Fundamentos de Ingeniería del Software

Apuntes Control 2

Ismael Sallami Moreno

Mayo de 2025

Índice general

1 Control 2: FIS **1**

1.1 Guía Detallada para Crear Diagramas de Comunicación 1

1.2 Apéndice: Uso de Estereotipos en UML 4

1.3 Guía General Paso a Paso para el Diagrama de Clases del Diseño 6

1 Control 2: FIS

1.1. Guía Detallada para Crear Diagramas de Comunicación

Este proceso nos ayudará a traducir los requisitos de una operación (descritos en su contrato) y la estructura de nuestro dominio (descrita en el modelo conceptual) en un diagrama que muestre cómo los objetos colaboran.

1.1.1. Preparación: Entender los Insumos

Antes de dibujar, debemos asegurarnos de comprender bien:

1. **El Contrato de la Operación:**
    - **Nombre y Parámetros**: Identificamos el nombre exacto de la operación y los parámetros que recibe. Esto será nuestro mensaje inicial.
    - **Responsabilidad**: Nos da una idea general de lo que la operación debe lograr.
    - **Tipo (Clase Responsable)**: Indica qué clase del sistema recibe el mensaje inicial. Este es nuestro primer objeto participante después del actor.
    - **Poscondiciones**: ¡Son cruciales! Describen el estado del sistema *después* de que la operación se ejecuta. Buscamos frases como:
        - "Se creó un objeto de la clase X…" (implica un mensaje de creación).
        - "Se creó un enlace entre objetoA y objetoB…" (implica que un objeto guarda una referencia a otro).
        - "El atributo Z del objeto W fue actualizado…" (implica un mensaje para modificar un atributo).
    - **Salida**: Define qué información debe devolver la operación. Esto nos ayuda a identificar los mensajes finales para recolectar datos.
    - **Excepciones**: Nos dan pistas sobre validaciones o búsquedas que podrían fallar (ej. "No existe el objeto X" implica que primero hay que buscar X).
2. **El Modelo Conceptual:**
    - **Clases y Atributos**: Identificamos las clases relevantes que participarán y los atributos que almacenan la información. Las poscondiciones suelen referirse a estas clases y atributos.
    - **Relaciones (Asociaciones, Composiciones, Agregaciones)**: Estas relaciones sugieren cómo los objetos podrían estar conectados y quién podría necesitar enviar mensajes a quién.

- Una composición (rombo relleno) implica que el objeto "contenedor" es responsable de crear/gestionar las partes.
- Una asociación simple indica que los objetos pueden interactuar.

1.1.2. Pasos para Dibujar el Diagrama de Comunicación

1. **Identificar el Actor y el Punto de Entrada:**
   - Dibujamos el **actor** que inicia la operación (puede ser un rol de usuario genérico si no se especifica).
   - Dibujamos el **objeto del sistema** que recibe el primer mensaje (indicado en el campo "Tipo" del contrato, ej. `:SGMenu`).
2. **Dibujar el Mensaje Inicial:**
   - Trazamos una flecha desde el actor al objeto del sistema.
   - Etiquetamos la flecha con el **nombre de la operación y sus parámetros** (del contrato, ej. `1: resultado = nombreOperacion(param1, param2)`). Asignamos un número de secuencia (ej. 1).
3. **Traducir las Poscondiciones en Interacciones (Iterativo):** Este es el núcleo del proceso. Para cada poscondición:
   - **Creación de Objetos**:
     - Si la poscondición dice "Se creó un objeto `miObjeto` de la clase `ClaseX`...".
     - Decidimos **quién es responsable** de crear `ClaseX`. A menudo es el objeto del sistema o un objeto que "contiene" a `ClaseX` según el modelo conceptual.
     - Dibujamos un mensaje desde el objeto responsable hacia la clase `ClaseX` (o un objeto existente de `ClaseX` si es un patrón factory, pero usualmente es a la clase para indicar instanciación).
     - Etiquetamos el mensaje como `N: miObjeto = crear(params_creacion)` (donde `N` es el siguiente número de secuencia). Los `params_creacion` son los datos necesarios para inicializar el objeto, que pueden venir de los parámetros de la operación principal o de otros objetos.
     - Dibujamos el nuevo objeto `miObjeto:ClaseX`. Podemos usar estereotipos como `<<L>>` (local) o `<<P>>` (parámetro) si es relevante.
   - **Establecimiento de Enlaces**:
     - Si la poscondición dice "Se creó un enlace entre `objetoA` y `objetoB`...".
     - `objetoA` necesita conocer a `objetoB`. Esto se logra generalmente de dos maneras:
       1. `objetoA` envía un mensaje a sí mismo para almacenar `objetoB` (ej. `N: incluir(objetoB)`).
       2. `objetoB` es pasado como parámetro al crear `objetoA`, o a un método "setter" en `objetoA`.
     - Nos aseguramos de que `objetoB` ya exista o sea creado antes de este paso.

