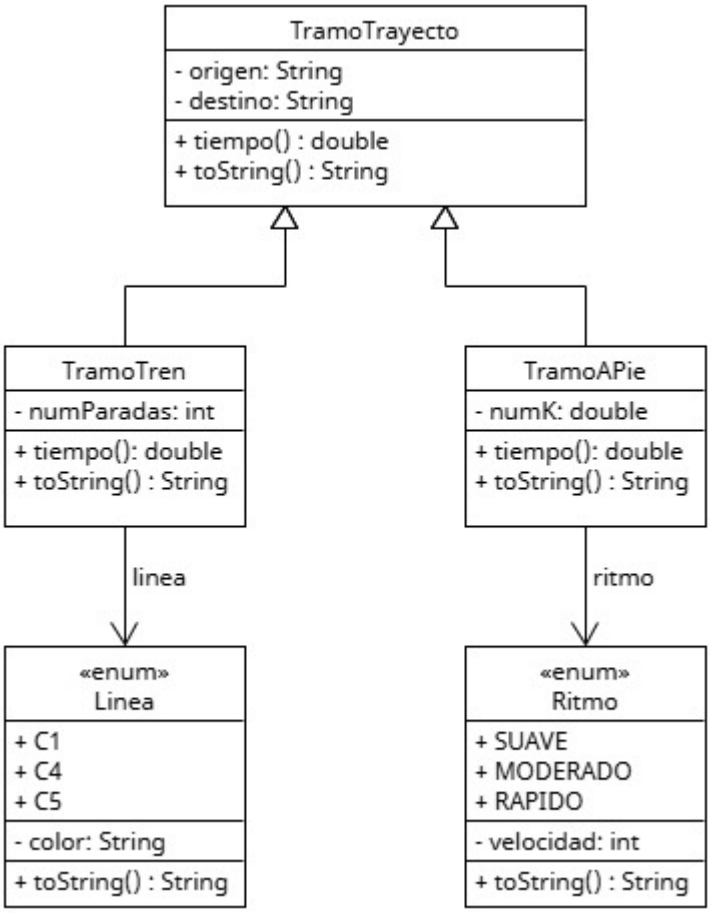
Alejandro González y Fernando Blanco

Memoria Práctica 2

ADSOF

Diagrama UML apartado 1:



El diagrama muestra la clase abstracta TramoTrayecto con sus hijas TramoTren y TramoAPie con sus respectivos atributos y métodos, ambas clases hijas contienen un enum para optimizar los objetos. El diagrama muestra el esquema que sigue el código adjuntado.

Apartado 2:

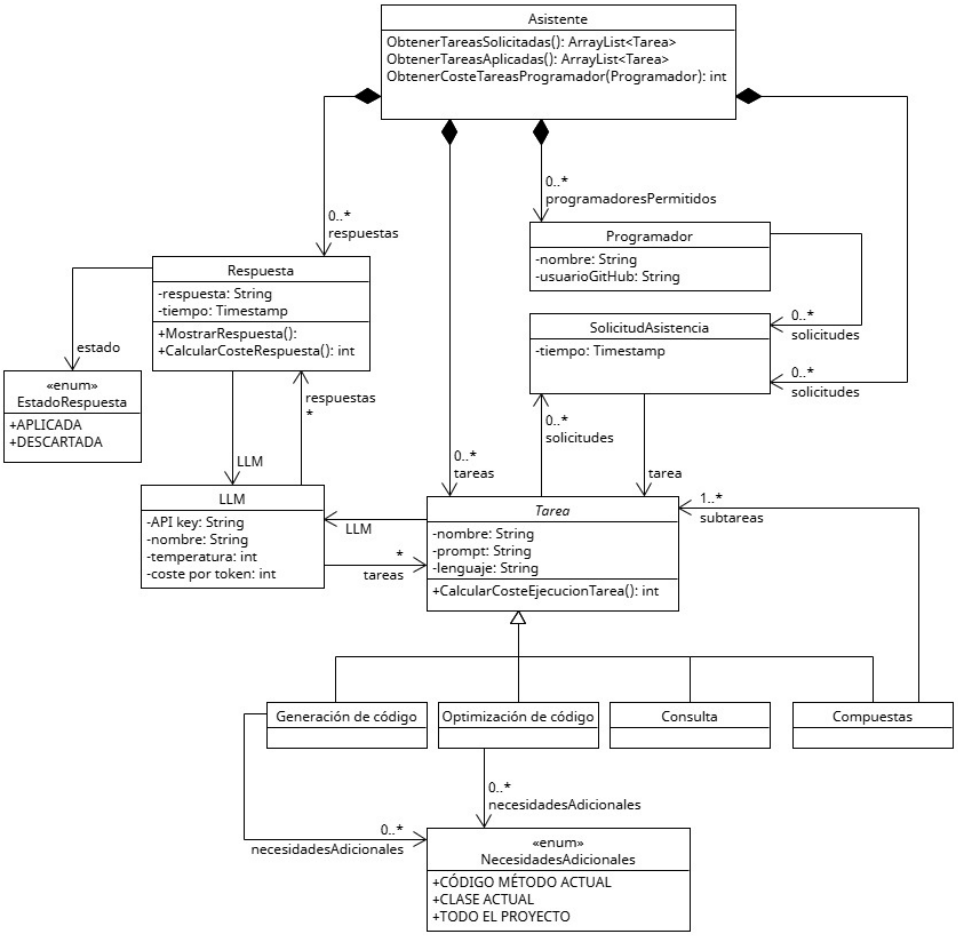
El siguiente diagrama de estados muestra un asistente que resuelve una serie de tareas predefinidas utilizando distintos LLMs.

