ânima EDUCAÇÃO

UNISUL
UNIVERSIDADE DO SUL DE SANTA CATARINA

PROGRAMAÇÃO DE SOLUÇÕES COMPUTACIONAIS

Prof. Ricardo Ribeiro Assink

POO – MODIFICADOR FINAL

Onde posso usar?

- Classes
- Atributos e outras variáveis
- Métodos

POO – MODIFICADOR FINAL

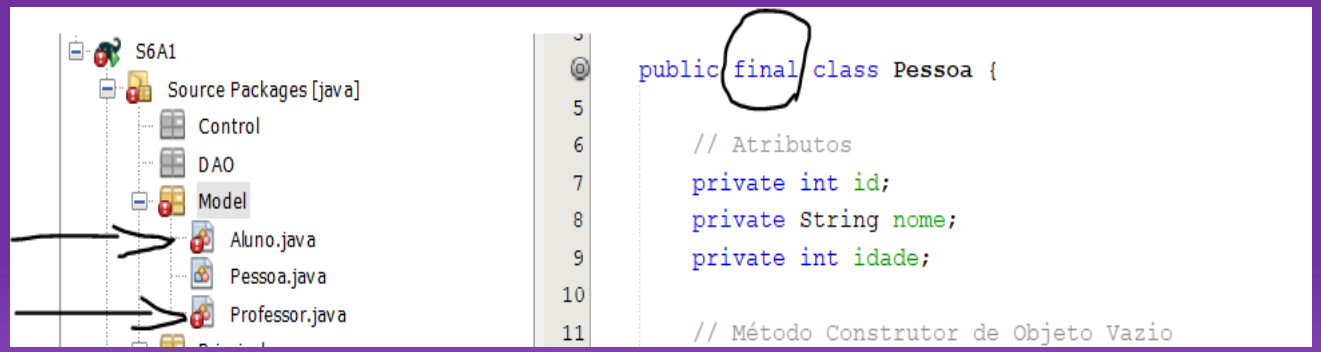
1 – Classe final

Classes ***final*** não podem ser estendidas, ou seja, ninguém pode ser herdeiro dessa classe.

Usamos quando queremos garantir que o comportamento da classe não seja modificado e que a mesma não tenha herdeiros.

POO – MODIFICADOR FINAL

1 – Classe final



A Classe final interrompe o fluxo de herdeiros.

Na última aula vimos as classes Pessoa, Aluno e Professor.

A Classe Pessoa não pode ser final.
Se isso acontecer, Aluno e Professor não podem estendê-la (Não podem herdar nada de Pessoa).

Descrição de imagem:
Mostra indicação gráfica de erro dentro da IDE NetBeans, ao tentar adicionar

POO – MODIFICADOR FINAL

1 – Classe final

Apenas SUPONDO.. Que Aluno e Professor são as últimas classes da abstração que fizemos, elas são candidatas a se tornarem classes *final*.

MAS ...
Dizemos "SUPONDO", pois a abstração pode se estender e criar por exemplo:

Aluno Regular e Aluno Ouvinte.
Ou
Professor contratado e Professor voluntário.

Poderiam ser novas subclasses herdando atributos e métodos das superclasses.

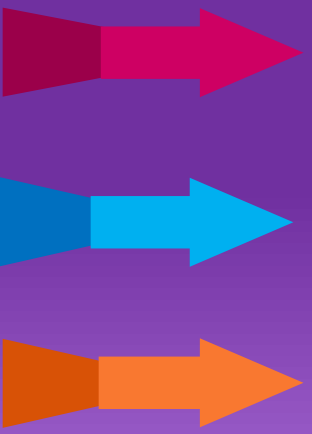
Neste caso, estas é que seriam candidatas a tornarem-se classes *final*!
Pois queremos interromper a cadeia de herança. Queremos que FINALIZE neste ponto!

POO – MODIFICADOR FINAL

2 – Atributo ou variável final

Um atributo ou variável com o modificador final, só pode ter seu valor atribuído uma única vez (CONSTANTE).

