

# FRAMEWORK CONVERTOR DE OBJETOS JAVA

**Cleverton Hoffmann, Rodrigo Curvello**

Instituto Federal Catarinense Campus Rio do Sul

clevertonhoffmann@com.br, rodrigo.curvello@ifc.edu.br

## ***Abstract.***

*This article aims to explain the creation and operation of the framework converter developed in the discipline of object-oriented programming II in the course of Computer Science of the Federal Institute Catarinense Campus Rio do Sul. The implemented framework uses concepts of structural and creative project standards, aiming at the transformation of Java objects into classes of any language generating the class file. Implemented initially for Python, PHP, JavaScript and C++, being able to be configured according to the need of the programmer for any language. It is thus realized through the implementation and theoretical investigation that the implementation of frameworks using project patterns and good programming practices are not always used by programmers, however the correct use of them and the knowledge about these standards facilitates the development of any framework or program.*

**Key-words:** *Computer Science; framework; project patterns.*

**Resumo.** *Este artigo tem como objetivo a explanação da criação e funcionamento do framework converter desenvolvido na disciplina de programação orientada a objetos II no curso de Ciências da Computação do Instituto Federal Catarinense Campus Rio do Sul. O framework implementado utiliza de conceitos de padrões de projetos estruturais e criacional, visando a transformação de objetos de Java em classes de qualquer linguagem gerando o arquivo da classe. Implementado inicialmente para Python, PHP, JavaScript e C++, podendo ser configurado conforme a necessidade do programador para qualquer linguagem. Percebe-se assim por meio da implementação e investigação teórica, que a implementação de frameworks utilizando padrões de projetos e boas práticas de programação nem sempre são utilizadas por programadores, porém o uso correto deles e o conhecimento sobre esses padrões facilita o desenvolvimento de qualquer framework ou programa desejado.*

**Palavras-chave:** *Ciências da Computação; framework; padrões de projetos.*

## **1. Introdução**

Programação Orientada a Objetos está presente nos mais diversos softwares e frameworks criados. Programas de computadores são desenvolvidos em grande escala atualmente. Um dos motivos desse crescimento refere-se as linguagens de programação. As mesmas podem ser divididas em linguagens de máquina, assembly, e de alto nível. Dentro dessas linguagens se destaca o Java que é uma linguagem de alto nível, tendo suas raízes baseadas em C++. (DEITEL, 2005). O Java é uma linguagem de

programação que tem algumas características como a portabilidade (pode ser executado em diferentes sistemas computacionais desde que tenha a máquina virtual Java instalada), multithreading (permite a execução de um mesmo programa simultaneamente), e suporte à comunicação (possibilita a programação em rede e comunicação). (SEBESTA, 2011). Como outras linguagens o Java também é conhecido como uma linguagem de programação orientada a objetos. (DEITEL, 2005)

Entre as características de programação orientada a objetos, destacamos as seguintes: **Objeto:** Pode ser comparado com um objeto do mundo real, um sinônimo de instância de uma classe. (MANSOUR, s.a.) Em outras palavras um objeto possui três características básicas, como estado, operações e identidade. O estado é definido pela sua configuração, em um tempo que as informações estão armazenadas. Cada objeto suporta um conjunto de operações. E a identidade é o que distingue um objeto de outro, podendo até ter o mesmo estado, mais os dados armazenados são diferentes. (GERMANIO, 2005). **Classe:** Segundo Germanio (2005, p.46): “A classe define uma coleção de objetos do mesmo tipo. [...]O que se faz, na prática é definir os campos e métodos de objetos nas classes às quais eles pertencem.” Uma instância de uma classe define um objeto. Em Java as instâncias das classes são definidas pela palavra new. E uma classe é definida pela palavra class e o nome da mesma. A classe deve sempre ter um construtor, que permite que o objeto seja configurado de forma correta e pode inicializar dados do objeto. Uma classe também possui métodos que ditam o funcionamento do objeto. (BARNES, KOLLING, 2009).

Assim como o Java outras linguagens também utilizam da programação orientada a objetos e da criação de classes, como por exemplo Python, C++, JavaScript, PHP, entre outras. Cada uma delas possui uma forma de escrita de classes, o JavaScript por exemplo entende tudo literalmente como uma variável. Os atributos em PHP são declarados com \$ em frente ao nome. E assim sucessivamente cada linguagem possui sua peculiaridade.

