## ANHANGUERA EDUCACIONAL

**CAMPUS MARTE**

**SISTEMAS DE INFORMAÇÃO – 4º SEMESTRE**

**YTALO MATOS**

# LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

## SÃO PAULO/SP

**2017**

**LISTA DE ILUSTRAÇÕES**

# FIGURA 1 – Pesquisa do Tiobe.com 08

# FIGURA 2 – Herança 11

# FIGURA 3 – Interface 20

**LISTA DE ABREVIATURAS E SIGLAS**

**C# -** Linguagem de Programação da Microsoft

**JVM –** Java Virtual Machine (Máquina Virtual Java)

**O.O –** Orientação a objetos

# SUMÁRIO

1. **INTRODUÇÃO 05**
2. **DESENVOLVIMENTO 05**

**2.1 HISTORICO DA LINGUAGEM 05**

**2.2 PRINCIPAIS RECURSOS 06**

**2.3 IDE UTILIZADA 07**

**3 PORQUE ESCOLHEU ESSA LINGUAGEM? 07**

**4 PILARES DA ORIENTAÇÃO A OBJETOS 08**

**4.1 Abstração 08**

**4.2 Herança 10**

**4.3 Encapsulamento 13**

**4.3.1 Métodos(Getter e Setter) 14**

**4.4 Polimorfismo 16**

**5 CLASSES 17**

**5.1 Instanciação/Objetos 19**

**6 INTERFACE 20**

**6.1 Interfaces X Abstratas 21**

**7 CONCLUSÃO 23**

**REFERÊNCIAS 24**

**1 INTRODUÇÃO**

Linguagem de Programação Orientada a Objetos

O que é?

É a linguagem de programação que tem como paradigma de programação a Orientação a Objetos.

Hoje existe no mercado várias linguagens que seguem este paradigma. Ex.: C++, C#, VB.NET, Java, Object Pascal, Objective-C, Python, SuperCollider, Ruby e Smalltalk. ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5) e Visual Basic (a partir da versão 4) são exemplos de linguagens de programação com suporte a orientação a objetos.

Buscamos, neste trabalho, passar as principais características da Orientação a Objetos, fizemos um resumo das características e usando a linguagem escolhida, fizemos uma comparação com a linguagem C# (Linguagem estudada em sala).

**2 DESENVOLVIMENTO**

JAVA

Linguagem escolhida para fazer a pesquisa e os apontamentos dentro das regras pré-estabelecidas, levantando as características de O.O e mostrando um exemplo de código com uma comparação ao C#.

**2.1 HISTORICO DA LINGUAGEM**

Linguagem Java? Um nome relativamente entranho para uma linguagem de programação se comparado a Pascal, Cobol, Basic e diversos outros nomes de linguagem. Ele surgiu por acaso, quando a equipe de engenheiros da Sun, a criadora do Java, foi tomar café na esquina.

Na realidade, não existe muito consenso entre os escritores americanos com relação à verdadeira história da linguagem.

Originalmente, a linguagem foi concebida para utilização em pequenos dispositivos eletrônicos inteligentes; entretanto, com as dificuldades de financiamento desse setor na época e, principalmente, devido ao surgimento da Internet a partir de 1993, novas oportunidades apareceram e a Sun “entrou de cabeça” nessa área.

A partir de 1995, a Sun anunciou o Java, não apenas como mais uma linguagem de programação, mas como uma nova plataforma de desenvolvimento. Desta forma, Java começou a ser utilizada na elaboração de páginas para a World Wide Web, proporcionando uma produção com conteúdo interativo e dinâmico, inicialmente com o uso de applets(pequeno software que executa atividades especificas) com imagens em movimento.

Desde o ano de 1996 até hoje, o Java não para de crescer, produzindo soluções desde pequenas aplicações até aplicativos corporativos, controle de servidores WWW, celulares, etc.

**2.2 PRINCIPAIS RECURSOS**

Uma das principais características de Java é o WORA (Write Once, Run Anywhere) ou simplesmente 'Escreva uma Vez e Execute em Qualquer Lugar'. Isso por si só traz vantagens, pois poupa tempo aprendendo bibliotecas específicas de Sistemas Operacionais. Poupa dinheiro, que seria gasto neste tempo. Poupa esforço, pois muitas vezes não há a necessidade de se conhecer alguns ou vários aspectos da plataforma subjacente.

