Engenharia de Software

Tecnologia de Software

Aula 4: Arquitetura e projeto (*design*)

Fábio Levy Siqueira
levy.siqueira@usp.br

Desenvolvimento

- Como criar o software a partir dos requisitos?
  - Quais componentes devem ser criados?
  - Qual a responsabilidade de cada componente?
    - Requisitos funcionais e não funcionais
    - Organização que facilite a evolução do software
  - Qual a relação entre os componentes?

- É possível codificar diretamente?

Arquitetura de software

Arquitetura de software

- O que é arquitetura de software?
  - *Design* == Arquitetura?

> Conceitos ou propriedades fundamentais de um sistema em seu ambiente, encarnadas em seus elementos, relações e nos princípios de seu projeto e evolução (ISO, 2011)

- É uma *abstração*
- Em geral há uma preocupação com os RNFs

Arquitetura de software

- Importância (Bass, Clements e Kazman, 2012)
  - Permite ou **restringe** características de qualidade
  - Permite predizer a qualidade do sistema
  - Restringe o ***design*** e a **implementação**
    - Encaminha decisões de design
      - Como o sistema será distribuído?
      - Como o sistema será estruturado?
      - Como os componentes do sistema serão decompostos?
      - Qual a melhor organização para tratar dos RNF?
      - Existe alguma arquitetura genérica que pode ser usada como base?
  - Permite incorporar componentes desenvolvidos independentemente

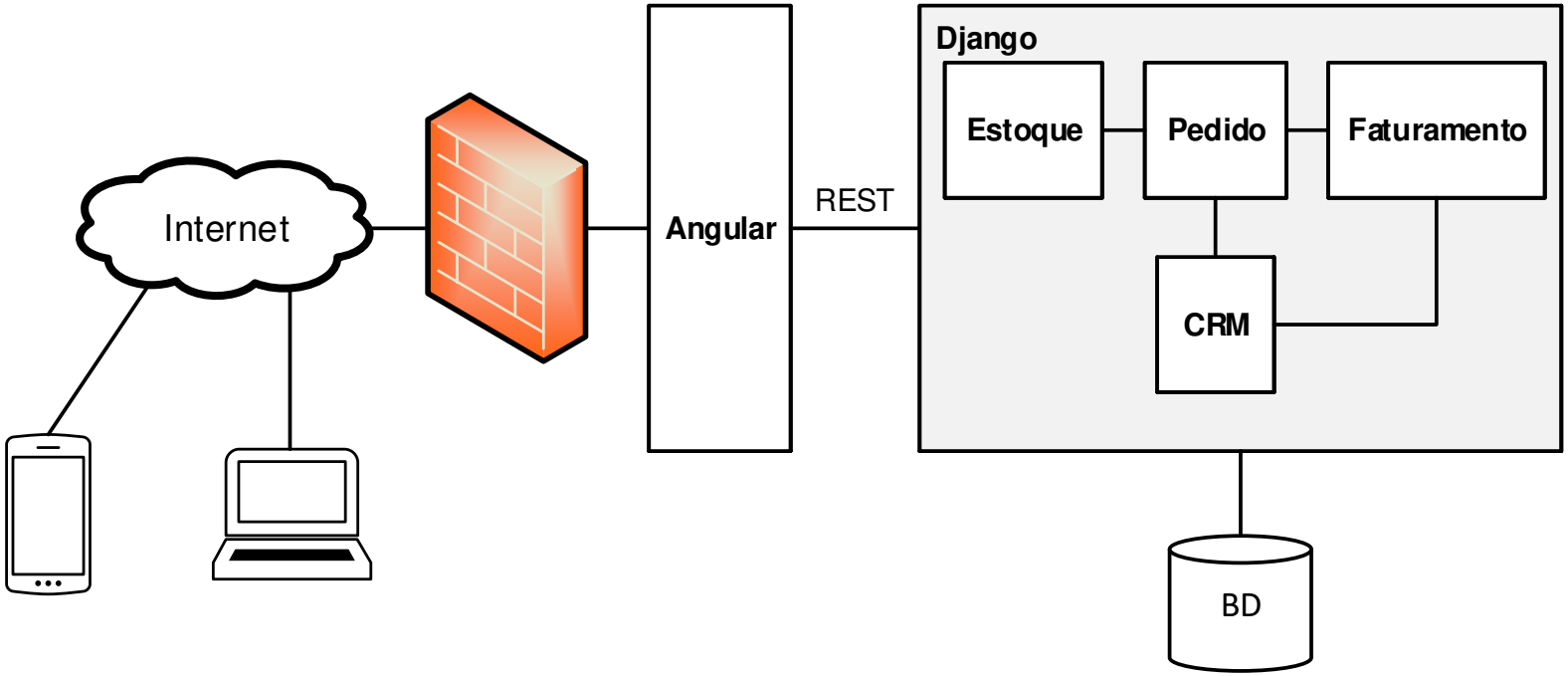
Arquitetura de software

- Importância (continuação)
  - Ajuda a pensar sobre mudanças
  - Melhora estimativas de custo e tempo
  - Influencia a divisão do trabalho
  - Melhora a comunicação com stakeholders
    - Cada stakeholder tem uma preocupação diferente

