Java Compilation process

The Java compilation process involves a 2-step-process. The first being run through a OS independent compiler and second in a Virtual Machine ( JVM ) which is custom-built for every OS. During the compilation, the java file id passed through the compiler which compiles the java code into a machine independent code known as Bytecode. While being compiled / converted from the java file / source code into the bytecode, the compiler follows several steps in order to be completed.

1. **Parse:** Reads a set of \*.java source files and maps the resulting token sequence into AST (Abstract Syntax Tree)-Nodes.
2. **Enter:** Enters symbols for the definitions into the symbol table.
3. **Process annotations:** If Requested, processes annotations found in the specified compilation units.
4. **Attribute:** Attributes the Syntax trees. This step includes name resolution, type checking and constant folding.
5. **Flow:** Performs dataflow analysis on the trees from the previous step. This includes checks for assignments and reachability.
6. **Desugar:** Rewrites the AST and translates away some syntactic sugar.
7. **Generate:** Generates ‘.Class’ files.

To execute the program, the class files, that are generated by the compiler, are independent of the machine or the OS. This allows them to be run on any system. To run the code, the main class file is passed to the JVM. It then goes through 2 main stages, which are:

1. **The Class Loader -** A class loader, itself an object, creates a flat name space of class bodies that are referenced by a string name.
2. **Bytecode Verifier -** After the bytecode of a class is loaded by the class loader, it has to be inspected by the bytecode verifier, whose job is to check that the instructions don’t perform damaging actions
3. **JIT Compiler ( Just-In-Time Compiler ) -** This is the final stage encountered by the java program, and its job is to convert the loaded bytecode into machine code.