JAVA: Introdução, Características, Estrutura e Ambientes de Programação

Hélder Alves Couto
Discente do Curso Engenharia da Computação
Faculdades Integradas de Três Lagoas (AEMS)

Thiago Alves Rodrigues
Discente do Curso Engenharia da Computação
Faculdades Integradas de Três Lagoas (AEMS)

Alan Pinheiro de Souza
Docente do Curso Engenharia da Computação
Faculdades Integradas de Três Lagoas (AEMS)
Mestre em Informática pela Universidade Federal do Rio de Janeiro (UFRJ)

Olavo Alves dos Santos Neto
Docente do Curso Engenharia da Computação
Faculdades Integradas de Três Lagoas (AEMS)

Diego Moreira Rodrigues
Docente do Curso Engenharia da Computação
Faculdades Integradas de Três Lagoas (AEMS)

RESUMO
Este projeto tem como objetivo realizar um levantamento bibliográfico referente às características e o contexto como plataforma de desenvolvimento da linguagem de programação Java. É abordado o conceito de multiplataforma, a estrutura de programação, exemplo de código Java e o ambiente de desenvolvimento da linguagem. O artigo discute ainda o contexto da orientação a objetos, principal característica dessa linguagem, tendência adotada como padrão de mercado no desenvolvimento de aplicações, uma síntese das principais divisões do Java e suas evoluções.

PALAVRAS-CHAVE: Java; História; Orientação a Objetos; Estrutura da Linguagem; Ambiente de Programação.

INTRODUÇÃO

A linguagem de programação Java foi idealizada para ser menor em linhas de código, mais simples e mais confiável para o desenvolvimento de aplicações. Preservou a característica de orientação a objetos[1], herdada da linguagem C++. A linguagem C++ permite o desenvolvimento orientado a objetos, porém, este não é um requisito obrigatório desta linguagem, sendo possível continuar programando de forma estrutural em sua plataforma. Nesse sentido, Java muda o conceito e introduz

como uma de suas características de estrutura básica a programação orientada a objetos, conforme Gomes (2004, p.01).

A linguagem foi lançada em 23 de Maio de 1995, pela *Sun Microsystem*, tendo como base ser simples, neutra, e com execução em vários tipos de *hardware*, propiciando a idéia de portabilidade, ou seja, a execução da mesma aplicação nos mais diversos sistemas operativos e/ou *hardwares,* tanto robustos, quanto básicos, necessitando apenas da máquina virtual Java instalada no sistema que receberá a aplicação, conforme Furgeri (2010, p.21).

Java foi desenvolvida a princípio para uso em dispositivos eletrônicos embarcados, como fornos, micro-ondas, torradeiras e sistemas de televisão, porém, com o surgimento da *Internet* a partir de 1993, a *Sun Microsystem* (*Oracle*) identificou o potencial em utilizar a linguagem para adicionar conteúdo dinâmico às páginas *web*. Com isso, lançou em 1995 a linguagem e, não apenas como uma ferramenta de programação, mas como uma nova plataforma de desenvolvimento, possibilitando adicionar interações e animações às exibições em navegadores, tornando o projeto ideal para desenvolver páginas mais aprazíveis, conforme Deitel e Deitel (2010, p.06).

A motivação da pesquisa sobre a linguagem Java foi devido ao grande sucesso e ascensão pelo mercado desenvolvedor (FURGERI, 2010, p.18), e ainda sua semelhança com a linguagem C++.Java traz características quem se assemelham a outras linguagens, porém, a linguagem aborda a portabilidade já como conceito nativo em sua plataforma, outro fator e a possibilidade de uso em conjunto com outras linguagens, a grande semelhança também com a linguagem C, uso de *frameworks*[2] que visam facilitar e automatizar o processo de desenvolvimento das aplicações e as divisões da linguagem (JSE[3], JEE[4], JME[5] e JavaFX[6]).

