Área de Formação: Ciências Informáticas
Itinerário de Formação: 481039 – Técnico de Informática e Sistemas
Modalidade: Educação e Formação de Adultos

UFCD: 21 - Fundamentos de linguagem Java
Formador: Rui Almeida

Validado – 29/05/14

Ana Paula Sarandeses

Nome: Luís Caldeira Nº 19 Data: 13-05-14

PRA – Reflexão Final do Módulo

UC21

Nesta UFCD (unidade de formação de curta duração), falámos sobre os fundamentos da linguagem Java. Este módulo foi a iniciação à linguagem *Java*.

Todo o contudo deste módulo é novo para mim, apenas tinha conhecimento do seu uso enquanto plataforma.

Começámos por instalar o *software "NetBeans IDE",* que nos foi facultado pelo formador Rui Almeida.

A tecnologia Java é usada para desenvolver aplicativos para uma ampla variedade de ambientes, de dispositivos para o consumidor a sistemas corporativos heterogéneos.

Aprendi o conceito da linguagem *Java*. A linguagem Java, como acontece em qualquer linguagem de programação, tem a sua própria estrutura, regras de sintaxe e paradigma de programação. O paradigma de programação da linguagem Java baseia-se no conceito de programação orientada a objectos (OOP), que os recursos da linguagem suportam.

A linguagem Java é uma derivação da linguagem C; portanto, as suas regras de sintaxe são muito semelhantes às regras dessa linguagem — por exemplo: os blocos de códigos são modulados em métodos e delimitados por chaves ({ e }) e as variáveis são declaradas antes de serem usadas.

Estruturalmente, a linguagem Java começa com pacotes. O pacote é o mecanismo de *namespace* da linguagem Java. Dentro do pacote há classes e dentro das classes há métodos, variáveis, constantes, entre outros.

O compilador Java escreve o código de origem em arquivos Java e em seguida compila-os. O compilador verifica o código em relação às regras de sintaxe da linguagem e, em seguida, grava *bytecodes* em arquivos *.class.* Os *bytecodes* são instruções padrão que devem ser executadas numa *Java virtual machine* (JVM). Ao incluir esse nível de abstracção, o compilador Java é diferente dos compiladores de outras linguagens, que gravam instruções adequadas para o conjunto de chips da CPU no qual o programa será executado.

No tempo de execução, a JVM lê e interpreta os *arquivos. class* e executa as instruções do programa na plataforma de hardware nativa para a qual a JVM foi escrita. A JVM interpreta os bytecodes da mesma forma que a CPU interpretaria as instruções da linguagem do conjunto. A diferença é que a JVM é um software escrito especificamente para uma plataforma. A JVM é o ponto central do princípio da linguagem Java: "escreva uma vez, execute em qualquer lugar". O código pode executar em qualquer conjunto de chips para o qual uma implementação adequada de JVM estiver disponível. Há JVMs disponíveis para as principais plataformas, como Linux e Windows, e subconjuntos da linguagem Java foram implementados em JVMs de chips para telemóveis e praticantes de hobby.

Em vez de me obrigar a acompanhar a alocação de memória (ou usar uma biblioteca de terceiros para fazer isso), a plataforma Java fornece uma gestão de memória de fábrica. Quando o seu aplicativo Java cria uma instância de objecto no tempo de execução, a JVM aloca automaticamente o espaço na memória para esse objecto no heap, que é um conjunto de memória separado para que o seu programa o use. O colector de lixo executa em segundo plano, verificando quais são os objectos que não são mais necessários para o aplicativo e tomando de volta a memória a eles reservada. Esta abordagem à manipulação da memória é chamada gestão implícita da memória porque não requer que se escreva o código para manipular a memória. A colecta de lixo é um dos recursos essenciais para o desempenho da plataforma Java.

Como qualquer linguagem de programação, a linguagem Java designa certas palavras que são especiais para o compilador e, por serem especiais, não é permitido usá-las para dar nomes às construções em Java. Gostei da iniciação à linguagem Java.

A lista de palavras reservadas é surpreendentemente curta. Alguns exemplos:

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