CLASSE E OBJETO

Alexandre Mendonça Fava
alexandre.fava@udesc.br

Universidade do Estado de Santa Catarina – UDESC
Programa de Pós-graduação em Computação Aplicada – PPGCA

Roteiro

- Definição
  - Classe
  - Objeto
- Construtor
- Classes Invólucro
- `String`
- `Scanner`
- Resumo
- Referências

Definição

Classe

SCHILDT, Herbert. **Java para iniciantes.**

Uma classe é um modelo que define a forma de um objeto.

Objeto

SCHILDT, Herbert. **Java para iniciantes.**

Os objetos são *instâncias* de uma classe.

TAVARES, Roberto; SILVA, Fábio. **Introdução à Programação para Engenharia.**

Na orientação a objetos, uma **classe** é um "esqueleto", uma estrutura que engloba dados e procedimentos.

Uma classe é *instanciada*, gerando **objetos**.

Definição

(dicionário)

Classe

Conjunto de **atributos** ou **comportamentos** comuns a um grupo de seres ou coisas

Objeto

Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas **características** e **atividades**

Definição

(dicionário)

Classificação

CARRO

Atributo:	Comportamento:
Cor	Ligar
Ano	Desligar
Placa	Acelerar

Coisa

Definição

(dicionário)

ADJETIVOS	VERBOS
Cor	Ligar
Ano	Desligar
Placa	Acelerar
Rádio	Frear
Marca	Direcionar
Chassi	Abastecer
Tanque	Calibrar
Câmbio	**Voar**
Velocidade	**Nadar**
Consumo	**Comer**
Gênero	**Atirar**
Munição	**Teletransportar**

Coisa

SCHILDT, Herbert. **Java para iniciantes.**

Uma classe bem projetada deve agrupar informações logicamente conectadas. A inserção de informações não relacionadas na mesma classe desestruturará rapidamente seu código!

Definição

(dicionário)

Atributo: | Comportamento:

Co
An
Pla
Rád
Mar
Cha
Tan
Câm
Veloci
Consu
Gên
Mun

https://pt.wikipedia.org/wiki/Abstração_(ciência_da_computação)

Abstração (ciência da computação)

Artigo Discussão

Origem: Wikipédia, a enciclopédia livre.

Em ciência da computação, a **abstração** é a habilidade de concentrar nos aspectos essenciais de um contexto, ignorando características menos importantes ou acidentais. Em modelagem ou programação orientada a objetos, uma classe é uma abstração de entidades existentes no contexto de um software.

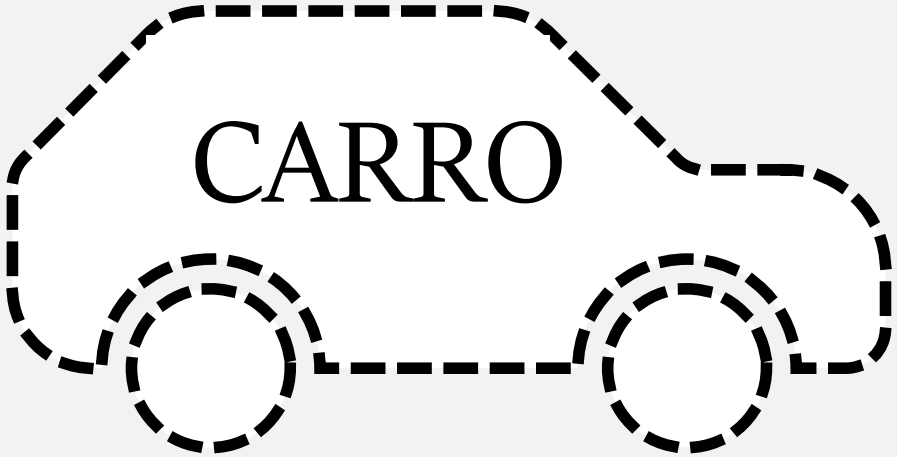
iniciantes.

agrupar
tadas.
relacionadas
rapidamente

Definição

(dicionário)

Classificação



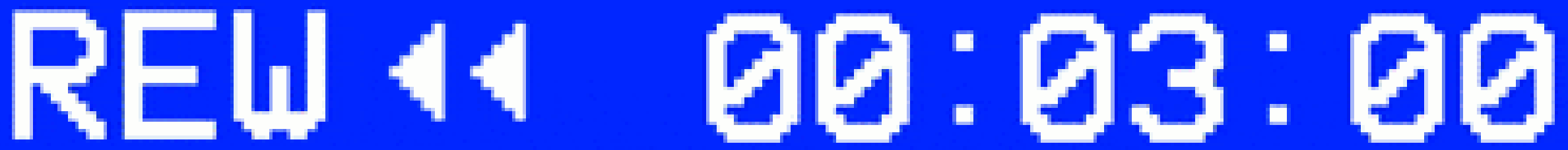
Atributo:	Comportamento:
Cor	Acelerar
Tanque	Frear
Velocidade	Direcionar

Coisa

SCHILDT, Herbert. **Java para iniciantes.**

Uma classe bem projetada deve agrupar informações logicamente conectadas.

