SENAI CIMATEC

SISTEMA FIEB

TECNOLOGIA E INOVAÇÃO EM PROL DA INDÚSTRIA

SENAI CIMATEC

SISTEMA FIEB

Curso Técnico em Informática

SENAI CIMATEC SISTEMA FIEB
Federação das Indústrias do Estado da Bahia

Polimorfismo e Classes Abstratas

Prof: Diego Corrêa

Conceito

- É originário do **grego**, e quer dizer “muitas formas”

- Significa que um mesmo tipo de objeto, sob certas condições, pode realizar ações diferentes ao receber uma mesma mensagem

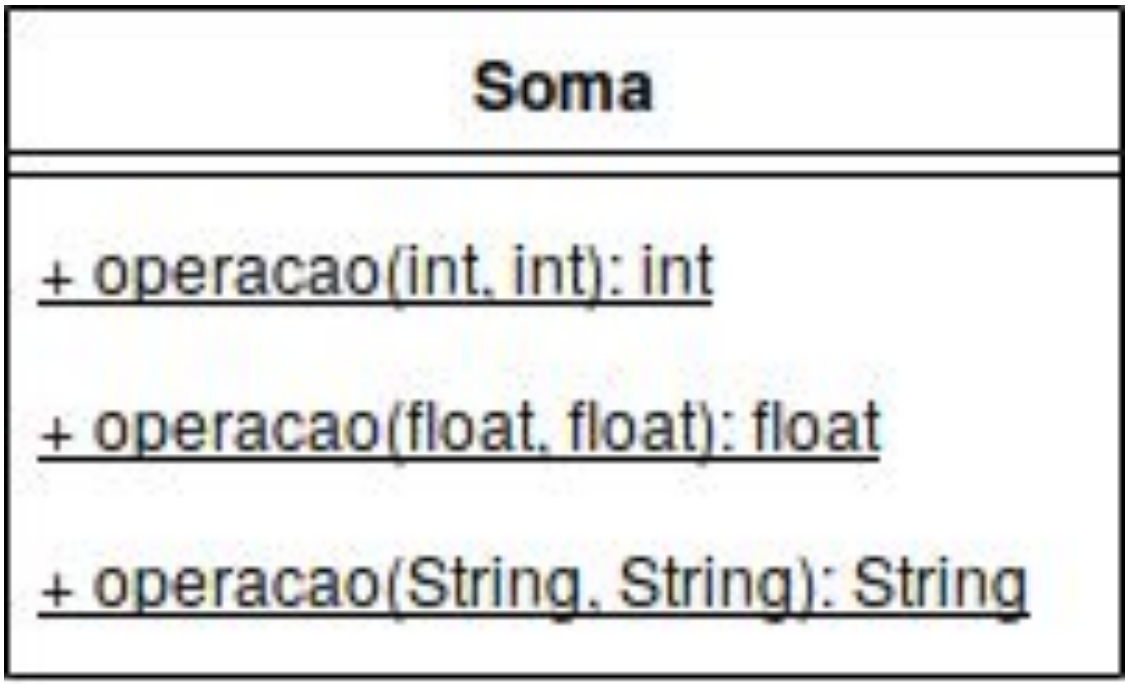
Conceito

- O **Polimorfismo** é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução

- Com o **Polimorfismo**, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes

Tipos de Polimorfismo

- **Sobrecarga**: Dependendo dos parâmetros métodos/construtores diferentes são ativados



