



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

Clases Abstractas

Las clases abstractas contienen definiciones que podrán ser heredadas, pero no permitirán instanciar objetos a partir de ellas. Tienen la jerarquía más alta en el grado de abstracción de manera que sólo a partir de clases heredadas se podrá tener una especificación de todos sus elementos y poder así crear objetos.

Las clases abstractas permiten declarar métodos y propiedades sin necesidad de definir el código que deberán ejecutar, y modificarlo posteriormente en una clase derivada.

Métodos virtuales

Los métodos virtuales permiten modificar un método que ha sido heredado de una clase abstracta.

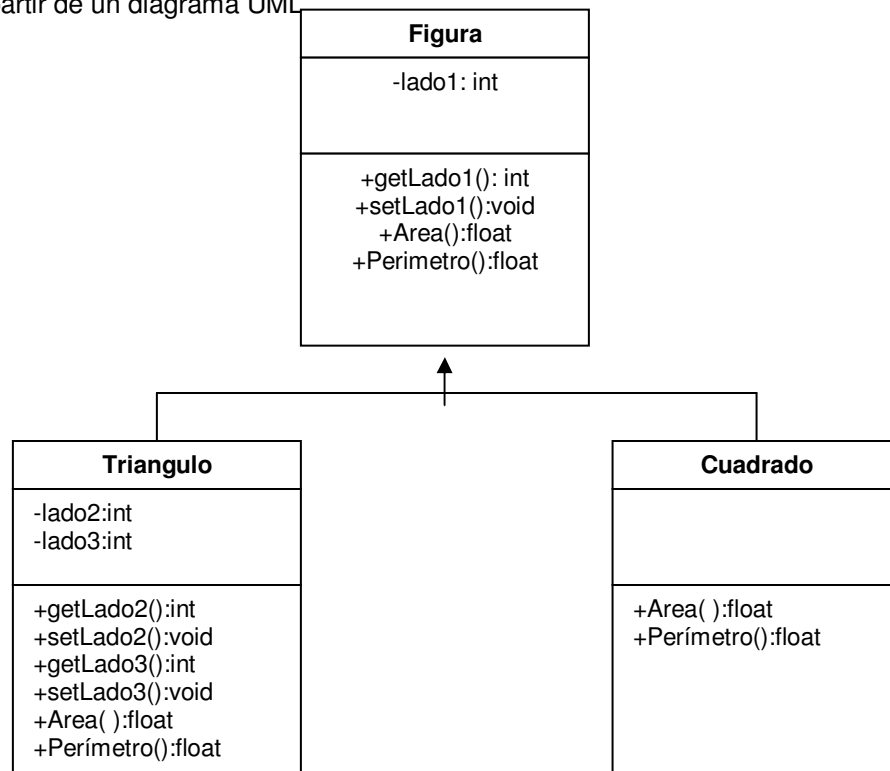
Clases parciales

Las clases parciales permiten dividir una clase en varios archivos donde cada uno de ellos contendrá variables, subclasses, propiedades y métodos; al ser compilados todas las partes se combinan. Este tipo de definición permite que varias personas puedan estar programando partes de la clase al mismo tiempo, así como dar mayor legibilidad a una clase que contenga demasiados elementos.

Todas las partes de la clase deberán llevar la palabra **partial**.

Polimorfismo. La herencia permite que una clase derivada redefina el comportamiento de una clase base. A esta forma de redefinir métodos lo llamamos polimorfismo.

Vamos a ver un ejemplo para entender este concepto. Para ello describiremos el siguiente problema a partir de un diagrama UML.





TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

Como podemos observar en el diagrama anterior tenemos una clase Base llamada Figura de la que derivan dos clases Triangulo y Cuadrado. La Clase derivada Triangulo hereda las variables y métodos de la clase Base Figura, además declara dos variables y redefine los métodos Perímetro() y Area(). En el caso de la clase derivada Cuadrado utiliza todas las variables y métodos de la clase Base Figura, y redefine los métodos Perímetro() y Area().

