

# Programação Orientada a Objetos

## **Aula Teórica: Herança em OO**

Prof. Dirson Santos de Campos  
dirson\_campos@ufg.br

**(09/10/2023)**

# Programação Orientada a Objetos

## Herança

- Mecanismo de reutilização de software onde uma nova classe é criada absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas;
- Permite que elementos mais específicos incorporem a estrutura e o comportamento de elementos mais genéricos;

# Programação Orientada a Objetos

## Tópicos principais desta aula

- Herança em OO (Orientação a Objetos)
- Superclasses X Subclasses
- Representação de Herança em Diagrama de Classes da UML
- Associação, Composição e Agregação em OO
- Métodos construtores explícitos

# Programação Orientada a Objetos

## Herança (ou generalização)

- Quando um objeto da classe filho é criado ele herda todas as propriedades da classe pai, além das propriedades definidas na própria classe filho;
- O homem naturalmente pensa dessa forma ...



# Programação Orientada a Objetos

É um tipo de ...

- Um objeto de uma subclasse (classe filha) **é um tipo de** objeto da superclasse (classe pai);



Beagle

É um tipo de Cachorro



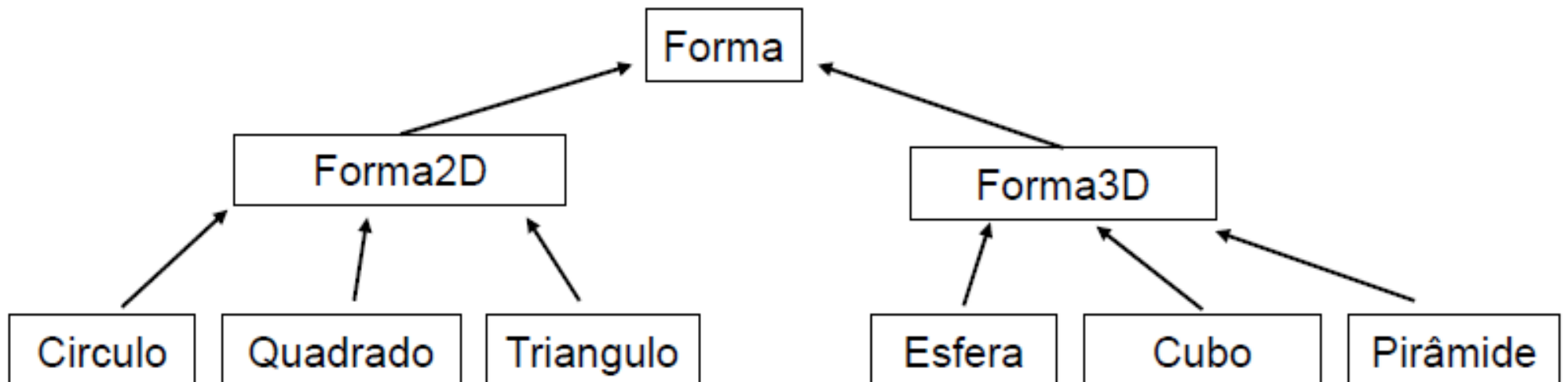