De forma geral o objetivo de uma boa arquitetura é **minimizar** o custo (em toda a vida do software) e **maximizar** a produtividade (Martin, 2017)

Arquitetura de software

- Como representar a arquitetura de software?
  - **Cuidado**: Arquitetura ≠ Descrição da Arquitetura
- Essa é uma boa representação da arquitetura?



Descrição

Dificilmente um único diagrama conseguiria representar tudo e ser fácil de ler e manter (Pfleeger e Atlee, 2010)

- Uso de descrições arquiteturais (ISO, 2011)
  - Base para as atividades de *design* e implementação
  - Base para analisar alternativas de implementação
  - Documentação de aspectos essenciais do sistema
  - Entrada para ferramentas de simulação, análise e geração de código
  - Comunicação entre *stakeholders*
  - Guia para o suporte da operação e da infraestrutura
  - Suporte para o planejamento, incluindo prazos e custos
  - ...

Visão

- A descrição da arquitetura *inclui* várias visões
  - Representação da arquitetura na perspectiva de um conjunto de **preocupações** (ISO, 2011)
    - Preocupações dos stakeholders
  - *Exemplo*: visões na engenharia civil
    - Encanamento, fiação elétrica, planta...

- Visões podem ser definidas para o projeto ou pode-se usar um framework
  - **Arcabouço de Zachman**: trata da arquitetura empresarial
  - **RM-ODP**: padrão para sistemas distribuídos e abertos
  - **Modelo 4+1**: modelo OO

Padrões de arquitetura

- Padrões ajudam a definir a arquitetura
  - **Padrões**: essência de soluções que podem ser usadas em problemas semelhantes
    - Capturam a experiência
  - Linguagem básica
  - *Exemplo*
    - Modelo em camadas
    - Model-View-Controller (MVC)
    - Dutos e filtros (*pipes and filters*)
    - Proxy

Na prática

- Todo software possui uma arquitetura?

- É fácil mudar uma arquitetura?

- É necessário representar a arquitetura?

Na prática

- Representação em métodos ágeis
  - *A arquitetura deve emergir*
    - *(Mas é difícil mudar após tomar decisões importantes)*
  - Em casos simples *não é necessário* representar
    - Software arquiteturalmente *comum*
    - Arquitetura e decisões pré-definidas
  - Mas e os outros casos?

Na prática

- Representação da arquitetura em métodos ágeis
  - Discussão entre desenvolvedores
  - Divulgação para o time (e para os mantenedores)
    - Não precisa ser um documento formal
  - Decisões arquiteturais devem ser documentadas
  - Um bom arquiteto deve definir políticas

> Um bom arquiteto maximiza o número de decisões não feitas
> (Martin, 2017, p.142)

Projeto (*design*)

Projeto (*design*)

- Objetivo

  Prover dado e informação suficientemente detalhados sobre o sistema e seus elementos para permitir a implementação consistente com as entidades arquiteturais (ISO, 2017)

- Atividade extremamente técnica
  - Conhecimento dos requisitos, da arquitetura, do código e dos testes
  - Proficiência em
    - Linguagem de programação
    - *Frameworks* e bibliotecas
    - Tecnologias em geral
    - Conceitos, princípios e padrões de *design*

Projeto (*design*)

- Ainda faz sentido falar em projeto?
  - Em métodos ágeis não é só codificação?

Projeto (*design*)

- Cada abordagem trata o projeto de maneira diferente
  - *Ex.*: *Extreme Programming*
    - Várias práticas apoiam o Projeto
    - *Exemplo*: design incremental (eliminar duplicação), TDD, cartões CRC (XP1), refatoração, todo time
    - Tradicionalmente é feita diretamente no código

- Algumas abordagens sugerem o uso de **modelos**
  - Mesmo abordagens ágeis
  - Permitem simplificar a comunicação e facilitam pensar sobre a solução
  - *Exemplo*: FDD, DDD e UP

Projeto dirigido por modelos

- Quais os problemas de usar modelos no *design*?

Projeto dirigido por modelos

- Devemos usar modelos no projeto?
  - Se sim, como?
  - Quão detalhado deve ser o modelo?

