Java

Índice

Introdução

- ## Um pouco sobre mim

Atualmente atuo como analista de performance. Na minha jornada fiz um curso técnico de informática no SENAI, onde o TCC foi um projeto protótipo real para uma startup que futuramente me garantiu uma vaga de estágio nessa mesma empresa.

Aprendi várias tecnologias diferentes em áreas diferentes, por mais que meu forte fosse o back-end, um estagiário ainda tinha que executar algumas funções para dar o suporte ao projeto, não tendo especificamente uma área onde trabalhar.

Por um desafio do meu CEO aos estagiários, quem conseguisse uma certificação Microsoft teria sua efetivação garantida, e eu consegui tirar a minha certificação na tecnologia/linguagem de programação C#, passando no exame “Exam 70-483: Programming in C#”.

- Observação 1: o exame me custou 100$ para poder me inscrever, mais ou menos 420 R$ naquele período.

- Observação 2: criei meu primeiro cartão de crédito só para fazer isso.

- Observação 3: nesse desafio os estagiários seriam reembolsados o valor da inscrição caso conseguissem passar em um desses exames.

Ao passar do tempo mudei de empresa, ao mesmo tempo que também mudei de área, agora sendo performance. Meu trabalho já foi desenvolver e fazer manutenção de aplicativos, hoje atuo mais em assegurar o bom funcionamento de aplicativos de clientes.

Um dos motivos desse guia é que claramente maioria desses clientes possuem aplicações em Java, então essa pequena jornada que estou desenvolvendo pode me ajudar em ter um melhor profissionalismo em como atuo. Porém o verdadeiro motivo de eu estar fazendo um guia de Java, além de querer ajudar amigos e conhecidos a se tornarem desenvolvedores, é consolidar um domínio de Java antes das certificações.

Nesse processo de querer tirar as duas certificações de Java, cheguei à conclusão de que a melhor forma de se aprender é ensinando. Porque criar algum tipo de conteúdo que pudesse ensinar algum leigo ou iniciante a conseguir me acompanhar seria a prova definitiva de que meus conhecimentos são validos.

Certificações citadas:

- Exam 70-483: Programming in C#
- Java SE 8 Programmer I 1Z0-808
- Java SE 8 Programmer II 1Z0-809

- **O que é uma Linguagem de programação?**

Um bom jeito de se imaginar o que é uma linguagem de programação é pensar em um idioma. É uma forma de conseguir se comunicar com o computador. Possuímos uma nacionalidade que na maioria dos casos determina o nosso idioma, seja português, inglês, francês ou qualquer outro no mundo. O mesmo se aplica a computadores ou dispositivos eletrônicos, eles podem compreender ordens e rotinas através de sua "linguagem", o sistema binário (0 e 1).

Programar não é necessariamente escrever em 0 e 1, é uma atividade em etapas: primeiro escrevemos o código fonte, que posteriormente um compilador ou interpretador converte para código de máquina. Nessa ocasião o compilador se comporta como um tradutor, conseguindo traduzir a nossa mensagem para máquina sem que perca o seu sentido.

Esse código fonte é escrito em alguma linguagem de programação, podendo ser C#, Java, Python, C++ etc. (uma forma boa de se compreender é imaginar de novo, português, inglês ou francês). A única diferença entre uma linguagem outra é quase a mesma que a diferença entre os idiomas: possuem regras e sintaxes diferentes. Porém as diferenças nas linguagens podem ter impacto nas ferramentas que oferecem, em suas peculiaridades, vantagens e desvantagens, porque além de uma linguagem também é uma tecnologia.

Entre as especialidades e características das linguagens, destaca-se os seus níveis. Antes de tudo, esses níveis não são sobre o grau de dificuldade em seu aprendizado especificamente, eles são sobre o nível de compreensão humana. Quanto maior o nível da linguagem mais próxima ela é dos humanos, enquanto mais distante das máquinas. O contrário também se aplica:

- Linguagem de alto nível: compreensão mais próxima de humanos

- Linguagem de baixo nível: compreensão mais próxima das máquinas

Esses níveis na prática resumem a abstração da linguagem e a complexidade em ser copiladas ou interpretadas para a máquina (tradução). É importante avisar que não é uma regra uma linguagem de baixo nível ser impossível de se aprender ao mesmo tempo que uma de alto nível tenha

pouco controle do hardware da máquina, as linguagens de programação também são um produto comercial e tentam evoluir para continuar ativas na competição entre qual os desenvolvedores vão escolher.

- Observação: Java é considerada uma linguagem de alto nível.

- **O que é um Algoritmo?**

Com um pouco de compreensão sobre linguagens de programação é mais fácil entender o que é um algoritmo. Por definição é uma rotina, uma sequência finita de passos:

- Exemplo:

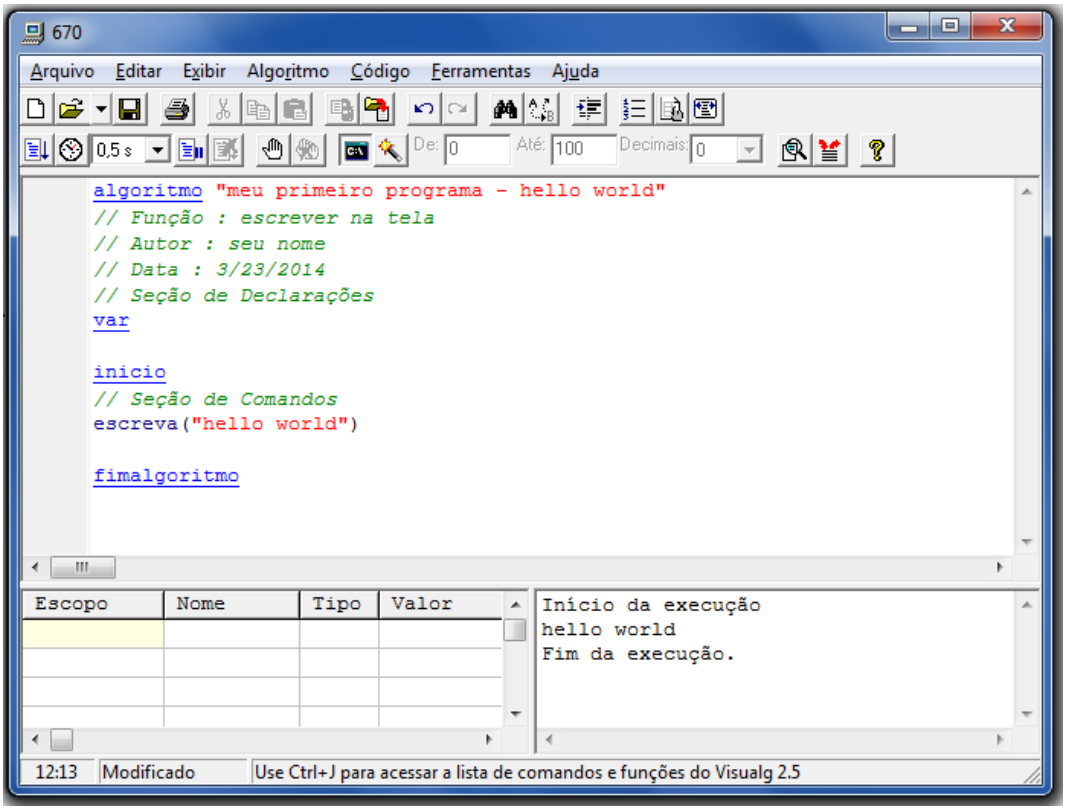
```
Algoritmo "soma"
Var Num1, num2, num3, resultado:inteiro

Inicio
escreval("este programa ira somar 3 números inteiros de sua escolha:")
escreval("digite um numero inteiro:")
leia(num1)
escreval("digite um numero para somar ao primeiro numero:")
Leia (num2)
escreval("digite um terceiro numero para somar aos outros 2 numeros:")
Leia (num3)
Resultado <- num1+num2+num3
escreval("O resultado é: ")
escreval (resultado)
fimalgoritmo
```

