Java
Fundamentos e
Orientação a
Objetos

Java – Aula 4

Java Fundamentos e Orientação a Objetos

O Paradigma da Orientação a Objetos

Criadores da UML

James Rumbaugh - Object Modeling Technique (OMT);

Grady Booch - Booch Method;

Ivar Jacobson - Objectory (OOSE) Process.

Cada autor adotava ideias dos métodos dos outros, então, evoluindo juntos produziriam melhorias; e

A unificação dos 3 métodos trariam estabilidade para o mercado.

Introdução

A orientação a objetos , também conhecida como " OO " , é chamada de paradigma pois requer uma forma diferente para:

Pensar o problema.

Modelar a solução.

Construir os algoritmos.

Introdução

Modelagem de sistemas consiste na criação de modelos (diagramas: estrutural e comportamental) capazes de representar o mundo real, sob a perspectiva do desenvolvimento de sistemas e está intimamente relacionado a dois fatores: ao paradigma de desenvolvimento e ao processo de desenvolvimento em uso;

Um sistema desenvolvido sob o paradigma da orientação a objetos representa os objetos que encontramos no mundo real, relacionados ao problema que estamos tratando, no negócio para o qual estamos desenvolvendo o sistema. O foco da equipe de desenvolvimento passa a ser a identificação e modelagem dos objetos do mundo real que afetam o sistema.

Introdução

A orientação a objetos está balizada em três princípios fundamentais:

- O encapsulamento que protege o acesso aos atributos;
- A herança; e
- O polimorfismo.

Juntos, eles permitem que classes sejam reaproveitadas, otimizando tempo e custo de desenvolvimento, além da segurança no reuso de classes já usadas e testadas.

Contextualização

Quando modelamos sistemas usando o paradigma orientado a objeto, a tarefa passa a ser a identificação dos objetos do mundo real envolvidos no contexto do sistema e a relação entre esses objetos.

Por exemplo, considerando o contexto de um sistema bancário, podemos citar os objetos agencia, cliente e conta, e a relação “Cliente possui uma conta em uma agencia”.

Ao identificar os objetos, são identificados também os dados (atributos) e as funcionalidades (funções) inerentes àquele objeto. O objeto Cliente possui os dados Nome, Endereço e CPF e sobre ele podem ser realizadas as funções de Incluir Novo Cliente, Excluir Cliente Inativo.

Conceituação Modelo OO

Na medida em que são identificados todos os objetos pertinentes a um sistema, já teremos os dados e procedimentos relacionados.

Um **modelo Orientado a Objeto (OO)** tem como entidade fundamental o objeto, que recebe e envia mensagens, executa procedimentos e possui um estado que, por proteção, apenas o próprio objeto pode modificar.

Problemas são resolvidos por meio de objetos, que enviam mensagens uns aos outros.

Elementos Básicos do Modelo OO

Classes

A POO tem como princípio básico categorizar e concentrar tudo relacionado a determinado item num único local, chamado de classe.

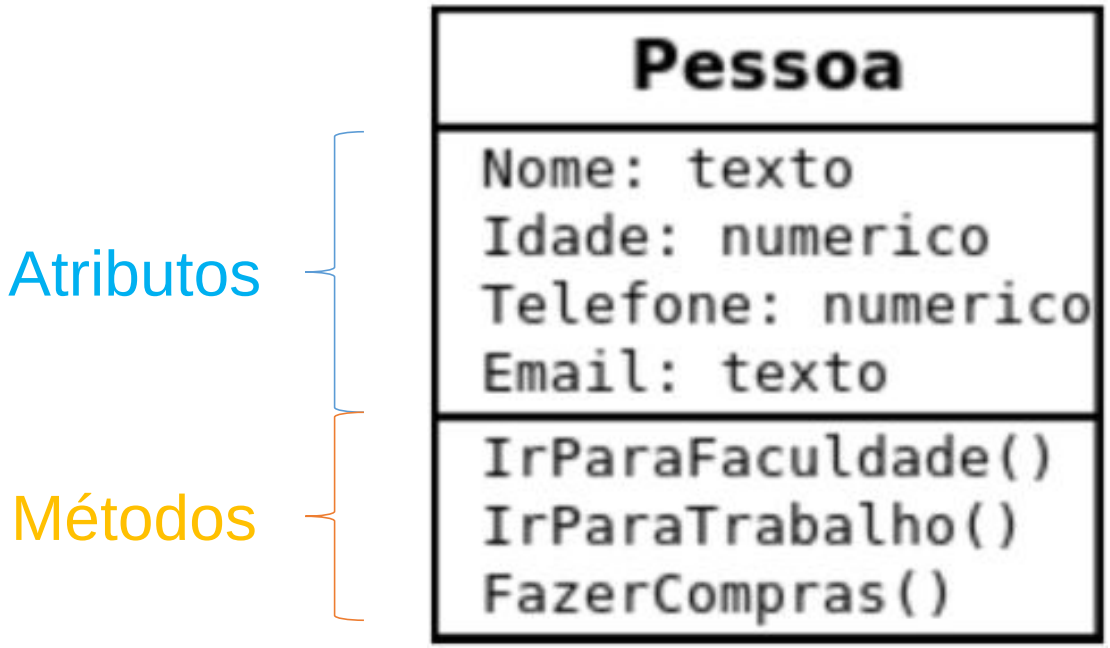
A classe, então, concentra todas as características de uma entidade qualquer e os chama de atributos.

