

# Programação Orientada a Objetos

Professor Isaac

Classe Object

# Classe Object

- Os desenvolvedores da linguagem julgaram que alguns métodos deveriam estar em todas as classes e por isso, em Java, existe a classe **Object** no pacote java.lang que agrupa esses métodos.
- Toda classe em Java herda direta ou indiretamente da classe **Object**. Este fato implica que todas as classes que escrevemos herdam todos os métodos de Object.

# Classe Object

Alguns dos métodos herdados de Object são:

- **toString()**
- **getClass()**
- **clone()**
- **equals(Object obj)**
- **hashCode()**

# Classe Object

- **toString()** – retorna uma string com a package, nome da classe e um hexadecimal que representa o objeto em questão. A recomendação da Oracle é que todas as classes sobrescrevam este método.
- **getClass()** – retorna informações do objeto atual, como o package e o nome da classe. Muito importante caso você tenha vários tipos de objeto, onde um herda do outro etc.
- **clone()** – retorna uma referência - ou cópia - de um objeto. A clonagem é arriscada porque pode causar efeitos indesejados. Portanto, como regra geral, clone( ) precisa ser sobreposto para que esses problemas sejam evitados.

# Classe Object

- **equals(Object obj)** – Retorna true se o objeto chamador for equivalente a objeto. É útil para saber se dois objetos apontam para o mesmo local na memória.
- **hashCode()** – Esse método retorna um inteiro único de cada objeto, muito usado em Collections. Como você já deve ter desconfiado, o hashCode tem uma ligação com o método equals(). É bastante útil quando trabalhamos com arquivos, para agilizar buscas e comparações.

# Exemplo da Classe Object

```
public class A {  
    public A() {  
        System.out.println("Construtor da classe A, super classe");  
    }  
  
    @Override  
    public Object clone() {  
        return this;  
    }  
}
```

```
public class B extends A {  
    public B() {  
        System.out.println("Construtor da classe B");  
    }  
}
```

# Exemplo da Classe Object

```
public class Principal {  
    public static void main(String[] args) {  
        A classe1 = new A();  
        System.out.println("=====");  
        B classe2 = new B();  
        System.out.println("=====");  
        System.out.println("getClass() da A: " + classe1.getClass());  
        System.out.println("getClass() da B: " + classe2.getClass());  
        System.out.println("=====");  
        B classe3 = (B) classe2.clone();  
        System.out.println("Objeto classe3 é clone da classe2? "+ classe3.equals(classe2));  
        System.out.println("=====");  
        System.out.println("toString da classe1: " + classe1.toString());  
        System.out.println("toString da classe2: " + classe2.toString());  
        System.out.println("=====");  
        System.out.println("Objeto classe1 é igual a classe2 ? " + classe2.equals(classe1));  
        System.out.println("Objeto classe1 é igual a classe1 ? " + classe1.equals(classe1));  
        System.out.println("Objeto classe2 é igual a classe3 ? " + classe2.equals(classe3));  
        System.out.println("=====");  
        System.out.println("Hash code da classe1: "+classe1.hashCode());  
        System.out.println("Hash code da classe2: "+classe2.hashCode());  
        System.out.println("Hash code da classe3: "+classe3.hashCode());  
        System.out.println("=====");  
    }  
}
```



# Exemplo da Classe Object

```
Saída - classeObject (run) x
run:
Construtor da classe A, super classe
=====
Construtor da classe A, super classe
Construtor da classe B
=====
getClass() da A: class classeobject.A
getClass() da B: class classeobject.B
=====
Objeto classe3 é clone da classe2? true
=====
toString da classe1: classeobject.A@15db9742
toString da classe2: classeobject.B@6d06d69c
=====
Objeto classe1 é igual a classe2 ? false
Objeto classe1 é igual a classe1 ? true
Objeto classe2 é igual a classe3 ? true
=====
Hash code da classe1: 366712642
Hash code da classe2: 1829164700
Hash code da classe3: 1829164700
=====
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

# Sobrecarga e Sobrescrita de Métodos

---

# Sobrecarga de Métodos

Ocorre quando os métodos tem o mesmo nome e se diferenciam pelos parâmetros:

- Tem quantidades diferentes de parâmetros.
- Tem a mesma quantidade de parâmetros, mas de tipos diferentes
- Tem a mesma quantidade de parâmetros de mesmos tipos, mas os tipos estão em ordem diferentes

# Exemplo de sobrecarga de métodos

