

CAPÍTULO 1

A plataforma Java

Diversas plataformas de desenvolvimento possuem grande penetração no mercado. A plataforma Java atingiu a liderança devido a algumas características relacionadas ao seu processo de evolução e especificação, junto com a participação forte e ativa da comunidade. Conhecer bem o ecossistema Java revela seus pontos fortes e também as desvantagens e limitações que podemos enfrentar ao adotá-la.

1.1. JAVA COMO PLATAFORMA, ALÉM DA LINGUAGEM

É fundamental conhecer com que objetivos a plataforma Java foi projetada, a fim de entender com profundidade os motivos que a levaram a ser fortemente adotada fortemente adotada no lado do servidor.

Java é uma plataforma de desenvolvimento criada pela Sun, que teve seu lançamento público em 1995. Ela vinha sendo desenvolvida desde 1991 com o nome *Oak*, liderada por James Gosling. O mercado inicial do projeto Oak compunha-se de dispositivos eletrônicos, como os *set-top box*.¹

Desde a sua concepção, a ideia de uma plataforma de desenvolvimento e execução sempre esteve presente. O Java idealizado era uma solução que facilitava o desenvolvimento de aplicativos portáteis, utilizando, para isso, uma linguagem de programação simplificada, segura, orientada a objetos, com uma extensa API e um ambiente de execução portátil.

Para obter a portabilidade, adotou-se a ideia de uma **máquina virtual** que executa um código de máquina próprio. Essa máquina virtual simplesmente traduziria as instruções para uma plataforma específica, ganhando independência de plataforma.

Logo após seu lançamento, o Java fez um enorme sucesso com seus **Applets**, pequenas aplicações embutidas em páginas Web executadas no navegador do cliente. Nessa época, propagou-se o conhecido slogan “*Write once, run anywhere*”. Com a ajuda da máquina virtual, qualquer navegador rodaria a mesma aplicação sem necessidade de alterações. Era um passo importante para adicionar mais recursos ao navegador, ainda tão limitado.

Atualmente, o Java está presente em vários ramos da tecnologia, e é uma das plataformas mais importantes do mundo. O mercado de Applets acabou não se tornando um grande destaque, e o Java firmou sua posição no mercado corporativo (Java EE) com soluções robustas para Web e aplicações distribuídas.

O Java é também uma das principais plataformas de desenvolvimento para dispositivos móveis (Java ME), com milhões de celulares, palms, *set-top boxes* e até nos aparelhos de *blu-ray*. Há também o caso do Google Android que usa a linguagem Java, mas é um caso à parte, já que sua máquina virtual não utiliza o bytecode Java.

No Desktop, houve avanços: em 2005, nos Estados Unidos, o Swing foi o toolkit gráfico mais usado em novas aplicações.² O Java FX, que nasceu muito tempo depois dos seus concorrentes Adobe Flex e Microsoft Silverlight, está sendo reformulado para aproveitar melhor a linguagem Java e marcar presença nas *Rich Internet Applications* (RIA). Mesmo assim, este ainda é o menor mercado do Java; sua portabilidade não tem vencido a facilidade de desenvolvimento de aplicativos em linguagens focadas em uma única plataforma.

É importante notar que o Java nasceu com o intuito de ser uma plataforma muito utilizada em dispositivos móveis, e ganhou força através dos Applets, mas o grande mercado que tornou a plataforma bem disseminada foi o corporativo, no *server-side*. Dada sua portabilidade e segurança, a plataforma foi a escolha feita por grandes bancos e empresas que precisavam ter maior facilidade de mudança. Esta facilidade evita o *vendor lock-in*, permitindo que não se fique dependente de

um único fabricante, um único sistema operacional ou um único banco de dados. Com a plataforma Java, essas empresas adquiriram um grau maior dessa liberdade.

Mas o que é exatamente o Java? **Mais que uma linguagem de programação, Java é uma completa plataforma de desenvolvimento e execução.** Esta plataforma é composta de três pilares: a máquina virtual Java (JVM), um grande conjunto de APIs e a linguagem Java.

O conceito de máquina virtual não é exclusividade do Java. As *Hardware Virtual Machines*, como VMWare, Parallels ou VirtualBox, também muito famosas, são usadas para rodar vários sistemas operacionais ao mesmo tempo, utilizando o recurso de virtualização que abstrai o hardware da máquina. Já a JVM, **Java Virtual Machine** (Máquina Virtual Java), é uma *Application Virtual Machine*, que abstrai não só a camada de hardware como a comunicação com o Sistema Operacional.

Uma aplicação tradicional em C, por exemplo, é escrita em uma linguagem que abstrai as operações de hardware e é compilada para linguagem de máquina. Nesse processo de compilação, é gerado um executável com instruções de máquina específicas para o sistema operacional e o hardware em questão.

