Universidad Tecnológica Nacional Facultad Regional Avellaneda		
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos		
Apellido y nombre: Kevin Martin Plucci		Fecha:25/07/2021
Trabajo Practico	Número Cuatro y número Tres.	Nota:
División:	2A	Firma: Kmp.

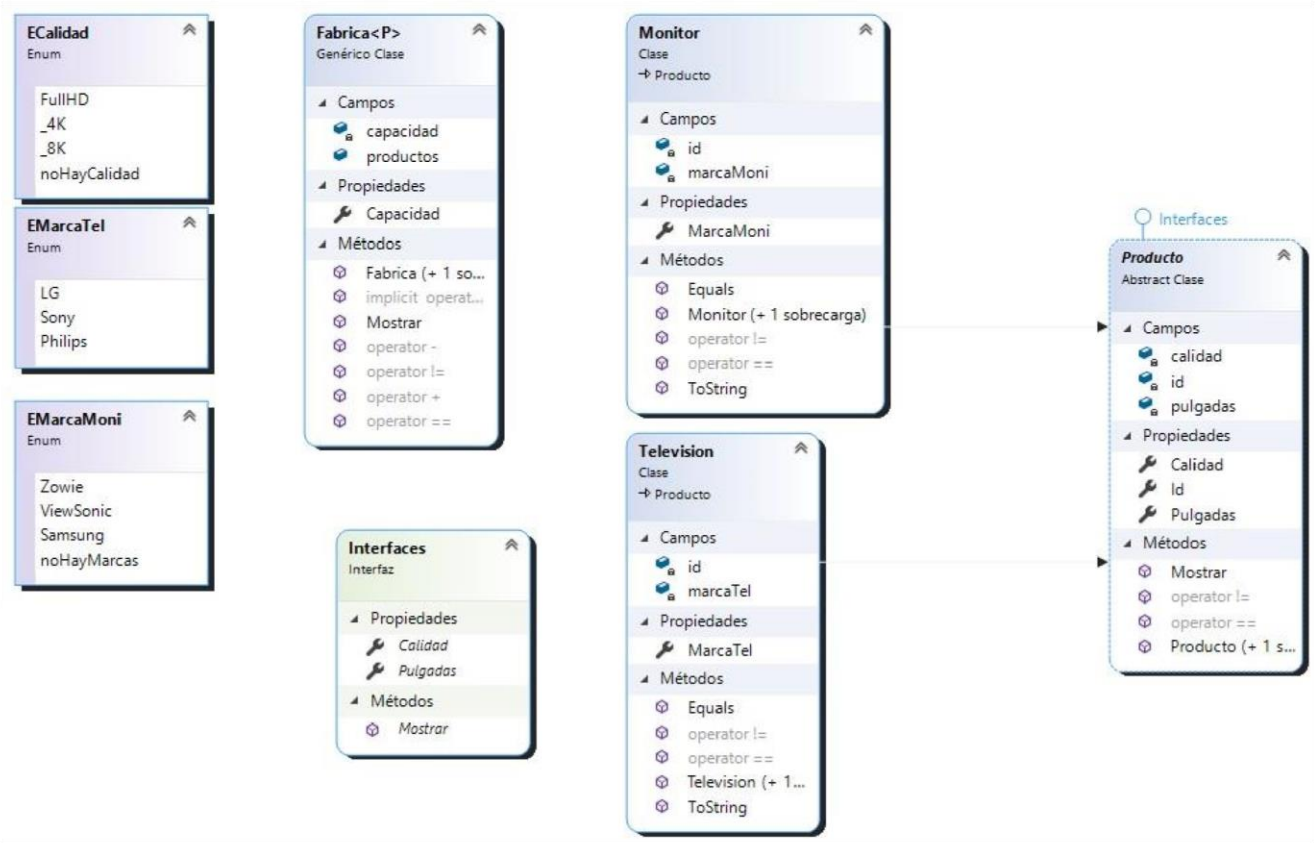


Diagrama de Clases:

Introducción

Este trabajo lleva a cabo a una fábrica de Monitores e Televisores. En la cual heredan de una clase abstracta llamada Producto.

Windows Forms

Consta en un menú de fabricación para poder elegir Monitores y

Televisión. En la cual, cuando seleccionaremos cualquiera de las dos, procederemos a dicha fabricación (agregarle tanto como la marca que usaremos, su calidad de video e su dimensión ósea pulgadas).

Explicación breve del Trabajo Practico

Consta de una lista genérica que almacena sin agotar stock, que pueden ser tanto televisor o monitores.

También, se descarga un archivo de la grilla (o factura) de dichos productos fabricados (interacción con el Forms, cuando fabricamos los productos) cada uno con sus caracterteristicas (su calidad, su pulgadas y su marca). También tendrá su método de guardado en cualquier lugar de su máquina e agregarle el nombre del archivo.

Excepciones

Dentro de la fábrica encontraremos el operador == (para evitar dicho problemas de comparación), en el operador + y en el operador menos (evitamos dichos conflictos a la hora de remover, agregar o igualar dichos productos).

En el Forms lo encontraremos en dichas serialización de cada producto (FormMonitor y FormTelevisor).

Esto evita que el programa no se interrumpa, aunque tenga problemas, pero que siga recorriendo el programa.

Test Unitarios

Se crea el proyecto Test Unitario, con su respectiva clase. Con esto corroboraremos la sobrecarga de operadores de distintos productos e iguales productos.

Tipos Genéricos

Tenemos una sola clase que sería Fábrica que contiene una lista genérica.

Interfaces

Solo una interfaz, que contiene 3 propiedades que comparte la clase Producto, (en ellas tenemos pulgadas y calidades). Y el método mostrar.

Archivos y Serializacion

Explicado anteriormente, lo tendremos en el Form tanto como el de Monitor como el de Televisor para serializar objetos genéricos de la Fábrica y de ahí sale una socialización de lo nombrado anteriormente. Se termina de producir los productos, se guardan en la ruta que el usuario quiera y con el nombre que quiera, al terminar se podrá cargar el archivo sin ningún problema (cabe mencionar que se guardara en .txt).Son dos .txt, uno tendrá todos los televisores/monitores fabricados(monitor por un lado y en televisores por otro lado) y el otro es un esquema de cómo es la grilla del DataTable (me refiero a que aparece lo que aparece en cada columna id/calidad/pulgadas/marcas, un esquema).

Correcciones del TP Nº 3 y TP Nº 4

Piucci, Kevin	SI	SI	D	SI	Métodos de test sin nombre TestMethod1 Cumplir con las reglas de estilo: fabrica_ Comentar código, en el caso de los test poner que deberían hacer (sin nombre y sin documentar, se hace dificil) Testear excepciones, archivos, cosas muy permeables a errores Agregar funcionalidad. Solo se puede cargar info y guardar, no levanta dicho archivo, al cerrar el form se pierde todo, no se ejecuta ningún proceso. Flojo. Al cerrar el form no cierra la aplicación. Puedo agregar muchas veces el mismo producto, no hay excepción como nombra el documento adjunto.

"Falla uno de los 2 test unitarios creados
Que aparezca _4K para cargar queda mal, el guión no es parte de la nomenclatura más allá de que sea como creaste el enumerado
Solo doy de alta 2 productos... No hay fabricación de por medio. Ya se había marcado esto en el TP3."

Métodos de test sin nombre TestMethod1

Cada TestMethod fue cambiado por su nombre de proceso que hace cada uno (Igualdad y distinto).

Cumplir con las reglas de estilo: fabrica_

Agregamos el Where y ordene más el Summary. Con su regla de estilo.

Comentar código, en el caso de los test poner que deberían hacer (sin nombre y sin documentar, se hace difícil)

Se comentó todo el código y acá haremos una explicación (e documentaremos) los test faltantes que tuvimos problemas con el primer punto a corregir.

```
/// <summary>
/// Test Unitario de Sobrecarga del operador != productos
/// </summary>
[TestMethod]
0 referencias
public void VerificarDistinto()
{
    int fabrica_ = 4;
    int idMonitor = 1;
    int idTelevisor = 1;
    int pulgadasMonitor_ = 1;
    int pulgadasTelevisor_ = 1;
    ECalidad unitario = ECalidad._4K;
    ECalidad secundario = ECalidad.FullHD;
    EMarcaTel marcaT = EMarcaTel.Philips;
    EMarcaMoni marcaM = EMarcaMoni.ViewSonic;

    Fabrica<Television> f1 = new Fabrica<Television>(fabrica_);
    Fabrica<Monitor> f2 = new Fabrica<Monitor>(fabrica_);
    Television t1 = new Television(idTelevisor,pulgadasTelevisor_, unitario, marcaT);
    Monitor m1 = new Monitor(idMonitor, pulgadasMonitor_, secundario, marcaM);

    //YA INGRESADO
    //f1 += c1;
    f1 += t1;

    //YA INGRESADO
    f2 += m1;

    Assert.AreEqual(t1.Id, idTelevisor);
    Assert.AreEqual(m1.Id, idMonitor);
    Assert.AreEqual(t1.Pulgadas, pulgadasTelevisor_);
    Assert.AreEqual(m1.Pulgadas, pulgadasMonitor_);
    Assert.AreEqual(t1.Calidad, unitario);
    Assert.AreEqual(m1.Calidad, secundario);
    Assert.AreEqual(t1.MarcaTel, marcaT);
    Assert.AreEqual(m1.MarcaMoni, marcaM);

    Assert.IsTrue(f1 != m1 && f2 != t1);
```

Comprobamos la sobrecarga de == de los productos.

Y el otro Test Unitario para probar los distintos productos

```
/// <summary>
/// Test Unitario de Sobrecarga del operador != productos
/// </summary>
[TestMethod]
0 referencias
public void VerificarDistinto()
{
    int fabrica_ = 4;
    int idMonitor = 1;
    int idTelevisor = 1;
    int pulgadasMonitor_ = 1;
    int pulgadasTelevisor_ = 1;
    ECalidad unitario = ECalidad._4K;
    ECalidad secundario = ECalidad.FullHD;
    EMarcaTel marcaT = EMarcaTel.Philips;
    EMarcaMoni marcaM = EMarcaMoni.ViewSonic;

    Fabrica<Television> f1 = new Fabrica<Television>(fabrica_);
    Fabrica<Monitor> f2 = new Fabrica<Monitor>(fabrica_);
    Television t1 = new Television(idTelevisor,pulgadasTelevisor_, unitario, marcaT);
    Monitor m1 = new Monitor(idMonitor, pulgadasMonitor_, secundario, marcaM);

    //YA INGRESADO
    //f1 += c1;
    f1 += t1;

    //YA INGRESADO
    f2 += m1;

    Assert.AreEqual(t1.Id, idTelevisor);
    Assert.AreEqual(m1.Id, idMonitor);
    Assert.AreEqual(t1.Pulgadas, pulgadasTelevisor_);
    Assert.AreEqual(m1.Pulgadas, pulgadasMonitor_);
    Assert.AreEqual(t1.Calidad, unitario);
    Assert.AreEqual(m1.Calidad, secundario);
    Assert.AreEqual(t1.MarcaTel, marcaT);
    Assert.AreEqual(m1.MarcaMoni, marcaM);

    Assert.IsTrue(f1 != m1 && f2 != t1);
```

(La corrección de cambiar el nombre de los test unitarios y de la falla de uno de los test unitarios se mostrara en la próxima imagen con una próxima corrección)

Testear excepciones, archivos, cosas muy permeables a errores

Se le agrego excepciones a los temas antiguos y a los incorporados. Un ejemplo: (este ejemplo aplica a lo que me remarco en una correcion de una falla del test unitario, se mostrara la correcion que le agregamos el try catch y la segunda correcion por la falla que nos generaba el test unitario, precisamente problema en la logica de la igualdad de monitores).

Antes arreglado con el try catch pero generando problemas en el Test Unitario

```
#region Sobrecargas de Operadores

/// <summary>
/// Sobrecarga del operador igual entre dos monitores
/// </summary>
/// <param name="m1">Primer monitor</param>
/// <param name="m2">Segundo monitor a ser comparado con el monitor</param>
/// <returns>retorna si los monitores de los productos son iguales</returns>
3 referencias
public static bool operator ==(Monitor m1, Monitor m2)
{
    bool rta = false;
    m1 = new Monitor();
    m2 = new Monitor();
    try
    {
        if (!((Producto)m1 == (Producto)m2 && m1.MarcaMoni == m2.MarcaMoni))
        {
            rta = true;
        }
    }
    catch(Exception ex) { }
    return rta;
}
```

Corrección actual a la lógica.

```
public static bool operator ==(Monitor m1, Monitor m2)
{
    bool rta = false;
    try
    {
        if ((Producto)m1 == (Producto)m2 && m1.MarcaMoni == m2.MarcaMoni)
        {
            rta = true;
        }
    }
    catch(Exception ex) { }
    return rta;
}
```

Corrección mencionada:

Dávila: “Falla uno de los test unitarios”.

A la sobrecarga de operador == de Monitor le agregamos una Excepciones para evitar errores (con su código comentado como los puntos anteriores).

Otro ejemplo con un tema más actual en Archivos:

```
1 referencia
private void GuardarDataTable()
{
    try
    {
        this.dt.WriteXmlSchema(xml_Monitores_Schema);

        this.dt.WriteXml(xml_Monitores);

        System.Threading.Thread.Sleep(2000);
        MessageBox.Show("Se han guardado el esquema y los datos del DataTable!!!");

    }
    catch
    {
        MessageBox.Show("Error al guardar el DataTable. ",
                        "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Y entre otros más…

Agregar funcionalidad. Solo se puede cargar info y guardar, no levanta dicho archivo, al cerrar el form se pierde todo, no se ejecuta ningún proceso.

```
private void btnVerLog_Click(object sender, EventArgs e)
{
    DialogResult rta = openFileDialog1.ShowDialog();//Reemplazar por la llamada al método correspondiente del OpenFileDialog

    if (rta == DialogResult.OK)
    {
        //leer el archivo seleccionado por el cliente y mostrarlo en txtVisorTickets

        System.Threading.Thread.Sleep(2000);
        string path = openFileDialog1.FileName;
        string texto;
        StreamReader f = new StreamReader(path);

        texto = f.ReadToEnd();

        this.txtVisorTickets.Text = texto;
    }
}
```

Agregue una visualización y carga de los tickets tanto monitor como televisor (tampoco se pierde la información, ya que uso el DataGreed para incorporar los datos que coloco, los cargo a la base de datos y de ahí los serializo en el mismo DataGreed.

Flojo.

Se documentó todo, se comentó todo, se agregó nuevas funcionalidades al forms, se arregló las excepciones, etc.

Al cerrar el form no cierra la aplicación.

Ya cierra y guarda los datos al cerrar el forms.

```
1 referencia
private void VolverM_Click(object sender, EventArgs e)
{
    //FormInicio frmInicio = new FormInicio();

    //this.Hide();

    //frmInicio.Show();

    this.Close();
}
```

Puedo agregar muchas veces el mismo producto, no hay excepción como nombra el documento adjunto.

Error de concepto a la hora de documentar, es ilimitado su stock.

Temas incorporados del TP Nº4

Hilos

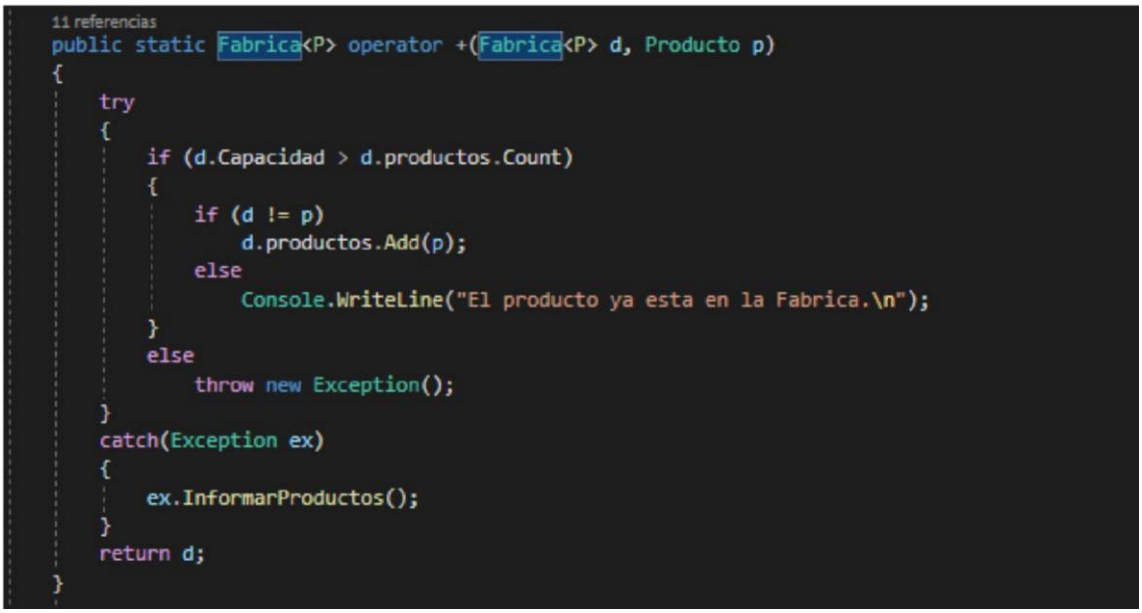
```
System.Threading.Thread.Sleep(2000);
```

Se le agrego este delay para evitar problemas a la hora de cargar un archivo o un Message Box Ejemplo:

```
{
    this.dt.WriteXmlSchema(xml_Monitores_Schema);

    this.dt.WriteXml(xml_Monitores);

    System.Threading.Thread.Sleep(2000);
    MessageBox.Show("Se han guardado el esquema y los datos del DataTable!!!");

}
```

Método de extensión:

Uno solo hay y es solo para evitar la saturación de productos.



```
11 referencias
public static Fabrica<P> operator +(Fabrica<P> d, Producto p)
{
    try
    {
        if (d.Capacidad > d.productos.Count)
        {
            if (d != p)
                d.productos.Add(p);
            else
                Console.WriteLine("El producto ya esta en la Fabrica.\n");
        }
        else
            throw new Exception();
    }
    catch(Exception ex)
    {
        ex.InformarProductos();
    }
    return d;
}
```

Base de datos:

Se establece una carga de datos hacia la base de datos que el empleado fabrica a la hora de hacer tantos Monitores o Televisores, cada uno con su ID de producto, Marca, Calidad de Video y Pulgadas, todo en un table ordenado ascendentemente por la ID:

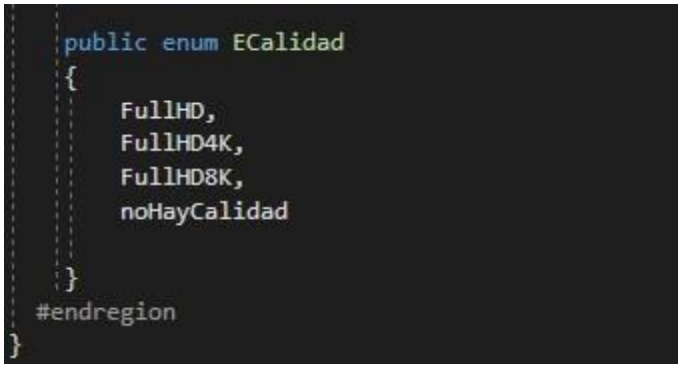
```
1 referencia
private void guardar_Click(object sender, EventArgs e)
{
    List<Monitor> auxMonitor = new List<Monitor>();

    try
    {
        string consultaP = "SELECT * FROM Monitor WHERE id = @id";

        cn.Open();


        SqlCommand comandP = new SqlCommand(consultaP, cn);


        GuardarDataTable();

        //comandP.Parameters.AddWithValue("@id", CrearM.Text);
        comandP.ExecuteNonQuery();
        SqlDataReader infoP = comandP.ExecuteReader();


        while (infoP.Read())
        {


            ECalidad Calidad = (ECalidad)Enum.Parse(typeof(ECalidad), infoP["calidad"].ToString());
            EMarcaMoni MarcaMoni = (EMarcaMoni)Enum.Parse(typeof(EMarcaMoni), infoP["marca"].ToString());
            int id = Convert.ToInt32(infoP["id"].ToString());
            int pulgadas = Convert.ToInt32(infoP["pulgadas"].ToString());

            Monitor monitores = new Monitor(id, pulgadas, Calidad, MarcaMoni);


        }
    }

    catch(Exception ex)
    {
```

Correcciones del Trabajo Practico Nº4

"Falla uno de los 2 test unitarios creados
Que aparezca _4K para cargar queda mal, el guión no es parte de la nomenclatura más allá de que sea como creaste el enumerado
Solo doy de alta 2 productos... No hay fabricación de por medio. Ya se había marcado esto en el TP3."

“Falla uno de los 2 Test Unitarios creados”= Solucionado en la página 5 y 6 del Word.

“Que aparezca _4K para cargar queda mal, el guion no es parte de la nomenclatura más allá de que sea como creaste el enumerado” Se cambió los nombres.



“Solo le doy de alta 2 productos…No hay fabricación de por medio. Ya se había marcado esto en el TP Nº3”

Antes de hacer click en crear los producto, hago un objeto para hacer la fabricación de por medio.

Ejemplo en monitores.

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


    try
    {
        DataRow fila = this.dt.NewRow();


        Monitor monitor = new Monitor(contador, int.Parse(pulgadasM.Text), (ECalidad)Enum.Parse(ty



        fila["marca"] = monitor.MarcaMoni;

        fila["calidad"] = monitor.Calidad;
        fila["pulgadas"] = monitor.Pulgadas;

        this.dt.Rows.Add(fila);

        da.Update(dt);


    }
    catch (Exception ex)
```