Annotációk, biztonság, nemzetköziesítés, reflection

Simon Károly
simon.karoly@codespring.ro

1. rész

Annotációk

2014.07.16. 17:14

Annotations

- Meta adatok beépítése a forráskódba.
- A programról adnak információkat, hozzárendelhetőek metódusokhoz, attribútumokhoz, osztályokhoz, illetve a program további elemeihez is.
  - Tulajdonképpen felfoghatóak speciális típusmódosítóként is – bárhol használhatóak, ahol típusmódosítók használhatóak.
  - A konvenciónak megfelelően a többi típusmódosító előtt használjuk ezeket.
- Felhasználás:
  - Információszolgáltatás a fordítónak (pl. figyelmeztetések elfojtása stb.)
  - Fordítási/telepítési időben történő feldolgozás: az annotációk alapján bizonyos eszközök forráskódot, állományokat generálhatnak, vagy hasonló műveleteket végezhetnek.
  - Futási idejű feldolgozás: egyes annotációk futási idejű ellenőrzések/műveletek elvégzésére adnak lehetőséget.
- Egyes keretrendszerek az annotációk alapján biztosíthatják szolgáltatásaikat (pl. perzisztencia keretrendszerek, tesztelési keretrendszerek alkalmazása stb.).

Deklaráció

- Az interfészek deklarációjához hasonlóan:

```
public @interface MyAnnotation {
        String toDo ();
        String date ()              default "[unimplemented]";
}
```

- Lehetőség alapértelmezett érték megadására.
- Típusokkal kapcsolatos megkötések: primitív, String, Class, enum, annotáció és ilyen típusú elemekből álló tömb elfogadható.
- Használat:

```
@MyAnnotation (
        toDo = "Sample task",
        date = "07-03-2012"
)
public void myMethod() {
        ...
}
```

Annotáció típusok

- Marker annotációk - nem tartalmaznak elemeket:
  - `public @interface MyAnnotation {}`
  - ```
    @MyAnnotation
    public void myMethod () { … }
    ```
- Egy elemet tartalmazó annotációk:
  - ```
    public @interface MyAnnotation {
      String toDo ();
    }
    ```
  - ```
    @MyAnnotation ("Sample Task")
    public void myMethod () { … }
    ```
- Kategóriák:
  - Egyszerű annotációk: a forráskódban használható alapvető annotációk (@Override, @Deprecated, @SuppressWarnings)
  - Meta-annotációk: annotációk annotálására → annotációk létrehozásánál (@Target, @Retention, @Documented, @Inherited)
    ```

Egyszerű annotációk

- ```
  @Override
  public String toString () {
          return "Sample String";
  }
  ```

- ```
  @Deprecated
  public void doSomething () { … }
  ```

- ```
  @SuppressWarnings ({"deprecation"})
  public void doSomethingNow () {
          …
          ref.doSomething ();
  }
  ```

- SuppressWarnings: all, deprecation, serial, unchecked, unused stb.
  ```

