TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN UTN-FRC

TUP

UTN

Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

PROGRAMACIÓN I

Unidad Temática 3: Programación Visual

Material de Estudio

1er Año - 1er Cuatrimestre

2019

V.0.1

Índice

Programación Visual

Programación Visual

La mayor parte de las soluciones necesitan ofrecer al usuario un modo de interactuar con la aplicación. En el ejemplo de aplicación comercial, un sitio Web permite al cliente ver productos y realizar pedidos, y una aplicación basada en el entorno operativo Microsoft Windows permite a los representantes de ventas escribir los datos de los pedidos de los clientes que han telefoneado a la empresa. Las interfaces de usuario se implementan utilizando formularios de Windows Forms, páginas Microsoft ASP.NET, controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos.

El modelo de programación Windows

El modelo de programación propuesto por Windows es totalmente diferente al modelo de ejecución secuencial de DOS. Al ser Windows un entorno multitarea los programas tienen que estar preparados para compartir los recursos de la máquina (procesador, memoria, teclado, ratón,…). Esto supone que Windows ha de disponer de métodos que permitan suspender tareas para activar otras en función de las circunstancias del momento (por ejemplo, por acción del usuario).

Pero por parte de las aplicaciones, este hecho supone que han de cooperar en la compartición de esos recursos. Las aplicaciones Windows se limitan a “esperar” mensajes procedentes del sistema, procesarlos y volver al estado de espera. Este modelo de programación se conoce como “orientado al evento”.

- **Mensaje:** Es una notificación a la aplicación de que ha ocurrido algo de interés y que por lo tanto debe de realizarse alguna acción específica. El origen del mensaje puede ser el usuario (haciendo click con el ratón dentro de una ventana), la propia aplicación (mandándose un mensaje a si misma) o Windows (pidiendo, por ejemplo, que se repinte la ventana tras ocultarse otra que tuviese delante). Dado que la unidad mínima de ejecución en Windows es una ventana, los mensajes van realmente dirigidos a ellas.
- **Ventana y procedimiento de ventana:** En Windows, una aplicación se representa físicamente por su ventana principal (aunque después pueda desplegar diversas ventanas hijas). Cada una de esas ventanas dispone de una serie de propiedades y un código asociado (lo que concuerda con el principio de la POO, en el concepto de objeto). Al código asociado a cada ventana se le denomina procedimiento de ventana. Es una función que recibe los mensajes, los procesa y devuelve el control a Windows para quedar en espera.

Otra de las características específicas de Windows frente a DOS es el uso de recursos por parte de las aplicaciones, como son iconos, menús, mapas de bits, cursores, plantillas de diálogos, etc. Las aplicaciones Windows disponen por tanto de recursos (gráficos generalmente) propios almacenados en lo que se llama el fichero de recursos). El proceso de construcción de programas en Windows incorpora una fase adicional al compilado y enlazado de los módulos objeto y las librerías. Hay un proceso final de compilación y de enlazado (bind) del fichero de recursos.

También se dice que Windows está orientado a eventos. Esto significa que cualquier programa está condicionado por lo eventos que ocurren en el entorno Windows (movimiento del ratón, pulsación de algún botón del ratón, pulsación de una tecla del teclado, desplazamiento o redimensionamiento de una ventana, etc.). Un programa Windows está continuamente sondeando al sistema ante la ocurrencia de cualquier evento y de ocurrir, Windows avisa al programa enviándole un mensaje. Un programa Windows está ocioso la mayor parte del tiempo esperando a que ocurra algún evento.

Entornos de programación Visual

En estos entornos, la labor de un programador se parece más a la de un “ensamblador” de piezas de software que la de un “constructor” de software. Con esto conseguimos mayor rapidez de desarrollo, y sobre todo, mayor simplicidad, ya que sólo tenemos que saber cómo “montar” esas piezas para que nuestro programa funcione. Delphi fue uno de los primeros entornos en aplicar con éxito esta filosofía, y hoy en día son muchos los que apuestan por esta idea, como por ejemplo Microsoft con su lenguaje de programación C#.

