Profesores: *Neiner Maximiliano, Villegas Octavio*

Parte 3 - Ejercicios con Funciones

Aplicación Nº 11 (Potencias de números)
Mostrar por pantalla las primeras 4 potencias de los números del uno 1 al 4 (hacer una función que las calcule invocando la función *pow*).

Aplicación Nº 12 (Invertir palabra)
Realizar el desarrollo de una función que reciba un Array de caracteres y que invierta el orden de las letras del Array.
Ejemplo: Se recibe la palabra "HOLA" y luego queda "ALOH".

Aplicación Nº 13 (Invertir palabra)
Crear una función que reciba como parámetro un string (***$palabra***) y un entero (***$max***). La función validará que la cantidad de caracteres que tiene *$palabra* no supere a *$max* y además deberá determinar si ese valor se encuentra dentro del siguiente listado de palabras válidas: "Recuperatorio", "Parcial" y "Programacion". Los valores de retorno serán:
1 si la palabra pertenece a algún elemento del listado.
0 en caso contrario.

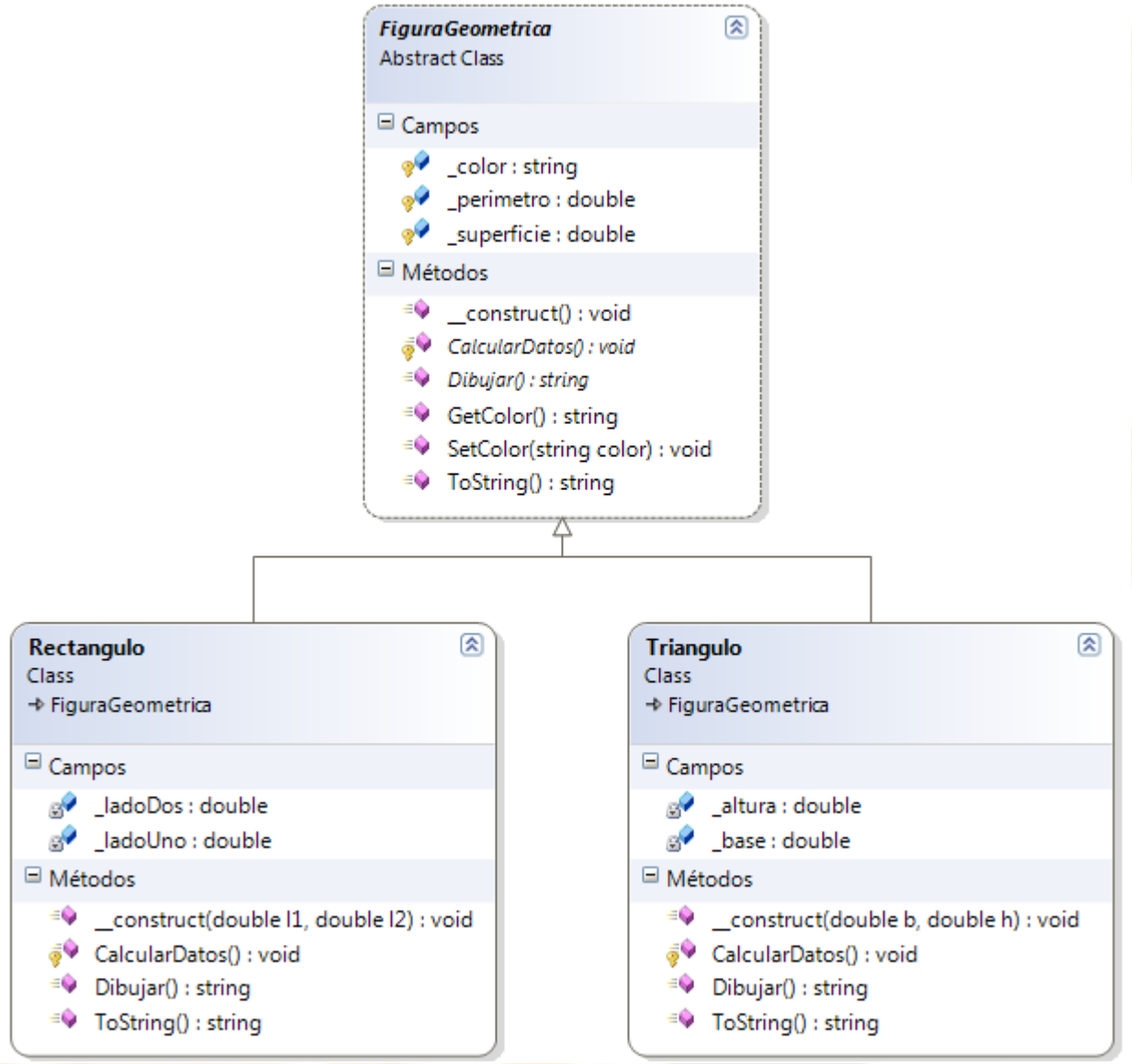
Aplicación Nº 14 (Par e impar)
Crear una función llamada **esPar** que reciba un valor entero como parámetro y devuelva *TRUE* si este número es par ó *FALSE* si es impar.
Reutilizando el código anterior, crear la función **esImpar**.