Las tareas tienen una prompt y pueden ser de cuatro tipos: de generación de código, optimización de código, consulta y compuestas.

Cada tarea tiene un LLM para resolver la prompt, este LLM genera una respuesta a la consulta dada.

El asistente puede ser utilizado por una serie de programadores que generan solicitudes de asistencia, estas solicitudes se corresponden con una tarea existente. Al procesarlas se muestra la respuesta de la tarea al programador, que puede aplicarla o descartarla.

El coste de ejecución de una tarea se calcula viendo el número de tokens (palabras) de una prompt y de la respuesta y multiplicándolo por el coste por token del LLM utilizado para procesar la prompt y generar la respuesta.



El código del apartado c) se encuentra en la carpeta de ej2

Apartado 3:

El diseño original mostraba los siguientes problemas de optimización y uso de recursos:

- Duplicación masiva de código: Las tres clases de suscripción (Papel, Basica, Premium) repetían los mismos atributos como codigoSuscripcion, fechaInicio y fechaFin. Esto hace que cualquier cambio en la lógica base obligue a modificar tres archivos distintos aparte de que no es óptimo.
- Baja eficiencia en las búsquedas: El Suscriptor guardaba una lista de números enteros (codigoSuscripcion: int[*]) en lugar de los objetos reales. Para obtener los detalles de una suscripción, el sistema tendría que buscar ese ID recorriendo todas las listas del Periodico, lo cual es muy lento.
- El Periodico tenía listas específicas para cada tipo de suscripción (sPapel, sBasicas, etc.). Si mañana quieres añadir una "Suscripción Fin de Semana" por ejemplo, tendrías que modificar la clase Periodico para añadir una nueva lista y nuevos métodos de búsqueda.
- Uso de flags: El Suscriptor tenía booleanos como isSuscriptorPremium. Esto es un error de optimización también que se puede solucionar muy fácilmente.

Los cambios realizados:

Clase Abstracta

- Cambio: Crear la clase madre Suscripcion.
- Razón: Centralizas los datos comunes. Ahora, si decides que todas las suscripciones deben tener un "ID de factura" por ejemplo, solo lo añades en un sitio. Las clases hijas solo se encargan de sus particularidades (papel, digital básica y premium).