```
public double soma() {  
    // comandos  
}  
  
public double soma(double num1, double num2) {  
    // comandos  
}  
  
public int soma(double num1, double num2, double num3) {  
    // comandos  
}  
  
public double soma(double [ ] numeros) {  
    // comandos  
}
```

# Exemplo de sobrecarga de métodos

```
public class Calculos {  
  
    public double soma() {  
        return 0;  
    }  
  
    public double soma(double num1, double num2) {  
        return num1 + num2;  
    }  
  
    public int soma(double num1, double num2, double num3) {  
        return (int) (num1 + num2 + num3);  
    }  
  
    public double soma(double[] numeros) {  
        double resultado = 0;  
        for( double i : numeros) {  
            resultado += i;  
        }  
        return resultado;  
    }  
}
```

# Exemplo de sobrecarga de métodos

```
6 public class Principal {
7     /**
8      * @param args the command line arguments
9      */
10    public static void main(String[] args) {
11        // TODO code application logic here
12        double[] numeros = {1,1,1,1,1,1,1};
13        Calculos calc = new Calculos();
14        System.out.println("Soma sem argumentos: " + calc.soma());
15        System.out.println("Soma com 2 argumentos: " + calc.soma(2,2));
16        System.out.println("Soma com 3 argumentos: " + calc.soma(4,4,4));
17        System.out.println("Soma com arrays: " + calc.soma(numeros));
18    }
19 }
```

Find:



Previous



Next



Select



run:

Soma sem argumentos: 0.0

Soma com 2 argumentos: 4.0

Soma com 3 argumentos: 12

Soma com arrays: 7.0






BUILD SUCCESSFUL (total time: 0 seconds)

# Outro exemplo de sobrecarga de métodos

```
6 public class Sobrerecarga{
7     public String tipos(int num1, int num2) {
8         return "int, int";
9     }
10
11     public String tipos(double num1, double num2) {
12         return "double, double";
13     }
14
15     public String tipos(String num1, String num2) {
16         return "String, String";
17     }
18
19     public String tipos(String num1, double num2) {
20         return "String, double";
21     }
22
23     public String tipos(double num1, String num2) {
24         return "double, String";
25     }
26 }
```

# Outro exemplo de sobrecarga de métodos

```
6 public class Principal {  
7     public static void main(String[] args) {  
8         // TODO code application logic here  
9         Sobrecarga objeto = new Sobrecarga();  
10        System.err.println("Metodo utilizado: " + objeto.tipos("exemplo", "aula"));  
11        System.err.println("Metodo utilizado: " + objeto.tipos(2.51, 7.82));  
12        System.err.println("Metodo utilizado: " + objeto.tipos("exemplo", 7.82));  
13        System.err.println("Metodo utilizado: " + objeto.tipos(2.51, "aula"));  
14        System.err.println("Metodo utilizado: " + objeto.tipos(10, 4));  
15    }  
16 }
```

Find:  | Previous Next Select |     

run:  
Metodo utilizado: String, String  
Metodo utilizado: double, double  
Metodo utilizado: String, double  
Metodo utilizado: double, String  
Metodo utilizado: int, int  
BUILD SUCCESSFUL (total time: 0 seconds)



# Argumentos em quantidade variável

Em algumas situações, podemos querer criar um método que use um número variável de argumentos.

Uma lista de argumentos de tamanho variável é especificada por três pontos (...).

# Exemplo de vários argumentos

```
12 public class VariosArgs {
13     public static void vaTeste(int ... v) {
14         System.out.println("Número de args: " + v.length);
15         System.out.println("Conteúdo: ");
16         for(int i=0; i < v.length; i++)
17             System.out.println(" arg " + i + ": " + v[i]);
18     }
19 }
```

# Exemplo de vários argumentos

```
3 public class Principal {  
4     public static void main(String[] args) {  
5         // Observe como vaTeste() pode ser chamado  
6         // com um número de argumentos variável.  
7         VariosArgs.vaTeste(10); // 1 argumento  
8         VariosArgs.vaTeste(1, 2, 3); // 3 argumentos  
9         VariosArgs.vaTeste(); // nenhum argumento  
10    }  
11 }
```

Saída - vaArgs (run) x



