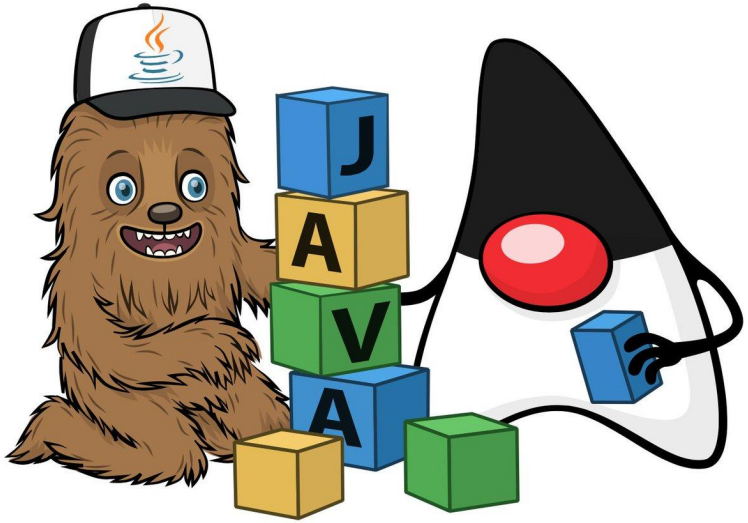
Clase 19: Serialización & JSON

Programación & Laboratorio III

Agenda

- **Serialización**
- Interfaz Serializable
- Serial Version UID
- Modificador transient
- Serialización y archivos
  - ObjectOutputStream
  - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



Serialización

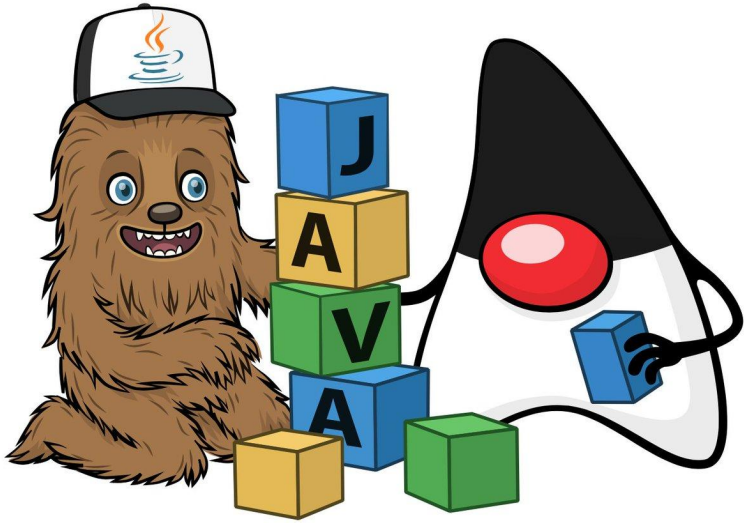
- La serialización permite convertir cualquier objeto (que implemente la interfaz Serializable) en una secuencia de bytes que pueden ser posteriormente leídos para restaurar el objeto original.
- Esta característica se mantiene incluso a través de la red, por lo que podemos crear un objeto en una máquina que corre bajo el sistema operativo Windows, serializarlo y enviarlo a través de la red a una máquina que corre bajo UNIX donde será correctamente reconstruido.

Serialización (2)

- La serialización es una característica añadida al lenguaje Java para dar soporte a:
  - La invocación remota de objetos
  - La persistencia.
- La invocación remota de objetos permite a los objetos que “viven” en otras máquinas comportarse como si vivieran en nuestra propia máquina.
- Para la persistencia, la serialización nos permite guardar el estado de un componente en disco, abandonar el IDE y restaurar el estado de dicho componente cuando se vuelve a correr el IDE.
-

Agenda

- ~~Serialización~~
- **Interfaz Serializable**
- Serial Version UID
- Modificador transient
- Serialización y archivos
  - ObjectOutputStream
  - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



Interfaz Serializable

- Un objeto se puede serializar si implementa la interfaz Serializable. Esta interfaz no declara ninguna función, se trata de una interfaz vacía.

```
public interface Serializable{
}
```