    - Los enlaces permanentes se suelen marcar con <<A>> (asociación) en el diagrama de comunicación.
- **Modificación de Atributos**:
    - Si la poscondición dice "El atributo Z del `objetoW` fue actualizado a `valor`...".
    - Dibujamos un mensaje hacia `objetoW` (ej. `N: setZ(valor)`).

4. **Obtención de Datos / Colaboraciones Intermedias:**
    - A menudo, para cumplir una poscondición o preparar la salida, un objeto necesitará información de otro.
    - Usamos el modelo conceptual para ver quién tiene qué dato.
    - Dibujamos mensajes de "solicitud de datos" (ej. `N: dato = getInfo()`).
    - Si la operación implica una búsqueda (ej. "buscar profesor por DNI"), representamos ese mensaje. Si la búsqueda puede fallar (como sugiere una excepción en el contrato), el diagrama muestra el flujo exitoso, pero tenemos en cuenta la lógica.
5. **Manejo de Colecciones / Bucles:**
    - Si una poscondición o parte de la lógica se aplica a "todos los elementos de una colección" (ej. "para todos los Platos"):
        - El objeto que posee la colección itera sobre ella.
        - Los mensajes que se repiten dentro del bucle se prefijan con `N.M* [condición_iteración]: ...` (ej. `2.1* [i=1..3]: ...`).
        - Primero, puede haber un mensaje para obtener el siguiente elemento (ej. `receta = siguiente()`).
        - Luego, los mensajes que operan sobre ese elemento.
6. **Preparación de la Salida:**
    - Consultamos la sección "Salida" del contrato.
    - El objeto del sistema (o un objeto delegado) es responsable de ensamblar esta salida.
    - Esto puede implicar enviar mensajes a otros objetos para obtener las piezas de información necesarias (ej. `obtenerDatosMenu()` que a su vez llama a `obtenerNombreReceta()`).
    - El valor de retorno final se muestra en el mensaje inicial (ej. `datosMenú = nuevoMenú(...)`).
7. **Numeración y Estereotipos:**
    - Numeramos los mensajes secuencialmente (1, 2, 2.1, 2.1.1, 3, etc.) para indicar el orden.
    - Utilizamos estereotipos como <<A>> (asociación), <<L>> (variable local), <<P>> (parámetro) para clarificar la naturaleza de los objetos y enlaces, como se ve en el ejemplo.
8. **Revisión y Refinamiento:**
    - **Completitud**: ¿Todas las poscondiciones están cubiertas por alguna secuencia de mensajes? ¿Se genera la salida especificada?
    - **Consistencia**: ¿Las interacciones son coherentes con el modelo conceptual (quién conoce a quién, quién tiene qué datos)?

- **Claridad**: ¿El diagrama es fácil de entender? ¿La numeración es correcta?

1.1.3. Ejemplo Rápido Aplicado (Conceptual)

- **Contrato**: `crearPedido(idCliente, listaItems)`
  - **Poscondiciones**:
    1. Se creó un objeto `Pedido`.
    2. Se asoció el `Cliente` (identificado por `idCliente`) al `Pedido`.
    3. Para cada `item` en `listaItems`, se creó un objeto `LineaPedido` y se asoció al `Pedido`.
  - **Conceptual**: `Sistema` – `Pedido` – `LineaPedido`; `Pedido` – `Cliente`.
- **Diagrama (resumido)**:
  1. `Actor` -> `:Sistema`: `p = crearPedido(idCliente, listaItems)`
  2. `:Sistema` -> `Cliente`: `cliente = buscar(idCliente)` (asumimos que hay que buscarlo)
  3. `:Sistema` -> `Pedido`: `miPedido = crear(cliente)`
  4. `:Sistema` -> `:Sistema`: `incluirPedido(miPedido)` (o similar, para guardar el pedido)
  5. `miPedido` -> `LineaPedido`: `* [para cada item]: linea = crear(item)` (bucle)
  6. `miPedido` -> `miPedido`: `* [para cada linea]: incluirLinea(linea)`

