FABRICA DE JUEGOS: “Sueños Felices”

Descripción: Es una aplicación para crear juegos de mesa y video juegos. Algunos videojuegos que se fabrican, pueden tener algunos bugs y la misma fabrica permite arreglarlos.

Para guardar los datos tanto de la fabrica y de los juegos fabricados, la aplicación permite almacenar en archivos de texto y Xml los mismos de manera tal que se pueda seguir trabajando una vez que se cerro la aplicación.

También se puede ver el listado de juegos fabricados, con datos extras como por ejemplo: el diseño de la caja de los juegos de mesa o el peso en GB de los videojuegos.

Nota: Hay archivos ya hechos para cargarlos en la app para ver. Los archivos se guardan/crean en la carpeta: FrmFabrica\bin\Debug.

Entidades:

- Producto: Clase abstracta (Base)
  *Atributos:
  String nombre
  Int clasificacionPorEdad
  String descripción

  *Constructores:
  Un constructor con tres parametros(nombre,clasificación,descripcion)
  Un constructor sin parámetros

  *Propiedades:
  Public string Nombre (get;set;)
  Public string ClasificacionPorEdad (get;set;)
  Public string Descripcion (get;set;)

  *Metodo:
  Public virtual string Mostrar() : Muestra los datos del producto.

  *Sobreecargas:
  Operador = /como consecuente operador != : compara dos productos por su nombre, retorna true si son iguales, false si no lo son

  *Sobreescrituras:

  Public override string ToString(): retorna el nombre del producto

---

- VideoJuego: Clase hija(derivada de producto)
  Interfaces implementadas: IBugs, IPesoByte (desplayadas mas adelante)
  *Enumerados:
  ETipoVideoJuego: Plataformas,Carreras,RPG,Aventura,Terror,Accion,Puzzle
  EPlataforma: Playstation,Xbox,Switch,SmartPhone,PC.
  EFormato: Fisico,Digital,Ambos

  *Atributos:

  ETipoVideoJuego tipo;

  List<EPlataforma> plataformas;

  EFormato formato;

  Int duración;

  Int bugsDeFabrica;

  Bool testeado;

  *Constructores:

  Un constructor con sobrecarga que recibe por parámetros todos los atributos tanto de

  VideoJuego como Producto.

  Un constructor vacio.

  *Propiedades:

  Hay una propiedad de lectura y escritura para cada uno de los atributos de la clase VideoJuego.

  Cabe destacar las Propiedades:

  Public int BugsDeFabrica(get;): retorna la cantidad de bugs de un juego

  Public string ListadoDePlataformas(get;) : Retorna un string de las las plataformas que se encuentren dentro del listado.

  *Metodos:

  Sobreescritura del método Mostrar(): retorna un string con los datos de la clase base junto a los datos del videojuego.

  *Metodos Implementados por Interfaces:

  -Interface IBugs:----------

  Public int CalcularBugs(): genera un numero random de bugs entre 0 y 20 que se da como resultado al fabricar un juego

  Public bool FixearBugs(): Permite arreglar los números de bugs que tenga un videojuego:

-Interface IPesoByte:----------

Public int CalcularPesoDelJuego(): Calcula un peso en GB dependiendo de la plataforma que se haya elegido y el fomato del videojuego.

- JuegoDeMesa: Clase Hija de Producto, Interface: IArmado
  *Enum:

  EElementos: Cartas,Dados,Piezas.

  *Atributos:

  List<EElementos> elementos;

  Bool tablero;

  String Objetivo;

  *Constructores:

  Un Constructor que recibe por parámetro tanto los datos del Producto como del JuegoDeMesa.

  Un Constructor vacio.

  *Propiedades: Una propiedad de escritura y lectura para cada uno de los atributos de JuegoDeMesa.

  Cabe destacar: public string ListadoDeElementos: solo lectura y retorna los datos de los elementos que se encuentran dentro de la lista de elementos.

  *Metodos:

  String Mostrar(): Sobrescritura del método Mostrar que muestra los datos del producto y los datos del JuegoDeMesa.

  *Metodos implementados por interface:

  -IArmado----------

  Public string CrearCaja(): Crea una caja para el juego de mesa teniendo en cuenta que elementos lleva y si tiene tablero o no . Retorna un string con los datos de la caja.