Meta annotációk

- ```
  @Retention (RetentionPolicy.RUNTIME)
  @Target (ElementType.METHOD)
  public @interface Test { }
  ```
- Retention: a RetentionPolicy.RUNTIME érték jelzi a virtuális gépnek, hogy az annotációnak futási időben is olvashatónak kell lennie a reflection mechanizmus által (SOURCE – a fordító már nem veszi figyelembe, CLASS – a fordító figyelembe veszi, de a VM nem, futási időben nem olvasható).
- Target: jelzi, hogy hol használható az annotáció (TYPE, METHOD, FIELD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE)
- Documented: az annotáció dokumentálva kell legyen a javadoc eszköz által (részét fogja képezni a generált dokumentumnak).
- Inherited: az ilyen típusú annotáció automatikusan öröklődik az annotált osztályból származtatott osztályok által.
  ```

2. rész

Java – biztonsági megoldások

Sandbox, Security

2014.07.16. 17:14

Biztonság

- Platform security:
  - Erősen típusos nyelv, automatikus memóriamenedzsment, a bájtkód ellenőrzése
  - Biztonságos osztálybetöltés, homokverem (sandbox) mechanizmus, JSA
- Kriptográfia:
  - JCA (Java Cryptography Architecture)
- Azonosítás és hozzáférési jogok (authentication and authorization/ access control):
  - JSA (Java Security Architecture)
  - JAAS (Java Authentication and Authorization Service)
- Biztonságos kommunikáció
  - JSSE (Java Secure Socket Extension)
  - Java GSS-API (Generic Security Services)
  - Java SASL API (Simple Authentication and Security Layer)
- Nyilvános kulcsok és tanúsítványok (certificate) menedzsmentjére szolgáló API-k

Sandbox, JSA

- A programokkal szemben gyakori elvárás, hogy meg kell felelniük bizonyos adatvédelmi és biztonsági követelményeknek, be kell tartaniuk bizonyos szabályokat.
- A Java biztonsági mechanizmusának az alapja a „homokverem” (sandbox) modell.
- Az alkalmazások a JVM által létrehozott elszigetelt környezetben futnak, így nincs lehetőség arra, hogy külső (pl. hálózaton keresztül elérhető), nem megbízható programok közvetlenül, ellenőrzés nélkül férjenek hozza lokális erőforrásokhoz. Az ilyen programok, csak a sandbox-on belül biztosított korlátozott erőforrásokat használhatják.
- A mechanizmusnak köszönhetően különböző programokhoz különböző jogosultságok rendelhetőek.
- Minden programot jellemeznek bizonyos jogosultságok, amelyek meghatározzák például, hogy az illető program milyen állományokat módosíthat, milyen információkhoz férhet hozzá, nyithat-e meg hálózati kapcsolatokat, indíthat-e el más programokat és így tovább.

Security manager, policy

- SecurityManager objektumok felelősek azért, hogy a programok betartsák a szabályokat.
- A „jogkörök” meghatározása az alkalmazott biztonsági stratégiának, vezérelvnek (security policy) megfelelően történik.
- A jogok általában .policy kiterjesztésű konfigurációs állományokban vannak tárolva, innen olvashatjuk ki őket és programon belüli reprezentációjuk java.security.Policy típusú objektumok formájában adott.
- Az alapértelmezett .policy állomány a JRE lib/security alkönyvtárában található.
- Az állományok jogosultságokat meghatározó bejegyzéseket tartalmaznak, az alábbi szintaxis szerint:
-

```
grant signedBy "signer_names", codeBase "URL" {
permission permission_class_name "target_name",
           "action", signedBy "signer_names";
                                    ....
};
```

Permissions

- **signedBy**: a kód szerzőjének meghatározása: a jogkör olyan kódra vonatkozik, amelyet a megadott szerzők digitális aláírása hitelesít. A mező opcionális, elhagyása azt jelenti, hogy a jogok az aláírástól függetlenül érvényesek.
- **codeBase**: a kód helyét azonosító URL. A jogok az illető helyen található kódokra érvényesek. A mező szintén elhagyható, így a jogok a kód helyétől függetlenül érvényeseknek lesznek tekintve.
- A **permission** mezők határozzák meg magukat a jogokat, „engedélyeket”.

  A **permission_class_name** az engedély típusa, például java.io.FilePermission. A **target_name** értéke az engedélyezett művelet célpontja, tárgya. Ez egy FilePermission típusú engedély esetében például egy állomány lenne. Az **action** mező az olyan engedélytípusok esetében fontos, ahol az engedélyezett műveletnek több formája lehet. Például a FilePermission esetében különböző hozzáférési formákról beszélhetünk, az action mező értéke lehet például „read”. A **signedBy** mező itt is digitális aláírással kapcsolatos, de ebben az esetben maga az engedély típusának megfelelő osztály kell hitelesítve legyen a megadott szerző aláírásával, ahhoz, hogy az engedélyt érvényesnek tekintse a rendszer.
- A mezők nagyrésze opcionális, például az alábbi módon egy program részére minden jog megadható:

```
grant {
        permission java.security.AllPermissions;
};
```

Policy

- A Policy objektumoknak megfelelő jogok betartásának ellenőrzéséért a SecurityManager objektum felelős, és SecurityException típusú kivételt generál, ha valamilyen program sérteni próbálja a rá vonatkozó szabályokat.
- A System osztály getSecurityManager nevű metódusával kérhetünk egy az aktuális SecurityManager objektumra mutató referenciát. Az osztály check előtagú (checkXXX formájú) metódusokat biztosít, különböző jogosultságok ellenőrzésére.
- Az alapértelmezett .policy állománynak megfelelő jogokat kiegészíthetjük, vagy lecserélhetjük. Ez a program futtatásakor történhet a következő módon:

```
java -Djava.security.manager -Djava.security.policy=URL App
```

- Az URL a .policy állomány helyét határozza meg. A –D kapcsoló a rendszer konfigurációs paramétereinek beállítására szolgál.

Policy

- `java -Djava.security.manager -Djava.security.policy=URL App`
- Az első rész biztosítja, hogy a program futtatása során az alapértelmezett SecurityManager objektum ellenőrizze a jogok betartását. Erre csak akkor van szükség, ha az alkalmazás (az App a .class állomány neve) nem biztosít saját SecurityManager-t (ezt a System osztály setSecurityManager metódusának segítségével teheti meg).
- A második rész az alapértelmezetten használt .policy állományban meghatározott jogokat kiegészíti az URL által meghatározott állományban meghatározott jogokkal. A második résznél, az URL előtt „=” helyett „==” is használható, és ebben az esetben csak az URL által beazonosított .policy állományban található jogok lesznek érvényesek.
- Megjegyzendő, hogy a fenti műveletek elvégzésére, az alapértelmezett beállítások módosítására csak akkor van lehetőség, ha ezt a rendszer biztonsági beállításai megengedik, azaz, ha a java.security konfigurációs állományon belül a policy.allowSystemProperty tulajdonság értéke true (ez az alapértelmezett beállítás).

JCA

- java.security csomag
- Provider osztály – hozzáférés kriptográfiai algoritmusokat megvalósító csomagokhoz.
- SecureRandom osztály – kriptográfiai algoritmusokban alkalmazható véletlen számok generálása.
- MessageDigest osztály: üzenetek biztonsága (pl. SHA1, MD5 hash algoritmusok).
- Signature: digitális aláírások (pl. DSA/RSA alapú algoritmusok).
- Cipher osztály: kriptográfiai algoritmusok (AES, DES stb.).
- KeyFactory, key interfészek és osztályok: kulcsok menedzsmentje, kulcsgenerálás.
- CertificateFactory, tanúsítványok generálása és menedzsmentje.

MessageDigest példa

```
package edu.codespring.bibliospring.backend.util;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public final class PasswordEncrypter {

    private static final Logger LOG = LoggerFactory.getLogger (PasswordEncrypter.class);

    public static String generateHashedPassword (final String password, final String salt) {
        String hashedPassword = "";
        byte[] initialBytes;

        try {
            initialBytes = (password + salt).getBytes (PropertyProvider.INSTANCE.getProperty ("passwordEncoding"));
            final MessageDigest algorithm = MessageDigest.getInstance (PropertyProvider.INSTANCE
                    .getProperty ("passwordHashingAlgorithm"));
            algorithm.reset ();
            algorithm.update (initialBytes);
            final byte[] hashedBytes = algorithm.digest ();
            hashedPassword = new String (hashedBytes);
        } catch (final UnsupportedEncodingException e) {
            LOG.error ("Password encryption failed: unsupported encoding", e);
        } catch (final NoSuchAlgorithmException nsae) {
            LOG.error ("Password encryption failed: hashing algorithm not supported", nsae);
        }

        return hashedPassword;
    }
}
```

User – setPassword:

```
public void setPassword (final String password) {
        this.password = PasswordEncrypter.generateHashedPassword (password, "");
}
```

3. rész

Tulajdonságok, nemzetköziesítés és lokalizáció

2014.07.16. 17:14

Properties

- Általános szabály, hogy a különböző konfigurációs jellemzőket (amelyek változhatnak) ne építsük be a kódunkba (hardcoding), mivel ez minden változás esetén az újrafordítás kényszeréhez vezetne.
- Pl.: hálózati kliens-szerver alkalmazás esetén a cím vagy port változása ne eredményezze a kód újrafordítását.
- A paramétereket konfigurációs állományokban tároljuk.
- Properties osztály: egy alkalmazás esetében előforduló perzisztens tulajdonságok kezelésére alkalmas.
- Tulajdonképpen egy hasító tábla alapú szótár implementáció, a Hashtable osztály leszármazottja. A tulajdonságokat azonosító-érték párok formájában állományokban tárolhatjuk, a következő módon:

```
identifier1 = value1
identifier2 = value2
…
```

Properties

- Az állományok tipikusan .properties kiterjesztésűek. A Properties osztály megfelelő metódusaival (load és store) beolvashatjuk, vagy lementhetjük (módosíthatjuk) ezeket a tulajdonságokat. Ezen kívül az osztály lehetőséget biztosít arra, hogy lekérjünk, vagy módosítsunk egy adott azonosítóval rendelkező tulajdonságot:
- `public String getProperty (String key)`
- `public Object setProperty (String key, String value)`
- A második metódus az illető azonosítónak megfelelő előző értéket téríti vissza, ha volt ilyen, egyébként null értéket ad.

Properties

```
package edu.codespring.bibliospring.swingclient.util;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class PropertyProvider {

    private static Properties properties;

    static {
        properties = new Properties ();
        FileInputStream fis = null;
        try {
            fis = new FileInputStream ("./bin/edu/codespring/bibliospring/swingclient/res/BSSwingGeneral.properties");
            properties.load (fis);
        } catch (final IOException e) {
            e.printStackTrace ();
        } finally {
            try {
                if (fis != null) {
                    fis.close ();
                }
            } catch (final IOException e) {
                e.printStackTrace ();
            }
        }
    }

    public static String getProperty (final String key) {
        return properties.getProperty (key);
    }

}
```

BSSwingGeneral.properties tartalma lehet például:

```
encoding = UTF8
default_language = en
default_region = US
```

i18n, L10n, g11n

- A számítógépes alkalmazásoknál általában, de főként a grafikus felhasználói felületek esetében, általános követelmény a nemzetköziesítés és lokalizáció.
- A nemzetköziesítés többnyelvűséget jelent: a különböző anyanyelvű felhasználók könnyen (a program módosítása/újrafordítása nélkül) állíthassák át a felület nyelvét.
- Fontos továbbá a különböző helyi, a felhasználó országában megszokott megjelenítési módok, konvenciók betartása, például numerikus adatok, dátumok, pénzösszegek megjelenítésének esetén. A már nemzetköziesített, több nyelvet támogató programok esetében a lokalizáció folyamata felelős az adatok megfelelő megjelenítéséért.
- Az angol nyelvű szakterminológiában a két folyamat neve **internationalization**, illetve **localization**, és gyakran használják az i18n és L10n rövidítéseket, ahol a számok a megnevezések első és utolsó betűi közötti karakterek számát jelölik. Mivel a két műveletet általában együtt alkalmazzák, a szakirodalomban néhol találkozhatunk a **globalization** (g11n) kifejezéssel is, a két művelet együttes megnevezéseként.

Locale

- ISO Language Code (ISO 639), ISO Country Code (ISO 3166)

- ```
  Locale l1 = new Locale ("en", "US");
  Locale l2 = new Locale ("en", "GB");
  ```

```
import java.util.Locale;
public class LocaleExample {
   public static void main(String[] args) {
          Locale[] list = Locale.getAvailableLocales ();
          for (Locale l:list)
                  System.out.println(l.getLanguage() +
                          " "  +l.getCountry());
   }
}
```
```

Locale

- Alapértelmezett érték megadása: a user.language és user.region tulajdonságok beállításával, vagy a Locale osztály setDefault() metódusának segítségével.
- A user.region és user.language rendszertulajdonságok a VM indulásakor lesznek beállítva, ha változtatni akarjuk ezeket, akkor a VM argumentumaként kell megadnunk az értékeket, annak indításakor. Az értékek futási idejű változtatása (System.setProperties metódus segítségével nem vonja maga után a defaultLocal érték módosítását).
- Következmény: ha futási időben akarjuk változtatni az alapértelmezett nyelvet/régiót a Locale osztály setDefault metódusát alkalmazhatjuk.

