Universidad Tecnológica Nacional
Facultad Regional Avellaneda

Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos

Guía de Ejercicios POO 2017

Antes de empezar lea lo siguiente:
Para los ejercicios de esta guía, deberá asignarle al atributo 'Title' de la clase Console, el número de ejercicio, por ejemplo ***Console.Title = "Ejercicio Nro ##"***, donde ## será el número del ejercicio.
Del mismo modo se deberán nombrar las clases que contengan al método ***Main***, por ejemplo, ***Class Ejercicio_##***.
Para visualizar los valores decimales de los ejercicios, Ud. deberá dar el siguiente formato de salida al método Write/WriteLine: ***"#,###.00"***.

CONCEPTOS BASICOS

1. Ingresar 5 números por consola, guardándolos en una variable escalar. Luego calcular y mostrar: el valor máximo, el valor mínimo y el promedio.

2. Ingresar un número y mostrar: el cuadrado y el cubo del mismo. Se debe validar que el número sea mayor que cero, caso contrario, mostrar el mensaje: ***"ERROR. ¡Reingresar número!"***.
   Nota: *Utilizar el método '**Pow**' de la clase **Math*** para realizar la operación.

3. Mostrar por pantalla todos los **números primos** que haya hasta el número que ingrese el usuario por consola.
   Nota: *Utilizar estructuras repetitivas, selectivas y la función módulo (%).*

4. Un **número perfecto** es un entero positivo, que es igual a la suma de todos los enteros positivos (excluido el mismo) que son divisores del número.
   El primer número perfecto es 6, ya que los divisores de 6 son 1, 2 y 3; y 1 + 2 + 3 = 6.
   Escribir una aplicación que encuentre los 4 primeros números perfectos.
   Nota: *Utilizar estructuras repetitivas, selectivas y la función módulo (%).*

5. Un **centro numérico** es un número que separa una lista de números enteros (comenzando en 1) en dos grupos de números, cuyas sumas son iguales.
   El primer centro numérico es el 6, el cual separa la lista (1 a 8) en los grupos: (1; 2; 3; 4; 5) y (7; 8) cuyas sumas son ambas iguales a 15. El segundo centro numérico es el 35, el cual separa la lista (1 a 49) en los grupos: (1 a 34) y (36 a 49) cuyas sumas son ambas iguales a 595.
   Se pide elaborar una aplicación que calcule los centros numéricos entre 1 y el número que el usuario ingrese por consola.
   Nota: *Utilizar estructuras repetitivas, selectivas y la función módulo (%).*

6. Escribir un programa que determine si un año es bisiesto.
   Un año es bisiesto si es múltiplo de 4. Los años múltiplos de 100 no son bisiestos, salvo si ellos también son múltiplos de 400. Por ejemplo: el año 2000 es bisiesto pero 1900 no.
   Nota: *Utilizar estructuras repetitivas, selectivas y la función módulo (%).*

7. Hacer un programa que pida por pantalla la fecha de nacimiento de una persona (día, mes y año) y calcule el número de días vividos por esa persona hasta la fecha actual (tomar la fecha del sistema

con ***DateTime.Now***).
Nota*: Utilizar estructuras selectivas. Tener en cuenta los años bisiestos.*

8. Por teclado se ingresa el valor hora, el nombre, la antigüedad (en años) y la cantidad de horas trabajadas en el mes de ‗n' empleados de una fábrica.
   Se pide calcular el importe a cobrar teniendo en cuenta que el total (que resulta de multiplicar el valor hora por la cantidad de horas trabajadas), hay que sumarle la cantidad de años trabajados multiplicados por $ 150, y al total de todas esas operaciones restarle el 13% en concepto de descuentos.
   Mostrar el recibo correspondiente con el nombre, la antigüedad, el valor hora, el total a cobrar en bruto, el total de descuentos y el valor neto a cobrar de todos los empleados ingresados.
   Nota: *Utilizar estructuras repetitivas y selectivas.*

9. Escribir un programa que imprima por pantalla una pirámide como la siguiente:

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

   El usuario indicará cuál será la altura de la pirámide ingresando un número entero positivo. Para el ejemplo anterior la altura ingresada fue de 5.
   Nota: *Utilizar estructuras repetitivas y selectivas.*

10. Partiendo de la base del ejercicio anterior, se pide realizar un programa que imprima por pantalla una pirámide como la siguiente:

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

    Nota: *Utilizar estructuras repetitivas y selectivas.*

METODOS ESTATICOS (DE CLASE)

11. Ingresar 10 números enteros que pueden estar dentro de un rango de entre -100 y 100.
    Para ello realizar una clase llamada **Validacion** que posea un método estático llamado **Validar**, que posea la siguiente firma: ***bool Validar(int valor, int min, int max)***:
    a. valor: dato a validar
    b. min y max: rango en el cual deberá estar la variable valor.

