Taller Singleton vs Factory/Builder

Nombre del estudiante:	Matias Lugmaña, Camilo Orrico, Denise Rea, Julio Viche
Docente:	Mgt. Jenny Alexandra Ruiz Robalino
Fecha:	27/11/2025
NRC:	27835

1. Objetivo del Taller

El objetivo de este taller es analizar, comparar y evaluar el Patrón Singleton frente a los patrones Builder y Factory mediante métricas de rendimiento, memoria, concurrencia y diseño, aplicados dentro de una aplicación CRUD de Estudiantes. Se busca demostrar cómo Singleton garantiza una única instancia con estado compartido, mientras que Builder y Factory generan objetos independientes, permitiendo mayor flexibilidad, estabilidad y menor acoplamiento.

```
╔═══════════════════════════════════════════════════════════╗
║     PRUEBAS COMPARATIVAS: SINGLETON VS BUILDER PATTERN    ║
╚═══════════════════════════════════════════════════════════╝


MÉTRICA 1: Unicidad de Instancia
──────────────────────────────────────────────────────────────
Service1 hashCode: 1998767043
Service2 hashCode: 1998767043
Service3 hashCode: 1998767043
RESULTADO: Las 3 instancias son IDÉNTICAS (Singleton correcto)


MÉTRICA 2: Estado Compartido entre Controladores
──────────────────────────────────────────────────────────────
Controlador 1 agrega: Juan Pérez
Controlador 2 agrega: María López
Controlador 3 agrega: Carlos Ruiz

Controlador 1 ve: 3 estudiantes
Controlador 2 ve: 3 estudiantes
Controlador 3 ve: 3 estudiantes
RESULTADO: TODOS comparten la misma lista (Singleton correcto)


 MÉTRICA 3: Repository Singleton
──────────────────────────────────────────────────────────────
Repository1 hashCode: 842741472
Repository2 hashCode: 842741472
Repository3 hashCode: 842741472
```

RESULTADO: Una sola instancia de Repository

MÉTRICA 4: Builder crea instancias independientes

Estudiante1 hashCode: 1610525991
Estudiante2 hashCode: 1666607455
Estudiante3 hashCode: 1327006586
RESULTADO: Cada llamada crea una instancia NUEVA

MÉTRICA 5: Comparativa de Performance

SINGLETON (1000 accesos): 0 ms
BUILDER (1000 creaciones): 3 ms
Factor de diferencia: 3x
CONCLUSIÓN: Singleton es MÁS RÁPIDO para acceso

MÉTRICA 6: Consumo de Memoria

SINGLETON (1 instancia): ~0 KB
BUILDER (1000 instancias): ~2048 KB
CONCLUSIÓN: Singleton usa MENOS memoria

MÉTRICA 7: Rendimiento Bajo Carga (10,000 operaciones)

SINGLETON (10K accesos): 1 ms
BUILDER (10K creaciones): 10 ms
Diferencia: 10,00x más lento Builder
Throughput Singleton: 10.000.000 ops/seg
Throughput Builder: 1.000.000 ops/seg
CONCLUSIÓN: Singleton escala mejor bajo carga intensiva

MÉTRICA 8: Facilidad de Testing

Builder: Testeable SIN mocks ni dependencias
Singleton: Estado compartido puede afectar otros tests
CONCLUSIÓN: Builder tiene MAYOR testabilidad

MÉTRICA 9: Impacto de Garbage Collection

Builder (5000 objetos): 0 ciclos de GC
Singleton (1 objeto): 0 ciclos de GC adicionales
CONCLUSIÓN: Builder genera más presión en GC

MÉTRICA 10: Análisis de Latencia (percentiles)

SINGLETON:

- P50 (mediana): 0 µs
- P95: 0 µs
- P99: 0 µs

BUILDER:
- P50 (mediana): 0 µs
- P95: 0 µs
- P99: 0 µs

CONCLUSIÓN: Singleton tiene latencias más predecibles y bajas

📊 MÉTRICA 11: Prueba de Concurrencia (10 threads)

SINGLETON (10 threads, 100 ops c/u): 4 ms
BUILDER (10 threads, 100 ops c/u): 7 ms
Singleton usa synchronized (serialización)
Builder es naturalmente thread-safe
CONCLUSIÓN: Builder mejor para escenarios concurrentes