Parte 4 - Ejercicios con POO

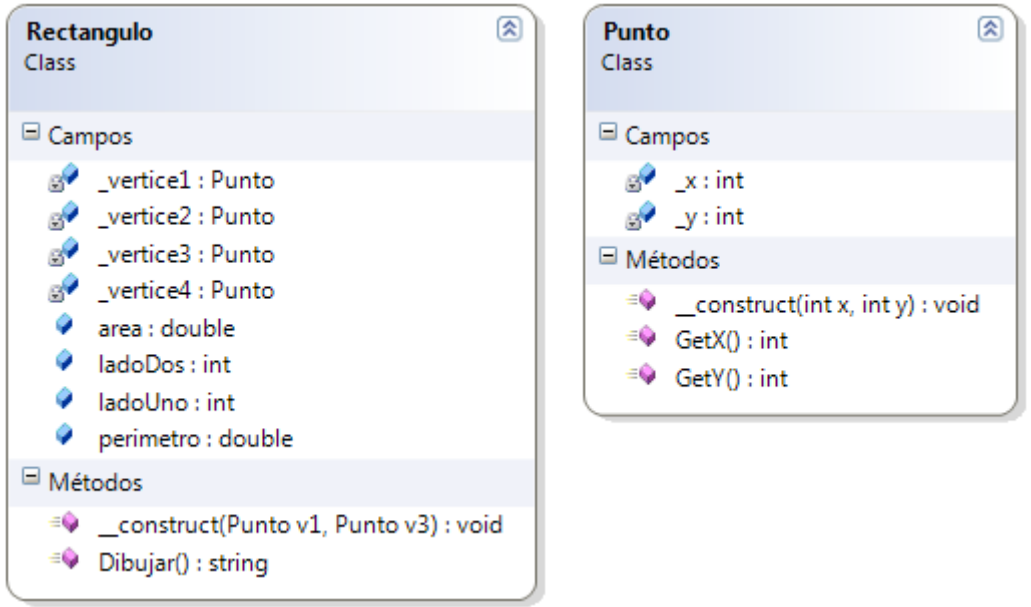
Aplicación Nº 15 (Figuras geométricas)
La clase ***FiguraGeometrica*** posee: todos sus atributos protegidos, un constructor por defecto, un método getter y setter para el atributo ***_color***, un método virtual (***ToString***) y dos métodos abstractos: ***Dibujar*** (público) y ***CalcularDatos*** (protegido).
CalcularDatos será invocado en el constructor de la clase derivada que corresponda, su funcionalidad será la de inicializar los atributos _superficie y _perimetro.
Dibujar, retornará un string (con el color que corresponda) formando la figura geométrica del objeto que lo invoque (retornar una serie de asteriscos que modele el objeto).
Ejemplo:

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

Utilizar el método ToString para obtener toda la información completa del objeto, y luego dibujarlo por pantalla.
Jerarquía de clases:



Aplicación Nº 16 (Rectangulo - Punto)
Codificar las clases **Punto** y **Rectangulo**.