Esta guía nos proporciona un marco de trabajo. Cada problema tendrá sus matices, pero los principios de derivar interacciones de las responsabilidades y poscondiciones, usando el modelo conceptual como mapa, son universales.

1.2. Apéndice: Uso de Estereotipos en UML

Los estereotipos en UML son una forma de extender el vocabulario del lenguaje, permitiéndonos definir nuevos tipos de elementos de modelado o añadir semántica específica a los existentes. Se representan generalmente con el nombre del estereotipo entre dobles comillas angulares (ej. `<<estereotipo>>`).

Aquí explicamos cuándo usar algunos de los estereotipos comunes, especialmente en el contexto de los diagramas de comunicación y de clases de diseño que hemos estado viendo, basándonos en los ejemplos y la teoría:

1.2.1. Estereotipos en Diagramas de Comunicación

Estos estereotipos se suelen aplicar a los enlaces entre los objetos (lifelines) o a los propios objetos para clarificar su rol o naturaleza en la interacción:

1. **<<A>> (Asociación)**
   - **Cuándo usarlo**: Lo utilizamos en el enlace entre dos objetos para indicar que existe una asociación estructural y duradera entre ellos en el diagrama de clases. Los objetos se conocen y pueden interactuar más allá del contexto de la operación actual.
   - **Ejemplo**: En el diagrama de comunicación de la solución, el enlace entre `:SGMenú` y `miMenú:Menú` (una vez que `miMenú` es "incluido" en el sistema) o entre `plato:Plato` y su `:Receta` llevan este estereotipo.
2. **<<L>> (Local)**
   - **Cuándo usarlo**: Indica que un objeto es una variable local dentro del ámbito de la operación que estamos describiendo, o es creado y utilizado solo durante esa operación, sin que necesariamente exista una relación persistente con otros objetos a largo plazo.
   - **Ejemplo**: `miMenú:Menú` podría considerarse inicialmente local a la operación en `:SGMenú` hasta que se establece una asociación más formal. `miplato:Plato` es creado y utilizado localmente por `miMenú`.
3. **<<P>> (Parámetro)**
   - **Cuándo usarlo**: Lo usamos para indicar que un objeto o una colección de objetos se pasa como parámetro en un mensaje de una operación.
   - **Ejemplo**: En el diagrama de comunicación de la solución, `listaRecetas:Receta` es tratada como un parámetro (o derivada de uno) que `miMenú:Menú` utiliza internamente (implícito en el mensaje `2: crear` y luego en `2.1: receta = siguiente()`).

1.2.2. Estereotipos en Diagramas de Clases de Diseño

1. **Estereotipos de Dependencia (<<dependency>>)**
   - La relación de dependencia indica que un cambio en la especificación de un elemento puede afectar a otro elemento que la usa. Los enlaces estereotipados con <<L>>, <<P>> o <<G>> (Global) en los diagramas de interacción se traducirán en una dependencia en el diagrama de clases de diseño.
   - **<<use>>**: Es un estereotipo genérico para la dependencia. Lo usamos cuando una clase necesita a otra para su correcta implementación o especificación, pero no es una relación estructural fuerte como una asociación.
   - **<<create>> o <<instantiate>>**: Lo usamos para indicar que una clase (el cliente) crea instancias de otra clase (el proveedor). Es una forma específica de dependencia.
   - **<<call>>**: Indica que una clase llama a operaciones de otra.
   - **<<parameter>>**: Si una clase se utiliza como tipo de un parámetro en una operación de otra clase.
   - **<<local>>**: Si una instancia de una clase es una variable local dentro de una operación de otra clase.
   - *Nota*: No es necesario dibujar una dependencia si ya existe una asociación entre las clases, ya que la asociación implica una dependencia.

