Contenido

Enunciado Versión 1

Simulación de un Videoclub, el cual vamos a realizar mediante un desarrollo incremental y siguiendo la práctica de programación en parejas (*pair programming*).

Cada clase debe ir en un archivo php separado. Para facilitar su implementación, se muestra la estructura UML del modelo y un fragmento de código para probar las clases.

Creamos el Soporte

Crea una clase para almacenar soportes (Soporte.php). Esta clase será la clase padre de los diferentes soportes con los que trabaje nuestro videoclub (cintas de vídeo, videojuegos, etc...):

- Crea el constructor que inicialice sus propiedades. Fíjate que la clase no tiene métodos *setters*.
- Definir una constante mediante un propiedad privada y estática denominada IVA con un valor del 21%.
- Crear un archivo (inicio.php) para usar las clases y copia el siguiente fragmento:

```php
<?php
include "Soporte.php";

$soporte1 = new Soporte("Tenet", 22, 3);
echo "<strong>" . $soporte1->titulo . "</strong>";
echo "<br>Precio: " . $soporte1->getPrecio() . " euros";
echo "<br>Precio IVA incluido: " . $soporte1->getPrecioConIVA() . " euros";
$soporte1->muestraResumen();
```

Salida:

Tenet
Precio: 3 euros
Precio IVA incluido: 3.48 euros
Tenet
3 € (IVA no incluido)

Creamos Cinta de Video

Crea la clase CintaVideo la cual hereda de Soporte. Añade el atributo duración y sobreescribe tanto el constructor como el método muestraResumen (desde CintaVideo deberás llamar al método muestraResumen del padre).

Añade a inicio.php el código para probar la clase:

```php
<?php
include "CintaVideo.php";

$miCinta = new CintaVideo("Los cazafantasmas", 23, 3.5, 107);
echo "<strong>" . $miCinta->titulo . "</strong>";
echo "<br>Precio: " . $miCinta->getPrecio() . " euros";
echo "<br>Precio IVA incluido: " . $miCinta->getPrecioConIva() . " euros";
$miCinta->muestraResumen();
```

Salida:

Los cazafantasmas
Precio: 3.5 euros
Precio IVA incluido: 4.06 euros
Película en VHS:
Los cazafantasmas
3.5 € (IVA no incluido)
Duración: 107 minutos

Añadimos Dvd

Crea la clase Dvd la cual hereda de Soporte. Añade los atributos idiomas y formatoPantalla. A continuación, sobrescribe tanto el constructor como el método muestraResumen.

Añade a inicio.php el código para probar la clase:

```php
<?php
include "Dvd.php";

$miDvd = new Dvd("Origen", 24, 15, "es,en,fr", "16:9");
echo "<strong>" . $miDvd->titulo . "</strong>";
echo "<br>Precio: " . $miDvd->getPrecio() . " euros";
echo "<br>Precio IVA incluido: " . $miDvd->getPrecioConIva() . " euros";
$miDvd->muestraResumen();
```

Origen
Precio: 15 euros
Precio IVA incluido: 17.4 euros
Película en DVD:
Origen
15 € (IVA no incluido)
Idiomas:es,en,fr
Formato Pantalla:16:9

Añadimos Juego

Crea la clase Juego la cual hereda de Soporte. Añade los atributos consola, minNumJugadores y maxNumJugadores. A continuación, añade el método muestraJugadoresPosibles, el cual debe mostrar *Para un jugador*, *Para X jugadores* o *De X a Y jugadores* dependiendo de los valores de las atributos creados. Finalmente, sobreescribe tanto el contructor como el método muestraResumen.

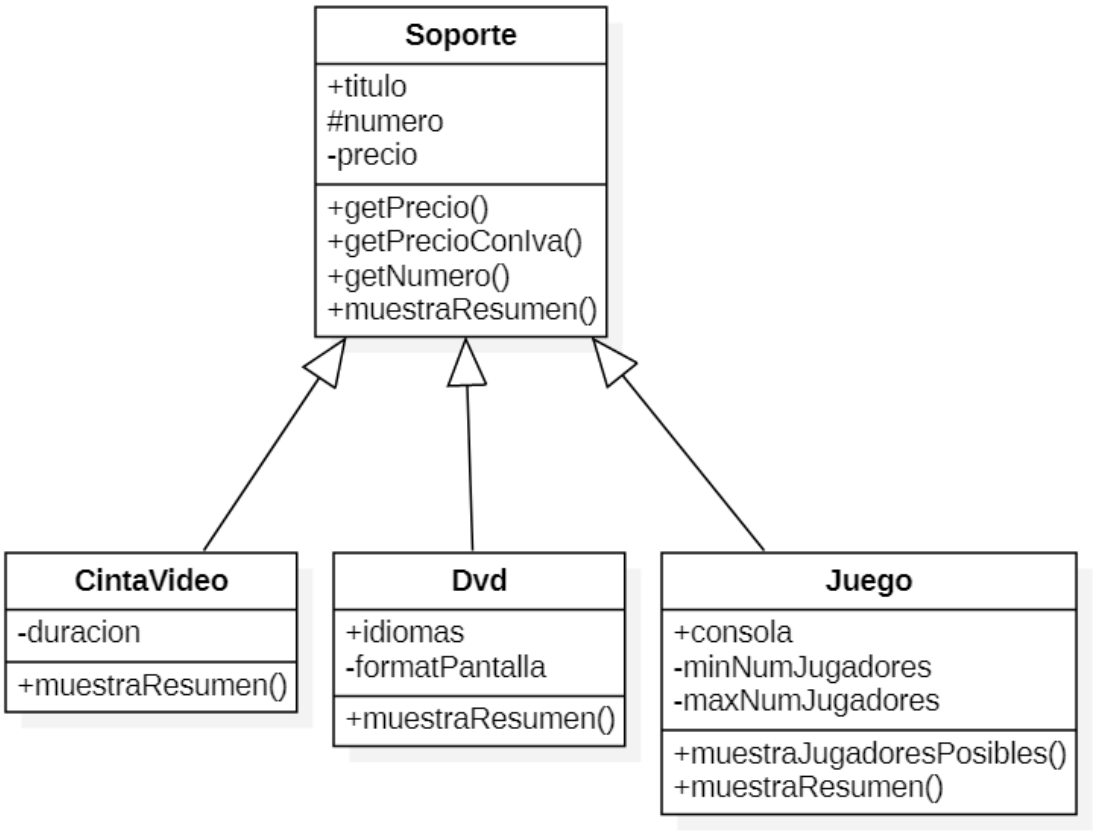
Añade a inicio.php el código para probar la clase:

```php
<?php
include "Juego.php";

$miJuego = new Juego("The Last of Us Part II", 26, 49.99, "PS4", 1, 1);
echo "<strong>" . $miJuego->titulo . "</strong>";
echo "<br>Precio: " . $miJuego->getPrecio() . " euros";
echo "<br>Precio IVA incluido: " . $miJuego->getPrecioConIva() . " euros";
$miJuego->muestraResumen();
```

The Last of Us Part II
Precio: 49.99 euros
Precio IVA incluido: 57.9884 euros
Juego para: PS4
The Last of Us Part II
49.99 € (IVA no incluido)
Para un jugador

Llegados a este punto, nuestro modelo es similar al siguiente diagrama (Modelo inicial de Videoclub)



Añadimos Cliente

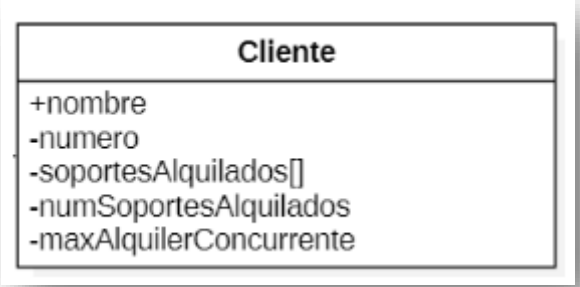
Crear la clase Cliente. El constructor recibirá el nombre, número y maxAlquilerConcurrente, este último pudiendo ser opcional y tomando como valor por defecto 3. Tras ello, añade getter/setter únicamente a número, y un getter a numSoportesAlquilados (este campo va a almacenar un contador del total de alquileres que ha realizado). El array de soportes alquilados contendrá clases que hereden de Soporte. Finalmente, añade el método muestraResumen que muestre el nombre y la cantidad de alquileres (tamaño del array soportesAlquilados).

Dentro de Cliente, añade las siguiente operaciones:

- **tieneAlquilado**(Soporte $s): bool → Recorre el array de soportes y comprueba si está el soporte.
- **alquilar**(Soporte $s): bool -→ Debe comprobar si el soporte está alquilado y si no ha superado el cupo de alquileres. Al alquilar, incrementará el numSoportesAlquilados y almacenará el soporte en el array. Para cada caso debe mostrar un mensaje informando de lo ocurrido.

Seguimos con Cliente para añadir las operaciones:

- **devolver**(int $numSoporte): bool → Debe comprobar que el soporte estaba alquilado y actualizar la cantidad de soportes alquilados. Para cada caso debe mostrar un mensaje informando de lo ocurrido
- **listarAlquileres**(): void → Informa de cuantos alquileres tiene el cliente y los muestra.



Inicio 2

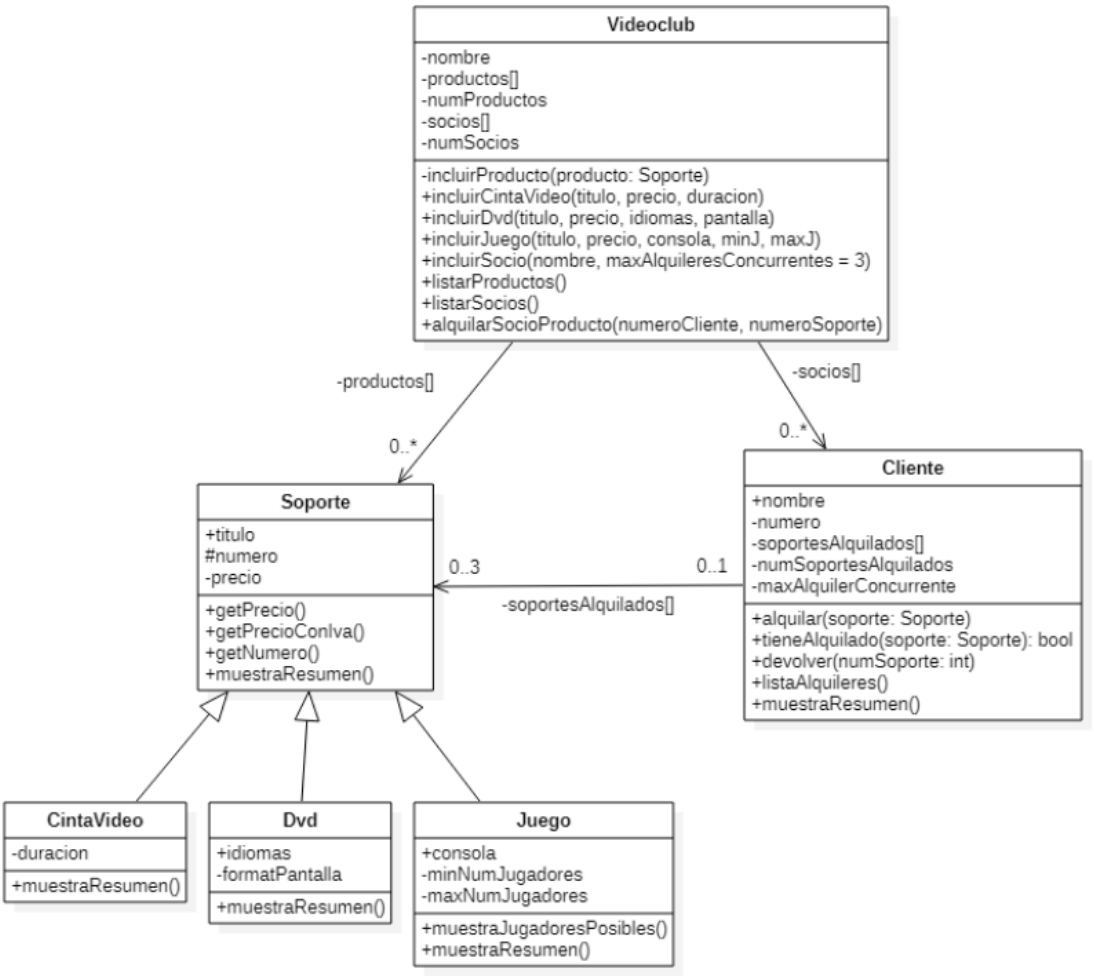
Carga el archivo inicio2.php para probar la clase.

Relación Clientes y Soportes

Vamos a relacionar los clientes y los soportes mediante la clase Videoclub. Crea la clase que se representa en este gráfico, teniendo en cuenta que:

- ***productos*** es un array de *Soporte*
- ***socios*** es una array de *Cliente*
- Los métodos públicos de incluir algún soporte crearán la clase y llamarán al método privado de *incluirProducto*, el cual es el encargado de introducirlo dentro del array.

El modelo completo quedará de la siguiente manera:



Modelo completo de Videoclub

Inicio 3

Carga el archivo inicio3.php para probar la clase.

Fin Versión 1 de VideoClub

- Transforma Soporte a una **clase abstracta** y comprueba que todo sigue funcionando. ¿Qué conseguimos al hacerla abstracta?

Al declarar Soporte como una clase abstracta, estamos estableciendo que Soporte es un concepto, una plantilla, y no un objeto que pueda existir por sí mismo. No tiene sentido tener un "soporte" genérico en el videoclub; lo que tenemos son tipos específicos de soportes: CintaVideo, Dvd o Juego.

1. **Coherencia del modelo**: Forzamos a que solo se puedan crear objetos que tienen un sentido completo en nuestro dominio (DVDs, Juegos, etc.), evitando objetos genéricos e incompletos.
2. **Seguridad en el diseño**: Evitamos que otros programadores (o nosotros mismos en el futuro) creen instancias de Soporte por error.
3. **Define un contrato base**: Una clase abstracta garantiza que todas las clases que hereden de ella compartirán una estructura y comportamiento comunes (propiedades y métodos definidos en Soporte).

El código de las clases CintaVideo, Dvd, Juego, Cliente y Videoclub seguirá funcionando perfectamente, ya que solo trabajan con las clases hijas (concretas).

- Crea un interfaz **Resumible**, de manera que las clases que lo implementen deben ofrecer el método muestraResumen(). Modifica la clase **Soporte** y haz que implemente el interfaz. ¿Hace falta que también lo implementen los hijos?

Una interfaz es un contrato que define qué métodos debe implementar una clase, pero sin decir cómo debe hacerlo. Es una forma de garantizar que diferentes clases, aunque no estén relacionadas por herencia, puedan realizar una misma acción.

Cuando una clase padre (Soporte) implementa una interfaz, todas sus clases hijas (CintaVideo, Dvd, Juego) heredan automáticamente esa implementación y también se consideran del tipo de la interfaz.

- Soporte ya cumple el contrato de Resumible porque tiene un método muestraResumen().
- Las clases hijas heredan este cumplimiento. Aunque ellas sobreescriben el método muestraResumen() para añadir su propia lógica, ya están cumpliendo el contrato que su padre firmó.

Hacer esto nos permitiría, por ejemplo, tener un array de objetos Resumible que podrían incluir Soportes, Clientes (si también implementara la interfaz), etc., y podríamos recorrerlo llamando a muestraResumen() en cada uno, sin importar de qué clase son.

Sugerencia:

Antes de nada, crea un repositorio privado en GitHub y sube el proyecto actual de *Videoclub*. Una vez creado, invita a tu compañero al repositorio como colaborador.

- Inicializa en local tu repostorio de git, mediante git init
- Añade y sube los cambios a tu repositorio, mediante git add . y luego git commit -m 'Inicializando proyecto'.
- Conecta tu repositorio con GitHub y sube los cambios (mira la instrucciones de GitHub: comandos git remote y git push).
- Tu compañero deberá descargar el proyecto con sus credenciales.