Java labor segédlet

az

Alkalmazás fejlesztési környezetek c. tárgyhoz

készítette: Filep Szabolcs
2017.

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

Bevezetés

A labor célja, hogy megismertesse a hallgatókat a rendszerek közötti kommunikációs formák közül az egyik legelterjettebbel, a REST alapú kommunikációval.

A tárgy előadásai részletesen tárgyalják a különböző Java alapú keret-rendszereket és eszközöket és bemutatják a REST fogalmát és alapelveit.

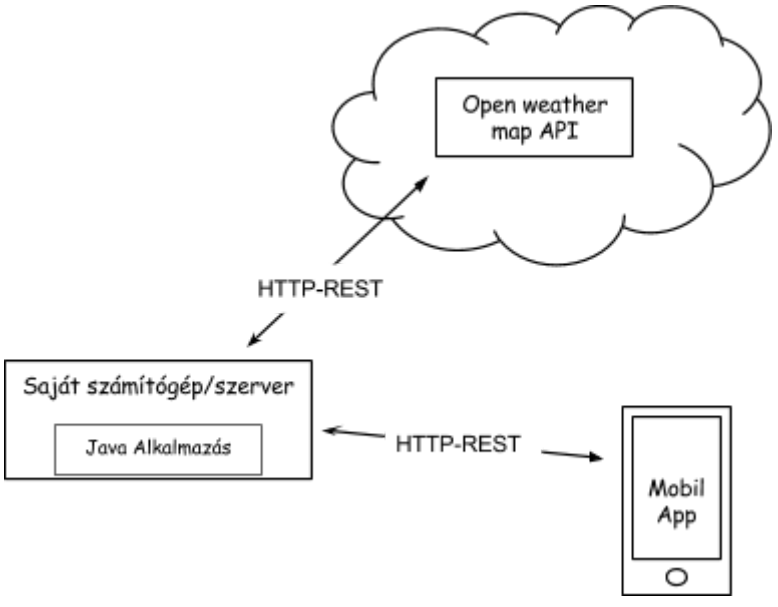
A labor témaválasztása mögötti szándék az, hogy bemutassuk, hogyan készíthetünk olyan alkalmazást, ami valós idejű külső adatokat képes feldolgozni és az eredményt elérhetővé tenni más rendszerek számára, pl. mobil alkalmazások számára.

A labor tartalma és menete:

1. Bevezetés és a megvalósítandó feladat rövid ismertetése
2. OpenWeatherAPI regisztráció (10 perc szükséges a regisztráció aktiváláshoz, ezért ezzel érdemes kezdeni).
3. Közös kutatómunka a lehetséges megoldásokról és időjárási adatforrásokról
4. Elméleti háttér rövid ismertetése (IoT, REST, HTTP)
5. Fejlesztési eszközök rövid ismertetése (Java, Eclipse, Maven, HttpComponents, Gson, REST kliensek (böngésző, Postman, Restlet stb.))
6. Projekt váz áttekintése
7. Önálló feladatok

A feladat

A feladat egy olyan alkalmazás készítése, ami képes megmondani, hogy jelenleg melyik városban van a leghidegebb és melyikben a legmelegebb az országban.



Kérdések:

- Milyen lehetséges megoldások jöhetnek szóba?
- Melyiknek milyen költség vonzata van?
- Mennyire megbízható az adat forrása?
- Megvalósítható lenne-e saját hőmérők üzembe helyezése a városokban?
- Ha megvan az eredmény, hogyan szolgáltassuk ezt a felhasználónak vagy más rendszereknek?

Projekt váz áttekintése

Projekt váz letöltése és importálása az Eclipse fejlesztő eszközbe.

