LINGUAGEM BASEADA EM CLASSES V.S. LINGUAGEM NÃO BASEADA EM CLASSE

ANDRÉ LUIZ KUCZNER PEREIRA, ANDREI RICARDO RODRIGUES, ELINEY SABINO, LUIZ CLAUDIO BARRETO, THISSIANY BEATRIZ ALMEIDA, LUIZ DA ROCHA FARIAS

RESUMO

Este trabalho visa a comparação entre duas linguagens orientadas a objeto que possuem características distintas, uma se baseia em classes e a outra não. Para este trabalho usou como metodologia a escolha de um objeto para compará-las dentro de um conjunto de critérios que foram percebidos como comum em linguagens orientadas a objeto.

Palavras-Chave: Programação Orientada a Objeto, Java, JavaScript

ABSTRACT

This paper aims at comparing two languages oriented to an object that has different, a database in classes and others not. For this work, as a methodology, a choice of an object to compare it within a set of criteria that so many perceived as common in object-oriented languages.

KEYWORDS: Object Oriented Programing, Java, JavaScript

INTRODUÇÃO

Programação Orientada a Objetos ou simplesmente POO, é um paradigma de programação onde se usam classes e objetos, para representar e processar dados usando programas de computador [1]. Esse paradigma traz três pilares para que ela seja eficiente e prática. Estes pilares são: (i) **Encapsulamento** (proteção ao acesso das propriedades do objeto por meio níveis de acesso e de métodos especiais), (ii) **Herança** (reuso de propriedades e métodos comuns para alguns objetos que são passados de uma classe "pai" para seus "filhos") e (iii) **Polimorfismo** (mudança ou reescrita do funcionamento de um método que foi herdado para um fim diferente para aquele objeto específico)[1].

Este trabalho tem por objetivo trazer uma comparação sobre os principais aspectos da programação das linguagens baseada em classe entre as linguagens não baseada em classe. A fim de mostrar que o paradigma(POO) é funcional, prático, organizado e rápido para desenvolvimento de sistemas em linguagens que foram criadas para trabalhar dessa maneira.

JAVA E JAVASCRIPT

As Linguagens *JAVA* e *JAVASCRIPT* são linguagens orientadas a objetos com diferenças dentro do próprio paradigma.

A linguagem *JAVA* surgiu em meados de 1991, criada por James Gosling e financiada pela antiga *SUN* que queria uma linguagem a qual pudesse ser aplicada em qualquer computador ou dispositivo logicamente complexo. Essa linguagem herdou funções de antigas linguagem como C e C++(orientada a objeto)[2].

A linguagem *JAVASCRIPT* é uma linguagem de alto nível criada pela *NetScape*(hoje fundação Mozilla) no início da *Web*. Apesar do nome sugestivo, não tem nada haver com *JAVA*. A linguagem foi padronizado pela *European Computer Manufacturer's Association (ECMA)* sob o nome de *ECMAscript* mas é comum chamar de *JAVASCRIPT.[3]*.

COMPARAÇÃO

Neste trabalho será comparado a aplicação do paradigma com suas características nas linguagens para as quais foram desenvolvidas, independente das versões mais recentes, tomando essa comparação de forma genérica. Para tal usou-se um exemplo, o objeto "Pessoa" para o estudo, onde uma pessoa tem um nome, uma idade e uma ação, falar.

Foi construído um modelo de tópicos como metodologia de comparação teórica sobre as duas linguagens criadas sobre o paradigma da POO. Assim podemos equiparar as semelhanças e ou diferenças entre elas. Este modelo é baseado nos seguintes itens: (i)Classes e objetos, (ii)Visibilidade de atributos e métodos, (iii)Métodos especiais, (iv)Herança, (v)Polimorfismo. Os códigos para as duas linguagens foram escritos em suas formatações de fonte padrão.

CLASSES E OBJETOS

Java é uma linguagem de programação estruturada orientada a objetos interpretada pela JVM (*Java virtual machine*).

Por ser uma linguagem estruturada ela possui o conceito de classes e objetos, classes em java é usado para criar um modelo de algo real que queremos em nossa aplicação, através desse modelo (classe) podemos gerar vários objetos para usar em nossa aplicação.

Esse modelo de classe e objeto é utilizado para facilitar a manipulação de dados e métodos dentro de sua aplicação.