Veja um trecho de código da classe Pessoa.



```
public class Pessoa {

    // Atributos
    private int id;
    private String nome;
    private int idade;
    public final String instituicao = "Universidade X";
```

O valor da variável(atributo) instituicao não pode mais ser alterado.

POO – MODIFICADOR FINAL

3 – Método final

É um método que não pode ser sobrescrito (@Override) nas subclasses.
Usamos quando não queremos que as subclasses modifiquem o método.

Exemplo:
Se definíssemos o método mostrarDados() da Classe Pessoa como final, ele não poderia ser sobrescrito na classe Aluno. Mas certamente não é este comportamento que esperamos no nosso exemplo.

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

POO – MODIFICADOR STATIC

Com o static o atributo ou método pertence a Classe e não ao objeto.

Útil para compartilhar entre as instâncias de objeto, um determinado valor ou método.

Mais utilizados em:

- Métodos
- Atributos

POO – MODIFICADOR STATIC

1 – Métodos static

Métodos static podem ser chamados sem a necessidade de instanciar um objeto.

Exemplo:

```
// Dentro da Classe Pessoa
 public static boolean VerificarContratacao(){
        return true;
 }

// Dentro da Classe Principal
if(Pessoa.VerificarContratacao() == true){
System.out.println("Contratado!");
 }
```

POO – MODIFICADOR STATIC

2 – Atributos static

Os atributos static mantém o mesmo valor para todas as instâncias de Objeto.

Não precisa de objeto para acessar.

Exemplo:

```
// Dentro da Classe Pessoa

public class Pessoa {

    // Atributos
    private int id;
    private String nome;
    private int idade;
    public static double pi = 3.1415;

// Dentro da Classe Principal
System.out.println(Pessoa.pi);
```

POO – CLASSES ABSTRATAS (ABSTRACT)

São classes "modelo" para outras subclasses que dela herdam, no entanto, não podem ser instanciadas.

No nosso exemplo da aula anterior temos Pessoa como superclasse e Aluno e Professor como subclasses.

Se Pessoa fosse abstrata(*abstract*), dentro deste sistema, só seria possível instanciar objetos de Aluno e Professor. Já a classe Pessoa, por ser abstrata(*abstract*), não permitiria instanciação.

Exemplo:
Não quero que ninguém instancie objetos de Pessoa, só de Aluno e Professor.

```
// Dentro da Classe Pessoa

public abstract class Pessoa {

}
```

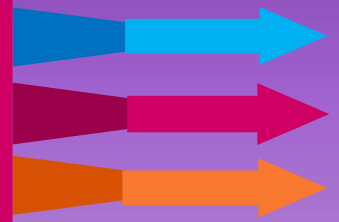
POO – MÉTODOS ABSTRATOS (ABSTRACT)

Métodos abstratos só podem ser escritos dentro de classe abstratas.

Este tipo particular de método não possui corpo, pois a sua implementação deve ser DIFERENTE em cada uma das subclasses.

Usamos quando queremos obrigar as subclasses desta classe abstrata, a sobrescrever (@Override) os métodos marcados como abstratos.

Exemplo:
Quero que as classes Aluno e Professor sejam OBRIGADAS a sobrescrever(@Override) um determinado método dito como obrigatório



```
// Dentro da Classe Pessoa
public abstract class Pessoa {

        // método abstrato que PRECISA ser sobrescrito(@Override) nas subclasses
        public abstract void metodoobrigatorio( ) ;

}
```

POO – INTERFACES (INTERFACE)

Interfaces são CLASSES TOTALMENTE ABSTRATAS;

Enquanto que em classes abstratas podemos ter atributos e métodos que não são abstratos, nas Interfaces TODOS os métodos são naturalmente abstratos e os atributos são sempre estáticos e constantes(*static final*).

Então, dizemos que Interfaces são MODELOS PUROS que não podem ser instanciados.

