David Torres Aranha

Actividad 1.10: Incluir Excepción de proyecto en Mapamundi

Investiga sobre las excepciones

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

Las excepciones *checked* son aquellas que heredan de `Exception` (sin ser `RuntimeException`), como `IOException` o `SQLException`. El compilador obliga a tratarlas: o se capturan con `try/catch` o se declaran en la firma del método con `throws`. Si no se hace, el código no compila.

Las excepciones *unchecked* son las que heredan de `RuntimeException` (y los `Error`), como `NullPointerException` o `IllegalArgumentException`. El compilador no obliga a capturarlas ni a declararlas.

Al escribir el código:

- Con las *checked*, el `throws` se va propagando por todos los métodos de la cadena (interfaces, implementaciones y capas superiores), lo que hace el código más verboso pero obliga al programador a tener en cuenta el error.
- Con las *unchecked*, el código es más limpio, pero es más fácil olvidarse de tratarlas y que aparezcan errores en tiempo de ejecución.

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

El criterio principal es si el llamador puede y debe hacer algo para recuperarse del error:

- Heredar de `Exception` (*checked*): cuando el error es previsible y recuperable, y queremos obligar a quien llama al método a tratarlo (por ejemplo, un fichero que no existe, un saldo insuficiente, una regla de negocio incumplida).
- Heredar de `RuntimeException` (*unchecked*): cuando el error se debe a un fallo de programación o a una situación que el llamador normalmente no puede resolver (argumentos inválidos, configuración incorrecta, errores técnicos irrecuperables), o cuando no queremos "contaminar" todas las firmas con throws.

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?

Depende del tipo de excepción:

- Si es *checked*: se produce un error de compilación, porque el compilador exige capturarla o declararla con `throws`.
- Si es *unchecked*: el código compila. En ejecución, al lanzarse la excepción se interrumpe el flujo normal, se ejecuta el bloque `finally` (si existe) y la excepción se propaga hacia arriba por la pila de llamadas. Si ningún método la captura, llega al `main` y el programa termina abruptamente mostrando el *stack trace*.

Es válido únicamente cuando la excepción es unchecked (`RuntimeException` y sus subclases, o `Error`), ya que el compilador no exige tratarlas.

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.

Los tres motivos principales son:

1. Evitar contaminar las firmas de los métodos: en una aplicación Spring Boot hay muchas capas (controller, service, repository). Con checked habría que añadir `throws` en todas ellas, aunque las capas intermedias no puedan hacer nada útil con el error. Con unchecked, la excepción sube sola hasta quien sabe tratarla.
2. Gestión de transacciones (rollback): por defecto, Spring con `@Transactional` hace rollback automático solo ante excepciones unchecked (`RuntimeException` y `Error`). Con checked, la transacción se confirma (commit) salvo que se configure explícitamente con `rollbackFor`, lo que puede dejar datos inconsistentes.
3. Tratamiento centralizado de errores: Spring permite capturar las excepciones en un único punto mediante `@ControllerAdvice` y `@ExceptionHandler`, convirtiéndolas en respuestas HTTP adecuadas (404, 400, 500...). Esto elimina la necesidad de bloques `try/catch` repartidos por el código. Además, el propio Spring ya traduce las excepciones checked de acceso a datos (`SQLException`) a su jerarquía unchecked `DataAccessException`, siguiendo esta misma filosofía.