**Pesquisa**

**Estrutura de Dados**



**Athauan Marques da Cunha – 136246-1**

**Orientação ao Objeto**

*Primeiramente antes de começar a fala sobre OO (orientação ao objetos), vamos enteder quando e como surgiu os conceitos e a suas vantagens no desenvolvimento de softwares.*

**Um pouco sobre a história da orientação ao objeto**

A **orientação a objetos** tem sua origem nos anos 60 na Noruega, com Kristen Nygaard e Ole-Johan Dahl, no Centro Norueguês de Computação. Através da linguagem Simula 67, foram introduzidos os conceitos de classe e herança. A orientação a objetos foi mais bem conceituada no laboratório da Xerox, em Palo Alto, sendo refinada numa seqüência de protótipos da linguagem Smalltalk. O líder desse projeto foi Alan Curtis Kay, considerado um dos criadores do termo “programação orientada a objetos”;

**Conceitos e principios**

Formalmente, para ser considerada uma linguagem OO, precisa implementar quatro conceitos básicos: abstração, encapsulamento, herança e polimorfismo.

Os princípios da orientação a objetos:

- Qualquer coisa é um objeto.

- Objetos realizam tarefas através da requisição de serviços.

- Cada objeto pertence a uma determinada classe.

- Uma classe agrupa objetos similares.

- Um classe possui comportamentos associados ao objeto.

- Classes são organizadas em hierarquias.

**Sobre Java e sua aplicação na Orientação Objeto**

O Java é ao mesmo tempo um ambiente e uma linguagem de programação desenvolvida pela Sun Microsystems Inc.  
Trata-se de mais um representante da geração de linguagens orientadas a objetos e foi projetado para resolver os problemas da área de programação cliente/servidor.  
Os aplicativos em Java são compilados em um código de bytes (“bytecodes”) independente de arquitetura.  
Esse código de bytes pode então ser executado em qualquer plataforma que suporte um interpretador Java.  
O Java requer somente uma fonte e um binário e, mesmo assim, é capaz de funcionar em diversas plataformas, o que faz dele um sonho de todos os que realizam manutenção em programas.

Um sistema orientado a objetos no java é composto por um conjunto de classes e objetos bem definidos que interagem entre si, de modo a gerar o resultado esperado.A Orientação à Objetos modela o mundo real com classes e instâncias.

Variáveis de uma classe são chamadas de instâncias de classe. Por exemplo: Se existe uma classe Pessoa existe a instância (variável) pessoa que é do tipo Pessoa. Um Objeto, ou instância, é uma entidade cujas informações podem incluir desde suas características até suas atividades, ou seja, uma abstração de um domínio de um problema.

Veja como java organiza as suas aplicações de OO:

Pacote: Conjunto de classes e demais arquivos que possuem interesses comuns ou atuam com dependências entre si. Fisicamente são pastas do sistema operacional.

Instância, objeto: Uma variável do tipo de uma classe.

Construtor: Responsável por iniciar a criação e inicialização de uma instância de classe.

Método: Funções referenciados aquela classe.

Modificador de acesso: Descreve que outras classes podem ter acesso a classe que está se criando. Também é usado para indicar que uma classe pode ser acessada de fora de seu pacote.

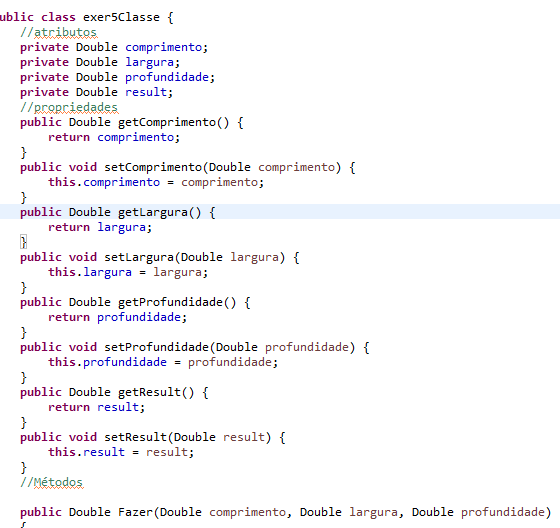
Hierarquia de classes: Um grupo de classes que estão relacionadas por herança.

Superclasse: É a classe que é estendida por uma determinada classe.

Subclasse: É a classe que estende determinada classe.

Classe base: A classe de determinada hierarquia que é uma superclasse de todas as outras classes. A classe? pai de todas

**Exemplo de codigo:**



**Vantagens**

Orientação a objetos vai te ajudar em muito em se organizar e escrever menos, além de concentrar as responsabilidades nos pontos certos, flexibilizando sua aplicação,encapsulando a lógica de negócios.

