Execution of Java Programs:

1. You have multiple .java files.
2. These files are compiled into their corresponding .class files by the Java compiler. The .class files contain the equivalent byteCode.
3. The .class files are then loaded into RAM by the Class Loader of the JVM, which itself resides on RAM.
4. Then the byteCode is verified by the JVM's byteCode verifier.
5. If all good, the byteCode is converted into corresponding Machine Code which is then executed by the hardware.

Java Stack, Heap & Garbage Collection:

1. Suppose you have 2 functions with local variables.
2. The first function has only local variables while the second one also has an instance variable or object created with the new operator.
3. When Java will encounter the first function, it will create a stack frame on the stack with the local variables and push it to the stack.
4. When Java will encounter the second function, it will create a stack frame on the stack with the local variables and push it to the stack as well. The object created with the new operator will be created in the heap.
5. The difference will be that the reference to the object will be pointing to the object, that gets created in the heap because of the new operator. The reference will itself be in the stack frame.
6. When the execution has taken place or the function variables go out of scope, the stack frame with the object reference as well as the local variables will be flushed out.
7. Since, no reference will be pointing to the object in the heap, it will be cleared by the Garbage Collector.
8. Then, the stack frame of the first function will also be flushed.

Just In Time (JIT) Compilation:

Java code goes through the entire process of being converted to byteCode, verification and conversion to Machine Code which is time consuming. JIT makes sure to compile the most frequently used code as well as the code that is currently being used and keeps it in memory to avoid Java code going through the entire life cycle again. The unused code is ignored and not converted.