

ACTIVIDAD 1.10 - Excepciones en Mapamundi

A. Investigación sobre excepciones

¿Qué diferencia hay entre excepciones checked y unchecked?

Jaba divide las excepciones en dos grandes grupos según si el compilador obliga o no a tratarlas.

- **Checked (comprobadas):** heredan de `Exception` sin pasar por `RuntimeException`. Ejemplos: `IOException`, `SQLException`. El compilador me obliga a hacer una de esta sdos cosas: capturarla con `try/catch` o avisar de que mi método puede lanzarla co n`throws`. Si no hago ninguna, el código no compila. Suelen representar problemas externos que pueden pasar aunque mi código esté bien, como que un fichero no exista.
- **Unchecked (no comprobadas):** heredan de `RuntimeException` (o de `Error`). Ejemplos: `NullPointerException`, `ArrayIndexOutOfBoundsException`. El compilador no me obliga a tratarlas. Suelen representar errores de programación, como usar una variable que vale `null` o acceder a una posición que no existe.

Con las checked, el `throws` se va propagando por todos los métodos de la cadena: si un método de abajo la lanza, el de encima tiene que capturarla o declararla también, y así sucesivamente. Con las unchecked el código queda más limpio, pero es más fácil olvidarse de tratarlas, porque el compilador no avisa.

▼

¿Cuándo una excepción propia debe heredar de Exception y cuándo de RuntimeException?

El criterio que he entendido es este: **¿puede quien llama al método hacer algo útil para recuperarse del error?**

- Si sí puede (reintentar, avisar al usuario, usar otro valor), la excepción debe ser **checked** y heredar de `Exception`. Así el compilador le obliga a pensar qué hacer.
- Si no puede)es un fallo de programación o algo irrecuperable), debe ser **unchecked** y heredar de de `RuntimeException`. Obligar a capturarla solo llenaría el código de `try/Catch` que no sirven para nada.

En esta práctica se usa `Exception` porque queremos que cualquier método que acceda a datos tenga que gestionar el posible fallo.

▼ ¿Qué pasa si una excepción no se captura ni se declara con throws?

La excepción no se queda en ese método. Sube por la pila de llamadas: pasa al método que lo llamó, luego al que llamó a ese, y así hasta llegar al `main`. Si nadie la captura por el camino, la JVM termina el programa y muestra por pantalla el *stack trace* (la lista de métodos por los que pasó el error). Por el camino sí se ejecutan los bloques `finally` y se cierran recursos de un `try-with-resources`.

Es válido cuando la excepción es unchecked. Con una checked no compila, porque el compilador exige capturarla o declararla.

▼

¿Por qué en un proyecto profesional con Spring Boot estas excepciones serían unchecked?

Estos son los tres motivos principales que he encontrado:

1. **No "ensucian" las firmas de los métodos.** Con checked, el `throws` tendría qeu repetirse en el controlador, el servicio y el DAO. Con unchecked, las capas intermedias no declaran nada y el código queda más limpio.
2. **Desacoplan las capas de la tecnología.** Si el servicio declarase `throws SQLException` o `IOException`, estaría atado a cómo se guardan los datos. Spring convierte esos errores en su propia jerarquía (`DataAccessException`, que es unchecked), de modo que si cambio de base de datos no tengo que tocar las capas superiores.
3. **Transacciones y gestión centralizada de errores.** Con `@Transactional`, Spring hace rollback automático con excepciones unchecked, pero no con las checked. Además, con unchecked se

pueden tratar todos los errores en un único sitio (`@ControllerAdvice`) en vez de capturarlos uno a uno.

▼ B. Tratamiento de excepciones en Mapamundi

▼ 1. Preparación del proyecto

Partí de mi proyecto de la actividad 1.9 (`Ficheros_09`) y creé una copia con el nombre `Ficheros_10`. Dentro creé el paquete `org.palomafp.aadd.rmr.excepciones` y metí las 4 clases de excepciones que da el profesor.

Los ficheros del profesor venían con el paquete `org.palomafp.aadd.inm.excepciones` (`inm` son sus iniciales), y por eso me daban error en Eclipse. Lo solucioné cambiando la primera línea de cada uno a `package org.palomafp.aadd.rmr.excepciones;`.

La jerarquía queda así:

- `MapaMundiException` (abstracta, hereda de `Exception`)
  - `NegocioException`
  - `TecnicoException`
    - `AccesoDatosException`

Todas son checked, porque la clase de arriba hereda de `Exception`. Cada una se usa para algo distinto:

- **`AccesoDatosException`**: fallos técnicos al leer o escribir los datos (por ejemplo, los ficheros CSV).
- **`TecnicoException`**: fallos técnicos en general (por ejemplo, al leer el `.properties`).
- **`NegocioException`**: cuando se rompe una regla de negocio.

