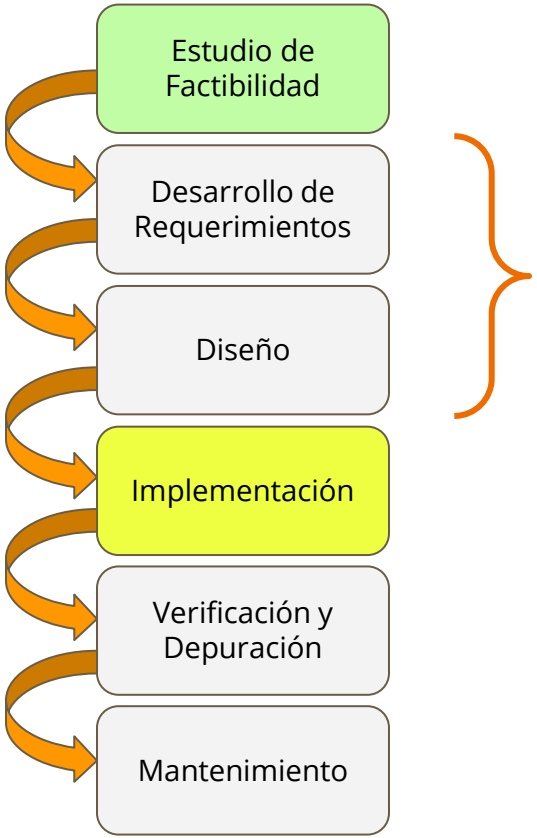
Programación 2

Clases y objetos. Estado interno. Parámetros tipo clase. identidad, igualdad y equivalencia

En esta clase

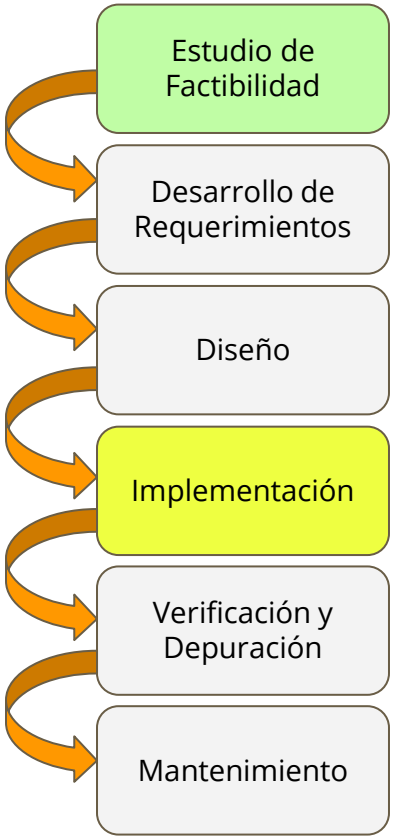
- Repaso. Las etapas del proceso de desarrollo
- Clases como tipos
- Variables y tipos
- Referencias y valores
- El estado interno de un objeto
- Alcance de las variables
- Asignación de variables de tipo clase
- Parámetros de tipo clase
- Resultados de tipo clase

Repaso - ciclo de vida



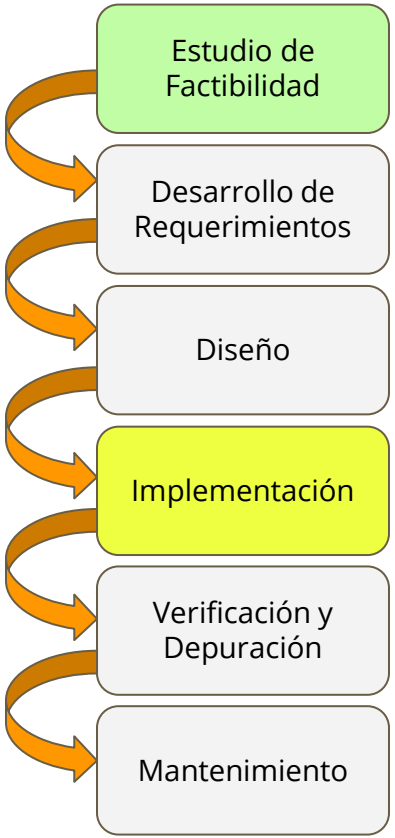
Durante el desarrollo de requerimientos y el diseño del sistema se identifican los **objetos del problema** y se los agrupa en **clases** en base a sus atributos y su comportamiento.

Repaso - ciclo de vida



La implementación consiste en escribir el **código** de cada clase usando un lenguaje de programación.

Repaso - ciclo de vida



Si el proceso se completa exitosamente el **producto final** es un **sistema de software** que satisface una **necesidad**, **oportunidad** o **idea**.

En la etapa de verificación se chequea que el comportamiento de cada clase sea correcto para un conjunto de casos de prueba y que el sistema satisfaga los requerimientos.

Repaso - Concepto de clase

Durante las etapas de desarrollo de requerimientos y diseño de un sistema orientado a objetos, se identifican los **objetos del problema** y se los agrupa en **clases**.

Cada clase es un patrón que define los **atributos** y el **comportamiento** de un conjunto de objetos del problema.

La colección de clases y sus relaciones se grafican en un **diagrama de clases**.

Repaso - Concepto de clase

En la **implementación** de un sistema orientado a objetos, cada **clase** del diagrama, se traduce en un **módulo de código** escrito en un lenguaje de programación.

Así, una clase, es un módulo de software que se desarrolla con cierta independencia de los demás y se integra luego de ser verificado, para formar parte del sistema.

En la **ejecución** del sistema se crean **objetos de software** cuyo **estado interno** y **servicios** dependen de su **clase**.

Repaso - Clases como tipos

Un **tipo de dato**, establece un **conjunto de valores** y un **conjunto de operaciones** que se aplican a esos valores.

Una clase que especifica un conjunto de atributos y servicios, define un **tipo de dato** a partir del cual es posible **declarar variables** y **crear objetos**.

El **conjunto de valores** queda determinado por los valores de los atributos.

El **conjunto de operaciones** lo definen los servicios provistos por la clase.

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.

