ânima EDUCAÇÃO

UNISUL
UNIVERSIDADE DO SUL DE SANTA CATARINA

PROGRAMAÇÃO DE SOLUÇÕES COMPUTACIONAIS

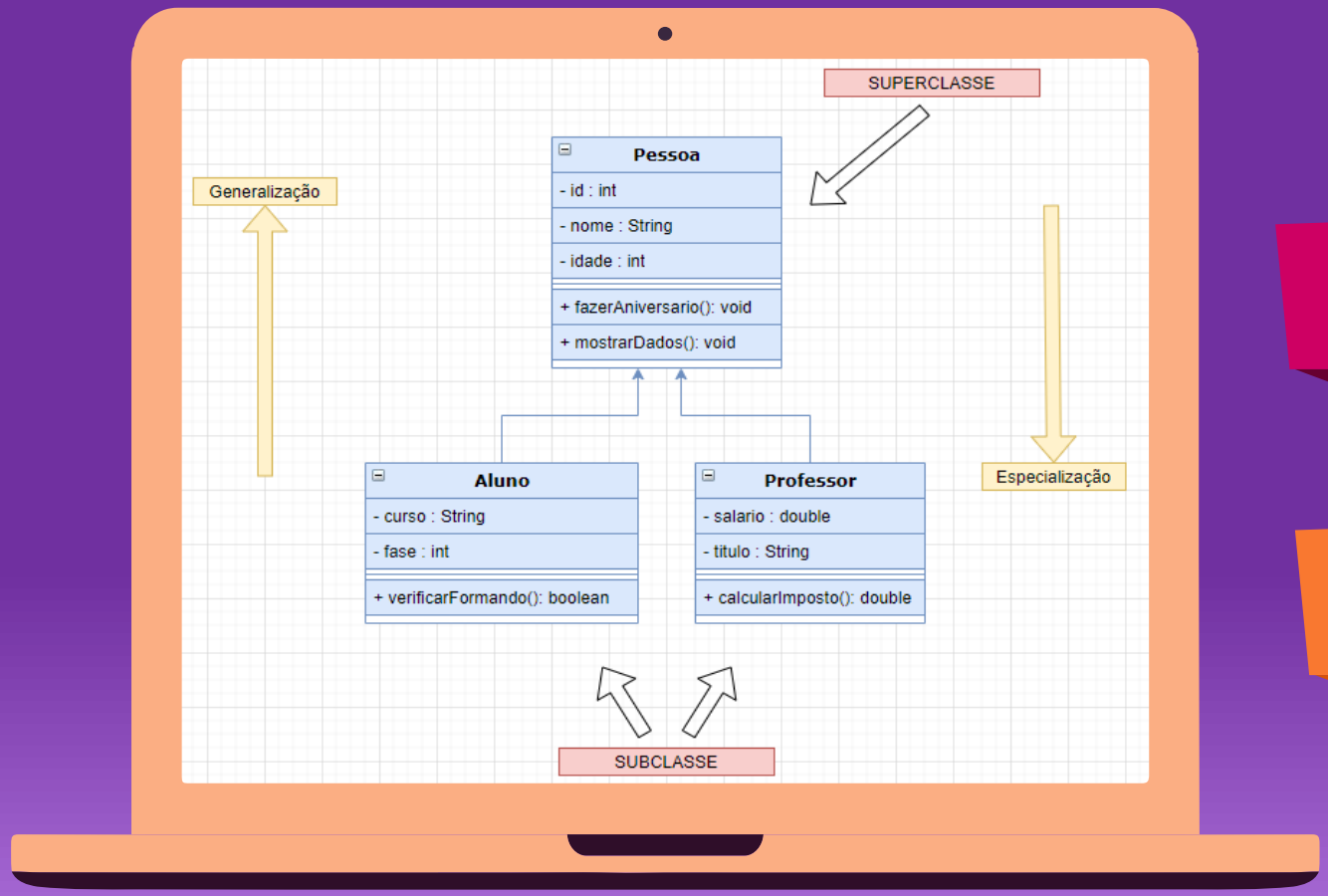
Prof. Ricardo Ribeiro Assink

POO - HERANÇA

O QUE É HERANÇA?

