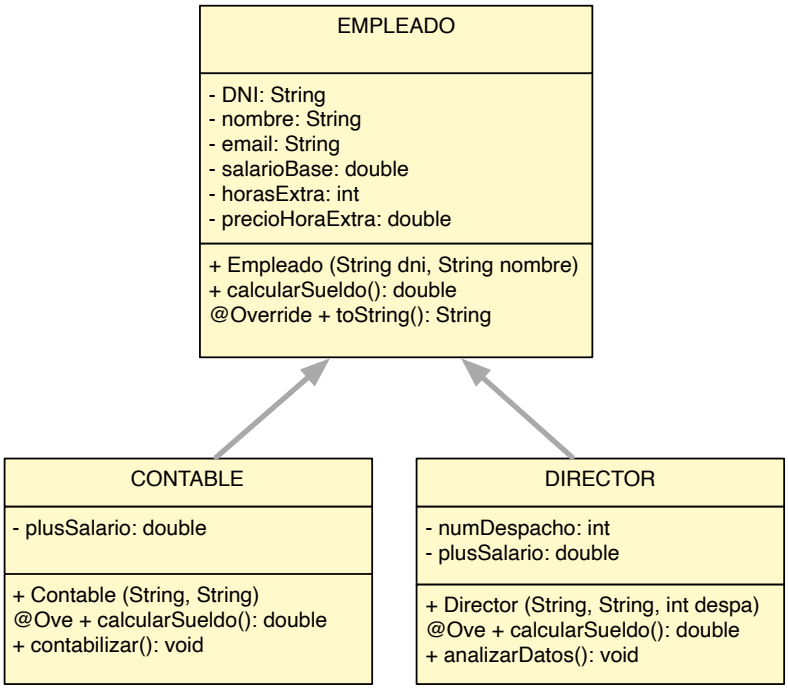
UT5 - Ejercicios HERENCIA - Relación1

Ejercicio01

Considere la siguiente jerarquía de clases.



- Para crear un empleado y un contable necesitaremos su DNI y su nombre. Adicionalmente, para crear un director necesitaremos indicar un despacho.
- El sueldo de cualquier empleado es equivalente al salario base mas las horas extra que haya echado.
- Las horas extra de los empleados y de los contables son a 10€/hora. Los directivos las cobran al doble, 20€/hora.
- El salario base es de 1000€. El plus para los contables es 200€ y para los directores es de 400€.
- Los contables contabilizan facturas (este método solo muestra un mensaje "Estoy contabilizando…."). Los directores analizan datos. Este método solo muestra un mensaje ("Estoy analizando muchos datos….").
- Redefine el método *toString()* en Empleado. Imprime el nombre y el DNI con la forma que puedes ver aquí –>

```
Celia Blue (DNI:3333A)
```

Hacer un programa principal (*Ejercicio1.java*) que haga lo siguiente:

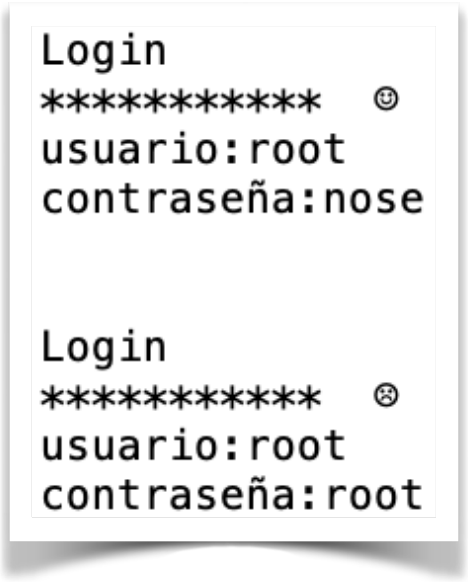
- Cree un empleado, un contable y un director.
- Añadir 5 horas extras a cada unos de ellos.
- Imprimir los tres y mostrar su sueldo. Poner el contable a contabilizar facturas y el director a analizar datos.

Hacer un programa principal (*Ejercicio1_polimorfismo.java*) que haga lo siguiente:

- Cree un empleado, un contable y un director y los añada a un ArrayList
- Recorrer el array y añadir 5 horas extras a cada unos de ellos, imprimirlos y mostrar su sueldo. Además, poner el contable a contabilizar facturas y el director a analizar datos (esto último usando el operador *instanceof*)

