# PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 2

Prof. Leonardo Gomes

## CONVERSA INICIAL

Nesta aula, vamos abordar o principal conceito dentro do paradigma de programação orientado a objetos: as classes. As classes dentro de orientação a objetos são compostas principalmente por atributos e métodos, e é por meio delas que geramos os objetos que dão nome ao paradigma. Além da definição dos conceitos, teremos exemplos do uso das classes e objetos com maiores detalhes de como aplicar esses conceitos.

Ao final desta aula, esperamos atingir os seguintes objetivos, que serão avaliados ao longo da disciplina da forma indicada:

Quadro 1 – Objetivos

| **Objetivos** | **Avaliação** |
|---|---|
| 1. Aplicar os conceitos de classes e objetos em linguagem Java. | Questionário e questões dissertativas |
| 2. Desenvolver algoritmos que combinam, agrupam e interagem com diferentes objetos. | Questionário e questões dissertativas |
| 3. Aplicar o conceito de construtor em linguagem Java. | Questionário e questões dissertativas |

## TEMA 1 – CLASSES E ATRIBUTOS

Neste tema, vamos debater os dois conceitos mais importantes da Programação Orientada a Objetos (POO), as classes e objetos.

Quando programamos pensando em POO, o objetivo é modelar o mundo real dentro do contexto que nos interessa. Esse modelo deve ser simples e considerar apenas os elementos que forem relevantes para o problema abordado. Os objetos são os elementos em si que compõem o nosso sistema, enquanto as classes são a descrição desses objetos, e como o nome sugere, classificam um conjunto de objetos que pertençam a um mesmo conjunto.

Por exemplo, suponha que desejamos criar um jogo eletrônico que implemente uma corrida de carros. É pertinente representar todos os diferentes

carros que participam dessa corrida. Cada carro individualmente possui suas características próprias, uma cor, um valor de velocidade máxima, um nível de combustível, uma localização na pista de corrida, entre outros. Cada carro individualmente, portanto, é um **objeto**.

Ainda no exemplo da corrida, observe que todos os carros são representados pelo mesmo conjunto de características, embora possuam valores diferentes para cada variável. E dentro da POO o código que descreve os atributos que todos os carros possuem é chamado de **classe**.

Em resumo, enquanto o termo *carro* em si de maneira global se refere à classe, se falamos do carro vermelho que está liderando a corrida, estamos falando de um carro específico, portanto, de um objeto.

De forma geral cada objeto possui três aspectos principais:

- **Atributos**: São as variáveis que descrevem o objeto.
- **Métodos**: São como funções que dizem o que cada objeto faz.
- **Estado:** Seria o valor de cada atributo que representa aquele objeto específico.

O conceito de registro (ou *struct)* é presente em diversas linguagens de programação e, semelhante ao registro, as classes possuem atributos que caracterizam seus objetos. Na Figura 1, novamente utilizamos o exemplo de uma classe Carro para ilustrar esse conceito. Todos os carros possuem os mesmos atributos, porém, em estados diferentes.

Figura 1 – Representação da classe Carro com os atributos cor, modelo e velocidade com 3 objetos distintos cada um com um estado


Crédito: Colorlife/Shutterstock.

Já os métodos representam o que cada objeto da classe é capaz de fazer, são funções associadas a cada objeto. A classe carro, por exemplo, poderia ter o método *acelerar(int)*, que receberia um valor inteiro e modificaria a velocidade do objeto pelo valor passado por parâmetro.

A chamada do método acelerar com o parâmetro 50 associado ao objeto Sally: *Sally.acelerar(50)* faria o estado do atributo *Velocidade* do objeto Sally ser modificado de *210* para *260*, por exemplo.

## 1.1 Classe em Java

O Java é uma linguagem orientada a objetos, o que significa que todo o código que escrevemos está dentro de alguma classe e as interações entre os dados se dão por meio de métodos e objetos. A convenção entre os programadores Java é a de criar um arquivo separado para cada Classe Java. O arquivo que contém o método main também é uma classe própria geralmente.

