DIAGRAMAS DE CLASES

ÍNDICE-CONTENIDO:

INTRODUCCIÓN:

- En esta práctica se analizarán, diseñarán y transformarán diagramas de clases a código.

OBJETIVOS:

- El objetivo principal de los diagramas de clase a tratar es representar la estructura estática de un sistema o aplicación en forma visual, utilizando elementos como clases, atributos, métodos y relaciones entre ellos.

ALCANCE-EJERCICIO 1:

- El ejercicio plantea la creación de un sistema de matriculación de alumnos en un instituto, donde se deben modelar diferentes entidades y relaciones utilizando un diagrama de clases en UML.

ALCANCE-EJERCICIO 2:

- El ejercicio plantea realizar ingeniería inversa de los ejercicios de OOP, de 3 ejercicios de OOP que hayamos trabajo en clase en PRO. Comprendemos así la estructura, las clases, los atributos y los métodos implementados.

DESARROLLO - EJERCICIO 1:

- **Forma que desarrollaremos este ejercicios y sus diferentes apartados:**

1. **Creación de clases:**

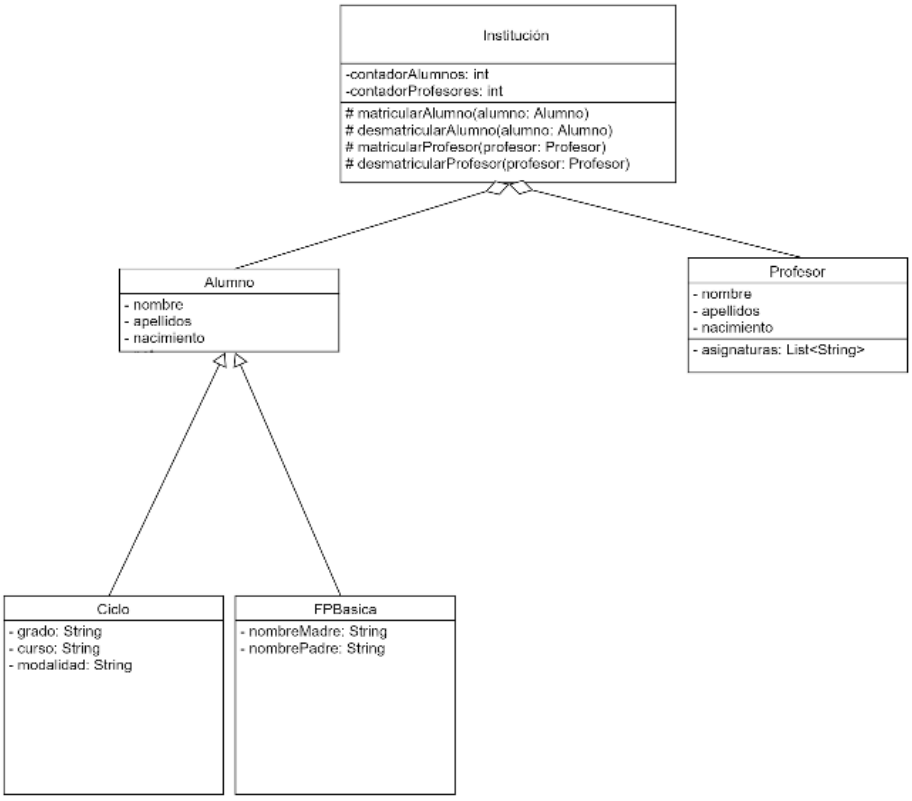
   Clase Profesor: Contendrá los atributos nombre, apellidos, fecha de nacimiento y asignaturas que imparte. También se implementará un contador de profesores.

   Clase Alumno: Tendrá los atributos nombre, apellidos, fecha de nacimiento y notas. Se implementará un contador de alumnos.

   Clase AlumnoCiclo: Hereda de la clase Alumno e incluye atributos adicionales como el grado al que pertenece, el curso (primero o segundo) y la modalidad (presencial o semipresencial).

   Clase AlumnoFPBasica: Hereda de la clase Alumno e incluye atributos adicionales como el nombre de la madre y el padre.

   Clase Curso: Representa un curso en el que los alumnos pueden ser escolarizados. Los alumnos y profesores pueden matricularse o desmatricularse de un curso. Los profesores pueden ser tutores de un curso.


