**Introduction**

## **Introduction**

The Java Platform Module System was release with the Java SE 9 at march 18, 2014.

The primary goals of this Project were to:

* Make it easier for developers to construct and maintain libraries  
  and large applications;
* Improve the security and maintainability of Java SE Platform Implementations in general, and the JDK in particular;
* Enable improved application performance; and
* Enable the Java SE Platform, and the JDK, to scale down for use in small computing devices and dense cloud deployments.

In a more practical way now the platform has reached out these two main changes:

The entire JDK’s source and Java runtime were redesigned and built upon on modules

The java language has its own module system

For the developers its mean that now we can distribute our java application using our own customize Java Runtime, for example suppose you’ve wrote program that don’t make use of neither swing package nor RMI. Now you can distribute you program that run in a Java Runtime resized that don’t contain these modules, bringing a lot of benefits such as performance, bootstrap, archives size and others.

Another big improvement for developers that using java module system we can avoid the nasty issue called JAR hell

Jar hell rise because we can have multiples versions of the same Jar lying on the classpath, that. It could result in the classloader loading the first jar version found in the classpath, behavior that could cause tremendous issues in our application.

Another issue very knew by developers caused by jars in classpath was that sometimes on compiling the application any error is throw but in runtime the classnotfoundexception is throw, due the missing of jar on the classpath in runtime.

But Besides these practical things solved with modules in more formal way we can list the benefits of implementing modules at Java Platform were:

* loose coupling between components
* clear contracts and dependencies between components
* hidden implementation using strong encapsulation

And in terms of JEP we can fit the Jigsaw in these following:

* **create a module system for the language** – implemented under [**JEP 261**](https://openjdk.java.net/jeps/261)
* **apply it to the JDK source** – implemented under [**JEP 201**](https://openjdk.java.net/jeps/201)
* **modularize the JDK** **libraries** – implemented under [**JEP 200**](https://openjdk.java.net/jeps/200)
* **update the runtime to support modularity** – implemented under [**JEP 220**](https://openjdk.java.net/jeps/220)

and

* **be able to create smaller runtime with a subset of modules from JDK** – implemented under [**JEP 282**](https://openjdk.java.net/jeps/282)

**Besides this JEP another reason to implement modules system was** encapsulate the internal APIs in the JDK like the classes under the sun.\* packages and other non-standard APIs.

## **Module System Applied to JDK Source**

Now let’s see how the JDK Java APIs was.

Firstly, look at the some JDK installed directory. We can see one file named src.zip. This archive contains the code base for the JDK Java APIs. If you extract the archive, you will find multiple folders, few starting with java, few with javafx and the rest with jdk. Each folder represents a module

The modules starting with *java* are the JDK modules, those starting with *javafx* are the JavaFX modules and others starting with *jdk* are the JDK tools modules.

All JDK modules and all the user defined modules implicitly depend on the *java.base* module. The *java.base* module contains commonly used JDK APIs like Utils, Collections, IO, Concurrency among others. The dependency graph of the JDK modules is:

The next class we’ll see the new tools for the Java Module System delivered along the JDK.

## **4. New Tools for Modularity**

**jdeps** – helps in analyzing the code base, jar and etc to identify the dependencies on JDK APIs and the third party JARs.

**jdeprscan** – helps in analyzing the code base for usage of any deprecated APIs

**jlink** – helps in creating a smaller runtime by combining the application's and the JDK's modules

**jmod** – helps in working with jmod files. jmod is a new format for packaging the modules. This format allows including native code, configuration files, and other data that do not fit into JAR files