Tudo o que essa entidade pode realizar, é chamado de método.

Elementos Básicos do Modelo OO

Classe “Pessoa”

São conjuntos de objetos com as mesmas características (atributos e métodos).



Elementos Básicos do Modelo OO

Objeto

É o principal elemento do Modelo Orientado a Objeto. Será visto como uma cópia de uma “classe”.

Os objetos representam as “coisas” a serem modeladas do mundo real. Um objeto pode ser algo concreto como um carro, um aluno ou algo abstrato como uma disciplina.

Cada objeto possui os dados inerentes a ele, como por exemplo, Nome (José) e Matrícula (20110127220l) de um aluno e possui as operações que ele executa, como incluir novo aluno ou alterar dados de um aluno existente.

Elementos Básicos do Modelo OO

Mensagens

As mensagens são enviadas entre os objetos. Quando um objeto deseja que seja executada uma operação, da responsabilidade de outro objeto, ele manda uma mensagem a esse outro objeto informando o que ele deseja que seja feito.

A operação desejada será implementada por meio de um método.

Atributos

São dados que caracterizam o objeto.

Elementos Básicos do Modelo OO

Métodos

São procedimentos (implementados por uma rotina ou função) que executam uma operação em um objeto e definem parte de seu comportamento.

Pilares do Modelo OO

Encapsulamento

O encapsulamento é uma das principais e mais importantes técnicas do Programação Orientada a Objeto.

Como as propriedades e os métodos ficam encapsulados na classe, elas podem ficar protegidos e seguros, sem que o código fique visível para outros programadores.

O encapsulamento é uma técnica para minimizar a interdependência entre as classes, pois apenas os métodos da respectiva classe podem alterar seus dados (atributos), facilitando a identificação de erros e a alteração dos programas.

Pilares do Modelo OO

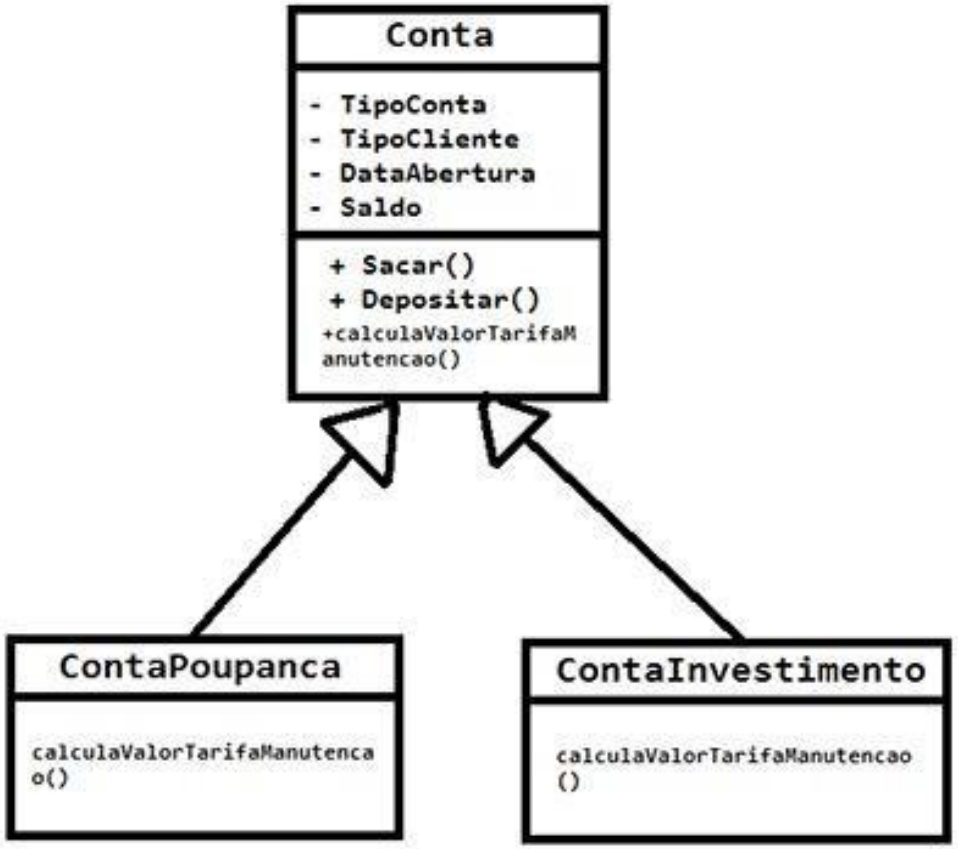
Herança

Trata-se de um mecanismo para derivar novas classes a partir da definição de classes existentes, que se dá por meio de um processo de refinamento. Uma classe derivada ou descendente herda os dados (atributos) e comportamento (métodos) da classe base ou ancestral ou ascendente.

