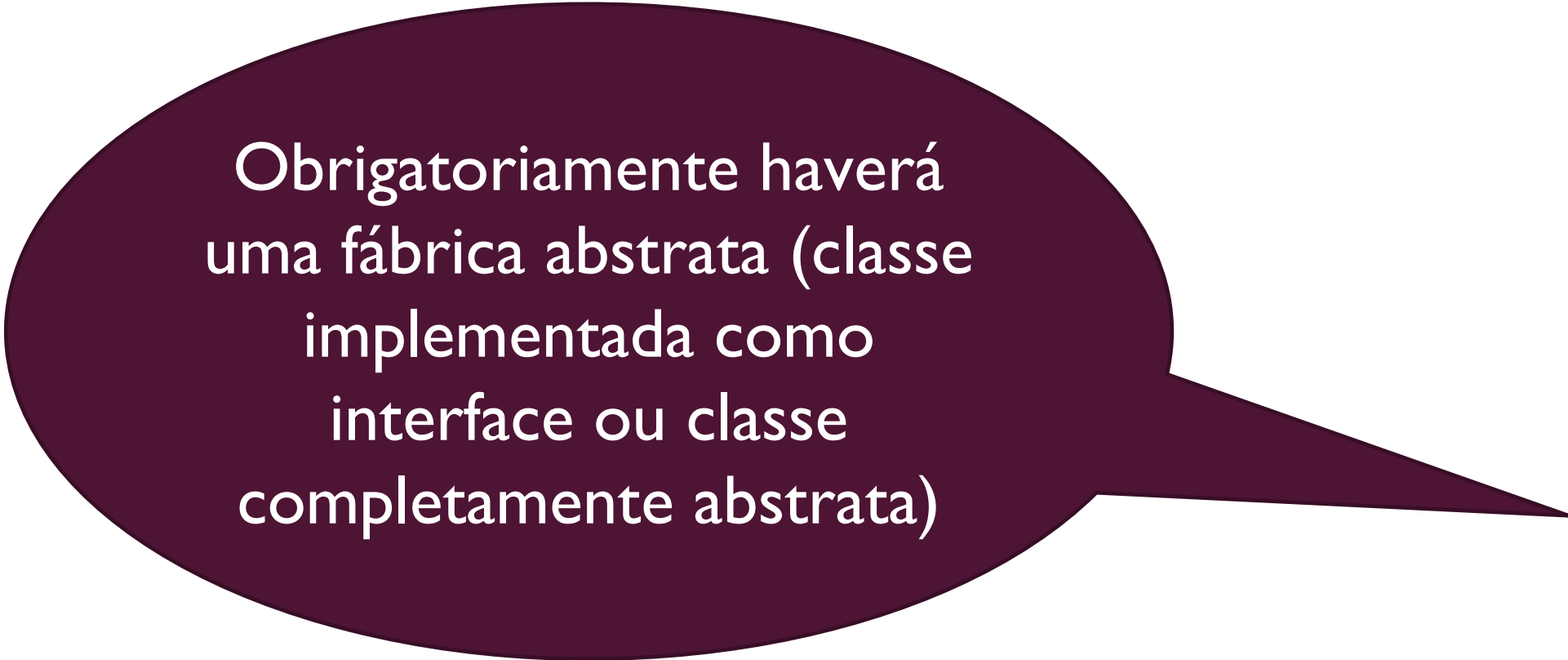
# Abstract Factory

## Solução





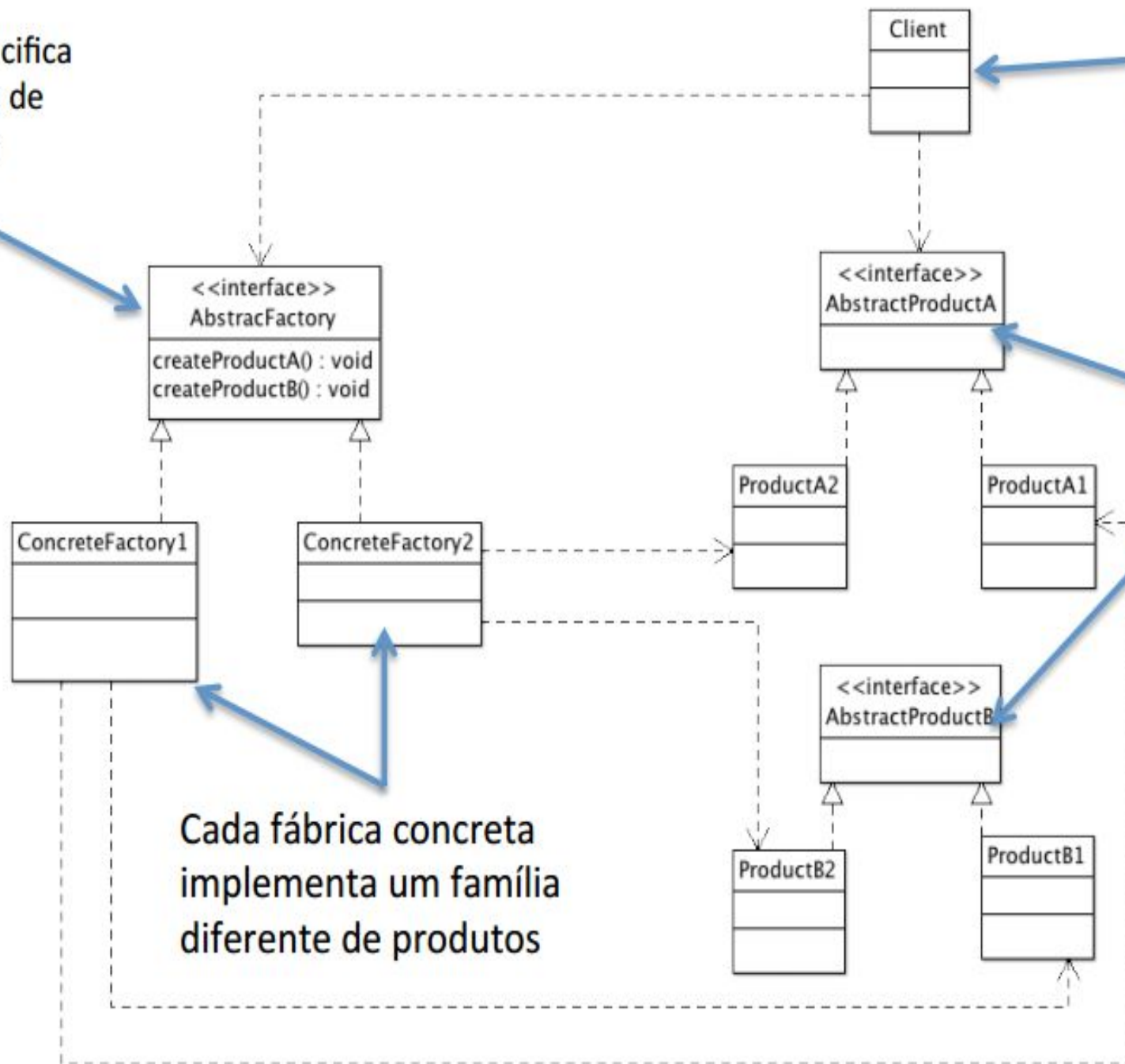
# Abstract Factory Solução



Obrigatoriamente haverá  
uma fábrica abstrata (classe  
implementada como  
interface ou classe  
completamente abstrata)

A fábrica abstrata especifica como criar uma família de produtos relacionados

O Cliente é escrito em termos da fábrica abstrata. Em tempo de execução é ligado a uma fábrica concreta



Cada fábrica concreta implementa um família diferente de produtos

Família de produtos

# Abstract Factory

## Consequências

- O padrão isola classes concretas
  - Uma factory encapsula a responsabilidade e o processo de criação de objetos de produtos
  - Isola clientes das classes de implementação
  - O cliente manipula instâncias através de suas interfaces abstratas
- Facilita o câmbio de famílias de produtos
  - A classe da FactoryConcreta só aparece em um lugar: onde ela é instanciada
  - Uma mudança numa única linha de código pode ser suficiente para mudar a FactoryConcreta que a aplicação usa
    - A família inteira de produtos muda de uma vez
- Do lado negativo: dá suporte a novos tipos de produtos não é trivial