SENAI CIMATEC SISTEMA FIEB
Federação das Indústrias do Estado da Bahia

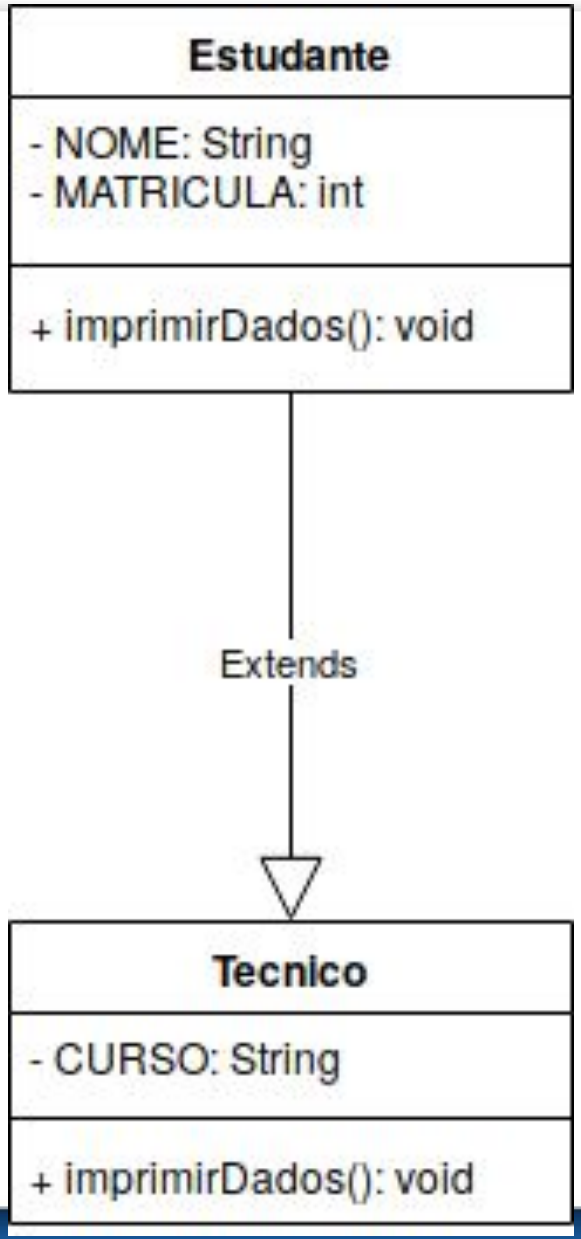
Código

```
public class Soma {
    public static int operacao(int a, int b) {
        return a + b;
    }

    public static float operacao(float a, float b) {
        return a + b;
    }

    public static String operacao(String a, String b) {
        return a + b;
    }
}
```

Tipos de Polimorfismo

- **Sobreposição**: Métodos das classes têm uma tomada lógica diferente dependendo
  - Através do decorador @Override métodos podem ser sobrepor outros

Tipos de Polimorfismo



Código

```java
public class Estudante {
    private final String NOME;
    private final long MATRICULA;

    Estudante(String nome, long matricula){
        this.NOME = nome;
        this.MATRICULA = matricula;
    }

    public void imprimirDados() {
        System.out.println("Nome: " + this.NOME);
        System.out.println("Matricula: " + this.MATRICULA);
    }
}
```

```java
public class Tecnico extends Estudante{
    private String CURSO;

    Tecnico(String curso, String nome, long matricula) {
        super(nome, matricula);
        this.CURSO = curso;
    }

    @Override
    public void imprimirDados() {
        System.out.println("################################################");
        System.out.println("############# Aluno do curso Tecnico #############");
        super.imprimirDados();
        System.out.println("Curso: " + this.CURSO);
    }
}
```

Generalização da Classe

- A herança mantém as características da superclasse na subclasse
- Assim como mantém uma herança do tipo também
- Ou seja, a subclasse pode ser identificado como dois tipos

```
public static Estudante tecnico() {
    return new Tecnico("Desenvolvimento de Software", "Tecnaldo Tecnilson", 20020191);
}
```

Generalização da Classe

Para diferenciar e identificar a qual **classe** aquela **instância** pertence utiliza-se o **instanceof**

Generalização da Classe

```
if (estudante instanceof Tecnico) {
    System.out.println("############## Aluno do curso Tecnico ##############");
    System.out.println("+ Local: Todos os alunos assistem aula no CIMATEC 4");
    estudante.imprimirDados();
} else if(estudante instanceof Graduacao) {
    System.out.println("############## Aluno do curso Gradução ##############");
    estudante.imprimirDados();
}
```