    Terminado el ingreso mostrar el valor mínimo, el valor máximo y el promedio.
    Nota: *Utilizar variables escalares, NO utilizar vectores.*

12. Realizar un programa que sume números enteros hasta que el usuario lo determine, por medio de un mensaje "**¿Continuar? (S/N)**".
    En el método estático ***ValidaS_N(char c)*** de la clase **ValidarRespuesta**, se validará el ingreso de opciones.
    El método devolverá un valor de tipo booleano, *TRUE* si se ingresó una 'S' y *FALSE* si se ingresó cualquier otro valor.

13. Desarrollar una clase llamada **Conversor**, que posea dos métodos de clase (**estáticos**):
string DecimalBinario(double). Convierte un número de decimal a binario.
double BinarioDecimal(string). Convierte un número binario a decimal.

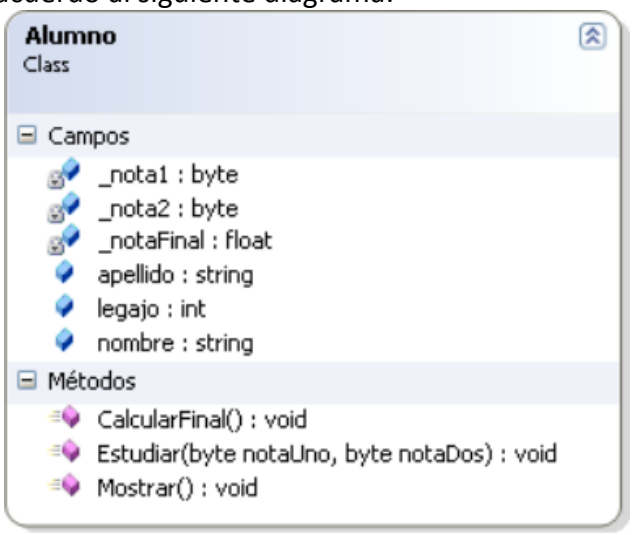
14. Realizar una clase llamada **CalculoDeArea** que posea 3 métodos de clase (**estáticos**) que realicen el cálculo del área que corresponda:
    a. ***double CalcularCuadrado(double)***
    b. ***double CalcularTriangulo(double, double)***
    c. ***double CalcularCirculo(double)***

    El ingreso de los datos como la visualización se deberán realizar desde el método Main().

15. Realizar un programa que permita realizar operaciones matemáticas simples (suma, resta, multiplicación y división). Para ello se le debe pedir al usuario que ingrese dos números y la operación que desea realizar (pulsando el caracter +, -, * ó /).
    El usuario decidirá cuándo finalizar el programa.
    Crear una clase llamada **Calculadora** que posea tres métodos estáticos (de clase):
    a. **Calcular** (público): Recibirá tres parámetros, el primer número, el segundo número y la operación matemática. El método devolverá el resultado de la operación.
    b. **Validar** (privado): Recibirá como parámetro el segundo número. Este método se debe utilizar sólo cuando la operación elegida sea la DIVISIÓN. Este método devolverá TRUE si el número es distinto de CERO.
    c. **Mostrar** (público): Este método recibe como parámetro el resultado de la operación y lo muestra por pantalla. No posee valor de retorno.