A inserção de informações não relacionadas na mesma classe desestruturará rapidamente seu código!


REW
00:03:00

Paradigma: Orientado a Objetos

Primeira Aula

Definição

(programação)

Classe

É a declaração de um novo tipo de dado que **abstrai** as **características** e **comportamentos** importantes de um determinado conceito em uma única estrutura

Objeto

É a confecção de um determinado conceito **abstrato** baseado na união dos seus principais **atributos** e **métodos** em uma única variável

Definição

(programação)

Classe

Objeto

Uma **classe** é o componente que modela computacionalmente **objetos** e/ou conceitos do mundo material

Definição

(programação)

Uma **classe** é o componente que modela computacionalmente **objetos** e/ou conceitos do mundo material

Classes não se transformam em objetos

Objetos são construídos a partir de classes

O processo de criação de objetos
é conhecido como **instanciar**

Definição

(programação)

Classes não se transformam em objetos

Objetos são construídos a partir de classes

O processo de criação de objetos
é conhecido como **instanciar**

SCHILDT, Herbert. **Java para iniciantes.**

É importante deixar uma coisa bem clara: uma classe é uma abstração lógica. Só quando um objeto dessa classe é criado é que existe uma representação física dela na memória.

Classe

(programação)

Código (Main.java)

```
1  import java.lang.System;
2
3  public class Main{
4      public static void main(String[] args){
5          System.out.print("Olá, Mundo!");
6      }
7  }
8
9
10
11
12
13
14
```

Classe

(programação)

Código (Main.java)

```
1  public class Main{
2    public static void main(String[] args){
3      char veiculo1_cor         = 'A';
4      int  veiculo1_tanque      = 100;
5      int  veiculo1_velocidade = 0;
6
7      veiculo1_velocidade++; //acelerando
8      veiculo1_velocidade--; //freando
9
10     System.out.println(veiculo1_velocidade);
11
12
13
14 }
```

ISSO SÃO VARIÁVEIS E NÃO UM OBJETO

Classe

(programação)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     veiculo1.cor         = 'A';
5     veiculo1.tanque      = 100;
6     veiculo1.velocidade = 0;
7
8     veiculo1.acelerar();
9
10    System.out.println(veiculo1.velocidade);
11
12   }
13 }
14
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7       velocidade++;
8   }
9   void freiar(){
10      velocidade--;
11  }
12
13 }
14
```

Classe

(programação)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     veiculo1.cor         = 'A';
5     veiculo1.tanque      = 100;
6     veiculo1.velocidade = 0;
7
8     veiculo1.acelerar();
9
10    System.out.println(veiculo1.velocidade);
11
12  }
13 }
14
```

Ei, isso está me parecendo uma: STRUCT.

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7       velocidade++;
8   }
9   void freiar(){
10      velocidade--;
11  }
12
13 }
14
```

ATRIBUTOS

MÉTODOS

Classe

(programação)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     veiculo1.cor         = 'A';
5     veiculo1.tanque      = 100;
6     veiculo1.velocidade = 0;
7
8     veiculo1.acelerar();
9
10
11
12
13
14
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7       velocidade++;
8   }
9   void freiar(){
```

ATRIBUTOS

MÉTODO

SCHILDT, Herbert. **Java para iniciantes.**

Uma abordagem como essa nunca seria usada em um código Java escrito profissionalmente. Além de ser propensa a erros (você pode se esquecer de configurar um dos campos), há uma maneira melhor de executar essa tarefa: o construtor.

Construtor

TAVARES, Roberto; SILVA, Fábio. **Introdução à Programação para Engenharia.**

Em praticamente todas as linguagens que adotam o paradigma da orientação a objeto é dada a possibilidade de se criar um método especial, que é executado imediatamente após a criação do objeto. Esse método é chamado *Construtor*.

O que é um construtor?

