Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Properties, preferences

Bevezetés

- Számos esetben szükség lehet arra, hogy egy program két futtatása között megőrizzünk bizonyos **beállítások**at, **paraméterek**et.
- Ezt megtehetjük saját formátumú file-ok formájában is, de a Java közvetlenül is támogatja ennek megvalósítását.
- Két ilyen szolgáltatás létezik:
  - a **Properties** osztály, és
  - J2SE 1.4 óta a **Preferences API**

Properties

- A Properties osztály **név-érték párok** tárolását teszi lehetővé.
- Mind a **nevek**, mind az **értékek** kizárólag **String**-ek lehetnek.
- Szolgáltatások:
  - Név-érték párokat bármikor **betehetünk** a Properties objektumba.
  - Egy Properties objektum tartalmát **kimenthetjük egy stream-be**, illetve **betölthetjük egy stream-ből**.
  - **Kilistázhatjuk** az összes **név-érték párt**, illetve **lekérdezhetjük** a **nevek listáját**.
  - A Properties objektum létrehozásakor megadhatunk egy másik Properties objektumot, amelyekben **default értékek** találhatók.

Lekérdezés Properties objektumokból

- Egy névhez tartozó lekérdezés eredménye az alábbiak szerint alakul:
  - Ha a név-érték párok között a megadott névhez **tartozik leképezés**, akkor a megfelelő érték lesz az eredmény.
  - Ha adtunk meg **default érték**eket tartalmazó Properties objektumot, és az azon végrehajtott lekérdezés eredménye nem null, akkor ez az érték lesz az eredmény.
  - Ha adtunk meg **default értéket a lekérdezéshez**, akkor ez az érték lesz az eredmény.
  - Egyébként az eredmény **null** lesz.

Property file-ok

- Egy Property objektum tartalmát rendszerint egy .properties kiterjesztésű file-ban tároljuk.
- A **neveket** és **értékeket "="** illetve **":"** jelek **választják el** egymástól.
- A **név-érték párokat újsor választja el**, de ha a sor utolsó karaktere "\", a következő sor is az érték részének tekintendő.
- A név a sor első nem szóköz karakterével kezdődik, és a következő szóközig, illetve "=", vagy ":" jelig tart. A nevek "=", ":" jeleket, illetve szóközöket "\" után tartalmazhatnak.
- A **"#"**-al vagy **"!"**-lel kezdődő sorok **megjegyzések**.
- A property file-ok **ISO 8859-1**-es kódolásban tárolódnak, a speciális karaktereket Unicode escape formájában lehet megadni. A property file-ok a **native2ascii** programmal konvertálhatók ISO 8859-1-be.

Megjegyzések

- Mivel a Properties osztály a Hashtable osztályból származik, használhatjuk a Hashtable put, illetve putAll metódusait is név-érték párok felvételére, de ha ezeken keresztül nem String típusú nevet vagy értéket viszünk be, a legközelebbi kimentési művelet sikertelen lesz.
- A Properties osztályt leszármaztatva létrehozhatunk speciális Properties osztályokat, amelyek támogatják a számunkra szükséges egyéb típusok tárolását. Ilyen esetben az adott típust String-gé, illetve egy String-et az adott típussá konvertáló metódusokra van szükség.

```
public void setProperty (String key,int value) {
   setProperty (key,Integer.toString (value));
}
public int getIntProperty (String key,int default)
  throws NumberFormatException {
   String stringValue = getProperty (key,Integer.toString (default));
   return Integer.parseInt (stringValue);
}
```

Példa I.

- Az alábbi példában egy boolean típusú értéket szeretnénk egy property file-ban tárolni. A default értékeket is meg akarjuk adni egy Properties objektum formájában.

```
import java.util.*;

public class Program {

   public void metodus () {
      if (Boolean.getBoolean (properties.getProperty ("enabled"))) {
         ...
      }
   }

   ...

   private Properties properties;
}
```

Példa II.

```
import java.util.*;
public class Program {
   ...
   public void loadProperties () {
      Properties default = new Properties ();
      default.setProperty ("enabled",Boolean.toString (true));
      properties = new Properties (default);
      try {
         properties.load (new FileInputStream ("p.properties"));
      } catch (IOException e) {}
   }
   public void saveProperties () {
      try {
         FileOutputStream s = new FileOutputStream ("p.properties");
         properties.store (s,"Program properties");
      } catch (IOException e) {}
   }
   ...
}
```

Példa III.

