Jilai Xu

Actividad 1.10: Incluir Excepción de proyecto en Mapamundi

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

Excepciones *Checked* (Verificadas): Son aquellas que hereden directamente de Exception (pero no de RuntimeException). El compilador de Java obliga obligatoriamente a gestionarlas de forma explícita. Esto significa que el código que invoque a un método lanzador debe envolverlo en un bloque try-catch o propagar la excepción declarándola en la firma del método mediante la cláusula throws. Afecta escribiendo un código más defensivo, rígido y estructurado, ya que obliga al programador a anticiparse a errores recuperables externos (como fallos de red o de ficheros).

Excepciones *Unchecked* (No verificadas): Son aquellas que heredan de RuntimeException (como NullPointerException o IllegalArgumentException). El compilador **no** exige su captura ni su declaración obligatoria con throws. Están diseñadas para representar errores de programación o fallos lógicos graves que el código cliente no suele poder manejar de forma recuperable en ese mismo punto. Afecta escribiendo un código más limpio y libre de bloques try-catch repetitivos cuando los errores no se pueden solucionar de inmediato.

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

Heredar de Exception (*Checked*): Se debe utilizar cuando el error representa una situación anómala pero esperada y recuperable por el flujo de la aplicación, donde el código que realiza la llamada tiene la capacidad lógica y la obligación de solucionar el problema o notificar al usuario de forma controlada (por ejemplo, un fallo de validación de negocio o un archivo de configuración ausente que se puede volver a solicitar).

Heredar de RuntimeException (*Unchecked*): Se debe utilizar cuando la excepción se debe a un error de programación, fallo lógico o estado inválido irrecuperable (por ejemplo, pasar un argumento null a un método que no lo tolera, violar una precondición interna, o errores de configuración del sistema de los cuales la aplicación no puede recuperarse automáticamente).

Jilai Xu

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?

Qué ocurre: Si la excepción lanzada es de tipo *checked*, el código no compilará, ya que el compilador de Java detectará que no está gestionada ni propagada. Si la excepción es de tipo *unchecked* (RuntimeException), el código compilará perfectamente, pero al ejecutarse interrumpirá el flujo normal del hilo actual, propagándose hacia arriba en la pila de llamadas (*stack trace*) hasta llegar al método main o al hilo principal, lo que provocará la terminación abrupta del programa (si no hay un mecanismo global de captura de excepciones no controladas).

Cuándo es válido: Es perfectamente válido únicamente cuando se trata de excepciones de tipo *unchecked* (RuntimeException), ya que Java no exige formalmente su declaración ni captura obligatoria en la compilación.

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

Evitar la contaminación de la interfaz (*API Pollution*): Las excepciones *checked* obligan a propagar la cláusula throws a través de todas las capas intermedias (controladores, servicios, repositorios), acoplando fuertemente las firmas de los métodos a detalles de implementación de nivel inferior.

Manejo centralizado y global de errores: Spring Boot proporciona mecanismos potentes y limpios como @ControllerAdvice y @ExceptionHandler. Esto permite capturar todas las excepciones de negocio o técnicas de forma centralizada en una sola clase global, evitando saturar el código de bloques try-catch repetitivos en cada método.

Adaptación a contratos transaccionales y arquitecturas REST: Los frameworks modernos manejan la lógica de rollback de transacciones de base de datos automáticamente basándose en excepciones *unchecked*. Además, en servicios web RESTful, los errores técnicos o de negocio se traducen directamente en códigos de estado HTTP (como un 404 o 500) mediante interceptores globales, haciendo innecesario que las capas de negocio manejen excepciones de forma intrusiva.