Institución
-contadorAlumnos: int
-contadorProfesores: int
matricularAlumno(alumno: Alumno)
desmatricularAlumno(alumno: Alumno)
matricularProfesor(profesor: Profesor)
desmatricularProfesor(profesor: Profesor)
Alumno
- nombre
- apellidos
- nacimiento
Profesor
- nombre
- apellidos
- nacimiento
- asignaturas: List<String>
Ciclo
- grado: String
- curso: String
- modalidad: String
FPBasica
- nombreMadre: String
- nombrePadre: String

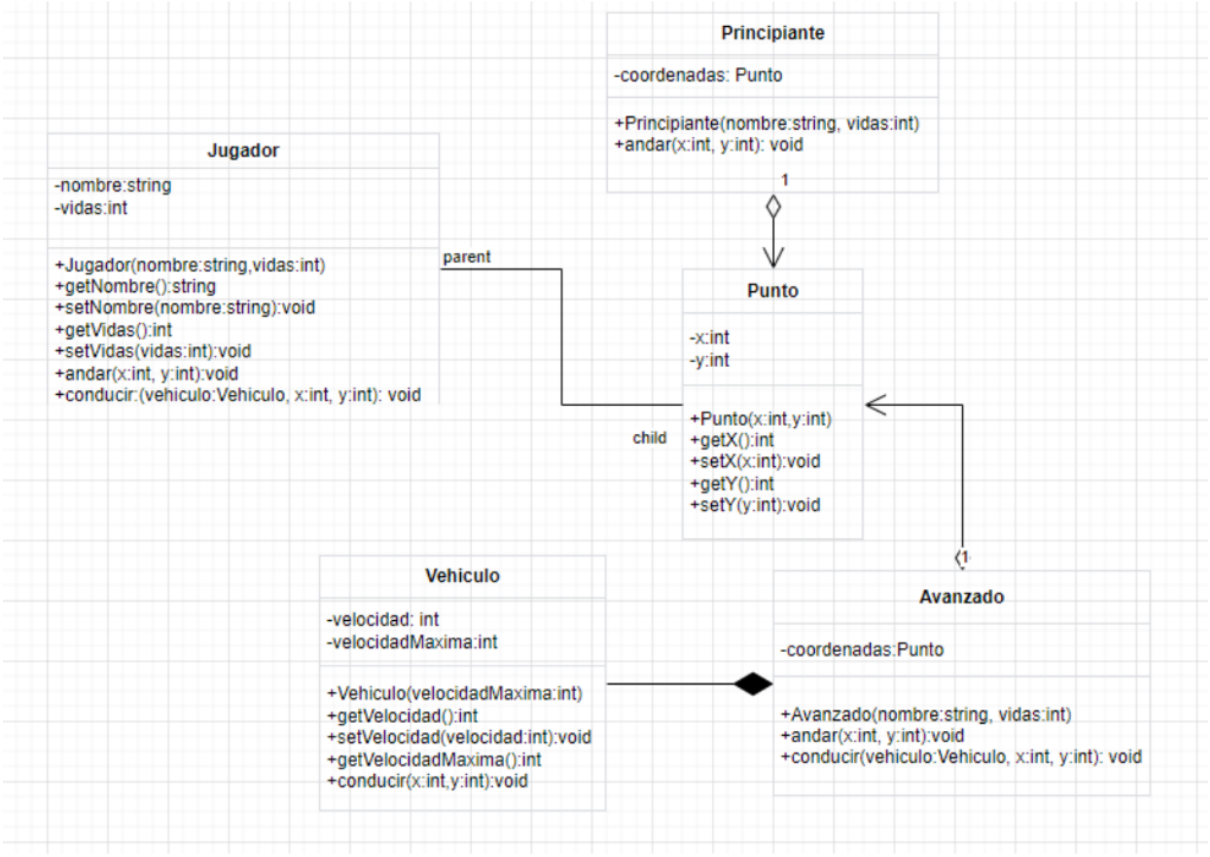
2. **Creación de clases para el juego de ordenador:**

Clase Jugador: Contendrá los atributos nombre y número de vidas. También se incluirá el método andar para desplazarse a unas coordenadas (x, y).

Clase JugadorPrincipiante: Hereda de la clase Jugador y añade el método andar para el desplazamiento a pie.

Clase Vehículo: Contendrá el atributo velocidad y métodos para leer y ajustar su valor. No puede superar una velocidad máxima predefinida.

Clase JugadorAvanzado: Hereda de la clase Jugador y agrega el atributo vehículo. Implementa el método conducir para desplazarse a unas coordenadas utilizando el vehículo.



3. **Creación de clases para la gestión de satélites:**

   Clase CuerpoSolido: Contendrá los atributos imagen, dirección, posición y velocidad.

