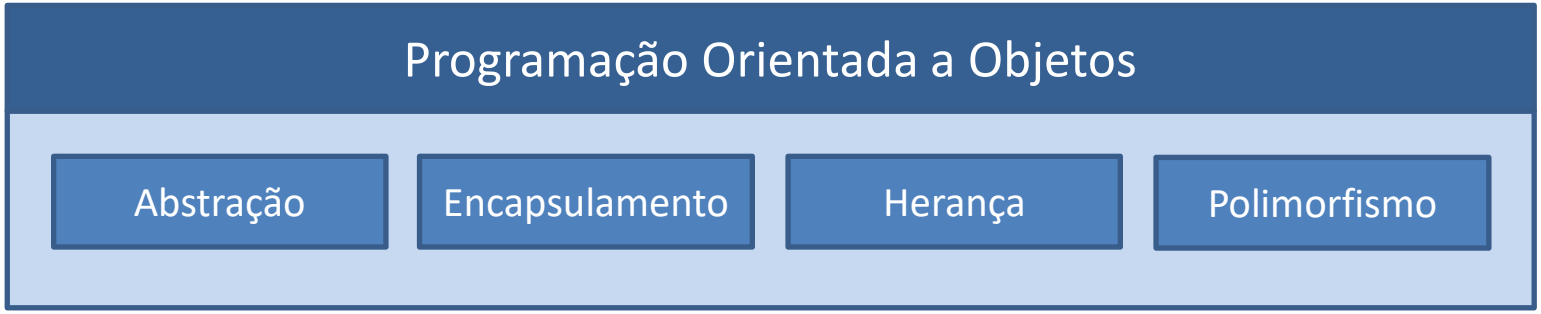
Herança

Professor Isaac

Herança

É um dos pilares da Orientação a Objetos

Pilares da Orientação a Objetos



Fonte: http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264

Conceitos de Herança

- É uma forma de reutilização de código!
- O conceito de herança se baseia no princípio de que toda codificação mais genérica pode ser transmitida para classes mais específicas.
- Cria uma nova classe a partir de uma classe existente.
- Relação é um

Herança

- Cria uma nova classe como uma **extensão** de uma classe já existente.
- A nova classe é um tipo da classe já existente.
- Permite utilizar a forma da classe existente, adicionando código, sem destruir a classe existente.
- A nova classe herda os atributos e os métodos da classe existente.

Exemplo - Carros

- Uma Ferrari é um Carro
- Uma BMW é um Carro
- Um Fusca é um Carro

Podemos ter uma classe **Carro** que tem características comuns a todos esses veículos!

A classe carro pode conter os seguintes atributos:

- **Rodas**
- **Cor**
- **ano de fabricação**

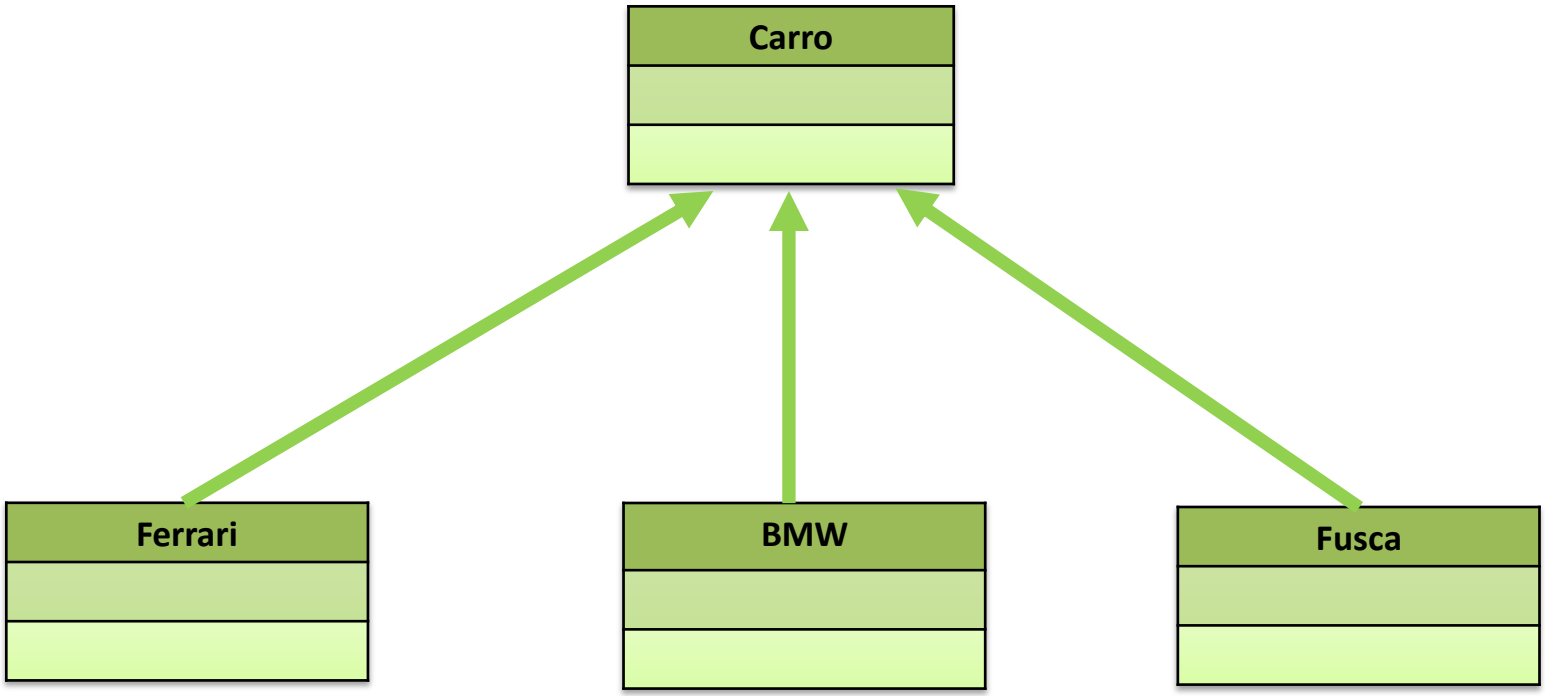
Superclasse e Subclasse

- Considerando os exemplos dados:
  - As classes Mamífero e Carro são superclasses / classes bases
  - As classes Ferrari, BMW, Fusca são subclasses / classes derivadas
- Superclasses tendem a ser mais genéricas
- Subclasses tendem a ser mais específicas