A implementação da herança garante a reutilização de código que, além de economizar tempo e dinheiro, propicia mais segurança ao processo de desenvolvimento, posto que as funcionalidades da classe base já foram usadas e testadas.

Pilares do Modelo OO

Herança “Diagrama de Classe”



Pilares do Modelo OO

Polimorfismo

A palavra polimorfismo deriva do grego e significa “muitas formas”. A partir do momento em que uma classe herda atributos e métodos de uma (herança simples) ou mais (herança múltipla) classes base, ela tem o poder de alterar o comportamento de cada um desses procedimentos (métodos).

Isso amplia o poder do reaproveitamento de código promovido pela herança, permitindo que se aproveite alguns métodos e se altere outros. Um método com mesmo nome, em classes distintas, pode ter diferentes comportamentos.

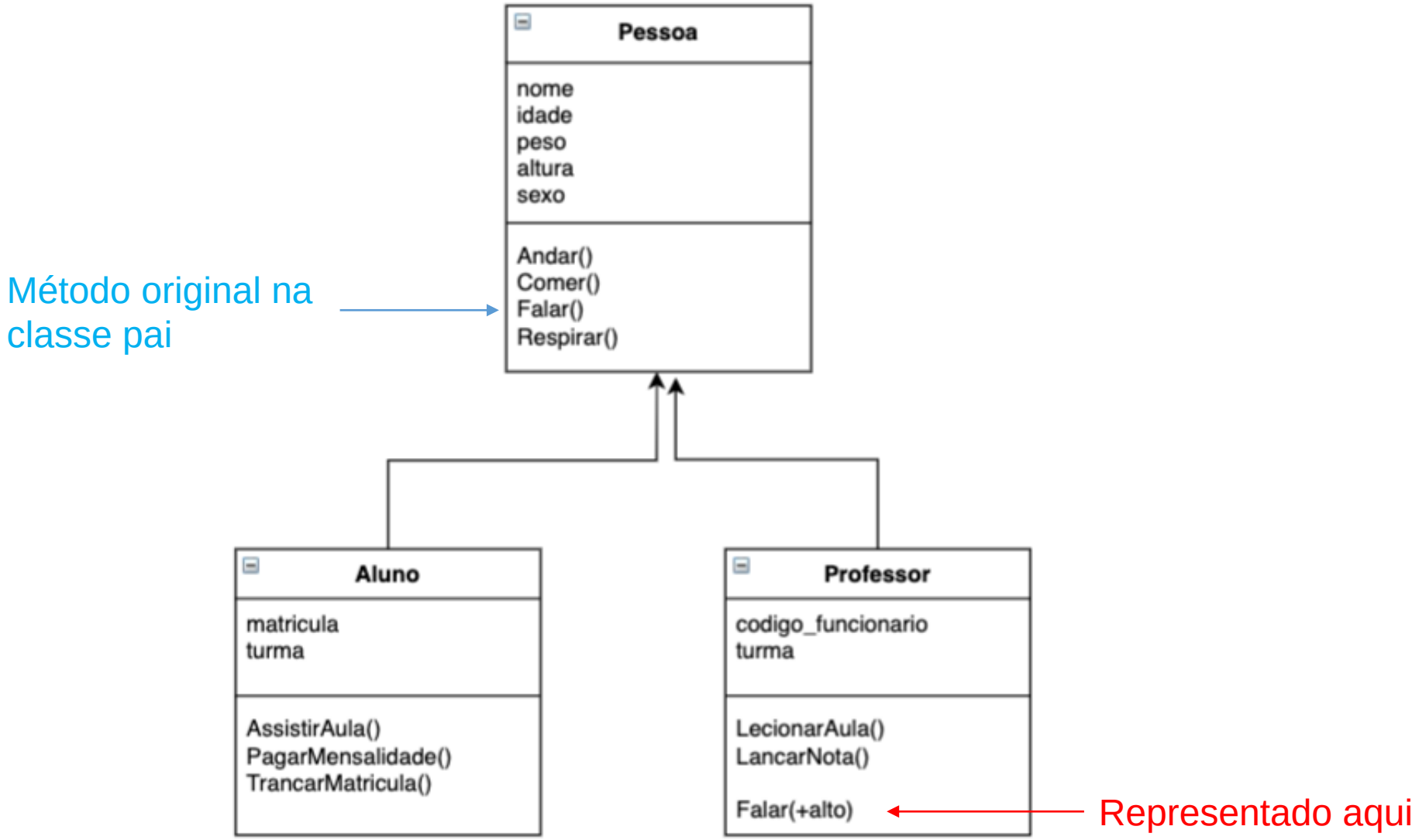
Pilares do Modelo OO

Polimorfismo “sobrescrita”