Java possui "Coleta de Lixo" automática. Isso significa que ela desloca automaticamente qualquer espaço de memória não utilizado sem que o programador se preocupe com isso.

Java é uma linguagem de alto nível. Por ser uma linguagem de alto nível Java tem construtores e características que permitem escrever programas mais rapidamente do que em C/C++. Sendo assim há uma maior produtividade. Aliado ao fato de que Java sendo uma linguagem orientada a objetos, faz Java uma ótima escolha para desenvolvimento.

Java está presente em uma variedade de lugares. Os aplicativos Java estão presentes em diversas áreas, desde celulares, até servidores, o que dá ao desenvolvedor uma ampla gama de possibilidades de sistemas a serem desenvolvidos.

Java tem uma boa documentação, é gratuito e de código aberto. Como não há custo para se desenvolver em Java, pelo menos no início, há um incentivo aos novatos que não necessitam gastar com programas pagos, mas ao invés disso possuem escolhas de altíssima qualidade gratuitas e amplamente usadas.

**2.3 IDE UTILIZADA**

Neste trabalho utilizamos o Eclipse, Eclipse é um IDE para desenvolvimento Java, porém suporta várias outras linguagens a partir de plug-ins como C/C++, PHP, ColdFusion, Python, Scala e plataforma Android. Ele foi feito em Java e segue o modelo open source de desenvolvimento de software. Atualmente faz parte do kit de desenvolvimento de software recomendado para desenvolvedores Android.

O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares. Hoje, o Eclipse é o IDE Java mais utilizado no mundo. Possui como característica marcante, o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

1. **PORQUE ESCOLHEU ESSA LINGUAGEM?**

Java continua a ser a linguagem de programação nº 1 – Pesquisa do tiobe.com (mar/17)

Hoje em dia, apesar das várias linguagens que estão disponíveis no mercado para desenvolvimento, algumas apareceram para responder a novos modelos de computação e outras vão se mantendo.

Apesar de altos e baixos o Java se mantém como a mais popular, devido à plataforma Android.

Além de ser uma plataforma de desenvolvimento, desenvolve em qualquer sistema operacional para qualquer sistema operacional.

As comunidades têm os grupos muito fortes, é fácil conseguir conteúdo para estudar e ajuda para tirar dúvidas.

É grátis e tem grande coleção de bibliotecas Open Source. Grande número de frameworks, a JVM roda cerca de 350 linguagens.

Fora estas facilidades, ainda há muito mercado de trabalho para programador Java, e é a linguagem que tem a maior probabilidade de trazer retorno financeiro.

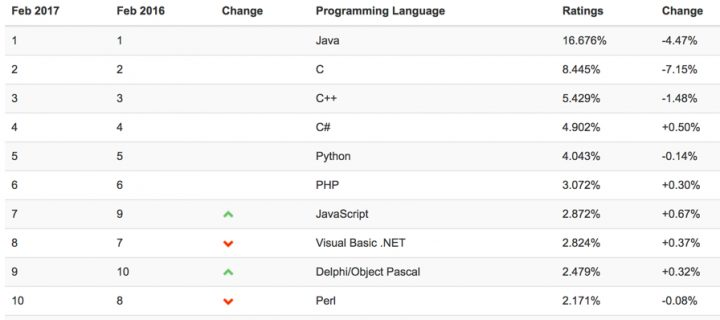


Figura 1 - Pesquisa do tiobe.com (mar/17)

1. **PILARES DA ORIENTAÇÃO A OBJETOS**

Temos como característica da O.O, e são denominados de pilares da O.O, o seguinte:

**4.1 Abstração**

É utilizada para definir as entidades (classes) do mundo real. Essas entidades são consideradas tudo que é real, levando em consideração suas características e ações, ou seja, atributos e métodos respectivamente.

As classes abstratas, são classes que servem como modelo para uma classe concreta. Por servirem de modelo para as classes que herdarão suas características, elas não podem ser instanciadas diretamente.