Se conocen con el nombre de IDEs, es una herramienta esencial a la hora de desarrollar software. Incluye:

- Editor
- Compilador o intérprete
- Depurador
- Ayuda en línea

Componentes

Para explicar esto debemos conocer, al menos básicamente, la programación orientada a objetos, ya que la programación basada en componentes se apoya sobre ella.

La Programación Orientada a objetos, comúnmente POO, es un modelo de programación que estructura los programas dando más énfasis a los datos que a los procedimientos. En la programación procedural (la de Pascal, C, Basic, etc.) se utilizan funciones que hacen algo con los datos. Sin embargo en la POO (con C++, C#, Java, Object Pascal, etc.) se utilizan objetos (datos) que hacen cosas.

Un componente es una clase de uso específico, lista para usar, que puede ser configurada o utilizada de forma visual, desde el entorno de desarrollo.

La principal diferencia, respecto a una clase normal, es que la mayor parte del trabajo lo podemos hacer de forma visual, con el ratón y ajustando las opciones que se nos ofrece en nuestro entorno.

En la programación orientada a objetos, debemos codificar una serie de operaciones, más o menos laboriosas, para preparar los objetos para su uso. Programar estas operaciones requiere su tiempo, su complejidad y pueden ser origen de errores. Sin embargo, en la programación basada en componentes, todas estas operaciones las realizamos de forma visual, para así poder dedicar la atención a nuestro problema.

Propiedades y eventos

Las propiedades son datos públicos del componente, muy parecidas a los atributos de una clase, aunque se accede a ellas a través de dos métodos: un método para leer su valor, y otro para modificarlo. Existen propiedades de sólo lectura, en las que podemos consultar pero no modificar su valor, y propiedades de sólo escritura. Por ejemplo, las propiedades “Alto” (Width) y “Ancho” (Height) de un botón permiten que un programador pueda cambiar las dimensiones del componente. Cuando el programador cambia alguna de ellas, el componente debe redibujarse en la pantalla, para mostrar los nuevos cambios.

Los eventos son funciones del componente, que se ejecutarán automáticamente cuando ocurra “algo importante”. Un evento es cualquier suceso que puede ocurrirle a un componente (movimiento del ratón, pulsación de algún botón del ratón, pulsación de una tecla del teclado, desplazamiento o redimensionamiento de una ventana, etc.) que pueden condicionar el comportamiento y apariencia del programa. Un programador puede poner el código que quiera en el evento, para así poder hacer una acción cuando ese “algo importante” ocurra. Cada componente poseerá una serie de eventos que puede recibir o generar. Se pueden tratar los eventos de un componente que necesitemos, y dejar que los demás sean tratados por defecto.

Los métodos son funciones, que permiten realizar acciones. Normalmente, se utilizan métodos para dos tareas distintas: realizar algo importante (como repintar en pantalla, cambiar el foco o algo así), o para establecer el valor de los atributos internos, haciendo algún tipo de comprobación previa. Como las propiedades pueden ser leídas o escritas a través de métodos, a veces es equivalente la llamada a un método y el cambio de una propiedad. Por ejemplo, para mostrar un botón tenemos dos posibilidades.

Y por último, los atributos. Tienen la misma misión que en programación orientada a objetos, es decir: almacenar datos internos al objeto (o clase). En el maravilloso mundo de los componentes, los atributos siempre son internos y de uso privado, y debemos utilizar las propiedades para que un programador pueda leer o establecer un dato.

Sabiendo esto, podemos decir que la principal “misión” del programador de componentes es definir un grupo de propiedades, métodos y eventos para que otros programadores puedan utilizar el componente de forma sencilla y rápida.

Componentes visuales y no visuales

Se pueden establecer muchas clasificaciones para los componentes. Una de ellas es la de visuales o controles, frente a no visuales.

Los componentes visuales son aquellos que, al utilizarlos, muestran algún elemento (o dibujo) en la pantalla y es el usuario de nuestros programas el que interactúa con él. El componente es el principal responsable de dibujar en la pantalla lo que sea oportuno, dependiendo de su estado, del valor de sus atributos, etc. Hay muchos componentes de este tipo, como pueden ser los botones, etiquetas de texto, formas, etc.

