Ejercicios Propuestos

La presente relación no tiene mayor objeto que permitir que nos vayamos familiarizando con el lenguaje. Los ejercicios están organizados por temas, de acuerdo con lo que hemos ido viendo y van presentando una dificultad creciente.

Clases

1. Nombre de espacio prueba

Si no lo has hecho antes, crea un nombre de espacio *Utilidades.Prueba* con una clase y un pequeño programa que compruebe que funciona bien.

2. Nombre de espacio matemáticas

Crea ahora el nombre de espacio Utilidades.Matematicas con dos clases llamadas Sumar y Potenciar. La clase Sumar tendrá un método suma(int, int), sobrecargado como suma(double, double) para poder sumar números reales.

La clase Potenciar tendrá un método potencia(int, int) sobrecargado como potencia (double, int), donde el primer parámetro será la base y el segundo el exponente. Luego escribe una clase que permita probar el nombre de espacio para ver si todo resulta como debería.

3. Clase animal

Escribe una clase Animal con los atributos nombre (string) y edad (int), con sus respectivos getters y setters.

Crea un constructor que permita asignar valores a ambos atributos durante la creación del objeto. Implementa para esta clase el método Clone() y el método Equals() heredados de Object para poder copiar y comparar en profundidad, y escribe un programa de prueba que lo demuestre.

4. Contador de animales

Para la clase Animal anterior, crea un atributo numAnimales(int) que cuente cuantos objetos de la clase Animal han sido creados, y comprueba que funciona correctamente.

5. Constructor copia

Añade un constructor copia a la clase Animal y (no debería ser necesario decirlo a estas alturas) comprueba que funciona.

6. Herencia animal

Crea dos clases llamadas Perro y Gato que deriven de la clase Animal. Añade a la clase Perro el atributo raza (String) con su getter y su setter. Añade a la clase Gato el atributo vidas (int) con un valor inicial de 7 y dos métodos: getVidas() y restarVida(). Éste último restará una vida al gato, claro.

Haz un programa que compruebe que todos los métodos de la clase madre (animal) están disponibles en Perro y Gato y funcionan correctamente.

7. Adivinanza

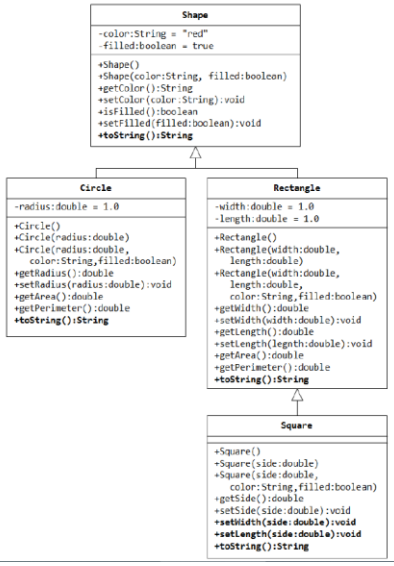
Blanco por dentro, verde por fuera. Que no, que es broma. Se trata de averiguar qué demonios mostrará este programa por pantalla sin teclearlo ni ejecutarlo en la máquina:

```
namespace Ejercicio02 {
    public class Bebe {
        Bebe(int i) : this("Soy un bebé") {
            Console.WriteLine("Hola, tengo " + i + "meses");
        }
        Bebe(String s) {
            Console.WriteLine(s);
        }
        void llora() {
            Console.WriteLine("¡¡Buaaaaaaaa!!");
        }
        public static void main(String[] args) {
            new Bebe(8).llora();
        }
    }

}
```

7. Herencia geométrica

Observa el diagrama adjunto. Es un diagrama de clases. Expresa las relaciones de herencia y muestra los atributos y métodos de cada clase (por cierto: + significa public, y - significa private):



