Programowanie w JAVA
Lab. 9 – REST API Client

1. Cel zadania: Udowodnić teorię sześciu stopni oddalenia
2. Szczegóły zadania:

"Według teorii sześciu stopni oddalenia każdego z nas dzieli tylko sześć kontaktów od dowolnie wybranej osoby na świecie"

Źródło: https://www.youtube.com/watch?v=lPXaHBkGUBI

PS. Wg najnowszych badań, wartość 6 może być zawyżona:
https://research.fb.com/three-and-a-half-degrees-of-separation/

Celem zadania jest powtórzenie eksperymentu Patricka Reynoldsa z 1999 (https://oracleofbacon.org/), który badał powiązania pomiędzy aktorami grającymi wspólnie w filmach. Jako dane wejściowe wykorzystaj bazę danych filmów i aktorów serwisu IMDB przygotowaną na potrzeby tego zadania w postaci REST API.

Opis API:

[GET] **https://java.kisim.eu.org/actors/{imdb}**
Zwraca dane aktora o podanym identyfikatorze {imdb},
np. https://java.kisim.eu.org/actors/nm0000102

[GET] **https://java.kisim.eu.org/actors/{imdb}/movies**
Zwraca filmy w których grał aktor o podanym identyfikatorze {imdb},
np. https://java.kisim.eu.org/actors/nm0000102/movies

[GET] **https://java.kisim.eu.org/movies/{imdb}**
Zwraca dane filmu o podanym identyfikatorze {imdb},
np. https://java.kisim.eu.org/movies/tt0117571

[GET] **https://java.kisim.eu.org/actors/search/{wzor}**
Zwraca listę aktorów których imię i nazwisko zawiera podany {wzór}, lista zwiera maksymalnie 20 wpisów, działa również na częściowych danych, wielkość liter nie ma znaczenia
np. https://java.kisim.eu.org/actors/search/brandon
https://java.kisim.eu.org/actors/search/brad pitt

W każdym przypadku API zwraca informację w formacie JSON.
Baza obejmuje tylko filmy, seriale i aktorów (bez reżyserów, monterów i innych ludzi kina).

Identyfikator IMDB znajdziesz w adresie strony profilowej danego aktora: https://www.imdb.com/name/**nm0000102** lub danego filmu: https://www.imdb.com/title/**tt0117571**

Przykładową implementację można oprzeć na grafie nieskierowanym, w którym wierzchołki stanowią aktorzy, a krawędź film - poprzez który się poznali.

3. Implementacja:
   a. Zaimplementuj aplikację pozwalającą na wyznaczenie najkrótszej ścieżki znajomości pomiędzy dwoma aktorami.
   b. Aplikacja powinna pozwolić na podanie nazwiska i imienia aktora i na ich podstawie pobrać z bazy jego ID z serwisu IMDB.
   Posiadając ID aktora A i ID aktora B należy zbudować graf połączeń zakładając, że aktor zna innego aktora jeśli wystąpili wspólnie w filmie. Wg. teorii, nie powinno być więcej niż 6 stopni (osób) łączących aktora A z aktorem B.
   c. Wykorzystaj bibliotekę `Jackson` do deserializacji danych z API.
   (https://www.baeldung.com/jackson-deserialization)
   d. Implementacje aplikacji najlepiej oprzeć na grafie nieskierowanym. Możesz napisać własną implementację grafu lub skorzystać z **JGraphT**.
4. Wskazówki
   a. Jeśli zdecydujesz się na użycie biblioteki JGraphT to pamiętaj aby:
      i. Klasa reprezentująca wierzchołek MUSI mieć przeładowane metody `equals` i `hashCode` inaczej może dojść do duplikacji wierzchołków reprezentujących tych samych aktorów. W

IntelliJ IDEA można te metody wygenerować za pomocą ALT+INSERT. Porównując obiekty weź pod uwagę co tak naprawdę je od siebie odróżnia. Czy może być kilku aktorów o tych samych imionach i nazwiskach? W omawianym przypadku tylko identyfikator IMDB jest unikalny i niepowtarzalny

ii. Klasa reprezentująca krawędź grafu MUSI dziedziczyć pod `org.jgrapht.graph.DefaultEdge`

iii. Dodanie do grafu takiej samej krawędzi dwukrotnie spowoduje nadpisane starej – łatwo rozwiązywane za pomocą metody `clone()`

iv. Zapoznaj się z dokumentacją http://jgrapht.org/javadoc/ oraz sposobem instalacji biblioteki http://jgrapht.org/