A busca da implementação deu-se por sugestão do professor da disciplina, buscando aplicar conceitos aprendidos sobre orientação a objetos. Assim entre os conceitos que ficaram implicitamente implementados no código destacamos a injeção de dependência, na qual utiliza-se de uma interface a qual as classes estruturais de cada linguagem alvo a serem geradas implementam, podendo assim ser gerado o código de qualquer linguagem.

Alguns dos padrões de projetos pretendidos a serem implementados implicitamente, foi o Builder e Factory método que são padrões de projeto de criação que significam que são responsáveis pelas técnicas de criação de classes, instanciação tornando o sistema independente dos objetos criados. Diferente dos padrões estruturais que são responsáveis pela estruturação das classes de forma a reutilização de código em tempo de execução. (CRUZ; PAZOTI; MARACCI, 2016).

Por mais que buscadas as boas práticas de programação, percebemos que existe uma distância entre conseguir aplicar todos os conceitos vistos na disciplina de forma prática. E principalmente na implementação, saber qual padrão de projeto correto a usar na implementação. (CRUZ; PAZOTI; MARACCI, 2016).

## 2. Metodologia

O trabalho foi desenvolvido como trabalho final da disciplina de Programação Orientada a Objetos II no curso superior de Bacharelado em Ciência da Computação do Instituto Federal Catarinense Campus Rio do Sul.

Para a implementação do código utilizou-se a pesquisa na internet em artigos científicos e também na documentação das linguagens utilizadas, Java, PHP, JavaScript, C++, Python.

A linguagem padrão utilizada no desenvolvimento do framework é o Java, usando a IDE NetBeans 8.2.

Inicialmente foram buscados os métodos de leitura de arquivos e escrita por meio da linguagem Java, bem como a forma de transformar um objeto em um array de string de atributos. Por fim deu-se a busca de como utilizar os conceitos de Padrão de Projetos aprendidos durante a disciplina no framework a ser desenvolvido.

Após a busca e implementação do framework principal, teve-se a necessidade da investigação de como as linguagens foco eram estruturadas em formato de classe.

A escrita do artigo se baseia em artigos científicos, visando a pesquisa qualitativa, explorando e explanando a implementação do framework.

Na Imagem 1 consta o diagrama de classe do Framework desenvolvido, para a elaboração do diagrama foi utilizada a ferramenta Astah.