O objetivo que levou a produção deste artigo foi elucidar tópicos relevantes sobre a linguagem Java, com relação ao seu lançamento e as evoluções da linguagem ao longo dos últimos anos. Portanto, a metodologia de pesquisa do artigo adotada é um levantamento bibliográfico e está dividida em três seções. A primeira seção é destinada a referenciar as características da linguagem, que abordará a evolução, o paradigma de orientação a objetos, portabilidade, alto desempenho, multiprocessamento e segurança.

A segunda seção referenciará a estrutura de programação, destacando um exemplo de programa Java simples, os tipos de dados, as declarações das

variáveis, tipos de operadores aritméticos, relacionais e lógicos. Nesta seção não serão abordados os temas suporte a comunicação, coletor de lixo, operadores de atribuição composta, ternário e *bitwise*, *strings*, entrada e saída, fluxos de controle, números grandes e *arrays*, devido a sua complexidade que demandará mais estudos e detalhes sobre o assunto e ficará para abordagem em trabalhos futuros. A terceira seção referenciará dois ambientes dos mais utilizados pelos programadores no desenvolvimento de aplicações usando a linguagem de programação Java.

1. CARACTERÍSTICAS DA LINGUAGEM

As linguagens de programação são divididas em quatro categorias: imperativas, funcionais, lógicas e orientada a objetos. Java se enquadra na categoria de orientação a objetos, um paradigma de programação já sólido no mercado desenvolvedor e a maioria das linguagens atuais permitem trabalhar desta forma (FURGERI, 2010, p.19). Além da orientação a objetos, outras características desta linguagem são: funcionalidades *web*, independência de plataforma, alto desempenho, segurança, oferta de recursos com múltiplas rotinas e a coleta de lixo, conforme Junior (2007, p.22) e Sebesta (2011, p.42).

Desde o lançamento da linguagem ocorreram alterações substanciais em sua estrutura, desde a ampliação de sua API[7] (*Application Programming Interface*) até um novo esquema de numeração. A partir da versão 5, recebeu o codinome *Tiger*, então chamada de Java 5.0. A versão 6.0, ficou conhecida como *Mustang*. A versão 7.0, recebe o nome *Dolphin* que também recebeu várias melhorias na API. É possível ver na Tabela 1 essas alterações nas versões do Java, sendo esta última versão aquela que obteve maior implementação, conforme Junior (2013, p.19).

Neste contexto de melhorias é possível ainda ver a progressão na versão interna da linguagem, a mudança nos nomes de referência de cada versão e as características abordadas em cada versão. Desta forma, segundo a *Sun* as alterações refletiriam no nível de maturidade da linguagem, estabilidade, escalabilidade e segurança da plataforma, indicando as perspectivas de longo prazo desta tecnologia e também seu sucesso, segundo Junior (2013, p.20) e Deitel e Deitel (2010, p.07).

Tabela 1: Versões da plataforma Java.

Ano	Versão	Versão Interna	Nome da Versão	Características da Linguagem
1996	1.0	1.0/44	Oak	Versão inicial
1997	1.1	1.1/45		Classes internas
1998	1.2	1.2/46	Playground	Declaração strictfp. Reflexão. Compilação JIT
2000	1.3	1.3/47	Kestrel	Tecnologia *HotSpot* para JVM (Java *Virtual Machine*)
2002	1.4	1.4/48	Merlin	Diretiva assert. Encadeamento de exceções.
2004	5.0	1.5/49	Tiger	*Autoboxing*. Enumerações. Genéricos. *Metadata* (Anotações). Lista variável de argumentos.
2006	6.0	1.6/50	Mustang	Suporte para *scripting*
2011	7.0	1.7/51	Dolphin	*Strings* em *switch*. *Automatic resource management* (ARM) ou*try-with-resources*. *Multi-catch*. Inferência de tipos genéricos. *Varargs* melhorado. Novos literais.

Fonte: Retirado de Junior (2013, p.19).