Anteriormente, fizemos o nosso primeiro programa (código abaixo) e nele observe a linha public class PrimeiraClasse {}.

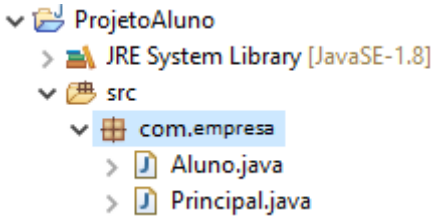
A palavra *public* diz respeito à visibilidade quais outras classes poderão visualizar o código. Estudaremos mais detalhes sobre visibilidade posteriormente, por hora basta entender que é necessário o comando *public*. Na

sequência, temos o comando *class PrimeiraClasse*, que indica que entre chaves está o código de uma classe chamada *PrimeiraClasse*.

```
package com.empresa;

public class PrimeiraClasse {

	public static void main(String[] args) {
		System.out.println("Alo Mamae");
	}

}
```

Como exemplo, vamos fazer um programa Java com duas classes. A classe **Principal** e uma classe **Aluno** que representará quais dados cadastrais possuem os alunos de uma instituição. Você já aprendeu, anteriormente, como fazer um programa básico com uma única classe contendo o método main. Para a primeira classe, dê o nome de Principal e marque a opção no checkbox a opção **public static void main(String[] args)**. Para a segunda classe, certifique-se de que a nova classe fique no mesmo pacote que a classe Principal e dê um nome de Aluno. As demais opções devem ser as padrões do sistema, especialmente a opção **public static void main(String[] args),** só pode existir um único método main no projeto. A estrutura do seu projeto deve ficar como na Figura abaixo.

Figura 2 – Projeto Eclipse contendo duas classes




Na classe Aluno, vamos declarar como atributo dados que desejamos representar. Para uma classe com nome, número de matrícula e CPF, ficaria assim:

```
package com.empresa;
public class Aluno {
	String nome;
	int matricula;
	String cpf;
}
```

De volta à classe Principal, dentro do método main, podemos criar objetos do tipo Aluno, modificar e comparar seus atributos. No código a seguir, declaramos dois alunos (Mario e Luigi) e queremos imprimir o nome do aluno mais antigo, ou seja, aquele que possui a menor matrícula. Confira o código:

```
1. public static void main(String[] args) {
2.       Aluno mario = new Aluno();
3.       mario.cpf="111.111.111-1";
4.       mario.nome="Super Mario";
5.       mario.matricula=1001;
6.
7.       Aluno luigi = new Aluno();
8.       luigi.cpf="222.222.222-2";
9.       luigi.nome="Super Luigi";
10.      luigi.matricula=1002;
11.      Aluno antigo;
12.      if(mario.matricula<luigi.matricula) {
13.           antigo=mario;
14.      }
15.      else {
16.           antigo=luigi;
17.      }
18.      System.out.println("Aluno mais antigo: " +
   antigo.nome);
19. }
```

No POO, um objeto pode ser chamado de *instância de uma classe* e criar uma nova instância, o que é chamado de *instanciação*. Na linha 2 e 7, temos os

objetos sendo instanciados pelo comando ***new***, seguido do nome da classe com abre fecha parênteses, esses parênteses estão presentes, pois se trata de um construtor, que funciona semelhante a um método que especifica detalhes de como o objeto deve ser inicializado. Veremos mais sobre construtores no futuro.