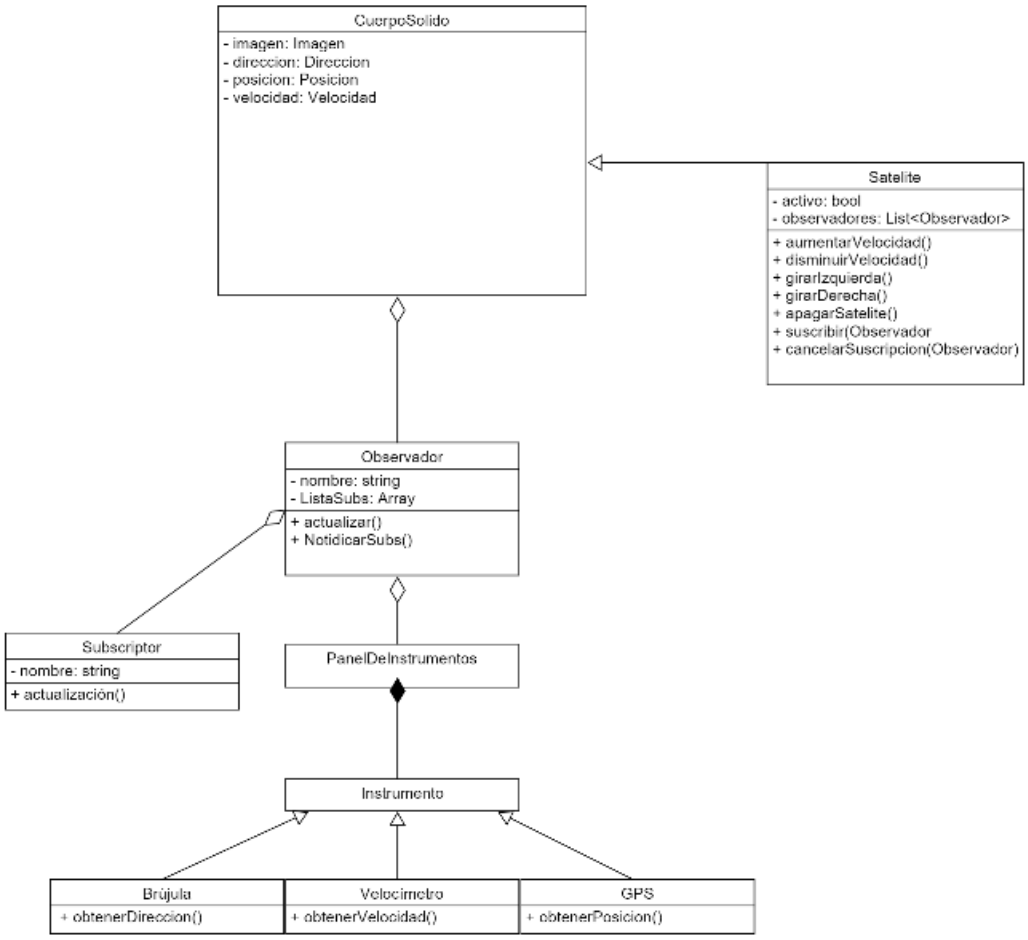
   Clase Satélite: Hereda de la clase CuerpoSolido y agrega métodos para controlar el satélite, como aumentar, disminuir, girar a la izquierda, girar a la derecha y apagar.

   Clase Instrumento: Representa un instrumento utilizado para medir diferentes valores, como velocidad, dirección e imagen. Cada instrumento tratará la información de manera diferente.

   Clase PanelInstrumentos: Contendrá una lista de instrumentos, permitiendo activar, desactivar y visualizar la información obtenida.

   Clase Publicador: Representa a un publicador en el patrón de arquitectura Observador. Puede notificar a sus suscriptores, permitir suscripciones o bajas, y almacena una lista de suscriptores.

   Clase Suscriptor: Representa a un suscriptor en el patrón de arquitectura Observador. Contendrá el método de actualización, que será llamado por el publicador cuando ocurra un evento observado.