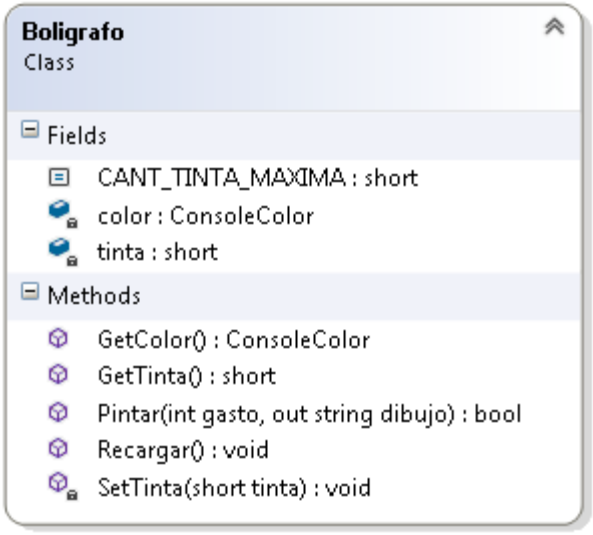
OBJETOS

16. Crear la clase Alumno de acuerdo al siguiente diagrama:



    a. Se pide crear 3 instancias de la clase Alumno (3 objetos) en la función Main. Colocarle nombre, apellido y legajo a cada uno de ellos.
    b. Sólo se podrá ingresar las notas (nota1 y nota2) a través del método **Estudiar**.
    c. El método **CalcularFinal** deberá colocar la nota del final sólo si las notas 1 y 2 son mayores o iguales a 4, caso contrario la inicializará con -1. Para darle un valor a la nota final utilice el método de instancia **Next** de la clase **Random**.
    d. El método Mostrar, expondrá en la consola todos los datos de los alumnos. La nota final se mostrará sólo si es distinto de -1, caso contrario se mostrará la leyenda "Alumno desaprobado".

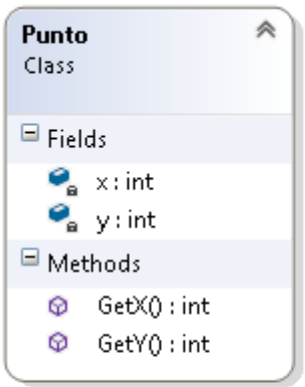
17. Crear la clase **Bolígrafo** a partir del siguiente diagrama:

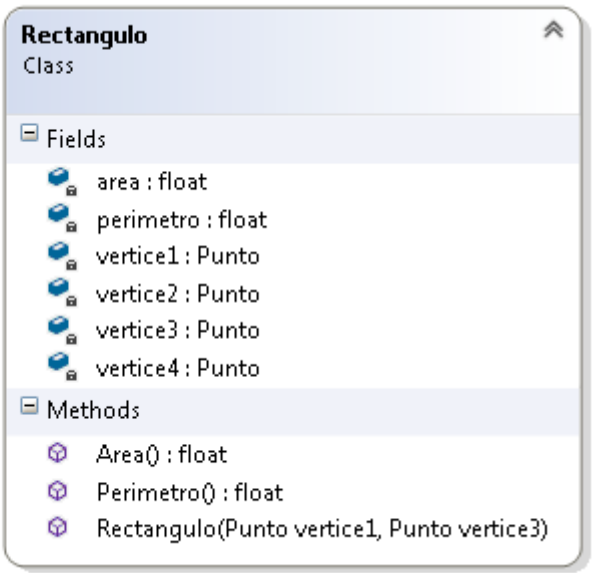


a. La cantidad máxima de tinta para todos los bolígrafos será de 100. Generar una constante estática en el Boligrafo llamada CANT_TINTA_MAXIMA donde se guardará dicho valor.
b. Generar los métodos ***GetColor y GetTinta*** para los correspondientes atributos (sólo retornarán el valor del mismo).
c. Generar el método privado SetTinta que valide el nivel de tinta y asigne en el atributo.
d. Recargar() colocará a tinta en su nivel máximo de tinta. Reutilizar código.
e. En el Main, crear un bolígrafo de tinta azul (***ConsoleColor.Blue***) y una cantidad inicial de tinta de 100 y otro de tinta roja (***ConsoleColor.Red***) y 50 de tinta.
f. El método Pintar(int, out string) restará la tinta gastada, sin poder quedar el nivel en negativo, avisando si pudo pintar (nivel de tinta mayor a 0). También informará mediante el out string tantos "*" como haya podido "gastar" del nivel de tinta. O sea, si nivel de tinta es 10 y gasto es 2 valdrá "**" y si nivel de tinta es 3 y gasto es 10 "***".
g. Utilizar todos los métodos en el Main.
h. Al utilizar Pintar, si corresponde, se deberá dibujar por pantalla con el color de dicho bolígrafo.