Elas podem conter ou não métodos abstratos, podem implementar ou não um método. A classe filha, ou seja, que herdar da classe abstrata, se esta for concreta, deve obrigatoriamente implementar os métodos abstratos da classe Pai. Se for abstrata também, não há necessidade de implementação.

Usamos classes abstratas para representar grupos que tem características comum, mas que em alguns detalhes específicos, agem de maneira diferente.

As classes abstratas são bastante relacionadas com Polimorfismo.

Exemplo de uma abstração

Classe abstrata Conta

abstract class Conta {

private double saldo;

public void setSaldo(double saldo) {

this.saldo = saldo;

}

public double getSaldo() {

return saldo;

}

public abstract void imprimeExtrato();

}

Classe Conta Poupança que herda Conta

public class ContaPoupanca extends Conta {

@Override

public void imprimeExtrato() {

System.out.println(“### Extrato da Conta ###”);

SimpleDateFormat sdf = new SimpleDateFormat(“dd/MM/aaaa HH:mm:ss”);

Date date = new Date();

System.out.println(“Saldo: “+this.getSaldo());

System.out.println(“Data: “+sdf.format(date));

}

}

Classe principal

public class TestaConta {

public static void main(String[] args) {

Conta cp = new ContaPoupanca();

cp.setSaldo(2121);

cp.imprimeExtrato();

}

}

Este mesmo exemplo em C#:

Classe abstrata Conta

public abstract class Conta

{

private double saldo;

public void setSaldo(double saldo)

{

this.saldo = saldo;

}

public double getSaldo()

{

return saldo;

}

public abstract void imprimeExtrato();

}

Classe Conta Poupança que herda Conta

public class ContaPoupanca : Conta

{

public override void imprimeExtrato()

{

Console.WriteLine(“### Extrato da Conta ###”);

Console.WriteLine(“Saldo: “ + this.getSaldo());

Console.WriteLine(“Data: “ + DateTime.Now.ToString(“dd/MM/yyyy HH:mm”));

}

}

Classe principal

public class Program

{

static void Main(string[] args)

{

Conta conta = new ContaPoupanca();

conta.setSaldo(2121);

conta.imprimeExtrato();

Console.ReadLine();

}

}

**4.2 Herança**

Uma das principais características da orientação a objetos, a herança tem como objetivo ajudar, permitindo com que o usuário crie classes que reutilizem de uma forma bem ampla os códigos de outra classe (classe pai ou base).

A classe base herda quase tudo da classe principal, menos seus métodos construtores e finalizadores. Um ponto positivo é que além de herdar ela não se limita somente a isso, podendo assim criar seus próprios métodos, membros, funcionalidades etc.

Como o próprio nome sugere, na O.O, o termo herança se refere a algo herdado. Em Java, a herança ocorre quando uma classe passa a herdar características (variáveis e métodos) definidas em uma outra classe, especificada como sua ancestral ou superclasse. A técnica da herança possibilita o compartilhamento ou reaproveitamento de recursos definidos anteriormente em outra classe. A classe que fornece recebe o nome de superclasse e a que recebe subclasse.

É uma técnica extremamente utilizada em Java.

Obs. Em Java pode se utilizar múltipla herança enquanto em C# não.

Exemplo em Java:

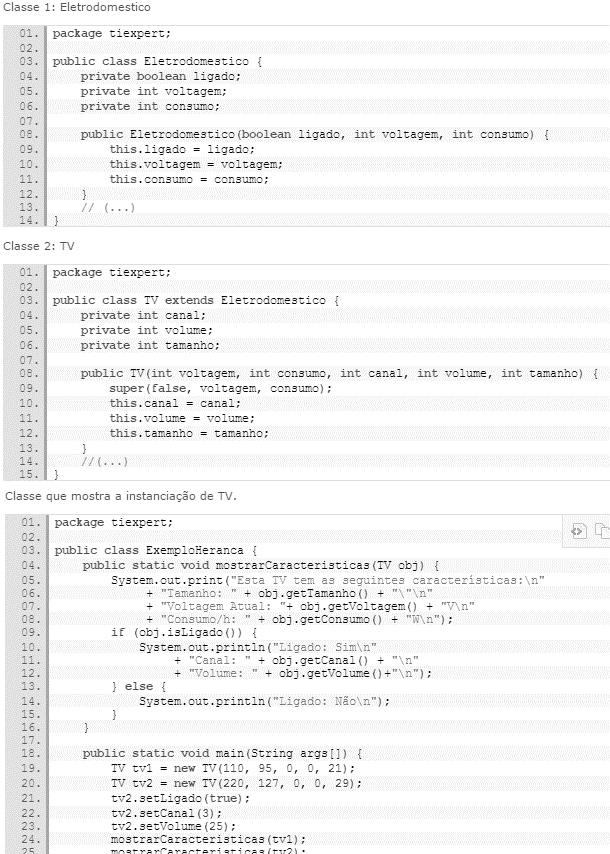


Figura 2 – Herança

Exemplo em C#

public class Eletrodomestico

{

private bool ligado;

private int voltagem;

private int consumo;

public Eletrodomestico(bool ligado, int voltagem, int consumo)

{

this.ligado = ligado;

this.voltagem = voltagem;

this.consumo = consumo;

}

public int Voltagem { get { return voltagem; } set { voltagem = value; } }

public int Consumo { get { return consumo; } set { consumo = value; } }

public bool Ligado { get { return true; } set { ligado = value; } }

}

public class TV : Eletrodomestico

{

private int canal;

private int volume;

private int tamanho;

public TV(int voltagem, int consumo, int canal, int volume, int tamanho)

: base(false, voltagem, consumo)

{

this.canal = canal;

this.volume = volume;

this.tamanho = tamanho;

}

public int Tamanho { get { return tamanho; } set { tamanho = value; } }

public int Canal { get { return canal; } set { canal = value; } }

public int Volume { get { return volume; } set { volume = value; } }

}

public static void mostrarCaracteristicas(TV obj)

{

Console.Write(“Esta TV tem as seguintes características:\n”

+ “Tamanho: “ + obj.Tamanho + “\”\n”

+ “Voltagem Atual: “ + obj.Voltagem + “W\n”

+ “Consumo/h: “ + obj.Consumo + “W\n”);

if (obj.Ligado)

{

Console.WriteLine(“Ligado: Sim\n”

+ “Canal: “ + obj.Canal + “\n”

+ “Volume: “ + obj.Volume + “\n”);

}

else

{

Console.WriteLine(“Ligado: Não\n”);

}

}

static void Main(string[] args)

{

TV tv1 = new TV(110, 95, 0, 0, 21);

TV tv2 = new TV(220, 127, 0, 0, 29);

tv2.Ligado = true;

tv2.Canal = 3;

tv2.Volume = 25;

mostrarCaracteristicas(tv1);

mostrarCaracteristicas(tv2);

Console.ReadLine();

}

}

**4.3 Encapsulamento**

É um conceito da Programação Orientada a Objetos onde o estado de objetos (as variáveis da classe) e seus comportamentos (os métodos da classe) são agrupados em conjuntos segundo o seu grau de relação.

O propósito do encapsulamento é o de organizar os dados que sejam relacionados, agrupando-os (encapsulando-os) em objetos (classes), reduzindo as colisões de nomes de variáveis (dado que variáveis com o mesmo nome estarão em namespaces distintos) e, da mesma forma, reunindo métodos relacionados às suas propriedades (ou variáveis de classe). Este padrão ajuda a manter um programa com centenas ou milhares de linhas de código mais legível e fácil de trabalhar e manter.

O uso do encapsulamento fornece uma série de vantagens, além dos citados anteriormente, minimizar erros de programação e facilitar a ampliação do código em função de novas atualizações.

Para determinar o nível de acesso dos elementos de uma classe, são usados qualificadores de acesso:

Public: um nível sem restrições, equivalente a não encapsular.

Protected: um nível intermediário de encapsulamento em que as variáveis e métodos podem ser acessados pela própria classe ou por suas subclasses.

Private: o nível mais rígido em que apenas os métodos da própria classe têm acesso a variáveis e outros métodos.