CuerpoSolido
- imagen: Imagen
- direccion: Direccion
- posicion: Posicion
- velocidad: Velocidad
Satelite
- activo: bool
- observadores: List<Observador>
+ aumentarVelocidad()
+ disminuirVelocidad()
+ girarIzquierda()
+ girarDerecha()
+ apagarSatelite()
+ suscribir(Observador
+ cancelarSuscripcion(Observador)
Observador
- nombre: string
- ListaSubs: Array
+ actualizar()
+ NotificarSubs()
Subscriptor
- nombre: string
+ actualización()
PanelDeInstrumentos
Instrumento
Brújula
+ obtenerDireccion()
Velocimetro
+ obtenerVelocidad()
GPS
+ obtenerPosicion()

DESARROLLO - EJERCICIO 2:

Presentamos un enfoque general sobre cómo abordar este proceso:

1. Primero revisaremos el código existente: Analizamos el código fuente de los ejercicios de OOP realizados en la asignatura de programación y en el examen. Comprendemos la estructura, las clases, los atributos y los métodos implementados.

2. Identificar las clases: Identifica las clases presentes en el código y sus relaciones. Observa cómo interactúan entre sí y cómo se organizan en el código.

3. Crear el diagrama de clases: Utilizando el plugin de diagramas de clase integrado en Visual Studio, crea un nuevo diagrama de clases en el proyecto correspondiente. Agrega las clases identificadas en el paso anterior al diagrama y establece las relaciones entre ellas, como asociaciones, agregaciones, herencias, etc.

4. Generamos código a partir del diagrama: Si el código existente está incompleto o mal implementado, utilizaremos el diagrama de clases como referencia para completar o corregir el código correspondiente.

EJERCICIO 1:

- **Enunciado:**

  La cafetería del centro nos ha pedido que realicemos un programa que permita gestionar los pedidos que realiza el alumnado. Estos pedidos se componen de una serie de productos, tantos como quiera el usuario, a elegir entre los que ofrece la cafetería en su menú. Los productos que contiene el menú se especificarán en el código.

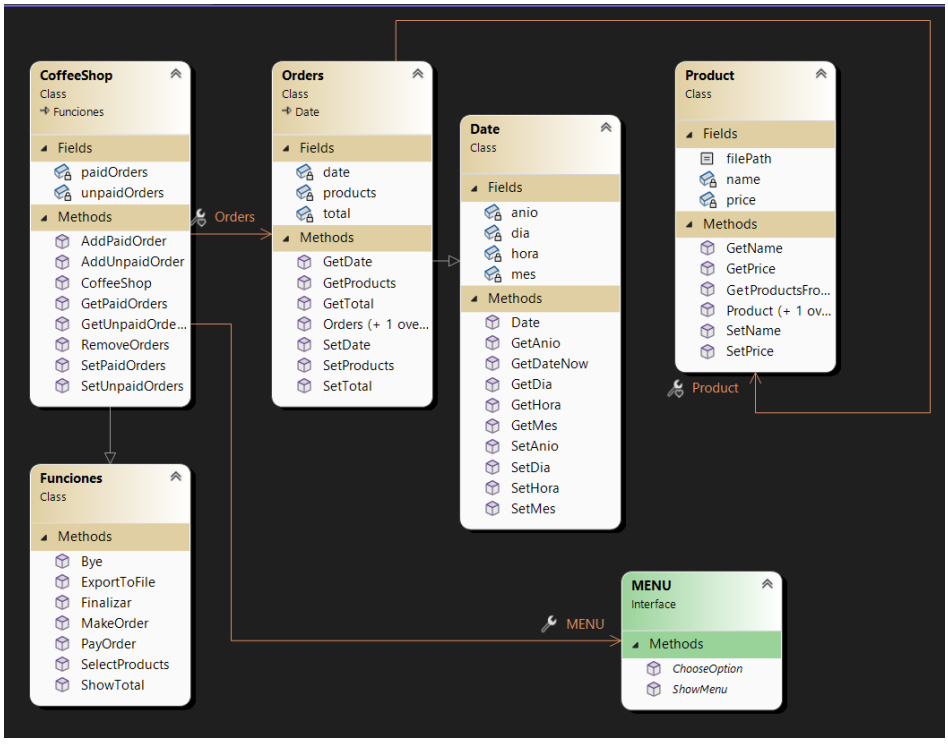
  La cafetería tendrá que ir anotando los pedidos en una cola de pedidos que solo admite un máximo de 5, de manera que si la cola está llena no se podrán añadir más pedidos hasta que se sirva alguno de los pendientes.

  Los pedidos han de registrar un conjunto de productos y la fecha en la que fue pedido (dia, mes, año y hora).

  Los pedidos se sirven por orden de llegada, cuando el usuario así lo elija en el menú, teniendo que mostrar el coste que tenía el pedido servido.