Lancha

É um tipo de veículo

# Programação Orientada a Objetos

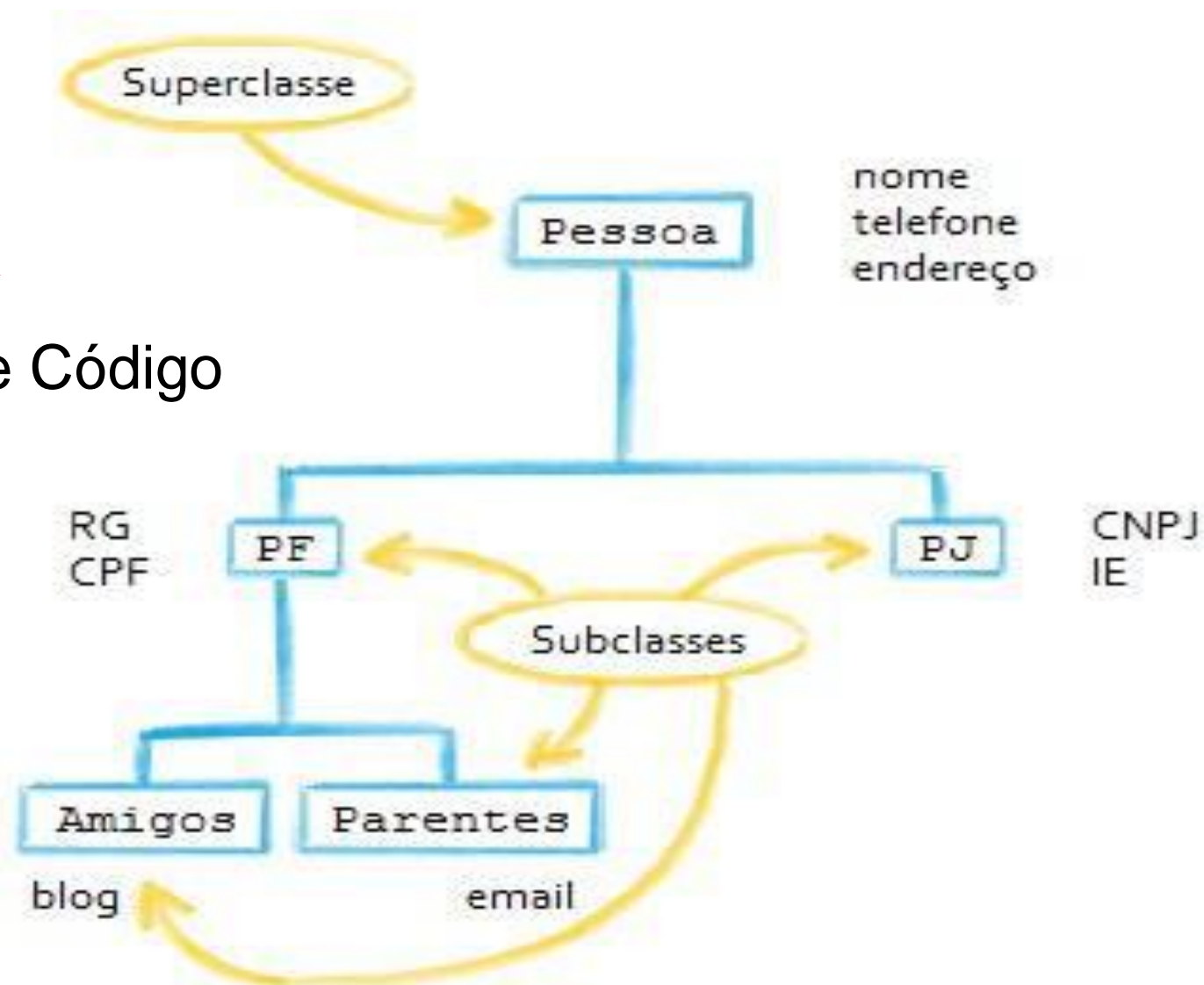
## Herança (cont.)

- Frequentemente um objeto de uma determinada classe também **é um** objeto de outra classe.
- Este tipo de relação normalmente é hierarquizada ...



# Programação Orientada a Objetos

- **HERANÇA**
- Reutilização de Código



# Programação Orientada a Objetos

## Superclasses X Subclasses

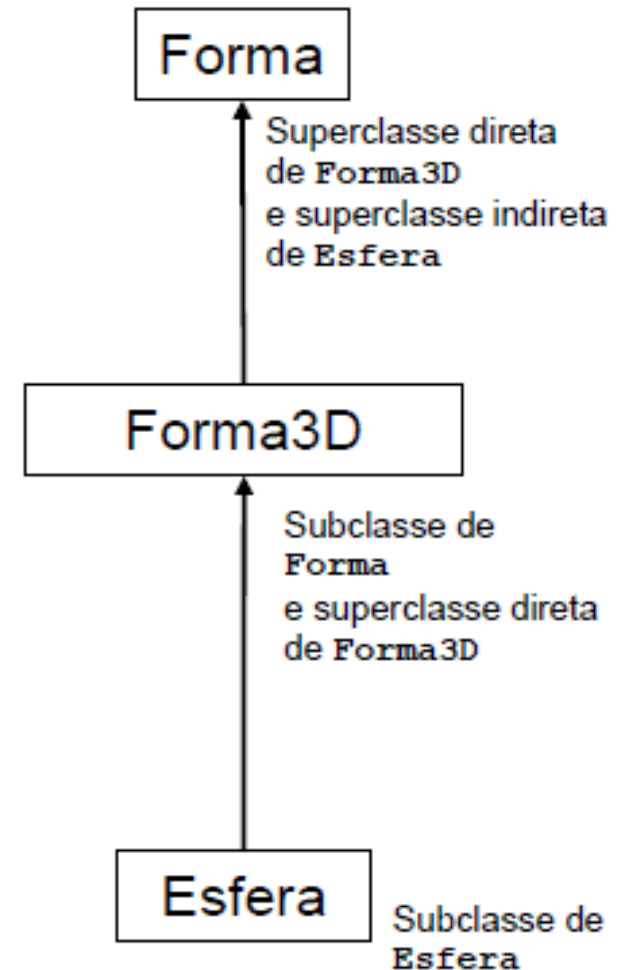
- Quando especificamos uma classe, ao invés de começar do zero, declarando atributos e métodos que talvez já existam em outra classe, podemos designar a nova classe a herdar o comportamento e as ações de uma classe já existente;
- A classe existente é chamada de **superclasse** e a nova classe de **subclasse**.



# Programação Orientada a Objetos

## Superclasses X Subclasses

- Superclasse tendem a ser mais gerais enquanto que subclasses, mais específicas;
- Toda subclasse pode vir a tornar-se uma superclasse para futuras subclasses;
- A superclasse direta é aquela a partir do qual a subclasse herda explicitamente, uma superclasse indireta é qualquer superclasse acima da classe direta na hierarquia de classes.