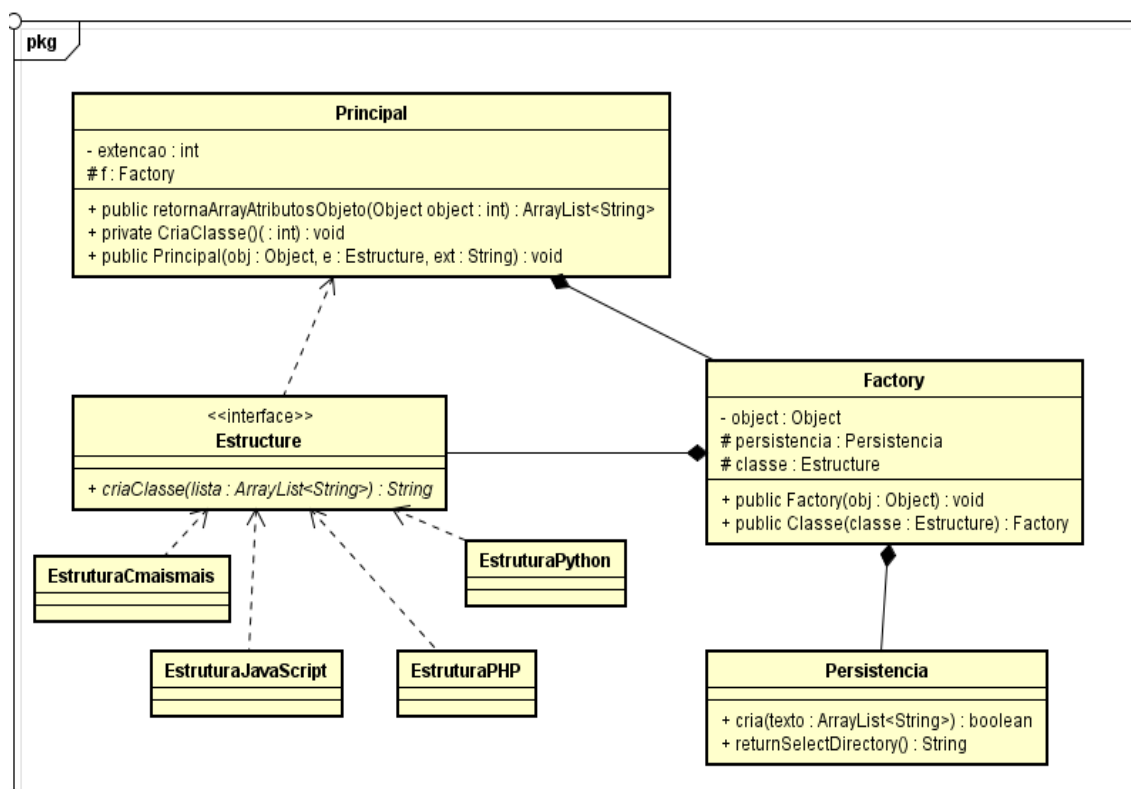


Imagem 1 – Diagrama de classes e métodos do Framework

### 3. Resultados e Discussões

Inicialmente foram desenvolvidos os métodos base, para a leitura e escrita de um documento de texto simples, bem como o retorno do diretório a salvar o documento. Para a criação de qualquer documento em um sistema operacional é necessário que ele tenha uma extensão e um nome, podendo ter ou não conteúdo dentro dele. Desta forma conseguimos salvar e gerar qualquer classe de linguagem apenas definindo o seu conteúdo e salvando o nome-ponto (Extensão do arquivo). Na Tabela 1 tem-se o método responsável por escrever e criar um novo arquivo, conforme na linha 11, cria um arquivo selecionando o local a salvar o arquivo, salvando com o nome da classe e extensão do arquivo, sendo estes últimos dois parâmetros passados como parâmetro no arrayList texto. Na linha 11 se cria o arquivo, na linha 29 é escrito no arquivo já criado.

```
1  /**
2   * MÉTODO RESPONSÁVEL POR CRIAR O DOCUMENTO DE UMA CLASSE
3   * @param texto ArrayList
4   * Posicao(0) = Nome da Classe
5   * Posicao(1) = Extensão do Arquivo
6   * Posicao(2) = String da Classe a ser gerada
7   * @return true or false criando o documento
8   */
9  public boolean cria(ArrayList<String> texto){
10
11      File f = new File("").
12      concat(this.returnSelectDirectory()).concat("\\").
13      concat(texto.get(0)).concat(texto.get(1));
14
15      try {
16          f.createNewFile();
17      } catch (IOException ex) {
18          return false;
19      }
20
21      PrintWriter pw;
22
23      try {
24          pw = new PrintWriter(new FileOutputStream(f));
25      } catch (FileNotFoundException ex) {
26          return false;
27      }
28
29      pw.append(texto.get(2));
30
31      pw.flush();
32      pw.close();
33
34      return true;
35  }
```

**Tabela 1 – Código método cria () contido na classe Persistência – Elaboração autor**

Na Tabela 2 tem-se o código que retorna um diretório selecionado, o mesmo é utilizado na criação do arquivo conforme Tabela 1 linha 11 onde que o método `this.returnSelectDirectory()` é chamado, método esse exposto na Tabela 2.

```
1  /**
2   * ABRE JANELA DE DIRETÓRIO PARA SALVAR
3   * @return String do diretório selecionado
4   */
5  public String returnSelectDirectory(){
6      JFileChooser fc = new JFileChooser();
7
8      // restringe a amostra a diretorios apenas
9      fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
10
11     int res = fc.showOpenDialog(null);
12
13     if(res == JFileChooser.APPROVE_OPTION){
14         File diretorio = fc.getSelectedFile();
15         return diretorio.getAbsolutePath();
16     }
17     else
18         return " ";
19 }
```

**Tabela 2 – Código método `returnSelectDirectory ()` na Persistência – Elaboração autor**

Visando utilizar injeção de dependência e a extensão do framework para a adição de novas linguagens usamos uma interface `Estructure`, a qual define o método que as classes que a implementam devem conter. Conforme a Tabela 3 a interface criada especifica um método `criaClasse ()` o qual retorna um `String` da estrutura da classe da linguagem foco, e receber como atributos, os nomes dos atributos do objeto Java, bem como o nome da classe e os tipos dos atributos, para a geração da estrutura.

```
1  /**
2   * Interface Responsável por fornecer dados
3   * para a criação da estrutura de classe
4   * em cada linguagem específica
5   * @author Cleverton
6   */
7  public interface Estructure {
8
9      /**
10     * Método a ser implementado para a geração do código em qualquer linguagem
11     * @param lista com:
12     * Nome da Classe;
13     * Nome de cada Atributo;
14     * Tipo de dados de cada atributo;
15     * @return String da estrutura da classe
16     */
17     public String criaClasse(ArrayList<String> lista);
18
19 }
```

