ALQUILER PELÍCULAS

En este ejercicio vamos a modelar el sistema de visión y pago de películas y series de una empresa de streaming de vídeo. Vamos a tener una empresa que será la clase PrimeVideo que tendrá un catálogo de películas y un conjunto de suscriptores. Tendremos que añadir películas, series y clientes a esa plataforma y al final calcular las ganancias de la empresa.

MULTIMEDIA

A continuación, se indican las clases que tenemos que crear:

Genero: enum {Thriller, Accion, Aventuras, Romántica, Terror, Infantil, Scifi, Drama, Comedia, Oriental}. En mayúsculas.

Cliente: dni, nombre, email, precioMensual. Constructores (defecto y con todos los parámetros menos precioMensual), getters, setters, toString(), equals por dni.
Método abstracto boolean esPro();
Método abstracto double getPrecioMensual();
Sólo habrá Clientes de los dos tipos siguientes, Prime y PrimePro.

ClientePrime: el precio mensual es de 3€ (static final). Se llama al constructor padre pasándole el precioMensual con el valor del atributo estático.
esPro() devuelve false
getPrecioMensual() devuelve 3
Paga el precio mensual más el precio de las películas y series que no son “plus”.

ClientePrimePro: el precio mensual es de 5€. Se llama al constructor padre pasándole el precioMensual de 5.
esPro() devuelve true
getPrecioMensual() devuelve 5
No paga por las pelis y series catalogadas “plus”.

Multimedia: código (long autoincremental) y título, plus (boolean), precio (se aplica si no es cliente plus) y Genero. Constructores (por defecto, paremetrizado solo con titulo, precio y esPlus), getters, setters, equals por código, toString.
Para hacer el autoincremento del código necesitas un atributo estático llamado autoincremento inicializado a uno, y cada vez que se crea un objeto se le suma uno a ese autoincremento estático y luego se le asigna a ‘código’.

Película: hereda de multimedia. int duración (en minutos). Constructor añadiéndole al del padre la duración, getters, setters, toString().

PrimeVideo: ArrayList<Multimedia> catalogo, ArrayList<Cliente> suscriptores, double ganancias.
Constructor sin parámetros que inicializa los ArrayList y las ganancias a cero.
Método addSuscriptor(Cliente): añade un suscriptor, si no estaba ya
Método addMultimedia(Multimedia): añade una película al catálogo, si no estaba ya
Método ver(Multimedia m, Cliente c): si el cliente no es pro se añade a ganancias el precio del Multimedia.
Método double getGanancias(): devuelve las ganancias. A las ganancias generadas cuando los clientes ven multimedia hay que sumarle las ganancias de los todos los suscriptores que haya (multiplicado por 12 meses).

En una clase Test calcula las ganancias anuales de la empresa si tiene 200 pelis (30 son plus a 10€ cada una) y 1.000.000 de clientes (25% son pro). Simula que cada cliente vea las 40 primeras películas y dime las ganancias.

SERIES

Ahora vamos a ampliar el ejercicio añadiendo un nuevo tipo de Multimedia, las Series. Estarán formadas por Temporadas y cada Temporada por Episodios. Aquí la descripción de estas clases:

Serie: hereda de Multimedia.
ArrayList<Temporada> temporadas.
Constructor llamando al del padre e iniciando el array de temporadas, getters, setters, toString(), getNumeroTemporadas(), addTemporada(Temporada temporada), delTemporada(Temporada temporada) .
El método addTemporada() cuando se añade una Temporada a una serie, se le pone a la Temporada la Serie que la está añadiendo como this, así:

```
temporada.setSerie(this);
this.temporadas.add(temporada);
```

Temporada: int numero, ArrayList<Episodio> episodios, Serie serie.
Constructores (por defecto y solo con numero, inicializar el ArrayList), getters, setters, getNumeroEpisodios(), toString() (número, título de la serie, episodios), addEpisodio(Episodio episodio), delEpisodio(Episodio episodio).

