**VIDEO 143**,**-- Excepciones comprobadas y declaracion ‘throws’**, Lo errores que heredan de la cl.

**VIDEO 144**,--clausula ‘**throws**’.  **SOLUCION A ERRORES QUE NO SABEMOS COMO SOLUCIONAR**. Metodos que lanzan excepciones.

**VIDEO 146,---Lanzar excepiones de forma manual; con la clausula ‘trow’ sin ‘s’.**

**VIDEO 147, ---CREAMOS NUESTRAS PROPIAS EXCEPCIONES.**

**VIDEO 148,** -***--CAPTURAR VARIAS EXCEPCIONES.***

**VIDEO 149,** -***--CLÁUSULA FINALLY.***

**VIDEO 142**,

A la hora de programar se pueden cometer errores. Esta es la gerarquia de las cl, que gestionan los errores en java.

Hay dos tipos de errores;

1º los que se dan en ‘Tiempo de compilación’ son errores de sintaxis -----¡¡¡ APRENDE A ESCRIBIR.

2º en ‘Tiempo de ejecucion’ de los que heredan varios tipos de errores.

Estudiamos a partir de los errores que heredan de la cl, Exception,

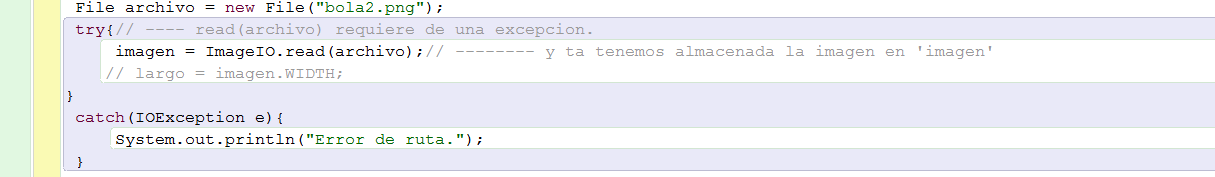


**TENEMOS**; la cl, ‘IOException’ y la cl ‘RuntimeException’

Lo errores que heredan de la cl ‘**IOException**’ ***no son errores del programador***. (ej, la imag de una carpeta…??)

Si heredan de la cl ‘**RuntimeException**’ son ***errores del programador*** (ej, recorrer 7 posiciones si solo hay 6).

Lo errores cl ‘***IOException’ java prevee que puedan suceder, (ej, la imag de una carpeta…?? Programamos para que aparecca una imagen de una carpeta, java prevee que alguien quite la imagen de esa carpeta, y para que que el programa no caiga, cuando se esté ejecutando, nos obliga a poner un código*** ‘ **try, catch**’)



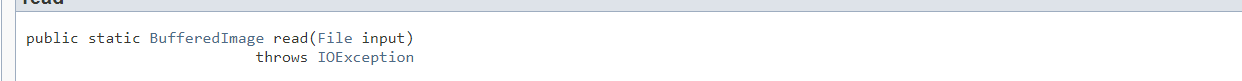
***Si la no puedes mostrar la imagen, ‘try’ muestra este mensaje; ‘catch’.***

**VIDEO 143**,

**-- Excepciones comprobadas y declaracion ‘throws’**, Lo errores que heredan de la cl ‘IOException’



En la api de java pueden verse mt acompañados de la clausula ‘**throws**’ esta quiere decir que el mt lanza una exception por lo que obligatoriamente tenemos que codifiar un **try catch**,.



1 – si todo va bien, se ejecuta el código,

2 – si se produce algún fallo, el programa va a lanzar 🡪’throws’ un objeto de tipo ‘**IOException**’ lo que nos obliga a codificar un ‘**try catch**’ por ser un error comprobado por java, para que el programa no caiga.

