

Paradigma orientado a objetos conceitos iniciais

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Programação Orientada a Objetos e Java

- A **programação orientada a objetos** surgiu como um estilo popular de programação no qual o refinamento de objetos se tornou uma preocupação central.
- Nos próximos slides, usaremos **Java** para explicar as principais características de uma linguagem orientada a objetos. O propósito **não é explicar Java**, mas, sim, identificar as principais ideias no paradigma orientadas a objetos, como **encapsulamento**, **polimorfismo** e **herança**.
- Para maiores informações sobre o Java acesse:
<https://www.caelum.com.br/apostila-java-orientacao-objetos/#null>

Definição de Linguagem Orientada a Objetos

- Um conceito importante na programação imperativa foi utilização da **abstração procedural** (funções) no desenvolvimento de programas.
- Na programação orientada a objetos tem se inicio com a **abstração de dados** ou **tipos abstratos de dados (TAD)**. A abstração de dados estende a noção de tipo, proporcionando o programador um mecanismo de **encapsulamento** para definir **novos tipos** de dados.
- O **encapsulamento** permite que **constantes logicamente relacionadas** (constantes, variáveis, funções) sejam agrupadas em uma **entidade** (classe).

Desenvolvendo uma calculadora de fração

Em matemática, uma **fração** é um modo de expressar uma **quantidade** a partir de uma **razão** de dois **números inteiros**. De modo simples, pode-se dizer que **uma fração**, representada de modo genérico como **a/b** , designa o inteiro **a** **dividido em b** partes iguais. Neste caso, **a** corresponde ao **numerador**, enquanto **b** corresponde ao **denominador**, que não pode ser igual a zero.

Operações entre frações:

Soma de fração: $(a/b) + (c/d) = (a \cdot d + c \cdot b) / b \cdot d$

Multiplicação de fração: $(a/b) * (c/d) = (a \cdot c) / (b \cdot d)$

Divisão de fração: $(a/b) / (c/d) = (a/b) \cdot (d/c) = (a \cdot d) / (b \cdot c)$

Igualdade: $(a/b) == (c/d)$ se $a \cdot d == b \cdot c$

Desenvolvendo uma calculadora de fração

- Já vimos como o código em linguagem imperativa, e se usarmos o paradigma orientado a objetos com a linguagem em Java, poderíamos ter as seguintes questões:
 - Como representar uma fração em uma linguagem orientados a objetos?
 - Que informações precisamos armazenar?
 - Como implementar as operações definidas para frações?
 - Como testar nossa implementação?

Classes

- Uma classe é uma declaração de tipo que encapsula constantes, variáveis e funções para manipulação dessas variáveis. Podemos dizer que **classe** é um **tipo abstrato de dados**.
- Na terminologia de classes em POO temos **variáveis locais** são chamadas de **variáveis de instância (atributos)**, suas inicializações são obtidas por chamadas a um **construtor** da classe, e a finalização por chamada ao **destrutor**, e as **funções** são considerados **métodos**.
- Cada **instância** de uma classe é um **objeto**.
- Um **cliente** de uma **classe** é qualquer outra classe ou método que declara ou instancia um objeto da classe.

Estrutura básica de uma classe em Java

```
public class <NomeDaClasse>
{
    // variáveis de instancia;
    private      <tipo> <atributos>
    //definição de construtores...
    public <NomeDaClasse>( <parâmetros> )
    {
        // corpo do construtor
    }
    // métodos...
    public <tipoDeRetorno>
        <nomeDoMétodo>(<parâmetros>)
    {
        //corpo do método
    }
    //outros métodos e construtores .....
}
```

Estrutura básica de uma classe em Java

- Os **objetos** são **criados** a partir de **classes**. Assim, uma **classe** é um **conjunto de objetos** que têm **os mesmos atributos e comportamentos**.
- Toda classe precisa ter um **nome**, um **conjunto de atributos (ou campos)**, **construtores** e **métodos**.
- Os atributos, construtores e métodos devem ficar **encapsulados** na classe. O **encapsulamento**, além de agrupar tudo numa só estrutura, também permite restringir o acesso aos elementos.

