Classes Wrappers

* java.lang.Integer
* java.lang.Double

Apresentação do grupo 02

## **Autores**

* [@Talita-Rocha](https://www.github.com/Talita-Rocha)
* [@KaikyScript](https://www.github.com/KaikyScript)
* [@rodrigoflu](https://www.github.com/rodrigoflu)
* [@leiasouza](https://www.github.com/leiasouzat)
* [@aurimartins](https://www.github.com/aurimartins)
* [@FERNANDAMSS](https://www.github.com/FERNANDAMSS)
* [@sheilansn](https://www.github.com/sheilansn)
* @claudineia

Em Java têm algumas palavras reservadas para a representação dos tipos de dados básicos que precisam ser manipulados para a construção de programas. Estes tipos de dados são conhecidos como tipos primitivos (Primitive Types).

São oito tipos primitivos: int, byte, short, long, float, double, boolean e char

Esses tipos não são considerados objetos, e portanto representam valores brutos. Eles são armazenados diretamente na pilha de memória. (Memory stack)

**Apresentando os tipos de dados primitivos:**

public class TipoDados {

public static void main(String[] args) {

byte idade = 123;

short ano = 2021;

int cep = 21070333;

// se começar com zero, talvez tenha que ser outro tipo

long cpf = 98765432109L;

// se começar com zero, talvez tenha que ser outro tipo

float pi = 3.14F;

double salario = 1275.33;

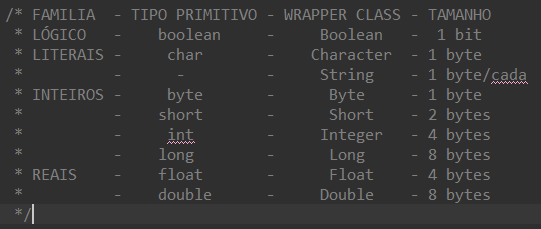
}

}

* **Classes Wrappers¹**

É uma classe empacotadora que encapsula um único valor de um tipo de dado primitivo.

Há uma classe wrapper correspondente a cada tipo primitivo em Java. Por exemplo, a classe Integer deve tratar um valor int.



Uma outra situação em que as classes wrapper são úteis é quando precisamos converter um objeto em um tipo de dado primitivo.

Por exemplo, digamos que em nossa aplicação, uma interface gráfica do usuário solicite que seja digitada a idade de uma pessoa em uma caixa de texto JTextField.

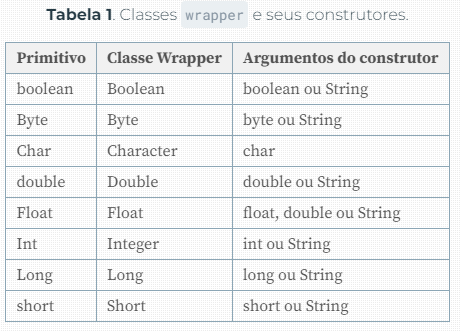
Contudo, sabemos que caixas de textos retornam seu conteúdo em uma String, o que requer uma conversão para int de maneira que possamos realizar cálculos com a idade na aplicação.

**Visão geral das classes wrapper**

Cada tipo primitivo em Java possui uma classe empacotadora correspondente. Por exemplo, a classe wrapper para int é Integer, para float é Float, e assim por diante, conforme pode ser visto na Tabela 1.

Observe que o nome do tipo primitivo é simplesmente o mesmo nome da classe wrapper em minúsculas, exceto o tipo char, que corresponde a Character, e int, cuja classe correspondente é Integer.

Na Tabela 1 podemos observar também quais os possíveis argumentos que podem ser passados ao construtor de cada classe.