Construtor é um método especial responsável por inicializar os atributos de uma determinar classe

Construtor

Se você **não** criar um Construtor, Java fornecerá automaticamente um construtor padrão que inicializará todos atributos com seus valores padrão

Não possui retorno
(nem mesmo `void`)

Construtor é um método especial responsável por inicializar os atributos de uma determinar classe

Sua chamada ocorre de forma implícita
(através do operador `new`)

SCHILDT, Herbert. **Java para iniciantes.**

Normalmente, usamos um construtor para fornecer valores iniciais para as variáveis de instância definidas pela classe ou para executar algum outro procedimento de inicialização necessário à criação de um objeto totalmente formado.

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     veiculo1.cor         = 'A';
5     veiculo1.tanque      = 100;
6     veiculo1.velocidade = 0;
7
8     veiculo1.acelerar();
9
10    System.out.println(veiculo1.velocidade);
11  }
12 }
13
14
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7       velocidade++;
8   }
9   void freiar(){
10      velocidade--;
11  }
12
13 }
14
```

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1        = new Carro();
4     veiculo1.acelerar();
5
6     System.out.println(veiculo1.cor);
7   }
8  }
9
10
11
12
13
14
```

Deve possuir o mesmo nome da classe

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(){
7    cor = 'B';
8    tanque = 100;
9    velocidade = 0;
10  }
11
12  void acelerar(){
13      velocidade++;
14  }
```

CONSTRUTOR

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1        = new Carro();
4     veiculo1.acelerar();
5
6     System.out.println(veiculo1.cor);
7   }
8  }
9
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(){
7    cor = 'B';
8    tanque = 100;
9    velocidade = 0;
```

CONSTRUT

SCHILDT, Herbert. Java para iniciantes.

Se uma classe não definir seu próprio construtor, **new** usará o construtor padrão fornecido por Java. Logo, **new** pode ser usado para criar um objeto de qualquer tipo de classe. O operador **new** retorna uma referência ao objeto recém criado.

Construtor

Código (Main.java)

```java
public class Main{
 public static void main(String[] args){
   Carro veiculo1        = new Carro();
   veiculo1.acelerar();

   System.out.println(veiculo1.cor);
 }
}
```

Retorna uma referência ao objeto recém criado? Isso está me parecendo um: MALLOC.

Código (Carro.java)

```java
public class Carro{
 char cor;
 int tanque;
 int velocidade;

 Carro(){
  cor = 'B';
  tanque = 100;
  velocidade = 0;
 }

 void acelerar(){
     velocidade++;
 }
```

CONSTRUTOR

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro('A');
4     veiculo1.acelerar();
5
6     System.out.println(veiculo1.cor);
7   }
8  }
9
10
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(char colorir){
7     cor = colorir;
8     tanque = 100;
9     velocidade = 0;
10  }
```

Método Construtor Parametrizado

SCHILDT, Herbert. Java para iniciantes.

Os parâmetros são adicionados a um construtor da mesma forma que são adicionados a um método: apenas declare-os dentro de parênteses após o nome do construtor.

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro('A');
4     veiculo1.acelerar();
5
6     System.out.println(veiculo1.cor);
7   }
8  }
9
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(char colorir){
7     cor = colorir;
8     tanque = 100;
9     velocidade = 0;
```

Método Construtor Parametrizado

SCHILDT, Herbert. Java para iniciantes.

Já que a memória é finita, é possível que **new** não consiga alocar memória para um objeto por não existir memória suficiente. Se isso ocorrer, haverá uma exceção de tempo de execução.

Exceção será visto em aulas futuras

Construtor

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro('A');
4     veiculo1.acelerar();
5
6     System.out.println(veiculo1.cor);
7   }
8  }
9
```

É fácil, é só liberar mais memória!

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(char colorir){
7     cor = colorir;
8     tanque = 100;
9     velocidade = 0;
```

Método Construtor Parametrizado

SCHILDT, Herbert. Java para iniciantes.

Já que a memória é finita, é possível que **new** não consiga alocar memória para um objeto por não existir memória suficiente. Se isso ocorrer, haverá uma exceção de tempo de execução.

Exceção será visto em aulas futuras

Destrutor

CENÁRIO HIPOTÉTICO

Caso um programa tenha alocado muitos objetos na memória

Basta **DESTRUIR** objetos que não são mais usados para criar novos

JAVA NÃO POSSUI MÉTODO DESTRUTOR

Olá, eu sou o **Coletor de Lixo**.
(Garbage Collector)

