educación IT
CODE YOUR FUTURE

# Introducción a serializadores JSON y Gson

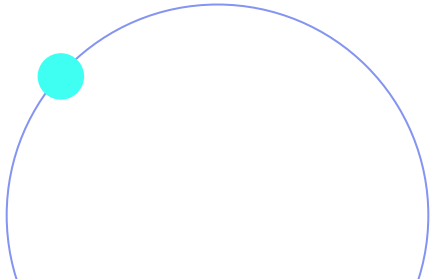
# ¿Qué son los serializadores?

En el contexto de una API RESTful en Java, **un "serializador" es un componente que convierte objetos Java (modelos de datos) en un formato que puede ser enviado a través de la red, típicamente JSON o XML, y viceversa.**

Este proceso se conoce como:

- Serialización, cuando se convierte de objeto a formato de intercambio de datos.
- Deserialización, cuando se convierte de formato de intercambio de datos a objeto.

Estos procesos **son fundamentales para convertir datos entre objetos Java y formatos de texto estándar utilizados en protocolos como HTTP, permitiendo tanto el envío como la recepción de información de manera estructurada y eficiente.**



# Funciones clave de un serializador

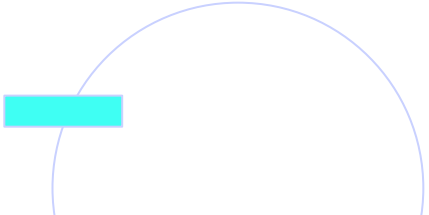
- **Transformar datos a un formato portable:** convierte datos complejos a un formato estándar (como JSON) para compartirlos con otros sistemas.
- **Validación de datos:** algunos serializadores pueden validar los datos durante el proceso, asegurándose de que cumplan con ciertos criterios.

**Ventajas**

- Facilita la comunicación entre diferentes sistemas.
- Esencial para el almacenamiento y recuperación de datos en formatos estándar.
- Proporciona validación de datos, asegurando consistencia y seguridad.

# Funciones del serializador en una API Rest

| Serialización | Deserialización |
| --- | --- |
| Convertir objetos Java en JSON o XML para ser enviados en la respuesta HTTP. | Convertir datos en formato JSON o XML recibidos en la solicitud HTTP en objetos Java. |