La clase **Punto** ha de tener dos atributos privados con acceso de sólo lectura (sólo con **getters**), que serán las coordenadas del punto. Su constructor recibirá las coordenadas del punto.
La clase **Rectangulo** tiene los atributos privados de tipo **Punto** *_vertice1*, *_vertice2*, *_vertice3* y *_vertice4* (que corresponden a los cuatro vértices del rectángulo).
La base de todos los rectángulos de esta clase será siempre horizontal. Por lo tanto, debe tener un constructor para construir el rectángulo por medio de los vértices 1 y 3.
Los atributos *ladoUno*, *ladoDos*, *área* y *perímetro* se deberán inicializar una vez construido el rectángulo.
Desarrollar una aplicación que muestre todos los datos del rectángulo y lo dibuje en la página.

Aplicación Nº 17 (Auto)
Realizar una clase llamada **"Auto"** que posea los siguientes atributos **privados**:

_color (String)
_precio (Double)
_marca (String).
_fecha (DateTime)

Realizar un constructor capaz de poder instanciar objetos pasándole como parámetros:

i. La marca y el color.
ii. La marca, color y el precio.
iii. La marca, color, precio y fecha.

Realizar un método de **instancia** llamado "**AgregarImpuestos",** que recibirá un doble por parámetro y que se sumará al precio del objeto.
Realizar un método de **clase** llamado **"MostrarAuto",** que recibirá un objeto de tipo **"Auto"** por parámetro y que mostrará todos los atributos de dicho objeto.
Crear el método de instancia **"Equals"** que permita comparar dos objetos de tipo **"Auto"**. Sólo devolverá *TRUE* si ambos **"Autos"** son de la misma marca.

Crear un método de clase, llamado **"Add"** que permita sumar dos objetos **"Auto"** (sólo si son de la misma marca, y del mismo color, de lo contrario informarlo) y que retorne un **Double** con la suma de los precios o cero si no se pudo realizar la operación.
Ejemplo: *$importeDouble = Auto::Add($autoUno, $autoDos);*
En *testAuto.php*:

- Crear **dos** objetos **"Auto"** de la misma marca y distinto color.
- Crear **dos** objetos **"Auto"** de la misma marca, mismo color y distinto precio.
- Crear **un** objeto **"Auto"** utilizando la sobrecarga restante.
- Utilizar el método **"AgregarImpuesto"** en los últimos tres objetos, agregando $ 1500 al atributo precio.
- Obtener el importe sumado del primer objeto **"Auto"** más el segundo y mostrar el resultado obtenido.
- Comparar el primer **"Auto"** con el segundo y quinto objeto e informar si son iguales o no.
- Utilizar el método de clase "**MostrarAuto**" para mostrar cada los objetos impares (1, 3, 5)

Aplicación Nº 18 (Auto - Garage)
Crear la clase **Garage** que posea como atributos privados:
_razonSocial (String)
_precioPorHora (Double)
_autos (Autos[], reutilizar la clase Auto del ejercicio anterior)
Realizar un constructor capaz de poder instanciar objetos pasándole como parámetros:
i. La razón social.
ii. La razón social, y el precio por hora.
Realizar un método de **instancia** llamado **"MostrarGarage",** que no recibirá parámetros y que mostrará todos los atributos del objeto.
Crear el método de instancia **"Equals"** que permita comparar al objeto de tipo **Garaje** con un objeto de tipo **Auto.** Sólo devolverá *TRUE* si el auto está en el garaje.
Crear el método de instancia **"Add"** para que permita sumar un objeto **"Auto"** al **"Garage"** (sólo si el auto **no** está en el garaje, de lo contrario informarlo).
Ejemplo: *$miGarage->Add($autoUno);*
Crear el método de instancia **"Remove"** para que permita quitar un objeto **"Auto"** del **"Garage"** (sólo si el auto **está** en el garaje, de lo contrario informarlo).
Ejemplo: *$miGarage->Remove($autoUno);*
En *testGarage.php*, crear autos y un garage. Probar el buen funcionamiento de todos los métodos.