- Técnica de programação orientada a objetos usada para organizar e criar classes para reuso;
- Possibilita derivar uma classe de outra já existente;
- A classe existente é chamada de **SUPERCLASSE** ou classe base;
- A classe derivada é chamada de **SUBCLASSE**;
- A **SUBCLASSE** herda as características da superclasse, ou seja, a subclasse herda os métodos e os atributos definidos pela superclasse;

POO - HERANÇA



A SUPERCLASSE define tudo que SE REPETE nas SUBCLASSE.

A SUBCLASSE ESTENDE a SUPERCLASSE.

Descrição de imagem:
Mostra caixas representando a SUPERCLASSE (Pessoa) no topo e duas SUBCLASSE na base(Aluno e Professor).

POO - HERANÇA

O programador pode implementar uma classe derivada como for preciso, adicionando novos atributos ou métodos, ou até mesmo, modificando os herdados.

Reuso de software é uma vantagem no uso de herança.

Em Java, usa-se a palavra reservada **extends** para estabelecer uma relação de herança.

POO - HERANÇA

```
// A SUPERCLASSE deve ser importada
import Model.Pessoa;


// A classe Aluno é SUBCLASSE da SUPERCLASSE Pessoa
public class Aluno extends Pessoa {

}
```

O programador pode implementar uma classe derivada como for preciso, adicionando novos atributos ou métodos, ou até mesmo, modificando os herdados.

Reuso de software é uma vantagem no uso de herança.

Em JAVA, usa-se a palavra reservada extends para estabelecer uma relação de herança.

POO - OVERLOADING

Overloading (sobrecarregar) é uma característica que permite que uma classe tenha mais de um método com o mesmo nome, se seus parâmetros forem diferentes.

Permite:

- Múltiplos métodos com o mesmo nome (Assinaturas diferentes).
- Permite ao programador definir operações semelhantes com parâmetros diferentes.

POO - OVERLOADING

Três maneiras de sobrecarregar um método:

01 NÚMEROS DE PARÂMETROS

metodoAdicionar (int, int)
metodoAdicionar (int, int, int)

02 TIPO DE DADOS DOS PARÂMETROS

metodoAdicionar (int, int)
metodoAdicionar (int, double)

03 SEQUÊNCIA DO TIPO DE DADOS DOS PARÂMETROS

metodoAdicionar (int, double)
metodoAdicionar (double, int)

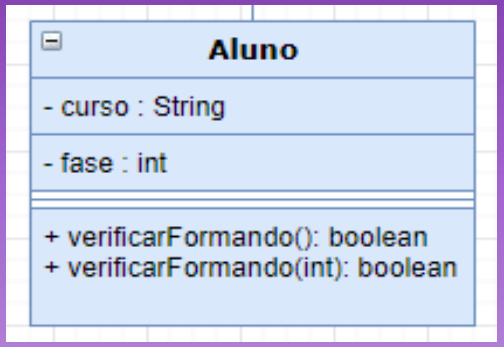
POO - OVERLOADING

```
public boolean verificarFormando( ) {
    if (this.fase == 10) {
        return true;
    } else {
        return false;
    }
}

// Exemplo de overload
public boolean verificarFormando(int fase) {
    if (fase == 10) {
        return true;
    } else {
        return false;
    }
}
```

É semelhante à sobrecarga de construtor em Java, que permite que uma classe tenha mais de um construtor com diferentes listas de argumentos.

Overloading significa sobrecarregar os métodos dentro da mesma classe.



POO - OVERRIDING

Declarar um método na subclasse que já está presente na classe pai é conhecido como método Overriding.

A substituição é feita para que uma classe filho possa dar sua própria implementação a um método que já é fornecido pela classe pai.

