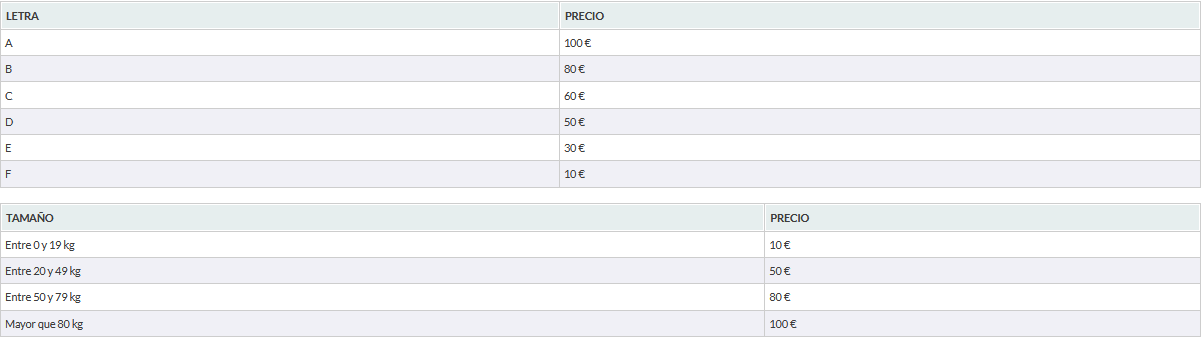
Ejercicios Herencia y Polimorfismo

1.- Crearemos una supeclase llamada **Electrodomestico** con las siguientes características:

* Sus atributos son **precio base, color, consumo energético** (letras entre A y F) y **peso**. Indica que se podrán heredar.
* Por defecto, el color sera blanco, el consumo energético sera F, el precioBase es de 100 € y el peso de 5 kg. Usa constantes para ello.
* Los colores disponibles son blanco, negro, rojo, azul y gris. No importa si el nombre esta en mayúsculas o 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 implementara serán:
  + Métodos get de todos los atributos.
  + **comprobarConsumoEnergetico(char letra)**: comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Se invocara al crear el objeto y no sera visible.
  + **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 sera visible.
  + **precioFinal()**: según el consumo energético, aumentara su precio, y según su tamaño, también. Esta es la lista de precios:



Crearemos una subclase llamada **Lavadora** con las siguientes características:

* Su atributo es **carga**, ademas de los atributos heredados.
* Por defecto, la carga es de 5 kg. Usa una constante para ello.
* 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 incrementara el precio. Llama al método padre y añade el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Crearemos una subclase llamada **Television** con las siguientes características:

* Sus atributos son **resolución** (en pulgadas) y **sintonizador TDT** (booleano), ademas de los atributos heredados.
* Por defecto, la resolución sera de 20 pulgadas y el sintonizador sera 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 la resolución, sintonizador TDT 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()**: si tiene una resolución mayor de 40 pulgadas, se incrementara el precio un 30% y si tiene un sintonizador TDT incorporado, aumentara 50 €. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Ahora crea una clase ejecutable que realice lo siguiente:

* Crea un array de Electrodomesticos de 10 posiciones.
* Asigna a cada posición un objeto de las clases anteriores con los valores que desees.
* Ahora, recorre este array y ejecuta el método precioFinal().
* Deberás mostrar el precio de cada clase, es decir, el precio de todas las televisiones por un lado, el de las lavadoras por otro y la suma de los Electrodomesticos (puedes crear objetos Electrodomestico, pero recuerda que Television y Lavadora también son electrodomésticos). Recuerda el uso operador instanceof.

Por ejemplo, si tenemos un Electrodomestico con un precio final de 300, una lavadora de 200 y una televisión de 500, el resultado final sera de 1000 (300+200+500) para electrodomésticos, 200 para lavadora y 500 para televisión.