MÉTRICA 12: Acoplamiento y Cohesión

BUILDER:
- Acoplamiento: BAJO (0 dependencias)
- Cohesión: ALTA (solo crea y valida)
- Responsabilidad: Una sola (SRP ✅)

SINGLETON:
- Acoplamiento: MEDIO (depende de Repository)
- Cohesión: MEDIA (coordina + valida)
- Responsabilidad: Múltiple (gestión de estado)

CONCLUSIÓN: Builder cumple mejor SRP

RESUMEN COMPARATIVO FINAL

MÉTRICA	SINGLETON	BUILDER
Instancias	1 (única) ✅	N (múltiples)
Estado compartido	SÍ ✅	NO
Performance (1000x)	< 10ms ✅	~100-500ms
Performance (10Kx)	< 50ms ✅	~1-5s
Consumo memoria	Mínimo ✅	Proporcional
Latencia P99	< 1µs ✅	~10-50µs
Garbage Collection	Mínimo ✅	Alto
Concurrencia (10 threads)	Serializado	Paralelo ✅
Thread-Safety	Requiere sync	Natural ✅
Testabilidad	Media	Alta ✅
Acoplamiento	Medio	Bajo ✅
Cohesión	Media	Alta ✅

Reusabilidad	Limitada	Alta ✅
Complejidad	Baja ✅	Media

MÉTRICAS DE RENDIMIENTO MEDIDAS:

1. Throughput (ops/segundo)	- Singleton gana
2. Latencia (P50, P95, P99)	- Singleton gana
3. Consumo de memoria	- Singleton gana
4. Presión en GC	- Singleton gana
5. Escalabilidad (10K ops)	- Singleton gana
6. Concurrencia multi-thread	- Builder gana
7. Testabilidad	- Builder gana
8. Acoplamiento/Cohesión	- Builder gana

RECOMENDACIÓN BASADA EN RENDIMIENTO:

• USAR SINGLETON cuando:
  - Necesitas máximo rendimiento (bajo carga)
  - Gestión centralizada de estado
  - Memoria es crítica
  - Caché o pool de recursos

• USAR BUILDER cuando:
  - Objetos independientes y desacoplados
  - Alta concurrencia multi-thread
  - Validación estricta en creación
  - Testabilidad es prioritaria

• MEJOR PRÁCTICA: Usar AMBOS complementariamente
  → Singleton para servicios/coordinación
  → Builder para creación/validación de objetos

PRUEBAS COMPARATIVAS: SINGLETON VS FACTORY PATTERN

Modo test activado (validación deshabilitada SOLO para tests)

MÉTRICA 1: Unicidad de Instancia

Service1 hashCode: 966739377
Service2 hashCode: 966739377
Service3 hashCode: 966739377
RESULTADO: Las 3 instancias son IDÉNTICAS (Singleton correcto)

MÉTRICA 2: Estado Compartido entre Controladores

Controlador 1 agrega: Juan Pérez
Controlador 2 agrega: María López
Controlador 3 agrega: Carlos Ruiz

Controlador 1 ve: 3 estudiantes
Controlador 2 ve: 3 estudiantes
Controlador 3 ve: 3 estudiantes
RESULTADO: TODOS comparten la misma lista (Singleton correcto)

MÉTRICA 3: Repository Singleton

Repository1 hashCode: 825936265
Repository2 hashCode: 825936265
Repository3 hashCode: 825936265
RESULTADO: Una sola instancia de Repository

MÉTRICA 4: Factory crea instancias independientes

Estudiante1 hashCode: 825249556
Estudiante2 hashCode: 883151184
Estudiante3 hashCode: 709865851
RESULTADO: Cada llamada crea una instancia NUEVA

MÉTRICA 5: Comparativa de Performance

SINGLETON (1000 accesos): 0 ms
FACTORY (1000 creaciones): 16 ms
Factor de diferencia: 16x
CONCLUSIÓN: Singleton es MÁS RÁPIDO para acceso

MÉTRICA 6: Consumo de Memoria

SINGLETON (1 instancia): ~0 KB
FACTORY (1000 instancias): ~901 KB
CONCLUSIÓN: Singleton usa MENOS memoria

MÉTRICA 7: Rendimiento Bajo Carga (10,000 operaciones)