- Para hacer una clase serializable simplemente debemos implementar esta interfaz:

```
public class Persona implements Serializable{
    private String nombre;
    ...
}
```

Interfaz Serializable (2)

- Si dentro de la clase hay atributos que son otras clases, éstos a su vez también deben implementar Serializable. Con los tipos de java (String, Integer, etc.) no hay problema porque lo son. Por ejemplo:

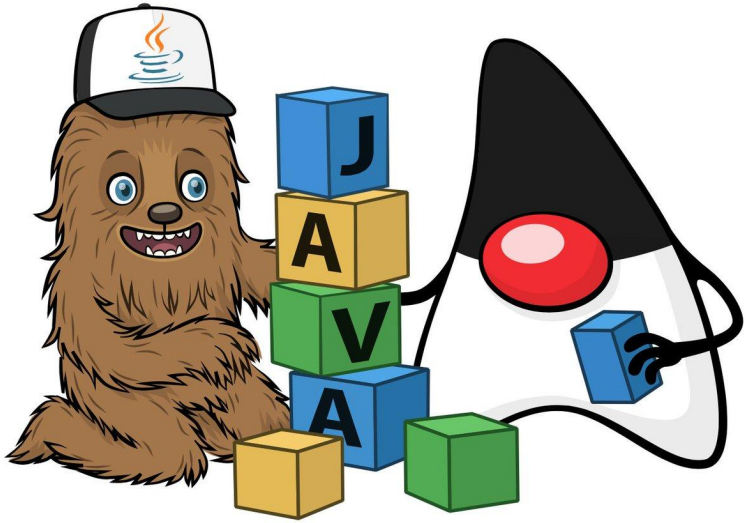
```
public class Persona implements Serializable {
    private String nombre;
    private Direccion direccion;
}

public class Direccion implements Serializable {
    private String calle;
    private String ciudad;
    ...
}
```

Interfaz Serializable (3)

- La serialización de un objeto como atributo de otro objeto se puede tomar como un árbol, donde las hojas son objetos que forman parte de otro objeto como un atributo, y todos ellos son marcados como serializables, para así entonces serializar primero las hojas del árbol y finalizar con la raíz.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- **Serial Version UID**
- Modificador transient
- Serialización y archivos
  - ObjectOutputStream
  - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



Serial Version UID

- Cuando pasamos objetos serializables de un lado a otro, puede llegar a pasar que en las distintas versiones de nuestro programa la clase cambie. De esta manera es posible que un lado tenga una versión más antigua que en el otro lado. Si esto sucede, la reconstrucción de la clase en el lado que recibe es imposible.
- Para evitar este problema, se aconseja que la clase que queremos serializar tenga un atributo de la siguiente forma:

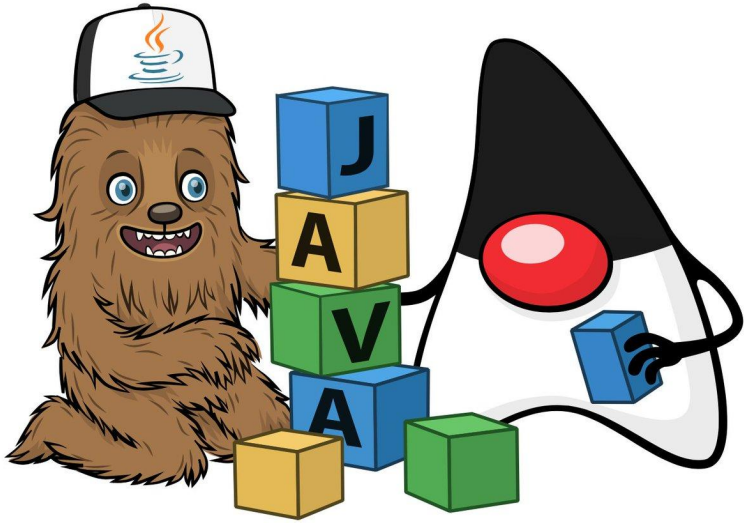
```
public class Persona implements Serializable {
     private static final long serialVersionUID = 879965647867471663 8L;

     ...
}
```