Nota: *Crear el constructor que crea conveniente. La clase Boligrafo y el Program deben estar en namespaces distintos.*

18. Escribir una aplicación con estos dos espacios de nombres (namespaces): **Geometría** y **PruebaGeometria**.
Dentro del espacio de nombres *Geometría* se deberán escribir dos clases: **Punto** y **Rectangulo**.





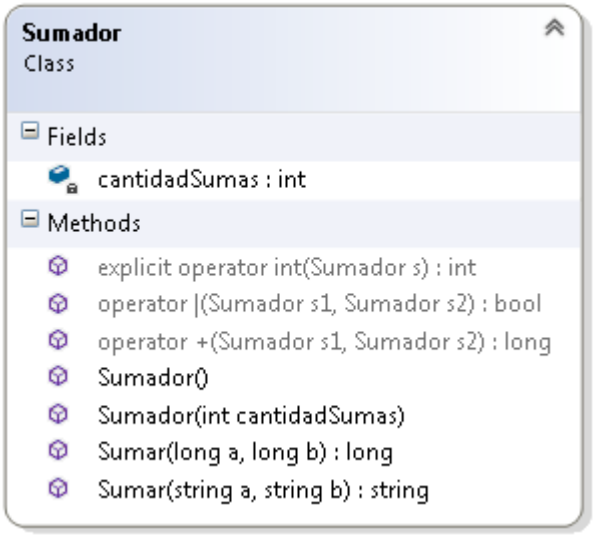
a. La clase **Punto** ha de tener dos atributos ***privados*** con acceso de sólo lectura (sólo con ***getters***), que serán las coordenadas del punto.
b. La clase **Rectangulo** tiene los atributos de tipo **Punto** vertice1, vertice2, vertice3 y vertice4 (que corresponden a los cuatro vértices del rectángulo).
c. 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 (utilizar el método **Abs** de la clase **Math**, dicho método retorna el valor absoluto de un número, para obtener la distancia entre puntos).
d. Realizar los métodos *getters* para los atributos privados lado, área y perímetro.
e. Los atributos área (*base * altura*) y perímetro (*(base + altura)*2*) se deberán calcular sólo una vez, al llamar por primera vez a su correspondiente método *getter*. Luego se deberá retornar siempre el mismo valor.

En el espacio de nombres **PruebaGeometria** es donde se escribirá una clase con el método **Main**.

f. Probar las funcionalidades de las clases Punto y Rectangulo.
    i. Generar un nuevo Rectangulo.
    ii. Imprimir por pantalla los valores de área y perímetro.
g. Desarrollar un método de **clase** que muestre todos los datos del rectángulo que recibe como parámetro.

SOBRECARGA DE OPERADORES

19. Realizar una aplicación de consola. Agregar la clase Sumador.



a. Crear dos constructores:
    i. Sumador(int) inicializa cantidadSumas en el valor recibido por parámetros.
    ii. Sumador() inicializa cantidadSumas en 0. Reutilizará al primer constructor.
b. El método Sumar incrementará cantidadSumas en 1 y adicionará sus parámetros con la siguiente lógica:
    i. En el caso de Sumar(long, long) sumará los valores numéricos
    ii. En el de Sumar(string, string) concatenará las cadenas de texto.

Antes de continuar, agregar un objeto del tipo Sumador en el Main y probar el código.

c. Generar una conversión explícita que retorne cantidadSumas.
d. Sobrecargar el operador + (suma) para que puedan sumar cantidadSumas y retornen un long con dicho valor.
e. Sobrecargar el operador | (pipe) para que retorne True si ambos sumadores tienen igual cantidadSumas.

Agregar un segundo objeto del tipo Sumador en el Main y probar el código.