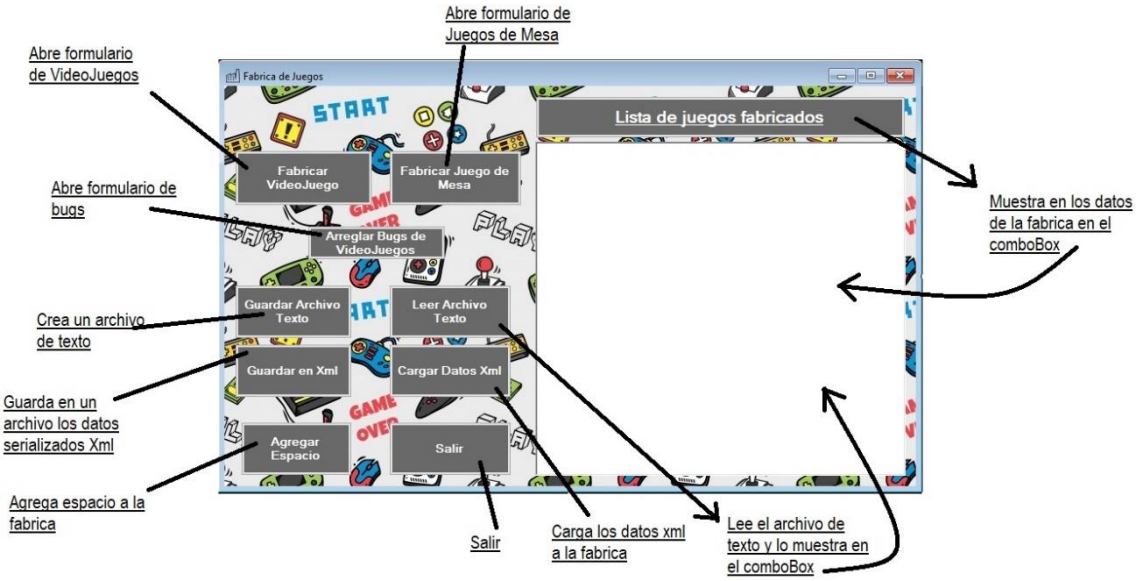
- Fabrica<T>: Clase Generica. Restricciones : T sea Producto
  *Atributos:
  List<T> juegosFabricados;
  String nombre;
  Int espacioLibre;
  *Constructores: Un constructor que recibe dos parámetros(nombre,stock) y otro constructor sin parámetros.

*Propiedades: una de lectura y escritura para cada uno de los atributos de la clase Fabrica.

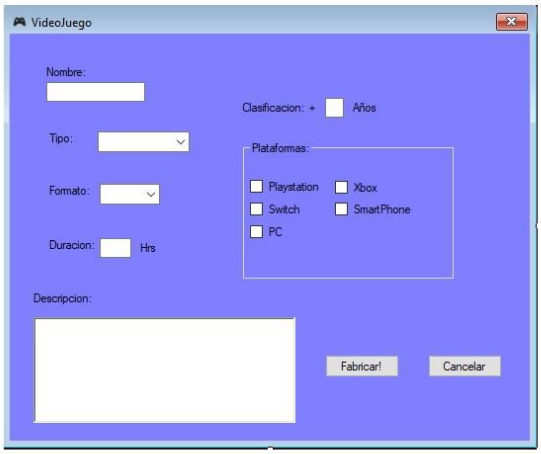
*Metodos:
Bool AgregarEspacioALaFabrica(int incremento): agrega espacio a la fabrica aumentando el numero del atributo espacioLibre.

String MostrarDatosDeLaFabrica(): Muestra todos los datos de la fabrica.
*Sobrecargas:
Operador +: Agrega un producto a la fabrica mientras sea un producto que no se encuentra en la fabrica y también si hay espacio libre en ella.
Operador = / como consecuente operador !=: Compara todos los productos dentro de la fabrica con el producto al que se le iguala. Retorna true si se encontró un producto igual, false si no se encontró.

- Formulario:
*FrmFabrica:



*FrmVideoJuego



*FrmJuegoDeMesa

*FrmBugs

*Metodos propios usados en los formularios:

-FrmPrincipal:

(Linea 110)Void DestinoXml(out string ruta,out string nombreDeArchivo): Donde se guarda/carga el archivo xml y que nombre tiene ese archivo xml.

