



# TRABAJO 1 DE EVALUACIÓN CONTINUA

Generación Automática de Código

Ismael Calvo Villalvilla

51689141E  
alaricoi@gmail.com

# Contenido

1	INTRODUCCIÓN .....	2
2	DESARROLLO DE LA PRÁCTICA .....	3
2.1	Recursos de programación utilizados.....	3
2.2	Descripción (especificación) detallada de la solución.....	3
2.3	Alcance y limitaciones de la solución propuesta.....	5
2.4	Descripción de los casos de Prueba.....	6
3	UBICACIÓN DE LOS PROGRAMAS FUENTE Y EXPLICACIÓN DEL RESTO DE DIRECTORIOS/FICHEROS ANEXADOS .....	8
4	MANUALES DE USO/INSTALACIÓN .....	9
5	REFERENCIAS.....	11

## 1 Introducción

En el marco de la asignatura de Generación Automática de Código del máster universitario Investigación en Ingeniería de Software y Sistemas Informáticos el presente documento describe la solución para resolver el trabajo 1, este trabajo consiste en construir un generador que transforme un fichero JSON en un fichero XML equivalente.

Primeramente, describiremos los recursos empleados para llevar a cabo el trabajo, así como el lenguaje de programación seleccionado, posteriormente se realizará una descripción detallada de la solución incluyendo las pruebas realizadas. Por último, se darán las indicaciones oportunas para su instalación y uso para acabar con la conclusión.

Antes de comenzar vamos a realizar una pequeña descripción de los formatos tanto de entrada como de salida que tendrá nuestro programa

**JSON** (JavaScript Object Notation), según la página [json.org](https://json.org), *es un formato ligero de intercambio de datos*, fácil tanto para las personas como para los programas de leer y comprender, presenta un formato independiente del lenguaje, pero utiliza convenciones que son familiares para C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen de JSON un lenguaje de intercambio de datos ideal. Se trata de una **colección de pares de nombre/valor**, comienza con llave izquierda, {, y termina con Llave derecha, }. A cada nombre le sigue dos puntos, y los pares nombre/valor están separados por coma.

```
{
  "markers": [
    {
      "name": "Rixos The Palm Dubai",
      "position": [25.1212, 55.1535],
    },
    {
      "name": "Grand Hyatt",
      "location": [25.2285, 55.3273]
    }
  ]
}
```

**XML** (eXtensible Markup Language) es un meta-lenguaje que permite definir marca, independiente del lenguaje de programación, al igual que JSON, para almacenar y transportar datos.

```
<markers>
  <name>Rixos The Palm Dubai</name>
  <position>25.1212</position>
  <position>55.1535</position>
</markers>
<markers>
  <name>Grand Hyatt</name>
  <location>25.2285</location>
  <location>55.3273</location>
</markers>
```

## 2 Desarrollo de la práctica

### 2.1 Recursos de programación utilizados

Dada la libertad de elección por parte del equipo docente a la hora de elegir lenguaje de progresión y recursos necesarios, el lenguaje elegido para el desarrollo es java (jdk 1.8), a pesar de que sea Ruby el lenguaje propuesto por el equipo docente y por el autor del libro *Code Generation in Action*. Esta elección se ha basado principalmente en los conocimientos previos que tengo del entorno y la comodidad a la hora de gestionar ficheros de marcar, existen una gran variedad de soluciones de terceros que nos permiten acceder fácilmente a los lenguajes tanto JON como XML y transformarlos en objetos fácilmente tratable. Una vez definido el lenguaje vamos a describir las herramientas empleadas:

- Sistema operativo Windows 10 64 bit.
- Como IDE (Integrated Development Environment) he utilizado Eclipse Oxygen en su instalación de Java.
- Librerías java de terceros:
  - librería de SWT (Standard Widget Toolkit), de eclipse y openSource, nos permite crear de aplicaciones de ventanas y que crea la interfaz de usuario a partir de componentes operativos de cada Sistema operativo.
  - Log4j: Para realizar logs y trazas de la aplicación
  - JSON: librería openSource de json.org que nos permite parsear y tratar ficheros como formato json y transfórmalos a objetos. Esta librería también nos permite tratar formatos XML.

### 2.2 Descripción (especificación) detallada de la solución

Se ha realizado una aplicación de escritorio en entorno gráfico que de manera sencilla nos va a permitir seleccionar un fichero de entrada y un fichero de salida. La solución se apoya en la librería **json.org** que nos va a permitir fácilmente pasar de formato texto a formato objeto.