  - **Cuidado**: A menos que o modelo seja um rascunho, ele precisa ser mantido

Paradigma

- Em geral o projeto (ou parte dele) considera um paradigma de programação

> Forma de conceituar o que significa realizar computação e como tarefas executadas no computador devem ser estruturadas e organizadas. (Budd, 2001)

Paradigma

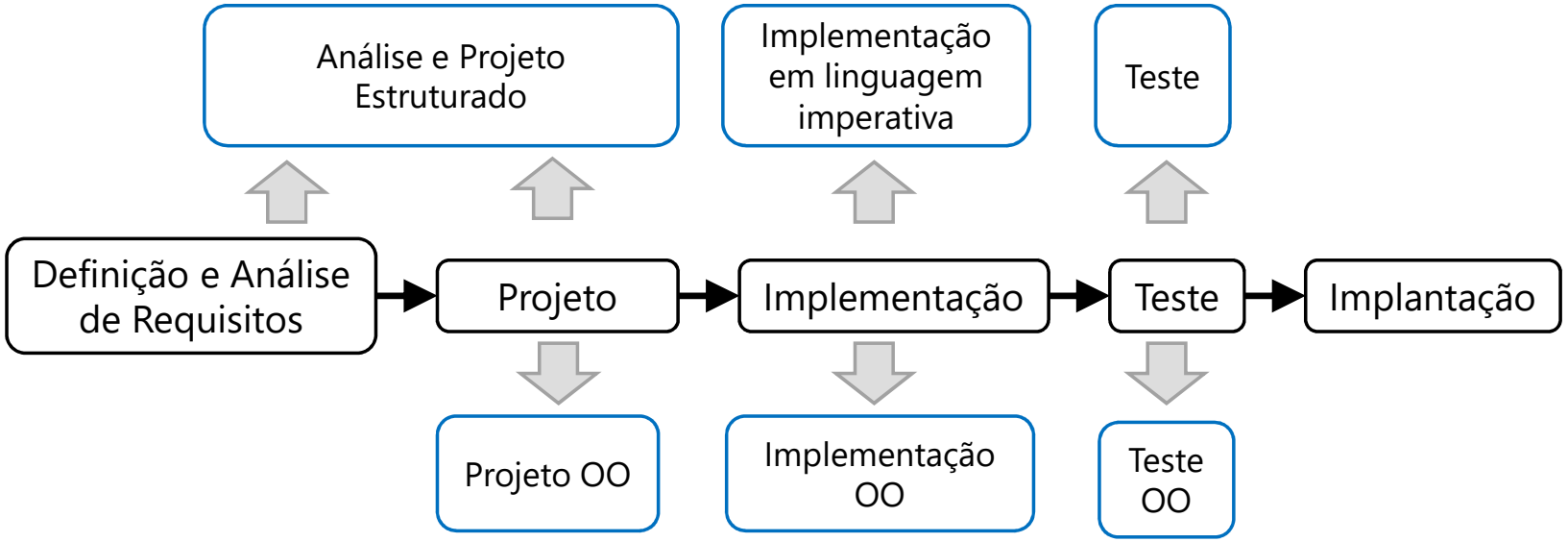
- Alguns paradigmas
  - Imperativo: foco no estado e comandos de mudanças de estado do programa (global)
    - Linguagens: Pascal, C e Cobol
  - Funcional: algoritmos (ponto de vista matemático)
    - Operações sobre listas
    - Linguagens: Lisp, Haskel, ML e Scala (também é OO)
  - Lógico: metas e lógica de predicados
    - Linguagens: Prolog
  - Orientado a objetos: conceitos do domínio do problema
    - Linguagens: C++, C#, Java, Objective-C, Ruby
- Algumas linguagens são multiparadigma

Paradigma

- Qual paradigma seguir?

Paradigma

- Definição dos requisitos é conceitualmente independente de paradigma
  - Ela pode ser feita independente de paradigma
  - No projeto se pode decidir o paradigma adequado
- Alguns métodos cobrem a análise e o projeto



Projeto Orientado a Objetos

Projeto OO

- Historicamente existiam vários métodos
  - Booch, Objectory, OMT, Fusion, Shlaer-Mellor etc.
  - *Processo Unificado*

- Atualmente
  - Processos de desenvolvimento
    - *Exemplo*
      - *Rational Unified Process*
      - ICONIX
      - *Feature Driven Development*
  - *Princípios e padrões de projeto*
    - GRASP, SOLID, GoF etc.

