## Resumo

Jvm é uma máquina virtual java pega qualquer códico escrito em java e o compila em seu próprio bytecode, para poder ser uma linguagem compreensível por qualquer sistema que utilize java.

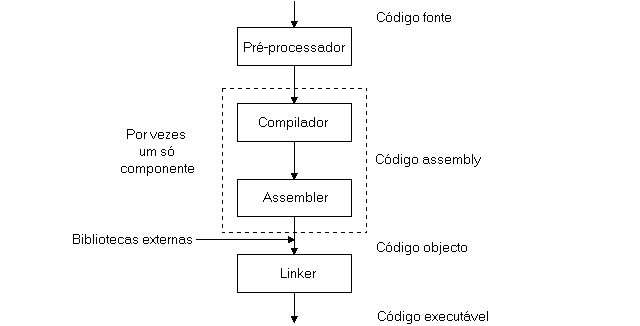
<https://www.devmedia.com.br/introducao-ao-java-virtual-machine-jvm/27624>

O [Java é uma linguagem de computador](https://www.devmedia.com.br/curso/curso-java/2105) muito utilizada atualmente em diversos Sistemas Operacionais (quando falamos em sistemas operacionais, não está relacionando apenas com computadores pessoais, mas também com tablets, equipamentos de Blu-Ray e etc.) devido a sua facilidade de ter o código portado. Ouvimos sempre por aí que o Java “roda em qualquer lugar” (write once, run everywhere), mas porque ele consegue “rodar” em qualquer plataforma? A resposta para esta pergunta é muito simples: devido a sua Máquina Virtual.

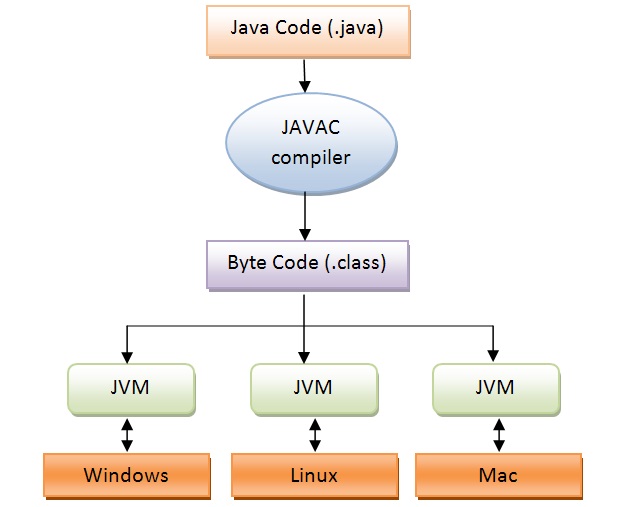
Em 1991, foi criada uma linguagem a partir do Green Project. James Gosling, Mike Sheridan e Patrick Naughton eram os idealizadores. O objetivo desse projeto não era criar uma nova linguagem de programação, mas sim, proporcionar a integração de computadores com outros tipos de equipamentos, como hardwares ou circuitos elétricos pré-fabricados. Em 1995, a empresa Sun MicroSystems introduziu a plataforma JAVA (batizada posteriormente) no mercado. Ela é constituída pela linguagem de programação Java, sua máquina virtual e muitas APIs de controle e desenvolvimento.

Antes de falarmos sobre a **JVM**, vamos pensar no conceito de máquina virtual. Uma máquina virtual é um software que simula uma máquina física e consegue executar vários programas, gerenciar processos, memória e arquivos. Resumindo, ele constitui de uma plataforma, onde a memória, o processador e seus outros recursos, são totalmente virtuais, não dependendo de hardwares.

Em uma [linguagem como a C](https://www.devmedia.com.br/curso/curso-de-introducao-a-c/274), o código é compilado para uma máquina especifica, ou seja, quando seu código seja compilado, ele poderá ser executado apenas naquele sistema operacional. Para executarmos esse código em outro Sistema Operacional, temos que ajustar as bibliotecas de acordo com as necessidades e recompilar. Segue abaixo um diagrama simples de códigos compilados, onde é gerado um código especifico para a máquina em questão pelo compilador e o assembler, já o linker irá “linkar” as bibliotecas necessárias para o código se tornar executável. Após ele se tornar executável, ele será executado na plataforma para o qual foi compilado, como mostra a **Figura 1**.

