

PROYECTO 2ª EVALUACIÓN DAM

PROGRAMACIÓN 1CFGS

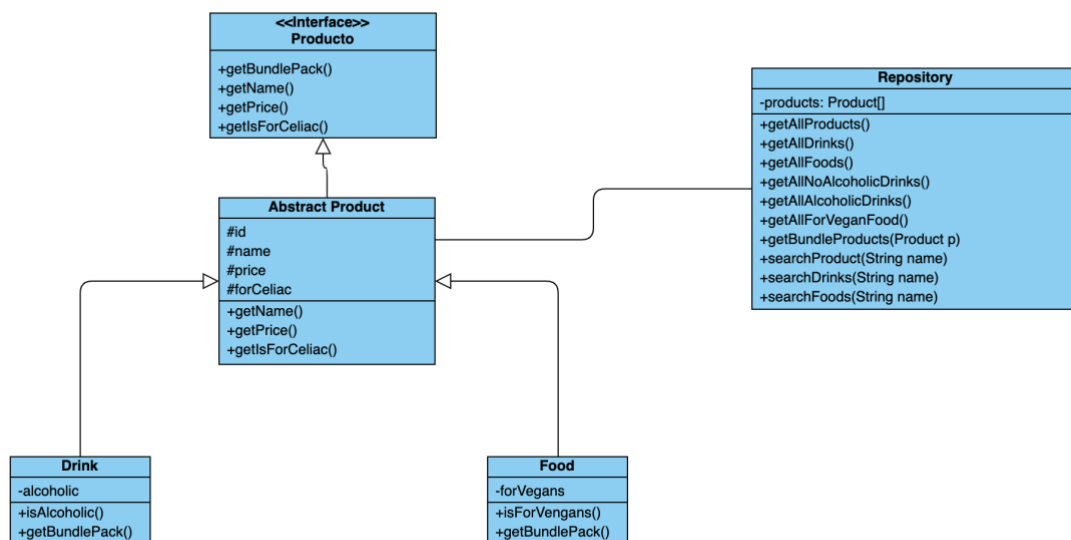


**DESARROLLO DE UNA APLICACIÓN PARA LA GESTIÓN DE ESTABLECIMIENTOS DE
VENTA DE ALIMENTACIÓN A DOMICILIO.**

El proyecto consiste en realizar una aplicación standalone (ejecutable, sin instalar, y sin necesidad de internet para ejecutarse en una máquina) para la gestión de un comercio de alimentación que ofrece comida a domicilio.

La aplicación será ejecutada por consola y será importante diseñar los menús y accesos teniendo en cuenta la usabilidad y accesibilidad de la misma.

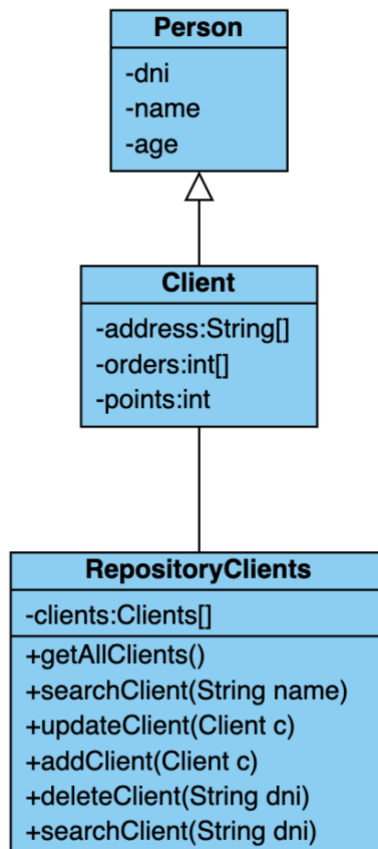
La aplicación tendrá cargada en memoria la carta disponible siguiendo las siguientes premisas:



Habr  productos que podr n “paquetizarse” para obtener descuentos. Por ejemplo, podemos tener una bebida concreta (Refresco de Cola) que al pedirse con una comida (Paquete de patatas) se obtenga un descuento del 10% en ambos productos. Por tanto, cada producto tendr  un array de los ids de aquellos productos con los que se puede combinar y obtener el descuento de 10%.

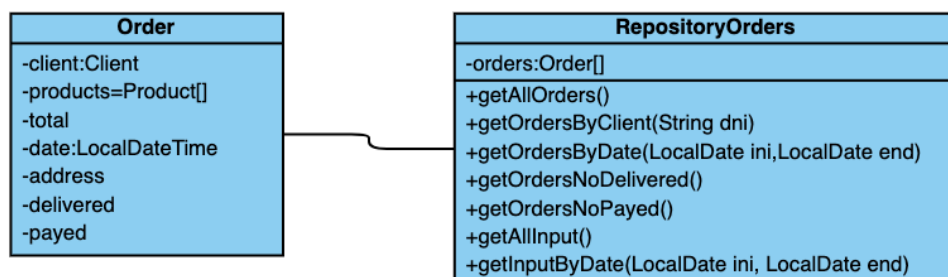
Repository ser  la clase que mantenga en memoria el array completo de productos disponibles en el restaurante. Por tanto, sobre esta clase el men  principal deber  realizar las operaciones de lectura para poder mostrar los productos disponibles para el pedido.

Por otra parte, tendremos un sistema de gesti n de clientes siguiendo las siguientes premisas:



Este repositorio será almacenado en un archivo (serializado) y cargado a memoria cuando se inicia la aplicación. Una vez realizada una modificación deberá ser actualizado y almacenado en el archivo de nuevo (clients.data). No podrán existir dos clientes con el mismo dni. Los clientes podrán tener más de una dirección de entrega no repetida.

