Conceitos de Orientação a Objetos

"Modelo de análise, projeto e paradigma de programação baseado na interação entre objetos"

Orientação a Objetos

- OO é um paradigma (assim como tradicional ou aspectos)
- Existe desde 1962 (Simula que influenciou o Smalltalk)
- Muitas linguagens são primariamente OO.

Aprenda OO e aplique em qualquer linguagem

Java vs C#

```java
public class Produto {
    private int codigo;
    private String descricao;
    private float preco;
    public Produto() {
        this.codigo = 0;
    }
    public float getPreco() {
        return preco;
    }
    public void setPreco(float preco) {
        this.preco = preco;
    }
}
```

```csharp
public class Produto {
    private int codigo;
    private String descricao;
    private float preco;
    public Produto() {
        this.codigo = 0;
    }
    public float Preco() {
        get {
            return preco;
        }
        set {
            preco = value;
        }
    }
}
```

Classes

O que é uma classe em um sistema Orientado a Objetos?

Classes

- Representação de um conjunto de objetos
- Abstração de dados e ações de entidades do mundo real
- Mesmas características
- São uma espécie de Modelo/Matriz dos Objetos (logo falaremos destes), e porque não uma "classificação" destes

Possuem:

- Atributos
- Operações

Atributos

- Características da classe
- Estado da classe

Operações

- Funcionalidades
- Métodos
- Alteram o estado da classe

Visibilidade dos atributos e operações

private

Somente acessível pela própria classe

package

Acessível pelas classes do mesmo pacote

protected

Acessível pelas classes do mesmo pacote ou por subclasses

public

Acessível de qualquer outra classe

Visibilidade das classes

package

Acessível pelas classes do mesmo pacote

public

Acessível de qualquer outro pacote

Encapsulamento

- Atributos privados
- Acessíveis apenas através das Operações
- Uso de Getters e Setters (veremos depois no JEE)

Java Code Conventions

Normas de nomenclatura, identação, formatação, etc.

Nomenclatura de uma classe

- Substantivo no singular
- Primeira letra maiúscula e as demais palavras em CamelCase
  - `PessoaFisica` ou `MeioTransporte`
- Sem caracteres especiais ou espaços

Obs:

- Não pode haver classes com o mesmo nome no mesmo local (no mesmo pacote)

Nomenclatura de um atributo

- Normalmente são substantivos
- Primeira letra minúscula e as demais palavras em CamelCase
  - `valorTotal` ou `quantidadeEncomenda`
- Sem caracteres especiais ou espaços

Obs:

- No caso de uma constante, utilizar apenas maiúsculas e `_`
  - `VALOR_MAXIMO_PERMITIDO`

Nomenclatura de uma operação

- Normalmente possuem verbos no infinitivo
- Geralmente se omitem as preposições
  - `calcularValorNota`
- Primeira letra minúscula e as demais palavras em CamelCase
  - `calcularValorFinal` ou `verificarSituacao`
- Sem caracteres especiais ou espaços

Obs:

- No caso de uma constante, utilizar apenas maiúsculas e `_`
  - `VALOR_MAXIMO_PERMITIDO`

Para mais detalhes

Oficial:

https://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Google:

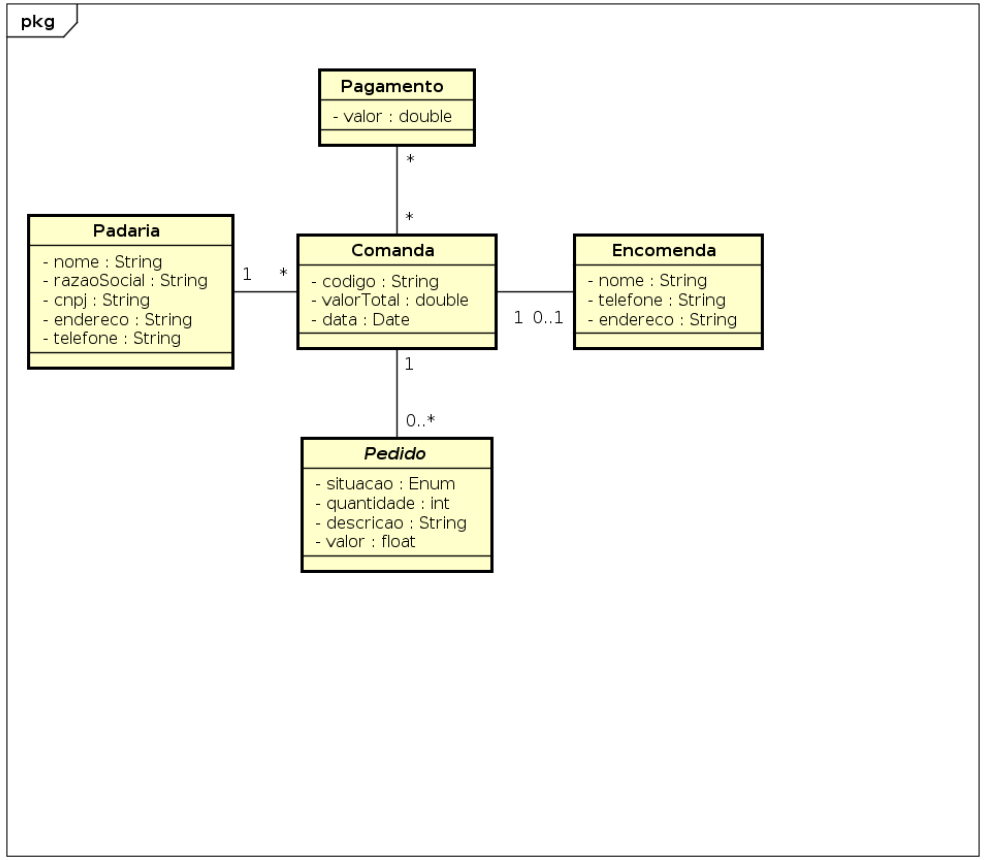
https://google.github.io/styleguide/javaguide.html

UML e o Diagrama de Classes

Unified Modeling Language e seu diagrama mais utilizado em OO

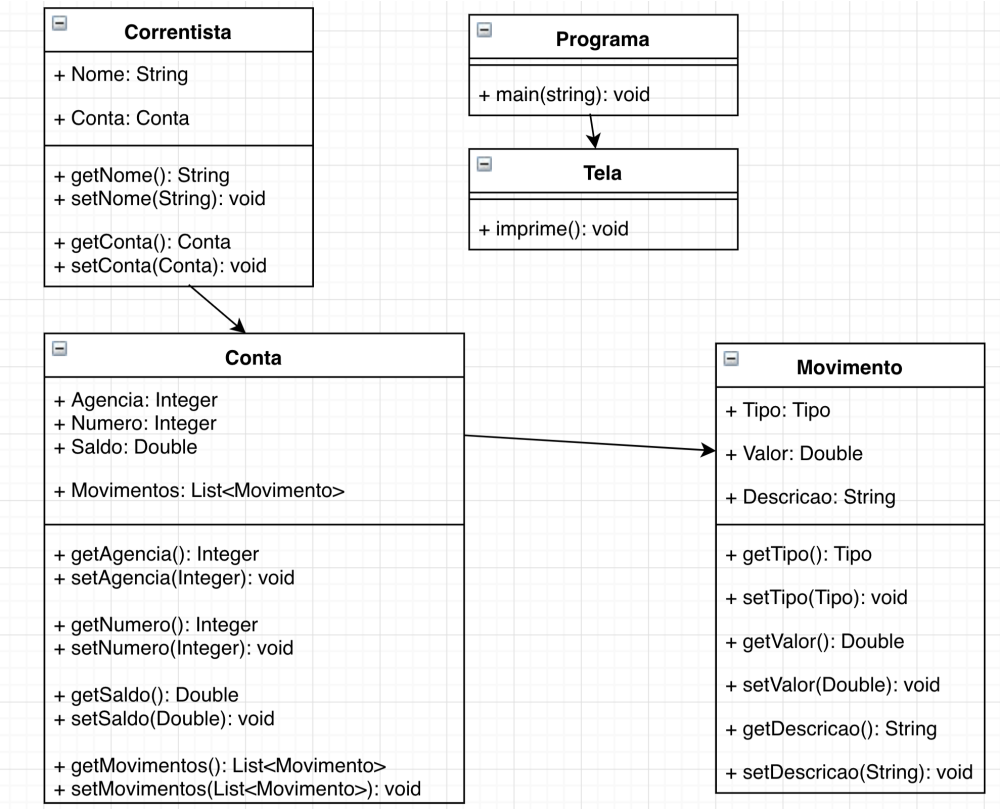
Diagrama de Classes

- Diagrama que contempla detalhes de uma especificação OO
- Perspectivas: Conceitual, Especificação e Implementação
- Exemplo de representação de Classe:



Exercício exercicio-banco - parte 1

Criar um novo projeto chamado "exercicio-banco" com as classes:



Exercício exercicio-banco - parte 1

Após criar as classes do diagrama conforme UML:

- O programa é a classe que contará o método `main`
- As classes Tela serão as opções do menu
- Criar Movimento, informar valores e imprimir. Criar método descricaoCompleta.

