# Java Excepciones Aprenda a crear, lanzar y controlar excepciones

## Conclusión 1: Pila de ejecución

* Qué es, para qué sirve y cómo funciona la pila de ejecución.
* Qué es la depuración (debug) y para qué sirve.
* Cómo utilizar Eclipse y su perspectiva de debug.
* Cómo cambiar entre perspectivas de Eclipse.

|  |
| --- |
|  |

## Conclusión 2: Tratamiento de excepciones

* Qué son las excepciones, para qué sirven y por qué se utilizarlas.
* Cómo analizar el seguimiento de excepciones o stacktrace.
* Manejar excepciones con bloques try-catch.
* Manejar una excepción lanzada dentro del bloque catch.
* Manejar múltiples excepciones con más de un bloque catcho usando Multi-Catch usando el pipe (|).

|  |
| --- |
|  |

## Conclusión 3: Lanzando excepciones

* Cómo lanzar excepciones.
* Cómo asignar un mensaje a la excepción.

|  |
| --- |
|  |

## Conclusión 4: Checked y unchecked

Si hiciste el ejercicio ¿será Miguel comprendió la clase ?, recordará lo que aprendimos. Para solucionarlo aún más, enumeramos los temas de esta clase:

* Existe una gran jerarquía de clases que representan excepciones. Por ejemplo, ArithmeticException es hija de RuntimeException, que hereda de Exception, que a su vez es hija de la clase de excepciones más ancestral, Throwable. Conocer bien esta jerarquía significa saber cómo usar las excepciones en su aplicación.
* Throwable es la clase que debe extenderse para poder lanzar un objeto en la pila (usando la palabra reservada throw)
* Es en la clase Throwable donde tenemos casi todo el código relacionado con las excepciones, incluyendo getMessage() e printStackTrace (). El resto de la jerarquía tiene solo algunas sobrecargas de constructores para comunicar mensajes específicos.
* La jerarquía que comenzó con la clase Throwable se divide en excepciones y errores. Las excepciones se utilizan en los códigos de aplicación. Los errores son utilizados exclusivamente por la máquina virtual.
* Las clases que heredan de Error se utilizan para informar errores en la máquina virtual. Los desarrolladores de aplicaciones no deben crear errores que hereden de Error.
* StackOverflowError es un error de máquina virtual para informar que la pila de ejecución no tiene más memoria.
* Las excepciones se dividen en dos categorías amplias: las que el compilador comprueba obligatoriamente y las que no.
* Los primeros se denominan checked y se crean por pertenecer a una jerarquía que no pasa RuntimeException.
* Los segundos están unchecked y se crean como descendientes de RuntimeException.

|  |
| --- |
|  |

## Conclusión 5: Aplicando excepciones

* cómo crear un bloque catch genérico utilizando la clase Exception;
* cómo crear una nueva excepción SaldoInsuficienteException;
* cómo transformar la excepción en checked o unchecked.

|  |
| --- |
|  |

## Conclusión 6: Finally y try with resources

* que existe un bloque finally, útil para cerrar recursos (como conexión);
* cuando hay un bloque finally, el bloque de catch es opcional;
* que el bloque \* finally \* se ejecuta siempre, sin o con excepción;
* cómo utilizar try-with-resources.

|  |
| --- |
|  |

## Anexos:

* Extendiendo a RuntimeException puede ser que se genere ese error, pero a nivel de Exception si genera ese error.

Repasando aquí. Throwable en la cabeza tiene dos grandes grupos de errores: errores y excepciones. Los errores son lanzados por los programadores que mantienen la máquina virtual de Java, la JVM, y las excepciones son lanzadas por nosotros que programamos sobre la JVM, la máquina virtual de Java. Dentro de este pequeño grupo de excepciones tenemos dos subgrupos: excepciones chequed y unchequed.

Las unchecked son las que extienden de RuntimeException, y por lo tanto las que no son verificadas por el compilador. Contrario a checked, que extienden de Exception y sí son verificadas en el momento pues de hacer la compilación. Entonces en resumen lo único que diferencia es al momento de la compilación, si tú quieres que el compilador te ayude a decidir cómo tratar la excepción y si tú quieres atrapar el error porque tú crees que pueda dar ese error.

* Try no puede estar solo, debe venir siempre acompañado con yn catch o un finally. Se puede acmpañar de varios catch, pero solo de 1 finally. El orden siempre debe ser try-catch-finally, si no se respeta deja de compilar.

## Código:

* Trow: palabra resevada que quiere decir lanza

## Vocabulario:

* Pila de ejecución: Stack o pila-🡪ciclo de ejecución método java

|  |
| --- |
| Interfaz de usuario gráfica, Texto, Aplicación  Descripción generada automáticamente |

* Punto de quiebre
* Debug: El debug es la ejecución de nuestro sistema, pero donde él encuentra este breakpoint él va a parar y nos va a mostrar todo lo que hay en la memoria, en ese momento. Vamos a verlo aquí en práctica. Para eso, para el debug, nosotros tenemos que ir al botón de acá, que es este bichito, debug flujo. Este bichito está al lado de play, del botón de play.
* Step int: entra al pedazo de codigo
* Step over: Pasa a la siguiente línea
* Error
* Las excepciones también son objetos
* Stacktrace: traza de la pila
* Try y catch: intenta y atrapa
* Serializar: cuando quiero mandar números por net
* Error: no controlable.
* JVM: errores retornados por los que programan la JVM en sí,
* Reglas de negocio
* Aplicaciones distribuidas: es un conjunto de aplicaciones que interactúan entre sí para conseguir satisfacer las necesidades de negocio de una misma empresa, por ejemplo. Para tomar un ejemplo de la vida real y sea más fácil de entender este concepto, vamos a hablar de Facebook.
* Codigo espagueti
* Recursos