Unidad 2 – Capitulo 2: Laboratorio POO

1. Clases y Herencia

Objetivos

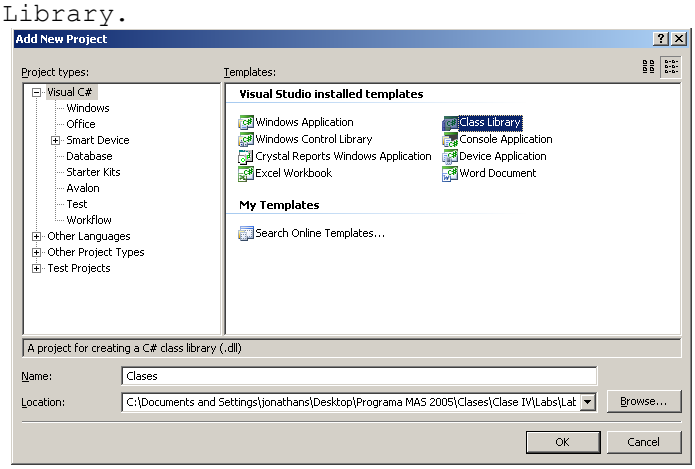
Escribir la primera aplicación que utilice clases y herencia. Crear una librería de clases

Duración Aproximada

30 minutos

Pasos

1) En Visual Studio, crear una nueva aplicación de consola, llamada LabClases1.
2) Agregar un nuevo proyecto a la solución llamado Clases, que sea de tipo Class Library.

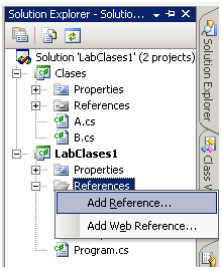


3) Al item class1.cs, renombrarlo a A.cs. Renombrar tambien el nombre de la clase dentro del archivo.
4) Agregar un using para System.Console, para acceder mas rapido a los elementos de escritura y lectura de la consola.
5) La clase A debe tener un property "NombreInstancia". Dos constructores, uno por defecto, que asigne nombre instancia como "Instancia sin nombre" y otro que permita pasarle por parametro el nombre. Además, 1 método "MostrarNombre" que devuelva por consola el nombre de la instancia y otros tres métodos M1, M2 y M3 que muestren por consola un mensaje avisando que el metodo fue invocado.

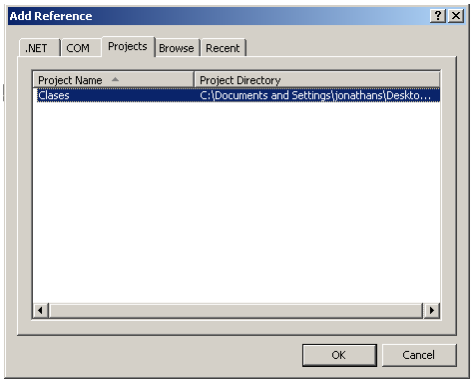
   Luego, agregar un nuevo item de tipo file a la solución, B.cs, donde se implementará la clase B, heredera de A. Esta clase debe implementar el método M4, que muestre por consola el mensaje "Metodo del hijo Invocado". El constructor de B debe pasar por defecto, como parametro al constructor de la clase A "Instancia de B".

   Compilar la clase. Observar que se genera un Assembly propio, con nombre Clases.
6) Para poder utilizar las clases en nuestra aplicación, será necesario referenciarla. Para esto, desde el Solution Explorer, accedemos a

las referencias del proyecto LabClases1, y agregamos una nueva referencia, haciendo click derecho sobre la carpeta References.



Desde el cuadro de diálogo, nos dirigimos a la ventana Projects, para poder agregar una referencia a un proyecto que existe dentro de la misma solución, y elegimos Clases. Ahora LabClases1 "conoce" al assembly Clases.



7) Definir el using para "Clases" en program.cs. Crear una instancia de A y otra de B, observar el intellisense, y los métodos disponibles para cada clase. Llamar a todos los métodos de A y B, observar que, si bien no fueron definidos en B, M1, M2 y M3 están disponibles.
8) Compilar y Ejecutar la aplicación.
9) Probar cambiar accesibilidad de los miembros, de public a protected y a private. Observar los errores en tiempo de compilación y analizarlos.