**Tabela 3 – Código interface `Estructure ()` – Elaboração autor**

A próxima classe a ser implementada foi a Factory visando usar o padrão Factory método, buscando criar todos os objetos e instancias ao instanciar ela, bem como o padrão Builder e principalmente usando injeção de dependência.

```

1  /**
2   * Classe que fabrica as classes de persistência e classe estrutural bem como armazena
3   o objeto
4   * @author Cleverton
5   */
6  public class Factory {
7
8      protected Object objeto;
9      protected Persistencia persistencia;
10     protected Estructure classe;
11     /**
12      * Método responsável por inicializar a persistencia e preencher o objeto
13      * @param obj Objeto
14      */
15     public Factory(Object obj){
16         objeto = obj;
17         persistencia = new Persistencia();
18     }
19     /**
20      * Classe responsável por
21      * @param classe
22      * @return Factory
23      */
24     public Factory Classe(Estructure classe){
25         this.classe = classe;
26         return this;
27     }
28 }

```

**Tabela 4 – Código classe Factory – Elaboração autor**

Na linha 8 da Tabela 4 é criado uma variável que receberá a classe Java instanciada, o objeto, em seguida na linha 9 temos a criação da variável persistência, a qual receberá a persistência, bem como a interface estrutural na linha 10. No momento que a classe Factory é instanciada é criada a instancia da persistência e necessariamente passado o objeto como parâmetro, conforme método na linha 15 da Tabela 4. Dentro da Factory também se tem um método que recebe a classe a interface como parâmetro, a qual retorna a própria classe Factory, conforme linha 24 da Tabela 4. A classe Factory apenas é instanciada pela classe principal, a qual utilizará dela para manipulação dos dados.

A classe principal que possui a lógica de manipulação das classes instanciadas será explanada conforme os métodos que a mesma possui. Inicialmente conforme a Tabela 5 mostra o construtor da classe à qual inicializa o atributo da classe Factory, a mesma possui e preenche a variável extensão do arquivo a ser criado. Na construção executa o método criaClasse, método que consta na Tabela 7.

```

1  /**
2   * Classe principal da Frame a ser Instanciada
3   * @author Cleverton
4   */

```

```

5 public class Principal {
6
7     private Factory f;
8     private String extensao;
9     /**
10      * Contrutor da classe principal
11      * @param obj Classe em Java
12      * @param e Arquivo de Configuração da Linguagem
13      * @param ext Extensão do arquivo da Linguagem a ser gerada a classe
14      */
15     public Principal(Object obj, Estructure e, String ext){
16         f = new Factory(obj).Classe(e);
17         extensao = ext;
18         this.CriaClasse();
19     }

```

**Tabela 5 – Parte inicial do código classe Principal com construtor – Elaboração autor**

Ao chamar o método `criaClasse` no construtor dentro dele existe a chamada ao método `retornaArrayAtributosObjeto(Object object)` que é o método implementado na Tabela 6, o mesmo é responsável por decodificar o objeto e extrair as informações das variáveis, nome conforme linha 14, as informações do tipo dos atributos linha 16 e o nome da classe linha 11. Retornando um Array com o nome da classe na posição 0 bem como os atributos e seus tipos em seguida.

```

1     /**
2      * RETORNA ARRAY DE STRING DO NOME DA CLASSE E DOS ATRIBUTOS E
3      SEUS TIPOS
4      * @param object
5      * @return String de atributos e seus tipos com o nome da classe na primeira posição
6      */
7     public ArrayList<String> retornaArrayAtributosObjeto(Object object){
8         Class<?> classe = object.getClass();
9         Field[] campos = classe.getDeclaredFields();
10        ArrayList<String> valores = new ArrayList<>();
11        valores.add(object.getClass().getSimpleName()); //Nome da Classe
12        for (Field campo : campos) {
13            //Pega o nome do atributo
14            valores.add(campo.getName());
15            //Pega o tipo do atributo
16            valores.add(campo.getAnnotatedType().getType().getTypeName().replaceAll("java.lang.",
17            ""));
18        }
19        int k=0;
20        //Parte de verificar o nome Double maiúsculo e transformar minúsculo para
21        padronizar as variáveis
22        for(String a: valores){
23            if(a.equals("Double")){
24                valores.set(k, "double");
25            }
26            k++;
27        }
28        return valores;
29    }