O comando **new** reserva um espaço de memória para aquele objeto. Veja que, na linha 11, uma variável chamada *antigo* do tipo Aluno está sendo criada, porém, sem ser instanciada com o comando **new**. Nesse caso, a variável é apenas um ponteiro e serve para referenciar outros objetos. No caso, ela irá referenciar qual dos dois alunos (Mario ou Luigi) é o que possui a menor matrícula. Na linha 18 antigo.nome irá imprimir o nome *Super Mario*. Se mario.nome fosse alterado para outro valor a alteração passaria a valer para antigo.nome e vice-versa, mario e antigo são referências (ponteiros) para o objeto criado na linha 2. Nas linhas 3, 4, 5, 8, 9 e 10 os atributos de cada objeto estão sendo acessados e modificados.

O código orientado a objeto no caso geral é mais legível que na programação estruturada. Em uma versão estruturada cada aluno deveria ser representando com 3 variáveis diferentes para cada aluno, assumindo um sistema em que os alunos sejam representados por 50 atributos diferentes rapidamente o código ficaria muito difícil de administrar sem uso de classe ou registro/*struct*.

Como desafio, considere o código anterior e modifique-o para termos um terceiro aluno, Yoshi. Descubra qual é o mais antigo dentre os três e imprima todos os dados dele.

## TEMA 2 – MÉTODOS

Neste tema, vamos discutir os métodos que representam as ações que podem desempenhar cada objeto de uma determinada classe.

Mais do que agrupar um conjunto de variáveis, as classes também possuem o que chamamos de métodos, que são equivalentes às funções em programação estruturada, um bloco de código que só é executado quando chamado. Os métodos podem receber dados de entrada (parâmetros) e opcionalmente um valor de retorno. A diferença básica do método em relação à função, é que o método está sempre associado a um objeto e consegue acessar os dados internos do objeto associado.

Suponha a classe Aluno que criamos anteriormente. Dentro da classe, poderíamos criar um método conforme no código abaixo:

```
1. public class Aluno {
2.     String nome;
3.     int matricula;
4.     String cpf;
5.     public void info(){
6.         System.out.println("nome: " + nome);
7.         System.out.println("matricula: " + matricula);
8.         System.out.println("cpf: " + cpf);
9.     }
10.}
```

Na linha 5, criamos o método *info()* que irá imprimir na tela todos os dados referentes ao aluno, os atributos nome, matricula e CPF são acessados de dentro do contexto do método. No entanto, vale o questionamento: qual aluno está tendo os atributos acessados pelo método? Será aquele que está associado ao método? Vamos exemplificar com o código a seguir, supondo o seguinte código na main:

```
1. Aluno mario = new Aluno();
2. mario.cpf="111.111.111-1";
3. mario.nome="Super Mario";
4. mario.matricula=1001;
5.
6. Aluno luigi = new Aluno();
7. luigi.cpf="222.222.222-2";
8. luigi.nome="Super Luigi";
9. luigi.matricula=1002;
10.
11.mario.info();
12.luigi.info();
```

Nas linhas 1 até 9, declaramos os objetos. Na sequência, quando acessamos *mario.info();* o método info() será executado considerando os dados dos atributos do objeto mario, enquanto *luigi.info();* será executado considerando os atributos do objeto luigi.

## 2.1 Exercícios

Desenvolva os exercícios listados aqui e depois compare sua resposta com os códigos apresentados na seção *Apêndice* ao final deste documento.

1. Crie uma classe Nota, com três atributos reais chamados: nota1, nota2, nota3. E crie também dois métodos, cada um para calcular e retornar diferentes tipos de média. *Média Aritmética;* e *Média ponderada* (pesos 2, 3 e 4 respectivamente).
2. Crie uma classe Conta, para administrar contas correntes de um banco com os seguintes atributos:

   String correntista;
   float saldo;
   float limiteSaque;

   E os métodos:

   void sacar( float valor)
   void depositar( float valor)
   void info()

3. Complemente o exercício anterior implementando um método abaixo:

   transfererir(Conta destino, float valor);

   O método deve transferir o valor passado por parâmetro do objeto conta que executa o método para a conta destino passada por parâmetro.