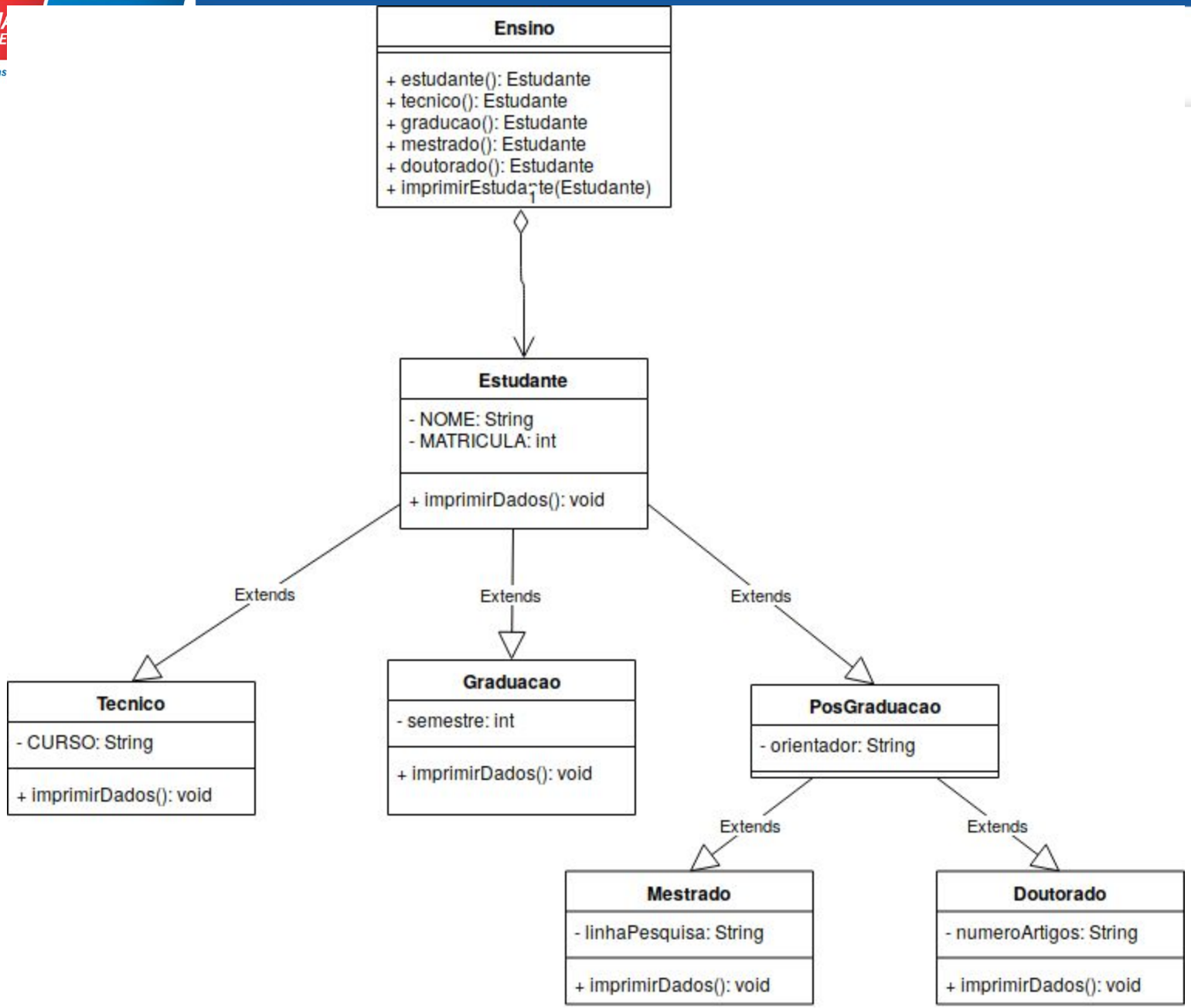
Exercício

Continuando o exercício dos estudantes:

- Criar uma instância de cada estudante em métodos separados
- Retornar essas instâncias como a superclasse “Estudante”
- Criar um método que recebe uma instância da superclasse “Estudante” e dependendo do tipo do estudante imprimir uma mensagem diferente, além de chamar o método para imprimir os dados

Exercício

```
public static void main(String[] args) {
    imprimirEstudante(estudante());
    imprimirEstudante(tecnico());
    imprimirEstudante(graduacao());
    imprimirEstudante(mestrado());
    imprimirEstudante(doutorado());
}

public static Estudante estudante() {

}

public static Estudante tecnico() {

}

public static Estudante graduacao() {

}

public static Estudante mestrado() {

}

public static Estudante doutorado() {

}

public static void imprimirEstudante(Estudante estudante) {

}
```


Ensino
+ estudante(): Estudante
+ tecnico(): Estudante
+ graducao(): Estudante
+ mestrado(): Estudante
+ doutorado(): Estudante
+ imprimirEstudante(Estudante)
1
Estudante
- NOME: String
- MATRICULA: int
+ imprimirDados(): void
Extends
Extends
Extends
Tecnico
- CURSO: String
+ imprimirDados(): void
Graduacao
- semestre: int
+ imprimirDados(): void
PosGraduacao
- orientador: String
Extends
Extends
Mestrado
- linhaPesquisa: String
+ imprimirDados(): void
Doutorado
- numeroArtigos: String
+ imprimirDados(): void

Abstração

Generalização

- Nos momento em que uma classe de alta ordem na herança não pode ser instanciado

- Utiliza-se as classes abstratas as quais as implementações podem ser deixadas para as subclasses

Generalização

- Uma classe declarada como **Abstrata (abstract)** é conhecida como classe abstrata
- Pode possuir métodos abstratos ou métodos não abstratos
- Para usar é necessário herança
- A superclasse declara o escopo do método para a subclasse implementar o corpo do método

Generalização

- Um classe abstrata necessita ser declarada com a palavra reservada abstract
- Não pode ser instanciada
- Pode possuir métodos estáticos e construtores
- Pode possuir métodos declarados como final para que a subclasse não implemente ou altere implementações prévias com o **@Override**

Generalização

```
public abstract class Estudante {
    private final String NOME;
    private final long MATRICULA;

    Estudante(String nome, long matricula){
        this.NOME = nome;
        this.MATRICULA = matricula;
    }

    public void imprimirDados() {
        System.out.println("Nome: " + this.NOME);
        System.out.println("Matricula: " + this.MATRICULA);
    }
}
```

Generalização

- Assim a superclasse não pode ir instanciada e suas subclasses podem utilizar os atributos e métodos