Ahora se trata de que hagas lo siguiente:

a) Escribir la superclase Shape (figura geométrica), que contendrá:
   - Dos atributos de instancia: color (String) y filled (boolean).
   - Dos constructores: uno sin argumentos que inicializará el color a "rojo" y filled a true; y otro que recibirá dos argumentos para inicializar los atributos.
   - Getters y setters.
   - Sobreescritura del método toString() de Object para devolver la cadena: "Soy una figura de color xxx y rellena/no rellena".

b) Escribe una clase que compruebe que los métodos de Shape funcionan.

c) Escribe dos subclases de Shape llamadas Circle y Rectangle.
   a. La clase Circle contendrá:
   - Una variable llamada Radius (double).
   - Tres constructores, como se ve en el diagrama de clases. El constructor sin argumentos establecerá el radio en 1.
   - Getter y setter para Radius.
   - Los métodos getArea() y getPerimeter().
   - Sobreescritura del método toString() heredado de Shape. Ahora, el método
   - devolverá: "Soy un círculo con radio = x, esta es mi superclase: yyy", donde yyy es la salida del método toString() de la superclase de Circle.

   b. La clase Rectangle se comportará igual, con las lógicas diferencias en atributos y métodos getters y setters. Mira el diagrama de clases si tienes alguna duda.

d) Escribe una clase llamada Square como subclase de Rectangle. Esta clase podía haberse modelado como subclase de Shape, pero es más cómodo hacerlo como subclase de Rectangle porque podemos aprovechar casi todo el código de su superclase. Basta con crear el siguiente constructor:

```
public Square(double side) : base(side,side ) { }; // Llama a la superclase Rectangle(double, double)
```

- Además de crear el constructor, sobreescribe, como en los otros casos, el método toString().
- Atención, pregunta: ¿necesitarás sobreescribir getArea() y getPerimeter(), o funcionarán tal y como han sido heredados de Rectangle? Haz la prueba a ver qué pasa...
- Sobreescribe los setters setLength() and setWidth() para evitar que el largo y el ancho del cuadrado puedan tener dimensiones diferentes.

e) Finalmente, escribe una clase de prueba que testee que todo lo anterior funciona como se espera.

8. Empleados

Crea una clase Empleado con los atributos nombre y sueldo y los getters y setters correspondientes.

Crea luego una subclase Encargado. Los encargados cobran un 10% más, aunque realicen el mismo trabajo. Sobrescribe el método getSueldo() y comprueba que funciona.

9. Huevos

Escribe dos clases, Clara y Yema, dentro de una clase Huevo. Crea dos métodos hazClara() y hazYema() dentro de Huevo, que se encarguen de instanciar la Clara y la Yema. Estas últimas mostrarán un mensaje en su constructor, para comprobar que funcionan.

10. Calculadora

Utiliza wrappers para escribir una clase que lea por teclado dos números y muestre la suma, la multiplicación, la división y el módulo. En el caso de que el segundo número sea 0, habrá que manejar la excepción aritmética.

11. Formas

Implementa una jerarquía de clases constituida por una clase Forma (abstracta) de la que hereden Circulo, Cuadrado, Triangulo y Rombo.

La clase Forma tendrá un método abstracto toString() y otro identidad(), que mostrará un identificador interno del objeto. Las demás clases heredarán de Forma e implementarán el método abstracto toString()

Después, prueba la jerarquía con este código, pero, cuidado, contiene varios errores. Arréglalo para que funcione.

```
class TestForma {
    public static void main(String[] args) {
        Forma f = new Circulo();
        f.indentidad();
        Circulo c = new Circulo();
        ((Forma)c).identidad();
        ((Circulo)f).identidad();
        Forma f2 = new Forma();
        f2.identidad();
        (Forma)f.identidad();
    }
}
```

12. Hidden class

Crea una clase privada que implemente un interfaz público. Escribe un método que devuelva una referencia a una instancia de la clase privada haciendo upcasting al interfaz. Comprueba entonces que la clase está completamente oculta intentando hacer downcasting hasta ella.