# TEMA 3 – PADRÕES E MODIFICADOR STATIC

Neste tema, vamos discutir dois assuntos importantes dentro da programação Java: os padrões de nomenclatura da linguagem e os usos do modificador *static.*

## 3.1 Padrões de nomenclatura

Embora a linguagem Java em si não imponha um padrão de nomenclatura, aceitando códigos escritos com qualquer estilo, a comunidade de programadores Java adota certos padrões que são amplamente utilizados. Desde as principais e mais elaboradas bibliotecas internas do Java até os

projetos mais simples utilizam os mesmos padrões que apresentaremos aqui facilitando a leitura dos códigos e padronizando códigos desenvolvidos por equipes de diversos programadores.

O padrão principal da linguagem é o Camel Case, que consiste em descrever uma palavra composta ou frase sem dar espaços ou utilizar underline (ou sublinha), mas utilizando letras maiúsculas para indicar a letra inicial da próxima palavra.

Suponha uma variável que representa o nome completo de um usuário poderia ser declarada no código como String ***nomeCompleto*** (letra maiúscula quando começa outra palavra) ao invés de *nome_completo* ou *nomecompleto* dentre outras opções. O nome Camel Case vem da semelhança das letras maiúsculas se sobressaindo no nome com as corcovas nas costas de um camelo. A Figura 2 ilustra a brincadeira com o nome.

Figura 2 – Camel Case, padrão de escrita na programação Java


Crédito: Eric Isselee/Shutterstock.

Dentro do Java, os padrões são os seguintes:

- **Pacotes:** são descritos inteiramente em letras minúsculas. Ex: com.empresa
- **Classes:** inicia com letra maiúscula e segue o Camel Case. Ex: Aluno
- **Métodos, atributos e variáveis:** iniciam com letra minúscula e seguem o Camel Case. Ex: nomeCompleto

- **Constantes:** inteiramente com letras maiúsculas separadas por *underline*. Ex: VALOR_PI

A palavra reservada **static** possui dois usos na linguagem JAVA. Um quando é associada a um método e outro quando é associada a um atributo. Nos dois casos significa que o atributo ou método poderá ser acessado de forma independente de instâncias. Métodos e atributos sempre são relativos a um objeto, porém métodos e atributos estáticos são independentes.

Um atributo estático pode ser entendido como uma variável global da classe, todas as instâncias podem trabalhar sobre a mesma variável. Veja o exemplo a seguir. As variáveis **numeroComum** e **numeroEstatico** se comportam de maneira diferente quando o método incremento for invocado por diferentes objetos. Se dois objetos invocarem o método incremento duas vezes cada um. numeroComum apresentará os valores 0 e 1 duas vezes, enquanto numeroEstatico apresentará os valores 0, 1, 2 e 3.

```
1.  public class Exemplo {
2.      static int numeroEstatico=0;
3.      int numeroComum=0;
4.      public void incremento(){
5.          numero++;
6.          matricula++;
7.          System.out.println("numeroComum: " + numeroComum);
8.          System.out.println("numeroEstatico: " + numeroEstatico);
9.      }
10.}
```

No caso dos métodos estáticos, eles funcionam de forma semelhante a uma função comum do paradigma estruturado. Um bloco de código associado a um nome. Observe que o método **main** que inicia a execução dos códigos Java também possui o modificador **static.**

```
1. public class Aritmetica {
2.      static public int somar(int a, int b){
3.          return a +b;
4.      }
5. }
```

No exemplo anterior, temos uma classe com um método estático, que pode ser invocado por qualquer outra classe sem a necessidade de instanciar, mas colocando nome da classe antes do método com no comando: `resultado = Aritmetica.somar(3,2);`

Esse comando irá chamar o método somar da classe Aritmética passando os valores 3 e 2 como parâmetro e retornando 5 para armazenar em uma variável de nome resultado.

Como desafio experimente implementar os métodos somar, subtrair, multiplicar e dividir de forma estática dentro de uma classe denominada Aritmética. Acesse depois estes métodos a partir de outra classe.

