

Programação Descomplicada



Orientação a Objeto

Linguagem Java

2017

V2.0

Índice

Programação Descomplicada	1
Orientação a Objeto	1
Linguagem Java	1
Programação Orientada a Objetos	3
Conceito Inicial	3
Definição de Objeto	4
Metodologia de Programação	5
Declaração de Classe	6
Convenção de Nomes	7
Declaração de Atributos	8
Encapsulamento	10
Setters e Getters	10
Herança de Classes	12
Polimorfismo	14
Sobre Carga de Métodos	14
Interface	16
Introdução	16
Utilizando interface em Java	16
Créditos	18

Programação Orientada a Objetos

Quando falamos de programação orientada a objetos no encontramos com um conceito básico, muitas pessoas se confundem quando falamos do mesmo conceito, então trouxe a explicação de forma mais objetiva possível.

A POO consiste em um só conceito e esse conceito é trazer objetos do mundo real para o virtual, também veremos um pouco da distribuição e ação das classes sobre a memória, vamos entender melhor no conteúdo a seguir.

Conceito Inicial



Segundo o conceito de orientação a objeto, qualquer coisa do mundo real pode ser considerado um objeto, por exemplo: uma lâmpada, caneta, carro, pessoa e etc. Segundo o conceito deve-se atribuir ou transpassar um objeto do mundo real para o chamado mundo virtual.

Distribuição de Memória

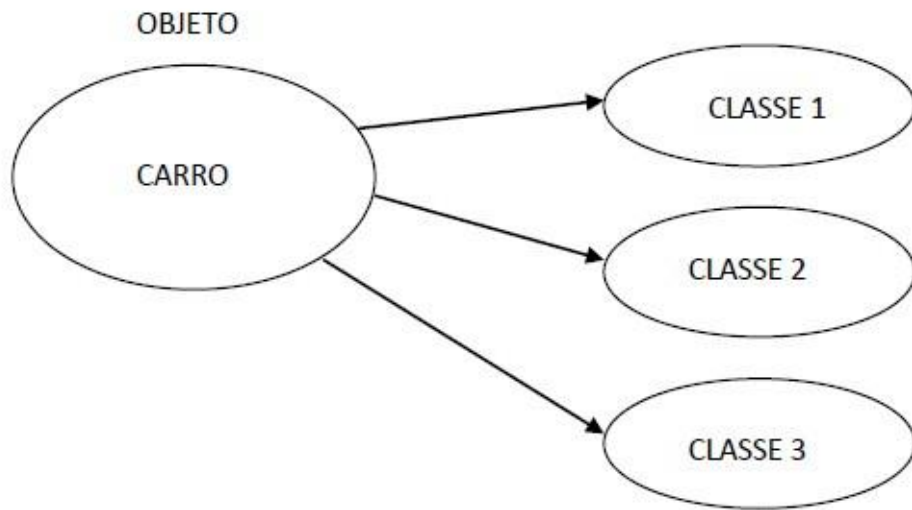
A utilização de memória com a OO se divide em duas partes onde temos:

Memória curta (RAM) = Objeto e a visão dinâmica

Memória Longa (HD) = Classes e a visão estática

Quando trabalhamos com objetos estamos trabalhando com memória curta, porém quando trabalhamos com classes estamos utilizando a memória longa que necessita de armazenamento de métodos e atributos.

Uma classe pode gerar a vários objetos dentro da OO, por exemplo:



Nesse exemplo acima podemos verificar que um único objeto representado pelo objeto carro gera três classes.

Definição de Objeto

Um objeto é definido ou trazido do mundo real para o virtual através de características e funcionalidades. Podemos ter uma predefinição de características e funcionalidades.

Característica = pode ser definida como aparência de um objeto.

Funcionalidade = pode ser definida como ação de um objeto.

Carro	
Características	Funcionalidades
Marca	Ação
Modelo	
Cor	
Motor	

Carro	
Características	Funcionalidades
Ford	Transportar
EcoSport	
Preto	
1.6	

Na orientação a objeto quando nos referimos a objeto e as suas características e funcionalidades também damos uma definição específica através da convenção de nomes.

Características são chamadas de **Atributos**.

Funcionalidades são chamadas de **Método**.

Um dos maiores problemas na orientação a objeto é a abstração do mesmo, neste momento podemos encontrar alguns tópicos como:

Complexidade (Quanto mais complexo mais difícil se torna).

Contexto

Escopo

Nível de Abstração (Deparamos com a dificuldade de abstração, alguns objetos tem problemas).