Por exemplo, uma ação normal de uma pessoa é falar, representado na classe Pessoa pelo método Falar(), mas um Professor em sala de aula tem que falar mais alto e claro para que os seus Alunos ouçam melhor. Então, o Professor não quer reaproveitar a forma como Falar () que acontece na classe pai.

Neste caso, descartamos todo o código existente e criamos "do zero" o método Falar(+ alto). Como estamos descartando tudo e escrevendo novamente, estamos fazendo uma **sobrescrita** do método Falar(). Isto pode ser interpretado no diagrama de classes da seguinte forma:

Pilares do Modelo OO



UML

A UML (Unified Modeling Language) é uma linguagem padrão para construção de projetos de sistemas, voltada para a visualização, especificação, construção e documentação de artefatos de um sistema. A UML foi projetada para ser independente do método de desenvolvimento a ser utilizado.

A UML é uma linguagem de modelagem. Não se trata de um método de desenvolvimento, nem tampouco de uma metodologia ou de um processo de desenvolvimento de sistemas. Isso porque ela não determina a ordem e nem como os diagramas devem ser usados. A UML simplesmente disponibiliza os diagramas, sob as várias visões necessárias ao desenvolvimento de sistemas.

Principais Características da UML

Visualização → modelagem gráfica tende a facilitar a compreensão, permitindo sua interpretação sem ambiguidades;

Especificação → permite a construção de modelos precisos, completos e sem ambiguidades. A UML atende a esses quesitos sob o ponto de vista da análise, do projeto e da implementação;

Construção → os diagramas UML podem ser diretamente integrados a várias linguagens de programação tais como Java e C++; e

Documentação → UML abrange a documentação da arquitetura do sistema e de todos os seus detalhes.

Diagramas da UML

Elementos Estruturais	Elementos Comportamentais
Diagrama de classe Diagrama de objeto Diagrama de componente Diagrama de instalação ou complementação Diagrama de pacote Diagrama de estrutura composta Diagrama de perfil	Diagrama de caso de uso Diagrama de sequência Diagrama de colaboração Diagrama de transição de estado Diagrama de atividade

Diagrama de Classe

Classe

Classes são os elementos mais importante de qualquer sistema orientado a objetos.

Uma classe é uma descrição de um conjunto de objetos com os mesmos **atributos**, **relacionamentos**, **operações** e **semântica.**

Classes são usadas para capturar o **vocabulário** de um sistema.

Classes são abstrações de elementos do domínio do problema, como “Cliente”, “Banco”, “Conta”.

Diagrama de Classe

Classe “Pessoa”

