

Moduláris programozás Java-ban

Jeszenszky Péter

Debreceni Egyetem, Informatikai Kar

jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2019. december 14.

JDK 9 (1)

- A fejlesztés javasolt ütemterve (2015. május 5.):
 - *Proposed schedule for JDK 9*
<http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-May/002172.html>
- Az eredetileg 2016. szeptember 22-re kitűzött kiadást többször is elhalasztották.
 - Lásd:
 - *Proposed schedule change for JDK 9* (2015. december 1.)
<http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-December/003149.html>
 - Az új célkitűzés: 2017. március 23.
 - *Proposed schedule change for JDK 9* (2016. szeptember 13.)
<http://mail.openjdk.java.net/pipermail/jdk9-dev/2016-September/004887.html>
 - Az új célkitűzés: 2017. július
 - *Proposed schedule change for JDK 9* (2017. május 30.)
<http://mail.openjdk.java.net/pipermail/jdk9-dev/2017-May/005864.html>
 - Az új célkitűzés: 2017. szeptember 21.

JDK 9 (2)

- A kiadás többszöri elhalasztása után végül 2017. szeptember 21-én jelent meg a JDK 9.
 - Lásd:
 - *JDK 9: General Availability*
<http://mail.openjdk.java.net/pipermail/announce/2017-September/000230.html>
 - *Oracle Announces Java SE 9 and Java EE 8*
<https://www.oracle.com/corporate/pressrelease/java-se-9-and-ee-8-092117.html>

JDK 9 (3)

- Újdonságok:
 - *What's New in Oracle JDK 9*
<https://docs.oracle.com/javase/9/whatsnew/>
 - *JDK 9 – Features*
<http://openjdk.java.net/projects/jdk9/>
- Áttérés:
 - *Java Platform, Standard Edition Oracle JDK 9 Migration Guide*
<https://docs.oracle.com/javase/9/migrate/>

Project Jigsaw (1)

- A projekt célja egy szabványos modulrendszer tervezése és implementálása volt a Java SE platformhoz, valamint ennek alkalmazása magára a platformra és a JDK-ra.
- Honlap: <http://openjdk.java.net/projects/jigsaw/>

Project Jigsaw (2)

- Elsődleges célok:
 - Megkönnyíteni a fejlesztők számára könyvtárak és nagy alkalmazások létrehozását és karbantartását.
 - Általában a Java SE platform implementációk, főleg a JDK biztonságának és karbantarthatóságának javítása.
 - Lehetővé tenni az alkalmazások teljesítményének növelését.
 - Lehetővé tenni a Java SE platform és a JDK kis számítási teljesítményű eszközökre való szabását.

Project Jigsaw (3)

- A projekt céljainak eléréséhez az alábbi két alapvető lehetőség biztosítása:
 - **Megbízható konfigurálás (*reliable configuration*)**: a törékeny, hibára hajlamos *class path* mechanizmus helyettesítése a programkomponensek számára egy olyan eszközzel, melynek révén explicit módon deklarálhatják az egymástól való függéseiket.
 - Lásd: JAR pokol (*JAR hell*)
 - **Erős egységbezárás (*strong encapsulation*)**: lehetővé tenni egy komponens számára annak deklarálását, hogy mely nyilvános típusai elérhetők más komponensek számára.

Project Jigsaw történet

- A projekt 2008 augusztusában indult.
- Eredetileg a JDK 7-hez szánták, majd előbb a JDK 8-ra, végül a JDK 9-re halasztották el a kivitelezést.
- 2014-ben indult a tervezés és megvalósítás a Java 9-hez.
- A Java platform modulrendszert meghatározó JSR 376 specifikációt 2014 decemberében hagyta jóvá a JCP végrehajtó bizottsága.
- A projekt fejlesztése nem a várt ütemben haladt, ezért a JDK 9 eredetileg 2016. szeptember 22-re kitűzött kiadást többször is elhalasztották.
- A projekttel kapcsolatos munka 2017 júliusában fejeződött be, az eredmény a 2017. szeptember 21-én megjelent JDK 9 részeként elérhető általános használatra.
- Lásd:
 - *Project Jigsaw: Context & History* <http://openjdk.java.net/projects/jigsaw/history>
 - *Project Jigsaw – Development history* <http://openjdk.java.net/projects/jigsaw/>

