Padrões de Projeto

O que são?

- Soluções provenientes de diversos projetos e utilizados por diversos programadores;
- Documentados em catálogos como Padrões de Projeto (Design Patterns);
- Catálogo GoF (Gang of Four);
- Visam a melhorar o acoplamento e a coesão do projeto de software;
- Devem ser adaptados ao contexto;
- Devem ser reutilizáveis;

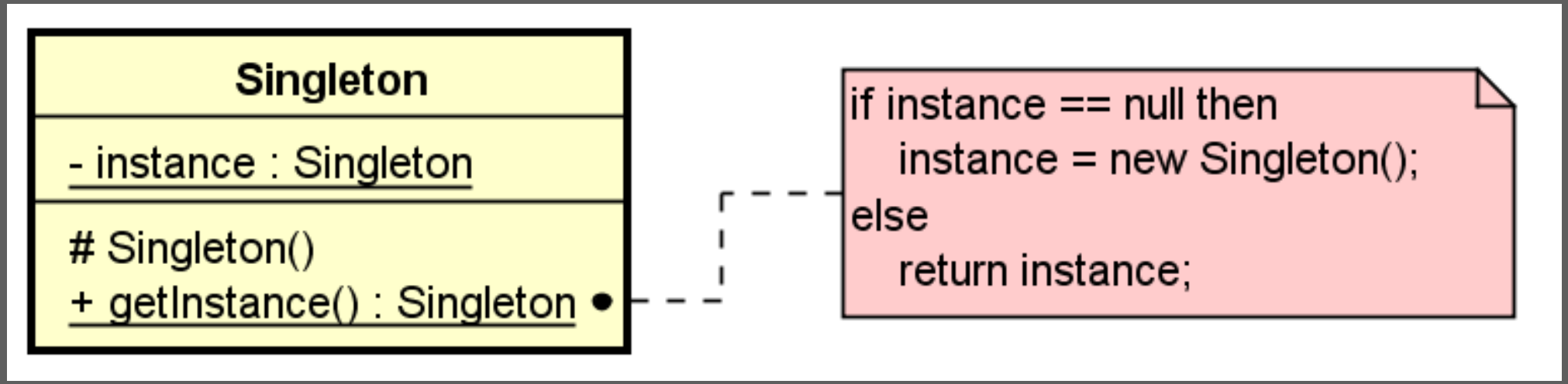
Os Padrões GoF

		Propósito		
		Criação	**Estrutura**	**Comportamento**
Escopo	**Classe**	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Criação

- **Singleton;**
- **Factory Method;**
- Abstract Factory;
- Builder;
- Prototype;

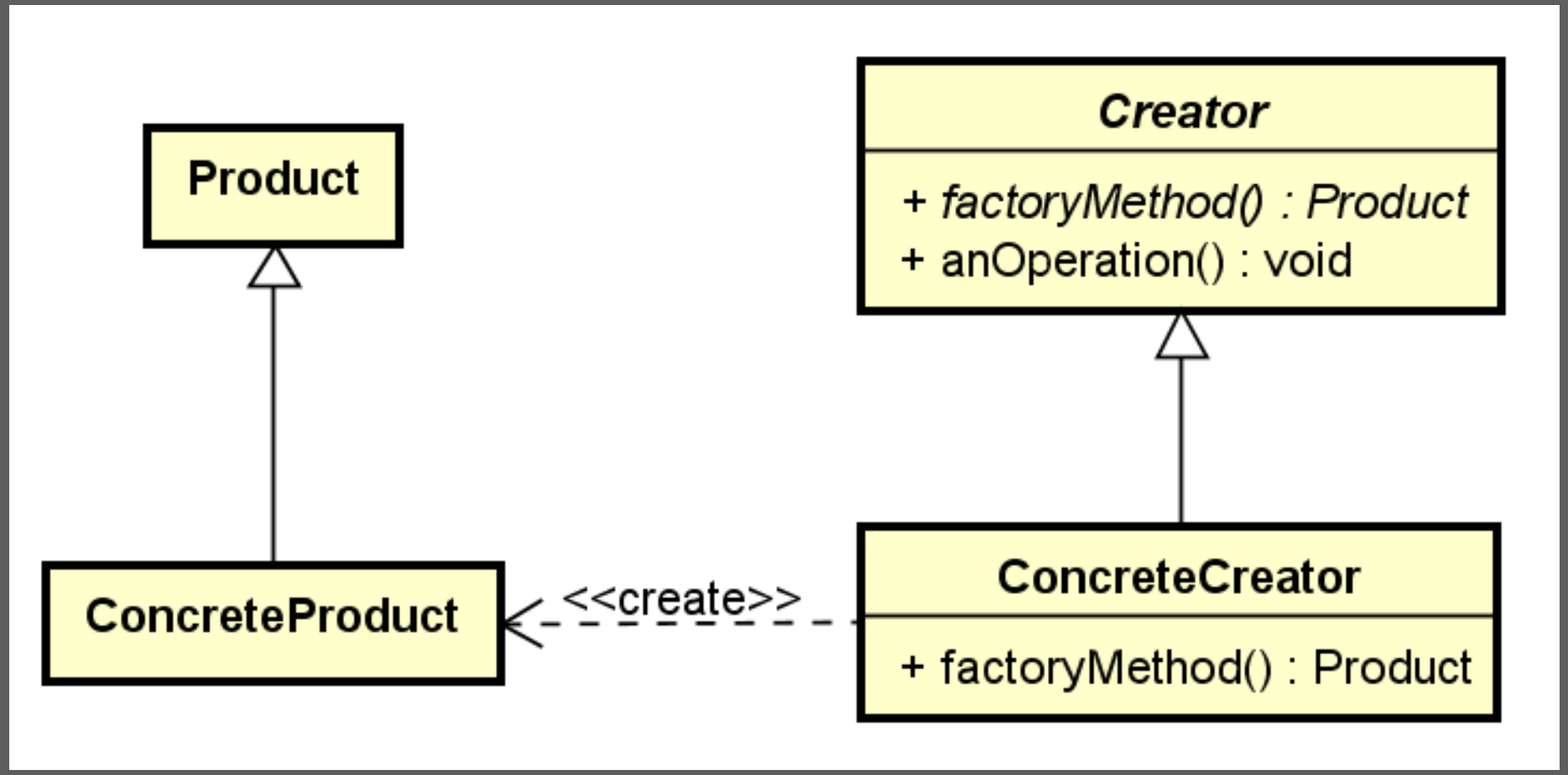
Singleton


Singleton
- instance : Singleton
Singleton()
+ getInstance() : Singleton
if instance == null then
instance = new Singleton();
else
return instance;

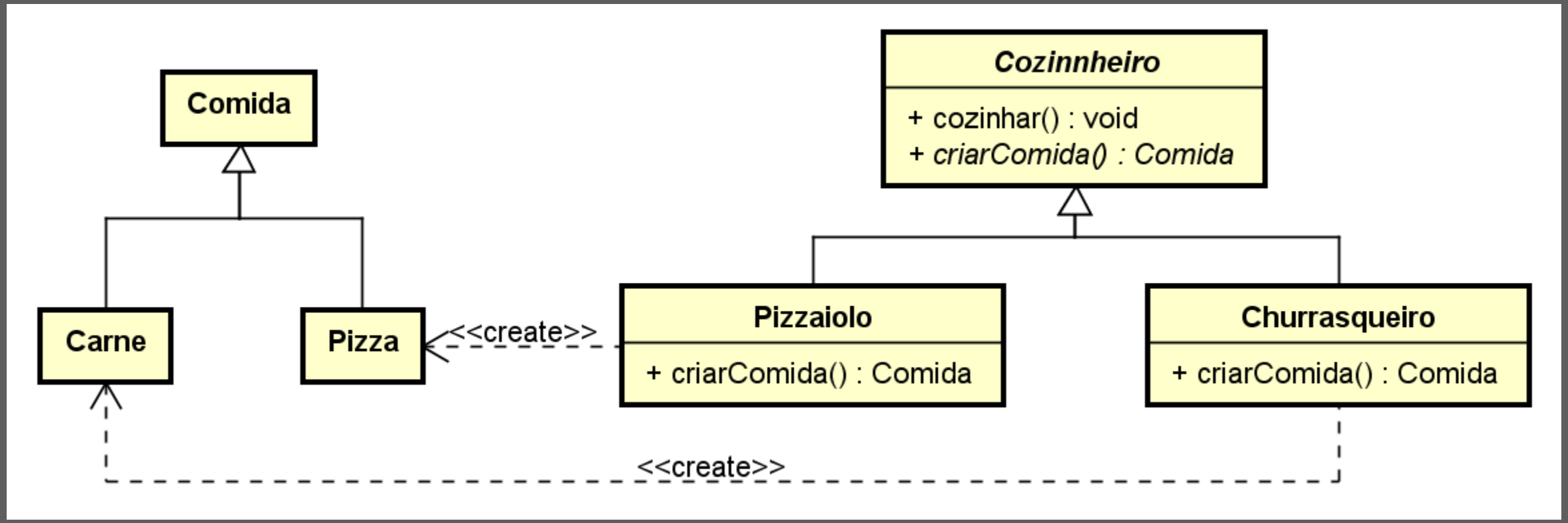
Singleton

- Exemplo de aplicação:
  - Contador de acessos ao website;
  - Conexão com o Banco de Dados;
  - Runtime (Java);
- Vantagens:
  - Compartilhamento de objetos comuns;
  - Garante apenas um objeto;
- Desvantagens:
  - Necessita de cuidado com a implementação de concorrência;

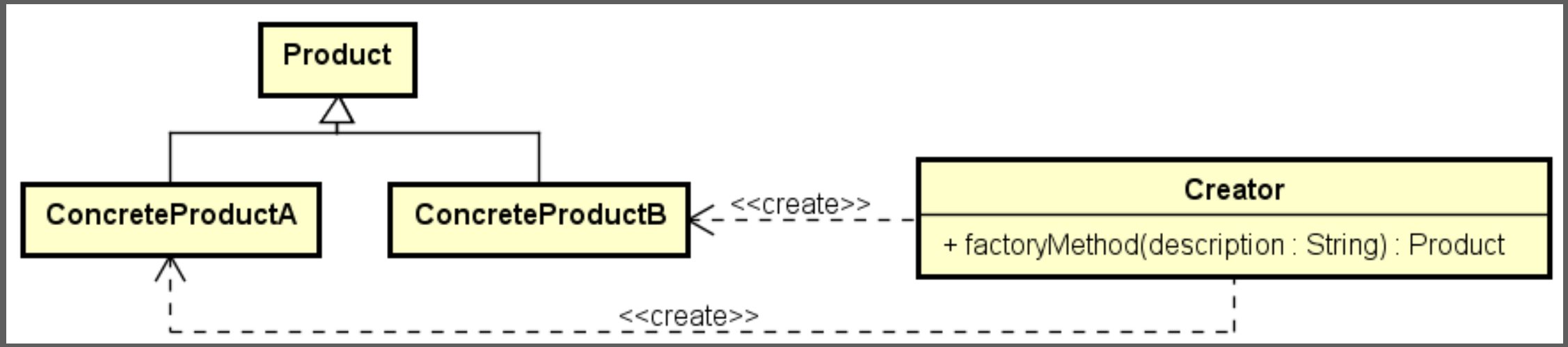
Factory Method



Factory Method



Factory Method



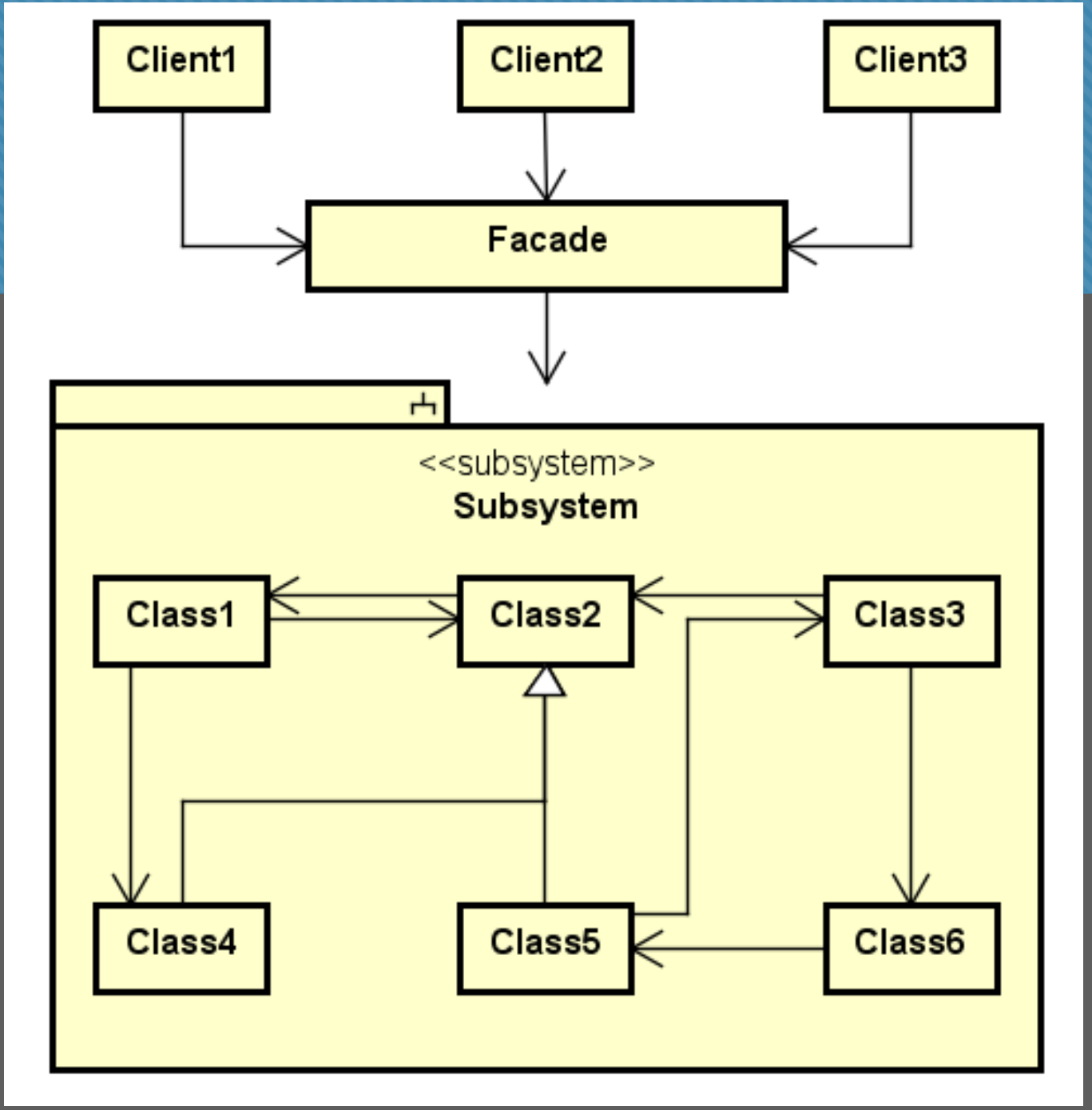
Factory Method

- Exemplo de Aplicação:
  - Sempre que houver muitas diferentes classes a serem instanciadas;
- Vantagens:
  - Instanciação indireta;
  - Fácil adição de novas classes;
  - Desacoplamento;
  - Pretinho básico: vai bem com tudo;
- Desvantagens:
  - Uma classe concreta criadora para cada item produzido;
  - Nem sempre o desacoplamento é garantido;
    - Factory Method com aninhamento de *ifs* ou um *switch* gigante;

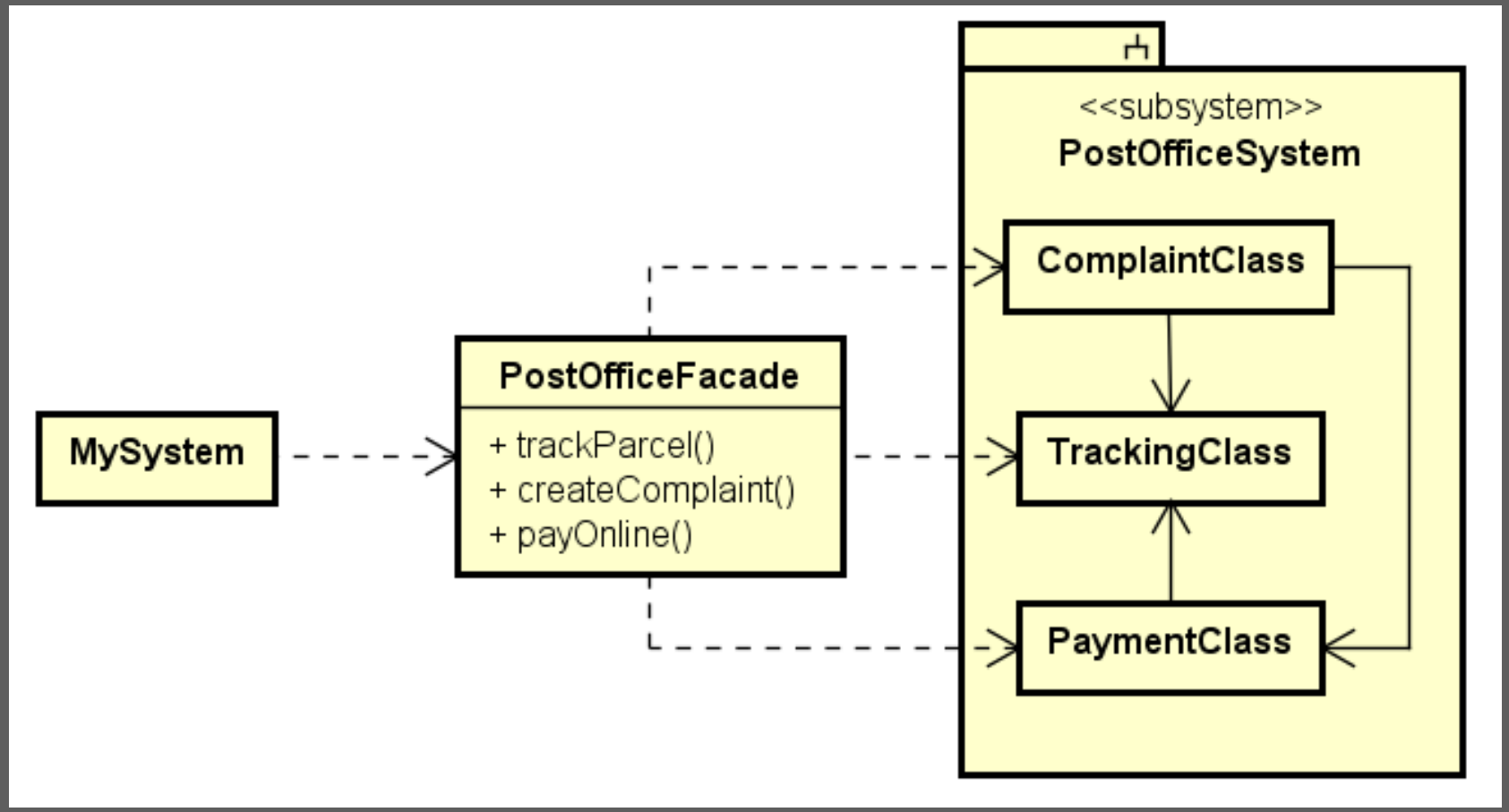
Estrutura

- **Facade;**
- **Proxy;**
- Adapter;
- Bridge;
- Composite;
- Decorator;
- Flyweight;

Facade


Client1
Client2
Client3
Facade
<<subsystem>>
Subsystem
Class1
Class2
Class3
Class4
Class5
Class6

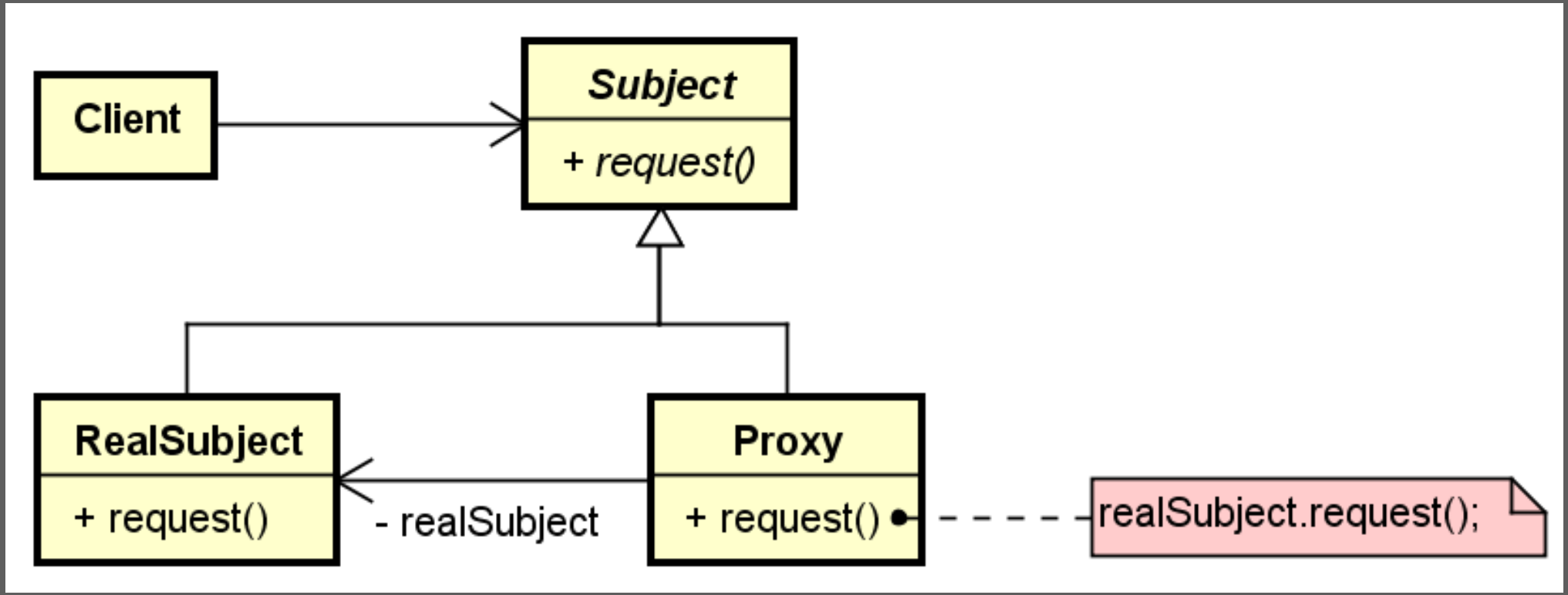
Facade


MySystem
PostOfficeFacade
+ trackParcel()
+ createComplaint()
+ payOnline()
<<subsystem>>
PostOfficeSystem
ComplaintClass
TrackingClass
PaymentClass

Facade

- Vantagens:
  - Um ponto de entrada;
  - Fácil saber quais são e onde estão as funcionalidades;
  - Se a funcionalidade do subsistema mudar, apenas uma classe muda;
- Desvantagens:
  - Facade é altamente acoplado ao subsistema;
  - Facade deve ser completo e sem defeitos;

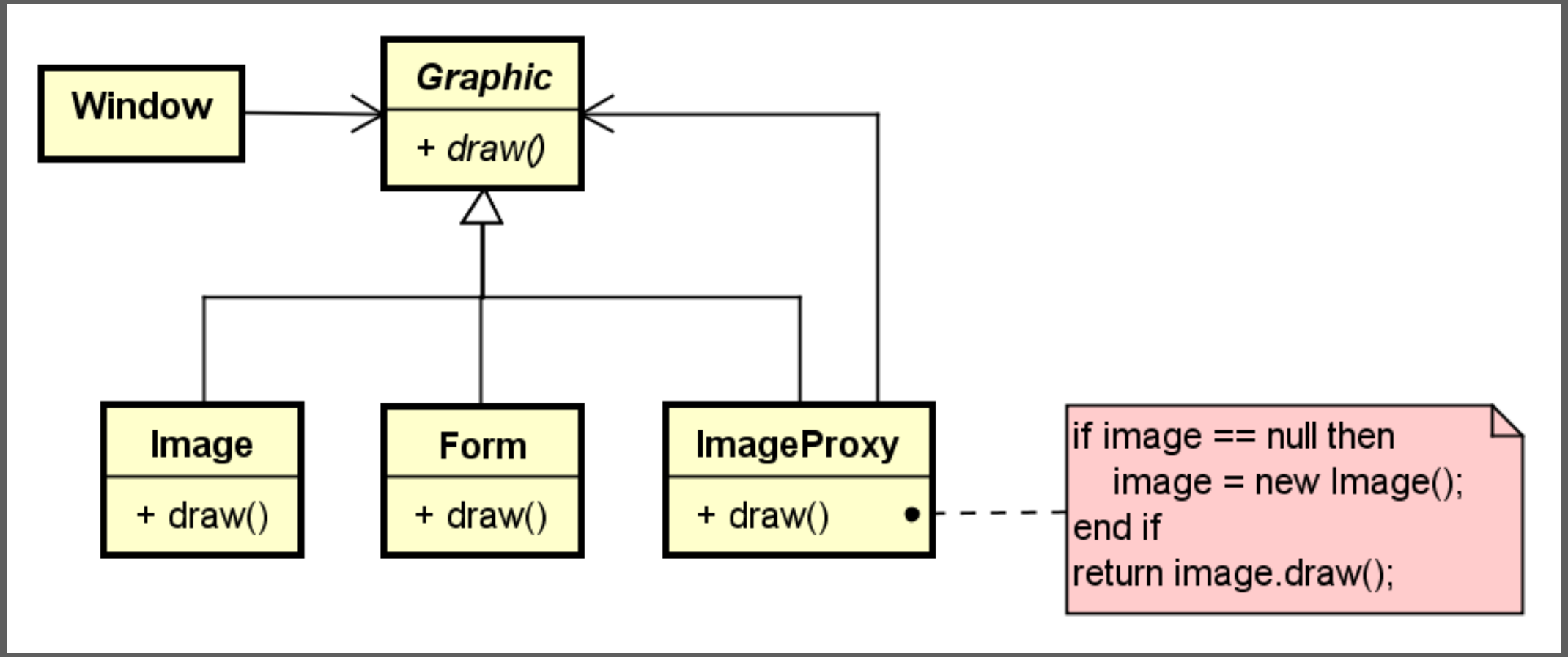
Proxy


Client
Subject
+ request()
RealSubject
+ request()
- realSubject
Proxy
+ request()
realSubject.request();

Proxy

- Aplicações:
  - Remote Proxy;
  - Virtual Proxy;
  - Protection Proxy;
  - Smart Reference;

Proxy


Window
Graphic
+ draw()
Image
+ draw()
Form
+ draw()
ImageProxy
+ draw()
if image == null then
image = new Image();
end if
return image.draw();

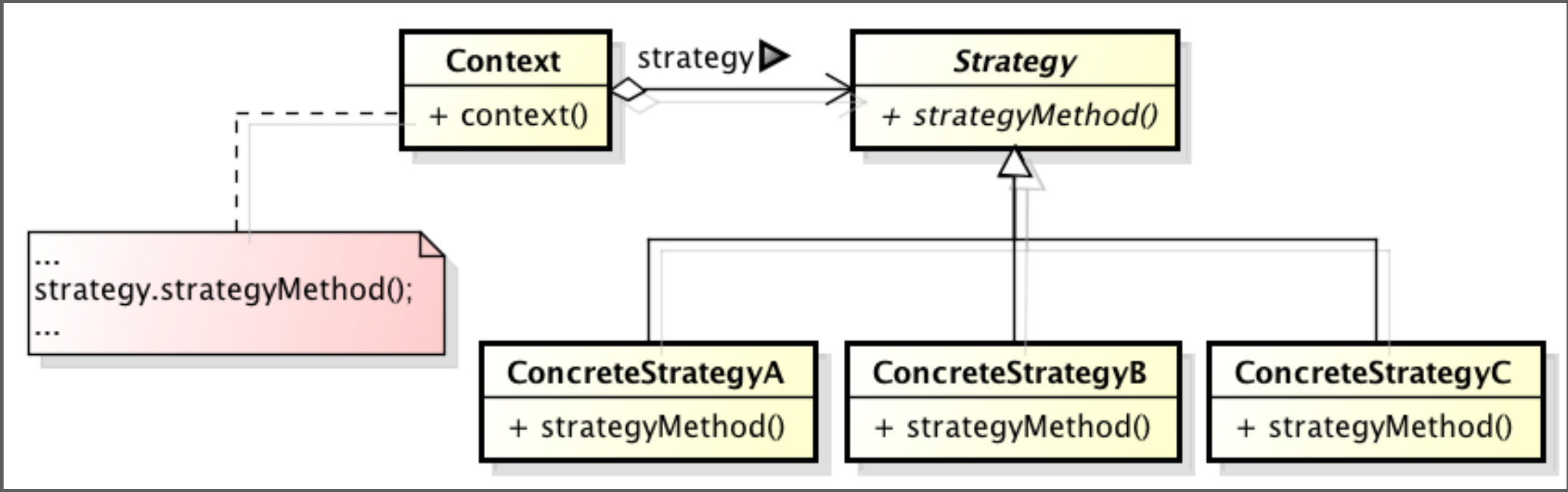
Proxy

- Vantagens:
  - Inserção de funcionalidade de forma desacoplada;
  - Possibilidade de troca de objetos em tempo de execução;
  - Flexível;
- Desvantagens:
  - O objeto proxy deve ser criado em algum momento;
  - Funcionalidade “escondida”;

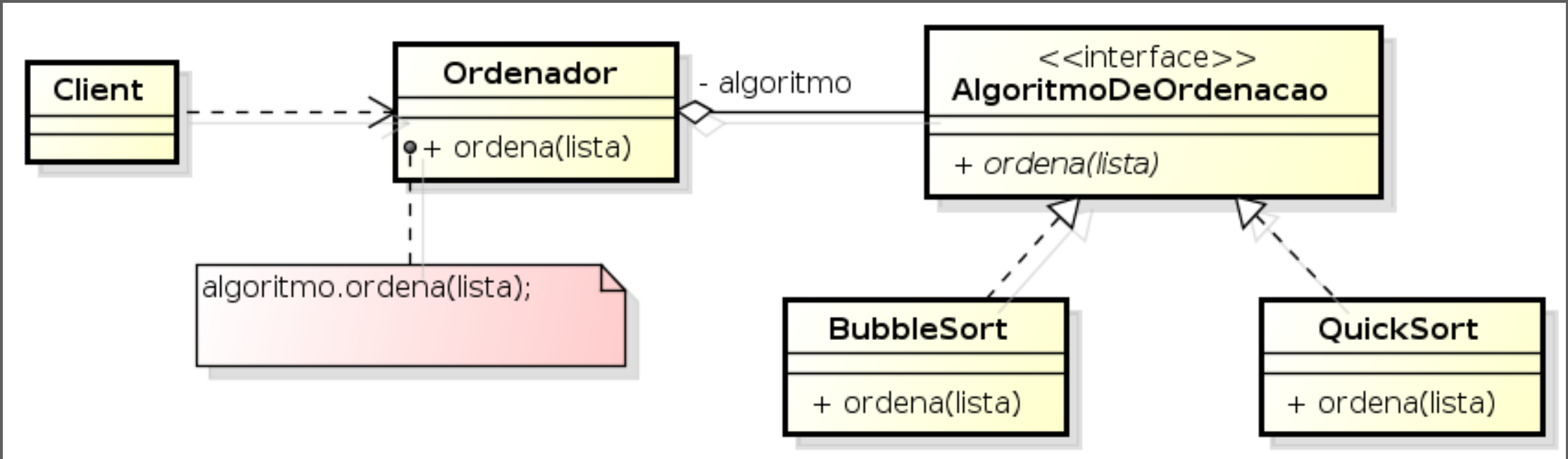
Comportamento

- **Strategy;**
- **Observer;**
- Chain of Responsibility;
- Command;
- Interpreter;
- Iterator;
- Mediator;
- Memento;
- State;
- Template Method;
- Visitor;

Strategy


Context
+ context()
strategy
Strategy
+ strategyMethod()
...
strategy.strategyMethod();
...
ConcreteStrategyA
+ strategyMethod()
ConcreteStrategyB
+ strategyMethod()
ConcreteStrategyC
+ strategyMethod()

Strategy

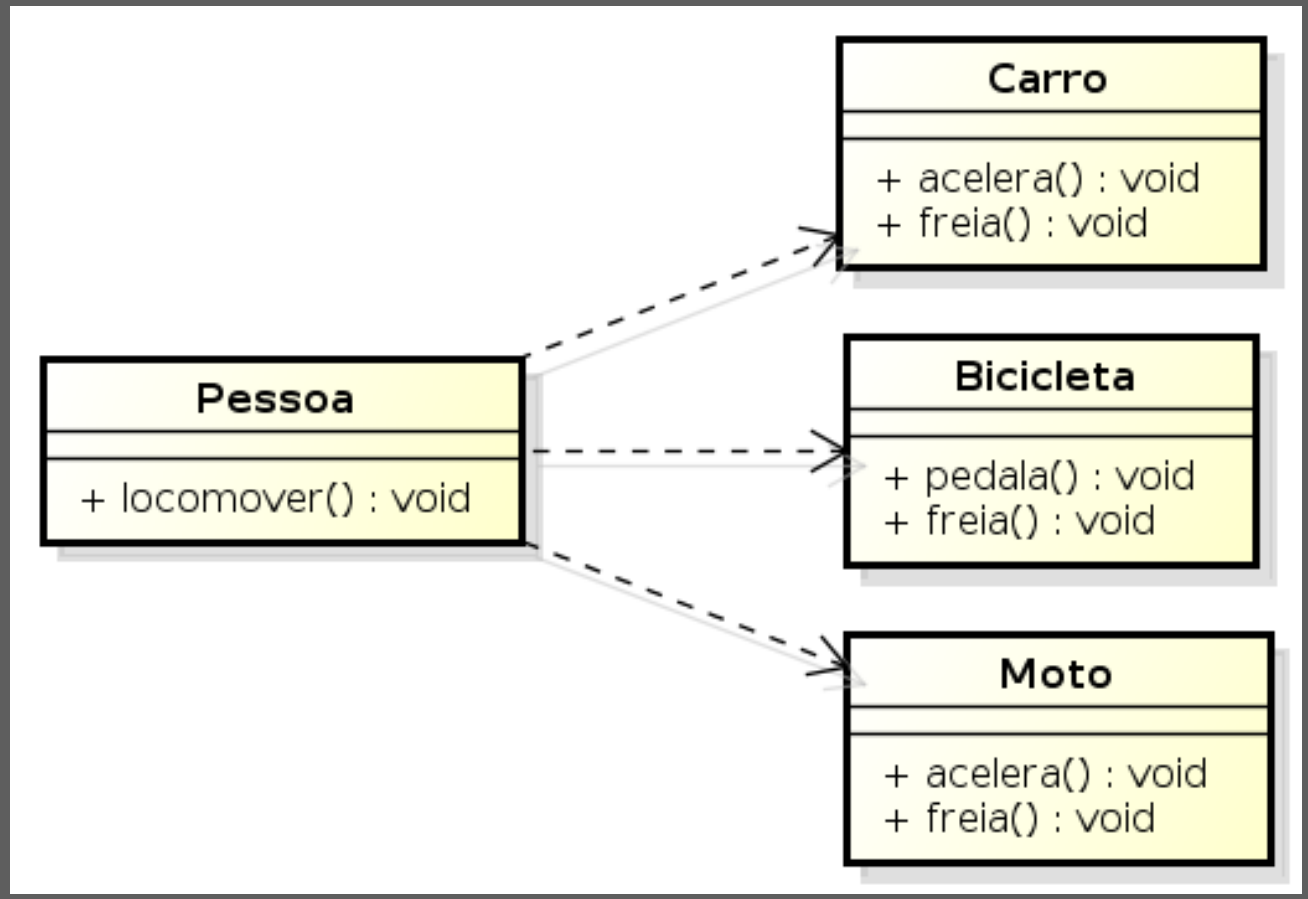

Client
Ordenador
+ ordena(lista)
- algoritmo
<<interface>>
AlgoritmoDeOrdenacao
+ ordena(lista)
algoritmo.ordena(lista);
BubbleSort
+ ordena(lista)
QuickSort
+ ordena(lista)

Strategy

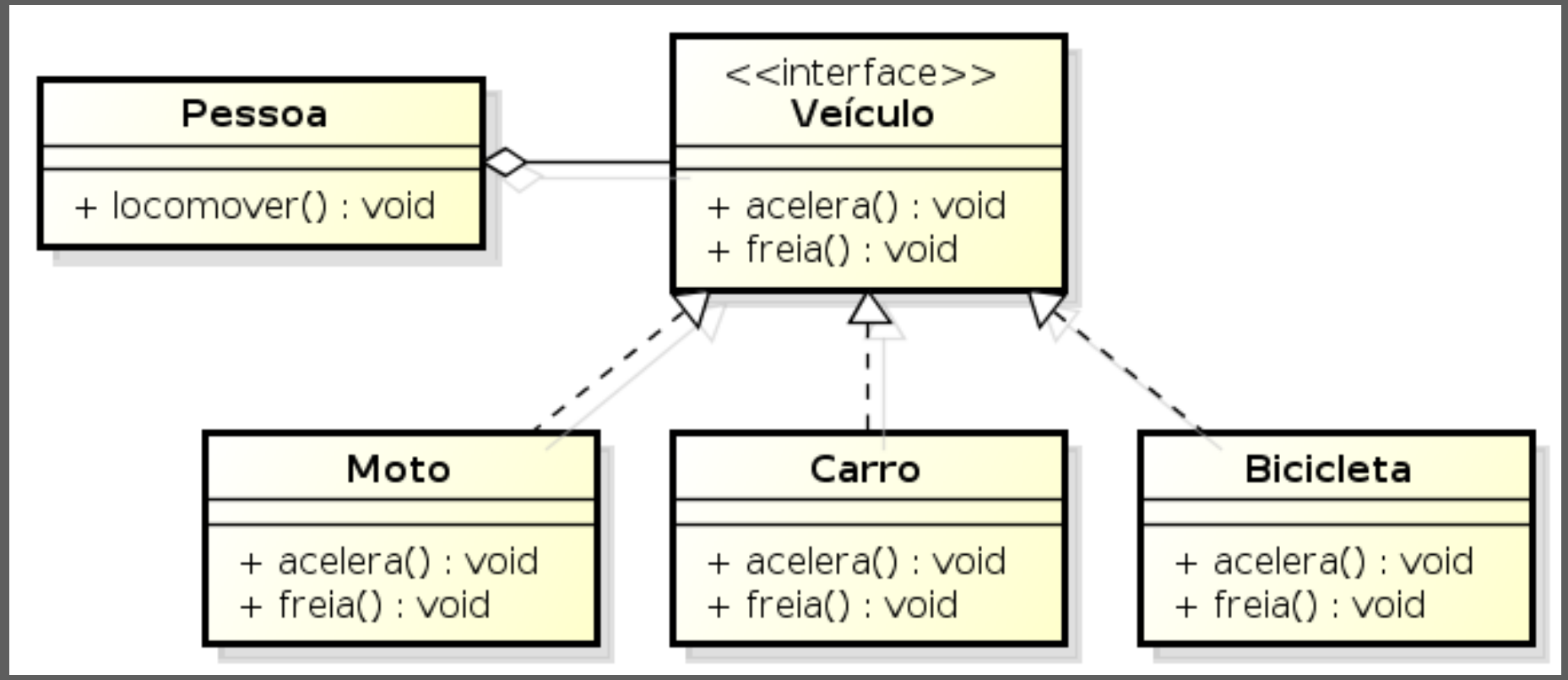
- Vantagens:
  - Simples;
  - Alto desacoplamento;
  - Intercâmbio de Algoritmos;
- Desvantagens:
  - Precisa construir um objeto de uma classe concreta;
  - Nem sempre é a solução para tudo;

Exercício

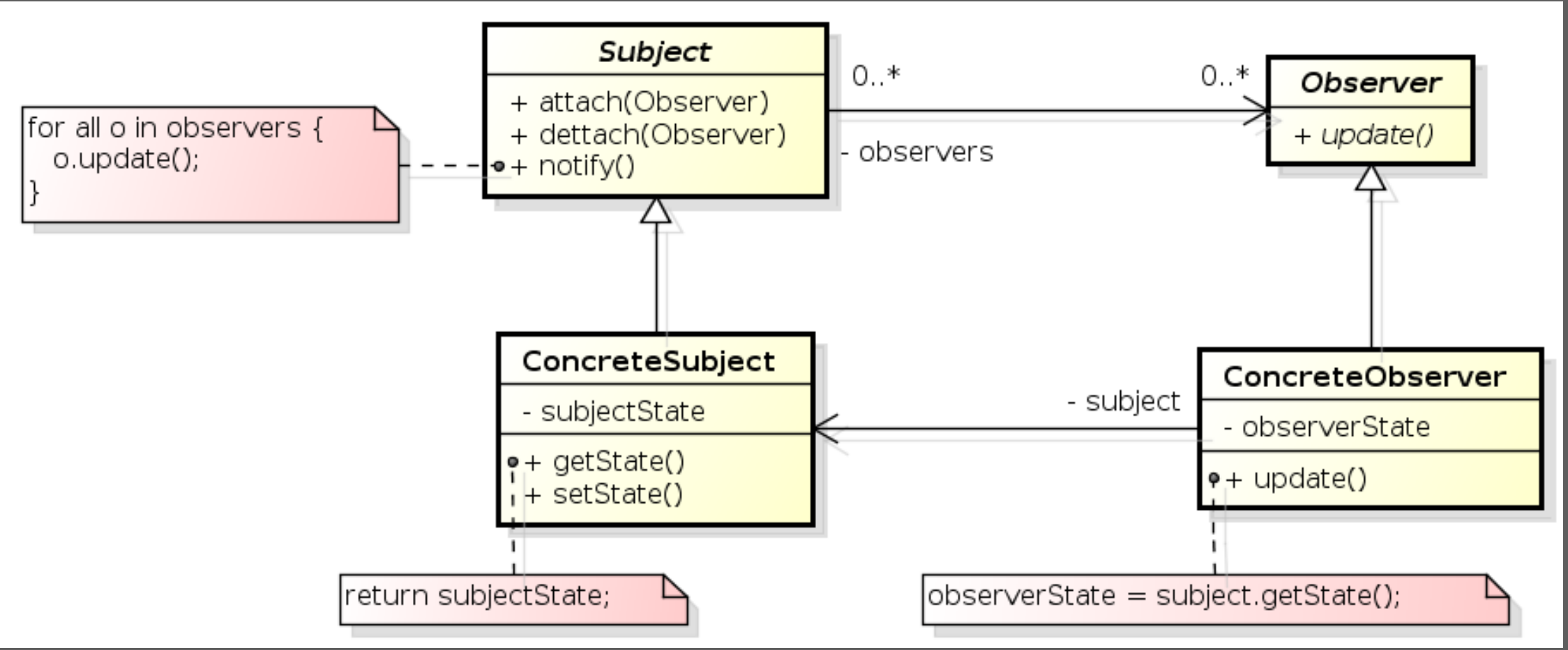
- Melhore o diagrama de classes ao lado com a aplicação de pelo menos um padrão de projeto;
- Sugestão: Strategy



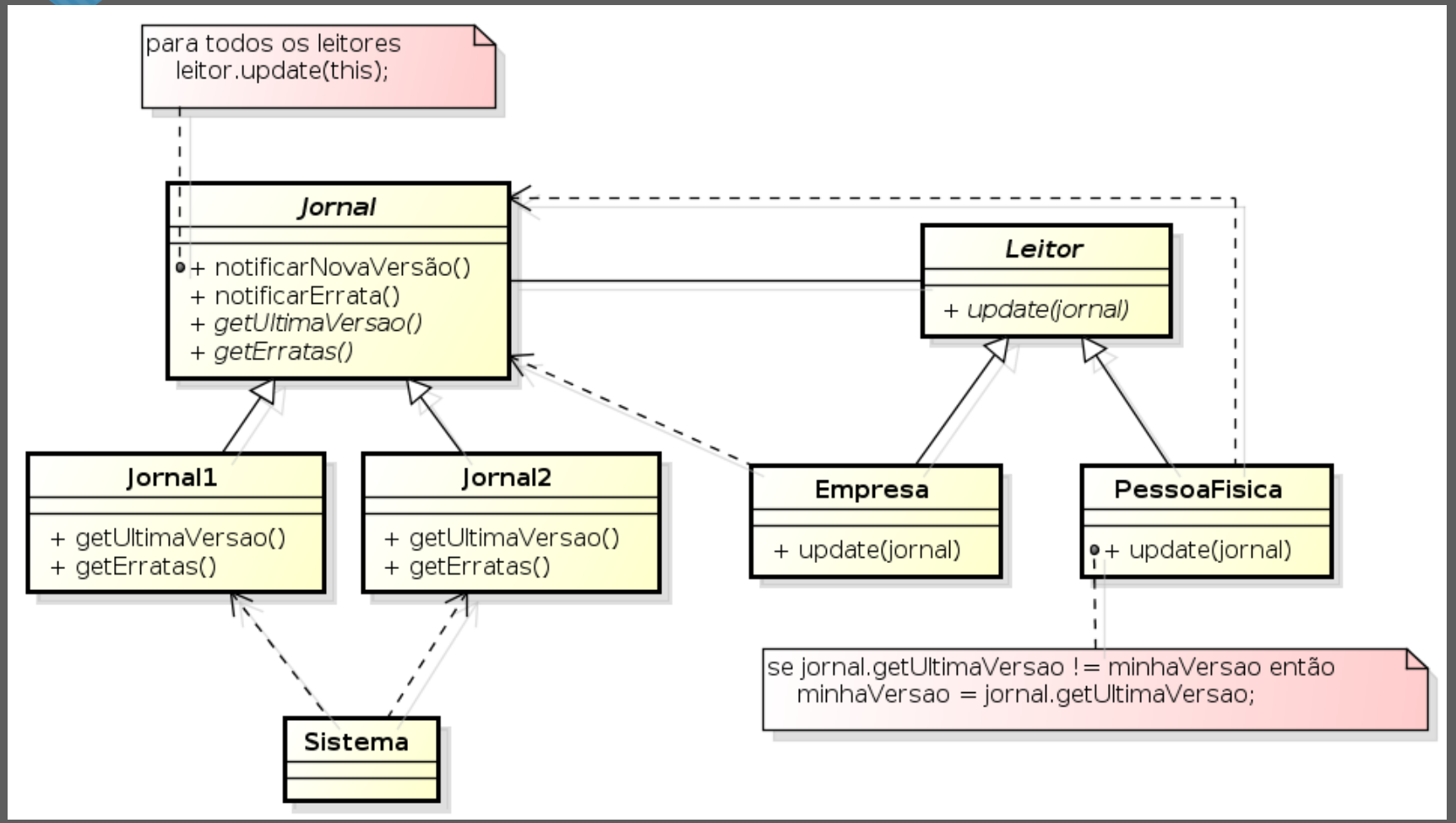
Exercício


Pessoa
+ locomover() : void
<<interface>>
Veículo
+ acelera() : void
+ freia() : void
Moto
+ acelera() : void
+ freia() : void
Carro
+ acelera() : void
+ freia() : void
Bicicleta
+ acelera() : void
+ freia() : void

Observer


Subject
+ attach(Observer)
+ dettach(Observer)
+ notify()
0..*
0..*
Observer
+ update()
- observers
for all o in observers {
o.update();
}
ConcreteSubject
- subjectState
+ getState()
+ setState()
- subject
ConcreteObserver
- observerState
+ update()
return subjectState;
observerState = subject.getState();

Observer


para todos os leitores
leitor.update(this);
Jornal
+ notificarNovaVersão()
+ notificarErrata()
+ getUltimaVersao()
+ getErratas()
Leitor
+ update(jornal)
Jornal1
+ getUltimaVersao()
+ getErratas()
Jornal2
+ getUltimaVersao()
+ getErratas()
Empresa
+ update(jornal)
PessoaFisica
+ update(jornal)
Sistema
se jornal.getUltimaVersao != minhaVersao então
minhaVersao = jornal.getUltimaVersao;

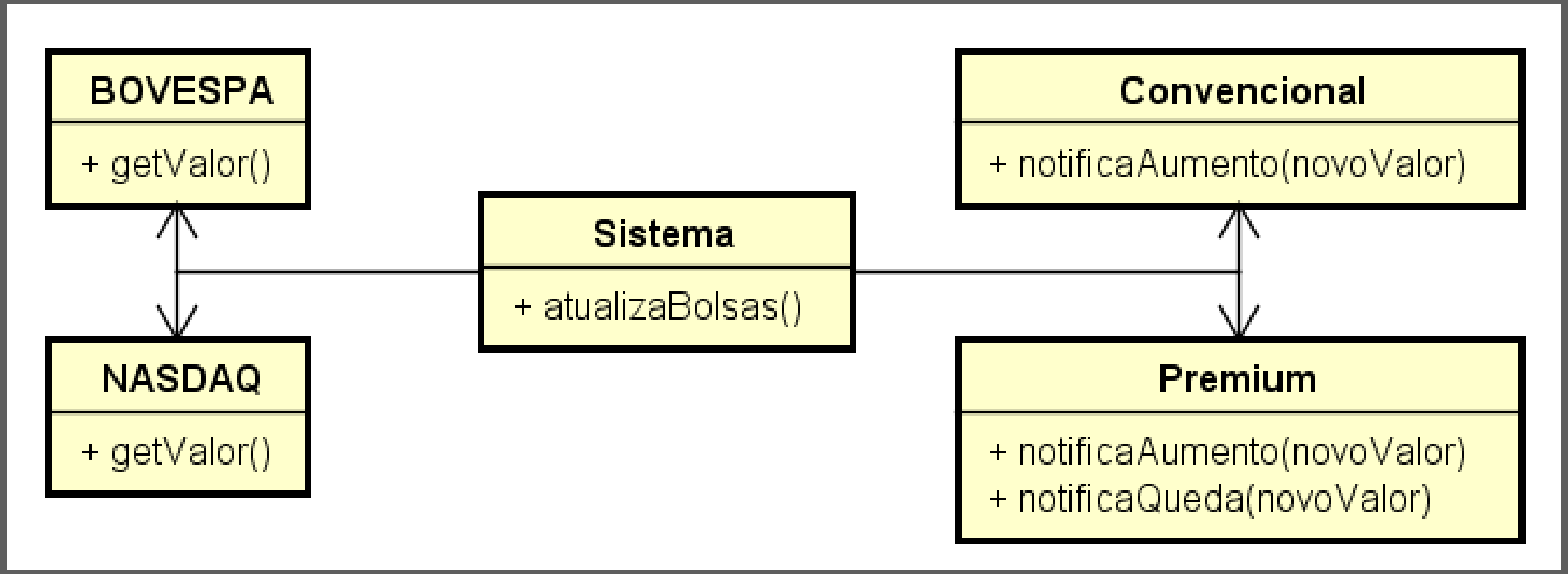
Observer

- Vantagens:
  - Desacopla notificadores e observadores;
  - Aumenta a coesão;
- Desvantagens:
  - Notificações desnecessárias;
  - Complexidade;