# Programação Orientada a Objetos

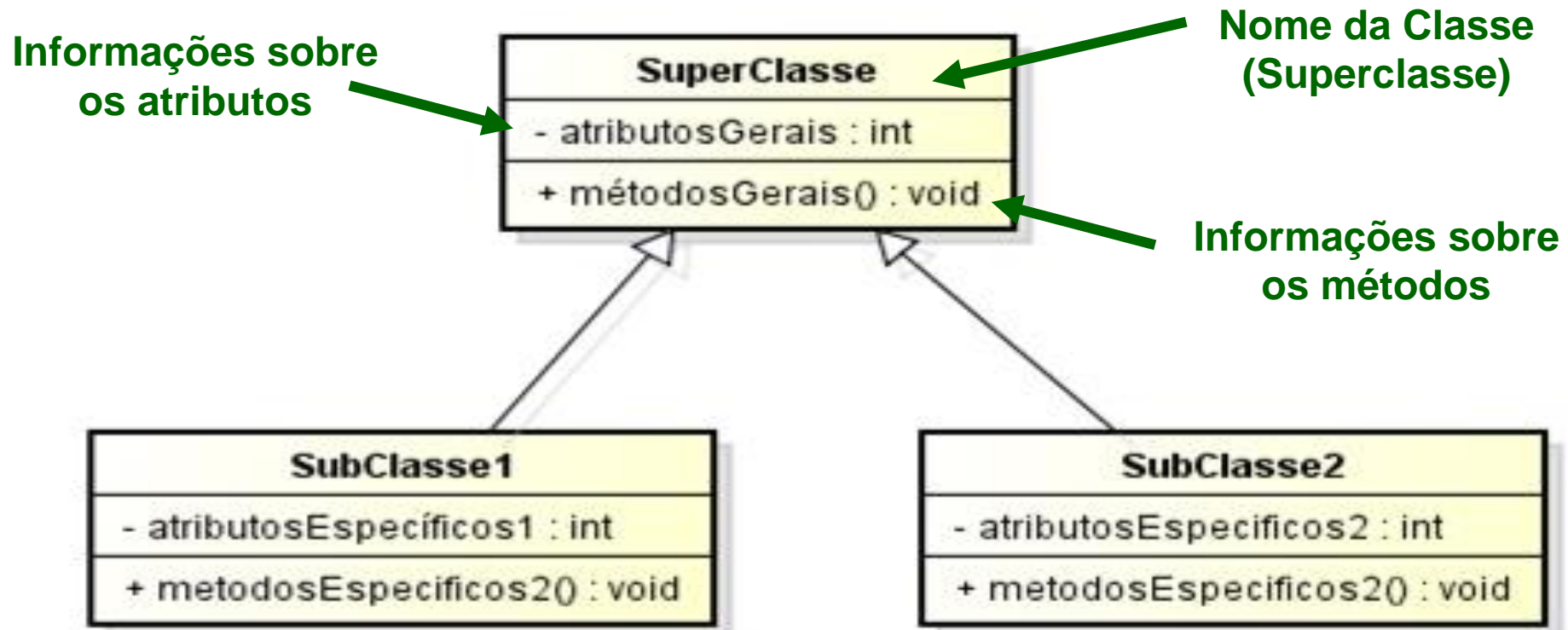
## Como identificar e modelar a herança ...

- Identificar as entidades importantes do contexto;
- Identificar as características (dados) e comportamentos (operações) de cada uma;
- Identificar características e comportamentos comuns (gerais) nas entidades;
- Identificar características e comportamentos específicos em cada entidades;
- Agrupar características e comportamentos comuns em uma superclasse (classe pai);
- Manter características e comportamentos específicos em cada classe;

# Programação Orientada a Objetos

## Herança - Representação em Diagrama de Classes da UML

(Desenhada sem uma ferramenta específica – é uma figura)





# Exemplo Abstração de uma Loja

Mapeamento do Diagrama  
UML de Classes para o  
código-fonte em Java

# Programação Orientada a Objetos

## Exemplo – Abstração de uma Loja (enunciado)

Uma loja deseja modelar e desenvolver um sistema e para isso ele identificou algumas entidades/atores importantes nas suas operações diárias: **clientes, fornecedores e funcionários**.

Verificou-se que todos podiam ser considerados **Pessoas** pois têm várias características comuns, porém alguns têm características especiais que também devem ser consideradas e representadas no modelo.

Para facilitar a modelagem dos dados e a implementação do sistema, foram utilizados os conceitos de herança para criar as classes, tentando reaproveitar o máximo de código possível.