```

**Tabela 6 – Método `retornaArrayAtributosObjeto ()` da classe principal – Elaboração autor**

O Método `criaClasse` especificado na tabela 7 é responsável montar o array com as informações e conteúdo da classe em foco na linguagem a ser gerada, ele é privado até para possibilitar acesso interno somente a classe principal, evitando possíveis erros.

```
1  /**
2   * Classe principal responsável por trabalhar com os métodos para criar classe dos
3   objetos
4   */
5   private void CriaClasse(){
6       ArrayList<String> valores = this.retornaArrayAtributosObjeto(f.objeto);
7       String estrutura = f.classe.criaClasse(valores);
8       String nome = valores.get(0);
9       valores.clear();
10      valores.add(nome);
11      valores.add(extensao);
12      valores.add(estrutura);
13      f.persistencia.cria(valores);
14  }
15
16 }
```

**Tabela 7 – Método `criaClasse` na classe Principal – Elaboração autor**

Após a criação da classe os testes necessários para funcionamento foram executados, tanto usando a biblioteca do Java JUnit de testes unitários para a verificação do funcionamento dos métodos utilizados, como a própria classe `main ()`.

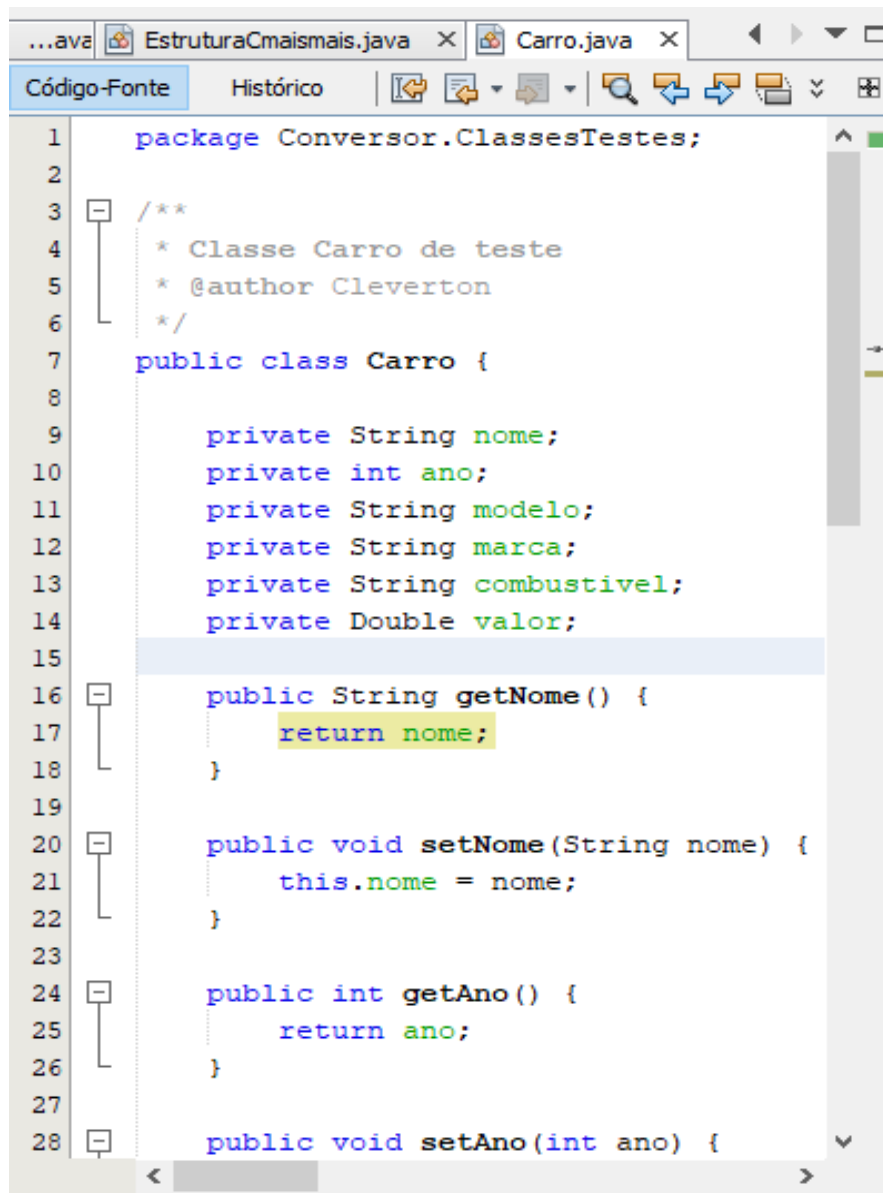
A classe `main ()` instancia o objeto em Java e passa ele como parâmetro para a classe principal a qual se encarrega de criar o documento de acordo com a string da classe estrutural da linguagem requerida. Na Tabela 8 vemos a classe `main ()`, onde é definido o tipo da linguagem a gerar a classe do objeto passado no método, bem como a extensão do arquivo.

```
1  public class Main {
2      public static void main(String[] args) {
3
4          Carro c = new Carro();
5          // Pessoa pes = new Pessoa();
6          // Principal p = new Principal(pes, new EstruturaPython(), ".py");
7          Principal p = new Principal(c, new EstruturaPython(), ".py");
8          // p = new Principal(c, new EstruturaPHP(), ".php");
9          // p = new Principal(c, new EstruturaJavaScript(), ".js");
10         // p = new Principal(c, new EstruturaCmais(), ".cpp");
11     }
12 }
```