Los componentes no visuales son aquellos que no aparecen en la ventana, y se insertan en un formulario para que el programador los utilice. Son más fáciles de programar que los componentes visuales, ya que no tienen ningún tipo de interfaz gráfico. Ejemplos de componentes no visuales podrían ser un temporizador, una tabla o una conexión a base de datos.

Componentes básicos

Formularios o Ventanas

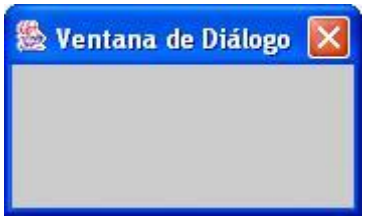
En la mayoría de aplicaciones, el formulario es la parte de la interfaz que permite que el usuario pueda introducir datos. Su diseño es crítico dado que una mala concepción de él puede convertirlo en una barrera para la interacción e inducir al usuario a cometer errores potenciando la frustración en el desempeño de su tarea.

Actualmente los formularios, al menos como estilo de interacción, los podemos encontrar en las aplicaciones más diversas. Utilizamos formularios cuando en una página web introducimos nuestros datos y nuestro número de tarjeta de crédito, pero también cuando en un cajero automático indicamos la cantidad de dinero que deseamos reintegrar o cuando en un editor seleccionamos las opciones de formato del texto.

La siguiente lista de directrices de diseño son consejos para realizar un buen diseño de formularios:

- Dar un título al formulario que exprese claramente su función.
- Las instrucciones han de ser breves y comprensibles.
- Hacer grupos lógicos de campos y separarlos con blancos. Por ejemplo: nombre, primer apellido y segundo apellido es un grupo lógico.
- Aspecto ordenado alineando los campos y las etiquetas.
- Las etiquetas de los campos deben usar terminología familiar.
- Ser consistente en el uso de los términos, es decir, usar siempre las mismas palabras para los mismos conceptos.
- El tamaño visible del campo debe corresponderse con la longitud del contenido que ha de introducir el usuario.
- Permitir el movimiento del cursor por medio del teclado y no solo con el mouse.
- Permitir que el usuario pueda corregir con libertad los caracteres que ha introducido en los campos.
- En donde sea posible, impedir que el usuario introduzca valores incorrectos. Por ejemplo, impedir que introduzca caracteres alfabéticos en campos que solo admiten valores numéricos.
- Si introduce valores incorrectos, indicar en un mensaje cuales son los correctos.
- Avisar cuanto antes al usuario si ha introducido valores incorrectos. Si es posible, no esperar a que haya rellenado el formulario totalmente. Esto no debe tomarse tan literal, ya que muchas veces las validaciones inmediatas pueden ser vividas por el usuario como un control excesivo y frustrante.
- Marcar claramente los campos obligatorios o los opcionales.
- Si es posible, colocar explicaciones o la lista de los valores válidos al lado de los campos.
- Dejar clara la acción que debe hacer el usuario al terminar de rellenar el formulario.

Diálogos

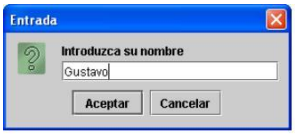


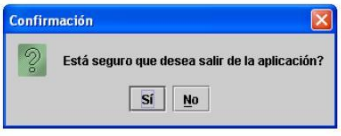
Una ventana de diálogo es muy parecida a una ventana con la diferencia de que no puede ser minimizada ni maximizada. Se utiliza cuando se requiere una respuesta o interacción precisa del usuario, de allí su nombre de ventana de “diálogo”.

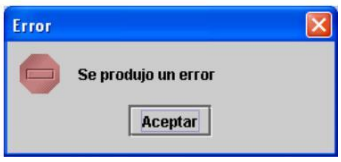
Ofrece el mismo comportamiento que una ventana, agregando la capacidad de tomar el control del foco para una aplicación hasta ser cerrada. Cuando una ventana de diálogo posee esta capacidad se dice que es una ventana modal (se convierte en una ventana exclusiva en el sentido de que deja inhabilitada cualquier operación con el resto de la aplicación).

Diálogos con opción

Son diálogos prefabricados, que tienen los componentes básicos para permitir, por ejemplo, mostrar un mensaje, o pedir algún valor o confirmación al usuario.