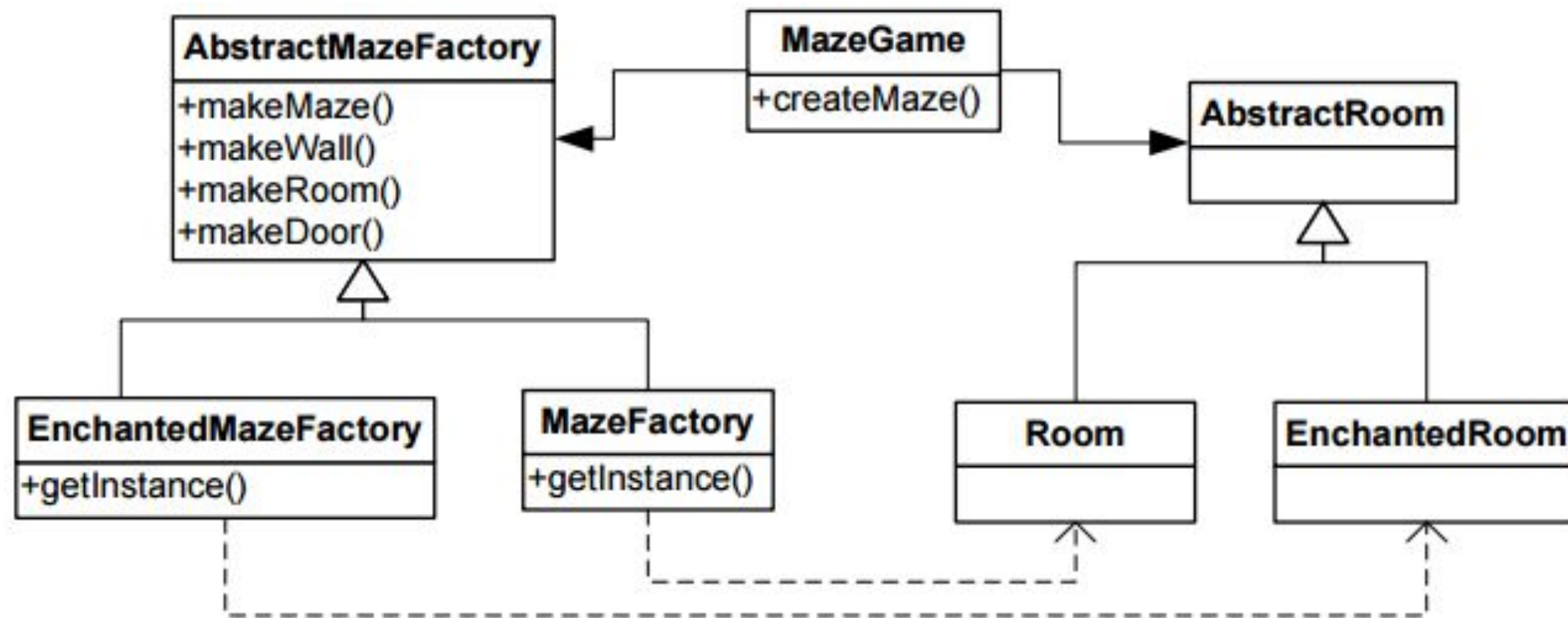
# Abstract Factory

## Quando usar?

- Quando um sistema deve ser independente de como seus produtos são criados, compostos e representados
- Quando um sistema deve ser configurado com uma entre várias famílias de produtos
- Quando uma família de produtos relacionados foi projetada para uso conjunto e você deve implementar essa restrição
- Quando você quer fornecer uma biblioteca de classes e quer revelar sua interface e não sua implementação
  - Não permita portanto que objetos sejam diretamente criados com new

