Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Java programok a filerendszerben

Bevezetés

- A Java-ban a programokat alkotó osztályok csoportosítása két szinten történik:
  - **logikai szint**: az osztályok és interfészek – rendszerint funkció alapján – **hierarchikus csomagok**ba történő szervezése,
  - **fizikai szint**: az osztályokat illetve interfészeket definiáló **forráskódot tartalmazó file-ok (forrásfile), valamint a byte kódot tartalmazó file-ok (classfile) tárolásá**nak szervezése.
    - A forrásfile-ok neve **.java**-ra, a classfile-ok neve **.class**-ra végződik.

A file-ok elhelyezkedése

- Mind a forrásfile-okat, mind a classfile-okat a **csomaghierarchiának megfelelő könyvtárszerkezet**ben kell elhelyezni.
- A könyvtárszerkezet **gyökerének felel meg a névtelen csomag**, a többi csomaghoz tartozó file-ok a **megfelelő relatív útvonalon** helyezkednek el.
- Ennek az elrendezésnek a célja az, hogy mind a fordító, mind a futtató környezet megtalálja a szükséges file-okat.
- Például:

Csomagnév	**Útvonal**
(névtelen)	(gyökér)
program	(gyökér)/program
org.acme.tool	(gyökér)/org/acme/tool

Forrásfile-ok tartalma és elnevezése

- Egy file-ban **tetszőleges számú osztályt, illetve interfészt definiálhatunk**.
- **Egy file**-ban definiált osztályok és interfészek **egy csomag**ban vannak: vagy a névtelen csomagban, vagy a file elején megadott package deklarációnak megfelelő csomagban.
- Egy file-ban **legfeljebb egy publikus** osztályt, vagy interfészt definiálhatunk.
- Ha a file-ban van publikus osztály vagy interfész, a file neve ennek nevével kell megegyezzen, egyébként a file neve tetszőleges.
  - A publikus `Osztály` nevű osztály definíciója tehát csak az `Osztály.java` nevű file-ban lehet.

Classfile-ok tartalma és elnevezése I.

- Egy classfile-ban **csak egyetlen osztály, vagy interfész byte kódja található**. Ezt azt jelenti, hogy a beágyazott osztályok is külön file-ba kerülnek!
- **Nem beágyazott osztály** esetén a classfile neve az osztály nevével egyezik meg.
- **Beágyazott osztály** esetén a classfile neve a beágyazó osztály(ok) nevéből (neveiből), és a beágyazott osztály nevéből áll, a neveket $ jel választja el egymástól.
- A **névtelen beágyazott osztály**ok classfile-jának neve a legkülső beágyazó osztály nevéből, egy $ jelből, és egy sorszámból áll. A legkülső osztályon belüli névtelen belső osztályok folyamatosan számozottak.
- A **lokális osztályok** classfile-jainak nevei a legkülső beágyazó osztály nevéből, a 1 karaktersorozatból, és a lokális osztály nevéből állnak.

Classfile-ok tartalma és elnevezése II.

```
public class Enclosing {              // Nem beágyazott: Enclosing.class

   public Enclosing () {
      Class anonymous = new Class () {  // Névtelen: Enclosing$1.class
         public void m () {}
      };
      class Local {}                 // Lokális: Enclosing$1$Local.class
   }
   public class Inner {}                 // Belső: Enclosing$Inner.class
   public static class Nested {} // Beágyazott: Enclosing$Nested.class
}

class Class {                             // Nem beágyazott: Class.class

   public Class () {
      Class inner = new Class () {          // Névtelen: Class$1.class
         public void m () {}
      };
   }
}
```

Java nevek és a file-ok megfeleltetése

- A fenti szabályok betartásával a Java nevek és a file-ok megfeleltethetők egymásnak.
- Egy osztály **csomagnevéből** a file-t tartalmazó **könyvtár** meghatározható, mind a forrásfile-ok, mind a classfile-ok esetén.
- A keresett **file nevé**t a két esetben eltérően kell meghatározni:
  - **forrásfile**: a keresett file neve az osztály nevével egyezik meg,
  - **classfile**: a nem beágyazott, illetve a beágyazott, de nem névtelen és nem lokális osztályok esetén a file neve egyértelműen meghatározható, a névtelen és a lokális osztályok esetén viszont a legkülső beágyazó osztály betöltése után derül ki a file-név.
- A Java szabvány lehetővé teszi a byte kódok classfile-ok helyett **adatbázis**ban való tárolását.

A classpath szerepe

- A fenti szabályok alapján egy tetszőleges minősített névből elő lehet állítani egy file-nevet, és egy **relatív útvonal**at.
- Ahhoz azonban, hogy a kérdéses file-t meg lehessen találni, **abszolút útvonal**ra van szükség.
- A relatív útvonalakat a Java rendszer a **classpath**-nak nevezett útvonallista elemeivel egészíti ki abszolút útvonallá.
- Amikor egy relatív útvonalat ki kell egészíteni abszolút útvonallá, a rendszer **sorra veszi** a classpath elemeit, amíg vagy meg nem találja a kérdéses file-t, vagy a classpath végére nem ér.
  - Ebből következik, hogy **a classpath elemeinek a sorrendje nem közömbös**.