Package: nível em que as variáveis e métodos podem ser acessados apenas por outras classes pertencentes ao mesmo pacote.

**4.3.1 Métodos(Getter e Setter)**

A forma mais correta para alcançarmos os atributos de uma classe é utilizando métodos e um dos métodos que foram padronizados para isso foram os métodos GET e SET. Através desses métodos determinamos quando será modificado um atributo e sua entrada. Deixando seu manejo e alterações com maior praticidade e clareza.

Para poder acessar os determinados atributos, tem que ser colocado Get e SET antes do nome do atributo para poder assim criar os métodos, cada método terá sua determinada função.

Método Get

Para poder ter acesso ao atributo deve-se utilizar os métodos GET para assim poder retornar um valor, podendo ser do tipo int, double etc.

public String getNomeEscola() {

return nomeescola;

}

public int getunidades() {

return unidades;

}

public String getValorUnitario() {

return valorunitario;

}

Método Set

Para que os atributos sejam alterados de maneira segura devemos, aplicar os métodos SET ele não possui retorno como o GET por que nessa etapa o atributo será apenas modificado, criando assim um método do tipo VOID (Sem Retorno). Mas ele deve ter alguma razão para poder acontecer tal alteração.

Public void setNomeEscola(string nomeescola) {

this.nomeescola = nomeescola;

}

public void getUnidades(int unidades) {

this.unidades = unidades;

}

public void getValorUnitario(double valorunitario) {

this.valorunitario = valorunitario;

}

Na IDE Eclipse é possível gerar Getters e Setters automaticamente evitando possíveis erros ao inserir manualmente.

Exemplo em Java:

public class Ponto {

private double x;

private double y;

public double getX() {

return x;

}

public void setX(double x) {

this.x = x;

}

public double getY() {

return y;

}

public void setY(double y) {

this.y = y;

}

}

Exemplo em C#

Public class Ponto

{

Private double x;

Private double y;

Public double X { get { return x; } set { x = value; } }

Public double Y { get { return y; } set { y = value; } }

Na linguagem C# usamos características com getters e setters, o que simplifica a leitura de dados de um objeto quando a lógica é executada. Já no Java essas atividades são realizadas através de métodos quando é necessária lógica para ler um campo e escrever neles podendo ser os métodos getCampo e setCampo.

**4.4 Polimorfismo**

Representa uma técnica avançada de programação e seu uso pode gerar economia de recursos computacionais. A ideia geral é que uma determinada classe mais genérica possa assumir diferentes comportamentos, gerando objetos distintos, dependendo de certas condições. Na prática quer dizer que um mesmo objeto pode executar métodos diferentes, dependendo do momento de sua criação. O objeto possui comportamentos variados que dependem da classe ao qual foi criado.

Exemplo em Java

public class Animal {

public void comer() {

System.out.println( “Animal Comendo…” );

}

}

public class Cao extends Animal {

public void comer() {

System.out.println( “Cão Comendo…” );

}

}

public class Tigre extends Animal {

public void comer() {

System.out.println( “Tirgre Comendo…” );

}

}

public class Test {

public void fazerAnimalComer( Animal animal ) {

animal.comer();

}

public static void main( String[] args ) {

Test t = new Test();

t.fazerAnimalComer( new Animal() );

t.fazerAnimalComer( new Cao() );

t.fazerAnimalComer( new Trigre() );

}

}

Exemplo em C#

public class Animal

{

public void Comer()

{

Console.WriteLine(“Animal Comendo...”);

}

}

public class Cao : Animal

{

public void Comer()

{

Console.WriteLine(“Cão Comendo...”);

}

}

public class Tigre : Animal

{

public void Comer()

{

Console.WriteLine(“Tigre Comendo...”);

}

}

public class Test

{

public void fazerAnimalComer(Animal animal)

{

animal.Comer();

}

}

class Program

{

static void Main(string[] args)

{

Test t = new Test();

t.fazerAnimalComer(new Animal());

t.fazerAnimalComer(new Cao());

t.fazerAnimalComer(new Tigre());

Console.ReadLine();

}

}

Podemos notar que cada chamada vai fazer uma coisa diferente, porem como existe herança entre as classes todos os métodos irão funcionar, porque são do tipo do mais genérico (Animal) ou são filhos do mais genérico.

