

FACULTATEA CALCULATOARE, INFORMATICA SI MICROELECTRONICA
UNIVERSITATEA TEHNICA A MOLDOVEI

PROGRAMAREA ÎN REȚEA

LUCRAREA DE LABORATOR#2

HTTP Client with concurrency superpowers

Autor:

Marina JECHIU

asistent universitar:

Alex GAVRIȘCO

1 Laboratory work implementation

1.1 Tasks and Points

There's a legacy system your new client want to extend (without access to its source code). The existing system allows to get a list of orders made by client's customers and a list of categories. The orders are quite simple - created-at, total, user-id, category-id. And a category isn't complicated as well - id, name, category-id, nullable. Category is a simple recursive data structure (what means that some categories can have as parent another category, and there can be "root" categories which doesn't have a parent).

Your client wants from you a tool which can generate a new type of report - total per categories (including data from child descending categories).

You've got a simple sketch from the client.

Name	Total
Electronics	1,000,000.00
Usable	700,000.00
Smartswitches	700,000.00
VR	50,000.00
Headphones	250,000.00
Name	5,000.00
Surveillance	42,000.00

Figure 1.1 – Template de afisare a rezultatelor

There is some info about the legacy system you managed to get from your client:

- Categories URL <https://evil-legacy-service.herokuapp.com/api/v101/categories/>
- Orders URL <https://evil-legacy-service.herokuapp.com/api/v101/orders/>
- The client said that he had found a mysterious "key" 55193451-1409-4729-9cd4-7c65d63b8e76 for the legacy system

Note: The legacy system isn't documented properly (all you know about it - URL and that it must return CSV), so you need to read about HTTP and discover what additional info you may need to supply to get a response with requested data. The application must offer next functionality:

- retrieve the list of orders (since it's a legacy system, it exports data in CSV format :() within a date interval
- retrieve the list of available categories (also CSV :()
- parse and validate received data
- aggregate data
- display results to the user
- cache received data locally

1.2 Analiza lucrării de laborator

Pentru realiarea task-urilor descrise anterior a fost folosit limbajul Java. Astfel, programul realizat conține următoarele clase:

Request - această clasă are 3 parametri: url(de tip string), key(de tip string) și list(de tip ArrayList). Această clasă are o funcție principală, și anume getData(). Cu ajutorul parametrilor url și key este realizat reuest-ul, iar datele obținute(în format CSV) sunt salvate și returnate prin parametrul list.(Listing 1)

Orders - această clasă are 4 parametri id(int), total(double), category-id(int) , created(string). Datele obținute în urma request-ului sunt parsate și transformate în obiecte de tip Orders. Funcția de parsare este prezentată în Listing 2.

Categories - această clasă are 3 parametri id(int), category-id(int), name(string). Funcționalitatea acesteia este asemănătoare cu cea a clasei Orders, doar că structura obiectelor este diferită.

ReadOrders - pentru a crea request-ul și parsarea comenzilor și categoriilor, acestea au fost plasate în 2 fire de execuție diferite, dirijate de un alt fir de execuție. Pentru a crea aceste fire a fost implementată interfața Callable.

ThreadAgregation - această clasă reprezintă la rândul ei un alt fir de execuție, care va porni simultan cele 2 fire de execuție, reîntorcând controlul thread-ului principal abia după ce ambele fire își vor fi terminat execuția. Implementarea aceasta este prezentată în Listing 3

Tree - Cu ajutorul acestei clase a fost implementată afișarea în cascadă. Astfel, fiecare nod creat devine root pentru un nou tree, până când nodurile nu mai copii. Funcția de afișare este prezentată în Listing 4

În urma implementărilor realizate, rezultatele obținute pot fi vizualizate în figura 1.2.