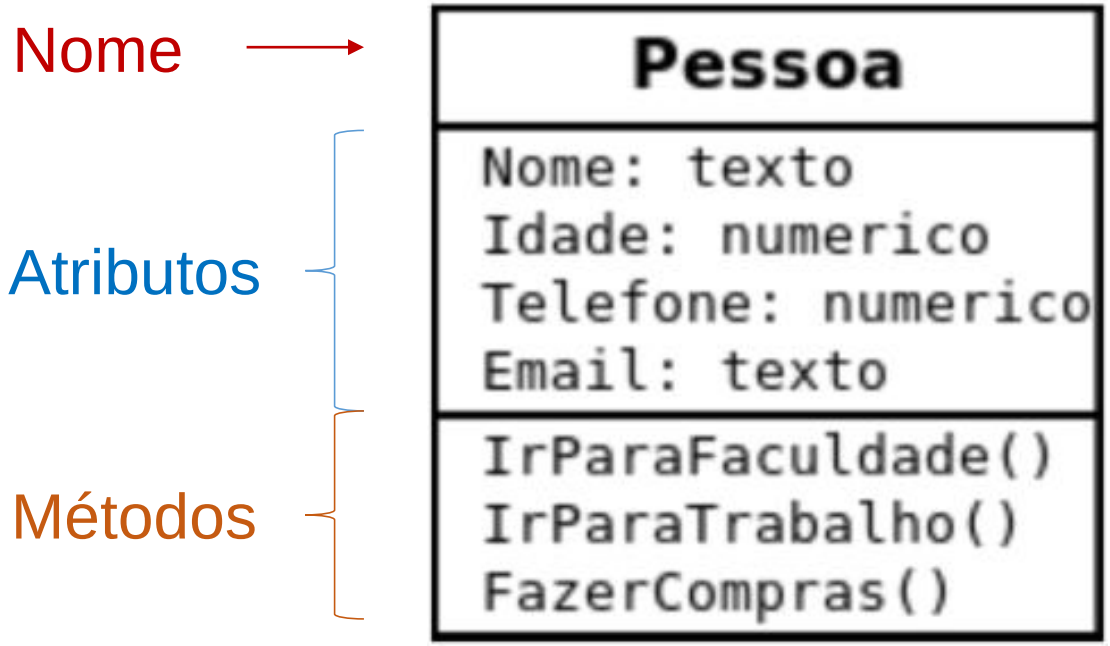


Diagrama de Classe

Classe

Atributos

As classes costumam ter atributos que armazenam os dados dos objetos da classe, além de métodos, também chamados operações, que são como funções que uma instância da classe pode executar.

Valores dos atributos

Os valores dos atributos podem variar de uma instância para outra. Graças a essa característica, aliás, é possível identificar cada objeto individual, ao passo que os métodos são idênticos para todas as características de uma classe específica.

Métodos

Embora os métodos sejam declarados no diagrama de classes, identificando os parâmetros possíveis que são por eles recebidos e os valores possíveis por eles retornados, o diagrama de classes não se preocupa em definir as etapas que tais métodos percorrer quando primeiros chamados.

Diagrama de Classe

Classe “Pessoa”

Criação da classe no java.

```java
public class Pessoa {

    private String Nome;
    private String RG;
    private String CPF;
    private double Salario;

    public void setNome(String nome) {
        this.Nome = nome;
    }
    public void setRG(String rg) {
        this.RG = rg;
    }
    public void setCPF(String cpf) {
        this.CPF = cpf;
    }
    public void setSalario(double salario) {
       this.Salario = salario;
    }
}
```

Diagrama de Classe

Relacionamentos

Os relacionamentos ligam as classes/objetos entre si criando relações lógicas entre estas entidades.

Tipos de relacionamentos especialmente importantes na modelagem orientada a objetos:

- Associações;
- Agregação;
- Composição;
- Dependências;
- Generalizações; e
- Realização.

Diagrama de Classe

Relacionamentos

Associação → especifica que objetos de um elemento (classe) estão conectados a objetos de outros elementos.

Agregação → relacionamento fraco do tipo “é parte de”. É um tipo especial de associação;

Composição → relacionamento forte do tipo “é parte de”. A composição entre um elemento (o “todo”) e outros elementos (“as partes”) indica que as partes só existem em função do “todo”.

Diagrama de Classe

Relacionamentos

Dependência → relacionamento de uso, no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente.

Generalização (herança) → relacionamento entre descrições mais gerais e descrições mais específicas, com mais detalhes sobre alguns dos elementos gerais.

Realização → relacionamento entre uma interface e o elemento que a implementa.

Diagrama de Classe

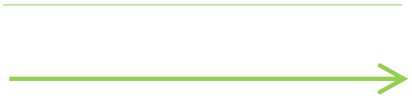
Relacionamentos (símbolos)

Agregação

Generalização

Associação

Sem/com navegação



Dependência

Composição

Realização

Diagrama de Classe

Multiplicidade

É a cardinalidade de uma associação.

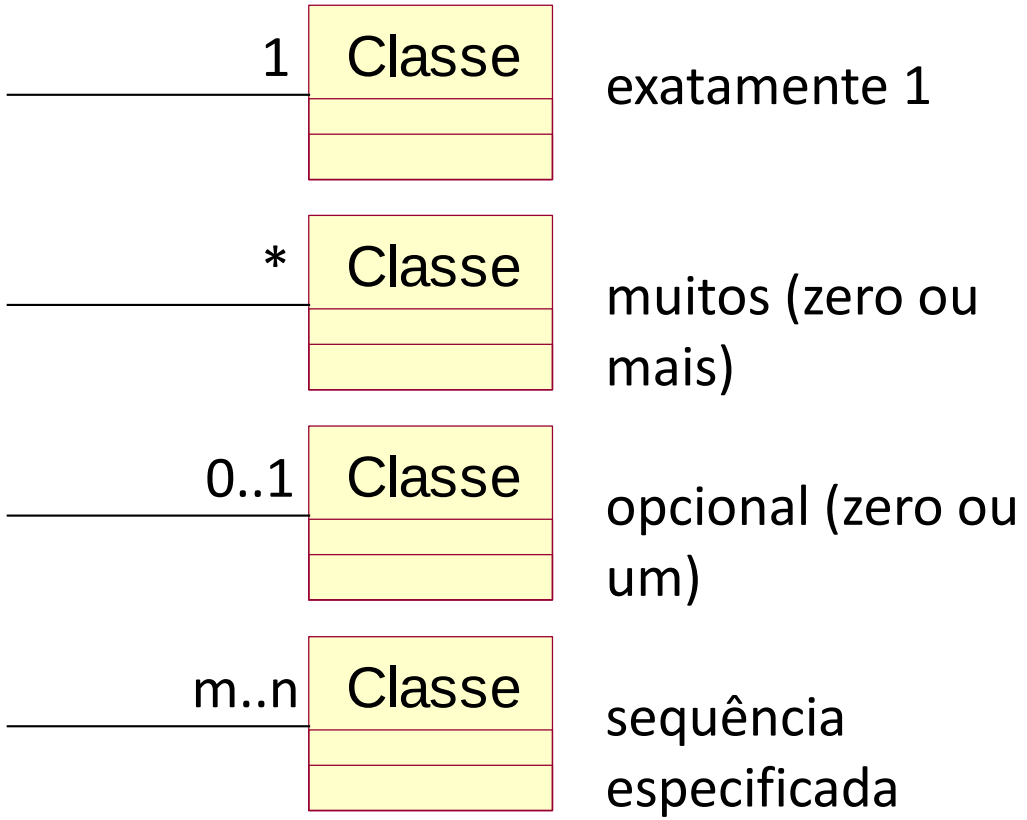


Diagrama de Classe

Associação Unária

Quando há um relacionamento de uma classe para consigo própria.

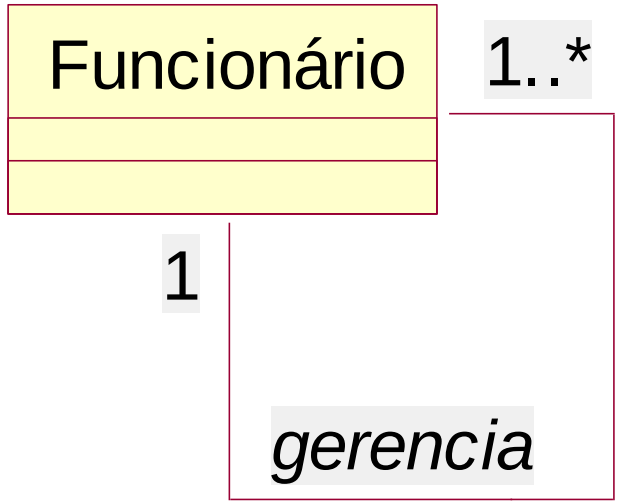


Diagrama de Classe

Associações Binárias

Associações binárias ocorrem quando são identificados entre objetos de duas classes distintas. Esse tipo de associação é, em geral, a mais comumente encontrada.

Exemplo de associação com uma classe cliente e pedido, nesse modelo um objeto da classe cliente pode se relacionar com nenhum ou vários objetos da classe pedido (0 .. *) e um objeto da classe pedido pode se relacionar com apenas um objeto da classe cliente (1).

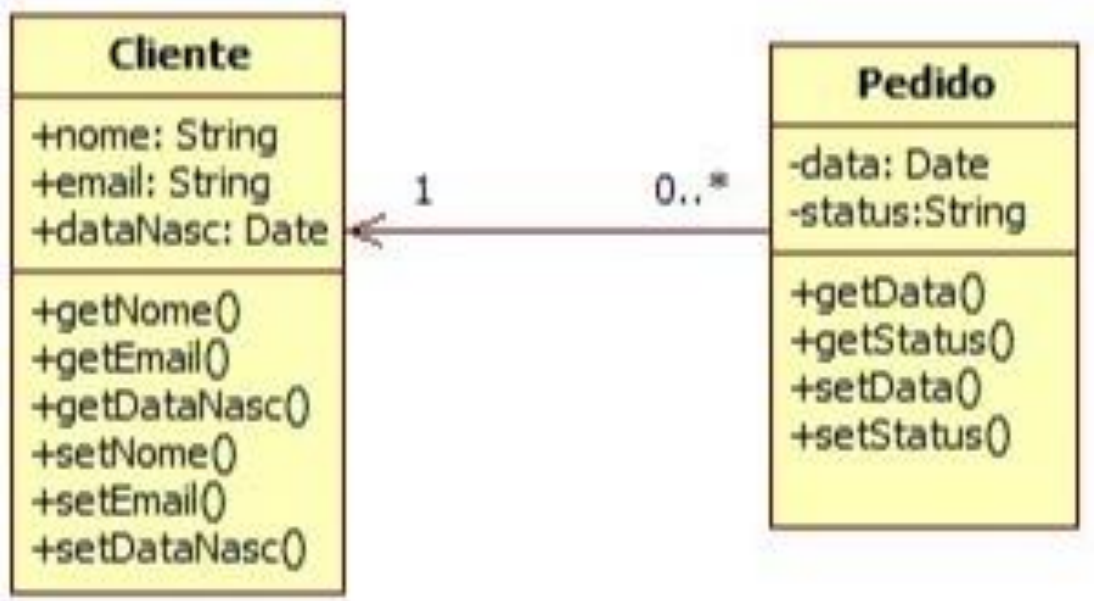


Diagrama de Classe

Código em java

```java
public class Cliente {

	private String nome;
	private String email;
	private Date dataNasc;


	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public Date getDataNasc() {
		return dataNasc;
	}

	public void setDataNasc(Date dataNasc) {
		this.dataNasc = dataNasc;
	}

}
```

```java
public class Pedido {

	private Date data;
	private String status;
	private Cliente cliente;
// Atributo do tipo cliente, a classe cliente compõe a classe pedido, esse atributo é uma cópia da classe cliente, portanto possui todos seus atributos e métodos.


	public Date getData() {
		return data;
	}
	public void setData(Date data) {
		this.data = data;
	}
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
	public Cliente getCliente() {
		return cliente;
	}
	public void setCliente(Cliente cliente) {
		this.cliente = cliente;
	}

}
```

Diagrama de Classe

Composição

Em uma composição os objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo ao qual estão relacionados. O símbolo de composição diferencia-se graficamente do símbolo de agregação por utilizar um losango preenchido.

Um ou vários itens (1 .. *) compõe um pedido e um item do pedido (1) só pode estar determinado pedido.

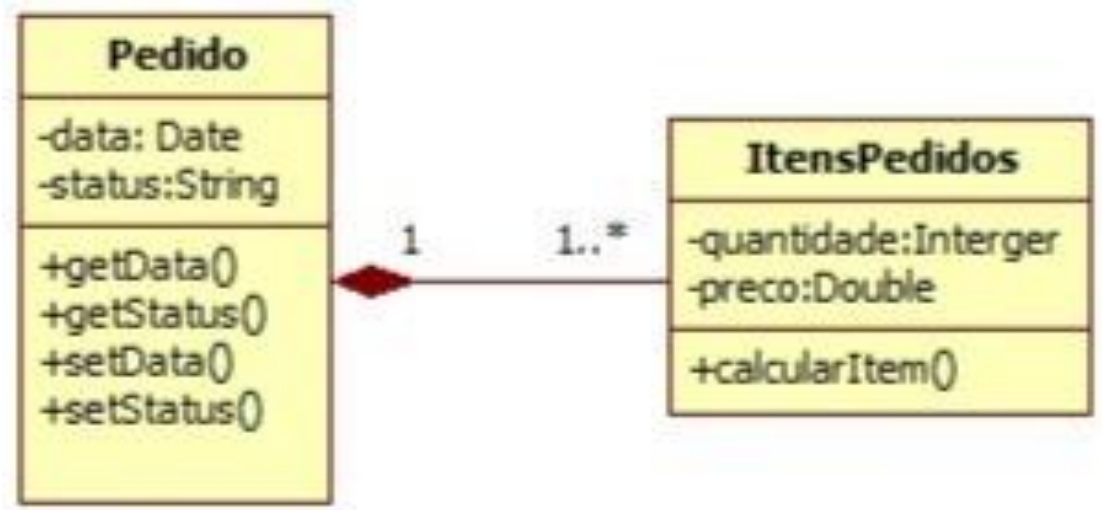


Diagrama de Classe

Código em java

```java
public class Pedido {

	private Date data;
	private String status;
	private Cliente cliente; // Atributo do tipo cliente, a classe cliente compõe a classe pedido, esse atributo é uma cópia da classe cliente, portanto possui todos seus atributos e métodos.

	private List<ItensPedido> itens = new ArrayList<>();

// Atributo do tipo ItensPedido, como um objeto da classe Pedido pode conter um ou vários itens, esse atributo deve ser uma lista da classe ArrayList ou de qualquer outra coleção que seja uma lista em Java.


	public Date getData() {
		return data;
	}
	public void setData(Date data) {
		this.data = data;
	}
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
	public Cliente getCliente() {
		return cliente;
	}
	public void setCliente(Cliente cliente) {
		this.cliente = cliente;
	}

}
```

```java
public class ItensPedido {

	private Integer quantidade;
	private Double preco;

	public Integer getQuantidade() {
		return quantidade;
	}
	public void setQuantidade(Integer quantidade) {
		this.quantidade = quantidade;
	}
	public Double getPreco() {
		return preco;
	}
	public void setPreco(Double preco) {
		this.preco = preco;
	}

}
```

Diagrama de Classe

Herança

Herança (ou generalização) é o mecanismo pelo qual uma classe (subclasse) pode estender outra classe (superclasse), aproveitando seus padrões (métodos) e variáveis possíveis (atributos).

Nesse modelo temos uma vantagem / generalização, uma classe Cliente é base para as outras duas classes ClientePF e ClientePJ, ou seja, todos os atributos da classe cliente serão herdados pelas suas subclasses, então como subclasses tem quatro atributos sendo dois da classe mãe cliente e seus próprios atributos.

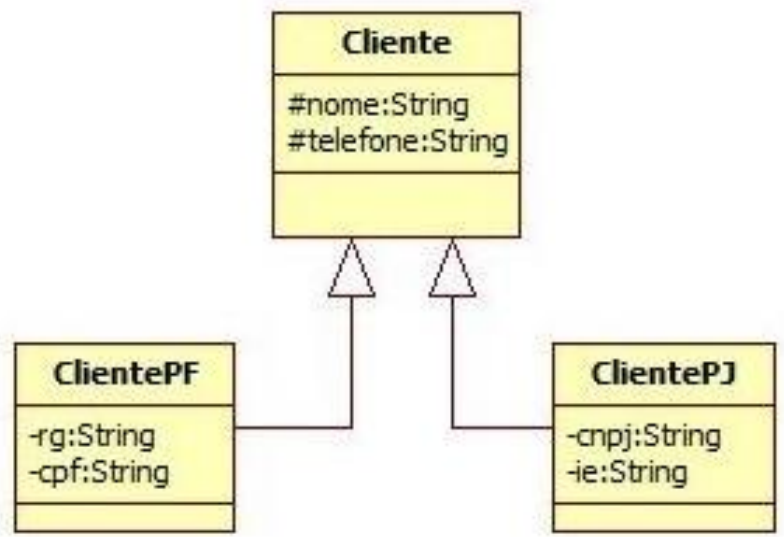


Diagrama de Classe

Código em java

```java
public class Cliente {
    String nome;
    String telefone;

    public Cliente(String nome, String telefone) {
        this.nome = nome;
        this.telefone = telefone;
    }

    public Cliente() {

    }

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getTelefone() {
        return telefone;
    }
    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

}
```

```java
public class ClientePF extends Cliente {

// A palavra reservada extends indica que a classe ClientePF
// vai herdar tudo que a classe Cliente tiver disponível para a
// Herança (atributos e métodos se houver).

    String rg;
    String cpf;

    public ClientePF() {
        super();
    }
// Método construtor da classe chamando o construtor da classe
mãe, responsável por contribuir o objeto da classe na memória


    public ClientePF(String nome, String telefone, String rg,
String cpf) {
        super(nome, telefone);
        this.rg = rg;
        this.cpf = cpf;
    }

// Método construtor da classe chamando o construtor da classe
mãe, e recebendo seus próprios atributos por parâmetros.


    public String getRg() {
        return rg;
    }
    public void setRg(String rg) {
        this.rg = rg;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

}
```

```java
public class ClientePJ extends Cliente{
// A palavra reservada extends indica que a classe ClientePJ
// vai herdar tudo que a classe Cliente tiver disponível para a
// Herança (atributos e métodos se houver).


    String cnpj;
    String ie;

    public ClientePJ() {
        super();
    }
// Método construtor da classe chamando o construtor da classe
mãe, responsável por contribuir o objeto da classe na memória


    public ClientePJ(String nome, String telefone, String cnpj,
String ie) {
        super(nome, telefone);
        this.cnpj = cnpj;
        this.ie = ie;
    }
// Método construtor da classe chamando o construtor da classe
mãe, e recebendo seus próprios atributos por parâmetros.


    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }

    public String getIe() {
        return ie;
    }

    public void setIe(String ie) {
        this.ie = ie;
    }

}
```

Diagrama de Classe

Software para trabalhar com o Diagrama de Classe

Draw.io

É uma ferramenta competente e muito utilizada para a criação de diagramas UML (e de outros tipos). O site apresenta componentes que se adequam à criação de vários tipos de diagramas, como o de sequência, de atividades, de estados e o tradicional caso de uso.

https://app.diagrams.net/

Diagrama de Classe

Estudo de Caso 1

Precisamos informatizar um pequeno curso de informática. Para isso, pretende-se fazer um cadastro de alunos, contendo sua matrícula, nome, endereço.

Ministramos vários tipos de curso (código, descrição), temos que cada aluno pode se matricular em um ou mais curso.

A empresa possui vários instrutores (código, nome) que estão habilitados, cada um, em lecionar em vários cursos.

Diagrama de Classe

Estudo de Caso 2

O hospital RADIALMEDIC, possui um respectivo número de alas (código, descrição), nas quais abrigam várias especialidades para atendimento (código, descrição).

Em cada especialidade é sabido que há um ou mais médicos (matricula, nome) habilitados para tal.

O paciente (matricula, nome, endereço) ao chegar no hospital tem seus dados registrados e são encaminhados para uma determinada especialidade.