Controles

 **MainMenu:** Barra de menú (que actúa como menú principal).

 **PopupMenu**: Menú desplegable (también llamado menú contextual) que aparecen cuando se presiona con el botón derecho del ratón.

 **Label:** Muestra texto que el usuario no puede seleccionar ni manipular. Se usa para mostrar textos de título, encabezamientos, o incluso para mostrar resultados.

 **Edit:** Muestra un área de edición de texto en la que el usuario puede introducir y modificar una única línea de texto.

 **Memo:** Muestra un área de edición de texto en la que el usuario puede introducir y modificar múltiples líneas de texto.

 **Button:** Crea un botón que el usuario puede presionar para efectuar acciones.

 **CheckBox:** Presenta una opción binaria (Si/No - Verdad/Falso) de manera que cuando se selecciona este control, se permuta entre ambos valores. Este control puede emplearse para crear un grupo de estos controles que representen elecciones que no sean mutuamente exclusivas (al contrario que los RadioButton, por lo que el usuario puede seleccionar más de una opción en un grupo.

 **RadioButton:** Presenta una opción binaria (Si/No - Verdad/Falso) de manera que cuando se selecciona este control, se permuta entre ambos valores. Este control puede emplearse para crear un grupo de estos controles que representen elecciones mutuamente exclusivas (al contrario que los CheckBox, por lo que el usuario puede seleccionar sólo una en un grupo.

 **ScrollBar:** Proporciona una forma cómoda de modificar el área visible de un formulario o de una lista. También puede usarse para desplazarse en un rango amplio de valores por incrementos prefijados.

 **GroupBox:** Contenedor para agrupar opciones relacionadas en un formulario.

 **RadioGroup:** Contenedor que crea un grupo de componentes RadioButton en un formulario.

 **Panel:** Contenedor que puede contener otros componentes en un formulario. Se usa para crear barras de herra-mientas y líneas de estado. Los componentes que contiene están asociados al panel.

 **ListBox:** Muestra una lista de elecciones que está acompañada de una barra de scroll.

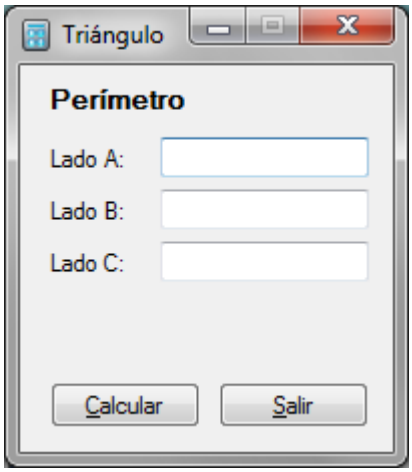
 **ComboBox:** Muestra una lista de elecciones. Es un control que combina aspectos de un componente ListBox y de un componente Edit: el usuario puede introducir datos en el área de edición o seleccionar en el área de lista.

Problemas modelo

Problema 1.1: Triángulo con constructores:

Calcular el perímetro de un triángulo en una interfaz gráfica.