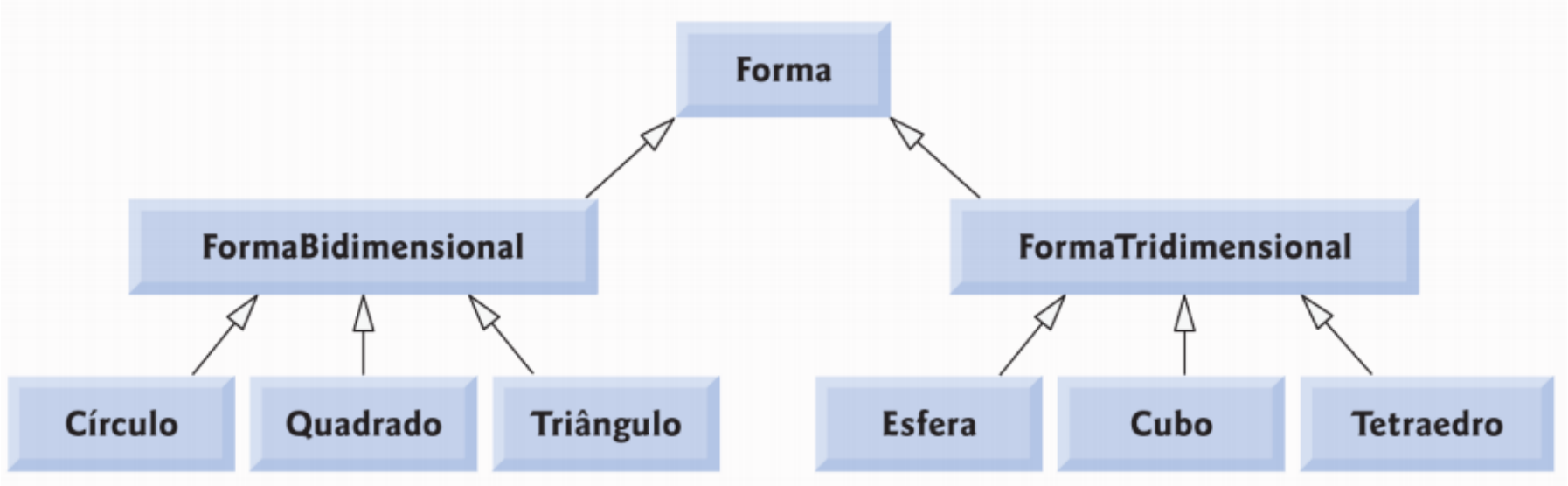
A Herança Permite que as Subclasses:

- Herdem os atributos e os métodos da superclasse:
  - atributos e métodos herdados podem ser diretamente utilizados - não é preciso escrevê-los novamente.
- Definam novos atributos e métodos.
- Modifiquem um método definido na superclasse (sobrescrita - override)