CENÁRIO HIPOTÉTICO

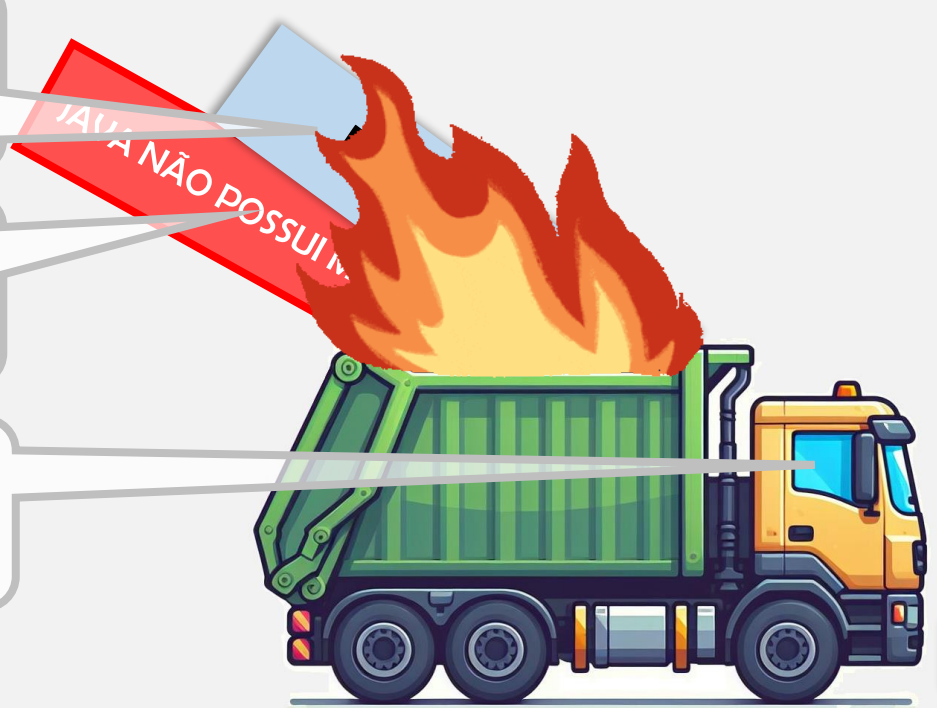
Caso um programa tenha alocado muitos objetos na memória

Basta **DESTRUIR** objetos que não são mais usados para criar novos

Estou te chamando **`finalize`**

`finalize`, **`finalize`**, **`finalize`**

Já chamaram o **`finalize`**?
Então: **FOGO!!!**



finalize

Alguém me chamou?

SCHILDT, Herbert. Java para iniciantes.

A coleta de lixo só ocorre esporadicamente durante a execução do programa. Ela não ocorrerá só porque existem um ou mais objetos que não são mais usados.

SCHILDT, Herbert. Java para iniciantes.

O sistema de coleta de lixo de Java reclama objetos automaticamente – ocorrendo de maneira transparente em segundo plano, sem nenhuma intervenção do programador. Funciona assim: quando não existe nenhuma referência a um objeto, ele não é mais considerado necessário e a memória ocupada é liberada. Essa memória reciclada pode então ser usada para uma alocação subsequente.

AVISO: Um objeto precisa 'perder' todas as suas referências de memória dentro do programa para ser removido da memória (caso isso não seja feito, a memória estará sendo desperdiçada e isso poderá gerar falhas de segurança)

finalize

DEITEL, Paul. Java: como programar.

A JVM executa **coleta de lixo** automática para recuperar a *memória* ocupada por objetos que não são mais usados. Quando *não* há *mais referências* a um objeto, o objeto é *marcado* para coleta de lixo. A coleta normalmente ocorre quando a JVM executa o **coletor de lixo**, o que pode não acontecer por um tempo, ou até mesmo absolutamente antes de um programa terminar.

finalize

É um método chamado imediatamente antes da destruição final de um objeto pelo coletor de lixo

Condição 1: Há objetos a serem reciclados?

Condição 2: Há a necessidade de reciclados?

VOCÊ COMO PROGRAMADOR APENAS DECLARA O MÉTODO, QUEM REALIZA A CHAMDA DO MÉTODO É A JVM*

*se a JVM estiver afim de chamar o método

SE AMBAS AS CONDIÇÕES NÃO FOREM SATISFEITAS
A JVM FICA SEM FAZER NADA

finalize

SCHILDT, Herbert. **Java para iniciantes.**

P: **C++ define elementos chamados *destruidores*, que são executados automaticamente quando um objeto é destruído. O método** finalize( ) **é semelhante a um destruidor?**

R:

this

(um código sem this) 

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = new Carro();
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
7
8     veiculo1.acelerar();
9     veiculo1.acelerar();
10
11    System.out.println(veiculo2.velocidade);
```

Saída

```
2
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7       velocidade++;
8   }
9   void freiar(){
10      velocidade--;
11  }
```

As referências paras as variáveis são compartilhadas entre os objetos

this

(um código sem this) 

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = new Carro();
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
```

DEITEL, Paul. Java: como programar.