- Az előző példában a saveProperties metódus végrehajtásának hatására az alábbi file jön létre p.properties néven:

```
#Program properties
#Wed Nov 27 09:46:47 CEST 2002
enabled=true
```

- Természetesen az is gyakori eset, hogy a property file-t kézzel hozzuk létre, a program pedig csak kiolvassa belőle a számára szükséges opciókat.

System property-k I.

- A Java futtató környezetnek (de lényegében a háttérben elinduló VM-nek) indításkor a –D<név>=<érték> opcióval átadhatunk paramétereket, amelyek értékét a futó programból kiolvashatjuk.
- Ezek a paraméterek az úgynevezett **system property**-k.
- A System osztály statikus **getProperties** metódusával elkérhetünk egy Properties objektumot, ami a system property-ket tartalmazza, vagy a **getProperty** metódussal egy-egy property értékét kaphatjuk meg.
- Számos system property név **foglalt**, ezeket a futtató környezet használja. Ezekből megtudhatjuk a futtató környezet és az operációs rendszer verzióját, gyártóját, a classpath értékét, az aktuális felhasználó nevét és home könyvtárát, az aktuális könyvtárat, a rendszerben használt file és útvonal elválasztó karaktereket, és még sok minden mást.

System property-k II.

- Az alábbi példaprogram kiírja az operációs rendszer nevét és verzióját, a Java implementáció gyártóját és verzióját, a classpath értékét, és az aktuális felhasználó nevét.

```
public class SystemProperties {

   public static void main (String[] args) {
      java.util.Properties sp = System.getProperties ();
      System.out.println ("OS name: " + sp.getProperty ("os.name"));
      System.out.println ("OS version: " +
         sp.getProperty ("os.version"));
      System.out.println ("Java: " + sp.getProperty ("java.vendor"));
      System.out.println ("Java version: " +
         sp.getProperty ("java.version"));
      System.out.println ("classpath: "+
         sp.getProperty ("java.class.path"));
      System.out.println ("username: "+sp.getProperty ("user.name"));
   }
}
```

System property-k III.

- Az előbbi program futtatásának eredménye valami hasonló lesz:

```
OS name: Windows XP
OS version: 5.1
Java: Sun Microsystems Inc.
Java version: 1.4.1
classpath: .
username: ambi
```

- ... vagy:

```
OS name: Linux
OS version: 2.2.19
Java: Sun Microsystems Inc.
Java version: 1.4.0
classpath: .
username: webteam
```

A Preferences API

- A Preferences API a Properties osztály által nyújtott funkcionalitást terjeszti ki:
  - bejegyzések **hierarchikus elrendezés**e,
  - **eseményvezérelt értesítés** a hierarchia struktúrájának és a bejegyzések értékének változásairól,
  - String kulcsok, de az **értékek tetszőleges típusúak** lehetnek,
  - **rendszerszintű** és **felhasználói bejegyzések** külön tárolása,
  - **automatikus perzisztencia** (platform- és implementációfüggő tárolás),
  - **import/export** szöveg file helyett **XML-be**.
- A Preferences API elemei a **java.util.prefs** csomagban kaptak helyet.

Hierarchikus felépítés I.

- A Preferences API-ban a bejegyzések egy **fa csomópontjaiban** helyezkednek el, egy csomópontban **több bejegyzés** is lehet.
- A csomópontoknak **nevük**, illetve **abszolút**, valamint **relatív elérési útvonaluk** van.
- A csomópontok neve tetszőleges, de "/" jelet nem tartalmazhat, ez a karakter a hierarchiaszinteket választja el az útvonalakban. A gyökér csomópont neve az üres string.
- A gyökér csomópont abszolút útvonala /, a gyökér gyermekeié /<a csomópont neve>, a többi csomóponté pedig <a szülő abszolút útvonala>/<a csomópont neve>.
- A **relatív útvonalak** a filerendszerekben szokásos módon képezhetők: a relatív útvonal a viszonyítási alapként szolgáló ős abszolút útvonalát a kérdéses csomópont abszolút útvonalára kiegészítő string.

Hierarchikus felépítés II.