Criando a classe pessoa.

```
public Class Pessoa{
// atributos e métodos
String nome;
int idade;

public Pessoa(){ // construtor
  this.nome;
  this.idade;
};

public void falar(){
   System.out.println(this.nome + "Está Falando");
};
```

Criação do objeto fulano a partir da classe pessoa.

```
Pessoa fulano = new Pessoa();
fulano.nome = "Fulano";
fulano.idade = 35;
System.out.println(" Eu sou " + fulano.nome + " e tenho " + fulano.idade + " anos.");
fulano.falar();
```

JAVASCRIPT é uma linguagem de tipagem solta orientada a objeto, onde foi criada a princípio para que somente os navegadores interpretam a linguagem através do motor da google chamado V8. Atualmente existe o "*node.js''* que é um interpretador em tempo real da linguagem JAVASCRIPT, que faz uso do motor V8 da google. Tanto o nodejs e o motor V8 da google são programados em cima da linguagem C, onde a linguagem C converte a sintaxe JAVASCRIPT para linguagem de máquina, onde o processador poderá entender. Dessa forma a linguagem JAVASCRIPT graças à linguagem C se torna uma linguagem muito rápida e eficiente.

Em JAVASCRIPT temos o seguinte conceito tudo é objeto, diferente de uma linguagem estruturada onde só temos objetos quando são criados a partir de uma classe no JAVASCRIPT sempre estamos lidando com objetos.

Exemplo de objeto literal em *Javascript*.

```
var pessoa = {
        nome: 'Fulano',
        idade: 35,
        falar: function(nome){
     console.log(nome + "Está Falando");
    }
};
```

Outra forma de fazer objeto em forma de expressão em *Javascript*.

```
var Pessoa = function(){
        this.nome;
        this.idade;
        this.falar = function(nome){
     console.log(nome + 'Está Falando');
    }
};
```

Instanciando o objeto.

```
var fulano = new Pessoa();
fulano.nome = 'fulano';
fulano.idade = 35;
fulano.falar(fulano.nome);
```

VISIBILIDADE DE ATRIBUTOS E MÉTODOS

Essas classes possuem seus atributos e métodos que podem ter um nível de visibilidade como private, public ou protected (Privado, público ou protegido respectivamente) que é utilizado para manter o controle em sua aplicação e segurança para o desenvolvedor no momento de desenvolvimento.

O encapsulamento não mais é que a proteção dos atributos do objeto por meio da visibilidade, que no caso se trata da visibilidade *private* ou privado. Também se tem essa proteção com o *protected (protegido)*, embora o encapsulamento seja feito com o *private*. Dessa forma se tem acesso a esses atributos somente com os métodos *GET* e *SET*, o que impede que outros objetos acessem os atributos de objetos alheios. Outra forma de usar o encapsulamento é o uso de interfaces(classes abstratas que contém somente métodos vazios, garantido que aquelas funções serão incorporadas no código).

Public = forma com que o objeto consegue acessar os métodos de forma direta porém aberta.

```
public String nome;
public int idade;
public void falar(){};
```

Private = forma com que o objeto não tem mais acesso indireto aos atributos e métodos (encapsulamento).

```
private String nome; // atributos encapsulados como privados só podem
private int idade;  //  ser acessados pelos métodos set e get.
private void falar(){};
```

Protected = forma com que o objeto pode acessar os atributos e métodos somente na classe que foram estendidas.

```
Protected String nome; // também podem ser considerados encapsulados,
Protected int idade;  //  porém não são tão seguros.
Protected void falar(){};
```

Em *JAVASCRIPT* nós temos como dito anteriormente o conceito de que tudo são objetos, logo quando criamos atributos ou métodos nós temos um conceito chamado de *escopo*, cada objeto tem seu escopo, dessa forma um atributo criado dentro de um objeto não pode ser acessado ou mudado diretamente por outro objeto.

