TEC-UCT
INSTITUTO TECNOLÓGICO
UNIVERSIDAD CATÓLICA DE TEMUCO

Informe Técnico - Marret Coffee

Nombre: Nicolás Huenchual , Felipe salazar , Michael Flores
Fecha: 24-10-2025
Profesor: Cristian Alejandro Iglesias Vera
Asignatura: Desarrollo y el diseño de software
Equipo: *Team Marret*

1. Introducción

El presente informe corresponde a la **Actividad Evaluada N°4**, orientada a la aplicación de **patrones de diseño de software** y la **planificación colaborativa de implementación** en el contexto del proyecto **Marret Café**. Este proyecto busca digitalizar la gestión de productos, reservas y pedidos en una cafetería, integrando tecnologías modernas que permiten una experiencia funcional, mantenible y escalable.

El propósito principal de esta entrega es demostrar la **aplicación correcta de los patrones Singleton y Factory** en el desarrollo del sistema, asegurando una arquitectura limpia, modular y controlada. Además, se incluye la planificación colaborativa de trabajo en equipo, el uso de control de versiones profesional y una reflexión técnica individual sobre el proceso de desarrollo.

2. Aplicación y Justificación de Patrones de Diseño en Marret Café

Patrones seleccionados

Patrón	Aplicación en el sistema	Beneficio
Singleton	Control de una única conexión a la base de datos MySQL.	Evita duplicidad de conexiones y mejora el rendimiento.
Factory	Creación dinámica de productos del menú (cafés, pasteles).	Centraliza la lógica de instanciación y mejora la extensibilidad.

Patrón Singleton – Conexión a la base de datos

Qué problema resuelve

Durante el desarrollo del sistema **Marret Café**, se detectó la necesidad de mantener una **única conexión controlada a la base de datos MySQL**.
En aplicaciones PHP tradicionales, cada script puede abrir múltiples conexiones, generando **sobrecarga del servidor** y **riesgo de inconsistencias**.

El patrón **Singleton** soluciona este problema al garantizar que **solo exista una instancia** de la conexión durante toda la ejecución del sistema, permitiendo que los módulos de productos, reservas y administración compartan el mismo acceso de forma eficiente.

Por qué fue elegido

Se eligió este patrón porque:

- Permite **compartir la conexión** entre distintos módulos sin crear nuevas instancias.

- **Centraliza la configuración** (credenciales y parámetros), evitando duplicación de código.

- Su implementación en PHP es **simple, reutilizable y fácilmente integrable** en cualquier script del proyecto.

Impacto en la mantenibilidad y escalabilidad

- **Mantenibilidad:** al concentrar la lógica de conexión en una sola clase (`Database`), futuras modificaciones son más simples.
- **Escalabilidad:** nuevos módulos pueden reutilizar la misma instancia sin ajustes adicionales.
- **Estabilidad:** garantiza una comunicación consistente entre los componentes y la base de datos.

3.3 Patrón Factory – Creación de productos

Qué problema resuelve

En el sistema **Marret Café**, el módulo de gestión de productos debía manejar distintos tipos de artículos —como **cafés** y **pasteles** —, cada uno con atributos y comportamientos específicos.
Crear estos objetos directamente en el código con new generaba **duplicación de lógica** y **alto acoplamiento**, lo que dificultaba mantener y ampliar el sistema.

El patrón **Factory** resuelve este problema al **centralizar la creación de objetos** en una sola clase responsable de instanciar el tipo correcto, garantizando coherencia y reduciendo errores.

Por qué fue elegido

Fue elegido porque:

- Encapsula la lógica de creación en una clase única (ProductoFactory).
- Permite **agregar nuevos tipos de productos** sin alterar el código existente.
- Cumple con el principio de **abierto/cerrado (OCP)**, mejorando la extensibilidad.
- Favorece una arquitectura más organizada y modular.

Impacto en la mantenibilidad y escalabilidad

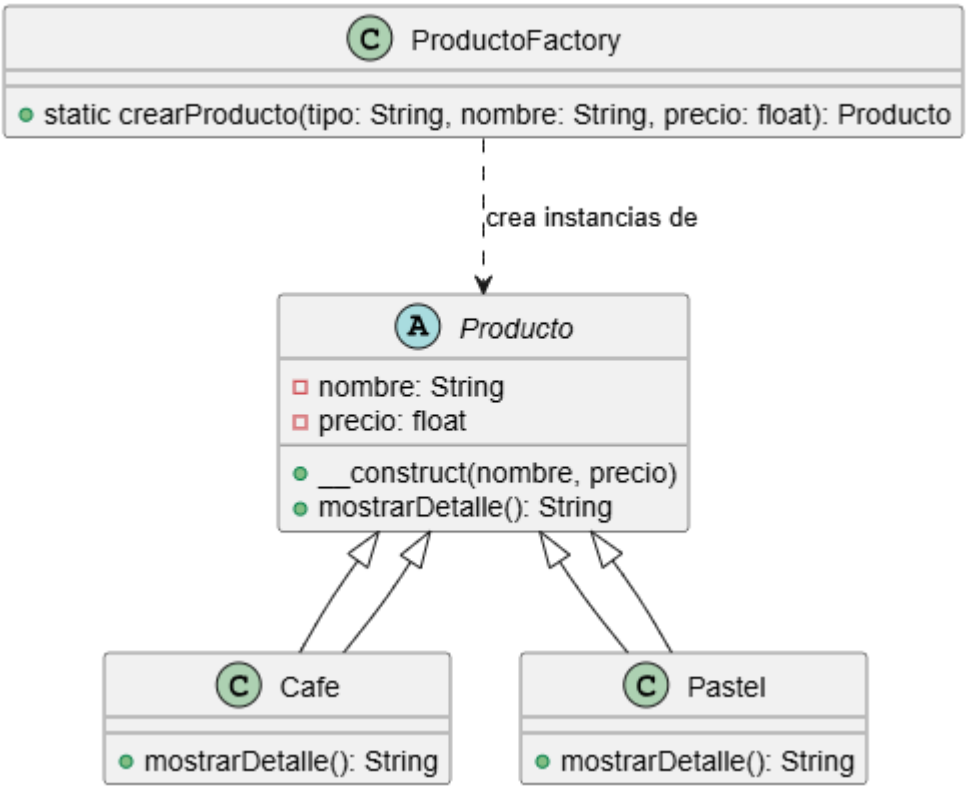
- **Mantenibilidad:** simplifica la creación de productos al concentrar la lógica en una única clase.
- **Escalabilidad:** facilita la incorporación de nuevos tipos de productos (por ejemplo, bebidas frías o snacks).
- **Consistencia:** asegura que todos los módulos utilicen la misma forma de crear y gestionar productos.

En conclusión, el patrón **Factory** proporciona una estructura más flexible y extensible, fortaleciendo la arquitectura del sistema **Marret Café** y facilitando su crecimiento futuro.

3. Diseño detallado y diagramas UML

Se presentan los diagramas UML de los patrones aplicados (Figuras 1 y 2), junto con sus implementaciones en PHP. Estos diagramas reflejan la estructura de clases y las responsabilidades asignadas, manteniendo la cohesión del diseño y bajo acoplamiento.

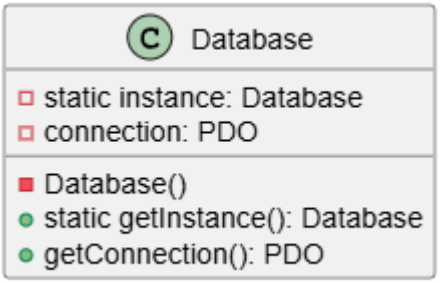
Diagrama UML de Factory :



Código PHP:

```
abstract class Producto {
    protected $nombre;
    protected $precio;

    public function __construct($nombre, $precio) {
        $this->nombre = $nombre;
        $this->precio = $precio;
    }

    public abstract function mostrarDetalle();
}

class Cafe extends Producto {
    public function mostrarDetalle() {
        return "☕ Café: {$this->nombre} - $ {$this->precio}";
    }
}

class Pastel extends Producto {
    public function mostrarDetalle() {
        return "🍰 Pastel: {$this->nombre} - $ {$this->precio}";
    }
}

class ProductoFactory {
    public static function crearProducto($tipo, $nombre, $precio) {
        switch ($tipo) {
            case "cafe": return new Cafe($nombre, $precio);
            case "pastel": return new Pastel($nombre, $precio);
            default: throw new Exception("Tipo de producto no válido.");
        }
    }
}
```

Diagrama UML de Patrón Singleton:



Implementación en PHP:

```
class Database {
    private static $instance = null;
    private $connection;

    private function __construct() {
        $this->connection = new PDO(
            "mysql:host=db;dbname=marret_cafe;charset=utf8",
            "marret",
            "marret123"
        );

    public static function getInstance() {

  if (self::$instance === null) {

  self::$instance = new Database();
        }
        return self::$instance;
    }
    public function getConnection() {
        return $this->connection;
    }
```

TEC-UCT
INSTITUTO TECNOLÓGICO
UNIVERSIDAD CATÓLICA DE TEMUCO

}
}

4. Plan de implementación colaborativa Herramientas utilizadas

Repositorio: GitHub.

Entorno: Visual Studio Code con Docker y Git integrados.

Comunicación: WhatsApp y reuniones presenciales.

Control de tareas: GitHub y coordinación directa entre los integrantes.

Organización del trabajo

El desarrollo se concentró principalmente en la rama **master**, desde la cual se realizaron los avances y correcciones del proyecto.
Aunque no se implementó una estructura avanzada de ramas (como `develop` o `feature`), el equipo mantuvo un control manual de versiones mediante **commits descriptivos** y respaldos frecuentes del código.
Cada integrante colaboró en secciones específicas (documentación, diagramas, diseño o desarrollo), manteniendo una comunicación constante para coordinar los cambios.

Roles del equipo Team Marret

Felipe Salazar Arias	Presentador Documentación	Exposición del proyecto.
Nicolás Huenchual	Líder / Coordinador	Organización del equipo y control de tiempo.
Michael Flores	Diseñador UML / QA Diagramas UML	Control de calidad y validación técnica.

Evidencias del trabajo colaborativo

Durante el desarrollo, el equipo concentró su trabajo en la rama principal (**master**) del repositorio GitHub, donde se realizaron múltiples commits que reflejan la evolución del proyecto.
Cada commit incluye mensajes descriptivos que registran mejoras, rediseños de diagramas UML y actualizaciones de documentación.

Las evidencias del historial de commits, junto con las contribuciones realizadas por los integrantes del equipo, se incluyen en los **Anexos**

5.Conclusión Técnica y de Trabajo en Equipo

El proyecto **Marret Café** fue una gran oportunidad para comprender cómo los **patrones de diseño** pueden hacer que un sistema sea más ordenado, eficiente y fácil de mantener.
El **patrón Singleton** tuvo un papel clave al manejar la conexión con la base de datos, evitando la creación de múltiples instancias innecesarias y mejorando así el rendimiento y la estabilidad del sistema.
Asimismo, el **patrón Factory** aportó una gran flexibilidad, ya que permite crear nuevos productos sin modificar el código base, haciendo que el sistema sea más **escalable** y **adaptable** a futuros cambios.

En cuanto al **trabajo en equipo**, el uso de **Docker** y **GitHub** fue fundamental. Estas herramientas permitieron mantener un entorno unificado, facilitar la colaboración y garantizar que todos los integrantes trabajáramos en la misma dirección. Además, fortalecieron la comunicación, la organización y la coordinación del grupo, aspectos esenciales en cualquier proyecto de desarrollo profesional.

6.Reflexión Personal

Este proyecto permitió reforzar habilidades tanto **técnicas** como **colaborativas**. Aprendimos la importancia de diseñar pensando en el futuro: que el código no solo funcione hoy, sino que sea **sostenible, escalable y mantenible**.

En resumen, fue una experiencia que unió **práctica, aprendizaje y trabajo en equipo**, demostrando que la ingeniería de software va más allá de programar: se trata de **crear soluciones sólidas, bien pensadas y sostenibles**.

Anexos

1. Uso de herramientas de Inteligencia Artificial (IA)

Durante el desarrollo del informe y diseño de patrones, el equipo utilizó la herramienta Chat GPT (modelo GPT-5) como apoyo en la estructuración del documento, generación de ejemplos de código y redacción técnica.

Significa que el diseño por ejemplo:

- ☐ El uso de IA se centró en las siguientes tareas.
- ☐ Generación de ejemplos base en PHP para los patrones Singleton y Factory.
- ☐ Redacción técnica y coherente con la rúbrica de evaluación.
- ☐ Estandarización del lenguaje profesional y formal.

Sin embargo, el contenido final fue analizado, adaptado y validado manualmente por los integrantes del Team Marret, asegurando que la aplicación técnica y la documentación reflejan el trabajo real del grupo.

Este proceso permitió aprender a integrar herramientas de IA de forma ética y responsable, utilizando su potencial como asistente, pero manteniendo la autoría técnica y conceptual del equipo.

ejemplo de promt:

Prompt 1: “Explícame cómo aplicar el patrón Singleton en PHP con conexión a base de datos.”
Prompt 2: “Redáctame la sección de análisis y selección de patrones para el informe AE4.”

“El uso de inteligencia artificial fue una herramienta complementaria de apoyo, sin sustituir el razonamiento ni el desarrollo propio del equipo.”

TEC-UCT
INSTITUTO TECNOLÓGICO
UNIVERSIDAD CATÓLICA DE TEMUCO

2.Codigo del Diagrama UML de Patrón Singleton

```
@startuml

class Database {

  - static instance: Database

  - connection: PDO

  - Database()

  + static getInstance(): Database

  + getConnection(): PDO

}

@enduml
```

Nota:

El Singleton es una única caja registradora que todos los empleados usan para cobrar, No crea una nueva cada vez que entra un cliente, solo usa la misma para mantener el control del dinero.

TEC-UCT
INSTITUTO TECNOLÓGICO
UNIVERSIDAD CATÓLICA DE TEMUCO

3.Código del diagrama UML de Patrón Factory

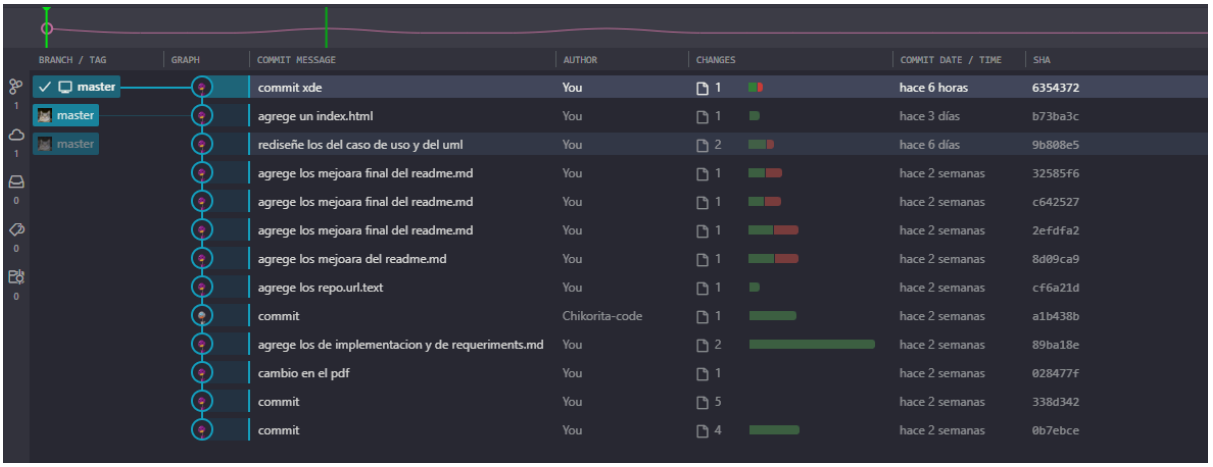
```
@startuml

' ====== Clases principales ======
abstract class Producto {
  - nombre: String
  - precio: float
  + __construct(nombre, precio)
  + mostrarDetalle(): String
}

class Cafe extends Producto {
  + mostrarDetalle(): String
}

class Pastel extends Producto {
  + mostrarDetalle(): String
}

class ProductoFactory {
  + static crearProducto(tipo: String, nombre: String, precio: float):
Producto
}

' ====== Relaciones ======
Producto <|-- Cafe
Producto <|-- Pastel
ProductoFactory ..> Producto : crea instancias de

@enduml
```

Nota:

El diagrama representa la implementación del *patrón de diseño Factory Method* en el sistema “Marret Café”. Este modelo permite crear distintos tipos de productos (como cafés y pasteles) de forma flexible y organizada. La clase ProductoFactory centraliza la creación de objetos, facilitando la extensibilidad del sistema y reduciendo la duplicación de código.

4.Historial de commits en GitHub

Descripción: Captura del historial de commits del repositorio del proyecto *Marret Café* en la rama principal (*master*), donde se evidencian los aportes de los integrantes, las actualizaciones del código y la documentación.



BRANCH / TAG	GRAPH	COMMIT MESSAGE	AUTHOR	CHANGES	COMMIT DATE / TIME	SHA
master		commit xde	You	1	hace 6 horas	6354372
master		agrege un index.html	You	1	hace 3 días	b73ba3c
master		rediseñe los del caso de uso y del uml	You	2	hace 6 días	9b808e5
		agrege los mejoara final del readme.md	You	1	hace 2 semanas	32585f6
		agrege los mejoara final del readme.md	You	1	hace 2 semanas	c642527
		agrege los mejoara final del readme.md	You	1	hace 2 semanas	2efdfa2
		agrege los mejoara del readme.md	You	1	hace 2 semanas	8d09ca9
		agrege los repo.url.text	You	1	hace 2 semanas	cf6a21d
		commit	Chikorita-code	1	hace 2 semanas	a1b438b
		agrege los de implementacion y de requeriments.md	You	2	hace 2 semanas	89ba18e
		cambio en el pdf	You	1	hace 2 semanas	028477f
		commit	You	5	hace 2 semanas	338d342
		commit	You	4	hace 2 semanas	0b7ebce