**Tabela 8 – Método `Main` – Elaboração autor**

Ao executar a classe `main ()` temos o objeto Carro em Java transformado nas demais linguagens. A classe Carro em Java consta na Figura 1.



The image shows a screenshot of a Java IDE window. The title bar at the top displays three open files: "...ava", "EstruturaCmaismais.java", and "Carro.java". The "Código-Fonte" (Source Code) tab is active. The code is written in Java and defines a class named "Carro" within the package "Conversor.Classestestes". The class has several private attributes: "nome" (String), "ano" (int), "modelo" (String), "marca" (String), "combustivel" (String), and "valor" (Double). It also includes four public methods: "getNome()" which returns the "nome" attribute, "setNome(String nome)" which sets the "nome" attribute, "getAno()" which returns the "ano" attribute, and "setAno(int ano)" which sets the "ano" attribute. The code is formatted with standard Java syntax highlighting, and the IDE interface includes a line number margin on the left and a scrollbar on the right.

```
1 package Conversor.Classestestes;
2
3 /**
4  * Classe Carro de teste
5  * @author Cleverton
6  */
7 public class Carro {
8
9     private String nome;
10    private int ano;
11    private String modelo;
12    private String marca;
13    private String combustivel;
14    private Double valor;
15
16    public String getNome() {
17        return nome;
18    }
19
20    public void setNome(String nome) {
21        this.nome = nome;
22    }
23
24    public int getAno() {
25        return ano;
26    }
27
28    public void setAno(int ano) {
```

**Figura 1 – Classe carro.java (JAVA) – Elaboração autor.**

Algo importante a se destacar ainda é sobre a interface ela é implementada pelas classes específicas das linguagens alvo. Isso significa, por exemplo, que a classe estruturaPython, instanciada no main é responsável por receber o array com as propriedades de atributos e nome do objeto Java instanciado, criando uma String da classe do tipo Python, que é gravada em um arquivo NomeClasse.py.

Além da criação das classes, faz-se necessário o teste com as classes geradas para verificar o funcionamento das mesmas. Assim para teste de cada uma das classes geradas foi utilizada um programa online ou compilador da linguagem e construção do código executor de cada linguagem. As classes geradas estão disponíveis no GitHub.