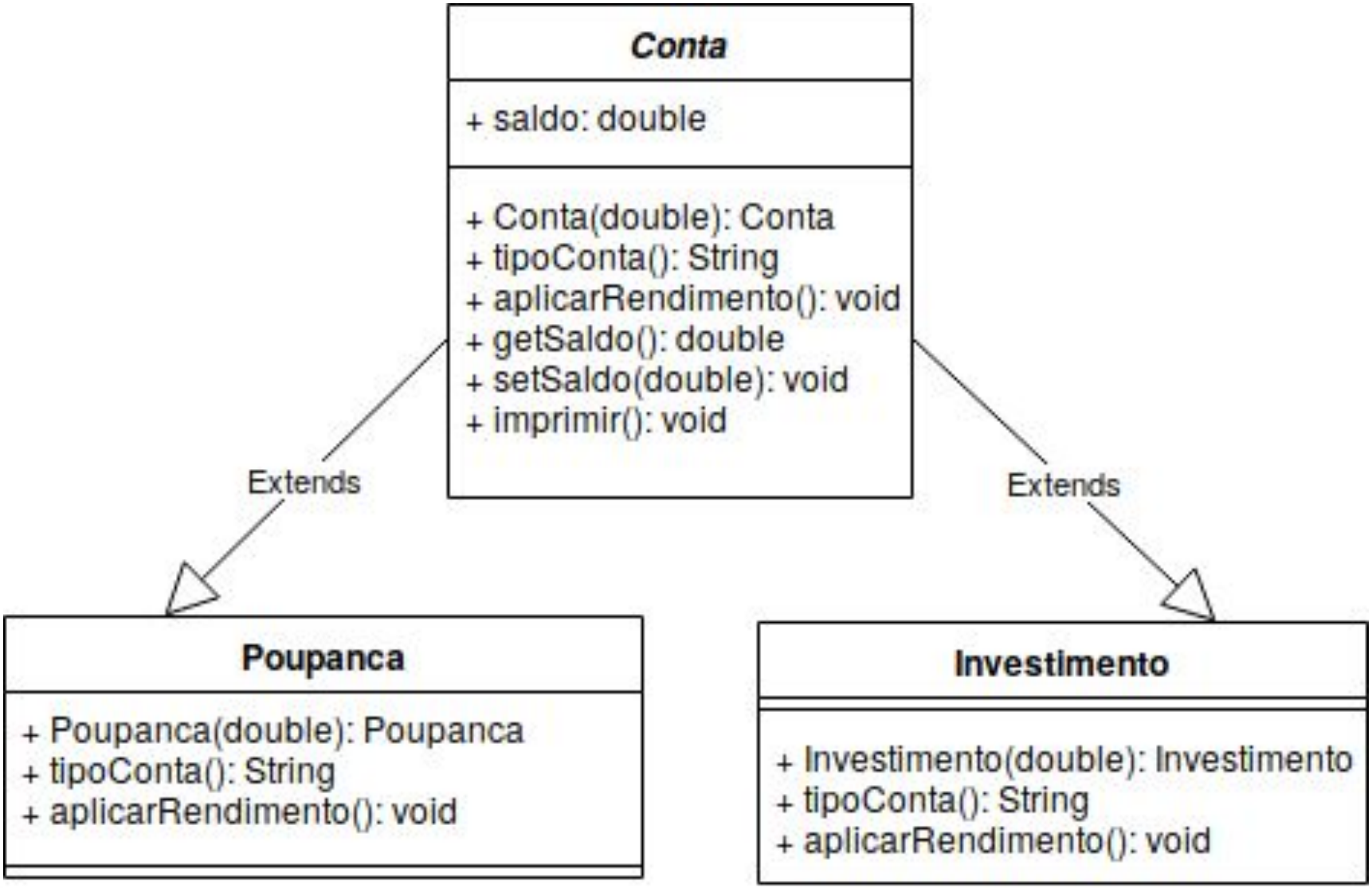
- Classes e métodos abstratos possuem seus nomes em itálico no **UML**

Exercicio

Implementar o esquema do UML de contas a seguir, criando métodos abstratos e não abstratos na classe Conta

```
public class AplicacaoConta {
    public static void main(String[] args) {
        Poupanca p = new Poupanca(500.0);
        System.out.println("Tipo da Conta: " + p.tipoConta());
        System.out.println("Saldo: " + p.getSaldo());
        p.aplicarRendimento();
        System.out.println("Saldo depois do rendimento: " + p.getSaldo());
        Investimento i = new Investimento(1000.0);
        System.out.println("Tipo da Conta: " + i.tipoConta());
        System.out.println("Saldo: " +i.getSaldo());
        p.aplicarRendimento();
        System.out.println("Saldo depois do rendimento: " + p.getSaldo());
    }
}
```

Exercicio


Conta
+ saldo: double
+ Conta(double): Conta
+ tipoConta(): String
+ aplicarRendimento(): void
+ getSaldo(): double
+ setSaldo(double): void
+ imprimir(): void
Extends
Extends
Poupanca
+ Poupanca(double): Poupanca
+ tipoConta(): String
+ aplicarRendimento(): void
Investimento
+ Investimento(double): Investimento
+ tipoConta(): String
+ aplicarRendimento(): void

SENAI CIMATEC SISTEMA FIEB
Federação das Indústrias do Estado da Bahia

Exercicio