Címkefelhő generálása – Java esettanulmány

Generating tag cloud – Java case study

KACZUR Sándor
Eötvös Loránd Tudományegyetem Informatikai Doktori Iskola,
1117 Budapest, Pázmány Péter sétány 1/C, kaczursandor@gmail.com

Abstract

An important part of the teaching of programming languages is the introduction to the construction and selection operations of known data structures. For example array, text, generic list, generic map, stream. These can consist of primitive types of data or own-typed objects. Starting from the available object-oriented operations, the functional operations can also be introduced. The article describes the specification, the 10-step implementation in Java, and presents the result.

Keywords: OO and functional programming, software development, Java programming language, collections

Kivonat

A programozási nyelvek tanításának fontos eleme az ismert adatszerkezetek konstrukciós és szelekciós műveleteinek megismertetése. Például tömb, szöveg, generikus lista, generikus map, folyam. Ezek állhatnak primitív típusú adatokból, illetve saját típusú objektumokból is. Kiindulva a rendelkezésre álló objektumorientált műveletekből, megismertethetőek a funkcionális műveletek is. A cikk ismerteti a specifikációt, 10 lépésben a Java nyelvű megvalósítást, és bemutatja az eredményt.

Kulcsszavak: OO és funkcionális programozás, szoftverfejlesztés, Java programozási nyelv, kollekciók

1. Bevezetés

A címkefelhők/szófelhők népszerűek, sok weboldalon megtalálhatóak. A CMS rendszerekben beépített szolgáltatás is lehet, vagy külön bővítmény/plugin is megvalósíthatja. Egy szövegben előforduló szavakból a gyakrabban előfordulókat nagyobb betűmérettel emeli ki. Eredménye lehet listás, táblázatos, esetleg képpé generált is. Kétféleképpen is megközelíthető, erre utal a Word Cloud és a Tag Cloud elnevezés. Utóbbi inkább egy blog taxonomiájához kapcsolódik és kategóriákra/címkékre érvényesül. A szófelhő a szöveg betűméretén túl megjelenítheti a szavak előfordulását is.

Az esettanulmányban tetszőleges szöveget dolgozunk fel. Ebből felépítünk egy előfordulást is mutató listás szófelhőt, amely rendezett, és a szavak betűmérete 32-16-ig változik. Azok a szavak kerülnek a szófelhőbe, amelyek legalább 5-ször előfordulnak. Kezelünk kivételeket is, például olyan szavakat, amiket nem érdemes szófelhőbe tenni. Lépésenként haladva ismertetjük a megvalósító forráskódot és az egyes lépések részeredményeit. Az esettanulmány a tanulók/hallgatók előképzettségtől függően feldolgozható egy 2-4 órás laborgyakorlaton.

A Java programozási nyelv csomagjait, osztályait, interfészeit, metódusait, műveleteit használjuk. Különböző adatszerkezetek kerülnek elő: tömb, generikus lista, generikus map, generikus folyam. Építünk a Stream API szolgáltatásaira és a lambda kifejezésekre [1-7]. A megvalósítás könnyen testreszabható, kezeli a tipikusan előforduló igényeket.

2. Tervezés és megvalósítás lépésenként

2.1. Szövegforrás előkészítése

Generálunk egy 10 bekezdésből álló szöveget a Lorem Ipsum - All the facts - Lipsum generator [8] weboldalon és a későbbi feldolgozáshoz mentjük a Java projekt `files` mappájába `lorem.txt` néven. A fájl mérete 5781 bájt. A teljes kiinduló adatforrás elérhető [9]. Részlet a fájl tartalmából:
`Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur quis mauris laoreet, lobortis orci eget, egestas dui. Vivamus pretium nunc sit amet ex dictum rutrum. Duis sodales augue dui, vitae porta eros auctor non. Pellentesque vehicula sollicitudin scelerisque. Sed urna massa, auctor nec tellus at, iaculis dictum ligula...`