Fonte: https://www.infoescola.com/informatica/logica-de-programacao/

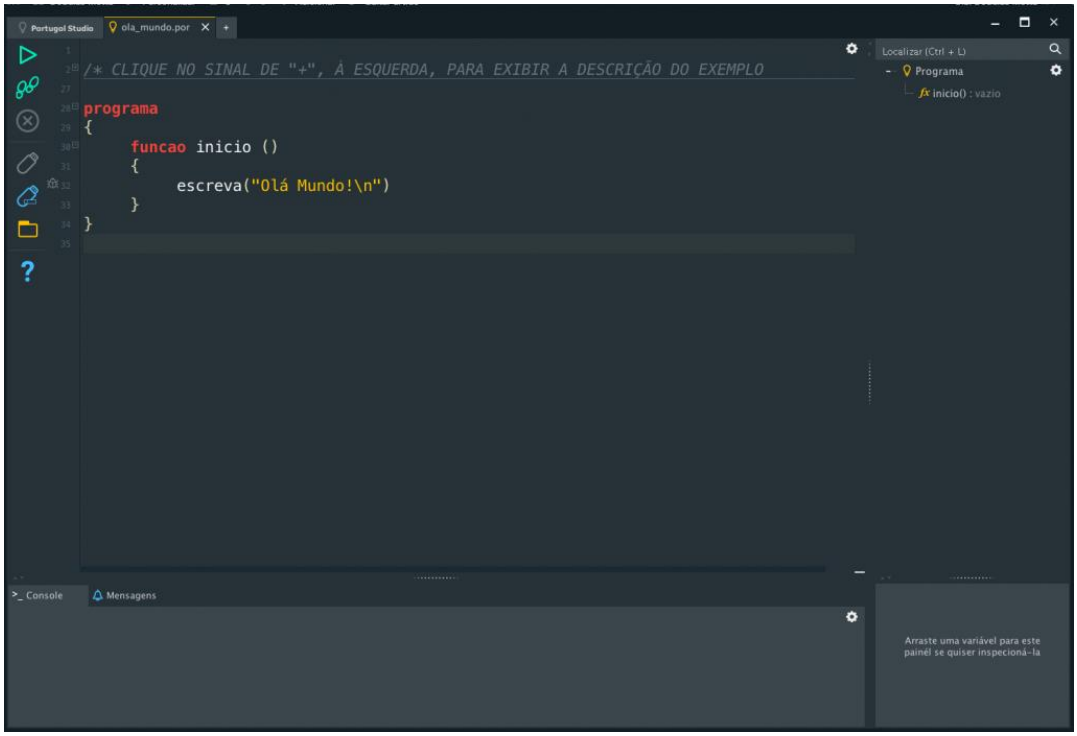
Existem algumas ferramentas que podem auxiliar novos desenvolvedores, entre elas temos o Portugol, um pseudocódigo feito para simular programação através de palavras em português para fácil entendimento. Portugol não é uma linguagem de programação, é importante frisar, apenas uma pseudolinguagem ou pseudocódigo (pseudo = superficialmente, falso ou mentira, mas que finge ser).

É muito comum o uso do Portugol para ensinar novos desenvolvedores a entender algoritmos e lógica de programação, por conta de palavras em português e alto explicativas que futuramente serão as mesmas só que em inglês em uma linguagem real.

- Hello World em Portugol:



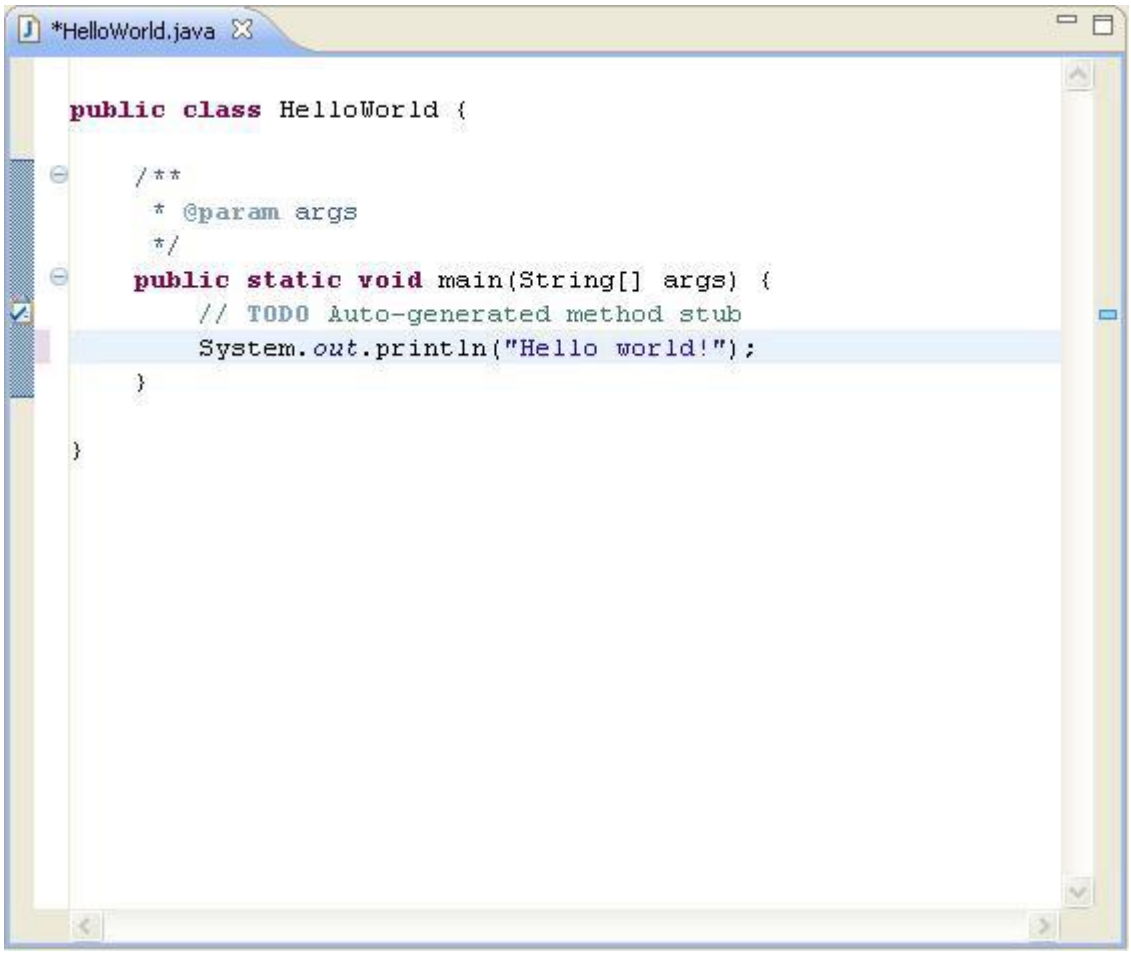
Fonte: https://cleversonabreu.wordpress.com/2014/03/25/hello-world-seu-primeiro-programa/



Fonte: https://douglasmotta.com.br/2019/02/05/aprender-programacao-na-pratica-portugol-studio/

- Observação: as duas imagens trata-se de Portugol, porém usando duas ferramentas (no caso editores) diferentes: a primeira é VisuAlg (icônica por ser muito usada em cursos e faculdades) e a segunda é Portugol Studio (referência a Visual Studio, uma ferramenta da Microsoft)

- Hello World em Java:



Fonte: https://medium.com/@tavocarvalho/hello-world-b70235f84590

- Observação: não há necessidade de entender o que cada linha do código significa, é apenas um exemplo do mesmo algoritmo escrito em pseudolinguagem Portugol em uma linguagem de programação de fato (em Java).

O algoritmo é como uma rotina, receita de bolo ou lista de compras, quaisquer conjuntos de ações em uma ordem de execução é um algoritmo. Somando a linguagens de programação, o algoritmo pode ser escrito em diversas linguagens e podendo ter o mesmo sentido, como visto anteriormente tanto em Portugol quanto Java, o mesmo pode ser aplicado a Javascript e Python, dificilmente a lógica muda.