**VIDEO 144**,

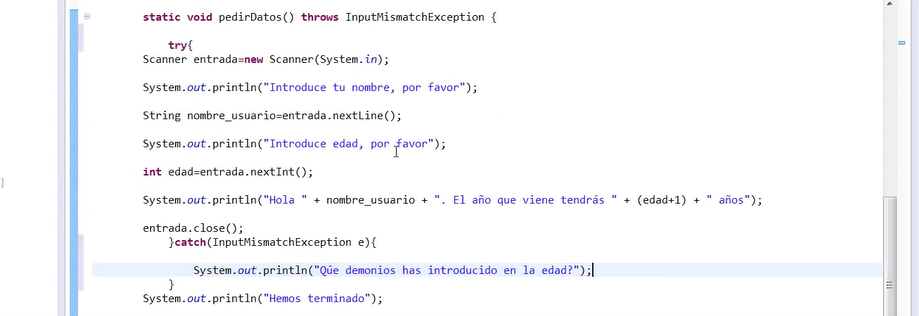
--clausula ‘**throws**’.  **SOLUCION A ERRORES QUE NO SABEMOS COMO SOLUCIONAR**. Metodos que lanzan excepciones.

Puede darse el caso de que apareccan errores, no comprobados! Es decir tenemos un error que ha cometido el programador. En estos casos no es ***obligado ni recomendable*** utilizar un ‘**try catch**’ pero para salir del apuro lo utilzamos, ¿Cómo poder utilizarlo?

Con la la clausula ‘**throws**’ en la cabecera del mt, acompañada del tipo de error que lanza.

En el ej; Cuando ponemos la edad solicitada y lo hacemos con texto en vez de con nº, el programa lanza un error de tipo InputMismatchException, y el programa se para.

Poniendo la clausula ‘**throws**’, en la cabecera del mt, seguida del tipo de excepcion que origina podemos aplicar un ‘**try catch**’, Try para que ejecute el mt, si todo va bien, y catch para que ejecute un mensaje si falla, de esta manera el programa sigue ejecutandose.

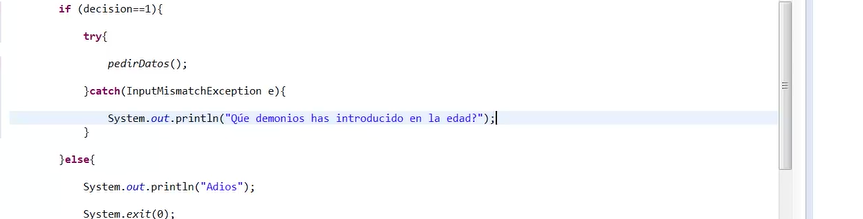


**VIDEO 145,**

**DATO**: objeto de tipo ‘**IOException**’ es para excepciones con ficheros externos.

PARECE POCO LÓGICO PONER LA LLAMADA A LA ESCEPCIÓN DENTRO DEL MT, POR LO QUE AHORA LO PONEMOS EN EL CONSTRUCTOR DONDE ES LLAMADO EL MT.

Crea la estructura fuera del mt, lo crea en el constructor, donde hace la llamada al mt,



**VIDEO 146,**

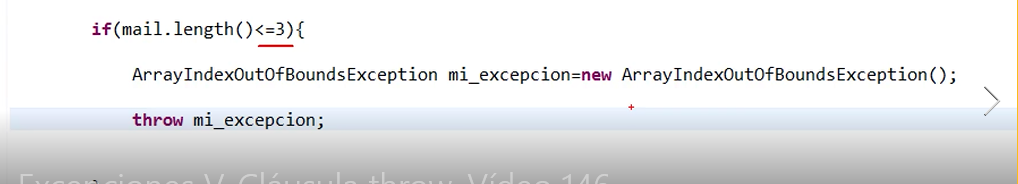
---Lanzar excepiones de forma manual; con la clausula ‘**trow**’ sin ‘s’.

Esta escepción se pude utilizar en cualquier punto del código, mientras que ‘**trows**’ solo se puede utilizar en la cabecera de los mt.

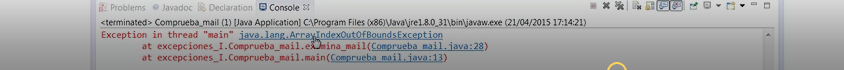
En el siguiente ejemplo, ***vemos como lanzar un error una excepción de forma manual***, en una dirección de correo incorrecta, no es la forma adecuada de utilizar esta clausula, pero sirve de forma ilustrativa para conocer el funcionamiento de la misma.

--- El 1º problema que se nos presenta es saber que escepción corresponde con el error de que la dirección de correo sea <= de 3; al no saberlo ponemos una cualquiera.

En este caso creamos una instancia de la cl, ArrayIndexOutOfBoundsException, y si la dirección de correo es <= de 3 caracteres lanza la excepción por tener clausula ‘**trow**’ sin ‘s’.