En las tres clases contamos con los métodos Area() y Perímetro(), sin embargo cada clase lo redefine para reflejar de manera correcta su comportamiento dentro de la clase.

Solo faltaría desarrollarlo en un lenguaje como C#.

Para poder utilizar la herencia de clases de C# se requiere declarar una clases y/o métodos abstractos. Para ello bastará con utilizar la palabra reservada `abstract` antes de la declaración de la clase o método.

```
public class abstract clase_principal
{
    public abstract int metodo1();
    public abstract float metodo2();
    public abstract void metodo3();
}
```

Posteriormente para indicar la herencia de clase y utilizar métodos abstractos deberemos declarar la clase heredada seguida por dos puntos y clase principal. Si se desea aplicar polimorfismo a un método heredado se utiliza la palabra reservada `override`.

```
class clase_heredada : clase_principal
{
    public override int metodo1()
    {
    }
}
```

En caso de clases abstractas las clases derivadas deberán contener una definición para todos los métodos de la clase base.

Otra manera de definir las clases sería utilizando métodos virtuales:

```
class clase_principal
{
    public virtual void metodo1(){ }
    public virtual void metodo1(){ }
    public virtual void metodo1(){ }
}

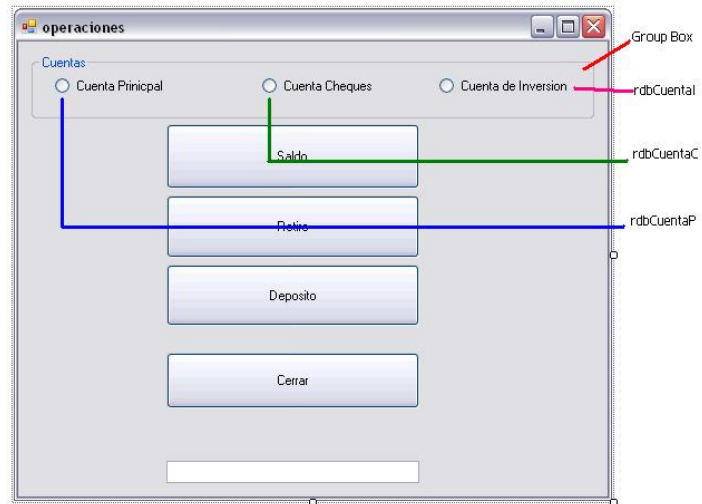
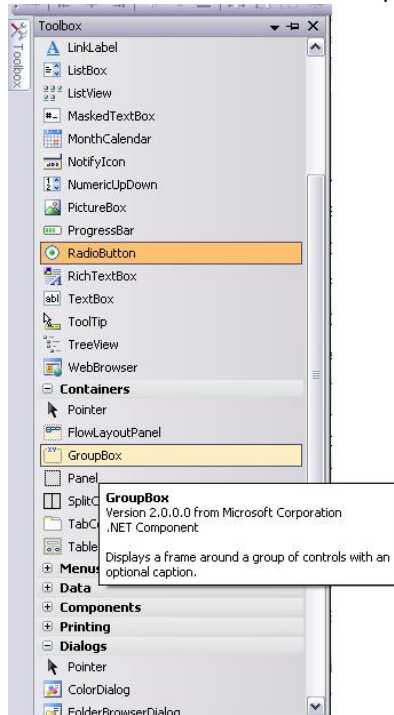
class clase_heredada:clase_principal
{
    public override void metodo1(){ }

    //En este caso podemos tener acceso al metodo1 de la clase padre con la palabra reservada base.
    public void metodo1_padre()
    {
        base.metodo1();
    }
}
```

En este segundo caso no es necesario implementar todos los métodos.

Ejercicios de la Práctica.

- I. Continuando con el ejercicio de la práctica anterior, **banco**, edita el formulario de **operaciones** de la siguiente manera:
 1. Adiciona al formulario un **GroupBox** y dentro del **GroupBox** tres **RadioButton**



Modifica su propiedad **Text** para cambiar las leyendas que aparecen al crear.
Cambia sus nombres por los indicados en la imagen.