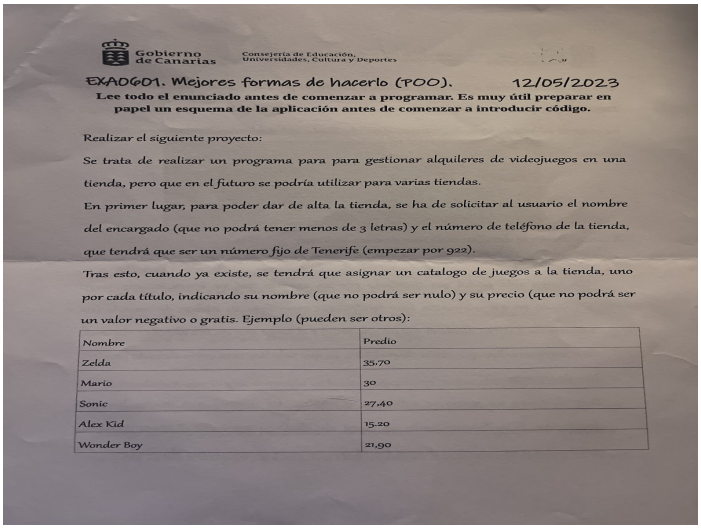
  El programa tendrá que permitir "hacer caja" mostrando todos los pedidos servidos hasta el momento, con sus productos y precio de pedido, y el total de dinero recaudado hasta el momento.

- **Estado:** FINALIZADO
- **Opción:** 1 (Generamos diagramas a partir del mismo)



EJERCICIO 2:

- **Enunciado:** EXAMEN FALLIDO DE OOP



Gobierno de Canarias — Consejería de Educación, Universidades, Cultura y Deportes

EXAD601. Mejores formas de hacerlo (POO). 12/05/2023

Lee todo el enunciado antes de comenzar a programar. Es muy útil preparar en papel un esquema de la aplicación antes de comenzar a introducir código.

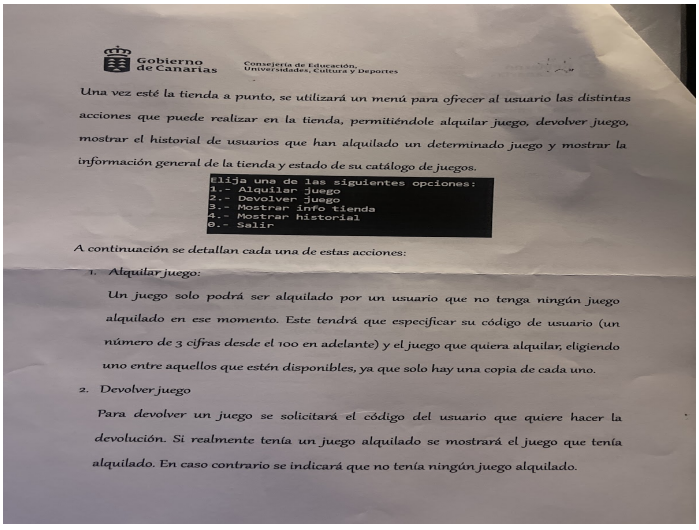
Realizar el siguiente proyecto:

Se trata de realizar un programa para para gestionar alquileres de videojuegos en una tienda, pero que en el futuro se podría utilizar para varias tiendas.

En primer lugar, para poder dar de alta la tienda, se ha de solicitar al usuario el nombre del encargado (que no podrá tener menos de 3 letras) y el número de teléfono de la tienda, que tendrá que ser un número fijo de Tenerife (empezar por 922).

Tras esto, cuando ya existe, se tendrá que asignar un catalogo de juegos a la tienda, uno por cada título, indicando su nombre (que no podrá ser nulo) y su precio (que no podrá ser un valor negativo o gratis. Ejemplo (pueden ser otros):

Nombre	Predio
Zelda	35,70
Mario	30
Sonic	27,40
Alex Kid	15,20
Wonder Boy	21,90



Gobierno de Canarias — Consejería de Educación, Universidades, Cultura y Deportes

Una vez esté la tienda a punto, se utilizará un menú para ofrecer al usuario las distintas acciones que puede realizar en la tienda, permitiéndole alquilar juego, devolver juego, mostrar el historial de usuarios que han alquilado un determinado juego y mostrar la información general de la tienda y estado de su catálogo de juegos.

```
Elija una de las siguientes opciones:
1.- Alquilar juego
2.- Devolver juego
3.- Mostrar info tienda
4.- Mostrar historial
0.- Salir
```