## Abstract Factory - Exercício 2

Implemente o jogo Maze Game (Jogo do Labirinto)



# Factory Method x Abstract Method

- Parece semelhante ao padrão Factory Method
  - Porém, em vez do cliente chamar um método de criação (Factory Method), ele possui um objeto de criação (Abstract Factory) e o usa para chamar os métodos de criação
- Onde Factory Method quer que você seja diferente (via herança) para criar objetos diferentes, o Abstract Factory quer que você tenha algo diferente (via interface ou classe abstrata)

- 
- “Garantir que uma classe só tenha uma única instância, e prover um ponto de acesso global a ela.” [Gof]

Singleton



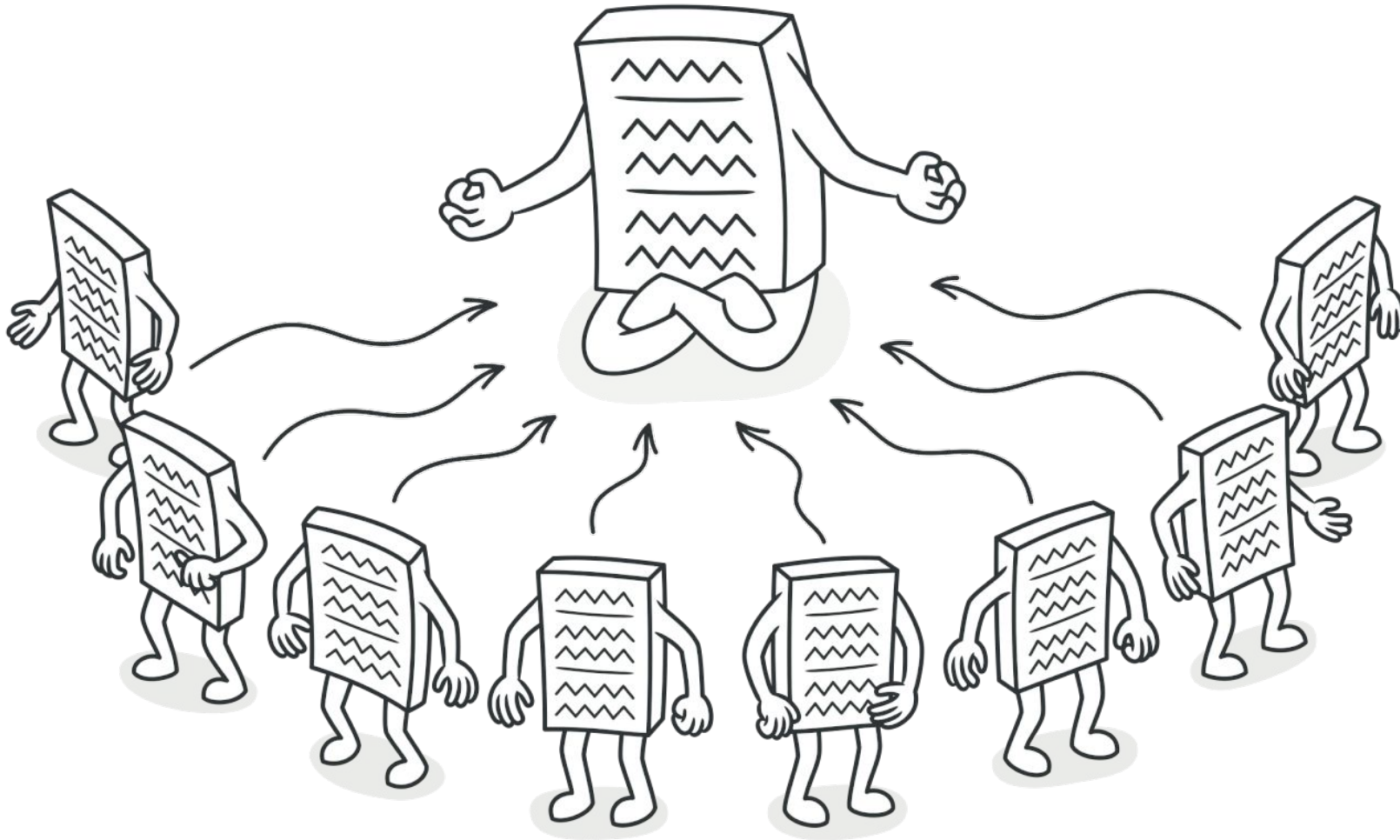


# Singleton

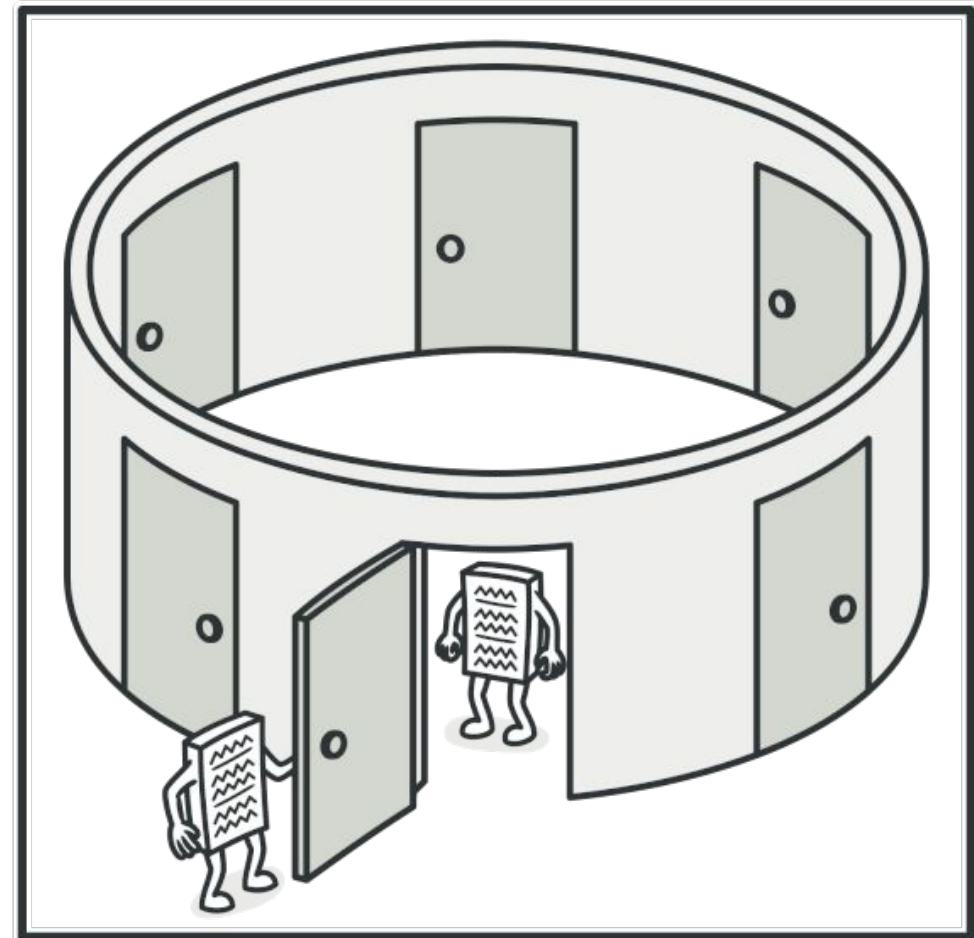
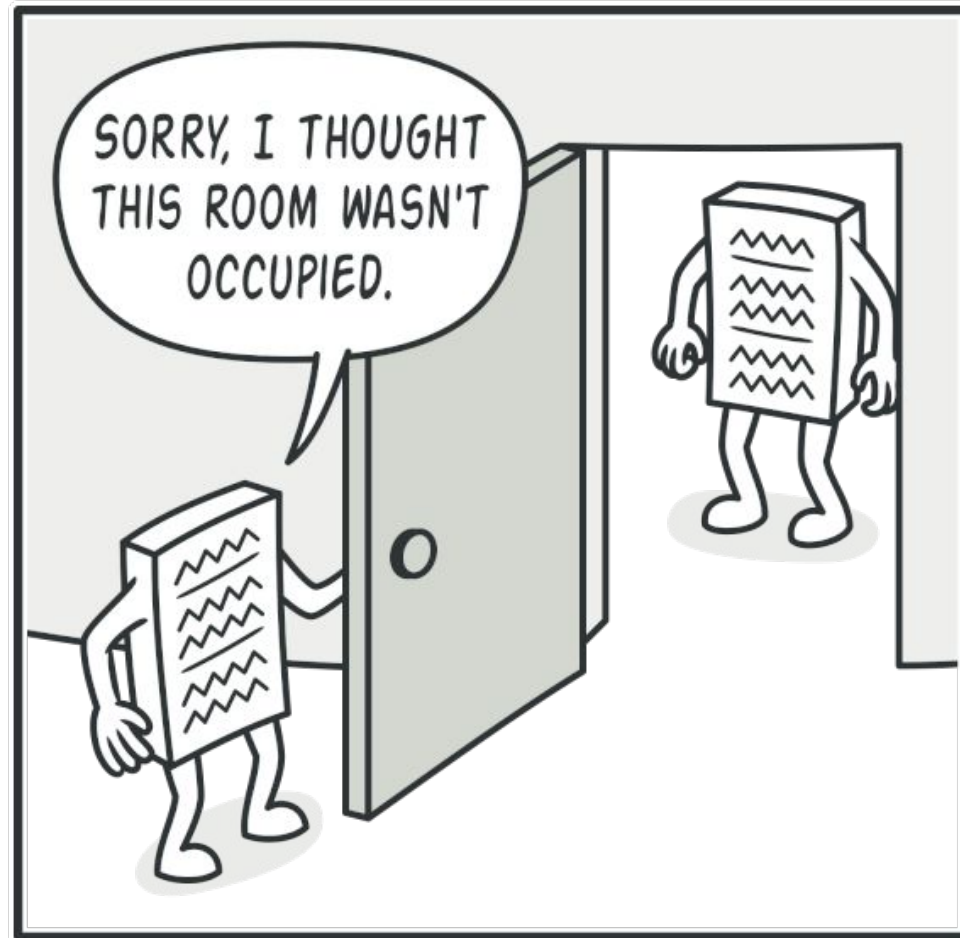
## Problema

