A Plataforma Java

Quando a linguagem Java nasceu, ela chamava a atenção por conta das seguintes características:

- **Orientado a Objeto (O.O.)**:

O Java segue os princípios da programação orientada a objetos (POO). Isso inclui conceitos como encapsulamento, herança e polimorfismo. No Java, tudo é considerado um objeto, o que ajuda na organização e estruturação do código. (Vamos ver tudo isso nas aulas).

- **Muitas Bibliotecas**:

Uma das grandes vantagens do Java é a extensa quantidade de bibliotecas (também conhecidas como APIs - Application Programming Interfaces) disponíveis. Isso facilita muito o desenvolvimento, pois você pode aproveitar funcionalidades já implementadas, economizando tempo e esforço. As bibliotecas padrão do Java cobrem uma ampla gama de funcionalidades, desde manipulação de arquivos até comunicação de rede.

- **Semelhança com C++**:

Na época em que o Java foi desenvolvido, a linguagem C++ era amplamente utilizada. O Java foi projetado para ser semelhante ao C++ em muitos aspectos, tornando a transição dos programadores de C++ para Java mais suave. No entanto, ao longo do tempo, algumas diferenças fundamentais se tornaram evidentes, e as comunidades de desenvolvimento em Java e C++ evoluíram de maneiras distintas.

- **Portabilidade (Roda em vários sistemas operacionais)**:

Uma característica fundamental do Java é a portabilidade. O código Java é compilado para um formato intermediário chamado bytecode, que pode ser executado em qualquer máquina virtual Java (JVM - Java Virtual Machine). Isso significa que, uma vez que você escreve um programa Java, ele pode ser executado em várias plataformas sem a necessidade de recompilação. Essa abordagem "write once, run anywhere" (escreva uma vez, execute em qualquer lugar) é uma das razões pelas quais o Java é tão popular para o desenvolvimento de aplicações corporativas.

Linguagem sem JVM (exemplo: C++):

- Compilação Direta para Código de Máquina:
Em linguagens como C++, o código fonte é escrito em um arquivo de texto. Este código é então compilado diretamente para código de máquina específico da arquitetura da CPU. O resultado é um arquivo executável que pode ser diretamente executado pelo sistema operacional.

- Dependência da Plataforma:
Como o código é compilado diretamente para a arquitetura de máquina específica, o executável resultante é dependente da plataforma. Isso significa que se você quiser executar o mesmo programa em diferentes sistemas operacionais ou arquiteturas de CPU, precisará recompilar o código para cada plataforma desejada.

Desempenho Otimizado:
A vantagem desse método é que o código é otimizado para a máquina específica, proporcionando um desempenho geralmente mais rápido. No entanto, a desvantagem é a falta de portabilidade entre diferentes plataformas sem recompilação.

Linguagem com JVM (Java):

- Compilação para Bytecode:
Em Java, o código fonte é escrito em arquivos .java. Este código é compilado para um formato intermediário chamado bytecode por meio do compilador Java (`javac`). O bytecode não é específico para nenhuma arquitetura de hardware e é independente da plataforma.

- Máquina Virtual Java (JVM):
O bytecode gerado é interpretado e executado pela JVM, uma máquina virtual que está disponível em diferentes sistemas operacionais. A JVM traduz o bytecode em código de máquina específico da plataforma durante a execução, permitindo que o mesmo bytecode seja executado em várias plataformas sem a necessidade de recompilação.

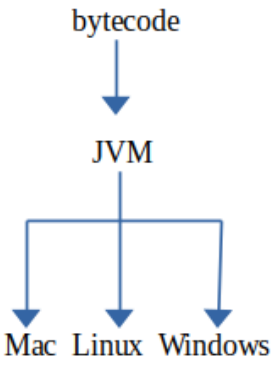
- Portabilidade:
A abordagem "write once, run anywhere" é possível porque o bytecode é portável. Você pode desenvolver um programa Java em um sistema operacional e executá-lo em qualquer outro sistema que tenha uma JVM compatível instalada, sem a necessidade de modificar o código-fonte.

Gerenciamento de Memória:
A JVM também gerencia automaticamente a alocação e desalocação de memória, proporcionando uma camada adicional de segurança e simplicidade para os desenvolvedores, em comparação com linguagens que exigem gerenciamento manual de memória, como C++.

Exemplo do formato entendido pela *virtual machine* (JVM), o *bytecode*, é o seguinte:

```
Compiled from "Onibus.java"
class Teste {

    public static void main(java.lang.String);
      Code:
      0: new           #2  // class Onibus
      3: dup
      4: invokespecial #3 Onibus."<init>":()V
      7: astore_1
      8: aload_1
      9: ldc           #4 // String Jabaquara...
     11: putfield      #5
        // Field Onibus.linha:Ljava/lang/String;
     14: return
}
```



Mas o código acima não parece ser de fácil leitura e compreensão. Para meios de comparação, segue um exemplo de um arquivo **.java,** a ser compilado e traduzido para **.class**, o tal do bytecode:

```
public class Onibus {
  String nome;
  String linha;
}

class Teste {
  public static void main(String args) {
    Onibus o = new Onibus();
    o.linha = "Jabaquara-Liberdade";
  }
}
```

Ao compilar esse codigo .**java** vai ser criado o **.class (bytecode)** que será entendido pela JVM.

Uma breve História e Evolução do Java:

Applets e Versatilidade Inicial: No seu início em 1995, o Java foi projetado para ser versátil, rodando em vários dispositivos e sistemas operacionais. Inicialmente, o foco estava em applets, que eram pequenos programas que podiam ser executados dentro de navegadores. Isso exigia a instalação do Java no navegador para funcionar.

Mudança para Server-Side:

Foco em Server-Side: Com o tempo, o Java encontrou forte aceitação no lado do servidor (server-side). Ao desenvolver aplicações e sistemas web complexos, a portabilidade do Java entre diferentes sistemas operacionais se tornou uma vantagem significativa. Empresas grandes, como bancos e órgãos governamentais, evitam depender de sistemas operacionais específicos ou de navegadores (Vendor lock-in).
Liberdade e Multiplataforma: O Java proporciona liberdade ao quebrar a dependência de versões específicas de sistemas operacionais e navegadores. Isso é crucial para evitar o Vendor lock-in, onde uma empresa fica "preso" a uma única solução ou tecnologia.

Conceito de Máquina Virtual Java (JVM):

Multiplataforma: A JVM é fundamental para a abordagem "write once, run anywhere". O código Java é compilado para bytecode, que é executado na JVM. Isso permite que um programa Java seja executado em qualquer lugar onde exista uma JVM, independentemente do sistema operacional subjacente.
Gerenciamento de Memória: A JVM cuida automaticamente da alocação e desalocação de memória, facilitando o desenvolvimento e tornando o código mais seguro contra vazamentos de memória.

Segurança e Sandbox: A JVM oferece uma camada de segurança através de um ambiente conhecido como "sandbox". Isso limita o acesso do programa a certos recursos do sistema, garantindo que ele não cause danos indesejados.
Otimizações e JIT Compiler: A JVM realiza otimizações no código Java durante a execução, e o compilador JIT (Just-In-Time) converte partes do bytecode em código de máquina nativo para melhorar o desempenho.

Enfoque na Plataforma e Ecossistema Java:

Além da Linguagem Java: O foco está mais na plataforma e no ecossistema Java do que apenas na linguagem Java em si (nesse curso será utilizado a linguagem java). Outras linguagens, como Ruby, Clojure e Scala, podem gerar bytecode Java. Isso significa que você pode utilizar diferentes linguagens de programação que são convertidas para bytecode Java e, em seguida, executadas na JVM.
Independência da Pilha Tecnológica: Empresas valorizam a JVM, pois ela oferece independência da "pilha tecnológica" abaixo dela. Isso significa que podem escolher diferentes tecnologias para o restante da aplicação, mantendo a portabilidade fornecida pela JVM.
Acesso a Bibliotecas: A JVM também oferece acesso a uma vasta variedade de bibliotecas e frameworks disponíveis no ecossistema Java, facilitando o desenvolvimento e evitando a necessidade de reinventar a roda.