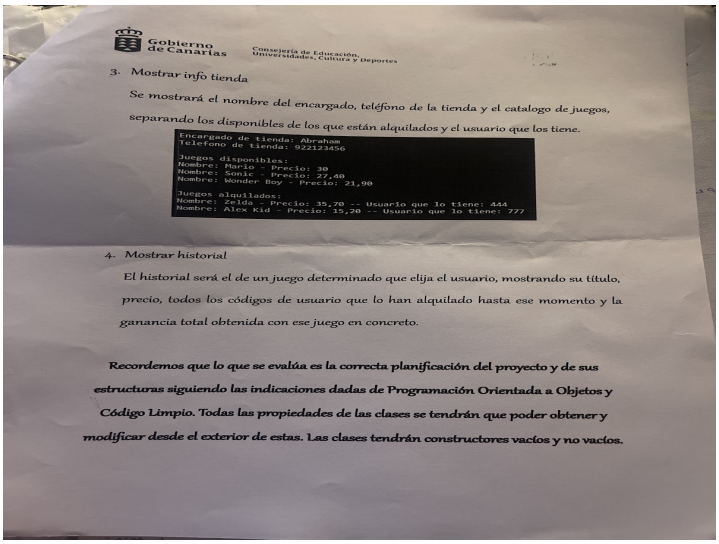
A continuación se detallan cada una de estas acciones:

1. *Alquilar juego:*

   Un juego solo podrá ser alquilado por un usuario que no tenga ningún juego alquilado en ese momento. Este tendrá que especificar su código de usuario (un número de 3 cifras desde el 100 en adelante) y el juego que quiera alquilar, eligiendo uno entre aquellos que estén disponibles, ya que solo hay una copia de cada uno.

2. *Devolver juego*

   Para devolver un juego se solicitará el código del usuario que quiere hacer la devolución. Si realmente tenía un juego alquilado se mostrará el juego que tenía alquilado. En caso contrario se indicará que no tenía ningún juego alquilado.



Gobierno de Canarias — Consejería de Educación, Universidades, Cultura y Deportes

3. *Mostrar info tienda*

   Se mostrará el nombre del encargado, teléfono de la tienda y el catalogo de juegos, separando los disponibles de los que están alquilados y el usuario que los tiene.

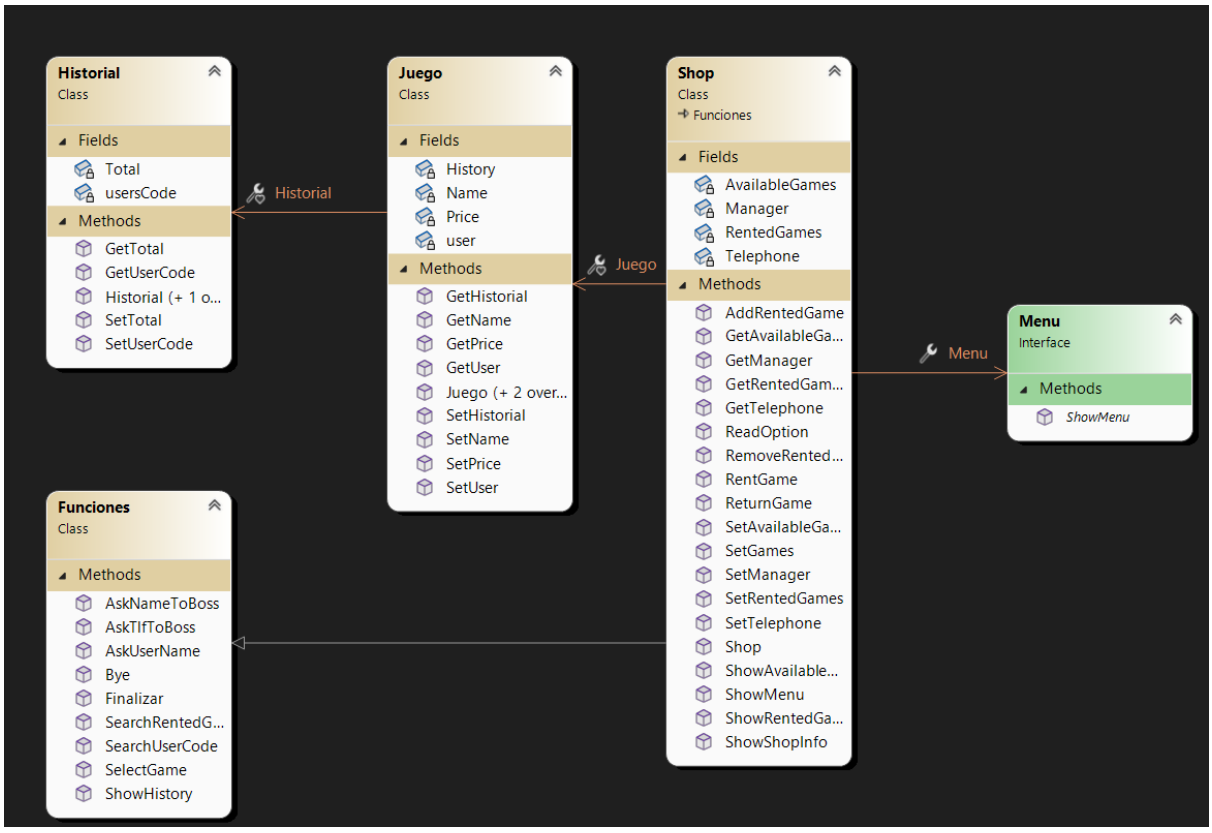
```
Encargado de tienda: Abraham
Telefono de tienda: 922123456
Juegos disponibles:
Nombre: Mario - Precio: 30
Nombre: Sonic - Precio: 27,40
Nombre: Wonder Boy - Precio: 21,90
Juegos alquilados:
Nombre: Zelda - Precio: 35,70 -- Usuario que lo tiene: 444
Nombre: Alex Kid - Precio: 15,20 -- Usuario que lo tiene: 777
```