Usado en el método void btnGuardar_Click(object sender, EventArgs e) y

Void btnCargar_Click(object sender, EventArgs e).

Nota a destacar: La ruta donde se genera/carga el archivo es : FrmFabrica\bin\Debug

(Linea 172)Void DestinoTexto(out string ruta,out string nombreDeArchivo): ruta y nombre del archivo texto.Donde se va a guardar/cargar.

Misma ruta que el archivo xml.

Usado en el método void btnGuardarTexto_Click(object sender, EventArgs e) y
Void btnLeerTexto_Click(object sender, EventArgs e).

-FrmVideoJuego:

(Linea 70)Private VideoJuego CargarVideoJuego(): Instancia un videojuego, con los datos validados ingresados en el formulario de video juego.

Usado en el método void btnAceptar_Click(object sender, EventArgs e)

(Linea 134)private void LimpiarDatosIngresados() : Una vez fabricado el videojuego, limpia los datos que estaban en los campos del formulario.

Usado en el método void btnAceptar_Click(object sender, EventArgs e)

(Linea 148)private List<EPlataforma> CargarListaPlataformas(): instancia una lista de plataformas y le agrega las plataformas que haya tildado el usuario desde el formulario y luego retorna esa lista.

Usado en el método private VideoJuego CargarVideoJuego()

-FrmJuegoDeMesa

(Linea 80) private JuegoDeMesa CargarJuegoDeMesa(): instancia un JuegoDeMesa, con los datos validados ingresados en el formulario de juego de mesa.

Usado en el método void btnAceptar_Click(object sender, EventArgs e)

(Linea 65)private void LimpiarDatosIngresados(): una vez fabricado el juego de mesa, limpia los datos del formulario.

Usado en el método void btnAceptar_Click(object sender, EventArgs e)

(Linea 150)private List<EElementos> CargarListaElementos(): Carga una lista con los elementos que haya tildado el usuario desde el formulario y la retorna.

Usado en el método private JuegoDeMesa CargarJuegoDeMesa().

-FrmBugs:

(Linea 53) private void ActualizarListaDeJuegosBugeados(): Actualiza la lista de juegos bugueados y quita a los juegos que ya han sido arreglados.

Usado en el método btnArreglar_Click(object sender, EventArgs e).

- Excepciones:
  *ArchivoNoEncontradoException: Se produce cuando no se encontró la ruta de un archivo, ya sea de texto o xml.
  -Throw en :
  (Linea 63) Metodo LeerTexto() en la clase ArchivoTexto.
  (Linea 66) Metodo Leer() en la clase SerializadorXml.
  -Catch en:
  (Linea 101) método btnCargar_Click en la clase FrmPrincipal
  (Linea 220) metodo btnLeerTexto_Click en la clase FrmPrincipal

*DatoInvalidoException: Se produce cuando el usuario no ingreso un dato valido en los formularios tanto de videojuego como de juego de mesa.
-Throw en:
(Lineas 94,103,108,116,132,140,142,170) en el método CargarJuegoDeMesa de la clase FrmJuegoDeMesa.
(Lineas 84,93,98,107,112,117,129,131,168) en el método CargarVideoJuego de la clase FrmVideoJuego.
(Lineas 142) del método CargarListaElementos() FrmJuegoDeMesa.
(Linea 168) del método CargarListaPlataformas() FrmVideoJuego.
-Catch en:
(Linea 41) Metodo btnAceptar_Click en FrmJuegoDeMesa.
(Linea 140) Metodo CargarJuegoDeMesa() en FrmJuegoDeMesa
(Linea 47) método btnAceptar_Click en FrmVideoJuego
(Linea 129)método CargarVideoJuego() en FrmVideoJuego

*DirectorioInvalidoException: Se produce cuando se le pasa un directorio invalido para la ruta de un archivo.
-Throw en:
(Linea 38) metodo GuardarTexto(T) de ArchivoTexto.cs
(Linea 39)método Guardar(T) de SerializadorXml.cs
-Catch en:
(Línea 66) método btnGuardar_Click() en FrmPrincipal
(Linea 194)método GuardarTexto_Click() en FrmPrincipal