Nesses exemplos de algoritmo é importante ressaltar que receitas são o exemplo mais correto, já que uma lista de compra e a fazeres podem ser feitos em qualquer ordem, enquanto uma receita de bolo o passo a passo tem que ser feito em uma ordem, sendo totalmente semelhante a um algoritmo.

- **O que é um Compilador?**

Compilador foi brevemente resumido como um tradutor na sessão “O que é programação?”, mas como funciona especificamente? Um compilador é um programa, ou seja, um software ou aplicação, que traduz um algoritmo em linguagem de alto nível para código de máquina ou linguagem de baixo nível, para finalmente ser passado ao processador.

Ele não gera diretamente um código de máquina, mas em linguagem simbólica (*assembly*) que é basicamente uma versão de baixo nível do que foi escrito no código fonte (nesse caso, alto nível). A aplicação, agora escrita em *assembly*, é traduzido em linguagem de máquina (binário) através dos montadores (*assemblers*, sim *assembler* é montador em inglês) do dispositivo que está executando o processo.

Esse mesmo arquivo *assembly* é o que contém diretamente as instruções e operações que devem ser realizadas pelo processador. Cada comando ou instrução dentro do código *assembly* se transforma em uma sequência de *bits* (digito binário, 0 ou 1). É importante lembrar que cada máquina tem um montador (*assembler*) criado pelo projetista do processador, por isso a relação entre montador e processador.

- Observação 1: a complexidade desse tópico deve-se ao processo de transformar uma linguagem de alto nível (fácil compreensão a humanos) em uma linguagem de baixo nível (fácil compreensão a máquinas), novamente pela diferença entre idioma e código binário.

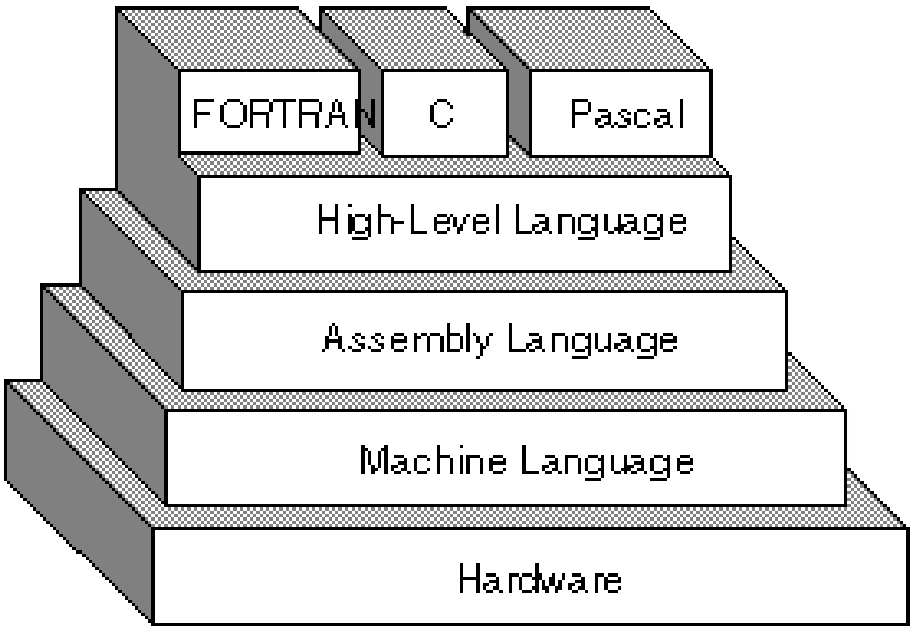
- Observação 2: é importante saber que nem todos os compiladores se comportam dessa maneira, alguns escrevem diretamente código-objeto, embora ainda tenham a opção de gerarem uma saída em *assembly* de forma tradicional

- Observação 3: alguns compiladores produzem um outro código de alto nível como saída. É o caso do primeiro compilador C++, o *cfront*, que produzia código C como saída. Assim como Lua e Java que são compiladas para *bytecode* e vão direto para código de máquina sem o processo de um montador (*assembler*).

- Observação 4: não é nem o compilador ou *assembly* que realmente produz um executável. Esse trabalho é feito pelo **Linker**, um programa do computador que consegue conectar (*linkar*) todos os

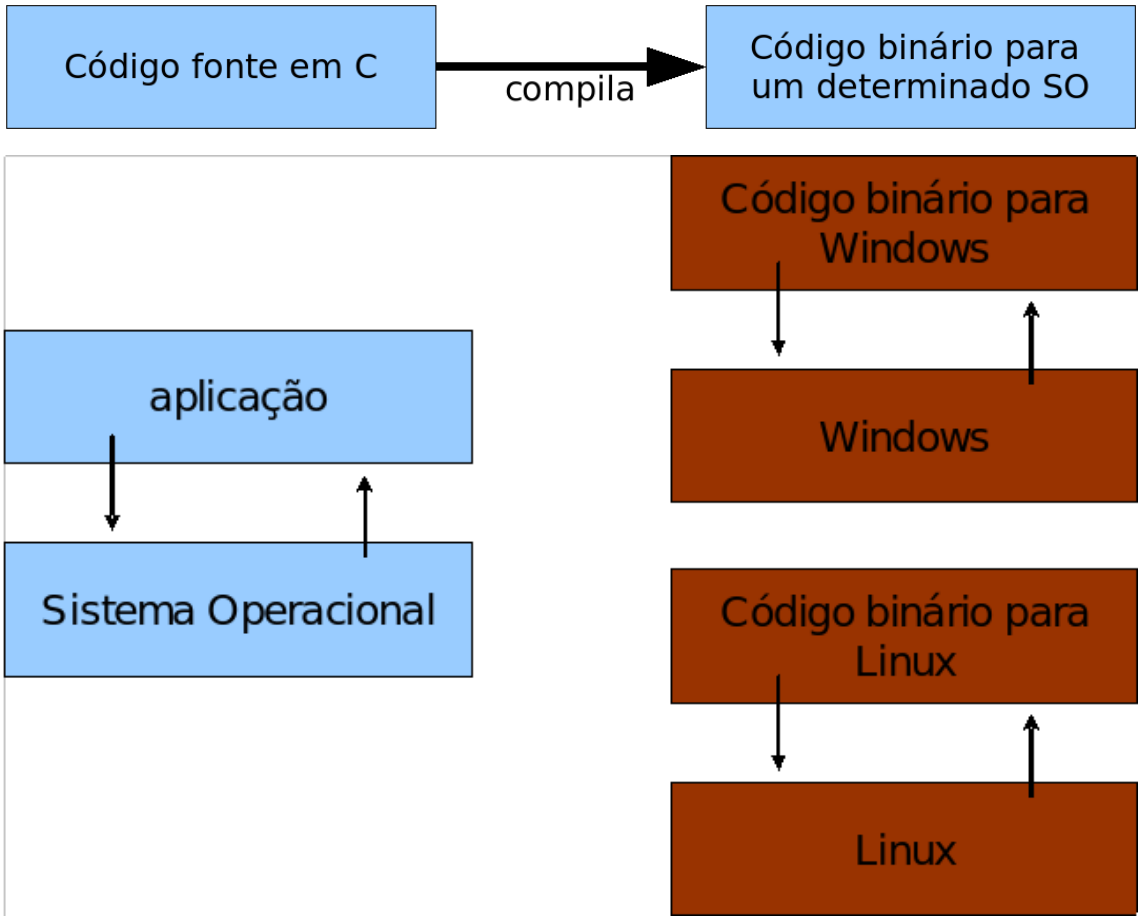
arquivos-objetos gerados pelo *assembly* ou compilação para finalmente um binário executável final (um .exe).

