UP
Universidad de Palermo

Técnicas Avanzadas de Programación

SOLID

Participante:

- Manuel Krivoy 0125450

Docente:

- María Alegre

Fecha:

06/09/2025

Condiciones: se debe a partir de los conocimientos adquiridos en el módulo y los conocimientos previos de programación orientada a objetos sobre el lenguaje JAVA, generar el código que mejor se adapta a la solución.

Entregables: código en Java. Adjunto PDF con las consideraciones y justificaciones de las decisiones tomadas.

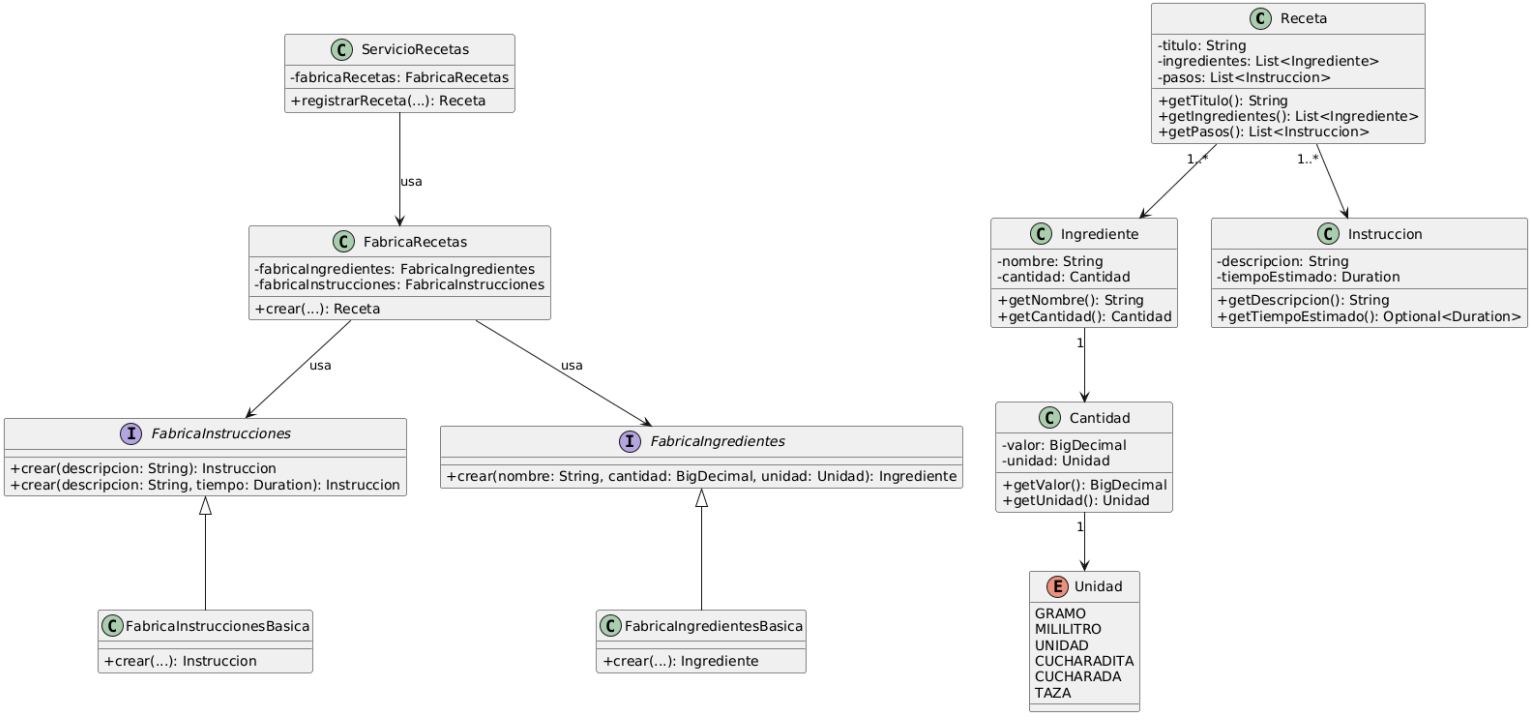
Fecha de entrega: 1 semana posterior a la fecha de publicación del módulo.

Condiciones de aprobación: la actividad se calificará con una puntuación máxima de 10, la demora en la entrega implicará una disminución en la nota máxima alcanzable.

Consigna: se quieren generar las clases correspondientes a una receta de cocina. Constará de una clase receta, la cual se compondrá de una clase ingrediente y otra clase instrucción. La clase principal podrá tener N ingredientes y N pasos.

Se deberá aplicar inyección de dependencias en el modelo, justificando su implementación.

Diagrama UML del sistema:



Justificación y explicación del diseño

1. **Clase Unidad (enum)**
   Se definió un tipo enumerado que contiene las posibles unidades de medida (gramo, mililitro, unidad, cucharadita, cucharada, taza). Esto evita el uso de cadenas de texto libres y brinda seguridad tipada al momento de definir cantidades de ingredientes. Además, facilita la validación y estandarización dentro del sistema, permitiendo extender las unidades sin modificar la lógica del resto de las clases.
2. **Clase Cantidad**
   Esta clase modela la cantidad de un ingrediente con dos atributos: el valor numérico (BigDecimal) y la unidad (Unidad). Se eligió BigDecimal para soportar cálculos precisos y evitar errores de redondeo comunes con double o float. Se validó que el valor sea positivo y que la unidad sea obligatoria. Esta clase encapsula la lógica de validación y asegura consistencia en los datos que recibe la clase Ingrediente.
3. **Clase Ingrediente**
   Representa un ingrediente de la receta. Está compuesta por un nombre (String) y una cantidad (Cantidad). La clase valida que el nombre no sea nulo ni vacío, y que la cantidad esté siempre presente. Esta separación permite abstraer el concepto de cantidad como una clase independiente, lo que aporta flexibilidad en el modelo y permite reutilizar lógica común de validación en distintos ingredientes.
4. **Clase Instruccion**
   Modela un paso dentro de la receta. Se compone de una descripción obligatoria (String) y un tiempo estimado opcional (Duration). La inclusión del tiempo es opcional para mantener flexibilidad, ya que no todos los pasos de cocina requieren un tiempo asociado. La clase también valida que la descripción no sea nula ni vacía. Con esta estructura se pueden crear pasos simples o más detallados dependiendo de las necesidades.
5. **Clase Receta**
   Es la entidad principal del modelo. Contiene un título, una lista de ingredientes y una lista de instrucciones. Se validó que la receta siempre tenga al menos un ingrediente y un paso para garantizar su coherencia. Además, se utilizó List.copyOf para proteger las listas de modificaciones externas, y se devuelve una vista inmodificable de los ingredientes y pasos. Esto asegura que una receta, una vez creada,

no pueda alterarse de manera insegura desde afuera.

6. **Interfaces de fábricas (FabricaIngredientes y FabricaInstrucciones)**
   Una fábrica es un patrón de diseño que tiene como objetivo centralizar la creación de objetos. En lugar de instanciar directamente con new, el código de más alto nivel utiliza las fábricas, lo que le da flexibilidad y reduce el acoplamiento.
   Las interfaces definen el contrato de cómo se deben crear ingredientes o instrucciones, pero no se preocupan por la implementación concreta. De esta forma, si en el futuro se quiere cambiar la forma de creación (por ejemplo, validando datos adicionales o consultando un repositorio externo), basta con crear una nueva implementación de la fábrica.
7. **Implementaciones básicas de fábricas (FabricaIngredientesBasica y FabricaInstruccionesBasica)**
   Estas clases son las implementaciones por defecto de las interfaces de fábrica. Cumplen con el contrato definido en las interfaces y se encargan de crear objetos del modelo de la manera más directa posible. **Funcionan como un "DAO" en el sentido de que encapsulan la lógica de creación, ocultando al resto del sistema cómo se instancia realmente cada objeto**. Así, si se reemplaza la implementación, no es necesario modificar el código que las utiliza.

8. **Clase FabricaRecetas**
   Esta clase compone a las dos fábricas anteriores (FabricaIngredientes y FabricaInstrucciones). Su objetivo es coordinar la creación de recetas completas a partir de ingredientes e instrucciones. El diseño cumple con el principio de inversión de dependencias (D de SOLID), ya que la fábrica de recetas no depende de implementaciones concretas sino de interfaces. Esto facilita la extensibilidad y el testeo.
9. **Clase ServicioRecetas**
   Representa la capa de aplicación que utiliza la fábrica de recetas para registrar nuevas recetas. Este servicio recibe por inyección de dependencias la instancia de FabricaRecetas y ofrece un método de alto nivel para crear recetas. Su rol es separar la lógica de negocio (cómo se gestionan las recetas) de la lógica de creación de objetos (cómo se instancian los ingredientes, instrucciones y recetas).