Orientação a objetos

- Foco nos conceitos do domínio do problema
  - Software é decomposto em conceitos
  - Dados e funções são organizadas por conceitos
    - Dados ficam escondidos
  - *Instâncias* dos conceitos colaboram para executar uma função do software

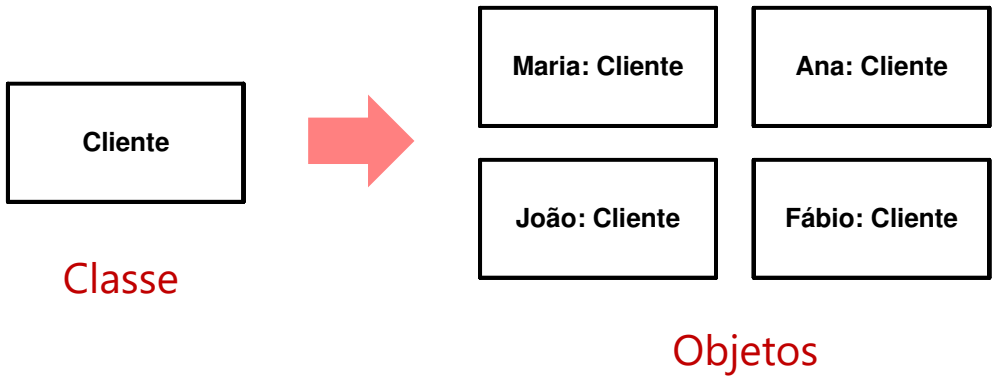
- *Exemplo*: conceitos em uma loja virtual

Orientação a objetos

- Terminologia básica (Armstrong, 2006)
  - Abstração
  - Classe
  - Encapsulamento
  - Herança
  - Objeto
  - Mensagem
  - Operação / método
  - Polimorfismo

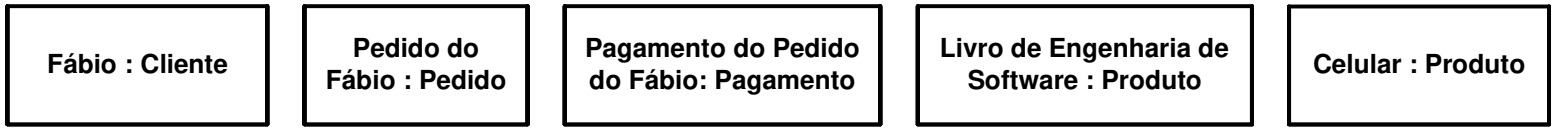
Orientação a objetos

- Conceitos são representados como **classes**
  - Definem um novo tipo
  - São instanciadas ("materializadas") em objetos