Y el programa caería,



PERO HABRÍAMOS LANZADO UNA EXCEPCIÓN DE FORMA MANUAL. ¡¡¡¡¡

**DATO**; la instancia se suele hacer de forma directa;

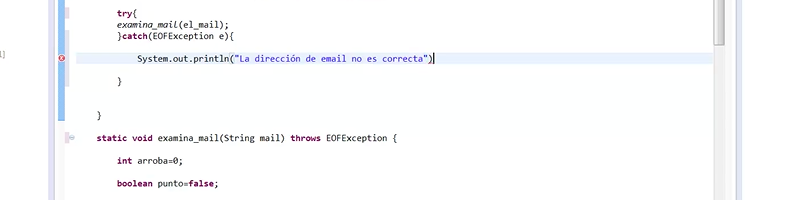
--- **throw = new ArrayIndexOutOfBoundsException();**

--- El 2º, decirle al programa que hacer cuando capture la excepción,

y otro dato importante es que si un mt, captura una excepción, tenemos que indicarlo en la cabecera del mt, con la clausula ‘**trows**’ seguido del tipo de excepción que lanzará.



**Recordemos**; que hemos puesto una excepción a boleo, ahora decidimos cambiarla por un tipo de ‘**excepción comprobada**’, para que el programa nos obligue a codificar el ‘try catch’, si todo va bien ‘try’ si no se ejecuta el código que tenemos en el ‘ catch’.



Dato,



**RECORDEMOS¡¡¡.** ES UN EJEMPLO ILUSTRATIVO, UTILIZAR UN EXCEPCIÓN DE ESTA MANERA ES UNA MALA PRÁCTICA.

**VIDEO 147,**

---CREAMOS NUESTRAS PROPIAS EXCEPCIONES.

\* ----------- CREAMOS LA CL PARA CREAR NUESTRA PROPIA EXCEPCION ----------esta

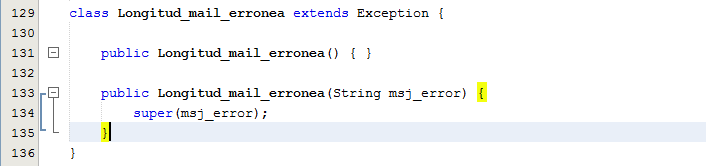
\* cl ***se recomienda tener dos constructores***, uno que pertencecca a la cl de la

\* que hereda -- y otro más que reciba por parámetro un String

\*

\* ES RECOMENDABLE QUE LA CL HEREDE DE LA CL Exception, para que obligue a poner el ‘**try, catch’**

\*



\* una vez creados los const, nos vamos al mt donde se crea el email y le

\* ponemos como lanzador de una excepcion con la clausula '**throws**' -->

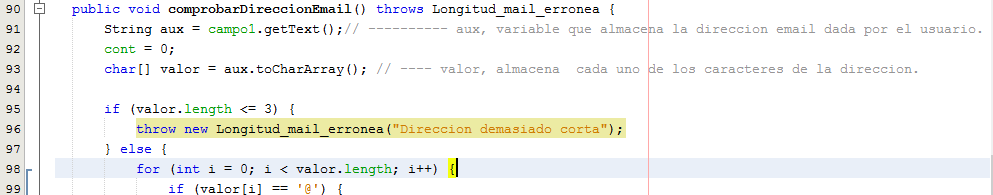
\* --> comprobarDireccionEmail() throws Longitud\_mail\_erronea \*



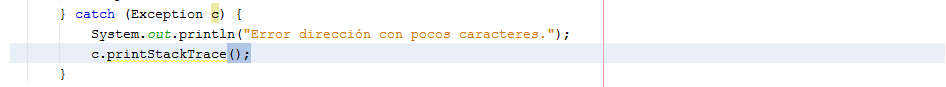
\*

\* dentro del condigo de este mt, hacemos una llamada a la cl, de nuestraExcepcion con la clausula ‘**throw**’ ¡¡¡ ***RECORDEMOS QUE ESTA CLAUSULA ES PARA LANZAR UNA EXCEPCION DE FORMA MANUAL EN EL PUNTO DEL CÓDIGO QUE QUERAMOS,***!!!