Dos detalles que me ayudaron a entenderlas. Primero, `MapaMundiException` es abstracta y sus constructores son `protected`, así que nadie puede hacer `new MapaMundiException(...)` directamente: solo sirve de "padre" común. Segundo, `NegocioException` no tiene constructor con causa, porque una regla de negocio incumplida no viene de otra excepción anterior, mientras que un error técnico sí suele venir de otro (por ejemplo, una `IOException`).

▼ 2. Cambios que hice en el código

a) `GestorPropiedades` lanza `TecnicoException`.

Antes, si fallaba la lectura del `.properties`, el `catch` solo imprimía un mensaje y el programa seguía adelante con las propiedades vacías. Eso causaba un `NullPointerException` más tarde, que

era confuso. Ahora lanza `TecnicoException` en dos casos: cuando no encuentra el fichero, y cuando hay una `IOException` al leerlo (en este caso le paso la excepción original como causa).

Para poder hacerlo tuve que mover la carga del fichero a un método `cargar()`. El motivo es que el código estaba en un bloque `static`, y un bloque `static` no puede declarar `throws`, así que no podía lanzar una excepción checked desde ahí. Como consecuencia, el método `get()` ahora declara `throws TecnicoException`.

---

b) Las interfaces declaran `throws`.

Todos los métodos de `IContinenteDAO` e `IPaisDAO` llevan `throws AccesoDatosException`. En `IPaisDAO` se añade también `NegocioException` (lo explico en la pregunta 5).

---

c) Las implementaciones lanzan la excepción en vez de imprimir.

En `ContinenteDaoFichero` y `PaisDaoFichero` hice tres cosas:

- Los atributos `FICHERO` y `SEPARADOR` ya no se asignan donde se declaran, sino dentro de un **constructor**. La razón es la misma de antes: ahora `GestorPropiedades.get()` lanza una excepción checked, y en la declaración de un atributo no puedo poner un `try/catch`.
- Si `GestorPropiedades` lanza una `TecnicoException`, el constructor la **convierte** en `AccesoDatosException` y le pasa la original como causa.
- Los `catch (IOException e)` ya no imprimen nada. Ahora hacen `throw new AccesoDatosException("mensaje", e)`. Así no pierdo el error real (la causa), y quien llame decide qué hacer con él.

---

d) El `main` recoge todas las excepciones.

Metí todo el `switch` de `GestorMapaMundi` dentro de un `try` con tres `catch`: `AccesoDatosException`, `TecnicoException` y `NegocioException`. El orden importa. `AccesoDatosException` hereda de `TecnicoException`, así que tiene que ir **antes**. Si pusiera primero el de `TecnicoException`, este capturaría también las de acceso a datos y el catch de `AccesoDatosException` nunca se ejecutaría (Java lo detecta y da error).

El enunciado no pedía llamar a los países desde el `main`, pero añadí los casos `6` (países de un continente) y `7` (país por id). Lo hice para poder probar la `NegocioException` de la pregunta 5 y que el `catch` fuera alcanzable.

▼ 3. Errores que me aparecieron y cómo los entendí

Error 1: “Exception AccesoDatosException is not compatible with throws clause in IContinenteDAO...”

- **Qué pasó:** en las clases que implementan la interfaz (`ContinenteDaoFichero`) me salían errores en las declaraciones de los métodos, mientras que la interfaz no daba ninguno.
- **Por qué:** una clase que implementa una interfaz no puede lanzar una excepción checked que la interfaz no haya declarado. La interfaz es como un contrato: si dice que el método no lanza nada, la implementación no puede lanzar `AccesoDatosException`. La interfaz no daba error porque una interfaz sin `throws` compila perfectamente. El error aparece en quien la implementa.
- **Solución:** añadir `throws AccesoDatosException` a cada método de las interfaces y guardar.

Error 2: "Unreachable catch block for NegocioException. This exception is never thrown from the try statement body"

- **Qué pasó:** puse un `catch` para `NegocioException` en el `main`, pero me daba error al compilar.
- **Por qué:** Java no permite capturar una excepción checked si nada dentro del `try` puede lanzarla, porque ese `catch` no serviría para nada. Es como poner un cartel de "salida de emergencia" en una pared sin puerta. En ese momento todavía no había escrito ningún `throw new NegocioException(...)`. Con `TecnicoException` no daba error porque `AccesoDatosException` hereda de ella, así que lo que lanza el `try` sí podía ser una `TecnicoException`.
- **Solución:** quité el `catch` de forma temporal y lo volví a poner cuando implementé la pregunta 5, cuando ya había algo que lanza `NegocioException`.