Ejercicio 02 - Login de usuarios

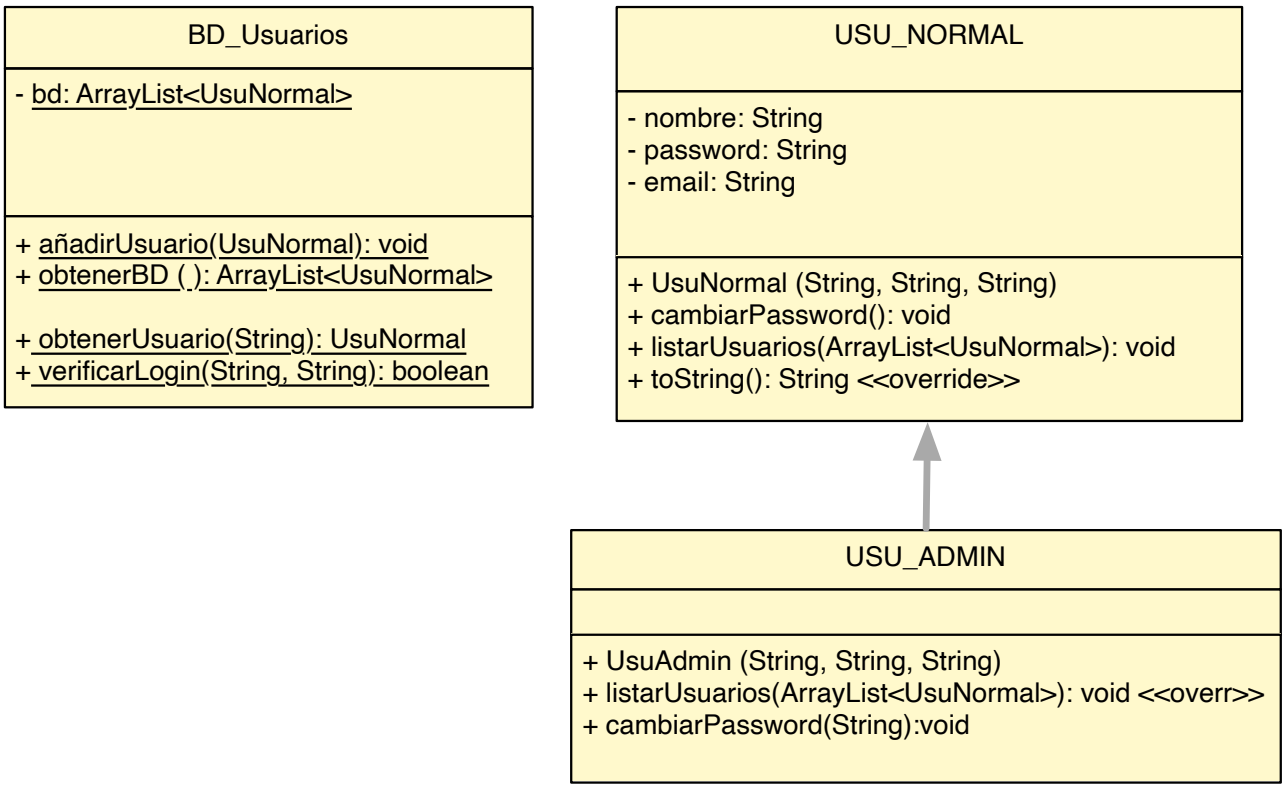
Hacer una aplicación que maneje el login de los usuarios de un sistema. Puedes ver el menú de login y el menú de usuario, así como las clases implicadas a continuación.




```
Login
********** 2/1
usuario:
contraseña:
```

```
*************************
Administrador: << root >>
*************************
1. Salir del sistema
2. Listar Usuarios
3. Cambiar Password
4. Cambiar Password Usu

Elige opción >
```

```
*************************
Usuario: << zipi >>
*************************
1. Salir del sistema
2. Listar Usuarios
3. Cambiar Password

Elige opción >
```



- Se permiten 2 intentos en el login. El menú mostrará una opción más si el usuario logeado es de tipo ADMIN.

- Los usuarios normales pueden cambiar su contraseña y pueden mostrar la lista de usuarios del sistema. En este último caso, las contraseñas se mostraran ocultas.
- Los administradores mostrarán la lista de usuarios en claro, es decir, mostrando sus contraseñas. Además, pueden cambiar la contraseña de cualquier usuario, además de la propia, obviamente.

```
                LISTA DE USUARIOS
=========================================
          zipi       zipi zipi@kk.com
          zape       zape zape@kk.com
A         root       root admin@admin.com
```

```
                LISTA DE USUARIOS
=========================================
          zipi ********** zipi@kk.com
          zape ********** zape@kk.com
A         root ********** admin@admin.com
```

- Al imprimir un usuario, mostrar el nombre, la contraseña y el email.
- Puesto que la clase *BD_Usuarios* va a ser única (no necesito crearme diversos objetos de este tipo), vamos a programarla con métodos estáticos. Esto me facilitará el acceso a los mismos, pues estarán disponibles en cualquier parte.
- El método *listarUsuarios(ArrayList<UsuNormal>):void* conceptualmente tiene más sentido haberlo programado en la clase BD_Usuarios, pero lo vamos a programar en los usuarios para practicar el concepto de <<override>> y ver como actúa correctamente la ligadura dinámica. Este método tiene como parámetro de entrada la base de datos, la cual la obtendremos con el método *BD_Usuarios.obtenerBD()*

PROGRAMA PRICIPAL - **Ejercicio02_menu.java**

- Crea el usuario normal "zipi", con contraseña "1234" y email "zipi@kk.com"
- Crea el usuario normal "zape", con contraseña "1234" y email "zipi@kk.com"
- Crea el usuario administrador "root", con contraseña "root" y email "root@kk.com"
- Logeate con zipi y muestra la lista de usuarios.
- Cambia el password de zipi a "zipi" y sal del sistema.
- Vuelve a logearte con "zipi" y comprueba que se ha cambiado el password.
- Logeate con "root" y muestra la lista de usuarios
- Cambia el password de "zape" por "zape"
- Vuelve a mostrar la lista de usuarios y comprueba que está todo ok