# Programação Orientada a Objetos

## Resolução - Exemplo Abstração de uma Loja

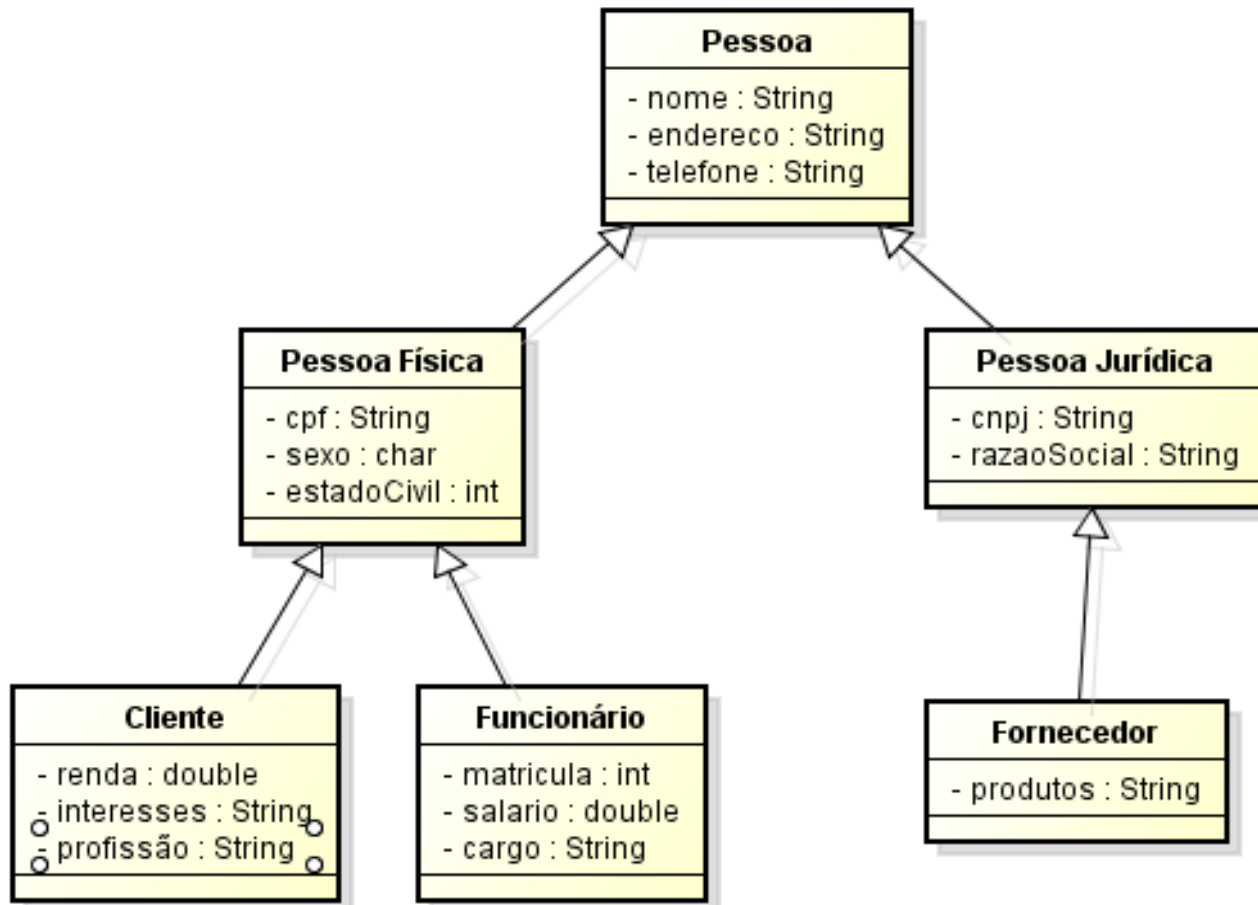
Como você faria isso ? Considerando as seguintes características:

- Um **cliente** é uma pessoa física que para ser cadastrado na loja precisa informar seu nome, cpf, endereço, telefone, sexo, estado civil, renda, interesses e profissão.
- Um **funcionário** é uma pessoa física e a loja guarda as seguintes informações sobre ele: nome, endereço, telefone, cpf, matrícula, cargo e salário
- Um **fornecedor** é uma entidade importante na loja e para ser cadastrado precisa informar o seu nome, endereço e telefone, cnpj, razão social e os produtos que ele fornece.

Tente gerar um modelo em UML com essas informações

# Programação Orientada a Objetos

## Resolução – Modelo do Diagrama de Classes do Exemplo Abstração de uma Loja



Observação:  
neste  
Diagrama de  
Classes parcial  
não estão  
descritos os  
Métodos,  
somente os  
nomes das  
classes e  
atributos

# Programação Orientada a Objetos

## Herança em Java da Classe PessoaFisica

```
public class PessoaFisica extends Pessoa {  
  
}
```

- A palavra reservada **extends** indica que a classe a ser especificada herda de uma outra classe;
- Na linguagem Java a hierarquia de classes inicia com a classe Object (do pacote java.lang), sendo assim toda classe Java é descendente em algum grau da classe Object.
- Uma sub-classe tem acesso aos atributos e métodos definidos com visibilidade **public** e **protected**, mas **não private**.



