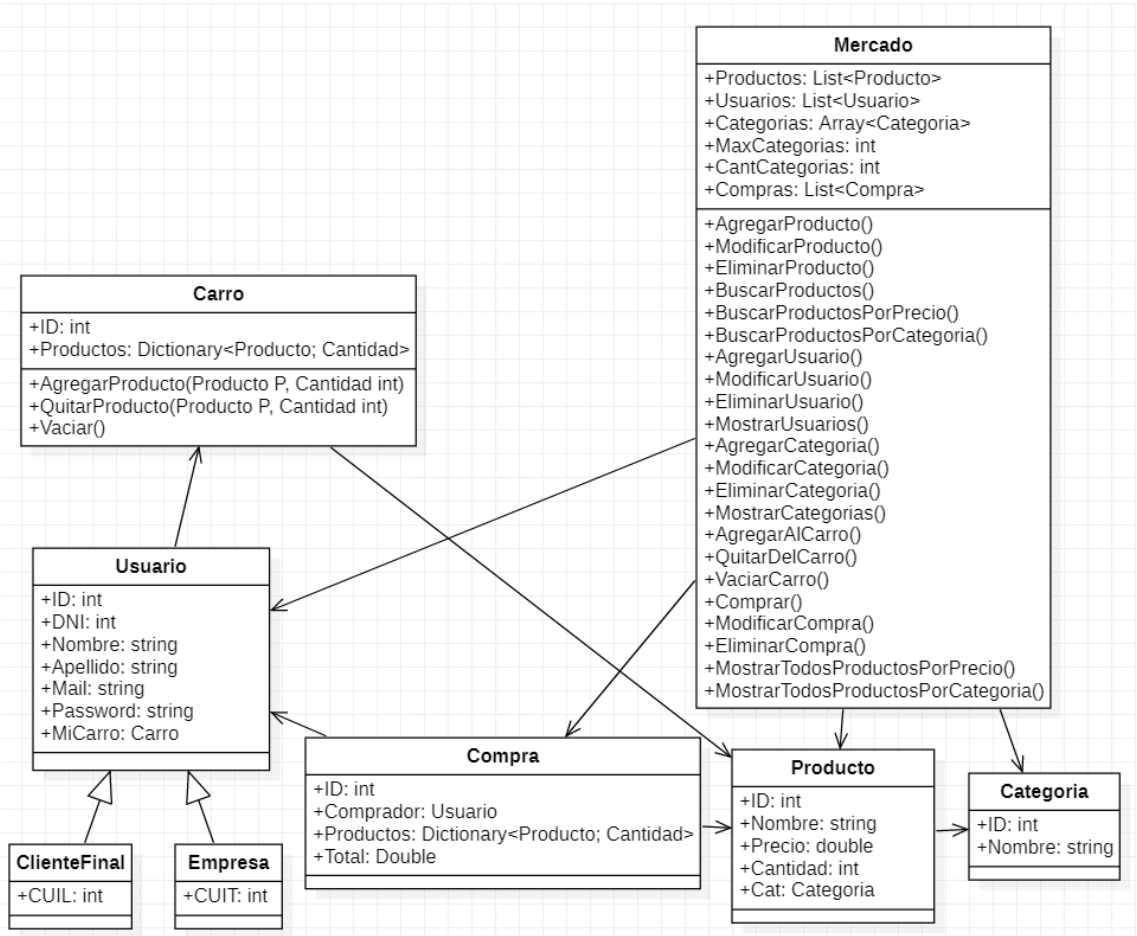
Trabajo Práctico 1

Definiciones

Para este trabajo implementaremos una **aplicación de consola** que simula una plataforma de comercio electrónico utilizando **C# ya sea en .NET Framework o .NET Core** (a elección por el grupo).

El siguiente diagrama UML muestra las clases involucradas en el modelo, debajo se brindan mayores detalles del sistema:



Detalles:

- Las clases contienen atributos que se pueden modelar como properties públicas con sus get y set correspondientes.
- Si bien no están en el diagrama (para optimizar espacio), todas las clases deben contar con su **constructor** correspondiente:
    - El campo ID es auto-incremental, es decir, se puede utilizar la cantidad de elementos de la lista correspondiente +1 como ID.
    - Usuario y Carro tienen el mismo ID. En su constructor, Usuario crea un nuevo carro con el mismo ID que le pasan por parámetro.

