


Universidad Tecnológica Nacional Facultad Regional Avellaneda											
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos											
Materia: LABORATORIO DE PROGRAMACIÓN II											
Apellido:						Fecha:					
Nombre:						Docente⁽²⁾:					
División:						Nota⁽²⁾:					
Legajo:						Firma⁽²⁾:					
Instancia⁽¹⁾:	PP		RPP	X	SP		RSP		FIN		

Generar una Solución nombrada como: *Apellido.Nombre.Division*, que contenga un proyecto de tipo *Biblioteca de Clases (Entidades)* el cual tendrá la clase base **Producto**.

- 1) Todos sus atributos son protegidos. Posee **sólo** un constructor de instancia. La propiedad **Marca**, retornará el valor correspondiente del atributo `_marca`. La propiedad **Precio**, retornará el valor asociado al atributo `_precio`.
El método **privado** de clase **MostrarProducto** retornará una cadena detallando los atributos de la clase.
- 2) La clase Producto posee sobrecarga de operadores:
Igualdad (Producto, Producto). Retornará *true*, si son del mismo tipo y sus las marcas y códigos de barra son iguales, *false*, caso contrario.
Igualdad (Producto, EMarcaProducto). Retornará *true*, si la marca del producto coincide con el enumerado pasado por parámetro, *false*, caso contrario.
Explícito. Retornará el código de barra del producto que recibe como parámetro.
Implícito. Retornará toda la información del producto, reutilizando código.
- 3) El método Equals deberá indicar si dos objetos son del mismo tipo. Realizarlo en 1 línea de código.
- 4) Contendrá una propiedad abstracta de sólo lectura llamada **CalcularCostoDeProduccion**. Retornará un float.
- 5) Contendrá un método virtual llamado Consumir que retornará el string "Parte de una mezcla."

También tendrá las siguiente clases derivadas de producto:

- 6) Jugo posee un único atributo propio, que será inicializado por su **único** constructor de instancia.
El constructor de clase inicializará el atributo DeConsumo en verdadero.
El método **privado** de instancia **MostrarJugo**, retornará una cadena conteniendo la información completa del objeto.
El método ToString hará publicos los datos del producto.
CalcularCostoDeProducción será el 40% del precio final.
Consumir retornará "Bebible".
- 7) Galletita posee un único atributo propio, que será inicializado por su **único** constructor.
El constructor de clase inicializará el atributo DeConsumo en verdadero.
El método **privado** de clase **MostrarGalletita**, retornará una cadena conteniendo la información completa del objeto recibido por parámetro.
CalcularCostoDeProducción será el 33% del precio final.

El método ToString hará publicos los datos del producto.
Consumir retornará "Comestible".

- 8) Gaseosa posee un único atributo propio, que será inicializado **sólo** por una de las sobrecargas del constructor (reutilizar código).
El constructor de clase inicializará el atributo DeConsumo en verdadero.
El método **privado** de instancia **MostrarGaseosa**, retornará una cadena conteniendo la información completa del objeto.
CalcularCostoDeProducción será el 42% del precio final.
El método ToString hará publicos los datos del producto.
Consumir retornará "Bebible".
- 9) Harina posee un único atributo propio, que será inicializado **sólo** por una de las sobrecargas del constructor (reutilizar código).
El constructor de clase inicializará el atributo DeConsumo en falso.
El método **privado** de instancia **MostrarHarina**, retornará una cadena conteniendo la información completa del objeto.
CalcularCostoDeProducción será el 60% del precio final.
El método ToString hará publicos los datos del producto.

La última clase que tendrá el proyecto será **Estante**. Dicha clase posee dos atributos, ambos protegidos. Uno indicará la capacidad máxima que tendrá el estante para almacenar productos. El otro es una colección genérica de tipo Producto.