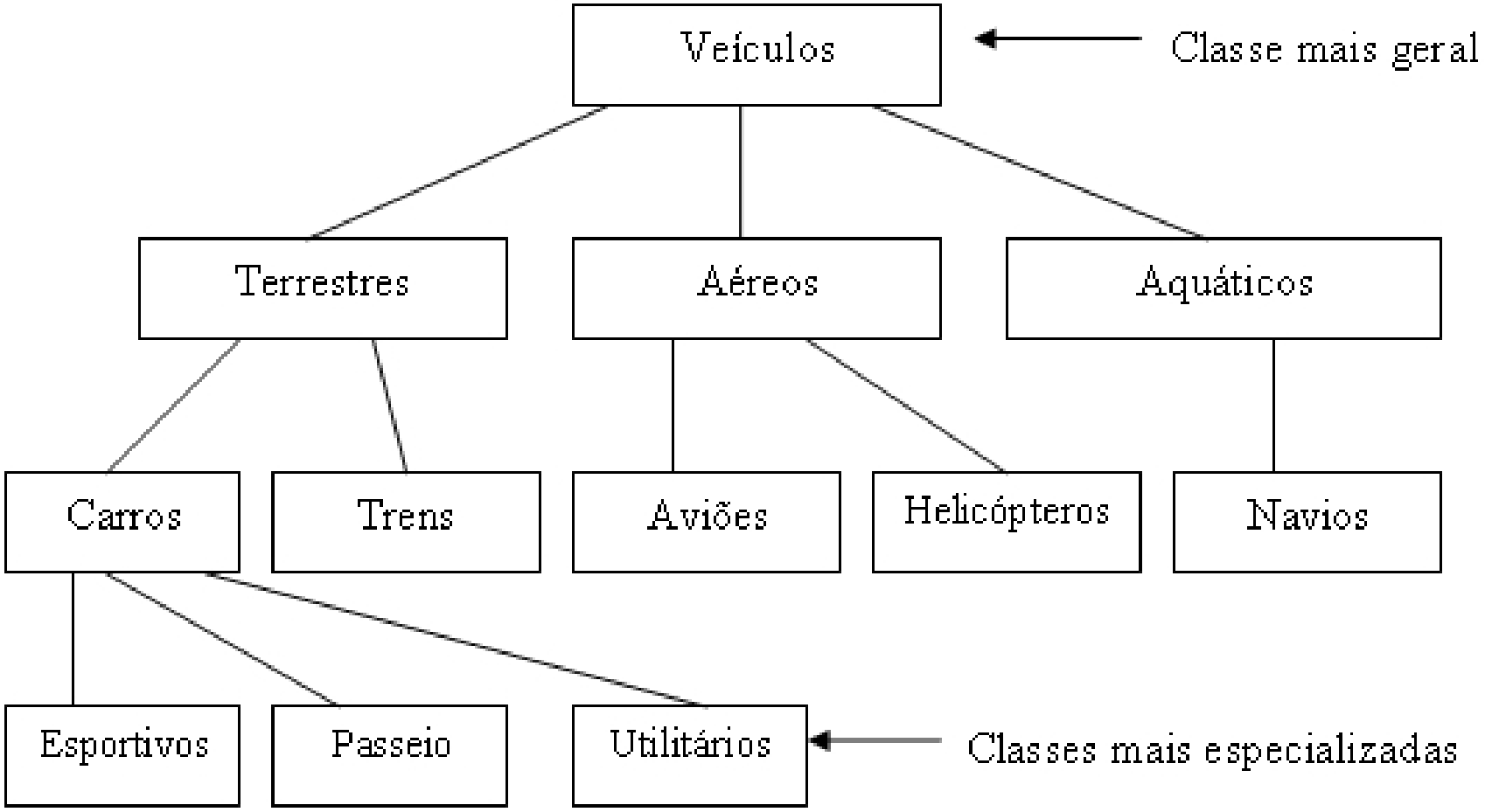
Herança - UML



Exemplo de Herança



Exemplo de Herança



Hierarquia de Classes - UML

- Superclasse Direta: É herdada explicitamente.
- Superclasse Indireta: É herdada de dois ou mais níveis da hierarquia.
- Herança Simples: Herda de somente uma classe básica.
- Herança Múltipla: Herda de múltiplas classes básicas (somente C++).

Herança

- Quando utilizar a herança?
- Realizar a pergunta “É um/uma?”
- Exemplos:
  - Funcionario **é uma** Pessoa?
  - Carro **é um** Veículo?
  - Aluno **é uma** Pessoa?
  - Gerente **é um** Empregado?

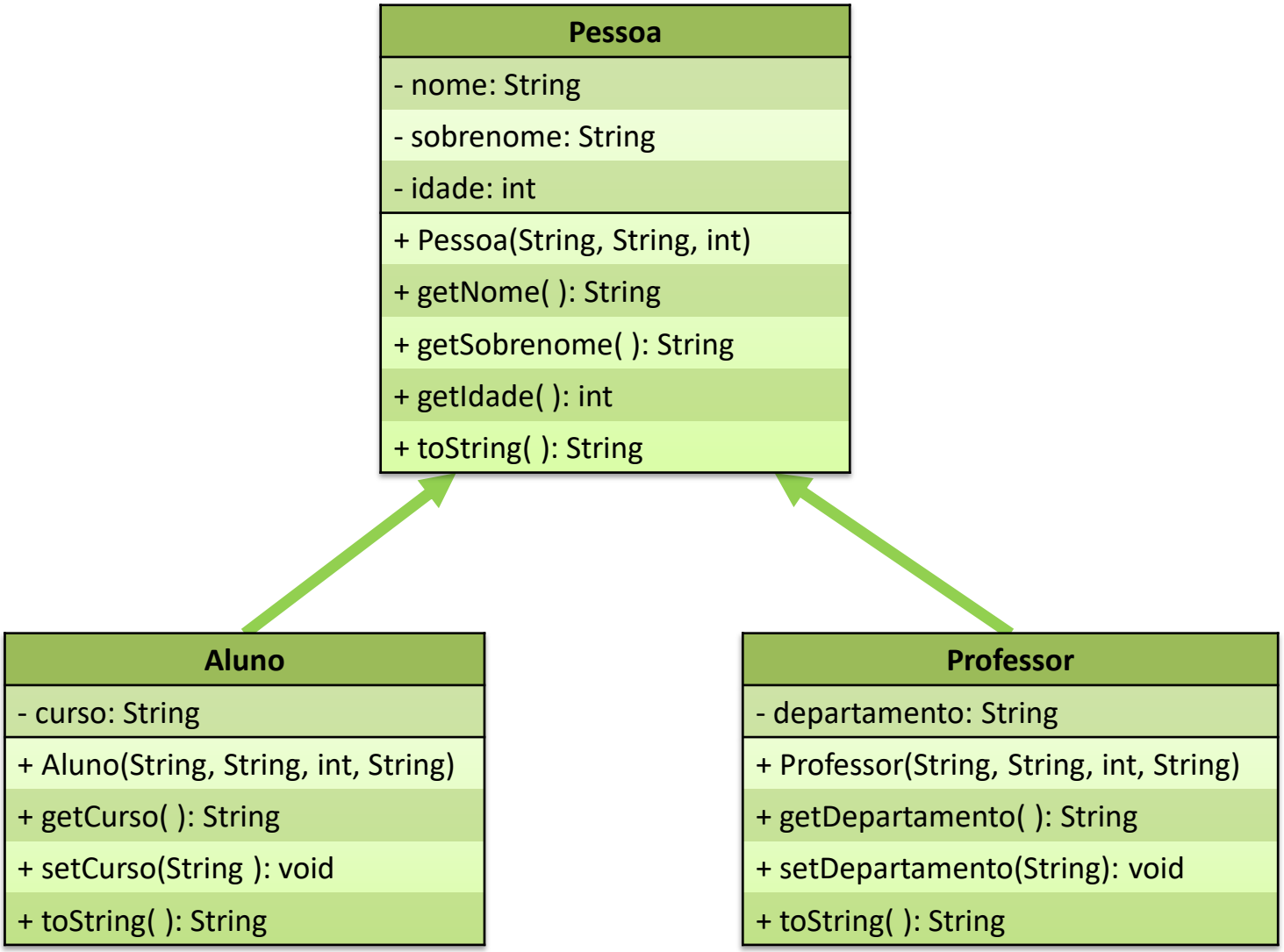
Exemplo

Duas classes que possui vários atributos e métodos iguais.

