Programacion 2

Asociación y dependencia entre clases

Asociación entre clases

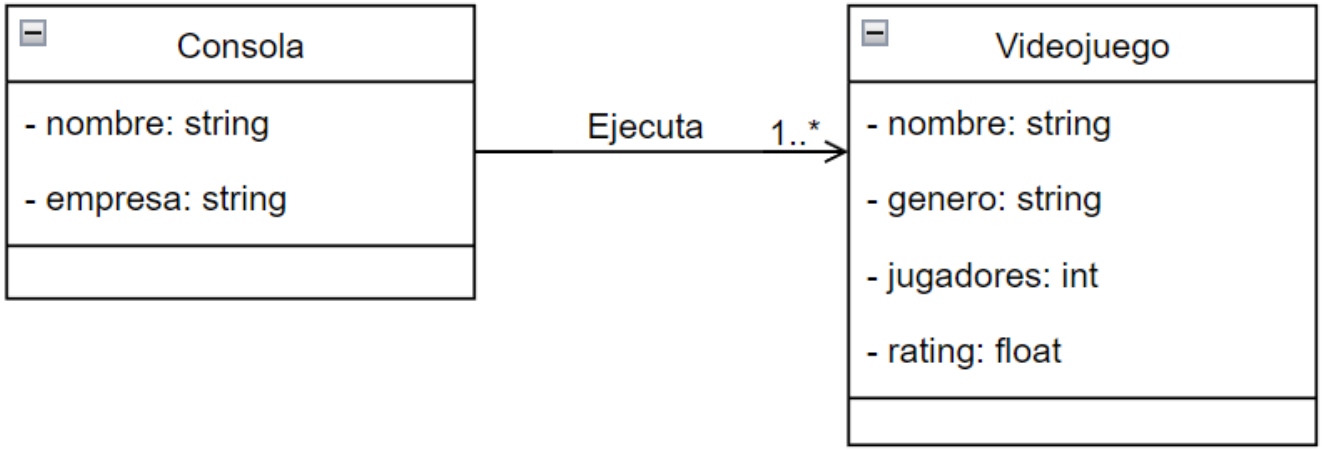
En la programación orientada a objetos el punto de partida para la construcción de un sistema es un proceso de abstracción y clasificación.

Los **objetos de una clase** se caracterizan por los mismos atributos y comportamiento, pero además **comparten entre sí el mismo modo de relacionarse con objetos de otras clases**.

Un objeto está **asociado** a otro objeto, si **tiene un** atributo de su clase.

La relación entre los objetos provoca una relación entre las clases, que se dicen **asociadas**.

Asociación entre clases



El diagrama modela las clases **consola** y **videojuego**, y la relación que presentan.

En este caso, una consola puede ejecutar 1 o más juegos.

Recordando la clase Videojuego

Videojuego
<<atributos de instancia>> - nombre : String - genero : String - jugadores : entero - rating : float
<<constructor>> + Videojuego (nombre: string, genero: string, cantJugadores: entero, rating: float) **<<consultas>>** ... obtener ... + masPopular(videojuego: Videojuego): Videojuego + esIgualQue(videojuego: Videojuego): boolean **<<comandos>>** + copiarValores(videojuego: Videojuego) + clonar(): Videojuego

La clase Videojuego es la que modelamos en la presentación anterior.

Consola
<<atributos de instancia>> nombre: string empresa: string juegos: **list<Videojuego>**
<<constructor>> + Consola(nombre: String, empresa: String) + Consola(nombre: String, empresa: String, juegos: list<Videojuego>) **<<consultas triviales>>** + obtenerNombre(): String + obtenerEmpresa(): String + obtenerJuegos(): **list<Videojuego>** **<<consultas>>** + tieneJuego(**videojuego: Videojuego**): boolean + obtenerCantidadJuegos(): entero + toString(): String **<<comandos>>** + agregarJuego(**videojuego: Videojuego**) + eliminarJuego(**videojuego: Videojuego**)

Clase Consola

La clase Consola tiene los atributos nombre, empresa y juegos.

Juegos es una lista de tipo clase Videojuego. Como aparece sólo en el segundo constructor, será un parámetro opcional en el constructor de la clase.

La clase también ofrece servicios que reciben parámetros de tipo clase Videojuego.

Aclaración

En los próximos ejemplos, el código de los métodos estará sin validaciones de tipo y rango para facilitar la comprensión del concepto a explicar.

Recordemos que las clases deben validar los datos que reciben antes de realizar operaciones con ellos.

Caso de estudio: consola

```
from Videojuego import Videojuego

class Consola:
    # constructor y atributos de instancia
    def __init__(self, nombre: str, empresa:str, juegos:list[Videojuego]=None):
        """Requiere que el nombre y la empresa no sean vacíos."""
        self.__nombre = nombre
        self.__empresa = empresa
        self.__juegos = []
        if juegos != None:
            if isinstance(juegos, Videojuego):
                self.agregarJuego(juegos)
            elif isinstance(juegos, list):
                for juego in juegos:
                    if isinstance(juego, Videojuego):
                        self.agregarJuego(juego)
```

Maneja el caso en que reciba un Videojuego

Maneja el caso en que reciba una lista de Videojuegos

La clase Consola **tieneUn** atributo de clase Videojuego.
La clase Consola puede acceder a cualquiera de los **miembros públicos** de la clase Videojuego.
Los atributos están encapsulados, no son accesibles, fuera de la clase.

Caso de estudio: consola

```
    #consultas triviales
    def obtenerNombre(self)->str:
        """Devuelve el nombre de la consola."""
        return self.__nombre

    def obtenerEmpresa(self)->str:
        """Devuelve la empresa de la consola."""
        return self.__empresa

    def obtenerJuegos(self)->list:
        """Devuelve la lista de juegos de la consola."""
        return self.__juegos
```

Caso de estudio: consola

```
#consultas
    def tieneJuego(self, juego:Videojuego)->bool:
        """Requiere que 'juego' esté ligado (no sea None).
        Devuelve True si la consola tiene el juego."""
        return juego in self.__juegos

    def obtenerCantidadJuegos(self)->int:
        """Devuelve la cantidad de juegos de la consola."""
        return len(self.__juegos)

    def __str__(self)->str:
        """Devuelve una representación de string de la consola."""
        return f"Nombre: {self.__nombre}, Empresa: {self.__empresa}, Cantidad de
juegos: {self.obtenerCantidadJuegos()}"
```

Caso de estudio: consola

```
    #comandos
    def agregarJuego(self, juego:Videojuego):
        """Requiere que 'juego' esté ligado (no sea None).
        Agrega un juego a la consola."""
        if not self.tieneJuego(juego):
            self.__juegos.append(juego)

    def eliminarJuego(self, juego:Videojuego):
        """Requiere que 'juego' esté ligado (no sea None).
        Elimina un juego de la consola."""
        if self.tieneJuego(juego):
            self.__juegos.remove(juego)
```

Caso de estudio: consola

```
class TesterConsola:
    @staticmethod
    def test():
        print("TESTEANDO LA CLASE CONSOLA")
        juego1 = Videojuego("Mario Bros", "Plataforma", 1000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 1000, 4.8)
        juego3 = Videojuego("FIFA 22", "Deportes", 2000, 4.2)
        consola = Consola("Nintendo Switch", "Nintendo", [juego1, juego2])
        print(f"Nombre: {consola.obtenerNombre()}")
        print(f"Empresa: {consola.obtenerEmpresa()}")
        print(f"¿Tiene el juego 1? {consola.tieneJuego(juego1)}")
        print(f"¿Tiene el juego 3? {consola.tieneJuego(juego3)}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("Agrego el juego 3")
        consola.agregarJuego(juego3)
        print("Juegos:")
        for juego in consola.obtenerJuegos():
            print(f"{juego} en la consola {consola.obtenerNombre()}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("Elimino el juego 2")
        consola.eliminarJuego(juego2)
        print(f"Juegos: {consola.obtenerJuegos()}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("¿Tiene el juego 2? ", consola.tieneJuego(juego2))
        print(consola)
        print("FIN DE LOS TESTS")
```

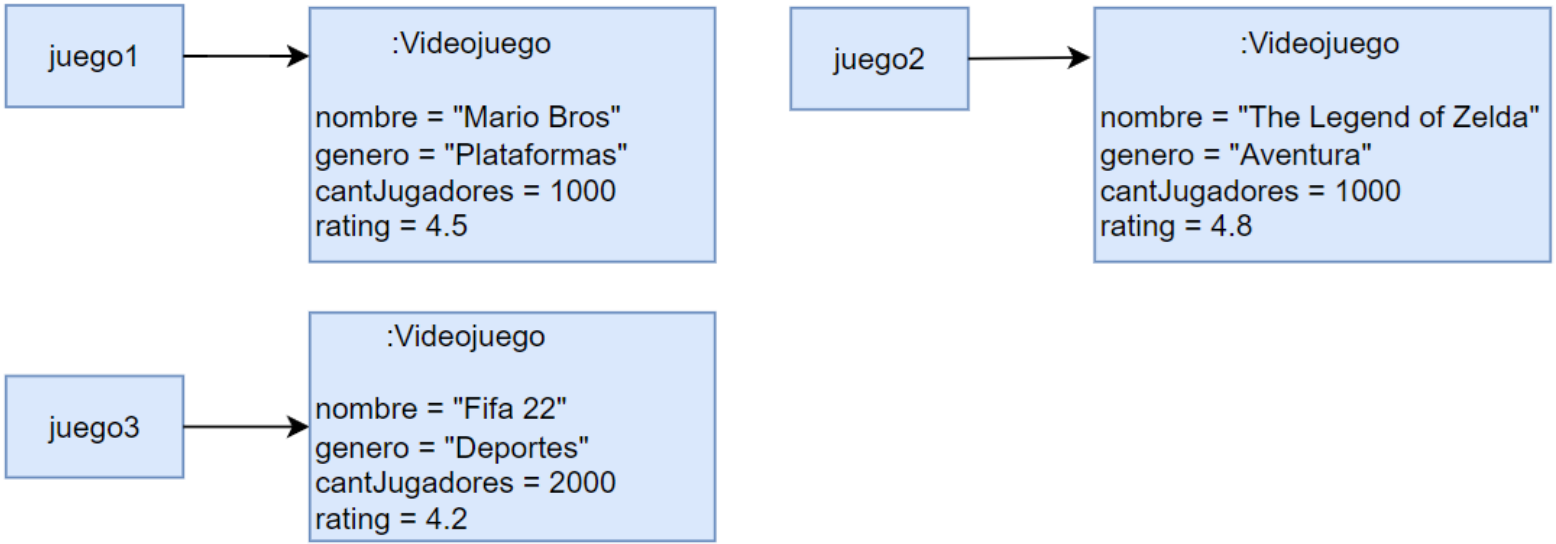
La clase tester define casos que buscan verificar los servicios de la clase Consola

Caso de estudio: consola

```
class TesterConsola:
    @staticmethod
    def test():
        print("TESTEANDO LA CLASE CONSOLA")
        juego1 = Videojuego("Mario Bros", "Plataforma", 1000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 1000, 4.8)
        juego3 = Videojuego("FIFA 22", "Deportes", 2000, 4.2)
```

La clase **TesterConsola** define su método estático **test()** que contiene las pruebas sobre los servicios de Consola.
Comienza creando tres objetos Videojuego.

Caso de estudio: consola - Diagrama de objetos

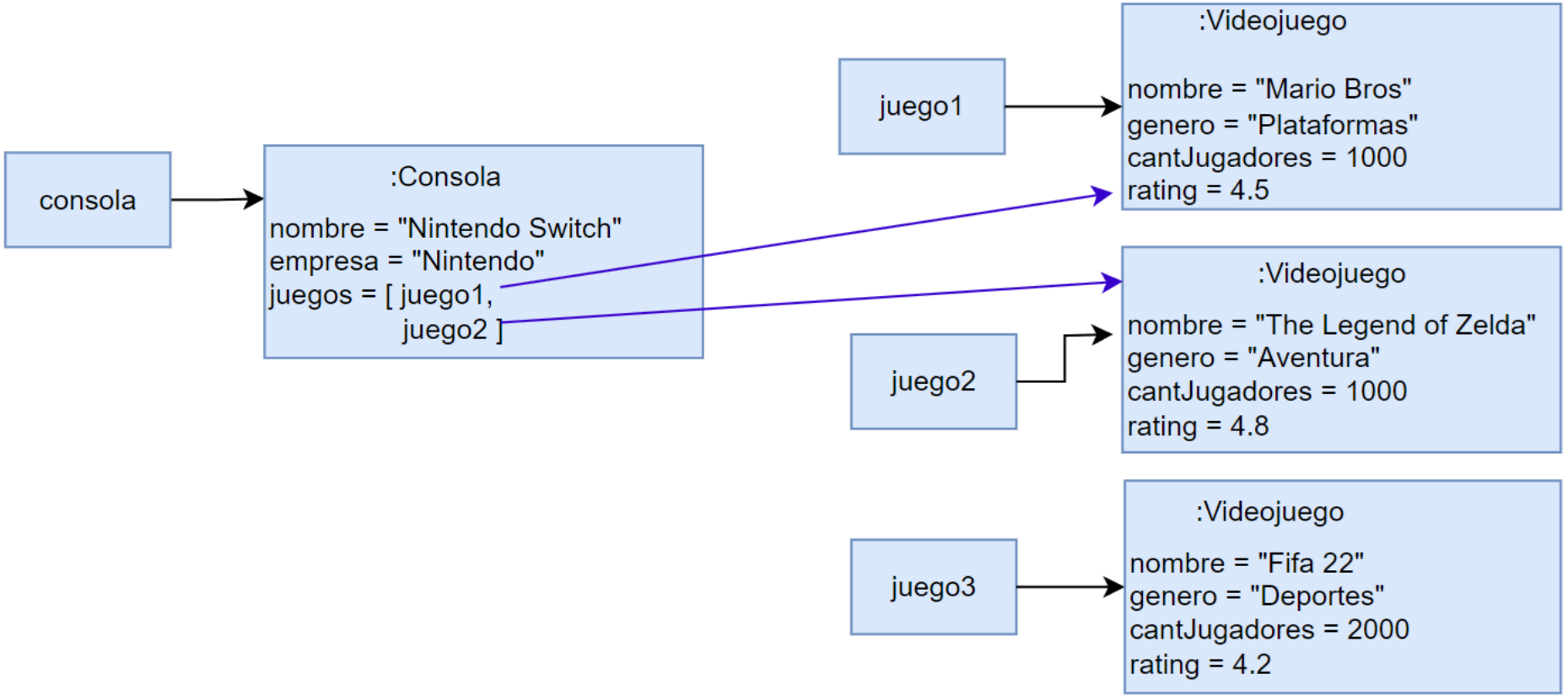


Caso de estudio: consola

```
consola = Consola("Nintendo Switch", "Nintendo", [juego1, juego2])
print(f"Nombre: {consola.obtenerNombre()}")
print(f"Empresa: {consola.obtenerEmpresa()}")
```

Crea un objeto Consola enviándole al constructor los parámetros Nombre, Empresa y una lista de Videojuegos.
Luego accede a los valores del objeto ***'consola'*** mediante los servicios provistos por la clase. Los **atributos están encapsulados**.

Caso de estudio: consola - Diagrama de objetos

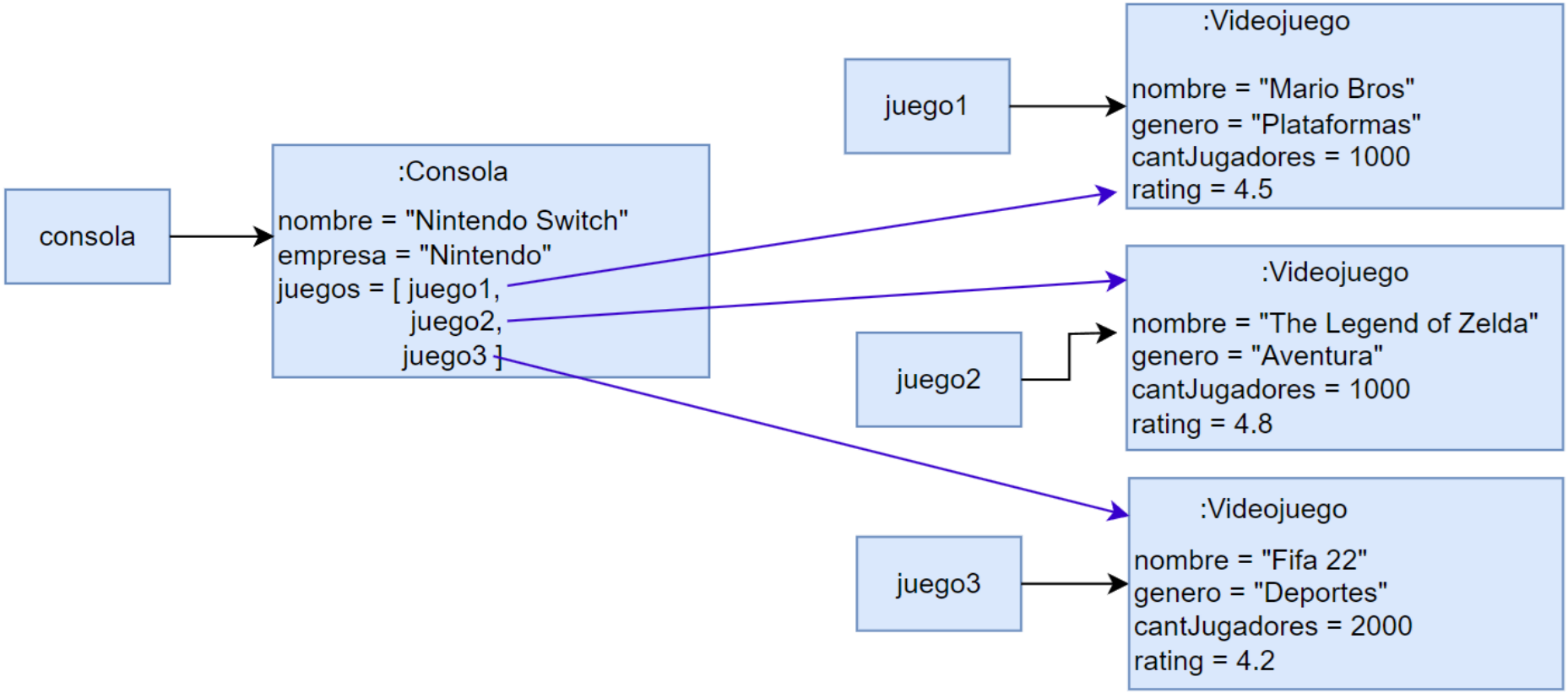


Caso de estudio: consola

```
print(f"¿Tiene el juego 1? {consola.tieneJuego(juego1)}")
print(f"¿Tiene el juego 3? {consola.tieneJuego(juego3)}")
print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
print("Agrego el juego 3")
consola.agregarJuego(juego3)
```

Consulta valores del estado interno mediante los métodos provistos por la clase.
Modifica el estado interno del objeto ***'consola'*** agregando otro juego mediante los servicios provistos por la clase.
Los **atributos están encapsulados**.

Caso de estudio: consola - Diagrama de objetos



Caso de estudio: consola

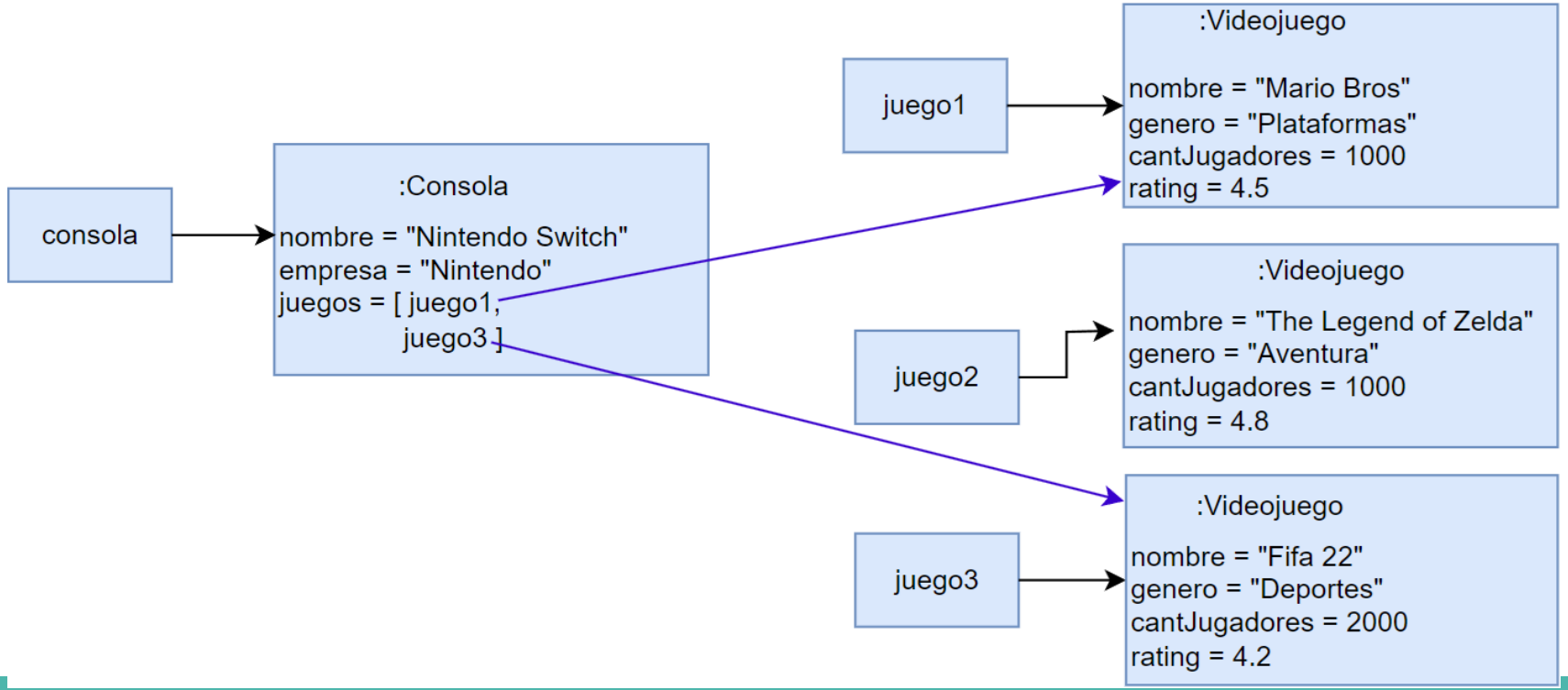
```
print("Juegos:")
for juego in consola.obtenerJuegos():
    print(f"{juego} en la consola {consola.obtenerNombre()}")
print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
```

La clase TesterConsola pide la lista de videojuegos de la consola, e itera sobre ella enviando el mensaje **__str__()** a cada objeto de clase Videojuego en la lista.

El método **__str__()** de una clase se utiliza cada vez que hagamos un **print()** o una conversión a string **str()** del objeto.

Caso de estudio: consola - Diagrama de objetos

```
print("Elimino el juego 2")
consola.eliminarJuego(juego2)
```



Caso de estudio: consola

```
print(f"Juegos: {consola.obtenerJuegos()}")
print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
print("¿Tiene el juego 2? ", consola.tieneJuego(juego2))
print(consola)
print("FIN DE LOS TESTS")
```

consola.obtenerJuegos() devuelve la **lista** de videojuegos.
Cuando queramos imprimir la lista de objetos Videojuego notaremos que no se envió el mensaje **__str__()** a cada objeto de la lista.
¿Por qué?: Cuando Python arma la representación de la lista para mostrar en pantalla, necesita una versión precisa y no ambigua de cada objeto en ella y para esto intenta enviar el mensaje **__repr__()** a la clase, si el método está implementado lo ejecuta, y si no está implementado python usa la representación por defecto (dirección de memoria) y la salida será una cadena que incluye el tipo del objeto y su dirección de memoria, algo como **<MiClase object at 0x7f0c8b0d3a60>**.

Caso de estudio: consola

```
TESTEANDO LA CLASE CONSOLA
Nombre: Nintendo Switch
Empresa: Nintendo
¿Tiene el juego 1? True
¿Tiene el juego 3? False
Cantidad de juegos: 2
Agrego el juego 3
Juegos:
Nombre: Mario Bros, Género: Plataforma, Cantidad de jugadores: 1000, Rating: 4.5 en la consola Nintendo Switch
Nombre: The Legend of Zelda, Género: Aventura, Cantidad de jugadores: 1000, Rating: 4.8 en la consola Nintendo
Switch
Nombre: FIFA 22, Género: Deportes, Cantidad de jugadores: 2000, Rating: 4.2 en la consola Nintendo Switch
Cantidad de juegos: 3
Elimino el juego 2
Juegos: [<Videojuego.Videojuego object at 0x0000013630CC13D0>, <Videojuego.Videojuego object at
0x0000013630CC1400>]
Cantidad de juegos: 2
¿Tiene el juego 2? False
Nombre: Nintendo Switch, Empresa: Nintendo, Cantidad de juegos: 2
FIN DE LOS TESTS
```

métodos `__str__` y `__repr__`

- `__str__` (String Representation) :
  - Propósito: El método *__str__* está diseñado para devolver una representación "legible" o informal del objeto. Es el método que se llama cuando usamos la función *print()* o *str()* en una instancia de la clase.
  - Ejemplo de uso: Ideal para mostrar información al usuario final.
- `__repr__` (Official Representation) :
  - Propósito: El método *__repr__* debe devolver una representación oficial del objeto que sea precisa y no ambigua. La intención es que esta representación sea útil para los desarrolladores, *permitiendo recrear el objeto utilizando el código generado por __repr__*.
  - Ejemplo de uso: Es utilizado por la función *repr()* y cuando inspeccionamos objetos en un entorno interactivo (por ejemplo, en el REPL).
- Comparación:
  - `__str__`: Enfocado en ser amigable y legible para el usuario.
  - `__repr__`: Enfocado en ser exacto y útil para los desarrolladores, con la posibilidad de recrear el objeto.

métodos `__str__` y `__repr__`

Ya vimos que pasa si tenemos implementado el método **__str__** y no tenemos implementado el método **__repr__** .

¿Que pasa al revés? (implementamos **__repr__** y no implementamos **__str__**)

Python usará **__repr__** cuando se requiera la representación precisa y no ambigua. Y también lo usará como reemplazo de **__str__** si éste no está implementado!

Implementamos `__repr__` en Videojuego

```
    def __str__(self)->str:
        """Devuelve una representación de string del videojuego."""
        return f"Nombre: {self.__nombre}, Género: {self.__genero}, Cantidad de jugadores:
{self.__cantJugadores}, Rating: {self.__rating}"

    #agregamos el método __repr__
    def __repr__(self) -> str:
        """Devuelve una representación formal del objeto videojuego."""
        return f"Videojuego('{self.__nombre}', '{self.__genero}', {self.__cantJugadores},{self.__rating})"
```

Caso de estudio: consola

```
class TesterConsola:
    @staticmethod
    def test():
        print("TESTEANDO LA CLASE CONSOLA")
        juego1 = Videojuego("Mario Bros", "Plataforma", 1000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 1000, 4.8)
        juego3 = Videojuego("FIFA 22", "Deportes", 2000, 4.2)
        consola = Consola("Nintendo Switch", "Nintendo", [juego1, juego2])
        print(f"Nombre: {consola.obtenerNombre()}")
        print(f"Empresa: {consola.obtenerEmpresa()}")
        print(f"¿Tiene el juego 1? {consola.tieneJuego(juego1)}")
        print(f"¿Tiene el juego 3? {consola.tieneJuego(juego3)}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("Agrego el juego 3")
        consola.agregarJuego(juego3)
        print("Juegos:")
        for juego in consola.obtenerJuegos():
            print(f"{juego} en la consola {consola.obtenerNombre()}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("Elimino el juego 2")
        consola.eliminarJuego(juego2)
        print(f"Juegos: {consola.obtenerJuegos()}")
        print(f"Cantidad de juegos: {consola.obtenerCantidadJuegos()}")
        print("¿Tiene el juego 2? ", consola.tieneJuego(juego2))
        print(consola)
        print("FIN DE LOS TESTS")
```

Volvemos a ejecutar el test

Caso de estudio: consola

```
TESTEANDO LA CLASE CONSOLA
Nombre: Nintendo Switch
Empresa: Nintendo
¿Tiene el juego 1? True
¿Tiene el juego 3? False
Cantidad de juegos: 2
Agrego el juego 3
Juegos:
Nombre: Mario Bros, Género: Plataforma, Cantidad de jugadores: 1000, Rating: 4.5 en la consola Nintendo Switch
Nombre: The Legend of Zelda, Género: Aventura, Cantidad de jugadores: 1000, Rating: 4.8 en la consola Nintendo Switch
Nombre: FIFA 22, Género: Deportes, Cantidad de jugadores: 2000, Rating: 4.2 en la consola Nintendo Switch
Cantidad de juegos: 3
Elimino el juego 2
Juegos: [Videojuego('Mario Bros', 'Plataforma', 1000, 4.5), Videojuego('FIFA 22', 'Deportes', 2000, 4.2)]
Cantidad de juegos: 2
¿Tiene el juego 2?  False
Nombre: Nintendo Switch, Empresa: Nintendo, Cantidad de juegos: 2
FIN DE LOS TESTS
```

Clase cliente y clase proveedora

La clase Videojuego **brinda** servicios que la clase Consola **usa**. Se dice entonces que la clase Videojuego es **proveedora** y la clase Consola es su **cliente**.

La clase Consola puede implementarse conociendo **qué** hace la clase Videojuego, pero no interesa **cómo** lo hace.

La clase Videojuego puede implementarse sin saber que va a ser usada por la clase Consola.

Cada clase debe conocer los servicios que brindan sus clases proveedoras, pero no necesita conocer quienes son sus clientes.

Cada clase va a ser verificada por separado y luego en conjunto con las demás clases relacionadas.

Para pensar

Una pista de carreras organiza un torneo donde varios autos compiten entre sí.

Cada auto tiene un piloto asignado, que es quien lo conduce, y una potencia específica medida en caballos de fuerza. El sistema necesita llevar el control de los autos y sus pilotos para realizar un seguimiento de quienes están compitiendo y cuán potente es cada auto.

Cada auto tiene un piloto asociado, que se identifica por su nombre, apellido, numero de inscripción y experiencia en carreras (medida en años).

El auto, por su parte, tiene una marca, un peso, una velocidad máxima y un valor que representa la potencia de su motor medido en caballos de fuerza.

- ¿Qué entidades aparecen en la descripción del problema?
- ¿Qué atributos podrías usar para describir tanto a los autos como a los pilotos?
- ¿Qué operaciones deberían poder realizar estas entidades?

Realiza el diagrama UML y luego la implementación en python