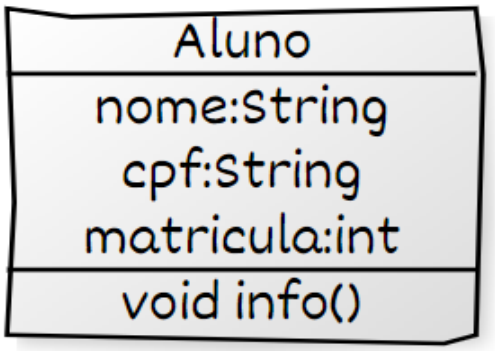
## TEMA 4 – INTERAÇÃO ENTRE OBJETOS

Neste tema, vamos reforçar o conceito de objetos e demonstrar as possibilidades de interação entre eles. A ideia da programação orientada a objetos consiste em mapear o problema considerando os objetos e a relação entre eles.

Uma forma visual e muito prática de representar graficamente classes e suas relações é por meio da *Unified Modeling Language* (UML), em português, Linguagem de Modelagem Unificada. A UML é muito utilizada na documentação de projetos, especialmente de grande escala por sua facilidade de compreensão. É especialmente útil durante a etapa de projeto, pois conseguimos representar os elementos que compõe o domínio do nosso problema sem a necessidade de implementação.

Na UML existem diversos tipos de diagramas que representam diferentes aspectos do projeto de software, entre eles o diagrama de classes que, como o nome sugere, representa justamente as classes. Cada classe é graficamente apresentada com um retângulo contendo geralmente três áreas. Na parte superior, o nome da classe; no meio, os atributos; e na parte inferior, os métodos. Na Figura 3, a classe Aluno é representada.

Figura 3 – Classe Aluno, com atributos métodos descritos




Outra vantagem do UML é a possibilidade de demonstrar a relação entre as classes. Podemos, por exemplo, ter atributos de uma classe sejam objetos de outra classe ou que seus métodos recebam objetos como parâmetros. Suponha, por exemplo, que desejamos criar uma classe que represente uma turma de alunos. Podemos representar a classe turma da seguinte forma.

```
1. package com.empresa;
2. import java.util.ArrayList;
3.
4. public class Turma {
5.       ArrayList<Aluno> listaAlunos = new ArrayList();
6.       Professor orientador;
7.       String sala;
8.
9.       void adicionarAluno(Aluno aluno) {
10.                listaAlunos.add(aluno);
11.          };
12.      }
```

Nas linhas 5 até 7, temos os atributos. Cada turma é composta por uma quantidade variável de alunos portanto o primeiro atributo é um ArrayList, uma classe que implementa arrays com grande facilidade, pois possui diversos métodos para adicionar, remover, contar e buscar elementos de forma dinâmica. Para utilizar essa classe, é necessário importar o arquivo onde está definida, o comando de importação está na linha 2. Dentro do operador <> podemos definir o tipo do ArrayList, que no caso é um ArrayList da classe Aluno, ou seja, todo o

