Programación 2

JSON
Serializando objetos compuestos

Repaso - JSON

JSON es un formato ligero de intercambio de datos basado en texto que se utiliza para representar objetos y estructuras de datos. Aunque se originó en el contexto de JavaScript, se ha convertido en un estándar ampliamente adoptado y se utiliza en una variedad de lenguajes de programación.

Repaso - Formato / composición

JSON está constituído por dos estructuras:

A. Una colección de pares de `"clave" : valor`. Dependiendo del lenguaje esto es conocido como un **objeto**, registro, estructura, **diccionario**, tabla hash, lista de claves o un arreglo asociativo.

B. Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Ejemplo de JSON

```
[
    {
        "nombre": "Juan",
        "apellido": "Perez",
        "edad": 30
    },
    {
        "nombre": "Pedro",
        "apellido": "González",
        "edad": 32
    }
]
```

Objeto persona

Objeto persona

Repaso - Características

Sintaxis Ligera: JSON utiliza una sintaxis simple y concisa para representar datos.

Estructuras de Datos Simples: Soporta tipos de datos como objetos, arreglos, cadenas, números, booleanos y valores nulos.

Facilidad de Lectura y Escritura: Es fácil de entender para los humanos y fácil de analizar y generar en aplicaciones.

Amplia Adopción: Se utiliza en servicios web, APIs y aplicaciones en línea debido a su eficiencia y simplicidad.

Trabajando con JSON en python

Para trabajar con JSON simplemente debemos importar la librería que python trae incorporada para codificar y decodificar archivos .json:

```
import json
```

La librería tiene dos métodos principales:

- **json.dumps(...)**: Convierte objetos nativos de Python en cadenas JSON. (dumps es abreviatura de '*dump to string*'). Lo usaremos para serializar un objeto.
- **json.loads(...)**: Convierte cadenas JSON en objetos Python.
(loads es la abreviación de '*load string'*). Lo usaremos en el método para Reconstruir un objeto con información de un string JSON. (Deserializacion)

Ejemplo de uso básico

```python
import json

# Convertir diccionario Python a JSON
persona = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
json_data_str = json.dumps(persona)
print(json_data_str)  # {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}

# Convertir JSON a diccionario Python
persona_dicc = json.loads(json_data_str)
print(persona_dicc["nombre"])  # Juan
```

Serialización de Objetos simples

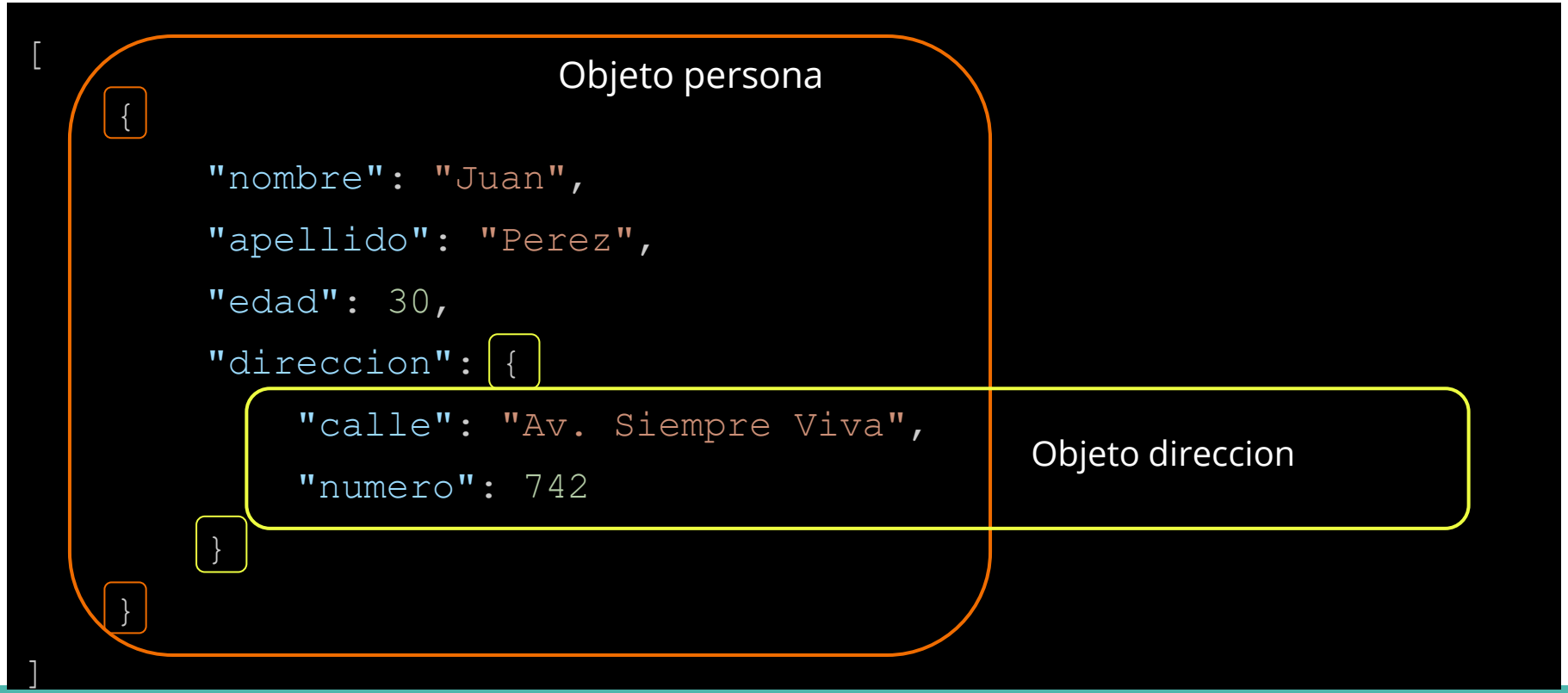
```python
import json

class Persona:
    def __init__(self, dni:int, nombre:str, apellido:str):
        if not isinstance(dni, int):
            raise ValueError("El DNI debe ser un entero.")
        if not isinstance(nombre, str) :
            raise ValueError("El nombre debe ser un string.")
        if not isinstance(apellido, str):
            raise ValueError("El apellido debe ser un string.")
        self.__dni = dni
        self.__nombre = nombre
        self.__apellido = apellido

    def to_json(self):
        dicc_persona = {"dni": self.__dni, "nombre": self.__nombre, "apellido": self.__apellido}
        return json.dumps(dicc_persona, ensure_ascii=False)
```

```python
import json

class Persona:
    def __init__(self, dni:int, nombre:str, apellido:str):
        if not isinstance(dni, int):
            raise ValueError("El DNI debe ser un entero.")
        if not isinstance(nombre, str) :
            raise ValueError("El nombre debe ser un string.")
        if not isinstance(apellido, str):
            raise ValueError("El apellido debe ser un string.")
        self.__dni = dni
        self.__nombre = nombre
        self.__apellido = apellido

    def to_json(self):
        dicc_persona = {"dni": self.__dni, "nombre": self.__nombre, "apellido": self.__apellido}
        return json.dumps(dicc_persona, ensure_ascii=False)

    @classmethod
    def from_json(cls, json_data):
        datos = json.loads(json_data)
        return cls(datos["dni"], datos["nombre"], datos["apellido"])
```

Serialización

Deserialización/reconstrucción

Ejemplo de JSON de objetos con atributos de tipo clase



Problemas de la serialización de objetos con referencias a otros objetos

1. **Serialización en Cadena dentro de Cadena**:

   Cuando llamamos a json.dumps() en cada nivel de anidación (por ejemplo, en los objetos relacionados), estamos generando un problema de "JSON dentro de JSON". Cada llamada a json.dumps() convierte el objeto en una cadena JSON, pero si luego esa cadena JSON se incluye en otro json.dumps(), aparecerán caracteres de escape adicionales (como \), creando una estructura que es difícil de leer y trabajar. Esto resulta en un JSON final que contiene demasiados caracteres de escape, haciendo que parezca desordenado y difícil de procesar.

Problemas de la serialización de objetos con referencias a otros objetos

2. **Pérdida de Flexibilidad en la Manipulación de Objetos**:

   Si cada atributo de tipo objeto ya está serializado en JSON, no se pueden acceder a los datos de esos objetos de manera sencilla. Esto limita la capacidad de manipular o extraer información del diccionario resultante en su forma de Python, porque en lugar de anidar diccionarios u otros objetos JSON directamente, tienes cadenas de texto. Esto hace que se pierda el beneficio de la estructura de datos y obliga a deserializar cada nivel de JSON para acceder a los datos.

Problemas de la serialización de objetos con referencias a otros objetos

3. **Duplicación de Trabajo de Serialización:**

   Cada llamada a json.dumps() implica un procesamiento adicional de serialización. Esto no solo es innecesario, sino que también consume recursos y aumenta la complejidad, especialmente si los datos deben ser reutilizados en varios lugares del programa.

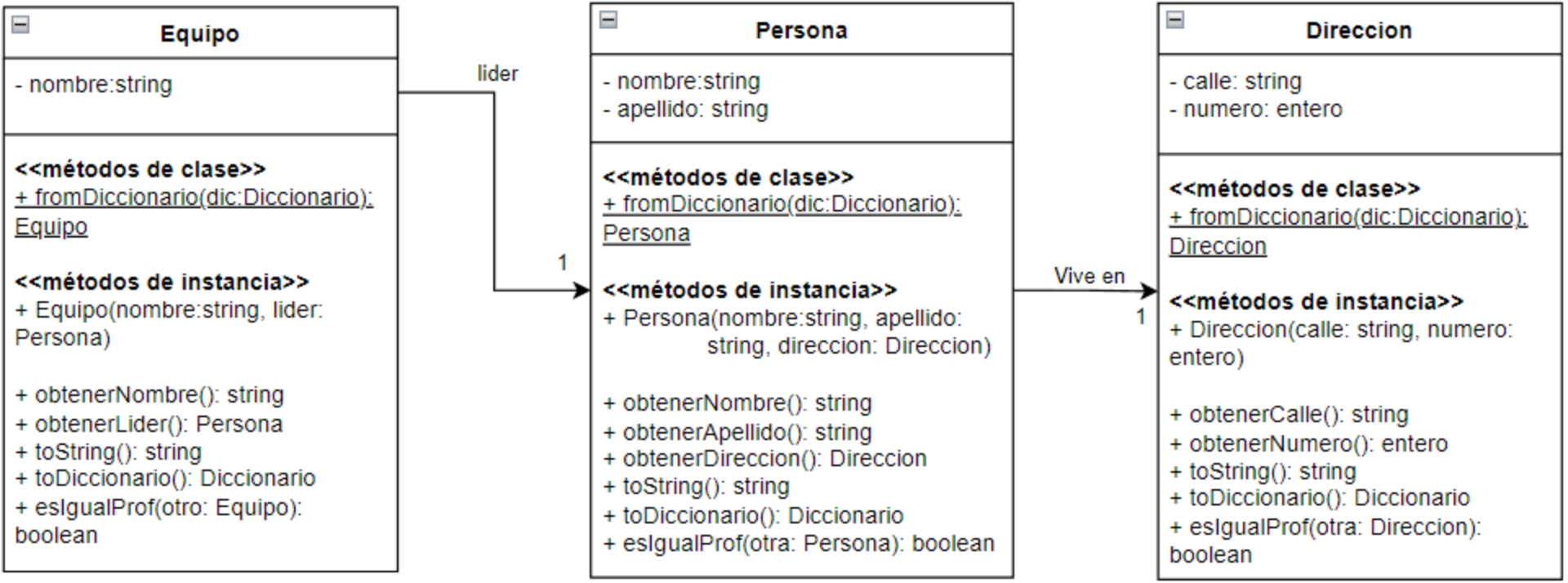
Problemas de la serialización - Solución

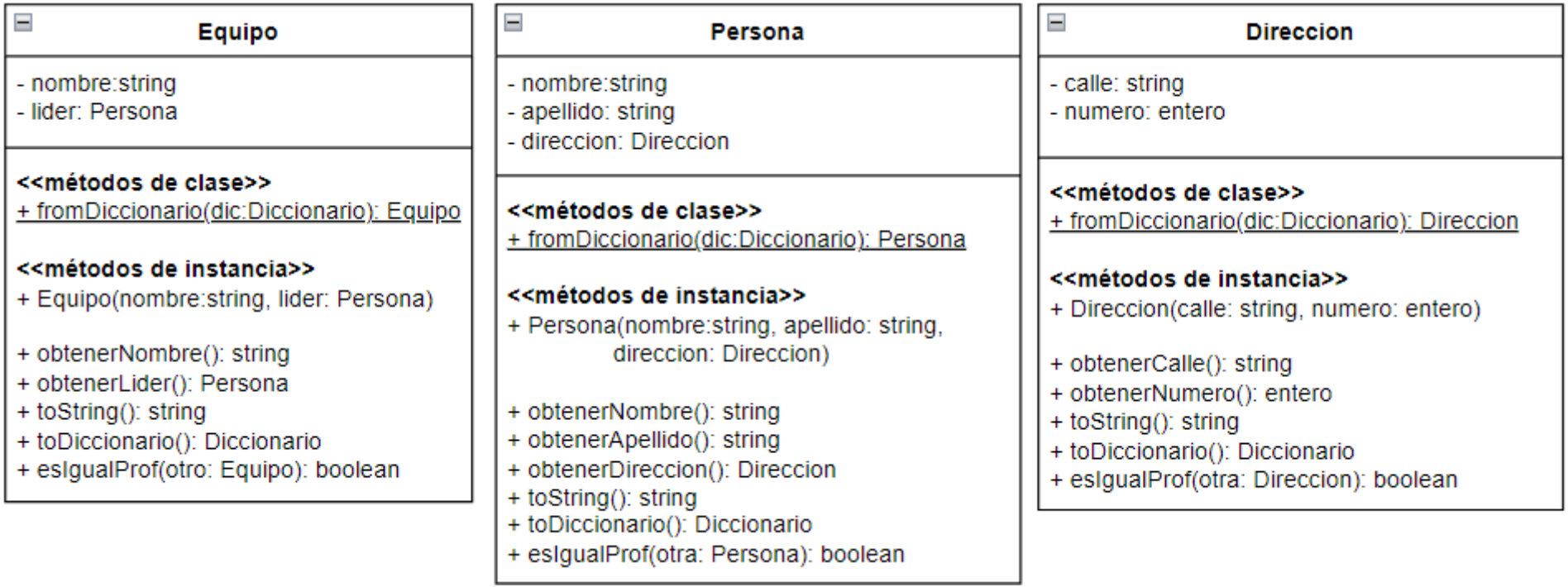
Usar Diccionarios en lugar de json.dumps() en cada Objeto.

En lugar de serializar los objetos anidados directamente con `json.dumps()`, es mejor reemplazar cada método `toJson` en las clases por un método `toDict()` ó `toDiccionario()` que retorne un diccionario en lugar de una cadena JSON.

De esta manera, el método `toDiccionario()` del objeto principal puede simplemente acumular todos los datos en un diccionario grande que finalmente puede ser serializado una sola vez por la clase que esté utilizando el objeto. Es decir, el objeto compuesto llamará al método `toDiccionario()` de sus atributos de tipo clase, obteniendo un diccionario por cada atributo de tipo clase.

Ejemplo de aplicación




Equipo
- nombre:string
- lider: Persona
<<métodos de clase>>
+ fromDiccionario(dic:Diccionario): Equipo
<<métodos de instancia>>
+ Equipo(nombre:string, lider: Persona)
+ obtenerNombre(): string
+ obtenerLider(): Persona
+ toString(): string
+ toDiccionario(): Diccionario
+ esIgualProf(otro: Equipo): boolean
Persona
- nombre:string
- apellido: string
- direccion: Direccion
<<métodos de clase>>
+ fromDiccionario(dic:Diccionario): Persona
<<métodos de instancia>>
+ Persona(nombre:string, apellido: string,
direccion: Direccion)
+ obtenerNombre(): string
+ obtenerApellido(): string
+ obtenerDireccion(): Direccion
+ toString(): string
+ toDiccionario(): Diccionario
+ esIgualProf(otra: Persona): boolean
Direccion
- calle: string
- numero: entero
<<métodos de clase>>
+ fromDiccionario(dic:Diccionario): Direccion
<<métodos de instancia>>
+ Direccion(calle: string, numero: entero)
+ obtenerCalle(): string
+ obtenerNumero(): entero
+ toString(): string
+ toDiccionario(): Diccionario
+ esIgualProf(otra: Direccion): boolean

```python
import json
class Direccion:
    #metodos de clase ------------------------------------------------------
    @classmethod
    def fromDiccionario(cls, data: dict)->"Direccion":
        if not isinstance(data, dict):
            raise ValueError("El parámetro data debe ser un diccionario.")
        return cls(data["calle"], data["numero"])

    #constructor y métodos de instancia ------------------------------------
    def __init__(self, calle, numero):
        if not isinstance(calle, str) or calle == "" or calle.isspace():
            raise ValueError("La calle debe ser un string válido.")
        if not isinstance(numero, int) or numero < 0:
            raise ValueError("El número debe ser un entero positivo.")
        self.__calle = calle
        self.__numero = numero

    def obtenerCalle(self)->str:
        return self.__calle

    def obtenerNumero(self)->int:
        return self.__numero

    def esIgualProf(self, otra:"Direccion")->bool:
        return self.__calle == otra.obtenerCalle() and self.__numero == otra.obtenerNumero()

    def toDiccionario(self):
        return {"calle":self.__calle, "numero":self.__numero}
```

```python
class Persona:
    #metodos de clase ------------------------------------------------------
    @classmethod
    def fromDiccionario(cls, data:dict)->"Persona":
        if not isinstance(data, dict):
            raise ValueError("El parámetro data debe ser un diccionario.")
        return cls(data["nombre"], data["apellido"], Direccion.fromDiccionario(data["direccion"]))

    #constructor y métodos de instancia -----------------------------------
    def __init__(self, nombre:str, apellido:str, direccion:Direccion):
        if not isinstance(nombre, str) or nombre == "" or nombre.isspace():
            raise ValueError("El nombre debe ser un string válido.")
        if not isinstance(apellido, str) or apellido == "" or apellido.isspace():
            raise ValueError("El apellido debe ser un string válido.")
        if not isinstance(direccion, Direccion):
            raise ValueError("La dirección debe ser una instancia de la clase direccion.")
        self.__nombre = nombre
        self.__apellido = apellido
        self.__direccion = direccion

    def obtenerNombre(self)->str:
        return self.__nombre

    def obtenerApellido(self)->str:
        return self.__apellido

    def obtenerDireccion(self)->Direccion:
        return self.__direccion

    def esIgualProf(self, otra:"Persona")->bool:
        return self.__nombre == otra.obtenerNombre() and self.__apellido == otra.obtenerApellido() and
self.__direccion.esIgualProf(otra.obtenerDireccion())

    def toDiccionario(self):
        return {"nombre":self.__nombre, "apellido":self.__apellido, "direccion":self.__direccion.toDiccionario()}
```

```python
class Equipo:
    #metodos de clase ------------------------------------------------------
    @classmethod
    def fromDiccionario(cls, data: dict)->"Equipo":
        if not isinstance(data, dict):
            raise ValueError("El parámetro data debe ser un diccionario.")
        return cls(data["nombre"], Persona.fromDiccionario(data["lider"]))

#constructor y métodos de instancia ------------------------------------
    def __init__(self, nombre:str, lider:Persona):
        if not isinstance(nombre, str) or nombre == "" or nombre.isspace():
            raise ValueError("El nombre debe ser un string válido.")
        if not isinstance(lider, Persona):
            raise ValueError("El líder debe ser una instancia de la clase Persona.")
        self.__nombre = nombre
        self.__lider = lider

    def obtenerNombre(self)->str:
        return self.__nombre

    def obtenerLider(self)->Persona:
        return self.__lider

    def toDiccionario(self):
        return {"nombre":self.__nombre, "lider":self.__lider.toDiccionario()}

    def esIgualProf(self, otro:"Equipo")->bool:
        return self.__nombre == otro.obtenerNombre() and self.__lider.esIgualProf(otro.obtenerLider())
```

```
class Tester:
    @staticmethod
    def test():
        lista_direcciones =[]
        lista_personas = []
        lista_equipos = []
        dir1 = Direccion("Av. Siempre Viva", 742)
        dir2 = Direccion("Av. Siempre Viva", 730)
        homero = Persona("Homero", "Simpson", dir1)
        ned = Persona("Ned", "Flanders", dir2)
        equipo_homero = Equipo("Los Simpsons", homero)
        equipo_ned = Equipo("Los Flanders", ned)

        lista_direcciones.append(dir1)
        lista_direcciones.append(dir2)
        lista_personas.append(homero)
        lista_personas.append(ned)
        lista_equipos.append(equipo_homero)
        lista_equipos.append(equipo_ned)
```

```python
print("print de objeto.toDiccionario()")
print(dir1.toDiccionario())
print(homero.toDiccionario())
print(equipo_homero.toDiccionario())
print("-----------------------------------------------\n")
print("Convierto equipo.toDiccionario() a JSON con json.dumps(obj.toDiccionario(), indent=4)")
str_equipo_homero = json.dumps(equipo_homero.toDiccionario(), indent=4)
print("print de JSON str_equipo_homero")
print(str_equipo_homero)
print("print de type(str_equipo_homero)")
print(type(str_equipo_homero))

print("-----------------------------------------------\n")
print("Convierto JSON a objeto con json.loads(str_equipo_homero)")
eq_recreado = Equipo.fromDiccionario(json.loads(str_equipo_homero))
print("print de obj.toDiccionario() de un objeto creado a partir de un JSON")
print(eq_recreado.toDiccionario())

print("-----------------------------------------------\n")
print("Guardo los equipos en un archivo JSON")
with open("equipos.json", "w") as file:
    diccionarios_equipos = [equipo.toDiccionario() for equipo in lista_equipos]
    json.dump(diccionarios_equipos, file, ensure_ascii=False, indent=4)
```

```
Salida (coloreada para mejorar legibilidad):
print de objeto.toDiccionario()
{'calle': 'Av. Siempre Viva', 'numero': 742}
{'nombre': 'Homero', 'apellido': 'Simpson', 'direccion': {'calle': 'Av. Siempre Viva', 'numero': 742}}
{'nombre': 'Los Simpsons', 'lider': {'nombre': 'Homero', 'apellido': 'Simpson', 'direccion': {'calle': 'Av. Siempre Viva', 'numero': 742}}}
------------------------------------------------

Convierto equipo.toDiccionario() a JSON con json.dumps(obj.toDiccionario(), indent=4)
print del JSON generado: str_equipo_homero
{
    "nombre": "Los Simpsons",
    "lider": {
        "nombre": "Homero",
        "apellido": "Simpson",
        "direccion": {
            "calle": "Av. Siempre Viva",
            "numero": 742
        }
    }
}
print de type(str_equipo_homero)
<class 'str'>
------------------------------------------------

Convierto JSON a objeto con json.loads(str_equipo_homero)
print de obj.toDiccionario() de un objeto creado a partir de un JSON
{'nombre': 'Los Simpsons', 'lider': {'nombre': 'Homero', 'apellido': 'Simpson', 'direccion': {'calle': 'Av. Siempre Viva', 'numero': 742}}}
------------------------------------------------

Guardo los equipos en un archivo JSON
```

Archivo Json “equipos.json”:

```json
[
    {
        "nombre": "Los Simpsons",
        "lider": {
            "nombre": "Homero",
            "apellido": "Simpson",
            "direccion": {
                "calle": "Av. Siempre Viva",
                "numero": 742
            }
        }
    },
    {
        "nombre": "Los Flanders",
        "lider": {
            "nombre": "Ned",
            "apellido": "Flanders",
            "direccion": {
                "calle": "Av. Siempre Viva",
                "numero": 730
            }
        }
    }
]
```