JSON fájlok

1. Városok listája egy fájlban: current.city.list.json (18MB!)
    a. Ezt lehet frissíteni kézzel az openweathermap.org-ról
2. Példa egy város adatainak megtekintésére (city.example.json)
    a. hozzá tartozó osztály: City.java
3. Példa válasz az OpenWeatherMap API-tól (weather.example.json)
    a. hozzá tartozó osztályok: Weather.java, Main.java,

Maven

pom.xml - a Maven konfigurációs fájl

- Gson - a JSON - Java objektum mappeléshez
- HttpComponents a HTTP híváshoz

Java forráskód

MinMaxWeatherApplication.java

- a **main()** metódus példányosítja az osztályt és start metódussal indítja
- a **minMaxValues** példányváltózóban tárolja az aktuális minimum és maximum hőmérsékletet

CityReadFromJson.java

- a **current.city.list.json** fájl beolvasását végzi, méghozzá stream-elve, hogy ne kelljen egyszerre memóriába tölteni az egész fájl tartalmát (18MB!)
- A Gson ad támogatást JSON fájlformátum feldolgozására
- A **json package**-ben találhatók azok a POJO-k (Plain Old Java Object), amiket a Gson használ, hogy a JSON fájlból Java objektumokat készítsen (vagy fordítva). Ezekbe az osztályokba csak azok a mezők kerültek bele, amiket az alkalmazás

használ, de tetszőlegesen bővíthetők bármilyen mezővel, amelyek a json fájlokban előfordulhatnak.

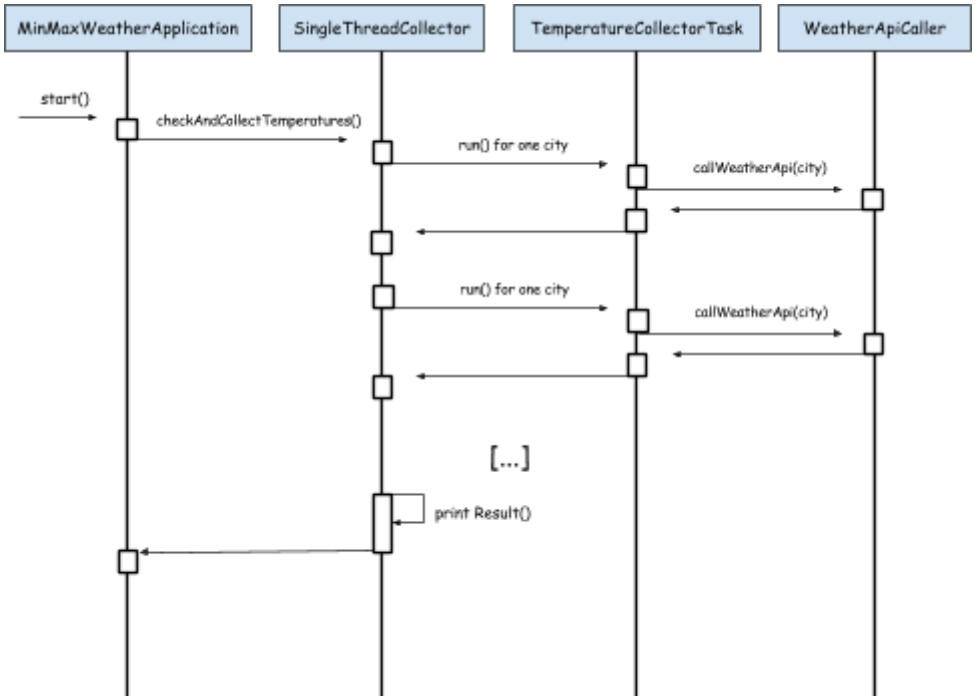
WeatherApiCaller.java:

- Ez az osztály felelős az OpenWeatherMap API currentWeather szolgáltatásának igénybevételéért: technikailag egy cityId-val és API-key kiegészített URL meghívásáért és a hőmérsékletet tartalmazó válasz objektum összeállításáért.
- Ha túl sokszor hívjuk, akkor előfordulhat, hogy 429-es hibát ad vissza az API, amit csak 1 napos várakozással, vagy új API kulcs regisztrációval oldhatunk fel.