Todo el código está contenido en la clase `t1APP`, el entorno gráfico se crea a partir de la librería SWT, a la hora de arrancar la aplicación se llama al método **open** que se encargará de llamar al método de creación del contenido y a levantar el layout.

El método de creación de contenido, **createContents**, se encarga de crear los objetos visuales y posicionarlos dentro de la pantalla. También definiéramos aquí los eventos de los botones, centrándonos en el objetivo del trabajo, ponemos el ejemplo de definición del botón que desencadenará las acciones de creación del fichero XML.

```
bCrearXML = new Button(shlPrcticaGac, SWT.NONE);
bCrearXML.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        crearXML();
    }
});
FormData fd_bCrearXML = new FormData();
fd_bCrearXML.top = new FormAttachment(tSalida, 18);
fd_bCrearXML.left = new FormAttachment(0, 25);
bCrearXML.setLayoutData(fd_bCrearXML);
bCrearXML.setText("Crear XML");
```

Todo el proceso de lectura, carga y exportación se realiza en el método **crearXML**.

Los pasos que se realizan para completar la transformación de formato son:

- 1º A partir del fichero de entrada se carga su contenido en un String

```
String contents = "";
try {
    /* Leemos el fichero de origen y lo guardamos en un String */
    contents = new String(Files.readAllBytes(Paths.get(tOrigen)));
} catch (IOException e1) {
    // Excepción porque ha habido un problema en la lectura del
    // fichero

    muestraDialogoModal(SWT.ICON_ERROR | SWT.OK, "Error de Lectura
I/O",
                        "No es posible leer el fichero
seleccionado de origen JSON");
    logger.error(e1);
}
```

- 2º El contenido se parsea a un objeto java de tipo **JSONObject** con el fin de poder tratarlo posteriormente como objeto java.

```
try {
    /*
    * Cargamos el contenido del fichero JSON en un OBJETO
    * JSONObject
    */
    JSONObject o = new JSONObject(contents);

    /* Parseamos el objeto JSONObject en un String con formato XML */
    String xml = XML.toString(o);

} catch (JSONException e1) {

    muestraDialogoModal(SWT.ICON_ERROR | SWT.OK, "JSON error",
                        "El fichero de entrada no cumple con formato JSON");
    logger.error(e1);
}
```

- 3º Una vez que disponemos del objeto JSONObject lo transformamos en XML apoyándonos también en la librería json.org, lo guardamos en la ruta de salida previamente seleccionada.

```
try {
    String xml = XML.toString(o);

    /* Se muestra el XML generado */
    tAreaDestino.setText(xml);
    /* Se guarda el fichero generado */
    Files.write(Paths.get(tSalida.getText()), xml.getBytes());

    muestraDialogoModal(SWT.ICON_WORKING | SWT.OK, "Correcto",
                        "Fichero xml creado correctamente.");

} catch (IOException e1) {
```

```
// Excepción porque ha habido un problema en la escritura del
// fichero

muestraDialogoModal(SWT.ICON_ERROR | SWT.OK, "Error de escritura
I/O",
    "No es posible escribir el fichero seleccionado de salida XML");
logger.error(e1);
}
```

El método *muestraDialogoModal* nos permite realizar un mensaje genérico indicándole el tipo de mensaje, los botones que queremos mostrar y el mensaje en sí.

```
private void muestraDialogoModal(int estilo, String titulo, String
texto) {
    MessageBox messageBox = new MessageBox(shlPrcticaGac,
estilo);
    messageBox.setText(titulo);
    messageBox.setMessage(texto);
    messageBox.open();
}
```

### 2.3 Alcance y limitaciones de la solución propuesta

Es necesario tener una máquina virtual java 1.8 y un sistema operativo Windows 64 bits. Para cada sistema operativo se debe realizar una importación de dependencias y definir el classpath,

El alcance de la solución abarca todos los formatos de entrada json con la limitación de aquellos ficheros con formato lista sin una clave raíz y que la librería json.org los considera mal formados:

```
{
  [
    {
      "name": "Rixos The Palm Dubai",
      "position": [25.1212, 55.1535],
    },
    {
      "name": "Shangri-La Hotel",
      "location": [25.2084, 55.2719]
    },
    {
      "name": "Grand Hyatt",
      "location": [25.2285, 55.3273]
    }
  ]
}
```