2.2. Szöveges tartalom előkészítése

```
String s=new String(Files.readAllBytes(
  Paths.get("./files/lorem.txt")));
s=s.replace("\n", "").replace("\r", "").replace(",", "").
  replace(".", "").toLowerCase();
```

A megadott útvonalról a `java.nio` csomag [10] metódusaival betöltjük a szövegfájl tartalmát a `byte[]`-be, majd az `s` szövegbe. A `replace()` metódus hívásaival eltávolítjuk a szövegből a sor és bekezdés végét jelző soremelés (`LF="\n"`) és kocsi vissza (`CR="\r"`) vezérlőkaraktereket, a vessző és a pont írásjeleket (mindet külön-külön cseréljük a semmire), végül a `toLowerCase()` metódus kisbetűssé alakítja a szöveget. A szöveg 5563 db karakterből áll. Részlet az előkészített szövegből:
`lorem ipsum dolor sit amet consectetur adipiscing elit curabitur quis mauris laoreet lobortis orci eget egestas dui vivamus pretium nunc sit amet ex dictum rutrum duis sodales augue dui vitae porta eros auctor non pellentesque vehicula sollicitudin scelerisque...`

2.3. Szólista elkészítése

```
List<String> wordList=Arrays.asList(s.split(" "));
```

A szóközök mentén darabolva (`split()`) a szöveget elkészül belőle egy névtelen szövegtömb (`String[]`), amit rögtön átalakítunk (`Arrays.asList()`) szöveg típusú generikus listává (`List<String>`) [11]. A lista 826 db elemből áll. Részlet a generikus listából:
`[lorem, ipsum, dolor, sit, amet, consectetur, adipiscing, elit, curabitur, quis, mauris, laoreet, lobortis, orci, eget, egestas, dui, vivamus, pretium, nunc, sit, amet, ex, dictum, rutrum, duis, sodales, augue, dui, vitae, porta, eros, auctor, ..., hendrerit]`

2.4. Csoportosítás és megszámolás

```
Map<String, Long> wordCountMap=wordList.stream().collect(
  Collectors.groupingBy(Function.identity(), Collectors.counting()));
```

A szólistát csoportosítjuk és megszámoljuk, hogy az egyes szavak hányszor fordulnak elő (másképpen: egy-egy csoport hány elemű). Elkészül a `wordCountMap` generikus map [12], amely kulcs-érték párok halmaza (leképezés). A kulcs a szó (`String`), az érték a darabszáma (`Long`). Alkalmazkodunk ahhoz, hogy a csoportosítás során használt `counting()` megszámoló művelet `Long` típusú értéket ad vissza. 188 db kulcs-érték párt kapunk. Részlet a generikus map-ből:
`{aenean=4, elementum=3, efficitur=3, mollis=3, tempor=6, potenti=2, bibendum=4, commodo=3, purus=5, augue=4, justo=5, lorem=6, leo=7, id=4, nam=2, per=2, habitant=1, semper=3, volutpat=3, ac=13, ad=1, sodales=6, in=12, finibus=3, velit=7, ..., consectetur=6}`

2.5. Szűrés és rendezés

```
List<String> exceptList=
  Arrays.asList(new String[] {"at", "et", "in", "ut"});
Stream<Entry<String, Long>> sortedWordCountStream=
  wordCountMap.entrySet().stream().
  filter(e -> !exceptList.contains(e.getKey())).
  filter(e -> e.getValue()>=5).
  sorted((e1, e2) ->
    (e1.getValue().equals(e2.getValue())) ?
    e1.getKey().compareTo(e2.getKey()) :
    e2.getValue().compareTo(e1.getValue())
  );
```

