



KEEP
CALM
AND
OVERRIDE
toString()



UML e Desenvolvimento Java

@Override, static, super, this e protected
Etapa 7

Gustavo de Miranda Gonçalves
gustavo.miranda@prof.infnet.edu.br

Sobrescrevendo métodos

@Override



A sobrescrita de métodos é a criação de um novo método, na subclasse, contendo a mesma assinatura e mesmo tipo de retorno do método sobrescrito já existente na superclasse.

O conceito de sobrescrita está intimamente ligado ao de herança, pois nos permite criar métodos mais específicos às necessidades de nossas subclasses.

@Override é importante para que o compilador verifique se de fato um método está sendo sobrescrito.



```
public class Empregado {  
    protected String nome;  
    protected String matricula;  
    protected double salario;  
  
    public double calcularSalario() {  
        return this.salario;  
    }  
}  
  
public class GerenteSetorial extends Empregado {  
    public double calcularSalario() {  
        return this.salario * 1.20;  
    }  
}
```



```
@Override  
public class GerenteSetorial extends Funcionario {  
    public double calcularSalario() {  
        return this.salario * 1.20;  
    }  
}
```

Static



A palavra reservada ***static*** muda o escopo de métodos e atributos onde é usada, pois faz com que métodos e atributos pertençam não à instância do objeto, mas à classe.

Um método **estático** pode ser chamado sem que exista uma instância da classe.



```
public final class Integer extends Number implements Comparable<Integer> {  
    public static Integer valueOf(String s, int radix) throws NumberFormatException {  
        return new Integer(parseInt(s, radix));  
    }  
}
```

```
Integer valor = Integer.valueOf("FF", 16);
```




Exemplo com **static**


```
public class UmaClasse {  
    static int valorTotal = 10;  
    public static int getValorTotal() {  
        return valorTotal;  
    }  
}
```

```
UmaClasse a1 = new UmaClasse();  
UmaClasse a2 = new UmaClasse();  
UmaClasse.valorTotal = 20;  
System.out.println(a1); // imprime 20  
System.out.println(a2); // imprime 20
```

super & this

Exemplo com **super** & **this**

```
@Override  
public class GerenteSetorial extends Funcionario {  
    public double calcularSalario() {  
        return super.calcularSalario() * 1.20;  
    }  
}
```



```
public class Empregado {  
    protected String nome;  
    protected String matricula;  
    protected double salario;  
  
    public Empregado(String nome, String matricula){  
        this.nome = nome;  
        this.matricula = matricula;  
    }  
    public double calcularSalario() {  
        return this.salario;  
    }  
}  
  
public class GerenteSetorial extends Empregado {  
  
    public GerenteSetorial(String nome, String matricula){  
        super(nome, matricula);  
    }  
  
    @Override  
    public double calcularSalario() {  
        return super.calcularSalario() * 1.20;  
    }  
}
```



protected

```
package br.edu.infnet.sisponto.domain;

public abstract class Pessoa {

    protected String nome;
    protected String cpf;
    protected String matricula;

    public String getNome() {
        return nome;
    }

    public String getCpf() {
        return cpf;
    }

    public String getMatricula() {
        return matricula;
    }

}
```

```
package br.edu.infnet.sisponto.domain;

public class Aluno extends Pessoa {

    private int[][] notasSemestrais;

    // Este método fica disponível para uma classe Professor do
    // mesmo pacote, mas não para classes de fora do pacote.
    protected consultarHistorico(){
        System.out.println(this.nome); // Acesso ao atributo nome
        só é possível pois nome é protected, e não private.
        // percorrer matriz de notas e imprimir...
    }

}
```