- Visualização das etapas de algumas linguagens até a comunicação com o hardware (a parte física da máquina):

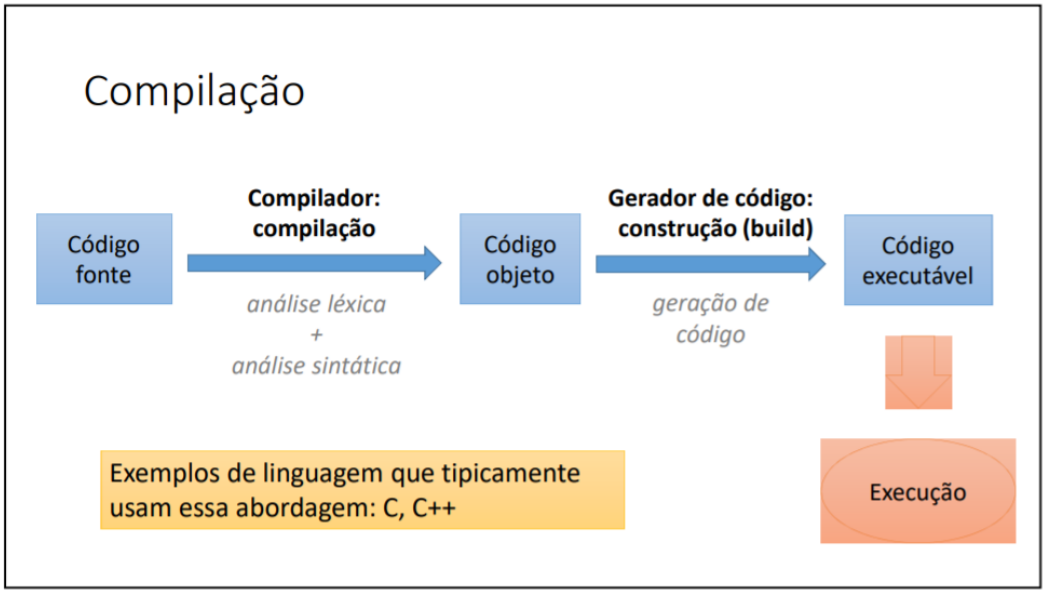


Fonte: https://minutodaseguranca.blog.br/importancia-da-linguem-de-programacao-de-baixo-nivel/

- Exemplos do processo de um compilador:



Fonte: https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java#maquina-virtual



Fonte: https://www.udemy.com/course/java-curso-completo/

Quase como “contraponto” a compilação existe a interpretação. Nota-se, o compilador produz um novo código a partir de outro, o código alvo que na maioria dos casos é código de máquina ou *assembly*. Nesse processo um novo arquivo está sendo gerado, logo todo o código fonte é lido inteiramente e compilado para o código alvo. O que é oposto do interpretador, que não gera um arquivo de código alvo, mas ao invés disso interpreta o código fonte em tempo real.

Exemplificando: compilador seria escrever um texto em português para depois ser traduzido para um novo texto em inglês, enquanto o interpretador seria escrever um texto em português e ele ser lido e traduzido em tempo real por um poliglota.

E tanto nesse exemplo quanto no processo real dá para observar claras vantagens e desvantagens nas duas soluções:

- Compilador: no exemplo da tradução vão ser sempre dois textos feitos, se houver algum erro, além da necessidade de corrigir a cópia original (em português) é preciso traduzir novamente para uma nova cópia corrigida (em inglês).

- Interpretador: seguindo o exemplo nesse caso não possuí o problema do compilador porque é apenas um texto e não dois, mas caso haja algum erro, esse erro não foi revisado e corrigido porque o texto ainda não foi lido por completo.

Compilador no processo de gerar um novo código é necessária a leitura inteira do código-fonte, logo se houver algum erro o programador será alertado a impossibilidade de tradução (na grande maioria das vezes apontando qual erro e o porquê). O interpretador executa em tempo real, ele não tem o problema de realizar manutenção em dois arquivos, porém no processo de execução, por ser em tempo real, podem ocorrer erros no processo sem aviso prévio.

Vantagens da compilação:

- Velocidade do programa.
- Auxílio do compilador antes da execução.

Vantagens da interpretação:

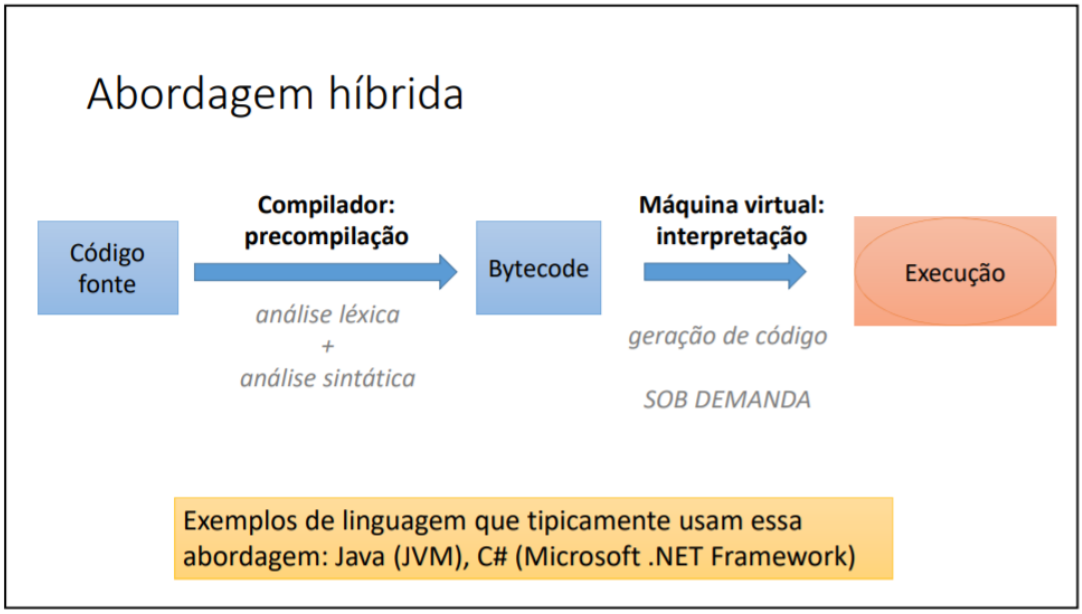
- Flexibilidade da manutenção do software.
- Linguagem expressiva, que fornece expressões complexas e de fácil entendimento em poucas linhas.
- Código fonte não precisa ser recompilado para funcionar em outras plataformas ou sistemas operacionais (Windows, Mac ou Linux).

Qual dos dois escolher então? A resposta é que não importa. Já passou o tempo em que isso era uma questão pertinente. Hoje as linguagens de programação atingiram um patamar de poderem ser ambos interpretadas quanto compiladas, mas especificamente variando entre esses dois a fim de manter uma melhor performance da aplicação.

Esse processo híbrido é feito especificamente pelo JITer (compilador JIT, Just-In-Time, ou compilador sob demanda), um compilador diferente do tradicional que citamos (que também é chamado de AOT, Ahead Of Time, antes do tempo). Ele compila partes do código antes da execução, igual um compilador tradicional, e compila outras partes em tempo de execução, que é praticamente o conceito do interpretador.

