9. Gyakorlat

legendi@inf.elte.hu

2010. április 13.

Reflection

"Önelemzés" - futás közben a program lekérdezheti a lehetőségeit, milyen részekből áll, etc. Lehetőségek: program passzív vizsgálata (pl. ahol a program publikus szolgáltatásainak felderítése, *Java Beans*), ill. korlátozottan egyes részek módosítása (pl. láthatóság).

A *Java Core Reflection* erősen típusos, biztonságos felület osztályok, objektumok vizsgálatára, használható a következőkre (amennyiben a biztonsági szabályok engedélyezik):

- új objektumok, tömbök létrehozása
- adattagok lekérdezése, módosítása
- függvények lekérdezése, meghívása
- tömbelemek lekérdezése, módosítása
- új osztályok létrehozása

Fontos osztályok: `java.lang.reflect.*` csomag:

- `Field, Method, Constructor` adattagok, függvények (`invoke(...)`, konstruktorok (`newInstance(...)`) lekérdezéséhez
- `Class` osztály-, ill. interfész információk eléréséhez
- `Package` csomagok kezeléséhez
- `Proxy` új osztályok létrehozásához
- `Array` tömbök dinamikus létrehozása, lekérdezése
- `Modifier` módosítók visszafejtésében segít (public, protected, etc.)

Class

Objektumreferencia megszerzése:

1. Objektumtól lekérdezhető:
   `Class<?> clazz = this.getClass();`
2. Osztálytól lekérdezhető:
   `Class<?> intClazz = int.class;`
3. Közvetlenül név szerint lekérdezhető:
   `Class<?> clazz = Class.forName("java.lang.Boolean");`
4. Új osztály létrehozása:
   `Proxy.getProxyClass(clazz.getClassLoader(), clazz.getInterfaces())`

Példa

```
package gyak9;

import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

public class ReflectionTest {
    public static void analyze(final Class<?> clazz) {
        System.out.println("Osztaly neve: " +
                 clazz.getName());
        System.out.println("Csomagja: " +
                clazz.getPackage());
        System.out.println("Ososztalyanak neve: " +
                clazz.getSuperclass());

        System.out.println("Deklaralt public fuggvenyek:");
        for ( final Method act : clazz.getDeclaredMethods() ) {
            if ( Modifier.isPublic( act.getModifiers() ) ) {
                System.out.println(act.getName());
            }
        }
    }

    public static void main(final String[] args) {
        analyze( ReflectionTest.class );
    }
}
```

Feladatok

Készíts egy programot, amely egyetlen parancssori argumentumot kap, egy osztály teljes hivatkozási nevét (*fully qualified name*), és

1. eldönti, hogy az adott osztály belső, névtelen, vagy lokális osztály-e. Ha egyik sem, akkor nézze meg, hogy felsorolási típus, annotáció vagy interfész-e (*a sorrendre figyeljetek, mert az annotáció is interfésznek minősül*).

2. bejárja az osztályban deklarált összes adattagot, és kiírja azok nevét, típusát, és módosítószavait.

3. megkeresi az osztály összes publikus, deklarlált konstruktorát, valamint kiírja azok paramétereinek a számát.

4. lekérdezi az összes adattagot, és megnézi, hogy hányhoz van getter, setter függvény definiálva, valamint hány olyan van, amelyhez mindkettő definiálva van.

5. megvizsgálja, hogy az adott osztálynak van-e nullary (zero-arg) konstruktora, és ha talál ilyet, csinál belőle egy példányt, valamint az alapértelmezett `toString()` függvényének a segítségével kiírja a képernyőre.

6. kiírja az összes (azaz nem csak a deklarált!) függvény módosítószavait, visszatérési értékének típusát, nevét, valamint paramétereinek típusát.

7. végigmegy az adott osztály összes statikus függvényén, és készít egy listát az ezen függvények által dobható kivételekről.

Részletek `http://java.sun.com/javase/6/docs/api/java/lang/reflect/package-frame.html`

Megjegyzés Primitív (+`void`) típusok, pl. boolean reprezentációjának `Class` példánya `java.lang.Boolean.TYPE`, rövidebb formája `boolean.class`, többinél ugyanígy.

Tömbök

Az `Array` osztály segítségével manipulálhatók az elemek (getter, setter függvények), új tömbök hozhatók létre (`newInstance()`), ill. a `Class` osztálynak vannak hasznos függvényei, pl.:

```
package gyak9;

public class ReflectionArrayTest {
    public static void arrayTest(final Class<?> clazz) {
```

```
        if ( ! clazz.isArray()) {
            System.out.println("Nem tomb");
            return;
        }

        Class<?> act = clazz;
        int dim = 0;
        while (act.isArray()) {
            act = act.getComponentType();
            dim++;
        }

        System.out.println( dim + " dimenzios");
        System.out.println( "Belso tipusa: " +
                act.getSimpleName());
    }

    public static void main(final String[] args) {
        arrayTest( new int[][] { {1, 2}, {3}}.getClass() );
    }
}
```

Megjegyzés Hülye jelölés, nem szívrohamot kapni:

```
int[][] arr = { {1, 2}, {3} };
System.out.println( arr );

// Eredmeny: [[I@42e816
```

Ok: B - byte, C - char, D - double, F - float, I- int, J - long, L*osztálynév* - osztály vagy interfész, S - short, Z - boolean, [- tömb

Feladat

Készítsünk egy programot, amely képes létrehozni egy adott típusú tömböt, majd az elemeit beállítani egy kitüntetett értékre. A program a paramétereket parancssori argumentumként kapja (elég a 8 primitív típusra felkészülni, valamint `String` objektumokra).