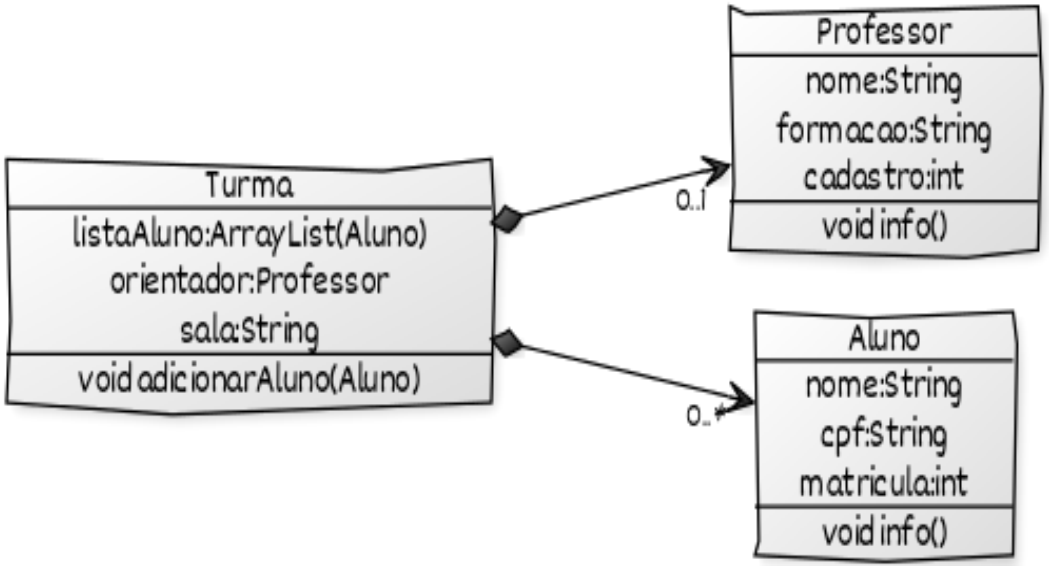
elemento desse array representará um aluno diferente que está matriculado naquela turma, lembrando novamente que o comando *new ArrayList()* gera uma instância nova da classe ArrayList, sem esse comando listaAlunos seria uma referência, um ponteiro. Na linha 6, temos um elemento da classe Professor que representa o orientador daquela turma, também um objeto de uma classe que é um atributo nesta. Suponha que a classe Professor composta por alguns poucos atributos e um método info() análogo ao aluno.

```
1. public class Professor {
2.       String nome;
3.       String formacao;
4.       int cadastro;
5.       void info();
6. }
```

Nas linhas 9 até 11 da classe Turma, temos o método que adiciona um novo aluno no ArrayList que compõem todos os alunos. O método *add* da classe ArrayList adiciona ao final do array, se necessário, o método de forma transparente aloca memória extra para realizar a adição do elemento.

Abaixo, na Figura 4 representamos a relação entre essas classes no diagrama de classes UML. No caso Turma, possui uma relação que chamamos de composição com relação às duas outras classes. O paradigma orientado a objetos tenta representar o domínio dos problemas de forma mais natural e semelhante ao problema real. Portanto, assim como na vida real, as turmas são compostas por um grupo de alunos da mesma forma a relação que entre elas é chamada de composição. Na Figura a indicação **0..1** significa que cada turma possui de zero até um professor e **0..*** significa que a turma possui de zero até vários alunos. Mais detalhes sobre o digrama de classes UML serão apresentados na nossa disciplina em um momento oportuno.

Figura 4 – Classes Aluno, Professor e Turma representadas




## TEMA 5 – CONSTRUTORES

No momento que instanciamos um objeto no Java internamente um bloco de código é executado, esse bloco de código é denominado **construtor**. Neste Tema vamos discorrer sobre como criar e utilizar os construtores.

No bloco de código de um construtor qualquer código pode ser escrito, mas usualmente utilizamos para carregar alguma informação, especialmente informações que sejam cruciais para o funcionamento do objeto no momento da sua criação.

Os construtores são criados de forma semelhante aos métodos, porém, devem possuir o mesmo nome da classe e não possuem valor de retorno. É possível também possuir diversos construtores para uma mesma classe variando apenas os parâmetros de entrada. Abaixo segue um código que exemplifica isso:

```
1. public class Aluno {
2.       String nome;
3.       int matricula;
4.       String cpf;
5.
6.       Aluno(String nome, int matricula, String cpf){
7.             this.nome = nome;
8.             this.matricula = matricula;
9.             this.cpf = cpf;
10.        }
11.        Aluno(String nome){
12.              this.nome = nome;
13.        }
14.        Aluno(){
15.              System.out.println("Construtor sem
   parâmetro");
16.        }
17.     }
```

Mais uma vez exemplificamos utilizando a classe Aluno. Dessa vez criamos três construtores diferentes, linhas 6, 11 e 14. No primeiro construtor, temos três parâmetros sendo passados, ou seja, é permitido instanciar o objeto passando o nome, a matrícula e o CPF. Conforme vemos nas linhas 7, 8 e 9, os parâmetros passados para os atributos da classe Aluno. A palavra reservada **this** serve para distinguir os atributos dos parâmetros, visto que ambos possuem os mesmos nomes.

