Test Practic

**Descriere exercitiu**

In cadrul acestui exercitiu, esti pus in ipostaza de-a face un query la un server printr-un API si de a interpreta rezultatul primit de la server. Scopul exercitiului nu este de a modifica functionalitatea “serverului”, ci de a implementa o serie de functii care fac un apel la acesta, primesc un raspuns si care interpreteaza informatia primita. In general, modul in care se realizeaza un query la un server este anevoios si trebuie tinut cont de foarte multe scenarii pentru ca programul sa ruleze fara probleme si erori: lipsa conectivitatii la internet, latenta comunicarii, functionalitatea serverului, tipul variat de raspunsuri pe care il poate trimite acesta user-ului, etc.

In general, pentru a nu se ingreuna procesul de dezvoltare, s-au creat librarii specializate care abstractizeaza intregul proces si care pun la dispozitie o interfata mai prietenoasa pentru programator. Acest execitiu doreste sa imite exact acest proces.

**Descriere program**

Functiile “librariei” sunt implementate deja in cadrul clasei NetworkUtils.

Singura functie publica din cadrul librariei cu care lucrati direct este **makeHttpRequest() .** Pentru a obtine diferite informatii de la server prin API-ul sau, se construiesc diferite URL-uri. Un URL poate fi privit ca un query la o baza de date MySQL. In cadrul exercitiului, veti construi doua URL-uri:

-><https://api.randomapi.com/16.4/questions?private_token=Ty151J88M&page=1>

-><https://api.randomapi.com/16.4/users?private_token=Ty151J88M&page=1>

In general, URL-urile pe care le folosim nu difera decat la nivel de cativa parametrii, asa ca e practic sa definim partile refolosibile drept constante stringuri la nivel global in clasa.

Vom folosi primul URL pentru a ne lua informatiile legate de intrebarile incarcate pe un forum. Prin al doilea URL, ne vom lua toate informatiile legate de utilizatorii de pe forum.

De fiecare data cand facem un apel si primim un fisier cu informatii, trebuie sa stim protocolul folosit in scrierea fisierului. Pentru asta, incercati sa cititi informatiile din obiectul de tip Response primit de la server.

(HINT: consola)

Un apel de tip HTTP poate intoarce diferite coduri care ne spun in ce masura a esuat sau a fost de succes query-ul. Exemple de coduri: **200 (Success)**, **404 (Not found).** Odata ce facem un apel la functia **makeHttpRequest(),** trebuie sa fim in stare sa interpretam raspunsul si sa luam diferite actiuni in functie de codul primit.

Codurile cu care lucram sunt:

* 200 – OK;
* 504 – Gateway Timeout
* 503 – Service Unavailable
* 408 – Request Timeout
* 406 – Protocol Misused

In caz ca primim codul 200, vrem sa parsam informatia mai departe. Pe de alta parte, in cazul celorlalte coduri, dorim doar sa afisam un mesaj corespunzator in consola (ex. No response, HTTP code is: … )

De asemenea, in cazul pasarii unui URL necorespunzator ca format, metoda va intoarce o exceptie de tipul **InvalidUrlFormatException.** Programul trebuie sa implementeze o modalitate de tratare a acestei exceptii, astfel ca, in cazul in care este ridicata, sa se afiseze un mesaj corespunzator in consola.

Gasiti definitia exceptiei in clasa NetworkUtils.

Atunci cand programul ruleaza, se alege la intamplare, prin functia **AutomaticServerRouterStatus()**, o configuratie Router-Server. Codurile HTTP vor fi in concordanta cu aceasta pereche (ex. Daca routerul si serverul sunt functionale, se returneaza cod 200). La final, programul trebuie sa fie in stare sa faca fata oricarei configuratii de stari. Pentru dezvoltare, exista functia **SetDefaultConfiguration()** prin care va puteti alege configuratia preferata by default. Ideal, pentru dezvoltare, puteti seta automat configuratia ideala (Router – TURNED ON, Server – UP) ca sa puteti lucra cu obiectele de tip Response).

In a doua parte, aveti de implementat o metoda de parsare a informatiilor primite de la server. Metoda de completat, **parseResponse(),** primeste un argument de tip **Response**.

Clasa **Response** din **NetworkUtils** are doua campuri: o variabila int **responseCode** care contine codul HTTP si un obiect de tipul **ResponseDetails**. Clasa ResponseDetails are urmatoarele campuri:

* **responseType**  : este de tipul USERS sau QUESTIONS (Hint: a se vedea enum Response\_Type) ;
* **usersCount** : in caz ca s-a facut apel la server pentru toti userii, variabila este nenula;
* **questionsCount**: analog, in caz ca s-a facut apel pentru toate intrebarile,
* **usersResponse**: in cazul apelului catre useri, acest string contine toate informatiile necesare parsarii ulterioare; (altfel e un string nul)
* **questionsResponse**: analog, ca in cazul userilor, petru intrebari.

**ResponseDetails** nu avea niciodata simultan stringurile **usersResponse** sau **questionsResponse** nenule.

La fel e valabil si pentru **usersCount** sau **questionsCount**.

Cele 2 stringuri, **usersResponse** si **questionsResponse** trebuie analizate ca format( fara a se interveni in implementarea clasei NetworkUtils! ) si pe baza observatiilor facute, implementata o functionalitate corespunzatoare de extragere a informatiilor.

Clasa **ParsedResponse** contine doua subclase : **User** si **Question**. Veti avea de implementat metodele, respective campurile acestor clase ca sa stocheze informatiile primite in mod corespunzator.

In cazul unui apel pentru intrebari, veti crea doar obiecte de tip Question, analog pentru utilizatori veti crea doar obiecte User.

Aveti doua functii deja definite ca semnatura de completat: queryServer() si parseResponse(). Functia queryServer() va intoarce intotdeauna un Response (asta daca nu e o exceptie ridicata), iar in cazul in care obiectul de tip Response are un cod de tip 200, doar atunci se va apelasi functia parseResponse().

Gasiti TODO-urile, precum si mai multe comentarii ajutatoare in cod. Exercitiile au fost gandite sa fie relative independente unu de celalalt, asa ca daca nu va iese legarea finala a tuturor functiilor, cel mai bine sa aveti macar corpurile si clasele definite.

Folositi-va atat de Internet, cat si de documentatia pusa deja la dispozitie in cod. Pentru depanari utilizati ori utilitarul de Debug din Visual Studio sau pentru erori subliniate, mergeti cu cursorul pe cuvantul subliniat cu rosu si apasati ALT+ENTER pentru diferite sugestii de rezolvare.

NU AVETI VOIE! Sa modificati clasa NetworkUtils. Folositi-va de functiile si clasele deja definite si ajutati-va cat se poate de documentatia pusa la dispozitie in comentarii.

Daca aveti intrebari, neclaritati, intrebati-ne!

Mult success!