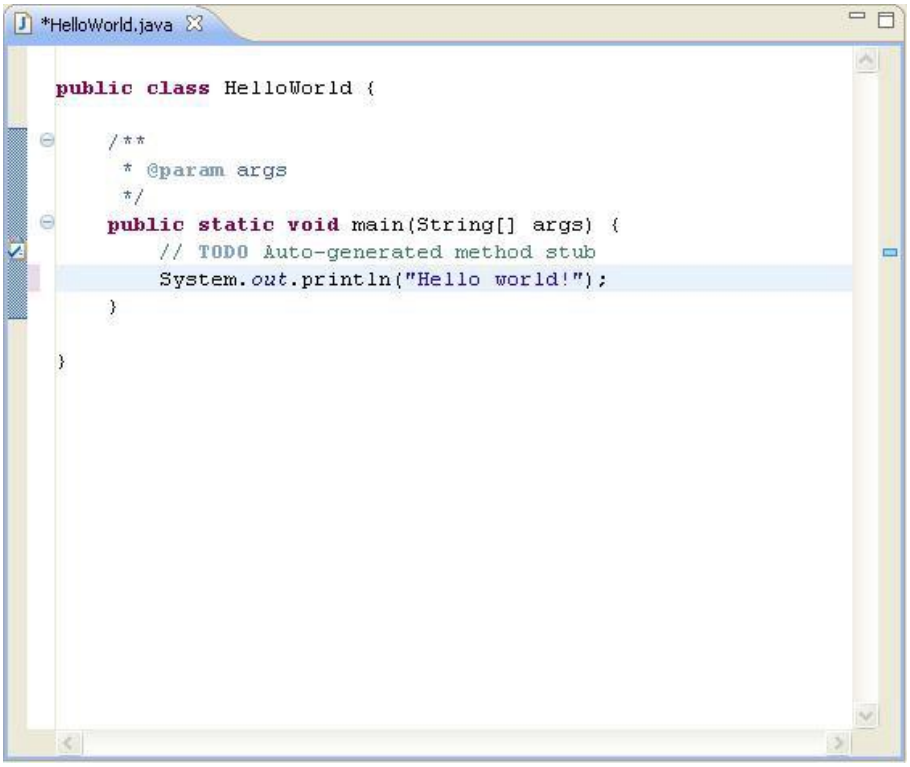
A fundo desse processo, a forma que o JIT decide entre compilar e interpretar é através dos *hotspots* (pontos quentes), compilando apenas o que ele considera como *hot* (quente) que são as partes usadas várias e várias vezes pela aplicação, enquanto o resto é interpretado. Exemplificando melhor, o JIT em primeira mão compila tudo em tempo real (interpretação) e monitora alguma parte se há bastante repetição, caso ele a encontre ela é marcada (*track*) e compilada (gerando de fato um *assembly* ou o próprio código de máquina).

- Funcionamento da abordagem híbrida (JIT):



Nessa sessão foi mostrado o processo do compilador, de um código fonte de fácil entendimento humano a um código alvo de baixo nível. Mas especificamente quais são as diferenças na prática?

- Hello World em Java:



Fonte: https://medium.com/@tavocarvalho/hello-world-b70235f84590

- Hello World em Assembly

```
    .equ    LAST_RAM_WORD,   0x007FFFFC
    .equ    JTAG_UART_BASE,  0x10001000
    .equ    DATA_OFFSET,     0
    .equ    STATUS_OFFSET,   4
    .equ    WSPACE_MASK,     0xFFFF

    .text
    .global _start
    .org    0x00000000

_start:
    movia       sp, LAST_RAM_WORD
    movi        r2, '\n'
    call        PrintChar
    movia       r2, MSG
    call        PrintString
_end:
    br          _end

PrintChar:
    subi        sp, sp, 8
    stw         r3, 4(sp)
    stw         r4, 0(sp)
    movia       r3, JTAG_UART_BASE
pc_loop:
    ldwio       r4, STATUS_OFFSET(r3)
    andhi       r4, r4, WSPACE_MASK
    beq         r4, r0, pc_loop
    stwio       r2, DATA_OFFSET(r3)
    ldw         r3, 4(sp)
    ldw         r4, 0(sp)
    addi        sp, sp, 8
    ret

PrintString:
    subi        sp, sp, 12
    stw         ra, 8(sp)
    stw         r3, 4(sp)
    stw         r2, 0(sp)
    mov         r3, r2
ps_loop:
    ldb         r2, 0(r3)
    beq         r2, r0, end_ps_loop
    call        PrintChar
    addi        r3, r3, 1
    br          ps_loop
end_ps_loop:
    ldw         ra, 8(sp)
    ldw         r3, 4(sp)
    ldw         r2, 0(sp)
    addi        sp, sp, 12
    ret

        .org        0x1000
MSG:    .asciz      "Hello World\n"
        .end
```

Fonte: https://towardsdatascience.com/hello-world-not-so-easy-in-assembly-23da6644ff0d

- Observação: é importante lembrar que em nenhuma dessas duas imagens é necessário saber o que cada linha de código representa, é um exemplo da conversão de alta compreensão humana para alta compreensão de máquina.

- **O que é Java?**

Java é uma tecnologia, mais precisamente uma linguagem de programação, mas também é uma plataforma desenvolvimento e execução, já que pode usado tanto para desenvolver aplicativos quanto para executar esses mesmos em dispositivos com o Java instalado. Essa tecnologia foi lançada pela primeira vez em maio de 1995 pela empresa Sun Microsystems, que agora pertence a Oracle.

- Observação 1: em 2010 a Oracle comprou a tecnologia Java e seus direitos da Sun Microsystems por 7.4 bilhões de dólares.

- Observação 2: duas semanas atrás desse evento histórico a IBM tentou o Java por 9,4 bilhões de dólares, mas as negociações falharam.

Java já foi por muito tempo foi a linguagem de programação mais usada no mundo. São mais de 3 bilhões de dispositivos com Java instalado (computadores, celulares, Smart TVs etc.). Um exemplo que muitos devs fazem é considerar Java como um "inglês" entre os idiomas existentes. De fato, quando algum iniciante entra no mundo da programação e não sabe qual linguagem começar provavelmente Java é a tecnologia mais incentivada.

- Observação: em novembro de 2020 Python conseguiu superar Java como linguagem de programação mais popular, título esse que pertencia ao Java por duas décadas.

Os destaques do Java vêm muito das necessidades da época em que foi criado. Na década de 90, linguagens de programação além de não serem muito amigaveis apresentavam algumas situações desconfortáveis aos devs (desenvolvedores):

- Ponteiros.
- Gerenciamento de memória.
- Organização confusa.
- Falta de bibliotecas.
- Reescrever parte do código para cada sistema operacional (Windows, Linux ou Mac).
- Custo financeiro da tecnologia.

A proposta do Java foi tornar essas situações mais agradáveis ao mesmo que oferecia algumas qualidades como por exemplo a sua simplicidade.

- Observação: hoje Java não é mais considerado uma linguagem tão simples em comparação com outras mais modernas, o que só demonstra a dificuldade de se programar na década de 90.

Por um fim de construir um senso crítico sobre a linguagem é importante também mencionar alguns problemas e polémicas sobre a linguagem. Alguns devs e empresas se esforçam para não adotar o Java como sua tecnologia principal. Conversando com alguns amigos, conhecidos e colegas de trabalho, sempre escutei que faculdades e cursos aos poucos estão mudando a forma de ensinar programação, mais precisamente optando por Python, C# ou Javascript ao invés de Java que era ensinado anteriormente.

- Observação: algumas dessas faculdades ou cursos tem um escopo de primeiro ensinar Portugol ou C, depois partir para Java como linguagem para fim de curso.

Agora é um pouco de opinião minha: acredito que um pouco dessa queda de popularidade do Java deve-se ao incidente que ocorreu em 2012. Nesse mesmo ano foi descoberto uma falha de segurança no plugin do Java, onde uma pessoa mal-intencionada poderia executar comandos remotos no computador da vítima. Esse problema levou mais de um ano para ter alguma correção. Para se ter um nível de ideia do ocorrido, na época o governo dos EUA emitiu um comunicado oficial recomendando que os usuários deixassem de utilizar Java.

- Observação: a empresa de segurança AlienVault conseguiu na época reproduzir essa falha executando um comando remoto que abria o app Calculadora em uma máquina com Windows XP e Java 7, confirmando mais uma vez a gravidade do assunto.

Até então Java poderia ser considerado o futuro da internet, e agora Python e Javascript disputam esse título. Toda essa história cria algumas questões: Java ainda é relevante? Por qual linguagem eu devo começar? Qual a melhor linguagem de programação?

Java ainda é uma das linguagens de programação mais utilizada no mundo, um bom paralelo de se imaginar é pensar nele como o "inglês", enquanto Python e Javascript como "mandarim", uma nova tendencia de idioma (assim como espanhol já foi uma vez). É claro que existe muito mercado para essas linguagens, recentemente dúvidas no Stack Overflow sobre essas tecnologias passaram o Java diversas vezes, assim como as mudanças em faculdades e cursos são um reflexo das mudanças no mercado de trabalho.

Não há uma resposta sobre qual a melhor linguagem, ou qual é a melhor para iniciantes, porém a minha pessoa indicaria Java para uma boa jornada de programador (e não sou o único com essa ideia).

- Observação 1: Stack Overflow é um site de perguntas e respostas sobre tecnologia, o principal entre concorrentes, é como um "Yahoo! Respostas" para devs, que consultam constantemente pare resolver tantos problemas nos estudos quanto no próprio trabalho.

- Observação 2: é importante lembrar que Javascript e Python são muito amigaveis a novos devs, principalmente Python que é mais facial de ser compreendido por alguém que não é programador do que um programador de fato (isso deve-se ao fato de ser muito diferente do estilo padrão que as linguagens compartilham). Mesmo assim, acredito que o mercado de trabalho esteja muito voltado ao Java, o que conversando com outros devs parece que essa realidade já mudou ou está nesse processo de "revolução".

Se sua decisão foi Java como linguagem de programação a se aprender, ou pelo menos está indeciso, acredito que algumas características e qualidades podem responder se vale ou não apena investir seu tempo em aprender:

- Java está na maioria das plataformas: maioria das vagas, dos sistemas e dos dispositivos possuem relação com essa tecnologia.

- Simples: possui bastante semelhança com outras linguagens como C, C++ e C#, o que pode ser intuitivo para devs que adotaram Java como segunda linguagem a se aprender.

- Baseado em classes e orientado a objetos: possui conceitos e abstração próximos com a nossa realidade, uma característica bem popular entre as linguagens.

- E uma linguagem e uma plataforma: além de ser uma linguagem de programação também é uma plataforma para rodar apps, já que é necessário ter uma versão do Java para usuários rodarem aplicativos dessa tecnologia (alguns dispositivos já vêm com Java instalado)

- Independente de plataforma: graças a tecnologia da JVM (máquina virtual Java), o código fonte Java é convertido em uma linguagem intermediaria chamada de Byte Codes. A JVM consegue compilar bytecodes para código da máquina o qual está instalado, seja um computador com Windows ou Linus, ou até uma Smart TV. Assim não é necessário reescrever o código para cada plataforma.

- Multi-Thread: possui a tecnologia de threads, onde pode executar múltiplas tarefas de uma vez.

- Coleta de lixo (Garbage Collector): a própria tecnologia lida com o gerenciamento de memória evitando possiveis leaks (fuga).

- Distribuição: além de ser gratuito oferece diversas bibliotecas e ferramentas para os devs.

Opinião pessoal: é uma clássica e importante linguagem de programação para se adicionar em seu conhecimento.

- **O que é JVM?**

A melhor forma de se começar esse tópico é com um bordão:

"Write Once, Run Anywhere".

Também conhecido como "WORA" (as iniciais da frase), ou em português "escreva uma vez, execute em qualquer lugar", é o bordão do Java que conquistou o mundo. Pode-se dizer que o Java foi o pioneiro entre as linguagens em ser multiplataforma.

Cada linguagem poderia sim ser compilada para cada sistema operacional ou dispositivo, contato que você a programasse de novo e usasse um compilador para código de máquina legível daquele dispositivo, então era um dos principais trabalhos de um programador reescrever o código para cada sistema operacional ou dispositivo dentro do escopo do seu projeto (Windows, Mac, Linux, desde um celular a um cartão e os exemplos continuam).

O Java possuí uma tecnologia capaz de conseguir escrever apenas uma vez e funcionar para qualquer dispositivo que tenha Java instalado, o conceito de “WORA”. Esse processo deve-se a JVM.

Entender a JVM (Java Virtual Machine, Máquina Virtual Java) é um dos principais conceitos para se entender a própria tecnologia Java, já que o mérito do seu clássico bordão é graças a JVM.

Antes de aprofundar na JVM, dois conceitos são extremamente importantes: bytecode e máquina virtual

O que é uma máquina virtual? Pense em uma simulação de um computador, ele executa softwares, gerencia processo, memoria, arquivos etc., igual a um computador de fato. Porém por ser uma simulação (projeção) ele não possui hardware físico, já que é gerado por uma outra máquina. Ter essa noção é diferencial para não confundir a JVM com um simples compilador ou interpretador.

Mas e o bytecode o que seria? Quando compilamos o código fonte Java (.java) geramos arquivos escritos em bytecode (.class). Bytecode é uma linguagem intermediária feita para a JVM, é importante ressaltar que a JVM não compreende Java, apenas bytecode.

Esse processo é um pouco diferente de outras linguagens de programação:

- Outras linguagens:

1. Código fonte: feito pelo programador.

2. Compilador: transforma código fonte em código binário ou *assembly*.

3. Sistema Operacional: consegue processar as ordens porque entende o código binário, ou o seu montador (*assembler*) converte o *assembly* em finalmente código binário.

Nesse processo algumas linguagens de programação fazem compilação diretamente para o sistema operacional. Ou seja, para mudar de sistema operacional mudaria tanto o compilador quanto o código fonte feito (teria que ser adaptado), já que as referências a bibliotecas e recursos estavam voltadas a aquele sistema ou dispositivo (um bom exemplo é lembrar que a mesma biblioteca gráfica do Windows não é a mesma do Mac, mudando-se as referências e o próprio código).

- Java

1. Código fonte Java: feito pelo programador.

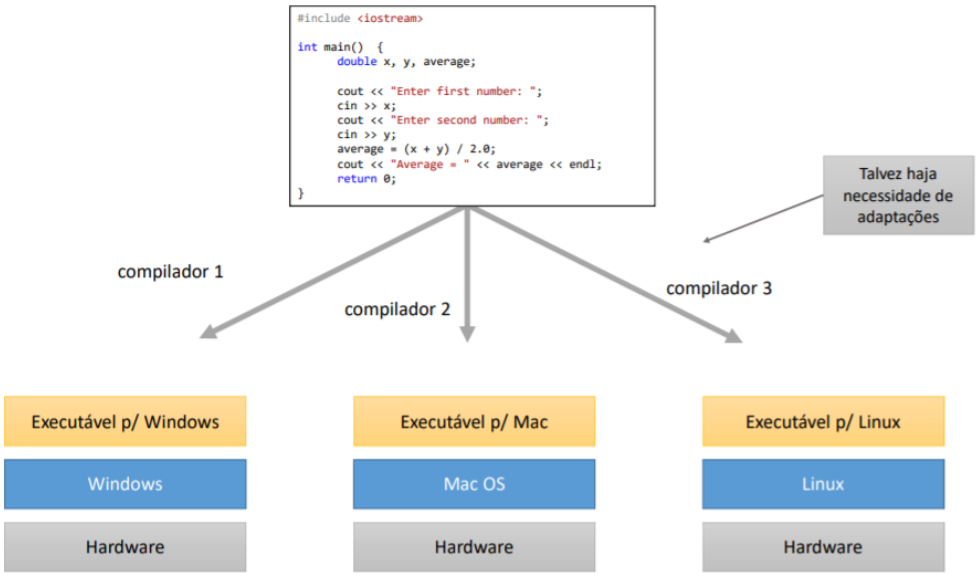
2. Compilador Java: transforma código fonte Java em bytecode.

3. JVM: transforma bytecodes em código nativo da máquina onde está instalado.

Sistemas operacionais ou dispositivos eletrônicos que possuem a JVM instalada conseguem converter bytecode para seu código nativo, mesmo sendo um computador com Mac ou uma Smart TV, contanto que possua uma versão da JVM para esse tipo de dispositivo. A responsabilidade de execução e link (*Linker*) de bibliotecas é da própria JVM projetada para aquele tipo de dispositivo, não mais do montador ou do programador.