**Aluno  mario = new Aluno(“Super Mario”, 1001, "111.111.111-1" );**

Ao instanciar aluno desta maneira, ele já terá valores em todos os seus atributos sem necessidade de indicar um a um individualmente.

**Aluno  luigi = new Aluno(“Super Luigi”);**

Por termos criado três construtores, temos três opções de instanciação. Esta instanciação por ter apenas um parâmetro String invoca o construtor da linha 11 que também possui apenas um parâmetro String e atribui um valor inicial

para o nome baseado no parâmetro do construtor, os demais atributos são ignorados. Note que os atributos são o que distingue diferentes construtores.

```
Aluno  yoshi = new Aluno();
```

Se instanciamos um objeto sem parâmetros, o construtor sem parâmetros é chamado, aquele presente na linha 14 irá imprimir uma mensagem. Quando criamos uma classe, um construtor vazio implícito que não executa nenhum código é criado. No entanto, a partir do momento que criamos um construtor qualquer, esse construtor vazio implícito deixa de existir. Para testar isso, observe que se apagarmos apenas o construtor da linha 14, essa instanciação vazia não vai funcionar, porém se apagarmos todos os construtores ele volta a funcionar novamente.

### 5.1 Exercício

4. Crie uma classe Horario com os atributos inteiros, hora, minuto e segundo. Crie três construtores, um que recebe três parâmetros e inicia os três atributos com os valores passados, outro que recebe apenas a hora e outro vazio. Caso algum dos valores inicializados não esteja no intervalo adequado (hora entre 0 e 23, minutos e segundos entre 0 e 59), inicialize o valor em questão em zero e emita uma mensagem de erro.

## FINALIZANDO

Nesta aula, iniciamos propriamente dito o conteúdo relativo à orientação a objetos. Vimos o conceito de classe e objeto que é o mais importante dentro desse Paradigma.

Também trabalhamos com métodos, o modificador static e vimos o conceito de construtores. Com o conteúdo desta aula, o(a) aluno(a) dá um passo importante na compreensão da orientação a objetos. Nessa fundamentação, o aluno tem a base para os conceitos mais aprofundados que seguem posteriormente.

Em outra oportunidade, exploraremos os conceitos de visibilidade e encapsulamento, importantes para diminuir inconsistências nos códigos orientados a objeto.

## APÊNDICE

1. Crie uma classe Nota, com três atributos reais chamados: nota1, nota2, nota3. E crie também dois métodos, cada um para calcular e retornar diferentes tipos de média. *Média Aritmética;* e *Média ponderada* (pesos 2 3 4).

```
1. public class Media {
2.       float n1,n2,n3;
3.
4.       float aritmetica() {
5.
6.            return (n1 + n2 + n3)/3;
7.       }
8.       float ponderada() {
9.            //na media ponderada cada nota é multiplicada
   pelo peso e depois dividia pela soma dos pesos.
10.               return (n1*2 + n2*3 + n3*4)/9;
11.          }
12.      }
```

E na classe com o método main para testar:

```
1. public static void main(String[] args) {
2.       Media mario = new Media();
3.       //letra f garante que o número será interpretado
   como float
4.       mario.n1=9f;
5.       mario.n2=7f;
6.       mario.n3=8.5f;
7.
8.       float resultado = mario.aritmetica();
9.       System.out.println("Aritmetica: " + resultado);
10.           resultado = mario.ponderada();
11.           System.out.println("Ponderada: " + resultado);
12.      }
```