Para teste da classe carro.js, foi utilizado um executor online disponível no link <<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class>>, em que o resultado da execução mostra-se na Figura 3.



```
54  * @return {string}
55  */
56  Carro.prototype.toString = function () {
57      return "Carro{" + "nome=" + this.nome + ", ano=" + this.ano + ", modelo=" + this.modelo + ", marca=" + th
58  };
59  return Carro;
60  }();
61  Carro["__class"] = "Carro";
62
63  console.log(Carro.prototype);
64  // expected output: 12
65
```

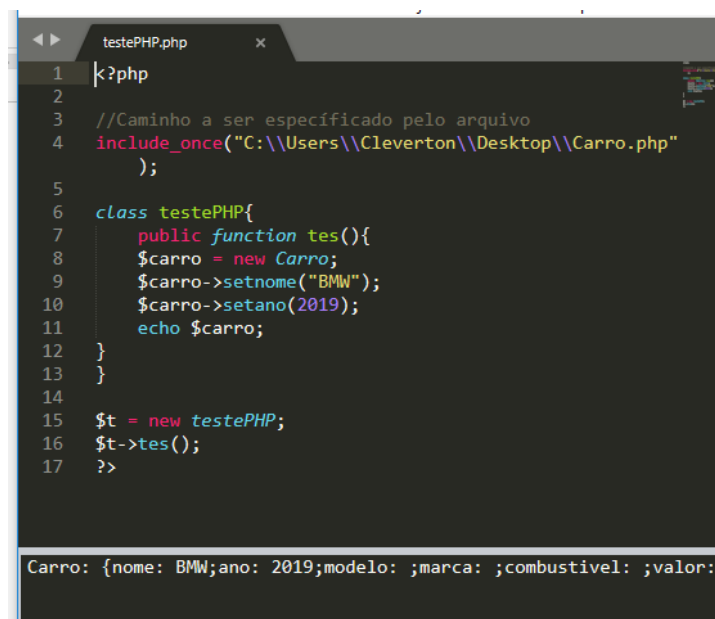
Run >

Reset

> Carro{nome=undefined, ano=undefined, modelo=undefined, marca=undefined, combustivel=undefined, valor=ur

**Figura 3 – Execução teste classe Carro.js – Elaboração autor**

Para a execução da classe carro.php utilizou-se o editor de texto Sublime Text, o qual foi baixado o compilador do PHP integrado ao mesmo, e no mesmo foi criada a classe teste e executado a classe PHP gerada, conforme a Figura 4. A classe testePHP.php também se encontra disponível no GitHub.

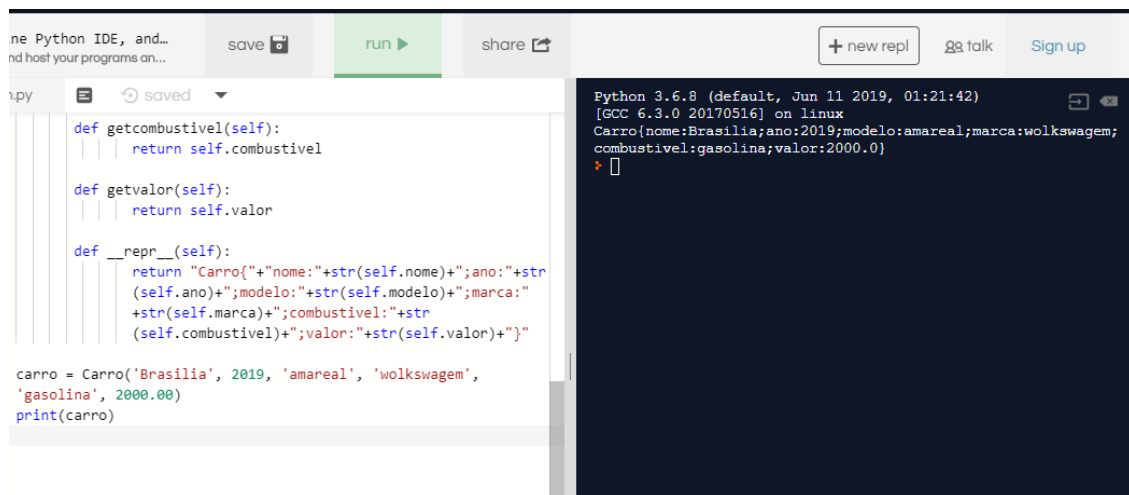


```
1  <?php
2
3  //Caminho a ser especificado pelo arquivo
4  include_once("C:\\Users\\Cleverton\\Desktop\\Carro.php"
5  );
6
7  class testePHP{
8      public function tes(){
9          $carro = new Carro;
10         $carro->setnome("BMW");
11         $carro->setano(2019);
12         echo $carro;
13     }
14 }
15
16 $t = new testePHP;
17 $t->tes();
18 ?>
```

Carro: {nome: BMW;ano: 2019;modelo: ;marca: ;combustivel: ;valor:

**Figura 4 – Execução teste classe Carro.php com a classe Teste.php – Elaboração autor**

Para o teste da classe Carro.py gerada na linguagem Python, foi utilizado um compilador e executor online disponível no link: <<https://repl.it/languages/python3>>, sendo definido alguns valores para as variáveis por meio do construtor da classe e em seguida dado um print na classe, possibilitando assim o uso do método `__repr__` da classe Carro.py, similar ao `toString` da linguagem Java. Na Figura 5 podemos ver o teste realizado da classe Carro.py.



```
Python 3.6.8 (default, Jun 11 2019, 01:21:42)
[GCC 6.3.0 20170516] on linux
Carro(nome:Brasilia;ano:2019;modelo:amareal;marca:volkswagem;
combustivel:gasolina;valor:2000.0)
>
```

```
def getcombustivel(self):
    return self.combustivel

def getvalor(self):
    return self.valor

def __repr__(self):
    return "Carro("+nome:"+str(self.nome)+";ano:"+str
(self.ano)+";modelo:"+str(self.modelo)+";marca:"
+str(self.marca)+";combustivel:"+str
(self.combustivel)+";valor:"+str(self.valor)+")"
```

```
carro = Carro('Brasilia', 2019, 'amareal', 'volkswagem',
'gasolina', 2000.00)
print(carro)
```

**Figura 5 – Execução teste classe Carro.py online com alguns valores – Elaboração autor**

A classe Carro.cpp é a única não usual, devido a não familiaridade com a linguagem, teve-se dificuldade de compreender a escrita da mesma. E a estrutura de criação dela estará em desenvolvimento e será desenvolvida futuramente.

## 4. Conclusão

O trabalho mostra de forma clara a importância da programação orientada a objetos na implementação de frameworks utilizando padrões de projetos e boas práticas de programação, sendo que estes nem sempre são utilizadas por programadores. O uso correto deles e o conhecimento sobre esses padrões facilita o desenvolvimento de qualquer framework ou programa desejado.

Percebeu-se durante o desenvolvimento a dificuldade de implementar um código em outra linguagem quando não se tem o conhecimento da linguagem. Para acrescentar outras linguagens a framework o programador deve conhecer a linguagem e sua estrutura antes de acrescenta-la, pois deverá implementar a estrutura texto em formato de string que gera a classe alvo a partir de um objeto de uma classe em Java. Também pode-se destacar que o trabalho e a framework não se considera finalizada, pois sempre existe necessidade de melhorias e aperfeiçoamento. E até a possível implementação de padrões de projetos diferentes. Para um cientista da computação destaca-se a necessidade do conhecimento dos padrões e de atualização, principalmente para a carreira de desenvolvedor de frameworks e sistemas programados.

## Referências

BARNES, David J. KOLLING, Michael. **Programação orientada a objetos com Java**. 4. Ed. São Paulo: Person Prentice Hall, 2009.

CRUZ, Caroline Teixeira da; PAZOTI, Mario Augusto; MARACCI, Francisco Virginio. **LEPATTERN -UMA FERRAMENTA PARA ENSINO DE PADRÕES**

**DE PROJETO: IMPLEMENTAÇÃO DOS PADRÕES STRATEGY E FACADE.** 2016. Disponível em: <<http://journal.unoeste.br/index.php/ce/article/view/1550/1612>>. Acesso em: 04 jul. 2019.

DEITEL, Harvey M. DEITEL, P.J. **Java: Como programar.** 6. Ed. São Paulo: Person Prentice Hall, 2005.

GERMANIO, Leibnitz. **Implementação orientada a objetos da solução de problemas estruturais dinâmicos via método dos elementos finitos.** Disponível em: <<http://www.bibliotecadigital.ufmg.br/dspace/bitstream/handle/1843/BUOS-8CJHM3/174.pdf?sequence=1>>. Acesso em: 09 dez. 2018.

MANSOUR, Isabel H. **Paradigma orientado a objetos.** Disponível em: <<https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula12.pdf>> Acesso em: 08 dez. 2018.

SEBESTA, Robert W. **Conceitos de linguagens de programação.** 9. Ed. Porto Alegre: Bookman, 2011.

**Link GitHub Projeto** (Possui um documento referencias.txt todas as referências de sites pesquisados para a implementação das classes específicas de cada linguagem):

<<https://github.com/ClevertonHoffmann/Framework-Projeto-Converter-----JAVA.git>>