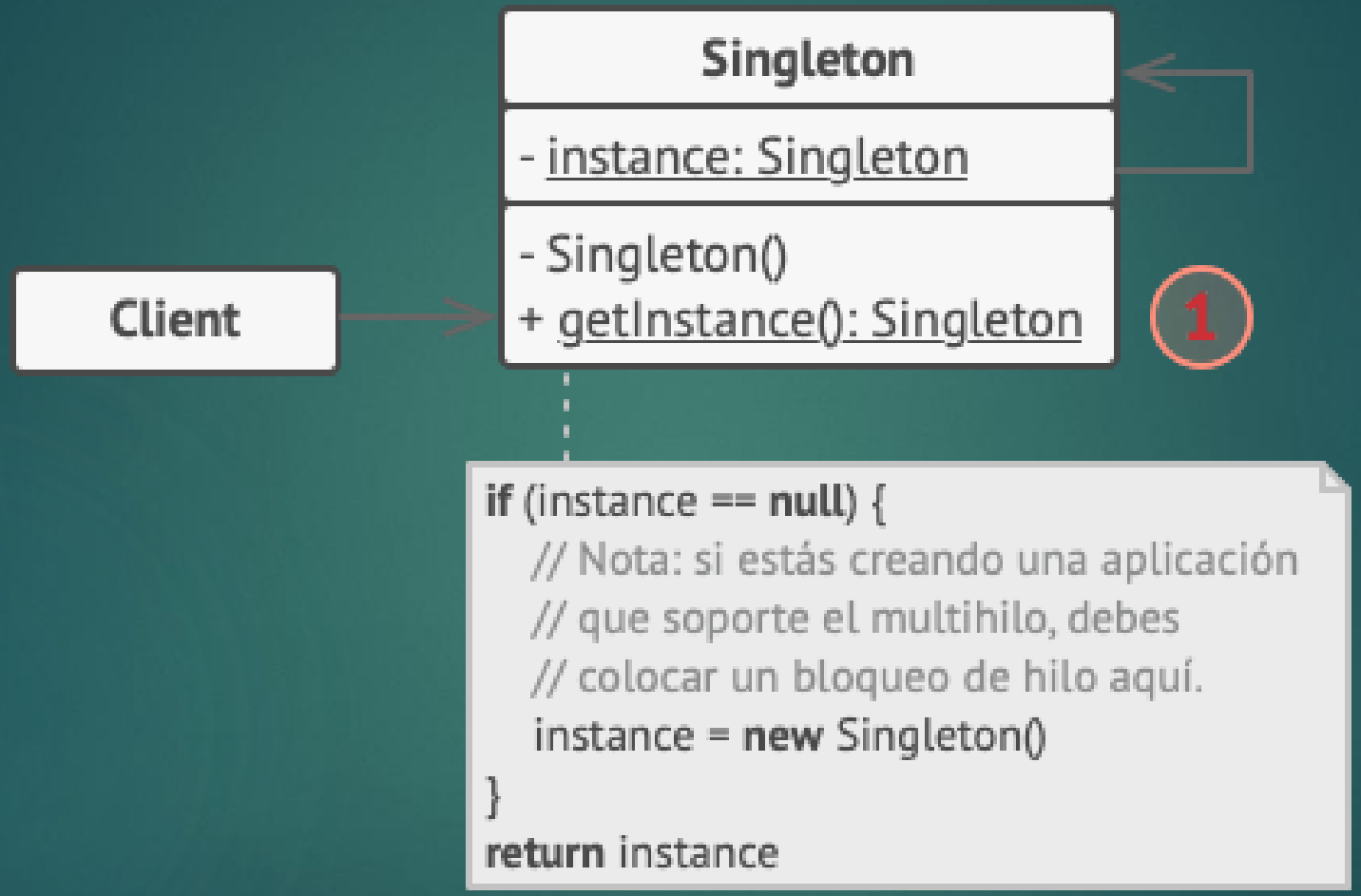
Grupo 12

MATERIA: DESARROLLO DE SOFTWARE 9

Integrantes:
Gustavo Cerrud,
Carlos Gonzalez,
Juan Samudio

🧠 Patrón Singleton

Patrón Singleton


Singleton
- instance: Singleton
- Singleton()
+ getInstance(): Singleton
Client
1
if (instance == null) {
// Nota: si estás creando una aplicación
// que soporte el multihilo, debes
// colocar un bloqueo de hilo aquí.
instance = new Singleton()
}
return instance

🧩 Características del patrón Singleton:

Una sola instancia

Acceso global

Control de creación

Constructor privado

Ideal para recursos compartidos

Gestor de Conexiones a Bases de Datos

DatabaseConnection.getInstance()

Logger (Sistema de logs)

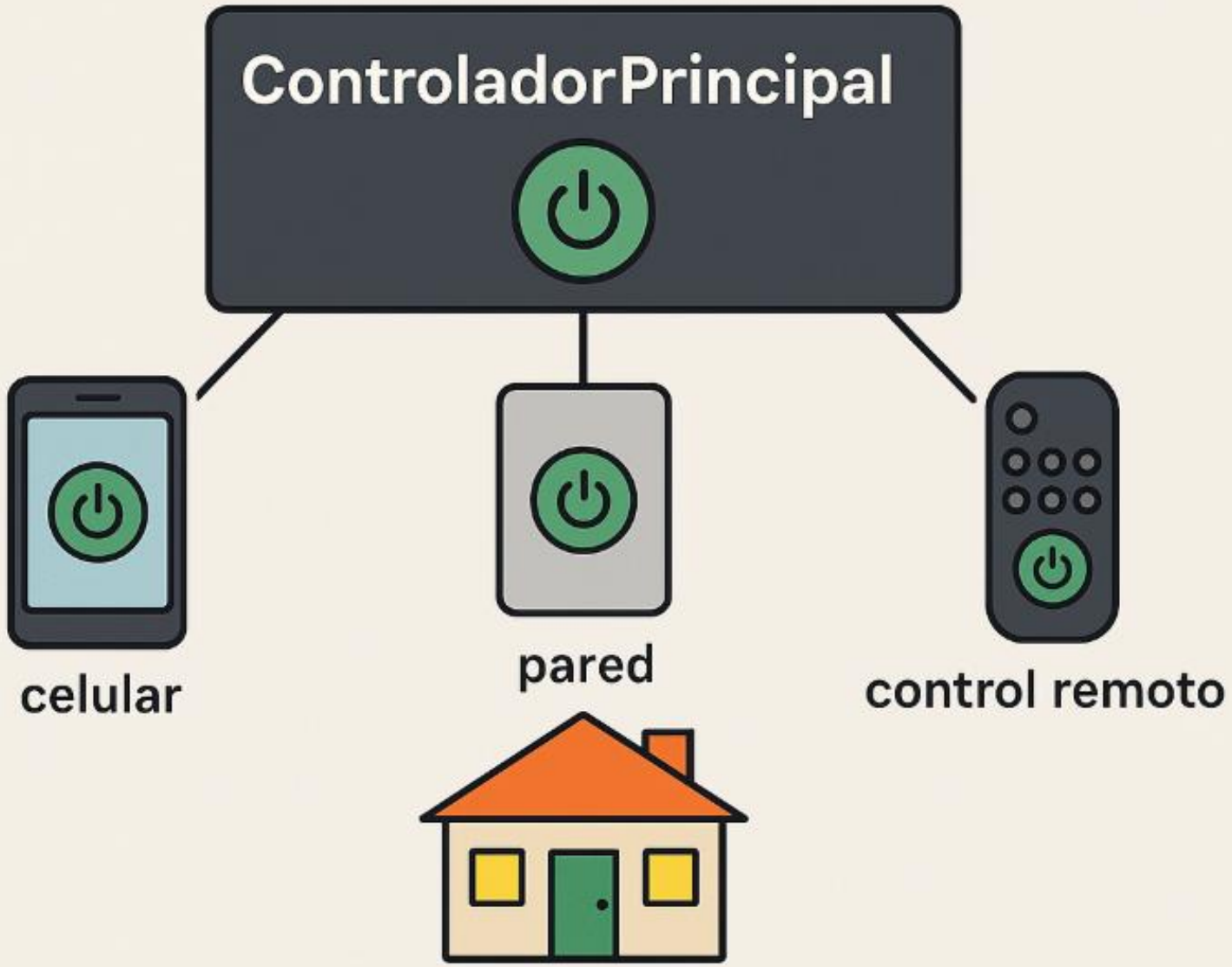
Logger.getInstance().log("Algo pasó")

Administrador de Colas (Thread Pool / Task Manager)

TaskManager.getInstance().addTask(task)


Caja1
Caja2
Caja
Registradora

Singleton



❌ ¿Qué pasa si no usas el patrón Singleton cuando deberías?

Múltiples instancias innecesarias

Inconsistencias en los datos

Confusión en el acceso a recursos

Dificultad para mantener el código

Problemas de sincronización (en aplicaciones con múltiples hilos)

🧠 Patrón Iterator

📌 **¿Para qué sirve?**

Sirve para acceder secuencialmente a los elementos de una colección sin necesidad de conocer cómo está implementada.

PROPÓSITO

permite recorrer colecciones de objetos sin exponer su estructura interna.

Estructura

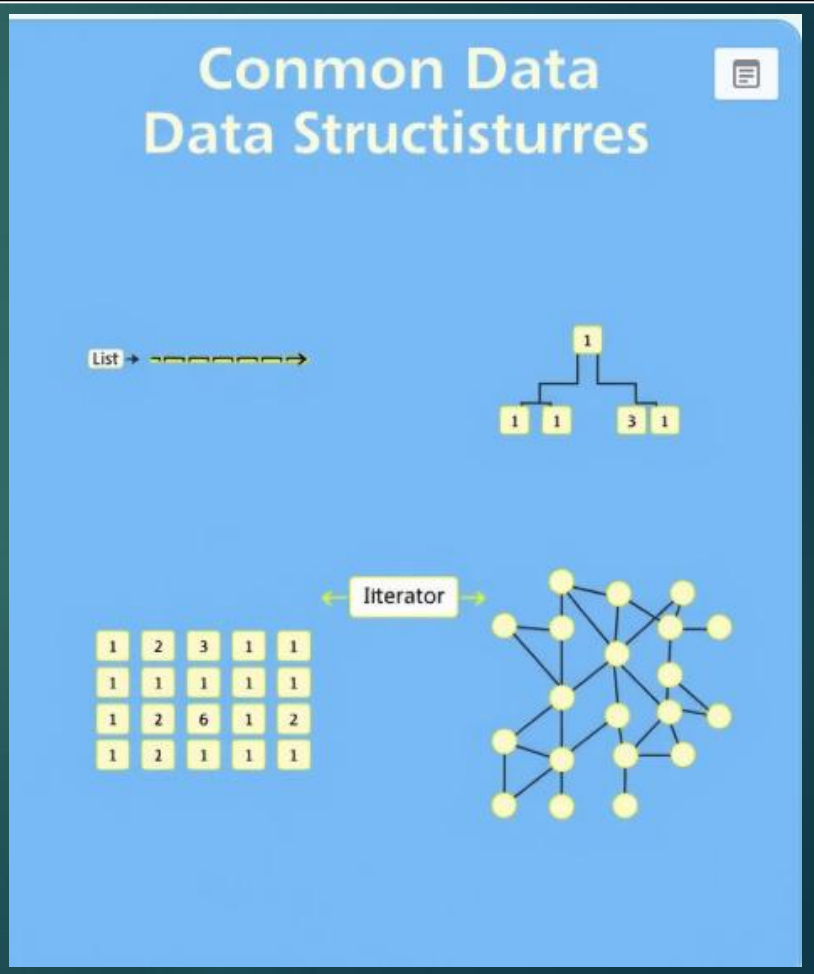
Consiste en dos componentes principales: el iterador y la colección.

Iterar sobre Diferentes Tipos de Colecciones

1.Listas 🟰 estructuras lineales (como arreglos o listas enlazadas).

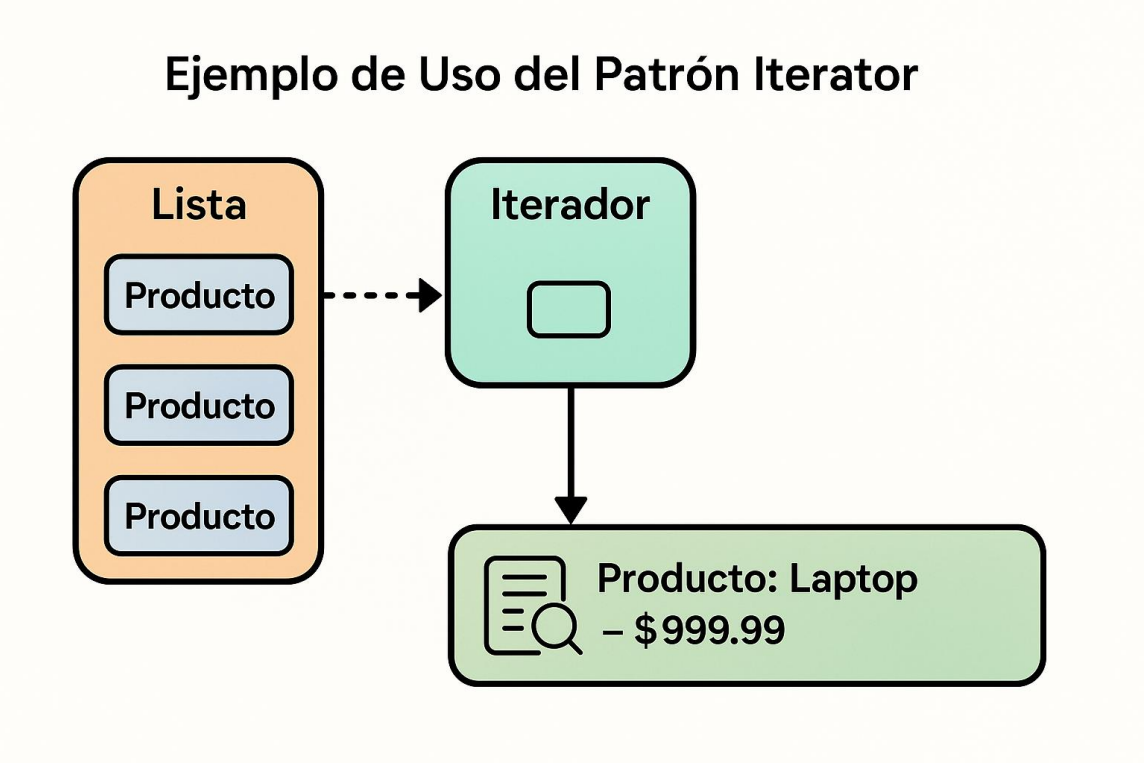
2.Árboles 🌲 estructuras jerárquicas (por ejemplo, árboles binarios).

3.Gráficas 🗺️ estructuras de nodos conectados entre sí (como mapas de rutas o redes sociales).



Ventajas del Patrón Iterator

- Separa la lógica de recorrido de la estructura de datos.
- Permite múltiples recorridos simultáneos (por diferentes iteradores).
- Hace que tu código sea más limpio y flexible.



✅ Comparación clara

Punto	`for` tradicional	Patrón Iterator
Simplicidad inicial	✅ Muy simple	🔸 Necesita más código
Reutilización	🔴 Baja (repetís el `for`)	✅ Alta (puedes reutilizar el iterador)
Independencia	🔴 Depende de que sea un array	✅ Funciona con cualquier estructura
Flexibilidad	🔴 Difícil cambiar el recorrido	✅ Puedes cambiar el tipo de recorrido
Código limpio	✅ en estructuras simples	✅ en estructuras complejas
Escalabilidad	🔴 Mala para árboles o grafos	✅ Perfecto para estructuras complejas

Conclusión

- El patrón Singleton es apropiado cuando se requiere garantizar que una clase tenga una única instancia accesible globalmente. Su correcta aplicación ayuda a mantener el orden en el sistema al evitar la duplicación innecesaria de objetos críticos. Sin embargo, un mal uso puede derivar en problemas graves como pérdida de control sobre recursos compartidos, datos inconsistentes y errores difíciles de depurar.

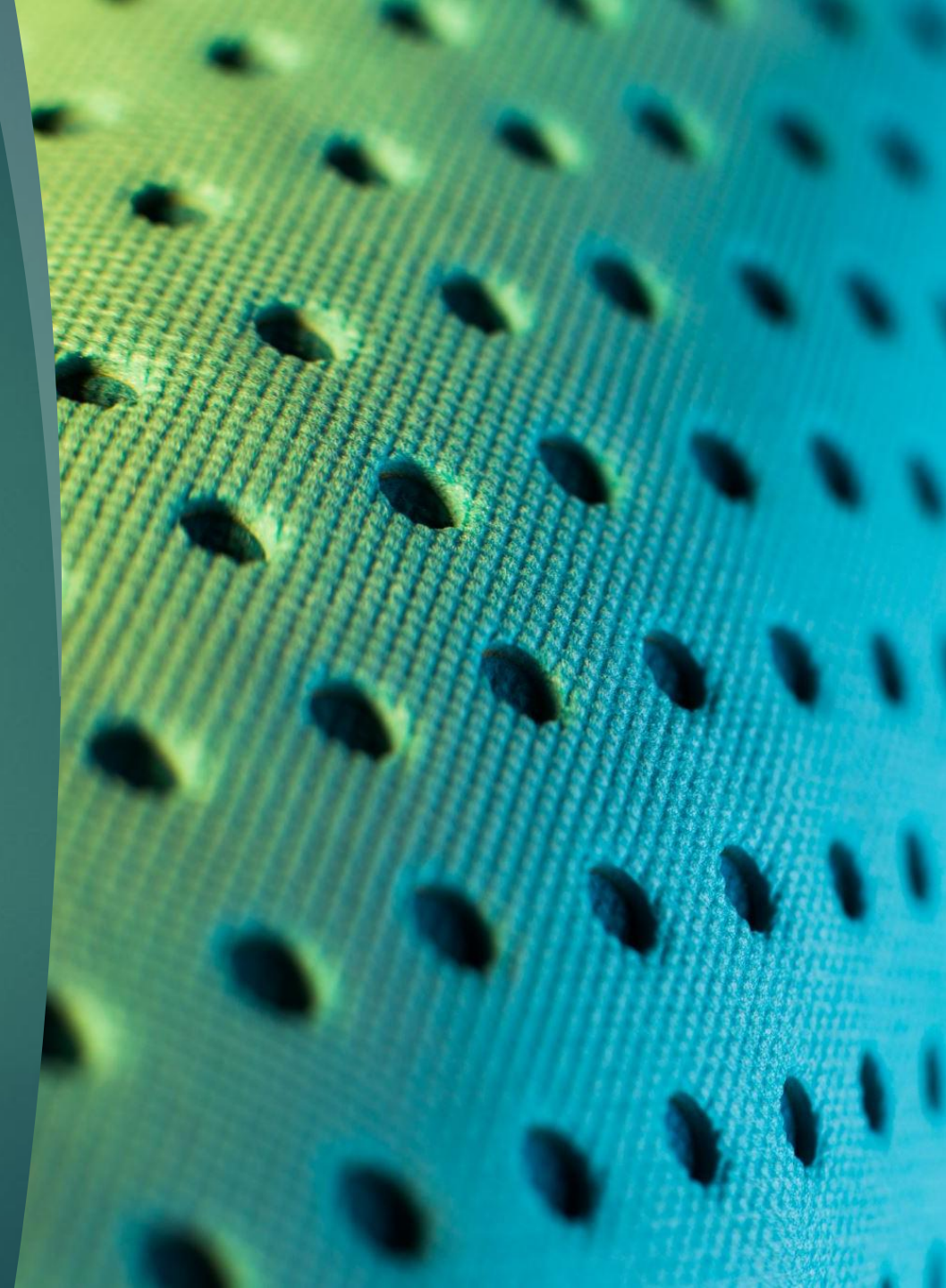
- Carlos Gonzalez

- Por otro lado, tenemos lo que es el patrón Iterator que nos proporciona un mecanismo mas estandarizado para recorrer los elementos dentro de una colección sin exponer su estructura interna. Esto favorece la encapsulación y facilita la implementación de algoritmos sobre estructuras de datos complejas de manera uniforme y segura.Con esto mencionado al darle un uso adecuado a los patrones de diseño en la programación orientada a objetos permite mejorar la calidad, mantenibilidad y escalabilidad del software.

- Gustavo Cerrud

- El patrón iterador me ha sorprendido por lo funcional que es, especialmente al aplicarlo a colecciones como si fueran listas simples, aunque por dentro tengan una lógica más compleja. Me llamó la atención cómo puedo recorrer una colección personalizada (como una lista de libros, objetos o incluso registros de base de datos) con foreach como si fuera un array, sin tener que saber cómo están guardados o manejados internamente los datos.Esto es especialmente potente en el contexto de bases de datos u otras estructuras complejas. En lugar de exponer directamente la estructura interna como un array, una consulta SQL o una API, el iterador me permite ocultar toda esa lógica y simplemente entregar los elementos de uno en uno, de forma limpia y controlada.Lo mejor de todo es que este patrón me separa completamente el "cómo recorro" de "qué estoy recorriendo", lo cual hace que el código sea mucho más legible, mantenible y reutilizable. No necesito saber si los datos vienen de una consulta, un archivo, o están generados dinámicamente: el iterador se encarga del trabajo sucio, y yo solo me enfoco en qué hacer con cada elemento.

- Juan Samudio



Bibliografía

Singleton. (n.d.). Refactoring.guru. Retrieved April 19, 2025, from https://refactoring.guru/es/design-patterns/singleton

Wikipedia contributors. (n.d.). *Singleton*. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Singleton&oldid=166493099

Iterator. (n.d.). Refactoring.guru. Retrieved April 19, 2025, from https://refactoring.guru/es/design-patterns/iterator

Iterator. (n.d.-a). Refactoring.guru. Retrieved April 19, 2025, from https://refactoring.guru/es/design-patterns/iterator