Cada objeto pode acessar uma *referência a si próprio* com a palavra-chave **this** (às vezes chamada de **referência this**). Quando um método de instância é chamado para um objeto particular, o corpo do método utiliza *implicitamente* a palavra-chave this para referenciar as variáveis de instância do objeto e outros métodos.

this

(um código com this) 

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = new Carro();
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
7
8     veiculo1.acelerar();
9     veiculo1.acelerar();
10
11    System.out.println(veiculo2.velocidade);
12  }
13 }
14
```

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7     this.velocidade++;
8   }
9   void freiar(){
10    this.velocidade--;
11  }
12
13 }
14
```

this

(um código com this)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = new Carro();
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
7
8     veiculo1.acelerar();
9     veiculo1.acelerar();
10
11    System.out.println(veiculo2.velocidade);
12  }
13
14
```

Saída

```
0
```

Ah, nem precisava fazer isso.

Bastava passar como referência o ponteiro do objeto.

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7    veiculo1 .velocidade++;
8   }
9   void freiar(){
10    this.velocidade--;
11  }
12
13 }
14
```

this

(um código com this)

Bloco de notas

Lembrar a linguagem C que JAVA não tem ponteiro.

Obs: Graças a referência **this** é possível reaproveitar nomes de variáveis sem ocasionar o problema de **variável oculta**.

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(char cor){
7    cor = cor;
8    tanque = 100;
9    velocidade = 0;
10  }
```

Ah, nem precisava fazer isso.

Bastava passar como referência o ponteiro do objeto.

this

(um código com this)

Código (Main.java)

Código (Carro.java)

Bloco de notas

Lembrar a linguagem C que JAVA não tem ponteiro.

Obs: Graças a referência **this** é possível reaproveitar nomes de variáveis sem ocasionar o problema de **variável não oculta**.

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   Carro(char cor){
7    this.cor = cor;
8    tanque = 100;
9    velocidade = 0;
10  }
```

Ah, nem precisava fazer isso.

Bastava passar como referência o ponteiro do objeto.

this

(exercício)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = veiculo1;
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
7
8     veiculo1.acelerar();
9     veiculo1.acelerar();
10
11    System.out.println(veiculo2.velocidade);
12  }
13 }
14
```

Saída

```
2
```

Ele não é um new carro (ele é o mesmo carro)

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7     this.velocidade++;
8   }
9   void freiar(){
10    this.velocidade--;
11  }
12
13 }
14
```

this

(exercício)

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Carro veiculo1       = new Carro();
4     Carro veiculo2       = new Carro();
5     veiculo1.velocidade = 0;
6     veiculo2.velocidade = 0;
7
8     veiculo1.acelerar();
9     veiculo1.acelerar();
10
11    System.out.println(veiculo2.velocidade);
12  }
13 }
14
```

Saída

```
0
```

Agora ele é de fato um new carro (outro carro)

Código (Carro.java)

```
1  public class Carro{
2   char cor;
3   int tanque;
4   int velocidade;
5
6   void acelerar(){
7     this.velocidade++;
8   }
9   void freiar(){
10    this.velocidade--;
11  }
12
13 }
14
```

Classes em Código

SCHILDT, Herbert. **Java para iniciantes.**

Uma definição **class** cria um novo tipo de dado.

Classes Invólucro

PRIMITIVOS
boolean
int
short
byte
long
char
float
double

ORIENTADOS A OBJETOS
Boolean
Integer
Short
Byte
Long
Character
Float
Double

Classes Invólucro

Por causa desses tipos primitivos JAVA não é considerado totalmente orientado a objetos

boolean
int
short
byte
long
char
float
double

Essas classes encapsulam os tipos primitivos, permitindo usar eles como objetos

Boolean
Integer
Short
Byte
Long
Character
Float
Double

Classes Invólucro

Código (Main.java)

```java
public class Main{
 public static void main(String[] args){
   int     var1 = (int)    10;
   byte    var2 = (byte)   10;
   short   var3 = (short)  10;
   float   var4 = (float)  10.0;
   double  var5 = (double) 10.0;
   long    var6 = (long)   10;
   char    var7 = (char)   'A';
   boolean var8 = (boolean) true;
 }
}
```

Código (Main.java)

@depreciado

