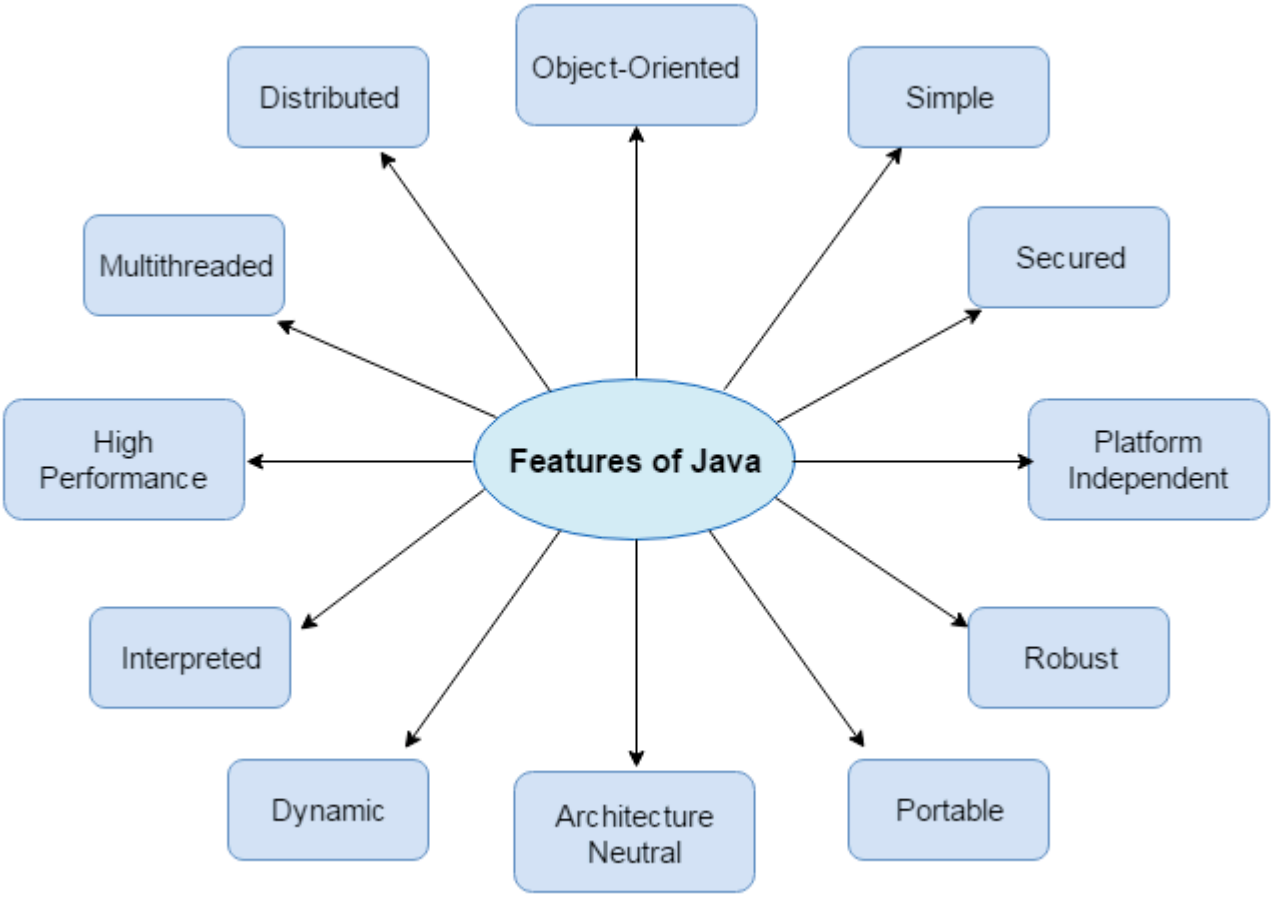
Java Features and Program Execution

Java is a **programming language** and a **platform**.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and Application Programming Interface (API), it is called platform.

Features of Java :

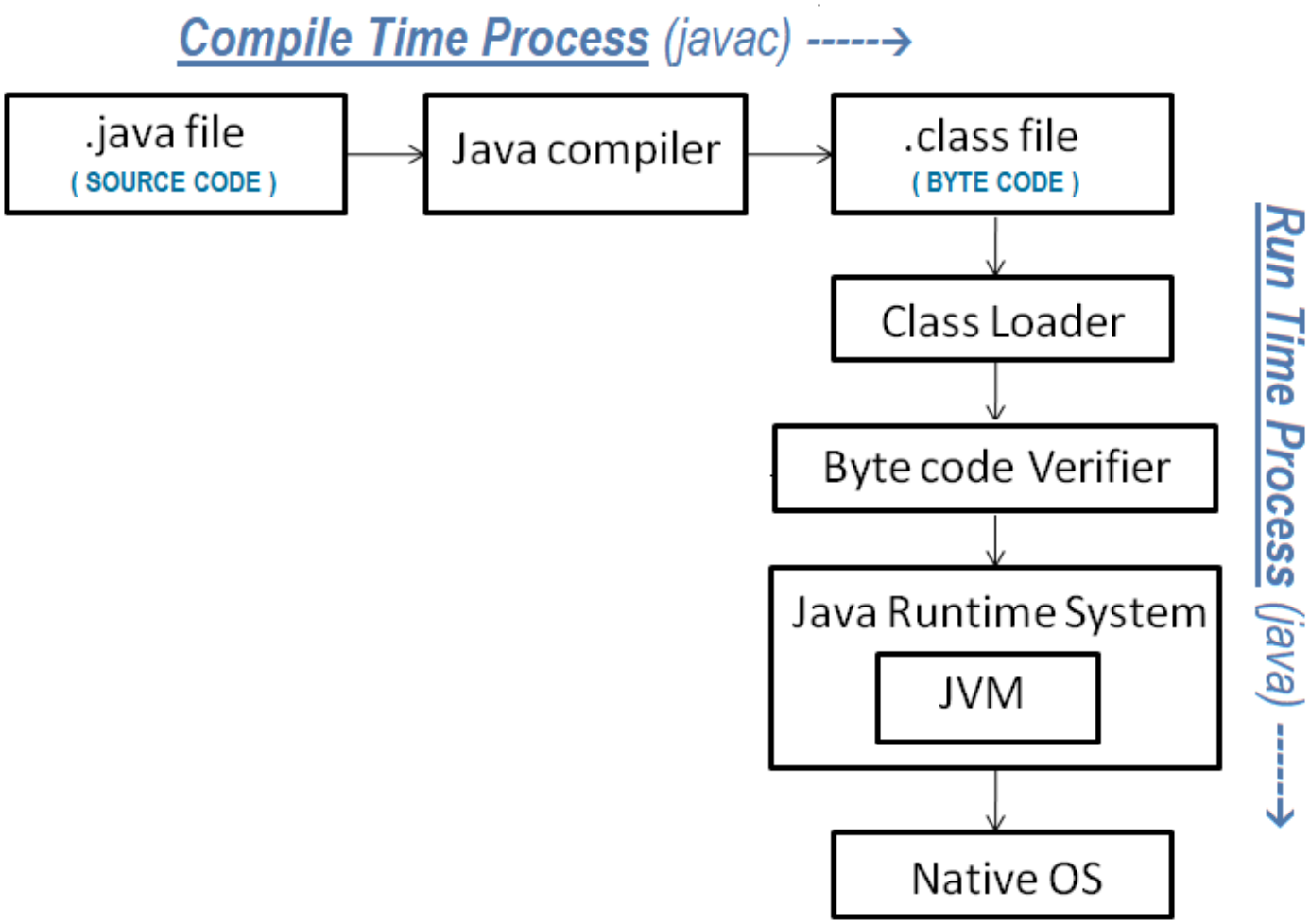


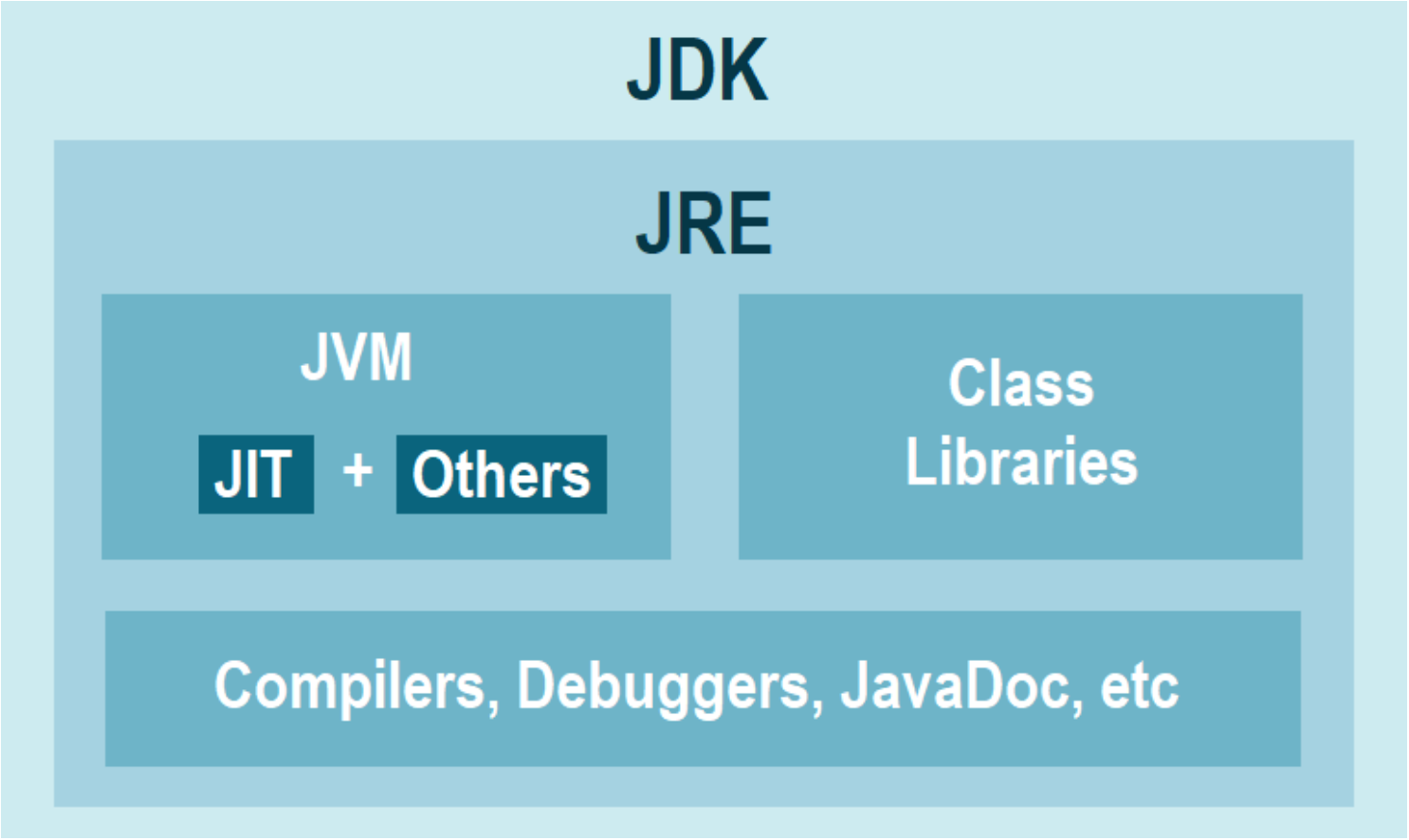
- **Simple:**
  - User friendly syntax based on C++
  - It has Automatic Garbage Collection
  - It has Rich set of APIs
  - Removed confusing features - explicit pointers, operator overloading, multiple inheritance, etc

- **Object-Oriented:**
  - In Java, we organize the software as a combination of different types of objects that incorporates both data and behavior.
  - Based on the concept of Objects, Class, Inheritance, Polymorphism, Abstraction, Encapsulation
- **Platform Independent:**
  - Java provides software-based platform. It has two components:
    - JRE (Runtime Environment)
    - API (Application Programming Interface)
  - Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent. Can run on many platforms - Windows, Linux, Mac, etc.
- **Secured:**
  - **No explicit pointer**
  - **JVM** - java Programs run inside virtual machine sandbox
  - **Classloader** - adds security by separating the package for the classes of the local file system from those that are imported from network sources.
  - **Bytecode Verifier** - checks the code fragments for illegal code that can violate access right to objects.
  - **Security Manager** - determines what resources a class can access such as reading and writing to the local disk.
  - **More** - developers can add extra security through SSL, JAAS, Cryptography etc.
- **Robust:**
  - **Good memory management** - automatic garbage collection.
  - **No pointers** - increases security.
  - **Exception handling** - increases robustness against errors.
  - **Strongly typed** - every variable must be declared with a data type.
  - **Statically typed** - type checking of variables is performed at compile time.
- **Architecture-Neutral:**
  - There is no implementation dependent features. e.g. size of primitive types is fixed.
- **Portable:**
  - Write Once and Run Anywhere.

- **Interpreted:**
    - Java is compiled to bytecodes, which are interpreted by a Java run-time environment.
    - The interpreter reads bytecode stream then execute the instructions.
- **High-Performance:**
    - **Uses ByteCode -** Java is faster than traditional interpreted languages since byte code is "close" to native code.
    - **Just-In-Time (JIT) -** it is designed to support JIT compilers, which dynamically compile bytecodes to machine code.
    - **Garbage collector -** collect the unused memory space and improve the performance of the application.
    - NOTE: Java is still slower than a compiled language like C/C++.
- **Distributed:**
    - We can create distributed applications in java. RMI and EJB are used for creating distributed applications.
    - We may access files by calling the methods from any machine on the internet.
- **Multi-threaded:**
    - A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.
    - The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
    - Threads are important for multi-media, Web applications etc.
- **Dynamic:**
    - **Dynamic Compilation (JIT) -** Implementations to gain performance during program execution. The machine code emitted by a dynamic compiler is constructed and optimized at program runtime, the use of dynamic compilation enables optimizations for efficiency.
    - **Load on Demand -** Loads in classes as they are needed, even from across the network.
    - **Dynamic memory allocation -** All Java objects are dynamically allocated.
    - **Dynamic Polymorphism -** Compiler doesn’t know which method to be called in advance. JVM decides which method to called at run time.

Java Program Execution Process :


Compile Time Process (javac) ----->
.java file
(SOURCE CODE)
Java compiler
.class file
(BYTE CODE)
Class Loader
Byte code Verifier
Java Runtime System
JVM
Native OS
Run Time Process (java) ----->
JDK - JRE - JVM - JIT :



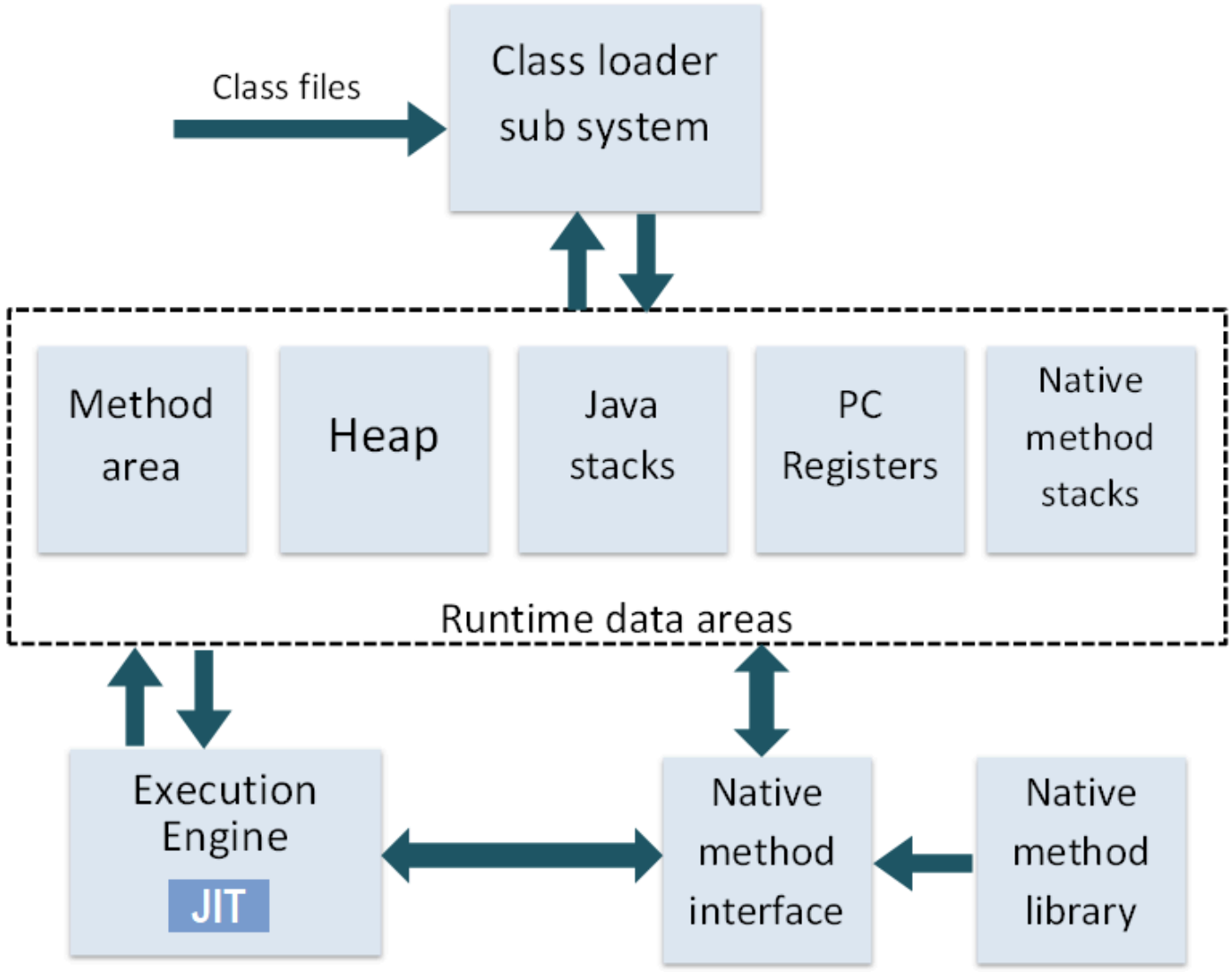
Java Development Kit (JDK): It is a collection of development tools including JRE.

Java Runtime Environment (JRE): It contains set of libraries and the JVM.

Java Virtual Machine (JVM): It is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. The JVM performs following main tasks: Loads code, Verifies code, Executes code and Provides runtime environment.

NOTE - JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

Internal Architecture of JVM :



JVM (Java Virtual Machine) has various sub components internally. You can see the most important ones in the above diagram.

- **Class loader sub system:** JVM's class loader sub system performs 3 tasks
  - It loads .class file into memory.
  - It verifies byte code instructions.
  - It allots memory required for the program.
- **Run time data area:** This is the memory resource used by JVM and it is divided into 5 parts
  - **Class (Method) area:** Stores constant pool, field and method data, the code for methods.
  - **Heap:** Objects are allocated on the heap.
  - **Java stacks:** Java stacks are the places where the Java methods are executed. A Java stack contains frames. It holds local variables and partial results, and plays a part in method

invocation and return. On each frame, a separate method is executed. Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

- **Program counter registers:** PC (program counter) register. It contains the address of the JVM instruction currently being executed.
- **Native method stacks:** Are places where native methods (eg. C language programs, etc) are executed.

- **Native method interface:** Native method interface is a program that connects native methods libraries (C header files) with JVM for executing native methods.
- **Native method library:** Holds the native libraries information.
- **Execution engine:**
  - **Just-In-Time(JIT) compiler:** It is used to improve the performance. It coverts byte code into machine code. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.
  - **Interpreter:** Read bytecode stream then execute the instructions.
  - **Virtual processor**
  - **NOTE -** JVM uses optimization technique to decide which part to be interpreted and which part to be used with JIT compiler.