A linguagem Java

Entender um pouco da história da plataforma Java é muito importante para enxergar os motivos que a levaram ao sucesso.

Quais eram os maiores problemas para programadores na década de 1990?

- ponteiros?
- gerenciamento de memória?
- organização?
- falta de bibliotecas?
- ter de reescrever parte do código ao mudar de sistema operacional?
- custo financeiro de usar a tecnologia?

A linguagem Java resolve bem esses problemas, que até então apareciam com frequência nas outras linguagens. Alguns desses problemas foram particularmente atacados porque uma das grandes motivações para a criação da plataforma Java era de que essa linguagem fosse usada em pequenos dispositivos, como tvs, videocassetes, aspiradores, liquidificadores e outros. Apesar disso a linguagem teve seu lançamento focado no uso em clientes web (browsers) para rodar pequenas aplicações (**applets**). Hoje em dia esse não é o grande mercado do Java: apesar de ter sido idealizado com um propósito e lançado com outro, o Java ganhou destaque no lado do servidor.

O Java foi criado pela Sun (http://www.sun.com) e mantida através de um comitê (http://www.jcp.org). Seu site principal era o java.sun.com, e java.com um site mais institucional, voltado ao consumidor de produtos e usuários leigos, não desenvolvedores. Com a compra da Sun pela Oracle em 2009, muitas URLs e nomes tem sido trocados para refletir a marca da Oracle. A página principal do Java é: http://www.oracle.com/technetwork/java/.

No Brasil, diversos grupos de usuários se formaram para tentar disseminar o conhecimento da linguagem. Um deles é o **GUJ** (http://www.guj.com.br), uma comunidade virtual com artigos, tutoriais e fórum para tirar dúvidas, o maior em língua portuguesa com mais de cem mil usuários e 1 milhão de mensagens.

Encorajamos todos os alunos a usar muito os fóruns do mesmo, pois é uma das melhores maneiras para achar soluções para pequenos problemas que acontecem com grande frequência.

Máquina Virtual

Em uma linguagem de programação como C e Pascal, temos a seguinte situação

quando vamos compilar um programa: o código fonte é compilado para código de máquina específico de uma plataforma e sistema operacional. Muitas vezes o próprio código fonte é desenvolvido visando uma única plataforma!

Esse código executável (binário) resultante será executado pelo sistema operacional e, por esse motivo, ele deve saber conversar com o sistema operacional em questão.

Isto é, temos um código executável para cada sistema operacional. É necessário compilar uma vez para Windows, outra para o Linux, e assim por diante, caso a gente queira que esse nosso software possa ser utilizado em várias plataformas. Esse é o caso de aplicativos como o OpenOffice, Firefox e outros.

Na maioria das vezes, a sua aplicação se utiliza das bibliotecas do sistema operacional, como, por exemplo, a de interface gráfica para desenhar as "telas". A biblioteca de interface gráfica do Windows é bem diferente das do Linux: como criar então uma aplicação que rode de forma parecida nos dois sistemas operacionais?

Precisamos reescrever um mesmo pedaço da aplicação para diferentes sistemas operacionais, já que eles não são compatíveis.

Já o Java utiliza do conceito de **máquina virtual**, onde existe, entre o sistema operacional e a aplicação, uma camada extra responsável por "traduzir" - mas não apenas isso - o que sua aplicação deseja fazer para as respectivas chamadas do sistema operacional onde ela está rodando no momento.

Dessa forma, a maneira com a qual você abre uma janela no Linux ou no Windows é a mesma: você ganha independência de sistema operacional. Ou, melhor ainda, independência de plataforma em geral: não é preciso se preocupar em qual sistema operacional sua aplicação está rodando, nem em que tipo de máquina, configurações, etc.

Repare que uma máquina virtual é um conceito bem mais amplo que o de um interpretador. Como o próprio nome diz, uma máquina virtual é como um "computador de mentira": tem tudo que um computador tem. Em outras palavras, ela é responsável por gerenciar memória, threads, a pilha de execução, etc.

Sua aplicação roda sem nenhum envolvimento com o sistema operacional! Sempre conversando apenas com a Java Virtual Machine (JVM).

Essa característica é interessante: como tudo passa pela JVM, ela pode tirar métricas, decidir onde é melhor alocar a memória, entre outros. Uma JVM isola totalmente a aplicação do sistema operacional. Se uma JVM termina abruptamente,

só as aplicações que estavam rodando nela irão terminar: isso não afetará outras JVMs que estejam rodando no mesmo computador, nem afetará o sistema operacional.

Essa camada de isolamento também é interessante quando pensamos em um servidor que não pode se sujeitar a rodar código que possa interferir na boa execução de outras aplicações.

Essa camada, a máquina virtual, não entende código java, ela entende um código de máquina específico. Esse código de máquina é gerado por um compilador java, como o javac, e é conhecido por "bytecode", pois existem menos de 256 códigos de operação dessa linguagem, e cada "opcode" gasta um byte. O compilador Java gera esse bytecode que, diferente das linguagens sem máquina virtual, vai servir para diferentes sistemas operacionais, já que ele vai ser "traduzido" pela JVM.

Compilando o primeiro programa

Vamos para o nosso primeiro código! O programa que imprime uma linha simples.

Para mostrar uma linha, podemos fazer:

```
System.out.println("Minha primeira aplicação Java!");
```

Mas esse código não será aceito pelo compilador java. O Java é uma linguagem bastante burocrática, e precisa de mais do que isso para iniciar uma execução. Veremos os detalhes e os porquês durante os próximos capítulos. O mínimo que precisaríamos escrever é algo como:

```
class MeuPrograma {
    public static void main(String[] args) {
        System.out.println("Minha primeira aplicação Java!");
    }
}
```

Após digitar o código acima, grave-o como **MeuPrograma.java** em algum diretório. Para compilar, você deve pedir para que o compilador de Java da Oracle, chamado javac, gere o bytecode correspondente ao seu código Java.

```
javac MeuPrograma.java
```

Depois de compilar, o **bytecode** foi gerado. Quando o sistema operacional listar os arquivos contidos no diretório atual, você poderá ver que um arquivo **.class** foi gerado, com o mesmo nome da sua classe Java.

Executando seu primeiro programa

Os procedimentos para executar seu programa são muito simples. O javac é o compilador Java, e o java é o responsável por invocar a máquina virtual para interpretar o seu programa.

```
java MeuPrograma
```

Ao executar, pode ser que a acentuação resultante saia errada devido a algumas configurações que deixamos de fazer. Sem problemas.

O que aconteceu?

```
class MeuPrograma {
    public static void main(String[] args) {

        // miolo do programa começa aqui!
        System.out.println("Minha primeira aplicação Java!!");
        // fim do miolo do programa

    }
}
```

O miolo do programa é o que será executado quando chamamos a máquina virtual. Por enquanto, todas as linhas anteriores, onde há a declaração de uma classe e a de um método, não importam para nós nesse momento. Mas devemos saber que toda aplicação Java começa por um ponto de entrada, e este ponto de entrada é o método main.

Ainda não sabemos o que é método, mas veremos no capítulo 4. Até lá, não se preocupe com essas declarações. Sempre que um exercício for feito, o código que nos importa sempre estará nesse miolo.

No caso do nosso código, a linha do System.out.println faz com que o conteúdo entre aspas seja colocado na tela.

Para saber mais: como é o bytecode?

O `MeuPrograma.class` gerado não é legível por seres humanos (não que seja impossível). Ele está escrito no formato que a virtual machine sabe entender e que foi especificado que ela entendesse.

É como um assembly, escrito para esta máquina em específico. Podemos ler os mnemônicos utilizando a ferramenta javap que acompanha o JDK:

```
javap -c MeuPrograma.class
```

E a saída:

```
MeuPrograma();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return

public static void main(java.lang.String[]);
    Code:
    0:    getstatic    #2; //Field java/lang/System.out:Ljava/io/PrintStream;
    3:   ldc     #3; //String Minha primeira aplicação Java!!
    5:   invokevirtual   #4; //Method java/io/PrintStream.println:
                                (Ljava/lang/String;)V
    8:   return

}
```

É o código acima, que a JVM sabe ler. É o "código de máquina", da máquina virtual.

Um bytecode pode ser revertido para o .java original (com perda de comentários e nomes de variáveis locais). Caso seu software vá virar um produto de prateleira, é fundamental usar um ofuscador no seu código, que vai embaralhar classes, métodos e um monte de outros recursos (indicamos o http://proguard.sf.net).