Linguagens de Programação: *Java*

Patrick Araújo
Acadêmico do curso de Ciência da Computação
Universidade Federal do Tocantins
Palmas - TO, Brasil
patrick.araujo@uft.edu.br

***Resumo*—Diante da necessidade de elaborar um trabalho acadêmico para a disciplina de Linguagens de Computação, o presente artigo apresenta uma breve introdução sobre a linguagem de programação *Java*, suas principais características, estrutura gramatical e uma lista de palavras chaves, operadores, conectivos e regras que se deve respeitar para a elaboração de um algoritmo bem documentado e com facilidade de leitura.**

***Index Terms*—Linguagens de Programação, *Java*, Gramática**

I. Introdução

A linguagem *Java* surgiu em 1995 e foi criada por James Gosling, que trabalhava na empresa *Sun Microsystem*s, hoje *Oracle*. Bastante usada no meio acadêmico para ensino de orientação a objetos, é umas das linguagens de programação mais populares atualmente. Umas das razões dessa popularidade se justifica pelo fato de ela ser multiplataforma, isso é possível porque o seu o código é compilado para *bytecode* que, em seguida, é interpretado pela *Java Virtual Machine*.

Ela possui uma sintaxe muito parecida com *C*, *C++* e *C#*, mas com diferentes bibliotecas. Por ser multiplataforma os programas em Java podem ser executados em diversos sistemas operacionais, com a condição que o interpretador esteja na máquina.

A Máquina Virtual *Java* converte o código *Java* em instruções que o sistema operacional reconheça. Por existir diversos computadores com variados sistemas operacionais, os programas *Java* podem ser processados em qualquer lugar. O lado negativo é que o programa se torna mais lento do que em comparação com outras linguagens.

Linguagens como *C* ou *C++* por outro lado, necessitam de um compilador para cada plataforma o que pode alterar também a escrita de programas. Para cada plataforma é necessário recompilar o código fonte, isso geral um arquivo binário diferente. Embora pode ser mais custoso esse processo, ele pode ser benéfico pois o desempenho aumenta.

A *JVM* permite que um único arquivo binário seja gerado para ser executado em diversas plataformas.

Desde a sua criação, *Java* foi planejada em duas partes. Como inglês e português, *Java* tem sua gramática e nomes comumente usados. A linguagem de programação *Java* tem suas especificações (sua gramática) e sua interface de programação de aplicações.

A linguagem de especificação *Java* é um documento que inclui regras como "Sempre abra parênteses depois da palavra *for*" e "use asteriscos para multiplicar dois números". A linguagem de especificação é relativamente pequena em comparação com outras linguagens.

II. Gramática

A. Gramática Léxica

A gramática de *Java* tem como seus símbolos terminais os caracteres da codificação de caracteres ***Unicode***. Ela define um conjunto de produções, começando do símbolo principal *Input*, que descreve como sequencias de caracteres *Unicode* são traduzidos em uma sequência de elementos *input*.

Esses elementos *inputs* com espaço em branco e comentários descartados, formam os símbolos terminais da gramática sintática da linguagem de programação *Java*, e são chamados de *tokens*. Esses *tokens* são identificadores, palavras chaves, literais, separadores e operadores da linguagem.

B. Gramática Sintática

A gramática sintática possui *tokens* definidos pela gramática léxica como seus símbolos terminais. Ela define um conjunto de produções, começando com o símbolo principal *CompilationUnit*, que descreve como a sequência de *tokens* podem formar programas sintaticamente corretos.

III. Palavras chaves

50 sequencias de caracteres, formados a partir de letras do *ASCII*, são reservadas para uso de palavras chaves e não podem ser usadas como identificadores.

abstract	*double*	*int*	*super*
assert	*else*	*interface*	*switch*
boolean	*enum*	*long*	*synchronized*
break	*extends*	*native*	*this*
byte	*final*	*new*	*throw*
case	*finally*	*package*	*throws*
catch	*float*	*private*	*transient*
char	*for*	*protected*	*try*
class	*goto*	*public*	*void*
const	*if*	*return*	*volatile*
continue	*implements*	*short*	*while*
default	*import*	*static*	
do	*instanceof*	*strictfp*	

IV. Operadores e Conectivos

37 *tokens* são operadores, formados dos caracteres da tabela *ASCII*

=	-=	/	++	<<=
==	>	/=	∧	>>
+	>=	∼	∧ =	
+=	*	||	–	>>=
<	*=	|	%	>>>
<=	!	|=	%=	
-	!=	:	<<	<<<

V. Regras

Os algoritmos devem ser escritos priorizando a facilidade de leitura, entendimento, e a modificação por outras pessoas. As regras de estilo são importantes e muitos algoritmos são avaliados pelo estilo e sua exatidão. Abaixo será elencado algumas regras básicas de estilo de programação.

