Accenture

Academia Java Xideral

Entrega Semana 2
Singleton

Alumno
Daniel Israel Ceballos Uc

Profesor
Miguel Rugerio

Patron de diseño Singleton

El patrón Singleton es un patrón de diseño creacional cuyo objetivo es asegurar que una clase tenga una única instancia en todo el sistema, y que se proporcione un punto de acceso global a dicha instancia. Es decir, si se necesita que un objeto se cree solo una vez y se reutilice durante toda la ejecución del programa, el Singleton garantiza esa unicidad y control del ciclo de vida del objeto.

Este patrón encapsula su propia instancia y evita que otras clases creen nuevas mediante un constructor público. Solo se permite acceder a la instancia a través de un método estático, el cual verifica si la instancia ya existe; de no existir, la crea. Este mecanismo proporciona un control total sobre la creación del objeto.

Motivación

Durante el diseño de software orientado a objetos, puede haber situaciones donde se necesita exactamente una instancia de una clase. Ejemplos comunes incluyen gestores de configuración, controladores de impresión, conexiones a bases de datos o sistemas de logging. En estos casos, permitir la creación de múltiples instancias puede ser problemático, ya que introduciría inconsistencias, múltiples accesos no sincronizados a recursos compartidos o un consumo innecesario de memoria.

La motivación principal del patrón Singleton radica en garantizar la existencia de una sola instancia de la clase, lo cual simplifica el diseño, mejora el rendimiento y previene errores derivados de múltiples instancias. Además, este patrón permite que la instancia se cree únicamente cuando se necesita, lo que puede ser útil en sistemas donde la eficiencia en el uso de recursos es importante.

Características

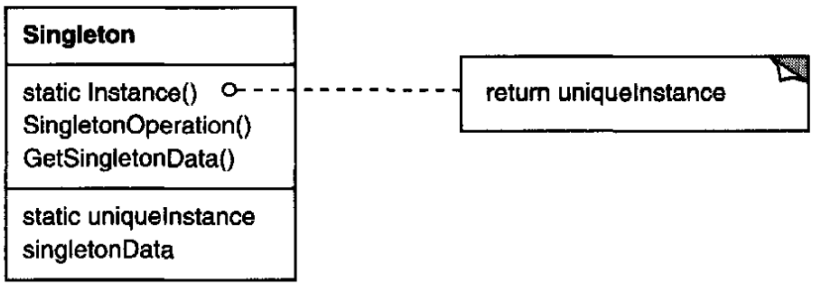
El patrón Singleton es especialmente útil cuando se requiere control y consistencia sobre objetos que coordinan recursos globales dentro de un sistema. Las principales características del patrón Singleton son:

- **Instancia única:** Solo se permite una instancia de la clase en todo el sistema.
- **Acceso global:** Se puede acceder a dicha instancia desde cualquier parte del programa mediante un método estático.
- **Encapsulamiento del constructor:** El constructor de la clase se declara como privado para evitar que otros objetos creen nuevas instancias.
- **Inicialización controlada:** Se puede implementar tanto eager (creación anticipada) como lazy (creación diferida), dependiendo de los requerimientos de rendimiento y uso de memoria.
- **Gestión de hilos:** En sistemas concurrentes, es importante asegurar que la creación de la instancia única sea segura frente a múltiples hilos. Esto se logra mediante sincronización o técnicas como el doble bloqueo.
- **Persistencia y serialización:** En entornos distribuidos o cuando se serializa un Singleton, es necesario implementar mecanismos para que no se creen nuevas instancias al deserializarlo.

Implementación

Para implementar el patrón Singleton en Java debe asegurarse que únicamente una referencia sea creada para su uso de manera global, para esto se pueden seguir la siguiente serie de pasos:

1. Añade un campo estático privado a la clase para almacenar la instancia Singleton.
2. Declara un método de creación estático público para obtener la instancia Singleton. Si la instancia de Singleton ya ha sido creada el método debe devolver dicha instancia en caso contrario es necesario crear una instancia y retornarla.
3. Declara el constructor de clase como privado. El método estático de la clase seguirá siendo capaz de invocar al constructor, pero esta no podrá ser instanciada externamente.



Referencias

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

Bloch, J. (2018). Effective Java (3rd ed.). Addison-Wesley.

Refactoring Guru. Singleton en Java. Refactoring.Guru. Recuperado el 15 de junio de 2025: https://refactoring.guru/design-patterns/singleton/java