Aluno
- nome: String
- sobrenome: String
- idade: int
- curso: String
+ Aluno(String, String, int, String)
+ getNome( ): String
+ getSobrenome( ): String
+ getIdade( ): int
+ getCurso( ): String
+ setCurso(String): void
+ toString( ): String

Professor
- nome: String
- sobrenome: String
- idade: int
- departamento: String
+ Professor(String, String, int, String)
+ getNome( ): String
+ getSobrenome( ): String
+ getIdade( ): int
+ getDepartamento( ): String
+ setDepartamento(String): void
+ toString( ): String

Exemplo de Herança



Criação de uma Classe em Java

- A criação de uma classe em Java usa a palavra “class”.

- Sintaxe:

```
<tipo de acesso> class <NomeDaClasse> {
    // atributos e métodos da classe
}
```

- Exemplo:

```
public class Pessoa {
        // comandos
}
```

Criação de uma Classe em Java usando Herança

- A criação de uma classe em Java usando a herança usa a palavra "**extends**"

- Sintaxe:

```
<tipo de acesso> class <NomeDaClasse> extends <NomedaSuperClasse> {
        // atributos e métodos da classe
}
```

- Exemplo:

```
public class Aluno extends Pessoa {
        // comandos
}
```

Criação de uma Classe em Java usando Herança

- Exemplo:

```
public class Aluno extends Pessoa {
        // comandos
}
```

- A classe **Aluno** é uma **sub-classe** de Pessoa

- A classe **Pessoa** é a **super-classe** de Aluno (ou classe base)

Tipo de Acesso aos métodos

	Classe	Pacote	Sub-Classe (mesmo pacote)	Sub-Classe (pacote diferente)	Fora da Classe e pacote
public	sim	sim	sim	sim	sim
protected	sim	sim	sim	sim	não
package	sim	sim	sim	não	não
private	sim	não	não	não	não

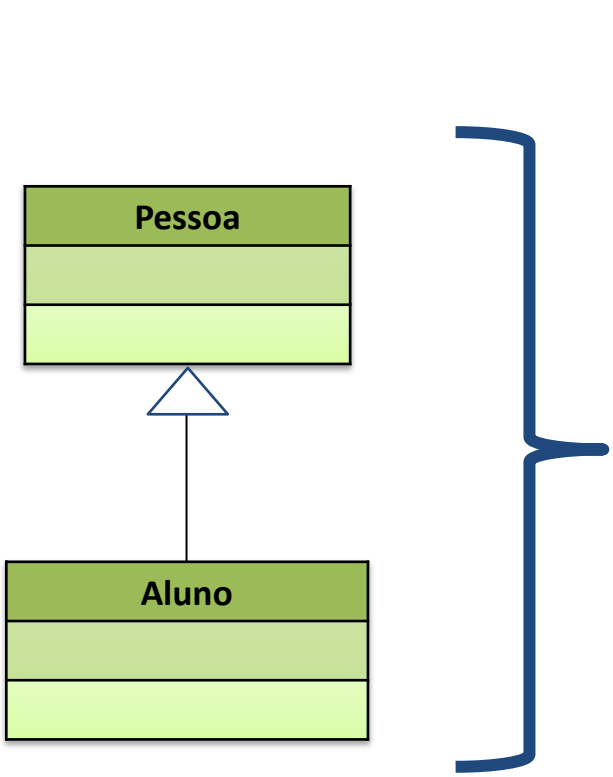
Tipo de Acesso

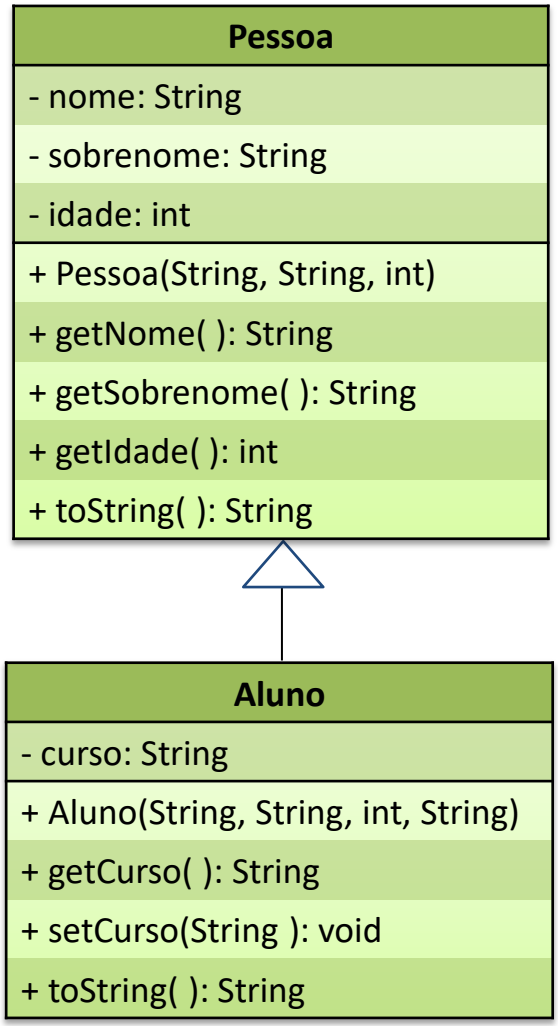
- private: membros private de uma superclasse são acessíveis somente por dentro da própria classe; não são herdados.
- package (default): acessível às classes do pacote.
- protected: O modificador protected especifica que o membro só pode ser acessado em seu próprio pacote e, além disso, por uma subclasse de sua classe em outro pacote.
- public: membros public de uma superclasse são acessíveis para os objetos de suas subclasses

Herança


public class Carro{ }
superclasse
public class Ferrari extends Carro{ }
subclasse
public class BMW extends Carro{ }
subclasse