Aplicación Nº 19 (Pasajero - Vuelo)
Dadas las siguientes clases:
Pasajero

Atributos **privados**: _apellido (string), _nombre (string), _dni (string), _esPlus (boolean)
Crear un constructor capaz de recibir los cuatro parámetros.
Crear el método de instancia **"Equals"** que permita comparar dos objetos Pasajero. Retornará *TRUE* cuando los _dni sean iguales.
Agregar un método getter llamado *GetInfoPasajero*, que retornará una cadena de caracteres con los atributos concatenados del objeto.

Agregar un método de clase llamado *MostrarPasajero* que mostrará los atributos en la página.

Vuelo

Atributos **privados**: _fecha (DateTime), _empresa (string) _precio (double), _listaDePasajeros (array de tipo Pasajero), _cantMaxima (int; con su getter). Tanto _listaDePasajero como _cantMaxima sólo se inicializarán en el constructor.

Crear el constructor capaz de que de poder instanciar objetos pasándole como parámetros:

i. La empresa y el precio.
ii. La empresa, el precio y la cantidad máxima de pasajeros.

Agregar un método getter, que devuelva en una cadena de caracteres toda la información de un vuelo: fecha, empresa, precio, cantidad máxima de pasajeros, y toda la información de **todos** los pasajeros.

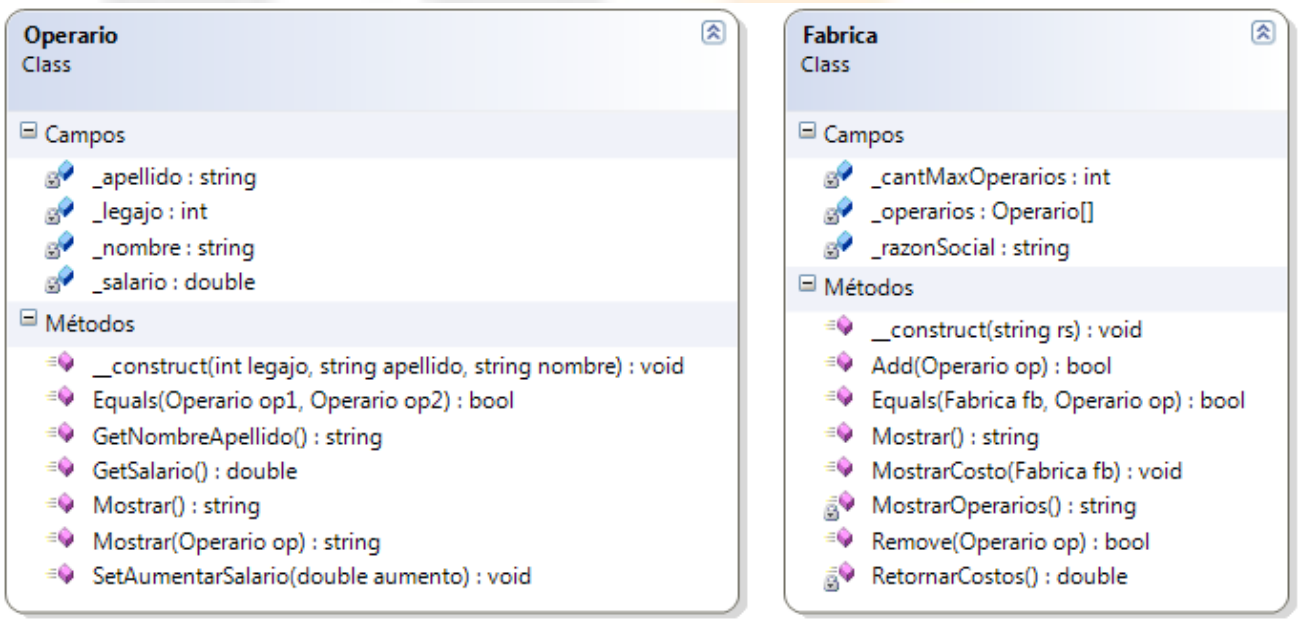
Crear un método de **instancia** llamado AgregarPasajero, en el caso que no exista en la lista, se agregará (utilizar Equals). Además tener en cuenta la capacidad del vuelo. El valor de retorno de este método indicará si se agregó o no.