Visibilidade em Javascript se dá pelo escopo.

```
// Escopo pessoa
var pessoa = {
        nome: 'Fulano',
        idade: 35
        falar: function(nome){
    console.log(nome + "Está Falando");
    }
};

// Escopo pessoa2
var pessoa2 = {
        nome: 'Sicrano',
        idade: 35,
        falar: function(nome){
    console.log(nome + "Está Falando");
    },
    altura: 1.84
};
```

MÉTODOS ESPECIAIS

As classes em java também possuem métodos construtores e métodos de encapsulamento como (*Getters* e *Setters*). O método construtor como o próprio nome diz, é usado para construtor um novo objeto após instanciar o objeto. Já os métodos getters e setters são utilizados para encapsular os atributos da classe, isto é feito para que traga mais segurança no momento do desenvolvimento não podendo alterar um atributo por descuido, sendo assim obrigado chamar o método get ou set para alterar.

```
// método construtor
Public Pessoa(){
  this. nome;
  this.idade;
};

// método Getter
public Pessoa(String nome, int Idade){
  this.nome;
  this.idade
};

fulano.setNome("fulano"); // método Setter
```

Criando o objeto dessa forma e usando o método *Getter*:

```
Pessoa fulano = new Pessoa(“fulano”, 35);
fulano.getNome();// método Getter
```

Podemos construir um objeto em javascript de duas forma de forma objeto literal (também conhecido de json) ou com uma expressão de função. A expressão de função é muito parecido como fazemos em outras linguagem. Desta forma nos permite ganhar muito mais tempo na linguagem javascript do que em java, pois não precisamos de toda a estrutura que o java necessita, apenas declaramos um objeto de forma literal e já está pronto para uso.

No Javascript acessamos os atributos via objeto construtor ou literal.

```
var pessoa = {
        nome: 'Fulano',
        idade: 35
        falar: function(nome){
     console.log(nome + "Está Falando");
    }
};
```

Herança

Podemos também em Java utilizar métodos e atributos de uma classe em outra classe, esse conceito é chamado de herança. O conceito de herança é utilizado para fazer com que classes se interajam entre si fazendo com que um certo componente de uma aplicação possa ser utilizado em outro componente da aplicação.

Para isso usa-se o termo *Extends*, que significa estender algo. Neste caso algo a alguém ou mais precisamente a um filho

CLASSE PAI.

```
public Class Pessoa{
String nome; // atributos comum para objetos filhos
int idade;

public Pessoa(){ // construtor
  this.nome;
  this.idade;
};

public void falar(){ // método de ação do objeto
   System.out.println(this.nome + "Está Falando");
};

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

public setIdade(int idade){
  this.idade = idade;
};
```

CLASSE FILHO.

```
public Class Funcionario Extends Pessoa {
  public String cargo; // atributos específicos de Funcionarios
  public Funcionario(){
    this.cargo;
  };

  public setCargo(String cargo){
    this.cargo = cargo;
};
};
```

Criando e manipulando a classe Funcionario.

```
Funcionario fulano = new Funcionario();
fulano.setNome("fulano);
fulano.setIdade(35);
fulano.setCargo("programador");
```

A herança em javascript acontece de forma diferente de outras linguagens, já que tudo em javascript é objeto. Utiliza-se o objeto *prototype* para compartilhar algo de um objeto para o outro objeto.

Herança em *Javascript*.

```
var Pessoa = function(){
```

```
        this.nome;
        this.idade;
        this.falar = function(nome){
      console.log(nome + 'Está Falando');
    }
};

var Pessoa2 = function(){
    Pessoa.call(this); // vai receber o objeto de pessoa
};

Pessoa2.prototype = Object.create(Pessoa.prototype); // criando a instância da herança

Pessoa2 fulano = new Pessoa2();
fulano.nome = 'Fulano';
```

POLIMORFISMO

O conceito de polimorfismo em *JAVA* consiste em utilizar o modelo (classe) para construir vários tipos de objetos. Assim, foram criadas duas classes diferentes: Pessoa e Cachorro, onde o objeto de uma classe pode utilizar atributos de outra classe.