Este tipo de json deberán tener un clave que defina la lista:

```
{
  "markers": [
    {
```

```

    "name": "Rixos The Palm Dubai",
    "position": [25.1212, 55.1535],
  },
  {
    "name": "Shangri-La Hotel",
    "location": [25.2084, 55.2719]
  },
  {
    "name": "Grand Hyatt",
    "location": [25.2285, 55.3273]
  }
]
}

```

## 2.4 Descripción de los casos de Prueba

El plan de pruebas se ha definido con la intención de abarcar las máximas situaciones que se pueden dar de formato json, los casos probados son:

- 1) JSON simple de clave valor. El formato de entrada es del tipo:

```

{
  "ID": "SGML",
  "SortAs": "SGML",
  "GlossTerm": "Standard Generalized Markup Language",
  "Acronym": "SGML",
  "Abbrev": "ISO 8879:1986",
  "GlossSee": "markup"
}

```

La salida en este caso es:

```

<GlossTerm>Standard Generalized Markup Language</GlossTerm>
<GlossSee>markup</GlossSee>
<SortAs>SGML</SortAs>
<ID>SGML</ID>
<Acronym>SGML</Acronym>
<Abbrev>ISO 8879:1986</Abbrev>

```

- 2) JSON anidados. Presenta objetos dentro de objetos:

```
{
  "markers": [
    {
      "name": "Rixos The Palm Dubai",
      "position": [25.1212, 55.1535],
    },
    {
      "name": "Shangri-La Hotel",
      "location": [25.2084, 55.2719]
    },
    {
      "name": "Grand Hyatt",
      "location": [25.2285, 55.3273]
    }
  ]
}
```

La salida en xml es:

```
<glossary>
  <title>example glossary</title>
  <GlossDiv>
    <GlossList>
      <GlossEntry>
        <GlossTerm>Standard Generalized Markup Language</GlossTerm>
        <GlossSee>markup</GlossSee>
        <SortAs>SGML</SortAs>
        <GlossDef>
          <para>A meta-markup language, used to create markup languages such
            as DocBook.</para>
          <GlossSeeAlso>GML</GlossSeeAlso>
          <GlossSeeAlso>XML</GlossSeeAlso>
        </GlossDef>
        <ID>SGML</ID>
        <Acronym>SGML</Acronym>
        <Abbrev>ISO 8879:1986</Abbrev>
      </GlossEntry>
    </GlossList>
    <title>S</title>
  </GlossDiv>
</glossary>
```

### 3) JSON con lista de objetos, que a su vez puedan estar anidados

```
{
  "markers": [
    {
      "name": "Rixos The Palm Dubai",
      "position": [25.1212, 55.1535],
    },
    {
      "name": "Shangri-La Hotel",
      "location": [25.2084, 55.2719]
    },
    {
      "name": "Grand Hyatt",
      "location": [25.2285, 55.3273]
    }
  ]
}
```

La salida XML es:



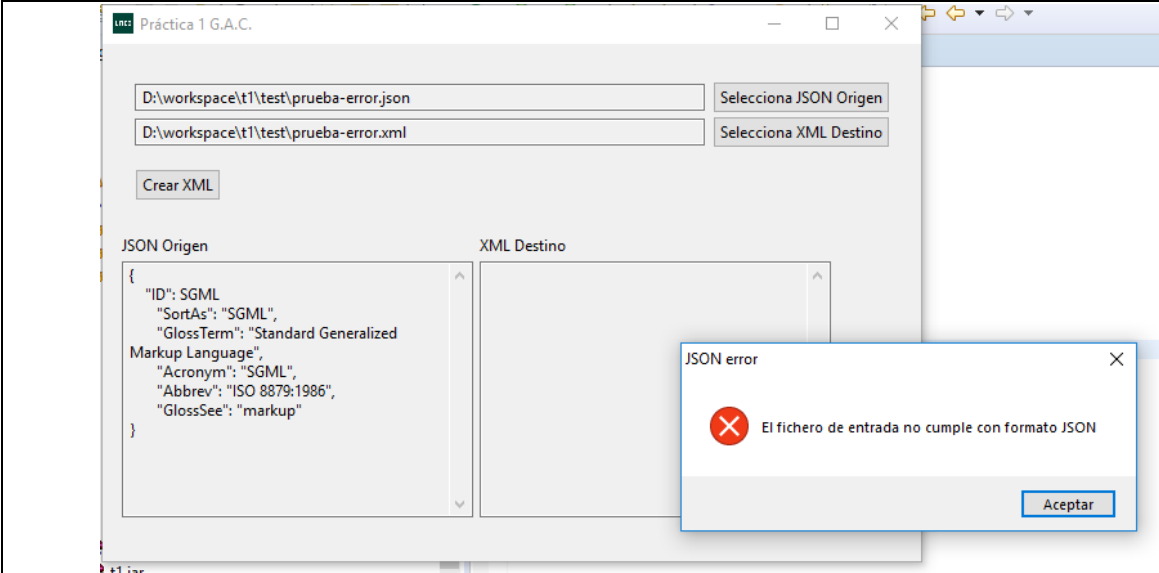
```

<markers>
  <name>Rixos The Palm Dubai</name>
  <position>25.1212</position>
  <position>55.1535</position>
</markers>
<markers>
  <name>Shangri-La Hotel</name>
  <location>25.2084</location>
  <location>55.2719</location>
</markers>
<markers>
  <name>Grand Hyatt</name>
  <location>25.2285</location>
  <location>55.3273</location>
</markers>