Agregar un método de instancia llamado MostrarVuelo, que mostrará la información de un vuelo.

Crear el método de clase **"Add"** para que permita sumar dos vuelos. El valor devuelto deberá ser de tipo numérico, y representará el valor recaudado por los vuelos. Tener en cuenta que si un pasajero es **Plus**, se le hará un descuento del 20% en el precio del vuelo.

Crear el método de clase **"Remove"**, que permite quitar un pasajero de un vuelo, siempre y cuando el pasajero esté en dicho vuelo, caso contrario, informarlo. El método retornará un objeto de tipo Vuelo.

Aplicación Nº 20 (Operario - Fabrica)



Métodos getters y setters (en ***Operario***):

GetSalario: Sólo retorna el salario del operario.

SetAumentarSalario: Sólo permite asignar un nuevo salario al operario. La asignación consiste en incrementar el salario de acuerdo al porcentaje que recibe como parámetro.
Constructores: realizar los constructores para cada clase (**Fabrica** y **Operario**) con los parámetros que se detallan en la imagen.
En la clase **Fabrica**, la cantidad máxima de operarios será inicializada en 5.
Métodos (en ***Operario***)
GetNombreApellido (de instancia): Retorna un ***String*** que tiene concatenado el nombre y el apellido del operario separado por una coma.
Mostrar (de instancia): Retorna un ***String*** con toda la información del operario. Utilizar el método *GetNombreApellido*.
Mostrar (de clase): Recibe un ***operario*** y retorna un ***String*** con toda la información del mismo (utilizar el método *Mostrar* de instancia)
Crear el método de instancia **"Equals"** que permita comparar al objeto actual con otro de tipo Operario. Retronará un booleano informando si el nombre, apellido y el legajo de los operarios coinciden al mismo tiempo.
Métodos (en ***Fabrica***)
RetornarCostos (de instancia, privado): Retorna el dinero que la fábrica tiene que gastar en concepto de salario de todos sus operarios.
MostrarOperarios (de instancia, privado): Recorre el Array de operarios de la fábrica y muestra el nombre, apellido y el salario de cada operario (utilizar el método *Mostrar* de operario).
MostrarCosto (de clase): muestra la cantidad total del costo de la fábrica en concepto de salarios (utilizar el método *RetornarCostos*).
Crear el método de clase **"Equals"**, recibe una Fabrica y un Operario. Retronará un booleano informando si el operario se encuentra en la fábrica o no. Reutilizar código.
Add (de instancia): Agrega un operario al Array de tipo ***Operario***, siempre y cuando haya lugar disponible en la fábrica y el operario no se encuentre ya ingresado. Reutilizar código. Retorna TRUE si pudo ingresar al operario, FALSE, caso contrario.
Remove (de instancia): Recibe a un objeto de tipo Operario y lo saca de la fábrica, siempre y cuando el operario se encuentre en el Array de tipo Operario. Retorna TRUE si pudo quitar al operario, FALSE, caso contrario.
Crear los objetos necesarios en *testFabrica.php* como para probar el buen funcionamiento de las clases.