Module in Java

- Java added this feature to collect Java packages and code into a single unit called module.
- The Java APIs are organized into methods, classes, packages, and - at the highest level - modules.
- In earlier versions of Java, there was no concept of module to create modular Java applications, that why size of application increased and difficult to move around. Even JDK itself was too heavy in size, in Java 8, rt.jar file size is around 64MB.
- To deal with situation, Java 9 restructured JDK into set of modules so that we can use only required module for our project.
- Java also allows us to create our own modules so that we can develop module based application.
- The module system includes various tools and options:
    - Includes various options to the Java tools javac, jlink and java where we can specify module paths that locates to the location of module.
    - Modular JAR file is introduced. This JAR contains module-info.class file in its root folder.
    - JMOD format is introduced, which is a packaging format similar to JAR except it can include native code and configuration files.
    - The JDK and JRE both are reconstructed to accommodate modules. It improves performance, security and maintainability.
    - Java defines a new URI scheme for naming modules, classes and resources.
- Module is a collection of Java programs or software.
- To describe a module, a Java file module-info.java is required. This file also known as module descriptor and defines the following:
    - Module name
    - What does it export
    - What does it require
- Module Declarations Example:
    - At the core of each module is the module declaration, a file with the name module-info.java that defines all properties of a module. As an example, here's the one for java.sql, the platform module that defines the JDBC API:

```
module java.sql
{
   requires transitive java.logging;
   requires transitive java.transaction.xa;
   requires transitive java.xml;

   exports java.sql;
   exports javax.sql;

   uses java.sql.Driver;
}
```

- It defines the module's name (java.sql), its dependencies on other modules (java.logging, java.transaction.xa, java.xml), the packages that make up its public API (java.sql and javax.sql), and which services it uses (java.sql.Driver).
- Generally speaking, a module declaration has the following basic form:

```
module $NAME {
   // for each dependency:
   requires $MODULE;

   // for each API package:
   exports $PACKAGE

   // for each package intended for reflection:
   opens $PACKAGE;
```

```
    // for each used service:
    uses $TYPE;

    // for each provided service:
    provides $TYPE with $CLASS;
}
```

- Create Java module:
  - Create a Directory Structure:
    - src
      - manish
        - abc
          - xyz

  - Create a file module-info.java, inside src folder, declare a module by using module identifier and provide module name same as the directory name that contains it (like manish).

- Create a module declarator

  ```
  module manish
  {

  }
  ```

  - Leave module body empty, if it does not has any module dependency. Save this file inside src/manish with module-info.java name.
- Java Source Code
  - Now, create a Java file to compile and execute module. In our example, we have a Hello.java file that contains the following code.

    ```
    package abc.xyz;
    class Hello
    {
        public static void main(String[] args)
        {
            System.out.println("Hello from the Java module");
        }
    }
    ```

  - Save this file inside src/manish/abc/xyz/ with Hello.java name.
- Compile Java Module
  - To compile the module use the following command.
    - javac -d mods --module-source-path src/ --module manish
- After compiling, it will create a new directory that contains the following structure.
  - mods
    - manish
      - abc
        - xyz
          - Hello.java
      - module-info.class
- Run Module
  - To run the compiled module, use the following command.
    java --module-path mods/ --module manish/abc.xyz.Hello