Clases como tipos

Videojuego

<<atributos de instancia>>
- nombre : String
- genero : String
- jugadores : entero
- rating : float

<<constructor>>
+ Videojuego (nombre: string, genero: string, cantJugadores: entero, rating: float)

<<consultas>>
+ obtenerNombre(): String
+ obtenerGenero(): String
+ obtenerJugadores(): entero
+ obtenerRating(): flotante
+ esPopular(): boolean
+ mismoGenero(videojuego: Videojuego): boolean
+ masPopularQue(videojuego: Videojuego): boolean
+ toString(): String
+ masPopular(videojuego: Videojuego): Videojuego

Requiere 0 <= rating <= 5

Requiere cantJugadores > 0, es el número de jugadores registrados en el juego

esPopular() devuelve True si rating >= 3.5 y jugadores > 10000

mismoGenero retorna True si el objeto que recibe el mensaje tiene el mismo género que el videojuego recibido en el parámetro

masPopularQue retorna True si el objeto que recibe el mensaje tiene mas jugadores y más rating que el objeto recibido por parámetro

masPopular retorna el objeto más popular

Clases como tipos

```
class Videojuego:
    # constructor y atributos de instancia
    def __init__(self, nombre:str, genero: str, cantJugadores:int, rating: float):
        """Requiere cantJugadores >= 0.
        Requiere rating >= 0 y rating <= 5."""
        self.__nombre = nombre
        self.__genero = genero
        self.__cantJugadores = cantJugadores
        self.__rating = rating
```

Clases como tipos

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

    def obtenerGenero(self)->str:
        """Devuelve el género del videojuego."""
        return self.__genero

    def obtenerCantJugadores(self)->int:
        """Devuelve la cantidad de jugadores registrados del videojuego."""
        return self.__cantJugadores

    def obtenerRating(self)->float:
        """Devuelve el rating del videojuego."""
        return self.__rating

    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}"
```

Clases como tipos

```
#consultas
def esPopular(self)->bool:
    """Devuelve True si el rating es mayor a 3.5 y tiene más de 10.000 jugadores."""
    return self.__rating > 3.5 and self.__cantJugadores > 10000

def mismoGenero(self, otro: 'Videojuego')->bool:
    """Devuelve True si ambos videojuegos son del mismo género."""
    return self.__genero == otro.obtenerGenero()

def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()

def masPopular(self, otro: 'Videojuego')->'Videojuego':
    """Devuelve el videojuego más popular."""
    if self.masPopularQue(otro):
        return self
    else:
        return otro
```

Clases como tipos

```
class Videojuego:
    #atributos de clase (si es que hay)

    #constructor y atributos de instancia

    #consultas

    #comandos
```

La clase Videojuego define un tipo de dato.

El **conjunto de valores** está formado por combinaciones de valores de un subconjunto {string, string, int, float}.

El **conjunto de operaciones** está formado por los servicios, en este caso el constructor y las consultas, provistos por la clase.

Clases como tipos

```
class testVideojuego:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

La clase **testVideojuego** usa a la clase **Videojuego**, declara una variable **juego1** de **tipo clase Videojuego**.

La variable **juego1** queda ligada a un objeto cuyos atributos toman valores dentro del conjunto de valores que establece el tipo clase **Videojuego**.

Clases como tipos

```
class testVideojuego:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        if juego1.esPopular():
```

El objeto ligado a la variable **juego1** recibe el mensaje **esPopular()**, que provoca la ejecución de una operación provista por el tipo.

Clases como tipos

```
class testVideojuego:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        if juego1.esPopular():
            print(f"{juego1.obtenerNombre()} es popular")
```

El objeto ligado a la variable **juego1** recibe el mensaje **obtenerNombre()**, que provoca la ejecución de una operación provista por el tipo.

Variables y tipos

El tipo de una variable puede ser:

- Elemental
- Una clase

El valor de una variable de un tipo elemental pertenece al conjunto de valores definido por el tipo.

El valor de una variable de un tipo clase en Python es una referencia a un objeto en memoria. Una referencia puede estar no definida, ser nula (‘None’), o estar ligada a un objeto existente.

Variables y objetos

La sentencia

```
juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

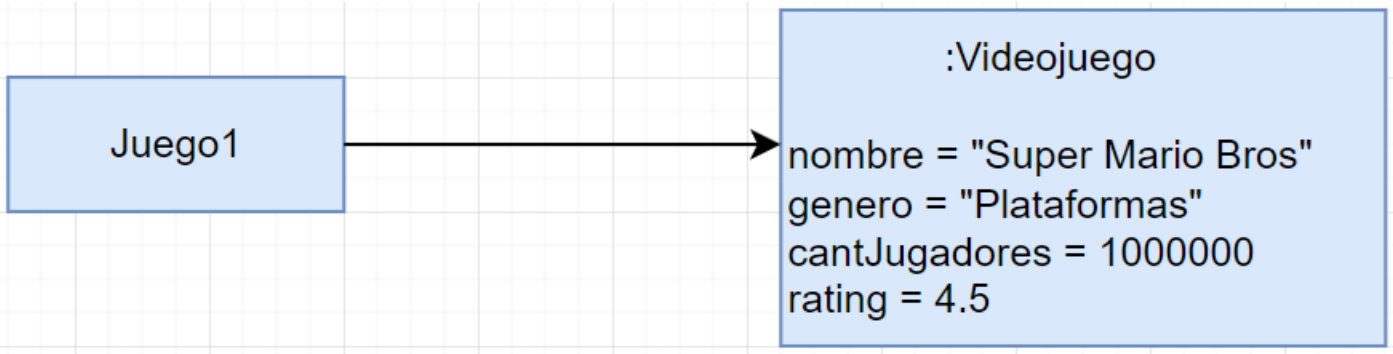
Crea una variable **juego1** de **tipo clase Videojuego**, cuyo valor es una **referencia ligada** a un objeto de clase Videojuego.

Variables y objetos

La sentencia

```
juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

Se modela dibujando la **variable**, el **objeto** y la **referencia ligada**.



El **objeto de software** de clase **`Videojuego`** representa a una entidad, esto es, un **objeto del problema**.