Asociación Directa

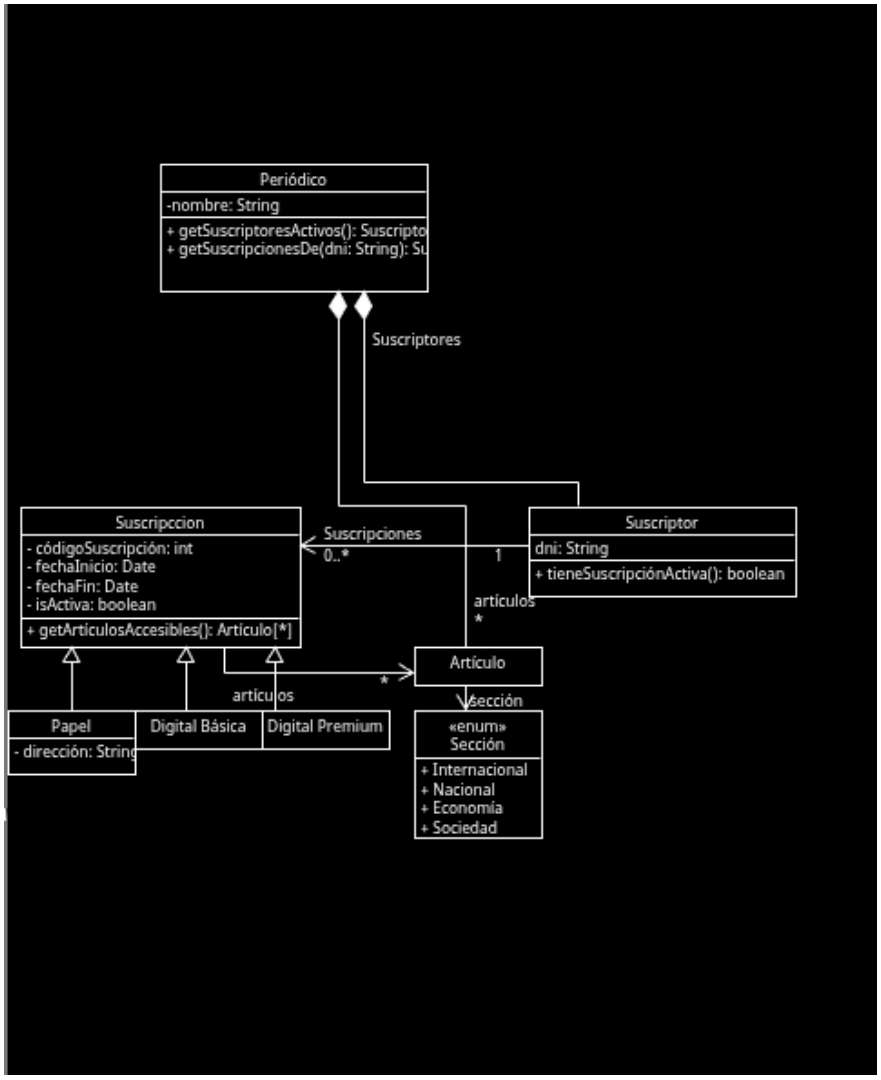
- Cambio: El Suscriptor ahora tiene una relación directa con la clase Suscripcion.
- Razón: Esto permite una navegación instantánea. Si tienes al objeto Suscriptor, ya tienes sus objetos Suscripcion sin tener que ir a buscarlos a ninguna lista global del periódico, también permite evitar los métodos booleanos para ver el tipo de suscripcion.

Polimorfismo con getArticulosAccesibles()

- Cambio: Definir este método en la clase abstracta que se implementa en las hijas.
- Razón: Permite que el código que muestra las noticias sea genérico: suscripcion.getArticulosAccesibles(). No importa si es papel, básica o premium; el objeto "sabe" qué devolver. Esto también optimizaría el hipotético código.

Uso de Enums

- Cambio: Sustituiste el String seccion por un «enum» Seccion.
- Razón: Evitas errores humanos al introducir otro tipo de sección no permitida. Con el enum, el sistema solo acepta los valores predefinidos: Internacional, Nacional, Economía o Sociedad.



Apartado 4:

a)

El siguiente diagrama muestra la aplicación de una pizzeria, que alberga distintos tipos de usuario.

Estos son los empleados (gestor y repartidores) y los clientes.

Además, la aplicación también permite guardar los pedidos de los clientes y trabajar con ellos.

