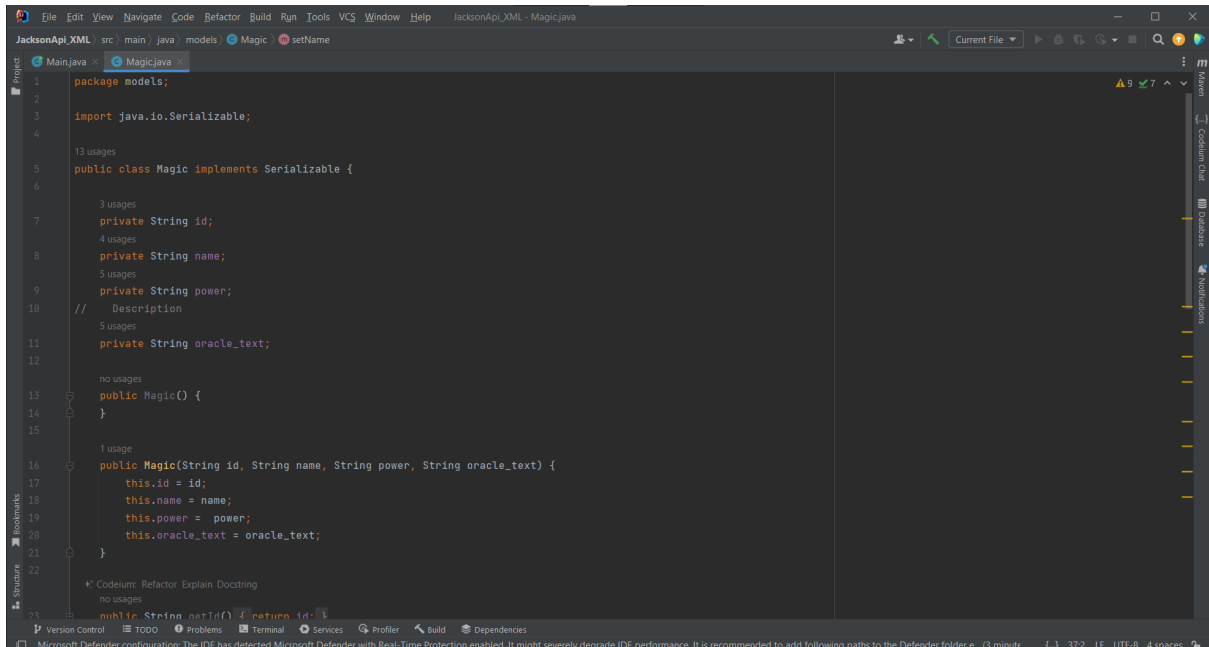


# LIBRERÍA JACKSON

## Introducción

Vamos a utilizar la api de Magic, para ello crearemos la clase de acuerdo a nuestras necesidades. La clase en cuestión sería como se muestra a continuación añadiendo sus getters y setters.



```
1 package models;
2
3 import java.io.Serializable;
4
5 13 usages
6 public class Magic implements Serializable {
7
8     3 usages
9     private String id;
10    4 usages
11    private String name;
12    5 usages
13    private String power;
14    // Description
15    5 usages
16    private String oracle_text;
17
18    no usages
19    public Magic() {
20    }
21
22    1 usage
23    public Magic(String id, String name, String power, String oracle_text) {
24        this.id = id;
25        this.name = name;
26        this.power = power;
27        this.oracle_text = oracle_text;
28    }
29
30    no usages
31    public String getId() { return id; }
```

En la clase añadiremos un implements serializable para realizar pruebas con listado de objetos.

Las dependencias añadidas para este proyecto son las siguientes:



```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.4.1</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.json/json -->
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20230227</version>
  </dependency>
</dependencies>
```

La primera es la dependencia de jackson, la cual nos proporcionará la clase ObjectMapper(nos ayudará a leer y escribir nuestro json) y la segunda necesaria para los archivos json.

Tendremos las rutas de los archivos json diferenciados por un solo objeto (una carta) o un listado de objetos (varias cartas)

```
//Manejando el fichero
File cardFile = new File( pathname: "src/main/resources/card.json");
File cardListFile = new File( pathname: "src/main/resources/cardList.json");
//Si no existen los ficheros los creamos
Utility.checkFile(cardFile);
Utility.checkFileComplete(cardListFile);
```

Si esos ficheros no existen los crearemos con los siguientes métodos:

```
public static void checkFile(File file) {
    if (!file.exists()) {
        try {
            file.createNewFile();
        } catch (IOException e) {
            //Cerramos el programa
            System.out.println("Error al crear el fichero");
            System.exit( status: 0);
        }
    }
}
```

```
1 usage
public static void checkFileComplete(File file) {
    if (!file.exists()) {
        try {
            file.createNewFile();
            BufferedWriter bf = new BufferedWriter(new FileWriter(file));
            bf.write( str: "[ ]");
            bf.close();
        } catch (IOException e) {
            //Cerramos el programa
            System.out.println("Error al crear el fichero");
            System.exit( status: 0);
        }
    }
}
```

El menú de esta aplicación es el siguiente:

```
=====
1. Leer fichero (Magic)
2. Escribir fichero (Magic)
3. Leer fichero completo (Magic)
4. Escribir fichero completo (Magic)
5. Salir
=====
Elige una opción:
```

## Leer un solo objeto en json

En el primer punto de nuestro menú, procederemos a leer un archivo que contiene un único objeto. Para lograrlo, emplearemos la biblioteca Jackson, donde instanciaremos un `ObjectMapper`. Este `ObjectMapper` nos permitirá tanto la lectura como la escritura en formato JSON, ya que el archivo en cuestión consta de un solo objeto. Posteriormente, mostraremos dicho objeto en pantalla. En este caso, utilizaremos el método "`readValue()`" para leer el JSON, dado que el archivo contiene objetos en este formato.

```
private static void readJsonFileMagic(File file) {

    Utilizando la librería jackson
    ObjectMapper mapper = new ObjectMapper();

    Leyendo y mostrando el fichero
    try {
        Obteniendo el objeto persona del fichero
        Magic magic = mapper.readValue(file, Magic.class);
        System.out.println(magic);
    } catch (IOException e) {
        System.out.println("Error al leer el fichero");
    }
}
```

## Escribir un solo objeto en json

Este sería el segundo punto de nuestro menú, para ello le pediremos al usuario que diga el nombre de la carta que desea buscar y traeremos los datos desde nuestro endpoint.

```
private static List<Magic> getMagics(String search) {
    try {
        URL url = new URL( spec: URL_BASE+search);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.connect();
        StringBuilder infoString = null;

        int responseCode = conn.getResponseCode();
        if (responseCode != 200) {
            System.out.println("Ha ocurrido un error " + responseCode);
        } else {
            infoString = new StringBuilder();
            Scanner scanner = new Scanner(url.openStream());
            while (scanner.hasNext()) {
                infoString.append(scanner.nextLine());
            }
            scanner.close();

            JSONObject jsonResponse = new JSONObject(infoString.toString());

            JSONArray jsonArray = jsonResponse.getJSONArray( key: "data");
            List<Magic> magics = new ArrayList<>();
        }
    }
}
```

```

// Iterate through the JSON array and create Magic objects
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject card = jsonArray.getJSONObject(i);

    String power;
    String oracle_text;
    try {
        power = card.getString( key: "power");
    } catch (Exception e) {
        power = "No tiene poder";
    }
    try {
        oracle_text = card.getString( key: "oracle_text");
    } catch (Exception e) {
        oracle_text = "No tiene descripción";
    }
    // Create a Magic object and populate its fields
    Magic magic = new Magic(card.getString( key: "id"), card.getString( key: "name"),
        power, oracle_text);

    // Add the Magic object to the list
    magics.add(magic);
}
return magics;

} catch (Exception e) {
    System.out.println("Ha habido un problema al obtener las Magic");
}

return new ArrayList<>();
}

```

Una vez extraídos los datos del endpoint escribiremos un objeto usando el método “writeValue()” de la clase ObjectMapper para escribir en formato json, en el caso de que en la búsqueda haya más de un objeto se escribirá uno aleatorio.

```

private static void writeJsonFileMagic(File file) {
    // Una nueva Magic para escribir en el fichero
    // Tomamos el nombre por teclado
    String search= Utility.readString( s: "Dime el nombre de las cartas que deseas buscar: ");
    List<Magic> magics = getMagics(search);
    Magic magic = magics.get(numRandom(magics.size()));

    // Utilizando la librería jackson para escribir el fichero
    ObjectMapper mapper = new ObjectMapper();

    // Escribiendo en el fichero
    try {
        mapper.writeValue(file, magic);
    } catch (IOException e) {
        System.out.println("Error al escribir en el fichero");
    }
}

```

```

{
  "id": "64bcc06a-de86-4387-882d-ead33e9c9e01",
  "name": "Wild Jhovall",
  "power": "",
  "oracle_text": ""
}

```

## Leer una lista de objetos en json (Deserialización de listas)

Como hicimos anteriormente para leer solo que en este caso utilizaremos el método `constructCollectionType`, el cual nos permitirá construir una lista de `Magics` en nuestro caso, después mostraremos todas las cartas.

```

private static void readJsonFileMagicComplete(File file) {
    // Utilizando la librería jackson
    ObjectMapper mapper = new ObjectMapper();

    // Leyendo y mostrando el fichero
    try {
        // Obteniendo los objetos persona del fichero
        List<Magic> magics = mapper.readValue(file, mapper.getTypeFactory().constructCollectionType(List.class, Magic.class));
        magics.forEach(System.out::println);
    } catch (IOException e) {
        System.out.println("Error al leer el fichero");
    }
}

```

## Escribir una lista de objetos en json (Serialización de listas)

En este caso lo que vamos a hacer es escribir una lista de objetos que nos vendrá del endpoint anteriormente mencionado. Primero leeremos nuestro fichero json por si existe alguna carta previamente y las guardaremos en una lista, después añadiremos a esa lista las nuevas cartas. Posteriormente escribiremos en nuestro fichero todas las cartas.

```
private static void writeJsonFileMagicComplete(File file) {
    // Crear una nueva persona para agregar al archivo

    String search = Utility.readString(s: "Dime el nombre de la carta que desea buscar: ");
    List<Magic> magics = getMagics(search);

    // Utilizando la librería Jackson para escribir el fichero sin borrar los datos
    ObjectMapper mapper = new ObjectMapper();

    try {
        // Leer el contenido actual del archivo (si existe)
        List<Magic> magicsDisk;
        magicsDisk = mapper.readValue(file, new TypeReference<>() {
        });
        magicsDisk.addAll(magics);
        // Escribir la lista actualizada en el archivo
        mapper.writeValue(file, magicsDisk);
    } catch (IOException e) {
        System.out.println("Error al escribir en el fichero");
    }
}
```

```
[
  {
    "id": "dd911980-1f5f-4420-96b8-1e4fe67e59f6",
    "name": "Jhoira, Ageless Innovator",
    "power": "2",
    "oracle_text": "{T}: Put two ingenuity counters on Jhoira, Ageless Innovator, then you may put an artifact card with mana value X",
  },
  {
    "id": "aa6c3c8a-aa35-46b3-8769-27b4244457b0",
    "name": "Jhoira of the Shitu",
    "power": "2",
    "oracle_text": "{2}, Exile a nonland card from your hand: Put four time counters on the exiled card. If it doesn't have suspend,",
  },
  {
    "id": "7b89a074-9aca-4f7f-953a-b401199be1cb",
    "name": "Jhoira's Familiar",
    "power": "2",
    "oracle_text": "Flying\nHistoric spells you cast cost {1} less to cast. (Artifacts, legendaries, and Sagas are historic.)",
  },
  {
    "id": "35d14746-3b57-4f8e-8155-e578cc84850d",
    "name": "Jhoira's Timebug",
    "power": "1",
    "oracle_text": "{T}: Choose target permanent you control or suspended card you own. If that permanent or card has a time counter",
  },
  {
    "id": "edb38309-c02c-496c-894f-786a2f6e3d1c",
    "name": "Jhoira's Toolbox",
    "power": "1",
    "oracle_text": "{2}: Regenerate target artifact creature.",
  }
]
```

Miguel Ángel Fernández y Adrián Lara Bonilla