Classe Pessoa:

```
public Class Pessoa{
// atributos e métodos
String nome;
int idade;

public Pessoa(){ // construtor
  this.nome;
  this.idade;
};

public void falar(){
   System.out.println(this.nome + "Está Falando");
};
```

Classe Cachorro:

```
public Class Cachorro{
// atributos e métodos
String pelo;
String raca;

public Cachorro(){ // construtor
  this.pelo;
  this.raca;
};
```

```
public void latir(){
  // sout = o cachorro está latindo
};
```

Instanciando os objetos pessoa e cachorro.

```
Pessoa fulano = new Pessoa();
Cachorro toto = new Cachorro();
fulano.nome = "Fulano";
toto.pelo = "preto";
```

Instanciando o objeto cachorro a a partir da classe pessoa.

```
Cachorro toto = new Pessoa();
toto.nome = "Toto";
toto.pelo = "preto";
```

Apesar do objeto toto ter sido instanciado a partir da classe pessoa ele tem sua própria classe com seus atributo e métodos e ainda caracteriza uma transformação, o que é bem diferente da herança.
O conceito é parecido com outras linguagem, mais em *JAVASCRIPT* é criado um objeto e a partir desse objeto podemos expor os seus atributos através do *prototype* para criar novos tipos de objeto.
Polimorfia em Javascript.

```
var Pessoa = function (nome, idade) {
  this.nome = nome;
  this.idade = idade;
};

var Cachorro = function (altura, largura) {
  this.altura = altura;
  this.largura = largura;
};

Pessoa.prototype = {
  getNome: ()=>{
    return this.nome;
  },
  getIdade: ()=>{
    return this.idade;
  },
  setName: (nome)=>{
    this.nome = nome;
  },
  setIdade: (idade)=>{
    this.idade = idade;
  }
};
```

```
Cachorro.prototype = new Pessoa();

var cachorro = new Cachorro(); // novo objeto criado a partir de pessoa
cachorro.setName('Toto');
console.log(cachorro.getNome());
```

DISCUSSÕES E RESULTADOS

A grande questão é uma comparação entre duas linguagens orientada a objeto e qual delas seria a melhor. Durante este trabalho observou-se que esta comparação acabou por tornar-se um paralelo. Cada uma delas possui suas peculiaridades como por exemplo o *JAVASCRIPT* não ser baseado em classe, mas somente em objeto (objeto literal), o que torna a torna diferente do *JAVA* que possui a classe como modelo para o objeto. Embora cada linguagem tenha sido desenvolvida para fins um pouco distintos, aos poucos uma "invadiu" a área da outra. O *JAVASCRIPT* encontrou uma forma dinâmica de assimilar o conceito de modelos(classes) para atrair desenvolvedores que estavam acostumados com o paradigma usada pelo *JAVA* e por outras linguagens como por exemplo o *PHP* e C#. O *JAVA* é uma linguagem robusta e completa que apesar de ser "verbosa", possui características de linguagem de alto nível com aplicabilidade em qualquer ambiente computacional.
O resultado trouxe um paralelo entre as duas linguagens mostrando faces de uma vertente diferente dentro do paradigma de programação orientada a objeto. Onde um desenvolvedor pode transitar entre estas duas linguagens sem grandes dificuldades.

CONCLUSÃO

As duas linguagens têm diferenças no que diz respeito a orientação a objetos, uma tem o conceito de classes (modelos) e objetos e outra somente em objetos (objeto literal), Em *JAVASCRIPT* a aplicação pode ser escrita de forma mais rápida, porém se não for bem organizada, a sua aplicação pode se tornar bagunçada e essa é uma desvantagem que o *JAVA* não possui.
Conclui-se que ambas são excelentes escolhas para se estudar e desenvolver e que suas características particulares são as diretrizes que movem a escolha por qual delas seguir dentro do paradigma da Programação Orientada a Objeto(POO).

REFERÊNCIA BIBILOGRAFICA

1. SANTOS, Rafael. Introdução à programação orientada a objetos usando Java 2a edição. Elsevier Brasil, 2013.

2. GUIMARÃES, Gleyser. A História da Linguagem JAVA. P@TNEWS - Departamento de Sistemas e Computação da Universidade Federal de Campina Grande (DSC-UFCG). Acessado em: 08/10/2017.

3. FLANAGAN, David. JavaScript: O guia definitivo. Tradução João Eduardo Nóbrega Tortello. Bookman Editora Ltda, 2007. 6ª edição. Av. Jerônimo de Ornelas, 670-Santana, 90040-340-Porto Alegre - RS.

4. ORACLE. JAVA SE Documentation. Acessado em 08/10/2017. Disponível em: <http://www.oracle.com/technetwork/pt/java/javase/documentation/index.html>.