Variables y objetos

La sentencia

```
juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

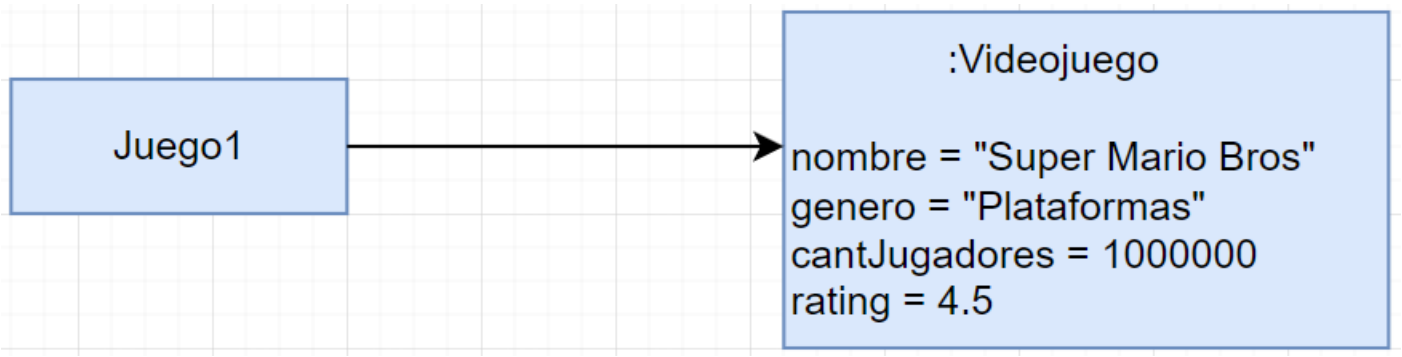
- Declara la variable Juego1 de tipo Videojuego
- Crea un objeto de clase Videojuego
- Liga el objeto a la variable

La **creación** de un objeto implica:

- Reservar espacio en memoria para mantener los valores de los atributos
- Invocar al constructor

Diagramas de objetos

Un **diagrama de objetos** es una representación gráfica que permite modelar objetos y referencias.



Cada **declaración**, **asignación** o **mensaje**, puede provocar un cambio en el diagrama de objetos.

Mensajes y métodos

Cuando un objeto recibe un **mensaje**, su clase determina el **método** que se va a ejecutar en respuesta a ese mensaje.

La clase **Videojuego** brinda la consulta **esPopular()** que retorna un valor booleano.

Cuando un objeto de clase **Videojuego** recibe el mensaje **esPopular()** se ejecuta el método provisto por esa clase y retorna un valor booleano como resultado.

Análogamente si un objeto de clase **Videojuego** recibe el mensaje **obtenerRating()** ejecuta el método provisto por su clase y retorna un valor float.

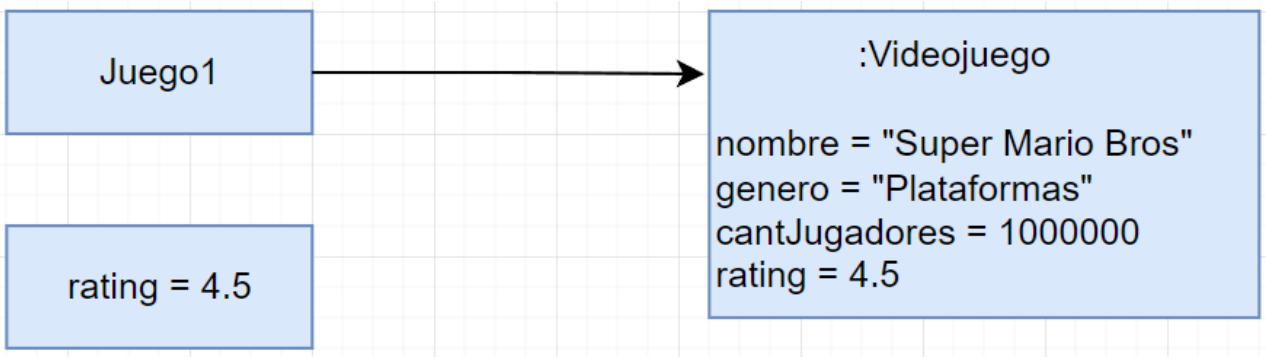
Mensajes y métodos

La instrucción

```
rating = juego1.obtenerRating()
```

Retorna el valor **4.5** y se lo asigna a la variable elemental **rating**

```
    def obtenerRating(self)->float:
        """Devuelve el rating del videojuego."""
        return self.__rating
```



Estado interno

El **estado interno** de un objeto queda determinado por los atributos de instancia de su clase.

Una variable de tipo clase está **ligada** si mantiene una **referencia** al **estado interno de un objeto**.

La programación orientada a objetos propone que una vez que se crea un objeto su estado interno solo puede ser modificado a través de los **comandos** provistos por la clase.

Estado interno - Encapsulamiento

Los atributos deben ser siempre privados (en python los declaramos con doble guión bajo al principio, aunque eso no garantiza que sean privados)

El estado interno de un objeto NO debe modificarse de forma directa desde el exterior de la clase.

Estado interno

```
juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

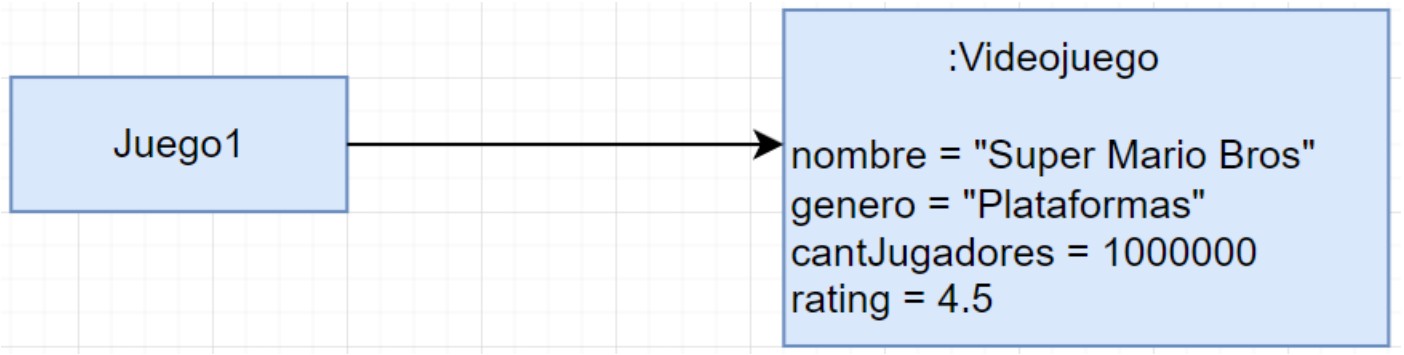
El constructor de la clase inicializa el estado interno del objeto.

```
def __init__(self, nombre:str, genero: str, cantJugadores:int, rating: float):
    """Requiere cantJugadores >= 0.
    Requiere rating >= 0 y rating <= 5."""
    self.__nombre = nombre
    self.__genero = genero
    self.__cantJugadores = cantJugadores
    self.__rating = rating
```

Estado interno

```
juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```

En el diagrama de objetos podemos modelar el estado interno del objeto y sus referencias.



Alcance de las variables

```python
class Videojuego:
    # constructor y atributos de instancia
    def __init__(self, nombre:str, genero: str,
cantJugadores:int, rating: float):
        """Requiere cantJugadores >= 0.
        Requiere rating >= 0 y rating <= 5."""
        self.__nombre = nombre
        self.__genero = genero
        self.__cantJugadores = cantJugadores
        self.__rating = rating
    def obtenerNombre(self)->str:
        """Devuelve el nombre del videojuego."""
        return self.__nombre

    def obtenerGenero(self)->str:
        """Devuelve el género del videojuego."""
        return self.__genero
```

