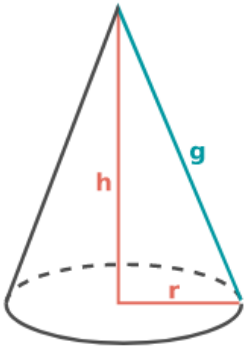
Parcial POO – Tema 2

1. Crear la clase Cono, según lo siguiente ***(1 punto)***:
   a. Tiene como atributos el ***radio*** de la base y la ***altura***, ambos de tipo entero y de sólo lectura.
   b. Tiene un atributo privado Generatriz, de tipo double.
      i. ***Generatriz***=$\sqrt{h^2 + r^2}$ donde ***h*** es la altura y ***r*** el radio
   c. La clase tiene un constructor donde se pasan los valores de sus atributos, si los datos no son válidos, arrojar una excepción de tipo ***ArgumentException***.
   d. La clase debe poder informar su área y volumen.
      i. Área=$AreaBase + AreaLateral$.
         1. AreaBase= $\pi * r^2$ donde ***r*** es el radio.
         2. AreaLateral=$= \pi * r * g$ donde ***r*** es el radio y ***g*** es la generatriz.
      ii. Volumen=$\frac{\pi * r^2 * h}{3}$ donde ***r*** es el radio y ***h*** la altura.
   e. La clase debe tener un método para informar todos sus datos, cada dato en una línea diferente.
   f. Utilizar la clase en un proyecto de consola.
      i. Lote de prueba:
         1. Radio de la base: 3
         2. Altura: 5
         3. Resultado:
            a. Volumen: 47.12388980384689
            b. Área: 83.22976079115258
            c. Generatriz: 5.830951894845301



2. Crear la clase estática `ValidadorPatente`, que debe tener un único método público `Validar`, que recibirá como parámetro un ***string***, tener en cuenta que las patentes en Argentina tienen 2 formatos válidos: **AAA NNN** o **AA NNN AA** ***(2 puntos)*.**
   a. Debe tener un método privado que indique si la patente tiene el nuevo formato o el anterior.
   b. Utilice la clase en un programa de consola.
3. Crear las clases `Kilómetros` y `Millas` con las siguientes características ***(2 puntos)***:
   a. Cada clase tiene 1 atributo privado `distancia` de tipo float.
   b. Cada clase tiene métodos para informar dicho atributo.
   c. En el constructor se proporciona el valor del atributo `distancia`, que por defecto inicializa en 100.
   d. Hacer la sobrecarga implícita entre float y las clases.
   e. Hacer la sobrecarga explícita entre las clases.
   f. Sobrecargar los operadores de igualdad en las clases. Los operadores de comparación == compararán las distancias.
   g. Se debe lograr que los objetos de estas clases se puedan sumar entre sí con total normalidad como si fueran tipos numéricos, teniendo en cuenta las siguientes equivalencias:
      i. 1 Km.=0.621371 millas.
      ii. 1 Milla = 1.6093 Kilómetros
   h. ***Siempre que se pueda se debe reutilizar código***.

i. Probar en programa de consola.

4. Crear un programa que permita almacenar una cantidad determinada de ***números perfectos***. ***(4 puntos)***

Números Perfectos: Un número perfecto es un número entero positivo que es igual a la suma de sus divisores propios positivos, excluido él mismo. Por ejemplo, 6 es un número perfecto porque sus divisores son 1, 2 y 3, y 1 + 2 + 3 = 6.

Clase NumeroPerfecto

a. Tiene un atributo Valor de tipo entero de lectura y escritura a través de propiedades.
b. Se adjunta el método para validar el número perfecto:
c. Sobrescribir los métodos heredados de Object.
d. Sobrecargar los operadores == y != para comparar 2 números por su Valor.

Clase RepositorioNumerosPerfectos

a. El atributo ***cantidad*** es de tipo entero y privado.
b. Debe tener un ***método público*** para informar dicho atributo.
c. Otro atributo privado de tipo array de ***NumeroPerfecto***, que acepte nulos, donde se almacenarán los números perfectos ingresados.
d. Definir 2 constructores:
    a. Uno que tome un parámetro que inicializará el atributo ***cantidad***.
    b. Otro que no reciba parámetros y por defecto inicializará el atributo ***cantidad*** en ***5***. En ambos casos se inicializa el array.
e. El repositorio debe definir métodos privados para:
    a. Establecer si está completo.
    b. Establecer si está vacío.
    c. Informar si un elemento ya existe.
f. Crear métodos para agregar y quitar un número perfecto del repositorio, los mismos devolverán una ***tupla<bool, string>***, indicando si se pudo realizar la operación y una descripción de la misma o del error.
g. Definir un método público que permita acceder a un elemento, el mismo puede devolver un nulo; si el índice estuviere fuera de rango, ***arrojar una excepción***.
h. Definir un método que devuelva un ***string*** con todos los números que tuviere, en el caso de un elemento nulo, mostrar ***"Elemento Nulo"***; si el vector está vacío, mostrar el mensaje ***"No hay elementos almacenados todavía".***
i. Definir un método que devolverá una ***tupla<bool, int>***, que informe si un número forma parte del repositorio y en ***qué posición se encuentra***, caso contrario informar ***falso*** y -1 como posición.
j. Crear una ***sobrecarga implícita*** del repositorio a un ***número entero***, devolviendo la ***suma*** de los números.
k. Utilizar las clases en un programa de consola con menú para realizar todas las operaciones pertinentes.

5. Testing ***(1 punto)***:
    a. Crear proyecto de testing de la clase del Ejercicio 1 para probar todos sus métodos.
    a. Crear un proyecto de testing para probar al menos 4 métodos del repositorio del Ejercicio 4.

Pautas para la resolución y entrega:

1. Solución en Blanco: Nombre y Apellido del alumno.
2. Cada ejercicio en una carpeta con el número de ejercicio.
3. Cada proyecto en su capa correspondiente y con el nombre correspondiente al ejercicio que está resolviendo.
4. Subir la solución a su repositorio en Git.
5. Adjuntar el link al classrrom en tiempo y forma, no serán tenidos en cuenta los trabajos enviados por otros medios y formatos.
6. Utilizar MiDLL para la captura de datos por consola, ya sea en la carpeta compartida o compilada.

Pautas para la corrección:

1. Cada ejercicio entregado NO debe tener errores de compilación, para su corrección.
2. Errores en las fórmulas serán tenidos en cuenta y dicho punto se considerará mal resuelto.
3. Se podrá solicitar al alumno la defensa oral de su trabajo.