```
run:  
Número de args: 1  
Conteúdo:  
  arg 0: 10  
Número de args: 3  
Conteúdo:  
  arg 0: 1  
  arg 1: 2  
  arg 2: 3  
Número de args: 0  
Conteúdo:  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

# Exercício de vários argumentos

Crie a classe soma com métodos que realize a soma recebendo vários parâmetros, a classe soma deverá receber inteiros e retornar inteiros, e receber double e retornar double.

```
6      public class Calculos {  
7  [ ]      public int soma(int ... nums) {  
8              int sum = 0;  
9              for (int i : nums)  
10                  sum += i;  
11              return sum;  
12          }  
13      }
```

# Exemplo de vários argumentos

```
3 public class Principal {  
4     public static void main(String[] args) {  
5         Calculos calc = new Calculos();  
6         System.out.println("soma = " + calc.soma());  
7         System.out.println("soma = " + calc.soma(2));  
8         System.out.println("soma = " + calc.soma(2, 2));  
9         System.out.println("soma = " + calc.soma(2, 2, 2));  
10        System.out.println("soma = " + calc.soma(1, 2, 3, 4));  
11    }  
12 }
```

Saída - variosParametros (run) ×

run:

soma = 0

soma = 2

soma = 4

soma = 6

soma = 10

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

# Sobreposição de Métodos

Em uma hierarquia de classes, quando um método de uma subclasse tem o mesmo tipo de retorno e assinatura de um método de sua superclasse, diz-se que o método da subclasse sobrepõe o método da superclasse.

Quando um método **sobreposto** é chamado de dentro de uma **subclasse**, a referência é sempre à versão definida pela subclasse. A versão do **método** definida pela **superclasse** será **ocultada**.

# Exemplo de sobreposição

```
⊙ public class A {  
⊙   public void show() {  
10     System.out.println("Método da classe A");  
11   }  
12 }
```

```
8   public class B extends A{  
9  
10     @Override  
⊙   public void show() {  
12     System.out.println("Método da classe B");  
13   }  
14 }
```

# Exemplo de sobreposição

```
12 public class Principal {
13
14     public static void main(String[] args) {
15         A classA = new A();
16         B classB = new B();
17
18         classA.show();
19         classB.show();
20     }
21 }
```

Saída - Override (run) ×

run:  
Método da classe A  
Método da classe B  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)



# Exemplo de sobreposição

Exemplo criando uma terceira classe (C) que herda a classe A.

```
1 public class A {  
2     public void show() {  
10         System.out.println("Método da classe A");  
11     }  
12 }
```

```
8 public class B extends A{  
9  
10     @Override  
11     public void show() {  
12         System.out.println("Método da classe B");  
13     }  
14 }
```

```
5 public class C extends A{  
6  
7 }
```

# Exemplo de sobreposição

```
12 public class Principal {
13
14     public static void main(String[] args) {
15         A classA = new A();
16         B classB = new B();
17         C classC = new C();
18
19         classA.show();
20         classB.show();
21         classC.show();
22     }
23 }
```

>

Saída - Override (run) x

run:  
Método da classe A  
Método da classe B  
Método da classe A  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

# Arquivos

---

Escrita em arquivo texto

# Escrita em Arquivo

- Classes: FileWriter e PrintWriter

Precisa importar as classes do pacote java.io:

```
import java.io.FileWriter;
```

```
import java.io.PrintWriter;
```