Metodologia de Programação

Na orientação a objeto nos deparamos com duas classes que denominamos como:

1º Classe = Classe Modelo

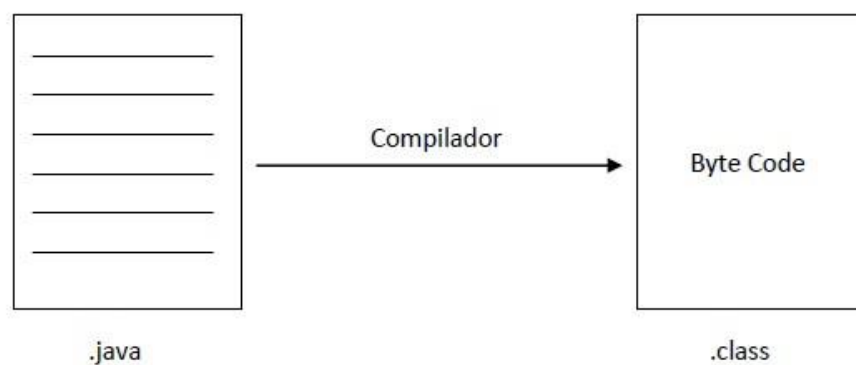
2º Classe = Classe Executável

Na classe modelo guardamos a descrição do objeto "Receita do bolo". Na verdade nesta classe nos atribuímos todos os atributos e criamos os métodos.

Na classe executável definimos quais e quantos objetos serão criados em memória e quais testes serão feitos. Na verdade nessa classe nos chamamos os métodos e passamos os atributos através dos métodos set e get criados na classe modelo.

Quando criamos uma classe seja modelo ou executável a extensão do arquivo fica como ".java", após a compilação o arquivo (classe) .java virá um arquivo bytecode com a extensão ".class".

Exemplo de conversão:



Devemos recordar que quando estamos falando de objeto é claro que vale lembrar que todo objeto é dinâmico.

Declaração de Classe

Quando declaramos uma classe sua estrutura é definida da seguinte forma:

```
public class [identificador]{  
  
....  
  
}
```

Posteriormente entra a declaração de atributos, métodos e construtores.

```
public class [identificador]{
```

Declaração de Atributos

Declaração de Métodos

Declaração de Construtores

```
}
```

Mas antes de declararmos a classe, devemos entender como funciona a convenção de nomes do Java. Você deve estar me perguntando convenção?

Convenção de Nomes

Sim convenção, na verdade isso é um padrão de modos de escritas que devem ser seguidos dentro do Java. Por exemplo:

Classe

Os nomes das classes devem ser escritos sem caracteres brasileiros como ç, acentos, "@#!" e principalmente espaço.

A primeira linha deve ser maiúscula e sem espaço, não pode começar com número, nome com a primeira letra em maiúscula e o resto minúsculo, exceto se o nome for composto.

Exemplo:

Pessoa (Nome Comum)

PessoaFisica (Nome Composto)

Atributos

No caso dos atributos todos os nomes devem ser criados em minúscula, porém se o nome for composto, os demais seguiram com as iniciais em maiúscula.

Exemplo:

cor (Nome Comum)

corDosOlhos (Nome Composto)

Métodos

No caso dos métodos todos os nomes devem ser criados em minúscula, porém se o nome for composto, os demais seguiram com as iniciais em maiúscula.

Exemplo: **correr()**

correrNaMontanha()

Vale lembrar que os métodos possuem (), que podem a qualquer momento pode receber ou não parâmetros para a sua execução.

Declaração de Atributos

Para declararmos atributos dentro de uma classe é preciso seguir alguns padrões, por exemplo, é necessário descrever a:

[Visibilidade] [Tipo de Dado] [Identificador]

Visibilidade

No Java nós temos alguns tipos de visibilidade, são elas:

+ Public

- Private

~ Package

Protected

+Public

Atributos declarados com essa visibilidade tornam o mesmo visível a todas as classes criadas no projeto.

-Private

Atributos declarados com essa visibilidade tornam o mesmo invisível as demais classes do projeto, se tornando apenas visível na classe que se encontra.

~Package

Atributos declarados com essa visibilidade tornam o visível a todas as classes que estão naquele pacote.

#Protected

Atributos declarados com essa visibilidade se tornam apenas acessíveis a classe pai, pois envolve o conceito de herança de classes que veremos mais a frente.

Tipos de Dados

Já no tipo de dados, temos o que é definido como dados primitivos, que são:

Números Inteiros	Byte Short Int Long
Reais	Float Double
Lógica	Boolean
Caractere	Char String

No caso dos tipos de dados caracteres vale ressaltar que o char é um tipo de dado limitado, por ter só 1 byte de tamanho ele comporta apenas um caractere em sua estrutura. Já o tipo de dado **String** é uma cadeia de caracteres que suporta inúmeros valores em sua estrutura.

