A HashMap, a TreeMap és a Hashtable...

...avagy a jó, a rossz és a csúf. A közös tulajdonságuk egyszer, implementálják a Map interfészt, s kulcs-érték mentén tárolnak adatokat, illetve a hatékony mködést a kulcs hash értéke alapján végzik. Mi a különbség a HashMap, a TreeMap és a Hashtable között? A legels implementáció a Hashtable, amely az 1.0 óta van jelen a Java nyelvben, mondhatni együtt ntt fel vele. Manapság eléggé mellzött szerepe van, szinte alig találkozunk a Hashtable alapon szervezett tárolókkal, s ennek egyik oka a szinkronizált mködés adta lassúsága. További érdekessége a Hashtable osztálynak, hogy nem szereti a *null* érté keket, az alábbi program:

```
Map map = new Hashtable();
map.put("a", null);
```

Szép kivételt okoz futásidben:

```
Exception in thread "main" java.lang.NullPointerException
        at java.util.Hashtable.put(Hashtable.java:394)
        at test.Main.test(Main.java:16)
```

Hasznos lehet tudni, hogy a Properties osztály a Hashtable osztályból származik, ezért nem tudunk *null* értéket vagy kulcsot felvenni, mint *property*.

A HashMap és a TreeMap egyidben, az Java 1.2 idejében került a nyelv alapvet osztályai közé, és mindketten megvalósítják a Map interfészt, így mind a három implementáció könnyedén cserélhet, ha jól írtuk meg a programot (érdekességképp megemlíteném, hogy a Hashtable csak az 1.2 verziótól implementálja a Map interfészt).

A HashMap két alapvet dologban különbözik a Hashtable osztálytól: elviseli a *null* értéket, illetve mködése nem szinkronizált, így nem kell vizsgálnunk a kulcsot vagy az értéket, azonban szálbiztos környezetben tudjuk csak használni.

A TreeMap - ellentétben az elz kettvel - képes rendezett sorrendben tartani a beledobott értékeket a kulcs alapján, tehát a hozzáadott értékeken a kulcs alapján növekv sorrendben tudunk végiglépkedni:

```
Map map = new TreeMap();
map.put("d", "d");
map.put("a", "a");
map.put("b", "b");
map.put("c", "c");
for (Object key : map.keySet())
{
  System.out.println(key);
}
```

Az eredmény a kulcs objektuma által diktált sorbarendezés (vagy a compareTo metódus):

```
a
b
c
d
```

A TreeMap ilyen viselkedése okán a kulcs általában nem lehet *null*, mivel ekkor a legtöbb osztály bizony kivételre fut:

```
Exception in thread "main" java.lang.NullPointerException
        at java.lang.String.compareTo(String.java:1167)
        at java.lang.String.compareTo(String.java:92)
```

Lezárásképp nézzük meg, hogy milyen teljesítményre számíthatunk a három implementáció esetén. A méréshez egy objektum tárat használunk, mivel a Map interfész nem fogad el primitív típust, így elre el kell készítenünk sok ezer különböz példányt, különben meghamisítanánk a mérést, a kód így néz ki:

```
Integer[] objectPool = new Integer[2000000];
for (Integer j = 0; j < 2000000; j++)
{
  objectPool[j] = j;
}

long startTime = System.nanoTime();
for (Integer i = 0; i < 100; i++)
{
  map = new HashMap();
  for (int j = 0; j < 2000000; j++)
  {
    map.put(objectPool[j], i);
  }
}
System.out.println(Math.round((System.nanoTime() - startTime) / 10000) / 100.0 + " ms");

startTime = System.nanoTime();
for (Integer i = 0; i < 100; i++)
{
  for (int j = 0; j < 2000000; j++)
  {
    map.get(objectPool[j]);
  }
}
System.out.println(Math.round((System.nanoTime() - startTime) / 10000) / 100.0 + " ms");
```

Ebben fogjuk a konkrét implementációt lecserélni sorban a három osztály egyikére. Nos, nézzük az eredményeket (vagyis 200 millió put és 200 millió get hívás idejét):

```
Hashtable:
90611.36 ms
3974.38 ms
HashMap:
86662.92 ms
3982.17 ms
TreeMap:
143784.58 ms
38699.3 ms
```

A következtetéseket mindenki vonja le magának.