# Programação Orientada a Objetos

Resolução – Modelo do Diagrama de Classes do  
Exemplo Abstração de uma Loja em Java

```
public class Pessoa {  
    protected String nome;  
    protected String endereco;  
    protected String telefone;  
}
```

# Programação Orientada a Objetos

## Resolução – Modelo do Diagrama de Classes do Exemplo Abstração de uma Loja em Java

```
public class PessoaFisica extends Pessoa {  
  
    protected String cpf;  
    protected char sexo;  
    protected int estadoCivil;  
}
```

# Programação Orientada a Objetos

## Resolução – Modelo do Diagrama de Classes do Exemplo Abstração de uma Loja em Java

```
public class PessoaJuridica extends Pessoa
{
    protected String cnpj;
    protected String razaoSocial;
}
```

# Programação Orientada a Objetos

## Resolução – Modelo do Diagrama de Classes do Exemplo Abstração de uma Loja em Java

```
public class Cliente extends PessoaFisica
{

    protected double renda;
    protected String interesses;
    protected String profissao;
}
```

# Programação Orientada a Objetos

## Resolução – Modelo do Diagrama de Classes do Exemplo Abstração de uma Loja em Java



```
public class Funcionario extends PessoaFisica
{

    protected int matricula;
    protected String cargo;
    protected double salario;
}
```

# Programação Orientada a Objetos

Resolução – Modelo do Diagrama de Classes do  
Exemplo Abstração de uma Loja em Java

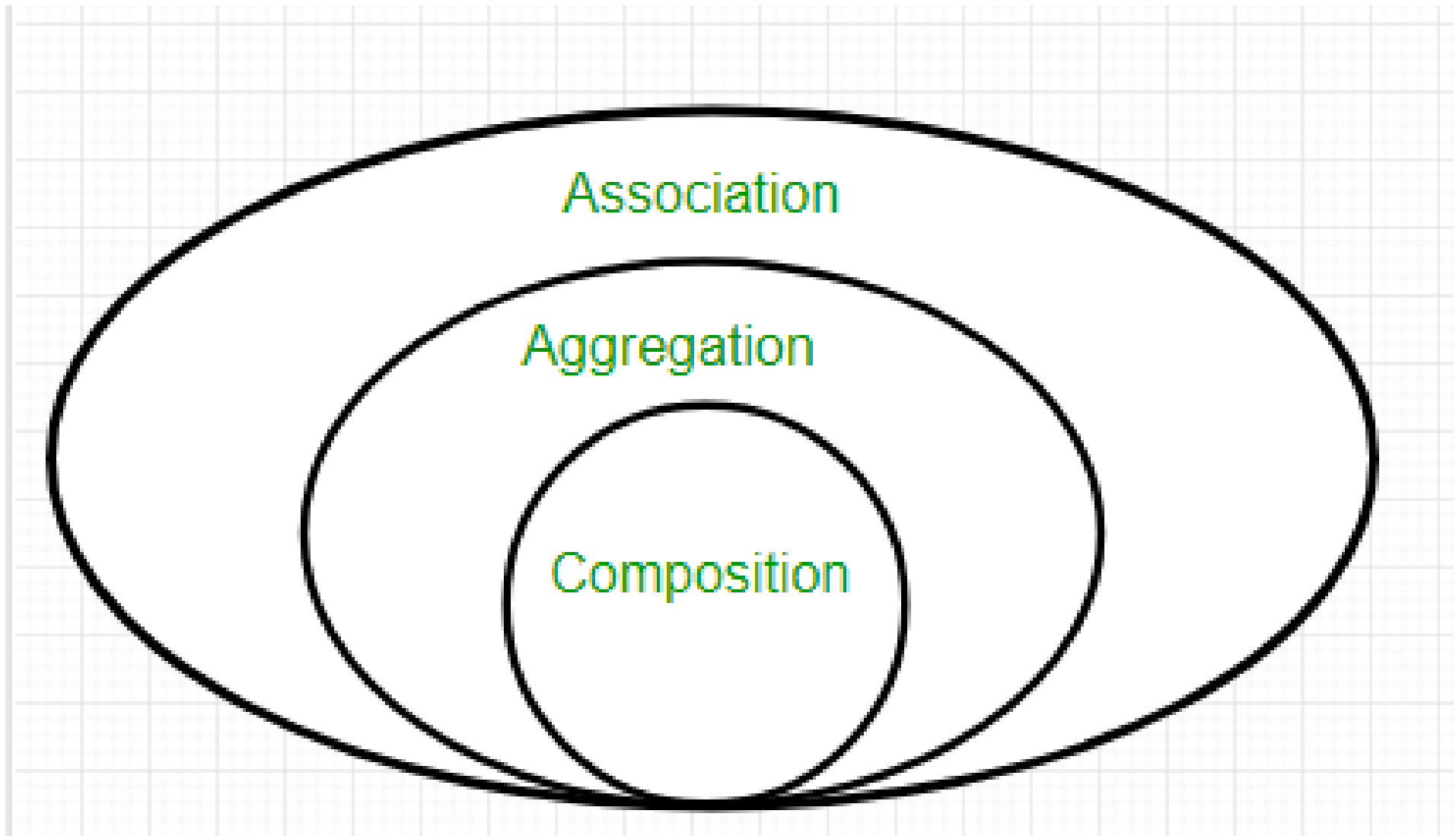
```
public class Fornecedor extends  
PessoaJuridica {  
  
    protected String produtos;  
  
}
```