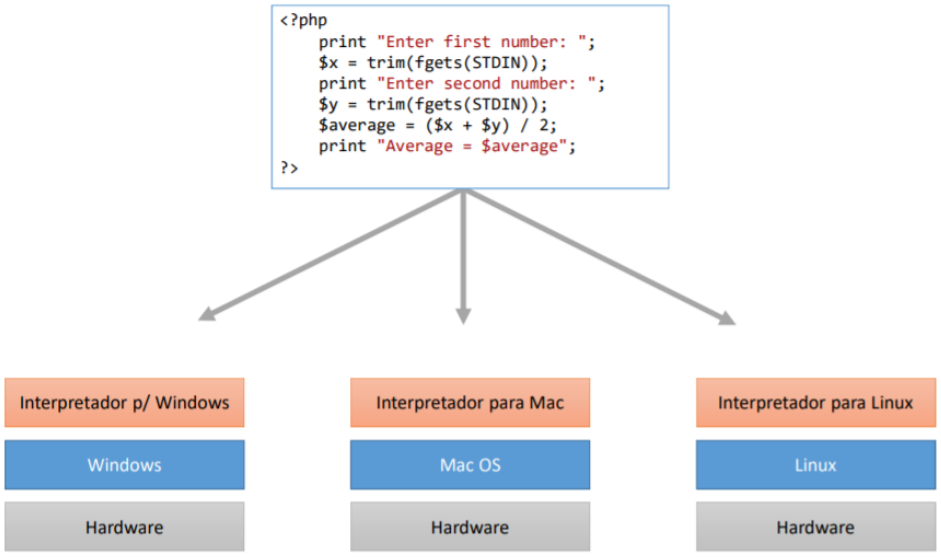
- Observação: a JVM de cada sistema é desenvolvida em código nativo daquele sistema pelos projetistas do Java, por isso a facilidade em não precisar se preocupar com a mudança de cada sistema operacional ou dispositivo.

- Exemplo de compilador:



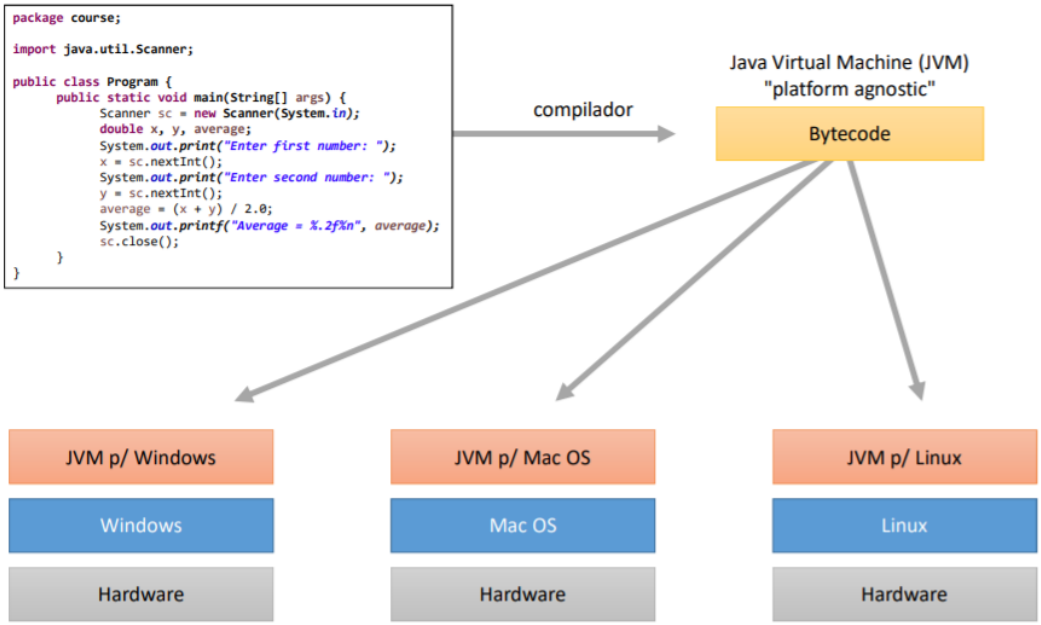
Fonte: https://www.udemy.com/course/java-curso-completo

- Exemplo de interpretador:

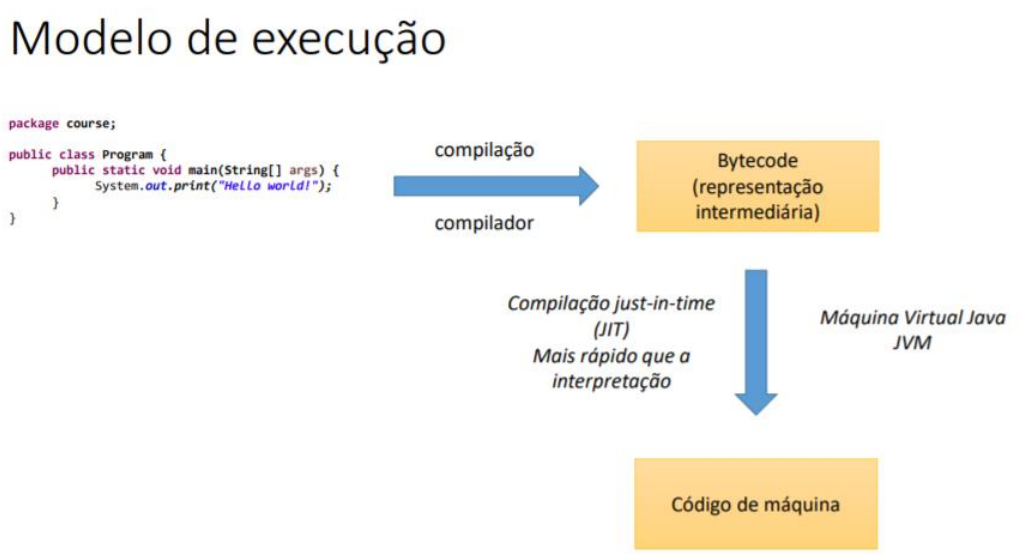


Fonte: https://www.udemy.com/course/java-curso-completo

- Exemplo do pré-compilador + máquina virtual:



Fonte: https://www.udemy.com/course/java-curso-completo



Fonte: https://www.udemy.com/course/java-curso-completo

Como mencionando anteriormente em “Compilador”, além de compilador e interpretador há a abordagem híbrida entre esses dois, e linguagens de programação C# e Java a utilizam. A linguagem é pré-compilada e depois a máquina virtual interpreta em tempo de execução, monitorando quando compilar ou interpretar, os *hotspots* também já mencionado.

Esquematizando: o código Java é passado para um pré-compilador; que transforma numa representação intermediaria (*bytecode*), já compilado e com garantias no ponto de vista sintático (não tem erros no código); o código já verificado é rodado numa máquina virtual para aquele dispositivo.

- Observação 1: o mesmo bytecode é utilizado para todas as máquinas virtuais, ou seja, não precisa de um compilador para cada dispositivo e reescrever o código como já mencionado.

- Observação 2: essa abordagem híbrida tem a vantagem dos dois mundos, compilador e interpretador, como já observado em “Compilador”

Uma dúvida que é muito comum é se a JVM é apenas um compilador ou se assemelha com ele, o mesmo é perguntado com interpretador. A resposta é não. Se observado anteriormente os processos que a JVM executa para fazer esse processo em que o Java roda em qualquer lugar (WORA), é possível perceber o tanto de funcionalidades que é executado pela JVM, que ela se assimila mais o que é em seu conceito, uma máquina virtual, um computador.

A JVM, além da função de transformar bytecode em código de máquina e realizar o JIT, gerencia pilhas, threads, memória etc., quase tudo “automatizado”, o que oferece um grau a mais de profundidade do que um simples compilador ou interpretador.

- Observação: o motivo da JVM se comportar como um computador que auxilia o desenvolvedor com vários tópicos é o objetivo principal de simplicidade. Coisas como gerenciamento de memória e coleta de lixo (Garbage Collector) que anteriormente eram tópicos muito complexos são feitos praticamente pela JVM sozinha.

Uma outra dúvida comum é porque não programar diretamente em bytecode? Antes de qualquer coisa a resposta é que é praticamente impossível.

O bytecode (.class) é escrito de uma forma quase irreconhecível para humanos, apenas a JVM pode entender o bytecode. É claro que existem métodos de engenharia reversa para poder converter o bytecode para Java, isso é claro se você não tiver o arquivo de código fonte (uma situação real é querer saber como funciona um aplicativo em Java de forma ilegal), ou converter eles de uma forma mais compreensível, mas ainda em bytecodes (como exemplo chinês e chinês simplificado).

Outra razão que desmotiva essa dúvida é que mesmo se fosse prático programar em bytecode ainda estaria violando todos os princípios básicos do Java, sua simplicidade e praticidade. Para todo o processo de multiplataforma ocorrer precisa de uma linguagem muito complexa, o compilador transforma uma linguagem simples nessa linguagem complexa e depois a JVM consegue executar nos dispositivos.

Um exemplo muito bom para entender esse tópico é imaginar aquele primeiro exemplo do “Compilador”. Passando para o JIT, seria um tradutor que primeiro passaria o seu português informal para formal, para então ser traduzido para inglês. É muito mais fácil e prático escrever de forma informal do que formal.

- Observação 1: é bem difícil escrever sobre esse assunto sem lembrar da CLR (Common Language Runtime), que é praticamente o bytecode da tecnologia .NET (C#).

- Observação 2: a JVM também consegue interpretar outras linguagens como Groovy, Kotlin e muitas outras. Existe implementações de outras linguagens já existentes que JVM consegue interpretar como por exemplo Python e PHP.

- Observação 3: é importante lembrar que a JVM não compreende Java, mas sim bytecode, por isso o processo de passar primeiro por um pré-compilador antes da JVM, onde se tornará bytecode.

Fontes

Essa parte é dedicada para as fontes que eu usei para fazer esse guia, desde cursos, livros, vídeo aulas ou o meu próprio conhecimento geral em programação.

- Java 2020 COMPLETO: Do Zero ao Profissional + Projetos!

https://www.udemy.com/course/fundamentos-de-programacao-com-java/

- Programação em Java 11 LTS do básico ao avançado

Autor: Geek University.

https://www.udemy.com/course/programacao-em-java-essencial/

- Java COMPLETO 2020 Programação Orientada a Objetos +Projetos

https://www.udemy.com/course/java-curso-completo/

- Qual a diferença entre linguagem compilada para linguagem interpretada?

https://pt.stackoverflow.com/questions/77070/qual-a-diferen%C3%A7a-entre-linguagem-compilada-para-linguagem-interpretada#:~:text=Primeiro%2C%20n%C3%A3o%20existe%20isso%20de,o%20c%C3%B3digo%20escrito%20naquela%20linguagem

- Java's Virtual Machine and CLR

https://stackoverflow.com/questions/453610/javas-virtual-machine-and-clr

- Java

https://www.devmedia.com.br/java/

- Java: Entenda para que serve o software e os problemas da sua ausência

https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2014/11/java-entenda-para-que-serve-o-software-e-os-problemas-da-sua-ausencia.html

- O que é Java? [Guia para iniciantes]

https://tecnoblog.net/416833/o-que-e-java-guia-para-iniciantes/

- Aprenda tudo sobre o que é uma linguagem de programação e para que ela serve

https://www.hostmidia.com.br/blog/linguagem-de-programacao/

- Introdução ao Java Virtual Machine (JVM)

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

- JVM por dentro

https://vepo.medium.com/jvm-por-dentro-553bab89a027

- Qual a diferença entre JDK, JRE e JVM

https://dicasdejava.com.br/qual-a-diferenca-entre-jdk-jre-e-jvm/

- Java: tudo o que você precisa saber para começar

https://www.zup.com.br/blog/java

- Falha de segurança no Java ainda não tem correção

https://www1.tecnoblog.net/122033/falha-exploit-java/

- O que é Java

https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java

- As linguagens de programação mais procuradas pelo mercado

https://www.digitalhouse.com/br/blog/linguagens-de-programacao-mais-procuradas

- O que é Java?

https://www.lenovo.com/br/pt/faqs/notebook-faqs/java/

- O que é uma linguagem de programação, para que serve e como funciona?

https://www.voitto.com.br/blog/artigo/linguagem-de-programacao

- Lógica de Programação

https://www.infoescola.com/informatica/logica-de-programacao/

- Oracle Certified Associate (OCA) Java 8

https://www.udemy.com/course/oracle-certified-associate-oca-java-8/

- Metodologia de Certificação Java 8 OCA: Simulados

https://www.udemy.com/course/metodologia-de-certificacao-java-8-oca-simulados/

- Use a Cabeça!: Java

https://www.amazon.com.br/Use-cabe%C3%A7a-Java-Bert-Bates/dp/8576081733/ref=sr_1_5?__mk_pt_BR=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=java&qid=1607348149&sr=8-5

- Java: The Complete Reference, Eleventh Edition

https://www.amazon.com.br/Java-Complete-Reference-Herbert-Schildt/dp/1260440230/ref=sr_1_3?__mk_pt_BR=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=java&qid=1607348149&sr=8-3

- OCA / OCP Java SE 8 Programmer Certification Kit: Exam 1Z0-808 and Exam 1Z0-809

https://www.amazon.com.br/OCA-OCP-Java-Programmer-Certification/dp/1119272092/ref=sr_1_7?__mk_pt_BR=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=OCJP&qid=1607348294&sr=8-7

- Aprender programação na prática: Portugol Studio

https://douglasmotta.com.br/2019/02/05/aprender-programacao-na-pratica-portugol-studio/

- Hello World – Seu Primeiro Programa

https://cleversonabreu.wordpress.com/2014/03/25/hello-world-seu-primeiro-programa/

- Hello World!

https://medium.com/@tavocarvalho/hello-world-b70235f84590

- Hello World!

https://www.dca.fee.unicamp.br/~eleri/ea876/04/cap10.pdf

- Universidade Estadual de Campinas - Unicamp

https://www.dca.fee.unicamp.br/~eleri/ea876/04/

- Os compiladores de linguagem de programação traduzem primeiro para Assembly ou diretamente para código de máquina?

https://www.ti-enxame.com/pt/compiler-construction/os-compiladores-de-linguagem-de-programacao-traduzem-primeiro-para-assembly-ou-diretamente-para-codigo-de-maquina/957545347/

- Importância da linguagem de Programação de Baixo Nível

https://minutodaseguranca.blog.br/importancia-da-linguem-de-programacao-de-baixo-nivel/

- Métodos de tradução: interpretador x compilador

https://imasters.com.br/desenvolvimento/metodos-de-traducao-interpretador-x-compilador

- O que é um JITter?

https://pt.stackoverflow.com/questions/146250/o-que-%c3%a9-um-jitter

- Métodos de tradução: interpretador x compilador

https://blog.geekhunter.com.br/metodos-de-traducao-compiladores-ou-interpretadores/

- Jvm JIT and Hotspot - What are the differences

https://stackoverflow.com/questions/6851525/jvm-jit-and-hotspot-what-are-the-differences

- Java Hello World - Your First Program

https://www.java-made-easy.com/java-hello-world.html

- Hello World…Not so easy in Assembly

https://towardsdatascience.com/hello-world-not-so-easy-in-assembly-23da6644ff0d

- Pela primeira vez em 20 anos, Python supera Java como a linguagem de programação mais popular

https://sempreupdate.com.br/pela-primeira-vez-em-20-anos-python-supera-java-como-a-linguagem-de-programacao-mais-popular/#:~:text=Python%20ultrapassa%20Java%20como%20a,%2C%20finalmente%2C%20Python%20supera%20Java