Objetos

O que diferencia uma Classe de um Objeto?

Objetos

- São gerados a partir de Classes
  - Uma Classe pode gerar N objetos (instâncias)
- Possuem valores nos atributos, específicos de cada instância
- Comunica-se com outros objetos
  - Utilizando "mensagens", através de Operações

Construtores

- Utilizamos o construtor de uma classe concreta para instanciarmos uma classe, com a palavra `new`
- Construtores podem possuir argumentos e passam a exigir os mesmos sempre que necessário criar uma classe
- Podem haver múltiplos construtores em uma classe
- São invocados apenas uma vez (por objeto) no início do seu ciclo de vida

Exercício exercicio-banco - parte 2

No mesmo projeto exercicio-banco criado anteriormente:

- Criar classe TelaCorrentista, criando método criarCorrentista e menu.
  - Neste método, solicitar nome, agência e conta. Criar uma conta, o correntista e deixar em um array.
- Criar classe TelaMovimento, criando método adicionarMovimento e menu.
  - Escolher o correntista, informar Tipo, Valor, Descricao.
- Alterar classe TelaMovimento, criando método imprimirExtrato.
  - Escolher o correntista, imprimir movimentos
- E por último: vamos criar construtores para nossas classes!

Interfaces e Classes Abstratas

Por que precisamos de algo além das classes?

Interfaces

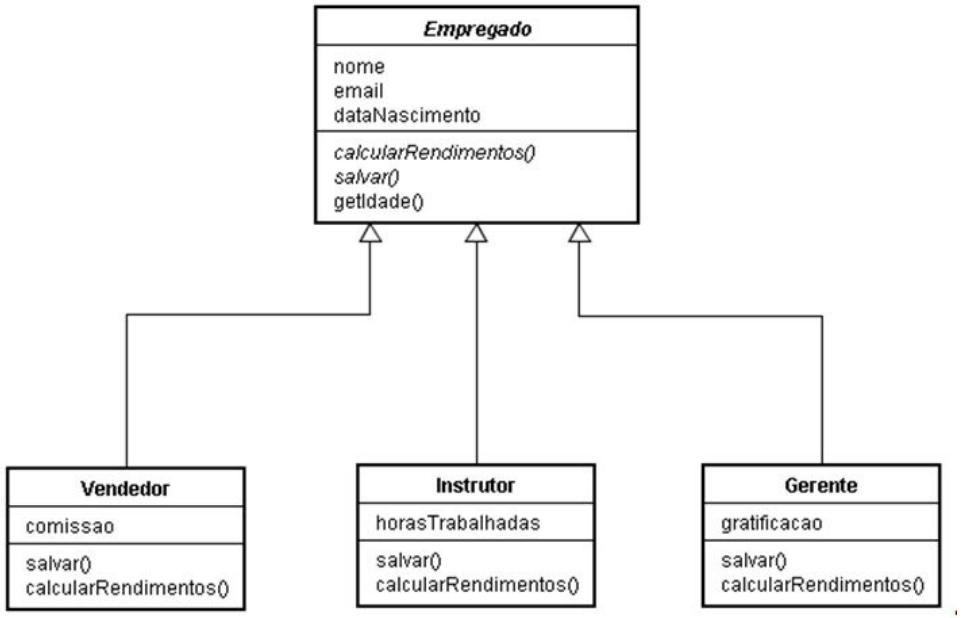
- São "Contratos"
- Apenas definem Operações
- Não possuem Atributos
  - Exceto se forem constantes
- São "implementadas" por classes
- É um exemplo de polimorfismo

Classes Abstratas

- Não podem ser instanciadas diretamente
- Devem ser herdadas para serem instanciadas

Classes Abstratas

Exemplo de representação em um Diagrama de Classes



Revisitando a OO: Relacionamentos

Associação, Dependência, Implementação e a famosa Herança...