POO – INTERFACES (INTERFACE)

Exemplo:

```
package Model;

public interface Produto {

    // Atributos da interface são sempre CONSTANTES
    public static final String codigoinicial = "BR01SC";
    public static final String codigofinal = "FLN";

    // Métodos abstratos, mesmo SEM USAR ABSTRACT
    public String Incluircodigo_inicio(String codigoatual);
    public String Incluircodigo_fim(String codigoatual);
}
```

POO – INTERFACES (INTEFACE)

Como as Interfaces são totalmente abstratas, para usá-las, criamos novas classes que IMPLEMENTAM (*implements*) esta interface.

```
package Model;
public class Bebida implements Produto {

    // construtor
    public Bebida() {
    }

    // métodos com implementação OBRIGATÓRIA
    @Override
    public String Incluircodigo_inicio(String codigoatual) {
        return Produto.codigoinicial + codigoatual;
    }

    @Override
    public String Incluircodigo_fim(String codigoatual) {
        return codigoatual + Produto.codigofinal;
    }
}
```

POO – INTERFACES (INTERFACE)

Veja este pequeno teste na classe Principal:

```
import Model.Bebida;
public class Principal {
    public static void main(String[] args) {

        String codigoatual = "2568715";
        String novocodigo = codigoatual;

        Bebida objetobebida1 = new Bebida();

        System.out.println("TESTE 1 : Cod.: " + novocodigo); // output será: TESTE 1 : Cod.: 2568715

        novocodigo = objetobebida1.Incluircodigo_inicio(codigoatual);

        System.out.println("TESTE 2 : Cod.: " + novocodigo); // output será: TESTE 2 : Cod.: BR01SC2568715

        novocodigo = objetobebida1.Incluircodigo_fim(novocodigo);

        System.out.println("TESTE 3 : Cod.: " + novocodigo); // output será: TESTE 3 : Cod.: BR01SC2568715FLN

    }
}
```

EXERCÍCIO 32

Crie um novo projeto na IDE usando as mesmas classes Pessoa e Aluno da aula passada.
Desta vez a classe Pessoa deve ser abstrata e deve possuir:

- Ao menos um método abstrato.
- Ao menos uma constante estática (static final)

Depois na classe Aluno, implemente a sobrescrita(@Override) dos métodos abstratos de Pessoa.
Em Aluno USE a constante estática que criou.

EXERCÍCIO 33

Vamos construir Interfaces ?
Imagine que você precisa criar uma Interface Animal com atributos e métodos.

Depois as classe Ave e Cachorro devem IMPLEMENTAR esta Interface. Veja que Ave e Cachorro podem ter seus próprios atributos e métodos se você quiser.

Use a Classe principal para testar seu exercício.

FIM

ANEXO 1 – INTERFACE PRODUTO

```
public interface Produto {

    // Atributos da interface
    public static final String codigoinicial = "BR01SC";
    public static final String codigofinal = "FLN";

    // Métodos abstratos, mesmo sem USAR ABSTRACT
    public String Incluircodigo_inicio(String codigoatual);
    public String Incluircodigo_fim(String codigoatual);

}
```

ANEXO 2 – CLASSE BEBIDA

```
public class Bebida implements Produto {

    // construtor
    public Bebida() {
    }

    // métodos com implementação OBRIGATÓRIA
    @Override
    public String Incluircodigo_inicio(String codigoatual) {
        return Produto.codigoinicial + codigoatual;
    }

    @Override
    public String Incluircodigo_fim(String codigoatual) {
        return codigoatual + Produto.codigofinal;
    }

}
```

ANEXO 3 – TESTE NA CLASSE PRINCIPAL

```
import Model.Bebida;

public class Principal {

    public static void main(String[] args) {

        String codigoatual = "2568715";
        String novocodigo = codigoatual;

        // Instanciação de objeto da classe Bebida que implementa PRODUTO.
        Bebida objetobebida1 = new Bebida();

        System.out.println("TESTE 1 : Cod.: " + novocodigo);

        novocodigo = objetobebida1.Incluircodigo_inicio(codigoatual);

        System.out.println("TESTE 2 : Cod.: " + novocodigo);

        novocodigo = objetobebida1.Incluircodigo_fim(novocodigo);

        System.out.println("TESTE 3 : Cod.: " + novocodigo);

    }

}
```