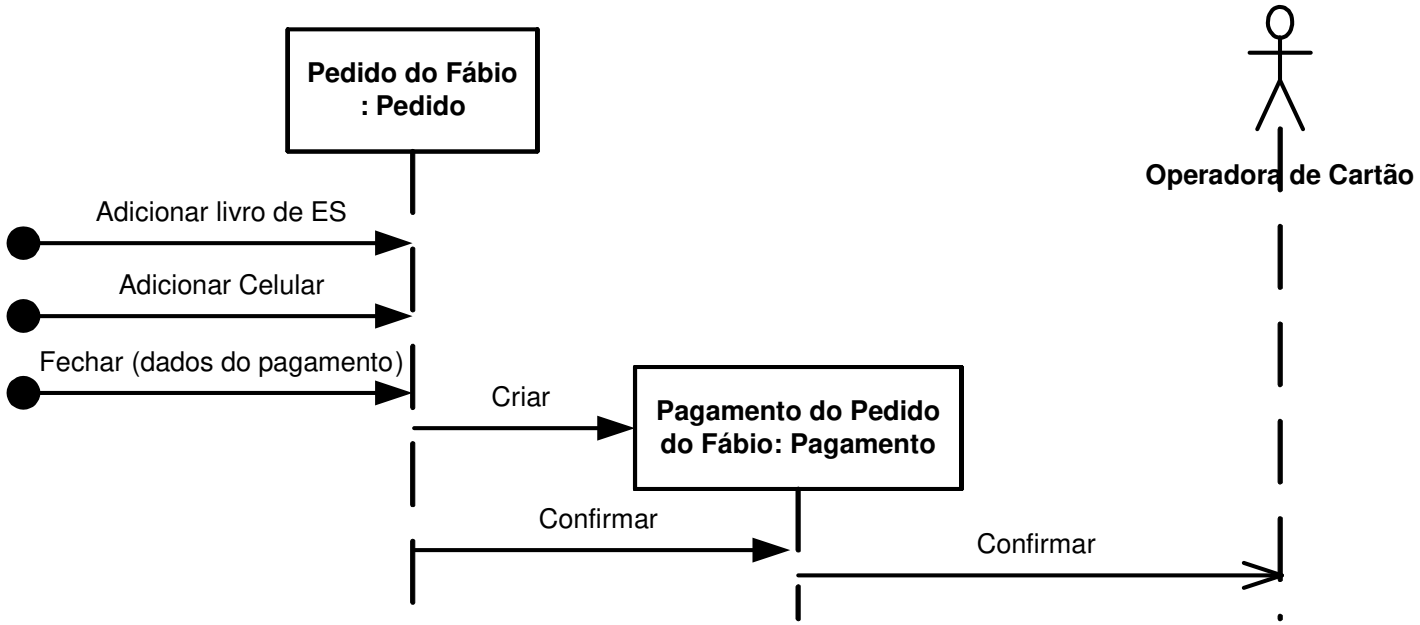


Orientação a objetos

- O software é um conjunto de objetos



  - Os objetos trocam *mensagens* para realizar algo

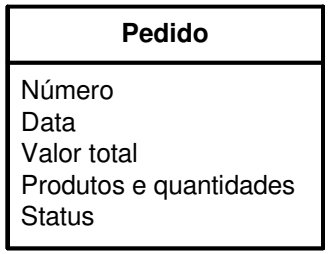


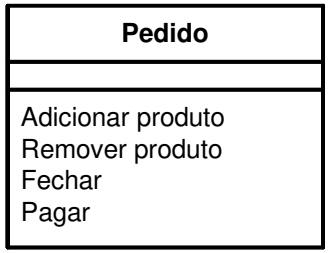
Orientação a objetos

- Classes definem *estrutura* e *comportamento* comum para um conjunto de objetos

  - **Estrutura**: atributos
    - Propriedades dos objetos
    - Definem o estado do objeto

  - **Comportamento**: operações
    - Serviços disponibilizados pelos objetos





Representação

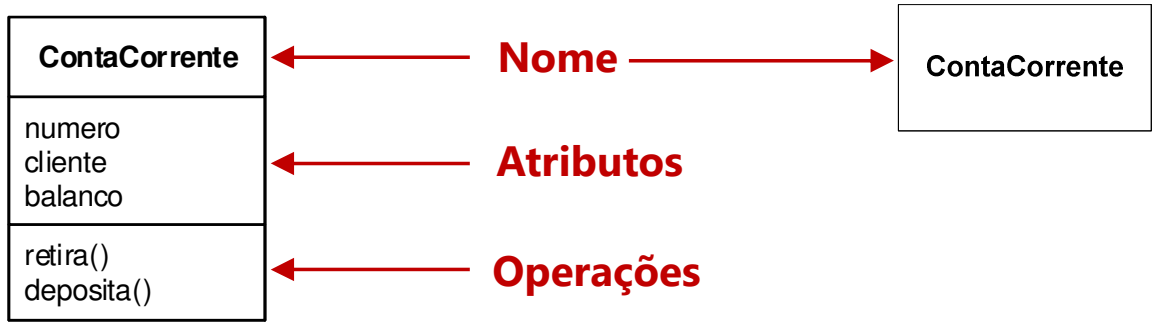
- Uma forma simples de representar as classes é através do **diagrama de classes** da UML
  - Linguagem para elaborar a estrutura de sistemas
    - Modelagem gráfica
    - *Principalmente* softwares Orientados a Objetos
  - Define 14 diagramas...

  - Abordagens dirigidas a modelos em geral usam a UML
    - Cada abordagem define como serão produzidos os diagramas
    - *Exemplo*: DDD, ICONIX e RUP

  - Veremos uma visão ***bem*** geral do diagrama de classes

Classes

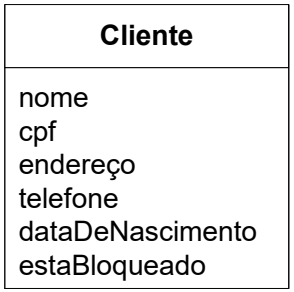
- Descrição de um conjunto de objetos com características comuns
  - (Pode ter outros *compartimentos*)