Exemplo de Herança





Exemplo de Herança

```
public class Pessoa {
    private String nome;
    private String sobrenome;
    private int idade;

    public Pessoa(String nome, String sobrenome, int idade) {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.idade = idade;
    }
    public String getNome() {
        return nome;
    }
    public String getSobrenome() {
        return sobrenome;
    }
    public int getIdade() {
        return idade;
    }
    @Override
    public String toString() {
        return "nome=" + nome + ", sobrenome=" + sobrenome +", idade=" + idade;
    }
}
```

Exemplo de Herança

```
public class Aluno extends Pessoa{
    private String curso;

    public Aluno(String curso, String nome, String sobrenome, int idade) {
        super(nome, sobrenome, idade);
        this.curso = curso;
    }
    public String getCurso() {
        return curso;
    }
    public void setCurso(String curso) {
        this.curso = curso;
    }
    @Override
    public String toString() {
        return "Aluno{" + "curso=" + curso + ", " + super.toString() + '}';
    }
}
```

Objetos e métodos na classe Principal

```java
package exemploherança;

public class Principal {

    public static void main(String[] args) {

        Aluno aluno01 = new Aluno("Engenharia", "Fulano", "Sicrano", 20);
        System.out.println("Aluno 01: " + aluno01);
        Aluno aluno02 = new Aluno("Ciencia da Comp", "Beltrano", "Name", 20);
        System.out.println("Aluno 01: " + aluno02);
        System.out.println("Nome do Aluno 01: " + aluno01.getNome());
        System.out.println("Nome do Aluno 02: " + aluno02.getNome());
    }
}
```

Saída - ExemploHerança (run)

```
run:
Aluno 01: Aluno{curso=Engenharia, nome=Fulano, sobrenome=Sicrano, idade=20}
Aluno 01: Aluno{curso=Ciencia da Comp, nome=Beltrano, sobrenome=Name, idade=20}
Nome do Aluno 01: Fulano
Nome do Aluno 02: Beltrano
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Construtores em subclasses

- O construtor da subclasse invoca primeiro o construtor da superclasse.

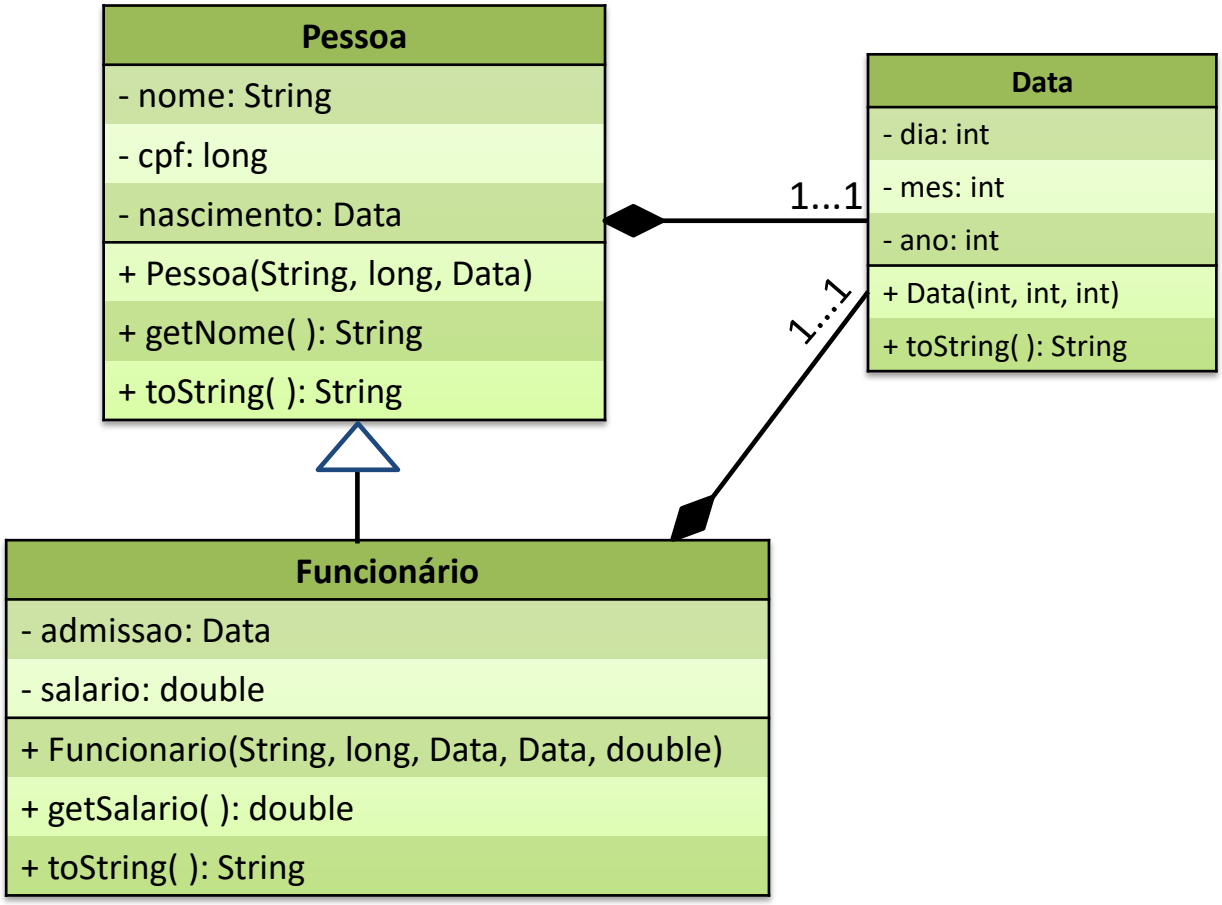
- Os construtores sobrecarregados de uma superclasse não são herdados pelas subclasses

Palavra-chave super

- Nos casos em que o construtor da superclasse é parametrizado, precisamos chamá-lo explicitamente na subclasse.

- A palavra-chave super permite que chamemos o construtor da superclasse na subclasse.

Exemplo de Herança com composição



Exemplo de Herança