A classpath megadása

- A classpath útvonalak listája, melyeket az adott platformra jellemző szeparátor választ el egymástól, ez Windows-ban a pontosvessző (;), UNIX rendszerekben a kettőspont (:).
- A classpath a Java rendszer programjainak kétféleképpen is megadható:
  - **Parancssori paraméter**ként, általában a -classpath opcióval, ez a jelenleg elfogadott, biztonságos megoldás.
  - A **CLASSPATH környezeti változó** értékeként. Régebben ez volt az elfogadott módszer, de több Java program egyidejű fordítása / használata esetén veszélyessé válik.

Java ARchive

- Egy Java programban több ezer osztály is lehet, kényelmetlen lenne ezeket telepítéskor, illetve egy könyvtár használatakor külön kezelni.
- A file-ok összefoghatók egy archívumba, ez a **JAR (Java ARchive)** file. A JAR file belsejében a file-ok hierarchikus elhelyezkedése megmarad, ez szükséges a file-ok megtalálásához.
- A JAR lényegében egy **ZIP archívum**, azzal a lényeges különbséggel, hogy "élő", vagyis a program **közvetlenül az archívumból futtatható**, annak kicsomagolása nélkül.
- A JAR file-ban nem csak classfile-ok, hanem a program futásához szükséges egyéb file-ok is elhelyezhetők (képek, paraméterek, stb.).
- **Speciális file-ok**ban utasításokat helyezhetünk el a futtató környezet számára.

A JAR file-ok használata I.

- A JAR file-okat a **jar** parancs segítségével hozhatjuk létre.
- Egy Java program futtatásakor három módja van a JAR file-ok használatának:
  - **A classpath elemei között felsorolhatunk JAR file-okat** is. Ekkor a keresett osztályokat a JAR-on belül is keresi a rendszer, a megfelelő relatív útvonalon.
  - A **Java Extension Mechanism** használata (lásd később).
  - A java futtató program **-jar opció**ja segítségével a megadott jar file fő osztályát automatikusan futtathatjuk. A fő osztályt a JAR-ban egy speciális file-ban kell megadni.

A JAR file-ok használata II.

- Az alábbi osztály classfile-ját helyezzük el egy teszt.jar nevű JAR file-ban:

```
public class Teszt2 {
   public Teszt2 () { System.out.println ("Ez a JAR-ban van."); }
}
```

- Ezt pedig ne tegyük JAR-ba

```
public class Teszt2 {
   public Teszt2 () { System.out.println ("Ez nem a JAR-ban van."); }
}
```

- Futtassuk az alábbi programot két módon:

```
public class Teszt {
   public static void main (String[] args) {
      new Teszt2 ();
   }
}
```

A JAR file-ok használata III.

```
> java –classpath teszt.jar;. Teszt
Ez a JAR-ban van.

> java –classpath .;teszt.jar Teszt
Ez nem a JAR-ban van.
```

- Az első esetben a JAR file-ban lévő `Teszt2` osztály töltődött be, míg a második esetben a JAR file-on kívül lévő.
- Ezzel a módszerrel egy JAR file egy osztályát egy futtatás erejéig lecserélhetjük.
- Biztonsági okokból ez megakadályozható.

Speciális file-ok egy JAR archívumban

- A speciális file-ok egy **`META-INF`** nevű könyvtárban helyezkednek el.
- A Java rendszer az alábbi file-okat ismeri fel és használja:
  - **`MANIFEST.MF`**: A JAR-ról és a benne lévő file-okról tartalmaz általános adatokat, illetve utasításokat a futtató környezet számára.
  - **`INDEX.LIST`**: Az osztálybetöltés felgyorsítására szolgál.
  - **`xxx.SF`** és **`xxx.DSA`**: Digitálisan aláírt JAR file-ok használják.
- A fenti file-ok név-érték párokat tartalmaznak, a nevet egy kettőspont választja el az értéktől, a párokat pedig újsor választja el egymástól.

A manifest file I.

- A manifest file elején az **egész JAR archívumra vonatkozó**, később az **egyes file-okra vonatkozó** bejegyzések állnak.
- A teljes JAR-ra vonatkozó bejegyzések közül a legfontosabbak:
  - **Manifest-Version**: a manifest file verziószáma,
  - **Created-By**: annak a Java implementációnak a gyártója és verziószáma, amely alatt a JAR file létrejött,
  - **Signature-Version**: a digitális aláírások verziószáma,
  - **Class-Path**: a JAR-ban lévő program vagy könyvtár futtatásához szükséges további könyvtárak relatív útvonalai,
  - **Main-Class**: a -jar opció használatakor a futtató környezet ezt az osztályt indítja el.

