Ejercicio 1. Practicar Enumerados

Define un tipo enumerado para representar los tipos de madera con los que trabaja una fábrica:

Tipo	Color	Peso en Kg/m³
ROBLE	Castaño verdoso	800
CAOBA	Marrón oscuro	770
NOGAL	Marrón rojizo	820
CEREZO	Marrón cereza	790
BOJ	Marrón negruzco	675

Define una clase principal que haciendo uso del tipo anterior, muestre en pantalla todos los tipos de madera con los que trabaja la empresa y solicite al usuario que seleccione uno, y a continuación muestre en pantalla las características de color y peso del tipo de madera seleccionado.

Ejercicio 2. Dada una clase Cuenta con atributos saldo y titular, con métodos ingreso y reintegro de una cantidad, teniendo en cuenta que al reintegrar el saldo nunca pueda ser negativo ….

Añade a la clase Cuenta:

- Modifica el atributo titular para que sea de tipo Persona (reutiliza la clase Persona hecha en un ejercicio anterior)
- Un atributo **estado** de tipo enumerado con los posibles estados {OPERATIVA, CERRADA, INMOVILIZADA y NUMEROS_ROJOS}
- Métodos setter y getter para este nuevo atributo.
- Sobrecarga los métodos ingreso y reintegro como métodos varargs.
- Un atributo **codigo** que será el código único para cada objeto de tipo Cuenta.
- Un atributo de clase llamado **ultimoCodigo** que contenga el último código de cuenta asignado. Este atributo permitirá asignar un nuevo código único a la siguiente Cuenta que sea creada.
- Método de clase **getCodigoCuenta**() que devuelva el siguiente código de cuenta que podemos utilizar al crear una nueva Cuenta.
- Una lista (array) de movimientos de la cuenta, que además de la cantidad del movimiento, tendrá un "concepto", que irá asociado a la cantidad. Cada vez que se produzca un ingreso o reintegro se deberá registrar dicho movimiento.
- Los métodos ingreso y reintegro se podrán utilizar indicando la cantidad y el concepto, pero también indicando solo la cantidad o cantidades, en cuyo caso el concepto será "Ingreso en cuenta" o "Reintegro en cuenta" respectivamente.
- un nuevo atributo **cotitulares,** que almacene una lista con los cotitulares de una cuenta →los cotitulares serán de tipo "Persona". Modifica los constructores para incluir esta propiedad.
- Piensa y añade los métodos getter y setter que consideres necesarios, a parte de los indicados, para los nuevos atributos. Justifica este apartado.
- Añade un método **addCotitulares** que pueda recibir un número variable de nombres de cotitulares y que los asigne al atributo correspondiente.
- Crea una aplicación principal que cree varias cuentas empleando los diferentes constructores, de forma que utilzando también los métodos set, todas tengan todos sus datos. Realiza varias operaciones sobre ellas de ingreso/reintegro.

- Finalmente muestra los datos de todas ellas así como sus movimientos y comprueba el efecto del Aliasign sobre el vector movimientos de Cuenta.

Ejercicio 3. Practicar la clientela.

Queremos desarrollar una aplicación para la **gestión de concursos** de programación. Un **concurso** se caracteriza por las siguientes **propiedades**:

- nombre que identifica al concurso. Esta propiedad se puede modificar.
- número de problemas: esta propiedad no se puede modificar una vez establecida. Este número establece el rango de índices válidos para los problemas, que va desde 0 hasta número de problemas – 1.
- equipos participantes: lista que almacena los nombres (cadena) de los equipos.
- número de equipos (propiedad calculada, es decir, no es un atributo).
- envíos: lista que contiene los envíos de los equipos participantes como respuestas a los problemas (se describe más adelante).

La clase ofrece dos **constructores**. En el primero se establece el nombre y el número de problemas. Inicialmente las listas están vacías (equipos y envíos). En el segundo constructor se omite el número de problemas, tomando el valor por defecto 5 (constante).

La clase ofrece la siguiente **funcionalidad** para la gestión de los equipos:

- Añadir equipos. Esta operación acepta una secuencia de nombres de equipos para registrarlos en el concurso (argumento tamaño variable). Nótese que debe evitarse introducir en la lista de equipos participantes un nombre de equipo repetido.
- Eliminar un equipo, estableciendo su nombre como parámetro. Retorna un valor booleano indicando si ha podido eliminarlo. Nótese que la eliminación de un equipo también implica el borrado de todos sus envíos.

La funcionalidad más importante de un concurso es la gestión de envíos.

Un **envío** representa la información sobre la respuesta de un equipo a un problema del concurso. Las **propiedades** de un envío son las siguientes:

- nombre del equipo.
- número de problema.
- respuesta al problema (cadena).

Todas las propiedades se establecen en el constructor y una vez establecidas no pueden cambiar. Esto significa que este tipo de datos define objetos inmutables.

El concurso ofrece una operación para registrar los envíos. La operación recibe como parámetros el nombre del equipo, el número de problema y la respuesta. Los pasos que realiza esta operación son los siguientes:

- En primer lugar, se comprueba si se cumplen los requisitos para aceptar un envío. Estos son: el equipo está en la lista de equipos, el índice del problema es válido, y la respuesta no es nula ni la cadena vacía. En caso de no cumplir estos requisitos, retorna un valor nulo indicando que el envío no ha sido registrado.

- Si se cumplen los requisitos anteriores, se construye un envío y lo almacena en la lista de envíos. Finalmente, retorna el envío.

Desarrolla un programa ejecutable que:

- Declare y construya un concurso con nombre “Sesión 1” y 2 problemas.
- Declare y construya un concurso con nombre “Sesión 2” y 3 problemas.
- Declare y construya un concurso con nombre “Sesión 3” y 3 problemas.
- Crea una lista de concursos y añade los tres concursos creados en los pasos anteriores.
- Recorre los concursos y realiza lo siguiente:
    - Muestra el nombre del concurso.
    - Añade los equipos “Equipo 1”, “Equipo 2” y “Equipo 3”.
    - Recorre los equipos para que cada uno realice el envío “(1, 2)” a todos los problemas.

Ejercicio 4. Practicar Herencia

Identifica la superclase Persona a partir de las siguientes clases

Piensa cuáles serían los atributos y métodos que tendría la superclase Persona, y cómo quedarían clases Futbolista, Entrenador y Masajista.

Ejercicio 5: Practicar Herencia

Partiendo de una clase **Bicicleta**, la cual tiene tres atributos: velocidadActual, platoActual y piñonActual, de tipo entero y cuatro métodos: acelerar(), frenar(), cambiarPlato(int pPlato) y cambiarPiñon(int pPiñon), Implementa las clases **BicicletaMontaña** y **BicicletaTandem**

- **BicicletaMontaña** tiene un atributo suspensión de tipo entero y un método cambiarSuspension(int pSuspension)
- **BicicletaTandem** tiene un atributo numAsientos de tipo entero.
- Crear los constructores de estas clases para inicializar todos sus atributos, haciendo uso de super.

Ejercicio 6. Practicar Herencia, enumerados

Crearemos una clase llamada **AparatoElectrico** con las siguientes características:

- Atributos: **precioBase, color, consumo energético** (letras entre A y F) y **peso**.
- Por defecto, el color será blanco, el consumo energético será F, el precioBase será de 100 € y el peso de 5 kg.
- Los colores disponibles son únicamente blanco, negro, rojo, azul y gris. Da igual en mayúsculas que en minúsculas.
- Los constructores que se implementaran serán:
  - § Un constructor por defecto.
  - § Un constructor con el precio y peso. El resto por defecto.
  - § Un constructor con todos los atributos.
- Los métodos que implementará serán:

§ Métodos **get** de todos los atributos.

§ **comprobarConsumoEnergetico(char letra)**: comprobará que la letra es correcta, sino es correcta usará la letra por defecto. Se invocará al crear el objeto y no será visible desde fuera de la clase.

§ **comprobarColor(String color)**: comprueba que el color es correcto, sino lo es usa el color por defecto. Se invocara al crear el objeto y no será visible desde fuera de la clase.

§ **precioFinal()**:según el consumo energético, aumentará su precio, y según su tamaño, también. El método devolverá el precio final. Esta es la lista de precios:

LETRA	PRECIO
A	100€
B	80€
C	60€
D	50€
E	30€
F	10€

TAMAÑO	PRECIO
Entre 0 y 19 Kg	10€
Entre 20 y 49Kg	50€
Entre 50 y 79 Kg	80€
Más de 80Kg	100€

Crearemos una subclase de AparatoElectrico llamada **Secadora** con las siguientes características:

- Atributo **carga**, además de los atributos heredados.

- Por defecto, la carga es de 5 kg.
- Los constructores que se implementaran serán:

  § Un constructor por defecto.

  § Un constructor con el precio y peso. El resto por defecto.

  § Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

- Los métodos que se implementara serán:

§ Método **get** de carga.

§ **precioFinal()**: si tiene una carga mayor de 30 kg, aumentara el precio 50 €, sino es así no se incrementará el precio. Llama al método padre y añade el código necesario. Recuerda que las condiciones que hemos visto en la clase AparatoElectrico también deben afectar al precio.

Crearemos una subclase de AparatoElectrico llamada **Vitroceramica** con las siguientes características:

- Atributos: **induccion** (booleano), además de los atributos heredados.
- Por defecto, la propiedad inducción será false.
- Los constructores que se implementaran serán:

  § Un constructor por defecto.

  § Un constructor con el precio y peso. El resto por defecto.

  § Un constructor con el valor para induccion y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

§ Método **get** de resolución y sintonizador TDT.

§ **precioFinal()**: las vitrocerámicas de inducción verán su precio incrementado un 30%. Recuerda que las condiciones que hemos visto en la clase AparatoElectrico también deben afectar al precio.

Crea un programa ejecutable que realice lo siguiente:

- Crea un array para 10 AparatosElectricos.
- Asigna a cada posición un objeto de las clases anteriores con los valores que desees (prueba a crear aparatos eléctricos con distintos tipos de consumo energético, distinto peso, secadoras de distinta carga, vitrocerámicas de inducción y convencionales …..
- Después recorre este array y ejecuta el método precioFinal() sobre cada objeto para visualizarlo en pantalla.
- Además deberás calcular y mostrar en pantalla la suma de los precios de los aparatos eléctricos según su clase (secadoras por un lado, vitroceramicas por otro, aparatos eléctricos …). Recuerda el uso del operador **instanceof**.

Ejercicio 7: Practicar Herencia

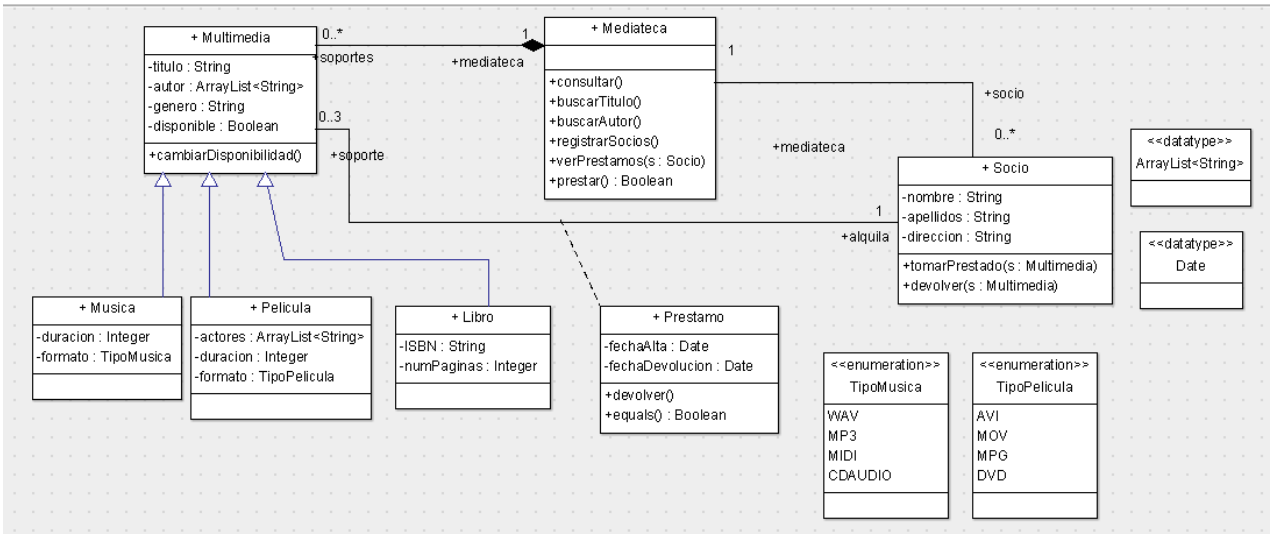
Escribe una aplicación Java para la gestión de una **Mediateca**. La mediateca se compone de objetos **Multimedia**, los cuales pueden ser películas, discos o libros. Esta clase contiene título, autor o autores, género y un atributo que indica si el objeto está disponible o no para su préstamo. Las **películas** disponen de un atributo con el formato de la misma que puede ser avi, mov, mpg o dvd, un atributo duración en minutos y una lista con los actores protagonistas que aparecen en la misma. La **música** tiene un atributo formato que puede ser wav, mp3, midi, cdAudio, un atributo con la duración en minutos. Los libros cuentan con un isbn y el número de páginas.

En la clase Multimedia el valor de todos los atributos se pasa por parámetro en el momento de crear el objeto. Esta clase tiene además, un método para devolver cada uno de los atributos, un método toString() que devuelve la información del objeto y un método para modificar la disponibilidad del objeto, también dispone de un método equals() que recibe un objeto de tipo Multimedia y devuelve true en caso de que el título y el autor sean iguales y false en caso contrario.

Para poder alquilar alguno de los objetos de la mediateca se debe ser socio. De un socio se almacenará nombre, apellidos y dirección. Los socios pueden coger en préstamo un objeto multimedia y devolverlo. El préstamo de un objeto dispondrá de una fecha de alta y devolución que deberá ser como mucho de 30 días. Los socios pueden tener como máximo 3 objetos en préstamo.

La aplicación mediateca debe permitir consultar los objetos disponibles, buscar un determinado título, buscar los objetos asociados a un determinado autor, registrar socios, prestar, ver los prestamos en curso… y otras opciones que se te ocurran.

El diagrama de clases es el siguiente:



Ejercicio 8: Practicar Herencia

El ejercicio consiste en la implementación de una aplicación para que los propietarios de un coche puedan compartir las plazas de su coche cuando vayan a realizar un viaje. Antes de presentar el concepto viaje, se introduce el concepto que representa la reserva de un viaje. Una **reserva** corresponde con la reserva de un número de las plazas ofertadas en un viaje. Se caracteriza por las siguientes propiedades:

- Código de reserva: una cadena de texto generada automáticamente en el constructor de forma aleatoria utilizando UUID.randomUUID().toString();
- Usuario: cadena de texto que identifica al usuario que realiza la reserva.
- Número de plazas: entero que representa el número de plazas que se han reservado.
- Fecha en la que se realiza la reserva. Esta propiedad se inicializa en la construcción y corresponde a la fecha actual.

El constructor de la reserva recibe como parámetro el usuario y el número de plazas. Además, las propiedades no se pueden modificar tras la construcción. Por tanto, esta clase define objetos inmutables.

Un **viaje** representa una oferta que hace un propietario para compartir su coche. Las propiedades que caracterizan a un viaje son:

- Propietario del vehículo (cadena de texto).
- Coche: cadena de texto con el modelo del coche utilizado para el viaje.
- Ruta: cadena que describe el viaje (por ejemplo, “Murcia – Valencia – Barcelona”).
- Fecha de salida.
- Plazas ofrecidas.
- Reservas: lista con las reservas que se han realizado.
- Número de plazas reservadas. Corresponde con la suma del número de plazas de las reservas realizadas.
- Plazas disponibles. Corresponde con la diferencia entre las plazas ofrecidas y elnúmero de plazas reservadas.

Las propiedades propietario, coche, ruta, fecha de salida y número de plazas no se pueden modificar una vez establecidas en la construcción. Las reservas serán añadidas a través de un método (se describe más adelante). Inicialmente la lista de reservas es vacía. En el constructor se establece el propietario, el coche, la ruta, la fecha de salida y las plazas ofrecidas. También se ofrece un segundo constructor en el que se omite el número de plazas tomando como valor por defecto 1 .

La funcionalidad que ofrece la clase viaje es:

- Realizar reserva: el método recibe como parámetro el usuario (cadena de texto) que realiza la reserva y el número de plazas que quiere reservar. Para poder formalizarla reserva se tiene que cumplir que: queden plazas disponibles y que el viaje no se haya realizado todavía (la fecha en la que se está realizando la reserva tiene que ser anterior a la fecha de salida). Si se cumplen estas restricciones, se generará un objeto reserva que se almacenará en la lista de reservas y que además se devuelve como resultado de la

ejecución. En el caso de que no se cumpla alguna de las condiciones el método devolverá null.

- Consultar la reserva asociada a un código: el método recibe como parámetro un código de reserva y devuelve el objeto reserva cuyo código es igual al recibido como parámetro, o null si no existe ninguna reserva con ese código.

Además, se definen dos tipos de viaje especiales:

- Viaje premium. Es un tipo de viaje que se caracteriza por permitir cancelar las reservas hasta el día antes de la fecha de salida. Para realizar la cancelación se proporciona el código de la reserva y devuelve un valor booleano indicando si se ha efectuado la cancelación.
- Viaje selectivo. Es un tipo de viaje que permite vetar a los usuarios. Por tanto, este tipo de viaje añade una nueva propiedad que contiene el conjunto de usuarios vetados. Inicialmente la colección estará vacía, pero se pueden añadir y eliminar usuarios vetados en cualquier momento. Por tanto, no se aceptará una reserva si el usuario que la realiza está vetado.

Por último, implementa el método toString de la clase Object en las clases que implementan los viajes siguiendo las recomendaciones de la asignatura.

Recuerda que:

- El método toString de la clase Viaje debe mostrar la información de todas las propiedades, incluidas las reservas. Por ello se debe implementar el método toString en la clase Reserva.
- toString sólo deben redefinirse en aquellas clases de la jerarquía que añadan nuevas propiedades, refinando en este caso la implementación de la clase padre.

Programa:

Implementa el siguiente programa para probar la funcionalidad:

- Crea los siguientes viajes cuyo propietario sea “José Antonio” y coche “Seat León”:
  - Viaje “Murcia-Cartagena” con fecha de salida 9/junio/2020 con el número de plazas por defecto.
  - Viaje selectivo “Murcia–Campus” con fecha de salida 1 0/junio/2020 y 4 plazas.
  - Vetar a “Beatriz” en el viaje selectivo.
  - Viaje premium “Murcia-Barcelona” con fecha de salida 1 5/junio/2020 y 6 plazas.
- Realiza las siguientes reservas y muestra el resultado de cada paso en la consola:
  - "Alberto" hace una reserva de dos plazas en el viaje “Murcia-Cartagena”. El resultado debe ser null porque el número de plazas ofertadas es 1 .
  - "Enrique" hace una reserva de 3 plazas en el viaje “Murcia-Campus”.
  - "Beatriz" hace una reserva de 1 plaza en el viaje “Murcia-Campus”. El resultado debe ser null, quedan plazas disponibles, pero “Beatriz” está vetada.
  - "Ana" hace una reserva de dos plazas en el viaje “Murcia-Barcelona”.
  - “Ana” cancela su reserva.
- Crea un conjunto con los tres viajes anteriores.
- Recorre el conjunto de viajes:

- Si el viaje es selectivo, quitar el veto a “Beatriz”.
- Imprime (toString) la información de cada viaje.