Project Jigsaw specifikációk

- JEP-ek:
 - *JEP 200: The Modular JDK* <http://openjdk.java.net/jeps/200>
 - *JEP 201: Modular Source Code* <http://openjdk.java.net/jeps/201>
 - *JEP 220: Modular Run-Time Images* <http://openjdk.java.net/jeps/220>
 - *JEP 260: Encapsulate Most Internal APIs* <http://openjdk.java.net/jeps/260>
 - *JEP 261: Module System* <http://openjdk.java.net/jeps/261>
 - *JEP 282: jlink: The Java Linker* <http://openjdk.java.net/jeps/282>
- JSR:
 - *JSR 376: Java Platform Module System*
 - <http://openjdk.java.net/projects/jigsaw/spec/>
 - <https://jcp.org/en/jsr/detail?id=376>

Modul fogalma (1)

- Elég szorosan összetartozó csomagok egy modulba csoportosíthatók.
- Egy modul kód és adatok egy névvel rendelkező, önleíró gyűjteménye. A kódja csomagokba van szervezve, melyek típusokat (osztályokat és interfészeket) tartalmaznak. Adatként erőforrásokat és más típusú statikus információkat tartalmaz.
 - Lásd: Mark Reinhold. *The State of the Module System*.
<http://openjdk.java.net/projects/jigsaw/spec/sotms/>

Modul fogalma (2)

- Egy modul exportáltként jelölheti meg néhány vagy minden csomagját, ami azt jelenti, hogy a típusaik hozzáférhetők a modulon kívüli kódból.
 - Egy felső szintű típus akkor, és csak akkor hozzáférhető a deklaráló modulon kívülről, ha nyilvánosnak deklarált és egy exportált csomag tagja.
 - Ha egy csomagot nem exportál egy modul, akkor csak a modulon belüli kód számára hozzáférhetők a típusai.
- Ha egy modul egy másik modul által exportált csomagokhoz szeretne hozzáférni, akkor explicit módon függenie kell tőle.

Modul deklarációk (1)

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith. *The Java Language Specification – Java SE 11 Edition*. 21 August 2018.
<https://docs.oracle.com/javase/specs/jls/se11/html/>
- Lásd a következő részt: 7.7. *Module Declarations*
<https://docs.oracle.com/javase/specs/jls/se11/html/jls-7.html#jls-7.7>

Modul deklarációk (2)

- Egy modul deklaráció egy nevesített modult határoz meg.
- Egy olyan modulnevet vezet be, mely modulok közötti kapcsolatok kifejezésére használható más modulok deklarációiban.
- Egy modulnév egy vagy több Java azonosítóból áll, melyeket `.` karakterek választanak el.
- Egy modul deklaráció direktívái határozzák meg a modul függését más moduloktól (`requires`), a más modulok számára elérhetővé tett csomagokat (`export` és `opens`), az általa felhasznált (`uses`) és nyújtott (`provides`) szolgáltatásokat.

Modul deklarációk (3)

- A modul deklarációt megelőzhetik import deklarációk.
 - Lehetővé teszik, hogy a modul deklarációban a modul vagy más modulok csomagjainak típusaira és a típusok statikus tagjaira az egyszerű nevükkel hivatkozassunk.

Modul deklarációk (4)

- Példa (OpenJDK 11):

```
module java.prefs {  
    requires java.xml;  
  
    exports java.util.prefs;  
  
    uses java.util.prefs.PreferencesFactory;  
}
```

Modul deklarációk (5)

- Példa (OpenJDK 11):

```
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;  
}
```

Modul deklarációk (6)

- Példa (OpenJDK 11):

```
module java.logging {  
    exports java.util.logging;  
  
    provides jdk.internal.logger.DefaultLoggerFinder with  
        sun.util.logging.internal.LoggingProviderImpl;  
}
```

Modul deklarációk (7)

- **requires:**

- A `requires` direktíva egy olyan modult határoz meg, melytől az aktuális modul függ.
- A `java.base` modul kivételével minden modul implicit módon függ a `java.base` modultól.
- A `requires` kulcsszót követheti a `transitive` módosító.
 - Az aktuális modultól függő modulok implicit módon függenek a `requires transitive` direktívával meghatározott modultól.
- A `requires` kulcsszót követheti a `static` módosító.
 - Ez azt jelenti, hogy a függőség fordítási időben kötelező, futási időben opcionális.