4. *Mostrar historial*

   El historial será el de un juego determinado que elija el usuario, mostrando su título, precio, todos los códigos de usuario que lo han alquilado hasta ese momento y la ganancia total obtenida con ese juego en concreto.

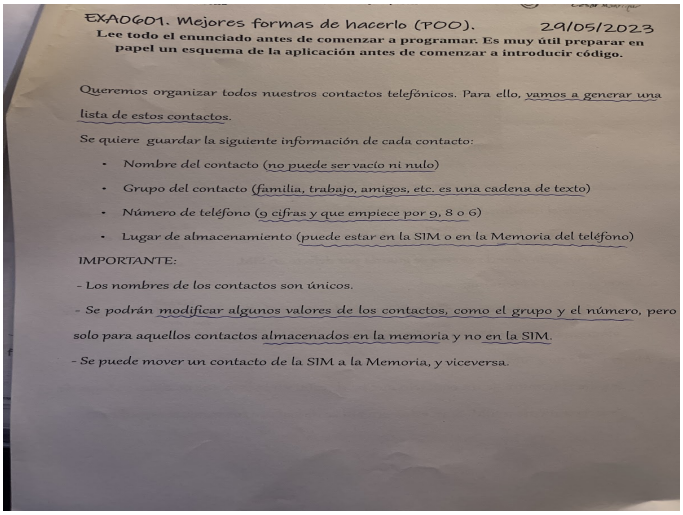
Recordemos que lo que se evalúa es la correcta planificación del proyecto y de sus estructuras siguiendo las indicaciones dadas de Programación Orientada a Objetos y Código Limpio. Todas las propiedades de las clases se tendrán que poder obtener y modificar desde el exterior de estas. Las clases tendrán constructores vacíos y no vacíos.

- **Estado:** SEMIACABADO
- **Opción:** 1 (Generamos diagramas a partir del mismo)



EJERCICIO 3:

- **Enunciado:** EXAMEN RECUPERACION-POO-C#



EXA0601. Mejores formas de hacerlo (POO). 29/05/2023

Lee todo el enunciado antes de comenzar a programar. Es muy útil preparar en papel un esquema de la aplicación antes de comenzar a introducir código.

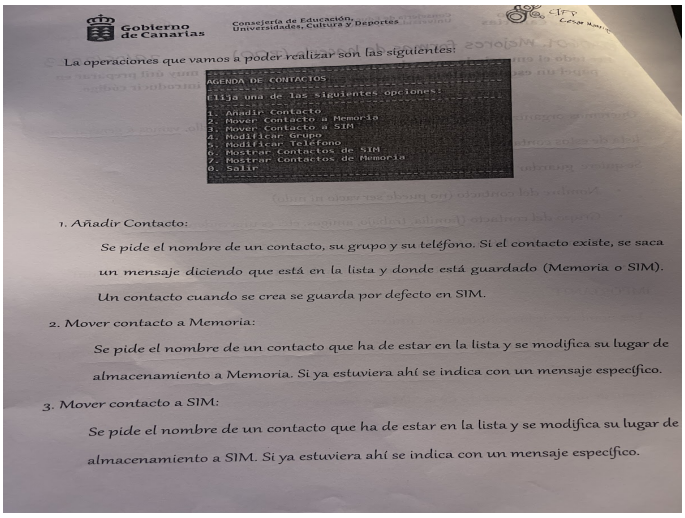
Queremos organizar todos nuestros contactos telefónicos. Para ello, vamos a generar una lista de estos contactos.