A linguagem Java oferece suporte tanto para inserção de dados de objetos quanto de não objetos, porém isso pode tornar um tanto quanto complexo para o programador. Para resolver essa situação em que é preciso usar objetos com dados do tipo primitivo, valores numéricos, caracteres simples e valores lógicos, é incluído em sua API um conjunto de classes denominado *wrapper*, capaz de encapsular os tipos de dados primitivos na forma de objetos da linguagem (SEBESTA, 2011, p.569). Assim para utilizar os tipos de dados primitivos é necessária a sua alocação com o operador *new* (novo), ver Figura 1 (JUNIOR, 2007, p.128), e tipos de dados primitivos com sua respectiva classe *wrapper*, ver Tabela 2 (JUNIOR, 2007, p.128).

Figura 1: Exemplo de alocação usando tipos de dados primitivos.

```
Integer inteiro = new Integer(4);
Double real = new Double(9.62);
Boolean logico = new Boolean("true");
Character caractere = new Character('J');
```

Fonte: Retirado de Junior (2007, p.128).

Tabela 2: Tipos de dados primitivos e a respectiva classe *wrapper*.

	Tipo Primitivo	Classe *Wrapper*
Lógico	Boolean	Boolean
Caractere	Char	Character
Integral	Byte	Byte
	Short	Short
	Int	Integer
	Long	Long
Ponto Flutuante	Float	Float
	Double	Double

Fonte: Retirado de Junior (2007, p.128).

Java também é portátil, pois a linguagem não depende de uma plataforma específica para execução, compilações, ao invés disso os códigos-fonte das aplicações desenvolvidas são compiladas para uma forma intermediária de código destinada à JVM. Esse formato intermediário é chamado de *bytecodes*[8]. Desta forma, uma mesma aplicação desenvolvida com a linguagem pode ser executada em qualquer tipo de arquitetura, bastando apenas à máquina virtual instalada no sistema e uma versão da máquina compatível com a plataforma Java utilizada, conforme Junior (2007, p.23) e Deitel e Deitel (2010, p.09-11).

Outra característica a ser observada sobre a linguagem Java é seu alto desempenho. Projetada para ser simples, neutra e compacta, independente de plataforma e utilização em rede, Java oferece excelente desempenho, devido à incorporação de um compilador JIT (*Just In Time*) na máquina virtual. Sendo assim, é possível converter os *bytecodes* em código nativo durante a carga da aplicação, melhorando consideravelmente a execução das aplicações escritas em código Java, conforme Ritchey (1997, p.08) e Junior (2007, p.23).

Além do compilador JIT incorporado para melhorar o desempenho, na execução das aplicações, é possível o uso de *multithreading* (multiprocessamento), ou seja, a execução de múltiplas rotinas concorrentemente, inclusive com sincronização, que por sua vez é um mecanismo capaz de controlar a ordem das tarefas quando são executadas. Deste modo, cada rotina é um fluxo de execução independente, que acontece simultaneamente em um programa. Essa concorrência no nível de sentença é um conjunto relativamente simples que trabalha em laços com sentenças inclusas que operam em elementos de uma matriz, conforme Sebesta (2011 p.597), Junior (2007, p.23) e Furgeri (2010, p.20).

Segurança é outro qualificador da linguagem. Pelo simples fato de aplicações poderem ser obtidas em uma rede, por meio da *Internet* ou outras fontes não confiáveis, o quesito segurança dessas aplicações fica comprometido. Ao levar em consideração o contexto e os problemas ocasionados por ataques intencionais, Java possui mecanismos de segurança intrínseco para minimizar esse problema e evitar que haja qualquer alteração e operação no sistema de arquivos da máquina-alvo, diminuindo consideravelmente os problemas relativos com a segurança, assim tal mecanismo é flexível o suficiente para determinar se uma aplicação é considerada segura, segundo Junior (2007, p.23) e Ritchey (1997, p.08).

2. ESTRUTURA DE PROGRAMAÇÃO

Um dos critérios para verificar a importância de uma linguagem de programação é a facilidade que os programadores terão para analisar e entender os códigos-fonte de um programa. Deve-se analisar o projeto que será desenvolvido e, de posse dos detalhes, escolherem a linguagem adequada a se utilizar, considerando suas características e sua estrutura de programação, ou seja, que tenha eficiência e legibilidade de máquina em um cenário de desenvolvimento, conforme Sebesta (2011, p.27).

