



Java labor segédlet

az

Alkalmazás fejlesztési környezetek c. tárgyhoz

készítette: Filep Szabolcs és Kövesdán Gábor
2017-2018.

laborvezető: Kövesdán Gábor
kovesdan.gabor@aut.bme.hu

Szerzői jogok

Jelen dokumentum a BME Villamosmérnöki és Informatikai Kar hallgatói számára készített elektronikus jegyzet. A dokumentumot az Alkalmazásfejlesztési környezetek c. tantárgyat felvevő hallgatók jogosultak használni, és saját céljukra 1 példányban kinyomtatni. A dokumentum módosítása, bármely eljárással részben vagy egészben történő másolása tilos, illetve csak a szerző előzetes engedélyével történhet.



Bevezetés

A labor célja, hogy megismertesse a hallgatókat a REST alapú kommunikációval.

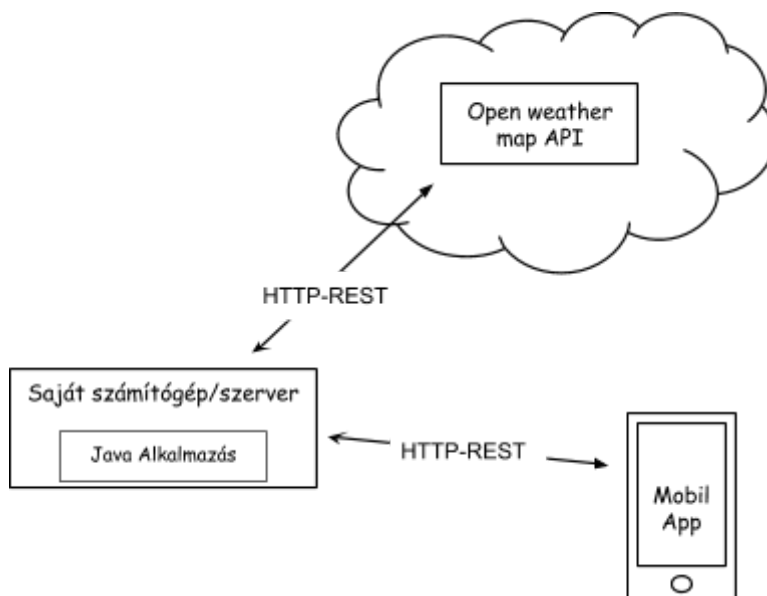
A labor témaválasztása mögötti szándék az, hogy bemutassuk, hogyan készíthetünk olyan alkalmazást, ami valós idejű külső adatokat képes feldolgozni és az eredményt elérhetővé tenni más rendszerek számára, pl. mobil alkalmazások számára.

A labor tartalma és menete:

1. A feladat rövid ismertetése
2. OpenWeatherAPI regisztráció (10 perc szükséges a regisztráció aktiváláshoz, ezért ezzel érdemes kezdeni).
3. Elméleti háttér rövid ismertetése (IoT, REST, HTTP)
4. Fejlesztési eszközök rövid ismertetése (Java, Eclipse, Maven, HttpComponents, Gson, REST kliensek (böngésző, Postman, Restlet stb.))
5. Projekt váz áttekintése
6. 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.



A hőmérsékleti adatokat az OpenWeatherService szolgáltatástól kapjuk meg REST API-n, és az eredményt mi is REST API-val szeretnénk közzétenni.

Projekt váz áttekintése

Projekt váz letöltése és importálása az Eclipse fejlesztő eszközbe.

JSON fájlok

Itt mintákat találunk rá, hogy milyen fájlokat kell majd feldolgoznunk, illetve előállítanunk.

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`** példányváltozóban tárolja az aktuális minimum és maximum hőmérsékletet

`CityReadFromJson.java`

- a **`current.city.list.json`** fájl beolvasását végzi, még hozzá stream-elve, hogy ne kelljen egyszerre memóriába tölteni az egész fájl tartalmát (18MB!)
- A Gson ad támogatást JSON fájlformátum feldolgozására
- A **`json package`**-ben található 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.