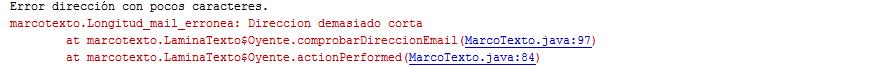
y la colocamos donde el mt haga saltar al programa donde no se cumpla una condicion.



EXISTE EN JAVA UNA INSTRUCCIÓN PARA INFORMARNOS DE LA PILA DE ERRORES; ‘**c.printStackTrace();**’



Sirve para informar de los herrores que se han producido. “**Dirección demasiado corta**”

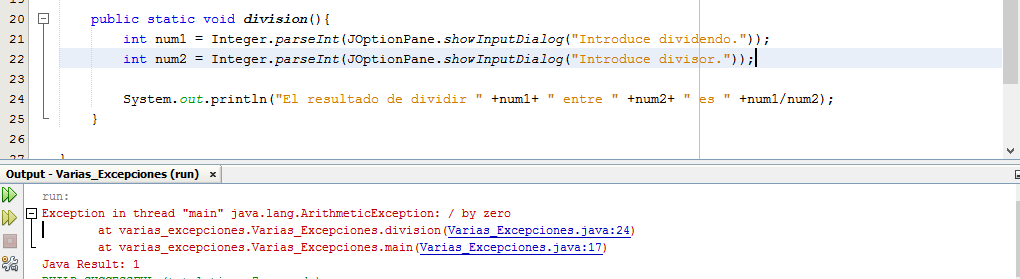


A un programador le viene bien tener esta información y este es el motivo por el que es recomendable crear estas cl con dos constructores, en el que ponemos un parametro String, la añadimos esta información.

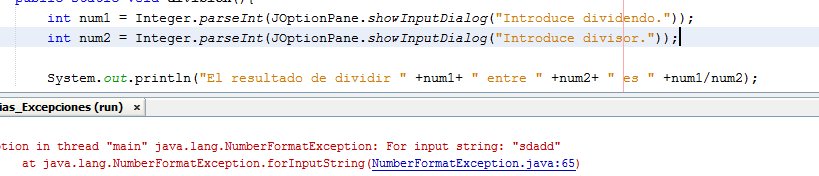
**VIDEO 148,** -***--CAPTURAR VARIAS EXCEPCIONES.***

Sucede cuando un mt, pude lanzar varios tipos de excepciones, todas ellas diferentes,

**En el ej; vemos la excepción que mandaría si hacemos la división entre ‘0’**. **AritmeticException**,

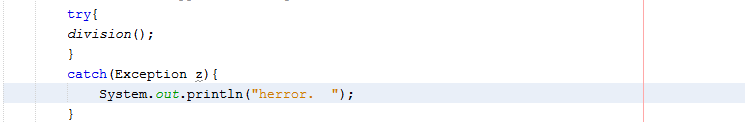


**O si ponemos texto en la división, manda la excepción’**. **NumberFormatException**



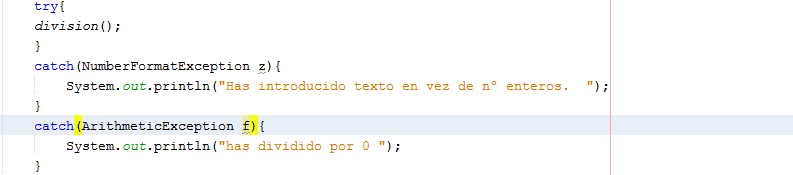
AMBAS SON EXCEPCIONES ‘NO CONTROLADAS’ Y NO ES RECOMENDABLE UTILIZAR EL CODIGO ‘try catch

Aún así podemos codificalo y evitar que el programa se pare.



Pero esto a parte de ser una mala práctica es poco útil, mejor es que el programa lance un mensaje por cada excepción que en el se produce.

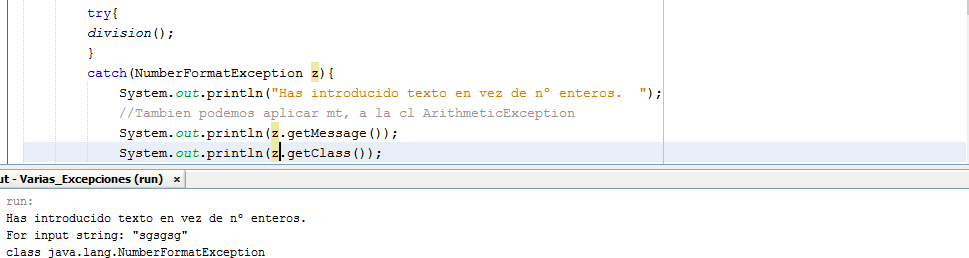
Para ello colocamos un catch para cada una de las excepciones que nos ha dado el programa.