Se quiere guardar la siguiente información de cada contacto:

- Nombre del contacto (no puede ser vacío ni nulo)
- Grupo del contacto (familia, trabajo, amigos, etc. es una cadena de texto)
- Número de teléfono (9 cifras y que empiece por 9, 8 o 6)
- Lugar de almacenamiento (puede estar en la SIM o en la Memoria del teléfono)

IMPORTANTE:

- Los nombres de los contactos son únicos.
- Se podrán modificar algunos valores de los contactos, como el grupo y el número, pero solo para aquellos contactos almacenados en la memoria y no en la SIM.
- Se puede mover un contacto de la SIM a la Memoria, y viceversa.



Gobierno de Canarias Consejería de Educación, Universidades, Cultura y Deportes

La operaciones que vamos a poder realizar son las siguientes:

```
AGENDA DE CONTACTOS
Elija una de las siguientes opciones:
1. Añadir Contacto
2. Mover Contacto a Memoria
3. Mover Contacto a SIM
4. Modificar Grupo
5. Modificar Telefono
6. Mostrar Contactos de SIM
7. Mostrar Contactos de Memoria
8. Salir
```

1. Añadir Contacto:

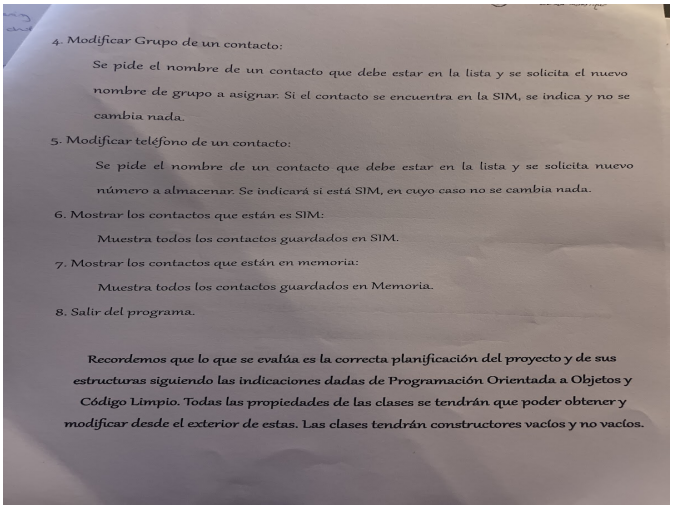
   Se pide el nombre de un contacto, su grupo y su teléfono. Si el contacto existe, se saca un mensaje diciendo que está en la lista y donde está guardado (Memoria o SIM). Un contacto cuando se crea se guarda por defecto en SIM.

2. Mover contacto a Memoria:

   Se pide el nombre de un contacto que ha de estar en la lista y se modifica su lugar de almacenamiento a Memoria. Si ya estuviera ahí se indica con un mensaje específico.

3. Mover contacto a SIM:

   Se pide el nombre de un contacto que ha de estar en la lista y se modifica su lugar de almacenamiento a SIM. Si ya estuviera ahí se indica con un mensaje específico.



4. Modificar Grupo de un contacto:

   Se pide el nombre de un contacto que debe estar en la lista y se solicita el nuevo nombre de grupo a asignar. Si el contacto se encuentra en la SIM, se indica y no se cambia nada.

5. Modificar teléfono de un contacto:

   Se pide el nombre de un contacto que debe estar en la lista y se solicita nuevo número a almacenar. Se indicará si está SIM, en cuyo caso no se cambia nada.

6. Mostrar los contactos que están es SIM:

   Muestra todos los contactos guardados en SIM.

7. Mostrar los contactos que están en memoria:

   Muestra todos los contactos guardados en Memoria.

8. Salir del programa.

Recordemos que lo que se evalúa es la correcta planificación del proyecto y de sus estructuras siguiendo las indicaciones dadas de Programación Orientada a Objetos y Código Limpio. Todas las propiedades de las clases se tendrán que poder obtener y modificar desde el exterior de estas. Las clases tendrán constructores vacíos y no vacíos.

- **Estado:** FINALIZADO
- **Opción:** 1 (Genera el diagrama a partir del mismo)