Identificador

O identificador é na verdade um nome que se dá ao atributo para que ele seja localizado e posteriormente atribuído valores pelos métodos.

Como exemplo a definição de um atributo ficaria assim:

```
public byte idade;  
public short ano;  
public char letra;  
public String nome;
```

Encapsulamento

Quando trabalhamos com alguns contextos da orientação a objeto é impossível não citar o encapsulamento, o encapsulamento na verdade é uma técnica aplicada a um método que serve para ocultar códigos, ações específicas criadas para o sistema.

Esse conceito é muito aplicado nos métodos chamados de **SETTERS** e **GETTERS** que agem de forma imprescindível, pois compõe um dos fundamentos do Java.

Setters e Getters

Os citados setters e getters são métodos que encadearão e constituirão todo o processo de desenvolvimento de qualquer sistema criado utilizando a linguagem Java. A seguir conheceremos um pouco dos dois.

O método Setter é responsável por atribuir, inserir, guardar valores em um atributo, sua estrutura é constituída da seguinte maneira:

```
public void setNome(String nome){  
  
    this.nome = nome;  
  
}
```

No exemplo acima nós temos um método que irá atribuir um nome a um respectivo atributo nome. O funcionamento do método é simples um nome será atribuído pelo usuário, esse nome é atribuído no campo nome que se encontra entre parênteses, todo conteúdo definido dentro desses parênteses recebe o nome de parâmetros (parâmetros são códigos complementares que irão compor a estrutura de um método e que também irão contribuir com a execução do código definido dentro da estrutura do mesmo) esse parâmetro nome é atribuído e recebido pelo atributo onde digitamos o código this que define o nome do this.nome sendo o atributo.

De uma forma mais simples seria assim:

```
public void setNome(Parâmetro){  
  
    this.Atributo = Parâmetro;  
  
}
```

O método Get é responsável por pegar, buscar, trazer valores de um atributo que foi setado pelo método set, sua estrutura é constituída da seguinte maneira:

```
public String getNome( ){  
  
    return nome;  
  
}
```

No exemplo acima nós temos um método que irá buscar um nome de um respectivo atributo nome. O funcionamento do método é simples um nome será atribuído pelo método setNome que por sua vez é atribuído pelo usuário e o método get irá trazer o valor deste atribuído setado pelo método set. O comando que permite o retorno do valor é o return

presente dentro do método get, esses métodos são simples e não necessitam de parâmetros para sua execução.

Vale lembrar que não se deve utilizar o método get se ainda não foi atribuído valor a um atributo, pois ele pode retornar o valor 0 para numérico e null para caractere seguindo pressuposto que todo atributo criado não contém nada em sua estrutura, ou seja, todo atributo criado vale zero.

Duas coisas diferem esses métodos, a primeira são os tipos de dados. Os métodos do tipo set contém em sua estrutura o tipo de dados “void”, que recebe a denominação e representação de tipo de dado nulo, vazio. Neste caso ele não irá tornar o seu método vazio, mas definirá que o código contido dentro dele será executado e finalizado sem retorno, armazenamento de nenhum valor.

Já no método get temos os tipos de dados definidos pelo atributo, pois se o atributo for do tipo String o método deve ser do tipo String, se for int deve ser int e assim sucessivamente. Neste caso ele armazenará, retornará um valor e o mesmo será um tipo de dado primitivo.

A segunda diferença é que o método set contém parâmetros e o método get não, ou seja, ambos executarão funções fundamentais, porém de jeitos diferentes.

Herança de Classes

Herança é um termo utilizado para algo que geralmente é um bem material que será deixado para alguém (Familiar, Filhos) obterem após a vida terrena do proprietário se findar. Mas temos o termo herança de classes no Java e na OO.