A estrutura de programação do Java exige alguns padrões que devem ser seguidos, desde a instalação correta do Java *Development Kit* (JDK, 2016), atenção nas declarações das variáveis, tidos de dados, tipos de operadores, *Strings* (objeto formado por uma sequência de caracteres), entrada e saída de dados, além de outras formas pré-definidas. A instalação da plataforma Java, JDK, é simples, basta baixar o pacote no próprio *site* da *Oracle*, realizar a instalação e escolher qual ambiente de desenvolvimento integrado será utilizado, conhecido também como IDE (*Integrated Development Environment*).

O Java é *sensitive case*, ou seja, diferencia letras maiúsculas de minúsculas, então muita atenção no momento de começar a programar com a linguagem Java. Existem algumas diferenças da linguagem relativas à linguagem C++, por exemplo, todas as funções são métodos de alguma classe, o método *main* sempre é estático, e como no C++ a palavra *void* sempre indica que este método não possui valor de retorno (HORSTMANN, 2009, p.685). Para análise desses conceitos é apresentado, na Figura 2, um código Java simples para melhor compreensão (JUNIOR, 2007, p.27).

Figura 2: Exemplo de código-fonte simples da linguagem.

```
1   //Declaração da classe pública Ola
2   public class Ola {
3       //Declaração do método main, que dá início ao programa
4       public static void main(String[] args) {
5           //impressão da mensagem
6           System.out.println("Início de Aprendizado!");
7           //impressão da mensagem
8           System.out.println("Artigo sobre a Linguagem de Programação Java!");
9       } //fecha as chaves do método main
10  } //fecha as chaves da classe Ola
```

Fonte: Adaptado de Junior (2007, p.27).

A segunda linha do código exemplo, *public class* Ola, inicia uma nova classe, a palavra *public* indica que a classe pode ser utilizada pelo “publico”, ou seja, por outras classes ou métodos (conjunto de instruções que descreve como executar uma tarefa). Já a quarta linha do código, *public static void main (String [ ] args)*, define um método chamado *main*, este indica ao compilador o ponto inicial da execução da classe. O parâmetro *String*[ ] *args* é uma parte obrigatória do método *main* que contém os argumentos da linha de comando. A palavra *static* é um qualificador ver que indica que o método pertence à classe e é controlado apenas por esta classe.

A sexta e oitava linha que contém os códigos: *System.out.println(“*Início de Aprendizado!*”)* e *System.out.println(“*Artigo sobre a linguagem de programação Java!*”)*, são instruções que imprimi uma linha de texto cada, “Início de Aprendizado!” e “Artigo sobre a linguagem de programação Java!”, porém faz-se necessária a indicação da saída do sistema para exibir a mensagem, neste caso, é utilizado um objeto chamado *out* que foi inserido pelos projetistas da biblioteca Java na classe *System* esta que contém diversos objetos e métodos para serem utilizados no momento de escrita dos códigos-fonte.

Todavia até o momento a mensagem não foi exibida para o usuário, ela foi armazenada para sua exibição na tela do usuário. Desta forma o método que o faz é chamado de *print*, que exibe a mensagem na tela do usuário, ou *println*. A aglutinação gerada com a adição de *ln* faz com seja inserida uma linha ao final da exibição de cada mensagem para o usuário, separando as mensagens umas das outras, no momento de sua execução. Para a correta compilação do código faz-se necessário à abertura e fechamento das chaves, {…}, conforme explicação nos comentários feitos utilizando barras duplas (//), esses comentários não são interpretados pelo compilador.