2. Adiciona dos nuevos objetos uno de la clase `cuenta_cheque` y otro de la clase `cuenta_inversion` y crea el siguiente método:

```
public void carga_cuentas_adicionales()
{
    cc = new cuenta_cheques();
    ci = new cuenta_inversion();
    cc.cuenta = cp.cuenta;
    cc.nombre_cliente = cp.nombre_cliente;
    cc.conectado = cp.conectado;

    ci.cuenta = cp.cuenta;
    ci.nombre_cliente = cp.nombre_cliente;
    ci.conectado = cp.conectado;
}
```

(Hay código sin implementar necesario para el funcionamiento de este método, identifícalo y corrígelo).

3. Da doble clic sobre el formulario para que se genere el evento **Load** y adiciona la siguiente instrucción:

```
private void operaciones_Load(object sender, EventArgs e)
{
    carga_cuentas_adicionales();
}
```

4. El código completo de tu formulario **operaciones** deberá quedar de la siguiente manera:

```
using System;
using System.Collections.Generic;
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace banco
{
    public partial class operaciones : Form
    {
        cuenta_principal cp;
        cuenta_cheques cc;
        cuenta_inversion ci;

        public void carga_cuentas_adicionales()
        {
            cc = new cuenta_cheques();
            ci = new cuenta_inversion();
            cc.cuenta = cp.cuenta;
            cc.nombre_cliente = cp.nombre_cliente;
            cc.conectado = cp.conectado;

            ci.cuenta = cp.cuenta;
            ci.nombre_cliente = cp.nombre_cliente;
            ci.conectado = cp.conectado;
        }

        public operaciones()
        {
            InitializeComponent();
        }

        public operaciones(cuenta_principal CP)
        {
            InitializeComponent();
            cp = CP;
        }

        private void btnSaldo_Click(object sender, EventArgs e)
        {
            if(rdbCuentaP.Checked==true)
                MessageBox.Show(" Saldo cuenta principal "+cp.Saldo);
            else if(rdbCuentaC.Checked==true)
                MessageBox.Show(" Saldo cuenta cheques " + cc.Saldo);
            else if (rdbCuentaI.Checked == true)
                MessageBox.Show(" Saldo cuenta inversion " + ci.Saldo);
        }

        private void btnRetiro_Click(object sender, EventArgs e)
        {
            retiro_deposito rd = new retiro_deposito(cp, 1, this);
            rd.Show();
        }

        private void btnCerrar_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnDeposito_Click(object sender, EventArgs e)
        {
            retiro_deposito rd=null;
        }
    }
}
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
if (rdbCuentaP.Checked == true)
    rd = new retiro_deposito(cp, 2, this);
else if (rdbCuentaC.Checked == true)
    rd = new retiro_deposito(cc, 2, this);
else if (rdbCuentaI.Checked == true)
    rd = new retiro_deposito(ci, 2, this);
rd.Show();
}

private void operaciones_Load(object sender, EventArgs e)
{
    carga_cuentas_adicionales();
}
}
```

5. Ahora edita el formulario retiro_deposito:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace banco
{
    public partial class retiro_deposito : Form
    {
        cuenta_principal cp;
        cuenta_cheques cc;
        cuenta_inversion ci;
        operaciones op;
        byte operacion;
        public retiro_deposito()
        {
            InitializeComponent();
        }

        public retiro_deposito(cuenta_principal CP, byte Operacion, operaciones OP)
        {
            InitializeComponent();
            cp = CP;
            operacion = Operacion;
            op = OP;
        }

        public retiro_deposito(cuenta_cheques CC, byte Operacion, operaciones OP)
        {
            InitializeComponent();
            cc = CC;
            operacion = Operacion;
            op = OP;
        }

        public retiro_deposito(cuenta_inversion CI, byte Operacion, operaciones OP)
        {
            InitializeComponent();
            ci = CI;
            operacion = Operacion;
            op = OP;
        }

