° JRE (Java Runtime Environment):

- O JRE é necessário para executar aplicações Java já compiladas. Ele inclui a Máquina Virtual Java (JVM), bibliotecas Java e outros componentes necessários para a execução de programas Java.
- O JRE é essencial para usuários que desejam **apenas executar** aplicativos Java, sem a necessidade de desenvolver ou compilar código-fonte.

- Principais componentes do JRE:

- Class Loader:
Carrega classes Java na JVM durante a execução. Pode ser personalizado para carregar classes a partir de diferentes fontes, como sistemas de arquivos locais ou servidores remotos.
- Byte Code Verifier:
Garante que o bytecode gerado durante a compilação é seguro e atende às restrições da JVM para prevenir potenciais problemas de segurança.
- Java API (Application Programming Interface):
Conjunto de classes e métodos que formam a biblioteca padrão do Java. Fornece funcionalidades para entrada/saída, manipulação de strings, coleções, rede, GUI, entre outros.
- Runtime Libraries:
Bibliotecas essenciais fornecidas pela JRE para suportar a execução de aplicações Java. Inclui classes para manipulação de exceções, threads, operações de E/S, entre outras.

° JDK (Java Development Kit):

- O JDK é um conjunto mais abrangente de ferramentas do que o JRE. Ele inclui tudo no JRE e também ferramentas adicionais para desenvolvimento e depuração de código Java.

- Principais componentes do JDK:

- java:
É a ferramenta usada para executar aplicações Java compiladas. Você fornece o nome da classe principal como argumento, e a JVM é iniciada para executar o programa.
- Javac (Java Compiler):
É a ferramenta responsável por compilar o código-fonte Java (.java) em bytecode Java (.class) que pode ser executado pela JVM.
- Javadoc:
Permite a geração de documentação automaticamente a partir de comentários no código-fonte Java.
- jdb (Java Debugger):
Ferramenta de depuração interativa que permite aos desenvolvedores depurar suas aplicações Java. Oferece funcionalidades como pontos de interrupção, inspeção de variáveis e execução passo a passo do código.
- Appletviewer:
Utilizado para executar e depurar applets Java fora de um navegador web. Applets eram componentes Java usados anteriormente para criar conteúdo interativo em páginas web.
- javah:
Gera arquivos de cabeçalho (header files) necessários para a implementação de métodos nativos em Java. Esses métodos podem ser implementados em linguagens como C ou C++.
- javaw:
É uma versão do comando `java` que é usada para iniciar aplicativos Java sem uma janela de console. Geralmente utilizado para aplicativos GUI.
- jar (Java Archive): Ferramenta para criar, visualizar e manipular arquivos JAR, que são arquivos compactados que podem conter classes Java e outros recursos.

- rmi (Remote Method Invocation): Suporte para o desenvolvimento de aplicações distribuídas em Java, permitindo que métodos de objetos Java sejam invocados remotamente.

° Na JVM (Java Virtual Machine):

Java Interpreter:
Interpreta o bytecode e executa as instruções da aplicação Java. A interpretação direta do bytecode pode resultar em uma execução mais lenta em comparação com outras técnicas.
JIT (Just-In-Time Compiler):
Compila parte do bytecode para código de máquina nativo do sistema durante a execução. Isso visa melhorar o desempenho, pois o código nativo pode ser executado mais eficientemente do que o bytecode interpretado.
Garbage Collector:
Gerencia a coleta de lixo, identificando e liberando a memória ocupada por objetos não utilizados. Isso ajuda a evitar vazamentos de memória.
Thread Synchronization:
Gerencia a sincronização entre threads para garantir que as operações concorrentes ocorram de maneira ordenada e sem conflitos.

Resumo do que foi falado:

Falando passo a passo sobre a compilação:

O **javac** é o compilador Java incluído no JDK. Ele transforma o código-fonte Java legível por humanos em bytecode Java, que é uma forma de código intermediário interpretado pela JVM.

Para compilar um arquivo Java chamado "MeuPrograma.java", você usaria o seguinte comando no terminal:

javac MeuPrograma.java

Isso gerará um arquivo "MeuPrograma.class", que pode ser executado pela JVM.

Para executar um programa Java após compilar o código-fonte com o javac, você usaria o comando **java** seguido pelo nome da classe principal (a classe que contém o método main). O comando para execução seria:

java MeuPrograma

O Java Virtual Machine (JVM) é responsável por interpretar e executar o bytecode gerado pelo compilador. Aqui está um resumo de como funciona internamente:

1- Compilação:
Quando você executa **javac MeuPrograma.java**, o compilador Java (javac) lê o código-fonte contido no arquivo "MeuPrograma.java" e o traduz para bytecode Java. Esse bytecode é salvo em um arquivo com a extensão ".class".

2- Execução:
2.1. Ao executar **java Programa**, a JVM é iniciada.
2.2. A JVM carrega a classe principal (no caso, "MeuPrograma") e procura pelo método **public static void main(String[] args)**. Esse método serve como ponto de entrada para a execução do programa Java.
2.3. A JVM interpreta o bytecode e executa as instruções contidas no método main. Durante a execução, a JVM gerencia a alocação de memória, a execução de threads e outras operações de baixo nível necessárias para a execução da aplicação.

3- Ciclo de Vida da Máquina Virtual Java:
3.1. A JVM é uma máquina virtual que executa programas Java em um ambiente independente de plataforma.
3.2. Durante a execução, a JVM realiza a just-in-time compilation (compilação sob demanda), onde partes do bytecode são compiladas para código de máquina nativo do sistema operacional no qual a JVM está sendo executada. Isso ajuda a melhorar o desempenho da execução.

4- Garbage Collection:
A JVM também gerencia automaticamente a coleta de lixo (garbage collection) para liberar a memória ocupada por objetos não utilizados, facilitando o desenvolvimento de programas livres de vazamentos de memória.