DAM1 - Programación con JSON

Introducción

- JSON.simple - Quick Guide

JSON es un formato de texto sencillo para el intercambio de datos.

Se utiliza en muchos servicios web para acceder a bases de datos.

Prueba estas URLs:

- Ficha de un producto en OpenFoodFacts:
  - Vista web:https://world.openfoodfacts.org/product/8424523060914
  - JSON: https://world.openfoodfacts.org/api/v0/product/8424523060914.json
- Ruta en coche OpenStreetMaps:
  - Vista web: https://www.openstreetmap.org/directions?engine=fossgis_osrm_car&route=42.4283%2C-8.6444%3B42.3895%2C-8.7101
  - JSON: https://router.project-osrm.org/route/v1/driving/-8.634701,42.428181;-8.6444,42.3895?overview=false

Algunos Servicios Web JSON

Open Street Maps: un mapa del mundo, creado por gente como tú y de uso libre bajo una licencia abierta.

API Lugares: Nominatim Vista Web: Nominatim

APIs Rutas: Project OSRM, GraphHopper, Valhalla Docs

Ejemplos:
https://nominatim.openstreetmap.org/search?q=IES%20Chan%20Do%20Monte&format=geocodejson
https://nominatim.openstreetmap.org/reverse?format=geojson&lat=42.38972175&lon=-8.710682
https://router.project-osrm.org/route/v1/driving/-8.634701,42.428181;-8.6444,42.3895?overview=false

Open Food Facts: Base de datos de productos alimenticios elaborada por todos, para todos.

API: https://es.openfoodfacts.org/data

Ejemplo: https://world.openfoodfacts.org/api/v2/product/737628064502.json

Open Beauty Facts: A collaborative, free and open database of cosmetic products from around the world.

https://world.openbeautyfacts.org/data

Ejemplo: https://world.openbeautyfacts.org/api/v0/product/7798178623352.json

Meteogalicia: Datos e información meteorolóxica.

MeteoGalicia - RSS, GeoRSS e JSON

Ejemplo: https://servizos.meteogalicia.gal/mgrss/predicion/jsonCPrazo.action?dia=-1&request_locale=gl

Instituto Nacional de Estadística (INE)

Productos y Servicios / Datos abiertos / API JSON / Inicio

Otros

- Open Weather Map API
- Google Maps API
- NASA API
- The Movie Database
- …

Ejemplos de uso en Java

Para utilizar servicios web que ofrezcan información en JSON debemos hacer lo siguiente:

1. Construir la URL adecuada para hacer la petición.
2. Conectar al servicio y descargar la respuesta en forma de cadena de caracteres.
3. Procesar el resultado para obtener la información que nos interesa.

Prompt: Explícame como si fuera un estudiante de programación en Java como usar la api nominatin de OpenStreetMap para implementar una función que obtenga el país al que pertenece una ubicación geográfica dados sus valores de latitud y longitud.

1. Construir la URL

En este paso declaramos un String y mediante operaciones de concatenación le asignamos el valor adecuado según la documentación de la API a la que queremos acceder.

Ejemplo:
La siguiente URL de la API de Nominatim nos permite acceder a la información disponible en Open Street Maps sobre un punto geográfico cuya latitud y longitud se muestran resaltadas:

https://nominatim.openstreetmap.org/reverse?format=geojson&lat=42.38972175&lon=-8.710682

Si en un programa disponemos de los valores de latitud y longitud en sendas variables, podemos construir y almacenar la URL resultante en una variable de tipo String con un código como el siguiente:

```
double latitud = 42.38972175;
double longitud = -8.710682;
String url = "https://nominatim.openstreetmap.org/reverse?format=geojson&lat=" + latitud + "&lon=" + longitud;
```

2. Conectar al servicio y descargar la respuesta

Existen diferentes formas en Java para acceder a una URL y obtener la respuesta en forma de String.

Una opción es usar un método como el siguiente implementado en una clase de Utilidades:

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

public class Utilidades {

    public static String leerUrl(String url) {
        String contenido = "";

        // Configuración del proxy del sistema
        System.setProperty("java.net.useSystemProxies", "true");

        try {
            URI uri = new URI(url);
            URLConnection conexion = uri.toURL().openConnection();
            conexion.setRequestProperty("Accept-Language", "es");
            BufferedReader lector = new BufferedReader(
                    new InputStreamReader(conexion.getInputStream()));

            String linea;
            while ((linea = lector.readLine()) != null) {
                contenido += linea + "\n";
            }
            lector.close();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            System.out.println("URL mal formada: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error de E/S: " + e.getMessage());
        }
        return contenido;
    }

}
```

El código anterior utiliza algunas clases de los paquetes java.net y java.io relacionadas con operaciones de entrada/salida y manejo de URLs:

1. URL: La clase URL representa una ubicación en la World Wide Web. Encapsula el protocolo, el nombre de host, la ruta y otros detalles relevantes de la URL. En este código, se utiliza para crear un objeto URL a partir de una cadena de dirección web.
2. URLConnection: Es una clase abstracta que representa una conexión a un recurso a través de una URL. Se utiliza para establecer una conexión y configurar propiedades de la conexión, como el idioma en este caso.

3. MalformedURLException: Es una excepción específica para manejar errores relacionados con URLs mal formadas al intentar crear instancias de la clase URL.
4. InputStreamReader: Convierte bytes de un flujo de entrada en caracteres. En este código, se utiliza para adaptar el flujo de entrada desde una conexión URL a caracteres legibles.
5. BufferedReader: Permite una lectura de caracteres más eficiente desde un flujo de entrada, como un archivo o una conexión de red, al almacenar en búfer los caracteres leídos.
6. IOException: Es una excepción que maneja errores relacionados con operaciones de entrada/salida. En el contexto de este código, se utiliza para atrapar posibles errores durante la lectura de la URL.

Puedes utilizar el siguiente prompt para consultar a alguna IA como chatGPT más detalles sobre el funcionamiento del código anterior.

Prompt: Explica el funcionamiento del siguiente código Java y de las clases que utiliza:

[*CÓDIGO*]

El código anterior es mejorable y optimizable. Puedes investigar las clases StringBuilder, HttpURLConnection de la API de Java y también bibliotecas externas más completas, como Apache HttpClient o OkHttp.

Usando el proxy del sistema

El siguiente código configura el sistema para que nuestro programa utilice la configuración del proxy del sistema. Esto es útil si se está ejecutando en un entorno que requiere un proxy para acceder a Internet.