TemperatureCollectorTask.java:

- Ez az osztály külön szálként is futtatható (implements Runnable). Feladata, hogy egy város hőmérsékletét összehasonlítsa az eddigi minimum és maximum hőmérsékletekkel és ha szükség, módosítsa azokat.

Szekvencia diagram az alkalmazás működéséről



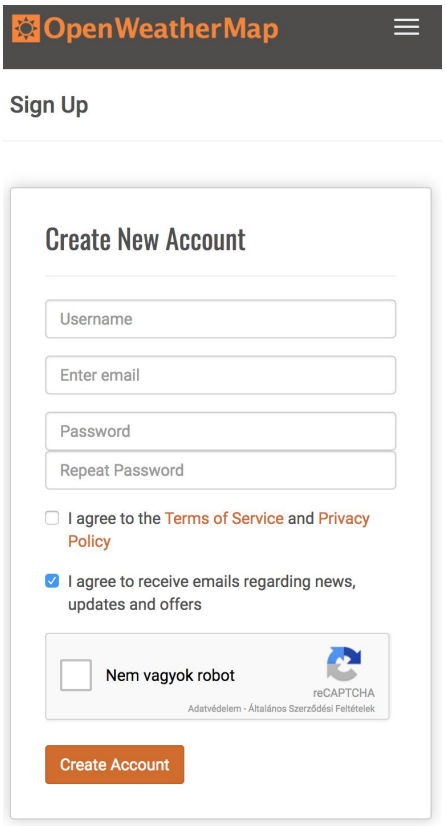
Feladatok

1. feladat: OpenWeatherMap.org regisztráció

Látogass el ide: https://home.openweathermap.org/

Sign up!

1. Regisztrálj be az oldalra!



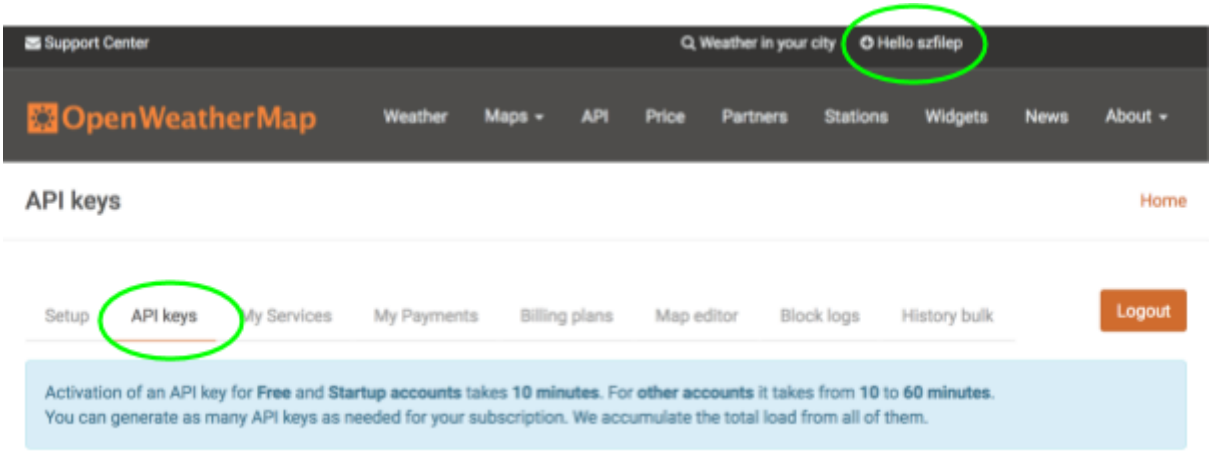
API keys

A fejlécben a névre kattintva tudod szerkeszteni az API kulcsaidat.

Az API kulcsok arra valók, hogy egy saját alkalmazásból is hozzáférhess az OpenWeatherMap szolgáltatásaihoz. Egy API kulcs jelen esetben például így néz ki:

```
d3cda285ea8c800ad9fb7264d687cb6a
```

Használhatod a *default* kulcsot vagy készíthetsz újat is.