# **Associação, Composição e Agregação em OO**

# Programação Orientada a Objetos

**Composição e Agregação são as duas formas de Associação.**





# Programação Orientada a Objetos

## Associação em OO

- Associação é a relação entre duas classes separadas que são estabelecidas por meio de seus objetos.
- Em orientada a objetos, um objeto se comunica com outro objeto para usar a funcionalidade e os serviços fornecidos por esse mesmo objeto.
- Uma agregação ou uma composição é um exemplo de Associação

# Programação Orientada a Objetos

## Exemplo de Associação em OO

- Uma agregação ou uma composição é um exemplo de Associação em OO

# Programação Orientada a Objetos

## Agregação em OO

- Agregação é uma forma especial de associação utilizada para mostrar que um tipo de objeto é composto, pelo menos em parte, de outro em uma relação todo/parte.
- Indicando que o objeto parte "é um atributo" do objeto todo.

# Programação Orientada a Objetos

## Exemplo de Agregação em OO

- As classes disciplina, professor e aluno forma uma agregação.
- O aluno é “ligado” a uma Disciplina, o professor também é “ligado” a uma disciplina.
- Em outras palavras, a disciplina, neste caso é o todo, enquanto os alunos e o professor são partes deste todo.

# Programação Orientada a Objetos

## Composição em OO

- A composição é uma outra forma de reaproveitarmos classes (também é conhecido por delegação);
- A composição é uma forma restrita de agregação, na qual duas entidades são altamente dependentes uma da outra.
- Representa parte do relacionamento.
- Na composição, ambas as entidades são dependentes uma da outra.
- Quando há uma composição entre duas entidades, o objeto composto não pode existir sem a outra entidade.

# Programação Orientada a Objetos

## Exemplo de Composição

- . A classe Instituto é “ligada” a Universidade, se o objeto da classe Universidade for destruído, todos os Institutos que são da Universidade em questão também devem ser excluídos.

# Programação Orientada a Objetos

## Exemplo de Relações entre Classes sem o uso de herança

- A classe **Aluno** contém na sua relação de atributos uma instância de **Curso** que tem uma instância de **Universidade** e métodos construtores explícitos, sem o uso de herança entre as classes

```
public class Aluno {
    String nome;
    int matricula;
    Curso curso;
    Aluno(int matricula, String nome) {
        this.nome = nome;
        this.matricula= matricula;
    }
}

public class Curso {
    String nome;
    int codigo;
    Universidade universidade;
    Curso (int codigo,String nome) {
        this.nome = nome;
        this.codigo = codigo;
    }
}
```

```
public class Universidade {
    String nome;
    String sigla;
    Universidade(String nome, String sigla)
    {
        this.nome = nome;
        this.sigla= sigla;
    }
}
```

# Programação Orientada a Objetos

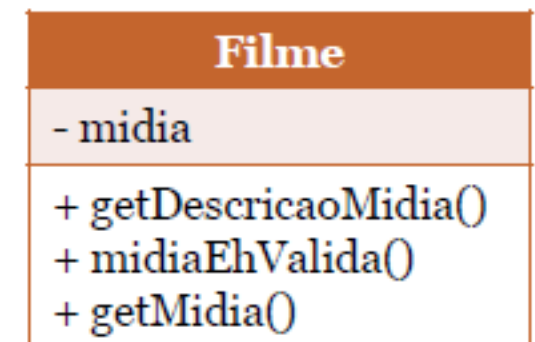
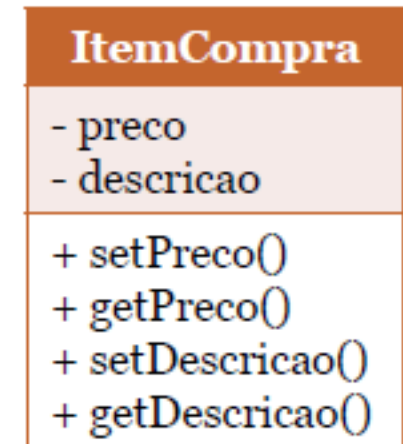
Diagrama  
de Classe  
desenhado  
à mão

## Herança em Java (Herança implícita)

```
class ItemCompra extends java.lang.Object {  
  
    private float preco;  
    private String descricao;  
  
    public ItemCompra(float p, String descr) {...}  
    public void setPreco(float novoPreco) {...}  
    public float getPreco() {...}  
    public void setDescricao(String d) {...}  
    public String getDescricao() {...}  
}
```

Neste caso  
a herança é  
opcional  
(pode ficar  
implícita)

```
class Filme extends ItemCompra {  
  
    private int midia;  
  
    public Filme(String descr, int midia) {...}  
    public String getDescricaoMidia() {...}  
    public static boolean midiaEhValida (int m) {...}  
    public int getMidia() {...}  
}
```





# Programação Orientada a Objetos

## Métodos construtores nas subclasses

- O construtor de uma subclasse **sempre** chama o construtor de sua superclasse, mesmo que a chamada não seja explícita.
- Se a chamada não for explícita (através da palavra-chave **super**) o construtor da subclasse tentará chamar o construtor vazio (sem argumentos) da superclasse – e se ele não estiver definido, ocorrerá um erro de compilação;
- Se uma classe não possui um construtor vazio (sem argumentos) e possui um construtor com argumentos, as classes herdeiras deverão obrigatoriamente chamar o construtor com argumentos da classe ancestral (**este é um tipo de erro que geralmente causa muita confusão**).

# Programação Orientada a Objetos

## Exercício 1: Uso de métodos construtores explícitos em herança

```
public class Funcionario {  
    protected String nome;  
    protected String cpf;  
    protected double salario;  
  
    Funcionario(String nome,String cpf){  
        this.nome = nome;  
        this.cpf = cpf;  
    }  
}
```

# Programação Orientada a Objetos

## Exercício 1: Uso de métodos construtores explícitos em herança

```
public class Gerente extends Funcionario{  
    private int senha;  
  
    Gerente (String nome,String cpf, int senha){  
        super(nome,cpf);  
        this.senha = senha;  
    }  
}
```

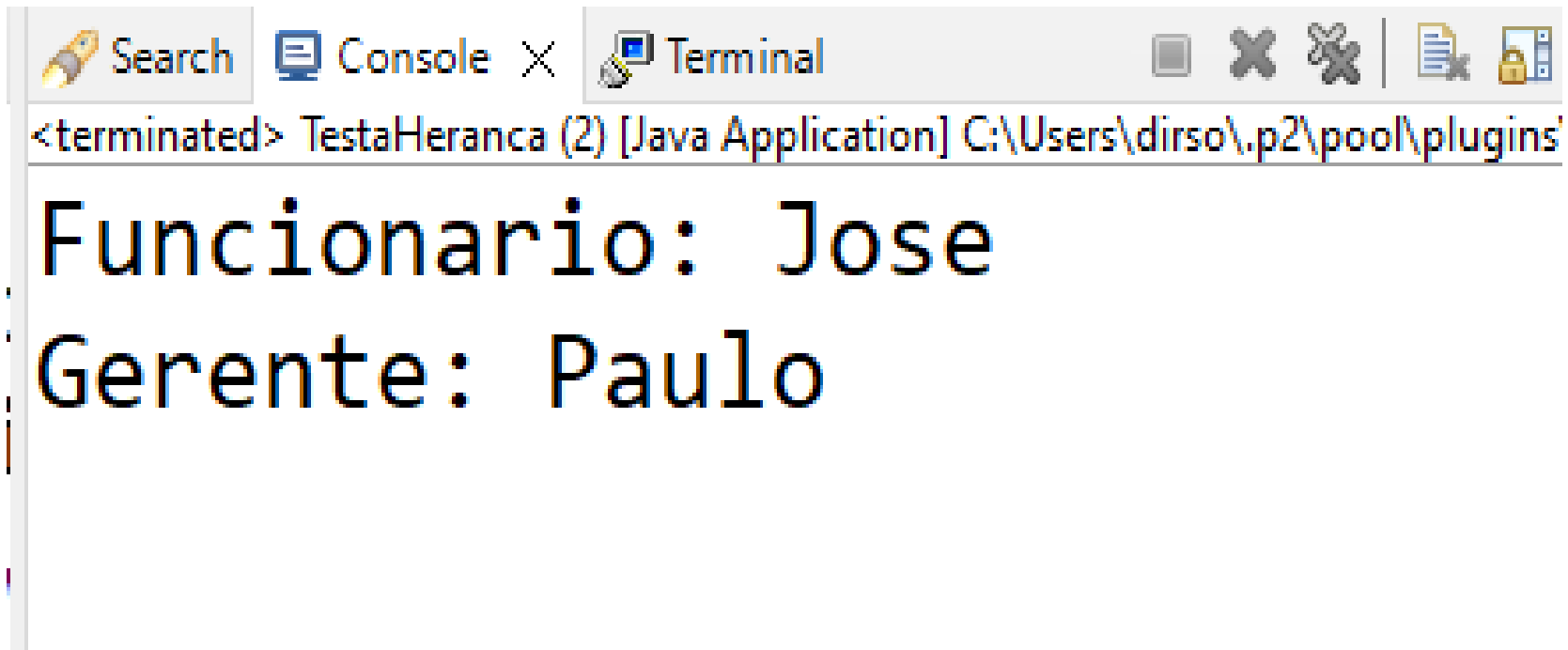
# Programação Orientada a Objetos

## Exercício 1: Uso de métodos construtores explícitos em herança

```
public class TestaHeranca{  
    public static void main(String [] args){  
        Funcionario f = new Funcionario("Jose", "23435678999");  
        Gerente g = new Gerente("Paulo", "9949595595", 3344);  
        System.out.println("Funcionario: " + f.nome);  
        System.out.println("Gerente: " + g.nome);  
    }  
}
```

# Programação Orientada a Objetos

## Rodando o exercício 1 no IDE Eclipse



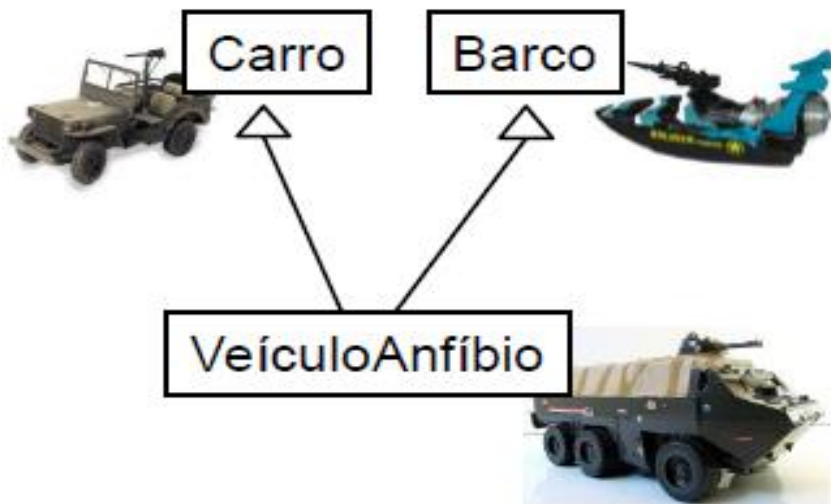
The screenshot shows the Eclipse IDE's integrated terminal. The terminal title bar includes tabs for 'Search', 'Console', and 'Terminal'. The terminal content displays the output of a Java application named 'TestaHeranca (2)'. The output consists of two lines: 'Funcionario: Jose' and 'Gerente: Paulo'.

```
<terminated> TestaHeranca (2) [Java Application] C:\Users\dirso\.p2\pool\plugins'  
Funcionario: Jose  
Gerente: Paulo
```

# Programação Orientada a Objetos

## Herança Múltipla (Não implementada em Java, existe em C++ e Python)

- Situação onde uma subclasse possui mais de uma superclasse;
- A linguagem Java não oferece suporte para herança múltipla;
- A alternativa para utilizar os benefícios da herança múltipla em Java é a utilização de interfaces;



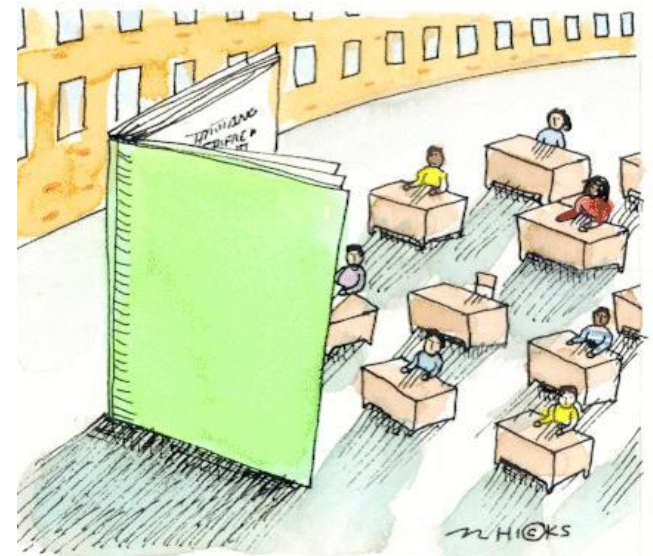
```
interface Carro {  
    public void puxarFreioDeMao();  
}  
  
interface Barco {  
    public void jogarAncora();  
}  
  
class VeiculoAnfibio implements Carro, Barco {  
    public void VeiculoAnfibio(Carro carro, Barco barco )  
    { ... }  
  
    public void puxarFreioDeMao() { ... }  
  
    public void jogarAncora() { ... }  
}
```

# Programação Orientada a Objetos

## Exercício:

### Organizando a bagunça

- Considere as pessoas presentes em um ambiente universitário (comunidade Acadêmica) e desenhe uma possível hierarquia de classes para representá-la
  - Alunos de graduação;
  - Alunos de pós-graduação;
  - Técnicos administrativos;
  - Docentes;
  - Coordenadores de curso (Administrador);
  - Professores;
  - Ex-Alunos



# Programação Orientada a Objetos

Uma possível hierarquia de Classes como solução

