## Resumo

Jit pega uma parte do código que é repetido várias vezes e transforma ele em código máquina que é lido mais rápido pela CPU e isso melhora a performance do programa

o jit pode estar relacionado com a componentização. pq é o mesmo conceito de de uma parte de código que vai ser lida na hora, sem conhecimento prévio da sua criação.(eu acho)

o jit do windows é o Common Language Runtime (CLR)

Just-in-time compilation is a method for improving the performance of interpreted programs. During execution the program may be compiled into native code to improve its performance. It is also known as dynamic compilation.

Dynamic compilation has some advantages over static compilation. When running Java or C# applications, the runtime environment can profile the application while it is being run. This allows for more optimized code to be generated. If the behavior of the application changes while it is running, the runtime environment can recompile the code.

Some of the disadvantages include startup delays and the overhead of compilation during runtime. To limit the overhead, many JIT compilers only compile the code paths that are frequently used.

## **Overview**

Traditionally there are two methods for converting source code into a form that can be run on a platform. Static compilation converts the code into a language for a specific platform. An interpreter directly executes the source code.

JIT compilation attempts to use the benefits of both. While the interpreted program is being run, the JIT compiler determines the most frequently used code and compiles it to machine code. Depending on the compiler, this can be done on a method or smaller section of code.

Dynamic compilation was first described in a paper by J. McCarthy on LISP in 1960.

Just In Time Compilation, JIT, or Dynamic Translation, is compilation that is being done during the execution of a program. Meaning, at run time, as opposed to prior to execution. What happens is the translation to machine code. The advantages of a JIT are due to the fact, that since the compilation takes place in run time, a JIT compiler has access to dynamic runtime information enabling it to make better optimizations (such as inlining functions).

What is important to understand about the JIT compilation, is that it will compile the bytecode into machine code instructions of the running machine. Meaning, that the resulting machine code is optimized for the running machine’s CPU architecture.

Some examples of JIT Compilers are JVM (Java Virtual Machine) in Java and CLR (Common Language Runtime), in C#.

## **History**

In the beginning, a compiler was responsible for turning a high-level language (defined as higher level than assembler) into object code (machine instructions), which would then be linked (by a linker) into an executable.

At one point in the evolution of languages, compilers would compile a high-level language into pseudo-code, which would then be interpreted (by an interpreter) to run your program. This eliminated the object code and executables, and allowed these languages to be portable to multiple operating systems and hardware platforms. Pascal (which compiled to P-Code) was one of the first; Java and C# are more recent examples. Eventually the term P-Code was replaced with bytecode, since most of the pseudo-operations are a byte long.

A Just-In-Time (JIT) compiler is a feature of the run-time interpreter, that instead of interpreting bytecode every time a method is invoked, will compile the bytecode into the machine code instructions of the running machine, and then invoke this object code instead. Ideally the efficiency of running object code will overcome the inefficiency of recompiling the program every time it runs.

### **Typical scenario**

The source code is completely converted into machine code

### **JIT scenario**

The source code will be converted into assembly language like structure [for ex IL (intermediate language) for C#, ByteCode for java].

The intermediate code is converted into machine language only when the application needs that is required codes are only converted to machine code.

## **JIT vs Non-JIT comparison**

In JIT not all the code is converted into machine code first a part of the code that is necessary will be converted into machine code then if a method or functionality called is not in machine then that will be turned into machine code, which reduces burden on the CPU. As the machine code will be generated on run time, the JIT compiler will produce machine code that is optimized for running machine’s CPU architecture.

Some examples of JIT are:

* Java: JVM (Java Virtual Machine)
* C#: CLR (Common Language Runtime)
* Android: DVM (Dalvik Virtual Machine) or ART (Android RunTime) in newer versions

The Java Virtual Machine (JVM) executes bytecode and maintains a count as of how many time a function is executed. If this count exceeds a predefined limit JIT compiles the code into machine language which can directly be executed by the processor (unlike the normal case in which javac compiles the code into bytecode and then Java, the interpreter interprets this bytecode line by line converts it into machine code and executes).

Also next time this function is calculated same compiled code is executed again unlike normal interpretation in which the code is interpreted again line by line. This makes execution faster.