1.3 Imagini

```
1 public ArrayList<String> getData() {
2     try {
3         URL url = new URL(this.url);
4         HttpURLConnection connection = (HttpURLConnection) url.openConnection
5     ();
6
7         connection.setRequestMethod("GET");
8         connection.setDoOutput(true);
9         connection.setRequestProperty("Accept", "text/csv");
10        connection.setRequestProperty("x-api-key", key);
11        InputStream content = (InputStream) connection.getInputStream();
12        BufferedReader in =
13            new BufferedReader(new InputStreamReader(content));
14        String line;
15        while ((line = in.readLine()) != null) {
16            list.add(line);
17        }
```

```

18
19     } catch (Exception e) {
20         e.printStackTrace();
21     }
22     return list;
23 }

```

Listing 1– Funcția ce realizează request-ul

```

1
2 public Orders(String string, ArrayList<String> ordine) {
3     int lastPosition = 0;
4     ArrayList<String> list = new ArrayList<>();
5     for (int i = 0; i < string.length(); i++) {
6         if (string.charAt(i) == ',') {
7             list.add(string.substring(lastPosition, i));
8             lastPosition = i + 1;
9         }
10    }
11    if (string.charAt(string.length() - 1) == ',') {
12        list.add("-1");
13    } else {
14
15        list.add(string.substring(lastPosition, string.length()));
16    }
17
18    for (int i = 0; i < 4; i++) {
19        if (ordine.get(i).equals("id")) {
20            this.id = list.get(i);
21        }
22
23        if (ordine.get(i).equals("category_id")) {
24            this.category_id = Integer.parseInt(list.get(i));
25        }
26
27
28        if (ordine.get(i).equals("total")) {
29            this.total = Double.parseDouble(list.get(i));
30        }
31
32        if (ordine.get(i).equals("created")) {
33            this.created = list.get(i);
34        }
35    }

```

Listing 2– Parsarea string-urilor și crearea obiectelor de tip Orders

```

1
2 public Orders(String string, ArrayList<String> ordine) {
3     int lastPosition = 0;

```

```

4      ArrayList<String> list = new ArrayList<>();
5      for (int i = 0; i < string.length(); i++) {
6          if (string.charAt(i) == ',') {
7              list.add(string.substring(lastPosition, i));
8              lastPosition = i + 1;
9          }
10     }
11     if (string.charAt(string.length() - 1) == ',') {
12         list.add("-1");
13     } else {
14
15         list.add(string.substring(lastPosition, string.length()));
16     }
17
18     for (int i = 0; i < 4; i++) {
19         if (ordine.get(i).equals("id")) {
20             this.id = list.get(i);
21         }
22
23         if (ordine.get(i).equals("category_id")) {
24             this.category_id = Integer.parseInt(list.get(i));
25
26         }
27
28         if (ordine.get(i).equals("total")) {
29             this.total = Double.parseDouble(list.get(i));
30         }
31
32         if (ordine.get(i).equals("created")) {
33             this.created = list.get(i);
34         }
35     }

```

Listing 3– Thread Agregation

```

1
2      public void show(int level) {
3          for (int i = 1; i < level; i++) {
4              System.out.print("\t");
5              this.getParent().setSuma(this.getSuma());
6
7          }
8          if(data != "root") {
9              System.out.println(data + " " + this.getSuma());
10         }
11         for (Tree child : children) {
12             child.print(level + 1);
13         }

```

Listing 4– Afișarea în cascadă

```
Automotive  242.2
  GPS & Cameras  52.56
  Wheels  502.69
  Tires  348.700000000000005
Electronics  1127.28
  Photo & Video  379.96
  Wearables  60.95
    Activity Trackers  21.54
    Action Cameras  577.26
    VR/AR  10.32
    Smartwatches  111.19
  Headphones  16.0
  TV  1023.24
Computers  175.74
  Laptops  65.19
  Network Accessories  160.37
  Tablets  446.94
  PC Components  355.24
    CPU  202.84
    GPU  30.67
    RAM  175.73
    SSD/HDD  350.66
    Motherboard  105.62
```

Figure 1.2– Afişarea rezultatelor

Concluzie

Pentru implementarea task-urilor propuse în java a fost nevoie analiza mai multor căi de soluționare. Pentru realizarea concurenței a fost aleasă implementarea interfeței Callable, deoarece aceasta ne oferă cea mai simplă cale de obținere a unor rezultate în urma executării unui thread.

Nu au fost implementate toate cerințele propuse. Astfel, în timpul executării firului ThreadAggregation main thread-ul este blocat, așteptând finisarea celui din urmă. Totodată, calcularea sumelor ținând cont de ierarhie nu este realizat, fiind implementat doar calcularea sumei din comenzi a fiecărui item individual.

References

- 1 <https://github.com/Alexx-G/PR-labs/blob/master/lab2-3.md>
- 2 , <https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html#header.cache-control>
- 3 , <https://www.callicoder.com/java-callable-and-future-tutorial/>