Construtores

- O processo de criação de um objeto a partir de sua classe é chamado **instanciação**. No processo de **instanciação**, alocamos **espaço em memória** com o operador **new** e inicializamos este espaço de memória com os parâmetros passados ao processo de instanciação.

NomeDaClasse **objeto** = **new** NomeDaClasse(parâmetros);

- Observe que o operador **new** não aparece sozinho, ele necessita de um bloco de código com mesmo nome da classe denominado **construtor**. Um **construtor** pode parecer, mas **não é** um método, já que **não possui retorno** e só é chamado durante a **construção do objeto**.

Atributos

- Um **atributo (ou variáveis de instância)** de uma classe serve para identificar uma característica comum dos objetos de uma classe e representam a **parte estrutural** (dados) da classe.
- Os valores atribuídos aos atributos pelos construtores, definem o **estado do objeto**.
- O estado de qualquer objeto pode ser alterado modificando-se as suas variáveis de instância.
- Os atributos também são encapsulados, existem métodos para alterar valor de um atributo **set** (acompanhado pelo **nome**) e consultar valor de um atributo **get** (acompanhado pelo nome do atributo), esse métodos são conhecidos por **setters** e **getters** de objetos.

Apontador de auto-referência **this**

- Quando criamos um objeto, a **variável associada ao objeto** (variável de referência do objeto) permite acessar todos os seus **métodos** declarados como **public**.
- Em Java o apontador de auto-referência **this** permite diferenciar dentro de um método os parâmetros dos atributo da classe, tornando explícita a referência aos atributos caso eles tenham o mesmo que os parâmetros.

Métodos

- Os métodos definem o **comportamento** dos objetos, ou seja, representam a parte **comportamental** (ou **dinâmica**) de uma classe.
- Os parâmetros de funções vistos no paradigma imperativo são passados de **forma explícita**, no POO o objeto que invoca o método também pode ser considerado um parâmetro no método, e nesse caso temos o objeto como **parâmetro implícito**.
- A declaração de um método segue os mesmos princípios das funções no paradigma imperativo, porém agora com os modificadores de acesso (**public**, **private**,...).

Modificadores de acesso

- O ocultamento de informação (encapsulamento) é tornado explícito requerendo que todo **método** e todo **atributo** em uma classe tenham um nível particular de visibilidade – **public**, **protected** ou **private** – com relação as suas subclasses e as classes cliente.
 - Uma variável ou um método **public** é visível a qualquer cliente e subclasse de uma classe.
 - Uma variável ou um método **protected** é visível somente para uma subclasse da classe.
 - Uma variável ou um método **private** é visível à classe corrente, mas não às suas subclasses ou aos seus clientes.
- Uma boa prática no projeto de software, é tornar **privadas** as **variáveis de instância** e permitir o acesso somente por meio do uso de **métodos públicos** ou **protegidos**.

Exercício

Suponha que teremos que desenvolver um sistema de computação gráfica e queremos começar modelando uma coordenada (x,y) , ou seja um ponto na tela.

Como estamos programando orientado a objetos, os pontos do sistema de computação gráfica devem ser representados por uma classe, basicamente as informações que a classe precisa armazenar são coordenadas x e y de um “ponto” na tela, ou seja, x e y são atributos da classe Ponto.

Implemente também um construtor para instanciar objetos da classe Pontos, o construtor ficará responsável em inicializar os atributos do objeto, lembrando que um construtor em Java deve ser implementado com o mesmo nome da classe.

Implemente também o método **igual(Ponto p)** na classe Ponto que verifica se dois pontos são iguais, o método recebe um parâmetro de forma **explícita** (Ponto p) e outro de **forma implícita**, ou seja, o objeto em que você invoca o método também é considerado um parâmetro da chamada do método. Implemente também o método **distancia()** que calcula e retorna a distância entre dois objetos. Por fim implemente um **cliente** para testar a classe Ponto.

Bons estudos !