Modul deklarációk (8)

- **exports:**

- Az `export` direktíva egy, az aktuális modul által exportált csomag nevét határozza meg.
- Más modulok számára a csomag `public` és `protected` típusai, valamint azok `public` és `protected` tagjai hozzáférhetők fordítási és futási időben.
 - Reflektív hozzáférés is engedélyezett ezekhez a típusokhoz és tagjaikhoz.

Modul deklarációk (9)

- **uses:**
 - A `uses` direktíva egy olyan szolgáltatást határoz meg, melyhez az aktuális modul szolgáltatókat találhat a `java.util.ServiceLoader` osztály révén.
- **provides:**
 - A `provides` direktíva egy olyan szolgáltatást határoz meg, melyhez a `with` záradék sorol fel egy vagy több szolgáltatót a `java.util.ServiceLoader` osztályhoz.

java.util.ServiceLoader (1)

- Szolgáltatások implementációinak betöltésére szolgáló lehetőség.
- Egy szolgáltatás egy olyan közismert interfész vagy (absztrakt) osztály, melyhez 0, 1 vagy több szolgáltató létezik.
- Egy szolgáltató (*service provider*, vagy röviden *provider*) egy olyan osztály, mely a közismert interfészt implementálja vagy alosztálya a közismert (absztrakt) osztálynak.
- Az alkalmazások kódja csak a szolgáltatásra hivatkozik, nem pedig a szolgáltatókra.
- Lásd: `java.util.ServiceLoader`
<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ServiceLoader.html>

java.util.ServiceLoader (2)

- Példa:

```
import java.sql.Driver;  
import java.util.ServiceLoader;  
  
ServiceLoader<Driver> loader = ServiceLoader.load(Driver.class);  
loader.forEach(driver ->  
    System.out.println(driver.getClass().getName()));  
// org.hsqldb.jdbc.JDBCDriver  
// org.sqlite.JDBC  
// org.h2.Driver  
// org.postgresql.Driver
```

java.util.ServiceLoader (3)

- Példa:

```
import java.util.ServiceLoader;
import java.security.Provider;

ServiceLoader<Provider> loader = ServiceLoader.load(Provider.class);
loader.stream()
    .map(ServiceLoader.Provider::type)
    .map(Class::getName)
    .forEach(System.out::println);
// class sun.security.ec.SunEC
// class com.sun.security.sasl.gsskerb.JdkSASL
// class org.jcp.xml.dsig.internal.dom.XMLDSigRI
// class sun.security.smartcardio.SunPCSC
// class sun.security.jgss.SunProvider
// class sun.security.pkcs11.SunPKCS11
// class com.sun.security.sasl.Provider
// class sun.security.provider.certpath.ldap.JdkLDAP
```

java.util.ServiceLoader (4)

- Egy modulban fejlesztett szolgáltatót a modul deklaráció egy `provides` direktívájában kell megadni.
 - A `provides` direktíva a szolgáltatást és a szolgáltatót is megadja.
- Egy JAR állományba csomagolt szolgáltatót egy szolgáltató konfigurációs állomány azonosít a `META-INF/services` könyvtárban, melynek neve a szolgáltatás teljesen minősített neve.
 - Az állomány a szolgáltatók teljesen minősített nevét tartalmazza, soronként egyet.

Olvashatóság (1)

- Ha az A modul függ a B modultól, akkor azt mondjuk, hogy A **olvassa** (*reads*) B -t, és hogy B **olvasható** (*readable*) A által.
 - Ilyenkor az A modul hozzáfér a B modul által exportált csomagok nyilvános felső szintű típusaihoz.
- Definíció szerint minden modul olvassa önmagát.
 - Az olvashatóság reflexív reláció.