Feladat

Készíts egy mélységi `String deepToString(Object[] arr)` segédeljárást tömbökhöz! A függvénynek egyetlen paramétere legyen: a kiírni kívánt tömb. A függvény menjen végig a az elemeken, és vizsgálja meg őket. Ha az

nem tömb, akkor fűzze hozzá a szöveges reprezentációját az objektumnak a visszaadott Stringhez. Ha tömb, akkor vizsgálja meg mind a nyolc primitív típusra (`elementClass == byte[].class`, etc.), és annak megfelelően dolgozza fel az elemeket. Ha nem primitív típus az elemtípus, akkor hívja meg a rekurzív függvényt újabb feldolgozásra!

Függvények

Függvényeket meg is tudunk hívni, ld. `Method#invoke(Object o, Object... args)` függvény. Ha a függvény statikus, akkor az első paraméter lehet `null` (különben reccs), paraméterlista lehet üres, visszatérési értéke egy `Object`. Példa:

```
package gyak9;

import java.lang.reflect.Method;

public class Invoking {
    public static int add(final int a, final int b) {
        return a + b;
    }

    public static void main(final String[] args) throws Exception {
        final Method method = Invoking.class.getMethod("add",
                new Class[] {
                    Integer.TYPE, Integer.TYPE
                });

        System.out.println( method.invoke(null, 1, 2) );
    }
}
```

Feladat

Készíts egy tetszőleges objektumot, majd reflection segítségével keresd meg az összes, paraméter nélküli getter függvényét! Ezeket hívd is meg reflection segítségével, és az eredményüket írd ki a képernyőre!

+/- Feladat

A feladatokat a `legendi@inf.elte.hu` címre küldjétek, **következő szombat éjfélig**! A subject a következőképp nézzen ki:

```
csop<csoportszám>_gyak<gyakorlat száma>_<EHA-kód>_<feladatok száma>
```

Például:

```
csop1_gyak9_LERIAAT_1
csop1_gyak9_LERIAAT_12
```

Mellékelni csak a Java forrásfájlokat mellékeljétek (semmiképp ne teljes Eclipse/NetBeans projecteket), esetleg rarolva, zipelve, és egy levelet küldjetek (több levél esetén az utolsó mellékleteit értékelem)! További fontos kritériumok:

- A konvenciókra figyeljetek plz!
- Ékezetes karaktereket **ne** használjatok! Főleg azonosítók esetében ne! A Java ugyan ezt megengedi, ugyanakkor a különböző környezetekbe való konvertáláskor ($latin2 \leftrightarrow UTF-8 \leftrightarrow Cp1250$) összetörnek a karakterek! Az ilyen forrásokat fordítani, következésképp értékelni sem tudom.

1. Feladat

Készítsünk egy programot, amely képes egy adott osztály osztály-hierarchiáját előállítani, valamint meg tudja mondani, hogy pontosan milyen interfészeket implementál (az ősosztályok által implementáltakat is)! Az osztály nevét parancssori argumentumként kapjuk. Példa:

```
interface I1 {}
interface I2 extends I1 {}

class A implements I2 {}
class B extends A {}
```

A `B` osztály vizsgálata esetén a következő listát adja vissza a program:

```
[class gyak9.B, class gyak9.A, interface gyak9.I2,
  interface gyak9.I1, class java.lang.Object]
```

2. Feladat

Bizonyítsd be, hogy a `String` osztály csak gyengén tekinthető *immutable* osztálynak (azaz reflection segítségével kiügyeskedhető az általa reprezentált szöveg megváltoztatása)!

A szöveg legyen ugyanolyan hosszú, de egy karakterét tetszőlegesen változtasd meg. A program megírásához szükséged lesz a String osztály forráskódjára (esetleg reflectionnel történő vizsgálatára), ezt a *JDK/src.zip* fájlban találhatod meg (vagy a különböző IDE-k segíthetnek a felkutatásában, pl. Eclipse alatt a *Ctrl + Snift + T* billentyűkombináció).

Megjegyzés Az ilyesmivel azért csak offtosan! Mivel magában a reprezentációban turkálunk, anélkül, hogy figyelnénk az objektum konzisztenciájának megőrzésére, gondok lehetnek a használatából (pl. a `String#hashCode()` is lazy instantiationnel cache-eli az eredményt)! Ugyanakkor - sajnos - néha elkerülhetetlen, ilyenkor rendkívül hasznos tud lenni ez a csel.