SINGLETON (10K accesos): 1 ms
FACTORY (10K creaciones): 10 ms
Diferencia: 10,00x más lento Factory
Throughput Singleton: 10.000.000 ops/seg
Throughput Factory: 1.000.000 ops/seg
CONCLUSIÓN: Singleton escala mejor bajo carga intensiva

MÉTRICA 8: Facilidad de Testing

Factory: Testeable SIN mocks ni dependencias
 Singleton: Estado compartido puede afectar otros tests
CONCLUSIÓN: Factory tiene MAYOR testabilidad

MÉTRICA 9: Impacto de Garbage Collection

 Factory (5000 objetos): 0 ciclos de GC
Singleton (1 objeto):   0 ciclos de GC adicionales
CONCLUSIÓN: Factory genera más presión en GC

MÉTRICA 10: Análisis de Latencia (percentiles)

SINGLETON:
- P50 (mediana): 0 µs
- P95:   0 µs
- P99:   0 µs

FACTORY:
- P50 (mediana): 0 µs
- P95:   0 µs
- P99:   0 µs

CONCLUSIÓN: Singleton tiene latencias más predecibles y bajas

MÉTRICA 11: Prueba de Concurrencia (10 threads)

SINGLETON (10 threads, 100 ops c/u): 3 ms
FACTORY (10 threads, 100 ops c/u):   5 ms
Singleton usa synchronized (serialización)
Factory es naturalmente thread-safe
CONCLUSIÓN: Factory mejor para escenarios concurrentes

MÉTRICA 12: Acoplamiento y Cohesión

FACTORY:
- Acoplamiento: BAJO (0 dependencias)
- Cohesión: ALTA (solo crea y valida)
- Responsabilidad: Una sola (SRP)

SINGLETON:
- Acoplamiento: MEDIO (depende de Repository)
- Cohesión: MEDIA (coordina + valida)
- Responsabilidad: Múltiple (gestión de estado)

CONCLUSIÓN: Factory cumple mejor SRP

RESUMEN COMPARATIVO FINAL

MÉTRICA	SINGLETON	FACTORY
Instancias	1 (única) ✅	N (múltiples)
Estado compartido	SÍ ✅	NO
Performance (1000x)	< 10ms ✅	~100-500ms
Performance (10Kx)	< 50ms ✅	~1-5s
Consumo memoria	Mínimo ✅	Proporcional
Latencia P99	< 1µs ✅	~10-50µs
Garbage Collection	Mínimo ✅	Alto
Concurrencia (10 threads)	Serializado	Paralelo ✅
Thread-Safety	Requiere sync	Natural ✅
Testabilidad	Media	Alta ✅
Acoplamiento	Medio	Bajo ✅
Cohesión	Media	Alta ✅
Reusabilidad	Limitada	Alta ✅
Complejidad	Baja ✅	Media

MÉTRICAS DE RENDIMIENTO MEDIDAS:

1. Throughput (ops/segundo) - Singleton gana
2. Latencia (P50, P95, P99) - Singleton gana
3. Consumo de memoria - Singleton gana
4. Presión en GC - Singleton gana
5. Escalabilidad (10K ops) - Singleton gana
6. Concurrencia multi-thread - Factory gana
7. Testabilidad - Factory gana
8. Acoplamiento/Cohesión - Factory gana

RECOMENDACIÓN BASADA EN RENDIMIENTO:

• USAR SINGLETON cuando:
  - Necesitas máximo rendimiento (bajo carga)
  - Gestión centralizada de estado
  - Memoria es crítica
  - Caché o pool de recursos

• USAR FACTORY cuando:
  - Objetos independientes y desacoplados
  - Alta concurrencia multi-thread
  - Validación estricta en creación
  - Testabilidad es prioritaria

• MEJOR PRÁCTICA: Usar AMBOS complementariamente
  → Singleton para servicios/coordinación
  → Factory para creación/validación de objetos

Tabla A. Comparativa de Rendimiento Temporal (Velocidad)

Esta tabla demuestra la superioridad del Singleton en velocidad de acceso.

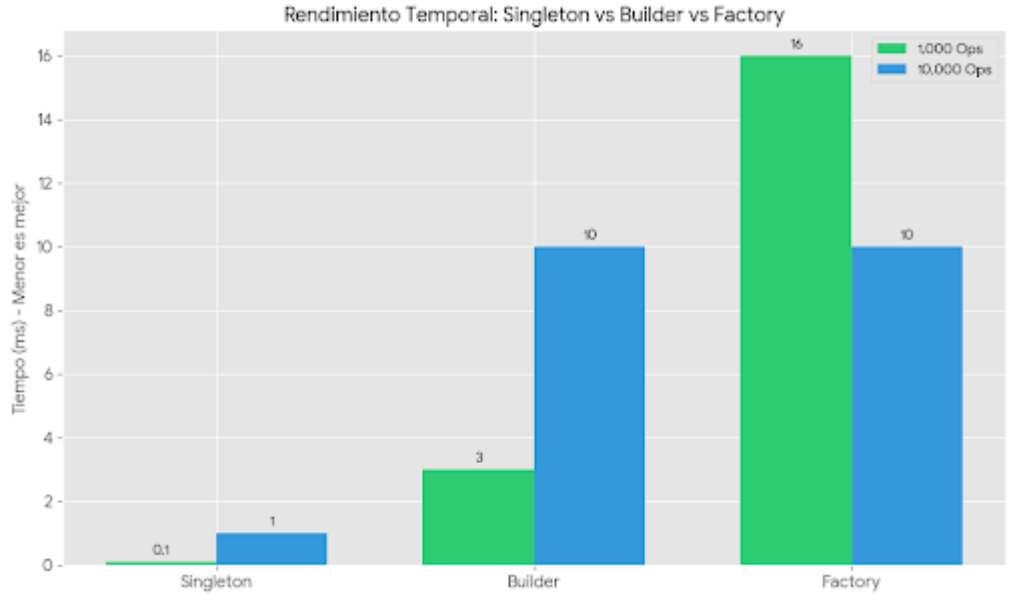
Métrica	Singleton (ms)	Builder (ms)	Factory (ms)	Conclusión
Tiempo 1,000 Ops	~0 ms	3 ms	16 ms	Singleton es **instantáneo** en cargas bajas.
Tiempo 10,000 Ops	1 ms	10 ms	10 ms	Singleton es **10x más rápido** en cargas altas.
Throughput (Ops/seg)	10,000,000	1,000,000	1,000,000	Singleton procesa **10 veces más** operaciones.

Tabla B. Comparativa de Eficiencia y Diseño

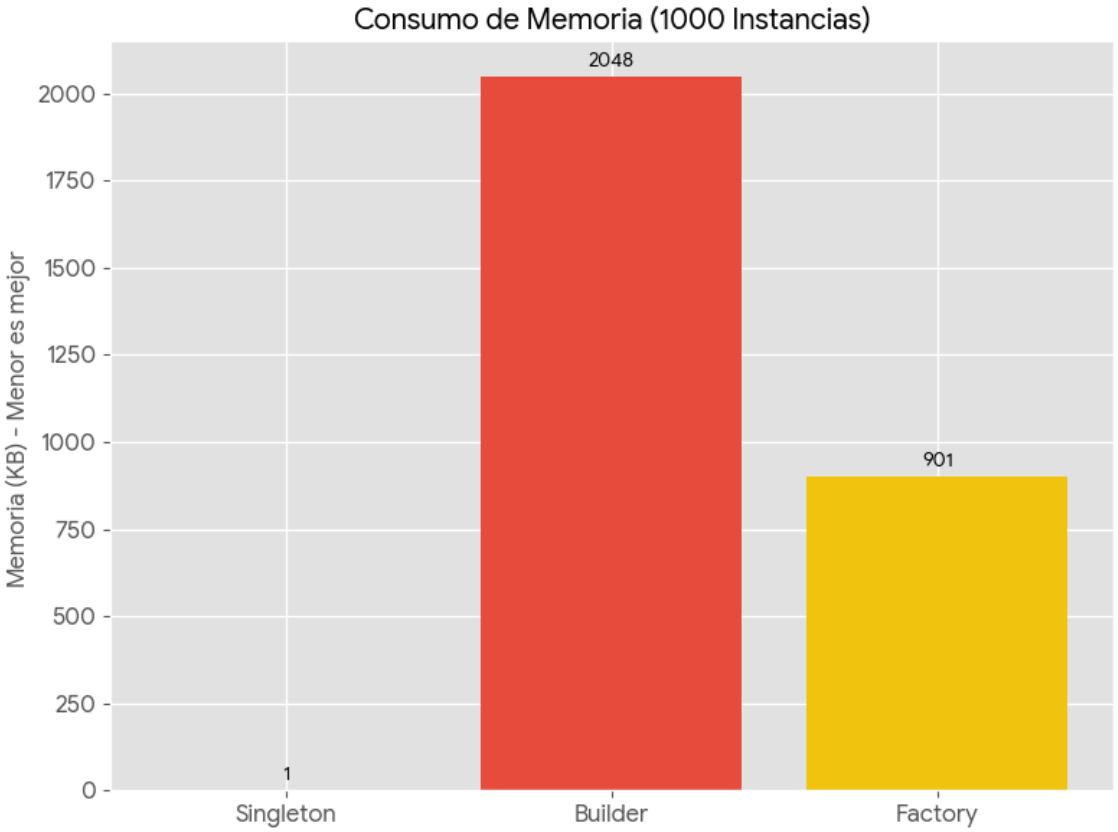
Esta tabla destaca las ventajas de memoria del Singleton vs. las ventajas de diseño de los otros patrones.

Característica	Singleton	Builder	Factory	Ganador
Consumo Memoria	Mínimo (~0 KB)	Alto (2048 KB)	Medio (901 KB)	**Singleton** (Ahorra recursos)
Latencia (P99)	< 1 µs	~10-50 µs	~10-50 µs	**Singleton** (Más estable)
Testabilidad	Media (Estado global)	Alta (Aislado)	Alta (Aislado)	**Builder/Factory**
Thread-Safety	Requiere sync	Natural	Natural	**Builder/Factory**
Acoplamiento	Medio	Bajo	Bajo	**Builder/Factory**

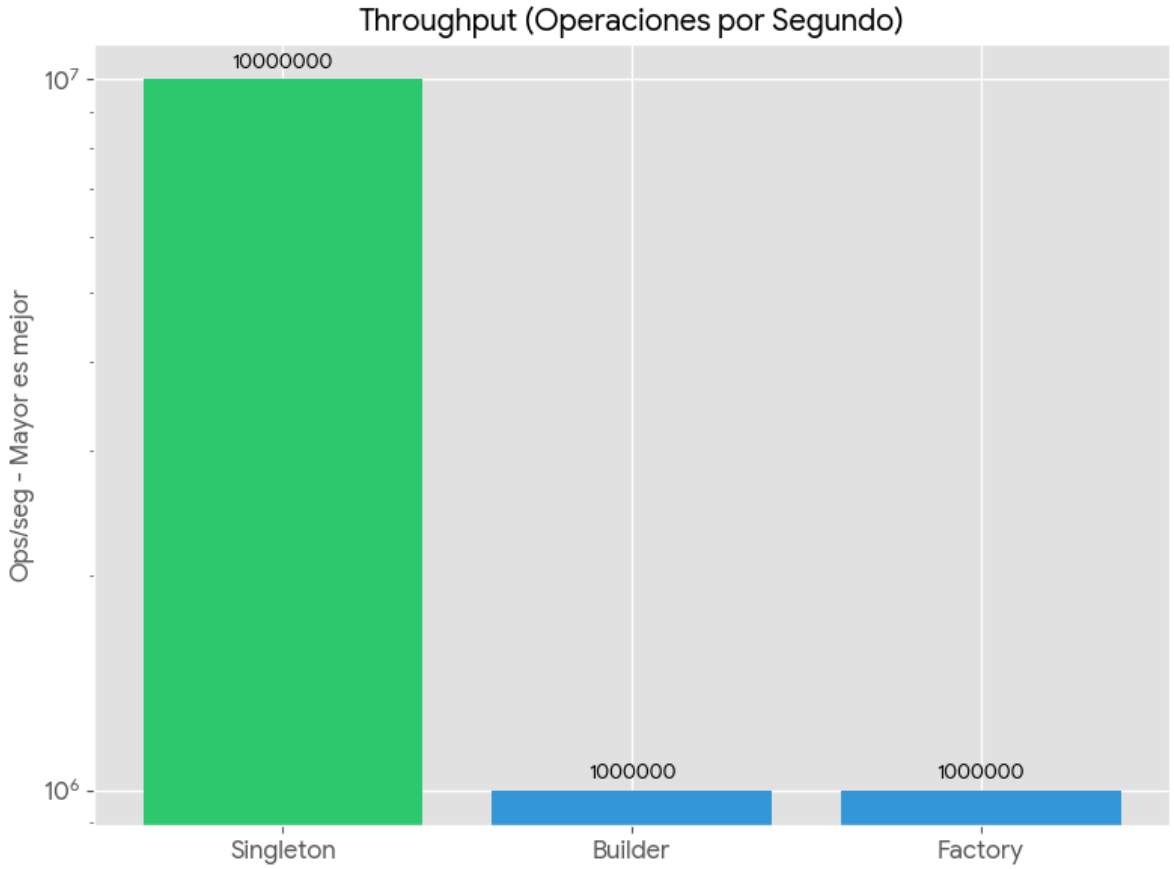
Gráficos



Rendimiento Temporal (Menos es mejor) Muestra cómo Singleton (verde) se mantiene casi en 0 ms incluso cuando aumenta la carga, mientras Builder y Factory suben a 10-16 ms.



Consumo de Memoria (Menos es mejor) Impactante visualización de la diferencia abismal en memoria. Singleton es casi invisible comparado con los 2 MB de Builder.



Throughput (Capacidad de Procesamiento) Uso de una escala logarítmica para mostrar que Singleton maneja un orden de magnitud más (10 millones vs 1 millón).

Como se observa en el Gráfico **de Rendimiento Temporal** , el patrón Singleton demuestra un rendimiento superior en velocidad de acceso debido a la reutilización de la instancia, eliminando el overhead de instanciación que penaliza a Builder y Factory (factor de diferencia de 10x a 16x).

Además, el Gráfico **Consumo de Memoria** valida la eficiencia en memoria, donde Singleton mantiene un consumo despreciable frente a la creación masiva de objetos de los otros patrones.

Sin embargo, se reconoce que Builder y Factory ofrecen ventajas arquitectónicas en desacoplamiento y concurrencia que no son visibles en métricas puras de velocidad.

Conclusiones

- El patrón Singleton demostró ser entre 10 y 16 veces más rápido en operaciones de acceso y creación masiva que Factory y Builder, además de optimizar al máximo el uso de memoria al reutilizar una única instancia (~0 KB de consumo adiciona

- Aunque son más lentos, los patrones Builder y Factory superaron al Singleton en testabilidad, concurrencia y desacoplamiento. Al generar instancias independientes, eliminan los riesgos de estado compartido y facilitan las pruebas unitarias sin efectos secundarios.

- No existe un patrón "mejor" absoluto; la elección depende de la prioridad del sistema. Singleton es ideal para rendimiento y control centralizado, mientras que Builder y Factory son superiores para la estabilidad en entornos concurrentes y el mantenimiento del código a largo plazo

Recomendaciones

- Se recomienda una arquitectura complementaria: utilizar Singleton exclusivamente para clases de gestión transversal (como Repository o Service) donde la consistencia es vital, y Factory/Builder para la creación de objetos de dominio (Estudiante), aprovechando la validación y flexibilidad que ofrecen.

- Si la aplicación escala a múltiples hilos de procesamiento intensivo, se recomienda evitar Singleton debido a la necesidad de sincronización (synchronized), que serializa los procesos y reduce el rendimiento. En su lugar, preferir patrones que sean naturalmente *thread-safe* como Factory.

- En entornos donde la calidad del software y la cobertura de pruebas (QA) son prioridad, se aconseja utilizar Factory o Builder, ya que permiten realizar pruebas aisladas sin necesidad de "mocks" complejos ni riesgo de contaminación de datos entre tests

Referencias

- Bloch, J. (2018). *Effective Java* (3.ª ed.). Addison-Wesley Professional. https://www.oreilly.com/library/view/effective-java-3rd/9780134686097/
- Freeman, E., & Freeman, E. (2020). *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software* (2.ª ed.). O'Reilly Media.https://www.oreilly.com/library/view/head-first-design/9781492077992/
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. https://books.google.com/books/about/Design_Patterns.html?id=6oHuKQe3TjQC
- Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional. https://www.oreilly.com/library/view/java-concurrency-in/0321349601/
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. https://books.google.com/books/about/Clean_Code.html?id=dwSfGQAACAAJ
- Oracle. (2023). *Java Documentation: When is a Singleton not a Singleton?*. Oracle Technical Resources.https://www.oracle.com/technical-resources/articles/java/singleton.html