- 10) El constructor de instancia **privado** será el **único** que inicializará la lista genérica. La sobrecarga pública del constructor inicializará la capacidad del estante. Reutilizar código.
- 11) El método público GetProductos, retornará el valor asociado del atributo *_productos*.
El método público de **clase MostrarEstante**, retornará una cadena con toda la información del estante, incluyendo el detalle de cada uno de sus productos. Reutilizar código.
- 12) Sobrecarga de operadores:
Igualdad, retornará *true*, si es que el producto ya se encuentra en el estante, *false*, caso contrario.
Adición, retornará *true*, si el estante posee capacidad de almacenar al menos un producto más y dicho producto no se encuentra en el estante, *false*, caso contrario. Reutilizar código.
Sustracción (Estante, Producto), retornará un estante sin el producto, siempre y cuando el producto se encuentre en el listado. Reutilizar código.
Sustracción (Estante, ETipoProducto), retornará un estante con todos los productos menos el que coincida con el enumerado que recibe como parámetro. Reutilizar código.
Ejemplo: estanteSinJugo = estante – ETipoProducto.Jugo;
- 13) Método público y de instancia **GetValorEstante**, retornará el valor del estante de acuerdo con el enumerado que recibe como parámetro.
Ejemplo: precioGalletitas = estante.GetValorEstante(ETipoProducto.Galletita);
//Retorna sólo el valor de la suma de las galletitas
La propiedad pública **ValorEstanteTotal** está asociada a la sobrecarga privada y de instancia del método GetValorEstante. Reutilizar código.
- 14) Agregar a la solución un proyecto de tipo Aplicación de Consola (**TestEstante**) y agregar un método de clase que permita ordenar la lista de productos del estante a través del método Sort

de dicha lista genérica a través de su Código de Barras. Agregar el Main entregado, sin modificar línea alguna.

- 15) Agregar a la solución un proyecto de tipo Formulario de Windows (**FormEstante**) y agregar un método de clase que permita ordenar la lista de productos del estante a través del método Sort de dicha lista genérica a través de su Marca.

Deberá contar con 2 botones y un RichTextBox no editable (rtxtSalida). Los botones serán Ordenar (btnOrdenar) que ordenará la lista y la mostrará en rtxtSalida, y Ejecutar (btnEjecutar). Utilizar el código provisto en Form, sin alterarlo.

Puntos Extras:

1. Investigar y realizar un método llamado GuardarEstante, estático y de clase. El método deberá guardar en un archivo de texto toda la información del estante y sus productos.
2. Investigar y realizar un método llamado SerializarEstante, estático y de clase. El método deberá guardar en un archivo XML toda la información del estante y sus productos.
3. Investigar y realizar un método llamado DeserializarEstante, estático y de clase. El método deberá leer el archivo XML con toda la información del estante y sus productos.

Es condición de aprobación:

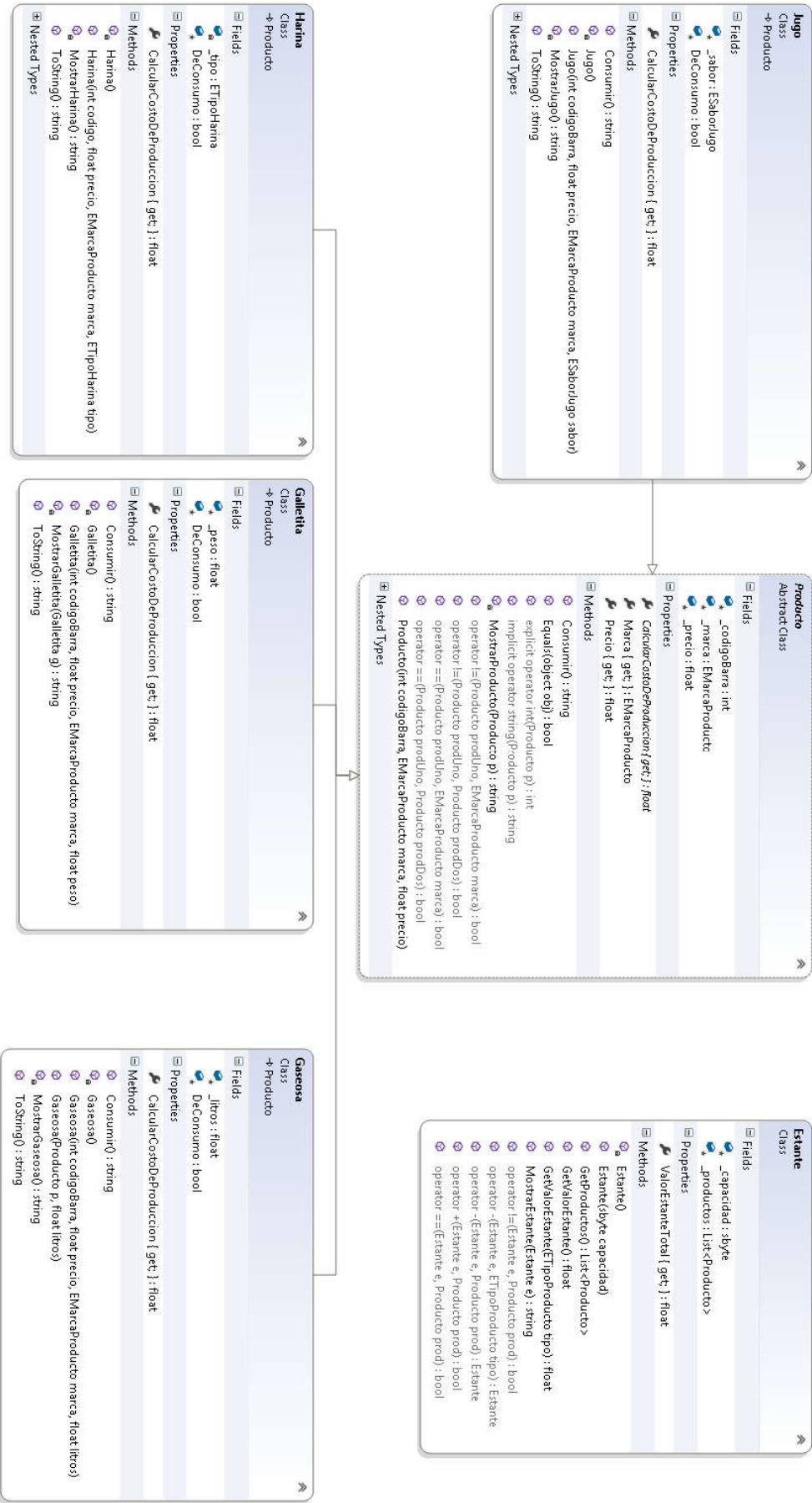
1. Qué se respeten todas las consignas dadas.
2. Qué todas las Clases, Métodos, Atributos, Propiedades, etc. sean nombrados exactamente como fue pedido en el enunciado.
3. Qué se introduzca el código de la función Main dada sin modificaciones.
4. Qué el proyecto no contenga errores de ningún tipo.
5. Qué el código compile y se ejecute de manera correcta.
6. Qué el código no se encuentre repetido con otro compañero (motivo por el cual desaprobarán la cursada).
7. Qué la salida por pantalla sea copia fiel de la entregada en este mismo documento.
8. Se deberá reutilizar código cada vez que se pueda, aunque no esté explicitado en el contenido del texto.
9. Se deberá documentar el código según las reglas de estilo de la cátedra.
10. La clase posterior a la corrección del ejercicio, los profesores harán preguntas sobre los métodos realizados, o posibles modificaciones del código a fin de completar la evaluación del mismo.