```
// Configuración del proxy del sistema
System.setProperty("java.net.useSystemProxies", "true");
```

3. Procesar la respuesta

Para procesar la respuesta JSON debemos familiarizarnos primero con su formato.

La consulta de la URL anterior devuelve el siguiente resultado en formato JSON:


```
{
    "type": "FeatureCollection",
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/copyright",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "place_id": 280605154,
                "osm_type": "way",
                "osm_id": 974440300,
                "place_rank": 30,
                "category": "amenity",
                "type": "school",
                "importance": 9.99999999995449e-06,
                "addresstype": "amenity",
                "name": "Instituto de Educación Secundaria Chan do Monte",
                "display_name": "Instituto de Educación Secundaria Chan do Monte, 23, Rúa Chan do Monte, A
Barriada, Mogor, Marín, Morrazo, Pontevedra, Galicia, 36911, España",
                "address": {
                    "amenity": "Instituto de Educación Secundaria Chan do Monte",
                    "house_number": "23",
                    "road": "Rúa Chan do Monte",
                    "hamlet": "A Barriada",
                    "borough": "Mogor",
                    "town": "Marín",
                    "county": "Morrazo",
                    "province": "Pontevedra",
                    "ISO3166-2-lvl6": "ES-PO",
                    "state": "Galicia",
                    "ISO3166-2-lvl4": "ES-GA",
                    "postcode": "36911",
                    "country": "España",
                    "country_code": "es"
                }
            },
            "bbox": [
                -8.7115201,
                42.3892082,
                -8.7095272,
                42.3902722
            ],
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -8.710682001062002,
                    42.38972175
                ]
            }
        }
    ]
}
```


En este caso se trata de un objeto en formato JSON que tiene una serie de atributos y valores, algunos de ellos a su vez son objetos anidados o arrays.

Una manera simple de procesar el resultado es tratarlo como una cadena de caracteres, buscando y extrayendo la información que nos interesa. Por ejemplo, el nombre del país al que pertenece el punto geográfico anterior corresponde al valor del atributo “country”. Podemos extraer esta subcadena con un código como el siguiente:

```
String url = ...
String json = Utilidades.leerUrl(url);
String country = null;
int index = json.indexOf("\"country\":");
if (index != -1) {
    int index2 = json.indexOf("\"", index + 11);
    country = json.substring(index + 11, index2);
}
```

Convertir Objetos a JSON y viceversa

- GitHub - **google/gson**: A Java serialization/deserialization library to convert Java Objects into JSON and back
  - Descargar .jar

1. Descarga y utiliza la librería Gson para:
   a. exportar el contenido de un array de personas a un String JSON
   b. importar un array en formato JSON a un array de objetos

```
import com.google.gson.Gson;

// Exportar Persona[] a JSON
        Gson gson = new Gson();
        Persona[] p = new Persona[];
        //…
        String json = gson.toJson(p);



// Importar JSON en Persona[]
        Gson gson = new Gson();
        Persona[] p;

        String json = ""; // contenido JSON;
        p = gson.fromJson(json, Persona[].class);
```

Ejemplos en Python

Ejemplo con Open Food Facts

Prueba este código en Python:

```
import urllib.request, json

print("# PRODUCTOS OPEN FOOD FACT")

codigo_barras = input("Introduce el código de barras [8424523060914]:")
if codigo_barras == "":
    codigo_barras = '8424523060914'

url_json =
'https://world.openfoodfacts.org/api/v0/product/'+codigo_barras+'.json'

resp = urllib.request.urlopen(url_json)
data = json.loads(resp.read())
print('Producto:',data['product']['generic_name'])
print('Marca:',data['product']['brands'],'\n\n')
```

Ejercicio: Sabrías implementar una función que imprima el nombre y la marca de un producto a partir de su código de barras?

Ejemplo con Open Street Map

```
import urllib.request, json

print("# RUTA OPEN STREET MAP")

latitud_inicio = 42.428181
longitud_inicio = -8.634701
latitud_fin = 42.389456
longitud_fin = -8.710098


tipo_ruta = 'driving'
url_json = 'https://router.project-osrm.org/route/v1/'+tipo_ruta+'/'
url_json += str(longitud_inicio)+','+str(latitud_inicio)+';'
url_json += str(longitud_fin)+','+str(latitud_fin)+'?overview=false'

resp = urllib.request.urlopen(url_json)
data = json.loads(resp.read())

print(data['routes'][0]['distance'],"metros")

# Imprime 10710.5 metros

url_web = 'https://www.openstreetmap.org/directions?engine=fossgis_osrm_car&route='
url_web += str(latitud_inicio)+','+str(longitud_inicio)+';'
url_web += str(latitud_fin)+','+str(longitud_fin)
print('url web:',url_web)

# Imprime la URL de la web con la ruta cargada para visualizar
```

Ejercicio: Sabrías implementar una función que devuelve la distancia en entre dos puntos en coche?