2. Métodos Virtuales y Ocultamiento

Objetivos

Comprender el uso de los modificadores virtual, new y override. Poder predecir las llamadas

Duración Aproximada

30 minutos

Pasos

1) Crear una nueva aplicación de consola llamada "LabClases2"
2) Agregar un proyecto Clases, de tipo Class Library. Renombrar el file a Clases.cs
3) Copiar el siguiente código:

```
namespace Clases
{
    public class A
    {
        public void F() { Console.WriteLine("A.F"); }
        public virtual void G() { Console.WriteLine("A.G"); }
    }

    public class B : A
    {
        new public void F() { Console.WriteLine("B.F"); }
        public override void G() { Console.WriteLine("B.G"); }
    }
}
```

Observar que se estan definiendo dos clases en el mismo archivo físico. Para la CLR esto es completamente transparente. Solo importa que se esta definiendo Clases.A y Clases.B dentro de un assembly llamado clases.

4) Compilar la librería. Observar que el compilador no arroja ningún warning ni error.
5) Referenciar Clases en el proyecto principal de la solución. Copiar el siguiente código dentro del main:

```
B b = new B();
A a = b;
a.F();
b.F();
a.G();
b.G();

Console.ReadKey();
```

6) Compilar y ejecutar el código. Analizar la salida por pantalla.
7) Eliminar el modificador new del método F de la clase B. Compilar. Observar que se arroja un warning, advirtiendo que si es intencional el redefinir el método F, debe ser declarado como new.
8) Quitar el operador override de B.G(). Observar que arroja un warning, y que al ejecutar la aplicación, llama a A.G().

3. Visor de Clases y Diseñados de Clases

Objetivos

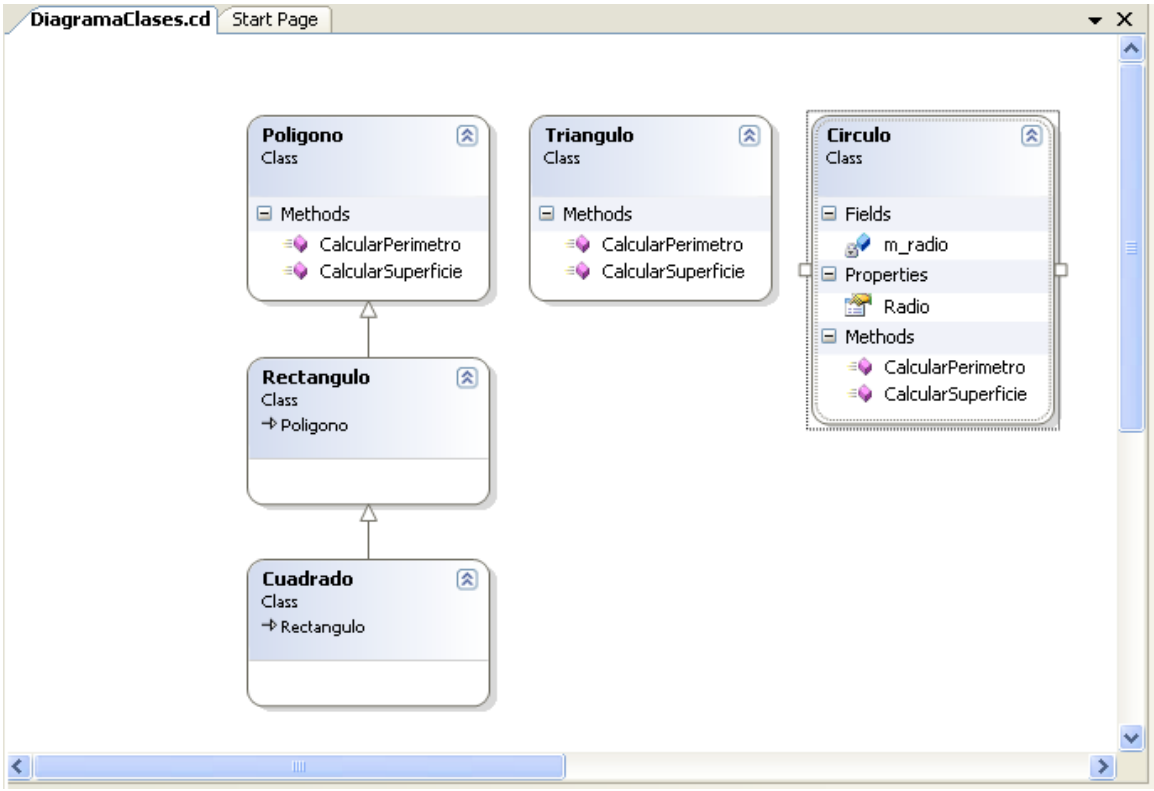
Utilizar el diseñador de clases (Class Designer) para crear una jerarquía de clases y los componentes y estructuras principales de cada clase. Visualizarlos mediante el Visor de Clases. Reforzar conceptos de POO básicos con un ejemplo sencillo.