```java
public
 public static void
                    ng[] args){
   Integer   var1 = new Integer(10);
   Byte      var2 = new Byte((byte)10);
   Short     var3 = new Short((short)10);
   Float     var4 = new Float(10f);
   Double    var5 = new Double(10d);
   Long      var6 = new Long(10l);
   Character var7 = new Character('A');
   Boolean   var8 = new Boolean(true);
   }
}
/*Essa era a forma antiga como era
 *feita a instanciação de objetos de
 *tipo primitivo. Ainda está presente
 *na linguagem mas está obsoleto e
 *"marcado" para remoção futuramente
 */
```

Classes Invólucro

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     int     var1 = 10;
4     byte    var2 = 10;
5     short   var3 = 10;
6     float   var4 = 10.0F;
7     double  var5 = 10.0D;
8     long    var6 = 10L;
9     char    var7 = 'A';
10    boolean var8 = true;
11   }
12 }
13
14
15
16
17
18
```

Essas classes são utilizadas para fornecer funcionalidades adicionais aos tipos primitivos

Cód... (Main.java)

@autoboxing

```
1  public cl...
2   public static void ma...(...args){
3     Integer   var1 = 10;
4     Byte      var2 = 10;
5     Short     var3 = 10;
6     Float     var4 = 10f;
7     Double    var5 = 10d;
8     Long      var6 = 10l;
9     Character var7 = 'A';
10    Boolean   var8 = true;
11   }
12 }
13 /*Essa é a forma atual como o
14  *compilador Java realiza a conversão
15  *automática entre tipos primitivos e
16  *suas respectivas classe, isso sem
17  *mais usar o construtor 'new'
18  */
```

Classes Invólucro

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     int      var1 = 10;
4     int      var2 = 10;
5
6     System.out.println(var1 == var2);
7     }
8  }
9
10
11
12
13
14
```

Saída

```
true
```

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Integer   var1 = 10;
4     Integer   var2 = 10;
5
6     System.out.println(var1 == var2);
7
8     }
9  }
10
11
12
13
14
```

Saída

```
false
```

Classes Invólucro

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     int      var1 = 10;
4     int      var2 = 10;
5
6     System.out.println(var1 == var2);
7     }
8  }
9
10
11
12
13
14
```

Saída

```
true
```

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     Integer   var1 = 10;
4     Integer   var2 = 10;
5
6     System.out.println(var1.equals(var2));
7
8     }
9  }
10
11
12
13
14
```

Saída

```
true
```

Classes Invólucro

Código (Main.java)

```java
public class Main{
 public static void main(String[] args){
   int      var1 = 10;
   int      var2 = 10;

   System.out.println(var1 == var2);
   }
}
```

PRIMITIVOS

- Executam mais rápido
- Consomem menos memória

Código (Main.java)

```java
public class Main{
 public static void main(String[] args){
   Integer   var1 = 10;
   Integer   var2 = 10;

   System.out.println(var1.equals(var2));

   }
}
```

ORIENTADOS A OBJETOS

- Possuem funcionalidades adicionais
- São usados em coleções
- São imutáveis

Classes Invólucro

São imutáveis

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3     byte      var1 = 10;
4     var1 = 11;
5     var1 = 12;
6     var1 = 13;
7     }
8  }
```

Endereços	Memória
0x00F	13
0x01F	
0x02F	
0x03F	

A JVM está guardando 13 no endereço 0x00F

Código (Main.java)

```
1  public class Main{
2   public static void main(String[] args){
3    Byte    var1 = 10;
4    var1 = 11;
5    var1 = 12;
6    var1 = 13;
7    }
8  }
```

Endereços	Memória
0x00F	12
0x01F	10
0x02F	11
0x03F	13

A JVM está colocando 13 no endereço 0x03F e chamando de var1

Classes Invólucro: `String`

https://docs.oracle.com/javase/tutorial/java/data/strings.html

Strings, que são amplamente utilizadas na programação Java, são uma sequência de caracteres. Na linguagem de programação Java, strings são objetos.

A plataforma Java fornece a classe `String` para criar e manipular cadeias de caracteres.

Classes Invólucro: `String`

https://docs.oracle.com/javase/tutorial/java/data/strings.html

Strings, que são amplamente utilizadas na programação Java, são uma sequência de caracteres. Na linguagem de programação Java, strings são objetos.

A plataforma Java fornece a classe `String` para criar e manipular cadeias de caracteres.

Criando cadeias de caracteres

A maneira mais direta de criar uma cadeia de caracteres é escrever:

```
String greeting = "Hello world!";
```

Classes Invólucro: String

Código (Main.java)

```
1  import java.lang.System;
2
3  public class Main{
4      public static void main(String[] args){
5          String   var1 = "10";
6          char     var2 = '1';
7          char[]   var3 = {'1', '0'};
8      }
9  }
```

String

Um tipo `string` é inicializado com uma sequência de caracteres entre aspas duplas

Um tipo `char` é inicializado com um único caráter entre aspas simples

Código (Main.java)