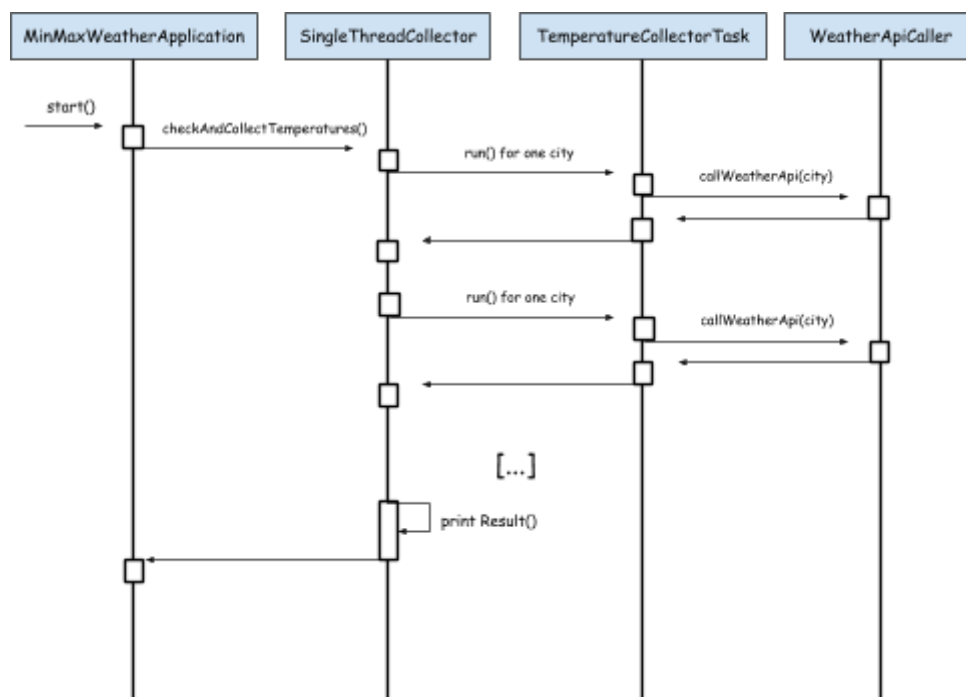
WeatherApiClient.java:

- 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!

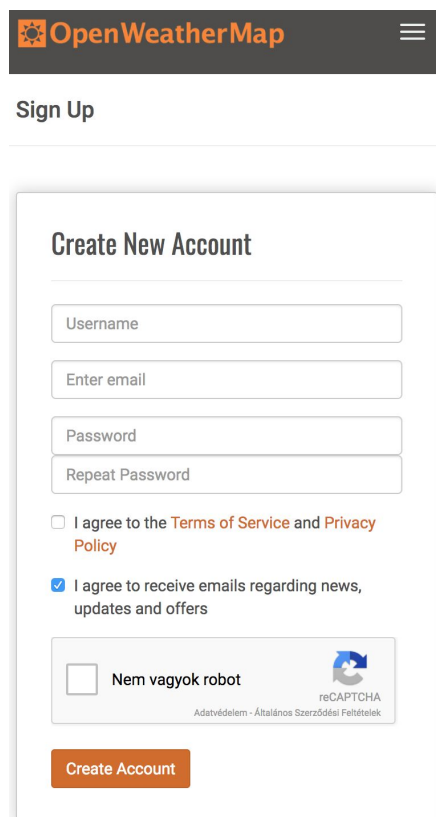
API keys

A fejlécben a névre kattintva tudod szerkeszteni az API kulcsaidat.

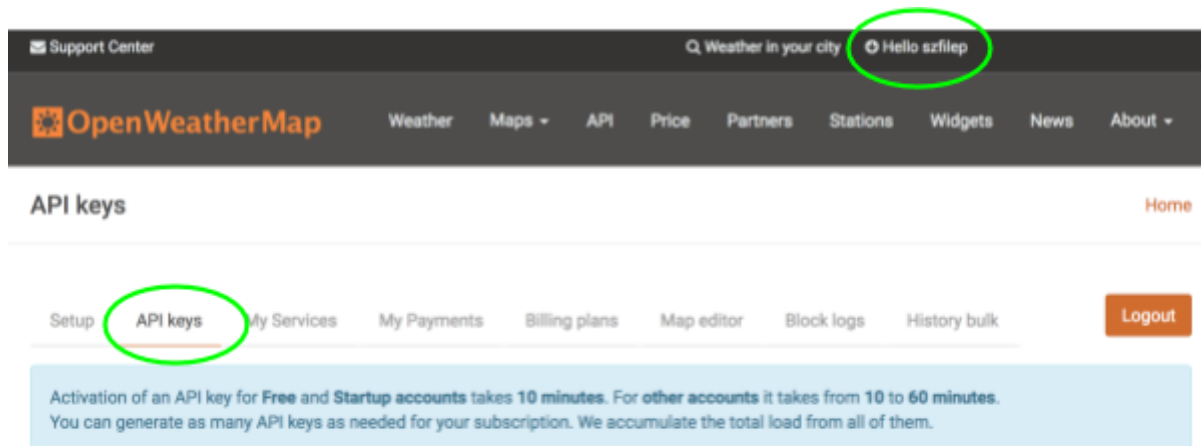
Az API kulcsok arra valók, hogy egy saját alkalmazásból is hozzáférhess az OpenWeatherMap szolgáltatásaihoz. Egy API kulcs jelen esetben például így néz ki:

d3cda285ea8c800ad9fb7264d687cb6a

Használhatod a *default* kulcsot vagy készíthetsz újat is.



The screenshot shows the 'Create New Account' form on the OpenWeatherMap website. It includes input fields for Username, Enter email, Password, and Repeat Password. There are two checkboxes: one for agreeing to the Terms of Service and Privacy Policy (unchecked), and another for agreeing to receive emails regarding news, updates, and offers (checked). A reCAPTCHA widget is present with the text 'Nem vagyok robot' and 'reCAPTCHA'. At the bottom, there is a 'Create Account' button.



The screenshot shows the 'API keys' page on the OpenWeatherMap website. The user's name 'Hello szfilep' is circled in green in the top navigation bar. In the main navigation bar, 'API keys' is also circled in green. Below the navigation bar, there is a light blue informational box with the following text: 'Activation of an API key for Free and Startup accounts takes 10 minutes. For other accounts it takes from 10 to 60 minutes. You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.'

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?
- 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");
```

Vizsgáld meg:

- 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)
```

Vizsgáld meg:

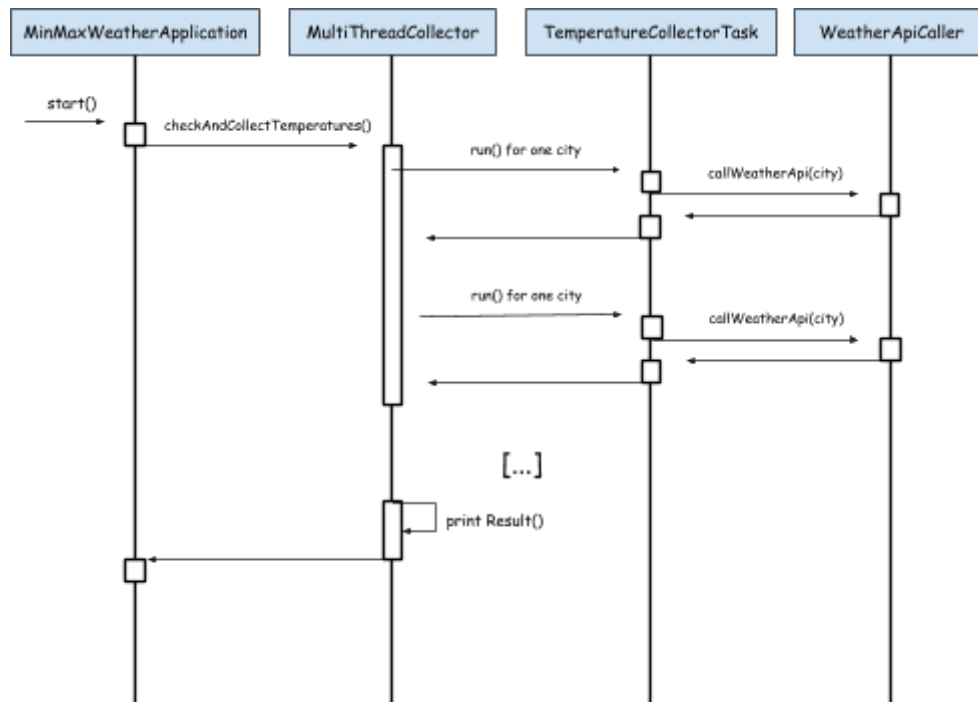
- Mennyi ideig fut így az alkalmazás, ameddig kiszámolja a végeredményt?
- Mitől ilyen lassú, hogy több másodpercig is eltart?
- Hogyan lehetne gyorsítani a futási időt?

4. feladat: MultiThreadCollector elkészítése

Annak érdekében, hogy gyorsabban eredményt adjon az alkalmazás, fejezd be a **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.



Vizsgáld meg:

- Mennyivel és mitől gyorsult az alkalmazás?
- 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 generál 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 szerveret (`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! :)