.

/***/Tambien podemos aplicar mt, de la cl NunberFormatException para obtener datos de la excepción***

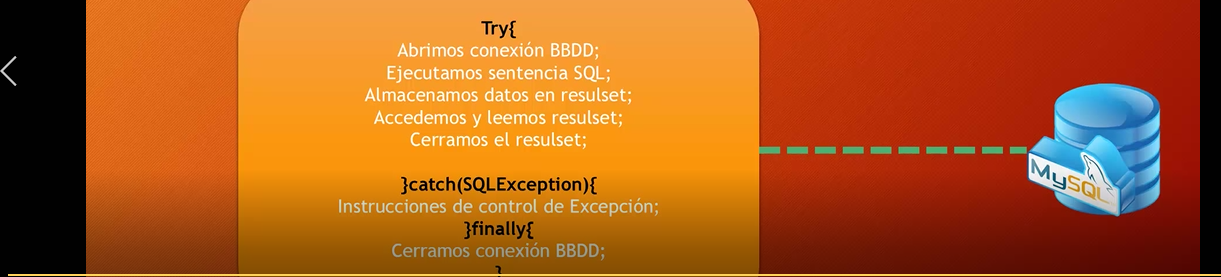
.



**VIDEO 149,** -***--CLÁUSULA FINALLY.***

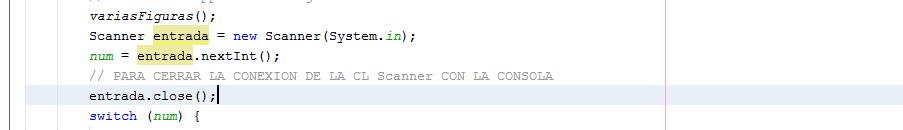
Principalmente sirve para cerrar los recursos de una BBDD.

Si estamos trabajando con una BBDD tenemos que cerrar el acceso a ella cuando hemos terminado la consulta, ya que si no estaríamos ocupando una entrada a la BBDD que no estamos utilzando y que otro usuario puede que necesite.



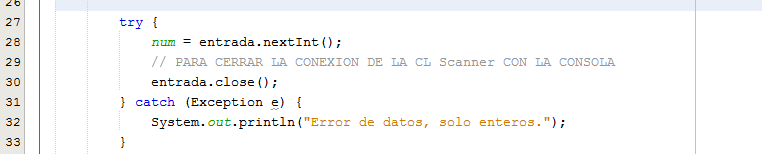
**Ejemplo ilustrativo**.

**DATO**: Ocurre que en algunas ocasiones, como es el caso de la cl, Scanner, abrimos una conexión externa con la consola para introducir datos en el programa, y esa conexión no es cerrada nunca. Esto hace que se esten gastando recursos de la máquina donde se esté ejecutando ese programa. Es recomendable cerrarla con el mt ‘close()’



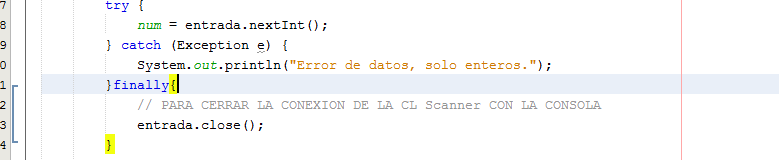
**ADVERSIDADES**:

En este proyecto del ejemplo, salta una excepción si ponemos datos de tipo String a la hora de seleccionar la figura que queremos, y el programa cae, por lo que montamos un ‘tri catch’



**CLAUSULA FINALLY**. 🡪Siempre se ejecuta.

Observemos en la imagen de arriba que si todo va bien el try se ejecuta, sino se ejecuta el catch, y la conexión quedaría abierta porque el mt ‘close()’ esta en try.



**El codigo que tengamos en FINALLY**. 🡪Siempre se ejecuta luego.