```python
class testVideojuego:
    @staticmethod
    def test():
        juego1 = Videojuego("Mario", "Plataforma", 1000, 4.5)
        if juego1.esPopular():
            print(f"{juego1.obtenerNombre()} es popular")
        else:
            print(f"{juego1.obtenerNombre()} no es popular")
        juego2 = Videojuego("Sonic", "Aventura", 500, 4.8)
        if juego1.mismoGenero(juego2):
            print("son del mismo género")
        if juego1.masPopularQue(juego2):
            print(f"{juego1.obtenerNombre()} es más popular
que {juego2.obtenerNombre()}")
```

- test() puede acceder a las variables juego1 y juego2 de tipo clase Videojuego.
- Los servicios de la clase Videojuego pueden acceder a __nombre, __genero, __cantJugadores y __rating

Parámetros de tipo clase

```
class testVideojuego:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.8)
        if juego1.masPopularQue(juego2):
            print(f"{juego1.obtenerNombre()} es más popular que {juego2.obtenerNombre()}")
            jugadores = juego1.obtenerCantJugadores()
        else:
            print(f"{juego1.obtenerNombre()} no es más popular que {juego2.obtenerNombre()}")
            jugadores = juego2.obtenerCantJugadores()
```

- jugadores es una variable elemental
- juego1 y juego2 son variables de tipo clase, mantienen referencias
- jugadores, juego1 y juego2 son locales a test()

Parámetros de tipo clase

```
juego1.masPopularQue(juego2):
```

Envía el mensaje **masPopularQue** al objeto ligado a la variable **juego1**

El parámetro efectivo (o real) **juego2** es una variable de tipo clase, es decir, es una referencia a un objeto de clase **Videojuego**

```
def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()
```

Parámetros de tipo clase

```
juego1.masPopularQue(juego2):
```

Al comenzar la ejecución del método **juego2** se asigna al parámetro **otro.**

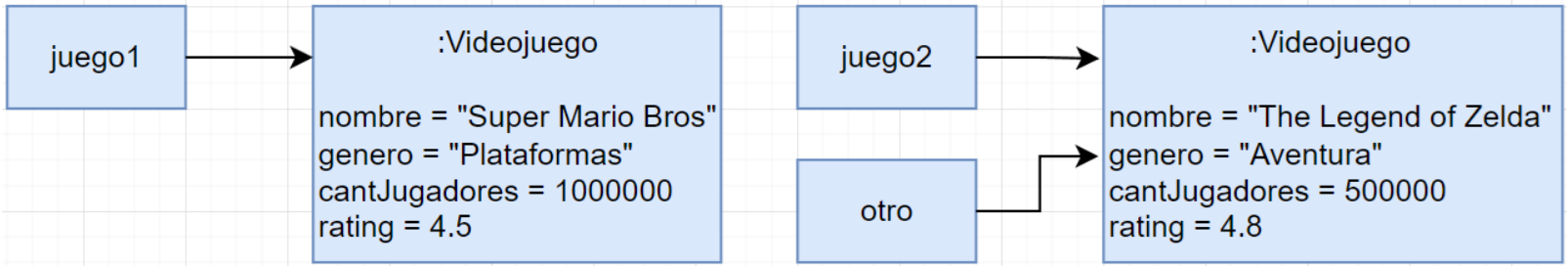
otro sólamente va a existir mientras se esté ejecutando el método **masPopularQue**

```
def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()
```

Parámetros de tipo clase

```
juego1.masPopularQue(juego2):
```

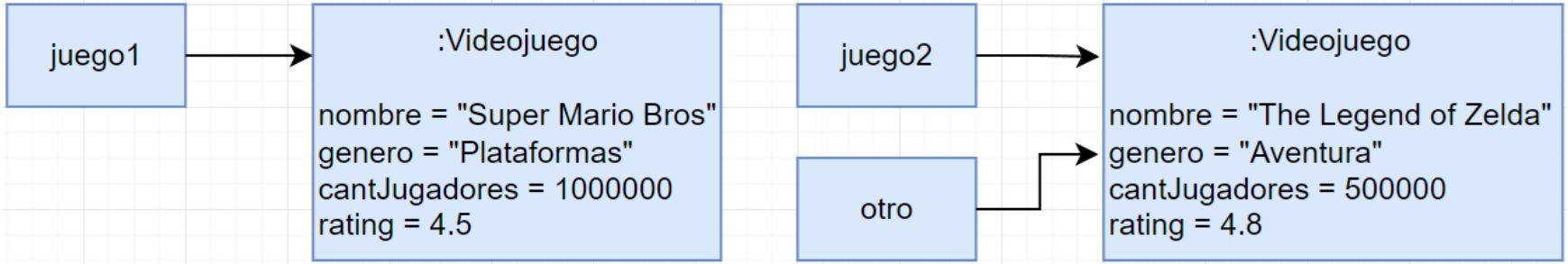
Las variables **juego2** y **otro** mantienen referencias al mismo objeto.




```
def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()
```

Parámetros de tipo clase

```
juego1.masPopularQue(juego2):
```

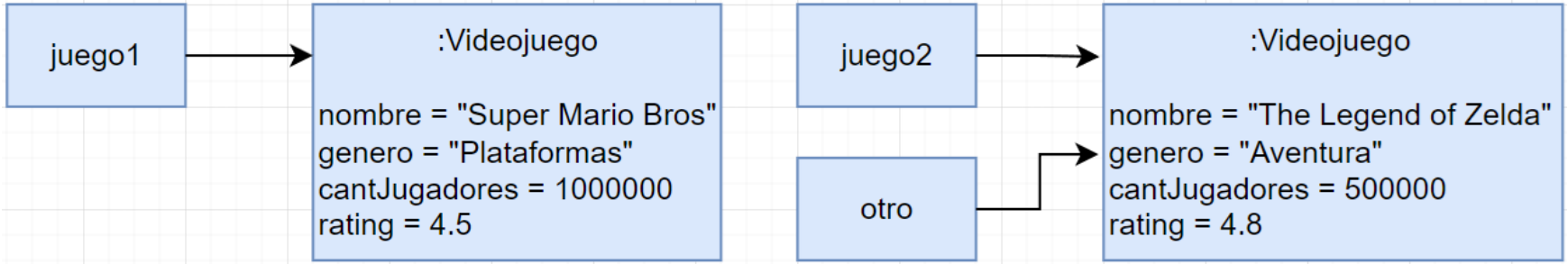


Compara la cantidad de jugadores y el rating del objeto que recibe el mensaje con la cantidad de jugadores y el rating del objeto recibido como parámetro

```
def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()
```

Parámetros de tipo clase

```
juego1.masPopularQue(juego2):
```



En nuestro ejemplo, retorna true porque **el objeto referenciado por *'juego1'*** tiene mas jugadores y mayor rating que **el objeto referenciado por *'otro'***

```
def masPopularQue(self, otro: 'Videojuego')->bool:
    """Devuelve True si el videojuego es más popular que otro."""
    return self.__cantJugadores > otro.obtenerCantJugadores() and self.__rating > otro.obtenerRating()
```

Identidad, igualdad y equivalencia (repaso)

Cada objeto de software tiene una **identidad**, una **propiedad** que lo distingue de los demás.

La **referencia** a un objeto puede ser usada como propiedad para identificarlo.

Si dos variables son iguales, mantienen una misma referencia, entonces están ligadas a un mismo objeto.

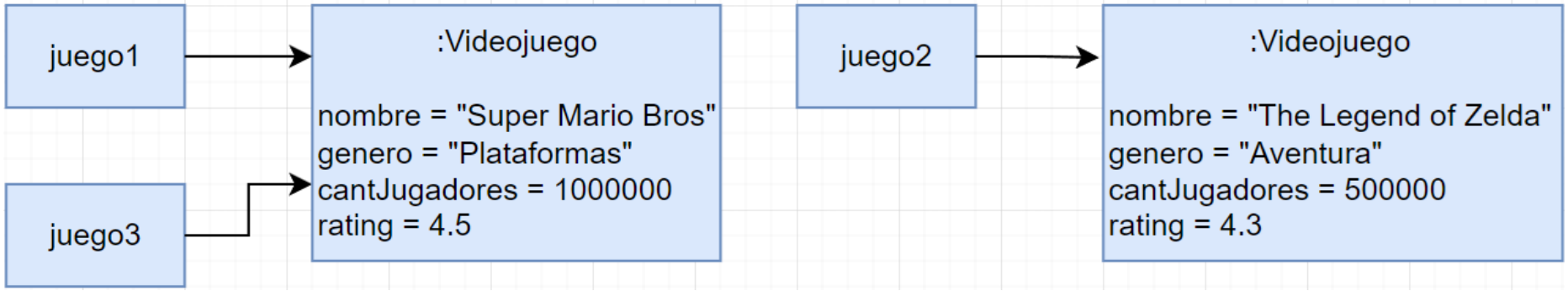
Cuando dos objetos mantienen el mismo estado interno, decimos que son **equivalentes**, aun cuando tienen diferente identidad.

Identidad, igualdad y equivalencia (repaso)

```
class TestReferencias:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.3)
        juego3 = juego1.masPopular(juego2)
        juego4 = juego1
        juego5 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        juego6 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        igualdad1 = juego1 == juego3
        igualdad2 = juego1 == juego4
        igualdad3 = juego5 == juego6
        print(f"juego1 == juego3 : {igualdad1}")
        print(f"juego1 == juego4 : {igualdad2}")
        print(f"juego5 == juego6 : {igualdad3}")
```

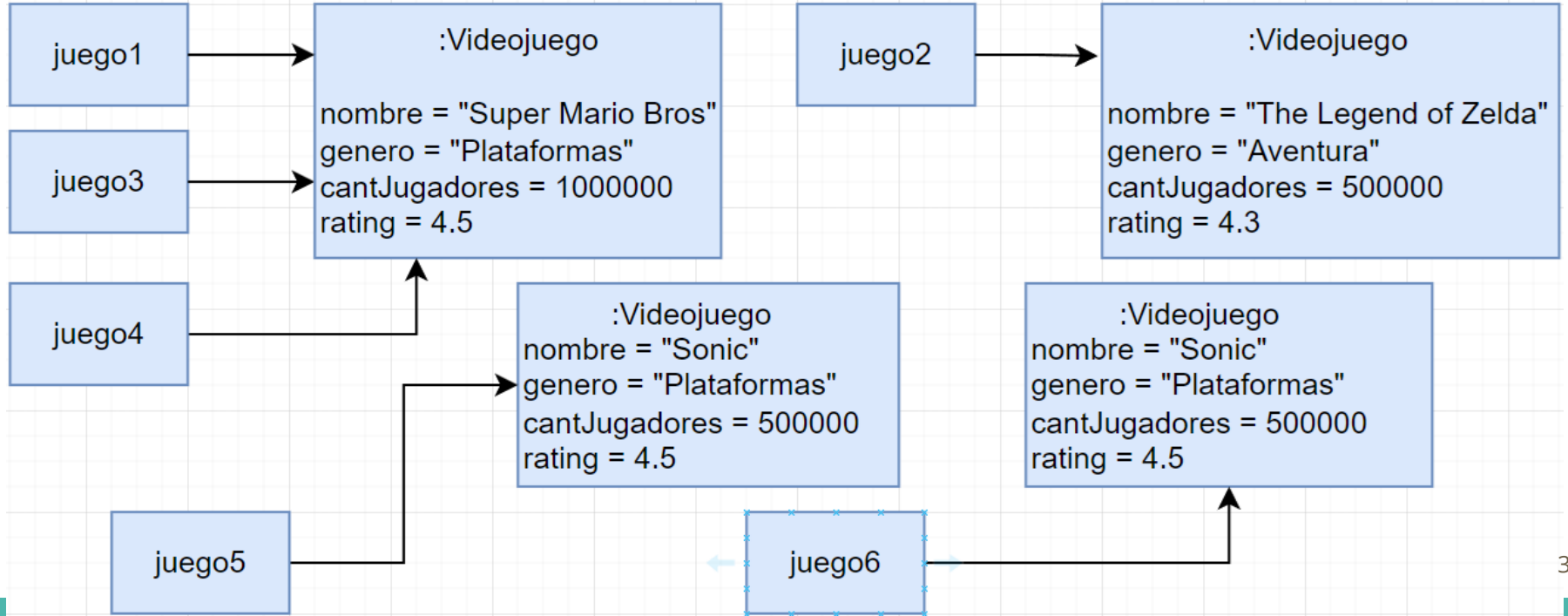
Identidad, igualdad y equivalencia (repaso)

```
class TestReferencias:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.3)
        juego3 = juego1.masPopular(juego2)
```



Identidad, igualdad y equivalencia (repaso)

```
juego4 = juego1
juego5 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
juego6 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
```




Juego 2
Juego1 Juego 4

Identidad, igualdad y equivalencia (repaso)

```
class TestReferencias:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.3)
        juego3 = juego1.masPopular(juego2)
        juego4 = juego1
        juego5 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        juego6 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        igualdad1 = juego1 == juego3
        igualdad2 = juego1 == juego4
        igualdad3 = juego5 == juego6
        print(f"juego1 == juego3 : {igualdad1}")
        print(f"juego1 == juego4 : {igualdad2}")
        print(f"juego5 == juego6 : {igualdad3}")
```

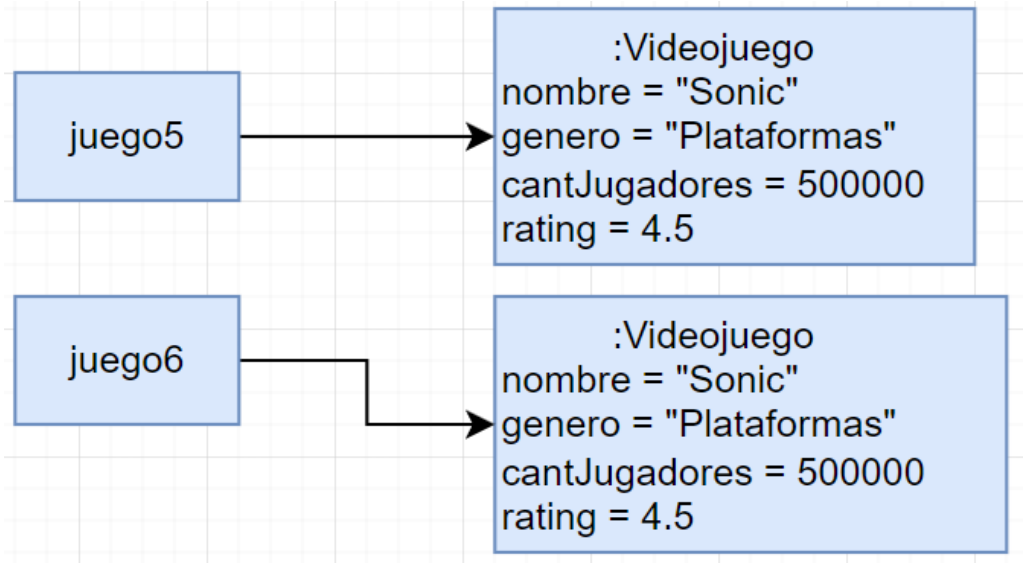
El operador relacional **==** compara variables de tipo clase, esto es **referencias**.

Identidad, igualdad y equivalencia (repaso)

```
if juego1.obtenerCantJugadores() == juego4.obtenerCantJugadores():
        print("La cantidad de jugadores de juego1 es igual a la de juego4")
if juego1.obtenerCantJugadores() == juego5.obtenerCantJugadores():
        print("La cantidad de jugadores de juego1 es igual a la de juego5")
```

El operador relacional **==** compara valores de **variables elementales**.

Identidad, igualdad y equivalencia (repaso)



Dos objetos que tienen el mismo estado interno son **equivalentes**.

Clases como tipos

Videojuego

<<atributos de instancia>>
- nombre : String
- genero : String
- jugadores : entero
- rating : float

<<constructor>>
+ Videojuego (nombre: string, genero: string, cantJugadores: entero, rating: float)

<<consultas>>
+ obtenerNombre(): String
+ obtenerGenero(): String
+ obtenerJugadores(): entero
+ obtenerRating(): flotante
+ esPopular(): boolean
+ mismoGenero(videojuego: Videojuego): boolean
+ masPopularQue(videojuego: Videojuego): boolean
+ toString(): String
+ masPopular(videojuego: Videojuego): Videojuego
+ esIgualQue(videojuego: Videojuego): boolean

Requiere 0 <= rating <= 5

Requiere cantJugadores > 0

esPopular() devuelve True si rating >= 3.5 y jugadores > 10000

mismoGenero retorna True si el objeto que recibe el mensaje tiene el mismo género que el videojuego recibido en el parámetro

masPopularQue retorna True si el objeto que recibe el mensaje tiene mas jugadores y más rating que el objeto recibido por parámetro

masPopular retorna el objeto más popular

esIgualQue retorna True si el objeto que recibe el mensaje tiene el mismo estado interno que el objeto recibido por parámetro

Identidad, igualdad y equivalencia

```
def esIgualQue(self, otro: 'Videojuego')->bool:
        """Devuelve True si ambos videojuegos son iguales (mismo estado interno)."""
        mismoNombre = self.__nombre == otro.obtenerNombre()
        mismoGenero = self.__genero == otro.obtenerGenero()
        mismaCantJugadores = self.__cantJugadores == otro.obtenerCantJugadores()
        mismoRating = self.__rating == otro.obtenerRating()
        return mismoNombre and mismoGenero and mismaCantJugadores and mismoRating
```

El método **esIgualQue()** recibe un objeto de tipo clase "**Videojuego**", compara el **estado interno de la instancia** que recibió el mensaje con el **estado interno del objeto recibido por parámetro**. Devuelve **True** cuando ambos estados internos son iguales, **False** en caso contrario.
Accede al estado interno del objeto recibido por parámetro a través de los *servicios ofrecidos por la clase*.

Clases como tipos

Videojuego

<<atributos de instancia>>
- nombre : String
- genero : String
- jugadores : entero
- rating : float

<<constructor>>
+ Videojuego (nombre: string, genero: string, cantJugadores: entero, rating: float)

<<consultas>>

. . .

. . .

. . .

+ clonar(): Videojuego

<<comandos>>

. . .

+ copiarValores(videojuego: Videojuego)

Requiere 0 <= rating <= 5

Requiere cantJugadores > 0

clonar retorna un nuevo Videojuego con el mismo estado interno de la instancia que recibe el mensaje

copiarValores actualiza el estado interno del objeto con los valores del estado interno del objeto Videojuego que recibe por parámetro

Copia de valores

```
def copiarValores(self, otro:"Videojuego")->None:
    """Copia los valores de 'otro' en self."""
    self.__nombre = otro.obtenerNombre()
    self.__genero = otro.obtenerGenero()
    self.__cantJugadores = otro.obtenerCantJugadores()
    self.__rating = otro.obtenerRating()
```

copiarValores actualiza el estado interno del objeto con los valores del estado interno del objeto Videojuego que recibe por parámetro (accedidos a través de los *métodos ofrecidos por la clase*).

Copia de valores

Al comenzar la ejecución del comando **copiarValores** se reserva una locación de memoria para el parámetro formal **‘otro’** y se le asigna el valor del parámetro real, **una referencia**.

El método requiere que **‘otro’** sea una **referencia ligada** a un objeto de clase **Videojuego**. Por lo tanto, si el valor de **‘otro’** es distinto a lo requerido, la ejecución termina anormalmente.

Cuando termina la ejecución del comando **copiarValores** la locación de memoria que almacena el valor de **‘otro’** se libera.

El videojuego que recibió el mensaje y el que pasó como parámetro **son equivalentes**, mantienen los mismos valores en sus atributos de instancia.

Clonar objeto

```
def clonar(self)->'Videojuego':
    """Devuelve un clon del videojuego."""
    return Videojuego(self.__nombre, self.__genero, self.__cantJugadores, self.__rating)
```

clonar() *retorna un nuevo videojuego con el mismo estado interno que la instancia que recibió el mensaje*.

La consulta **clonar** crea un nuevo videojuego y retorna la referencia.

Los parámetros reales del constructor son los atributos de instancia del videojuego que recibe el mensaje, de modo que los atributos de instancia del nuevo videojuego se inicializan con estos valores.

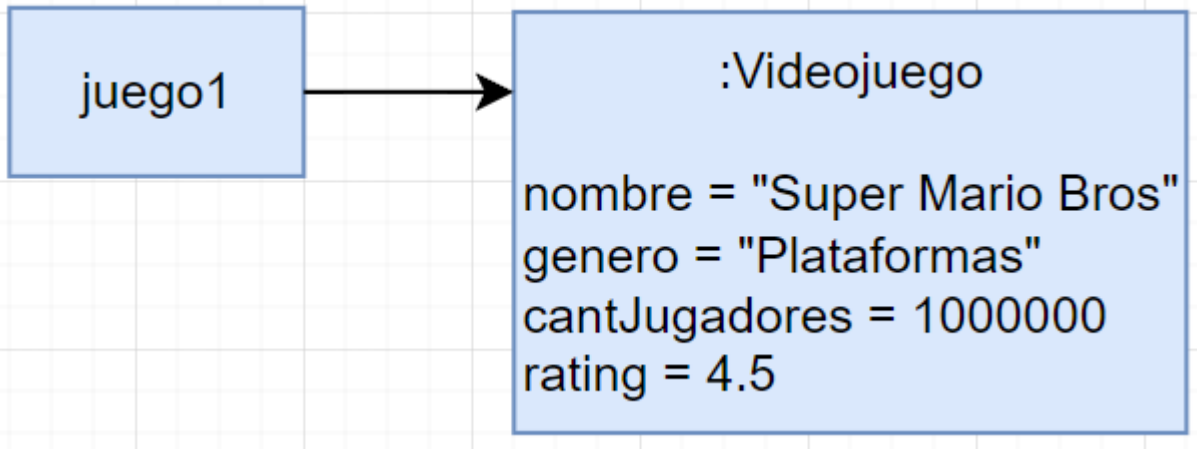
Entonces, el videojuego que recibe el mensaje y el que retorna como resultado **son equivalentes**, mantienen los mismos valores en sus atributos de instancia.

Identidad, igualdad y equivalencia

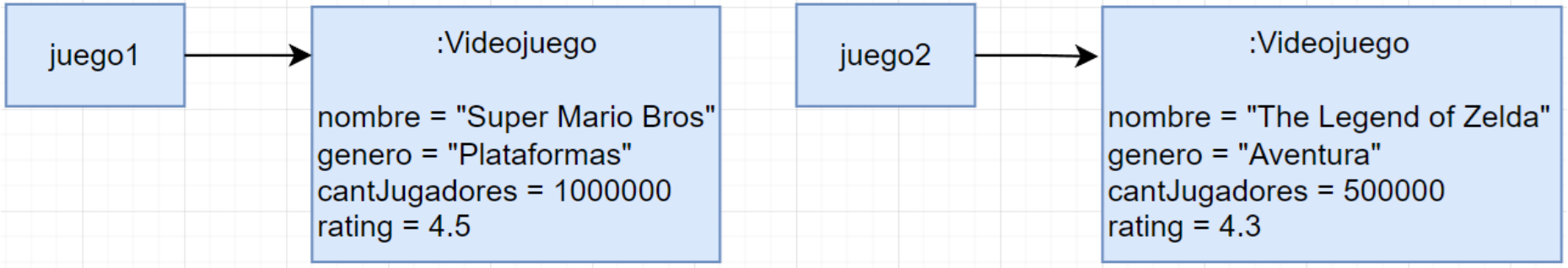
Para terminar de comprender estos conceptos, grafiquemos la ejecución del siguiente código

```
class TestReferencias:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
        juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.8)
        juego3 = juego1.masPopular(juego2)
        juego4 = juego1
        juego5 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        juego6 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
        juego6.copiarValores(juego1)
        juego7 = juego1.clonar()
```

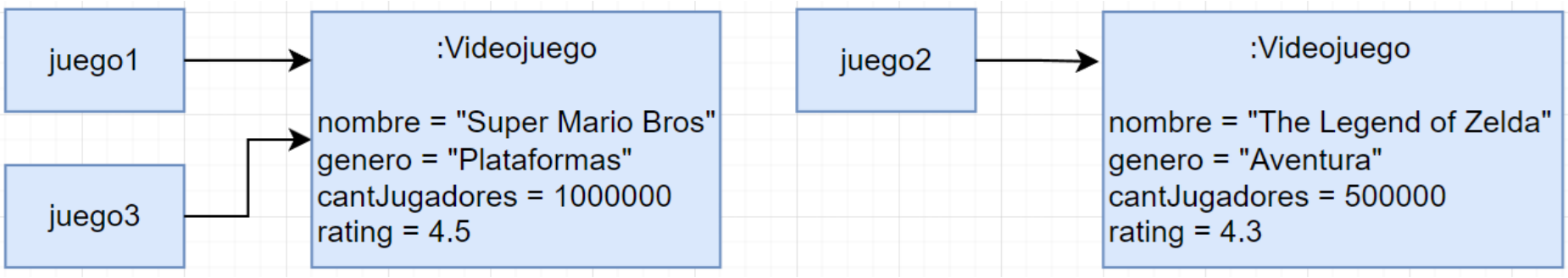
```python
class TestReferencias:
    @staticmethod
    def test():
        juego1 = Videojuego("Super Mario Bros", "Plataformas", 1000000, 4.5)
```



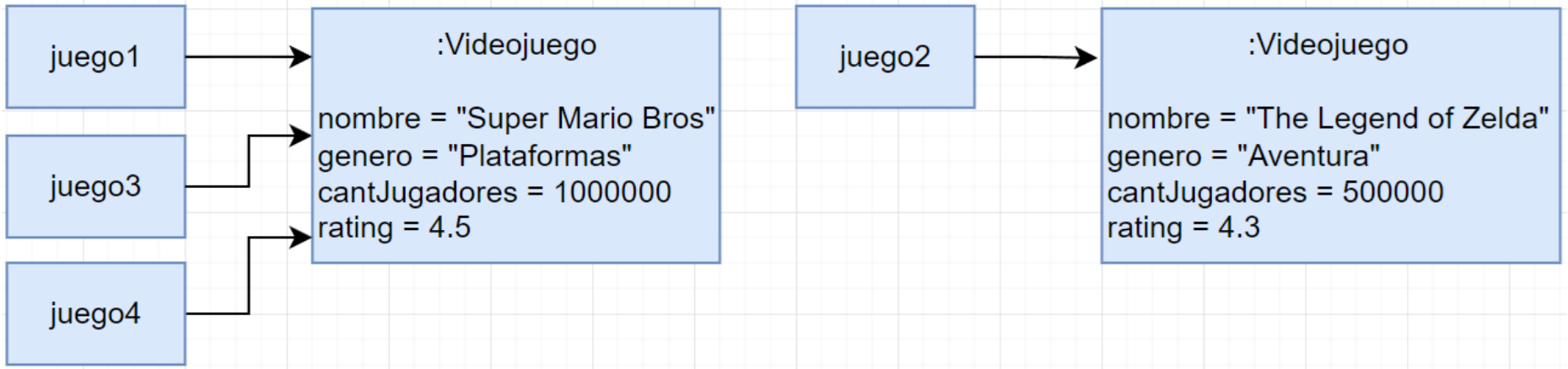
```
juego2 = Videojuego("The Legend of Zelda", "Aventura", 500000, 4.8)
```



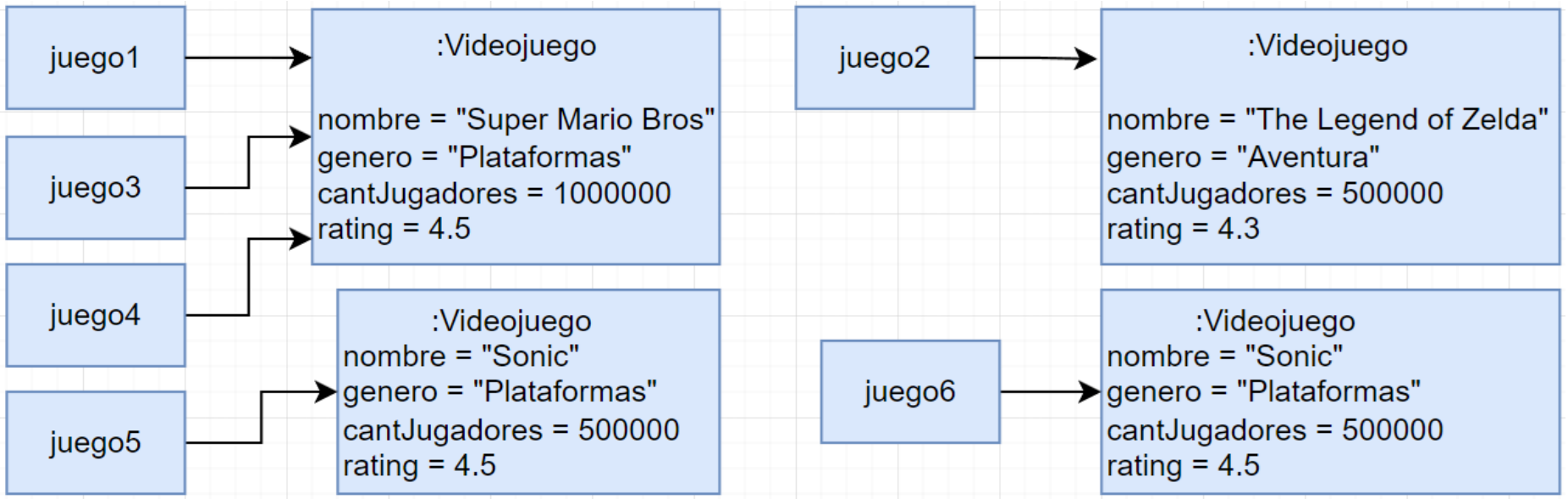
```
juego3 = juego1.masPopular(juego2)
```



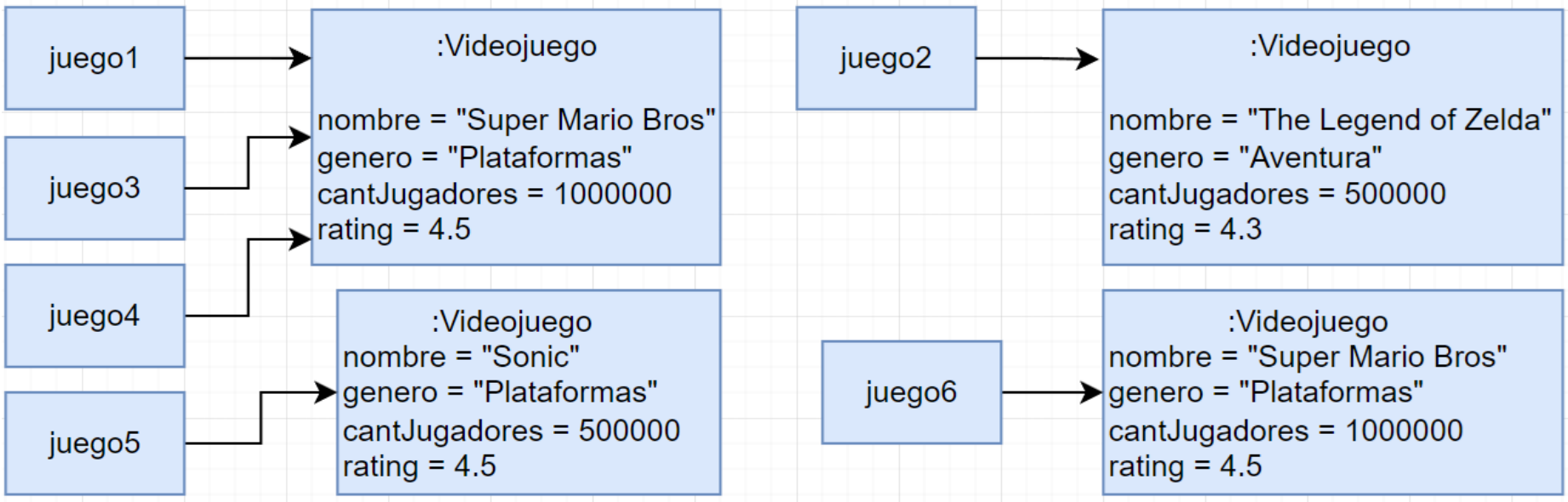
```
juego4 = juego1
```



```
juego5 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
juego6 = Videojuego("Sonic", "Plataformas", 500000, 4.5)
```



```
juego6.copiarValores(juego1)
```



```
juego7 = juego1.clonar()
```