**5 Classes**

Uma classe comumente define o estado e o comportamento de um objeto implementando atributos e métodos. Os atributos (por vezes referidos como “campos”, “membros de dados” ou “propriedades”), indicam as possíveis informações armazenadas por um objeto de uma classe, representando o estado de cada objeto. Os métodos (por vezes referidos como “operações” ou serviços) são procedimentos que formam os comportamentos e ações oferecidos por objetos de uma classe, sendo responsáveis por alterar o estado ou fornecer informações sobre um objeto.

As instâncias de uma classe compartilham o mesmo conjunto de atributos, embora sejam diferentes quanto ao conteúdo desses atributos. Por exemplo, a classe “Empregado” descreve os atributos comuns a todas as instâncias da classe “Empregado”. Os objetos dessa classe podem ser semelhantes, mas variam em atributos tais como “nome” e “salário”. A descrição da classe contém os itens correspondentes a esses atributos e define as operações ou ações relevantes para a classe, tais como “aumento de salário” ou “mudança do número de telefone”. Pode-se então falar sobre uma instância com o nome = “Joana Coelho” e outra com o nome = “João Coelho”.

Instância é a concretização de uma classe. Em termos intuitivos, uma classe é como um “molde” que gera instâncias de um certo tipo; um objeto é algo que existe fisicamente e que foi “moldado” na classe.

Assim, em programação orientada a objetos, a palavra “instanciar” significa criar. Quando falamos em “instanciar um objeto”, criamos fisicamente uma representação concreta da classe. Por exemplo: “animal” é uma classe ou um molde; “cachorro” é uma instância de “animal” e apesar de carregar todas as características do molde de “animal”, é completamente independente de outras instâncias de “animal”.

Exemplo em Java:

// Classe

Public class Animal{

// Atributo

protected string especie;

// Construtor

public Animal(string oolean) {

this.especie = oolean;

}

// Execução

static void Main(string[] args){

// Instâncias

Animal cachorro = new Animal(“Canis oole familiaris”);

Animal gato = new Animal(“Felis catus”);

Animal lobo = new Animal(“Canis oole”);

}

}

**5.1 INSTANCIAÇÃO/OBJETOS**

Objetos são estruturas de dados definidas e agrupadas dentro de uma classe. Sempre que utilizamos um objeto ou classe devemos reservar espaço em memória para que aquele objeto seja manipulado sem maiores problemas.

Além do mais, também podemos utilizar a mesma classe (com todos os seus métodos e atributos) para manipular outros objetos que serão tratados de forma diferente (mesmo se tiverem as mesmas características do objeto anterior), pois serão dois endereços de memória diferentes.

A vantagem de Java é nos possibilitar uma instanciação rápida e simples, sem termos que nos preocupar com referência a endereços e alocação dinâmica de memória, pois quem trata de manipular a memória é o próprio Java. Sem contar que, se um objeto não é mais referenciado dentro do programa, o próprio Java trata de liberar os recursos de memória consumidos pelo objeto usando o Garbage Collection – Coletor de Lixo.

Quem faz o papel de instanciador em Java é o new. New trata de reservar memória o suficiente para o objeto e criar automaticamente uma referência a ele. Para new conseguir determinar o objeto, precisamos usar o método construtor que será usado como base para instanciar a classe e gerar o objeto.

Tecnicamente, declaramos uma variável qualquer como sendo do tipo da classe (ex.: TV minhaTV;), depois instanciamos o objeto atribuindo a variável o resultado obtido por new mais o método construtor (ex.: minhaTV = new TV(); ).

Apesar de parecer muitos detalhes, isso fica mais facilmente visualizado no código abaixo.

Public class TV {

int tamanho;

int canal;

oolean ligada;

TV() {

tamanho = 21;

canal = 0;

ligada = false;

}

public static void main(String[] args) {

TV objeto1 = new TV();

TV objeto2;

objeto2 = new TV();

}

}

**6 INTERFACE**

A interface em Java é uma classe abstrata que pode ter apenas métodos abstratos e não pode ser instanciada e não tem nenhuma implementação

A interface serve para que as classes, baseadas nessas interfaces, consigam implementar os métodos da interface para que cada classe possa executar sua tarefa de forma diferente.