```

- 4) Para el control de errores, se realizan pruebas con datos mal formados. Aquí incluimos la restricción de ficheros sin al menos un elemento clave que ya hemos descrito en el punto 2.3.

En este caso de prueba, al intentar el parseo el programa eleva una excepción y no sigue con el proceso.



En el log:

```

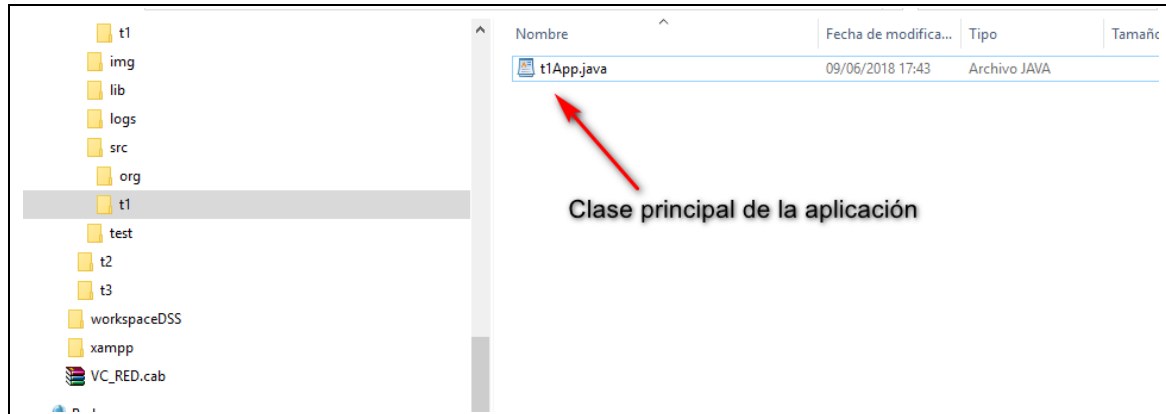
[main] ERROR t1.t1App - org.json.JSONException: Expected a ',' or ']' at 1000490 [character 1 line 3]
[main] ERROR t1.t1App - org.json.JSONException: A JSONObject text must begin with '{' at 1 [character 1 line 3]
[main] ERROR t1.t1App - org.json.JSONException: Expected a ',' or '}' at 21 [character 2 line 3]
[main] ERROR t1.t1App - org.json.JSONException: Expected a ',' or '}' at 21 [character 2 line 3]
[main] ERROR t1.t1App - org.json.JSONException: Expected a ',' or '}' at 21 [character 2 line 3]

```

### 3 Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados

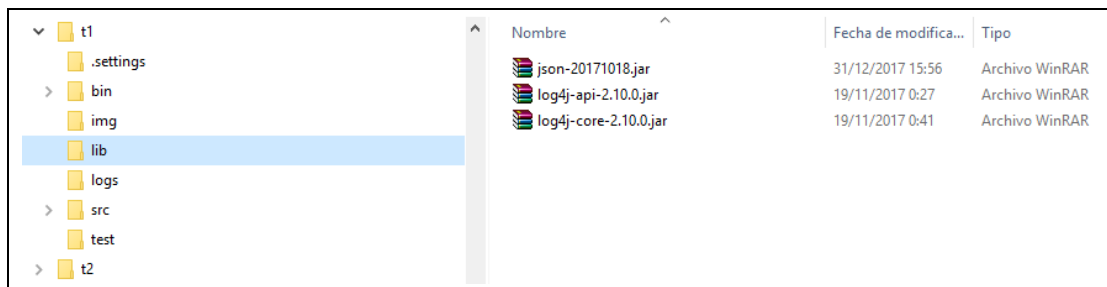
A continuación, se pasa a describir la entrega realizada, al descomprimir el fichero zip nos encontraremos con el siguiente árbol de directorios.

**Carpeta src:** Contiene los fuentes de la aplicación. En esta carpeta se encuentra el archivo como la clase **t1App.java** dentro de la carpeta t1, en esta clase esta la funcionalidad principal de la aplicación



Dentro de la carpeta src también encontramos el paquete org que eclipse genera automáticamente cuando creamos un proyecto SWT.

**Carpeta lib,** se sitúan las librerías de terceros que hemos utilizado en el proyecto



**Carpeta test.** Contiene los ficheros de pruebas y sus salidas

**Carpeta bin.** Contiene las clases compiladas puede estar vacío ya que solo tiene contenido en el momento de la compilación.

**Carpeta img.** Contiene las imágenes que se pueden utilizar en el proyecto

**Carpeta logs.** Contiene las trazas de ejecución del programa.

**Ficheros en el raíz.** Nos vamos a encontrar con los siguientes ficheros en el raíz.

**t1.jar**, librería ejecutable del programa

**swing2swt.jar**, librería necesaria para arrancar el entorno gráfico.

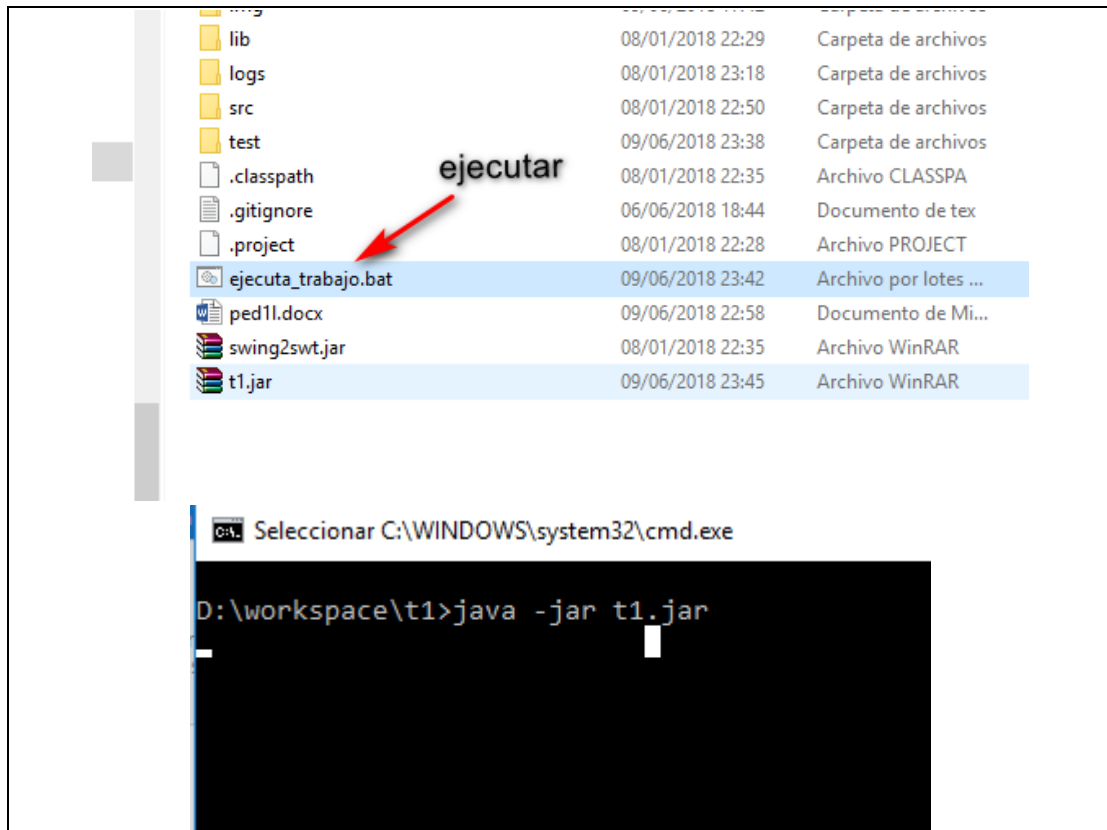
**ejecuta\_trabajo.bat**, en Windows, arranca el programa.

## 4 Manuales de uso/instalación

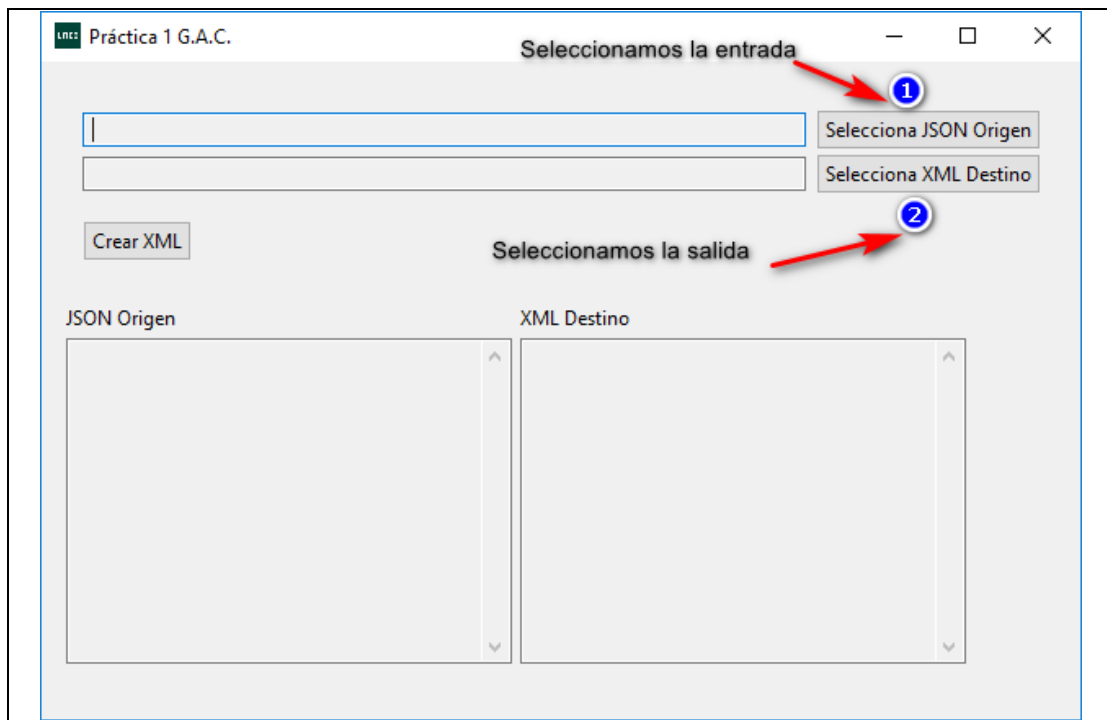
Para usar el programa solo es necesario disponer de una máquina virtual java 1.8 instalada en el equipo y el sistema operativo debe ser visual.

Los pasos para ejecutar el programa son los siguientes:

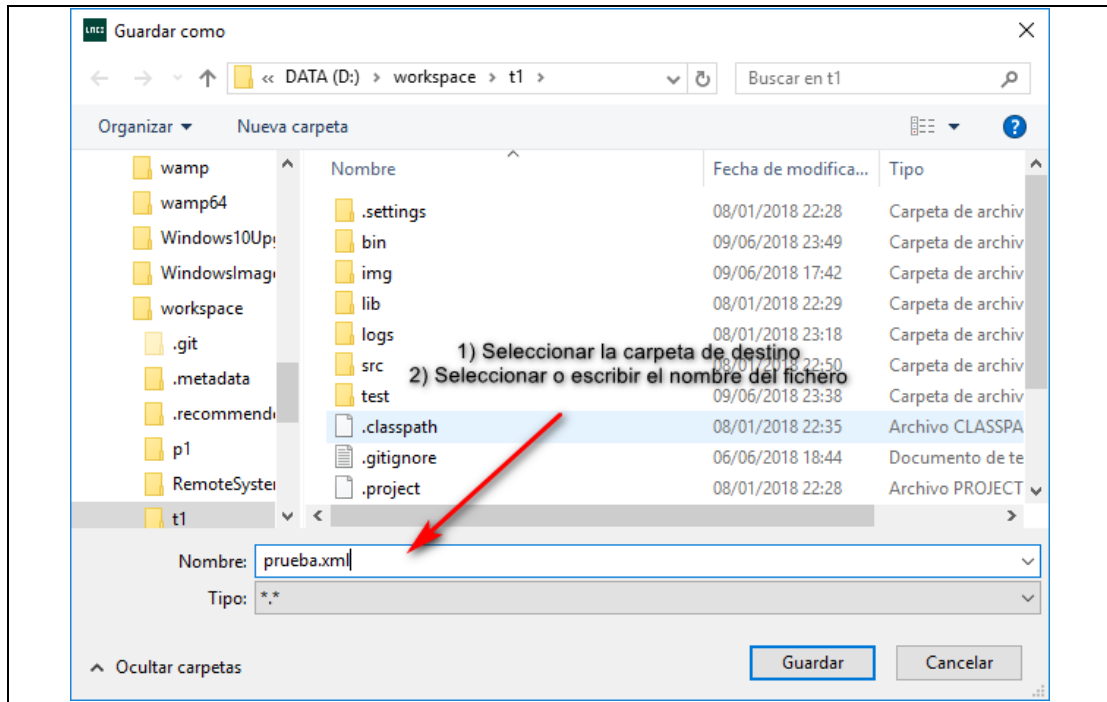
- 1º En Windows, ejecutar el archivo por lotes **ejecuta\_trabajo.bat** o realizar la llamada **java -jar t1.jar**



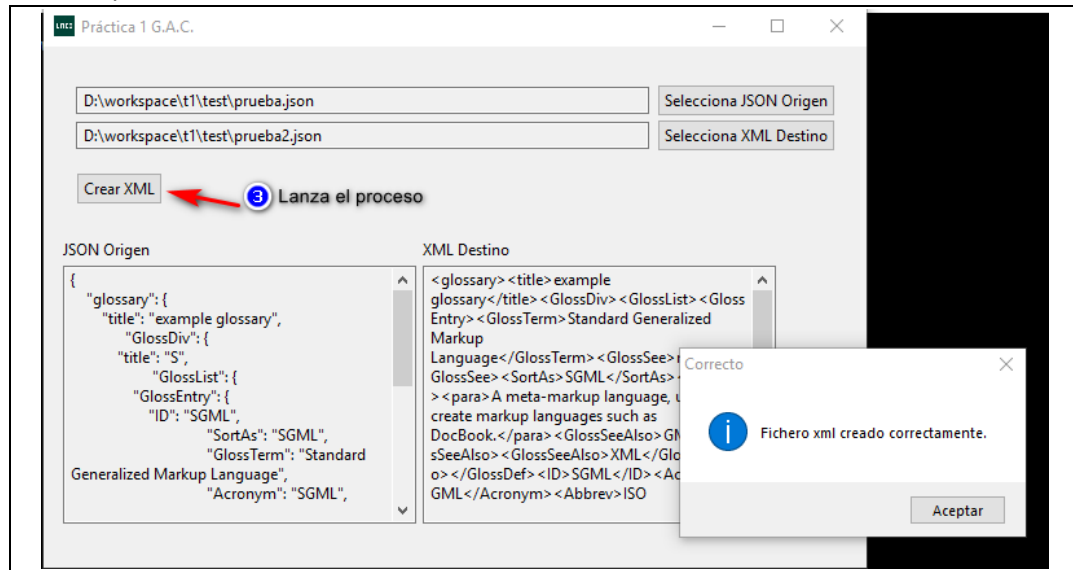
- 2º Nos mostrara la siguiente ventana donde deberemos seleccionar el fichero JSON de entrada y en fichero XML de salida.  
Las casillas de texto no son editables y solo se puede seleccionar mediante el cuadro de diálogo.



En el caso de la selección del fichero de salida se deberá escribir el nombre en la casilla del cuadro de dialogo.



3º Pulsamos sobre el botón **Crear XML** para lanzar el proceso. Nos mostrara la salida en pantalla de los ficheros tanto de entra como de salida.



## 5 Referencias

<http://json.org/>

<https://www.eclipse.org/swt/>