Poliformismo

Integrantes:

Lizbeth Guadalupe Ibáñez

Carlos Alberto Benítez Chilaca

Omar Tovar

Grupo: 7A

Carrera: TIC

### Significado Poliformismo

Se refiere a que una misma función (un método) puede tener varias formas, ya sea porque reciba distintos tipos de parámetros y/o en distinta cantidad, o porque incluso se aplique a distintos objetos.

### Tipos

**Sobrecarga**: Se llaman igual pero reciben distintos parámetros, y se comportan de forma que puede ser distinta, son ejemplos de "**sobrecarga**" (también conocida como "polimorfismo ad-hoc"). Es un tipo de polimorfismo en el que el compilador sabe en tiempo de compilación a qué método se debe llamar.

El caso opuesto es el "**polimorfismo puro**", en el que un mismo método se aplica a distintos objetos de una misma jerarquía, el compilador puede llegar a no ser capaz de saber en tiempo de compilación a qué método se debe llamar, y lo tiene que descubrir en tiempo de ejecución.

### Ejemplo

using System;

public class Animal

{

public Animal()

{

Console.WriteLine("Ha nacido un animal");

}

}

// ------------------

public class Perro: Animal

{

public Perro()

{

Console.WriteLine("Ha nacido un perro");

}

}

// ------------------

public class Gato: Animal

{

public Gato()

{

Console.WriteLine("Ha nacido un gato");

}

}

// ------------------

public class GatoSiames: Gato

{

public GatoSiames()

{

Console.WriteLine("Ha nacido un gato siamés");

}

}

// ------------------

public class Ejemplo62

{

public static void Main()

{

Animal a1 = new Animal();

GatoSiames a2 = new GatoSiames();

Perro a3 = new Perro();

Gato a4 = new Gato();

}

}

Clase Figura

using System;

using System.Drawing;

namespace App.Polimorfismo

{

//Clase Abstracta para obligar a implementar instancias

abstract class Figura

{

//Campo - Atributos

private Color Color { get; set; }

public int alto { get; set; }

public int ancho { get; set; }

public Size Tamaño

{

get

{

return new Size(ancho, alto);

}

set

{

Size temp = value;

alto = temp.Height;

ancho = temp.Width;

}

}

public int X { get; set; }

public int Y { get; set; }

//Método o función

/\*---------------------------------------------

Se establecen Virtual para obligar a

implementar un método con Override

---------------------------------------------\*/

public virtual string Dibujar()

{

return "Proceso realizado.";

}

public virtual void EstablecerColor(Color otrocolor)

{

this.Color = otrocolor;

}

public virtual Color ObtenerColor()

{

if (this.Color == null)

return Color.Black;

return this.Color;

}

public virtual Rectangle ObtenerFigura()

{

return new Rectangle();

}

}

}

Clase Circulo

using System;

using System.Drawing;

namespace App.Polimorfismo

{

//Heredad de Figura

class Circulo : Figura

{

//Más atributos

public int radio

{

get

{

return alto / 2;

}

}

//Métodos

public override string Dibujar()

{

return "Dibujar Circunferencia"; //return base.Dibujar();

}

//Nuevo color

public override void EstablecerColor(Color otrocolor)

{

if (otrocolor == null)

base.EstablecerColor(Color.Black);

else

base.EstablecerColor(otrocolor);

}

//Regresamos el color

public override Color ObtenerColor()

{

return base.ObtenerColor();

}

/\* ----------------------------------------

Segmento para dibujar en pantalla

-------------------------------------------\*/

//Construimos una circunferencia

public override Rectangle ObtenerFigura()

{

Rectangle circunferencia = new Rectangle();

circunferencia.X = this.X;

circunferencia.Y = this.Y;

circunferencia.Width = this.Tamaño.Width;

circunferencia.Height = this.Tamaño.Height;

return circunferencia;

}

//Construimos el color de relleno

public Brush ObtenerPincel()

{

return new SolidBrush(this.ObtenerColor()); ;

}

}

}

Clase Rectangulo

using System;

using System.Drawing;

namespace App.Polimorfismo

{

class Rectangulo : Figura

{

//Métodos

public override string Dibujar()

{

return "Dibujar Rectangulo"; //return base.Dibujar();