Associação

- É uma espécie de vínculo entre duas classes.
- Pode ser ainda ser dividido em:
  - agregação: objeto é parte de outro
    - a parte pode existir sem o todo.
  - e composição: O todo contém as partes
    - Quando o todo é destruído, as partes também são
- Representação na UML:

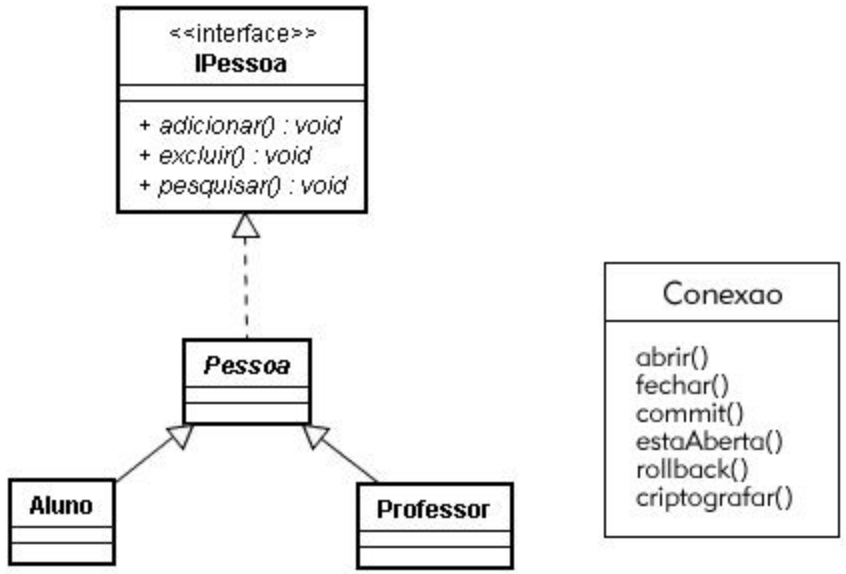
Dependência

Indica uso de uma classe por outra, sem vínculo de associação.

Implementação

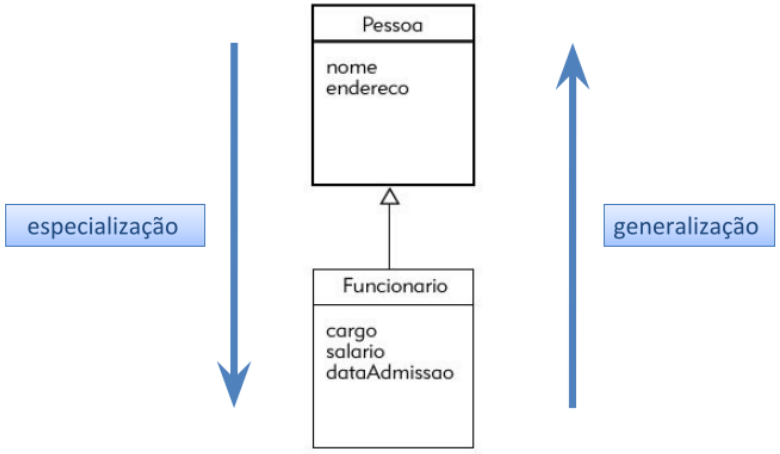
Utilizado para indicar que uma classe implementa uma interface

Exemplos:



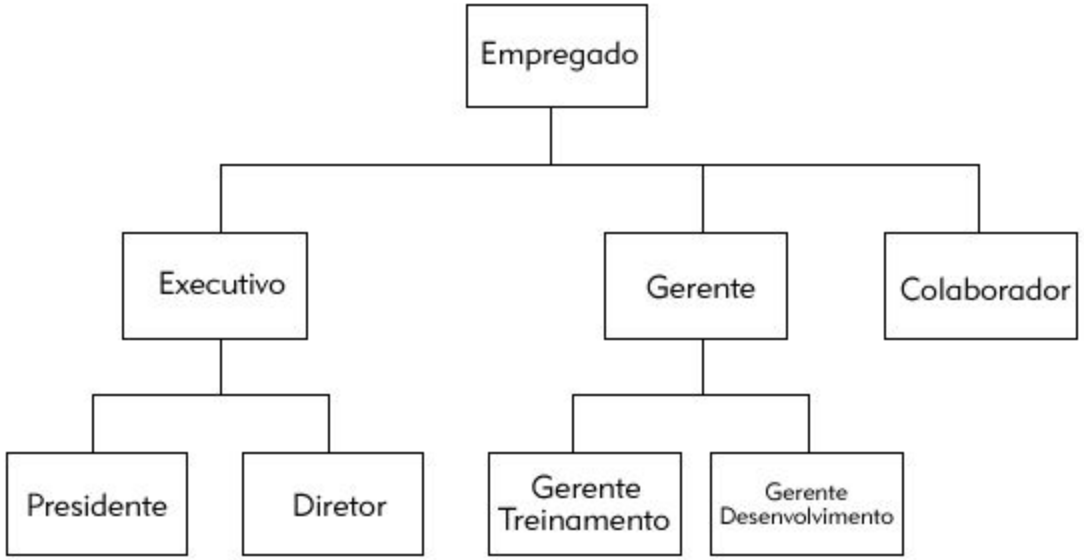
Herança

- Generalização
- Uma subclasse estende uma superclasse
- Herdam-se seus atributos e operações
  - Considerando a visibilidade
- Herança múltipla ocorre quando uma subclasse possui duas superclasses (mas maioria das linguagens não suporta)