Java também se utiliza de expressões, ou seja, palavras-chave e palavras reservadas que não são possíveis de utilização como nomes de variáveis, métodos ou outros operadores que não seja pelas definições pré-estabelecidas, por exemplo: *class*, *public, void, static, String, args, System, out* e *print*. É importante destacar ainda que algumas palavras como *true, false* e *null* ás vezes são confundidas como palavras-chave, nesse caso estas palavras são apenas literais reservados. Houve também a introdução em Java 5.0 da palavra-chave *enum* (Enumerações) usada em termos simples como um conjunto de objetos que representam um conjunto de

escolhas (LIGUORI; LIGUORI, 2016, p.10; HELLER; ROBERTS, 2004, p.07). Para mais expressões ver Tabela 3, conforme Heller e Roberts (2004, p.07).

Tabela 3: Palavras-chave e palavras reservadas de Java.

abstract	Default	If	Private	throw
Assert	Do	Implements	Protected	throws
boolean	Double	Import	Public	transient
Break	Else	instanceof	Return	true
Byte	Extends	Int	Short	try
Case	False	interface	Static	void
Catch	Final	Long	strictfp	volatile
Char	Finally	Native	super	while
Class	Float	New	switch	
Const	For	Null	synchronized	
continue	Goto	package	This	

Fonte: Retirado de Heller e Roberts (2004, p.07).

Os tipos de dados também devem ser bem definidos, por Java ser fortemente tipada é possível definir um determinado tipo de dado de forma errônea, desencadeando diversos erros de compilação e de recebimentos desses dados pelo programa. Há vários tipos de dados, de tipagem, nesse caso, o Java contém oito tipos primitivos, sendo quatro deles do tipo inteiro e dois do tipo numérico de ponto flutuante[9]. Um único caractere pode ser do tipo *char*, e o número um (1) é um tipo considerado booleano (*boolean*) com valor relativo a verdadeiro e o zero (0) com valor relativo falso, conforme Horstmann e Cornell (2010, p.22).

As declarações das variáveis no Java seguem padrões em seus tipos, as variáveis são identificadores cujo nome, escolhido pelo programador, é associado a um valor que pertence a certa tipagem de dados, ou seja, um identificador, seja ele uma variável, é a localização, endereço capaz de armazenar o valor de certo tipo. Java exige que todos os identificadores tenham um tipo de dado definido antes da sua utilização, que recebe um nome no momento da sua declaração ou criação. Em geral, a variável recebe um nome que a define, exemplo: *int idade* (a palavra idade é o nome definido para variável e o seu tipo *int*, por ser do tipo inteiro), conforme Furgeri (2010, p.32).

A linguagem Java oferece ainda uma gama de operadores usados para a realização de diversas operações desde aritmética, lógica e relacional. Os operadores aritméticos são destinados para cálculos matemáticos, como adição, subtração, multiplicação, divisão, resto da divisão, sinal negativo, sinal positivo,

incremento unitário e decremento unitário. Esses operadores podem ser observados na Tabela 4 (JUNIOR, 2007, p.42).

Tabela 4: Operadores aritméticos em Java.

Função	Operador	Exemplo
Adição	+	X + Y
Subtração	-	X – Y
Multiplicação	*	X * Y
Divisão	/	X / Y
Resto da divisão inteira	%	X % Y
Sinal negativo	-	-X
Sinal positivo	+	+X
Incremento unitário	++	++X ou X++
Decremento unitário	--	--X ou X--

Fonte: Retirado de Junior (2007, p.42).

Os operadores relacionais permitem a comparação de valores literais, variáveis ou até mesmo o resultado de expressões retornando um resultado da comparação do tipo lógica. É feita uma análise para verificar se o resultado da operação é verdadeiro ou falso, como ilustrado na Tabela 5, conforme Junior (2007, p.44).

Tabela 5: Operadores relacionais.

Função	Operador	Exemplo
Igual	==	X == Y
Diferente	!=	X != Y
Maior que	>	X > Y
Maior ou igual a	>=	X >= Y
Menor	<	X < Y
Menor ou igual a	<=	X <= Y

Fonte: Retirado de Junior (2007, p.44).