2. **`<<interface>>`**

    - **Cuándo usarlo**: Para denotar una interfaz, que es un clasificador que declara un conjunto de operaciones (un contrato) que otras clases (que realizan la interfaz) deben implementar. No tiene atributos ni implementaciones de métodos.
    - En la teoría se menciona explícitamente "Interfaces con sus operaciones y constantes" como elementos que puede contener un diagrama de clases de diseño.

3. **Estereotipos para Tipos de Clases (Arquitectónicos o de Análisis - EBC/MVC)**

    - Estos nos ayudan a categorizar las clases según su rol en la arquitectura del sistema.
    - **`<<actor>>`**: Representa un rol de usuario o un sistema externo que interactúa con el sistema modelado.
    - **`<<boundary>>` o `<<interfaz>>` (a veces se usa para clases de interfaz de usuario/sistema)**: Para clases que manejan la comunicación entre el sistema y los actores (ej. ventanas de GUI, APIs).
    - **`<<control>>` o `<<controller>>`**: Para clases que encapsulan la lógica de control de un caso de uso o coordinan las interacciones entre otras clases (ej. `:SGMenú` en la solución tiene un rol de controlador para la operación `nuevoMenú`).
    - **`<<entity>>`**: Para clases que representan objetos del dominio de negocio, usualmente con datos persistentes (ej. `Menú`, `Plato`, `Receta` en la solución son claramente entidades).

Es importante recordar que el uso de estereotipos debe aportar claridad al modelo. No es necesario estereotipar cada elemento si su naturaleza es obvia por el contexto o el nombre. La elección de qué estereotipos usar también puede depender de la metodología de desarrollo o las convenciones del equipo.

1.3. Guía General Paso a Paso para el Diagrama de Clases del Diseño

Este diagrama es esencial porque nos permite visualizar gráficamente las especificaciones de las clases e interfaces del software, así como las relaciones entre ellas dentro de una aplicación. Partimos del modelo de análisis, el modelo conceptual y los diagramas de interacción como referencia principal.

1.3.1. Paso 0: Preparación – Reunimos los Insumos

Antes de comenzar, nos aseguramos de contar con: - **Modelo Conceptual**: Nos proporciona una visión inicial de las clases, atributos y relaciones del dominio. - **Diagramas de Interacción (Secuencia o Comunicación)**: Son fundamentales, ya que muestran objetos específicos interactuando y los mensajes que intercambian para realizar las operaciones del sistema. - **Contratos de Operaciones del Sistema**: Definen el propósito de cada operación, sus parámetros, precondiciones, poscondiciones y salida.

1.3.2. Pasos para la Elaboración

Paso 1: Identificamos y Representamos las Clases

- **Identificación**:
  - A partir de los diagramas de interacción, cada objeto participante se traduce en una clase en el diagrama de clases de diseño.
  - Las clases del modelo conceptual también son candidatas directas.
  - Incorporamos clases de control o interfaz cuando sea necesario, por ejemplo, para manejar la lógica de casos de uso o la interacción con el usuario.
- **Representación**:
  - Dibujamos un rectángulo dividido en tres compartimentos: nombre, atributos y operaciones.
  - Inicialmente, incluimos los atributos identificados en el modelo conceptual y aquellos necesarios según los diagramas de interacción.

Paso 2: Identificamos y Añadimos las Operaciones (Métodos)

- **Identificación**:
  - Cada mensaje en los diagramas de interacción se convierte en una operación en la clase receptora.
  - Las operaciones del sistema definidas en los contratos se reflejan como operaciones públicas.
  - Incluimos constructores y operaciones internas según sea necesario.
- **Representación**:
  - Añadimos el nombre de la operación, sus parámetros y su visibilidad (+, −, #, ~) en el compartimento correspondiente.

Paso 3: Añadimos Tipos a Atributos y Parámetros

- Especificamos el tipo de dato de cada atributo y parámetro, así como el tipo de retorno de las operaciones.
- Indicamos la multiplicidad cuando se trate de colecciones.

