



Parcial de Diagramas UML

Por:

Samuel David Colmenarez Quero

Docente:

Christian David Jaimes Acevedo

Facultad de Ingenierías

Ingeniería de Software

Tecnológico de Antioquia - Institución Universitaria

Medellín, Colombia

2025

## Descripción de la actividad

La empresa gastronómica **CookMaster** necesita el diseño de un sistema para gestionar recetas culinarias. El sistema debe permitir crear, organizar y consultar recetas de diferentes tipos. El objetivo es modelar el sistema en **UML de clases** aplicando **uno o dos patrones de diseño** vistos en clase: Singleton, Factory o Builder.

## Diagrama de Clase

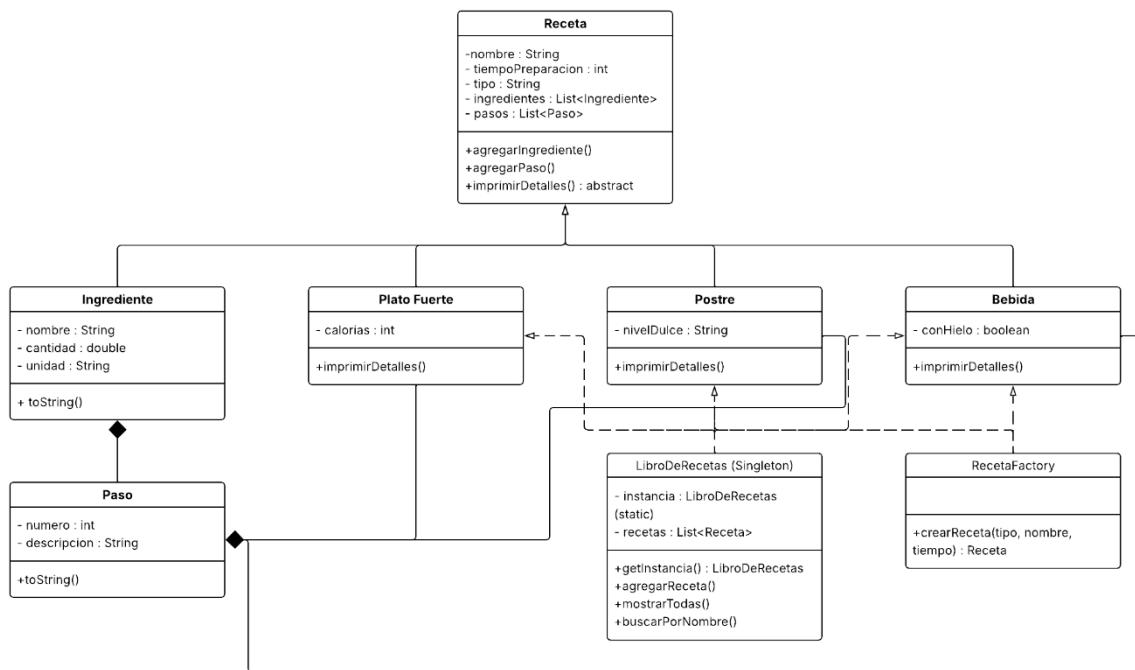


Ilustración 1. Diagrama de clases del ejercicio

## Código

### Receta.java

```

package Semana2.cookmaster.recetas;
import Semana2.cookmaster.ingredientes.Ingrediente;
import Semana2.cookmaster.pasos.Paso;
import java.util.ArrayList;
import java.util.List;

public abstract class Receta {
    protected String nombre;
    protected int tiempoPreparacion;
    protected String tipo;

    protected List<Ingrediente> ingredientes = new ArrayList<>();
    protected List<Paso> pasos = new ArrayList<>();

    public Receta(String nombre, int tiempo) {
        this.nombre = nombre;
        this.tiempoPreparacion = tiempo;
    }

    public void agregarIngrediente(Ingrediente ing) {
        ingredientes.add(ing);
    }

    public void agregarPaso(Paso paso) {
        pasos.add(paso);
    }

    public String getNombre() { return nombre; }

    public abstract void imprimirDetalles();
}

```

### Explicación

La clase abstracta Receta funciona como el molde principal para crear cualquier tipo de receta, ya sea un postre, una bebida o un plato fuerte. Dentro de esta clase se guardan los datos que todas las recetas tienen en común, como el nombre, el tiempo de preparación y el tipo de receta.

También incluye listas donde se guardan los ingredientes y los pasos necesarios.

Además, la clase ofrece métodos para agregar ingredientes, agregar pasos y obtener el nombre de la receta. Sin embargo, hay un método especial, llamado `imprimirDetalles()`, que no está completo aquí. Las clases hijas (como Postre, Bebida o PlatoFuerte) son las que deben completarlo, cada una a su manera.

Gracias a esta clase base, es más fácil organizar el código, evitar repeticiones y aprovechar la herencia, una idea importante en programación orientada a objetos.

## Ingrediente.java

```
package Semana2.cookmaster.ingredientes;

public class Ingrediente {
    private String nombre;
    private double cantidad;
    private String unidad;

    public Ingrediente(String nombre, double cantidad, String unidad) {
        this.nombre = nombre;
        this.cantidad = cantidad;
        this.unidad = unidad;
    }

    @Override
    public String toString() {
        return nombre + " " + cantidad + " " + unidad;
    }
}
```

## Explicación

La clase **Ingrediente** representa un componente esencial dentro de una receta culinaria. Cada ingrediente tiene tres características fundamentales: su nombre, la cantidad requerida y la unidad de medida (gramos, mililitros, unidades, etc.). Esta clase permite crear objetos simples y claros que se integran fácilmente dentro de la lista de ingredientes de una receta. Su constructor inicializa los valores y el método `toString()` devuelve una representación organizada y legible del ingrediente, facilitando su impresión cuando se consultan los detalles de una receta. En general, esta clase encapsula la información mínima necesaria para modelar un ingrediente de forma eficiente.

## Paso.java

```
package Semana2.cookmaster.pasos;

public class Paso {
    private int numero;
    private String descripcion;

    public Paso(int numero, String descripcion) {
        this.numero = numero;
        this.descripcion = descripcion;
    }

    @Override
    public String toString() {
        return numero + ". " + descripcion;
    }
}
```

## Explicación

La clase **Paso** describe cada instrucción necesaria para preparar una receta. Un paso contiene un número que indica el orden en el que debe ejecutarse y una descripción detallada de la acción a realizar. Su objetivo es organizar el proceso culinario de manera secuencial, permitiendo que el usuario siga las instrucciones fácilmente. Genera un formato adecuado para mostrar cada paso. Esta estructura facilita que las recetas sean claras, precisas y fáciles de comprender, permitiendo añadir tantos pasos como requiera la preparación.

## Bebida.java

```
package Semana2.cookmaster.recetas.tipos;
import Semana2.cookmaster.recetas.Receta;

public class Bebida extends Receta {
    private boolean conHielo;

    public Bebida(String nombre, int tiempo, boolean conHielo) {
        super(nombre, tiempo);
        this.tipo = "Bebida";
        this.conHielo = conHielo;
    }

    @Override
    public void imprimirDetalles() {
        System.out.println("==> Detalle de Receta ==>");
        System.out.println("Nombre: " + nombre);
        System.out.println("Tipo: " + tipo);
        System.out.println("Tiempo de preparación: " + tiempoPreparacion + " min");

        System.out.println("\nIngredientes:");
        int i = 1;
        for (var ing : ingredientes)
            System.out.println(" " + i++ + ") " + ing);

        System.out.println("\nPasos:");
        for (var p : pasos)
            System.out.println(" " + p);

        System.out.println("\nAtributos adicionales:");
        System.out.println("- Con hielo: " + (conHielo ? "Sí" : "No"));
    }
}
```

## Explicación

La clase Bebida es una clase que hereda de Receta, pero está pensada especialmente para recetas líquidas como jugos, batidos o refrescos. Además de las características básicas que ya trae de la clase Receta, esta clase agrega una propiedad extra llamada conHielo, que sirve para indicar si la bebida se sirve fría o no.

En su constructor, la clase recibe el nombre de la bebida, el tiempo de preparación y si lleva hielo. También tiene su propia versión del método imprimirDetalles(), donde se muestran los ingredientes, los pasos y esta característica especial.

Esta clase es un ejemplo de polimorfismo, porque cambia la forma de mostrar los detalles para adaptarse a lo que una bebida necesita, pero sin dejar de seguir la estructura general que define la clase Receta.

## PlatoFuerte.java

```
package Semana2.cookmaster.recetas.tipos;
import Semana2.cookmaster.recetas.Receta;

public class PlatoFuerte extends Receta {
    private int calorias;

    public PlatoFuerte(String nombre, int tiempo, int calorias) {
        super(nombre, tiempo);
        this.tipo = "Plato Fuerte";
        this.calorias = calorias;
    }

    @Override
    public void imprimirDetalles() {
        System.out.println("==== Detalle de Receta ===");
        System.out.println("Nombre: " + nombre);
        System.out.println("Tipo: " + tipo);
        System.out.println("Tiempo de preparación: " + tiempoPreparacion + " min");

        System.out.println("\nIngredientes:");
        int i = 1;
        for (var ing : ingredientes) {
            System.out.println(" " + i++ + ") " + ing);

        System.out.println("\nPasos:");
        for (var p : pasos)
            System.out.println(" " + p);

        System.out.println("\nAtributos adicionales:");
        System.out.println(" - Calorías: " + calorias);
    }
}
```

### Explicación

La clase `PlatoFuerte` representa comidas más completas o sustanciosas dentro del sistema, como almuerzos o cenas. Esta clase hereda de `Receta`, pero agrega un dato extra llamado `calorias`, que indica cuánta energía aporta el plato.

En su constructor, la clase recibe el nombre del plato, el tiempo de preparación y la cantidad de calorías. Además, tiene su propia versión del método `imprimirDetalles()`, donde muestra toda la información del plato: los ingredientes, los pasos y las calorías totales.

Esta clase sirve para distinguir platos más elaborados dentro del programa, manteniendo un orden claro al mostrar la información. Gracias a la herencia, solo se añade lo necesario sin repetir código.

## Postre.java

```
package Semana2.cookmaster.recetas.tipos;
import Semana2.cookmaster.recetas.Receta;

public class Postre extends Receta {
    private String nivelDulce;

    public Postre(String nombre, int tiempo, String nivelDulce) {
        super(nombre, tiempo);
        this.tipo = "Postre";
        this.nivelDulce = nivelDulce;
    }

    @Override
    public void imprimirDetalles() {
        System.out.println("==== Detalle de Receta ===");
        System.out.println("Nombre: " + nombre);
        System.out.println("Tipo: " + tipo);
        System.out.println("Tiempo de preparación: " + tiempoPreparacion + " min");

        System.out.println("\nIngredientes (" + ingredientes.size() + ")");
        int i = 1;
        for (var ing : ingredientes)
            System.out.println(" " + i++ + " " + ing);

        System.out.println("\nPasos (" + pasos.size() + ")");
        for (var p : pasos)
            System.out.println(" " + p);

        System.out.println("\nAtributos adicionales:");
        System.out.println(" - Nivel de dulce: " + nivelDulce);
    }
}
```

## Explicación

La clase Postre hereda de Receta y está pensada para representar preparaciones dulces como tartas, galletas o helados. Además de lo que ya trae la clase base, esta clase agrega un atributo llamado nivelDulce, que sirve para indicar qué tan dulce es el postre según lo que el usuario decida.

En su constructor, la clase recibe toda la información necesaria para crear el postre y también asigna el tipo "Postre" automáticamente. Luego, al sobrescribir el método imprimirDetalles(), muestra de manera ordenada los ingredientes, los pasos y el nivel de dulzor.

Gracias a esta clase, es posible crear postres personalizados con sus propias características, aprovechando toda la estructura general que ya ofrece el sistema de recetas sin tener que repetir código.

## RecetaFactory.java

```
package Semana2.cookmaster.recetas.factory;
import Semana2.cookmaster.recetas.Receta;
import Semana2.cookmaster.recetas.tipos.*;

public class RecetaFactory {
    public static Receta crearReceta(String tipo, String nombre, int tiempo) {
        switch (tipo.toUpperCase()) {
            case "POSTRE":
                return new Postre(nombre, tiempo, nivelDulce: "Medio");
            case "BEBIDA":
                return new Bebida(nombre, tiempo, conHielo: true);
            case "PLATO":
            case "PLATOFUERTE":
                return new PlatoFuerte(nombre, tiempo, calorias: 500);
            default:
                throw new IllegalArgumentException("Tipo de receta desconocido: " + tipo);
        }
    }
}
```

## Explicación

La clase **RecetaFactory** usa el *Patrón Factory*, cuya idea principal es facilitar la creación de objetos. En vez de que otras partes del programa tengan que crear directamente cada tipo de receta, esta clase recibe un texto (por ejemplo “Postre”, “Bebida” o “PlatoFuerte”) y devuelve una receta del tipo correcto.

De esta forma, el programa puede crear diferentes recetas sin preocuparse por cómo funciona cada clase por dentro. Toda la lógica para decidir qué objeto crear está guardada en un solo lugar.

Además, si algún día se agregan nuevos tipos de recetas, solo habría que modificar esta clase, lo que hace el sistema más fácil de mantener y extender.

## LibroDeRecetas.java

```
package Semana2.cookmaster.libro;
import Semana2.cookmaster.recetas.Receta;

import java.util.ArrayList;
import java.util.List;

public class LibroDeRecetas {
    private static LibroDeRecetas instancia;
    private List<Receta> recetas = new ArrayList<>();

    private LibroDeRecetas() {}

    public static LibroDeRecetas getInstancia() {
        if (instancia == null)
            | instancia = new LibroDeRecetas();
        return instancia;
    }

    public void agregarReceta(Receta r) {
        recetas.add(r);
        System.out.println(x: "Receta guardada en el Libro de Recetas.");
    }

    public void mostrarTodas() {
        System.out.println(x: "\nListado de recetas:");
        int i = 1;
        for (Receta r : recetas)
            System.out.println(i++ + ". " + r.getNombre());
    }
}
```

## Explicación

La clase **LibroDeRecetas** utiliza el patrón *Singleton*, cuya misión es asegurarse de que en todo el programa exista **solo un libro de recetas**. Para lograr esto, la clase tiene un **constructor privado** (que impide crear libros desde fuera) y un método estático llamado **getInstancia()**, que crea el libro una sola vez y después devuelve siempre la misma copia.

Gracias a esto, todas las recetas quedan guardadas en un único lugar, evitando duplicados o información desordenada. Además, esta clase se encarga de manejar todo lo relacionado con las recetas: buscarlas, registrarlas y mostrarlas.

El patrón Singleton es útil cuando se necesita un punto central de acceso, manteniendo el orden y la coherencia del sistema.

## Main.java

```
package Semana2.cookmaster;
import Semana2.cookmaster.pasos.Paso;
import Semana2.cookmaster.ingredientes.Ingrediente;
import Semana2.cookmaster.recetas.Receta;
import Semana2.cookmaster.recetas.factory.RecetaFactory;
import Semana2.cookmaster.libro.LibroDeRecetas;

public class Main {
    Run|Debug
    public static void main(String[] args) {
        LibroDeRecetas libro = LibroDeRecetas.getInstancia();

        // Crear postre
        Receta tarta = RecetaFactory.crearReceta(tipo: "POSTRE", nombre: "Tiramisú", tiempo: 40);
        tarta.agregarIngrediente(new Ingrediente(nombre: "Café", cantidad: 200, unidad: "ml"));
        tarta.agregarIngrediente(new Ingrediente(nombre: "Azúcar", cantidad: 80, unidad: "g"));

        tarta.agregarPaso(new Paso(numero: 1, descripcion: "Preparar café fuerte."));
        tarta.agregarPaso(new Paso(numero: 2, descripcion: "Mezclar mascarpone."));

        libro.agregarReceta(tarta);

        // Crear bebida
        Receta limonada = RecetaFactory.crearReceta(tipo: "BEBIDA", nombre: "Limonada Natural", tiempo: 10);
        limonada.agregarIngrediente(new Ingrediente(nombre: "Agua", cantidad: 500, unidad: "ml"));
        limonada.agregarIngrediente(new Ingrediente(nombre: "Limón", cantidad: 2, unidad: "u"));

        limonada.agregarPaso(new Paso(numero: 1, descripcion: "Exprimir limones."));
        limonada.agregarPaso(new Paso(numero: 2, descripcion: "Mezclar y servir."));

        libro.agregarReceta(limonada);
    }
}
```

## Explicación

La clase Main es donde realmente empieza el programa y sirve para mostrar cómo funciona todo el sistema de recetas. Primero, obtiene la única instancia del LibroDeRecetas, gracias al patrón *Singleton*, que asegura que solo existe un libro en todo el programa.

Luego, se crean diferentes recetas usando RecetaFactory, una clase especial que se encarga de fabricar objetos sin que tengamos que crearlos directamente. Después, a cada receta se le agregan sus ingredientes y sus pasos, y finalmente todas se guardan en el libro.

Al terminar, la clase imprime los detalles completos de una receta para demostrar que todo está funcionando como debe. En resumen, Main conecta todas las partes del sistema y muestra cómo trabajan juntas.