Java 8 - Lambda expressions

A Java 8 egyik – illetve a Project Jigsaw elkaszálása óta az egyetlen – nagy újítása a Lambda expressions beépítése a nyelvi eszközök közé.

Elször érdemes tisztázni, hogy mi is az a Lambda expression: a magyar szakmai nyelvbe a funkcionális nyelvekkel együtt került be a lambda-kalkulus elnevezés, amely lehetvé teszi, hogy paraméterként függvényt lehessen átadni egy metódus hívásánál, ezzel egyszerbben lehet leírni azt a feladatot, amelyet eddig a név nélküli bels osztályokkal oldottunk meg (anonymous inner class). Lássunk egy klasszikus példát, egy listát szeretnénk rendezni egy Comparable interfészt kiterjeszt saját osztállyal.

Az els megoldás mutatja a teljes programot, amely egy lista elemeit – az egyszerség kedvéért – összekever:

LambdaRunExample.java

```
public class LambdaRunExample
{
  private static final Random random = new Random();

  public static void main(String[] args)
  {
    List<String> list = new ArrayList<String>();
    list.add("a");
    list.add("ab");
    list.add("abc");
    list.add("abcd");
    list.add("abdce");
    Collections.sort(list, new Comparator<String>()
    {
      public int compare(String o1, String o2)
      {
        return random.nextInt();
      }
    });
    System.out.println(list);
  }
}
```

Módosítsuk a programot a Lambda expressions szabályai szerint:

LambdaRunExample.java

```
public class LambdaRunExample
{
  private static final Random random = new Random();

  public static void main(String[] args)
  {
    List<String> list = new ArrayList<String>();
    list.add("a");
    list.add("ab");
    list.add("abc");
    list.add("abcd");
    list.add("abdce");
    Collections.sort(list, (o1, o2) -> random.nextInt());
    System.out.println(list);
  }
}
```

Fordítsuk le egy Java 8 JDK használatával:

```
> ./jdk1.8.0/bin/javac -version
javac 1.8.0-ea
> ./jdk1.8.0/bin/javac LambdaRunExample.java
> ./jdk1.8.0/bin/java LambdaRunExample
[a, abc, abdce, ab, abcd]
```

Vessünk egy pillantást a két megoldás lényegi részeit illet különbségekre:

```
Collections.sort(list, new Comparator<String>()
{
  public int compare(String o1, String o2)
  {
    return random.nextInt();
  }
});
```

versus

```
Collections.sort(list, (o1, o2) -> random.nextInt());
```

Ezért fogjuk szeretni a Lambda expressions használatát, bár szoknia kell a Java utóbbi 10 évéhez szokott szemnek az ilyen sorokat, de meg lehet tanulni, ahogy az annotációkat és a generics-et is megtanultuk... 😷

Mi van a Comparator osztályon túl?

A Comparator ideális állatorvosi ló, mert nagyszeren meg lehet rajta mutatni a lehetséget, de természetesen más osztályok és metódusok is vannak, amelyek meg fogják könnyíteni az életünket, lássunk ebbl is néhányat a teljesség igénye nélkül:

List.filter

Hasznos funkció lehet egy listából leszrni bizonyos elemeket, például a három karakternél hosszabb szövegeket:

```
Iterable<String> filtered = list.filter( item -> item.length() <= 3 );
System.out.println(filtered);
```

Eredményül megkapjuk a rövid listaelemeket:

```
[a, ab, abc]
```

List.map

A lista elemébl egy új listát készíthetünk, például lista elemeinek a hosszát szeretnénk megkapni eredményül:

```
Iterable<Integer> lengths = list.map( item -> item.length() );
System.out.println(lengths);
```

```
[1, 2, 3, 4, 5]
```

List.reduce

A lista elemeit egy elemmé tudjuk redukálni, a redukálás mveletét tudjuk meghatározni, például adjuk össze a fentebb kialakult lista elemeit:

```
Integer sum = lengths.reduce(0, (left, right) -> left + right);
System.out.println(sum);
```

Az els érték az, ami az els *left* lesz, amelyhez hozzáadjuk a lista els elemét, majd a kapott értékkel végigiterálunk a listán, így az eredmény 15 lesz:

```
15
```

List.forEach

Végig tudunk iterálni egy lista elemein, így például össze tudjuk fzni a lista tartalmát egy szöveggé:

```
StringBuilder sb = new StringBuilder();
list.forEach( item -> { sb.append(item).append(' '); } );
System.out.println(sb);
```

Az eredmény természetesen:

```
a ab abc abcd abdce
```

...

A lehetségek tárháza "végtelen"... 😅

Jó-jó, de hogy mködik?

A *gyári* Lambda expressions vonzó lehetség, de ez a játék sokkal érdekesebb saját függvénytörzsekkel, s ha létrehozunk saját Lambda expressions-ön alapuló megoldásokat, akkor jobban megértjük a mködését is.

Elször is szükségünk van egy interfészre, amelybl majd Lambda expressions lesz, legyen a példa egy szr, amely a kapott objektum értéke szerint majd egy igaz vagy egy hamis választ ad:

Filter.java

```
public interface Filter<T>
{
  boolean filter(T o1);
}
```

Ezek után a fentebb már elkészített *LambdaRunExample.java* állományban hozzunk létre egy metódust, amely majd elvégzi a szrést:

LambdaRunExample.java

```
public static <T> void applyFilter(List<T> list, Filter<? super T> f)
{
  List<T> removeFromList = new ArrayList<T>();
  for (T item : list)
  {
    if (f.filter(item))
    {
      removeFromList.add(item);
    }
  }
  list.removeAll(removeFromList);
}
```

Amint látjuk, a metódus második paramétereként egy *Filter* interfészt implementáló osztály egy példányát szeretnénk megkapni, amelyet a *for each* ciklusban használunk fel, ahol a kapott *item* példányról döntjük el, hogy ki kell-e szrnünk a listából vagy sem.

A lényegi rész – maga a Lambda expression – a Collections.sort metódushoz hasonlóan épül fel, egyszeren megadjuk azt a feltételt, amely szerint a szrést el szeretnénk végezni, jelen esetben a három karakternél hosszabb szövegeket az *applyFilter* metódus el fogja távolítani a listából:

```
LambdaRunExample.applyFilter(list, (item) -> item.length() > 3);
```

Felmerül a kérdés, hogy a fordításkor honnan tudja a fordító, hogy melyik metódusról van szó, s ez a kérdés a Lambda expressions lényege... 😅

Rontsuk el a *Filter* interfészt, adjuk hozzá egy új metódust:

```
public interface Filter<T>
{
  boolean filter(T o1);
  boolean anotherFilter(T o1);
}
```

A *LambdaRunExample.java* osztályt lefordítva rögtön besír a *javac*, hogy nem tudja eldönteni, mit is szeretnék tle, mert az átadott interfészben több olyan metódus is van, amelyet használhatna:

```
LambdaRunExample.java:19: error: method applyFilter in class LambdaRunExample cannot be applied to given types;
    LambdaRunExample.applyFilter(list, (item) -> item.length() > 3);
                    ^
  required: List<T>,Filter<? super T>
  found: List<String>,lambda
  reason: multiple non-overriding abstract methods found in interface Filter
  where T is a type-variable:
    T extends Object declared in method <T>applyFilter(List<T>,Filter<? super T>)
1 error
```

Szóval elégedjünk meg egy darab metódussal azon interfészeinkben, amelyeket szeretnénk a fenti módon használni... 😑

Hol érhet el információ?

Lambda expressions: http://jdk8.java.net/lambda/