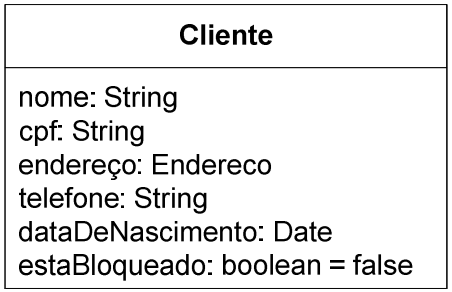


- **Nome**: substantivo e singular
  - Representa um conceito

Atributos

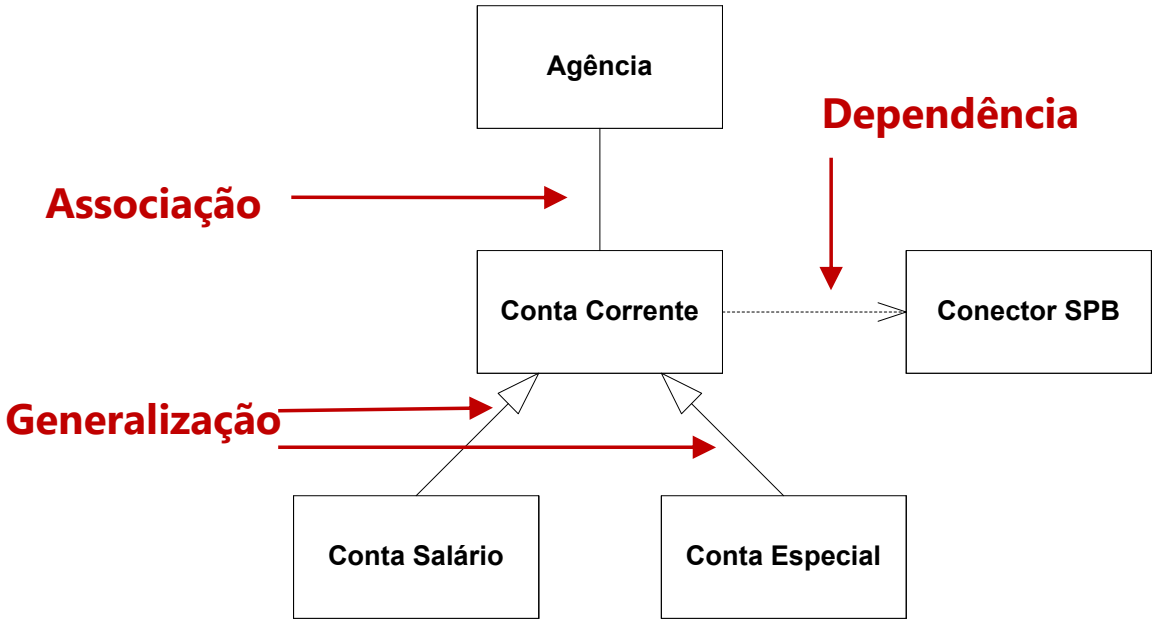
- Abstração dos estados: propriedade
- Intervalo de valores que uma propriedade pode apresentar
  - Nome
  - Tipo (*opcional*)
  - Valor padrão (*opcional*)





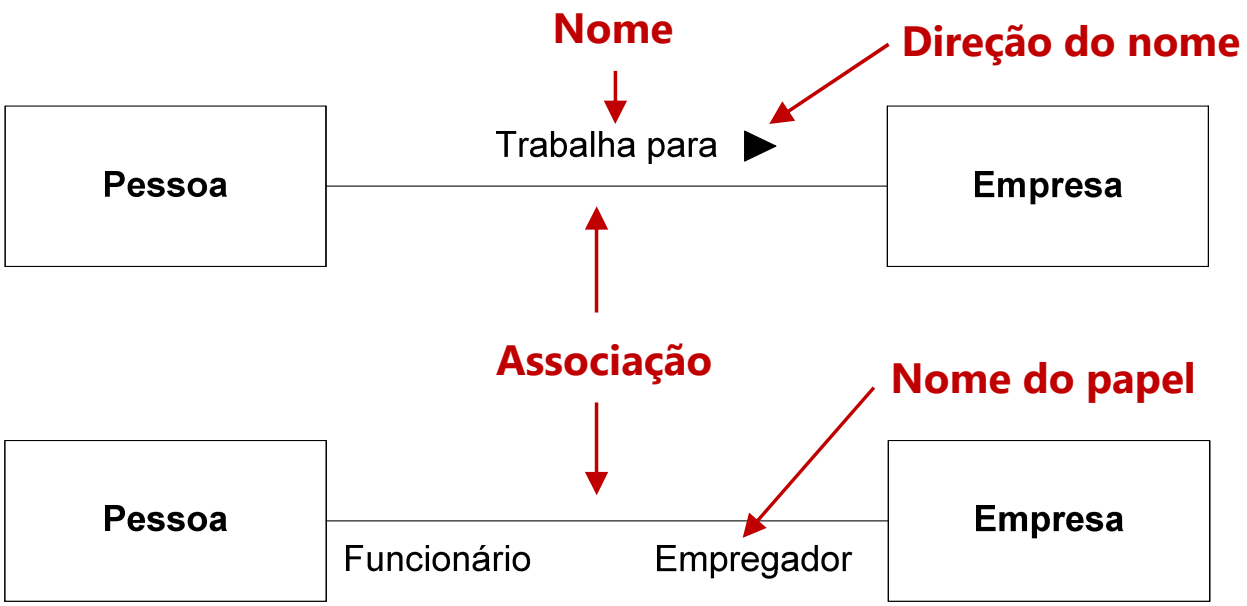
Relacionamento

- Forma de representar a colaboração entre classes
  - Associação
  - Generalização / especialização
  - Dependência