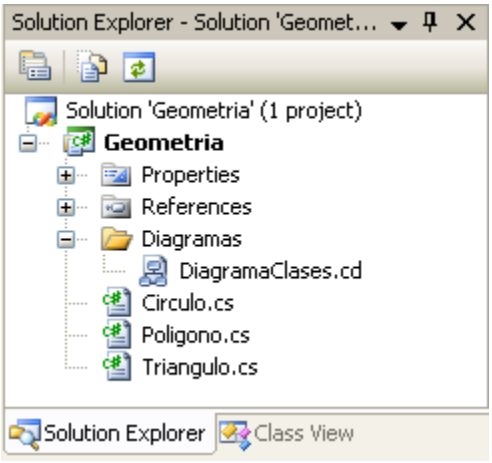
Duración Aproximada

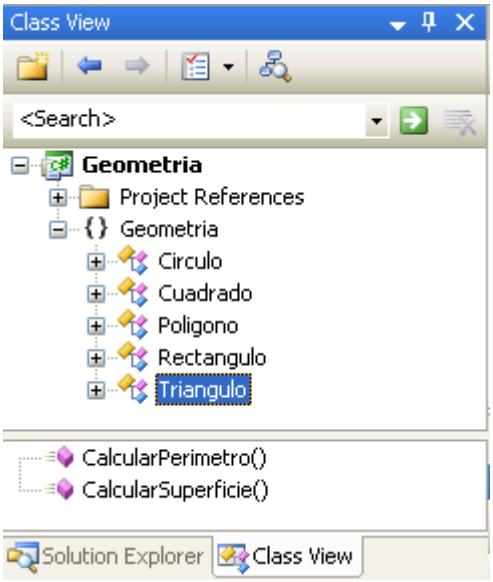
60 minutos

Pasos

1) Crear un proyecto tipo librería de clases (Class Library) de nombre "*Geometria*" al igual que la solucion que lo contenga.
2) Modificar el nombre a la clase (Class1) creada por defecto al crearse el proyecto por el nombre "*Circulo*"
3) Agregar al proyecto *Geometria* un nuevo item del tipo Diagrama de clases (Class Diagram) y ponerle de nombre "*DiagramaClases*".
4) Activar y visualizar (en caso de no estarlo aun) el Visor de Clases (Class View) y la caja de herramientas (Tool Box).
5) Ubicar la clase *Circulo* en el Visor de Clases y arrastrarla al Disenador de Clases. Se vera que se agrega graficamente la clase en el diseñador.
6) En el diseñador de clases ubicado sobre la clase *Circulo* elegir del menú de contexto la opción **Add** / **Field** agregar campo: *m_radio*.
7) Idem al anterior a excepcion de elegir opción **Add** / **Property** agregar propiedad: *Radio*.
8) Idem al anterior a excepcion de elegir opción **Add** / **Method** agregar métodos: *CalcularPerimetro* y *CalcularSuperfie*.
9) Acceder al código de la clase Triangulo (mediante doble clic o alguna otra forma alternativa) agregar al código la implementacion dentro de la propiedad para devolver el valor de la variable (o campo) y para setear su valor.
10) Siguiendo con la implementación de la clase Triangulo implementar los métodos agregados anteriormente.
    a. Triangulo
    b. Poligono
    c. Rectangulo
    d. Cuadrado







11) Siguiendo los pasos anteriores continuar con el desarrollo de tal forma de obtener la siguiente estructura jeraruica de clases.
12) Implementar los miembros (campos, metodos y propiedades) de cada clase agregada en el punto anterior según corresponda.