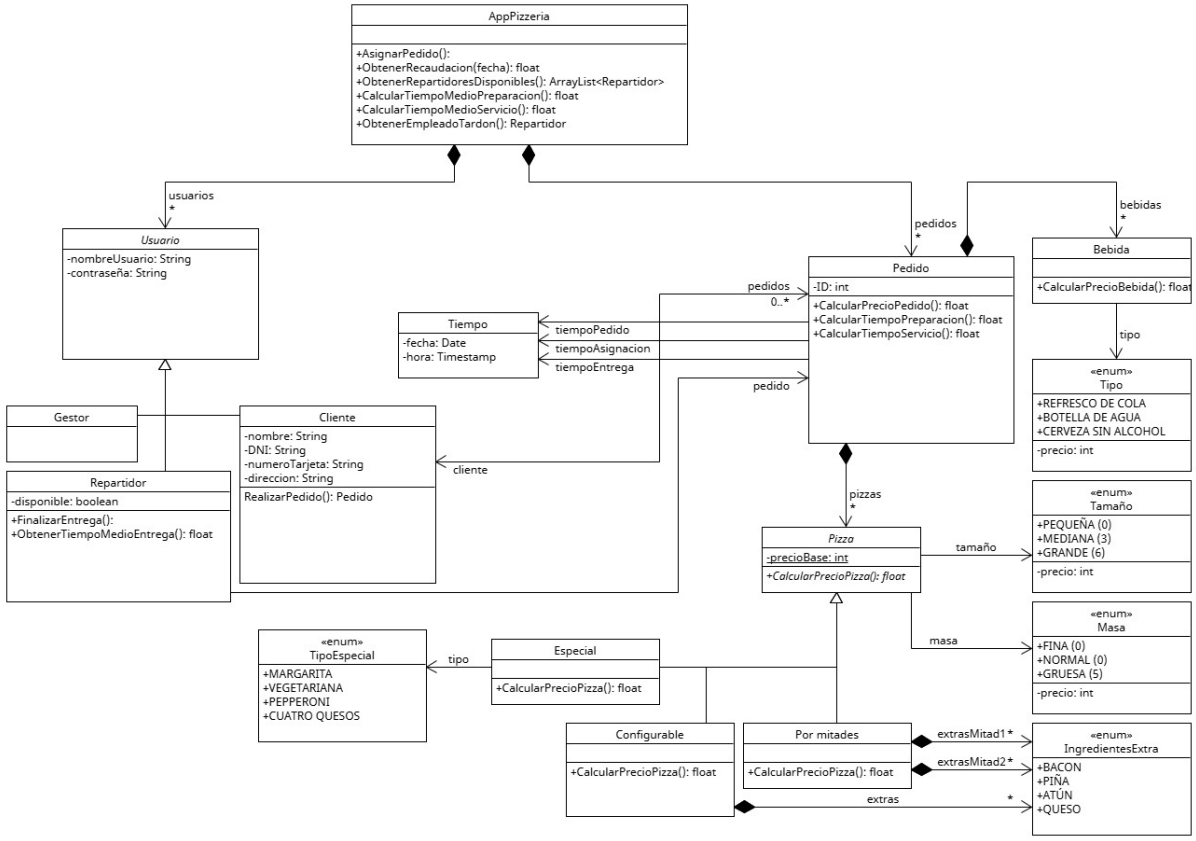
Los pedidos se componen de una serie de pizzas y bebidas. Las pizzas son personalizables y tienen distintos tipos, mientras que las bebidas se eligen entre una selección de tres.

Los pedidos tienen un precio que se calcula según sus componentes, y una serie de tiempos, estos son:

+ Tiempo de pedido: Tiempo en el que se realiza el pedido.

+ Tiempo de asignación: Tiempo en el que el pedido ya ha sido preparado, y es asignado a un repartidor.