2.- Crearemos una clase llamada **Serie** con las siguientes características:

* Sus atributos son **titulo, numero de temporadas**, **entregado, genero y creador.**
* Por defecto, el numero de temporadas es de 3 temporadas y entregado **false**. El resto de atributos serán valores por defecto según el tipo del atributo.
* Los constructores que se implementaran serán:
  + Un constructor por defecto.
  + Un constructor con el titulo y creador. El resto por defecto.
  + Un constructor con todos los atributos, excepto de entregado.
* Los métodos que se implementara serán:
  + Métodos get de todos los atributos, excepto de entregado.
  + Métodos set de todos los atributos, excepto de entregado.
  + Sobrescribe los métodos toString.

Crearemos una clase **Videojuego** con las siguientes características:

* Sus atributos son **titulo, horas estimadas, entregado, genero y compañia**.
* Por defecto, las horas estimadas serán de 10 horas y entregado false. El resto de atributos serán valores por defecto según el tipo del atributo.
* Los constructores que se implementaran serán:
  + Un constructor por defecto.
  + Un constructor con el titulo y horas estimadas. El resto por defecto.
  + Un constructor con todos los atributos, excepto de entregado.
* Los métodos que se implementara serán:
  + Métodos get de todos los atributos, excepto de entregado.
  + Métodos set de todos los atributos, excepto de entregado.
  + Sobrescribe los métodos toString.

Como vemos, en principio, las clases anteriores no son padre-hija, pero si tienen en común, por eso vamos a hacer una interfaz llamada **Entregable** con los siguientes métodos:

* **entregar()**: cambia el atributo prestado a true.
* **devolver()**: cambia el atributo prestado a false.
* **isEntregado()**: devuelve el estado del atributo prestado.
* Método **compareTo (Object a)**, compara las horas estimadas en los videojuegos y en las series el numero de temporadas. Como parámetro que tenga un objeto, no es necesario que implementes la interfaz Comparable. Recuerda el uso de los casting de objetos.

Implementa los anteriores métodos en las clases Videojuego y Serie. Ahora crea una aplicación ejecutable y realiza lo siguiente:

* Crea dos arrays, uno de **Series** y otro de **Videojuegos**, de 5 posiciones cada uno.
* Crea un objeto en cada posición del array, con los valores que desees, puedes usar distintos constructores.
* Entrega algunos **Videojuegos** y **Series** con el método **entregar()**.
* Cuenta cuantos **Series** y **Videojuegos** hay entregados. Al contarlos, devuélvelos.
* Por último, indica el **Videojuego** tiene más horas estimadas y la serie con mas temporadas. Muestralos en pantalla con toda su información (usa el método toString()).

3.- Queremos representar con programación orientada a objetos, un aula con estudiantes y un profesor.  
Tanto de los estudiantes como de los profesores necesitamos saber su nombre, edad y sexo. De los estudiantes, queremos saber también su calificación actual (entre 0 y 10) y del profesor que materia da.  
Las materias disponibles son matemáticas, filosofía y física.  
Los estudiantes tendrán un 50% de hacer novillos, por lo que si hacen novillos no van a clase pero aunque no vayan quedara registrado en el aula (como que cada uno tiene su sitio).  
El profesor tiene un 20% de no encontrarse disponible (reuniones, baja, etc.)  
Las dos operaciones anteriores deben llamarse igual en Estudiante y Profesor (polimorfismo).  
El aula debe tener un identificador numérico, el número máximo de estudiantes y para que esta destinada (matemáticas, filosofía o física). Piensa que más atributos necesita.  
Un aula para que se pueda dar clase necesita que el profesor esté disponible, que el profesor de la materia correspondiente en el aula correspondiente (un profesor de filosofía no puede dar en un aula de matemáticas) y que haya más del 50% de alumnos.  
El objetivo es crear un aula de alumnos y un profesor y determinar si puede darse clase, teniendo en cuenta las condiciones antes dichas.  
Si se puede dar clase mostrar cuantos alumnos y alumnas (por separado) están aprobados de momento (imaginad que les están entregando las notas).  
NOTA: Los datos pueden ser aleatorios (nombres, edad, calificaciones, etc.) siempre y cuando tengan sentido (edad no puede ser 80 en un estudiante o calificación ser 12).

 4.- Vamos a hacer una baraja de cartas españolas orientado a objetos.

Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas)

La baraja estará compuesta por un conjunto de cartas, 40 exactamente.

Las operaciones que podrá realizar la baraja son:

* barajar: cambia de posición todas las cartas aleatoriamente
* siguienteCarta: devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.
* cartasDisponibles: indica el número de cartas que aún puede repartir
* darCartas: dado un número de cartas que nos pidan, le devolveremos ese número de cartas (piensa que puedes devolver). En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.
* cartasMonton: mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario
* mostrarBaraja: muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.

5.- Nos piden hacer una un programa que gestione empleados.

Los empleados se definen por tener:

* Nombre
* Edad
* Salario

También tendremos una constante llamada PLUS, que tendrá un valor de 300€

Tenemos dos tipos de empleados: repartidor y comercial.

El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double).

El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).

Crea sus constructores, getters and setters y toString (piensa como aprovechar la herencia).

No se podrán crear objetos del tipo Empleado (la clase padre) pero si de sus hijas.

Las clases tendrán un método llamado plus, que según en cada clase tendrá una implementación distinta. Este plus básicamente aumenta el salario del empleado.

* En comercial, si tiene más de 30 años y cobra una comisión de más de 200 euros, se le aplicara el plus.
* En repartidor, si tiene menos de 25 y reparte en la “zona 3”, este recibirá el plus.

Puedes hacer que devuelva un booleano o que no devuelva nada, lo dejo a tu elección.

Crea una clase ejecutable donde crees distintos empleados y le apliques el plus para comprobar que funciona.

6.- Nos piden hacer que gestionemos una serie de productos.

Los productos tienen los siguientes atributos:

* Nombre
* Precio

Tenemos dos tipos de productos:

* Perecedero: tiene un atributo llamado días a caducar
* No perecedero: tiene un atributo llamado tipo

Crea sus constructores, getters, setters y toString.

Tendremos una función llamada calcular, que según cada clase hará una cosa u otra, a esta función le pasaremos un numero siendo la cantidad de productos

* En Producto, simplemente seria multiplicar el precio por la cantidad de productos pasados.
* En Perecedero, aparte de lo que hace producto, el precio se reducirá según los días a caducar:
  + Si le queda 1 día para caducar, se reducirá 4 veces el precio final.
  + Si le quedan 2 días para caducar, se reducirá 3 veces el precio final.
  + Si le quedan 3 días para caducar, se reducirá a la mitad de su precio final.
* En NoPerecedero, hace lo mismo que en producto

Crea una clase ejecutable y crea un array de productos y muestra el precio total de vender 5  productos de cada uno. Crea tú mismo los elementos del array.

7.- Nos piden hacer un almacén, vamos a usar programación orientado a objetos.

En un almacén se guardan un conjunto de  bebidas.

Estos productos son bebidas como agua mineral y bebidas azucaradas (coca-cola, fanta, etc). De los productos nos interesa saber su identificador (cada uno tiene uno distinto), cantidad de litros, precio y marca.

Si es agua mineral nos interesa saber también el origen (manantial tal sitio o donde sea).

Si es una bebida azucarada queremos saber el porcentaje que tiene de azúcar y si tiene o no alguna promoción (si la tiene tendrá un descuento del 10% en el precio).

En el almacén iremos almacenado estas bebidas por estanterías (que son las columnas de la matriz).

Las operaciones del almacén son las siguientes:

* **Calcular precio de todas las bebidas**: calcula el precio total de todos los productos del almacén.
* **Calcular el precio total de una marca de bebida**: dada una marca, calcular el precio total de esas bebidas.
* **Calcular el precio total de una estantería**: dada una estantería (columna) calcular el precio total de esas bebidas.
* **Agregar producto**: agrega un producto en la primera posición libre, si el identificador esta repetido en alguno de las bebidas, no se agregará esa bebida.
* **Eliminar un producto**: dado un ID, eliminar el producto del almacén.
* **Mostrar información**: mostramos para cada bebida toda su información.

Puedes usar un main para probar las funcionalidades (añade productos, calcula precios, muestra información, etc)

8.- Vamos a hacer unas mejoras a la clase Baraja del ejercicio 5 de POO de los videos.

Lo primero que haremos es que nuestra clase Baraja será la clase padre y será abstracta.

Le añadiremos el número de cartas en total y el número de cartas por palo.

El método crearBaraja() será abstracto.

La clase Carta tendrá un atributo genérico que será el palo de nuestra versión anterior.

Creamos dos Enum:

* PalosBarEspañola:
  + OROS
  + COPAS
  + ESPADAS
  + BASTOS
* PalosBarFrancesa:
  + DIAMANTES
  + PICAS
  + CORAZONES
  + TREBOLES

Creamos dos clases hijas:

* BarajaEspañola: tendrá un atributo boolean para indicar si queremos jugar con las cartas 8 y 9 (total 48 cartas) o no (total 40 cartas).
* BarajaFrancesa: no tendrá atributos, el total de cartas es 52 y el número de cartas por palo es de 13. Tendrá dos métodos llamados:
  + cartaRoja(Carta<PalosBarFrancesa> c): si el palo es de corazones y diamantes.
  + cartaNegra(Carta<PalosBarFrancesa> c): si el palo es de tréboles y picas.

De la carta modificaremos el método toString()

Si el palo es de tipo PalosBarFrancesa:

* La carta número 11 será Jota
* La carta numero 12 será Reina
* La carta numero 13 será Rey
* La carta numero 1 será As

Si el palo es de tipo PalosBarFrancesa:

* La carta numero 10 será Sota
* La carta numero 12 será Caballo
* La carta numero 13 será Rey
* La carta numero 1 será As

9.- Una academia nos pide hacer un programa para hacer un pequeño test a sus alumnos.

Estas preguntas, para facilitar la inclusión, estarán escritas en un txt (incluido en la descarga del proyecto).

Una opción se compone de:

* El texto de la opción (digamos la respuesta)
* Es correcto o no

Una pregunta consta de:

* Pregunta (tendrá delante dos puntos y coma ;P;)
* Opciones de la pregunta (entre 2 y 4)
* Opción correcta (tendrá delante dos puntos y coma ;R;)
* Puntos

La pregunta no será válida en los siguientes casos:

* Las opciones no están entre 2 y 4.
* La opción correcta esta entre el número de opciones y es un número.
* Los puntos es un número entero.

Sus métodos son:

* mostrarPregunta(): muestra la pregunta con sus opciones.
* comprobarRespuesta(int respuestaUsuario): comprueba la respuesta del usuario si es correcta o no.
* Getter de los atributos.

Un test está formado por un conjunto preguntas y los puntos acumulados. Piensa que debemos saber por cual pregunta vamos.

Sus métodos son:

* cargarPreguntas(String fichero): carga todas las preguntas del fichero
* siguientePregunta(): devuelve la siguiente pregunta
* reiniciarTest(): nos permite reiniciar el test.
* realizarTest(): empieza el test y empieza a formular las preguntas

El fichero de preguntas tiene el siguiente formato:

;P;Pregunta 1

Opción 1 pregunta 1

Opción 2 pregunta 1

Opción 3 pregunta 1

Opción 4 pregunta 1

;R;Numero opción correcta

Puntos pregunta 1

;P;Pregunta 2

Opción 1 pregunta 2

Opción 2 pregunta 2

10.- Crea una aplicación que nos calcule el área de un circulo, cuadrado o triangulo. Pediremos que figura queremos calcular su área y según lo introducido pedirá los valores necesarios para calcular el área. Crea un método por cada figura para calcular cada área, este devolverá un número real. Muestra el resultado por pantalla

Aquí te mostramos que necesita cada figura:

* **Circulo**: (radio^2)\*PI
* **Triangulo**: (base \* altura) / 2
* **Cuadrado**: lado \* lado

11.- Crea una aplicación que nos convierta un número en base decimal a binario. Esto lo realizara un método al que le pasaremos el numero como parámetro, devolverá un String con el numero convertido a binario. Para convertir un numero decimal a binario, debemos dividir entre 2 el numero y el resultado de esa división se divide entre 2 de nuevo hasta que no se pueda dividir mas, el resto que obtengamos de cada división formara el numero binario, de abajo a arriba.

Veamos un ejemplo: si introducimos un **8** nos deberá devolver **1000**

12.- Crea un aplicación que nos convierta una cantidad de euros introducida por teclado a otra moneda, estas pueden ser a dolares, yenes o libras. El método tendrá como parámetros, la cantidad de euros y la moneda a pasar que sera una cadena, este no devolverá ningún valor, mostrara un mensaje indicando el cambio (void).

El cambio de divisas son:

* 0.86 libras es un 1 €
* 1.28611 $ es un 1 €
* 129.852 yenes es un 1 €

13.- Crea una aplicación que nos calcule el factorial de un número pedido por teclado, lo realizara mediante un método al que le pasamos el número como parámetro. Para calcular el factorial, se multiplica los números anteriores hasta llegar a uno. Por ejemplo, si introducimos un 5, realizara esta operación 5\*4\*3\*2\*1=120.

14.- Se va a implementar un simulador de Vehículos. Existen dos tipos de Vehículo: Coche y Camión.

Sus características comunes son la matrícula y la velocidad. En el momento de crearlos, la matrícula se recibe por parámetro y la velocidad se inicializa a 0. El método toString() de los vehículos devuelve información acerca de la matrícula y la velocidad. Además se pueden acelerar, pasando por parámetro la cantidad en km/h que se tiene que acelerar.

Los coches tienen además un atributo para el número de puertas, que se recibe también por parámetro en el momento de crearlo. Tiene además un método que devuelve el número de puertas.

Los camiones tienen un atributo de tipo Remolque que inicializa a null (para indicar que no tiene un remolque). Además, tiene un método ponRemolque(), que recibe el Remolque por parámetro, y otro quitaRemolque(). Cuando se muestre la información de un camión que lleve remolque, además de la matrícula y velocidad del camión, debe aparecer la información del remolque.

En esta clase, hay que sobreescribir el método acelerar de manera que si el camión tiene remolque y la velocidad más la aceleración superan los 100 km/h, se lance una excepción de tipo DemasiadoRapidoException.

Hay que implementar la clase Remolque. Esta clase tiene un atributo de tipo entero que es el peso y cuyo valor se le da en el momento de crear el objeto. Debe tener un método toString() que devuelva la información del remolque.

Implementar la clase DemasiadoRapidoException.

## Ayudas y planteamiento

Se va a implementar una clase Vehículo que sirve de superclase para Coche y Camión. Un Remolque no es un Vehículo, así que se implementaría sin heredar de Vehículo.

En esta clase Vehículo, se declara e implementa la parte común entre Coche y Camión, es decir:

Los atributos matrícula y velocidad.

El constructor que inicializa los dos atributos.

El método acelerar().

El método toString().

En Coche se añade el atributo numeroPuertas y el método para devolverlo. En el constructor se llama al constructor de la superclase.

En Camión se añade el atributo remolque, de tipo Remolque y los métodos para poner y quitar el remolque, además de sobreescribir el método toString() para mostrar la información del remolque.

Al sobreescribir el método acelerar() se lanza una excepción de tipo DemasiadoRapidoException que hay que indicar en la cabecera mediante una cláusula throws. Java no permite, en un método que sobreescribe a otro, lanzar una excepción que no se está lanzando en el método sobreescrito. Por este motivo, en el método acelerar() de Vehículo hay que indicar en la cláusula throws que se pueden lanzar este tipo de excepciones.

La clase Remolque no hereda de Vehículo. La clase Camión tiene una relación de composición con Remolque.