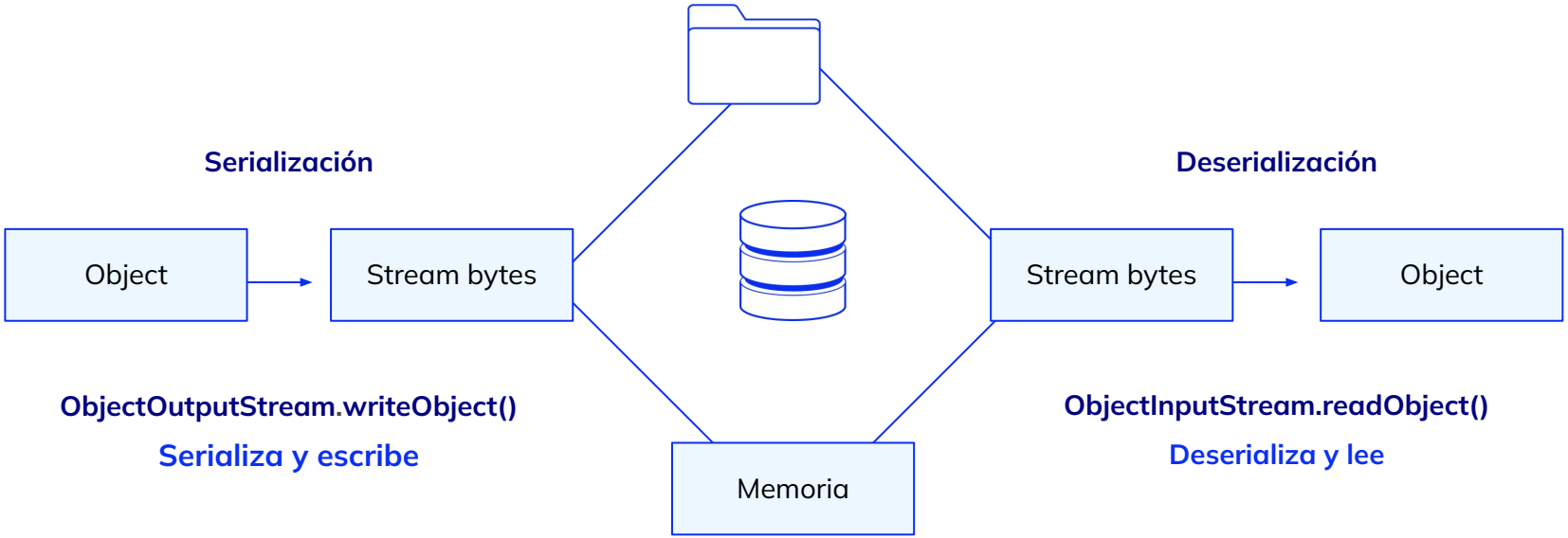


**La gestión eficiente de los datos implica procesos clave como la serialización y la deserialización.**

La imagen a continuación, en la siguiente *slide*, resume visualmente cómo los objetos en memoria pueden transformarse en flujos de datos para su almacenamiento o transmisión, y cómo esos datos pueden regresar a su estado original.

Este proceso es fundamental en la comunicación entre aplicaciones y en el manejo de datos persistentes.

## Serialización y deserialización: transformación de objetos en datos




Trabajando con ObjectOutputStream y ObjectInputStream

# Serializadores comunes en Java

Existen varias bibliotecas y *frameworks* en Java que facilitan la serialización y deserialización en el contexto de una API RESTful.

Algunos de los más populares son:

- **JSONP**: significa JSON Processing API para Java.
- **Jackson**: una de las bibliotecas más populares para trabajar con JSON en Java.
- **Gson**: otra biblioteca popular de Google para trabajar con JSON.
- **JAXB (Java Architecture for XML Binding):** Utilizado principalmente para trabajar con XML.

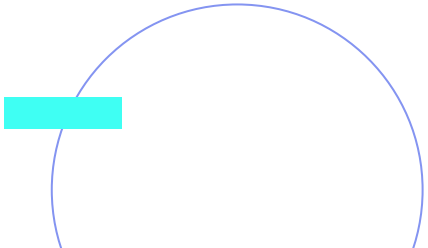
# JSON Processing API para Java

**JSONP es una API estándar de Java que proporciona soporte para procesar JSON de manera fácil y eficiente.**

Forma parte del paquete javax.json desde Java EE 7 y es una forma sencilla de leer, escribir y manipular datos en formato JSON, en aplicaciones Java.

Algunas de sus características son:

- **Proporciona clases y métodos para leer y escribir datos JSON.**
- **Ofrece soporte para análisis y generación de JSON.**
- **Es parte de la especificación Java EE 7 y está disponible en la plataforma Java EE.**



## Ejemplo a partir del JSON de un objeto de la clase Persona

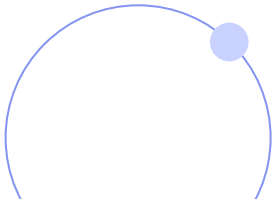
```
{
    "nombre": "Juan A. Pérez",
    "edad": 31,
    "direccion": {
        "calle": "Calle Falsa 123",
        "ciudad": "Madrid",
        "codigoPostal": 28013
    },
    "telefono": "+34 600 123 456",
    "email": "juan.perez@example.com"
}
```

## Ejemplo de serialización

```
import javax.json.bind.Jsonb;
import javax.json.bind.JsonbBuilder;

public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona("Juan A. Pérez", 31, new Direccion("Calle
Falsa 123"));

        Jsonb jsonb = JsonbBuilder.create();

        // Serialización a JSON
        String json = jsonb.toJson(persona);
        System.out.println("JSON generado con JSONB:");
        System.out.println(json);
    }
}
```