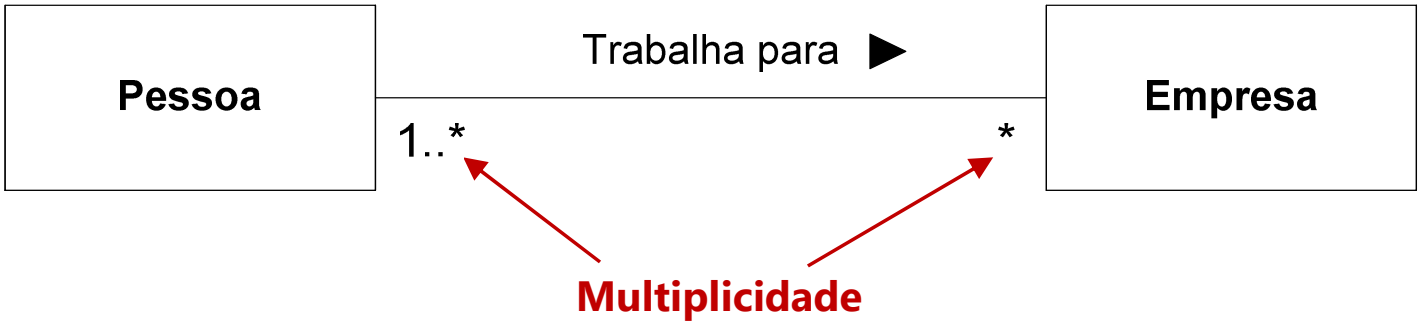
Associação

- Objetos de uma classe estão conectados a objetos de outra classe
  - Representam requisitos de informação



Associação

- Multiplicidade
  - Número de **objetos** que podem estar envolvidos na relação em um determinado ponto do tempo
    - É uma **faixa de valores**



(Uma pessoa trabalha para 0 ou mais empresas e uma empresa tem uma ou mais pessoas)

  - Não há uma multiplicidade padrão

Associação

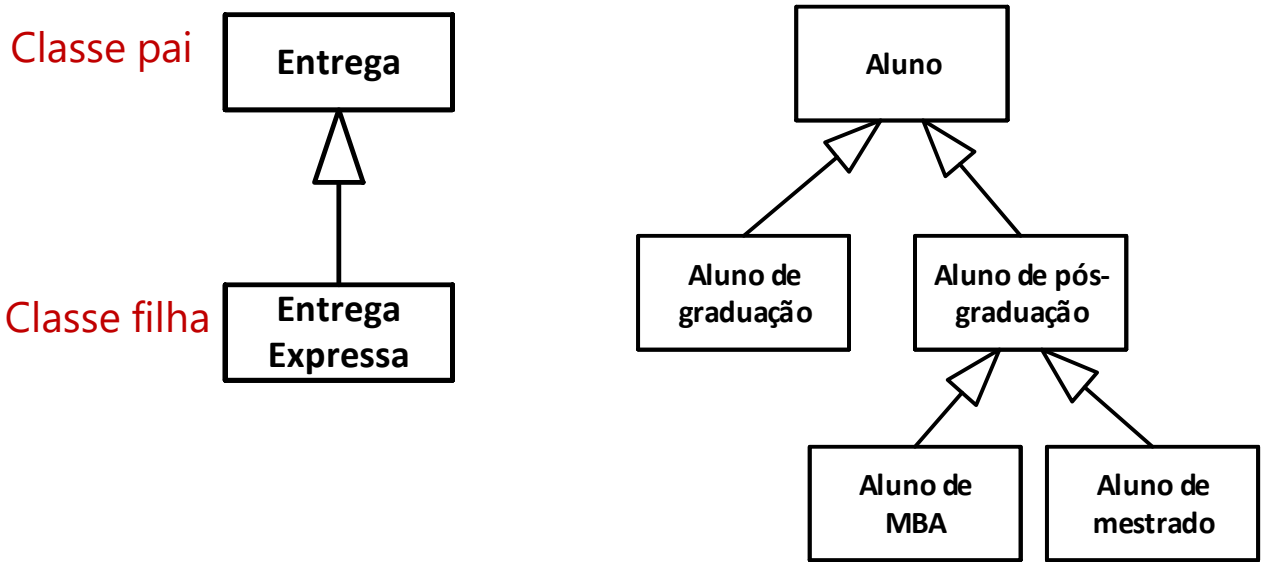
- Multiplicidade: valores

Valor	Representação	Exemplo
Valor fixo	número	1 (apenas 1 objeto) 3 (obrigatoriamente 3 objetos)
Faixa de valor	mínimo..máximo	1..5 (de 1 a 5 objetos) 1..* (1 ou mais objetos) 0..* (0 ou mais objetos)

