Consumir Api Rest Java

Pequeña guiá paso a paso para generar una aplicación que consuma los recursos desde un servidor API REST. El reto aquí es implementarlo en Java Swing (Interfaz usuario) usando lo que aquí presento con el server de api-pokemon-json:

Contenido

Dependencias

> Recordar crear un proyecto en Maven para el uso de dependencias. crear etiqueta `<dependencies></dependencies>` en el archivo pom.xml

```
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jaxrs</artifactId>
    <version>2.2.1.GA</version>
</dependency>
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jackson-provider</artifactId>
    <version>2.2.1.GA</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.6.3</version>
</dependency>
<dependency>
    <groupId>com.mashape.unirest</groupId>
    <artifactId>unirest-java</artifactId>
    <version>1.4.9</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
```

```
    <artifactId>httpclient</artifactId>
    <version>4.3.6</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpasyncclient</artifactId>
    <version>4.0.2</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpmime</artifactId>
    <version>4.3.6</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20140107</version>
</dependency>
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.3.1</version>
</dependency>
```

> Correr sobre el proyecto un Clean and Build (limpiar y construir)

Modelo Pokemon

Crear un java class Pokemon, con los atributos siguientes para poder manejar objetos desde java:

```
    private int id;
    private String nombre;
    private int generacion;
    private double peso;
    private double altura;
    private String habilidad;
```

Crear 3 constructores:

- Constructor vació.
- Constructor con todos los atributos menos id.
- Constructor con todos los atributos.

Crear Getters y Setters para todos los atributos del modelo Pokemon.

Metodo Get

Para obtener datos de una **API REST**, hay que usar el verbo de **HTTP Get**, generamos una solicitud al servidor, este lo devolverá en formato JSON el cual transformamos a String, que luego mapeamos para generar una lista de objetos pokemon. Primero importamos librerias:

```
import java.io.IOException;
import org.apache.http.client.ClientProtocolException;
import org.jboss.resteasy.client.ClientRequest;
import org.jboss.resteasy.client.ClientResponse;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.core.type.TypeReference;
import java.util.List;
```

Luego dentro de un try-catch, realizaremos la solicitud y transformaremos a una lista de objetos para utilizarla como queramos, en el siguiente código la lista la iteramos con `for` e imprimimos por consola la salida de datos.

```
try {
    ClientRequest request = new ClientRequest("https://api-pokemon-
json.herokuapp.com/api/v1/pokemons");
    request.accept("application/json");

    ClientResponse<String> response = request.get(String.class);
    String output= response.getEntity();

    ObjectMapper mapper = new ObjectMapper();
    List<Pokemon> list = mapper.readValue(output, new
TypeReference<List<Pokemon>>(){});

    for (int i = 0; i < list.size(); i++) {
        Pokemon poke = list.get(i);
        System.out.println(poke.getId()+" / "+poke.getNombre());
    }
} catch (ClientProtocolException e) {
            e.printStackTrace();
} catch (IOException e) {
            e.printStackTrace();
} catch (Exception e) {
            e.printStackTrace();
}
```

Metodo Post

Para poder enviar datos y que el servidor **API REST** lo guarde en su base de datos, hay que usar el verbo **HTTP Post**, primero importamos las librerías que usaremos:

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.google.gson.Gson;
```

Instanciamos un nuevo objeto Pokemon con los datos que queremos guardar, lo transformaremos con **Gson** a un **String** con formato json. Usaremos **Unirest.post()** para enviar a la url de nuestra API, un body con el json generado.

```
Pokemon poke = new Pokemon("stantler",2,71.20,1.4,"Cacheo");
Gson gson = new Gson();
String json = gson.toJson(poke);

try {
    HttpResponse<JsonNode> jsonResponse = Unirest.post("https://api-pokemon-
json.herokuapp.com/api/v1/pokemons")
        .header("accept", "application/json")
        .header("Content-Type", "application/json")
        .body(json)
        .asJson();
     System.out.println(jsonResponse.getStatus());

} catch (UnirestException ex) {
    Logger.getLogger(post.class.getName()).log(Level.SEVERE, null, ex);
}
```

> STATUS 201_: Significa que el registro fue creado satisfactoriamente.

Metodo Put

Para poder modificar un registro primero debemos obtener o saber su **id**, el servidor lo exigirá por medio de la **Url** para saber cual registró vamos a modificar, debemos usar el verbo **HTTP Put**. Primero importamos las librerías que usaremos:

```
import com.google.gson.Gson;
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.JsonNode;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
import java.util.logging.Level;
import java.util.logging.Logger;
```

Almacenamos en una variable el **id** del registro que modificaremos, generamos un nuevo objeto Pokemon con los datos que queremos modificar (lo que no, los dejamos igual), lo transformaremos con Gson a un String con formato json. Usaremos **Unirest.put** para enviar a la url con un id, un body con el json generado.

> url+id = dominio. com/api/version/controlador/id

```
int id = 11;
Pokemon poke = new Pokemon("pidgeot",1,39.5,1.5,"Tumbos");
Gson gson = new Gson();
String json = gson.toJson(poke);


try {
    HttpResponse<JsonNode> jsonResponse = Unirest.put("https://api-pokemon-
json.herokuapp.com/api/v1/pokemons/"+id)
        .header("accept", "application/json")
        .header("Content-Type", "application/json")
        .body(json)
        .asJson();
     System.out.println(jsonResponse.getStatus());

} catch (UnirestException ex) {
    Logger.getLogger(put.class.getName()).log(Level.SEVERE, null, ex);
}
```

> STATUS 200: el registro fue actualizado correctamente
>
> STATUS 400: no hay registro con el id suministrado

Metodo Delete

Para eliminar un registro es necesario obtener o saber su **id**, el servidor lo exigirá por medio de la Url para saber cual registró vamos a eliminar, además usar el verbo de **HTTP Delete**. Primero importamos las librerías que usaremos:

```
import com.mashape.unirest.http.HttpResponse;
import com.mashape.unirest.http.Unirest;
import com.mashape.unirest.http.exceptions.UnirestException;
import java.util.logging.Level;
import java.util.logging.Logger;
```

Almacenamos en una variable el id del registro que eliminaremos, usando **Unirest.delete** enviaremos al servidor una url + id, con el método **.asString();** la respuesta del servidor la almacenaremos en un objeto **HttpResponse**, el cual consultaremos el **status** de la solicitud.

```
int id = 14;
try{
    HttpResponse<String> response = Unirest.delete("https://api-pokemon-
json.herokuapp.com/api/v1/pokemons/"+id).asString();
    System.out.println(response.getStatus());
} catch (UnirestException ex) {
    Logger.getLogger(postpost.class.getName()).log(Level.SEVERE, null, ex);
}
```

> STATUS 204: el registro fue eliminado correctamente
>
> STATUS 400: el registro no se pudo eliminar o no existe registro con el id suministrado