ANEXO: Módulos en Java

Antes de Java 9, el sistema de paquetes se utilizaba para organizar las clases en diferentes espacios de nombre. De esta forma, pueden coexistir en la plataforma las clases `java.util.Date` y `java.sql.Date`, porque están declaradas en paquetes distintos. Pero, esta organización del código presentaba algunas limitaciones en cuanto a modularidad:

- **Accesibilidad no controlada**: No había un mecanismo formal para controlar qué clases de un paquete eran accesibles fuera del paquete. Todas las clases públicas son accesibles fuera del paquete para cualquier clase de otro paquete, no puedes restringir a qué clases se quiere dar acceso. De manera que, o se da acceso a todas las clases (public) o no se da a ninguna (visibilidad a nivel de paquete). Los módulos van a solucionar este problema estableciendo un nuevo nivel de control de acceso. En un módulo se puede establecer qué paquetes son visibles a otros paquetes.
- **Gestión de dependencias**: Hasta Java 9, las dependencias con otras librerías las tenía que gestionar el programador configurando el ***classpath*** (lista de rutas o librerías que contienen todas las clases necesarias para la ejecución del proyecto). En el caso de que existiesen dependencias transitivas, por ejemplo, el proyecto A depende de la librería X, pero X a su vez depende de la librería Y, si el programador olvida incluir Y en el classpath, el proyecto falla en tiempo de ejecución. Los módulos nos van a permitir definir claramente cuáles son sus dependencias, esto es, qué módulos necesita. **Nota**: Existen herramientas específicas para la construcción de aplicaciones Java y de gestión de dependencias como Maven o Gradle que permiten solucionar el problema de las dependencias transitivas.
- **Tamaño de la librería estándar**: Hasta Java 8, todas las clases estaban agrupadas en un solo gran conjunto, incluso si una aplicación solo necesitaba una pequeña parte de la librería tenía que incluirlas todas. Ahora, la plataforma está dividida en módulos de manera que sólo se van a utilizar los que sean necesarios para la aplicación. Por ejemplo, si una aplicación no requiere de los módulos de interfaz gráfica no tiene que utilizarlos y se reduce significativamente el tamaño en tiempo de ejecución.

A partir de Java 9, se introdujo el **sistema de módulos** para resolver los problemas mencionados:

- **Un módulo** es una unidad de agrupación lógica de paquetes y recursos, que define explícitamente qué se expone a otros módulos y qué se mantiene privado y las dependencias con otros módulos.
- El archivo **module-info.java** es el descriptor del módulo. Contiene las siguientes declaraciones clave:
    - **`module nombre.del.modulo {}`**: Define el nombre del módulo.
    - **`exports paquete.nombre;`**: Especifica qué paquetes están disponibles para ser usados por otros módulos. NOTA: si el paquete contiene subpaquetes, éstos no estarán accesibles si no se exportan explícitamente.
    - **`requires [transitive] nombre.del.modulo;`**: Especifica de qué otros módulos depende el módulo actual. Si un módulo A especifica `requires transitive B` significa que el módulo que requiera el módulo A también depende y requerirá el módulo B.