v. Przykładowy program wykorzystujący nieskierowany graf oraz algorytm znajdywania najkrótszej drogi Bellmana-Forda:

```
import org.jgrapht.Graph;
import org.jgrapht.GraphPath;
import org.jgrapht.alg.shortestpath.BellmanFordShortestPath;
import org.jgrapht.graph.SimpleGraph;

void runSimpleGraphExample(){
    Graph<Actor, Movie> g = new SimpleGraph<>(Movie.class);
    Actor a = new Actor("A1", "nm0001");
    Actor b = new Actor("A2", "nm0002");
    Actor c = new Actor("A3", "nm0003");
    Actor d = new Actor("A4", "nm0004");
    //Actor newA = new Actor("A1", "t0001");   // poeksperymentuj!

    g.addVertex(a);
    g.addVertex(b);
    g.addVertex(c);
    g.addVertex(d);
    //g.addVertex(newA);

    g.addEdge(a,b, new Movie("M1", ""));
    g.addEdge(b,c, new Movie("M2", ""));
    g.addEdge(b,d, new Movie("M3", ""));
    g.addEdge(d,a, new Movie("M4", ""));
    //g.addEdge(newA,c, new Movie("M5", ""));

    Set<Actor> vertices = g.vertexSet();

    BellmanFordShortestPath<Actor, Movie> bfsp = new BellmanFordShortestPath<>(g);
    GraphPath<Actor, Movie> shortestPath = bfsp.getPath(a,c);

    List<Movie> edges = shortestPath.getEdgeList();
    List<Actor> actors = shortestPath.getVertexList();

    for(int i = 0; i < actors.size(); ++i){
        if(i == actors.size()-1)
            System.out.print(actors.get(i));
        else
            System.out.print(actors.get(i) + " -> " + edges.get(i).toString()  + " -> ");
    }

}
```

vi. Do wychodzenia z wielu zagnieżdżonych pętli w pewnych sytuacjach możesz użyć tzw. nazwanych bloków (ang. named/labeled block): https://softwareengineering.stackexchange.com/questions/185944/java-labels-to-be-or-not-to-be Dobrze znać taką konstrukcję (odpowiednik owianego złą sławą goto z C++), ale nie należy jej nadużywać jeśli jest lepsze wyjście (np. refactoring kodu)!

vii. SimpleGraph nie pozwala na pętle

b. W zależności od implementacji w rozwiązaniu może przydać się kolejka:

```
Queue<Actor> actorsQueue = new ArrayDeque<>();
…
actorsQueue.add(new Actor());
Actor actor = actorsQueue.remove();
```

c. Projektując aplikację weź pod uwagę, że poszukiwania zabierają sporą długość czasu – wykorzystaj wątki!

d. Do zrealizowania połączenia HTTP wykorzystaj `OkHttpClient`: https://mvnrepository.com/artifact/com.squareup.okhttp3/okhttp

e. Postman (https://www.getpostman.com) oraz Insomnia (https://insomnia.rest/) to narzędzia pozwalające na wykonanie zapytań GET/POST do REST API w wygodniejszy sposób niż przeglądarka.

5. Przykłady do testowania własnej implementacji

a. **Boguslaw Linda** (nm0511277) -> Deserter's Gold (tt0191666) -> **Katarzyna Figura** (nm0276758)

b. **Boguslaw Linda** (nm0511277) -> Billboard (tt0424846) -> Andrzej Seweryn (nm0786614) -> Amok (tt0106263) -> Fanny Ardant (nm0000272) -> Episode dated 22 November 2008 (tt1338337) -> **Keanu Reeves** (nm0000206)

c. **Piotr Adamczyk** (nm0010611) -> Bezdroza serca i umyslu (tt0718677) -> Piotr Fronczewski (nm0296389) -> Moomins and the Winter Wonderland (tt6315872) -> Alicia Vikander (nm2539953) -> The Man from U.N.C.L.E. (tt1638355) -> **Henry Cavill** (nm0147147)

d. **Kit Harington** (nm3229685) -> Amanda Peet/Kit Harington (tt3759824) -> Amanda Peet (nm0001605) -> The Whole Ten Yards (tt0327247) -> **Bruce Willis** (nm0000246)

6. Teoria:
   a. Czym jest REST API?
   b. Format JSON

- **Ze względu na charakter zadania – jego prezentacja na zajęciach może być niewykonalna (ze względu na czas poszukiwania rozwiązania i ograniczenia techniczne), dlatego zalecam przygotować zrzuty ekranu z wynikami dla wybranych trzech par aktorów (inni niż w przykładach)!**

  … lub zaimplementować cache oparty o serializację wyników w plikach.

- Istnieje szansa, że połączenie pomiędzy aktorami może nie istnieć w bazie, ze względu na niekompletność danych udostępnionych bezpłatnie przez IMDB (https://www.imdb.com/interfaces/). Przykłady z punktu 6 zostały sprawdzone.
- Zwróć uwagę, że nie zawsze istnieje tylko jeden wynik (w grafie może być kilka ścieżek równorzędnie krótkich).

Po uzyskaniu zaliczenia na zajęciach, prześlij źródła w archiwum **zgodnie z konwencją nazewniczą** (patrz Lab0.pdf) do chmury na adres:
https://cloud.kisim.eu.org/s/5PJXWTywQX4cxMD
najpóźniej do następnych zajęć.