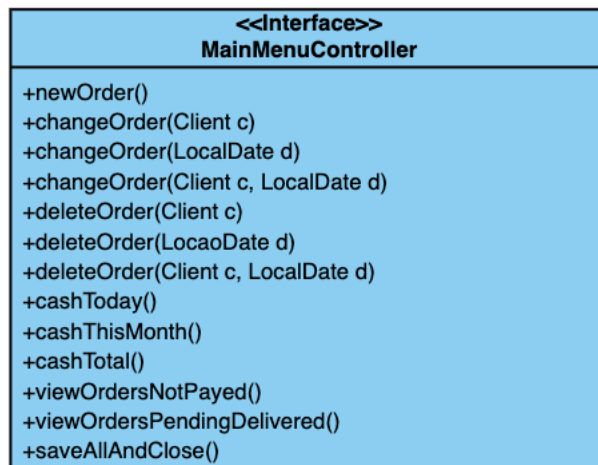
Los clientes realizarán pedidos que serán almacenados (serializados) en un archivo para mantener persistencia y su historial. Los pedidos seguirán las siguientes premisas:



Mientras no se termina el pedido se aconseja implementar un carrito de la compra (patrón Singleton). El carrito podrá ser editado en todo momento, pero cuando se almacena como Pedido, solo podrá ser modificado los parámetros de pagado y enviado (o eliminar el pedido).

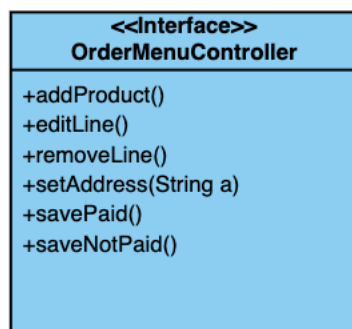
La interfaz de la aplicación será por consola, cuidando el patrón MVC (Modelo-Vista-Controlador) en tanto sea posible.

La interfaz del controlador del menú principal contará con las siguientes opciones:



A estas funcionalidades se les podrá añadir tantas como sean oportunas.

En cuanto al menú del pedido (mientras se está realizando uno y se gestiona el carrito de la compra):



Se da por hecho que en todo momento se van imprimiendo el cliente, la fecha y todas las líneas del pedido. A estas funcionalidades se le puede añadir todas las que sean necesarias.

```

classDiagram
    class Product {
        <<interface>>
        +getBundlePack()
        +getName()
        +getPrice()
        +getIsForCeliac()
    }
    class AbstractProduct {
        +id
        +fName
        +fPrice
        +fForCeliac
        +getName()
        +getPrice()
        +getIsForCeliac()
    }
    class Drink {
        +alcoholic()
        +isAlcoholic()
        +getBundlePack()
    }
    class Food {
        +isForVegans()
        +isForVegans()
        +getBundlePack()
    }
    class Chart {
        +orderOrder
    }
    class Repository {
        +products: Product[]
        +getAllProducts()
        +getAlAllDrinks()
        +getAlAllFoods()
        +getAlNoAlcoholicDrinks()
        +getAlAlcoholicDrinks()
        +getAlForVeganFood()
        +getBundleProducts(Product p)
        +searchProduct(String name)
        +searchDrinks(String name)
        +searchFoods(String name)
    }
    class Client {
        +address String[]
        +orders int[]
        +points int
    }
    class RepositoryClients {
        +clients: Client[]
        +getAlClients()
        +searchClient(String name)
        +updateClient(Client c)
        +addClient(Client c)
        +deleteClient(String dni)
        +searchClient(String dni)
    }
    class RepositoryOrders {
        +orders Order[]
        +getAlOrders()
        +getOrdersByClient(String dni)
        +getOrdersByDate(LocalDate ini, LocalDate end)
        +getOrdersNoDelivered()
        +getOrdersNoPaid()
        +getAlAllInput()
        +getInputByDate(LocalDate ini, LocalDate end)
    }
    class RepositoryUtils {
        +loadClients()
        +saveClients()
        +loadOrders()
        +saveOrders()
    }
    class MainMenuController {
        <<interface>>
        +newOrder(Client c, LocalDate time id)
        +changeOrder(Client c)
        +changeOrder(LocalDate d)
        +changeOrder(Client c, LocalDate d)
        +deleteOrder(Client c)
        +deleteOrder(LocalDate d)
        +viewOrder(Client c, LocalDate d)
        +cashToday()
        +cashThisMonth()
        +cashTotal()
        +viewOrdersNotPaid()
        +viewOrdersPendingDelivered()
        +saveAllAndClose()
    }
    class OrderMenuController {
        <<interface>>
        +addProduct()
        +editLine()
        +removeLine()
        +setAddress(String a)
        +savePaid()
        +saveNotPaid()
    }
    Product <|-- AbstractProduct
    AbstractProduct <|-- Drink
    AbstractProduct <|-- Food
    AbstractProduct <|-- Chart
    Repository <|-- RepositoryClients
    Repository <|-- RepositoryOrders
    Repository <|-- RepositoryUtils
    Client <|-- RepositoryClients
    Client <|-- RepositoryOrders
    Client <|-- RepositoryUtils
    MainMenuController <|-- OrderMenuController
    
```

Se valorará la usabilidad de la aplicación.

Nota: se empleará metodología SCRUM y tableros [TRELLO](#) para gestión del proyecto.

