14. Ejercicios propuestos

EJERCICIO P1:

- Crear una clase Libro que contenga los siguientes atributos: ISBN, Titulo, Autor, Número de páginas
- Crear sus respectivos métodos get y set correspondientes para cada atributo. Crear el método toString() para mostrar la información relativa al libro con el siguiente formato:
- "El libro con ISBN creado por el autor tiene páginas"

En el fichero main, crear 2 objetos Libro (los valores que se quieran) y mostrarlos por pantalla. Por último, indicar cuál de los 2 tiene más páginas.

EJERCICIO P2: Algunos seres pueden caminar y pueden nadar estas dos capacidades son interfaces (PuedeCaminar.java y PuedeNadar.java):

- Otros animales también pueden volar (interface PuedeVolar.java).
- Los mamíferos son animales que pueden moverse (Mamifero.java).
- El gato es un mamifero que puede nadar y caminar (Gato.java).Las aves son animales que pueden caminar (Aves.java).
- Un Loro es un ave que puede caminar y volar pero no puede nadar.
- El Avestruz es un ave que solo camina.

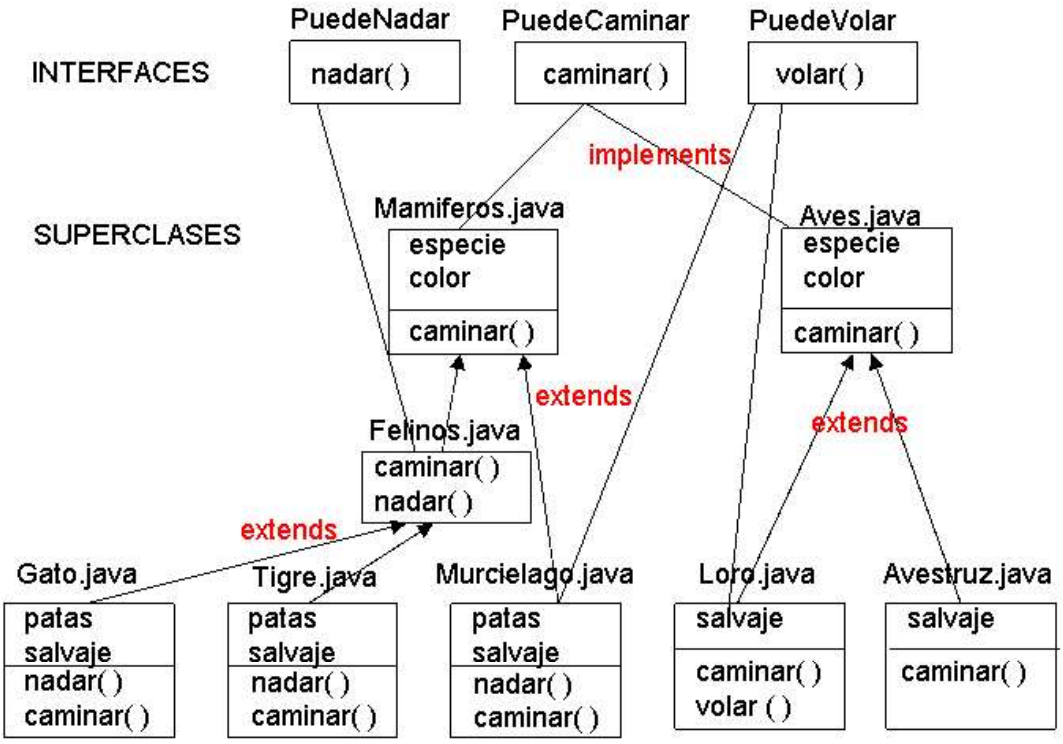
Crea las clases siguientes:

- Crea la clase Avestruz.java (ave que solo camina) hija de Aves.java. Crea la clase Loro.java (ave que camina y vuela) hija de Aves.java.
- Crea la clase Murciélago.java (mamífero que vuela y camina) hija de Mamifero.java Cree la clase Felino.java (mamífero que camina y puede nadar) hija de Mamifero.java. Cree la clase Gato.java (es un Felino)
- Cree la clase Tigre.java (es un Felino).

Crear un programa que cree varios seres (gato, loro, avestruz murciélago y tigre) y muestre qué es lo que puede hacer cada uno mediante la implementación de las tres interfaces anteriores y la creación de las superclases correspondientes (Mamífero, Aves y Felinos). Los métodos nadar(), caminar() y volar() simplemente mostrarán un mensaje en cada clase de animal que diga el animar que es y que puede hacer. P.E:

```
public void caminar(){
      System.out.println("El avestruz CAMINA");
}
```

El diagrama de clases del programa podría ser similar a este:



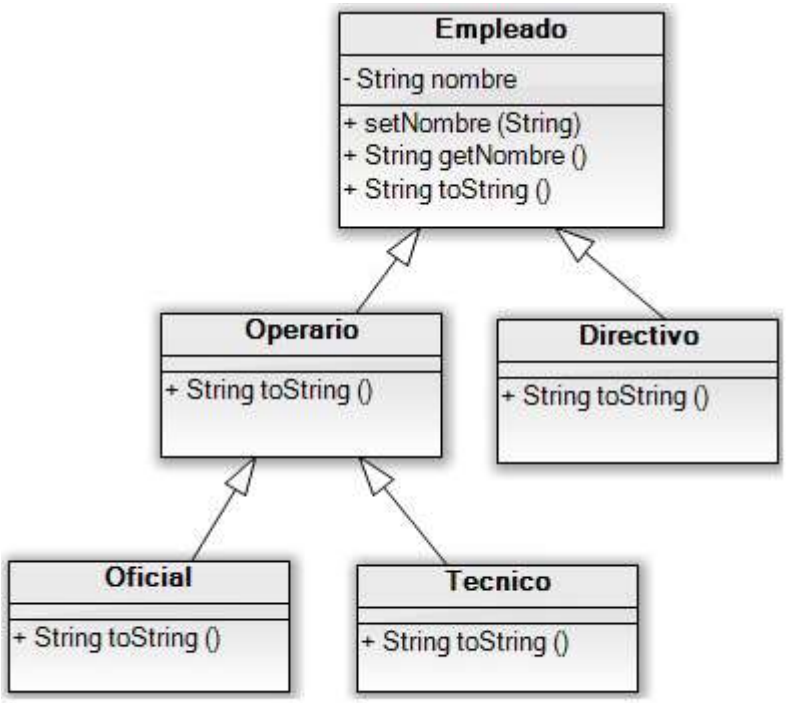
EJERCICIO P3:

Se plantea desarrollar un programa Java que permita representar la siguiente situación. Una instalación deportiva es un recinto delimitado donde se practican deportes, en Java interesa disponer de un método *int getTipoDeInstalacion().* Un edificio es una construcción cubierta y en Java interesa disponer de un método *double getSuperficieEdificio()*. Un polideportivo es al mismo tiempo una instalación deportiva y un edificio; en Java interesa conocer la superficie que tiene y el nombre que tiene. Un edificio de oficinas es un edificio; en Java interesa conocer el número de oficinas que tiene.

Definir dos interfaces y una clase que implemente ambas interfaces para representar la situación anterior. En una clase test con el método main, crear un ArrayList que contenga tres polideportivos y dos edificios de oficinas y utilizando un iterator, recorrer la colección y mostrar los atributos de cada elemento. ¿Entre qué clases existe una relación que se asemeja a la herencia múltiple?

EJERCICIO P4:

Codifica la siguiente jerarquía de clases representada por el diagrama UML siguiente:



La clase base es la clase Empleado. Esta clase contiene:

- Un atributo privado *nombre* de tipo String que heredan el resto de clases.
- Un constructor por defecto.
- Un constructor con parámetros que inicializa el nombre con el String que recibe.
- Método set y get para el atributo nombre.
- Un método toString() que devuelve el String: "Empleado " + nombre.

El resto de clases solo deben sobrescribir el método toString() en cada una de ellas y declarar el constructor adecuado de forma que cuando la ejecución de las siguientes instrucciones:

```
Empleado E1 = new Empleado("Rafa");
Directivo D1 = new Directivo("Mario");
Operario OP1 = new Operario("Alfonso");
Oficial OF1 = new Oficial("Luis");
Tecnico T1 = new Tecnico("Pablo");
System.out.println(E1);
System.out.println(D1);
System.out.println(OP1);
System.out.println(OF1);
System.out.println(T1);
```

Den como resultado:

```
Empleado Rafa
Empleado Mario -> Directivo
Empleado Alfonso -> Operario
Empleado Luis -> Operario -> Oficial
Empleado Pablo -> Operario -> Tecnico
```

EJERCICIO P5:

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()).

EJERCICIO P6 : Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2º grado.
Tendremos los 3 coeficientes como atributos, llamémosles a, b y c.
Hay que insertar estos 3 valores para construir el objeto.
Las operaciones que se podrán hacer son las siguientes:

- obtenerRaices(): imprime las 2 posibles soluciones
- obtenerRaiz(): imprime única raíz, que será cuando solo tenga una solución posible.
- getDiscriminante(): devuelve el valor del discriminante (double), el discriminante tiene la siguiente formula, (b^2)-4*a*c
- tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.
- tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.
- calcular(): mostrara por consola las posibles soluciones que tiene nuestra ecuación, en caso de no existir solución, mostrarlo también.

Formula ecuación 2º grado: (-b±√((b^2)-(4*a*c)))/(2*a)
Solo varia el signo delante de -b

EJERCICIO P7. Vamos a hacer el juego de la ruleta rusa en Java.

Como muchos sabéis, se trata de un número de jugadores que con un revolver con un sola bala en el tambor se dispara en la cabeza.

Las clases a hacer son:

- Revolver:
    - Atributos:
        - posición actual (posición del tambor donde se dispara, puede que esté la bala o no)
        - posición bala (la posición del tambor donde se encuentra la bala)

Estas dos posiciones, se generaran aleatoriamente.

- Funciones:
    - disparar(): devuelve true si la bala coincide con la posición actual
    - siguienteBala(): cambia a la siguiente posición del tambor
    - toString(): muestra información del revolver (posición actual y donde está la bala)

- Jugador:
    - Atributos
        - id (representa el número del jugador, empieza en 1)
        - nombre (Empezara con Jugador más su ID, “Jugador 1” por ejemplo)
        - vivo (indica si está vivo o no el jugador)
    - Funciones:
        - disparar(Revolver r): el jugador se apunta y se dispara, si la bala se dispara, el jugador muere.
- Juego:
    - Atributos:
        - Jugadores (conjunto de Jugadores)
        - Revolver
    - Funciones:
        - finJuego(): cuando un jugador muere, devuelve true
        - ronda(): cada jugador se apunta y se dispara, se informara del estado de la partida (El jugador se dispara, no ha muerto en esa ronda, etc.)

El número de jugadores será decidido por el usuario, pero debe ser entre 1 y 6. Si no está en este rango, por defecto será 6.

En cada turno uno de los jugadores, dispara el revólver, si este tiene la bala el jugador muere y el juego termina.

Recuerda usar una clase ejecutable para probarlo.