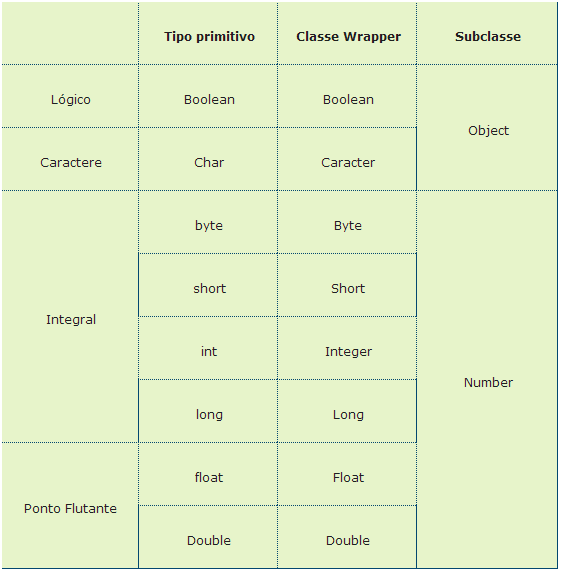
As classes wrapper são parte do pacote java.lang, o qual é importado por default em todos os programas Java. Dessas classes, as numéricas – Byte, Short, Integer, Long, Float e Double – estendem a classe Number. Number é uma classe abstrata e também é superclasse de

* **Pacote java.lang²**

Fornece classes que são fundamentais para o design da linguagem de programação Java. As classes mais importantes são *Object,* que é a raiz da hierarquia de classes, e *Class*, cujas instâncias representam classes em tempo de execução.

Frequentemente é necessário representar um valor de tipo primitivo como se fosse um objeto. As classes wrapper Boolean, Character, Integer, Long, Float e Double servem a esse propósito.

Um objeto do tipo Double, por exemplo, contém um campo cujo tipo é double, representando esse valor de forma que uma referência a ele possa ser armazenada em uma variável do tipo referência. Essas classes também fornecem vários métodos para conversão entre valores primitivos, além de oferecer suporte a métodos padrão como equals e hashCode. A Void class é uma classe não instanciável que contém uma referência a um Class objeto que representa o tipo void.



* **java.lang.Integer³**

**Integer class**

* java.lang.Object
  + java.lang.Number
    - java.lang.Integer

public final class Integer

extends Number

implementa Comparable < Integer >

A Integer Class envolve um valor do tipo primitivo int em um objeto. Um objeto do tipo Integer contém um único campo cujo tipo é int.

public class TiposDados {

public static void main(String[] args) {

int cep = 21070333;

// se começar com zero, talvez tenha que ser outro tipo

}

}

Além disso, essa classe fornece vários métodos para converter um int para a String e a String para a int, bem como outras constantes e métodos úteis ao lidar com um int.

* **java.lang.Double4**

**Double class**

* java.lang.Object
  + java.lang.Number
    - java.lang.Double

public final class Double

estende Number

implementa Comparable < Double >

A Double class envolve um valor do tipo primitivo double em um objeto. Um objeto do tipo Double contém um único campo cujo tipo é double.

public class TiposDados {

public static void main(String[] args) {

double salario = 1275.33;

}

}

Além disso, essa classe fornece vários métodos para converter a double em a String e a String em a double, bem como outras constantes e métodos úteis ao lidar com a double.

Apresentar alternativas de instanciação de objetos com construtor ou metódos instance

* **Construtor e Descrição**

**Integer (int value):** Constrói um objeto Integer recém-alocado que representa o valor especificado int.

**Integer(String s):** Constrói um objeto Integer recém-alocado que representa o valor int indicado pelo parâmetro String.

**Double(double value):** Constrói um objeto recém-alocado que representa o argumento Double primitivo .double

**Double(String s):** Constrói um objeto recém-alocado Double que representa o valor de ponto flutuante do tipo double representado pela string.

**Apresentar no mínimo 04 métodos mais utilizados destacando o seu contrato (tipo retorno + nome + parâmetros);**

**MÉTODOS ESTÁTICOS INTEGER:**

* **static int max(int a, int b)**

Retorna o maior de dois valores int, como se estivesse chamando Math.max.