A interface tem o objetivo de obrigar um determinado conjunto de classes a ter métodos e propriedades em comum para atingir um determinado objetivo, porém a interface permite que os métodos possam ser implementados em cada classe de maneiras diferentes. Exemplo simples de interface

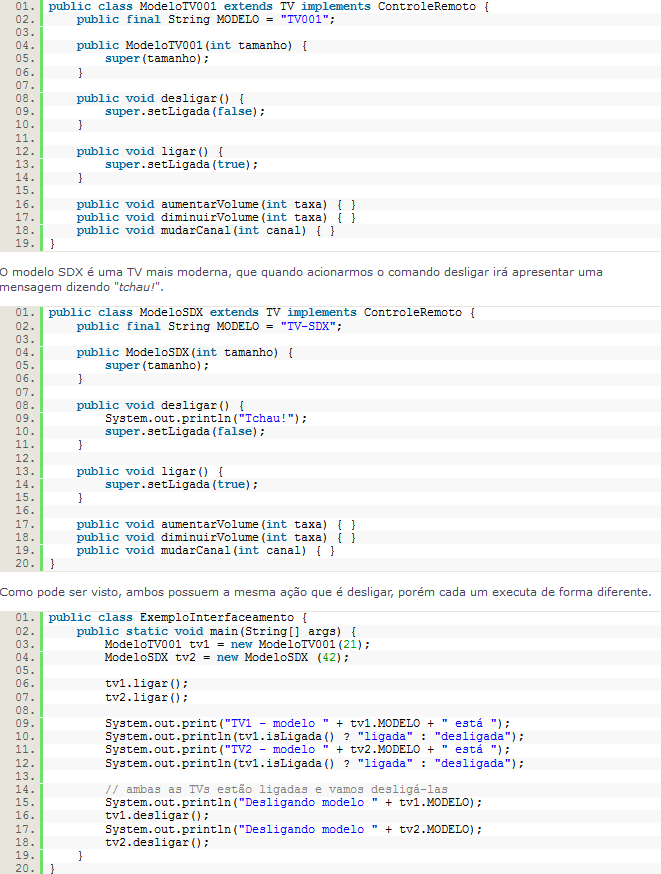


Figura 3 – Interface

Usando um exemplo bem simples, podemos ter uma interface chamada aéreo que possui o método voar(). Ou seja, toda classe que implementar aéreo deve dizer como voar(). Portanto, se eu tenho uma classe chamada pássaro e outra chamada avião, ambas implementando a interface aéreo, então, nestas duas classes devemos codificar a forma como cada um irá voar().

Uma interface é criada da mesma forma que uma classe, mas utilizando a palavra-chave interface no lugar de class.

Interface nomeDaInterface { métodoAbstrato (argumentos); }

**6.1 Interfaces X Classes Abstratas**

Interfaces:

Uma interface não é considerada uma Classe e sim uma Entidade.

Não possui implementação, apenas assinatura, ou seja, apenas a definição dos seus métodos sem o corpo.

Todos os métodos são abstratos.

Seus métodos são implicitamente Públicos e Abstratos.

Não há como fazer uma instância de uma Interface e nem como criar um Construtor.

Funcionam como um tipo de "contrato", onde são especificados os atributos, métodos e funções que as classes que implementem essa interface são obrigadas a implementar.

Classes Abstratas:

As classes abstratas devem conter pelo menos um método abstrato, que não tem corpo.

É um tipo especial de classe que não há como criar instâncias dela.

É usada apenas para ser herdada, funciona como uma superclasse.

Uma grande vantagem é que força a hierarquia para todas as subclasses.

É um tipo de contrato que faz com que as subclasses contemplem as mesmas hierarquias e/ou padrões.

Exemplo de uma interface

public interface Celular{

public String sistemaOperacional();

}

Classe Android

public class Android implements Celular {

@Override

public String sistemaOperacional (){

return “Meu sistema é Android.";

}

}

Classe IOS

public class Ios implements Celular {

@Override

public String sistemaOperacional (){

return "Meu sistema é IOS.";

}

}

Este mesmo exemplo em C#:

public interface Celular

{

public string sistemaOperacional() { }

}

public class Android : Celular

{

public string sistemaOperacional()

{

return "Android";

}

}

public class Ios : Celular

{

public string sistemaOperacional()

{

return "Ios";

}

}

class Program

{

static void Main(string[] args)

{

Android android = new Android();

Ios ios = new Ios();

string sistema1 = android.sistemaOperacional();

string sistema2 = ios.sistemaOperacional();

Console.WriteLine(sistema1);

Console.WriteLine(sistema2);

Console.ReadLine();

}

}

**7 CONCLUSÃO**

Fizemos uma breve pesquisa, sobre a linguagem Java, onde descrevemos um pouco de sua história, como foi criada.

Apresentamos os principais recursos da linguagem onde pode-se analisar e concluir os objetivos da linguagem, mesmo estando em constante evolução.

Foi realizada uma introdução da Orientação a Objetos, um paradigma de programação ao qual o Java se baseia, apesar de hoje em dia, já está sendo implementado em outros paradigmas.

Dentro da O.O, descrevemos suas principais características, e apresentamos um exemplo de código em Java e um outro código do mesmo exemplo em C#. Que descreve como a O.O, funciona ou como devemos aplica-la.

Utilizamos e descrevemos um pouco sobre um ambiente de desenvolvimento que foi criado para desenvolver a programação da linguagem, o Eclipse.

Analisamos, como o Java está na atualidade, tanto para estudo, quanto para o mercado de trabalho, e foi constatado que o Java, ainda, é a principal linguagem de programação, a mais popular e com mais usuários, conseguimos concluir que ainda se trata de uma ótima opção de linguagem de programação, para se investir.

Mesmo com todas as linguagens que apareceram depois dela.

Foi um ótimo trabalho, pois dentro de toda a pesquisa pode se levantar e tirar dúvidas, como está a atualidade da programação, e o mercado de trabalho. Além de poder se desenvolver pessoalmente, e absorver um pouco mais sobre Orientação a Objetos.

**REFERÊNCIAS**

FURGERI, Sérgio. **Java 6 – Ensino Didático – Desenvolvendo e Implementando Aplicações**. 2. Ed. São Paulo, Editora Érica, 2010. 17, 18, 152-157 p.

Eclipse – Disponível em: <https://pt.wikipedia.org/wiki/Eclipse(software)>

Por que Java? Disponível em: <http://www.devmedia.com.br/por-que-java/20384>

Abstração. Disponível em: <http://www.tiexpert.net/programacao/java/classes-abstratas.php>

Pilares da Orientação a Objetos. Disponível em: <http://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366>

Polimorfismo. Disponível em: <http://www.javaprogressivo.net/2012/10/Polimorfismo--Classes-abstratas-e-Metodos-abstratos.html>

Por que Java? Disponível em: <http://tek.sapo.pt/opiniao/artigos/opiniao-10-motivos-para-aprender-a-programar-java-e-porque-e-uma-das-melhores-linguagens>

Por que Java? Disponível em: <https://pplware.sapo.pt/informacao/java-continua-linguagem-programacao-no1/>

Principais recursos. Disponível em: <https://pt.wikibooks.org/wiki/Java/Introdu%C3%A7%C3%A3o>

Interface. Disponível em: <//http://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387>

Métodos Get/Set. Disponível em: <www.devmedia.com.br/criando-metodos-get-e-set-em-java/24623>

Herança. Disponível em: <http://www.tiexpert.net/programacao/java/heranca.php>

Herança. Disponível em: <https://www.caelum.com.br/apostila-csharp-orientacao-objetos/heranca/#10-2-reaproveitando-a-implementacao-da-classe-base>

Classes e Instanciação. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/inheritance>

Polimorfismo. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/>

Instanciação. Disponível em: <http://www.devmedia.com.br/conceitos-e-exemplos-instanciacao-estrutura-da-linguagem/18817>

Objetos. Disponível em: <http://objota.com.br/java/instanciando-objetos.html>