Olvashatóság (2)

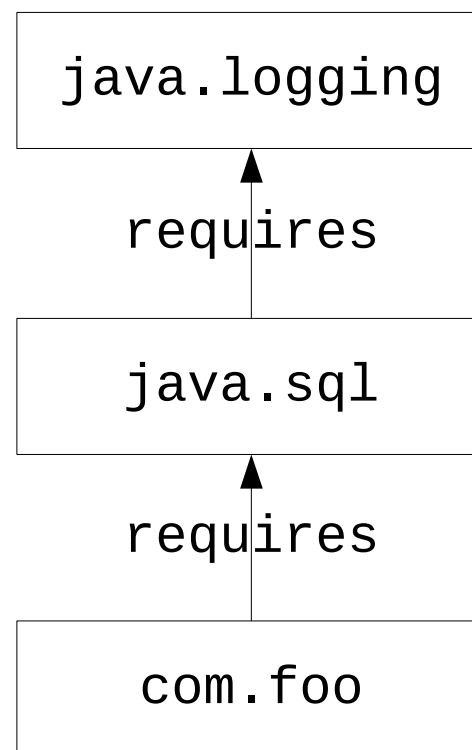
- Az olvashatóság nem tranzitív reláció!
 - Példa: az *A* modul nem olvassa a *C* modult!

```
module A {  
    requires B;  
}  
  
module B {  
    requires C;  
}
```

Olvashatóság (3)

- Az olvashatóság nem tranzitív reláció!
 - Példa:

```
module java.logging {  
  exports java.util.logging;  
}  
  
module java.sql {  
  requires java.logging;  
  requires javax.transaction.xa;  
  requires java.xml;  
  exports java.sql;  
  exports javax.sql;  
}  
  
module com.foo {  
  requires java.sql;  
  exports com.foo;  
}
```



Olvashatóság (4)

- Az olvashatóság nem tranzitív reláció!
 - Példa (folytatás):
 - A `com.foo` modulban fordítási hibát eredményez az alábbi kódrész megjelölt metódushívása, mivel a modul nem olvassa a `java.logging` modult.

```
import java.sql.*;
import java.util.Properties;

String url;
Properties info;

Driver driver = DriverManager.getDriver(url);
Connection conn = driver.connect(url, info);
driver.getParentLogger().info("Connection acquired");
```

Olvashatóság (5)

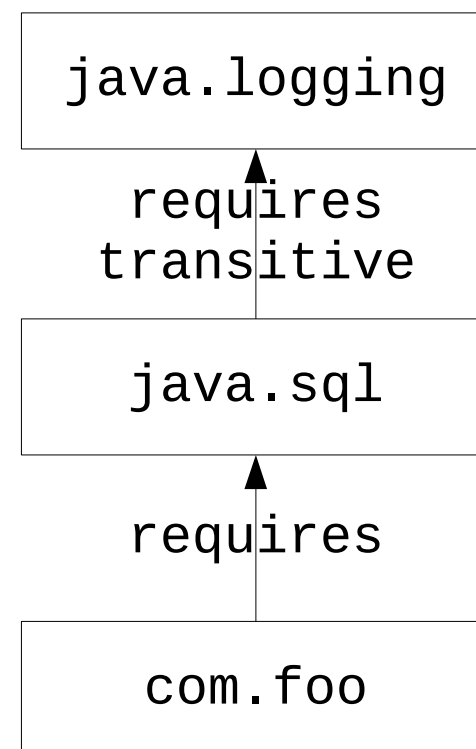
- Az olvashatóság nem tranzitív reláció!
 - Példa (folytatás):
 - A probléma egy lehetséges megoldása a `com.foo` modult függővé tenni a `java.logging` modultól is:

```
module com.foo {  
    requires java.logging;  
    requires java.sql;  
    exports com.foo;  
}
```

Olvashatóság (6)

- Az olvashatóság nem tranzitív reláció!
 - Példa (folytatás): a probléma elegáns megoldása:

```
module java.logging {  
  exports java.util.logging;  
}  
  
module java.sql {  
  requires transitive java.logging;  
  requires transitive javax.transaction.xa;  
  requires transitive java.xml;  
  exports java.sql;  
  exports javax.sql;  
}  
  
module com.foo {  
  requires java.sql;  
  exports com.foo;  
}
```



Olvashatóság (6)

- **Implicit olvashatóság (*implied readability*):**
 - A `requires transitive` direktíva azt fejezi ki, hogy aktuális modultól függő modulok által is olvasható a függőség.
 - A példában tehát a `com.foo` modul is olvassa `java.logging` modult.

Olvashatóság (7)

- Egyetlen modul sem olvashat két vagy több olyan modult, melyek ugyanazt a csomagot exportálják.
 - Ennek megsértése fordítási hibát eredményez.

Névtelen modulok

- Egy speciális, úgynevezett **névtelen modulhoz** (***unnamed module***) tartozik minden olyan típus, mely nem tartozik valamely nevesített modulhoz.
- A névtelen modul fogalma hasonló a névtelen csomag fogalmához.
- Azért szükséges, mert a Java SE 9 előtt nem volt lehetőség modulok deklarálására.
- A Java SE platform egy implementációja legalább egy névtelen modul használatát kell, hogy támogassa.
- Egy névtelen modul minden más modult olvas.
- Egy névtelen modul minden csomagját exportálja.

Modul útvonalak

- A `java`, `javac`, `jlink` és más parancssori eszközök számára a modulok megadása modul útvonalakban.
- A modul útvonalak olyan listák, melyek minden eleme az alábbiak valamelyike vagy az alábbiakat tartalmazó könyvtár:
 - A gyökeri könyvtárban egy `module-info.class` állományt tartalmazó JAR állomány (úgynevezett moduláris JAR állomány).
 - Egy JMOD állomány.
 - Egy modult tartalmazó könyvtár.
- Hiba, ha egy modul útvonal elemben (például egy könyvtárban) több azonos nevű modul szerepel.

JMOD állományok

- A JAR állományok lehetőségein túlmutató új formátum, mely támogatja például natív könyvtárak becsomagolását is.
- Kizárólag a fordítási és szerkesztési időben használhatók, végrehajtási időben nem!
 - Az utóbbihoz szükséges lenne például natív könyvtárak menet közbeni kitömörítésére és szerkesztésére.
- JMOD állományokba vannak csomagolva a JDK moduljai is.
 - Lásd a `$JAVA_HOME/jmods` könyvtárat.
- A JAR állományformátumhoz hasonlóan a JMOD állományformátum is a ZIP állományformátumon alapul.
- `jmod`: új parancssori eszköz JMOD állományok létrehozásához, manipulálásához és vizsgálatához.
 - Lásd: *JDK 11 Documentation – Tools Reference* – `jmod`
<https://docs.oracle.com/en/java/javase/11/tools/jmod.html>

Szerkesztés

- A szerkesztés a fordítás és a futtatás közötti opcionális fázis, melynek során modulok egy halmazából és függőségeikből egy egyéni futásidejű lemezkép kerül létrehozása.
 - Lásd: *JEP 282: jlink: The Java Linker*
<http://openjdk.java.net/jeps/282>
- `jlink`: a szerkesztésre szolgáló új parancssori eszköz.
 - Lásd: *JDK 11 Documentation – Tools Reference – jlink*
<https://docs.oracle.com/en/java/javase/11/tools/jlink.html>

Moduláris JDK (1)

- Lásd: *JEP 200: The Modular JDK* <http://openjdk.java.net/jeps/200>
- A JSR 376-ban meghatározott és a JEP 261-ben implementált Java platform modulrendszer használata a JDK modularizációjához.
- A JDK olyan modulokra történő felosztása, melyek fordítási időben, összeállítási időben és futási időben különféle konfigurációkká kombinálhatók össze, ideértve többek között:
 - A teljes Java SE platformnak, a teljes JRE-nek és a teljes JDK-nak megfelelő konfigurációkat.
 - Olyan egyéni konfigurációkat, melyek csak bizonyos meghatározott modulokat tartalmaznak esetlegesen külső könyvtár és alkalmazás modulokkal kibővítve, valamint mindezen modulokhoz tranzitíven szükséges modulokat.

Moduláris JDK (2)

- Kétféle modul megkülönböztetése:
 - **Szabványos modulok:**
 - Olyan modulok, melyek specifikációit a JCP szabályozza.
 - A szabványos modulok neve a `java.` karakterlánccal kezdődik.
 - **Nem szabványos modulok:**
 - A JDK részét alkotó egyéb modulok.
 - A nem szabványos modulok neve a `jdk.` karakterlánccal kezdődik.

Moduláris JDK (3)

- A JDK moduláris szerkezete irányított gráffal jeleníthető meg.
 - A gráfban a csúcsok a modulokat ábrázolják, az élek pedig a modulok közötti függéseket.
- Lásd:
<https://docs.oracle.com/en/java/javase/11/docs/api/java.se/module-graph.png>

Moduláris JDK (4)

- A legalsó `java.base` modul olyan alapvető osztályokat tartalmaz, mint például a `java.lang.Object` és a `java.lang.String`.
 - Ez a modul nem függ egyetlen modultól sem és minden modul függ tőle.
- Az ábra tetején a `java.se` modul látható, mely a Java SE platformot alkotó modulokat gyűjti egybe.
 - A modul az aggregátor modulok példája, melyek más modulok tartalmát gyűjtik össze implicit olvashatóságot biztosítva hozzájuk, de nem adva hozzá saját tartalmat.

Moduláris JDK forráskód

- Lásd: *JEP 201: Modular Source Code*
<http://openjdk.java.net/jeps/201>
- A JDK forráskódjának modulokba történő átszervezése (új könyvtárszerkezet kialakítása a JDK forráskódjához).

Moduláris futásidejű lemezképek (1)

- Lásd:
 - *JEP 220: Modular Run-Time Images* <http://openjdk.java.net/jeps/220>
 - Mark Reinhold. *Project Jigsaw: Modular run-time images*.
<https://mreinhold.org/blog/jigsaw-modular-images>
- A JDK és a JRE futásidejű lemezképek (*run-time image*) szerkezetének átalakítása a modulok alkalmazásához, a teljesítmény a biztonság és a karbantarthatóság javításához.
- Egy olyan új URI-séma (*j r t*) meghatározása a futásidejű lemezképekben tárolt modulok, osztályok és erőforrások megnevezéséhez a lemezkép belső szerkezetének vagy formátumának felfedése nélkül.

Moduláris futásidejű lemezképek (2)

- A JDK 9 előtt kétféle futásidejű lemezkép megkülönböztetése: JRE és JDK.
 - A JDK lemezkép a JRE lemezkép egy másolatát tartalmazza a `jre/` alkönyvtárban.
- Az új lemezkép szerkezet megszünteti ezt a megkülönböztetést, a JRE és JDK lemezképe azonos szerkezetű.
 - A JDK lemezkép egyszerűen egy olyan futásidejű lemezkép, mely történetesen a teljes fejlesztői eszközkészletet tartalmazza.
- Egy moduláris lemezkép JAR állományok helyett inkább modulokból áll.

Moduláris futásidejű lemezképek (3)

- Az alábbi lehetőségek megszüntetése:
 - *Java Endorsed Standards Override Mechanism*
<https://docs.oracle.com/javase/8/docs/technotes/guides/standards/>
 - Azt tette lehetővé, hogy a JCP-n kívül karbantartott szabványok újabb verzióinak vagy a Java SE platform részét képező, de attól függetlenül fejlődő különálló API-k implementációit telepítsük egy futásidejű lemezképbe.
 - *The Extension Mechanism for Support of Optional Packages*
<https://docs.oracle.com/javase/8/docs/technotes/guides/extensions/>

Moduláris futásidejű lemezképek (4)

- Az `rt.jar` és `tools.jar` állományok eltávolítása:
 - A korábban a `lib/rt.jar`, `lib/tools.jar` és egyéb belső JAR állományokban tárolt `.class` és erőforrás állományok hatékonyabb módon kerülnek tárolásra implementáció specifikus állományokban a `lib/` könyvtárban.
 - A `jimage` parancssori eszköz szolgál ezeknek a JDK implementáció specifikus konténer állományoknak a vizsgálatára.

Moduláris futásidejű lemezképek (5)

- Példa:

```
// JDK 8:  
System.out.println(ClassLoader  
    .getSystemResource("java/lang/Object.class"));  
// jar:file:/usr/lib/jvm/java-8-oracle/jre/lib/rt.jar!  
// java/lang/Object.class
```

```
// JDK 9 vagy későbbi verziók:  
System.out.println(ClassLoader  
    .getSystemResource("java/lang/Object.class"));  
// jrt:/java.base/java/lang/Class.class
```

Inkubáló modulok (1)

- Arra szolgálnak, hogy nem végleges, inkubáló (*incubating*) API-kat adjanak a fejlesztők kezébe.
- Egy inkubáló API egy olyan API, melynél kívánatos a szabványosítás vagy véglegesítés elhalasztása a következő kiadásra az API-val kapcsolatos további tapasztalatok szerzése céljából.
- Korlátozott egy API inkubációs élettartama: az API szabványosításra vagy véglegesítésre kerül a következő kiadásban, vagy pedig eltávolításra kerül.
- Az inkubáló modulok neve a `jdk.incubator.` előtaggal kezdődik.
- Lásd:
 - *JEP 11: Incubator Modules* <http://openjdk.java.net/jeps/11>

Inkubáló modulok (2)

- Példa: `jdk.incubator.httpclient`
 - Lásd:
 - JEP 110: HTTP/2 Client (Incubator) <https://openjdk.java.net/jeps/110>
 - *Module `jdk.incubator.httpclient`*
<https://docs.oracle.com/javase/9/docs/api/jdk.incubator.httpclient-summary.html>
 - Az API végül a JDK 11-ben került szabványosításra.
 - Lásd: *JEP 321: HTTP Client (Standard)* <https://openjdk.java.net/jeps/321>
 - A szabványosított API-t a `java.net.http` modul szolgáltatja a `java.net.http` csomagban.
 - Lásd: *Module `java.net.http`*
<https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/module-summary.html>

Eltávolított modulok

- A Java EE és CORBA technológiákat tartalmazó modulok elavultnak és a jövőben eltávolításra kerülőnek lettek jelölve a Java SE 9-ben:
 - `java.xml.bind` (JAXB), `java.xml.ws` (JAX-WS), `java.activation` (JAF), `java.transaction` (JTA), ...
- Ezek a modulok fordításnál és végrehajtásnál nem kerülnek feloldásra a JDK 9-ben!
 - A modulok feloldásához a `javac` és `java` parancsokhoz meg kell adni az `--add-modules` parancssori opciót.
- Ezek a modulok a Java SE 11-ben és a JDK 11-ben kerültek ténylegesen eltávolításra.
- Lásd: *JEP 320: Remove the Java EE and CORBA Modules*
<http://openjdk.java.net/jeps/320>

Moduláris alkalmazások

- Példák:
 - *Apache Derby* (licenc: *Apache License 2.0*)
<https://db.apache.org/derby/>
 - *SLF4J* (licenc: *MIT License*) <https://www.slf4j.org/>
 - *Logback* (licenc: *Eclipse Public License v1.0/GNU LGPL v2.1*) <https://logback.qos.ch/>
 - *OpenJFX* (licenc: *GNU GPL v2 + Classpath Exception*)
<http://openjdk.java.net/projects/openjfx/> <https://openjfx.io/>
 - ...

Moduláris „Helló, világ!” program (1)

- Könyvtárszerkezet és forráskód:

```
src/  
  org.hello/  
    module-info.java  
  org/  
    hello/  
      Main.java
```

```
// module-info.java:  
module org.hello {  
  exports org.hello;  
}
```

Moduláris „Helló, világ!” program (2)

- Fordítás:

```
$ javac -d mods/org.hello \ célkönyvtár  
src/org.hello/module-info.java \  
src/org.hello/org/hello/Main.java
```

Moduláris „Helló, világ!” program (3)

- Futtatás:

```
$ java --module-path mods \ a modulokat tartalmazó könyvtár(ak)  
  --module org.hello/org.hello.Main
```

Moduláris „Helló, világ!” program (4)

- JMOD készítés:

```
$ mkdir jmods  
$ jmod create --class-path mods/org.hello \  
  jmods/org.hello.jmod
```

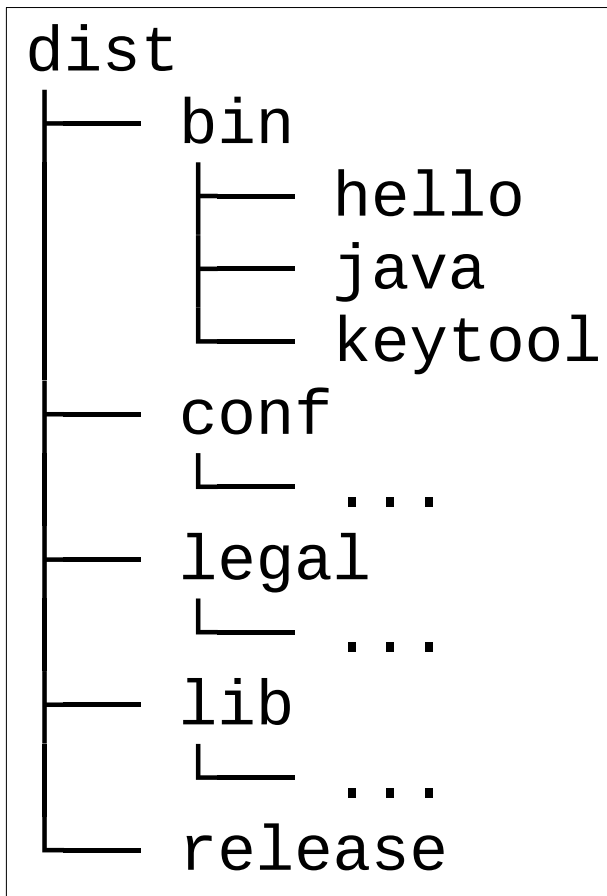
Moduláris „Helló, világ!” program (5)

- Szerkesztés:

```
$ jlink --module-path $JAVA_HOME/jmods:jmods \  
  --add-modules org.hello \  
  --output dist \  
  --strip-debug \  
  --compress 2 \  
  --no-header-files \  
  --no-man-pages \  
  --launcher hello=org.hello/org.hello.Main
```

Moduláris „Helló, világ!” program (6)

- A szerkesztés eredményeként kapott `dist/` könyvtár szerkezete (a teljes méret mindössze kb. 30 MB):



Integrált fejlesztői környezetek

- **Eclipse IDE:** *Configure Eclipse for Java 9*
https://wiki.eclipse.org/Configure_Eclipse_for_Java_9
- **IntelliJ IDEA:** *Getting Started with Java 9 Module System*
<https://www.jetbrains.com/help/idea/getting-started-with-java-9-module-system.html>
- **NetBeans IDE:** Java 9 támogatás a 9.0 verziótól
 - Lásd: *Apache NetBeans (incubating) 9.0 Features – Supporting the Jigsaw Module System*
https://netbeans.apache.org/download/nb90/#_supporting_the_jigsaw_module_system

Fejlesztőeszközök

- *Apache Maven JMod Plugin*
<https://maven.apache.org/plugins/maven-jmod-plugin/>
- *Apache Maven JLink Plugin*
<https://maven.apache.org/plugins/maven-jlink-plugin/>

További ajánlott irodalom

- Cay S. Horstmann. *Core Java SE 9 for the Impatient*. 2nd Edition. Addison-Wesley Professional, 2017.
<http://www.informit.com/store/core-java-se-9-for-the-impatient-9780134694726>
- Kishori Sharan. *Java 9 Revealed: For Early Adoption and Migration*. Apress, 2017.
<https://www.apress.com/br/book/9781484225912>
- Koushik Kothagal. *Modular Programming in Java 9*. Packt Publishing, 2017.
<https://www.packtpub.com/application-development/modular-programming-java-9>
- Sander Mak, Paul Bakker. *Java 9 Modularity: Patterns and Practices for Developing Maintainable Applications*. O'Reilly Media, 2017. <http://shop.oreilly.com/product/0636920049494.do>