Serial Version UID (2)

- El número del final debe ser distinto para cada versión de compilado que tengamos.
- De esta forma, la JVM es capaz de detectar rápidamente que las versiones en ambos lados son diferentes.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- **Modificador transient**
- Serialización y archivos
  - ObjectOutputStream
  - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



Modificador transient

- Habrá ocasiones donde no será necesario incluir un atributo del objeto en la serialización, y para esto se encuentra el modificador transient.
- Este modificador le indica a la JVM que dicho atributo deberá ser exento de la serialización, en otras palabras, ignorará este atributo.
- Por otro lado, los atributos que lleven el modificador static nunca se tomarán en cuenta al serializar un objeto, ya que este atributo pertenece a la clase y no al objeto.

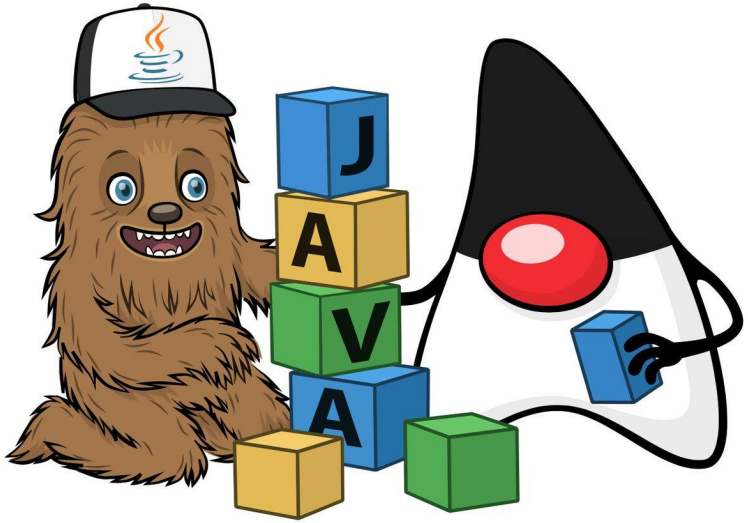
Modificador transient (2)

- Ejemplo:

```
public class Persona implements Serializable {
    private String nombre;
    private String apellidoCasada;
    private transient String apellidoSoltera;

    ...

}
```

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- **Serialización y archivos**
  - ObjectOutputStream
  - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



Serialización y archivos

- Un uso de la serialización es la lectura y escritura de objetos en un archivo.
- Ejemplo:

```
public class Persona implements Serializable {
    private String nombre;
    private Direccion direccion;
}

public class Direccion implements Serializable {
    private String calle;
    private String ciudad;
    ...
}
```

Serialización y archivos (2)

- Escribir en el archivo:

```
File file = new File(“mi_archivo.txt”);
ObjectOutputStream objOutputStream = new ObjectOutputStream(new
FileOutputStream(file));

Persona p = new Persona();
objOutputStream.writeObject(p);

objOutputStream.close();
```

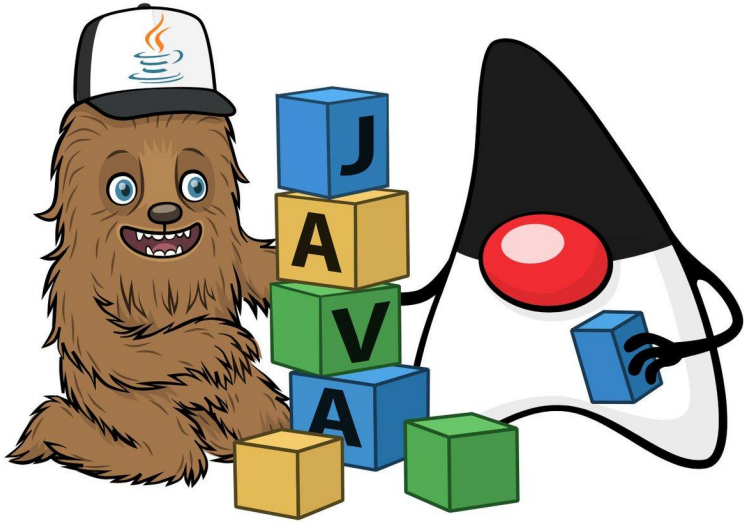
Serialización y archivos (3)

