Práctica 5. Programación Orientada a Objetos Python.

El objetivo de esta práctica es que se implemente un programa en Python utilizando la programación orientada a objetos. Para ello analizaremos el fichero .py proporcionado por la profesora y contestaremos a las siguientes preguntas:

1. Prepara el proyecto con los ficheros fuentes y revisa el código que se proporciona con la práctica.
   a) Identifica cuantas clases Python hay en el fichero CocheHerencia.py. ¿Cómo se sabe dónde finaliza una clase si no se está finalizando con llave, paréntesis o punto y coma?
   b) Identifica los constructores de cada clase.
   c) Identifica los atributos de la clase **Coche**. ¿Qué particularidad tiene el atributo **contadorcoches** y para qué se utiliza?
   d) Como puedes ver en el constructor de **Coche**, no existe declaración clara de los atributos en Python. ¿Cómo sabe entonces de qué tipo es cada atributo?
   e) Revisa los métodos **acelerar** y **frenar** de la clase **Coche**. ¿Para qué crees que es la palabra reservada *self* que aparece en dichos métodos?
2. Para el punto de entrada en el programa puedes ver que se dispone un código. Identifícalo. Revisa el código fuente de la clase principal y ejecútalo, observando la salida que se produce por pantalla y emparejándolo con cada una de las instrucciones.
3. En Python puede definirse herencia entre clases. Observa la clase **Cochemarchas**.
   a) ¿Cómo se indica que una clase es padre de otra?
   b) ¿Qué puedes decir del constructor de la clase **Cochemarchas**?
   c) Instancia un objeto de la clase **Cochemarchas** y haga uso de los atributos propios y de los atributos y métodos heredados.
   d) ¿Cuándo creas el nuevo objeto, cómo afecta al atributo de la clase **contadorcoches**?
4. En la clase **Cochemarchas** hay cuatro métodos que deben ser definidos. Por el momento hemos utilizado la palabra reservada *pass* que no hace nada, y nos sirve para que el programa compile y no de errores.
   a) Desarrolla cada uno de ellos, utilizando los atributos y métodos que necesites, tanto de la clase padre como de la hija. Estos métodos deben actualizar los atributos que sean necesarios, y genera algún tipo de mensaje por pantalla de la operación que se está realizando, y debe tener en cuenta las siguientes restricciones que se deben validar:

- `cambiomarcha`:
    - No se puede pasar de marchas positivas a marcha atrás directamente. El coche debe estar parado y en punto muerto.
    - No debe permitirse marchas largas con velocidades muy bajas porque se cala el coche.
    - Puede permitirse marchas cortas con velocidades altas, pero debe emitir un mensaje de aviso de que el coche está revolucionado.
- `ponerpuntomuerto`:
    - No hay restricciones, pero debe emitir un aviso indicando que se encuentra en punto muerto.
- `ponerfremoano`:
    - No se puede poner el freno de mano con el coche en movimiento
- `quitarfrenomano`:
    - No tiene restricciones, pero debe emitir un aviso indicando que se ha quitado el freno de mano.

b) Utilice estos métodos en la función principal y pruebe su funcionamiento.

c) La clase **Coche** tiene un método **rugir** que sirve para imprimir por pantalla el ruido que haría el coche al acelerar. Especialice el método **rugir** en la clase **Cochemarchas**, para que haga un ruido distinto.

5. Implemente una clase **Cocheautomatico** que herede de la clase **Coche**.
    a) Esta clase debe tener una propiedad que sea modo, que indica el modo que se ha fijado el cambio automático. Este modo puede tomar los valores:
        - P – parking
        - R – marcha atrás (Reverse)
        - N – punto muerto (Neutral)
        - D – Conducción (Drive)

    b) Debe especializar el método rugir de la clase Coche.

    c) Utilice la clase **Cocheautomatico** en la función principal y pruebe su funcionamiento.

6. Desde el punto de vista de la encapsulación, como puedes comprobar todos los atributos de una clase (atributos de datos y métodos) son públicos. Esto quiere decir que desde un código que use la clase, se puede acceder a todos los atributos y métodos de dicha clase.

    Existe un modo de indicar que un atributo o un método interno es una clase y no se debería utilizar fuera de ella. Como vimos en la teoría era poniendo un _ delante del atributo. Con esta solución, el atributo seguirá siendo accesible desde fuera de la clase, pero se está señalando que es un atributo privado, y, por tanto, no debe utilizarse porque no se sabe qué consecuencias puede tener.

    a) Prueba a cambiar el nombre del atributo **color** por **_color**. Utiliza la opción **Refactor** del menú contextual del código que ofrece **PyCharm**. ¿Qué ocurre?

    b) Prueba también a cambiar el nombre del atributo color por **__color**. ¿Qué ocurre?

c) Añade los métodos get y set de los atributos de las clases. En Python, una de las convenciones de nomenclatura posibles para los métodos getter y setter es utilizar el prefijo “get_” o “set_” para los métodos getter y setter respectivamente. Adapta el código para que haga uso de estos métodos.

7. Como puedes comprobar al ejecutar la función principal, la línea de código *“prit(str(c1)”* genera una salida poco amigable. Esto se debe a que la clase Coche no tiene definido el método “_str_”.
   a) Define este método para las tres clases, de forma que las dos clases hijas deben hacer uso del método “_str_” de la clase padre mediante el método reservado **super ()**. Este método debe mostrar el estado de cada objeto, es decir, una cadena de texto del valor de sus atributos.
   b) Comprueba cómo cambia la salida y añade líneas de código que permitan comprobar la salida de cada clase.
8. En la función principal, crea una lista y añade varios objetos de las distintas clases de coches, y recorre la lista imprimiendo por pantalla el sonido de los distintos objetos. Deberíamos poder ver cómo se comporta el polimorfismo en Python, y cómo muestra por pantalla los distintos mensajes en función de la clase concreta de cada objeto recorrido.