A manifest file II.

- Az egyes file-okra vonatkozó bejegyzések közül a fontosabbak:
  - **Content-Type**: a file MIME típusa,
  - **Java-Bean**: ha true, a file egy Java Beans objektum,
  - **Sealed**: ha true, a file-t tartalmazó csomag "le van pecsételve" (ez az opció a teljes JAR file-ra is alkalmazható),
  - ***x*-Digest**: a file *x* típusú lenyomata (*x* jelenleg SHA-1, vagy MD5),
  - **Magic**: a lenyomat kiszámításához szükséges plusz információ.
- A Sealed opció segítségével **a JAR file konzisztenciáját biztosíthatjuk**. Ha egy JAR file-ban egy csomag sealed, csak az adott JAR file-ból tölthetők be a csomag osztályai. Ha a csomag egy osztálya nem a JAR-ból töltődik be, a futtató környezet `SecurityException`-t dob.

Digitálisan aláírt JAR-ok I.

- A JAR file-okat biztonsági okokból digitálisan aláírhatjuk, biztosítva tartalmuk eredetiségét.
- Az aláíráshoz szükséges kulcsokat a **keytool** programmal hozhatjuk létre, a JAR file aláírására és későbbi ellenőrzésére a **jarsigner** program szolgál.
- Egy JAR file-t egyszerre **többen is aláírhatnak**.
- A JAR-ban speciális file-ok tárolják az aláírással kapcsolatos információkat.
  - a `MANIFEST.MF` file-ban az egyes **bejegyzések lenyomata** (digest) található,
  - az `xxx.SF` és `xxx.DSA` file-ok párban vannak, és egy aláíróhoz tartozó **digitális aláírás**t és **kísérőadatok**at tartalmaznak.

Digitálisan aláírt JAR-ok II.

- Egyszerű kulcsgenerálás

```
> keytool –alias littejoe -genkey
```

- Létrehozzuk a JAR-t

```
> jar –cvf myJar.jar –C projectroot/ .
```

- A JAR aláírása

```
> jarsigner –keystore "$HOME$\.keystore" myJar.jar littlejoe
```

- A JAR verifikálása

```
> jarsigner –verify myJar.jar
```

Digitálisan aláírt JAR-ok III.

- Az alábbi egy aláíratlan JAR file alap manifest-jének tartalma:

```
Manifest-Version: 1.0
Created-By: 1.4.1-rc (Sun Microsystems Inc.)
```

- A manifest tartalma DSA kulcspárral és SHA-1 lenyomattal történő aláírás után (a jarsigner jelenleg még RSA kulcspárral és MD5 lenyomattal tud aláírni):

```
Manifest-Version: 1.0
Created-By: 1.4.1-rc (Sun Microsystems Inc.)

Name: Test2.class
SHA1-Digest: xPgUOiN12cY/yfI34TReauPYdmI=

Name: Test.class
SHA1-Digest: rM4RIMVRNx38F7h6r+Cha+sqLBg=
```

Digitálisan aláírt JAR-ok IV.

- Ha a JAR-t egy user nevű személy írta alá, a USER.SF és USER.DSA nevű file-ok jöttek létre.
- A USER.DSA egy **bináris file**, amely a USER.SF file **aláírás**át, a **publikus kulcs**ot, és az ellenőrzéshez szükséges **X.509-es tanúsítványok**at tartalmazza.
- A USER.SF file tartalma hasonló a manifest-éhez:

```
Signature-Version: 1.0
Created-By: 1.4.1-rc (Sun Microsystems Inc.)
SHA1-Digest-Manifest: TFDeVvAJ7j8tb9wl3r+DyCY2MHA=

Name: Test2.class
SHA1-Digest: CZPWVfd+2/nTEI7bNL6cMvln/Bw=

Name: Test.class
SHA1-Digest: i65m3Rdr+vLd/vVi/941EnsVllA=
```

Digitálisan aláírt JAR-ok V.

- USER.SF file bejegyzéseinek jelentése a következő:
  - ***x*-Digest-Manifest**: a manifest file *x* lenyomata,
  - ***x*-Digest**: a manifest file-ban az adott file-hoz tartozó bejegyzés *x* lenyomata.
- A JAR ellenőrzésének menete (aláírónként):
  - Az **SF file aláírásának ellenőrzése** a DSA file-ban tárolt aláírás, publikus kulcs és tanúsítványok segítségével.
  - A **manifest file eredetiségének ellenőrzése** az SF-ben tárolt lenyomatok alapján (egyben, vagy bejegyzésenként).
  - A JAR-ban talált **file-ok lenyomatának kiszámítása**, és **összevetése a manifest-ben tárolt lenyomatokkal**.