**Figura 1**. Compilação de um Código

Já o Java não tem esse problema, pois sua execução não está diretamente relacionada com o Sistema Operacional, ele conversa diretamente com a **JVM (Java Virtual Machine)**, possibilitando assim a portabilidade de seu código. O que for escrito em um sistema operacional Windows, irá rodar em um sistema operacional Linux (salvo algumas exceções de códigos nativos). Esse processo cria uma independência do Sistema Operacional, dando ao desenvolvedor uma liberdade de desenvolver para múltiplas plataformas sem aquela preocupação de se o código irá funcionar corretamente. A Virtual Machine sim é desenvolvida em código nativo, pois ela conversa diretamente com o sistema operacional para que o programa Java funcione na máquina, como mostra a **Figura 2**.

**Figura 2**. Esquema de Funcionamento da JVM

Você pode estar pensando agora: “Ah, entendi. **A Java Virtual Machine é um interpretador de código**”. Não, ela é muito mais do que isso, além de interpretar código, é também responsável pela execução das pilhas, gerenciamento de memória, threads e etc., ou seja, é um “computador virtual”. Uma de suas funções que podemos notar aqui é o Garbage Collector. Ele é uma thread responsável pela “limpeza” da memória virtual, ou seja, quando existe muito “lixo” na memória virtual, o Garbage Collector entra em ação. Porém, é difícil prever quando isso irá acontecer, por ele ser uma thread, como comentado anteriormente, e as threads são lançadas de acordo com o escalonador de processos.

A JVM não entende código Java, e sim um código especifico chamado [ByteCode](https://www.devmedia.com.br/artigo-java-magazine-66-bytecode-escondendo-e-revelando/12302), que é gerado pelo compilador Java (javac). Esse código é o que será traduzido pela Virtual Machine para o código de cada máquina em questão. Os processos de execução de um software Java foram aperfeiçoados ao longo dos tempos, pois no início, a Virtual Machine interpretava apenas um ByteCode por vez. Hoje em dia, a JVM possui sistemas de compilação JIT (Just - In - Time) misturados com a interpretação do código. Essa técnica cria os chamados “Hot-Spots”, que nada mais são que áreas de código executadas com maior frequência. Isso ocorre com a análise dos ByteCodes à medida que eles são interpretados pela Virtual Machine.

Uma pergunta muito comum entre os desenvolvedores iniciantes de Java é: “Quero começar a programar em Java, o que devo baixar? JDK ou JRE?”.

No primeiro momento, temos o [JDK](https://www.devmedia.com.br/introducao-ao-java-jdk/28896), que é o kit de desenvolvimento (Java Development Kit), que é o utilizado pelo desenvolvedor, pois ele possui pacotes que possibilitam o desenvolvimento de aplicações Java em nosso ambiente, já possuindo a JVM.

Já o [JRE](https://www.devmedia.com.br/java-jre-introducao/28917), é o ambiente de execução (Java Runtime Enviroment). Todas as máquinas que rodam uma aplicação Java precisam desse runtime, pois é onde a JVM estará contida, e como já comentado anteriormente, irá fazer todo o controle das aplicações Java.

Com isso podemos perceber, mesmo que superficialmente, que a **Java Virtual Machine faz um trabalho gigantesco “por detrás dos panos”**, não permitindo ao desenvolvedor se preocupar com coisas que “atrapalhariam” o desenvolvimento de uma aplicação. Mas ao mesmo tempo, é importante que se saiba o que acontece lá no “baixo” nível, se o desenvolvedor quer ser realmente completo. Com o decorrer do tempo, o Java irá evoluir como tudo nesse mundo e muitas mudanças podem ocorrer, o que nunca irá mudar (pelo menos espera-se) é o seguinte: write once, run everywhere.

Espero que gostem deste artigo introdutório.