20. Construir tres clases: Pesos, **Euro** y **Dolar**. Se debe lograr que los objetos de estas clases se puedan sumar, restar, comparar, incrementar y disminuir con total normalidad como si fueran tipos numéricos, teniendo presente que 1 Euro equivale a 1,3642 dólares y 1 dólar equivale a 17,55 pesos.
    Además, tienen que ser compatibles entre sí y también con el tipo ***Double***. Sobrecargar los operadores **explicit** y/o **implicit**.
    Nota: *Las clases y el Program deben estar en namespaces distintos.*

21. Crear tres clases: **Fahrenheit**, **Celsius** y **Kelvin**. Realizar un ejercicio similar al anterior, teniendo en cuenta que:
    F = C * (9/5) + 32
    C = (F-32) * 5/9
    F = K * 9/5 – 459.67
    K = (F + 459.67) * 5/9

22. Tomando la clase **Conversor** del ejercicio 13, se pide:
    Agregar las clases:
    a. ***NumeroBinario***:
        i. único atributo número (string)
        ii. único constructor privado (recibe un parámetro de tipo string)
    b. ***NumeroDecimal***
        i. único atributo número (double)
        ii. único constructor privado (recibe un parámetro de tipo double)

    Utilizando los métodos de la clase Conversor donde corresponda, agregar las sobrecargas de operadores:
    c. NumeroBinario:
        i. string + (NumeroBinario, NumeroDecimal)
        ii. string - (NumeroBinario, NumeroDecimal)
        iii. bool == (NumeroBinario, NumeroDecimal)
        iv. bool != (NumeroBinario, NumeroDecimal)
    d. NumeroDecimal:
        i. double + (NumeroDecimal, NumeroBinario)
        ii. double - (NumeroDecimal, NumeroBinario)
        iii. bool == (NumeroDecimal, NumeroBinario)
        iv. bool != (NumeroDecimal, NumeroBinario)

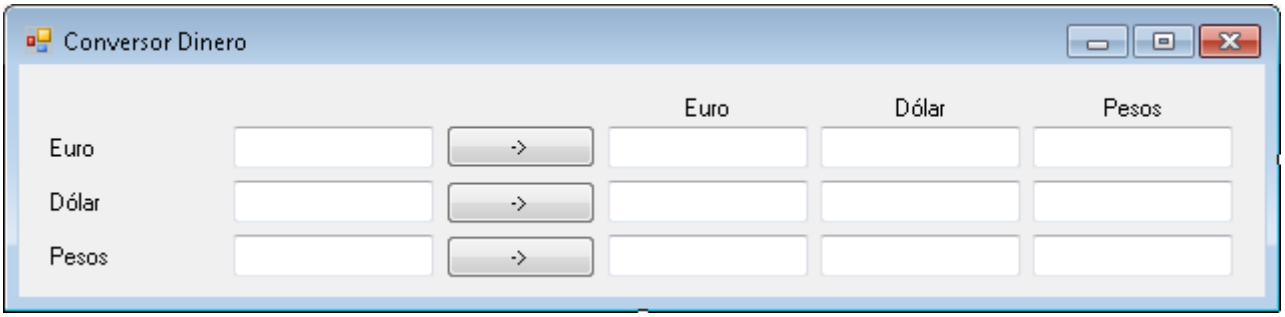
    Agregar conversiones **implícitas** para poder ejecutar:
    e. NumeroBinario objBinario = "1001";
    f. NumeroDecimal objDecimal = 9;

    Agregar conversiones **explícitas** para poder ejecutar:
    g. (string)objBinario
    h. (double)objDecimal

    Generar el código en el Main para instanciar un objeto de cada tipo y operar entre ellos, imprimiendo cada resultado por pantalla.

FORM

23. Tomar el Ejercicio 20. Realizar un Formulario con el siguiente formato:

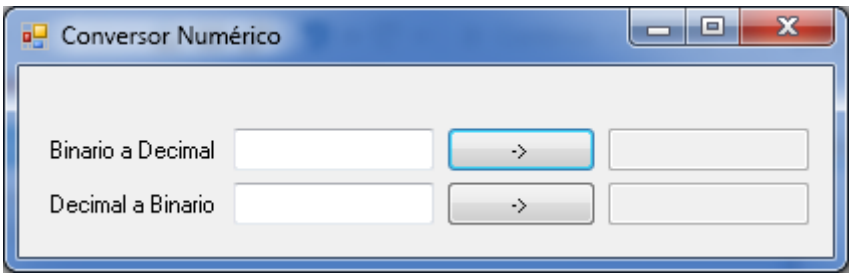


Implementarlo de tal forma que al ingresar un valor válido en la primer casilla (txtEuro, txtDolar y txtPesos respectivamente) y presionar el botón del medio (btnConvertEuro, btnConvertDolar y btnConvertPesos) el resultado de la conversión se vea reflejado en las casillas de la derecha (txtEuroAEuro, txtEuroADolar, txtEuroAPesos, txtDolarAEuro, txtDolarADolar, txtDolarAPesos, txtPesosAEuro, txtPesosADolar y txtPesosAPesos), las cuales no podrán ser editadas/escritas por el usuario.