* **static int min(int a, int b)**

Retorna o menor de dois valores int, como se estivesse chamando Math.min.

* **static int sum(int a, int b)**

Soma dois inteiros de acordo com o operador +.

* **static String toString(int i)**

Retorna um objeto String que representa o inteiro especificado.

* **static Integer valueOf(int i)**

Retorna uma instância Integer que representa o valor do int especificado.

* **static Integer valueOf(String s)**

Retorna um Integer objeto que contém o valor do especificado String.

**MÉTODOS ESTÁTICOS DOUBLE:**

* **static double max(double a, double b)**

Retorna o maior de dois valores double, como se estivesse chamando Math.max.

* **static double min(double a, double b)**

Retorna o menor de dois valores double, como se estivesse chamando Math.min.

* **static double sum(double a, double b)**

Soma dois valores doubles de acordo com o operador +.

* **static String toString(double d)**

Retorna uma representação de string do double argumento.

* **static Double valueOf(double d)**

Retorna uma instância Double que representa o valor double especificado.

* **static Double valueOf(String s)**

Retorna um objeto Double que contém o valor double representado pelo argumento string s.

**MÉTODOS DE INSTÂNCIA INTEGER**

* **float floatValue()**

Retorna o valor Integer como float, após uma conversão primitiva de ampliação.

* **int hashCode()**

Retorna um código hash para Integer.

* **int intValue()**

Retorna o valor Integer como um int.

* **long longValue()**

Retorna o valor Integer como long, após uma conversão primitiva de ampliação.

* **short shortValue()**

Retorna o valor Integer como short após uma conversão primitiva de restrição.

* **String toString()**

Retorna um objeto String representando o valor deste Integer .

**MÉTODOS DE INSTÂNCIA DOUBLE**

* **double doubleValue()**

Retorna o valor double deste objeto Double.

* **boolean equals(Object obj)**

Comparar um objeto com outro objeto especificado.

* **float floatValue()**

Retorna o valor Double como float, após uma conversão primitiva de restrição.

* **int intValue()**

Retorna o valor Double como int, após uma conversão primitiva de restrição.

* **long longValue()**

Retorna o valor Double como long, após uma conversão primitiva de restrição.

* **short shortValue()**

Retorna o valor Double como short, após uma conversão primitiva de restrição.

* **String toString()**

Retorna uma representação de string deste objeto Double.

**Apresentar se alguns dos métodos é sobrecarregado;**

**Sobrecarga (Overload) de métodos em Java8**

A sobrecarga, ou overload em inglês, permite a existência de vários métodos com o mesmo nome. Métodos sobrecarregados são escritos com o mesmo nome, mas com uma lista de argumentos diferente. São geralmente usados dentro de uma mesma classe.

As regras para usarmos métodos sobrecarregados são simples. O que você precisa lembrar para a usar sobrecarga de métodos é o seguinte:

1. Uma classe pode fazer a sobrecarga nos métodos que foram declarados dentro dela e também nos métodos herdados.
2. Métodos sobrecarregados devem possuir o mesmo nome.
3. Métodos sobrecarregados devem possuir listas de parâmetros diferentes.
4. Métodos sobrecarregados podem ter modificadores de acesso e tipos de retorno diferentes, mas eles não podem ser usados para diferenciar um método do outro.

**toString()**

A finalidade de toString() é dar alguma representação significativa de determinado objeto.

As classes wrapper contém a versão de uma instância do método toString(), não-static e sem argumentos. Este método tem como retorno uma String como valor do tipo primitivo encapsulado no objeto. Veja um exemplo:

1. Double double1 = new Double("5.17");
2. System.out.println(double1.toString());

As classes wrapper numéricas fornecem um método sobrecarregado e static toString() que utiliza o tipo numérico primitivo apropriado: Double.toString(), Long.toString() e assim por diante, retorna uma String.