}

//Nuevo color

public override void EstablecerColor(Color otrocolor)

{

if (otrocolor == null)

base.EstablecerColor(Color.Black);

else

base.EstablecerColor(otrocolor);

}

//Regresamos el color

public override Color ObtenerColor()

{

return base.ObtenerColor();

}

//Construimos un rectangulo

public override Rectangle ObtenerFigura()

{

Rectangle contorno = new Rectangle();

contorno.X = this.X;

contorno.Y = this.Y;

contorno.Width = this.Tamaño.Width;

contorno.Height = this.Tamaño.Height;

return contorno;

}

//Construimos el color de relleno

public Pen ObtenerLapiz()

{

return new Pen(this.ObtenerColor()); ;

}

//Construimos el color de relleno

public Brush ObtenerPincel()

{

return new SolidBrush(this.ObtenerColor()); ;

}

}

}

Clase Triangulo

using System;

using System.Drawing;

namespace App.Polimorfismo

{

class Triangulo : Figura

{

//Métodos

public override string Dibujar()

{

return "Dibujar Triangulo"; //return base.Dibujar();

}

//Nuevo color

public override void EstablecerColor(Color otrocolor)

{

if (otrocolor == null)

base.EstablecerColor(Color.Black);

else

base.EstablecerColor(otrocolor);

}

//Regresamos el color

public override Color ObtenerColor()

{

return base.ObtenerColor();

}

//Construimos un triángulo

public override Rectangle ObtenerFigura()

{

Rectangle contorno = new Rectangle();

contorno.X = this.X;

contorno.Y = this.Y;

contorno.Width = this.Tamaño.Width;

contorno.Height = this.Tamaño.Height;

return contorno;

}

//Construimos un triángulo

public Point[] DibujarTriangulo()

{

Point A = new Point(0, alto);

Point B = new Point((ancho / 2), 0);

Point C = new Point(ancho, alto);

Point[] list = new Point[] { A, B, C };

return list;

}

//Construimos el color de relleno

public Pen ObtenerLapiz()

{

return new Pen(this.ObtenerColor()); ;

}

}

}

Cambio Form

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace App.Polimorfismo

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void llamadacirculo(int x2, int y2)

{

Graphics grafico = CreateGraphics();

Brush pincel = Brushes.Black;

//Esto es para llamar la clase circulo

Circulo circunferencia = new Circulo() { alto = 300, ancho = 300, X = x2, Y = y2 };

circunferencia.EstablecerColor(Color.Green);

grafico.FillEllipse(circunferencia.ObtenerPincel(), circunferencia.ObtenerFigura());

grafico.DrawEllipse(Pens.Red, circunferencia.ObtenerFigura());

}

private void llamadatriangulo(int x2, int y2)

{

Graphics grafico = CreateGraphics();

Brush pincel = Brushes.Black;

//Esto es para llamar a la clase triangulo

Triangulo triangulo = new Triangulo() { alto = 100, ancho = 100, X = x2, Y = y2 };

triangulo.EstablecerColor(Color.Blue);

int diametro = 100;

grafico.DrawLine(Pens.Purple, 2, diametro, diametro, 2);

grafico.DrawPolygon(triangulo.ObtenerLapiz(), triangulo.DibujarTriangulo());

}

private void llamadarectangulo(int x2, int y2)

{

Graphics grafico = CreateGraphics();

Brush pincel = Brushes.Black;

//Instanciamos el objeto circunferencia

Rectangulo rect = new Rectangulo() { alto = 100, ancho = 100, X = x2, Y = y2 };

rect.EstablecerColor(Color.Gray);

//grafico.DrawRectangle (pincel, X, Y, alto, ancho)

grafico.DrawRectangle(rect.ObtenerLapiz(), rect.ObtenerFigura());

}

private void button1\_Click(object sender, EventArgs e)

{

llamadacirculo(Int32.Parse(x.Text), Int32.Parse(y.Text));

}

private void button2\_Click(object sender, EventArgs e)

{

llamadarectangulo(Int32.Parse(x.Text), Int32.Parse(y.Text));

}

private void button3\_Click(object sender, EventArgs e)

{

llamadatriangulo(Int32.Parse(x.Text), Int32.Parse(y.Text));

}

}

}