Operadores lógicos permitem avaliar o resultado de diferentes operações aritméticas ou relacionais, construindo uma expressão como resultado, composta de várias partes e mais complexa. Esses operadores estão descritos na Tabela 6, conforme Junior (2007, p.46).

Tabela 6: Operadores lógicos.

Função	Operador	Exemplo
E lógico (and)	&&	X && Y
Ou lógico (or)	||	X || Y
Negação (not)	!	!X

Fonte: Retirado de Junior (2007, p.46).

3. AMBIENTE DE DESENVOLVIMENTO

Como acontece com outras linguagens de programação, em Java também é possível utilizar ambientes de desenvolvimento. Esses ambientes facilitam a vida dos programadores no momento da escrita dos códigos-fonte, pois, é possível a escrita do código e a compilação em tempo real, desta forma simplificando a análise e verificação de possíveis erros existentes na escrita. De forma geral, os programas passam por cinco fases: edição, compilação, carregamento, verificação e execução, segundo Deitel e Deitel (2010, p.08).

Por existir diversos ambientes gráficos de desenvolvimento, serão referenciados os dois mais utilizados: Eclipse[10](ECLIPSE, 2016) e Netbeans[11] (NETBEANS, 2016). São dois ambientes que possuem grande aceitação por parte do mercado de desenvolvimento pela sua facilidade instalação e de uso. É possível realizar o *download* dos ambientes de desenvolvimento gráfico por intermédio dos *sites* destas empresas. E, apesar da existência desses ambientes de programação, ainda é possível a escrita de código-fonte e teste sem o uso destes, usando as variáveis de ambiente, conforme Furgeri (2010, p.22).

Para programar sem o uso destes ambientes, ainda é necessária à instalação do pacote JDK, ferramenta composta por um compilador Java (*javac*), máquina virtual Java (*java*), gerador de documentação (*javadoc*), processador de anotações (*apt*), visualizador de *applets*[12](*appletviewer*), criador Java *archive* (*jar*), depurador Java (*jdb*[14]), gerador *header*/*stub* JNI[15] (*javah*) e um decompilador Java (*javap*) e, a configuração das variáveis de ambiente, *patch*[16], do sistema operacional. Contudo, o pacote JDK não fornece um ambiente visual de desenvolvimento, é necessário o uso de um editor de texto, por exemplo: o bloco de notas do *Windows* para escrita do código-fonte das classes Java, conforme Junior (2007, p.24-29).

CONSIDERAÇÕES FINAIS

Em razão das características apresentadas da linguagem de desenvolvimento Java, é possível avaliar os benefícios em usá-la em diversos projetos. Por se tratar de uma linguagem que herda as características de orientação a objetos da linguagem C++, porém de uma forma nativa e com uma melhor abordagem no sentido de ser mais simples e neutra, tornou-se uma linguagem muito

utilizada no desenvolvimento de aplicações devido ao seu destaque na independência de plataforma, desempenho, segurança, múltiplas rotinas e compatibilidade com páginas *web*.

Esse levantamento bibliográfico sobre a linguagem Java, em virtude do que foi mencionado, pôde elucidar alguns tópicos relevantes, não somente sobre as características, mas sobre a estrutura de programação utilizada pela plataforma e os tipos de dados usados no Java. O programador deve atentar-se para evitar confusão quanto aos tipos, ou seja, a forma como é realizada a declaração das variáveis, verificando o não uso das palavras reservadas incorporadas na estrutura de escrita do Java, e também embutidas nos chamados *frameworks*, que neste caso visam à automatização de diversos processos apenas com a importação dos pacotes necessários.

Devido ao curto espaço de tempo na produção deste artigo a escolha do tema foi fator decisivo para a produção da pesquisa. E em virtude da abrangência do assunto, existe a necessidade de mais estudos. Em relação ao exposto, o estudo das características, paradigma de orientação a objetos, interpretação, funcionalidades *web*, alto desempenho, multiprocessamento, portabilidade, segurança, estrutura, processos e tipos de operadores da linguagem foram brevemente observados. Também não foi possível a realização de uma análise comparativa com outras linguagens que, também como o Java, possuem grande ascensão de mercado.