```
1  import java.lang.System;
2
3  public class Main{
4      public static void main(String[] args){
5          String   var1 = "10";
6          char     var2 = '1';
7           char[]   var3 = {'1', '0'};
```

https://docs.oracle.com/javase/tutorial/java/data/strings.html

Nota: A `String` classe é imutável, portanto, uma vez criada, um `String` objeto não pode ser alterado.

String

DEITEL, Paul. **Java: como programar.**

```
3
4  public class StringConstructors
5  {
6     public static void main(String[] args)
7     {
8        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y'};
9        String s = new String("hello");
10
11       // utiliza os construtores String
12       String s1 = new String();
13       String s2 = new String(s);
14       String s3 = new String(charArray);
15       String s4 = new String(charArray, 6, 3);
16
17       System.out.printf(
18          "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3, s4);
19    }
20 } // fim da classe StringConstructors
```

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

String

Código (Main.java)

```
1  import java.lang.System;
2
3  public class Main{
4      public static void main(String[] args){
5          String   var1 = new String ("10");
6          String   var1 = "10";
7      }
8  }
9  /* Qual a diferença entre usar e não usar o construtor?
10  * R: No construtor a string é armazenada no HEAP
11  * Na notação literal a string é armazenada em um
12  * String Pool
13  */
```

String

https://www.w3schools.com/java/java_ref_string.asp

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
codePointAt()	Returns the Unicode of the character at the specified index	int
codePointBefore()	Returns the Unicode of the character before the specified index	int
codePointCount()	Returns the number of Unicode values found in a string.	int
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String

String

toString()

length()

split()

charAt()

concat()

valueOf()

indexOf()

replace()

equals()

String

(exercício)

Código (Main.java)

```
1  public class Main{
2      public static void main(String[] args){
3        String   var1 = "café";
4        String   var2 = "café";
5        System.out.println(var1 == var2);
6        System.out.println(var1.equals(var2));
7        System.out.println(var1.intern()==var2.intern());
8  }
```

Saída

```
false
true
true
```

equals(): Compara duas strings. Retorna verdadeiro se as strings forem iguais e falso se não.

inter(): Retorna a representação canônica do objeto string.

Scanner

DEITEL, Paul. Java: como programar.

Um **Scanner** permite a um programa ler os dados (por exemplo, números e strings) para utilização nele.

SCHILDT, Herbert. Java para iniciantes.

Scanner pode ser usada na leitura de entradas de várias fontes, inclusive do console e de arquivos. Portanto, você pode usá-la para ler um string numérico inserido pelo teclado e atribuir seu valor a uma variável.

Scanner

É uma classe que viabiliza a leitura de dados da entrada padrão
(e de outras entradas)

Scanner

Para usar a classe Scanner, você **precisa** importar obrigatoriamente seu pacote correspondente

Código (Main.java)

```
1  import java.util.Scanner;
2
3  public class Main{
4     public static void main(String[] args){
5       Scanner var1 = new Scanner(System.in);
6       int     n1   = var1.nextInt();
7
```

DEITEL, Paul. Java: como programar.

A palavra-chave `new` cria um objeto `Scanner` que lê caracteres digitados pelo usuário no teclado. O **objeto de entrada padrão**, `System.in`, permite que aplicativos leiam *bytes* de informações digitadas pelo usuário. O `Scanner` traduz esses bytes em tipos (como `int`s) que podem ser utilizados em um programa.

Scanner

Para usar a classe Scanner, você precisa importar obrigatoriamente seu pacote correspondente

Código (Main.java)

```
1  import java.util.Scanner;
2
3  public class Main{
4      public static void main(String[] args){
5          Scanner var1 = new Scanner(System.in);
6          int     n1   = var1.nextInt();
7
8
9
10         var1.close();
11     }
12 }
```

Está sendo criando um objeto do tipo `Scanner` vinculado ao console (`System.in`)

Objetos Scanners são objetos descartáveis. Após seu uso, é possível descartá-los para liberar a memória

Scanner

Para usar a classe Scanner, você precisa importar obrigatoriamente seu pacote correspondente

Código (Main.java)

```
1  import java.util.Scanner;
2
3  public class Main{
4     public static void main(String[] args){
5       Scanner var1 = new Scanner(System.in);
6       int     n1   = var1.nextInt();
7       //File arquivo = new File("Documento.txt");
8       //Scanner var1 = new Scanner(arquivo);
9       //É preciso fazer mais algumas coisas para ler arquivos.
10      var1.close();
11    }
12 }
```

É possível declarar mais de um `Scanner` (apenas tenha cuidado para não dar conflito)

Scanner

Não é necessário realizar a importação do pacote caso você faça o uso do **nome de classe totalmente qualificado**

Código (Main.java)