2. Crie uma classe Conta, para administrar contas correntes de um banco com os atributos abaixo.
3. Complemente o exercício anterior implementando um método abaixo. (Código responde os exercícios 2 e 3)

transfererir(Conta destino, float valor);

```
String correntista;
float saldo, limiteSaque;

E os métodos:

void sacar( float valor)
void depositar( float valor)
void info()
```

```
1. public class Conta {
2.       String correntista;
3.       float saldo;
4.       float limiteSaque;
5.       void sacar(float valor) {
6.         // verificação do saque
7.         if (valor > saldo || valor > limiteSaque) {
8.            System.out.println("Saque não permitido");
9.         }
10.              saldo -= valor;
11.       }
12.
13.       void depositar(float valor) {
14.              saldo += valor;
15.       }
16.
17.       void info() {
18.              System.out.println("Nome: " + correntista);
```

```
19.             System.out.println("Saldo: " + saldo);
20.             System.out.println("Limite: " +
    limiteSaque);
21.         }
22.
23.         void transferir(Conta outra, float valor) {
24.             if (valor > saldo || valor > limiteSaque) {
25.                 System.out.println("Transferencia não
    permitida");
26.
27.             }
28.             saldo -= valor;
29.             outra.saldo += valor;
30.          }
31.      }
```

**Na main para testar o código:**

```
1.    Conta mario = new Conta();
2.    Conta luigi = new Conta();
3.
4.    mario.correntista = "Super Mario";
5.    mario.saldo = 1000;
6.    mario.limiteSaque = 200;
7.
8.    luigi.correntista = "Super Luigi";
9.    luigi.saldo = 2000;
10.     luigi.limiteSaque = 200;
11.
12.     mario.transferir(luigi, 150);
13.     mario.info();
14.     luigi.info();
```

O método deve transferir o valor passado por parâmetro do objeto conta que executa o método para a conta destino passada por parâmetro.

4. Crie uma classe Horario com os atributos inteiros, hora, minuto e segundo. Crie três construtores, um que recebe três parâmetros e inicia os três atributos com os valores passados. Outro que recebe apenas a hora e outro vazio. Caso algum dos valores inicializados não esteja no intervalo adequado (hora entre 0 e 23, minutos e segundos entre 0 e 59), inicialize o valor em questão em zero e emita uma mensagem de erro.

```
1. public class Horario {
2.     int hora, minuto, segundo;
3.
4.     public Horario(int hora, int minuto, int segundo) {
5.         this.hora = hora;
6.         this.minuto = minuto;
7.         this.segundo = segundo;
8.     }
9.
10.      public Horario(int hora) {
11.             this.hora = hora;
12.      }
13.
14.      public Horario() {
15.      }
16.     }
```

## REFERÊNCIAS


BARNES, D. J.; KÖLLING, M. **Programação orientada a objetos com Java**. 4. ed. São Paulo: Pearson Prentice Hall, 2009.

DEITEL, P.; DEITEL, H. **Java Como programar**. 10. ed. São Paulo: Pearson, 2017.

LARMAN, C. **Utilizando UML e Padrões**: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo. 3. ed. Porto Alegre: Bookman, 2007.

MEDEIROS, E. S. de. **Desenvolvendo software com UML 2.0**: definitivo. São Paulo: Pearson Makron Books, 2004.

PAGE-JONES, M. Fundamentos do desenho orientado a objetos com UML. São Paulo: Makron Book, 2001.

PFLEEGER, S. L. **Engenharia de software**: teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

SINTES, T. **Aprenda programação orientada a objetos em 21 dias**. 5. reimp. São Paulo: Pearson Education do Brasil, 2014.

SOMMERVILLE, I. Engenharia de software. 9. ed. São Paulo: Pearson, 2011.