Nesse caso, o método na classe pai é chamado de método substituído e o método na classe filho é chamado de método de substituição.

POO - OVERRIDING

Overriding (sobrescrever)

1. Métodos em classes diferentes (subclasse e superclasse).
2. Possuem a mesma assinatura.
3. Permite ao programador definir operações semelhantes.

POO - OVERRIDING

REGRAS DE IMPLEMENTAÇÃO DO MÉTODO EM JAVA:

01 A lista de argumentos do método de substituição (método da subclasse) deve corresponder ao método substituído (o método da superclasse). Os tipos de dados dos argumentos e sua sequência devem corresponder exatamente.

02 O modificador de acesso do método de substituição (método de subclasse) não pode ser mais restritivo do que o método substituído da classe pai.

Por exemplo, se o modificador de acesso do método da classe principal for público, o método de substituição não pode ter modificador de acesso privado ou protegido.

POO - OVERRIDING

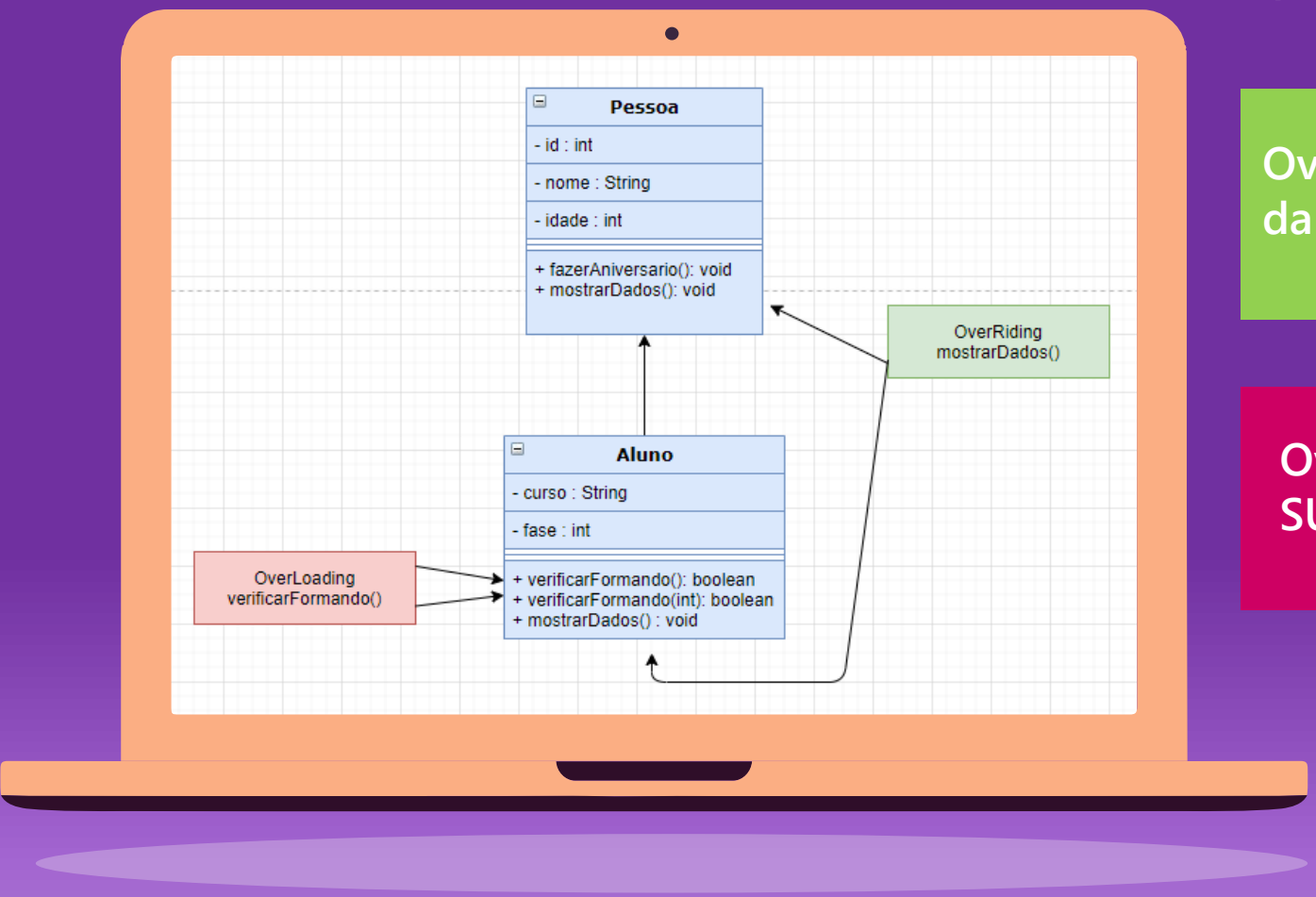
REGRAS DE IMPLEMENTAÇÃO DO MÉTODO EM JAVA:

03 Os métodos privados, estáticos e finais não podem ser substituídos, pois são locais para a classe.

No entanto, os métodos estáticos podem ser redeclarados na subclasse, neste caso, o método de subclasse atuaria de forma diferente e não terá nada a ver com o mesmo método estático da classe pai.

04 A ligação dos métodos substituídos ocorre no tempo de execução, que é conhecido como ligação dinâmica.

POO – OVERRIDING E OVERLOADING



Overloading significa sobrecarregar os métodos dentro da mesma classe.

Overriding significa sobrescrever os métodos entre a SUPERCLASSE e SUBCLASSE.

Descrição de imagem:
Mostra apenas a representação gráfica do texto deste slide.

POO - SUPER

O método SUPER invoca um método da SUPERCLASSE mesmo que este mesmo método exista na SUBCLASSE.

Quando ocorre Override entre SUPERCLASSE e SUBCLASSE, as vezes será necessário acessar métodos da SUPERCLASSE por meio de um OBJETO da SUBCLASSE.

Para isso usamos o método SUPER.

Veja a seguir um exemplo comum de uso do super com os construtores.
Lembre-se que o SUPER não serve apenas para os construtores, mas para qualquer método.

POO - SUPER

```
// Método Construtor de Objeto Vazio
  public Aluno() {
  }

  // Método Construtor de Objeto, inserindo dados
  public Aluno(String curso, int fase) {
      this.curso = curso;
      this.fase = fase;
  }

  // Método Construtor usando também o construtor da SUPERCLASSE
  public Aluno(String curso, int fase, int id, String nome, int idade) {
      super(id, nome, idade);
      this.curso = curso;
      this.fase = fase;
  }
```

Ao lado vemos parte do código da SUBCLASSE Aluno. Existem 3 construtores neste código:

1. construtor de objeto vazio
2. construtor de objeto inserindo dados
3. construtor de objeto completo com dados da SUPERCLASSE (Pessoa).

Como Aluno é subclasse de Pessoa, ao instanciar um objeto de Aluno, este herda os atributos e métodos de Pessoa.
Se apenas instanciarmos um objeto de Aluno sem usar o SUPER, o objeto terá apenas os atributos da subclasse preenchidos, os atributos da classe Pessoa são criados, porém, vazios.

Então para o objeto ficar "completo", ou seja, com todos os dados preenchidos, o ideal seria que ao instanciar o objetos da subclasse Aluno, este também carregasse o construtor da classe Pessoa.

POO - SUPER

Ao lado vemos parte do código da SUBCLASSE Aluno.

```
// Exemplo de Override e super na classe Aluno

@Override
    public void mostraDados() {
        super.mostraDados();
        System.out.println("Curso: " + this.curso);
        System.out.println("Fase: " + this.fase);
    }
```

Trata-se de um exemplo de uso de Override e acesso a método da SUPERCLASSE usando SUPER.

Note que o método mostraDados() existe tanto na SUPERCLASSE Pessoa como na SUBCLASSE Aluno, caracterizando o Override.