Paso 4: Identificamos e Incluimos Asociaciones y su Navegabilidad

- Detectamos asociaciones persistentes a partir de los diagramas de interacción y el modelo conceptual.
- Dibujamos líneas entre las clases asociadas, indicando multiplicidad, navegabilidad y, si es relevante, el rol de cada extremo.
- Representamos agregaciones y composiciones según corresponda.

Paso 5: Identificamos e Incluimos Relaciones de Dependencia

- Añadimos dependencias cuando una clase utiliza otra como parámetro, tipo de retorno o variable local, pero no existe una asociación estructural.
- Representamos estas dependencias con líneas discontinuas y flechas abiertas.
- Evitamos redundancias si ya existe una asociación.

Paso 6: Incluimos Relaciones de Generalización (Herencia)

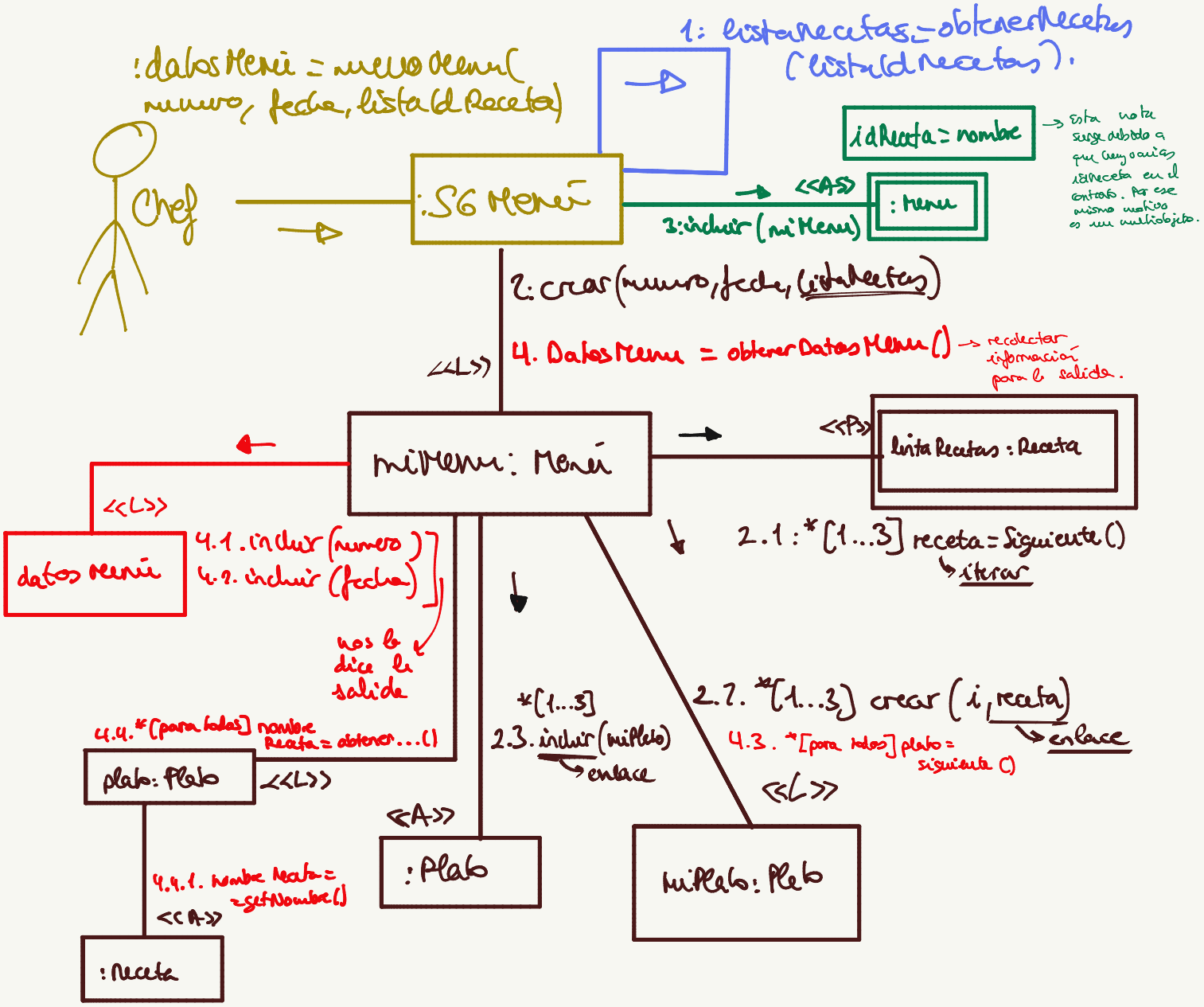
- Buscamos oportunidades de generalización cuando varias clases comparten atributos, operaciones o asociaciones.
- Representamos la herencia con una línea y una flecha triangular hueca desde la subclase hacia la superclase.
- Centralizamos los elementos comunes en la superclase.

