**Ejercicio 1**

El código viola el Principio de Responsabilidad Única (SRP), tras revisar exhaustivamente el código se ha llegado a la conclusión de que este estaba realizando múltiples tareas en un mismo método, cosa que hacía difícil de leer el código y complicaba el entendimiento de cómo funcionaba. De cara a la refactorización se ha separado en varios métodos las diferentes funciones que la clase debe cumplir para poder alcanzar el resultado deseado.

**Ejercicio 2**

El código viola el Principio de Responsabilidad Única (SRP), una vez más al revisar el código se ha determinado que la clase *CarManager()* estaba realizando múltiples tareas, cosa que no debería de ser ya que complica el entendimiento al leer el código. Para la refactorización se ha separado la creación de la “base de datos” de coches y el formateo para mostrarlo en dos clases distintas.

* **CarDatabase:** Una clase con un array de coches como parámetro, con métodos para instanciar la DB y mostrar la información de un coche por ID o todo el listado de coches.
* **CarFormatter:** Una clase que solo cuenta con dos métodos. *formatCars($car)* con el cual se le pasa por parámetro un array de objetos “Car” y lo formatee para posteriormente mostrarlo por pantalla. Y el método *getBestCar($cars),* el cual selecciona el mejor coche del array de objetos “Car” en base a su posición alfabética (A como menor, Z como mayor).

**Ejercicio 3**

El código viola el Principio Open/Closed (OCP), ya que la clase *AreaCalculator()* es estrictamente sólida para calcular el Area de un Circulo o un Cuadrado, y en caso de tener que añadir una nueva figura habría que modificarla para poder hacer dicho calculo. Es por ello que en la refactorización en vez de usar clases abstractas para hacer las formas, utilizamos una interfaz *“Shape()”* con el método para calcular el área de la forma, obtener el tipo de forma y dibujar la forma. De esta manera, la clase *“AreaCalculator()”* puede calcular el área de cada forma sin problemas.

**Ejercicio 4**

El código viola el Principio de Sustitución de Liskov (LSP), ya que la clase cuadrado hereda de la clase rectángulo, forzando a que el ancho y el alto sean iguales, cuando deberían actuar como en Rectángulo al tratarse de una herencia. Esto se soluciona en la refactorización al hacer a Cuadrado y Rectángulo clases independientes con sus propiedades y métodos únicos.

**Ejercicio 5**

El código no viola ninguno de los Principios SOLID, sin embargo, la clase “*Pool()”* se encuentra bastante cargada de responsabilidades por lo que de seguir creciendo, podría llegar a violar el Principio de Responsabilidad Única (SRP). Es por ello por lo que en la refactorización lo que se ha hecho ha sido separar la gestión de patos con la piscina en sí, de manera que, en caso de ampliarse el programa, este sea más escalable a futuro.

**Ejercicio 6**

**Ejercicio 7**

**Ejercicio 8**

**Ejercicio 9**