ResourceBundle

- ```
  System.out.println ("Hello.");
  System.out.println ("How are you?");
  System.out.println ("Goodbye.");
  ```

- ```
  import java.util.Locale;
  import java.util.ResourceBundle;
  public class LocaleExample {
     public static void main (String[] args) {
       Locale currentLocale;
       ResourceBundle messages;
       try {
                 currentLocale = new Locale(args[0], args[1]);
       } catch (Exception e) {
                 currentLocale = Locale.getDefault();
       }
       messages = ResourceBundle.getBundle(
                           "MessageBundle", currentLocale);
       System.out.println(messages.getString("greetings"));
       System.out.println(messages.getString("inquiry"));
       System.out.println(messages.getString("farewell"));
     }
  }
  ```

MessageBundle

- MessagesBundle.properties
  greetings = Hello.
  farewell = Goodbye.
  inquiry = How are you?

- MessagesBundle_de_DE.properties
  greetings = Hallo.
  farewell = Tschüß.
  inquiry = Wie geht's?

- MessagesBundle_en_US.properties
  greetings = Hello.
  farewell = Goodbye.
  inquiry = How are you?

- MessagesBundle_fr_FR.properties
  greetings = Bonjour.
  farewell = Au revoir.
  inquiry = Comment allez-vous?

MessageBundle

- Ha a megadott értékek nem felelnek meg az érvényes régió és nyelv kódoknak, a fordító nem jelez hibát, mivel a problémát megoldja a ResourceBundle osztály keresési stratégiája.
- A gyártómetódus először megpróbálja betölteni a megadott Locale-nak megfelelő erőforrás állományt. Amennyiben ez nem sikerül megpróbálja betölteni az alapértelmezett régiónak és nyelvnek megfelelő állományt.
- Megjegyezhetjük, hogy a getBundle metódus második paramétere el is hagyható, és ilyenkor is az alapértelmezett Locale-nak megfelelő állományt próbálja meg betölteni a rendszer.
- Ha az alapértelmezett Locale-nak megfelelő állomány sem létezik, a rendszer továbblépik a keresésben, és megpróbálja betölteni az alapértelmezettként megadott erőforrás állományt.
- Megjegyezhetjük, hogy ugyanezt az eredményt elérhetjük úgy is, hogy a Locale konstruktorának üres karakterláncokat adunk meg paraméterként. Ilyen esetben mindig az alapértelmezettként megadott állomány, és nem az alapértelmezett Locale-nak megfelelő állomány kerül betöltésre.
- Az erőforrások betöltésekor probléma csak akkor léphet fel, ha az alapértelmezett állomány sem található. Ebben az esetben MissingResourceException típusú futási idejű kivételt kapunk.

MessageBundle

- Oda kell figyelnünk az erőforrás állományok elhelyezésére, illetve az alap állománynév megadásakor az útvonal helyes megadására.
- Az erőforrás állományokat, az osztályokhoz hasonlóan, a rendszer az osztálybetöltő (classloader) segítségével tölti be (a ClassLoader.getResource metódus által). Ennek megfelelően a projekthez tartozó csomagok hierarchiájában keresi ezeket. Amennyiben nem használunk csomagokat, az állományoknak az osztályállományokkal egy könyvtárban kell lenniük. Ha a betöltést végző osztály valamilyen csomagban található, akkor az állományokat a projekt gyökérkönyvtárában kell elhelyeznünk.
- Általában az erőforrás állományokat egy projekten belül külön könyvtárakba csoportosítjuk. Ebben az esetben az alap állománynév megadásakor az állományt tartalmazó csomagot is meg kell határoznunk. Tételezzük fel, hogy példánk esetében az erőforrás állományokat a projekten belül egy külön "res" könyvtárban tároljuk. Ebben az esetben a program megfelelő sora a következőképpen módosul:

```
messages = ResourceBundle.getBundle("res.MessageBundle", currentLocale);
```

