Herencia en c#

La herencia no es más que la capacidad de un objeto de heredar las características de otro. Bueno, en programación, la herencia es la capacidad de una clase de heredar los métodos, las variables y las propiedades de otra (los constructores no se heredan). Esto permite ahorrar código, ya que varias funcionalidades no tendríamos que implementarlas nuevamente.

Una clase que herede de otra, puede usar los métodos, las propiedades o las variables de la clase padre. También se puede redefinir o modificar los métodos y propiedades, usando algunas palabras reservadas para esto, creando así un nuevo comportamiento del objeto. Una condición necesaria para que una clase herede de otra, es que la clase hija (la que hereda), debe poder usarse donde se use la clase padre (de la cual va a heredar). Si hay al menos un caso donde se puede usar la clase padre, y no se puede usar la clase hija, la herencia no tiene sentido.

**En C#, una clase solo puede heredar de otra. O sea, una clase no puede heredar de varias clases a la vez. Para que una clase herede de otra nada más hay que ponerle ‘:‘ después de su declaración, y seguido poner el nombre de la clase de la que se quiere heredar.**

**Ejemplo de herencia:**

public class Hijo:Padre

{

*//la clase Hijo hereda de la clase Padre*

public Hijo(int n):base(n){}

}

*//Declaramos la clase padre, de la que todas las figuras heredarán*

*//Es abstracta porque tendrá algunos metodos abstractos*

public abstract class Figura

{

*//Usamos <strong>protected,</strong> para que solo tengan acceso*

*//a estas variables, las clases que heredan de ella*

protected int x,y;

protected Control control;

protected Pen myPen;

protected bool visible;

*//Un constructor por defecto para todas las figuras*

public Figura(int x, int y, Color color, Control control)

{

*//Validar parametros*

this.x = x;

this.y = y;

this.control = control;

myPen = new Pen(color,4);

}

public abstract void Muestra();

public abstract int Area();

public virtual void Oculta()

{

Color temp = myPen.Color;

myPen.Color = control.BackColor;

Muestra();

myPen.Color = temp;

visible = false;

}

public virtual void Traslada(int plusX, int plusY)

{

if(visible)

{

Oculta();

x += plusX;

y += plusY;

Muestra();

}

else

{

x += plusX;

y += plusY;

}

public class Circulo:Figura

{

*//Vemos que en esta clase solo hay que declarar*

*//una variable*

protected int radio;

*//Este constructor tendrá todos los parámetros*

*//por defecto y un radio porque es un circulo*

public Circulo(int x, int y, Color miColor, Control miControl,

int radio)

:base(x,y,miColor,miControl)

{

*//Validacion*

this.radio = radio;

}

*//Ahora hay que programar como se mostrará la figura*

public override void Muestra()

{

Graphics g = control.CreateGraphics();

g.DrawEllipse(myPen,x-radio,y-radio,radio\*2,radio\*2);

visible = true;

}

*//Decimos como se calcula el area en este caso*

public override int Area()

{

return Math.PI \* Math.PI \* radio;

}

*//Le podemos añadir otro método independiente*

public int Perimetro()

{

return 2 \* Math.PI \* radio;

}

}

*//Esta clase también heredará de figura*

public class Rectangulo:Figura

{

protected int ancho,alto;

public Rectangulo(int x, int y, Color miColor, Control miControl,

int ancho, int alto)

:base(x,y,miColor,miControl)

{

*//Validar*

this.ancho = ancho;

this.alto = alto;

}

*//Redefinimos como mostrar la figura*

public override void Muestra()

{

Graphics g = control.CreateGraphics();

g.DrawRectangle(myPen,x,y,ancho,alto);

visible = true;

}

*//Redefinimos como calcular el area*

public override int Area()

{

return alto \* ancho;

}

}