4. Clase Persona

Objetivos

Familiarizarse con el uso del diseñador de clases (Class Designer).
Reforzar conceptos de POO y Sintaxis de lenguaje

Recomendaciones

Construir una aplicación que cumpla con los requisitos siguientes:

1) Declarar una clase "Persona" que contenga la información siguiente de una persona:
   - Nombre
   - Apellido
   - Edad
   - DNI

2) Implementar los métodos y propiedades siguientes:
   - Un constructor que reciba la información necesaria para iniciar la instancia y que proporcione un mensaje que indique la creación del objeto.
   - Un destructor que proporcione un mensaje que indique la destrucción del objeto.
   - Las propiedades correspondientes para acceder a cada uno de los atributos de la clase.
   - Un método GetFullName, el cual debe devolver la concatenación del nombre y apellido.
   - Un método GetAge, para calcular la edad de la persona.

5. Adivine el Número

Objetivos

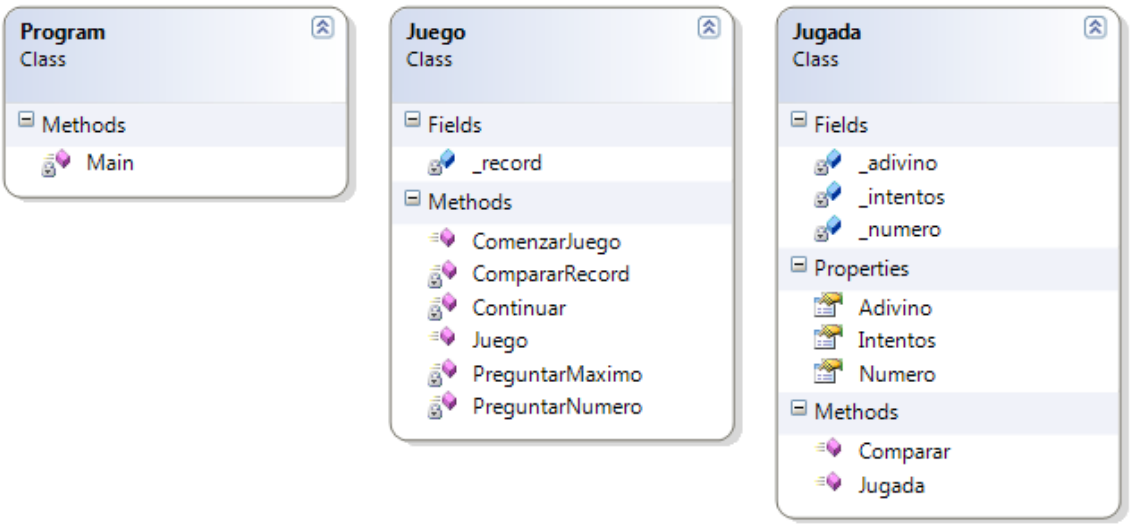
Familiarizarse con el uso del diseñador de clases (Class Designer).
Reforzar conceptos de POO y Sintaxis de lenguaje

Recomendaciones

1) Deben crearse dos clases, en archivos separados, dentro del proyecto, llamadas Juego y Jugada.
2) Juego debe tener un único método publico, ComenzarJuego(), llamado desde el main de la aplicación. Se debe encargar de mostrar mensajes al usuario, permitir realizar otra partida, llevar un control de record de menor cantidad de intentos para un acierto, etc.
3) Jugada debe generar, en el constructor, el número a adivinar, a partir de un máximo, para esto, se utilizará como constructor:

```
public Jugada(int maxNumero)
{
      Random rnd = new Random();
      Numero = rnd.Next(maxNumero);
}
```

4) Debe controlar la cantidad de intentos, en un atributo privado, pero que debe ser visto por Juego, gracias a una Property. Debe tener otra Property que le permita saber a Juego si ya fue acertado el número.
5) Deben capturarse errores y mostrar mensajes explicativos por pantalla.
6) Diagrama de Clases propuesto:



6. Adivine el Número con Ayuda

Objetivos

Familiarizarse con el uso del diseñador de clases (Class Designer).
Reforzar conceptos de POO y Sintaxis de lenguaje

Recomendaciones

1) Se debe implementar una nueva clase, JugadaConAyuda, heredera de Jugada. El método Comparar deberá brindar una ayuda, en función de un rango. Por ejemplo, si es mayor y dista 100 números del número esperado, debe anunciar que es mayor y está muy lejos. Si esta a 5 números, debe informar que está cerca. Modificar en la clase Juego solamente la instanciación de Jugada. Si los conceptos de POO fueron bien aplicados, simplemente debería reemplazarse la clase y funcionar todo de la misma manera que antes, con la nueva jugada.