Forma de entrega:

1. Se deberá subir al repositorio de Git publicado a comienzos de cuatrimestre en el Campus.
2. Dentro de una carpeta llamada RPP.
3. Deberá estar correctamente publicado, en el *branch* master y accesible para su descarga por cualquier método el día martes 18 de Octubre de 2016 antes de las 10am.

Aclaración:

- Los diagramas de clases son a fin ilustrativo.



No se pudo agregar el producto al estante!!!
No se pudo agregar el producto al estante!!!
Valor total Estante1: 114,65
Valor Estante1 sólo de Galletitas: 89,65
Contenido Estante1:
CAPACIDAD: 4
MARCA: Favorita
CÓDIGO DE BARRAS: 102
PRECIO: 37,5
TIPO: CuatroCeros

MARCA: Pitusas
CÓDIGO DE BARRAS: 113
PRECIO: 33,65
PESO: 250

MARCA: Diversión
CÓDIGO DE BARRAS: 111
PRECIO: 56
PESO: 500

MARCA: Naranjú
CÓDIGO DE BARRAS: 112
PRECIO: 25
SABOR: Pasable

Estante ordenado por Código de Barra....

CAPACIDAD: 4
MARCA: Favorita
CÓDIGO DE BARRAS: 102
PRECIO: 37,5
TIPO: CuatroCeros

MARCA: Diversión
CÓDIGO DE BARRAS: 111
PRECIO: 56
PESO: 500

MARCA: Naranjú
CÓDIGO DE BARRAS: 112
PRECIO: 25
SABOR: Pasable

MARCA: Pitusas
CÓDIGO DE BARRAS: 113
PRECIO: 33,65
PESO: 250

Estante1 sin Galletitas: CAPACIDAD: 4

MARCA: Favorita
CÓDIGO DE BARRAS: 102
PRECIO: 37,5
TIPO: CuatroCeros

MARCA: Naranjú
CÓDIGO DE BARRAS: 112
PRECIO: 25
SABOR: Pasable

Contenido Estante2:
CAPACIDAD: 3
MARCA: Favorita
CÓDIGO DE BARRAS: 103
PRECIO: 40,25
TIPO: Integral