A generikus map-et kétszer szűrjük (`filter()` művelet) úgy, hogy a kivételeket tartalmazó `exceptList`-ben ne szerepeljen a szó, valamint csak a legalább 5-ször előforduló szavakat hagyjuk meg. 71 db elemből álló folyam marad. Ebből a maradékból készítünk rendezett generikus folyamot (`sortedWordCountStream`). A `sorted()` művelet két kulcs-érték párt hasonlít össze. A rendezés érték/darabszám szerint (`getValue()`) csökkenő, azon belül kulcs/szavak szerint (`getKey()`) növekvő sorrendet biztosít. Másképpen: ha az értékek megegyeznek, akkor a növekvő sorrendet a szavak ábécé sorrendje határozza meg, egyébként a darabszámok csökkenő sorrendje dönti el. Most már könnyen látható, hogy a leggyakrabban előforduló kevés szóból 15 van, 14 előfordulás nincs... Részlet a rendezett generikus folyamból:

```
[nec=15, vel=15, ac=13, quis=13, vitae=13, eget=12, nunc=12, sed=12,
mauris=11, lectus=10, donec=9, dui=9, eu=9, ornare=9, pellentesque=9,
porttitor=9, aliquam=8, amet=8, dolor=8, fringilla=8, laoreet=8,
quam=8, sit=8, ante=7, ..., viverra=5]
```

2.6. Saját típusú listává konvertálás

Definiálunk egy `WordCount` POJO-t, `String` típusú `word` nevű, `Long` típusú `count` nevű, `int` típusú `fontSize` nevű tulajdonságokkal, getter/setter metódusokkal, és `toString()` függvénnyel.

```
List<WordCount> sortedWordCountList=
  sortedWordCountStream.
  map(e -> new WordCount(e.getKey(), e.getValue())).
  collect(Collectors.toList());
```

A `map()` intermediate művelettel a rendezett generikus folyamot bejárva, előállítjuk a POJO/`WordCount` típusú kimeneti objektumok rendezett generikus listáját. Továbbra is 71 elemmel dolgozunk. Részlet a rendezett generikus listából:

```
POJO{word: nec, count: 15, fontSize: 0}
POJO{word: vel, count: 15, fontSize: 0}
POJO{word: ac, count: 13, fontSize: 0}
POJO{word: quis, count: 13, fontSize: 0}
POJO{word: vitae, count: 13, fontSize: 0}
POJO{word: eget, count: 12, fontSize: 0}
...
POJO{word: viverra, count: 5, fontSize: 0}
```

2.7. Darabszámok összegyűjtése

```
List<Long> distinctCountList=
  sortedWordCountList.stream().map(e -> e.getCount()).distinct().
  collect(Collectors.toList());
```

A POJO típusú rendezett generikus listában lévő objektumoktól elkért darabszámok (`getCount()` POJO függvény) közül a különbözőeket (`distinct()` művelet) összegyűjtjük `Long` típusú generikus listába (`distinctCountList`). Az egyediesítő művelet nem hat az adatok sorrendjére. Tízféle előfordulást kapunk. Generikus lista: `[15, 13, 12, 11, 10, 9, 8, 7, 6, 5]`.

2.8. Betűméret lépésköze

```
final int MAX_FONT_SIZE=32;
final int MIN_FONT_SIZE=16;
long countCount=distinctCountList.size();
double stepFontSize=
  (double)(MAX_FONT_SIZE-MIN_FONT_SIZE+1)/countCount;
```

A szófelhőben a szavak gyakorisága alapján határozzuk meg a betűméretet. A betűméret 32-ről indul és fokozatosan csökken 16-ig. A betűméret léptetéséhez a tízféle gyakoriság/előfordulás meghatározza a `stepFontSize` lépésközt. A lépésköz `1.7` lett.

2.9. Betűméret kiszámítása

```
int i=0, gi=0;
while(i<sortedWordCountList.size()) {
  long count=sortedWordCountList.get(i).getCount();
  int fontSize=(int)Math.round(MAX_FONT_SIZE-gi*stepFontSize);
  while(i<sortedWordCountList.size() &&
      count==sortedWordCountList.get(i).getCount()) {
    sortedWordCountList.get(i).setFontSize(fontSize);
    i++;
  }
  gi++;
}
```