Portanto, devido aos fatores apresentados como limitações do projeto, não foram abordados alguns assuntos que são também de grande importância. Nesse sentido, ficarão para uma nova abordagem em trabalhos futuros, suporte a comunicação, coletor de lixo, operadores de atribuição composta, ternário e *bitwise*, *strings*, entrada e saída no Java, fluxos de controle, números grandes e *arrays*, devido as suas complexidades.

NOTAS

[1]Orientação a Objetos:modelagem de *software* em termos semelhantes aos usados para descrever objetos no mundo real (DEITEL; DEITEL, 2010, p.15).

[2]*Frameworks*: conjunto de classes que forma a base para construir funcionalidades avançadas - exemplo: *JSF*[17]*, Struts*[18]*, Stripes*[19]*, Wicret*[20]*, Hibernate*[21] - sem a necessidade de reescrever todo o código, bastando importá-lo para o projeto (HORSTMANN; CORNELL, 2010, p.314).

[3]*JSE (Java Standard Edition*): a base principal da linguagem, projetada para equipamentos que contem hardware básico (FURGERI, 2010, p.19).

[4]*JEE (Java Enterprise Edition*): usada para desenvolver aplicações baseadas em servidor (FURGERI, 2010, p.19).

[5]*JME (Java Micro Edition)*: projetada para máquinas com menor processamento e memória (ANTONIO; FERRO, 2009, p.04).

[6]*JavaFX:* plataforma de desenvolvimento de aplicações que podem ser executadas em diversos dispositivos, tornando a interface mais interativa e dinâmica (ANTONIO; FERRO, 2009, p.04).

[7]API (*Application Programming Interface*): é a *interface* de programação de aplicativo, documentação onde estão listadas as classes e os métodos da biblioteca Java (HORSTMANN, 2009, p.75).

[8]*Bytecodes*: é *uma* espécie de linguagem de máquina da JVM que utiliza instruções, tipos primitivos de tamanho fixo, ordenação e uma extensa biblioteca de classes padrões do Java (JUNIOR, 2007, p.23).

[9]Ponto Flutuante: são números decimais contendo partes fracionárias que podem ser do tipo *float* ou *double*, exemplo: 3.1415, 0.1, 2.997E8 (RITCHEY, 1997, p.159).

[10]Eclipse: Ferramenta utilizada para desenvolvimento de aplicações na linguagem Java (FURGERI, 2010, p.22)e (LIGUORI; LIGUORI, 2016, p.196).

[11]Netbeans: Ferramenta *Open Source* da *Oracle* para desenvolvimento de aplicações na linguagem Java (FURGERI, 2010, p.22; LIGUORI; LIGUORI, 2016, p.196).

[12]*Applets*: programas Java que são incorporados em páginas *web* tornando as páginas mais vivas e dinâmicas com o uso de animações (DEITEL; DEITEL, 2010, p.723).

[13]JAR (Java *Archive*): ferramenta utilitária para arquivamento e compressão, normalmente usada para combinar vários arquivos em um único arquivo (LIGUORI; LIGUORI, 2016, p.109).

[14]JDB (Java *Debugger*): ferramenta simples de depuração do código Java que fornece inspeção e depuração de uma Máquina Virtual Java local ou remota (JAVA, 2016).

[15]JNI (Java *Native Interface)*: Os métodos nativos são escritos em sua maioria com sintaxe da linguagem de programação C. Mecanismo utilizado para integrar o código de outra linguagem com um código Java (SCHILDT, 2011, p.313).

[16]*Patch* de configuração: é a configuração da variável de ambiente no sistema operacional utilizado para compilar as aplicações desenvolvidas em linguagem Java quando não há um ambiente de desenvolvimento integrado instalado, por exemplo *Eclipse*. Deste modo, o usuário utiliza o *prompt* de comando do *Windows* para testar os seus códigos escritos em um simples bloco de notas (FURGERI, 2010, p.24).