MARCA: Swift
CÓDIGO DE BARRAS: 333
PRECIO: 33
SABOR: Asqueroso

MARCA: Pitusas
CÓDIGO DE BARRAS: 113
PRECIO: 33,65
PESO: 250

Contenido Estante2:
CAPACIDAD: 3

Main

```
static void Main(string[] args)
{
    Console.Title = "Primer Parcial Laboratorio II - 2016 -";

    Estante est1 = new Estante(4);
    Estante est2 = new Estante(3);

    Harina h1 = new Harina(102, 37.5f, Producto.EMarcaProducto.Favorita,
Harina.ETipoHarina.CuatroCeros);
    Harina h2 = new Harina(103, 40.25f, Producto.EMarcaProducto.Favorita,
Harina.ETipoHarina.Integral);
    Galletita g1 = new Galletita(113, 33.65f, Producto.EMarcaProducto.Pitasas, 250f);
    Galletita g2 = new Galletita(111, 56f, Producto.EMarcaProducto.Diversión, 500f);
    Jugo j1 = new Jugo(112, 25f, Producto.EMarcaProducto.Naranja, Jugo.ESaborJugo.Pasable);
    Jugo j2 = new Jugo(333, 33f, Producto.EMarcaProducto.Swift, Jugo.ESaborJugo.Asqueroso);
    Gaseosa g = new Gaseosa(j2, 2250f);

    if (!(est1 + h1))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est1 + g1))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est1 + g2))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est1 + g1))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est1 + j1))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est2 + h2))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est2 + j2))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est2 + g))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }
    if (!(est2 + g1))
    {
        Console.WriteLine("No se pudo agregar el producto al estante!!!");
    }

    Console.WriteLine("Valor total Estante1: {0}", est1.ValorEstanteTotal);
    Console.WriteLine("Valor Estante1 sólo de Galletitas: {0}",
est1.GetValorEstante(Producto.ETipoProducto.Galletita));
    Console.WriteLine("Contenido Estante1:\n{0}", Estante.MostrarEstante(est1));

    Console.WriteLine("Estante ordenado por Código de Barra...");
    est1.GetProductos().Sort(Program.OrdenarProductos);
    Console.WriteLine(Estante.MostrarEstante(est1));

    est1 = est1 - Producto.ETipoProducto.Galletita;
    Console.WriteLine("Estante1 sin Galletitas: {0}", Estante.MostrarEstante(est1));
}
```

```
Console.WriteLine("Contenido Estante2:\n{0}", Estante.MostrarEstante(est2));  
est2 -= Producto.ETipoProducto.Todos;  
Console.WriteLine("Contenido Estante2:\n{0}", Estante.MostrarEstante(est2));  
  
Console.ReadLine();  
}
```

Form

```
private void btnEjecutar_Click(object sender, EventArgs e)
{
    rtxtSalida.Text = "";

    Estante est1;
    Estante est2;
    this.CargarEstante(out est1, out est2);

    rtxtSalida.Text += String.Format("Valor total Estante1: {0}", est1.ValorEstanteTotal);
    rtxtSalida.Text += String.Format("Valor Estante1 sólo de Galletitas: {0}",
    est1.GetValorEstante(Producto.ETipoProducto.Galletita));
    rtxtSalida.Text += String.Format("Contenido Estante1:\n{0}",
    Estante.MostrarEstante(est1));

    rtxtSalida.Text += "Estante ordenado por Marca....\n";
    est1.GetProductos().Sort(FormEstante.OrdenarProductos);
    rtxtSalida.Text += Estante.MostrarEstante(est1);

    est1 = est1 - Producto.ETipoProducto.Galletita;
    rtxtSalida.Text += String.Format("Estante1 sin Galletitas: {0}",
    Estante.MostrarEstante(est1));

    rtxtSalida.Text += String.Format("Contenido Estante2:\n{0}",
    Estante.MostrarEstante(est2));
    est2 -= Producto.ETipoProducto.Todos;
    rtxtSalida.Text += String.Format("Contenido Estante2:\n{0}",
    Estante.MostrarEstante(est2));
}

private void btnOrdenar_Click(object sender, EventArgs e)
{
    rtxtSalida.Text = "";

    Estante est1;
    Estante est2;
    this.CargarEstante(out est1, out est2);

    rtxtSalida.Text += "Estante 1 ordenado por Marca....\n";
    est1.GetProductos().Sort(FormEstante.OrdenarProductos);
    rtxtSalida.Text += Estante.MostrarEstante(est1);

    rtxtSalida.Text += "Estante 2 ordenado por Marca....\n";
    est2.GetProductos().Sort(FormEstante.OrdenarProductos);
    rtxtSalida.Text += Estante.MostrarEstante(est2);
}
```