UML

Unified Modeling Language (UML)

Linguagem de Modelagem Unificada

É uma linguagem de modelagem para a elaboração da estrutura de projetos de software.

Tem como **objetivo** especificar, documentar e estruturar para sub-visualização e maior visualização lógica do desenvolvimento completo de um sistema de informação.

Tipos de Diagramas do UML

UML 2 possui 14 tipos de diagramas divididos em duas categorias:

- **Diagramas Estruturais**
  Enfatizam os elementos que precisam estar presentes no sistema modelado.

- **Diagramas Comportamentais**
  Enfatizam o que precisa acontecer no sistema modelado.

Tipos de Diagramas do UML

UML 2 possui 14 tipos de diagramas divididos em duas categorias:

- **Diagramas Estruturais**
  Enfatizam os elementos que precisam estar presentes no sistema modelado.

  - Diagrama de Classes
  - Diagrama de Objetos
  - Diagrama de Componentes
  - Diagrama de Instalação ou de Implantação
  - Diagrama de Pacotes
  - Diagrama de Estrutura Composta
  - Diagrama de Perfil

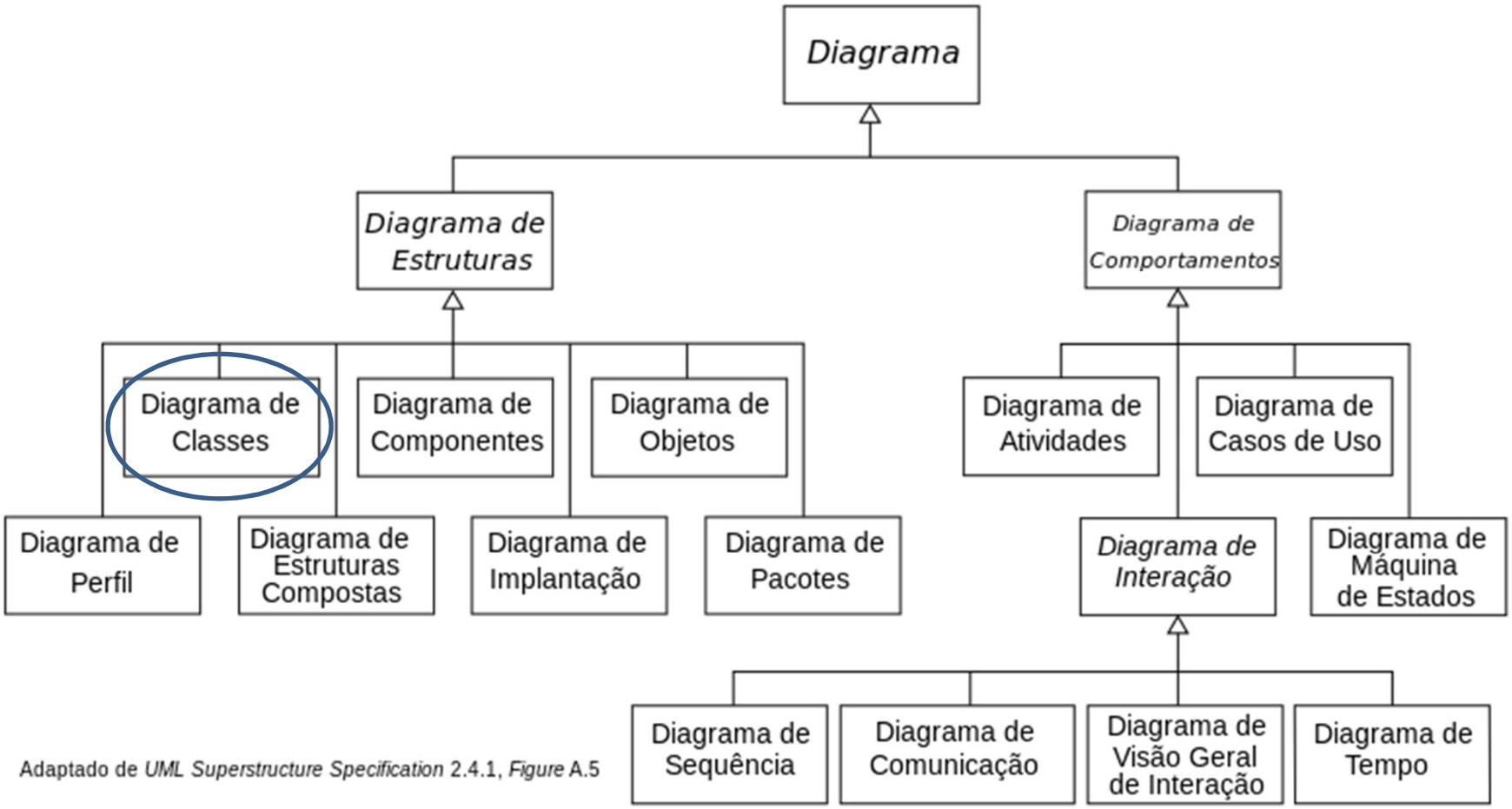
Tipos de Diagramas do UML