        private void button9_Click(object sender, EventArgs e)
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
{
    if (operacion == 1) //retiro
    {
        op.txtlno.Text = cp.retiro(Convert.ToDecimal(txtCantidad.Text));
    }
    else //Deposito
    {
        if (cp != null)
            op.txtlno.Text = cp.deposito(Convert.ToDecimal(txtCantidad.Text));
        else if (cc != null)
            op.txtlno.Text = cc.deposito(Convert.ToDecimal(txtCantidad.Text));
        else if (ci != null)
            op.txtlno.Text = ci.deposito(Convert.ToDecimal(txtCantidad.Text));
    }
    this.Close();
}

private void btnCantidad0_Click(object sender, EventArgs e)
{
    txtCantidad.Text = txtCantidad.Text + "0";
}

private void btnCantidad1_Click(object sender, EventArgs e)
{
    txtCantidad.Text = txtCantidad.Text + "1";
}
}
```

6. Edita la clase cuentas y adiciona el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace banco
{
    public class cuenta_principal
    {
        public string nombre_cliente;
        private decimal saldo;
        public long cuenta;
        public bool conectado;
        public decimal Saldo
        {
            get
            {
                if (conectado == true)
                    return saldo;
                else
                    return 0;
            }
            set
            {
                if (conectado == true)
                    saldo = value;
            }
        }

        public string retiro(decimal cantidad)
        {
            if (cantidad <= saldo)
            {
                saldo = saldo - cantidad;
                return "Entregando el dinero...";
            }
        }
    }
}
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
    }
    else
    {
        return "Saldo insuficiente...";
    }
}

public string deposito(decimal cantidad)
{
    if (cantidad <= 0)
    {
        return "Debe indicar la cantidad";
    }
    else
    {
        saldo = saldo + cantidad;
        return "Deposito realizado...";
    }
}

public class cuenta_cheques : cuenta_principal
{
    private decimal saldo;
}

public class cuenta_inversion : cuenta_principal
{
    private decimal saldo;
    public string deposito(decimal cantidad)
    {
        if (cantidad <= 0)
        {
            return "Debe indicar la cantidad";
        }
        else
        {
            saldo = saldo + cantidad + 30;
            return "Deposito realizado en inversion...";
        }
    }
}

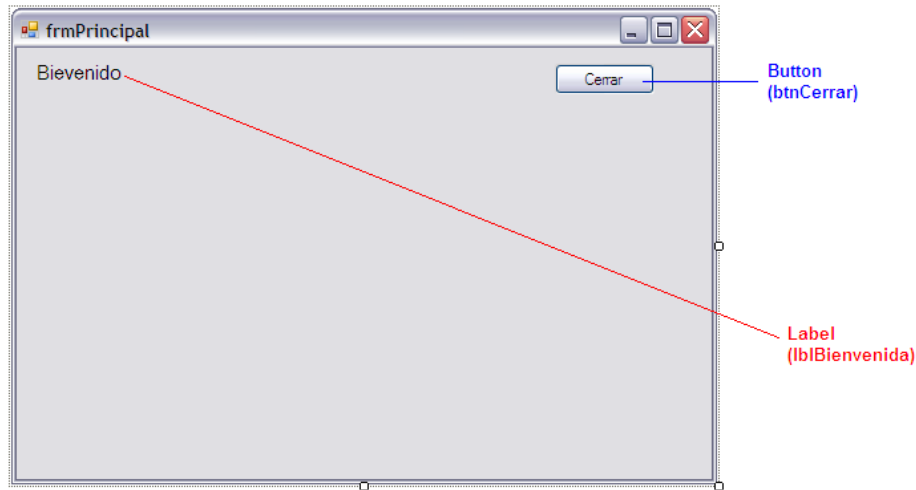
public decimal Saldo
{
    get
    {
        if (conectado == true)
            return saldo;
        else
            return 0;
    }
    set
    {
        if (conectado == true)
            saldo = value;
    }
}
}
```

Anota tus observaciones.