+ Tiempo de entrega: Tiempo en el que el pedido llega al cliente.



b)

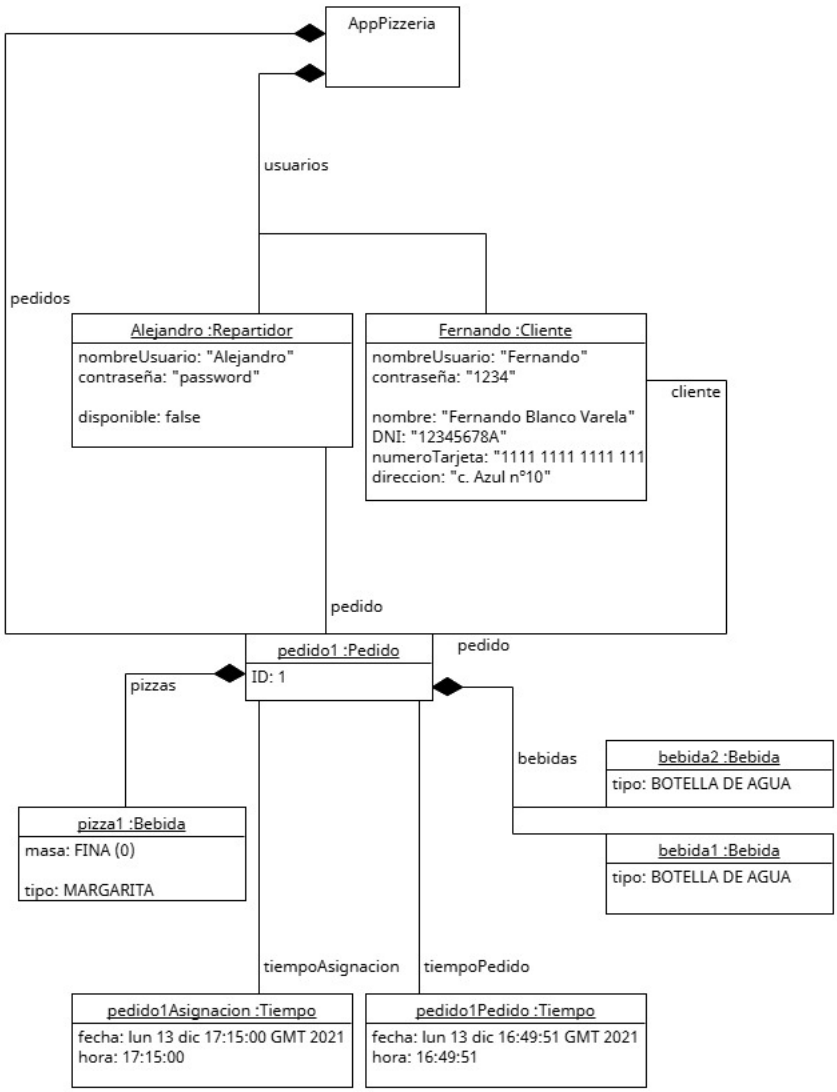
El siguiente diagrama de objetos muestra una posible instancia de nuestra aplicación para la pizzeria.

Encontramos los siguientes objetos:

+ Un repartidor de nombre Alejandro al que se le ha asignado un pedido.

+ Un cliente de nombre Fernando que ha realizado un pedido.

+ Un pedido compuesto de una pizza y dos bebidas. Este pedido ya ha sido asignado a un repartidor, pero todavía no ha sido entregado.


AppPizzeria
usuarios
pedidos
Alejandro :Repartidor
nombreUsuario: "Alejandro"
contraseña: "password"
disponible: false
Fernando :Cliente
nombreUsuario: "Fernando"
contraseña: "1234"
nombre: "Fernando Blanco Varela"
DNI: "12345678A"
numeroTarjeta: "1111 1111 1111 111
direccion: "c. Azul nº10"
cliente
pedido
pedido1 :Pedido
ID: 1
pedido
pizzas
bebidas
bebida2 :Bebida
tipo: BOTELLA DE AGUA
pizza1 :Bebida
masa: FINA (0)
tipo: MARGARITA
bebida1 :Bebida
tipo: BOTELLA DE AGUA
tiempoAsignacion
tiempoPedido
pedido1Asignacion :Tiempo
fecha: lun 13 dic 17:15:00 GMT 2021
hora: 17:15:00
pedido1Pedido :Tiempo
fecha: lun 13 dic 16:49:51 GMT 2021
hora: 16:49:51