El método addEpisodio(Episodio episodio) primero asigna el objeto this a la temporada del episodio y luego añade ese episodio a la temporada, de esta manera:

```
episodio.setTemporada(this);
this.episodios.add(episodio);
```

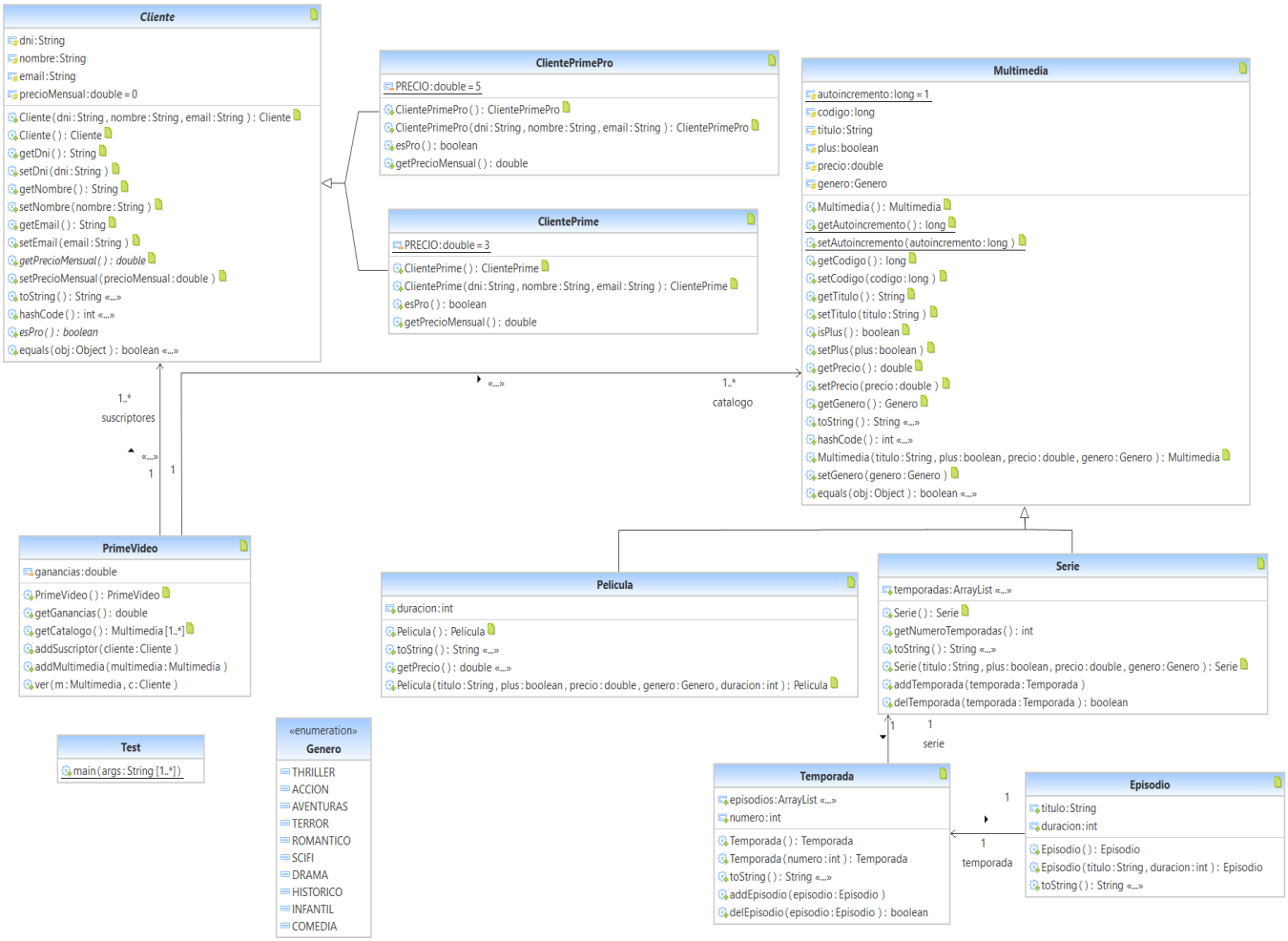
Episodio: String titulo, String duración (en minutos), Temporada temporada. Constructor por defecto, parametrizado con titulo y duración. Getters y setters, toString().

Añade 10 series al catálogo. Simula que cada cliente vea 5 series, y calcula las ganancias.

Ejemplo de cómo añadir crear una serie de 10 temporadas y 10 episodios por temporada:

```
Serie serie = new Serie("Breaking Bad", true, 10, Genero.THRILLER);
for(int i=0; i<10; i++) {
        Temporada t = new Temporada(i);
        for(int j=0; j<10; j++) {
                t.addEpisodio(new Episodio("Episodio "+j, 50));
        }
        serie.addTemporada(t);
}
```

Genera el **Javadoc** de todas las clases creadas con los comentarios pertinentes en los métodos y propiedades.


Cliente
dni:String
nombre:String
email:String
precioMensual:double = 0
Cliente(dni:String, nombre:String, email:String):Cliente
Cliente():Cliente
getDni():String
setDni(dni:String)
getNombre():String
setNombre(nombre:String)
getEmail():String
setEmail(email:String)
getPrecioMensual():double
setPrecioMensual(precioMensual:double)
toString():String «...»
hashCode():int «...»
esPro():boolean
equals(obj:Object):boolean «...»
ClientePrimePro
PRECIO:double = 5
ClientePrimePro():ClientePrimePro
ClientePrimePro(dni:String, nombre:String, email:String):ClientePrimePro
esPro():boolean
getPrecioMensual():double
ClientePrime
PRECIO:double = 3
ClientePrime():ClientePrime
ClientePrime(dni:String, nombre:String, email:String):ClientePrime
esPro():boolean
getPrecioMensual():double
Multimedia
autoincremento:long = 1
codigo:long
titulo:String
plus:boolean
precio:double
genero:Genero
Multimedia():Multimedia
getAutoincremento():long
setAutoincremento(autoincremento:long)
getCodigo():long
setCodigo(codigo:long)
getTitulo():String
setTitulo(titulo:String)
isPlus():boolean
setPlus(plus:boolean)
getPrecio():double
setPrecio(precio:double)
getGenero():Genero
toString():String «...»
hashCode():int «...»
Multimedia(titulo:String, plus:boolean, precio:double, genero:Genero):Multimedia
setGenero(genero:Genero)
equals(obj:Object):boolean «...»
1..*
suscriptores
«...»
1
1
«...»
1..*
catalogo
PrimeVideo
ganancias:double
PrimeVideo():PrimeVideo
getGanancias():double
getCatalogo():Multimedia[1..*]
addSuscriptor(cliente:Cliente)
addMultimedia(multimedia:Multimedia)
ver(m:Multimedia, c:Cliente)
Pelicula
duracion:int
Pelicula():Pelicula
toString():String «...»
getPrecio():double «...»
Pelicula(titulo:String, plus:boolean, precio:double, genero:Genero, duracion:int):Pelicula
Serie
temporadas:ArrayList «...»
Serie():Serie
getNumeroTemporadas():int
toString():String «...»
Serie(titulo:String, plus:boolean, precio:double, genero:Genero):Serie
addTemporada(temporada:Temporada)
delTemporada(temporada:Temporada):boolean
1
1
serie
Test
main(args:String[1..*])
«enumeration»
Genero
THRILLER
ACCION
AVENTURAS
TERROR
ROMANTICO
SCIFI
DRAMA
HISTORICO
INFANTIL
COMEDIA
Temporada
episodios:ArrayList «...»
numero:int
Temporada():Temporada
Temporada(numero:int):Temporada
toString():String «...»
addEpisodio(episodio:Episodio)
delEpisodio(episodio:Episodio):boolean
1
1
temporada
Episodio
titulo:String
duracion:int
Episodio():Episodio
Episodio(titulo:String, duracion:int):Episodio
toString():String «...»

BLACKJACK

En este ejercicio vamos a implementar el juego del **Blackjack**. Para ello primero nos crearemos varias clases que indicamos a continuación.

1. Clase **`Carta`**.

   - Atributo para el palo concreto de una carta (int).
   - Atributo para la figura concreta de una carta (String).
   - Constructor parametrizado Carta(palo, figura).
   - Constructor copia.
   - Métodos getters y setters.
   - Método getNombreCortoCarta() que devuelva un nombre corto de la carta tipo: 2C, 13D, …
   - Método toString() que devuelva en formato visual o en modo texto la carta: 2♥, A♦, …, 4Tréboles, etc.

2. Clase *abstracta* **`Baraja`**.

   Una baraja de cartas es una colección especifica de cartas.

   Sus propiedades son los siguientes:

   - ArrayList<Carta> baraja; // Formada por 13 * 4 cartas

   Crea los siguientes métodos:

   - Constructor por defecto. Crea un ArrayList vacío para el mazo de cartas.
   - public *abstract* Carta **repartirCarta**(). Devuelve la primera carta de la baraja y se elimina de la baraja.
   - public *abstract* Carta **azar**(). Devuelve una carta de la baraja al azar y la elimina
   - public void **barajar**(). Desordena toda la baraja. *Collections.shuffle(baraja)*
   - Sobrescribe el metodo **toString**(), para que se muestre toda la baraja.

3. Clase **BarajaInglesa**

   En los juegos de naipes inglés y americano las cartas tienes dos atributos:

   - Valor: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A
   - Palo: (♥, ♦, ♣, ♠)

   Para **implementar** esto podéis utilizar los valores como números del 1 al 13 y los palos como Strings “Corazones”, “Diamantes”, “Tréboles”, “Picas”. O también podéis usar representaciones con tipos enumerados (enum).

   Atributos:
   - Atributo estático con todos los palos.
   - Atributo estático con todas las figuras.
   - El resto heredado

   Métodos:
   - Heredados

- Constructor: deberá primero llamar al constructor padre y luego generar el mazo con los paros y figuras estáticos. Hay que generar las 52 cartas y meterlas al mazo, recorriendo los palos y figuras para conseguir las 52 cartas diferentes.
- **repartirCarta**(): hay que implementarlo.
- **azar**(): hay que implementarlo.

4. Clase **Jugador**

- Atributo “mano”, la mano del jugador, las cartas que haya ido robando. Array de cartas (5 máximo)
- El constructor creará un array vacío para el atributo mano.
- Método void “**nuevaCarta**(Carta)”, añade una Carta pasada como parámetro a la mano del jugador.
- Método String “**toString**()”, que muestre la mano del jugador.
- Método int “**valorMano**()”, que calcule el valor de la mano del jugador, la suma del valor de sus cartas.
- Método ArrayList<Cartas> **getMano**(), que devuelva las cartas actuales del jugador.
- Si necesitas más atributos o métodos, puedes añadirlos.

5. Clases **JugadorBlack** y **Crupier**.

Estas dos clases heredan de Jugador. Llevará un atributo nombre, que para el Crupier será siempre “crupier”, y para el jugador el nick del participante, o sea que cada clase tiene un constructor que hace cosas diferentes y llaman al constructor de Jugador. Para crupier se añade un atributo nuevo llamado *límite*, que por defecto será 17, valor de su mano donde tendrá que plantarse obligatoriamente.

6. Clase **Partida**. Llevará el jugador, el crupier y la baraja. Debes decidir tú qué implementar de esta clase y cómo. Tendrás que crear los jugadores (jugador y crupier), crear un baraja inglesa, …, y métodos como asignarCarta(Jugador j), etc. De esa baraja tendrás que ir repartiendo cartas, quitándolas de la baraja y asignándoselas a las manos de los jugadores. Tendrás que hacer un main(), donde en un menú pedirás nueva partida o salir. Una vez iniciada la partida, repartirás dos cartas al jugador y al crupier y luego tendrás dos opciones “Pedir carta” o “Plantarte”. Al pedir carta si te pasas pierdes, si haces black Jack ganas, sino puedes seguir pidiendo o plantarte. El crupier cuando tu pides carta el hace su juego automático, tiene que pedir cartas hasta tener más de 17, en cuyo caso se planta, si se ha pasado pierde, si no depende de lo que hagas tú.

   Puedes ir haciendo un toString mostrando cartas del jugador y del crupier cada vez.

   Ganador: si ninguno se ha pasado, ni tiene Blackjack, comprobamos si alguno suma 21. Gana el que tenga 21 con menos cartas. Si los dos están plantados gana el que más se acerque a 21, o en caso de empate el que tenga menos cartas.

   La idea del juego es ir mostrando las manos del jugador y del crupier hasta que uno se pase, o hasta que uno tenga blackjack o hasta que el jugador se plante, en cuyo caso ganará el de la mano con valor más cercano a 21 y con menos cartas.