```
public class Pessoa {
        private String nome;
        private long cpf;
        private Data nascimento;

        // construtor da classe Pessoa com três parâmetros usando um objeto Data
        public Pessoa(String nome, long cpf, Data nascimento){
                this.nome = nome;
                this.cpf = cpf;
                this.nascimento = nascimento;
        }
        // retorna o nome da pessoa
        public String getNome() {
                return this.nome;
        }
        // retorna uma String com os atributos da classe Data
        public String toString() {
                return "Nome = " + this.nome + ", CPF = " + this.cpf +
                        ", Data de Nascimento = " + this.nascimento;
        }
} // fim da classe Pessoa
```

Exemplo de Herança

```
public class Funcionario extends Pessoa {
        private Data admissao;
        private double salario;

        // construtor da classe Funcionario com cinco parâmetros
        public Funcionario(String nome, long cpf, Data nascimento, Data
                                                admissao, double salario) {
                super(nome, cpf, nascimento);
                this.admissao = admissao;
                this.salario = salario;
        }
        // retorna o salário
        public double getSalario() {
                return this.salario;
        }

        // retorna uma String com os atributos da classe Funcionario
        public String toString() {
                return super.toString() + ", Admissao = " + admissao + ", Salario =
                        " + salario;
        }
} // fim da classe Funcionario
```

Objetos e métodos na classe Principal

```
public class Principal {
        public static void main(String[] args) {
                Data nasc = new Data(22,5,1994);
                System.out.println("Data de nascimento = " + nasc);
                Pessoa pess1 = new Pessoa("Julia", 12345678901L, nasc);
                System.out.println("Pessoa 1: " + pess1);
                Pessoa pess2 = new Pessoa("Clarisse", 98765432109L, 1, 9, 1980);
                System.out.println("Pessoa 2: " + pess2);
                System.out.println("Nome da Pessoa 2: " + pess2.getNome());
                System.out.println("Classe Funcionario");
                Funcionario func1 = new Funcionario("Cristina", 4567891231L, new
                                        Data(17,4,1993), new Data(1,2,2015), 1500);
                System.out.println("Salario = " + func1.getSalario());
                System.out.println("Nome = " + func1.getNome());
                System.out.println("Funcionario 1: " + func1);

        } // fim do método main

} // fim da classe Principal
```

Objetos e métodos na classe Principal

```
6   package principal;
7   /**
8    * @author Isaac
9    */
10  public class Principal {
11      /**
12       * @param args the command line arguments
13       */
14      public static void main(String[] args) {
15          // TODO code application logic here
16          Data nasc;
17          nasc = new Data(22,5,1994);
18          System.out.println("Data de  nascimento  =  "  +  nasc);
19          Pessoa  pess1 = new  Pessoa("Julia",  "12345678901", nasc);
20          System.out.println("Pessoa 1:  "  +  pess1);
21          Pessoa  pess2 =  new  Pessoa("Clarisse",  "98765432109",  1,  9,  1980);
22          System.out.println("Pessoa 2:  "  +  pess2);
23          System.out.println("Nome da  Pessoa  2:  "  + pess2.getNome());
24          System.out.println("Classe Funcionario");
25          Funcionario func1 = new Funcionario("Cristina", "4567891231", new
26          System.out.println("Salario = " + func1.getSalario());
27          System.out.println("Nome = " + func1.getNome());
28          System.out.println("Funcionario 1: " + func1);
29      }
30  }
```

```
Data(17,4,1993), new Data(1,2,2015), 1500);
```

```
run:
Data de  nascimento  =  22/5/1994
Pessoa 1:  Nome  =  Julia,  CPF  =  12345678901,  Data  de  Nascimento  =  22/5/1994
Pessoa 2:  Nome  =  Clarisse,  CPF  =  98765432109,  Data  de  Nascimento  =  1/9/1980
Nome da  Pessoa  2:  Clarisse
Classe Funcionario
Salario = 1500.0
Nome = Cristina
Funcionario 1: Nome  =  Cristina,  CPF  =  4567891231,  Data  de  Nascimento  =  17/4/1993, Admissao = 1/2/2015, Salario = 1500.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Herança vs. Composição

- Composição e Herança são dois mecanismos para reutilizar funcionalidades.
- Na herança temos o conceito de Classe Base/Superclasse que é a classe que foi herdada pelas Classes Derivadas/Subclasses.
- A composição por sua vez estende uma classe pela delegação de trabalho para outro objeto.