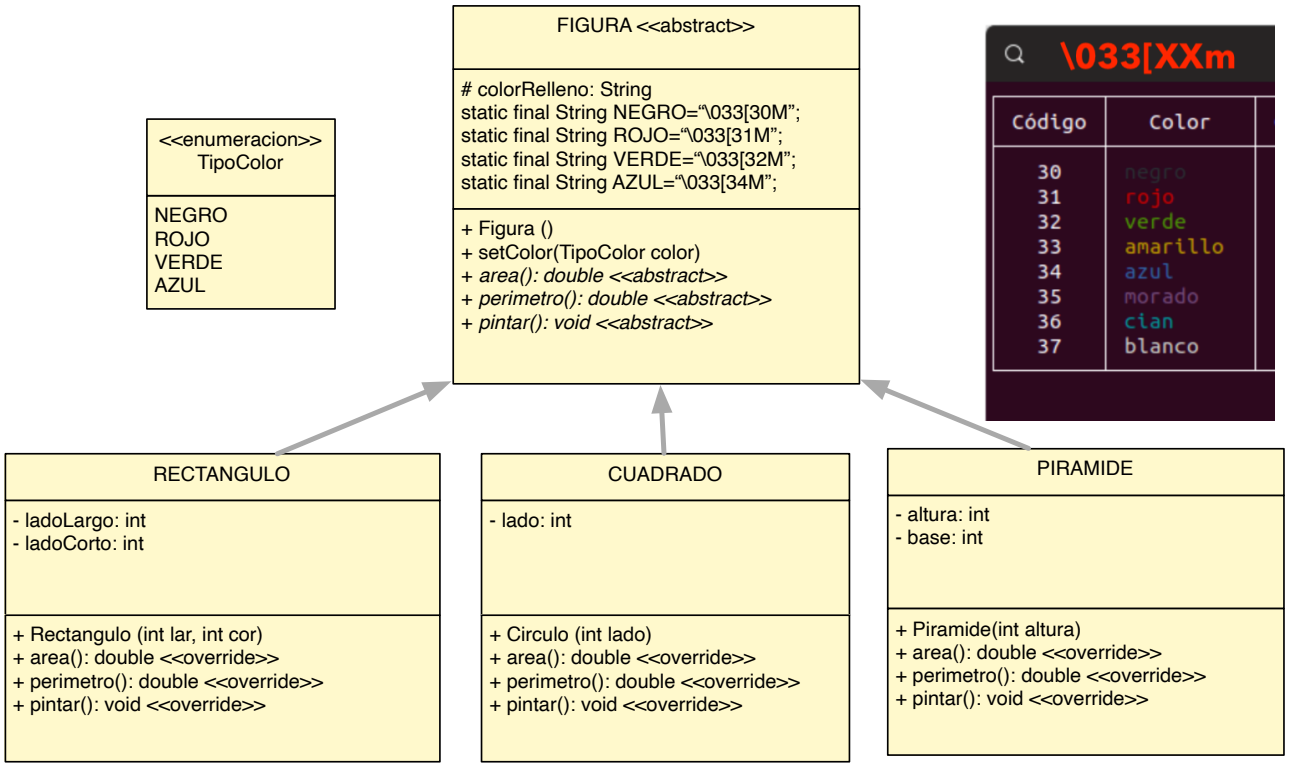
```
                LISTA DE USUARIOS
=========================================
          zipi       zipi zipi@kk.com
          zape       zape zape@kk.com
A         root       root admin@admin.com
```

- Sal del sistema y falla dos veces el login para salir

Ejercicio 03 - Figuras (clases abstractas)

Hacer una aplicación para dibujar "a lo cutre" figuras geométricas.

- Vamos a tener rectángulos, cuadrados y pirámides. Cada uno de ellos se podrá calcular el área y perímetro y pitarlo.
- La base de la pirámide será siempre fija dependiendo de la altura. La calcularemos siempre de la manera: `base=altura*2-1`
- Toda figura va a tener un *colorRelleno*, que será un String con el código del color. Definiremos 4 CONTANTES de tipo String correspondientes a la cadena que hay que usar con cada color.
- Por defecto, la figura será negra. Usaremos el método *setColor()* para rellenar el atributo del color. Usaremos como parámetro de entrada un enumerado.



Rectángulo 8x3

```
********
********
********
```

Cuadrado 3

```
* * *
* * *
* * *
```

Piramide altura 4

```
   *
  ***
 *****
*******
```

PROGRAMA PRINCIPAL 1 (Ejercicio3.java)

- Pinta un rectángulo rojo de 8x3, calcule su área y perímetro.
- Pinta un cuadrado azul de lado 3, calcule su área y perímetro.
- Pinta una pirámide verde de altura 4.

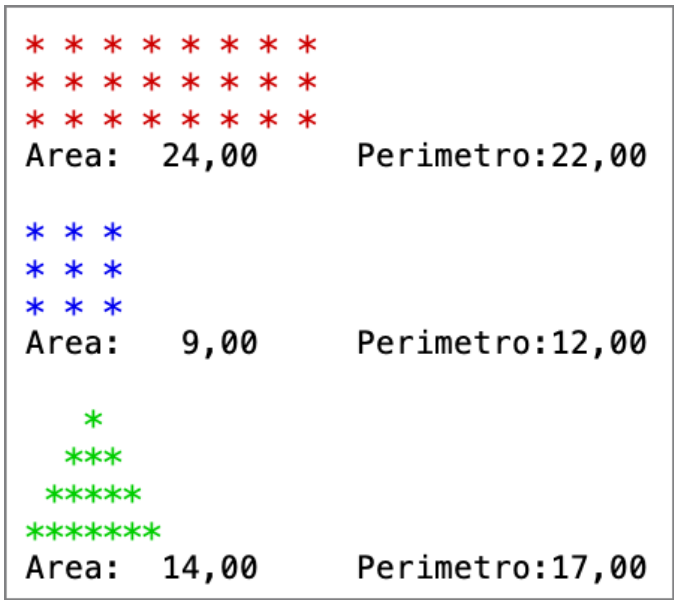
PROGRAMA PRINCIPAL 2 (Ejercicio3_polimorfismo.java)

- Crea un arraylist de figuras con las mismas que el main anterior.