1. String text = Double.toString(1.95);

Sendo assim, as classes Long e Integer fornecem um terceiro método toString(). Esse é static, seu primeiro argumento é do tipo primitivo e o segundo é uma base. O argumento da base informa o método para pegar o primeiro argumento, que por padrão está na base 10, e convertê-lo para a base fornecida, retornando, como resultado, uma String. Veja a seguir:

1. String exemplo = "resulta em "+ Long.toString(254, 16);//resulta em fe

**Realizar uma demonstração adaptando o uso dos métodos (mínimo 04) em situações do cotidiano (use a imaginação);**

**COLOCAR NOSSO EXEMPLOS AQUI (CÓDIGOS)**

**Referências:**

1. Devmedia - conceito classe wraper <https://www.devmedia.com.br/classes-wrapper-explorando-as-classes-que-empacotam-os-tipos-primitivos/24136>
2. Java lang <https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html>
3. Java.lang.Integer <https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html>
4. Java.lang.Double <https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html>

# Curso de Java 57: Wrappers: classes de tipos primitivos - Loiane Groner

<https://www.youtube.com/watch?v=U0dcVFvJIhk>

# Curso Java Completo - Aula 78: Classes Wrapper pt 01 - DEV DOJO

<https://www.youtube.com/watch?v=MBm7iyYt6NQ>

# Curso Java Completo - Aula 79: Classes Wrapper pt 02 - DEV DOJO <https://www.youtube.com/watch?v=s5AEuHhR2PY>

1. sobrecarga-overload-de-metodos-em-java <http://high5devs.com/2015/03/sobrecarga-overload-de-metodos-em-java/>
2. Visão geral das classes wrapper <https://www.devmedia.com.br/classes-wrapper-explorando-as-classes-que-empacotam-os-tipos-primitivos/24136>

**ANEXO:**

**Link da atividade em grupo:**

https://github.com/digytal-code/java-school/tree/main/exercicios/01%20-%20power-classes

## **Power Classes**

O nosso desafio é explorar as principais classes da linguagem Java.

#### **Requisitos**

1. Realizar uma breve descrição da classe em questão;
2. Apresentar alternativas de instanciação de objetos com construtor ou metódos instance ;
3. Apresentar no mínimo 04 métodos mais utilizados destacando o seu contrato (tipo retorno + nome + parâmetros);
4. Apresentar se alguns dos métodos é sobrecarregado;
5. Realizar uma demonstração adaptando o uso dos métodos (mínimo 04) em situações do cotidiando (use a imaginação);

#### **Classes por Grupo**

##### **Grupo 01**

* java.lang.String
* java.util.StringJoiner
* java.lang.StringBuilder

##### **Grupo 02**

* Descrição do que são classes Wrappers
* java.lang.Integer
* java.lang.Double

##### **Grupo 03**

* java.util.Calendar
* java.time.LocalDate
* java.time.LocalDateTime

##### **Grupo 04**

* java.lang.String.format()
* java.time.format.DateTimeFormatter
* java.text.DecimalFormat

##### **Grupo 05**

* java.io.File
* java.nio.file.Path
* java.nio.file.Paths

###### **05 - Links**

[Dev Dojo](https://www.youtube.com/watch?v=RX3kPWkuZMQ)

##### **Grupo 06**

* java.lang.Object
* java.lang.Math
* java.math.BigDecimal

#### **Dicas**

* Use a documentação oficial da Oracle + outras referências e exemplos na internet
* Atuar em conjunto para resolver o item 1 de requisitos e depois tentar distribuir entre os integrantes do grupo
* Testes, testes e mais testes, pensem previamente no resultado e validem no console
* O Grupo 04 poderá/deverá interagir com os demais grupos
* A apresentação será realizada em 10 minutos
* Definam antecipadamente a ordem dos apresentadores
* Usem o próprio README.md com os recursos de Markdown.
* Preferencialmente subam o conteúdo (apresentação + exemplos) no repositório da School