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

- II. Crea una nueva solución de tipo Windows Form o Windows Application y llámala geometría y llama a tu proyecto figuras_geometricas. Llama a tu formulario frmPrincipal y edítalo de la siguiente manera:



Ahora vamos a editar el código del formulario para ello puedes dar doble clic sobre cualquier parte del formulario o con el botón derecho del mouse seleccionar “Ver código”.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace geometria
{
    public partial class frmPrincipal : Form
    {
        public frmPrincipal()
        {
            InitializeComponent();
        }

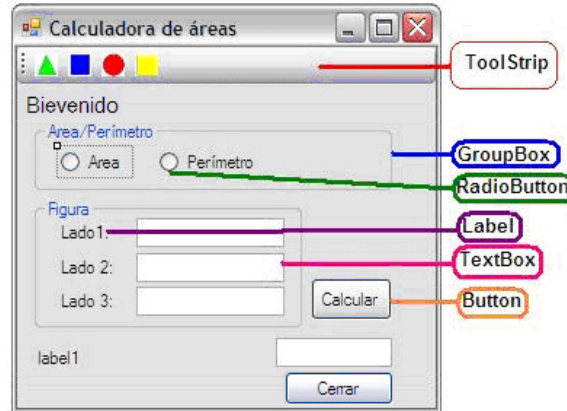
        private void frmPrincipal_Load(object sender, EventArgs e) //Para este método da doble clic sobre el formulario para que se genere todo el evento
        {
            lblBienvenida.Text = "Bienvenido a mi formulario ";
        }