- Recorre el arraylist para pintar cada una de las figuras, así como mostrar su área y perímetro.

```
 for (Figura f : listaFiguras) {
     f.pintar();
     System.out.printf("Area:%7.2f Perimetro %.2f\n",
f.area(),f.perimetro());
 }
```

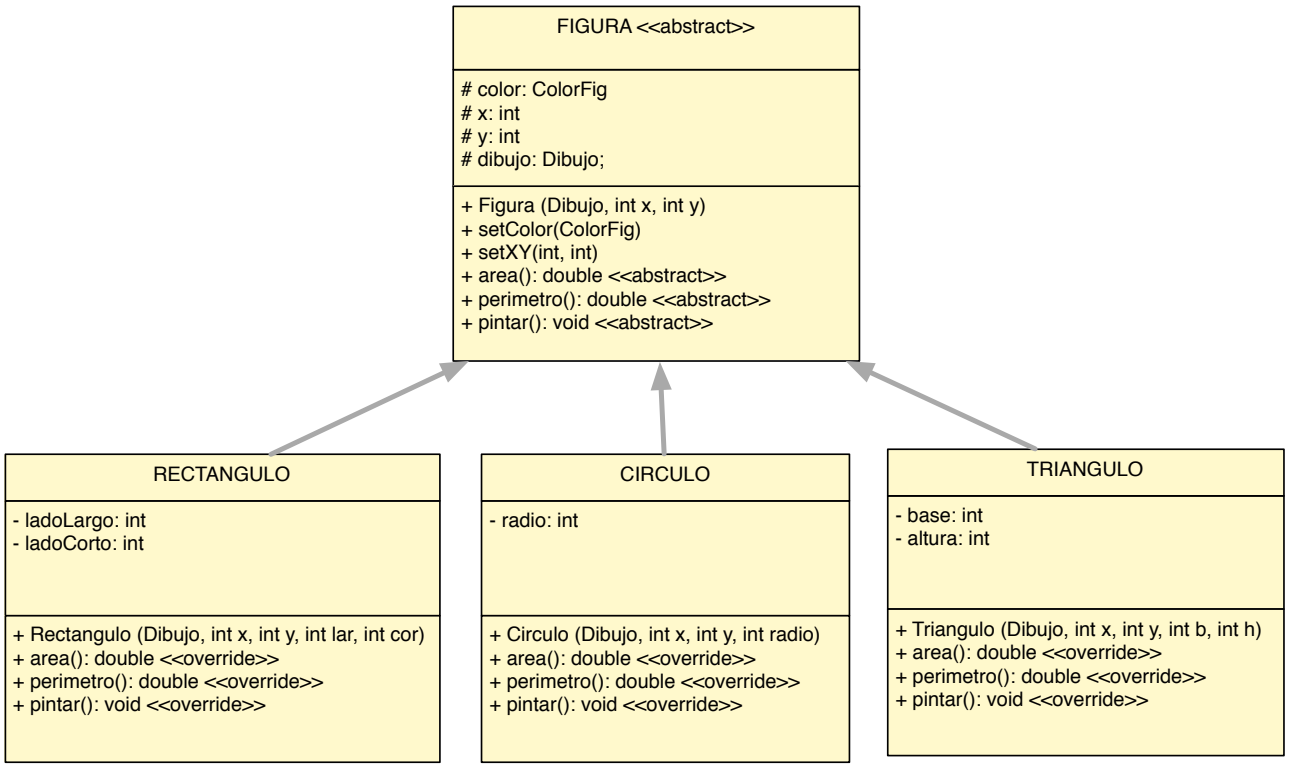
- El resultado debe ser el mismo que en el main anterior.



Ejercicio 04 - Figuras gráficas (clases abstractas y uso de librerías externas)
Hacer una aplicación para dibujar figuras geométricas con interfaz gráfica.

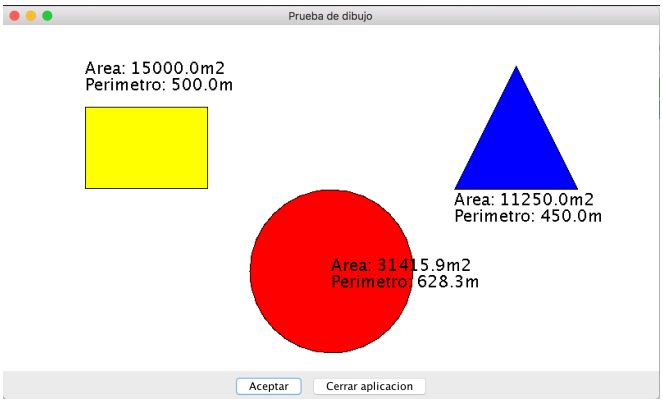
Para poder pintarlas, vamos a usar una librería llamada **fundamentos.jar** proporcionada por la Universidad de Cantabria (la podéis descargar de la plataforma). Esta librería nos proporciona muchas clases, entre ellas, la clase *Dibujo* que no permite dibujar figuras. Mirar la documentación para ver su uso.

- Toda figura va a tener un color, unas coordenadas iniciales (x,y) y se va a pintar en un dibujo. Por defecto, las figuras se pintan en negro.
- Vamos a tener rectángulos, círculos y triángulos. Cada uno de ellos se podrá calcular el área y perímetro y pitarlo.
- El área y perímetro deben de estar pintados debajo de la figura o cerca de ella.



