

Mi trabajo practico se basa en la fabricación de Computadoras y Celulares mediante la compatibilidad de ciertos componentes, tiene un criterio de ensamblado en el cual no se pueden usar componentes que no sean compatibles con el producto seleccionado, como, por ejemplo:

En consola:

```
FabricaPC PC1 = new FabricaPC(MarcaCPU.Intel, Procesador.i3_7100, Motherboard.GIGABYTE_GA_B250M_D3H,
```

Esto es un pc con productos compatibles, la marca de procesador es Intel, el procesador es Intel socket 1151 y la mother también es Intel y socket 1151, por lo cual se podrá fabricar, en cambio si ponemos componentes incompatibles como los siguientes no se podrá

```
FabricaPC PC3 = new FabricaPC(MarcaCPU.Mediatek, Procesador.i9_10900, Motherboard.GIGABYTE_GA_B250M_D3H,
```

Este pc cuenta con un procesador marca mediatek de celular, un procesador Intel socket 1200 y una mother socket 1151, por lo cual no se fabricará

```
Ya existe un producto con este codigo de barras...
La mother ingresada no corresponde a el procesador seleccionado: i9_10900
La marca del procesador no sirve para el ensamblado de PC...
Ya existe un producto con este codigo de barras...
```

Además de la compatibilidad es importante que 2 productos fabricados no cuenten con el mismo UPC (Código de Producto Universal), por lo cual, aunque un producto tenga los componentes correctos si el UPC se repite no podrá ser fabricado

Mi proyecto cuenta con una clase base abstracta llamada Producto, la cual contendrá los atributos que se pueden utilizar tanto en una PC como en un Celular, el listado de atributos es el siguiente:

UPC, GPU, RAM, Marca del procesador, Sistema operativo, Almacenamiento

De esta clase heredan las clases FabricaPC y FabricaCelular, las cuales implementan métodos que tiene la clase producto, estas clases cuentan con los siguientes atributos

FabricaPC: Lector de CD, Gabinete, Fuente, Motherboard y CPU.

FabricaCelular: Jack de auricular, Huella dactilar, Camara, Batería, Carcasa, Pulgadas y Resolución.

Trate de darle un enfoque "Realista" a los componentes y que tengan cierto sentido a la hora de fabricar los productos

Temas vistos entre las clases 15 y 19

Excepciones:

Utilice excepciones en todo el proyecto para que el este no rompa, al trabajar con prácticamente solo enumerados es muy raro que el programa rompa, pero por ejemplo en el caso del UPC que es un entero me puede llegar a romper el

programa si el usuario ingresa de manera forzosa un entero, para esto use la excepción `OverflowException` y `FormatException`, en caso de que llegue a ocurrir alguna de estas excepciones hice que el código de barras se asigne de manera aleatoria así el programa puede seguir funcionando con normalidad, informándole al usuario de lo sucedido.

```
try
{
    GRAFICOS = Gpu;
    MARCA_CPU = Marca;
    MEMORIA_RAM = memoriaRam;
    ALMACENAMIENTO = almacenamiento;
    SISTEMA_OPERATIVO = sistemaOperativo;
    CODIGO_DE_BARRAS =CodigoDeBarras;
}
catch (System.OverflowException ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
catch (System.FormatException ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message + "Se asigno un numero random como codigo de barras");
    Archivos<Producto>.LogErrores(ex.ToString());
    CODIGO_DE_BARRAS = Random();
}
```

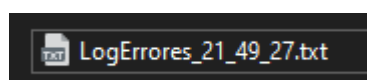
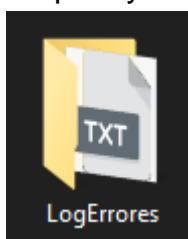
Además, cree un método que funcione como log de errores, así cada vez que suceda una excepción el programa guarde en un archivo de texto con la hora a la que sucedió el error, la finalidad de esto es tener una carpeta con todos los errores que tuvo el programa para poder ser arreglados en un futuro, hice que este log se encuentre en una carpeta aparte así su lectura es más fácil.

```
public static void LogErrores(string error)
{
    try
    {
        string Base = AppDomain.CurrentDomain.BaseDirectory;
        string Carpeta = System.IO.Path.Combine(Base, @"..\..\..\LogErrores\");
        string PathCarpeta = Path.GetFullPath(Carpeta);

        string RutaLogERROR = PathCarpeta + "LogErrores_" + DateTime.Now.ToString("HH_mm_ss") + ".txt";

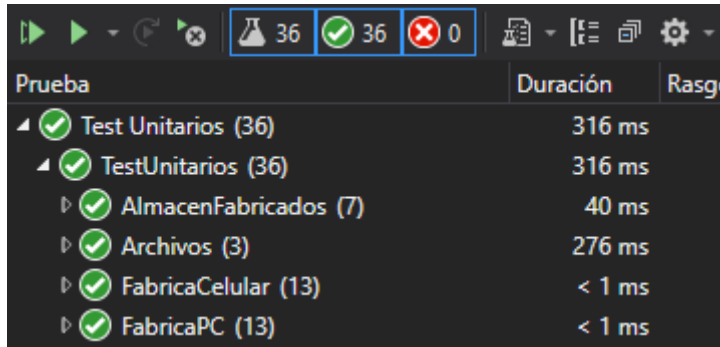
        using (StreamWriter Error = new StreamWriter(RutaLogERROR, true))
        {
            Error.WriteLine(error);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Carpeta y formato de los errores



Test Unitarios:

Utilice test unitarios para probar varias funcionalidades del programa, sobre todo para probar la compatibilidad de los productos y para ver si la carga de los productos fabricados en el almacén funcionaba correctamente.



Prueba	Duración	Rasgo
Test Unitarios (36)	316 ms	
TestUnitarios (36)	316 ms	
AlmacenFabricados (7)	40 ms	
Archivos (3)	276 ms	
FabricaCelular (13)	< 1 ms	
FabricaPC (13)	< 1 ms	

Tanto en **pc** como en **celular** por ejemplo testeé los Equals para probar que el funcionamiento de este sea correcto.

```
[TestMethod]
0 referencias
public void ProbarEquals_True()
{
    Entidades.FabricaPC PC1 = new Entidades.FabricaPC();
    Entidades.FabricaPC PC3 = new Entidades.FabricaPC();

    Assert.IsTrue(PC1.Equals(PC3));
}

[TestMethod]
0 referencias
public void ProbarEquals_False()
{
    Entidades.FabricaPC PC2 = new Entidades.FabricaPC();
    Entidades.FabricaPC PC3 = new Entidades.FabricaPC();

    Assert.IsFalse(PC2.Equals(PC3));
}
```

```
[TestMethod]
0 referencias
public void ProbarEquals_True()
{
    Entidades.FabricaCelular Celular1 = new Entidades.FabricaCelular();
    Entidades.FabricaCelular Celular3 = new Entidades.FabricaCelular();

    Assert.IsTrue(Celular1.Equals(Celular3));
}

[TestMethod]
0 referencias
public void ProbarEquals_False()
{
    Entidades.FabricaCelular Celular2 = new Entidades.FabricaCelular();
    Entidades.FabricaCelular Celular3 = new Entidades.FabricaCelular();

    Assert.IsFalse(Celular2.Equals(Celular3));
}
```

Equals devuelve true siempre que dos productos tengan el mismo código de barras o si tienen componentes incompatibles.

En el **almacén** probé por ejemplo que el funcionamiento del operador + o que la cantidad de productos fabricados funcionó correctamente.

```
[TestMethod]
0 referencias
public void ProbarSobrecargaOperadorMas_True()
{
    AlmacenProductosFabricados<Producto> Almacen;
    Almacen = "Test Method Francisco Rocha";

    Entidades.FabricaPC PC1 = new Entidades.FabricaPC();

    Almacen += PC1;

    int indice = Almacen | PC1;

    Assert.IsTrue(indice != -1);
}
```

```
[TestMethod]
0 referencias
public void ProbarCantidadGenerica_False()
{
    AlmacenProductosFabricados<Producto> Almacen;
    Almacen = "Test Method Francisco Rocha";

    Entidades.FabricaPC PC1 = new Entidades.FabricaPC();
    Entidades.FabricaCelular Celular1 = new Entidades.FabricaCelular();

    Almacen += PC1;
    Almacen += Celular1;

    Assert.IsFalse(1 == Almacen.CantidadGenerica);
}
```

En los archivos testee que el guardado y la carga de los datos funcione correctamente.

```
[TestMethod]
0 referencias
public void ProbarCargarListadoProductosFabricados()
{
    AlmacenProdutosFabricados<Producto> Almacen;
    Almacen = "TestMethod";

    AlmacenProdutosFabricados<Producto> Almacen2;
    Almacen2 = "TestMethod";

    Almacen.Directorío = AppDomain.CurrentDomain.BaseDirectory + "TestMethod.xml";
    Almacen2.Directorío = AppDomain.CurrentDomain.BaseDirectory + "TestMethod.xml";

    Entidades.FabricaPC PC1 = new Entidades.FabricaPC(MarcaCPU.Intel, Procesador.i3);
    Almacen += PC1;

    Archivos<Producto>.GuardarFabrica(Almacen);

    Almacen2 = Archivos<Producto>.CargarFabrica(Almacen2);

    Assert.IsTrue(Almacen.CantidadGenerica == Almacen2.CantidadGenerica);
}
```

Hice varios test mas como se ve en la primera imagen, pero esta es la vista general

Tipos Genéricos:

La clase AlmacenProdutosFabricados la hice de tipo genérico

```
0 referencias
public class AlmacenProdutosFabricados<T>
```

La cual se encarga de agregar mediante la sobrecarga del operador pipe y +, además se encarga listar todos los productos fabricados por el usuario

Cuenta con dos constructores privados, el primero se encarga de crear la lista genérica y el segundo se encarga de asignarle un nombre al almacén de productos fabricados

```
private AlmacenProdutosFabricados()
{
    try
    {
        Producto = new List<T>();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Archivos<Producto>.LogErrores(ex.ToString());
    }
}
```

```
private AlmacenProductosFabricados(string nombre) : this()
{
    try
    {
        this.nombreAlmacen = nombre;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Archivos<Producto>.LogErrores(ex.ToString());
    }
}
```

Interfaces:

Cree una interfaz producto la cual tiene 2 métodos y 6 propiedades que utilice en la clase abstracta producto

```
public interface IProductos
{
    8 referencias
    int CODIGO_DE_BARRAS { get; set; }

    9 referencias | 3/3 pasando
    GPU GRAFICOS { get; set; }

    9 referencias | 3/3 pasando
    RAM MEMORIA_RAM { get; set; }

    11 referencias | 3/3 pasando
    MarcaCPU MARCA_CPU { get; set; }

    9 referencias | 3/3 pasando
    SistemaOP SISTEMA_OPERATIVO { get; set; }

    9 referencias | 3/3 pasando
    Almacenamiento ALMACENAMIENTO { get; set; }

    12 referencias | 4/4 pasando
    bool NoMostrar();

    10 referencias | 2/2 pasando
    string Informacion();
}
```

```
public abstract class Producto : IProductos
{
```

En el almacén de productos fabricados establecí que T sea de tipo IProductos

```
31 referencias
public class AlmacenProductosFabricados<T> where T : IProductos
{
```

esto significa que el el almacén de productos fabricados acepta tanto a la clase Producto, FabricaPC y FabricaCelular, estas dos ultimas debido a que heredan de producto por lo tanto estas clases también son de tipo IProducto

Archivos y serialización:

Para archivos cree una clase Archivos la cual tiene 3 métodos, el primero se encarga de guardar todas las excepciones en un archivo de texto que se genera dependiendo de el horario del error en una carpeta destinada para todos los errores como mostré en el punto de excepciones

```
public static void LogErrores(string error)
{
    try
    {
        string Base = AppDomain.CurrentDomain.BaseDirectory;
        string Carpeta = System.IO.Path.Combine(Base, @"..\..\..\LogErrores\");
        string PathCarpeta = Path.GetFullPath(Carpeta);

        string RutaLogERROR = PathCarpeta + "LogErrores_" + DateTime.Now.ToString("HH_mm_ss") + ".txt";

        using (StreamWriter Error = new StreamWriter(RutaLogERROR, true))
        {
            Error.WriteLine(error);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Para la serialización utilice 2 métodos de guardado y cargado los cuales utilizan XML

El primero se encarga de guardar los datos del almacen de productos fabricados pasado por parámetro

```
public static void GuardarFabrica(AlmacenProdutosFabricados<T> Datos)
{
    try
    {
        using (XmlTextWriter auxArchivo = new XmlTextWriter(Datos.Directorio, Encoding.UTF8))
        {
            XmlSerializer auxEscriitor = new XmlSerializer(typeof(AlmacenProdutosFabricados<T>));
            auxEscriitor.Serialize(auxArchivo, Datos);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        LogErrores(ex.ToString());
    }
}
```

El segundo se encarga de retornar el almacen pasado por parámetro con todos los datos cargados

```
public static AlmacenProdutosFabricados<T> CargarFabrica(AlmacenProdutosFabricados<T> Datos)
{
    try
    {
        if (File.Exists(Datos.Directorio))
        {
            using (XmlTextReader auxArchivoLeer = new XmlTextReader(Datos.Directorio))
            {
                XmlSerializer auxLector = new XmlSerializer(typeof(AlmacenProdutosFabricados<T>));
                Datos = (AlmacenProdutosFabricados<T>)auxLector.Deserialize(auxArchivoLeer);
            }
        }
        else
        {
            Console.WriteLine("No existe el archivo");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        LogErrores(ex.ToString());
    }

    return Datos;
}
```