Java7 - java.util.Objects

A Java7 egyik csendes újítása a java.util.Objects osztály, amely közel tucatnyi statikus metódust ad az objektumok kezeléséhez, amely metódusok egy része nem esik zavarba, ha *null* értéket kap. Az Objects tipikus esete utility osztályoknak, *final* módosítóval van ellátva, illetve van egy privát konstruktora is:

Objects.java

```
private Objects() {
  throw new AssertionError("No java.util.Objects instances for you!");
}
```

Suhanjunk végig az eddig megvalósított metóduslistán... 🙂

equals

Az els metódus egy egyszer egyezség vizsgálat, amely levizsgálja a két összehasonlítandó paraméter közül az elst, hogy annak értéke nem *null*, majd ezek után végzi el az összevetést, így ez a metódus *null safe*: használata során nem kell NullPointerException kivételre számítanunk. A forrása egyszer:

Objects.java

```
public static boolean equals(Object a, Object b) {
  return (a == b) || (a != null && a.equals(b));
}
```

Használata:

Example.java

```
String a = null;
String b = null;
System.out.println(Objects.equals(a, b));
b = "nem üres";
System.out.println(Objects.equals(a, b));
a = "nem üres";
System.out.println(Objects.equals(a, b));
```

Az eredmény:

```
true
false
true
```

deepEquals

A *deepEquals* metódus hasonlít a fentire, de sima *equals* helyett az *Arrays.deepEquals* metódust hívja meg *null* ellenrzéssel, így egy sorban tudunk két tömböt összevetni:

Objects.java

```
public static boolean deepEquals(Object a, Object b) {
  if (a == b)
    return true;
  else if (a == null || b == null)
    return false;
  else
    return Arrays.deepEquals0(a, b);
}
```

Használata:

Example.java

```
String[] a = null;
String[] b = null;
System.out.println(Objects.deepEquals(a, b));
b = new String[]{"semmi", "újabb semmi"};
System.out.println(Objects.deepEquals(a, b));
a = new String[]{"semmi", "újabb semmi"};
System.out.println(Objects.deepEquals(a, b));
```

Az eredmény:

```
true
false
true
```

hashCode

A *hashCode* lehetséget ad *null safe* hívásra, az átadott objektumnak csak akkor hívódik meg a *hashCode* metódusa, ha a paraméter nem *null*:

Objects.java

```
public static int hashCode(Object o) {
  return o != null ? o.hashCode() : 0;
}
```

Használata:

Example.java

```
String a = null;
System.out.println(Objects.hashCode(a));
a = "semmi";
System.out.println(Objects.hashCode(a));
```

Az eredmény:

```
0
109322039
```

hash

Hasonlít az elzre, azzal a különbséggel, hogy a metódus paraméterlistája dinamikusan bvíthet, s az eredményül kapott objektum-tömb átadásra kerül a *Arrays.hashCode* metódusnak, így lehetségünk van *null* biztos hash képzésre több argumentumból:

Objects.java

```
public static int hash(Object... values) {
  return Arrays.hashCode(values);
}
```

Használata:

Example.java

```
String a = null;
String b = null;
System.out.println(Objects.hash(a, b));
b = "semmi";
System.out.println(Objects.hash(a, b));
a = "semmi";
System.out.println(Objects.hash(a, b));
```

Az eredmény:

```
961
109323000
-796661087
```

toString

Ez a metódus kissé kilóg a többi közül, mert *null* biztos konverzióra eddig is lehetség volt az alább is látható *String.valueOf(o)* használatával:

Objects.java

```
public static String toString(Object o) {
  return String.valueOf(o);
}
```

Használata:

Example.java

```
String a = null;
System.out.println(Objects.toString(a));
a = "semmi";
System.out.println(Objects.toString(a));
```

Az eredmény:

```
null
semmi
```

toString + alapértelmezett érték

Az elznél sokkal hasznosabb metódus a most következ, amelynél megadhatunk egy alapértelmezett értéket, ha az átadott objektum értéke *null*:

Objects.java

```
public static String toString(Object o, String nullDefault) {
  return (o != null) ? o.toString() : nullDefault;
}
```

Használata:

Example.java

```
String a = null;
System.out.println(Objects.toString(a, "hopsza, ez null"));
a = "semmi";
System.out.println(Objects.toString(a));
```

Az eredmény:

```
hopsza, ez null
semmi
```

compare

A *compare* metódus két objektum összehasonlítására jó, s ez a többi metódushoz képest két okból is rendhagyó:

- nem *null safe*
- kell neki egy Comparator implementáció, amely szükség esetén lehet *null safe*

Objects.java

```
public static <T> int compare(T a, T b, Comparator<? super T> c) {
  return (a == b) ? 0 :  c.compare(a, b);
}
```

Használata (hibát okoz):

Example.java

```
Comparator<String> comparator = new Comparator<String>() {
  public int compare(String o1, String o2)
  {
    return o1.compareTo(o2);
  }
};

String a = null;
String b = null;
System.out.println(Objects.compare(a, b, comparator));
b = "semmi";
System.out.println(Objects.compare(a, b, comparator));
a = "semmi";
System.out.println(Objects.compare(a, b, comparator));
```

Az eredmény:

```
0
Exception in thread "main" java.lang.NullPointerException
    at hu.javaforum.showobjects.App$1.compare(App.java:19)
    at hu.javaforum.showobjects.App$1.compare(App.java:15)
    at java.util.Objects.compare(Objects.java:181)
    at hu.javaforum.showobjects.App.main(App.java:27)
```

requireNonNull

Ez egy nagyon hasznos metódus, jegyezzük meg, mert nem csinál ugyan sokat, de azt gyakran használjuk, ha paramétereket ellenrzünk! Megvizsgálja, hogy az átadott els paraméter *null* érték-e, ha igen, akkor dob egy *NullPointerException* kivételt, ha nem, akkor visszaadja az objektumot:

Objects.java

```
public static <T> T requireNonNull(T obj) {
  if (obj == null)
    throw new NullPointerException();
  return obj;
}
```

Használata:

Example.java

```
String a = Objects.requireNonNull("semmi");
System.out.println(a);
String b = Objects.requireNonNull(null);
System.out.println(b);
```

Az eredmény:

```
semmi
Exception in thread "main" java.lang.NullPointerException
    at java.util.Objects.requireNonNull(Objects.java:201)
    at hu.javaforum.showobjects.App.main(App.java:17)
```

requireNonNull + szöveg

Hasonló a fentihez, ám ez esetben megadhatunk egy szöveget, amely adott esetben átadódik a *NullPointerException* kivételnek:

Objects.java

```
public static <T> T requireNonNull(T obj, String message) {
  if (obj == null)
    throw new NullPointerException(message);
  return obj;
}
```

Használata:

Example.java

```
String a = Objects.requireNonNull("semmi", "A paraméter értéke null");
System.out.println(a);
String b = Objects.requireNonNull(null, "A paraméter értéke null");
System.out.println(b);
```

Az eredmény:

```
semmi
Exception in thread "main" java.lang.NullPointerException: A paraméter értéke null
    at java.util.Objects.requireNonNull(Objects.java:226)
    at hu.javaforum.showobjects.App.main(App.java:17)
```

Konklúzió

Nagyon hasznos osztály, használja mindenki – aki teheti, ugyanis nagyon sok saját osztály van a projektjeinkben, amelyek hasonló vagy pontosan ilyen módon mködnek, mivel egy-egy fenti metódus használatával a forráskódunk egyszerbb és átláthatóbb lesz, másrészt kevesebb saját forráskódot kell majd karbantartanunk.

(inspiráció: www.javabeat.net/2012/06/java-util-objects-a-static-utility-for-objects-introduced-in-java-7/)