    - El constructor de “Mercado” no pide parámetros, la cantidad máxima de categorías está definida por la constante MaxCategorias mientras que la cantidad de categorías agregadas se guarda en CantCategorias (datos necesarios ya que Categorías es un arreglo).
- La clase Usuario es abstracta e implementa un orden por número de DNI.
- La clase Producto implementa orden por Nombre.
- La clase Categoría implementa orden por Nombre.
- La clase Compra implementa orden por ID.
- Las clases Carro y Mercado no implementan orden.
- Toda clase hace además override del método ToString(), retornando todas sus propiedades separadas por un “ – “, de modo que sea posible imprimir un objeto de cualquier clase.
    - Carro y Compra además muestran todos los ítems en su diccionario
    - Mercado no implmenta ToString().

Métodos:

- Los parámetros no se agregaron en el diagrama para optimizar espacio, debajo se encuentra el detalle completo.
- Todos los métodos de ABM (alta, baja o modificación) devuelven un boolean que indica si se pudo realizar la transacción correctamente.
- Recordar que a la hora de “Agregar” un nuevo elemento, no pedimos un ID porque todavía no lo tiene, Mercado se lo asigna en base a la cantidad de elementos que tiene.
- Clase Mercado:
    - bool AgregarProducto (string Nombre, double Precio, int Cantidad, int ID_Categoría)
    - bool ModificarProducto (int ID, string Nombre, double Precio, int Cantidad, int ID_Categoría)
    - bool EliminarProducto (int ID)
    - void BuscarProductos (string Query): Muestra por pantalla, **ordenado por Nombre**, los productos que **contienen** en su nombre la cadena ingresada por el usuario para la búsqueda.
    - void BuscarProductosPorPrecio (string Query): Muestra por pantalla, **ordenado por Precio de menor a mayor**, los productos que **contienen** en su nombre la cadena ingresada por el usuario para la búsqueda.
    - void BuscarProductosPorCategoria (int ID_Categoria): Muestra por pantalla, **ordenado por Nombre**, los productos que pertenecen a la categoría con ID ingresada por el usuario para la búsqueda.
    - bool AgregarUsuario (int DNI, string Nombre, string Apellido, string Mail, string Password, int CUIT_CUIL, bool EsEmpresa)
    - bool ModificarUsuario (int ID, int DNI, string Nombre, string Apellido, string Mail, string Password, int CUIT_CUIL, bool EsEmpresa)
    - bool EliminarUsuario (int ID)

- void MostrarUsuarios(): Muestra por pantalla todos los usuarios que existen en el mercado ordenados por defecto (DNI).
- bool AgregarCategoria (string Nombre)
- bool ModificarCategoria (int ID, string Nombre)
- bool EliminarCategoria (int ID): *Cuidado al eliminar elementos de un arreglo!*
- void MostrarCategoria (): Muestra por pantalla todos los usuarios que existen en el mercado ordenados por defecto (DNI).
- bool AgregarAlCarro(int ID_Producto, int Cantidad, int ID_Usuario):
  - Pide al usuario el carro.
  - Si el parámetro Cantidad es menor a la Cantidad (atributo) del producto "ID_Producto" (hay stock), agrega el Producto al carro del usuario.
  - Si no hay stock devuelve falso.
  - *Nota: En este punto no decremento el atributo Cantidad en la clase Producto ya que el usuario todavía NO realizó la compra.*
- bool QuitarDelCarro (int ID_Producto, int Cantidad, int ID_Usuario): Disminuye la cantidad del producto ID_Producto en el carro del usuario.
- bool VaciarCarro (int ID_Usuario): vacía el carro del usuario.
- bool Comprar (int ID_Usuario): Genera una nueva compra:
  - Busca el usuario pasado como parámetro
  - Le pide su carro y a este los productos con la cantidad respectiva.
  - En base a esto calcula el total según el tipo de usuario, a la hora de hacer una compra existe una diferencia si el usuario es ClienteFinal o Empresa ya que este último paga 21% menos (descuenta IVA), esto se debe ver reflejado en el total de la compra.
  - Crea un nuevo elemento compra con el detalle necesario (ID auto-incremental, Comprador = ID_Usuario, Productos copiando los elementos del carrito a un *nuevo* diccionario (*¡cuidado con las referencias!*) y el total según calculado).
  - Disminuye el stock de los productos según lo comprado por el usuario.
  - Luego vacía el carrito del usuario.
  - Muestra el resultado por pantalla (ToString de la compra recientemente creada) y devuelve el valor correspondiente indicando si la ejecución fue correcta.
- bool ModificarCompra (int ID, double Total): solo se permite modificar el total en caso que haya un error de facturación. El resto de los datos no pueden ser modificados.
- bool EliminarCompra (int ID): Nuestro "botón de arrepentimiento"…
- void MostrarTodosProductosPorPrecio(): Muestra todos los productos del mercado ordenados por precio.
- void MostrarTodosProductosPorCategoria(): Muestra todas las categorías del mercado y para cada una de ellas los productos dentro de la misma.