- Leer archivo:

```
    File file = new File(“mi_archivo.txt”);
    ObjectInputStream objInputStream = new ObjectInputStream(new
FileInputStream(file));

    Object aux = objInputStream.readObject();
    while (aux!=null) {
       if (aux instanceof Persona)
          System.out.println(aux);
          aux = objInputStream.readObject();
    }

    objInputStream.close();
```

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
    - **ObjectOutputStream**
    - ObjectInputStream
- JSON
- Librerías para procesamiento de JSON
    - Jackson
    - Gson

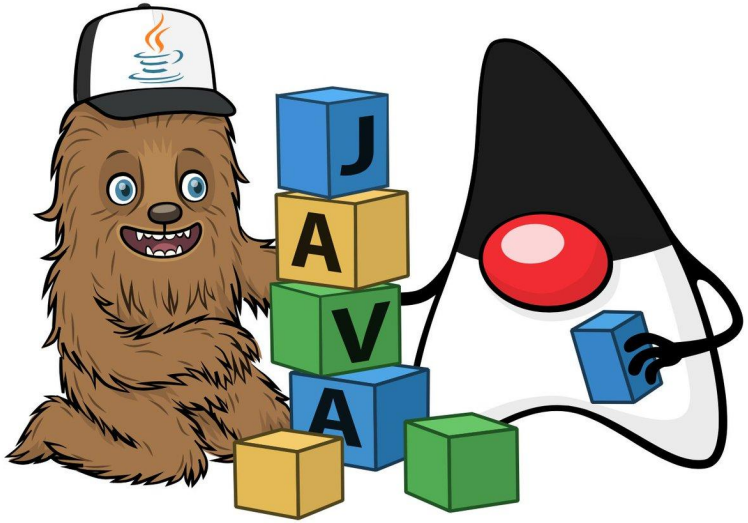


ObjectOutputStream

- Se utiliza para escribir objetos en un OutputStream en lugar de escribir el objeto convertido a bytes. De esta manera se encapsula el OutṕutStream en un ObjectOutputStream.
- Sólo los objetos que implementen la interfaz Serializable pueden escribirse en los streams.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
  - ~~ObjectOutputStream~~
  - **ObjectInputStream**
- JSON
- Librerías para procesamiento de JSON
  - Jackson
  - Gson

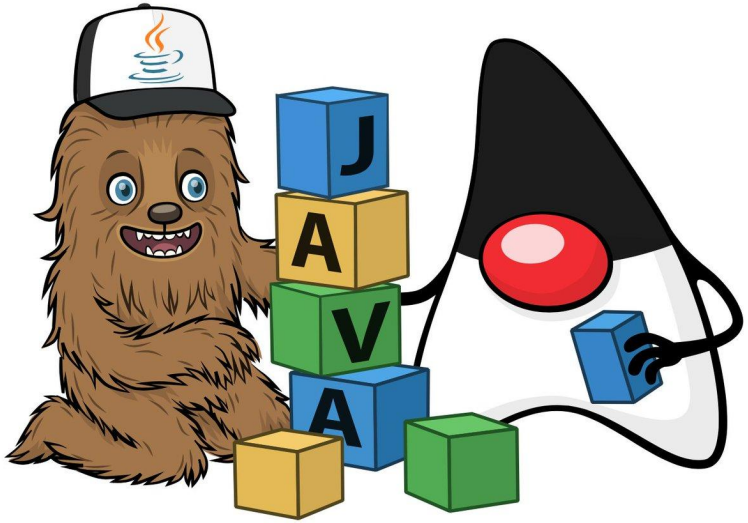


ObjectInputStream

- Se utiliza en conjunto con ObjectOutputStream para leer los objetos que fueron escritos.
- Se debe notar que al leer los objetos tal vez sea necesario realizar un casting sobre los mismos.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
  - ~~ObjectOutputStream~~
  - ~~ObjectInputStream~~
- **JSON**
- Librerías para procesamiento de JSON
  - Jackson
  - Gson



JSON

- JSON (JavaScript Object Notation) es un formato para intercambiar datos liviano, basado en texto e independiente del lenguaje de programación fácil de leer tanto para seres humanos como para las máquinas.
- Puede representar dos tipos estructurados: objetos y matrices.
    - Un objeto es una colección no ordenada de cero o más pares de nombres/valores.
    - Una matriz es una secuencia ordenada de cero o más valores.