24. Tomar el Ejercicio 21. Realizar un Formulario con el siguiente formato:

Implementarlo con la misma lógica que el ejercicio anterior.

25. Tomar el ejercicio 22. Realizar un Formulario con el siguiente formato:



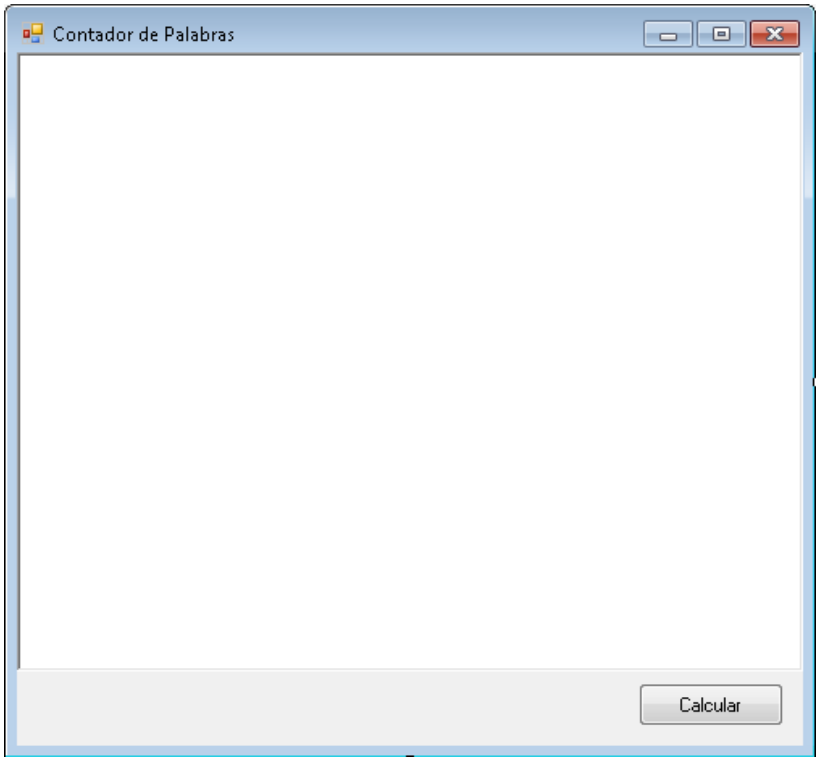
Implementarlo de tal forma que al ingresar un valor válido en la primer casilla (txtBinario y txtDecimal respectivamente) y presionar el botón del medio (btnBinToDec y btnDecToBin) el resultado de la conversión se vea reflejado en las casillas de la derecha (txtResultadoDec y ResultadoBin), las cuales no podrán ser editadas/escritas por el usuario.

Arrays y Colécciones

26. Leer 20 números enteros (positivos y negativos) distintos de cero. Mostrar el vector tal como fue ingresado y luego mostrar los positivos ordenados en forma decreciente y por último mostrar los negativos ordenados en forma creciente.

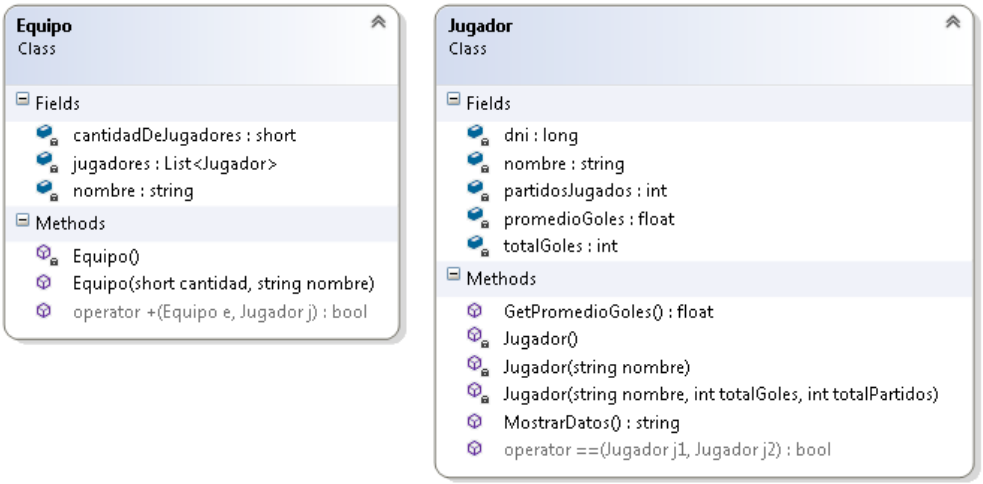
27. Realizar el ejercicio anterior pero esta vez con las siguientes colecciones: ***Pilas***, ***Colas*** y ***Listas***.