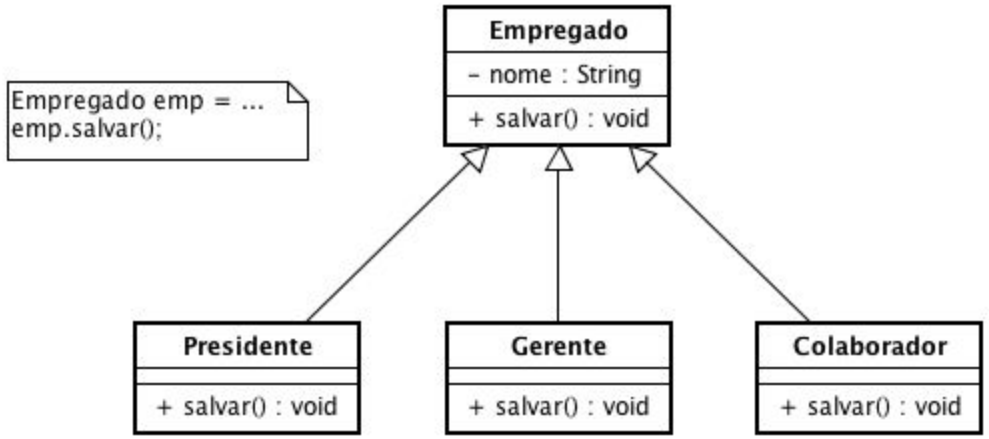


Herança: Níveis

- Herança pode ocorre em múltiplos níveis

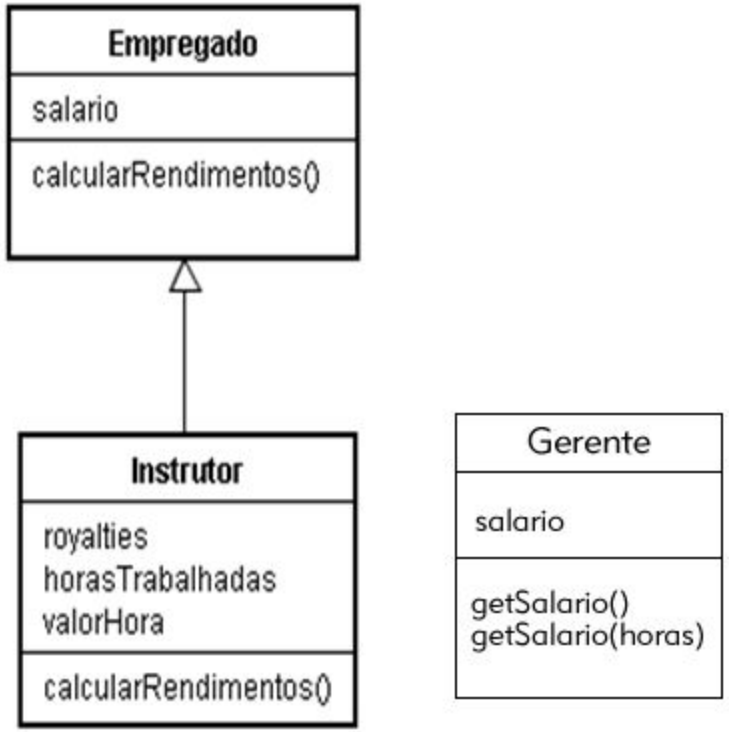


Polimorfismo




```
public void operacaoSalvar(Empregado emp) {
    emp.salvar();
}
```

Overriding vs Overloading



Exercício exercicio-banco - parte 3

No mesmo projeto exercicio-banco criado anteriormente:

- Vamos criar as classes CorrentistaPJ e CorrentistaPF
  - Aplicaremos herança com o Correntista
  - Criar o campo tipoDocumento no Pessoa Física
- Vamos criar uma interface para as nossa classes de Tela