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/FileWriter.html>

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/PrintWriter.html>

# Escrita em Arquivo

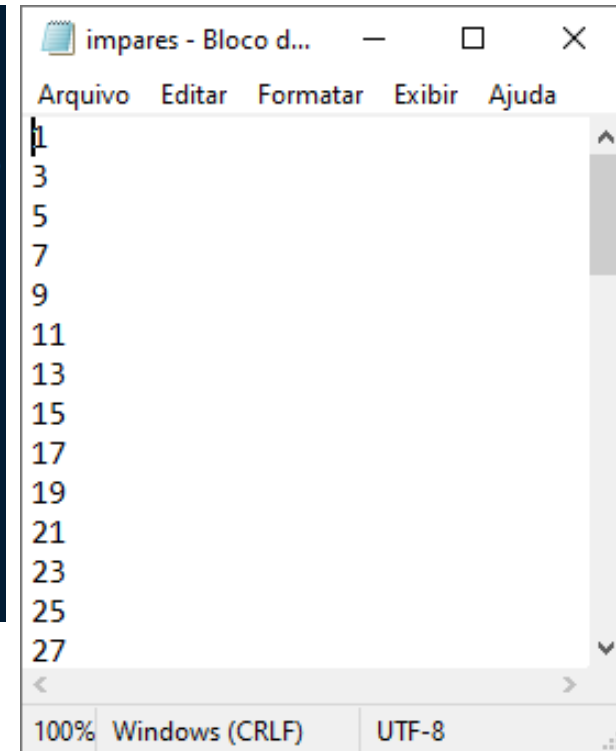
## *Exemplo: Arquivo com números ímpares*

```
1 // Escrita de arquivo é feita dentro do bloco try
2 try{
3     // o arquivo impares.txt é criado no modo escrita por meio do objeto arquivo
4     FileWriter arquivo = new FileWriter("impares.txt");
5     // PrintWriter recebe arquivo pelo construtor e cria um outro objeto, escritaArquivo
6     // que será responsável pelos métodos println() e printf()
7     PrintWriter escritaArquivo = new PrintWriter(arquivo);
8
9     for (i=1; i < 100; i++){
10         if (i % 2 != 0)
11             // escreve o número i no arquivo impares.txt
12             escritaArquivo.println(i);
13     }
14     // fecha o arquivo
15     arquivo.close();
16 }
17 catch (Exception e) {}
```

# Escrita em Arquivo


## *Exemplo: Arquivo com números ímpares*

```
1 // Escrita de arquivo é feita dentro do bloco try
2 try{
3     // o arquivo impares.txt é criado no modo escrita por meio do objeto arquivo
4     FileWriter arquivo = new FileWriter("impares.txt");
5     // PrintWriter recebe arquivo pelo construtor e cria um outro objeto, escritaArquivo
6     // que será responsável pelos métodos println() e printf()
7     PrintWriter escritaArquivo = new PrintWriter(arquivo);
8
9     for (i=1; i<100; i++){
10         if (i%2 != 0)
11             // escreve o número i no arquivo impares.txt
12             escritaArquivo.println(i);
13     }
14     // fecha o arquivo
15     arquivo.close();
16 }
17 catch (Exception e){}
```



# Escrita em Arquivo: Append

*Exemplo: Adicionando no arquivo sem deletar as informações que já estavam no arquivo*



```
public class Principal {  
    public static void main(String[] args) {  
        try{  
            FileWriter arquivo = new FileWriter("impares.txt", true);  
  
            PrintWriter escritaArquivo = new PrintWriter(arquivo);  
  
            for(int i=1; i<100; i++){  
                if(i%2!=0)  
                    escritaArquivo.printf("Número = %d\n", i);  
            }  
            arquivo.close();  
        }  
        catch(Exception e){}  
    }  
}
```



Leitura de um arquivo texto

# Leitura de Arquivo

- Classes: FileReader e BufferedReader

Precisa importar as classes do pacote java.io:

```
import java.io.FileReader;
```

```
import java.io.BufferedReader;
```

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/FileReader.html>

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/BufferedReader.html>

# Leitura de Arquivo

*Exemplo: Lendo o arquivo “impares.txt”*

```
try{
    FileReader arquivo = new FileReader("impares.txt");
    BufferedReader br = new BufferedReader(arquivo);

    ArrayList<String> impares = new ArrayList();
    String str;
    while((str = br.readLine()) != null)
        impares.add(str);
    System.out.println(impares);
}
catch(IOException e){}
```

# Leitura de Arquivo

```
8 public class Principal {
9     public static void main(String[] args) {
10         try{
11             FileReader arquivo = new FileReader("impares.txt");
12             BufferedReader br = new BufferedReader(arquivo);
13
14             ArrayList<String> impares = new ArrayList();
15             String str;
16             while((str = br.readLine()) != null)
17                 impares.add(str);
18             System.out.println(impares);
19         }
20         catch(IOException e){}
21     }
22 }
```

Saída - ArquivoLeitura (run) ×



run:

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```