- Garantir que apenas um objeto exista, independente do número de requisições que receber para criá-lo
- Preciso garantir que uma classe tem apenas uma instância e provê um ponto de acesso global a ela.
- Use Singleton quando:
  - Deve haver exatamente uma instância de uma classe, e ela deve ser acessível aos clientes a partir de um ponto de acesso conhecido

# Singleton Problema



# Singleton Problema



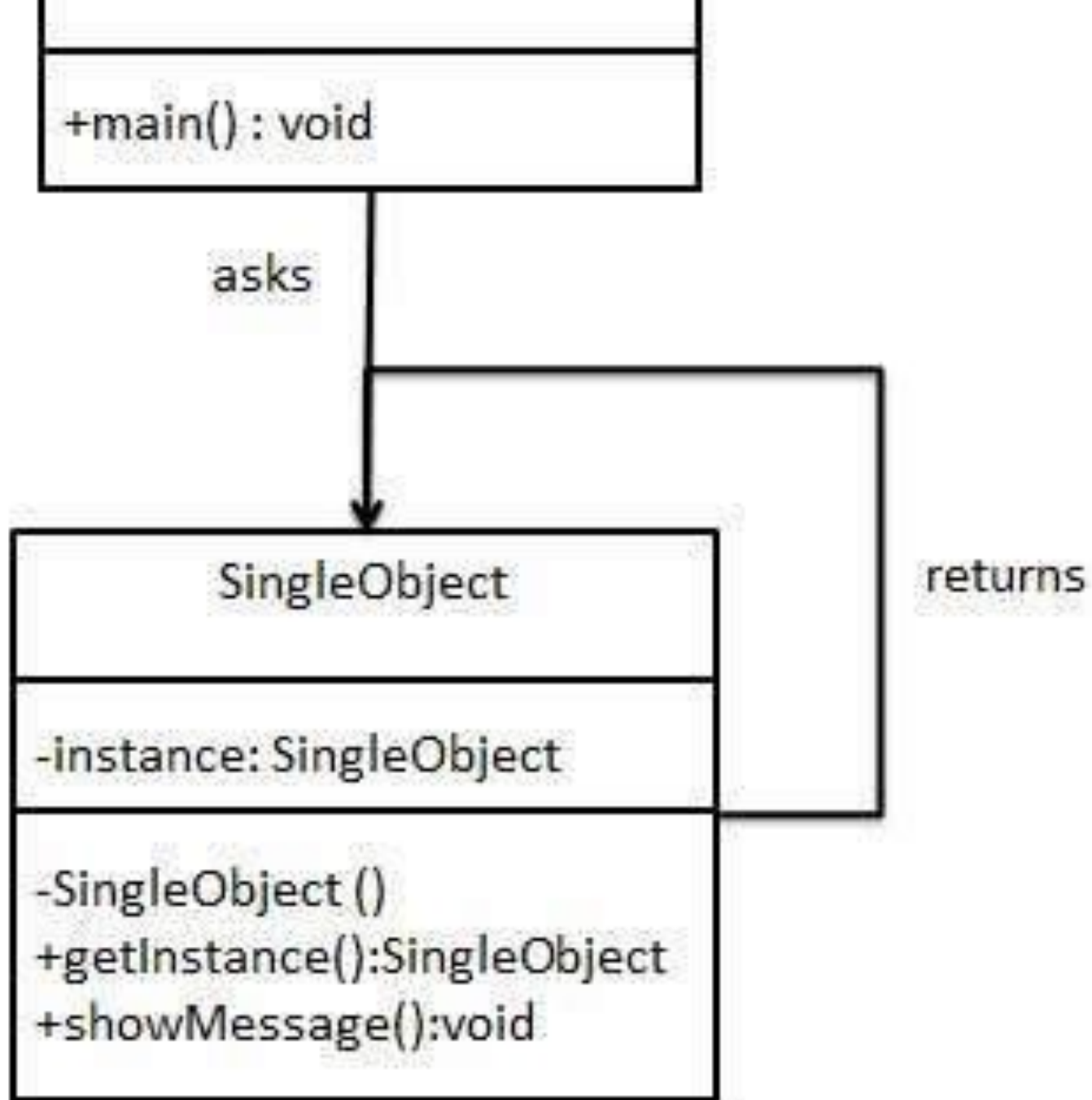
# Singleton

Como assegurar que haja uma única instância de uma classe?

# Singleton

- Formas de garantir que apenas um objeto exista:
  - Uma forma é de não permitir chamadas ao construtor
  - Construtor privativo
  - Ponto de acesso simples, estático e global

## Singleton Classes Env



# Singleton

## Vantagens

- Várias classes *Singleton* podem obedecer uma mesma interface, permitindo assim que um *Singleton* em particular seja escolhido para trabalhar com uma determinada aplicação em tempo de execução.
- Com apenas uma implementação interna do *Singleton* pode-se fazer com que o *Singleton* crie um número controlado de instâncias.
- É mais flexível que métodos estáticos por permitir o polimorfismo.



# Singleton

## Desvantagens

- Qualidade da implementação depende da linguagem
- Difícil de testar (simulações dependem de instância extra)
- Uso (abuso) como substituto para variáveis globais
- Inicialização *lazy* "preguiçosa" é complicada em ambiente *multithreaded* (é um anti-pattern)
- Difícil ou impossível de implementar em ambiente distribuído (é preciso garantir que cópias serializadas refiram-se ao mesmo objeto)

Hoje ...



## TED I

- Começar em sala e me enviar os exercício 1 e 2 da aula de hoje (slide 12 e 24).