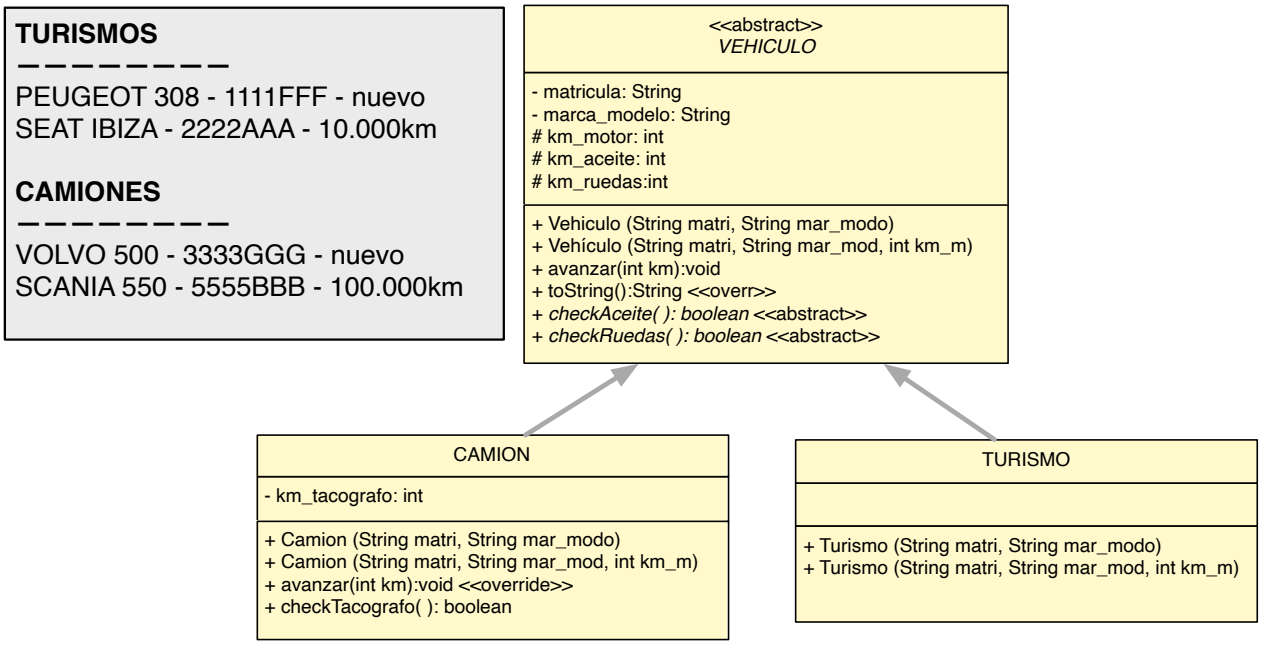
Hacer un programa principal que haga lo siguiente:

- Pinte un rectángulo amarillo, calcule su área y perímetro y lo indique debajo del dibujo.
- Pinte un circulo rojo, calcule su área y perímetro y lo indique debajo del dibujo.
- Pinte un triangulo azul, calcule su área y perímetro y lo indique debajo del dibujo.



Ejercicio 05 - Chequeo de vehículos (clases abstractas)
Hacer un programa que gestione el chequeo de los vehículos de una empresa.

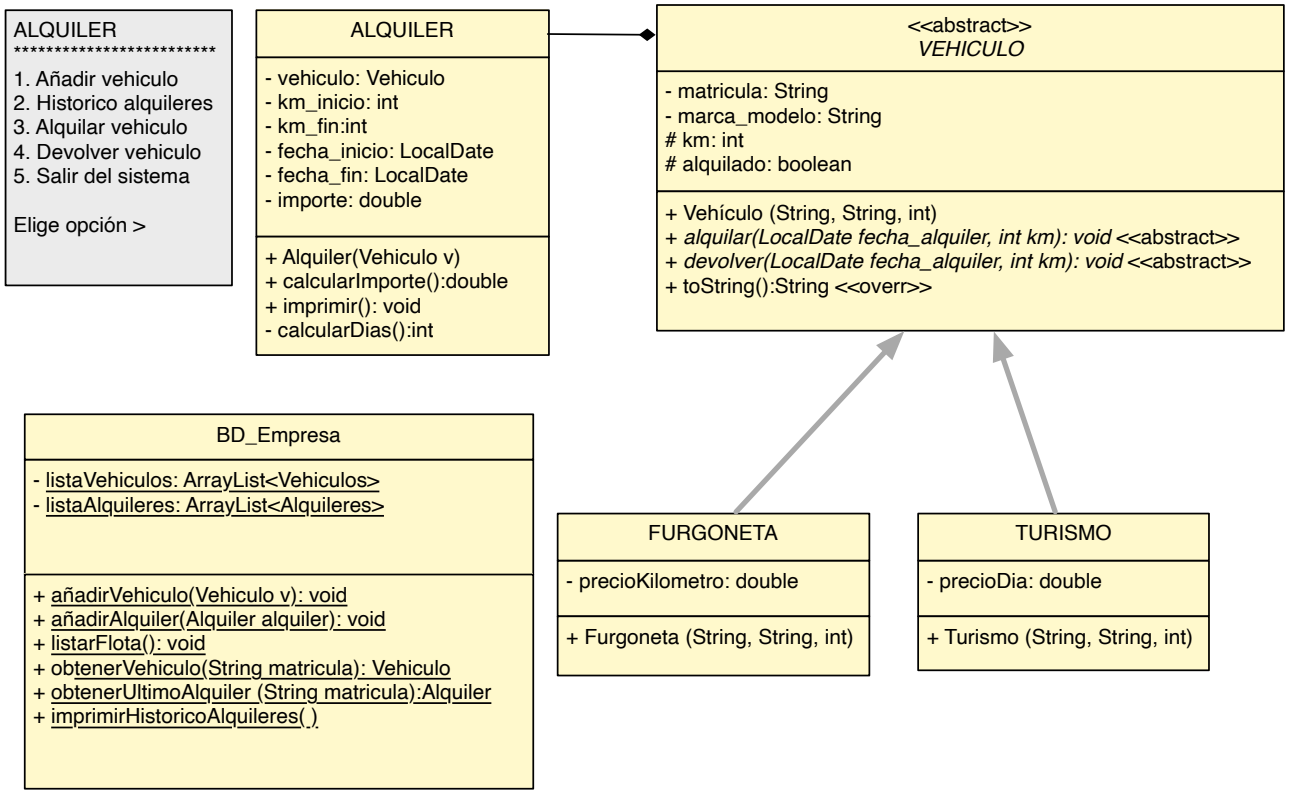
- Los vehículos se pueden comprar nuevos (con 0 kilómetros) o de segunda mano. Los vehículos de segunda se les pone a cero los km de las ruedas y del aceite (se les cambian siempre).
- Los camiones adicionalmente tienen un tacógrafo digital, del cual se guardan los km que lleva sin revisar.
- Se revisan periódicamente los kilómetros del aceite, de las ruedas y se revisa el funcionamiento del tacógrafo.
- En los camiones, el aceite se cambia a los 30.000km, las ruedas a los 50.000km y el tacógrafo se revisa cada 50.000 kilómetros.
- En los turismos, el aceite se cambia a los 15.000km y las ruedas a los 30.000km.
- El método *avanzar()* añade kilometros al motor y llama posteriormente a los métodos privados *checkAceite()* y *checkRuedas()*. Estos dos últimos métodos devuelve un booleano si ha habido que hacer un cambio. El método esta redefinido en la subclase Camion para contemplar el *checkTacografo()*