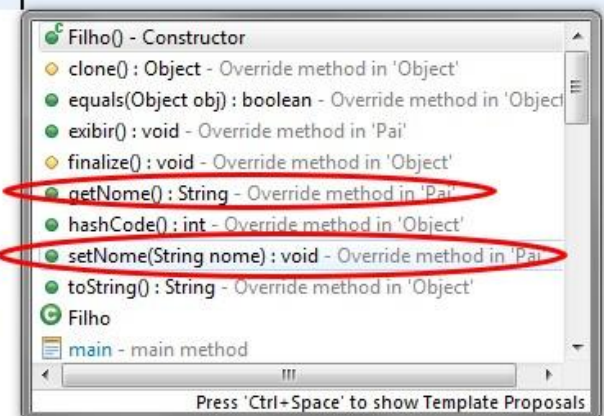
Esse termo é usado quando uma classe herda atributos e métodos do pai, para entendermos isso é necessário o aprofundamento do assunto que irá ocorrer a seguir. Suponhamos que exista uma classe, para essa classe serão criadas mais três classes e essas três classes desejam utilizar os atributos e métodos da classe principal, para tal é necessário defini-las como herdeiras da classe principal. Um dos conceitos do Java e POO é a reutilização dos códigos e a herança é um exemplo.

Neste caso sabemos que a classe pai possui seus próprios atributos e métodos e para as três classes se tornarem subclasses da classe principal é necessário à adição do código “extends”, abaixo temos um exemplo de uma classe comum denominada “Pai”.

```
Pai.java
1 public class Pai {
2
3     public String nome;
4
5
6
7
8     public String getNome() {
9         return nome;
10    }
11
12
13    public void setNome(String nome) {
14        this.nome = nome;
15    }
16
17
18    public void exibir(){
19        System.out.println(" O nome é "+this.nome);
20    }
21
22
23
24 }
25
26
27 }
28
```

Na classe abaixo temos o exemplo de herança onde a classe filho recebe o termo `extends` e em seguida recebe a descrição do pai, que no nosso caso será a própria classe de nome pai.

```
*Filho.java
1
2 public class Filho extends Pai {
3
4     public String parentesco;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 }
20
```



Podemos notar que a classe acima que recebeu a herança agora consegue observar os métodos da classe Pai. Isso ocorre devido à instância da classe ter sido criada.

Polimorfismo

Polimorfismo é a técnica de reuso de códigos, na verdade podemos definir este tipo de técnica como uma sobre escrita de método. O Polimorfismo permite que eu crie em uma classe pai um método fixo, por exemplo, suponhamos que eu crie o método `exibir` na classe pai e crie o método `exibir` na classe filho com o mesmo nome, geralmente se for feito isso em duas classes o compilador ou o próprio programa acusará erro, mas quando se usa a herança de classe a mesma é permitida, sendo tratada então como polimorfismo.

Existem outras formas de polimorfismo como polimorfismo paramétrico, polimorfismo de inclusão, sobrecarga e coerção.

Sobre Carga de Métodos

Sobre Carga é quando criamos um mesmo método várias vezes dentro de uma mesma classe, desde que eu mude a estrutura do mesmo para que eu possa reutilizar o mesmo método com tipos de dados diferentes, exemplo:

```
public int somar(int x){  
  
    int x = 2;  
    int y = 2;  
    int resultado = x+y;  
  
    return resultado;  
  
}
```

```
public float somar(float x){  
  
    float y = 2;  
    float resultado = x+y;  
  
    return resultado;  
  
}
```

No momento da chamada o Java reconhece o método pelo valor passado a ele pelo parâmetro, por exemplo:

Se eu chamar assim:

```
somar(2);
```

Ele saberá que é o método int, já se for assim:

```
Somar(2.2);
```

Ele saberá que é o método float.

Interface

Introdução

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para “obrigar” a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

Utilizando interface em Java

Dentro das interfaces existem somente assinaturas de métodos e propriedades, cabendo à classe que a utilizará realizar a implementação das assinaturas, dando comportamentos práticos aos métodos.

Abaixo é possível ver um exemplo de uma interface chamada *FiguraGeometrica* com três assinaturas de métodos que virão a ser implementados pelas classes referentes às figuras geométricas.

Listagem 1: Interface *FiguraGeometrica*

```
public interface FiguraGeometrica
{
    public String getNomeFigura();
    public int getArea();
    public int getPerimetro();
}
```

Para realizar a chamada/referência a uma interface por uma determinada classe, é necessário adicionar a palavra-chave `implements` ao final da assinatura da classe que irá implementar a interface escolhida.

Sintaxe:

```
public class nome_classe implements nome_interface
```

Onde:

nome_classe – Nome da classe a ser implementada.

nome_Interface – Nome da interface a se implementada pela classe.

Fonte: <http://www.devmedia.com.br/entendendo-interfaces-em-java/25502>

Créditos



Professor Edson Lael

Desenvolvedor Especialista na Linguagem Java

Desenvolvedor Especialista em Orientação a Objetos e Eventos

Desenvolvedor PHP

Desenvolvedor Especialista em Banco de Dados Oracle e Mysql

Formado em ADS e Pós Graduado em Desenvolvimento de Sistemas Web