        private void btnCerrar_Click(object sender, EventArgs e) //Da doble clic sobre el botón para que se genere el evento
        {
            Application.Exit(); //Sale de la aplicación y cierra todos los formularios abiertos
        }
    }
}
```

A continuación edite el formulario en modo diseño para que quede de la siguiente manera:

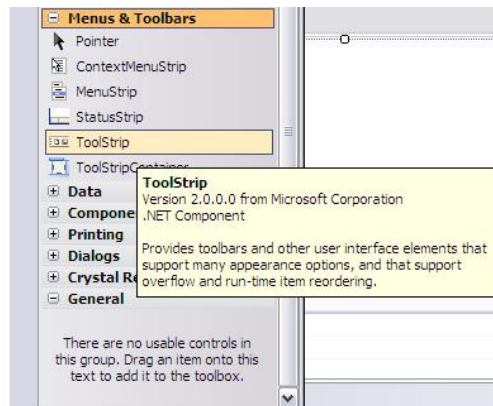


TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

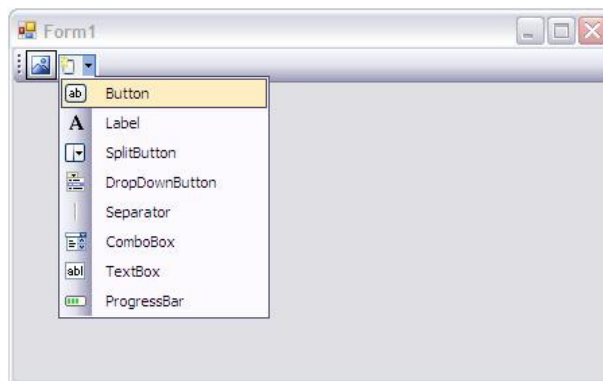


Cambia los nombres de los controles de manera que queden así: rdbArea, rdbPerímetro, lblLado1, lblLado2, lblLado3, txtLado1, txtLado2, txtLado3, btnCalcular, lblResultado, txtResultado

Para adicionar un ToolStrip, seleccione de la caja de herramientas del menu **Menus & Toolbars**



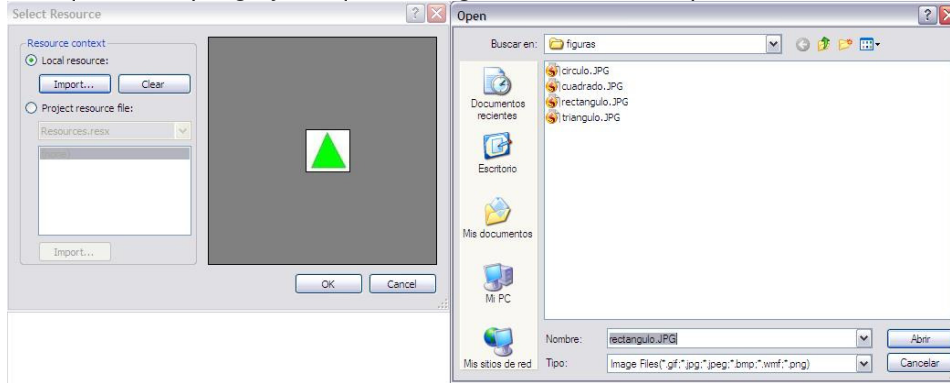
Posteriormente para colocarle los botones posicione sobre el objeto sobre el formulario y de clic en la flecha que le despliega y seleccione **Button**. Realice esta operación cuatro veces.





TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

Para adicionarle un icono a cada botón, seleccione el botón y vaya a la Ventana de Propiedades, busque la propiedad Image y de clic en el botón "...", posteriormente de clic en el botón "Import" de la ventana que se despliega y busque la imagen deseada del explorador.



- El **GroupBox** lo deberás seleccionar de la **Caja de Herramientas > Containers**.
 - Los **RadioButton** se despliegan en el menú **Common Controls**.
 - Para las tres etiquetas Lado1, Lado2 y Lado3, así como para las tres cajas de texto txtLado1, txtLado2, txtLado3 cambia su propiedad Visible a False.
 - Cambia la propiedad Enable del botón btnCalcular a False.
1. Selecciona el proyecto con el botón derecho del mouse y crea una nueva clase llamada figura.cs. Dentro del archivo escribe y comenta el programa del ejemplo de encapsulación, herencia y polimorfismo que se muestra a continuación:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace login_usuario
{
    abstract class figuras
    {
        private int lado1;

        public int Lado1
        {
            set
            {
                if (value < 0)
                    lado1 = 0;
                else
                    lado1 = value;
            }
            get
            {
                return lado1;
            }
        }

        public abstract float area();
        public abstract float perimetro();
    }

    class triangulo : figuras
    {
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
private int lado2;
private int lado3;

public int Lado2
{
    set
    {
        if (value < 0)
            lado2 = 0;
        else
            lado2 = value;
    }
    get
    {
        return lado2;
    }
}

public int Lado3
{
    set
    {
        if (value < 0)
            lado3 = 0;
        else
            lado3 = value;
    }
    get
    {
        return lado3;
    }
}

public triangulo(int tbase,int altura)
{
    Lado1 = tbase;
    Lado2 = altura;
}

public triangulo(int L1, int L2, int L3)
{
    Lado1 = L1;
    Lado2=L2;
    Lado3 = L3;
}

public override float area()
{
    return (Lado1 * Lado2) / 2.0F;
}

public override float perimetro()
{
    return Lado1 + Lado2 + Lado3;
}

}

class circulo : figuras
{
    circulo(int radio)
    {
        Lado1 = radio;
    }

    public override float area()
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
{  
    return 3.1416F * Lado1 * Lado1;  
}  
  
public override float perimetro()  
{  
    return 3.1416F * 2 * Lado1;  
}  
}
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

2. Posteriormente observa el siguiente código y aplícalo a tu formulario:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using validacionesDLL;

namespace login_usuario
{
    public partial class frmPrincipal : Form
    {
        private string sesion_usuario;
        private int tipo_figura;
        validaciones valida_variable = new validaciones();

        public frmPrincipal()
        {
            InitializeComponent();
        }
        public frmPrincipal(string usuario)
        {
            sesion_usuario = usuario;
            InitializeComponent();
        }

        private void frmPrincipal_Load(object sender, EventArgs e)
        {
            lblBienvenida.Text = "Bienvenido " + sesion_usuario;
        }

        private void btnCerrar_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnCalcular_Click(object sender, EventArgs e)
        {
            switch (tipo_figura)
            {
                case 1:
                    if (rdbArea.Checked == true)
                    {
                        triangulo tri = new triangulo(valida_variable.valida_entero(txtLado1.Text),
valida_variable.valida_entero(txtLado2.Text));
                        txtResultado.Text= Convert.ToString(tri.area());
                    }
                    else
                    {
                        triangulo tri = new triangulo(valida_variable.valida_entero(txtLado1.Text),
valida_variable.valida_entero(txtLado2.Text), valida_variable.valida_entero(txtLado3.Text));
                        txtResultado.Text = Convert.ToString(tri.perimetro());
                    }
                    break;
                case 2:
                case 3:
                case 4:
                    break;
            }
        }

        private void rdbPerimetro_CheckedChanged(object sender, EventArgs e)
```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
{
    reinicia_controles();
    if (rdbPerimetro.Checked)
    {
        selecciona_perimetroFigura();
    }
}

