Rafian Athallah Marchansyah

2440042380

Java’s compilation process does not work with a one-step compilation, but instead involves a two-step execution, where it goes through an OS-dependent compiler first then into the java virtual machine.

In the compilation stage, the source .java file is first passed through a compiler that encodes the source code into a machine independent encoder called Bytecode. The contents of each class is stored in a separate .class file. While converting to Bytecode, the compiler goes through a few steps.

First step is parsing, where it reads a set of .java source files and maps them into abstract syntax tree nodes. Next step is enter, where it enters symbols that can be defined into the symbol table. Next is process annotations, where if requested, it processes annotations found in the specified compilation units. Next is attributing, where it attributes syntax trees including name resolution, type checking and constant folding. Flow is up next, where it performs dataflow analysis on the trees from the previous steps, including checking for assignments and reachability. Then, it desugars, where it rewrites the abstract syntax tree nodes and translates some syntactic sugar. Finally, it generates the .class files.

After the compilation stage is done, the next stage is execution, where the class files generated by the compiler are passed to the java virtual machine, which goes through three stages before the code is executed.

The first step in execution is the class loader, where the main class is passed into the java virtual machine, and invoking it. The other classes referenced in the program are then loaded through the class loader, which is an object that creates a flat name space of class bodies which are referenced by a string name. After the bytecode of a class is loaded in, it is inspected by the bytecode verifier, which checks instructions that do not perform damaging actions, such as:

* Variables initialized before being used
* Method calls that match the types of object references
* Rules for accessing private data and methods that aren’t violated
* Local variable accesses fall within the runtime stack
* The run time stack does not overflow

If any of these actions occur, the verifier doesn’t allow the class to be loaded. If it succeeds, then it goes to the final stage, which is the just-in-time compiler. It’s job is to convert the loaded bytecode into machine code, which then executes it.