Herança vs. Composição

é um versus **tem um**

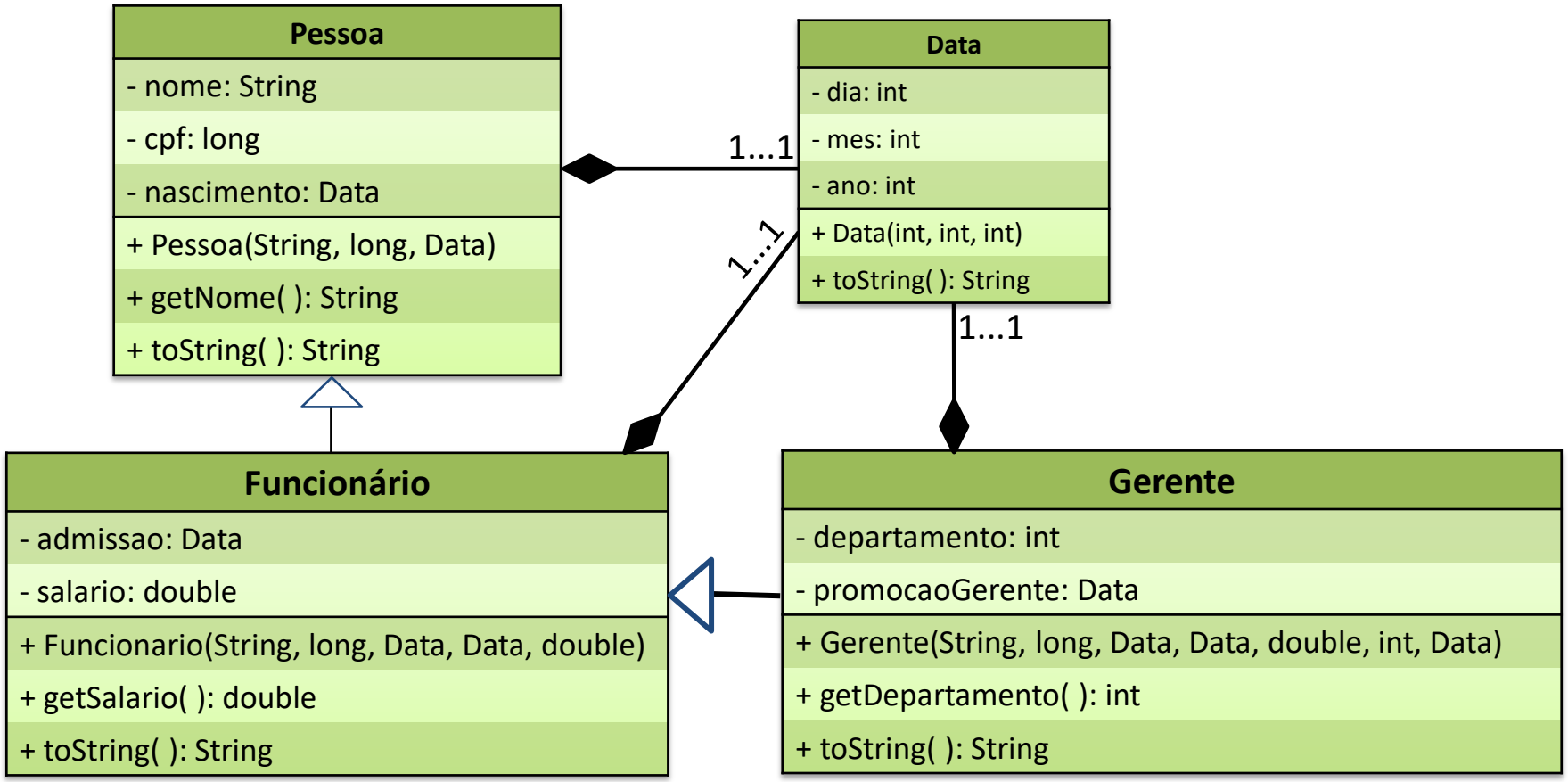
- é um (Herança)
  - O objeto da classe derivada pode ser tratado com objeto da classe básica.
  - Exemplo: O carro é um veículo.
    - Os atributos e métodos de veículos também se aplicam a carro

- tem um (Composição)
  - O objeto contém um ou mais objetos de outras classes como membros.
  - Exemplo: O carro tem (uma) direção.

Exercícios

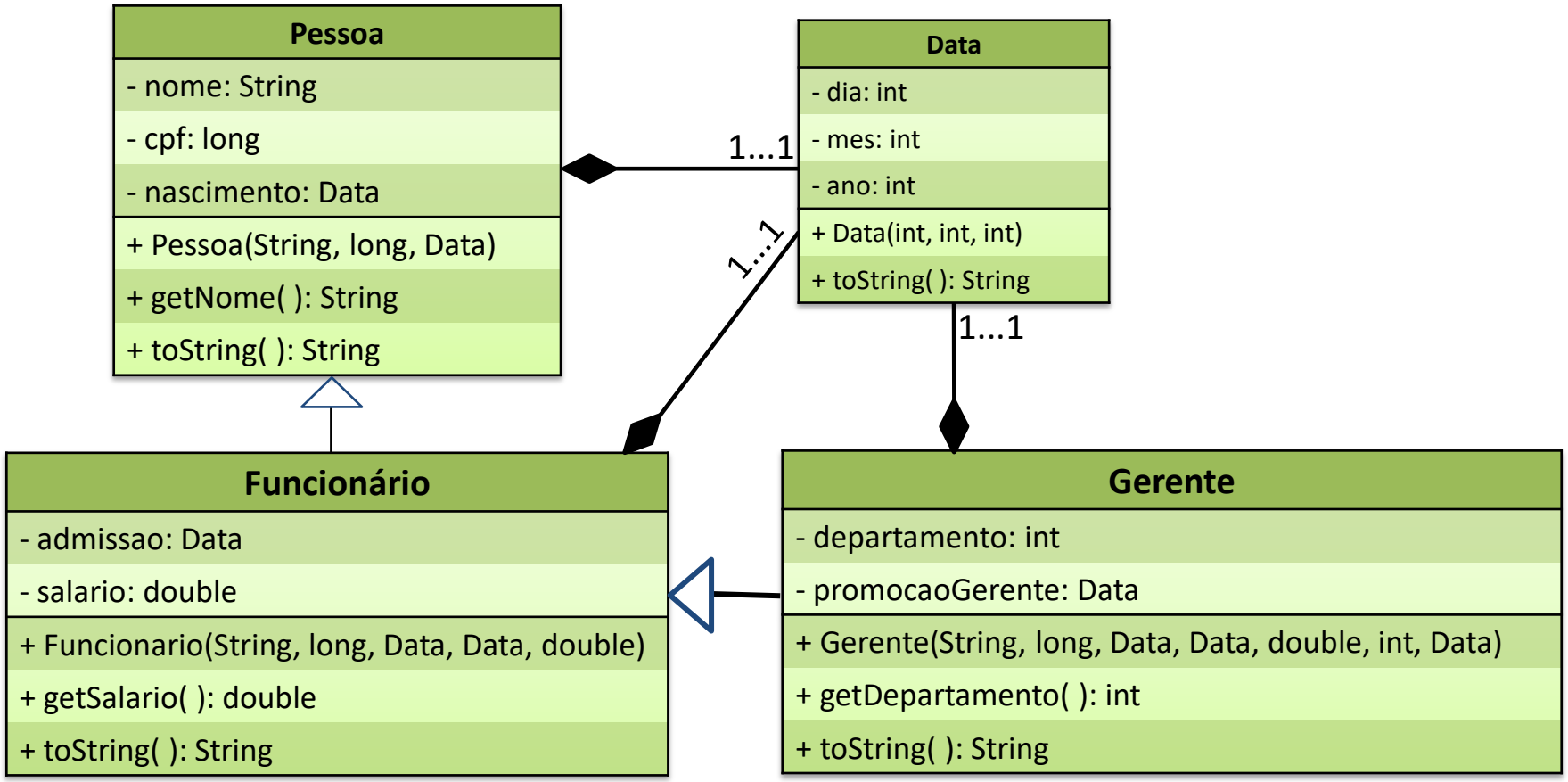
Exercício 01 – Implemente

Implemente no NetBeans as classes abaixo.