PROGRAMA - **Ejercicio05.java**
- Crea los vehículos anteriores (crea 4 objetos) y añádelos a una lista.
- Imprimir todos los vehículos indicando el tipo recorriendo la lista con un FOREACH.
- El vehículo matricula 1111FFF avanza 20.000km (cambio aceite)
- El vehículo 2222AAA avanza 30.000km (cambio aceite y ruedas)
- El vehículo 3333GGG avanza 20.000km (sin cambios)
- El vehículo 5555BBB avanza 60.000km (cambio aceite, ruedas y revisar tacógrafo)

Ejercicio 06 - Alquiler de coches (clases abstractas)

Hacer un programa que gestione los alquileres de un negocio de alquiler de furgonetas y turismos.



Los alquileres son creados por los vehículos, al llamar al método Vehiculo.alquilar(…..), al igual, para devolver un vehículo, llamaremos a Vehiculo.devolver(…….)

Se podía haber hecho de otro forma, que el main( ) creará los alquileres, pero así practicamos el this

Requisitos mínimos:

- Las furgonetas se alquilan por kilómetros. El precio del kilómetro es a 0.5€/km.
- Los coches se alquilan por días. El precio es de 30€/día.
- Almacenaré los vehículos de la empresa en una lista, así como los alquileres que se vayan generando. Estas dos listas estarán en una clase con métodos estáticos
- La opción de mostrar la flota de vehículos (aparecerá siempre antes del menú) mostrará primero de los turismos y luego de las furgonetas con sus datos. Al lado de cada vehículo aparecerá una letra «**A**» si está alquilado. Indicará los KM de cada uno.

```
========================================
      LISTADO DE TURISMOS
     1111TTT      Volvo XC60 [0 km]
     2222TTT         Audi A4 [0 km]

      LISTADO DE FURGONETAS
A    2222FFF     Citröen C16 [0 km]
     1111FFF   Mercedes VITO [10000 km]
========================================
```

- Método abstracto *alquila(…) y devolver(….)*. A estos métodos hay que pasarle siempre una fecha y unos kilómetros, dependiendo si es alquiler o devolución. A la hora de calcular el importe, usaremos unos datos u otros, pero pedimos ambos datos para que el alquiler contenga toda la información posible. ES POSIBLE QUE ESTOS MÉTODOS SEAN MUY PARECIDOS TANTO EN TURISMOS COMO EN FURGONETAS Y PODRÍAN NO SER ABSTRACTOS. Pero no pasa nada, los hacemos abstractos para practicar. Al devolver el vehículo, aparecerán los datos del alquiler

```
**********************************
1111-FFF         Mercedes Vito
Km realizados: 500km
Precio:0.5€/km
IMPORTE ALQUILER: 250.0€
**********************************
```

- Intentaremos pedir todos los datos en el main, en nuestra interfaz de entrada. Es por eso que a los métodos anteriores se le pasarán ya los datos recogidos en el main.
- Podemos hacer al pedir una fecha, que si pulsamos intro, se meta la fecha de hoy. Así nos ahorramos tiempo y es algo usual

```
TURISMO ENCONTRADO 1111TTT
   >Fecha de alquiler[dd/mm/aaaa] (INTRO PARA HOY):
18/05/2022
```

- El historico de alquileres mostrara linea a linea un resumen. Si un alquiler no está finalizado, lo indicará.

PROGRAMA - **Ejercicio06.java**

- Añade los 3 vehículos de la imagen anterior excepto el Mercedes Vito a la base de datos. Hazlo en código. Al añadirlos, todos van a tener 0 kilómetros.
- Añade con el menú la Mercedes Vito que tendrá 10.000km.
- Alquilar el 1111-TTT (VOLVO) en fecha 01-05-2022.
- Alquilar el 1111-FFF (Mercedes) hoy.
- Alquiler el 2222-TTT (Audi) hoy.
- Devolver el 1111-TTT (VOLVO). La fecha de devolución es 05-05-2022 y 1000km
- Devolver el 1111-FFF (Mercedes) hoy. En el marcador indica 10.500 km.
- Imprimir histórico

```
==========  HISTORICO DEL ALQUILERES  =======
   1111TTT - 2022-05-01 - 120,000000 €
   1111FFF - 2022-05-18 - 250,000000 €
   2222TTT - 2022-05-18 - NO FINALIZADO
=============================================
```