*ElementoNullException: Se produce si uno de los strings pasados por parámetro del constructor de ArchivoTexto o del constructor SerializadorXml es null o si la ruta completa es null.
-Throw en:
(Linea 20)Constructor ArchivoTexto(string,string) de ArchivoTexto
(Linea 34) método GuardarTexto(T) de ArchivoTexto.
(Linea 21,35)Constructor SerializadorXml en SerializadorXml

-Catch en:
(Linea 62, 97) Metodo btnGuardar_Click() y btnCargar_Click() del FrmPrincipal
(Linea 190,216)Metodo btnGuardarTexto_Click() y btnLeerTexto_Click() del FrmPrincipal

*ListaNullException: Se produce si una lista es null.
-Throw en:
(Linea 156) Metodo CargarListaElementos() del FrmJuegoDeMesa
(Linea 154) Metodo CargarListaPlataformas() de FrmVideoJuego
-Catch en:
(Lineas 54,144) método btnAceptar_Click y metodo CargarJuegoDeMesa en FrmJuegoDeMesa.
(Lineas 60,125) método btnAceptar_click y método CargarVideoJuego de FrmVideoJuego.

*NoEspacioException: Se produce cuando no hay espacio libre en la fabrica.
-Throw en:
(Linea 11) en la sobrecarga del operador + de Fabrica.cs
-Catch en:
(Linea 46) método btnAceptar_Click de FrmJuegoDeMesa
(Linea 52) método btnAceptar_Click de FrmVideoJuego.

*ProductosIgualesException: Se produce cuando dos productos son iguales.
-Throw en:
(Linea 146) sobrecarga operador + de Fabrica.cs
-Catch en:
(Linea 50) metodo btnAceptar_Click de FrmJuegoDeMesa.
(Linea 56) metodo btnAceptar_Click de FrmVideoJUego.

- Test Unit:
  VideoJuegoTest:
  3 Pruebas:
  PruebaDeInstancia(): Prueba que se inicialice correctamente un VideoJuego
  PruebaCalcularPesoJuego(): Prueba que funcione correctamente el método CalcularPesoDelJuego.
  PruebaDeFixearBugs(): Prueba que retorne true el método FixearBugs() si el juego tiene bugs.

  JuegoDeMesaTest:
  2 Pruebas:
  PruebaCrearCaja(): Prueba que funcione correctamente el método CrearCaja().
  PruebaJuegoDeMesaInstancia(): pruba que instancie correctamente un JuegoDeMesa.

- Tipos Genericos:
  En el proyecto Entidades se encuentra la clase Fabrica<T> que es genérica con la restricción de que T tiene que ser un Producto.

  En el proyecto Archivos tenemos la clase ArchivoTexto<T> con la restricción de que T sea una Fabrica<Producto> y tenga un constructor vacio y SerializadorXml<T> solo con la restricción de que tenga un constructor vacio.

- Interfaces:
  Se encuentran en el proyecto Entidades:
  *IBugs:
  -Propiedad:
  Int BugsDeFabrica(get;): propiedad que escritura que retorna la cantidad de bugs.
  -Metodos:
  Int CalcularBugs(); : Genera un numero random de bugs al fabricar un video juego
  Bool FixearBugs(); : Arregla un juego que tenga bugs. Retorna true si lo logro, false si no tenia bugs.

  *IPesoByte:
  -Metodo: CalcularPesoDelJuego(); Calcula el peso del juego teniendo en cuenta la plataforma y el formato.

  *IArmado:
  -Metodo: string CrearCaja() : Calcula las dimensiones que puede llegar a tener una caja a partir de los elementos del juego de mesa y retorna los datos en string.

- Archivos y Serializadores:
  Proyecto Archivos:
  *ArchivoTexto<T>: clase genérica con restricciones: T debe ser Fabrica<Producto> y debe tener un constructor sin parámetros.

  -Metodos:
  Void GuardarTexto(T obj) : guarda un objeto en formato texto.
  String LeerTexto() Lee un archivo de texto y retorna sus datos.

  *SerializadorXml<T>: clase genérica con la restricción de tener un constructor sin parámetros.

  -Metodos:
  Void Guardar(T obj): Crea un archivo xml y lo guarda.
  T Leer(): Carga un archivo xml y retorna los datos obtenidos.