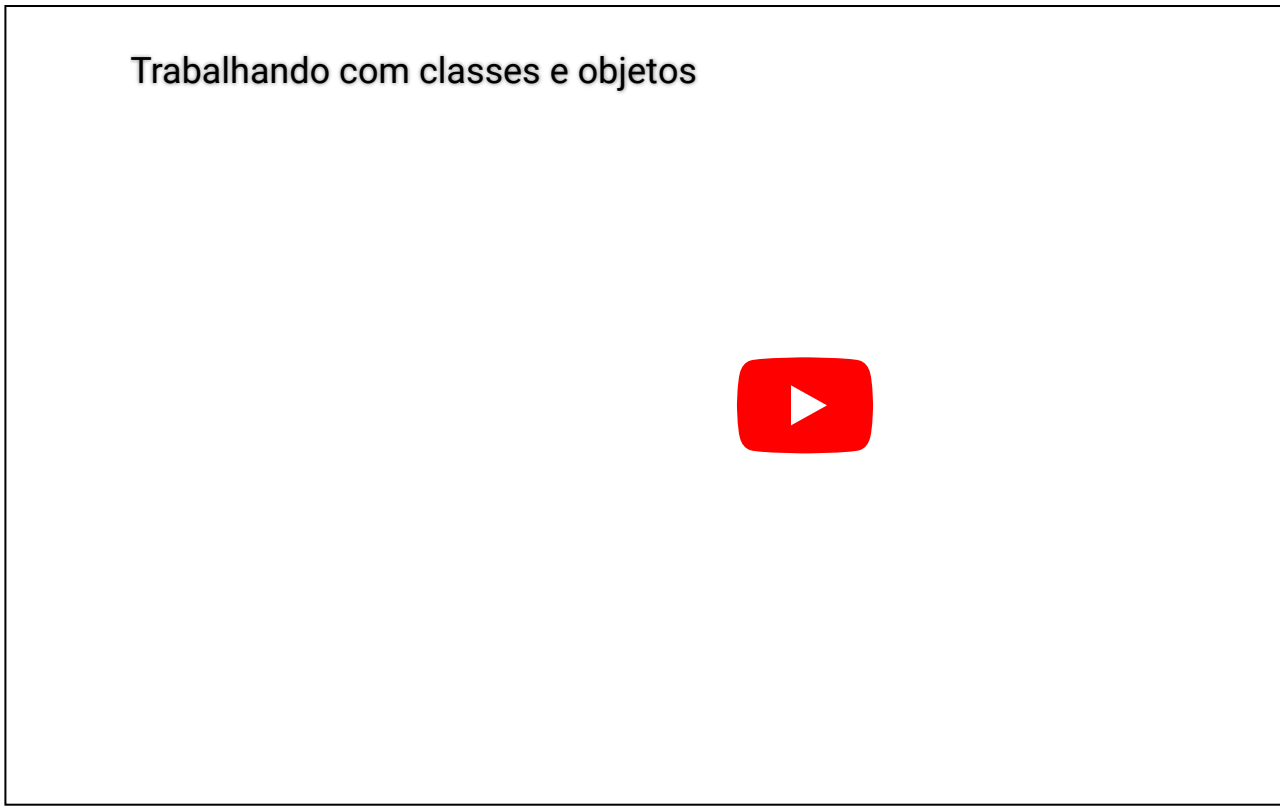
Fundamentos da Programação Orientada a Objetos

UNIDADE 02

Classes e objetos

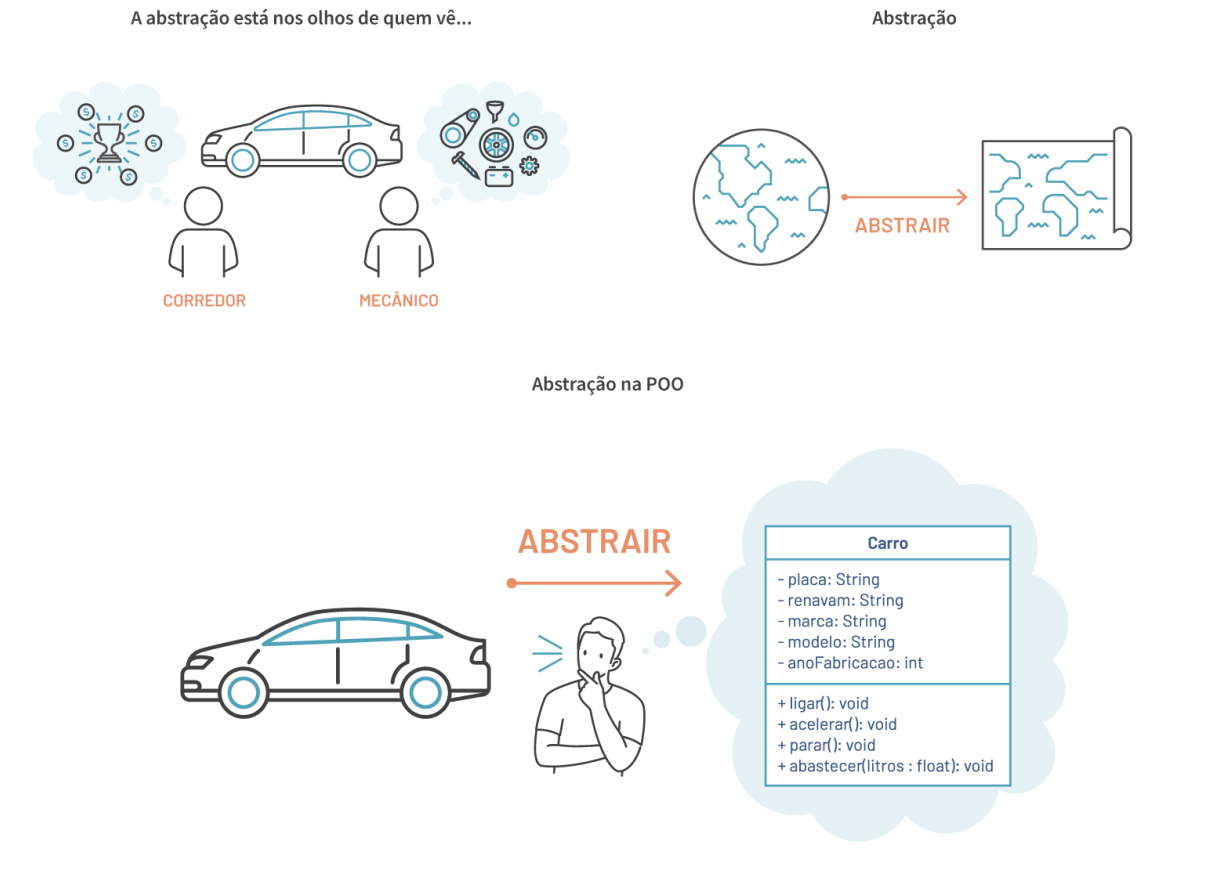
*Olá! Nesta Unidade, iremos criar programas com nossas primeiras **classes** e **objetos**, recursos essenciais nas linguagens orientadas a objetos, que permitem criar soluções computacionais de forma **intuitiva**.*



O ponto de partida é **representar entidades de mundo real** como **classes**, em um processo chamado de **abstração**, como demostrado na Figura 1 a seguir.

Abstrair é observar um ou mais elementos de um todo, avaliando suas características e propriedades de forma isolada, com o objetivo de **separar o elemento, ou o conceito, com o qual precisamos trabalhar**, excluindo todos os demais. Aplicada à Programação Orientada a Objetos (POO), a ação de abstrair consiste em separar conceitos e domínios do mundo real para representá-los como **classes**.

Figura 1 – Abstração na orientação a objetos



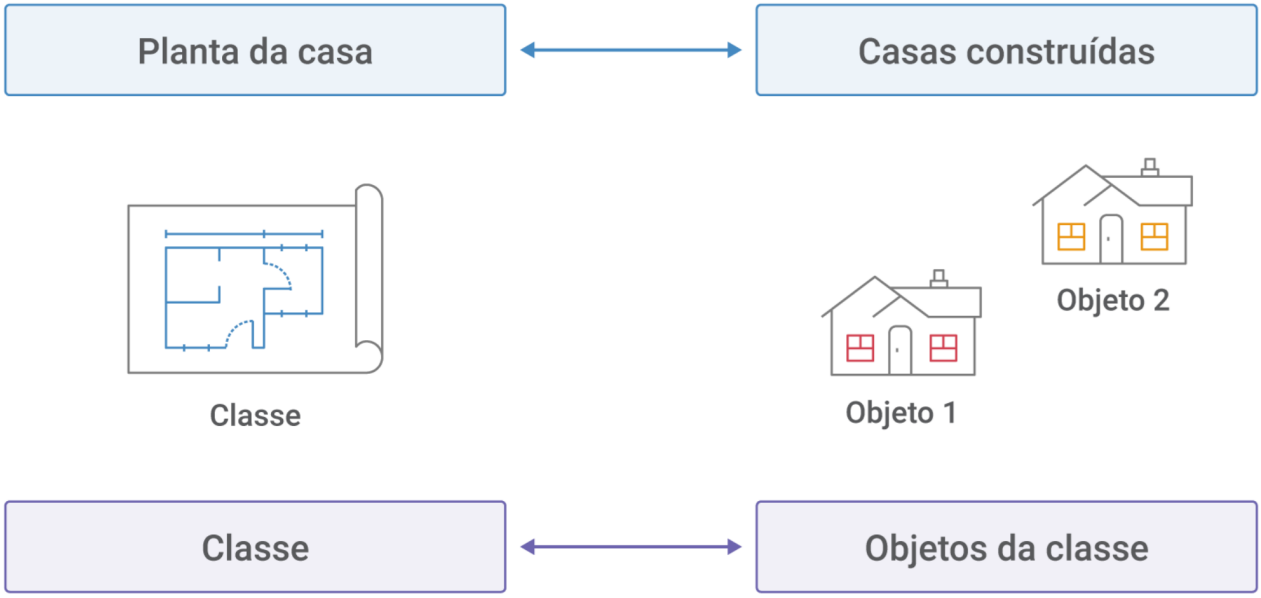
Fonte: Os autores (2020).

Membros da classe: atributos e métodos

Os principais conceitos da **POO** são **classes** e **objetos**, **comparados** na Figura 2 a seguir.

A **classe** representa um **conjunto de objetos** com **características semelhantes**. Podemos fazer uma analogia, ou comparação, na qual: uma classe é um modelo e o objeto é a realização desse modelo, da mesma forma como acontece com a “planta de uma casa” e a “casa construída”.

Figura 2 – Classe x objetos



Fonte: Os autores (2021).

Para exemplificar, vamos pensar em uma solução computacional que é concebida a partir de um **exercício de abstração**, em que precisamos identificar os elementos que interessam a um determinado *software.*

EXERCÍCIO DE ABSTRAÇÃO

Pense em um programa para controlar as **lâmpadas de uma casa**. Na orientação a objetos, cada lâmpada é representada como um **objeto**, ou **uma entidade**, que tem um **estado** que pode estar "aceso" ou "apagado", e os **comportamentos** de "acender a lâmpada e "apagar a lâmpada".

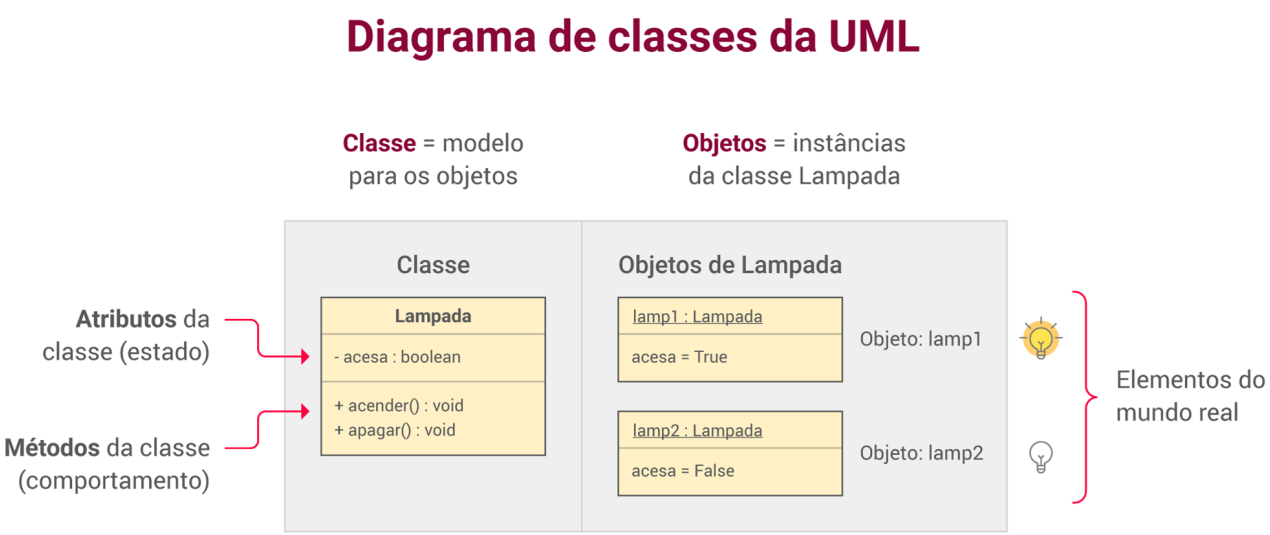
Essa modelagem inicial é melhor representada na orientação a objetos da seguinte forma:

- A **abstração**, ou ideia, de "lâmpada" é mapeada como uma **classe**.
- Qualquer classe tem uma estrutura composta por dois tipos de **membros**:
  - **Atributos**, ou **estado** da classe.
  - **Métodos**, ou **comportamento** da classe.
- As várias lâmpadas da casa – cada uma podendo estar acesa ou apagada – são as **instâncias da classe**, ou **objetos da classe**.

Veja como fica a representação dessa modelagem, feita com UML, uma ferramenta de diagramação própria para a orientação a objetos, na Figura 3 a seguir.

Observe a representação de **classes** e **objetos** na notação do **diagrama de classes** da *Unified Modeling Language* (UML), ou linguagem unificada de modelagem.

Figura 3 – Pensando em classes e objetos



Fonte: Os autores (2021).

Estado da classe:

- Situações representadas pelos valores dos seus **ATRIBUTOS** (ou propriedades).
- Interno ao objeto.
- Mudanças de estado podem ser realizadas pelo comportamento do objeto.
- No exemplo da Figura 3, é o atributo **acesa** que pode ter os valores "***true***" ou "***false***".

Comportamento da classe:

- Definido pelo conjunto de **MÉTODOS** que o objeto pode executar.
- Também conhecido como interface, operações ou funções do objeto.
- No exemplo da Figura 3, são os métodos **acender()** e **apagar()** que alteram o **valor do atributo da classe**, ou o estado da classe.

Instância e referência de objetos

Como pôde ser observado nas Figuras 2 e 3, quando a **classe**, que é um modelo, é concretizada, ela se torna um **objeto**. Essa transformação é denominada de **instanciação**.

OBJETO = **INSTÂNCIA** de classe

- Quando **instanciamos uma classe**, criamos **um objeto** da classe.
- Uma **classe** é um **tipo de objeto**.

Por ser uma linguagem puramente orientada a objetos, o Java fornece mecanismos para definir e criar nossas próprias classes, que serão instanciadas como objetos, para manter os dados dos nossos programas.

A abstração, portanto, é essencial para criar programas com objetos de mundo real que interagem. Essa estratégia nos permite lidar com a complexidade do *software*, pois possibilita a decomposição dos vários problemas em problemas menores. Isso porque, na prática, cada classe é como um pequeno programa, que tem atributos (dados ou estado) e métodos (funcionalidades que definem o comportamento do objeto) (GODOY, 2019).

Criando e instanciando nossa primeira classe

No Java, cada classe é definida em um arquivo próprio, que tem o nome idêntico ao da classe e extensão **.java**. Como primeiro exemplo, iremos trabalhar com algumas formas geométricas. Vamos começar com um **retângulo**. Ele representa uma entidade de mundo real, logo pode ser representado como uma classe, mantida em um arquivo próprio.

- **Retângulo**
  - Definido pelas suas medidas de “altura” e “largura” (atributos).
  - Um retângulo pode “calcular o seu perímetro” e também “apresentar os seus dados” (métodos).

BOAS PRÁTICAS DE PROGRAMAÇÃO

- **Nome de classe** = inicie com letra **maiúscula**.
- **Nome de atributo** = inicie com letra **minúscula** e use o nome de um **substantivo**.
- **Nome de método** = inicie com letra **minúscula** e use o nome de um **verbo**.
- Não use acentos da língua portuguesa em identificadores de qualquer linguagem de programação: **nome** de **método**, de **classe** ou de **atributo**.

NOME DO ARQUIVO Java: terá o mesmo **nome da classe**, com extensão **.java**:

- `Retangulo.java`

PARA DEFINIR UMA CLASSE no Java:

- Usar as palavras-chave ***`public class`*** seguidas do nome da classe:

```
public class Retangulo
```

PARA CRIAR UM OBJETO no Java:

- Usar a palavra-chave ***`new`*** da seguinte forma:

```
Retangulo retg1;          // nome de referência de objeto
retg1 = new Retangulo();  // instanciação da classe Retangulo
```

- A **referência de objeto** é o **nome do objeto** de uma determinada classe.
- No exemplo, `retg1` é um objeto da classe `retangulo` .

Figura 4 – Arquivo Retangulo.java

</>

```
 1  public class Retangulo {
 2      // Atributos da classe = ESTADO da
classe
 3      float altura;
 4      float largura;
 5
 6      // Métodos da classe = COMPORTAMENTO
da classe
 7      float calcularPerimetro() {
 8          float pm;                  //
variável local
 9          pm = 2 * altura + 2 * largura;
10          return pm;
11      }
12      void imprimirDados() {
13          float p;                    //
variável local
14          p = calcularPerimetro();    //
calcula o perímetro do retângulo
15          System.out.println("Retângulo: ");
16          System.out.println("- altura:    "
+ altura);
17          System.out.println("- largura:   "
+ largura);
18          System.out.println("- perimetro: "
+ p);
19      }
20      // Método principal - início de
execução do programa
21      public static void main(String[]
args) {
22          System.out.println("Mundo dos
retângulos");
23
24          // Objeto de Retangulo retg1
25          Retangulo retg1;          //
referência de objeto da classe Retangulo
26          retg1 = new Retangulo(); //
instanciação da classe Retangulo em objeto
27          retg1.altura  = 10;       //
atribuição de valor para atributo do objeto
28          retg1.largura = 20;       //
atribuição de valor para atributo do objeto
29          retg1.imprimirDados();    //
invocação de método do objeto
30
31          // Objeto de Retangulo retg2
32          Retangulo retg2;          //
referência de objeto da classe Retangulo
```

```
Console
Mundo dos retângulos
Retângulo:
- altura:    10.0
- largura:   20.0
- perimetro: 60.0
Retângulo:
- altura:    5.0
- largura:   15.0
- perimetro: 40.0
```

```
 33        retg2 = new Retangulo(); //
instanciação da classe Retangulo em objeto
 34        retg2.altura  = 5;       //
atribuição de valor para atributo do objeto
 35        retg2.largura = 15;      //
atribuição de valor para atributo do objeto
 36        retg2.imprimirDados();   //
invocação de método do objeto
 37     }
 38  }
```

Fonte: Autores (2021).

Vamos analisar o código da Figura 4.

Linha 1: criamos a classe, delimitada por { e }, nas Linhas 1 e 38, respectivamente.

Linhas 3 e 4: definimos dois atributos do tipo **`float`***:* `altura` *e* `largura`

Linha 7: definimos o método **`float`** `calcularPerimetro()`, que:

- Calcula o valor do perímetro do retângulo, na linha 9, atribuído à variável local ***`float`*** *`pm;`*.
- O valor de retorno, também do tipo ***`float`***, é retornado de *`pm`*, na linha 10.
- O método não tem não tem parâmetros de entrada, por isso tem um () logo após o nome do método – vamos falar mais de métodos na próxima seção.

Linha 12: definimos o método **`void`** `imprimirDados()`, que:

- Calcula o perímetro do retângulo, atribuído à variável local `p` na linha 14.
- Apresenta no console os valores dos atributos dos objetos da classe retângulo, nas linhas 16 e 17, e o valor calculado do perímetro na linha 18.
- Esse método não tem parâmetros e não tem retorno, por isso o nome do método é precedido pela palavra-chave **`void`**.

Linha 21: definimos o **método principal** (`main`), que é o ponto inicial do nosso programa.

Linha 25: definimos uma **referência para objeto** da classe `Retangulo retg1;`. Nesse momento, o objeto ainda **não foi instanciado**.

Linha 26: **instanciamos** um objeto `retg1 =` **`new`** `Retangulo()`, que tem o nome de (é referenciado por) `retg1`, da classe, ou do tipo, `retangulo`.

Linha 27 e 28: definimos valores para os atributos do objeto `retg1`.

Linha 29: invocamos o método `imprimirDados()` do objeto `retg1` – exibe no console informações do retângulo `retg1`.

Linha 32 a 36: repetimos os mesmos passos para um **novo objeto** da classe `Retangulo`, o `retg2`.

Quando criamos a referência de uma classe, estamos declarando uma variável do tipo da classe. Essa variável ainda não é um objeto, ela apenas pode referenciar o objeto, que é criado apenas quando instanciamos a classe. O operador ***new*** do Java aloca memória em tempo de execução para manter o objeto e retorna a referência, que é atribuída a uma variável do tipo da classe do objeto. Podemos definir e atribuir um objeto em uma mesma linha de comando, resumindo as linhas 25 e 26 e 32 e 33, conforme apresentado a seguir.

</>

```
  1  Retangulo retg1 = new Retangulo();   // define referência e
recebe o obj. de Retangulo
  2  Retangulo retg2 = new Retangulo();   // define referência e
recebe o obj. de Retangulo
```

Invocação de métodos dos objetos

Conforme mencionado anteriormente, os **métodos de uma classe são o comportamento que o objeto pode realizar**, ou seja, **operações** que o objeto pode executar. Por isso, é recomendado, sempre que possível, utilizar **verbos para o nome do método**, para indicar uma ação.

> Um **método**, na orientação a objetos, equivale a uma **função** na programação procedural.

A declaração de uma função segue a seguinte sintaxe para a **assinatura** de método (SCHILDT, 2015):

> **`tipo nomeMetodo (tipo parametro1, tipo parametro2, ..., tipo parametroN) {`**

```
        // corpo do método
    }
```

O **tipo de dado de retorno do método**, seguido do **nome do método** e da sua **lista de parâmetros**, define a **assinatura do método**. Cada parâmetro do método deve ter um tipo de dado declarado, como mostrado anteriormente.

Quando o **método não tem retorno**, é usada a palavra-chave ***void***. Se o método não tem parâmetros, sua lista de parâmetros fica vazia.

O **corpo do método** define um **bloco de código**, delimitado por { e }, que pode usar as operações e os atributos da classe, mas também pode usar novas variáveis, que apenas existem dentro do método e são chamadas de **variáveis locais.**

Veja o exemplo a seguir do **imprimirDados ()**, que é um método sem lista de parâmetros e sem retorno.

</>

```
1  void imprimirDados() {
2        float p;                    // variável local p do
tipo float
3        p = calcularPerimetro();    // calcula o perímetro do
retângulo e atribui à p
4        System.out.println("Retângulo: ");
5        System.out.println("- altura:    " + altura);  //
apresenta valor do atributo
6        System.out.println("- largura:   " + largura); //
apresenta valor do atributo
7        System.out.println("- perimetro: " + p);
8    }
```

O método **imprimirDados ()** também **invoca outro método**, quando executa o comando:

</>

```
1  p = calcularPerimetro();   // calcula o perímetro do
retângulo
```

Invocar um método, na orientação a objetos, equivale a **chamar uma função** na programação procedural.

No exemplo da Figura 4, temos **invocação de método** nas linhas 14, 29 e 36.

A palavra-chave "return"

Quando um método tem um **tipo de retorno**, é obrigatório utilizar o comando Java ***return*** ao final da execução do método. É dessa forma que o retorno do método é atribuído à uma variável. Veja o exemplo do método **calcularPerimetro ()** da Figura 4.

```
</>
```

```
1  float calcularPerimetro() {
2        float pm;   // variável local
3        pm = 2 * altura + 2 * largura;
4        return pm;  // retorno do método é do tipo float
5    }
```

Quando **calcularPerimetro ()** é invocado, ele atribui o seu retorno à uma variável do tipo ***float***.

```
</>
```

I

```
1  float p;                    // variável local p do tipo
float
2        p = calcularPerimetro();   // calcula o perímetro do
retângulo e atribui à p
```

Métodos construtores

Quando usamos o comando Java ***new*** para instanciar uma classe, ou seja, para criar um objeto, estamos usando um método especial chamado de **método construtor** (GODOY, 2019).

Um **construtor inicializa um objeto quando ele é criado**. Ele tem exatamente o nome da classe, contudo, não tem um tipo de retorno explícito. Os construtores são usados de forma geral para fornecer valores iniciais para os atributos da classe e executar alguma operação inicial para o objeto (SCHILDT, 2015).

Todas as classes têm construtores, mesmo quando não estão definidos no código, pois o Java fornece automaticamente um construtor-padrão que inicializa os atributos da classe para os seus valores-padrão: **zero**, **null** e **false**, respectivamente para valores **numéricos**, **tipos de referência** e ***booleans*** (SCHILDT, 2015).

Podemos criar um construtor para o exemplo da Figura 4 da forma apresentada a seguir.

</>

```
1  public class Retangulo {
2     // Atributos da classe = ESTADO da classe
3     float altura;
4     float largura;
5
6     public Retangulo (float alt, float larg) { // Define o
método construtor
7  altura = alt;
8  largura = larg;
9     }
10   ...
11  }
```

Com um construtor personalizado, a criação dos objetos pode ser realizada da forma apresentada a seguir.

</>

```
  1  retg1 = new Retangulo(10, 20);     // Construtor já
inicializa os atributos
  2  retg2 = new Retangulo(5, 15);      // da classe altura e
largura
```

IMPORTANTE

Quando criamos o nosso construtor, não é mais possível utilizar o construtor-padrão gerado automaticamente pelo Java.

</>

```
  1  retg1 = new Retangulo(); // gera erro quando existe um
construtor personalizado
  2  retg2 = new Retangulo(); // gera erro quando existe um
construtor personalizado
```

O comando apresentado passa a gerar erro de compilação "***The constructor* Retangulo*() is undefined***", ou "O construtor Retangulo() é indefinido", indicando que esse comando não existe no programa.

Palavras-chave do Java

Vamos ver como o Java trata com alguns aspectos importantes da orientação a objetos, utilizando palavras-chave da linguagem.

A palavra-chave "this"

Como os construtores servem para inicializar os atributos da classe, uma prática comum é **declarar o nome dos parâmetros exatamente igual ao nome dos atributos nas classes**. Contudo, o código dessa forma gera ambiguidade.

</>

```
 1  public class Retangulo {
 2      float altura;       // Atributos da classe = ESTADO da
classe
 3      float largura;      // Atributos da classe = ESTADO da
classe
 4
 5      public Retangulo (float altura, float largura) {
 6  altura = altura;      // PROBLEMA de ambiguidade:
 7  largura = largura;    // atributo e parâmetro têm o mesmo
nome
 8      }
 9  ...
10  }
```

Para resolver esse problema, o Java fornece a palavra-chave ***this***, que é uma referência para o objeto em questão, ou seja, referência para o próprio objeto (GODOY, 2019). O **código corrigido** fica como:

</>

```
 1  public class Retangulo {
 2      float altura;       // Atributos da classe = ESTADO da
classe
 3      float largura;      // Atributos da classe = ESTADO da
classe
 4
 5      public Retangulo (float altura, float largura) {
 6  this.altura = altura;     // atributo recebe o valor do
parâmetro
 7  this.largura = largura;   // atributo recebe o valor do
parâmetro
 8      }
 9  ...
10  }
```

A palavra-chave "*static*"

Em alguns casos, precisamos usar métodos e atributos de uma classe **sem precisar instanciar**, ou criar um objeto da classe. Nesses casos, usamos palavra-chave ***static*** antes do nome do atributo ou do método.

A Figura 5 a seguir apresenta um novo exemplo de código, que tem dois métodos ***static***: `somar(x, y)` e `multiplicar(x, y)`. Nos dois casos, **a invocação desses métodos usa no nome da classe, e não o nome de um objeto, da classe `Calculadora`** – invocações realizadas nas linhas 22 e 35.

Figura 5 – Exemplo de métodos *static*

</>

```
 1  import java.util.Scanner;
 2
 3  public class Calculadora {
 4      // métodos static são invocados diretamente da sua
classe
 5      static float somar (float a, float b) {
 6         return a + b;
 7      }
 8      static float multiplicar (float a, float b) {
 9         return a * b;
10      }
11
12      public static void main(String[] args) {
13         Scanner leitor = new Scanner(System.in); // leitor de
teclado
14         float x, y, z; // variáveis locais
15
16         System.out.print("Entre o valor de x: ");
17         x = leitor.nextFloat();
18
19         System.out.print("Entre o valor de y: ");
20         y = leitor.nextFloat();
21
22         z = Calculadora.somar(x, y);        // Método static:
invocado diretamente
23         System.out.println("x + y = " + z); // da classe
Calculadora
24
25         z = Calculadora.multiplicar(x, y);  // Método static:
invocado diretamente
26         System.out.println("x * y = " + z); // da classe
Calculadora
27      }
28  }
```

Fonte: Autores (2021).

> Atributos e métodos **estáticos** pertencem à classe, não a uma instância específica (objeto específico).

Quando declaramos um **atributo como *`static`***, ele é **compartilhado por todas as instâncias da classe** (GODOY, 2019): o atributo é mantido na mesma área de memória para todas as instâncias da classe. Isso significa que se um objeto da classe alterar o valor do atributo, todos os demais objetos da mesma classe terão o mesmo valor alterado no seu atributo.

Veja um exemplo do atributo ***`static`*** na Figura 6 a seguir.

Figura 6 – Exemplo de atributo *static*

</>

```
 1  public class Cambio {
 2      static float cotacaoDolar = 4.0f; //
Atributo static
 3
 4      float converteDolar (float valorReal)
{
 5          return valorReal / cotacaoDolar;
 6      }
 7      float converteReal (float
valorDolares) {
 8          return valorDolares *
cotacaoDolar;
 9      }
10
11      public static void main(String[]
args) {
12          float valorDolar;
13          float valorReal;
14
15          Cambio c1 = new Cambio(); // Cria
o objeto c1 da classe Cambio
16
17          // Apresenta cotacoes
18          System.out.printf("\n-- U$ 1,00
vale R$ %.2f\n",Cambio.cotacaoDolar);
19          System.out.printf("U$ 10,00 = R$
%.2f\n", c1.converteReal(10f));
20          System.out.printf("R$ 10,00 = U$
%.2f\n", c1.converteDolar(10f));
21
22          Cambio c2 = new Cambio(); // Cria
o objeto c2 da classe Cambio
23
24          Cambio.cotacaoDolar = 5f; //
alteração do atributo na classe
25
26          // Apresenta cotacoes
27          System.out.printf("\n-- U$ 1,00
vale R$ %.2f\n",Cambio.cotacaoDolar);
28          System.out.printf("U$ 10,00 = R$
%.2f\n", c2.converteReal(10f));
29          System.out.printf("R$ 10,00 = U$
%.2f\n", c2.converteDolar(10f));
30      }
31  }
```

Console

```
-- U$ 1,00 vale R$ 4,00
U$ 10,00 = R$ 40,00
R$ 10,00 = U$ 2,50

-- U$ 1,00 vale R$ 5,00
U$ 10,00 = R$ 50,00
R$ 10,00 = U$ 2,00
```

Fonte: Autores (2021).

Observe como o ***static*** é utilizado no exemplo da Figura 6.

Na linha 2, é declarado o único atributo ***static*** da classe `Cambio`: ***`static float cotacaoDolar`***, que recebe o valor de `4.0f`.

Na linha 15, é criado um objeto da classe: `Cambio c1 = new Cambio();`.

Nas linhas 19 e 20, são invocados os métodos `c1.converteReal(10f)` e `c1.converteDolar(10f)`, e usamos o método `System.out.printf` com uma máscara para formatar um número ***float*** com 2 casas decimais: `%.2f`; a sequência "\n" garante o salto de linha.

Na linha 22, é criado mais um objeto da classe: `Cambio c2 = new Cambio();`.

Na linha 24, é alterado o atributo ***`static float cotacaoDolar`*** da classe, que passa a ter o valor de `5f` - nesse momento, os dos objetos da classe, `c1` e `c2`, passam a compartilhar esse novo valor de ***`cotacaoDolar`***.

Nas linhas 28 e 29, são invocados os métodos `c1.converteReal(10f)` e `c1.converteDolar(10f)`, já usando o novo valor de `cotacaoDolar = 5f`.

As palavras-chave "*final static*" definem uma constante

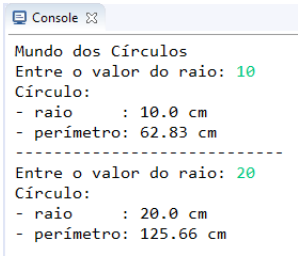
É comum utilizar a palavra-chave ***static*** junto com a palavra-chave ***final*** (que não permite alteração) para definir constantes.

Veja um exemplo desse tipo de uso no código completo da classe `Circulo`, exibido na Figura 7 a seguir.

Figura 7 – Exemplo de atributo constante definido como *final static*

</>

```
 1  import java.util.Scanner;
 2
 3  public class Circulo {
 4  // Atributo da classe --------
 5     float raio;
 6     final static float PI = 3.1415f; //
Contante: final static
 7
 8     public Circulo (float raio) {    //
Construtor
 9     this.raio = raio;
10     }
11
12  // Métodos da classe ---------
13     float calcularPerimetro() {
14        return 2 * PI * this.raio;
15     }
16     void imprimirDados() {
17        System.out.println("Círculo: ");
18        System.out.println("- raio     : "
+ raio);
19        System.out.println("- perímetro: "
+ calcularPerimetro());
20     }
21     // Método principal -------------
22     public static void main(String[]
args) {
23        Scanner leitor = new
Scanner(System.in); // leitor de teclado
24        float raio;
25        System.out.println("Mundo dos
Círculos");
26        System.out.print  ("Entre o valor
do raio: ");
27        raio = leitor.nextFloat();
// recebe valor float em raio
28        Circulo circ1 = new Circulo
(raio); // instancia Circulo em circ1
29        circ1.imprimirDados();
// invoca método
30        System.out.println("--------------
--------------");
31     System.out.print  ("Entre o valor do
raio: ");
32        raio = leitor.nextFloat();
// recebe valor float em raio
33        Circulo circ2 = new Circulo
(raio); // instancia Circulo em circ2
```



```
 34         circ2.imprimirDados();
// invoca método
 35     }
 36  }
```

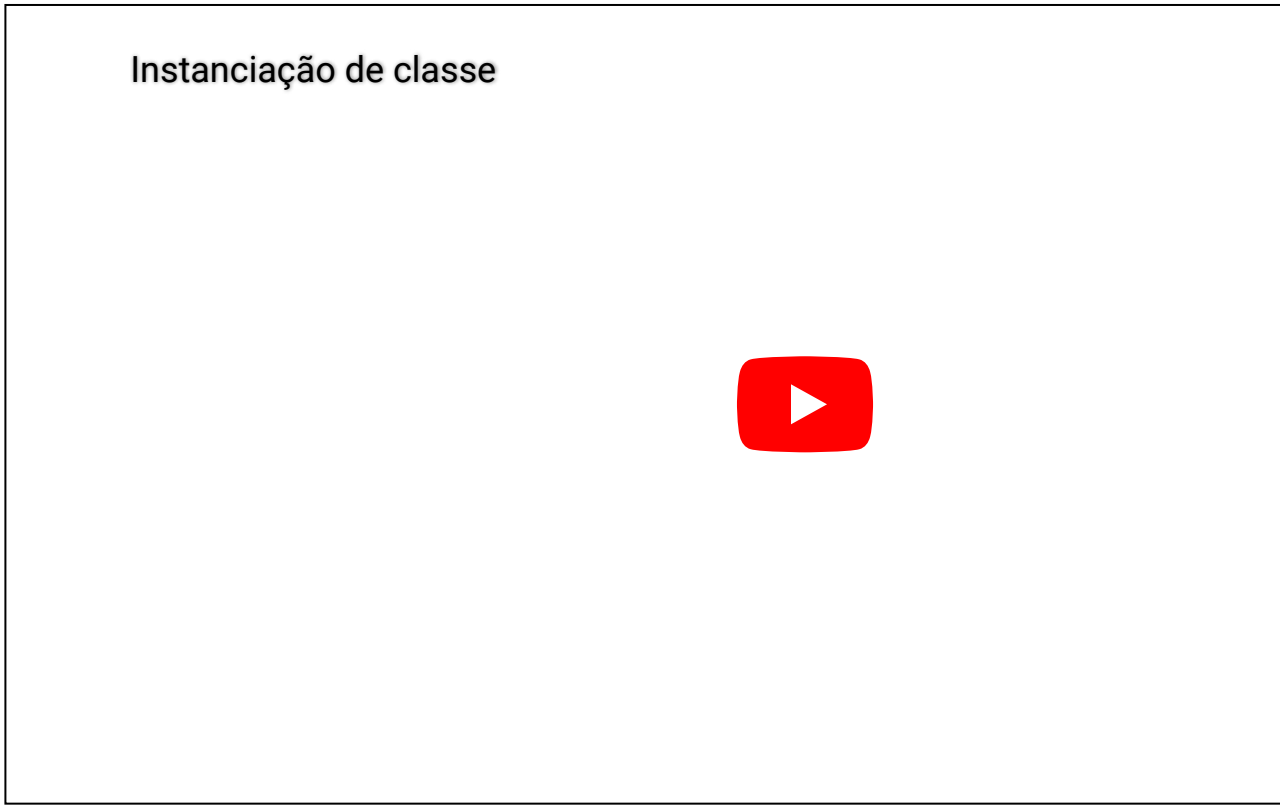
Fonte: Autores (2021).

O método ***main*** é estático justamente para poder iniciar o programa:

não é preciso instanciar a classe em que ele se encontra para poder invocar o método ***main***.

Praticando a utilização de atributos e métodos

Vamos recapitular e praticar novos exemplos de classes e de criação de objetos, com métodos e atributos, neste tutorial passo a passo.



EXPERIMENTE

Meu segundo programa: Olá, mundo da soma

Considerações finais

Nesta Unidade, aprendemos que uma classe é a representação de algo no mundo real: uma entidade ou um conceito. Uma classe tem determinada estrutura especial para manter seus membros: atributos e métodos.

- Os **atributos** definem o que precisamos manter do objeto: são seus dados, suas variáveis de classe, ou são o **estado** da classe.
- Os **métodos** definem as ações ou operações que os objetos de uma classe podem realizar; são o **comportamento** da classe.

Aprendemos também que **criamos objetos quando instanciamos uma classe**. Cada um mantém um estado próprio, e todos os objetos de uma classe podem executar as mesmas operações, ou métodos.

Vimos também que uma classe possui um método especial, chamado quando desejamos criar o objeto da classe: o **método construtor**, usado para inicializar os atributos da classe. No Java, para invocar o método construtor que instancia sua classe, usamos a palavra-chave ***new***.

Por fim, aprendemos que vários aspectos da orientação a objetos são tratados no Java com algumas palavras-chave da linguagem, como:

- ***This***: referência do objeto para si próprio.
- ***Static***: precede o nome de **atributos** e **métodos** da classe que são utilizados sem a necessidade de instanciar a classe (ou criar objetos da classe); são **atributos** e **métodos** de classe, e não de objeto:
  - Atributo ***static***: atributo de classe, cujo valor é o mesmo para todas as instâncias da classe; uma vez alterado, todas as instâncias da classe visualizam o mesmo novo valor.
  - ***Final static***: permite a criação de atributos de classe que são constantes, cujos valores não podem ser alterados.

Referências

GODOY, V. **Programação Orientada a Objetos I**. Curitiba: IESDE, 2019.

HORSTMANN, C. S.; CORNELL, G. **Core Java**. 8. ed. São Paulo: Pearson, 2010.

SCHILDT, H. **Java para Iniciantes**. Porto Alegre: Bookman, 2015.