```
1  //import java.util.Scanner;
2
3  public class Main{
4     public static void main(String[] args){
5        //Scanner var1 = new Scanner(System.in);
6        java.util.Scanner var1 = new java.util.Scanner(System.in);
7        int      n1    = var1.nextInt();
8
9
10       var1.close();
11    }
12 }
```

Scanner

Não é necessário realizara importação do pacote caso você faça o uso do **nome de classe totalmente qualificado**

Código (Main.java)

DEITEL, Paul. **Java: como programar.**

O compilador Java não requer declarações `import` *em um arquivo de código-fonte Java se o nome de classe totalmente qualificado for especificado sempre que um nome de classe é usado.*

A maioria dos programadores Java prefere o estilo de programação mais conciso que as declarações `import` *fornecem .*

```
11
12  }
```

Scanner

É importante averiguar se um tipo específico está disponível chamando um dos métodos `hasNextX` (X é o tipo de dado desejado)

Código (Main.java)

```
1  //import java.util.Scanner;
2
3  public class Main{
4     public static void main(String[] args){
5        Scanner var1 = new Scanner(System.in);
6        if (var1.hasNextInt()){
7           int n1 = var1.nextInt();
8        }
9        var1.close();
10    }
11 }
12
```

Scanner

É importante averiguar se um tipo específico está disponível chamando um dos métodos `hasNextX` (`X` é o tipo de dado desejado)

Código (Main.java)

```
1  //import java.util.Scanner;
2
3  public class Main{
```

SCHILDT, Herbert. Java para iniciantes.

Tecnicamente, você pode chamar um método **next** sem antes chamar um método **hasNext**. No entanto, pode não ser uma boa ideia. Se um método **next** não puder encontrar o tipo de dado que estiver procurando, lançará uma **InputMismatchException**. Logo, é melhor confirmar primeiro se o tipo de dado desejado está disponível chamando um método **hasNext** antes de chamar o método **next** correspondente.

Scanner

Assumindo o USUÁRIO IDEAL

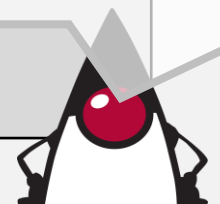
Código (Main.java)

```java
import java.util.Scanner;

public class Main{
    public static void main(String[] args){
        Scanner var1 = new Scanner(System.in);
        int     n1 = var1.nextInt();
        short   n2 = var1.nextShort();
        byte    n3 = var1.nextByte();
        double  n4 = var1.nextDouble();
        char    n5 = var1.next().charAt(0);
        String  n6 = var1.nextLine();
        boolean n7 = var1.nextBoolean();
        long    n8 = var1.nextLong();
        float   n9 = var1.nextFloat();
    }
}
```

Eu já vi isso:
"%d"
"%hd"
"%g"
"%c"
"%s"
"%ld"
"%f"

Chega! Cansei de ver você menosprezar o java!



Resumo

- **Classe:** É a declaração de um novo tipo de dado no programa.
- **Objeto:** É a manifestação uma certa entidade no programa.
- **Abstração:** Considera apenas os aspectos essenciais para a modelagem.
- **Atributos:** São as características de um objeto (adjetivo).
- **Métodos**: São as atividades de um objeto (verbo).
- **Classes Invólucro:** São uma forma de trabalhar com os tipos primitivos como objetos, ao invés de trabalhar como variáveis.
- **`String`:** É a classe invólucro do `char` (possibilitando texto no JAVA).
- **`Scanner`:** É uma classe que viabiliza a leitura de dados da entrada padrão.

Referências

SCHILDT, Herbert. **Java para iniciantes: crie, compile e execute**. 6. ed. Porto Alegre: Bookman, 2015. Disponível em: <https://app.minhabiblioteca.com.br/reader/books/9788582603376>.

TAVARES, Roberto; SILVA, Fábio. **Introdução à Programação para Engenharia**. Rio de Janeiro: GEN, 2022. Disponível em: <https://app.minhabiblioteca.com.br/reader/books/9788521638346>.

DEITEL, Paul. **Java: como programar**. 4. ed. Porto Alegre: Bookman, 2003.

DEITEL, Paul. **Java: como programar**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

CLASSE E OBJETO

Alexandre Mendonça Fava
alexandre.fava@udesc.br

Universidade do Estado de Santa Catarina – UDESC
Programa de Pós-graduação em Computação Aplicada – PPGCA