- Clase Carro:

- bool AgregarProducto (Producto P, int Cantidad): Crea una nueva entrada en el diccionario del carro para el producto. Si el producto ya existe, se actualiza la entrada correspondiente.
- bool AgregarProducto (Producto P, int Cantidad): Disminuye la cantidad del producto P. Si la cantidad es 0, se elimina la entrada del diccionario.
- Vaciar(): Reinicia el diccionario de Productos.

Condiciones de aprobación:

Para aprobar el trabajo se deben cumplir con los siguientes escenarios:

- Se deben implementar todas las clases que componen el modelo abordado previamente.
- La implementación de los métodos relacionados con la compra de productos es opcional (aunque recomendable) en esta etapa, siendo posible dejar la implementación de los mismos para el TP2.
- Se debe crear una clase program con una aplicación por consola que:
  - Al iniciar el programa, se crea un "mercado" que es la clase principal con un arreglo de categorías de tamaño definido en la constante MaxCategorias y el resto de las listas vacías.
  - Al ingresar el usuario visualiza una pantalla con las opciones para:
    1. Administrar
    2. Comprar
  - Si elije la opción 1, administrar, se muestran las opciones para:
    1. Alta de categoría
    2. Baja de categoría
    3. Modificación de categoría
    4. Mostrar categorías que existen en el sistema
    5. Alta de usuario
    6. Baja de usuario
    7. Modificación de usuario
    8. Mostrar usuarios que existen en el sistema
    9. Alta de producto
    10. Baja de producto
    11. Modificación de producto
    12. Mostrar productos que existen en el sistema
    13. Mostrar productos que existen en el sistema ordenados por precio
    14. Mostrar productos que existen en el sistema ordenados por categoría
  - Si elije la opción 2, comprar, se muestra: "Bajo construcción, próximamente en TP2!"
- Cada vez que se muestren los elementos de una clase deben estar ordenados según lo establecido en el punto anterior, salvo que el método especifique lo contrario (por ejemplo ProductosPorPrecio).

Condiciones de entrega

- La solución completa con todo el código se debe entregar en un archivo .zip con nombre TP1 – Grupo X. Reemplazar X por identificador del grupo (nombre o número).
- Se debe incluir dentro del .zip un archivo “ReadMe” que explique cómo utilizar ambos programas y cualquier aclaración que consideren necesaria o decisión de diseño que hayan tomado (puede ser en formato Word, txt, Excel, etc. No se evalúa presentación pero si contenido del mismo).
- La entrega se debe realizar por mail a mi casilla: walter.gomez@davinci.edu.ar
- El sujeto debe ser: Plataformas de programación: TP1 – Entrega grupo X. Reemplazar X por identificador del grupo (nombre o número).
- El cuerpo debe contener el detalle de los alumnos que componen el grupo, uno solo de ellos hace la entrega por el grupo pero debe poner en copia al resto, siendo responsabilidad de todos la entrega del trabajo.
- La fecha de entrega es 12/08/21. Entregar fuera de término afecta sobre la nota del trabajo práctico.
- Tengan en cuenta que la entrega se puede complicar ya que el mail no acepta archivos .EXE aun si los mismos están dentro de otro archivo .zip. También pueden compartir su proyecto por drive o enviarme el link de un repositorio.
- El proyecto debe compilar tal como fue entregado, caso contrario, será desaprobado.