1) Regra principal
   a) Um programa deve ser legível.
2) Comentários
   a) Toda classe (exceto para classes internas anônimas) deve conter uma *Javadoc* comentado que especifica o propósito da classe de descrever a sua interface pública em termos gerais. Exceto por classes aninhadas, o comentário *Javadoc* deve incluir um tag autor (@*author*).
   b) Todo método deve incluir um *Javadoc* que comenta a sua função, métodos privados não se aplicam. O comentário deve incluir uma declaração de todas as condições que devem ser mantidas quando o método é chamado, como restrições sobre os valores aceitáveis de seus parâmetros (Também chamado de "pré-condições"do método). O propósito de cada parâmetro deve ser claramente documentado. Se há algum valor retornado, seu significado deve ser documentado. Se um método pode invocar alguma exceção, é uma boa ideia também documentar isso. É encorajado usar as tags @*param*, @*return*, e @*throws* para documentar.
   c) Toda variável que tem papel não trivial no programa deve ser comentada para explicar seu propósito. Isso inclui tanto variáveis locais e globais. Para variáveis que não são privadas globais, o comentário deve ser no formato *Javadoc*.
   d) Comentários podem ser incluídos no corpo de um método quando é necessário explicar a lógica de um código. Códigos bem escritos em geral possuem poucos comentários.
   e) Comentários nunca devem ser usados para explicar a linguagem *Java*. Um comentário como "declarar uma variável float com nome y" ou "incrementar uma variável com nome x" é inútil.
3) Formatação
   a) Use indentação para mostrar a estrutura do programa. O corpo de uma definição de classe ou de método deve ser indentado. Quando uma instrução é aninhada dentro de outra instrução, ela deve ser recuada em um nível adicional.
   b) A abertura de um colchete "{" pode estar no final de uma linha. O fechamento de um colchete "}" deve estar em uma linha com nenhum outro caractere.
   c) Não coloque mais que uma declaração em uma linha.
   d) Evite longas linhas. Em geral, linhas não devem possuir mais que 80 caracteres. Declarações longas devem possuir mais de uma linha.
   e) Evite aninhamento muito profundo de declarações.
   f) Espaços e linhas em brancos, podem facilitar a leitura de um programa. É ideal deixar espaços entre operadores, como =, ==, != e outros. Linhas em branco podem ajudar a leitura no caso de definições de método.
4) Nomeação
   a) Use nomes significantes para variáveis, métodos e classes.
   b) Para variáveis, métodos e pacotes devem ser usados palavras com caixa baixa. Nomes de classes devem conter palavras com a primeira letra maiúscula. Se um nome contém mais que duas palavras, é ideal deixar as palavras extras com letra maíscula.
   c) Use "*final static*" para declarar constantes nomeadas para representar dados constantes. Constantes geralmente tem nomes que estão de forma integral em caixa alta e com as palavras separadas com o traço inferior.
5) Métodos
   a) Métodos devem conter uma clara e única tarefa identificável.
   b) Uma definição de método individual não deve ser muito longa. Em geral uma função não deve ser maior que uma página impressa.
   c) Os métodos de instância podem acessar as variáveis de instâncias que representam o estado de um objeto, mas deve-se evitar usar variáveis de instâncias para passar informações de um método a outro, para tanto, deve-se usar parâmetros de variáveis de retorno.
6) Classe
   a) Uma classe deve representar um claro, único e identificável conceito
   b) Deve-se usar os modificadores *public*, *protected* e *private* para controlar o acesso as variáveis
   c) Variáveis membros devem em geral, serem declaradas em privado. Métodos *getters* e *setters* podem ser providos para acessar e manipular as variáveis membros privadas.

Conclusão

As particularidades de *Java* não ficam só no fato da linguagem ser portátil e de possuir uma máquina virtual. A gramática de Java a torna uma linguagem com características únicas em comparação com outras linguagens. Além de possuir palavras chaves que em outras linguagens como *C* estão em falta. Em comparação os operadores e conectivos notamos a falta de operadores para ponteiros pelo fato de Java não trabalhar com ponteiros. Por fim temos as regras que os programadores

devem seguir para a criação de algoritmos bem documentados, livres de redundância e legíveis.

REFERÊNCIAS

[1] D. Eck, ”Java Programming Style Rules”, Math.hws.edu, 2018. [Online]. Available: http://math.hws.edu/eck/cs124/f11/style_guide.html. [Accessed: 27- Aug- 2018].

[2] ”Linguagens de programação - Hardware.com.br”, Hardware.com.br, 2007. [Online]. Available: https://www.hardware.com.br/artigos/linguagens/. [Accessed: 27- Aug- 2018].

[3] Gosling, B. Joy, G. Steele, G. Bracha and A. Buckley, ”The Java™ Language Specification: Java SE 7 Edition”, Oracle, 2011. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se7/jls7.pdf. [Accessed: 27- Aug- 2018].