Ejercicio 06bis - Alquiler de coches (clases abstractas) con BBDD

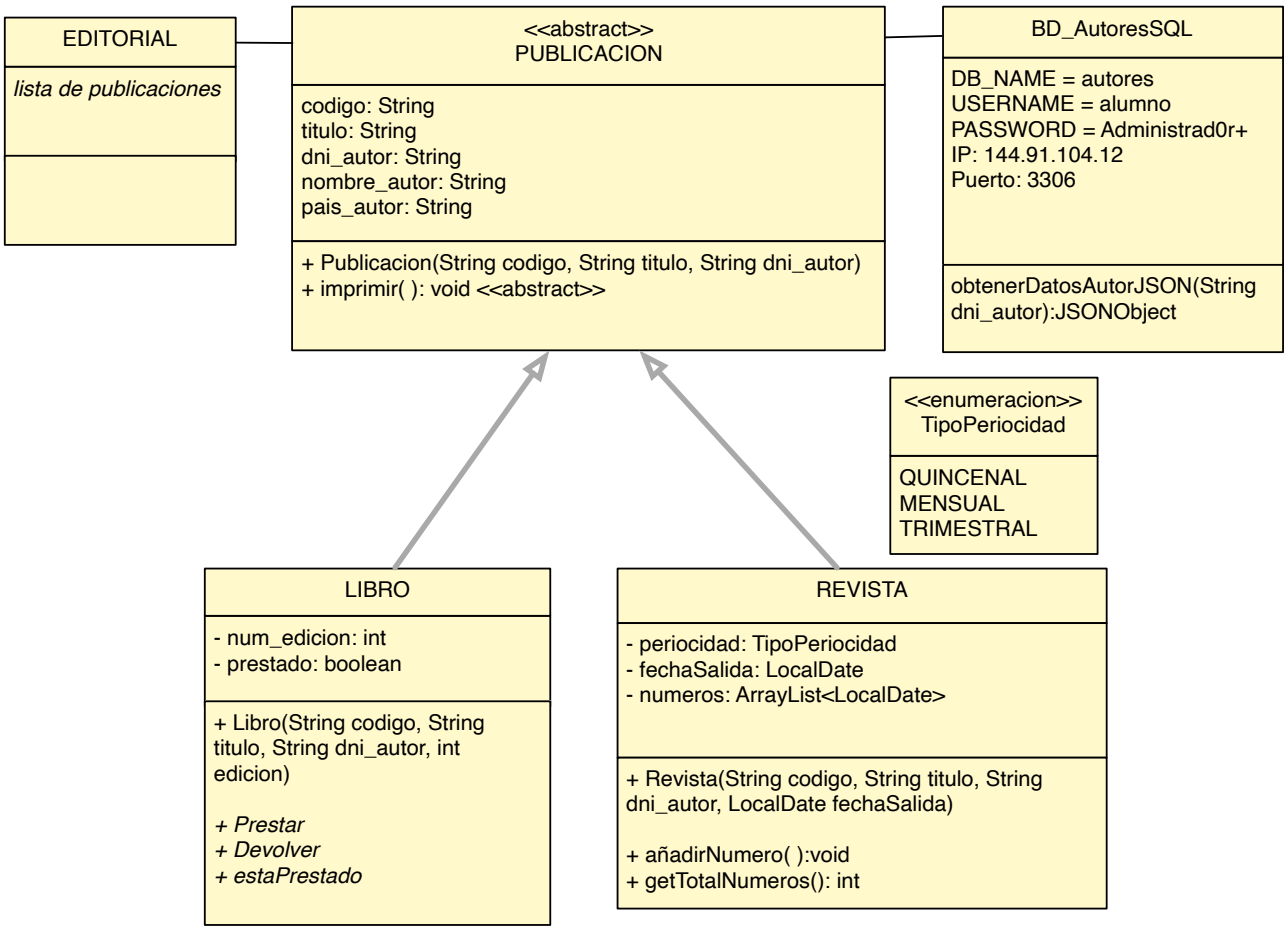
Modificar el ejercicio usando base de datos relacional para almacenar los datos del programa. No es necesario hacer todas las opciones del menú. Hacer solo las opciones de AÑADIR VEHICULO y de MOSTRAR ALQUILERES

Os proporciono el script con algunos datos ya metidos de vehículos y alquileres

```sql
DROP DATABASE IF EXISTS alquilervehiculos;
CREATE DATABASE alquilervehiculos CHARACTER SET utf8mb4;
USE alquilervehiculos;

CREATE TABLE vehiculo(
    matricula       varchar(50) NOT NULL,
    marca_modelo    varchar(50),
    km              int UNSIGNED DEFAULT 0,
    alquilado       boolean,
    precioKilometro DECIMAL(10,2),
    precioDia       DECIMAL(10,2),
    tipoVehiculo    varchar(50) NOT NULL,
    PRIMARY KEY (matricula)
);

CREATE TABLE alquiler(
    id                  int UNSIGNED NOT NULL AUTO_INCREMENT,
    matricula    varchar(50) NOT NULL,
    km_inicio   int UNSIGNED NOT NULL,
    km_fin   int UNSIGNED NOT NULL,
    fecha_inicio date,
    fecha_fin   date,
    importe DECIMAL(10,2),
    PRIMARY KEY (id),
    CONSTRAINT matricula_vehiculo FOREIGN KEY(matricula) REFERENCES vehiculo(matricula)
);

INSERT INTO vehiculo VALUES ('1111TTT','Volvo XC60',100,true,null,30,'turismo');
INSERT INTO vehiculo VALUES ('2222TTT','Audi A4',0,false,null,30,'turismo');
INSERT INTO vehiculo VALUES ('3333FFF','Citroen C16',250,false,0.5,null,'furgoneta');

INSERT INTO alquiler VALUES (null,'1111TTT',0,100,'2023-01-01','2023-01-05',120);
INSERT INTO alquiler VALUES (null,'3333FFF',0,250,'2023-01-01','2023-01-02',125);
INSERT INTO alquiler VALUES (null,'1111TTT',100,0,'2023-05-19',null,0);
```

- La clase BD_Empresa se llamará BDEmpresa_SQL, y mantendremos la clase Empresa cómo clase controlador con la que se comunica el main.
- Crearemos una tabla para los alquileres y tablas para los vehículos.
- Para transformar la jerarquía, crear una tabla solo con un campo *tipoVehiculo* y nulos. Puedes mirar el enlace https://www.youtube.com/watch?v=1C8hXHB1RcU&t=194s
- Usar JSON para mostrar el histórico de los alquileres