1.3.3. Paso Final: Revisión y Refinamiento

- Verificamos la consistencia con los diagramas de interacción, asegurándonos de que cada mensaje tenga su operación correspondiente.
- Comprobamos la completitud del diagrama, la claridad y la ausencia de elementos redundantes.
- Confirmamos que el diseño cubre todos los requisitos funcionales definidos en los contratos y casos de uso.

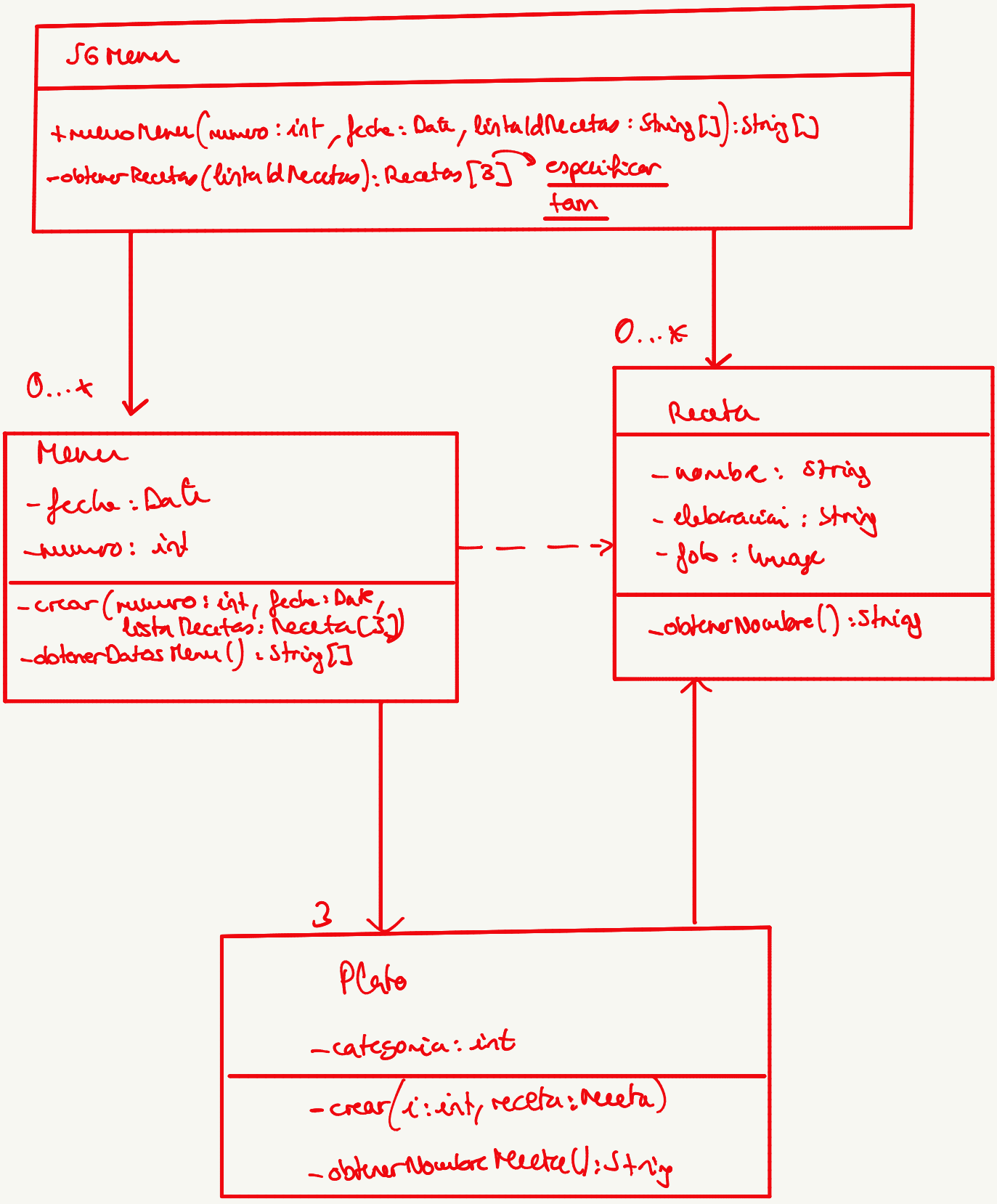
Siguiendo este enfoque iterativo y metódico, podemos construir diagramas de clases de diseño sólidos y alineados con los requisitos del sistema.