MessageBundle

- A rendszer a megadott "res.MessageBundle" karakterláncból felépíti a megfelelő elérési útvonalat (a "." karaktereket "/" karakterekkel helyettesítve, és az állománynevet a Locale-nak megfelelő utótagokkal, illetve a .properties kiterjesztéssel kiegészítve).
- Megjegyzendő, hogy a keresés ebben az esetben is a csomaghierarchián belül, a classpath gyökerétől kiindulva történik. Amennyiben abszolút elérési útvonalat szeretnénk megadni (bár ez ritkán szükséges, előfordulhat, ha az állományok a projekten kívül találhatóak), akkor egy, a célnak megfelelő osztálybetöltőt (pl. URLClassLoader) alkalmazhatunk (a getBundle metódus háromparaméteres változatának segítségével adhatunk át a metódusnak egy erre mutató referenciát).

Number format and currency

```
import java.text.NumberFormat;
import java.util.Locale;

public class NumberFormatDemo {

   public static void displayNumber (Locale currentLocale) {
      Integer quantity = new Integer (123456);
      Double amount = new Double (345987.246);
      NumberFormat numberFormatter;
      String quantityOut;
      String amountOut;
      numberFormatter = NumberFormat.getNumberInstance (currentLocale);
      quantityOut = numberFormatter.format (quantity);
      amountOut = numberFormatter.format (amount);
      System.out.println (quantityOut + "   " + currentLocale);
      System.out.println (amountOut + "   " + currentLocale);
   }

   public static void displayCurrency (Locale currentLocale) {
      Double currency = new Double (9876543.21);
      NumberFormat currencyFormatter;
      String currencyOut;
      currencyFormatter = NumberFormat.getCurrencyInstance (currentLocale);
      currencyOut = currencyFormatter.format (currency);
      System.out.println (currencyOut + "   " + currentLocale.toString ());
   }
```

Number format and currency

```
public static void displayPercent (Locale currentLocale) {
      Double percent = new Double (0.75);
      NumberFormat percentFormatter;
      String percentOut;
      percentFormatter = NumberFormat.getPercentInstance (currentLocale);
      percentOut = percentFormatter.format (percent);
      System.out.println(percentOut + "   " + currentLocale.toString());
}
```

```
public static void main (String[] args) {
      Locale[] locales = {
          new Locale ("fr","FR"),
          new Locale ("de","DE"),
          new Locale ("en","US")
      };
      for (int i = 0; i < locales.length; i++) {
         System.out.println();
         displayNumber (locales[i]);
         displayCurrency (locales[i]);
         displayPercent (locales[i]);
      }
   }
}
```

Példa: BiblioSpring i18n

```
package edu.codespring.bibliospring.swingclient.i18n;

import java.io.UnsupportedEncodingException;
import java.util.Locale;
import java.util.MissingResourceException;
import java.util.ResourceBundle;

import edu.codespring.bibliospring.swingclient.util.PropertyProvider;

public class LabelProvider {

    private static final String    BUNDLE_NAME     = "edu.codespring.bibliospring.swingclient.res.BSSwingLabels";
    private static ResourceBundle resourceBundle = ResourceBundle.getBundle (BUNDLE_NAME,
                                        new Locale (PropertyProvider.getProperty ("default_language"),
                                                    PropertyProvider.getProperty ("default_region")));

    public static String getLabel (final String key) {
        try {
            return new String (resourceBundle.getString (key).getBytes (),
                               PropertyProvider.getProperty ("encoding"));
        } catch (final MissingResourceException e) {
            return '!' + key + '!';
        } catch (final UnsupportedEncodingException e) {
            return resourceBundle.getString (key);
        }
    }

    public static void setLocale (final Locale locale) {
        resourceBundle = ResourceBundle.getBundle (BUNDLE_NAME, locale);
    }
}
```

Példa: BiblioSpring i18n

- BSSwingLabels.properties:

```
…
MainFrame.loanButton=Loans
MainFrame.reservationButton=Reservations
MainFrame.title=BiblioSpring Admin Interface
MainFrame.titleButton=Title Management
MainFrame.userButton=User Managemenet
…
```

- BSSwingLabels_hu_HU.properties:

```
…
MainFrame.loanButton=Kölcsönzés
MainFrame.reservationButton=Foglalás
MainFrame.title=BiblioSpring Admin felület
MainFrame.titleButton=Címek menedzsmentje
MainFrame.userButton=Felhasználók menedzsmentje
…
```

Példa: BiblioSpring i18n

```
package edu.codespring.bibliospring.swingclient.gui;

…            //imports
import edu.codespring.bibliospring.swingclient.i18n.LabelProvider;

public class MainFrame extends JFrame {

…

      private void initializeGUI () {
            …
            bookButton = new JButton (LabelProvider.getLabel ("MainFrame.titleButton"));
            userButton = new JButton (LabelProvider.getLabel ("MainFrame.userButton"));
            reservationButton = new JButton (LabelProvider.getLabel ("MainFrame.reservationButton"));
            loanButton = new JButton (LabelProvider.getLabel ("MainFrame.loanButton"));
            …
          }
      …
}
```

Példa: BS – BackEnd properties

```
package edu.codespring.bibliospring.backend.util;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public enum PropertyProvider {
    INSTANCE;

    private static final Logger LOG          = LoggerFactory.getLogger (PropertyProvider.class);
    private static final String BUNDLE_NAME = "edu.codespring.bibliospring.backend.res.BiblioSpringGeneral";

    private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle (BUNDLE_NAME);

    public String getProperty (final String key) {
        try {
            return RESOURCE_BUNDLE.getString (key);
        } catch (final MissingResourceException e) {
            LOG.error ("Missing resource", e);
            throw e;
        }
    }
}
```

BiblioSpringGeneral tartalma lehet például:

```
passwordHashingAlgorithm = SHA
passwordEncoding = utf-8
dbURL = jdbc:mysql://localhost:3306/bibliospring
dbUser = root
dbPassword = root
dbConnectionPoolSize = 10
```

```
public final class ConnectionManager {

…
private final String dbURL =
    PropertyProvider.INSTANCE.getProperty ("dbURL");
…
```

4. rész

Reflection

Class

- ```
  String str = "string";
  Class c = str.getClass ();
  ```
- ```
  System.out.println (c.getName ());
  ```
- ```
  try {
      String str2 = (String) c.newInstance ();
  } catch (InstantiationException e1) { ...
  } catch (IllegalAccessException e2) { ... }
  ```
- ```
  try {
      Class cls = Class.forName ("Class_name");
  } catch (ClassNotFoundException e) { ... }
  ```

- Dinamikus betöltés - példa:

```
interface Typewriter {
    void typeLine (String s);
}
class Printer implements Typewriter {
    ...
}
```

```
class MyApplication {
    ...
 String deviceName = "Printer";
 try {
    Class c = Class.forName (deviceName);
    Typewriter device =
           (Typewriter) c.newInstance ();
    device.typeLine ("Hello");
 } catch (Exception e) {...}
}
```

Reflection

CODESPRING
OUR CODE YOUR SUCCESS

- java.lang.reflect.Field
- java.lang.reflect.Method
- java.lang.reflect.Constructor

Method	Description
`Field[] getFields ();`	A publikus attribútumok (öröklöttek is) visszafordítása
`Field getField (String name);`	A paraméter által meghatározott publikus attribútum
`Field[] getDeclaredFields ();`	Minden attribútum (a nem publikusak is)
`Field getDeclaredField (String name);`	A paraméter által meghatározott attribútum
`Method [] getMethods ();`	Publikus metódusok (öröklöttek is)
`Method getMethod (String name, Class[] argumentTypes);`	A paraméterek által meghatározott publikus metódus
`Method[] getDeclaredMethods ();`	Minden metódus
`Method getDeclaredMethod (String name, Class[] argumentTypes);`	A paraméterek által meghatározott metódus
`Constructor[] getConstructors ();`	Publikus konstruktorok (öröklöttek is)
`Constructor getConstructor (Class [] argumentTypes);`	A paraméterek által meghatározott publikus konstruktor
`Constructor[] getDeclaredConstructors ();`	Minden konstruktor
`Constructor getDeclaredConstructor (Class[] argumentTypes);`	A paraméterek által meghatározott konstruktor

Reflection

- Hozzáférés az attribútumokhoz:

```
class BankingAccount {
    public int deposit;
}

BankingAccount account;
try {
    Field fdeposit = BankingAccount.class.getField ("deposit");
    int deposit = fdeposit.getInt (account);
    fdeposit.setInt (account, 42);
} catch (NoSuchFieldException e 1) {
      ...
} catch (IllegalAccessException e2){
      ...
}
```

- A **`setXXX ()`** és **`getXXX ()`** metódusok csak a publikus attribútumok esetében használhatóak (egyébként **`IllegalAccessException`**)

Reflection, security

- Argumentumok nélküli statikus metódus meghívása:

```
import java.lang.reflect.Method;
import java.lang.reflect.InvocationTargetException;

public class MethodInvocation {

    public static void main (String[] args) {
        try {
            Class c = Class.forName (args[0]);
            Method m = c.getMethod (args[1], new Class[]{});
            Object o = m.invoke (null, null);
            System.out.println ("Static method invocation (without arguments): "
                                + args[1] + " from the class: " + args[0] + "\nResult: " + o );
        } catch (ClassNotFoundException e1) {
            e1.printStackTrace ();
        } catch (NoSuchMethodException e2) {
            e2.printStackTrace ();
        } catch (IllegalAccessException e3) {
            e3.printStackTrace ();
        } catch (InvocationTargetException e4) {
            e4.printStackTrace ();
        }
    }
}
```

Az invoke () metódus paraméterei:

- A megfelelő objektumra mutató referencia (statikus metódusok esetében null)
- A metódus paramétereinek típusait tartalmazó tömb (primitív adattípusoknál Class.TYPE – pl. Integer.TYPE)

- Használat példa:

```
java MethodInvocation java.lang.System currentTimeMillis
```

Reflection, security

- Konstruktorok meghívása:

```
try {
  Constructor c = Date.class.getConstructor (new Class[] {String.class});
  Object o = c.newInstance (new Object[] {"Jan 1,2001"});
  Date d = (Date) o;
  System.out.println(d);
} catch (ClassNotFoundException e1) {
      e1.printStackTrace ();
} catch (NoSuchMethodException e2) {
      e2.printStackTrace ();
} catch (IllegalAccessException e3){
      e3.printStackTrace ();
} catch (InvocationTargetException e4) {
      e4.printStackTrace ();
}
```

Reflection és annotációk

- ```
  @Retention(RetentionPolicy.RUNTIME)
  @Target(ElementType.TYPE)
  public @interface MyAnnotation {
     public String name();
     public String value();
  }
  ```

- ```
  @MyAnnotation(name="someName", value = "Hello World")
  public class TheClass { }
  ```

- ```
  Class aClass = TheClass.class;
  Annotation[] annotations = aClass.getAnnotations();
  for(Annotation annotation : annotations){
     if (annotation instanceof MyAnnotation){
           MyAnnotation myAnnotation = (MyAnnotation) annotation;
           System.out.println("name: " + myAnnotation.name());
           System.out.println("value: " + myAnnotation.value());
     }
  }
  ```

- Vagy: `Annotation annotation = aClass.getAnnotation(MyAnnotation.class);`
  ```

Reflection és annotációk

- ```
  Method method = ... //obtain method object
  Annotation[] annotations = method.getDeclaredAnnotations ();
  ```

- ```
  Method method = ... // obtain method object
  Annotation annotation = method.getAnnotation (MyAnnotation.class);
  ```

- ```
  Field field = ... //obtain field object
  Annotation[] annotations = field.getDeclaredAnnotations ();
  ```

- ```
  Method method = ... //obtain method object
  Annotation[][] parameterAnnotations = method.getParameterAnnotations ();
  ```