

Java8 - A Collection interfész változásai

A Java8 – jobban mondva a Lambda csomag – lehetőséget ad arra, hogy [interfészekbe lehessen tenni kódrészleteket](#), amelyeknek els nekifutásra nem sok értelme van, hiszen erre eddig is lehetőséget kaptunk az absztrakt osztályokon keresztül... a többszörös öröklésre pedig nincs akkora igény, hogy komolyan el kellene gondolkodni rajta a nyelvet módosító fejlesztőknek. Ha jobban beleássuk magunkat a Lambda rejtelmeibe, akkor láthatunk egy logikus folyamatot, amely megmagyarázza a Virtual Extension Method néven nevezett nyelvi eszköz szükségességét. Írjunk egy rövid programot, amely létrehoz egy listát, benne számokkal 0 és 9 között:

LambdaRun.java

```
public class LambdaRun
{
    public static void main(String[] args)
    {
        java.util.List<Integer> numbers =
            new java.util.ArrayList<Integer>(java.util.Arrays.asList(new Integer[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}));
    }
}
```

A feladat legyen egyszer: írassuk ki a listában található számokat... a hagyományos módszer szerint valami ilyesmit írunk a program végére:

Hagyományos módszer

```
for(Integer number : numbers) System.out.println(number);
```

Az új módszer se sokkal kevesebb, a *List* interfész kapott egy új – *forEach* nev – metódust, amelybe tudunk írni Lambda kifejezéseket:

Lambda módszer

```
numbers.forEach(number -> {System.out.println(number)});
```

Az ördög a részletekben rejlik, nézzünk bele a [List](#) interfész forráskódjába, de ott nem találunk benne *forEach* metódust, amelyet a [Collection](#) interfészben se találunk (amelyből a *List* származik), mivel ez a metódus az [Iterable](#) interfészben van, amely a *Collection* egyik se, de lássuk gyorsan az alapértelmezett metódust, amelyben egy *delegate* metódustörzs szerepel:

Iterable.java

```
void forEach(Block<? super T> block) default {
    Iterables.forEach(this, block);
}
```

Ha tovább nyomozunk, akkor megtaláljuk a konkrét implementációt az [Iterables](#) osztályban, amely tulajdonképpen végrehajtja a Lambda blokkot:

Iterables.java

```
public static <T> Iterable<T> forEach(final Iterable<? extends T> iterable, final Block<? super T> block) {
    Objects.requireNonNull(iterable);
    Objects.requireNonNull(block);
    for (T each : iterable) {
        block.apply(each);
    }

    return (Iterable<T>) iterable;
}
```

Kerekedik a dolog... ezért kell a *default* metódustörzs az interfészekbe, mert a Lambda expression miatt az összes alapvet interfészbe fel kell venni a fentihez hasonló új metódusokat, amelyeknek **mkódnie kell** a régi implementációkkal – hiszen a platform alapvet interfészeit sok helyen sokszor implementáltuk saját osztályokban! Csak úgy lehet ezt megoldani, ha az interfészekben lehetőség van olyan metódusokat felvenni, amelyeket nem kell implementálni, a kialakult megoldás pedig a platform egészében használható eszköz lesz... csak szoknunk és tanulnunk kell még ezt a megoldást.