## Ejemplo de deserialización

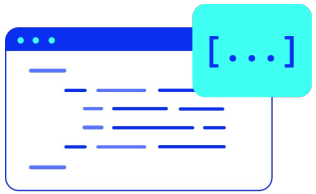
```
// Deserialización de JSON
Persona personaDeserializada = jsonb.fromJson(json,
Persona.class);
System.out.println("Objeto Persona deserializado:");
System.out.println(personaDeserializada);
    }
}
```



# Jackson

**Esta biblioteca proporciona un conjunto completo de funciones para leer, escribir y manipular datos JSON de manera eficiente, ofreciendo una solución robusta para trabajar con este formato ampliamente utilizado en aplicaciones modernas.**

Permite a los desarrolladores manejar datos JSON de manera sencilla y flexible, ya sea extrayendo información específica, modificando contenido o generando nuevos documentos JSON desde cero.



## Características

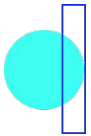
- Proporciona una alta velocidad de procesamiento.
- Ofrece una amplia gama de características, como enlaces directos de datos, árboles de objetos, modelo de datos de *streaming* y otros.
- Permite la personalización y configuración avanzada.
- Es ampliamente utilizado en proyectos Java y es compatible con muchos *frameworks* y tecnologías.

## Ejemplo de serialización en Jackson

```
import com.fasterxml.jackson.databind.ObjectMapper;

public class Main {
    public static void main(String[] args) throws Exception {
        Persona persona = new Persona("Juan A. Pérez", 31, new Direccion("Calle
Falsa 123"));

        ObjectMapper objectMapper = new ObjectMapper();

        // Serialización a JSON
        String json = objectMapper.writeValueAsString(persona);
        System.out.println("JSON generado con Jackson:");
        System.out.println(json);
    }
}
```

## Ejemplo de deserialización en Jackson

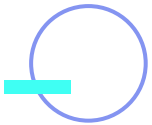
```
// Deserialización de JSON
Persona personaDeserializada = objectMapper.readValue(json, Persona.class);
System.out.println("Objeto Persona deserializado:");
System.out.println(personaDeserializada);
```



# Gson (Google Gson)

Al igual que Jackson, **Gson permite la serialización y deserialización de objetos Java en formato JSON y viceversa.**

Es conocida por su facilidad de uso y su capacidad para manejar tipos de datos complejos de manera transparente.



### Ventajas

- **Transparente y ligero:** No añade configuraciones innecesarias, lo que lo hace muy intuitivo para los desarrolladores.
- **Eficiente para prototipado:** Ideal para aplicaciones pequeñas y medianas que necesitan rapidez en el desarrollo.

## Características

- Fácil de usar y configurar.
- Ofrece un rendimiento decente, en la mayoría de los casos.
- Proporciona anotaciones personalizadas para controlar el proceso de serialización y deserialización.
- Es compatible con tipos de datos complejos, como listas, mapas, clases anidadas, y otros.
- Se lleva muy bien con Spring Boot.

## Ejemplo de Gson para la serialización y deserialización

```
public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona("Juan A. Pérez", 31, new
Direccion("Calle Falsa 123"));

        Gson gson = new Gson();

        // Serialización a JSON
        String json = gson.toJson(persona);
        System.out.println("JSON generado con Gson:");
        System.out.println(json);

        // Deserialización de JSON
        Persona personaDeserializada = gson.fromJson(json, Persona.class);
        System.out.println("Objeto Persona deserializado:");
        System.out.println(personaDeserializada);
    }
}
```

# JAXB (Java Architecture for XML Binding)

**Es una API de Java que permite convertir objetos Java en XML y viceversa. Esto simplifica el manejo de datos en aplicaciones.** Además, facilita la serialización y deserialización mediante anotaciones en las clases Java.

JAXB permite generar clases a partir de un esquema XML (XSD) o crear un esquema a partir de clases.

Es útil en integraciones donde se intercambian datos en formato XML de manera eficiente.