Error 3: `ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0`

- **Qué pasó:** salió al ejecutar `GestorMapaMundi` con *Run As → Java Application*.
- **Por qué:** el `main` lee `args[0]` para saber qué operación hacer, y como no le pasé ningún argumento, el array estaba vacío.
- **Solución:** ejecutar desde *Run Configurations → pestaña Arguments* y escribir el argumento, por ejemplo `1`.

▼ 4. Pruebas que hice

Argumentos	Qué comprobaba	Resultado
`1`	Listar continentes	Salen los 5 continentes
`3 06 Prueba`, `5 06 Modificado`, `4 06`	Añadir, modificar y borrar	Funcionan y vuelven los 5 originales
`7 105`	País con continente correcto	`Filipinas (105) con capital en Manila pertenece al continente Asia (03)`

Argumentos	Qué comprobaba	Resultado
`7 105` tras cambiar el continente de Filipinas a `99` en el fichero de países	Excepción de negocio	`Error de negocio: El país Filipinas referencia un continente que no existe: 99`
`1` tras añadir `XXX` a la ruta de continentes en el `.properties`	Excepción de acceso a datos	`Error de acceso a datos: ...` y `Causa: ...NoSuchFileException`

En la última prueba se ve bien el encadenamiento: el fichero no existe, `Files.readAllLines` lanza una `NoSuchFileException` (que es una `IOException`), el DAO la captura y lanza una `AccesoDatosException` guardando la original como causa, y el `main` imprime las dos cosas. Después de cada prueba restauré el fichero o la ruta originales.

▼ 5. ¿Qué excepción usaría para un país con un continente que no existe?

Usaría **`NegocioException`**.

El razonamiento es este. Cuando un país referencia un continente que no existe, el fichero se lee perfectamente: no hay ningún fallo de lectura ni de configuración, así que no es un problema técnico (`TecnicoException` y `AccesoDatosException` no encajan). Lo que ocurre es que los **datos son incoherentes y rompen una regla del negocio**: "todo país debe pertenecer a un continente que exista".

También encaja que `NegocioException` no lleve causa, porque aquí no hay ninguna excepción previa. El problema es simplemente que `obtenerContinentePorCodigo` devuelve `null`.

En el código, después de buscar el continente del país, compruebo `if (continente == null)` y lanzo `NegocioException`. Como esta excepción no hereda de `AccesoDatosException`, los métodos de `IPaisDAO` y de `PaisDaoFichero` tienen que declarar los dos: `throws AccesoDatosException, NegocioException`.

La respuesta de la IA fue la siguiente:

```
# Pregunta a la IA:
Estoy en la implementación PaisDaoFichero. La operación de consulta
de país por ID usa el DAO de continentes para cargar la información
del continente (obtener continente por código). Quiero lanzar una e
xcepción en el caso de que un país contenga la referencia a un cont
inente que no exista (la operación devuelve null). ¿Cuál de las exc
epciones del proyecto (NegocioException, TecnicoException o AccesoD
atosException) utilizarías?

# Respuesta de la IA:
Utilizaría NegocioException.

El motivo es que el fallo no es técnico. El fichero de países se ha
```

```
leído correctamente y el de continentes también: no hay ningún prob
lema de lectura, de configuración ni de acceso a los datos. Por eso
AccesoDatosException y TecnicoException no encajan, ya que están pe
nsadas para fallos de infraestructura, como un fichero que no exist
e o un .properties ilegible.
Lo que ocurre es que los datos son incoherentes y rompen una regla
del dominio: "todo país debe pertenecer a un continente que exist
a". Es una violación de una regla de negocio, y esa es justo la sit
uación para la que sirve NegocioException.
También encaja con el diseño de la jerarquía, porque NegocioExcepti
on no tiene constructor con causa. Aquí no hay ninguna excepción pr
evia que encadenar: el problema es simplemente que obtenerContinent
ePorCodigo devuelve null.
Implementación recomendada: después de buscar el continente, compro
bar si es null y lanzar la excepción con un mensaje que identifique
el dato problemático (país y código de continente). Como NegocioExc
eption no hereda de AccesoDatosException, los métodos de IPaisDAO y
de PaisDaoFichero deben declarar ambas: throws AccesoDatosExceptio
n, NegocioException.
Matiz: si el null se debiera a un fallo al leer el fichero de conti
nentes, ese error ya se lanzaría como AccesoDatosException desde Co
ntinenteDaoFichero. El null solo llega cuando la lectura ha ido bie
n y el código simplemente no está en el fichero, y eso es un proble
ma de datos, no técnico.
```