Se um objeto da SUBCLASSE Aluno invocar o método mostrarDados(), este TAMBÉM invoca o método mostrarDados() da SUPERCLASSE Pessoa.

VAMOS VER OS DETALHES DIRETO NO CÓDIGO?

Acompanhe o professor para explicações mais detalhadas. Observe o funcionamento e uso dos itens estudados anteriormente direto em uma IDE.

O código fonte dos arquivos estudados estarão nos anexos.

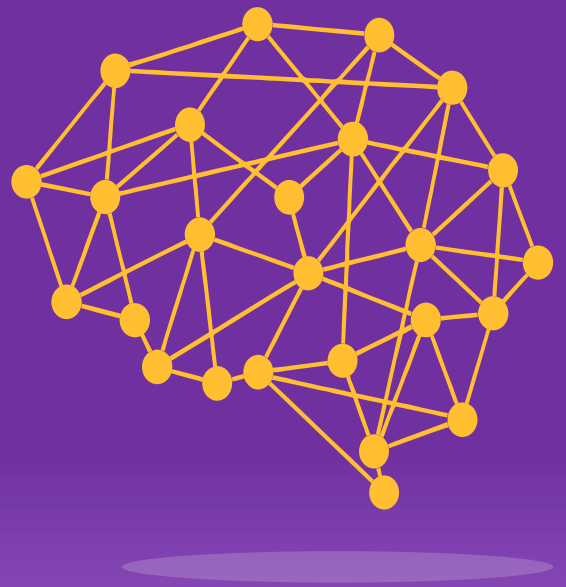
Presta atenção!!!

Busca Ativa!

1. Procure e assista vídeos na internet que falem de Herança, Overriding e Overloading.

2. Implemente e execute TODOS os exemplos da aula de hoje.

3. Adicione exemplos de Override e Overload usando as classes Pessoa e Aluno. Os exemplos já estão comentados para você se guiar.



FIM

ANEXO 1 – SUPERCLASSE PESSOA

```
public class Pessoa {

    // Atributos
    private int id;
    private String nome;
    private int idade;

    // Método Construtor de Objeto Vazio
    public Pessoa() {
    }

    // Método Construtor de Objeto, inserindo dados
    public Pessoa(int id, String nome, int idade) {
        this.id = id;
        this.nome = nome;
        this.idade = idade;
    }

    // Métodos GET e SET
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

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

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    // Método adicional
    public void fazerAniversario() {
        this.idade++;
    }

    public void mostraDados() {
        System.out.println("ID: " + this.id);
        System.out.println("Nome: " + this.nome);
        System.out.println("Idade: " + this.idade);
    }

}
```

ANEXO 2 – SUBCLASSE ALUNO

```
public class Aluno extends Pessoa {

    // Atributos
    private String curso;
    private int fase;

    // Método Construtor de Objeto Vazio
    public Aluno() {
    }

    // Método Construtor de Objeto, inserindo dados
    public Aluno(String curso, int fase) {
        this.curso = curso;
        this.fase = fase;
    }

    // Método Construtor usando também o construtor da SUPERCLASSE
    public Aluno(String curso, int fase, int id, String nome, int idade) {
        super(id, nome, idade);
        this.curso = curso;
        this.fase = fase;
    }



    // Métodos GET e SET
    public String getCurso() {
        return curso;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    public int getFase() {
        return fase;
    }

    public void setFase(int fase) {
        this.fase = fase;
    }

    // Método adicional
    public boolean verificarFormando() {

        if (this.fase == 10) {
            return true;
        } else {
            return false;
        }
    }


    // Exemplo de overload
    public boolean verificarFormando(int fase) {

        if (fase == 10) {
            return true;
        } else {
            return false;
        }
    }


    // Exemplo de Override
    @Override
    public void mostraDados() {
        super.mostraDados();
        System.out.println("Curso: " + this.curso);
        System.out.println("Fase: " + this.fase);
    }


}
```