Exercício 01 – Implemente

Acessar os métodos da classe Gerente e Funcionário no método main.



Exercício 02 – Sistema político brasileiro

Implemente a Hierarquia de herança do Sistema Político Brasileiro.

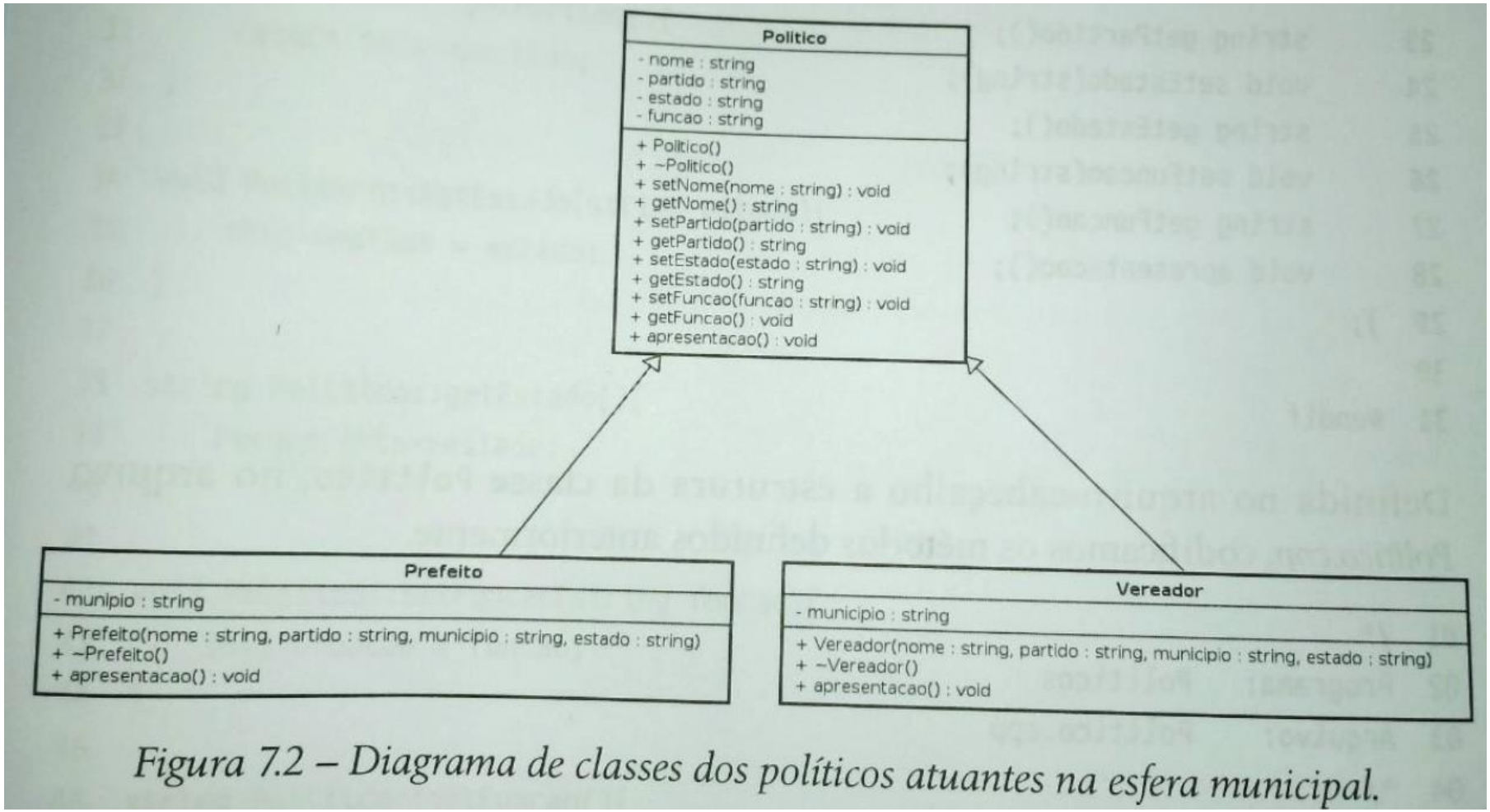


Figura 7.2 – Diagrama de classes dos políticos atuantes na esfera municipal.

Fonte: Introdução a Orientação a Objetos com C++ e Python - Orlando Saraiva Jr.

Exercício 03 – Implemente

(Deitel 9.8) Escreva uma hierarquia de herança para as classes Quadrilatero, Trapezio, Paralelogramo, Retangulo e Quadrado. Use Quadrilatero como superclasse da hierarquia. Crie e use uma classe Ponto para representar os pontos (x, y) de cada forma. Faça a hierarquia o mais profunda (isto é, com muitos níveis) possível. Especifique as variáveis de instância e os métodos para cada classe. As variáveis de instância private de Quadrilatero devem ser Pontos npara os quatro pontos que delimitam o quadrilátero. Escreva um programa que instancia objetos de suas classes e gera saída da área de cada objeto (exceto Quadrilatero). A entrada será feita com a posição de 4 pontos (x, y).

Exercício 03 – Implemente

Área Trapézio: