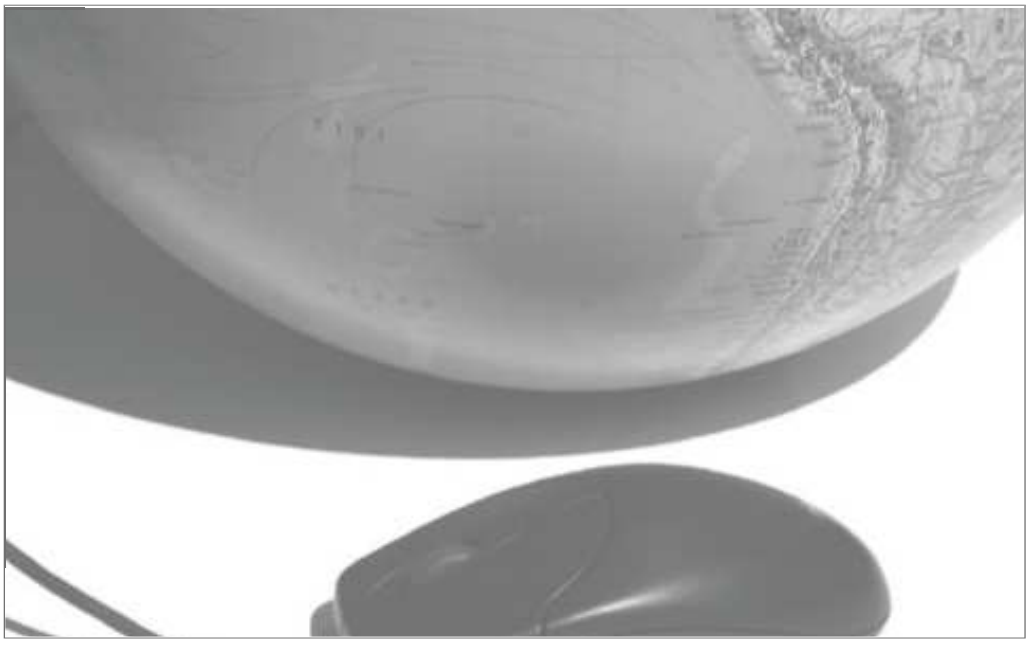
4

Orientação a Objetos: Classes, Objetos e Métodos



Meta

Nesta aula, serão discutidos os conceitos básicos da Orientação a Objetos, e, em seguida, como que estes conceitos básicos são implementados na Linguagem Java.

Objetivos

Ao final desta aula, você deverá ser capaz de:

- ✓ Compreender os principais conceitos da Orientação a Objetos (OO) envolvendo classe, objetos e métodos.
- ✓ Implementar programas em java utilizando dos conceitos da Orientação a Objetos.

1 – Introdução à Orientação a Objetos

Boa parte de nosso entendimento e relacionamento com o mundo se dá através do conceito de objetos. Ao observarmos as coisas e seres que existem ao nosso redor, existe uma natural tendência a tentarmos identificar o que é cada uma dessas diferentes entidades. Ao olharmos para uma bola de basquete, reconhecemos sua forma esférica, seu tamanho usual, colorido típico e aplicação. Mas a bola que observamos não é a única que existe, inúmeras outras estão "por aí" e, mesmo possuindo características idênticas, são objetos diferentes.

Como o próprio nome indica, a bola é de basquete, ou seja, para ser usada no esporte que denominamos basquete, é um tipo de bola específica. Assim, bola de basquete representa uma classe de objetos que apresentam características próprias. Cada uma das bolas de basquete existentes é um objeto desta classe de objetos. Existem outras bolas (outros tipos) que não são de basquete. Bolas especiais para os múltiplos esportes diferentes e bolas de recreação são exemplos de outras classes de objetos que compartilham das mesmas características de uma bola, ou melhor, de uma esfera.

Pode parecer óbvio, mas a dificuldade inicial do paradigma da Orientação a Objetos é justamente saber distinguir o que é **classe** e o que é **objeto**. É comum o iniciante utilizar, obviamente de forma errada, essas duas palavras como sinônimas.

Nas próximas sessões deste e dos próximos capítulos, mostraremos algumas das principais terminologias utilizadas na POO que serão utilizadas em todo este material, como **classes, objetos, herança, métodos, polimorfismo e encapsulamento.**

2 – Classe e Objeto

Uma **classe** é um tipo de objeto que pode ser definido pelo programador para descrever uma entidade real ou abstrata. Podemos entender uma classe como um "modelo" ou como uma "especificação" para certos objetos, ou seja, a descrição genérica dos objetos individuais pertencentes a um dado conjunto. A bola de basquete pode ser uma classe que descreve os objetos bola de basquete. A definição de uma classe envolve a definição de variáveis às quais se associam funções que controlam o acesso a esses objetos. Assim, a criação de uma classe implica definir um tipo de objeto em termos de seus atributos (variáveis que conterão os dados) e seus métodos (funções que manipulam tais dados).

Uma **classe** representa uma entidade e, como tal, define o seu comportamento (**métodos**) e as suas características (propriedades). Os acontecimentos que podem ocorrer para uma dada entidade em determinada altura são chamados de **eventos**.

Classe é um componente de programa que descreve a "estrutura" e o "comportamento" de um grupo de objetos semelhantes – isto é, as informações que caracterizam o estado desses objetos e as ações (ou operações) que eles podem realizar.

Já um **objeto** é uma **instância** de uma classe. Ele é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. **João**, **José** e **Maria**, por exemplo, podem ser exemplos de objetos da classe **Pessoa**, descrita acima.

Objeto é a criação de uma instância da classe. Quando **instanciamos** um objeto, criamos fisicamente uma representação concreta de sua classe. "Ser Humano" é uma classe, um molde, já "Roberto" é uma instância de "Ser Humano". Apesar de carregar todas as características do molde "Ser Humano", ele é completamente diferente (independente) das outras instâncias de "Ser Humano".

Um outro exemplo: uma receita de bolo. Você come uma receita de bolo? A resposta é "Não". Precisamos **instanciá-la**, criar um **objeto** bolo a partir dessa especificação (a classe) para utilizá-la. Podemos criar centenas de bolos a partir dessa classe (a receita, no caso) e eles podem ser bem semelhantes, alguns até idênticos, mas são **objetos** diferentes.

Podemos fazer várias analogias semelhantes. A planta de uma casa é uma casa? Definitivamente não. Não podemos morar dentro da planta de uma casa, nem podemos abrir sua porta ou pintar suas paredes. Precisamos, antes, construir instâncias a partir dessa planta. Essas instâncias, sim, podemos pintar, decorar ou morar dentro delas.

3 – Classes e Objetos em Java

Para exemplificar a implementação de classes em Java, vamos criar uma classe que armazena os dados da conta de um usuário no banco. Por enquanto, vamos escrever apenas o que a conta possui (atributos), e não o que ela faz (métodos).

```
class Conta {
  int numero;
  String nome;
  double saldo;
  double limite;
}
```

Esses são os **atributos** que toda conta, quando criada, irá possuir. Repare que essas variáveis foram declaradas fora de um bloco, diferente do que fizemos na classe que possuía um método ***main***. Quando uma variável é declarada diretamente dentro do escopo da classe, ela é chamada de **variável de objeto**, ou **atributo**. O diagrama da Figura 23 - mostra o que temos em nossa classe até agora:

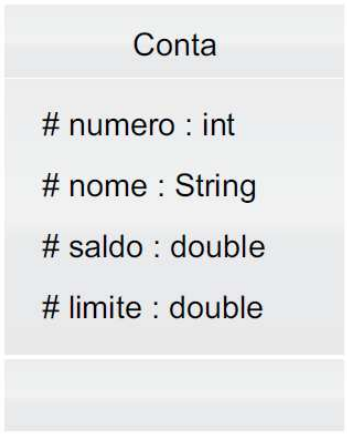


Figura 23 - Atributos da classe Conta

A classe **Conta** especifica o que todos os objetos dessa classe devem conter. Neste caso, ela não possuirá um método chamado ***main*** porque ela não será a classe principal do nosso programa. Para isso, criaremos uma classe **Programa**, que criará um objeto da classe **Conta**.

```
class Programa {
  public static void main (String[] args) {
    Conta minhaConta; /* Declarando a variável minhaConta,
que conterá um objeto da classe Conta */
     minhaConta = new Conta(); /* Criando efetivamente um
objeto da classe Conta e atribuindo-o à variável minha-
Conta */
```

```
minhaConta.nome = "Hilário"; /* Atribuindo o valor de uma
String à variável nome do objeto minhaConta */
    minhaConta.saldo = 1000.0; /* Atribuindo o valor de um
double à variável saldo do objeto minhaConta */
     System.out.println("Saldo atual: " +
                              minhaConta.saldo);
}
```

Preste atenção no ***ponto*** que foi utilizado em "minhaConta.nome" e "minhaConta.saldo". É assim que acessamos uma variável de objeto. Agora, sempre que o programa for iniciado, a conta pertencerá ao usuário "Hilário" e seu saldo será de mil reais.

A criação de um objeto a partir de uma classe é chamada de processo de **instanciação**. Para realizar a instanciação de um objeto qualquer, é usado o operador new (FURGERI, 2005). Se quiséssemos criar mais de uma conta, bastaria criarmos vários objetos da classe conta com o operador new, dando um nome diferente para cada conta. Cada uma delas teria um valor independente para seus atributos.

Ao salvarmos o código correspondente a uma classe em um arquivo, devemos tomar os seguintes cuidados:

1. Em um único arquivo Java podem existir várias diferentes definições de classes, mas apenas uma delas pode ser pública.

2. O nome do arquivo deve ser o nome da classe pública, observando-se cuidadosamente o uso do mesmo caixa (maiúsculas e minúsculas) tanto para o nome da classe como para o nome do arquivo.

No nosso caso, o programa deve se chamar “Programa.java”.

Exercício 33 - Implemente a classe **Pessoa**, contendo os atributos “nome”, “altura” e “peso”, pelo menos. Crie alguns objetos dessa classe e defina valores diferentes para os atributos de cada um deles.

Exercício 34 - Crie 10 objetos da classe **Pessoa** e insira-os em um *array* de objetos dessa classe. Utilize um for para imprimir todos os atributos dos objetos do *array*.

Exercício 35 - Crie uma classe para um “**Funcionário**”. Ela deve ter o nome do funcionário, o departamento onde trabalha, seu salário (*double*), a data de entrada no banco (*String*), seu RG (*String*) e um valor booleano que indique se o funcionário está na empresa no momento ou se já foi embora.

Exercício 36 - Crie uma classe **Empresa** que possua “nome”, “cnpj”, “qtde_de_funcionario” e um *array* de objetos da classe **Funcionario** (o array pode armazenar até 100 funcionários).

4 – Métodos

Dentro da classe, também declaramos o que cada conta faz e como isso é feito (os comportamentos que cada classe tem, isto é, o que ela faz). As operações que um objeto oferece são chamadas de **métodos**. **Andar**, **Correr** e **Comer**, por exemplo, podem ser métodos da classe **Pessoa**. **Métodos** são comportamentos que os objetos de uma classe podem possuir. A operação de ligar um objeto da classe **Moto**, por exemplo, pode ser feita com o método **ligar**( ).

Por exemplo, de que maneira uma quantia em dinheiro é sacada de uma conta? Especificaremos isso dentro da própria classe **Conta**. É por isso que essas "funções" são chamadas de métodos: pois esta é a maneira (o método) de fazermos alguma operação com um objeto. No código a seguir, adaptaremos a classe **Conta** criada anteriormente para incluir o método "sacar".

```
class Conta {
 int numero;
  /* Deixe aqui os demais atributos que já criamos para a
classe Conta */

  void sacar (double quantidade) {
    this.saldo = this.saldo - quantidade; /* É o mesmo que
fazer this.saldo -= quantidade */
  }
}
```

O método "sacar" recebe como argumento (ou parâmetro) um valor do tipo "*double*" contendo a quantidade de dinheiro a ser sacada da conta do usuário. Sem o argumento, é impossível sabermos o quanto devemos subtrair do saldo da conta. Essa variável chamada "quantidade" pode ser acessada durante todo o método e deixa de existir ao final do método. A palavra "*void*" indica que nenhum valor será retornado por esse método.

A palavra "*this*" (que já estava presente no código da interface que criamos no primeiro capítulo) permite que acessemos o valor de um atributo da própria classe em questão, no nosso caso a classe **Conta**.

A palavra reservada ***this*** é muito importante e útil. Ela referencia a própria instância de dentro de sua classe. Cuidado, ***this*** não pode ser usada em métodos estáticos, como veremos mais adiante.Agora criaremos um método parecido com o anterior, mas para depositar dinheiro na conta.

```
class Conta {
  // Deixe aqui os atributos e métodos anteriores

  void depositar (double quantidade) {
    this.saldo += quantidade;
  }
}
```

O "+=" soma o valor de "quantidade" ao valor antigo do "saldo" e guarda o resultado no próprio "saldo". Para fazer com que um objeto execute algum método, também colocamos um "ponto".

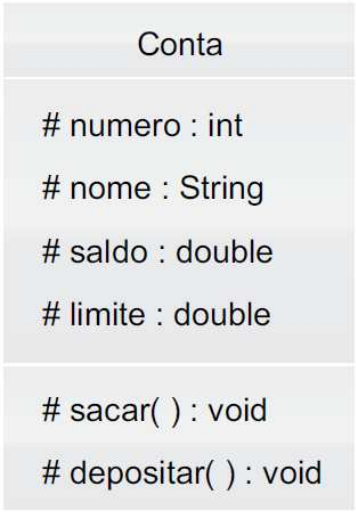


Figura 24 - Atributos e métodos da classe Conta

A Figura 24 - mostra o diagrama da classe Conta com os métodos criados acima:

No código a seguir, alteramos a classe "Programa" para sacarmos dinheiro da conta e depois depositarmos.

```
class Programa {
  public static void main (String[] args) {
    Conta minhaConta;
    minhaConta = new Conta();

    minhaConta.nome = "Hilário";
minhaConta.saldo = 1000.0;

    minhaConta.sacar(200); // Saca 200 reais
    minhaConta.depositar(500); // Deposita 200 reais

    System.out.println("Saldo atual: " +
                          minhaConta.saldo);
}
```

Exercício 37 - Crie os seguintes métodos para a classe **Funcionario**, criada anteriormente:

a) Método "bonificar", que aumenta o salário do funcionário de acordo com o parâmetro passado como argumento.

b) Método "demitir", que não recebe parâmetro algum, apenas modifica o valor booleano indicando que o funcionário não trabalha mais na empresa.

c) Método "mostrarDados", que simplesmente imprime todos os atributos de um funcionário.

Exercício 38 - Crie dois objetos da classe **Funcionario** com os mesmos atributos. Compare os dois com o operador "= =" e veja se o resultado é *true* ou *false*.

Exercício 39 - Insira na classe **Pessoa**, também criada anteriormente, o atributo "idade" e o método "aniversario", que incrementa sua "idade". Crie um objeto desta classe, defina a "idade", chame o método "aniversario" algumas vezes e depois imprima novamente a "idade".

Dicas:

1. Vá compilando seus arquivos à medida que for incluindo classes, variáveis e métodos. Se criar um programa grande e compilá-lo todo de uma vez, pode ficar mais difícil encontrar os vários erros de

digitação que podem aparecer.

2. Por enquanto, você pode colocar várias classes em um mesmo arquivo. Mais adiante, veremos que em alguns casos é importante escrever algumas classes em arquivos separados. Não se preocupe com isso neste momento!

Em todo método devemos especificar o tipo de seu retorno. Nos casos anteriores, utilizamos a palavra "*void*", que significa que não há um valor de retorno do método.

Quando um método possui um valor de retorno, esse valor é devolvido para o código que o chamou. O método "sacar", por exemplo, poderia retornar um valor booleano indicando se a operação foi bem sucedida (o que acontece se a quantidade a ser sacada não for maior que o valor do saldo mais o limite da conta). Veja no exemplo a seguir:

```
boolean sacar (double quantidade) {
  if (quantidade > (this.saldo + this.limite)) {
    return false;
  }
  else {
    this.saldo = this.saldo - quantidade;
    return true;
  }
}
```

Note que agora a declaração do método mudou. O retorno do método não é mais "*void*", e sim "*boolean*". A palavra "*return*" indica que o método será retornado ali, e o valor ao lado dela é que será devolvido como retorno. Abaixo temos um exemplo de como utilizar o valor de retorno da nova função "sacar".

```
minhaConta.saldo = 1000.0;
boolean consegui = minhaConta.sacar(2000);

if (consegui) {
  System.out.println("Operação realizada com sucesso.");
}
else {
  System.out.println("ERRO: Saldo insuficiente.");
}
```

Para simplificar o código, podemos obter o mesmo resultado da seguinte forma:

```
minhaConta.saldo = 1000.0;

if (minhaConta.sacar(2000)) {
  System.out.println("Operação realizada com sucesso.");
}
else {
  System.out.println("ERRO: Saldo insuficiente.");

}
```

Por fim, um método também pode ser criado com estático. Para isso, basta adicionarmos a palavra **static** à declaração do método. Um método estático não nos obriga a instanciarmos um objeto de sua classe para que tenhamos acesso a seus serviços: ele serve apenas para que possamos aproveitá-lo em pequenas computações.

Os métodos estáticos foram criados para resolver situações especiais na orientação a objetos, e também para simplificar certas operações. Imagine que exista uma classe chamada **CPF**, criada por outro programador, que tenha apenas um método chamado **validaCPF** que verifica se um CPF é válido ou não. Segundo a lógica da orientação a objetos, precisaríamos instanciar um objeto desse tipo para utilizarmos seu método para a verificação do CPF. Isso é um tanto quanto incômodo, pois o que necessitamos, neste caso, é simplesmente de uma funcionalidade, e não de um objeto. Da mesma maneira, existem dezenas de funções matemáticas, físicas, estatísticas que precisamos utilizar sem a necessidade de instanciarmos um objeto.

O que queremos é apenas enviar os parâmetros e receber resultados, como se esses métodos fossem funções. Assim, para verificarmos se um CPF é válido, precisaríamos apenas da seguinte instrução:

```
boolean valido = CPF.validaCPF("095.345.672-18");
```

Veja que o método "validaCPF" pode ser utilizado mesmo que não tenhamos criado uma instância da classe CPF. Mas para que um método possa ser criado como estático, ele só pode usar as variáveis da classe que forem estáticas (além das variáveis recebidas como parâmetro pelo método). Uma **variável estática**, declarada em uma determinada classe, é compartilhada por todos os objetos dessa classe e é criada uma única vez durante a execução do programa. Uma **variável de objeto**, ao contrário, é criada a cada vez que um novo objeto dessa classe é criado. Uma variável estática também é definida através da palavra **static**.

Exercício 40 - Por que o código abaixo não irá compilar?

```
class Exercicio75 {
  int x = 37;
  public static void main(String [] args) {
      System.out.println(x);
  }
}
```

A partir da próxima aula, mostraremos como incluir estes novos conceitos na interface criada nas aulas 01 e 02.