Csoportváltást alkalmazunk és a csoportot `gi`-vel indexeljük. Egy csoportba azok a POJO objektumok tartoznak, amelyeknél a szavak előfordulása megegyezik. Az algoritmus 2. lépésében az aktuális csoportra érvényesen kiszámítjuk a betűméretet (`fontSize`), ami az algoritmus 3. lépésében a csoportba tartozó minden POJO objektumnál beállításra kerül a `setFontSize()` POJO eljárással. Az algoritmus 4. lépésében léptetjük a csoport `gi` indexét. A POJO-k esetén először csak a `word` és `count` tulajdonságok kerültek beállításra, de most már a `fontSize` tulajdonság is értéket kapott. Részlet a generikus listából:

```
POJO{word: nec, count: 15, fontSize: 32}
POJO{word: vel, count: 15, fontSize: 32}
POJO{word: ac, count: 13, fontSize: 30}
POJO{word: quis, count: 13, fontSize: 30}
POJO{word: vitae, count: 13, fontSize: 30}
POJO{word: eget, count: 12, fontSize: 29}
...
POJO{word: viverra, count: 5, fontSize: 17}
```

2.10. HTML tartalom előállítása

```
StringBuilder sbHTML=new StringBuilder("<p>");
sortedWordCountList.forEach(wordCount ->
  sbHTML.append("<span style=\"font-size: ").
    append(wordCount.getFontSize()).append("px\">").
    append(wordCount.toString()).append(" </span>")
  );
sbHTML.append("</p>");
```

A generikus lista POJO objektumain végighaladva, a `forEach()` záró művelettel összeállítható a weboldal szófelhőt tartalmazó része (`sbHTML`). A 71 db szóból álló szófelhő HTML forráskódjának mérete 3409 bájt. Részlet a HTML forráskódból:

```
<p><span style="font-size: 32px">nec (15) </span><span style="font-
size: 32px">vel (15) </span><span style="font-size: 30px">ac (13)
</span><span style="font-size: 30px">quis (13) </span><span
style="font-size: 30px">vitae (13) </span><span style="font-size:
29px">eget (12) </span>...<span style="font-size: 17px">viverra (5)
</span></p>
```

3. Eredmény

Kétféleképpen jeleníthető meg az eredmény. Az 1. ábrán a 10. lépés eredménye látható (a generált HTML bekezdést kiegészítve, fájlba mentve). A 2. ábrán a 3. lépés eredményét betöltve és testre szabva, a WordClouds.com [13] weboldalon kapott grafikus eredmény látható.

nec (15) vel (15) ac (13) quis (13) vitae (13) eget (12) nunc (12) sed (12) mauris (11) lectus (10) donec (9) dui (9) eu (9) ornare (9) pellentesque (9) porttitor (9) aliquam (8) amet (8) dolor (8) fringilla (8) laoreet (8) quam (8) sit (8) ante (7) elit (7) eros (7) euismod (7) leo (7) maecenas (7) morbi (7) nisl (7) odio (7) pulvinar (7) scelerisque (7) sem (7) tellus (7) ullamcorper (7) velit (7) aliquet (6) arcu (6) blandit (6) consectetur (6) dapibus (6) eleifend (6) ipsum (6) lorem (6) maximus (6) placerat (6) porta (6) sapien (6) sodales (6) suspendisse (6) tempor (6) urna (6) vestibulum (6) cursus (5) dictum (5) est (5) felis (5) justo (5) ligula (5) malesuada (5) neque (5) nibh (5) non (5) nulla (5) purus (5) rutrum (5) suscipit (5) turpis (5) viverra (5)

1. ábra: *Lorem ipsum szófelhő (eredmény szövegként)*

2. ábra: *Lorem ipsum szófelhő (eredmény képként)*

4. Továbbfejlesztési lehetőségek

Evolúciós projektként több fázisban megvalósítható az esettanulmány:

- OO alapokkal, ha a `String` típus/osztály műveletei már ismertek, és `String[]` is beépült már, akkor közvetlen konstans szövegből, rendezés, csoportváltás és sorozatos konkatenációval előállítható a HTML tartalom és kiíratható a konzolra
- kivételkezeléssel és fájlkezeléssel kiegészítve az input szöveg betölthető fájlból, illetve a szintén fájlban lévő HTML tartalom fix része betölthető, kiegészíthető a generált szöveggel és felülírható, vagy másik fájlba menthető,

- saját osztály/objektum/POJO típussal szépen egyensúlyozhatunk a műveletekkel a modell és a vezérlés között MVC-ben, és kezelhető az összetartozó 3 tulajdonság: szó, db, betűméret,
- alap hálózatkezeléssel kiegészítve a generált címkefelhő/szófelhő HTML szövegével kiegészíthető/felülírható FTP-szerű műveletekkel távoli/felhőbeli tárhelyen lévő konstans szöveges tartalom, vagy dinamikus JSP/PHP által generált forráskód,
- fejlettebb hálózatkezeléssel a megfelelő formátumú szólistát URL paraméterként elküldve vagy API-ba feltöltve a paraméterezett alakú, betűtípusú, betűméretű, színű szófelhő kép formátumban menthető, authentikációval továbbítható másik URL-re, illetve API nélkül HTML scrapinggel is hozzájuthatunk,
- adatbázis-kezeléssel is előállíthatunk szófelhőt weboldalon lévő blog tartalomból, vagy a hozzászólások szövegéből (vagy a blog taxonomiájából: kategória, címke) és az előbbi funkciókat még valamilyen szerveres időzítővel is kiegészíthetjük és automatizálható, pl. heti 1 frissítés, vagy küldjük el hetente e-mailben vagy a telefonjára Push Notification-ként egy webáruház ügyvezetőjének a termékek szabad szavas értékelését,
- a Google Translate API-n átfuttatva a szófelhőt, többnyelvű webáruházhoz is generálhatunk szófelhőt, valamint egy webáruházban elemezhető az, hogy mire keresnek rá, amikre nincs pontos találat csak valamilyen reguláris mintaillesztővel adható részben releváns találat; döntéstámogatásra is használható: hogyan igazítsák a raktárkészletet a dinamikusan/szezonálisan változó kereslethez, vagy online reklámkampányok periódusaival alkalmazkodva; tipikus helyesírási hibáknál is adható értelmes találat.

Irodalmi hivatkozások

[1] Functional programming for Java developers, Part 1, https://www.infoworld.com/article/3314640/functional-programming-for-java-developers-part-1.html, 2020.08.20.
[2] Java SE 8: Lambda Quick Start, https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html, 2020.08.20.
[3] A Java 8 újdonságai, http://nyelvek.inf.elte.hu/leirasok/Java8/index.php, 2020.08.20.
[4] Prasad R.: Using lambda expressions in Java 11 [Tutorial], https://hub.packtpub.com/using-lambda-expressions-in-java-11-tutorial/, 2020.08.20.
[5] Kaczur S.: Ismerkedjünk lambda kifejezésekkel!, https://it-tanfolyam.hu/ismerkedjunk-lambda-kifejezesekkel/, 2020.08.20.
[6] Balogh P.: Stream API lambda kifejezésekkel, https://it-tanfolyam.hu/stream-api-lambda-kifejezesekkel/, 2020.08.20.
[7] The Java 8 Stream API Tutorial, https://www.baeldung.com/java-8-streams, 2020.08.20.
[8] Lorem Ipsum - All the facts - Lipsum generator, https://www.lipsum.com/, 2020.08.20.
[9] Véletlenszerűen generált lorem ipsum szöveg, kiinduló adatforrás, https://it-tanfolyam.hu/cimkefelho-generalasa/, 2020.08.20.
[10] java.nio csomag: Java NIO Package, https://www.javatpoint.com/java-nio-package, 2020.08.20.
[11] ArrayList (Java Platform SE 8), https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html, 2020.08.20.
[12] HashMap (Java Platform SE 8, https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html, 2020.08.20.
[13] Free online Wordcloud generator, https://www.wordclouds.com/, 2020.08.20.