JSON con el API JavaScript de Java

1. **JSON en Java (con Scripting API)**
    1.1. **Implementación de JSON con Scripting API de Java**
        Código de ejemplo:
    1.2. **Dependencias con GraalVM (para versiones actuales)**
    1.3. **Ejecución de scripts JavaScript desde Java**
        Salida esperada:
        Compilación y Ejecución:
2. **Análisis de JSON con** `JSON.parse()`
    2.1. **Conversión de texto JSON a objetos**
        Ejemplo de código:
        Salida esperada:
    2.2. **Errores de Sintaxis en** `JSON.parse()`
        Ejemplo de error:
        Salida esperada (con error de sintaxis):
3. **Resumen del funcionamiento en Java con JSON**
4. **Conclusión**

1. JSON en Java (con Scripting API)

Aunque JSON no está integrado en la API estándar de Java, es posible trabajar con él utilizando la **API de Scripting** de Java. A continuación se explica cómo

usar un motor de JavaScript en Java para manejar JSON.

1.1. Implementación de JSON con Scripting API de Java

- **Motor de scripting:** Java proporciona el motor **Nashorn** (obsoleto en versiones más recientes de JDK). Si necesitas usar JSON en Java, puedes aprovechar el motor de JavaScript con la **Scripting API**. (No entra en el examen)

Código de ejemplo:

```
java
Copiar código
import java.io.FileReader;
import java.io.IOException;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;
import static java.lang.System.*;

public class RunJSScript {
    public static void main(String[] args) {
        if (args.length != 1) {
            err.println("uso: java RunJSScript scriptEnJS");
            return;
        }

        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine engine = manager.getEngineByName("nashorn");

        try {
            engine.eval(new FileReader(args[0]));
        } catch (ScriptException se) {
            err.println(se.getMessage());
        } catch (IOException ioe) {
            err.println(ioe.getMessage());
        }
    }
```

```
}
```

1.2. Dependencias con GraalVM (para versiones actuales)

- En versiones actuales de Java quizás debas añadir un motor de JavaScript a tu proyecto Maven, como ECMAScript como el proporcionado por Oracle GraalVM Oracle GraalVM for JDK 21:

```
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js</artifactId>
  <version>23.0.1</version>
</dependency>
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js-scriptengine</artifactId>
  <version>23.0.1</version>
</dependency>
```

1.3. Ejecución de scripts JavaScript desde Java

El código anterior ejecuta un archivo JavaScript proporcionado como argumento desde la línea de comandos. Ejemplo de un script `poeta.js` en formato JSON:

```
var poeta = {
  "nombre": "Sylvia",
  "apellidos": "Plath",
  "estaViva": false,
  "edad": 30,
  "direccion": {
    "direccionCalle": "21 2nd Street",
    "ciudad": "New York",
    "estado": "NY",
    "codigoPostal": "10021-3100"
  },
```

```
    "telefonos": [
        { "tipo": "casa", "numero": "212 555-1234" },
        { "tipo": "oficina", "numero": "646 555-4567" }
    ],
    "hijos": [],
    "marido": null
};

print(poeta.nombre);
print(poeta.apellidos);
print(poeta.direccion.ciudad);
print(poeta.telefonos[1].numero);
```

Salida esperada:

```
Copiar código
Sylvia
Plath
New York
646 555-4567
```

Compilación y Ejecución:

Suponiendo el Script se llama `poeta.js`, ejecuta la aplicación de la siguiente manera:

```
javac RunJSScript.java
java RunJSScript poeta.js
```

---

2. Análisis de JSON con `JSON.parse()`

2.1. Conversión de texto JSON a objetos

- Un objeto JSON existe como **texto independiente del lenguaje**. Para convertir el texto en un objeto dependiente del lenguaje, necesitas analizar el texto. **JavaScript proporciona un objeto `JSON` con un**

método `parse()` para esta tarea.

Ejemplo de código:

```
var tarjetaJSON = "{ \"numero\": \"1234567890123456\", \"caducidad\": \"20/04\", \"tipo\": \"visa\" }";
var tarjeta = JSON.parse(tarjetaJSON);

print(tarjeta.numero);
print(tarjeta.caducidad);
print(tarjeta.tipo);
```

Salida esperada:

```
1234567890123456
20/04
visa
```

2.2. Errores de Sintaxis en `JSON.parse()`

- Los nombres y claves en JSON **deben estar entre comillas dobles**. Si se usan comillas simples o hay otro error de formato, se lanzará un error de sintaxis.

Ejemplo de error:

```
var tarjetaJSON2 = "{ 'tipo': 'visa' }";
var tarjeta2 = JSON.parse(tarjetaJSON2);
```

Salida esperada (con error de sintaxis):

```
typescript
Copiar código
SyntaxError: JSON no válido: <json>:1:2 Se esperaba , o } pero se encontró
'
```

```
{ 'type': 'visa' }
^ en <eval> en la línea número 11
```

3. Resumen del funcionamiento en Java con JSON

1. **Scripting API de Java:**
   - Java permite ejecutar scripts de JavaScript (incluyendo JSON) a través de la Scripting API.
   - Es necesario obtener un motor de script (como ECMAScript o GraalVM) y evaluar el script JSON.
2. **Uso de JSON en Java:**
   - El JSON puede ser manejado como un objeto a través del motor JavaScript embebido en Java.
   - Esto requiere una correcta configuración del entorno (dependencias como GraalVM en versiones modernas).
3. **Análisis de JSON en JavaScript (`JSON.parse()`):**
   - El método `JSON.parse()` convierte texto JSON en un objeto JavaScript.
   - Es fundamental respetar la sintaxis de JSON (usar **comillas dobles** para los nombres de las claves).
4. **Compatibilidad en versiones recientes de Java:**
   - Si bien **Nashorn** está obsoleto, GraalVM es una alternativa moderna que permite ejecutar código JavaScript en Java.

4. Conclusión

El manejo de JSON en Java requiere utilizar la API de Scripting para ejecutar scripts en JavaScript que procesan JSON. Con la evolución de Java, se deben usar motores modernos como **GraalVM** para asegurar compatibilidad y rendimiento adecuado al ejecutar código JavaScript dentro de aplicaciones Java.