Solución



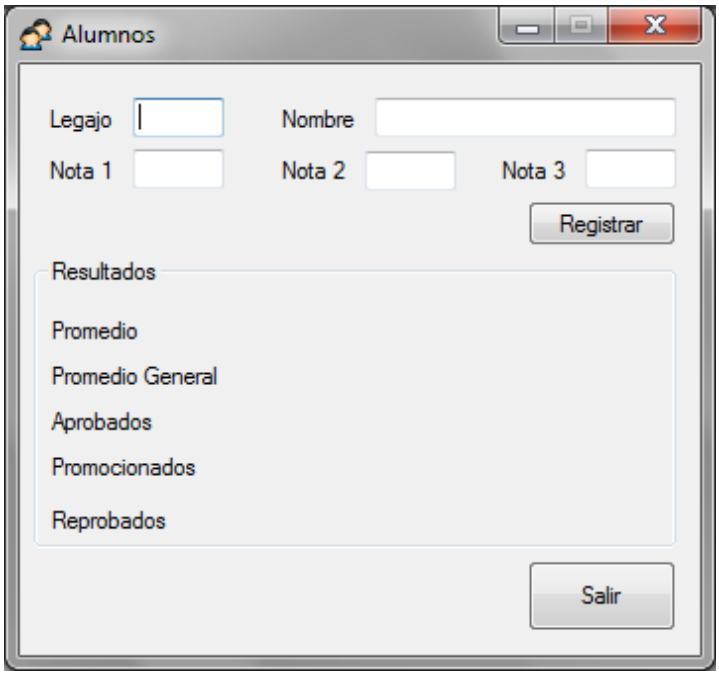

```
namespace Triang
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            lblPerimetro.Visible = false;
        }
        //Clase Triangulo:
        public class Triangulo
        {
            double ladoA, ladoB, ladoC;
            //Constructores:
            public Triangulo()
            {
                ladoA = ladoB = ladoC = 0;
            }
            public Triangulo(float a, float b, float c)
            {
                ladoA = a;
                ladoB = b;
                ladoC = c;
          }
          //Propiedades:
          public double pLadoA
          {
              set { ladoA =   value; }
              get { return ladoA; }
          }
          public double pLadoB
          {
              set { ladoB = value; }
              get { return ladoB; }
          }
```

```
        public double pLadoC
        {
            set { ladoC = value; }
            get { return ladoC; }
        }
        //Métodos:
        public double perimetro()
        {
            return ladoA + ladoB + ladoC;
        }
    }//Fin Clase Triángulo
    //Botón Calcular:
    private void button1_Click(object sender, EventArgs e)
    {
        Triangulo triang = new Triangulo();
        triang.pLadoA = Convert.ToDouble(txtA.Text);
        triang.pLadoB = Convert.ToDouble(txtB.Text);
        triang.pLadoC = Convert.ToDouble(txtC.Text);
        lblPerimetro.Text = "Perímetro: " + triang.perimetro().ToString();
        lblPerimetro.Visible = true;
        txtA.Text = "";
        txtB.Text = "";
        txtC.Text = "";
        txtA.Focus();
    }//Fin Botón Calcular
    //Botón Salir:
    private void button2_Click(object sender, EventArgs e)
    {
        Close();
    }//Fin Botón Salir
  }
}
```

Problema 1.2: Alumnos:

Dados el legajo, el nombre y tres notas de alumnos, calcular el promedio del alumno, el promedio general, la cantidad de aprobados, promocionados y reprobados.

Solución



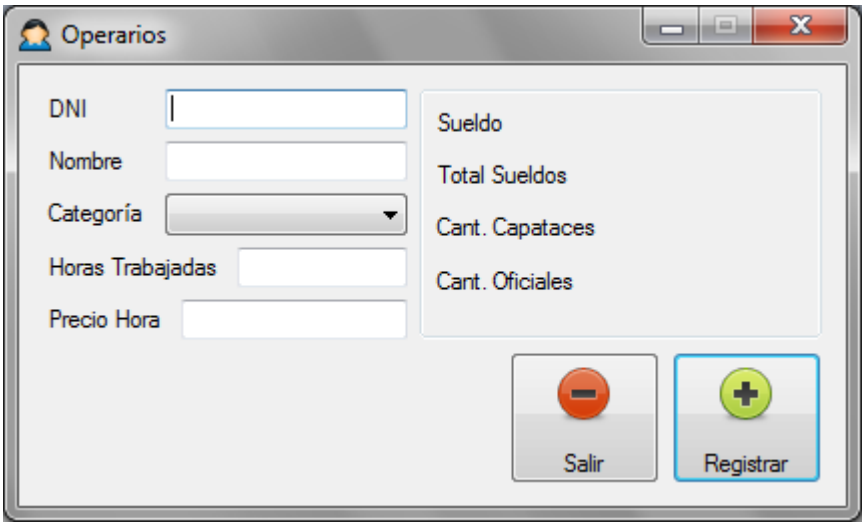
```
/*********************************************************************/
/* Alumno.cs */
/*********************************************************************/
namespace ProgrI_Ej008
{
    class Alumno
    {
        int legajo;
        string nombre;
        double nota1, nota2, nota3;
        //Constructores:
        public Alumno()
        {
            legajo = 0;
            nombre = "";
            nota1 = nota2 = nota3 = 0;
        }
        public Alumno(int leg, string nomb, double n1, double n2, double n3)
        {
           legajo = leg;
           nombre = nomb;
           nota1 = n1;
           nota2 = n2;
           nota3 = n3;
       }
       //Propiedades:
       public int pLegajo
       {
            set { legajo = value; }
            get { return legajo; }
       }
       public string pNombre
       {
            set { nombre = value; }
            get { return nombre; }
       }
       public double pNota1
       {
            set { nota1 = value; }
            get { return nota1; }
       }
       public double pNota2
       {
            set { nota2 = value; }
            get { return nota2; }
       }
       public double pNota3
       {
            set { nota3 = value; }
            get { return nota3; }
       }
       //Métodos:
       public double calcularPromedio()
       {
            return (nota1 + nota2 + nota3) / 3;
       }
    }
}
```

```
/*****************************************************************/

/* Form1.cs*/
/*****************************************************************/
namespace ProgrI_Ej008

{
    public partial class frmAlumnos : Form
    {
        int cantidadAlumnos;
        double sumaPromedios;
        int aprobados, reprobados, promocionados;
        public frmAlumnos()
        {
            InitializeComponent();
            cantidadAlumnos = 0;
            sumaPromedios = 0;
            aprobados = 0;
            reprobados = 0;
            promocionados = 0;
        }
        private void btnSalir_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            Alumno alumn = new Alumno();
            alumn.pLegajo = Convert.ToInt32(tbxLegajo.Text); alumn.pNombre =
            tbxNombre.Text;
            alumn.pNota1 = Convert.ToDouble(tbxNota1.Text);
            alumn.pNota2 = Convert.ToDouble(tbxNota2.Text);
            alumn.pNota3 = Convert.ToDouble(tbxNota3.Text);
            //Calculo y muestro promedio del alumno:
            double promedio = alumn.calcularPromedio();
            lblPromedio.Text = "Promedio: " + Math.Round(promedio, 2).ToString();
            //Cuento la cantidad de alumnos ingresados:
            cantidadAlumnos++;
            //calculo y muestro promedio acumulado:
            sumaPromedios += promedio;
            lblPromedioGeneral.Text = "Promedio General: " +
            Math.Round((sumaPromedios / cantidadAlumnos), 2).ToString();
            //Determino y muestro la condición del alumno:
            if (promedio >= 4)
                aprobados++;
            else
                reprobados++;
            if (promedio >= 8 &&alumn.pNota1 >= 4 &&alumn.pNota2 >= 4 &&alumn.pNota3 >=4)
                promocionados++;
            lblAprobados.Text = "Aprobados: " + aprobados.ToString();
            lblPromocionados.Text = "Promocionados: " + promocionados.ToString();
            lblReprobados.Text = "Reprobados: " + reprobados.ToString();
            limpiarCampos();
            tbxLegajo.Focus();
        }
        void limpiarCampos()
        {
            foreach (Control c in this.Controls)
            {
                        if (c isTextBox)
                            c.Text = "";
            }
        }
    }
}
```

Problema 1.3: Operarios:

Dados el DNI, nombre, categoría (capataz y oficial), horas trabajadas y precio de la hora de operarios, calcular el sueldo, el total de sueldos, la cantidad de capataces y la cantidad de oficiales.

Solución




```
/*************************************************************************/
/*                                Operario.cs                            */
/*************************************************************************/
namespace ProgrI_Ej011
{
    class Operario
    {
        //Atributos:
        int dni;
        string nombre;
        int horasTrabajadas;
        int idCategoria;
        double precioPorHora;
        //Constructores:
        public Operario()
        {
            dni = 0;
            nombre = "";
            horasTrabajadas = 0;
            idCategoria = 1;
            precioPorHora = 0;
        }
        public Operario(int doc, string nom, int horas, int idCat, double precio)
        {

            dni = doc;
            nombre = nom;
            horasTrabajadas = horas;
            idCategoria = idCat;
            precioPorHora = precio;

        }
        //Propiedades:
        public int pDni
        {
            set { dni = value; }
            get { return dni; }
        }
        public string pNombre
        {
            set { nombre = value; }
```

```
            get { return nombre; }
        }
        public int pHorasTrabajadas
        {
            set { horasTrabajadas = value; }
            get { return horasTrabajadas; }
        }
        public int pIdCategoria
        {
            set { idCategoria = value; }
            get { return idCategoria; }
        }
        public double pPrecioPorHora
        {
            set { precioPorHora = value; }
            get { return precioPorHora; }
        }
        //Métodos:
        public string toString()
        {
            string cat, texto;
            switch (idCategoria)
            {
                case 1: cat = "Oficial"; break;
                case 2: cat = "Capataz"; break;
                default: cat = "Otra"; break;
            }
            texto = "DNI: " + dni.ToString()
              +     "\nNombre: " + nombre
              +     "\nCategoría: "+ cat
              +     "\nHoras Trabajadas: " + horasTrabajadas.ToString()
            +       "\nPrecio Hora: " + precioPorHora.ToString(); return
            texto;
        }
        public double calcularSueldo()
        {
                  return precioPorHora * horasTrabajadas;
             }
    }
}

/*************************************************************************/
/* Form1.cs */
/*************************************************************************/
namespace ProgrI_Ej011
{
    public partial class frmOperario : Form
    {
        int contOficiales;
        int contCapataces;
        double totalSueldos;
        public frmOperario()
        {
            InitializeComponent();
            contOficiales = 0;
            contCapataces = 0;
            totalSueldos = 0;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            DialogResult opcion = MessageBox.Show("¿Está seguro que desea salir?", "Salir",
                MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if(opcion==DialogResult.Yes)
                Close();
        }

        private void button1_Click(object sender, EventArgs e)
        {
```

```
            if (validarDatos())
            {
                Operario oper = new Operario();
                oper.pDni = Convert.ToInt32(txtDni.Text);
                oper.pNombre = txtNombre.Text;
                oper.pIdCategoria = cmbCategoria.SelectedIndex + 1;
                oper.pHorasTrabajadas = Convert.ToInt32(txtHorasTrabajadas.Text);
                oper.pPrecioPorHora = Convert.ToDouble(txtPrecioPorHora.Text);
                switch (oper.pIdCategoria)
                {
                    case 1: contOficiales++; break;
                    case 2: contCapataces++; break;
                    default: break;
                }
                totalSueldos += oper.calcularSueldo(); lblSueldo.Text = "sueldo: " +
            oper.calcularSueldo(); lblCapataces.Text = "capataces: " + contCapataces.ToString();
            lblOficiales.Text = "oficiales: " + contOficiales.ToString(); lblTotalSueldos.Text = "total
            sueldos: " + totalSueldos.ToString(); MessageBox.Show("Operario: \n" + oper.toString(),
            "resumen",
                    MessageBoxButtons.OK, MessageBoxIcon.Information); limpiarDatos();
                txtDni.Focus();
            }
        }
        public bool validarDatos()
        {
            foreach(Control c in this.Controls)
            {
                if (c isTextBox&&string.IsNullOrEmpty(c.Text))
                {
                    MessageBox.Show("Complete con los datos faltantes", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                    c.Focus();
                    return false;
                }
            }
            return true;
        }
        public void limpiarDatos()
        {
            foreach (Control c in this.Controls)
            {
                        if (c isTextBox)
                            c.Text = "";
            }
        }
    }
}
```

Bibliografía

- P.Bishop – Fundamentos de Informática – Anaya 1992.
- G.Brookshear – Computer Sciense: An Overview – Benjamin/Cummings. 1994.
- L. Joyanes – Problemas de Metodología de la Programación – McGraw Hill. 1990.
- L. Joyanes - Fundamentos de Programación: Algoritmos y Estructura de Datos – McGraw Hill. 1993.
- P. de Miguel – Fundamentos de los Computadores – Paraninfo. 1994.
- P. Norton – Introducción a la Computación – McGraw Hill. 1995.
- A. Prieto, A. Lloris y J.C. Torres – Introducción a la Informática – McGraw Hill. 1989.
- A. Tucker, W. Bradley, R. Cupper y D. Garnick – Fundamentos de Informática (Lógica, resolución de problemas, programas y computadoras). – McGraw Hill. 1994.