DIAGRAMA DE COMUNICACIÓN



Redactamos info para la salida.

DIAGRAMA DE CLASES

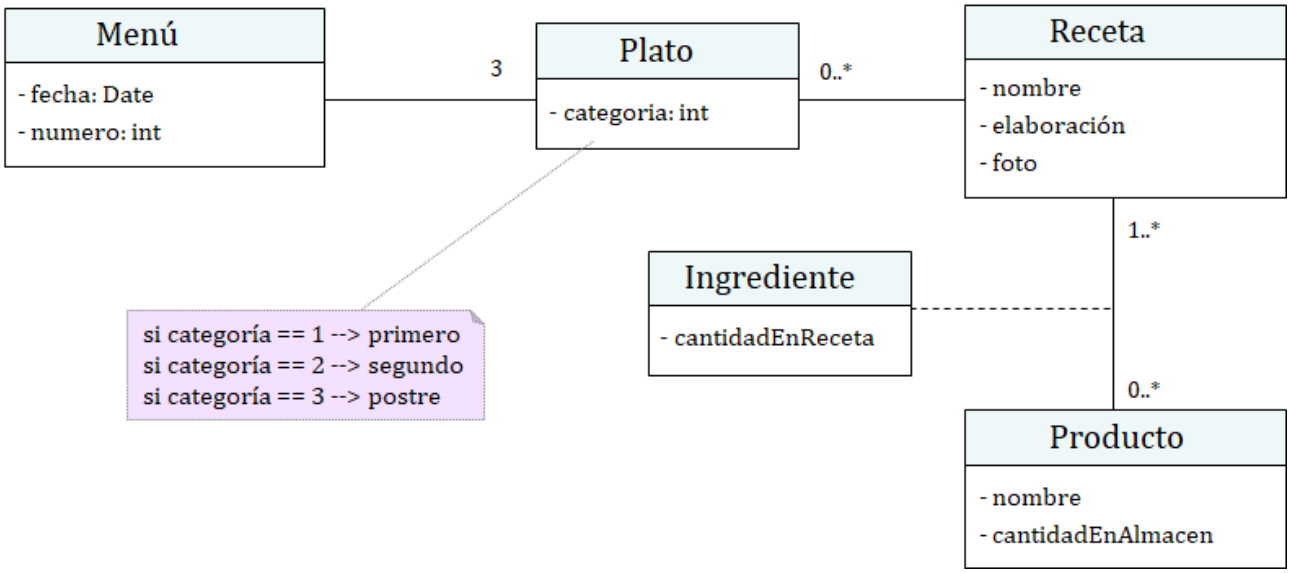


Tomando como base el modelo conceptual y el contrato para la operación nuevoMenú que se adjuntan:

A. Realice el diagrama de comunicación de la operación nuevoMenú**.**

B. Partiendo del diagrama anterior, realice el diagrama de clases de diseño

Modelo conceptual:



Contrato:

Nombre	nuevoMenú (número, fecha, listaIdReceta)
Responsabilidad	Incluir un nuevo menú, junto con las recetas que le corresponde a cada plato: primero, segundo y postre.
Tipo	SGMenú (Sistema)
Notas	listaIdReceta [1] == idReceta para el primero listaIdReceta [2] == idReceta para el segundo listaIdReceta [3] == idReceta para el postre
Referencias	
Excepciones	- No existe alguno de los objetos de la clase *Receta* identificado por los elementos de *listaIdReceta*.
Salida	- La *fecha* y el *número* del *Menú* Para todos los *Platos* - El *nombre* de la *Receta* correspondiente
Precondiciones	
Poscondiciones	▪ Se creó un objeto de la clase *Menú*, *miMenú*, debidamente inicializado. De 1 a 3 ▪ Se creó un objeto de la clase *Plato*, *miPlato*, debidamente inicializado. ▪ Se creó un enlace entre *miPlato* y el objeto *Receta* identificado por el correspondiente elemento de *listaIdReceta*. ▪ Se creó un enlace entre el objeto *miPlato* y el objeto *miMenú*.

Diagrama de comunicación:

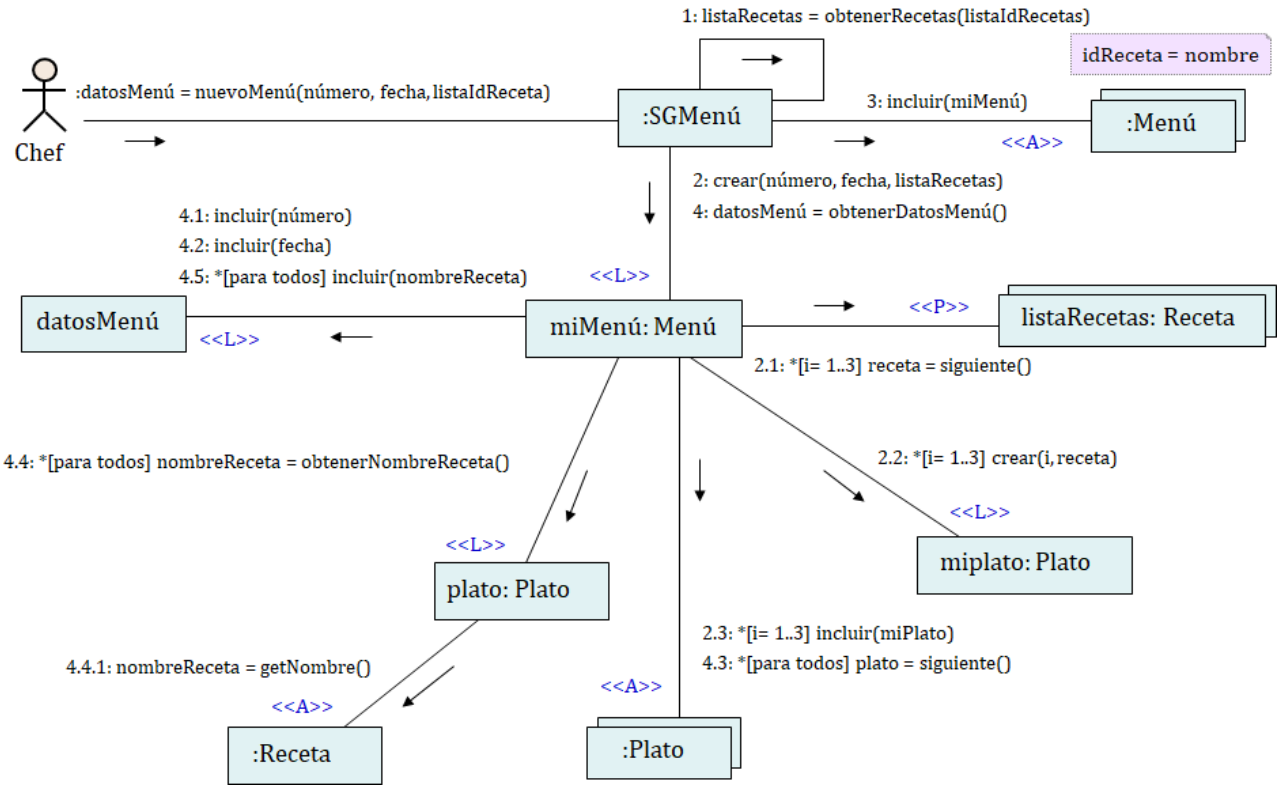


Diagrama de clases del diseño: