Inyección de Dependencias

La inyección de dependencias es una forma de darle estructura al código, esta forma consiste en que las clases reciban los objetos que necesitan desde afuera, en lugar de crearlos ellas mismas. Esto es útil para que las clases estén menos conectadas o sean menos dependientes entre sí y sean más fáciles de probar, modificar y reutilizar.

Un ejemplo para poder comprenderlo mejor es cuando se tiene una clase, como EmailService, la cual se encarga de enviar correos electrónicos. Normalmente se podría crear un objeto EmailSender directamente desde la clase que haga el envío, pero con la inyección de dependencias, ese objeto EmailSender se le pasa a EmailService desde fuera por un constructor, por ejemplo. De esta forma la clase no necesita saber cómo funciona el envío, solo está utilizando el objeto que se le entrega.

Hay tres formas comunes de aplicar la inyección de dependencias:

- Por constructor: el objeto necesario se pasa cuando se crea la clase.
- Por método setter: se le asigna después de haber sido creada.
- Por campo: cuando un framework (como Spring) lo gestiona automáticamente usando anotaciones.

La inyección de dependencias es útil en aplicaciones grandes, donde una clase puede depender de muchas otras. Una clase que muestra la información al usuario no debería conectarse directamente a la base de datos, sino recibir un servicio que se encargue de eso. Así cada parte hace lo suyo y no se mezclan responsabilidades para que se tenga una estructura mejor organizada.

Además, este método facilita mucho las pruebas. Si se quiere probar una clase que normalmente se conecta a una base de datos o que envía correos, se le puede inyectar una versión simulada que no haga esas acciones reales. Esto ahorra tiempo, evita errores y hace más sencillo comprobar que el código funciona como debería.

Al final, lo que se busca con la inyección de dependencias es mantener el código limpio y más ordenado, permitiendo que cada clase se enfoque en una sola tarea y pueda trabajar con diferentes implementaciones sin necesidad de ser modificada.

Excepciones en Java

Las excepciones en Java son errores que pueden ocurrir mientras el programa se está ejecutando e interrumpen el flujo normal del código. Por ejemplo, si una aplicación trata de abrir un archivo que no existe o si intenta hacer una operación matemática inválida, como dividir entre cero, entonces se produce una excepción. Las excepciones evitan que el programa de detenga de forma inesperada, permitiendo que se capturen los errores en el momento que ocurren para decidir cómo solucionarlo.

Hay dos tipos de excepciones:

1. Checked (Checadas): Son las que java obliga a manejar. Están relacionadas con operaciones que dependen de factores externos que pueden fallar, como leer un archivo, conectarse a una base de datos o interactuar con la red. Cuando se trabaja con este tipo de excepciones, es necesario usar bloques try-catch para controlarlas, o declarar en el método que pueden ocurrir usando throws. Ejemplos comunes de estas excepciones son IOException o FileNotFoundException.
2. Unchecked (No checadas): Estas excepciones no son obligatorias de manejar, pero pueden causar que el programa falle si no se controlan adecuadamente. Normalmente están relacionadas con errores de lógica o situaciones imprevistas dentro del propio código, como cuando se accede a una posición que no existe en un arreglo, por lo que se sale del rango (ArrayIndexOutOfBoundsException) o intentar utilizar una variable que no ha sido inicializada (NullPointerException). Aunque Java no exige manejarlas, es importante saber anticiparse y poder controlar estos casos para evitar errores.

Cuando se trabaja con excepciones es muy útil el uso del multicatch, ya que este permite capturar varios tipos de excepciones diferentes en un mismo bloque catch, pero solo si se quiere realizar la misma acción para todas las excepciones. Esto hace que el código sea más limpio y fácil de leer.

Otra característica muy práctica es el try-with-resources, el cual se usa para trabajar con recursos como archivos o conexiones a bases de datos. Este tipo de bloques permite que java cierre automáticamente esos recursos al terminar de usarse. Por lo que se reduce el riesgo de olvidos o errores relacionados con los recursos que se queden abiertos.

Preguntas de dudas

¿Cuándo se recomienda usar la inyección por constructor y cuándo por setter?

¿Se pueden combinar excepciones checked y unchecked en el mismo bloque try?