Um executável C não é portátil; não podemos executá-lo no Windows e no Linux sem recompilação. Mas o problema de portabilidade não se restringe ao executável, já que muitas das APIs são também específicas do sistema operacional. Assim, não basta recompilar para outra plataforma, é preciso reescrever boa parte do código (Figura 1.1).

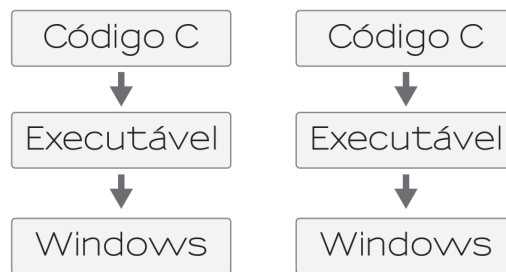


Figura 1.1 – Programa em C sendo executado em dois sistemas operacionais diferentes.

Ao usar o conceito de máquina virtual, o Java elimina o problema da portabilidade do código executável. Em vez de gerar um executável específico, como

“Linux PPC” ou “Windows i386”, o compilador Java gera um executável para uma máquina genérica, virtual, não física, a JVM.

JVM é uma máquina completa que roda em cima da real. Ela possui suas próprias instruções de máquina (*assembly*) e suas APIs. Seu papel é executar as instruções de máquina genéricas no Sistema Operacional e no hardware específico sob o qual estiver rodando (Figura 1.2).

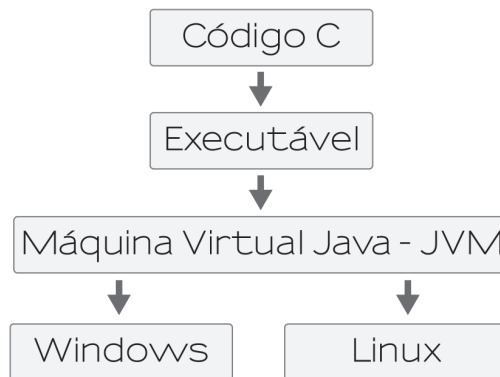


Figura 1.2 – Programa em Java sendo executado em dois sistemas operacionais diferentes.

No fim das contas, o problema da portabilidade continua existindo, mas o Java puxa essa responsabilidade para a própria JVM, em vez de deixá-la a cargo do desenvolvedor. É preciso instalar uma JVM específica para o sistema operacional e o hardware que se vai usar. Mas, feito isso, o desenvolvimento do aplicativo é totalmente portátil.

É importante perceber que, por causa deste conceito da JVM, o usuário final, que deseja apenas rodar o programa (não desenvolvê-lo), não pode simplesmente executá-lo. Tanto os desenvolvedores quanto os usuários precisam da JVM. Mas os primeiros, além da JVM, precisam do compilador e de outras ferramentas.

Por esta razão, o Java possui duas distribuições diferentes: o **JDK (Java Development Kit)**, que é focado no desenvolvedor e traz, além da VM, compilador e outras ferramentas úteis; e o **JRE (Java Runtime Environment)**, que traz apenas o necessário para se executar um aplicativo Java (a VM e as APIs), voltado ao usuário final.

Um ponto fundamental a respeito da JVM é que ela é uma especificação, diferente de muitos outros produtos e linguagens. Quando você comprava o Microsoft Visual Basic 6.0, só havia uma opção: o da Microsoft.

A JVM, assim como a linguagem, possui especificações muito bem definidas e abertas: a *Java Virtual Machine Specification*³ e a *Java Language Specification*.⁴ Em outras palavras, pode-se tomar a liberdade de escrever um compilador e uma máquina virtual para o Java. E é isto o que acontece: não existe apenas a JVM HotSpot da Sun/Oracle, mas também a J9 da IBM, a JRockit da Oracle/BEA, a Apple JVM, entre outras. Ficamos independentes até de fabricante, o que é muito importante para grandes empresas que já sofreram muito com a falta de alternativas para tecnologias adotadas antigamente.

Essa variedade de JVMs também dá competitividade ao mercado. Cada uma das empresas fabricantes tenta melhorar sua implementação, seja aperfeiçoando o JIT *compiler*, fazendo *tweaks* ou mudando o gerenciamento de memória. Isto também traz tranquilidade para as empresas que usam a tecnologia. Se algum dia o preço da sua fornecedora estiver muito alto, ou ela não atingir mais seus requisitos mínimos de performance, é possível trocar de implementação. Temos a garantia de que isso funcionará, pois uma JVM, para ganhar este nome, tem de passar por uma bateria de testes da Sun, garantindo compatibilidade com as especificações.

A plataforma Microsoft .NET também é uma especificação e, por este motivo, o grupo Mono pode implementar uma versão para o Linux.

Os bytecodes Java

A JVM é, então, o ponto-chave para a portabilidade e a performance da plataforma Java. Como vimos, ela executa instruções genéricas compiladas a partir do nosso código, traduzindo-as para as instruções específicas do sistema operacional e do hardware utilizados.

Essas instruções de máquina virtual são os **bytecodes**. São equivalentes aos *mne-mônicos* (comandos) do assembly, com a diferença de que seu alvo é uma máquina virtual. O nome *bytecode* vem do fato de que cada *opcode* (instrução) tem o tamanho de um byte e, portanto, a JVM tem capacidade de rodar até 256 bytecodes diferentes (embora o Java 7 possua apenas 205). Isto faz dela uma máquina teoricamente simples de se implementar e de rodar até em dispositivos com pouca capacidade.

Podemos, inclusive, visualizar esses bytecodes. O download do JDK traz uma ferramenta, o **javap**, capaz de mostrar de maneira mais legível o bytecode contido em um arquivo *.class* compilado. O bytecode será mostrado através dos nomes das instruções, e não de bytes puros.

O código a seguir ilustra um exemplo de classe Java:

```
public class Ano {
    private final int valor;
    public Ano(int valor) {
        this.valor = valor;
    }
    public int getValor() {
        return valor;
    }
}
```

E a execução da linha do comando `javap -c Ano` exibe o conjunto de instruções que formam a classe `Ano`:

```
public class Ano extends java.lang.Object{
public Ano(int);
    Code:
    0:  aload_0
    1:  invokespecial #10; //Method java/lang/Object."<init>":()V
    4:  aload_0
    5:  iload_1
    6:  putfield #13; //Field valor:I
    9:  return

public int getValor();
    Code:
    0:  aload_0
    1:  getfield #13; //Field valor:I
    4:  ireturn
}
```

Embora a linguagem Java tenha um papel essencial na plataforma, tê-la como uma executora de bytecodes, e não exatamente de código Java, abre possibilidades interessantes. A plataforma Java tem ido cada vez mais na direção de ser um ambiente de execução multilinguagem, tanto através da *Scripting API* quanto por linguagens que compilam direto para o bytecode. É uma ideia presente, por exemplo, no .NET desde o início, com sua *Common Language Runtime* (CLR) executando diversas linguagens diferentes.

A linguagem Scala mostra esta força do bytecode para a plataforma Java. Para a JVM, o que interessa são os bytecodes, e não a partir de qual linguagem eles foram gerados. Em Scala, teríamos o seguinte código fonte, que possui a mesma funcionalidade que o código Java mostrado anteriormente:

```
class Ano(val valor:Int)
```

Ao compilarmos a classe Scala anterior com o compilador `scalac`, os bytecodes gerados serão muito próximos dos vistos na classe Java. É possível observar isso rodando o comando `javap` novamente. A única grande mudança é que o getter em Scala chama-se apenas `valor()` e não `getValor()` como em Java. Veremos mais sobre outras linguagens rodando em cima da JVM neste capítulo. Além disso, muitos frameworks e servidores de aplicação geram bytecode dinamicamente durante a execução da aplicação.

A JVM é, portanto, um poderoso executor de bytecodes, e não interessa de onde eles vêm, independentemente de onde estiver rodando. É por este motivo que muitos afirmam que a linguagem Java não é o componente mais importante da plataforma, mas, sim, a JVM e o bytecode.

1.2. ESPECIFICAÇÕES AJUDAM OU ATRAPALHAM?

A Plataforma Java é bastante lembrada por suas características de abertura, portabilidade e até liberdade. Há o aspecto técnico, com independência de plataforma e portabilidade de sistema operacional desde o início da plataforma. Há também a comunidade com seus grupos de usuários por todo o mundo, sempre apoiados e incentivados pela Sun/Oracle. Há ainda o mundo open source, que abraçou o Java com diversos projetos cruciais para o ecossistema da plataforma, muitos deles presentes no dia a dia da maioria dos desenvolvedores. E há o JCP e suas especificações.

Em 1998, três anos após o nascimento do Java dentro da Sun, foi criado o *Java Community Process* (JCP), como um órgão independente da própria Sun, cujo objetivo era guiar os passos do Java de maneira mais aberta e independente. Hoje, a Oracle detém os direitos da marca Java, mas as decisões sobre os rumos da plataforma não estão apenas em suas mãos. Quem guia os rumos do Java é o JCP, com suas centenas de empresas e desenvolvedores participantes, que ajudam a especificar as novas tecnologias e a decidir novos caminhos. Há um *Comitê Executivo* eleito a cada três anos que comanda o órgão. Grandes empresas, como Google, IBM, HP, RedHat, Nokia, além da Oracle, fazem parte dele, inclusive organizações livres, como a SouJava.

Cada nova funcionalidade ou mudança que se queira realizar na plataforma é proposta ao JCP no formato **JSR**, uma *Java Specification Request*. São documentos que relatam essas propostas, para que possam ser votadas pelos membros do JCP. Depois de aceita uma JSR, é formado um **Expert Group** para especificá-la em