Outra enorme vantagem, onde você realmente vai economizar montanhas de código, é o polimorfismo das referências, que veremos em um posterior capítulo.

**Gettes and Settes**

**Set**

Nomeamos um método acessor com **set** toda vez que este método for modificar algum campo ou atributo de uma classe, ou seja, se não criarmos um método acessor **set** para algum atributo, isso quer dizer que este atributo não deve ser modificado.

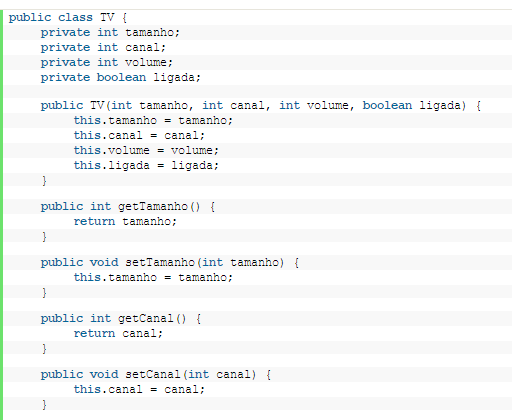
Portanto, como o valor de um atributo da classe será modificado, não é necessário que este método retorne nenhum valor, por isso, os métodos setters são **void**. Porém, obrigatoriamente, eles tem que receber um argumento que será o novo valor do campo.

**Get**

Nomeamos um método acessor com **get** toda vez que este método for verificar algum campo ou atributo de uma classe.

Como este método irá verificar um valor, ele sempre terá um retorno como String, int, float, etc. Mas não terá nenhum argumento.

Exemplo:



**Sobrecarga**

Java nos permite criar vários métodos com o mesmo nome desde que tenham parâmetros diferentes. Isso é o que chamamos de sobrecarga de métodos.

A sobrecarga de métodos consiste em criarmos o mesmo método com possibilidades de entradas diferentes. Essas entradas, caracterizadas como parâmetros, devem sempre ser de tipos diferentes, quantidades de parâmetros diferentes ou posições dos tipos diferentes.

**Herança**

Enquanto programamos em Java, há a necessidade de trabalharmos com várias classes. Muitas vezes, classes diferentes tem características comuns, então, ao invés de criarmos uma nova classe com todas essas características usamos as características de um objeto ou classe já existente.

Ou seja, herança é, na verdade, uma classe derivada de outra classe.

Para simplificar de uma forma mais direta, vejamos:

Vamos imaginar que exista uma classe chamada Eletrodomestico, e nela estão definidos os seguintes atributos: ligado (boolean), voltagem (int) e consumo (int).

Se levarmos em conta a classe TV que estamos usando de exemplo até agora, podemos dizer que TV deriva de Eletrodomestico. Ou seja, a classe TV possui todas as características da classe Eletrodomestico, além de ter suas próprias características

**Polimorfismo**

Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C# e C++, que permite ao desenvolvedor usar o mesmo elemento de formas diferentes. Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. No Polimorfismo temos dois tipos:

Polimorfismo Estático ou Sobrecarga

Polimorfismo Dinâmico ou Sobreposição

O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

**Palavras reservadas**

**this**

Denota uma referência ao próprio objeto sobre o qual está

sendo executado o método.

**break, continue, return**

Interrupção de fluxo

**Private**

-Aplicado à declaração de um método ou variável, o modificador

private torna este elemento acessível apenas ao código da classe

onde está declarado.

- Aplicado à declaração de uma classe ou interface torna esta classe ou

interface acessível apenas no escopo da classe onde está declarada.

**Protected**

Aplicado à declaração de um método ou variável de uma classe, torna

este elemento acessível apenas às classes que pertencem ao mesmo

pacote desta classe, e ao código das classes que herdam desta

primeira.

Aplicado à declaração de uma classe ou interface torna esta classe ou

interface visível apenas no escopo do pacote onde está declarada.

**Public**

Aplicado à declaração de um método, variável, classe ou interface

elimina qualquer restrição de visibilidade a este elemento.

**Abstract**

Aplicado à declaração de um método ou classe

Indica que a implementação deste método ou classe será efetuada

posteriormente concluída na subclasses.

**Static**

Aplicado à declaração de uma variável ou método

Aplicado à declaração de uma variável indica que só existe uma

ocorrência desta variável no ‘runtime’, a qual está presente no objeto

que representa a informação de 'runtime' da classe na qual a variável

está declarada.

Pode se dizer que uma variável static é uma variável de classe (à la

Smalltalk).

Aplicado à declaração de um método indica que o objeto (this) ao

qual este método se referencia é o objeto que representa a informação

de 'runtime’ da classe na qual o método está declarado.

Pode se dizer que um método static é um método de classe.