private void rdbArea_CheckedChanged(object sender, EventArgs e)
{
    reinicia_controles();
    selecciona_areafigura();
}

private void reinicia_controles()
{
    foreach (Control ctr in this.groupBox2.Controls)
    {
        ctr.Visible = false;
        ctr.Text = "";
    }
    txtResultado.Text = "";
    btnCalcular.Text = "Calcular";
    btnCalcular.Visible = true;
    btnCalcular.Enabled = true;
}

private void selecciona_areafigura()
{
    switch(tipo_figura)
    {
        case 4:
        case 1:
            lblLado1.Text="Base";
            lblLado2.Text="Altura";
            lblLado1.Visible = true;
            lblLado2.Visible = true;
            txtLado1.Visible = true;
            txtLado2.Visible = true;
            break;
        case 2:
            lblLado1.Text = "Lado";
            lblLado1.Visible = true;
            txtLado1.Visible = true;
            break;
        case 3:
            lblLado1.Text="Radio";
            lblLado1.Visible = true;
            txtLado1.Visible = true;
            break;
    }
}

private void selecciona_perimetroFigura()
{
    switch (tipo_figura)
    {
        case 1:
        case 4:
            lblLado1.Text = "Lado1:";
            lblLado2.Text = "Lado2:";
            lblLado3.Text = "Lado3:";
            foreach (Control ctr in this.groupBox2.Controls)
            {

```



TEMA4.PROGRAMACIÓN ORIENTADA A OBJETOS

```
        ctr.Visible = true;
    }
    break;
case 2:
    lblLado1.Text = "Lado";
    lblLado1.Visible = true;
    txtLado1.Visible = true;
    break;
case 3:
    lblLado1.Text = "Radio";
    lblLado1.Visible = true;
    txtLado1.Visible = true;
    break;
}
}

private void toolStripButton1_Click(object sender, EventArgs e)
{
    //Triangulo
    tipo_figura = 1;
    groupBox2.Text = "Triangulo";
    rdbArea.Checked = true;
    reinicia_controles();
    selecciona_areafigura();
}

private void toolStripButton2_Click(object sender, EventArgs e)
{
    //Cuadrado
    tipo_figura = 2;
    groupBox2.Text = "Cuadrado";
    rdbArea.Checked = true;
    reinicia_controles();
    selecciona_areafigura();
}

private void toolStripButton3_Click(object sender, EventArgs e)
{
    //Circulo
    tipo_figura = 3;
    groupBox2.Text = "Circulo";
    rdbArea.Checked = true;
    reinicia_controles();
    selecciona_areafigura();
}

private void toolStripButton4_Click(object sender, EventArgs e)
{
    //Rectangulo
    tipo_figura = 4;
    groupBox2.Text = "Rectangulo";
    rdbArea.Checked = true;
    reinicia_controles();
    selecciona_areafigura();
}
}
}
```

3. Completa el código para que calcule las áreas y perímetros de todas las figuras.