28. Generar un WindowsForm con el siguiente formato:



Utilizar Diccionarios para realizar un contador de palabras. Ordenar los resultados de forma descendente por cantidad de apariciones de cada palabra. Informar mediante un MessageBox el TOP 3 de palabras con más apariciones.

29. Crear una Clase llamada **Jugador** y otra Equipo con la siguiente estructura:



Jugador:

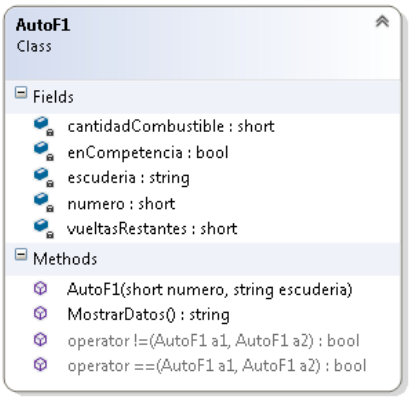
a. Todos los datos estadísticos del Jugador se inicializarán en 0 dentro del constructor privado.
b. El promedio de gol sólo se calculará cuando invoquen al método GetPromedioGoles.
c. MostrarDatos retornará una cadena de string con todos los datos y estadística del Jugador.
d. Dos jugadores serán iguales si tienen el mismo DNI.

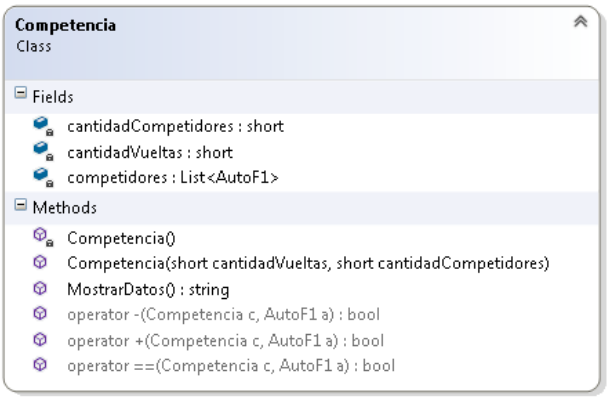
Equipo:

e. La lista de jugadores se inicializará sólo en el constructor privado de Equipo.
f. La sobrecarga del operador + agregará jugadores a la lista siempre y cuando este no exista aun en el equipo y la cantidad de jugadores no supere el límite establecido por el atributo cantidadDeJugadores.

Generar los métodos en el Main para probar el código.

30. Diseñar una clase llamada **Competencia** y otra **AutoF1** con los siguientes atributos y métodos:





AutoF1:

a. Al generar un auto se cargará enCompetencia como falso y cantidadCombustible y vueltasRestantes en 0.
b. Dos autos serán iguales si el número y escudería son iguales.
c. Realizar los métodos getters y setters para cantidadCombustible, enCompetencia y vueltasRestantes.

