Actividad 1.10 Tratamiento de ficheros con Excepciones

Marcos Alcalde

A. Investiga sobre las Excepciones .

Cuando se habla de excepciones en JAVA:

1. ¿Qué diferencia hay entre las excepciones checked y unchecked en Java, y cómo afecta esto a la forma de escribir el código?

Checked (verificadas): Heredan de Exception. El compilador te obliga obligatoriamente a tratarlas con un bloque try-catch o declarándolas con throws. Obligan a prever errores externos inevitables (como que un fichero no exista).

Unchecked (no verificadas): Heredan de RuntimeException. Ocurren en ejecución y el compilador no te obliga a capturarlas. Se usan para errores lógicos de programación sin saturar el código de bloques obligatorios.

2. ¿Qué criterio se debe usar para decidir si una excepción personalizada debe heredar de Exception o de RuntimeException?

Heredar de Exception (checked): Se usa si el error es recuperable y quieres obligar al código llamante a gestionarlo o informar al usuario.

Heredar de RuntimeException (unchecked): Se usa para fallos de lógica o programación donde el llamante no puede solucionarlo al momento y se quiere evitar llenar el código de try-catch innecesarios.

3. ¿Qué ocurre si en un bloque try se lanza una excepción (ocurre una excepción y se interrumpe el flujo normal) que no se captura en ningún catch y tampoco se declara con throws (el método no declara el throws)? ¿En qué casos eso es válido?

Si es **checked** el programa no compila. Si es unchecked, compila pero al ejecutarse se interrumpe el flujo normal, muestra la traza (*stack trace*) por consola y el programa finaliza de forma abrupta.
Solo es válido con excepciones *unchecked* cuando ocurre un error grave e inesperado y no tiene sentido que el programa siga funcionando en un estado incorrecto.

4. En el tratamiento que te voy a proponer a continuación se van a utilizar “exceciones checked”. ¿Qué hace que un proyecto profesional basado en Spring Boot dichas excepciones se definirían como “unchecked”? Investiga sobre 3 de los principales motivos.

Evita acoplamiento: Impide tener que escribir throws en cadena por todas las firmas de métodos, interfaces y servicios, mejorando la legibilidad.

Manejadores globales: Permite centralizar la gestión de errores en una sola clase global mediante anotaciones (@ControllerAdvice), evitando repetir bloques try-catch.

Rollback automático: Las operaciones con bases de datos (@Transactional) deshacen los cambios automáticamente solo ante excepciones unchecked (RuntimeException), facilitando la gestión de transacciones.