UML 2 possui 14 tipos de diagramas divididos em duas categorias:

- **Diagramas Comportamentais**
  Enfatizam o que precisa acontecer no sistema modelado.

  - Diagrama de Caso de Uso
  - Diagrama de Transição de Estados (ou de Estados)
  - Diagrama de Atividade
  - Diagrama de Sequência
  - Diagrama Visão Geral de Interação (ou de Interação)
  - Diagrama de Colaboração (ou Comunicação)
  - Diagrama de Tempo (ou Temporal)

Tipos de Diagramas do UML



Adaptado de *UML Superstructure Specification* 2.4.1, *Figure* A.5

UML: ***Diagrama de Classes***

Diagramas de Classes são uma "foto" das classes de um Sistema Orientado a Objetos. Contém os campos/atributos, métodos e conexões/relacionamentos entre as classes.

UML: ***Diagrama de Classes***

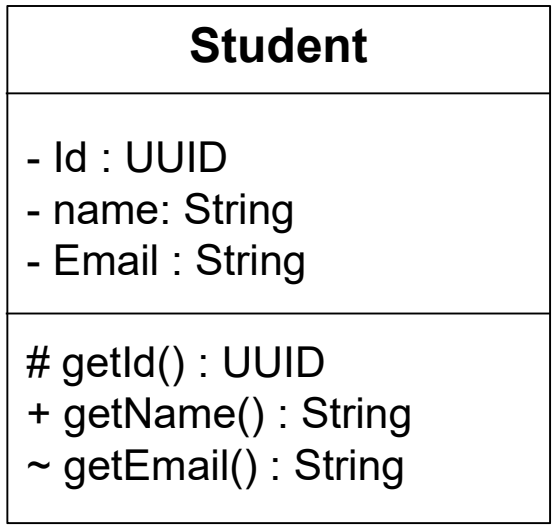
O que não é representado em um diagrama de classes?

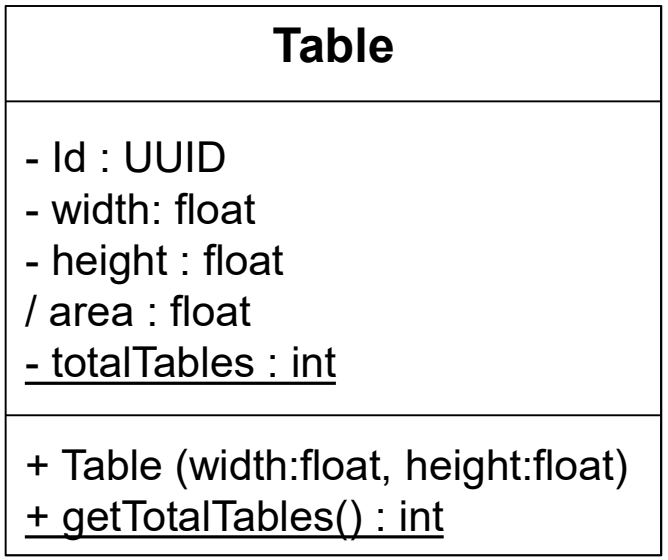
- Detalhes de como as classes interagem
- Algoritmos – como os comportamentos são implementados
- Métodos triviais podem ser omitidos (get/set)
- Classes que vem de bibliotecas também podem ser omitidas (Ex: ArrayList)

UML: ***Diagrama de Classes***

Nome da **classe** fica no topo da caixa

- **<<interface>>**
- Nome da classe em *itálico* caso seja ***abstrata***





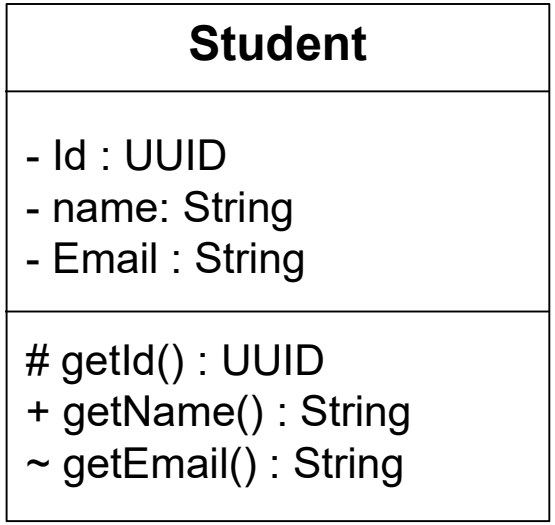
UML: ***Diagrama de Classes***

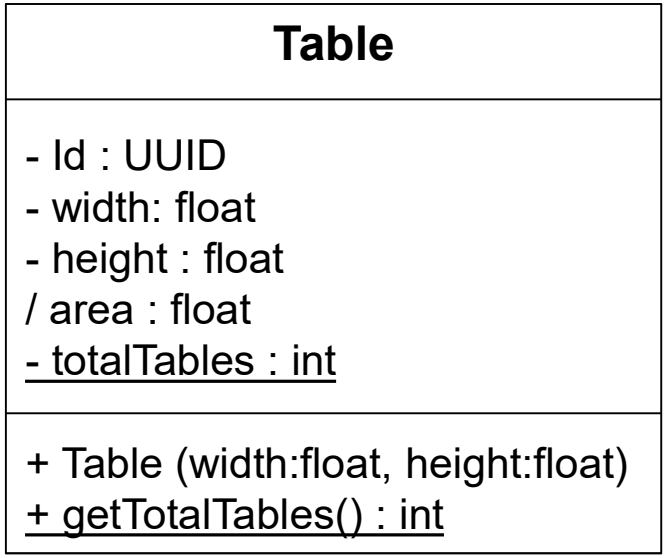
Atributos

- Incluem todos os campos de um objeto.
- Incluem propriedades “derivadas” de propriedades.

Operações/Métodos/Funções

- Podem ser omitidos métodos triviais (com exceção de interfaces)
- Não deve conter métodos herdados.





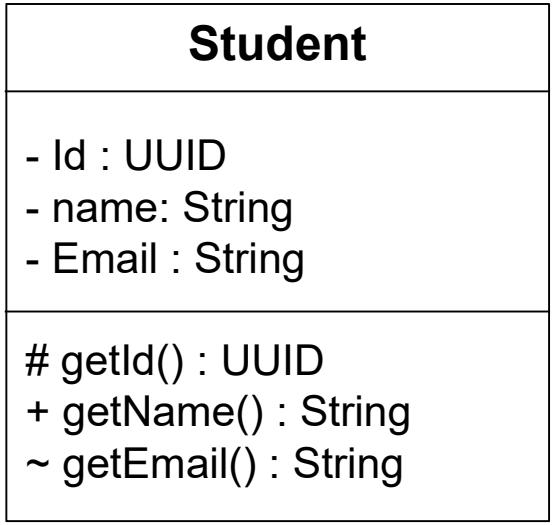
UML: ***Diagrama de Classes***

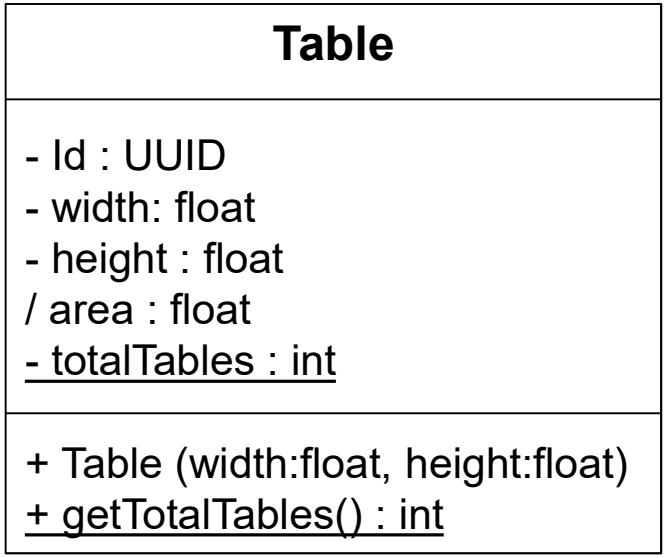
Sintaxe de Atributos

- nome: tipo [count] = defaultValue

Símbolo	Visibilidade
+	public
#	protected
-	private
~	package (default)
/	derived

Atributos <u>sublinhados</u> são **estáticos**.





UML: ***Diagrama de Classes***

Sintaxe de Métodos

- nome (parâmetros): tipo de retorno

Parâmetros são representados no formato:

nome: tipo

Omitir retornos do tipo *void*

Métodos <u>sublinhados</u> são **estáticos**.

Student
- Id : UUID - name: String - Email : String
getId() : UUID + getName() : String ~ getEmail() : String

Table
- Id : UUID - width: float - height : float / area : float <u>- totalTables : int</u>
+ Table (width:float, height:float) <u>+ getTotalTables() : int</u>

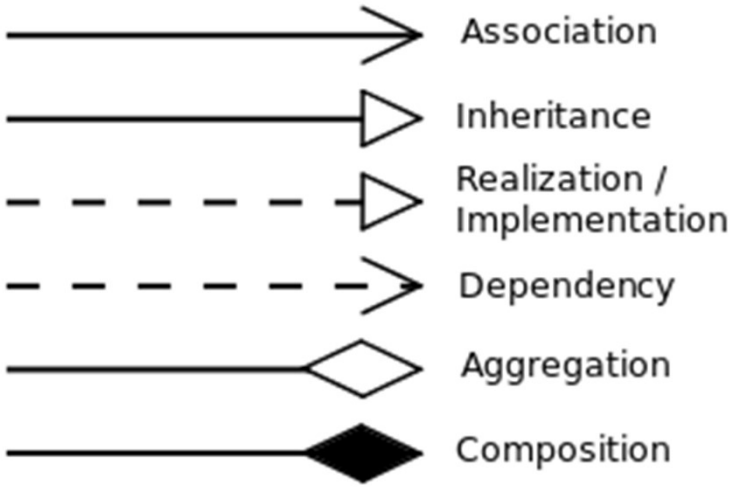
UML: ***Diagrama de Classes***

Generalização: relação de herança

- Herança entre classes
- Implementação de Interface

Associação: relação de uso

- Dependência
- Agregação
- Composição

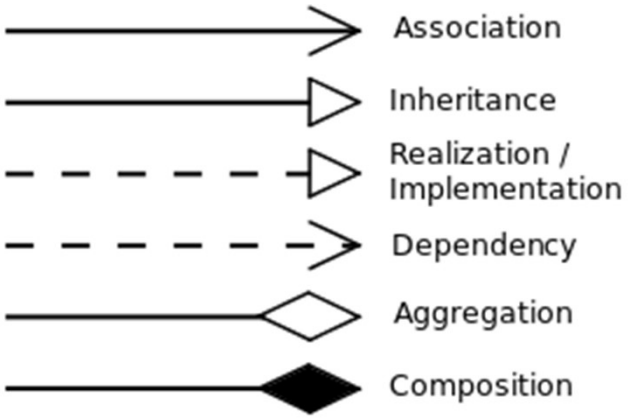


Fonte: Free licensed media from Commons

UML: ***Diagrama de Classes***

Generalização

Parent	Estilo da Linha/Seta
class	sólida, seta preta
abstract class	sólida, seta branca
interface	tracejada, seta branca



Fonte: Free licensed media from Commons

Relações óbvias geralmente não são representadas. Exemplo: Java Object

UML: ***Diagrama de Classes***

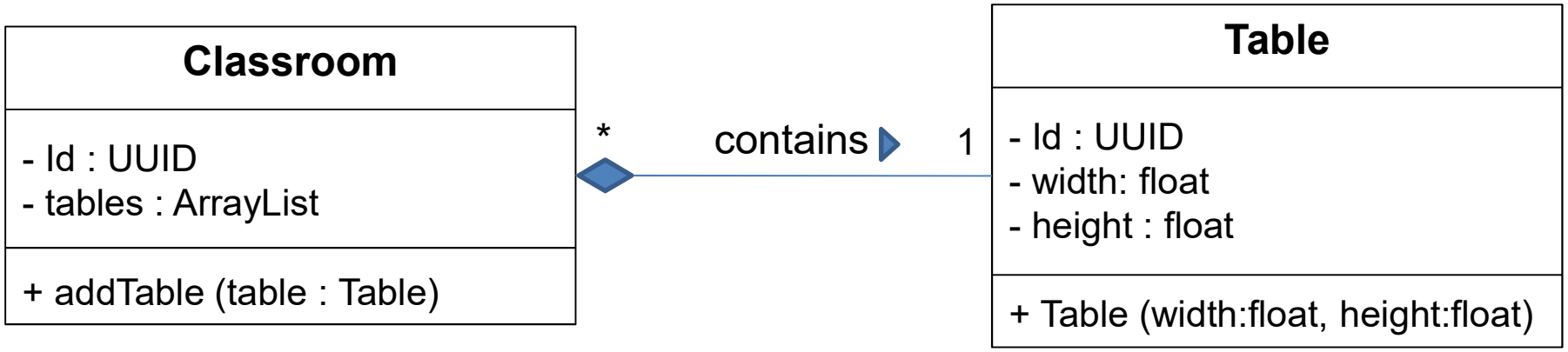
Associação

Multiplicidade

Símbolo	Quantidade
*	0, 1, ou mais
1	exatamente 1
2..4	entre 2 e 4
5..*	5 ou mais

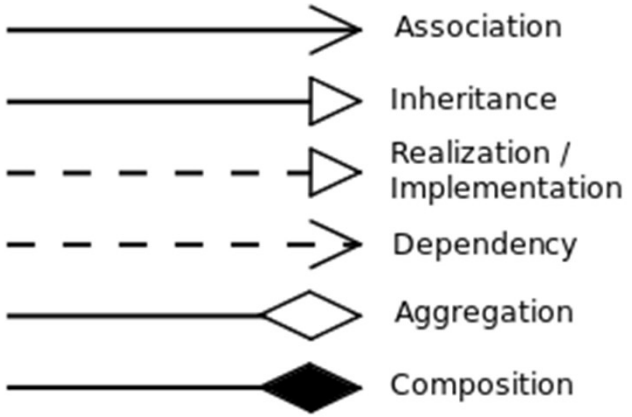
Nome
Relações que o objeto possui.

Navegabilidade
Direção



UML: ***Diagrama de Classes***

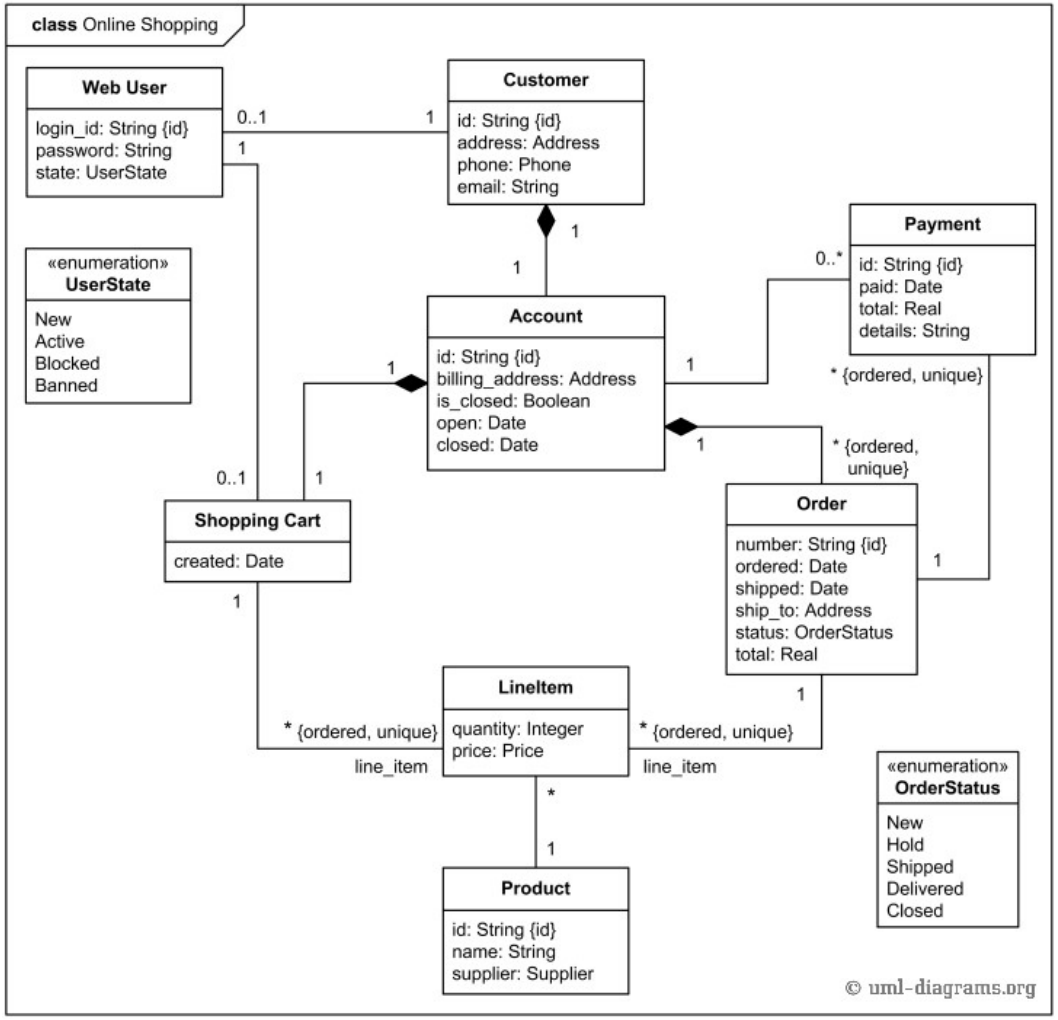
Tipo de Associação



Fonte: Free licensed media from Commons

- **aggregation:** "é parte de"
  - Diamante branco
- **composition:** "é totalmente feito de"
  - Versão mais forte da agregação/aggregation
  - Só pode existir se o outro existe
  - Diamante preto
- **dependency:** "usa temporariamente"
  - Seta tracejada ou uma linha

UML: ***Diagrama de Classes***



http://www.uml-diagrams.org/examples/online-shopping-domain-uml-diagram-example.html?context=cls-examples

Design Patterns

Padrões de Projeto (*Design Patterns*)

Padrões de projeto são soluções gerais para problemas que ocorrem com frequência dentro de um determinado contexto no projeto de software.

Não são projetos finalizados que podem ser transformados diretamente em código fonte. **São descrições ou modelos (*templates*) que se propõem a resolver diferentes problemas.**

São práticas formalizadas que o programador pode utilizar para resolver esses problemas comuns.

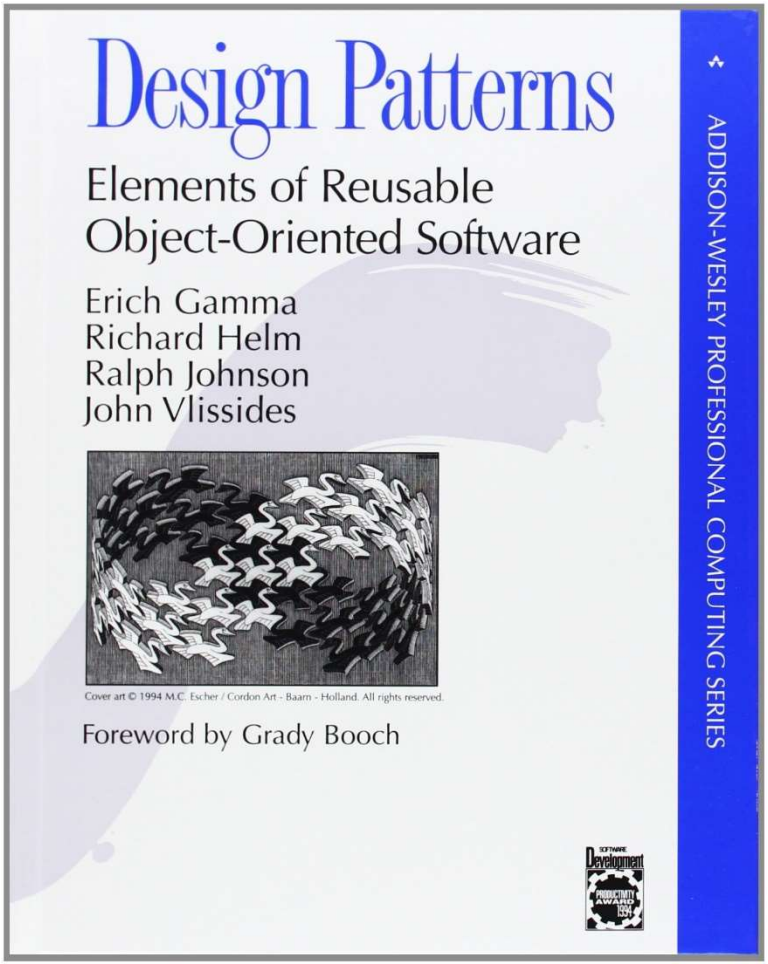
Padrões de Projeto (*Design Patterns*)

- **Padrões GOF**
  G*ang of Four*

- **Padrões GRASP**
  General Responsibility Assignment Software Patterns (or Principles)

Design Patterns: **Tipos de Padrões GOF (Gang of Four)**

- Padrões de Criação
- Padrões Estruturais
- Padrões Comportamentais



Design Patterns: **Tipos de Padrões GOF (Gang of Four)**

Criação
Abstract Factory
Builder
Factory Method
Prototype
Singleton

Estrutural
Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Proxy

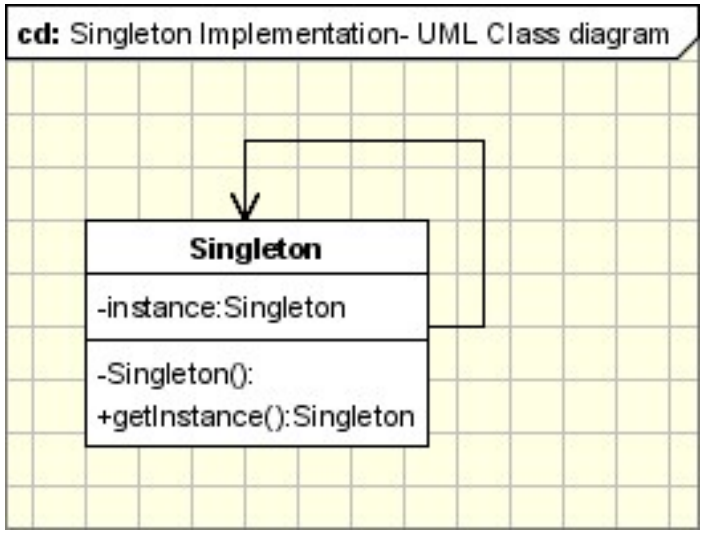
Comportamental	
Chain of Responsibility	State
Command	Strategy
Interpreter	Template Method
Iterator	Visitor
Mediator	
Memento	
Observer	

Design Patterns: ***Singleton***

Problema
Uma classe precisa ter uma única instância.

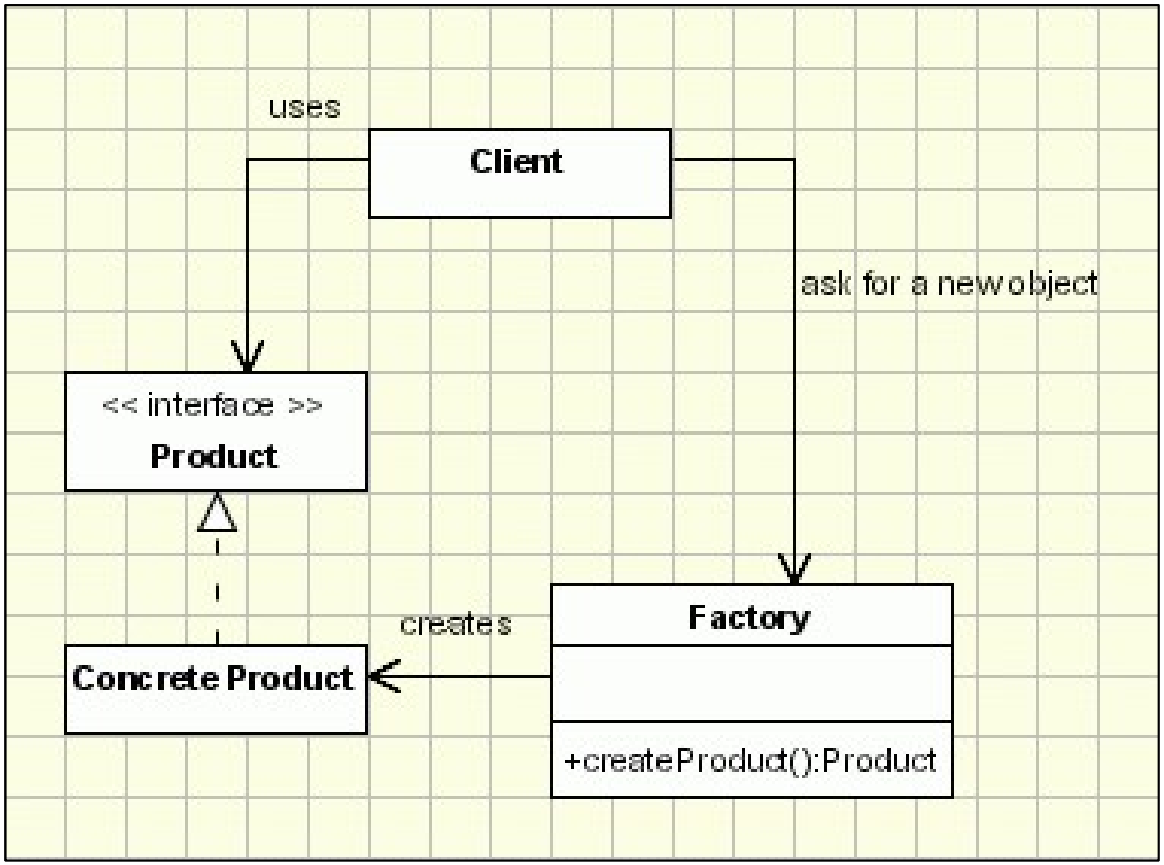
Solução
Garante que uma classe terá apenas uma instância.

Consequência
Fácil acesso a gerência de recursos compartilhados, como variáveis globais.



http://www.oodesign.com/singleton-pattern.html

Design Patterns: ***Factory Method***



http://www.oodesign.com/factory-pattern.html

Design Patterns: **Tipos de Padrões GRASP**
General Responsibility Assignment Software Patterns (or Principles)

Assim como os padrões de projeto do GOF, os padrões GRASP são utilizados para resolução de problemas comuns e bastante típicos de desenvolvimento de software orientado a objeto. Portanto, tais técnicas apenas documentam e normatizam as práticas já consolidadas, testadas e conhecidas no mercado.

Design Patterns: **Tipos de Padrões GRASP**
General Responsibility Assignment Software Patterns (or Principles)

GRASP	
Controller	Polymorphism
Creator	Protected Variations
Indirection	Pure Fabrication
Information Expert	
High Cohesion	
Loose Coupling	

Importância dos Design Patterns

Design Patterns

- *Visam facilitar a reutilização de soluções de desenho – isto é, soluções na fase de projeto do software.*

- *Estabelecem um vocabulário comum de desenho, facilitando comunicação, documentação e aprendizado dos sistemas de software.*