1. Másold be ezt a kulcsot az **WeatherApiCaller** osztály megfelelő helyére:
   private final static String ***API_KEY*** = **"<Your API key comes here>"**;
2. Próbáld ki az API néhány hívását a https://openweathermap.org/current oldalon található minták alapján a böngésző segítségével

Kérdések

- Mit csinál a &mode=html és a &mode=xml paraméter, ha hozzáilleszted az URL-hez, amikor meghívod a szolgáltatást?
- Hogyan viszonyulnak ezek a különböző reprezentációk a REST név feloldásában a Representational State Transfer-ben szereplő a representational szóhoz? :)
- Mi a válasz tartalom alapértelmezett formátuma?

2. feladat: városok beolvasása

Add hozzá az alábbi kódrésztet a **MinMaxWeatherApplication** osztály **start()** metódusához, azért, hogy induláskor beolvassa a magyar városokat.

```
System.out.println("Start reading cities from file...");
List<City> cities = new CityReaderFromJson().readCitiesForCountry("hu");
```

Kérdések:

- Mennyi magyar várost talált?
- Összesen hány város van a current.city.list.json fájlban?

3. feladat: SingleThreadCollector elkészítése

Implementáld a **checkAndCollectTemperatures()** metódus belsejét. Ennek a metódusnak két feladata van:

1. A start() metódusból hívd meg a **collectInSingleThread()** metódust.
2. egy foreach segítségével, a paraméterben kapott **city** listán, minden városra hívjon meg egy **új TemperatureCollectorTask** példány **run**() metódusát.
3. ezek után írja ki a minMax értékét a consolra:
   System.*out*.println("**RESULT: \n**" + minMax);
4. Fejezd be a **TemperatureCollectorTask** osztály **setMinAndMax()** metódusát, ami a paraméterben kapott időjárási adattal frissíti a **minMax** objektum hőmérsékleti adatait, amennyiben hidegebb vagy melegebb várost talált. Figyelj rá, hogy a Float-okat compareTo() metódussal hasonlítsd össze:
   pl. `float1.compareTo(float2)`

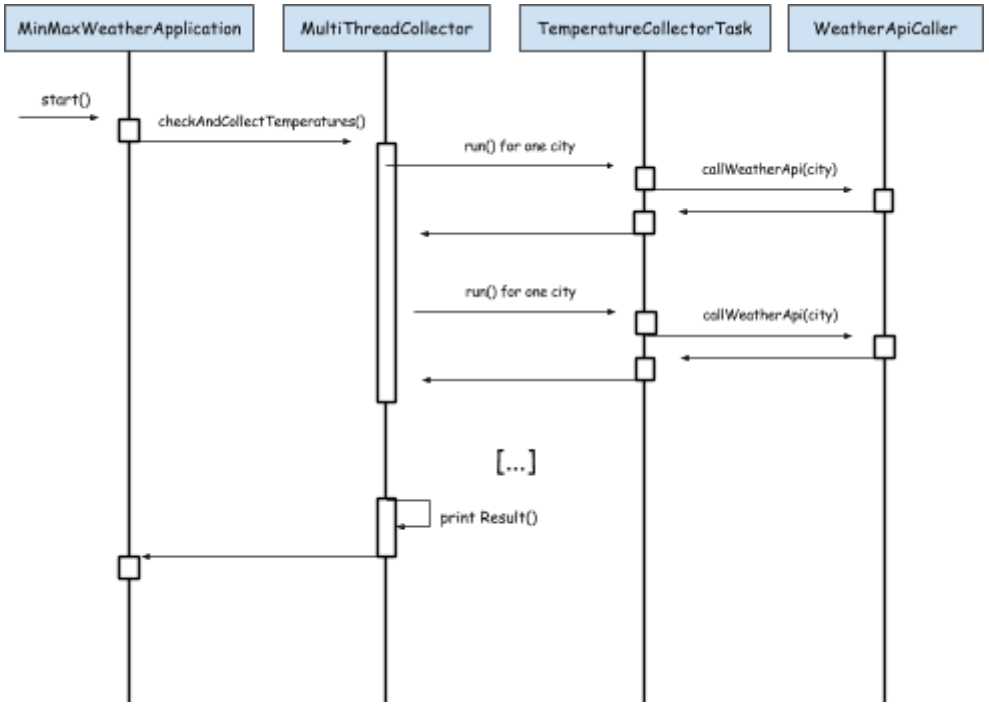
Kérdések:

- Mennyi ideig fut így az alkalmazás, ameddig kiszámolja a végeredményt?
- Mitől ilyen lassú, hogy több másodpercig is eltart?
- Hogyan lehetne gyorsítani a futási időt?

4. feladat: MultiThreadCollector elkészítése

Annak érdekében, hogy gyorsabban eredményt adjon az alkalmazás, fejezd be a a **MultiThreadCollector** osztály **checkAndCollectTemperatures()** metódusát:

1. A foreach belsejében ahelyett, hogy közvetlenül a TemperatureCollectorTask run() metódusát hívnád, használd az **ExecutorService**-t, ami a példányváltozó szinten már rendelkezésre áll. Ezzel párhuzamosítani lehet a task futtatásokat, és így a HTTP hívásokat.
2. Az MinMaxWeatherApplication osztályban készíts egy **collectInMultiThread()** metódust az előző alapján és írd át benne a 'one by one' szöveget 'parallely'-re és, hogy MultiThreadCollector-t használja.



Kérdések:

- Mennyivel gyorsult az alkalmazás?
- Mitől gyorsult így fel?
- Hogyan lehet szabályozni a párhuzamosan futtatott HTTP hívások számát az alkalmazásban?
- Figyeld meg a **finishAndPrintResult()** metódust! A jelenlegi beállítások szerint legfeljebb mennyi ideje van az alkalmazásnak, hogy minden OpenWeatherMap API hívás lefusson? (keresd meg a kódban a választ!)

5. feladat: saját REST API készítése

Mit tudunk tenni, ha mi is szeretnénk más rendszereket adatot szolgáltatni hasonló felületen? Először is szükségünk van egy HTTP szerverre, ami fogadja a kéréseket és válaszokat rá a HTTP protokollnak megfelelően.

A legelterjedtebb Java-s HTTP szerverek: Apache Tomcat, Jetty, Glassfish, Wildfly, WebLogic (utóbbiak már sokkal többet tudnak, mint egy sima HTTP szerver). A Java 6-os verziója óta azonban az Oracle JDK-ban is van egy beépített kis HTTP szerver, amit tudunk használni erre a célra.

Az alkalmazásban már van egy MinMaxHttpServer, ami a 8000-es porton figyel, ha elindítjuk. Ehhez kell elkészíteni a **RestHttpHandler handle()** metódusát, hogy tetszőleges hívás esetén JSON formátumban visszaadja a kiszámolt minimum és maximum hőmérsékleti adatok.

1. Készítsd el a **RestHttpHandler** osztály **handle()** metódusát. Használd a Gson osztályt az adatok JSON formátumúvá alakításához. Ha a böngészőből hívod, akkor az eredmény valami ilyesmi kell legyen:

```
{
 "minTempWeather": {
  "main": {
   "temp": 8.2
  },
  "id": 3046526,
  "name": "Pecs"
 },
 "maxTempWeather": {
  "main": {
   "temp": 11
  },
  "id": 717771,
  "name": "Mateszalka"
 }
}
```

2. Egészítsd ki ezt a metódust, hogy ha a requestURI egyenlő ‘/stop’, akkor állítsa le a HTTP szervert (httpExchange.getRequestURI().toString().equals(**"/stop"**)).
3. Próbáld ki az alkalmazást böngészőből.
4. Próbáld ki az alkalmazást Postman-ből (böngésző kiegészítő) is, ha ennek telepítése lehetséges.

...Itt a vége, további jó Java programozást! :)