[17]JSF (Java *Server Face*): conjunto de tecnologias para desenvolvimento de *interface* do usuário, *Server side,* proporcionando maior facilidade no desenvolvimento de aplicações, fáceis e customizadas (ANTONIO; FERRO, 2009, p.04; LIGUORI; LIGUORI, 2016, p.197).

[18]*Struts*: *framework* usado para o desenvolvimento e gerenciamento de aplicações *web* usando uma arquitetura modelo para visualização o MVC (*Model-view-Controller*) (ANTONIO; FERRO, 2009, p.05; LIGUORI; LIGUORI, 2016, p.197).

[19]*Stripes*: o projeto também visa o desenvolvimento de aplicações *web*, porém com uma abordagem mais fácil e eliminando a série de configurações necessárias com o *framework Struts* (ANTONIO; FERRO, 2009, p.06).

[20]*Wicret*: tem como objetivo proporcionar o uso de POJO (*Plain Old Java Object*) para o desenvolvimento de aplicações *web,* tornando o desenvolvimento mais fácil e agradável (ANTONIO; FERRO, 2009, p.06).

[21]*Hibernate*: *framework* de desempenho para persistência de objetos, usa modelo relacional e serviços de consulta (*query*) (ANTONIO; FERRO, 2009, p.07; LIGUORI; LIGUORI, 2016, p.194).

REFERÊNCIAS

ANTONIO, Eric Aceiro; FERRO, Milene. **Análise Comparativa entre os Principais Frameworks de Desenvolvimento Java**. In: 4º Congresso Nacional de Ambientes Hipermídia para Aprendizagem, Florianópolis, 2009.

DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar.** 8ª Ed., São Paulo: Pearson Prentice Hall, 2010.

ECLIPSE. **Eclipse.** Disponível em: https://eclipse.org/downloads/. Acessado em: 18 de Maio de 2016.

FURGERI, Sergio. **Java 7: Ensino Didático.** São Paulo: Érica, 2010.

GOMES, Everton Barbosa. **Dante Explica: Java2 v1.4.** 2ª Ed., Rio de Janeiro: Ciência Moderna, 2004.

HELLER, Philip; ROBERTS, Simon. **Guia Completo de Estudos para Certificação em Java 2**. Rio de Janeiro: Ciência Moderna, 2004.

HORSTMANN, Cay. **Conceitos de Computação com Java.** 5ª Ed., Porto Alegre: Bookman, 2009.

HORSTMANN, Cay; CORNELL, Gary. **Core Java, Volume 1: Fundamentos.** 8ª Ed., São Paulo, Pearson Prentice Hall, 2010.

JAVA. ***Java SE API Documentation***. Disponível em: http://docs.oracle.com/javase/8/docs/api/. Acessado em: 22 de Maio de 2016.

JDK. **JDK: Kit desenvolvimento Java**. Disponível em: http://www.oracle.com/technetwork/pt/java/. Acessado em: 18 de Maio de 2016.

JUNIOR, Peter Jandl. **Java: Guia do Programador Atualizado para Java 6.** São Paulo: Novatec, 2007.

JUNIOR, Peter Jandl. **Java: Guia do Programador Atualizado para Java 7.** 2ª Ed., São Paulo: Novatec, 2013.

LIGUORI, Robert; LIGUORI, Patrícia. **Java 8 Pocket Guide.** Sebastopol: O´Reilly, 2014.

NETBEANS. **Netbeans.** Disponível em: https://netbeans.org/downloads/. Acessado em: 18 de Maio de 2016.

RITCHEY, Timothy. **Programando com Java! Beta 2.0**. Rio de Janeiro: Campus, 1997.

SCHILDT, Herbert. **Java: The Complete Reference Eighth Edition.** 8ª Ed., New York: The McGraw-Hill, 2011.

SEBESTA, Robert. **Conceitos de Linguagem de Programação.** 9ª Ed., Porto Alegre: Bookman, 2011.

SERSON, Roberto Rubisten. **Programação Orientada a Objetos com Java.** Rio de Janeiro: Brasport, 2007.