



Sobrescrevendo toString() hashCode()



**Certified
Developer**
The Ultimate Tech Degree

DigitalHouse >
Coding School



Temas

1

`toString()`

2

`hashCode()`

1 | .toString()



.toString()

Cada classe herda o método toString() da classe Object, ou seja, se não o implementarmos, os objetos que instanciamos terão este método.

Por exemplo, em nossa classe Empregado, com nome, arquivo, salário e descontos como atributos, o que aconteceria se usássemos o método toString():

```
public class Empregado{  
    private String nome;  
    private String arquivo;  
    protected double salario;  
    protected double descontos;  
  
}
```



.toString()

Ao usar o método, não teríamos erro, mas a informação exibida não seria algo compreensível:

```
public static void main(String[] args) {  
  
    Empregado novoEmpregado=new Empregado("Jose","1111");  
  
    System.out.println(novoEmpregado.toString());  
}
```

Esta é a saída que obtemos:

```
com.company.Empregado@1540e19d
```



.toString(): por que sobrescrever?



O método `.toString()` tenta representar o objeto com texto, mas como não o sobrescrevemos, obtemos esse tipo de saída.

A solução é substituir o método mostrando apenas as informações que queremos mostrar, e dando à string de saída o formato mais adequado.



Sobrescrevendo toString()

Lembre-se que é importante não alterar a assinatura do método, caso contrário estaremos sobrecarregando.

```
public class Empregado{  
    private String nome;  
    private String arquivo;  
    protected double salario;  
    protected double descontos;  
  
    @Override  
    public String toString(){  
        return "Nome: " + nome + "\n" +  
               "Arquivo: " + arquivo;  
    }  
}
```

Adicionamos o método toString() e retornamos a string com as informações do objeto que queremos retornar.



A saída que obtemos é aquela que programamos, neste caso, o nome e o arquivo.

```
public static void main(String[] args) {  
  
    Empregado novoEmpregado=new Empregado("Jose","1111");  
  
    System.out.println(novoEmpregado.toString());  
}
```

Esta é a saída:

```
Nome: Juan  
Arquivo: 1111
```



2 | .hashCode()

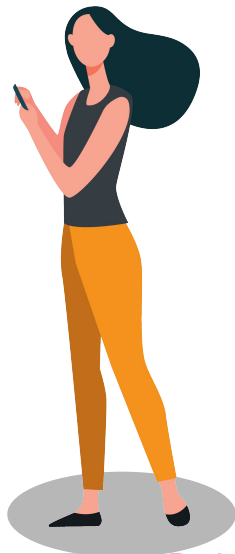


.hashCode()

Este é outro dos métodos herdados da classe Object. Quando este método é utilizado, ele retorna um número único que identifica o objeto, ou seja, se eu tiver dois objetos da mesma classe, o hashCode() geraria um número diferente para cada um e esse número me ajudará a identificá-lo.

A utilidade desse identificador veremos mais tarde, quando virmos estruturas de dados mais avançadas.

Agora, o que precisamos ter certeza é que esse código identificador é único para cada objeto.





.hashCode(): como sobrescrever?

```
public class Empregado{
    private String nome;
    private String arquivo;
    protected double salario;
    protected double descontos;

    @Override
    public int hashCode(){
        int hash=31;
        hash= hash* nome.hashCode();
        hash= hash* arquivo.hashCode();
        return hash;
    }
}
```

A classe
Empregados com
o .hashCode ()
sobrecarregado.
A seguir, veremos
como isso é feito.



Para gerar um número único, você trabalha com números primos. Pode ser qualquer número primo, neste caso 31 foi usado.

Como nome e arquivo são strings, ou seja, também são objetos, possuem seu próprio hashCode(). Multiplicamos todos os números e obtemos o hashCode do objeto.

Em uma string, o hashCode é gerado a partir dos caracteres. Por exemplo, o número do arquivo é sempre diferente.

```
@Override
public int hashCode(){
    int hash=31;
    hash= hash* nome.hashCode();
    hash= hash* arquivo.hashCode();
    return hash;}
```



Com a sobrecarga que fizemos, obtemos o valor mostrado:

```
public static void main(String[] args) {  
  
    Empregado novoEmpregado=new Empregado("Jose", "1111");  
  
    System.out.println(novoEmpregado.hashCode());  
}
```

Esta é a saída:

-1480218112



DigitalHouse>
Coding School