  - Observação: `*` e `0..*` são equivalentes

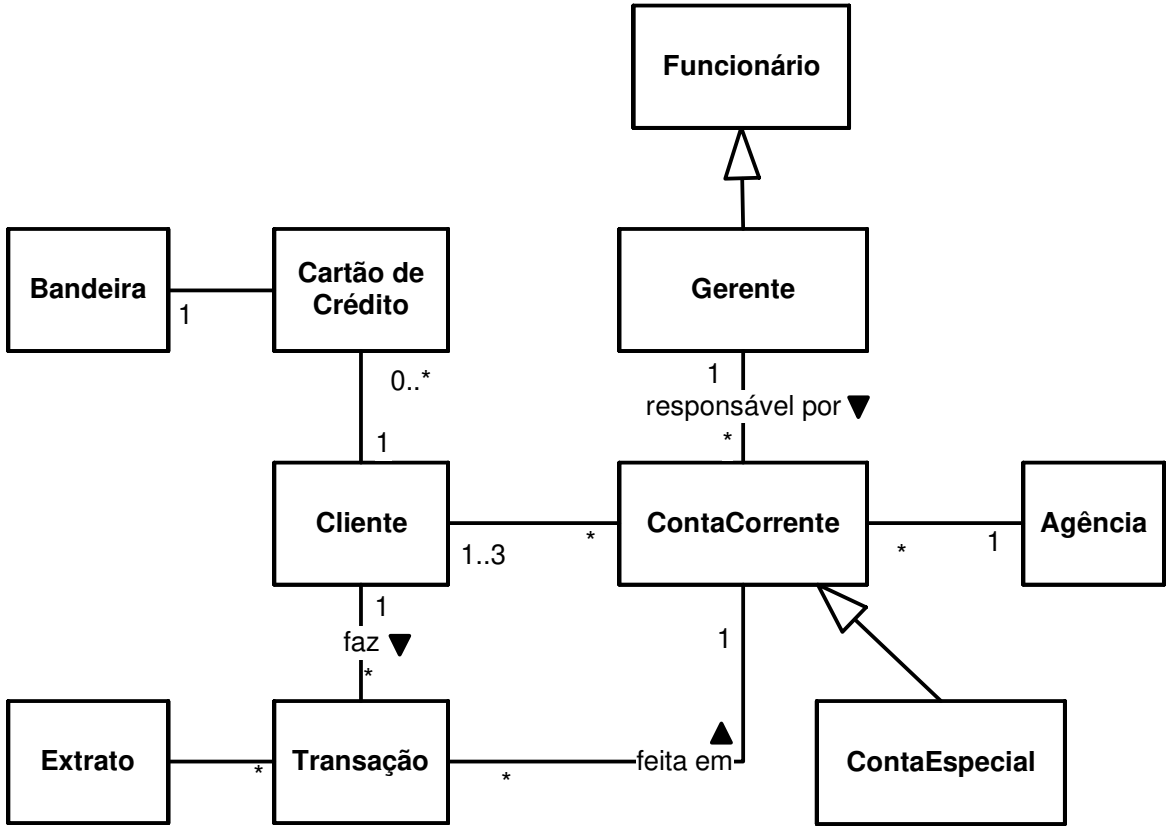
Generalização / especialização

- Classe pode ser uma *especialização* de uma outra
  - Herda operações e atributos
  - Pode definir outras operações e atributos
  - *Exemplo*



Exemplo

- Classes de um sistema bancário
  - Defina multiplicidade e nomes (sempre que possível)



Referências

- ARMSTRONG, D. **The Quarks of Object-Oriented Development**. Communications of the ACM. v.49, n.2, p.123-128, February, 2006.
- BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3ª edição. Addison-Wesley, 2012
- BUDD, T. **An Introduction to Object-Oriented Programming**. 3rd Edition. Addison-Wesley. 2001.
- FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. Addison-Wesley, 3a edição, 2003.
- ISO/IEC/IEEE. **Systems and software engineering - Architecture description**. ISO/IEC/ IEEE 42010 . 2011.
- MARTIN, R.C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Prentice Hall, 2017.
- PFLEEGER, S. L.; ATLEE, J. M. **Software Engineering: Theory and Practice**. 4. ed. Upper Saddle River: Prentice Hall, 2010.