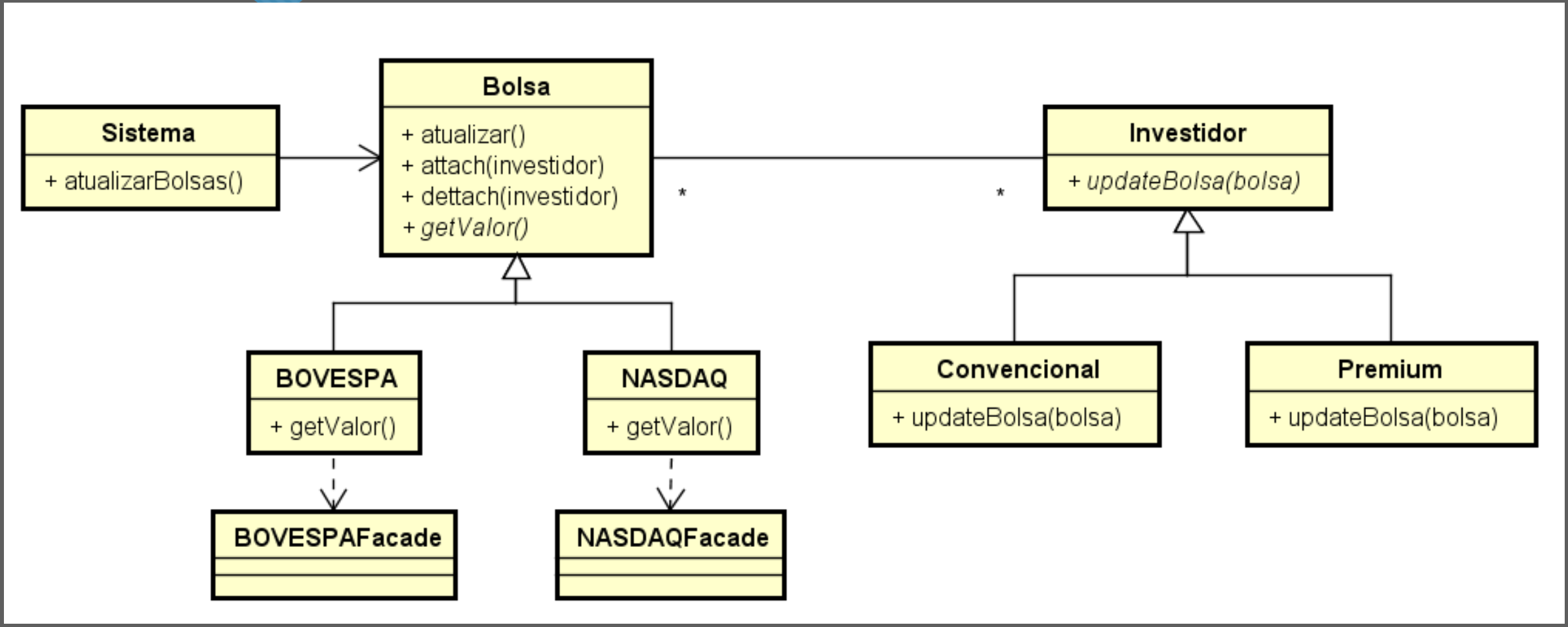
Exercício

- Você precisa criar um software para uma empresa que trabalha com bolsa de valores;
- Essa empresa trabalha com a NASDAQ e com a BOVESPA;
- Sempre que uma das duas bolsas sobe, todos os clientes da empresa devem ser notificados, mas quando uma das duas bolsas cai, somente os clientes premium podem ser notificados;
- A NASDAQ e a BOVESPA possuem seus próprios subsistemas que podem ser acessados remotamente;
- Faça o diagrama de classes para o sistema;

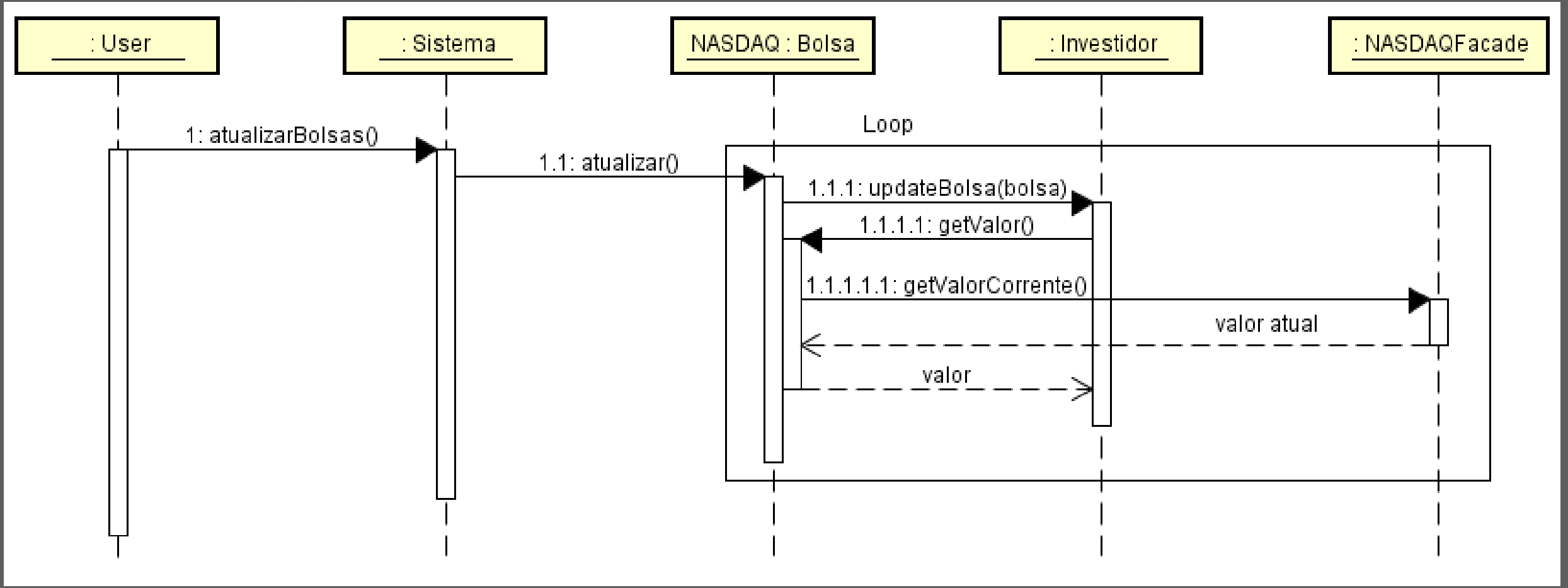
Exercício – Exemplo Ruim



Exercício – Solução



Exercício – Diagrama de Sequência



Conclusão

- Soluções quase prontas;
- Precisa de adaptações;
- Melhora a coesão e o acoplamento dos elementos do projeto;
- Podem ser complexos;
- Não podem ser aplicados sempre;
- É possível combiná-los;
- Anti-patterns;