- Az érvényes útvonalak az alábbi két feltételnek kell eleget tegyenek:
  - Nem tartalmazhatnak egymás után több "/" jelet.
  - A gyökér csomópont útvonalát kivéve nem végződhetnek "/" jellel.
- Szokásos, hogy **egy csomaghoz tartozó bejegyzések útvonala** a **csomag nevé**vel egyezik meg: a java.util.prefs csomag bejegyzései általában a /java/util/prefs csomópontban lennének.
- A rendszerszintű bejegyzésekhez egy fa tartozik, míg az egyes felhasználókhoz egy-egy külön fa. Ez lehetővé teszi, hogy az **összes felhasználót érintő beállítások** (például egy programhoz telepített kiterjesztések listája) és az **egyes felhasználók egyéni beállításai** (színek, ablakok pozíciói) **elkülönülhessenek egymástól**.

Eseményvezérelt értesítés

- Két fajta listener-t regisztrálhatunk egy csomóponthoz:
  - NodeChangeListener: a **csomópont strukturájának** változásait jelzi, vagyis azt, hogy a csomóponthoz hozzáadtunk, vagy eltávolítottunk egy gyermeket.
  - PreferenceChangeListener: a **csomópont tartalmának** változásait jelzi, vagyis azt, hogy egy bejegyzést hozzáadtunk, eltávolítottunk, vagy megváltoztattunk.

Automatikus perzisztencia

- A Preferences API mögött egy úgynevezett **backing store** van, amely az egyes csomópontok, és azok tartalmának tárolását végzi el.
- A backing store megvalósítása platform- és implementációfüggő, **Windows**-ban a **Registry**-ben tárolódnak az adatok, **Unix**-ban és **Linux**-ban pedig egy-egy **file**-ban (minden fa külön file-ban).
- A backing store-t egyszerre több VM is használhatja, a konkurrens hozzáféréssel kapcsolatban csak annyit kell a backing store megvalósításnak garantálnia, hogy az adatok ne sérüljenek meg, konzisztencia feltételek nincsenek.
- A backing store-ral kapcsolatos metódusok BackingStoreException-t dobnak, ha a backing store nem elérhető.

XML kapcsolat

- Az egyes fák illetve részfák tartalmát **XML-be ki lehet menteni**, majd később az **XML file-ból be lehet olvasni**.
- Az XML file a http://java.sun.com/dtd/preferences.dtd nevű DTD-nek (Document Type Descriptor) kell megfeleljen. (Természetesen az export funkció által létrehozott XML file megfelel ennek a DTD-nek.
- Az XML kimentés illetve betöltés célja az, hogy a backing store-ban lévő adatokat elmenthessük például **archiválás**, vagy **szállítás** céljából.
- Ha a visszatöltendő XML file nem felel meg a fenti DTD-nek, a Preferences osztály importPreferences metódusa InvalidPreferencesFormatException-t dob.

A Preferences API elemei

- A **Preferences** osztály egy csomópontot reprezentál.
- Absztrakt osztály, a konkrét backing store megvalósítás tartalmazza az ebből származtatott, komplett osztályt.
- Az **AbstractPreferences** osztály a Preferences osztályból származik, és a backing store funkciók kivételével mindent megvalósít.
- A **PreferencesFactory** interfészt implementáló tényleges osztályt a backing store megvalósítás tartalmazza. Ez az osztály hozza létre a rendszer-, illetve a felhasználói fákat.
- A **NodeChangeListener**, **NodeChangeEvent**, **PreferenceChangeListener**, **PreferenceChangeEvent** interfészek és osztályok az eseménykezelést valósítják meg.

Példa I.

- Az alábbi példában egy rendszerszintű, és egy felhasználói szintű bejegyzést akarunk kezelni.

```
import java.util.prefs.*;

public class Program {
   public void metodus () {
      Preferences sysNode =
         Preferences.systemNodeForPackage (Program.class);
      boolean enabled = sysNode.getBoolean (ENABLED,ENABLED_DEFAULT);
      Preferences userNode =
         Preferences.userRoot ().node ("windows/file");
      byte[] color = userNode.getByteArray (COLOR,null);
      ...
   }
   private static final String ENABLED = "enabled";
   private static final booelan ENABLED_DEFAULT = true;
   private static final String COLOR = "color";
}
```

Példa II.

- A következő példában egy rendszerbejegyzést figyelünk.

```
import java.util.prefs.*;

public class Program implements PreferenceChangeListener {

   public Program () {
      Preferences sysNode =
         Preferences.systemRoot ().node ("program/settings");
      sysNode.addPreferenceChangeListener (this);
   }

   public void preferenceChange (PreferenceChangeEvent e) {
      if ("mode".equals (e.getKey ()) {
         ...
      }
   }
}
```