- Los valores pueden ser cadenas, números, booleanos, nulos y estos dos tipos estructurados.

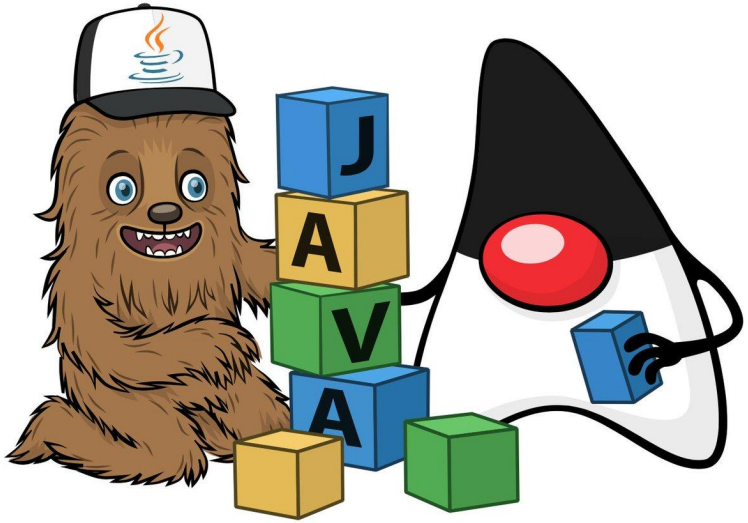
JSON (2)

```
{
   "nombre": "Homero",
   "apellido": "Simpson",
   "edad": 35,
   "direccion": {
      "calle": "Av. Colón 1145",
      "ciudad": "Mar del Plata",
      "codigoPostal": 7600
   },
   "telefonos": [
      {
        "tipo": "casa",
        "numero": "212  555-1234"
      },
      {
            "tipo": "celular",
            "numero": "646  555-4567"
      }
   ]
}
```

- Ejemplo: representación JSON de un objeto Persona, que tiene valores de cadena para nombre y apellido, un valor numérico para la edad, un valor de objeto que representa el domicilio de la persona y un valor de matriz de objetos de números telefónicos.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
  - ~~ObjectOutputStream~~
  - ~~ObjectInputStream~~
- ~~JSON~~
- **Librerías para procesamiento de JSON**
  - Jackson
  - Gson



Librerías para procesamiento de JSON

- En la actualidad existen varias librerías para transformar un objeto Java en una cadena JSON. Entre ellas:

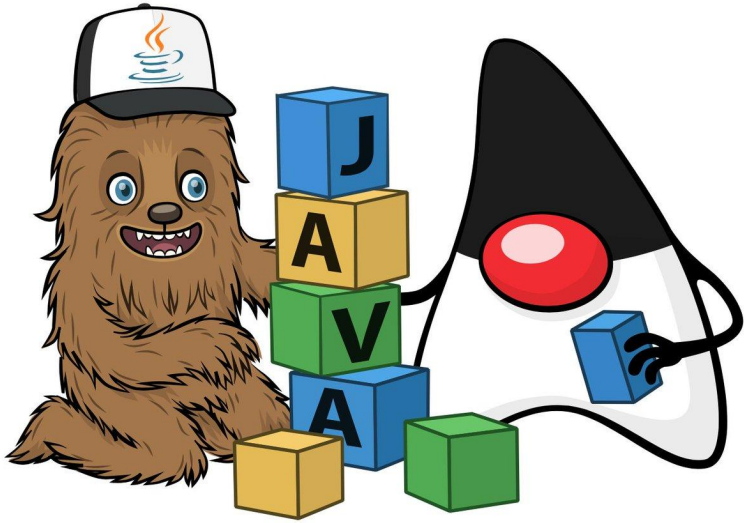
1) **Jackson:** es una librería de utilidad de Java que nos simplifica el trabajo de serializar (convertir un objeto Java en una cadena de texto con su representación JSON), y deserializar (convertir una cadena de texto con una representación de JSON de un objeto en un objeto real de Java) objetos JSON. Para ello usa básicamente la introspección de manera que si en el objeto JSON tenemos un atributo “name”, para la serialización buscará un método “getName()” y para la deserialización buscará un método “setName(String s)”.

Librerías para procesamiento de JSON (2)

2) **Gson**: el uso de esta librería se basa en el uso de una instancia de la clase Gson. Dicha instancia se puede crear de manera directa (new Gson()) para transformaciones sencillas o de forma más compleja con GsonBuilder para añadir distintos comportamientos.

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
  - ~~ObjectOutputStream~~
  - ~~ObjectInputStream~~
- ~~JSON~~
- ~~Librerías para procesamiento de JSON~~
  - **Jackson**
  - Gson



Jackson

- Escribir en archivo:

```
File file = new File(“mi_archivo.json”);

ObjectMapper mapper = new ObjectMapper();
Persona persona = new Persona();

//Object to JSON in file
mapper.writeValue(file, persona);
```

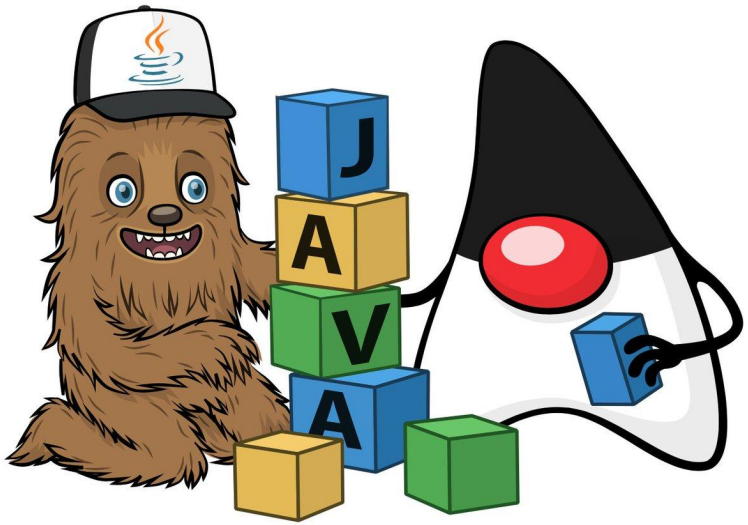
Jackson (2)

- Leer desde archivo:

```
File file = new File(“mi_archivo.json”);

ObjectMapper mapper = new ObjectMapper();

Persona p = mapper.readValue(file, Persona.class);

System.out.println(p);
```

Agenda

- ~~Serialización~~
- ~~Interfaz Serializable~~
- ~~Serial Version UID~~
- ~~Modificador transient~~
- ~~Serialización y archivos~~
  - ~~ObjectOutputStream~~
  - ~~ObjectInputStream~~
- ~~JSON~~
- ~~Librerías para procesamiento de JSON~~
  - ~~Jackson~~
  - **Gson**



Gson

- Escribir en archivo:

```
File file = new File(“mi_archivo.json”);

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file));

Persona persona = new Persona("Juan", "Gson");

Gson gson = new Gson();

gson.toJson(persona, Persona.class, bufferedWriter);
```

Gson (2)

- Leer desde archivo:

```
File file = new File(“mi_archivo.json”);

BufferedReader bufferedReader = new BufferedReader(new FileReader(file));

Gson gson = new Gson();

Persona persona = gson.fromJson(bufferedReader, Persona.class);

System.out.println(persona);
```

Bibliografía oficial

- https://www.tutorialspoint.com/java/java_serialization.htm
- https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html
- https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html
- https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html
- http://jackson.codehaus.org/
- https://sites.google.com/site/gson/gson-user-guide