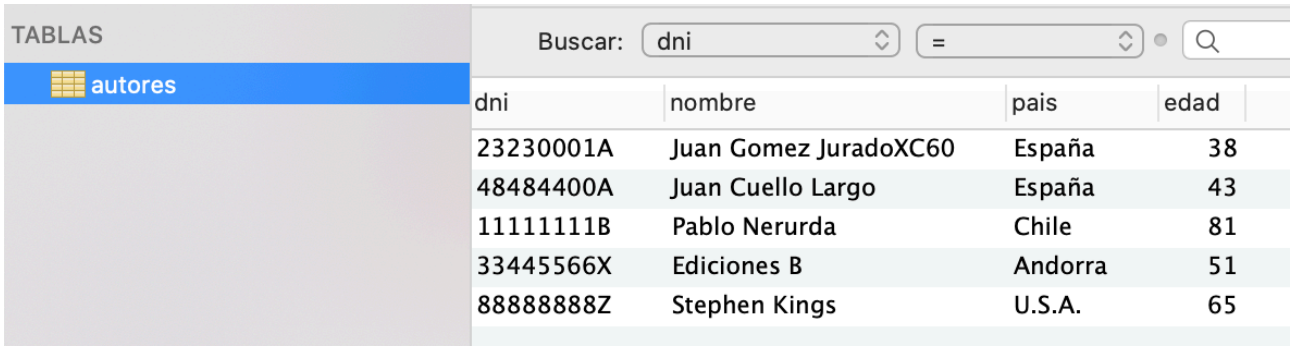
Ejercicio 07 - Editorial con Publicaciones




```
 EJERCICIO 7 - PUBLICACIONES
============================================================
LIBRO   [L-100]   EL RESPLANDOR 10ª edicion(DISPONIBLE)
LIBRO   [L-212] JAVA para todos  5ª edicion(PRESTADO)
REVISTA [ R-45]            HOLA  3 ejemplares - MENSUAL
REVISTA [ R-65]   TODO CONSOLAS  5 ejemplares - QUINCENAL
============================================================
1. Añadir libro
2. Añadir revista
3. Prestar/devolver libro
4. Sacar nuevo numero de revista
5. Detalles de publicacion
6. Salir
Elige una opcion >
```

LAS OPCIONES 1 Y 2 NO SON OPCIONALES. AÑADIREMOS EN CÓDIGO LOS LIBROS Y REVISTAS

La estructura de la BBDD es la siguiente. Los datos de conexión están en el UML



dni	nombre	pais	edad
23230001A	Juan Gomez JuradoXC60	España	38
48484400A	Juan Cuello Largo	España	43
11111111B	Pablo Nerurda	Chile	81
33445566X	Ediciones B	Andorra	51
88888888Z	Stephen Kings	U.S.A.	65

PUBLICACIONES

- Las publicaciones tendrán código, título, dni_autor, nombre_autor y pais_autor. En el constructor indicaremos código, título y dni_autor. El nombre_autor y pais_autor los sacaremos de una base de datos proporcionada.

LIBROS

- Los libros tendrán un código con formato L-XX
- Los libros se pueden prestar y devolver. Solo cambiamos el atributo.

REVISTAS

- Las revistas tendrán un código con formato R-XX y por defecto, serán de tipo MENSUAL.
- Una revista (del mismo título) esta formada por diversos números. Cada numero tendrá una fecha. La fecha de salida de la revista corresponde al primer número de la revista. Los diferentes números se guardarán en un array. Solo es necesario guardar la fecha de cada numero. El índice del array será el número de la revista.
- El método *añadirNumero( )* añadirá un nuevo numero al array, obteniendo la fecha de forma automática, mirando la periocidad de la revista. LocalDate tiene métodos para sumar días a una fecha.

```java
//*************** inicializacion de datos ******************************
Editorial editorialArcas = new Editorial();

Libro libro1=new Libro("L-100","EL RESPLANDOR","88888888Z",10);
editorialArcas.addPublicacion(libro1);

Libro libro2=new Libro("L-212","JAVA para todos","48484400A",5);
editorialArcas.addPublicacion(libro2);
libro2.prestar();

unaFecha=LocalDate.parse("01/01/2023", dtf);
Revista revista1=new Revista("R-45","HOLA","33445566X",unaFecha);
editorialArcas.addPublicacion(revista1);
revista1.añadirNumero();
revista1.añadirNumero();

unaFecha=LocalDate.parse("01/01/2023", dtf);
Revista revista2=new Revista("R-65","TODO CONSOLAS","33445566X",unaFecha);
revista2.setPeriocidad(TipoPeriocidad.QUINCENAL);
editorialArcas.addPublicacion(revista2);
revista2.añadirNumero();
revista2.añadirNumero();
revista2.añadirNumero();
revista2.añadirNumero();
//*************** fin inicializacion de datos ******************************
```

Aspecto de los detalles de un libro y una revista –> **método imprimir()**

```
Detalles del LIBRO
==================
Codigo:L-212
Titulo:JAVA para todos
Numero edicion:5
Nombre autor:Juan Cuello Largo
Pais autor:España
ESTADO: PRESTADO
```

```
Detalles de REVISTA
=========================
Codigo:R-45
Titulo:HOLA
Nombre autor:Ediciones B
Pais autor:Andorra
Periocidad: MENSUAL
Total numeros:3
  Numero 0 -->01/01/2023
  Numero 1 -->01/02/2023
  Numero 2 -->01/03/2023
```

Aspecto del toString() de libro y revista. Usado en el listado principal

```
LIBRO   [L-100]   EL RESPLANDOR 10ª edicion(DISPONIBLE)
LIBRO   [L-212] JAVA para todos  5ª edicion(PRESTADO)
REVISTA [ R-45]             HOLA  3 ejemplares - MENSUAL
REVISTA [ R-65]   TODO CONSOLAS  5 ejemplares - QUINCENAL
```