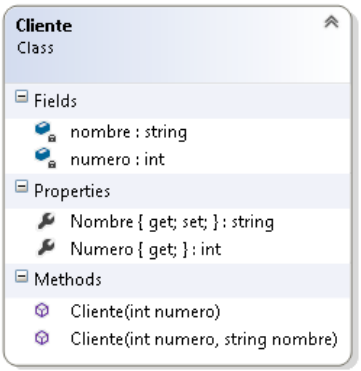
Competencia:

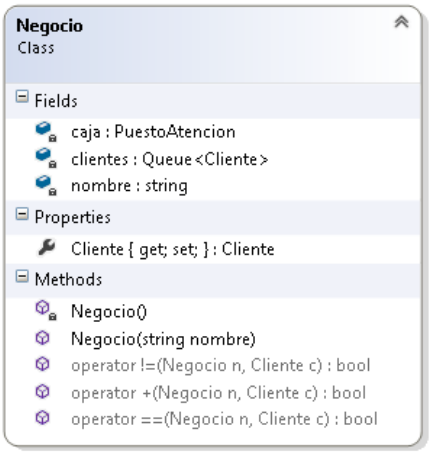
d. El constructor privado será el único capaz de inicializar la lista de competidores.
e. La sobrecarga + agregará un competidor si es que aún hay espacio (validar con cantidadCompetidores) y el competidor no forma parte de la lista (== entre Competencia y AutoF1).
f. Al ser agregado, el competidor cambiará su estado enCompetencia a verdadero, la cantidad de vueltasRestantes será igual a la cantidadVueltas de Competencia y se le asignará un número random entre 15 y 100 a cantidadCombustible.

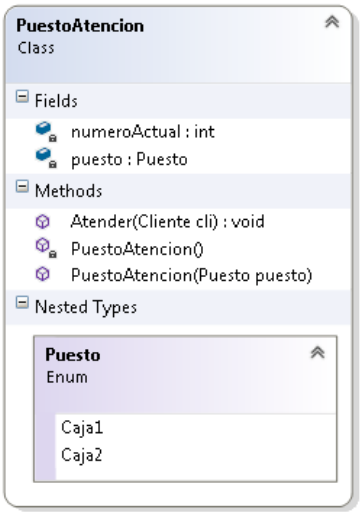
Generar los métodos en el Main para probar el código.

ENCAPSULAMIENTO

31. Generar un sistema de atención al cliente mediante las clases Cliente, Negocio y PuestoAtencion:







Puesto Atención:

a. numeroActual es estático y privado. Se inicializará en el constructor de clase con valor 0.
b. El método Atender recibirá un cliente, simulará un tiempo de atención mediante el método de clase Sleep de la clase Thread y retornará true para indicar el final de la atención.
c. La propiedad de clase NumeroActual incrementará en 1 a numeroActual y lo retornará.

Cliente:

d. Dos clientes serán iguales si tienen el mismo número.

Negocio:

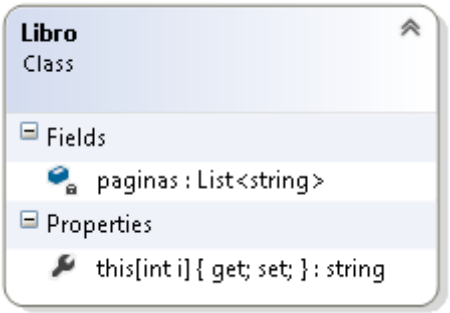
e. El constructor privado inicializará la lista y el puesto de atención como Caja1.
f. El operador + será el único capaz de agregar un Cliente a la cola de atención.
g. La propiedad Cliente retornará el próximo cliente en la cola de atención en el get. El set deberá controla que el Cliente no figure ya en la cola de atención, caso contrario lo agregará.

Generar los métodos en el Main para probar el código.

32. Tomar el ejercicio 29 como base. Agregar propiedades de sólo lectura a los atributos partidosJugados, promedioGoles y totalGoles de Jugador, y de lectura/escritura a nombre y dni. Quitar el método GetPromedioGoles, colocando dicha lógica a la respectiva propiedad. Realizar todos los cambios necesarios para que vuelva a funcionar como antes.

33. Tomar el ejercicio 30 como base. Agregar propiedades de sólo lectura a los atributos nombre y número de AutoF1, y de lectura/escritura al resto de sus atributos.

34. Crear la clase Libro:



El indexador leerá la página pedida o la asignará si esta ya existe. Si el índice es superior al máximo existente, agregará una nueva página.

Generar los métodos en el Main para probar el código.

HERENCIA

35. Tomar el ejercicio 32 de esta guía. Generar una nueva clase DirectorTecnico y hacer que esta y la clase Jugador hereden de Persona, quedando el siguiente formato:
36.

THREADS

37. Tomar el ejercicio XX de esta guía (AutoF1). Hacerlos correr.