DSD Lab 8: RESTful requests with HttpClient (max: 12p)

In this lab you write Java apps that use the Apache HttpClient library to communicate with public RESTful APIs.
You do NOT need to create Spring beans today – just basic Java!

For each task, create a new Spring Legacy Project > Simple Spring Maven (or Maven Project) and add the following dependencies:

- Apache HttpClient
  - Group ID: org.apache.httpcomponents
  - Artifact ID: httpclient
  - Version: 4.5.1 (or greater)
- Google Gson
  - Group ID: com.google.code.gson
  - Artifact ID: gson
  - Version: 2.3.1 (or greater)

TIP: you can generate Java classes from JSON at http://www.jsonschema2pojo.org/

- Source type: JSON
- Annotation Style: Gson

TASK 1: REST Countries API (4p)

Write a Java application that sends GET requests to search the REST Countries API (http://www.restcountries.eu) for the following information:

- Portuguese-speaking countries (language code: *pt*)
- Countries in *eastern asia*
- Countries that use the US dollar (*usd*) as the currency

Print out the resulting country names for each case.

TIP 1: Use HTTP, not HTTPS!

TIP 2: If URL has a space character, use %20 instead! For example: "http://.../eastern%20asia"

TIP 3: To marshall/unmarshall a list of objects with Gson, you need to tell Gson what type of list it is. For example, if the base class for JSON data is called CountryData, then we have to manually define a Type (java.lang.reflect) object that corresponds a list of CountryData objects:

```
Type listType = new TypeToken<List<CountryData>>(){}.getType();
List<CountryData> data = new Gson().fromJson(json, listType);
```

TASK 2: OMDb API (4p)

Write a Java application that makes a GET request to the OMDb API (http://www.omdbapi.com/?). The application asks a movie name from the user and prints out the following information (or an error message if the movie is not found):

- Movie title
- Release date
- Genre
- Actors
- IMDB rating

TIP 1: If the movie name has a space character, replace them with %20 instead (use String's replace() method)
TIP 2: OMDb API returns just one JSON object, so you don't need to use a List.

Example:

```
Give a movie name:
Shawshank Redemption
Fetching data...
Name: The Shawshank Redemption
Release date: 14 Oct 1994
Genre: Crime, Drama
Actors: Tim Robbins, Morgan Freeman, Bob Gunton, William Sadler
IMDB rating: 9.3
```

TASK 3: POST & GET request to JSONPlaceholder API (4p)

Write a Java application that makes a GET and a POST request to JSONPlaceHolder API, which is a test environment for developers: http://jsonplaceholder.typicode.com/

In both cases, use Gson to marshall/unmarshall data between Java and JSON.

1. Make a GET request to get any of the sample posts
   - URL: http://jsonplaceholder.typicode.com/posts/10 ('10' is post ID)
   - Here is an example result:

```
{
  "userId": 1,
  "id": 10,
  "title": "optio molestias id quia eum",
  "body": "quo et expedita modi cum officia vel magni\ndoloribus qui repudiandae\nvero nisi sit\nquos
veniam quod sed accusamus veritatis error"
}
```

   - You must unmarshal this JSON into Java, and then print it out. To do this:
     1. Manually create a Java class that corresponds to the JSON string (see above)
     2. Unmarshal the JSON string into a Java object.
     3. Print out the details of the Java object. In case of an error, print an error message.

2. Make a POST request to send a new post to the API:
   - URL: http://jsonplaceholder.typicode.com/posts
   - Create a Java object, marshal it into JSON, and send it by POST. The API expects this JSON format (id can be null!):

```
{
    "title": "Test title",
    "body": "Test Body",
    "userId": 1
}
```

   - IMPORTANT: successful status code is 201, which means "created".
   - If request is successful, you will get new post in JSON as a result:

```
{
    "id": 101,
    "title": "Test title",
    "body": "Test Body",
    "userId": 1
}
```

This is basically the same than the original post, but “id” is added by the server when new post is created.

   - Unmarshall the result JSON into a Java object and print out its details. In case of an error, print an error message.

BONUS TASK 4: Connect to an API of your choice (2p)

Search the Internet for an open RESTful API. Then write a Java application that sends a request to the API and prints out the result. It should be an API that doesn't require registration so that TAs can easily test your solution.

NOTE: if you use an API that requires registration, then include your API-key in your source code.