**Introducción a Excepciones**

**Que es una excepción?**

Una excepción es un evento que ocurre durante la ejecución de un programa, que interrumpe el flujo normal de ejecución. Java utiliza excepciones para el manejo de errores, error handling.

Cuando se da un error (desbordamiento de memoria, disco duro estropeado, división por cero, acceder a un vector fuera de sus limites) la JVM crea un objeto de la clase Exception.

No necesariamente es un error o un problema en el software. Podría ser el caso de un dato repetido.

Ejecutamos el siguiente codigo:

public static void main(String[] args){

int num = 10 / 0; // intento division por cero

}

Java lanzo una excepción porque no puedo resolver la división por cero.

Vemos el detalle de la excepción que aparece en la consola de salida al ejecutar el código

**Bloques try / catch / finally**

**Sintaxis**

public static void main(String[] args){

try {

// codigo . . .

} catch(Exception e){

// codigo . . .

} finally {

// codigo . . .

}

}

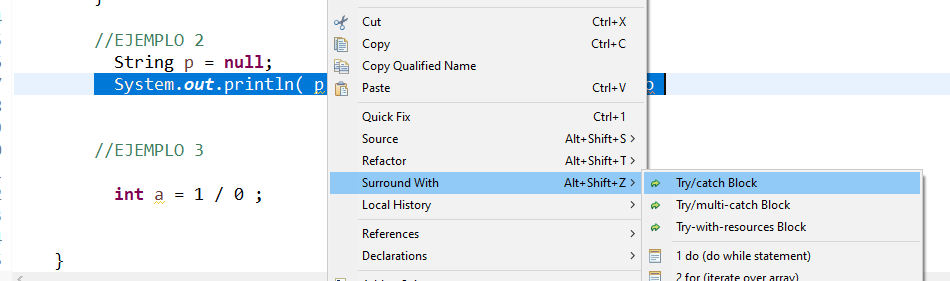
**Codigo Netbeans:**



**Codigo Eclipse:**

****

Crear automáticamente **try-catch** desde Eclipse:



Desde Netbeans:

Marcar en bloque el codigo y oprimir ALT + ENTER => Surround with try-catch

**Bloques**

***Bloque try:*** aquí va el **código de proceso** si todo va bien, no se lanzara ninguna excepción.

Si hay algún inconveniente, se lanzara una excepción. la cual será capturada en el bloque catch.

***Bloque catch***: aquí va el **código de tratamiento de error**. Donde se ubican las **medidas correctivas**, como por ejemplo levantar una caja de dialogo con el error o escribir algún archivo de log con el error.

Este bloque **es obligatorio si se pone el bloque try y no se pone el finally**. Si se pone el finally el bloque catch es opcional.

***Bloque finally(opcional si se pone CATCH)***: aquí va el código que se ejecuta tanto si se lanza una excepción como si no se lanza.

Su función mas importante es **cerrar** todo tipo de **conexiones**. Ej: conexiones a la base, cerrar un archivo, conexiones a redes, streams etc. Esto sucede porque el error puede darse antes de cerrar conexiones.

**Exception:** Es la clase a partir de la cual se crean los objetos e.

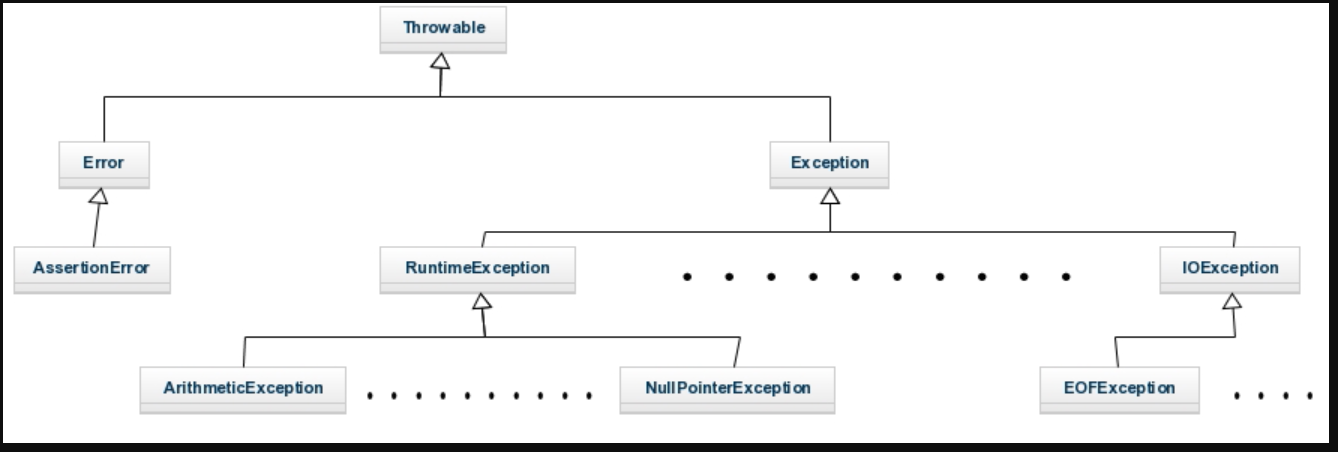
Exception e = new Exception();

Para definir una excepción será necesario con extender la clase Exception (herencia) y proporcionar la funcionalidad extra que requiera el tratamiento de esa excepción.

***Objeto “e”***:

El objeto generado “excepción” contiene información sobre el acontecimiento ocurrido y transmite esta información al método desde el que se ha generado la excepción.

**Diagrama** de la clase Trowable (clase padre de todas las clases que manejan las excepciones)

******

**Multiples catch**

Si se quiere discriminar el error, se debe hacer un catch por cada error

*Codificamos y probamos el siguiente codigo (Probamos con la división a cero y luego con la referencia a nulo, comentamos lo que no queremos probar).*

**Codigo Eclipse:**

****

package clase5.excepciones;

public class Clase5Excepciones {

public static void main(String[] args) {

try { System.out.println("try . . .");

int num = 10 / 0; // intento division por cero

String p = null;

System.out.println( p.length() ); // referencia a nulo

System.out.println("FIN try . . .");

} catch(ArithmeticException e){

System.out.println("Hubo un error: Division por cero");

} catch(NullPointerException e){

System.out.println("Hubo un error: Puntero a nulo");

} catch(Exception e){

System.out.println("Hubo un error");

} finally {

System.out.println("finally . . .");

}

}

}

Entonces lo que ocurre es que si se da alguna excepción manejara primero esa excepción y ya no manejara las otras. No evaluara los catch siguientes.

Se estila poner un bloque catch a lo último que capture la excepcion “Exception”, que es la excepción genérica y representa a cualquier excepción.

Si se pone antes no compila (probar poner el catch de Exception delante de otro catch y ver que no compila).

**Checked Exceptions vs. Unchecked Exceptions**

Hay dos tipos de categorías de excepciones en JAVA que tienen distinto comportamiento.

Unchecked Exception => Excepciones que el programador no chequeó o no detectadas al momento de escribir el código y compilar. Errores en el codigo.

NullPointerException

IndexOutOfBoundsException

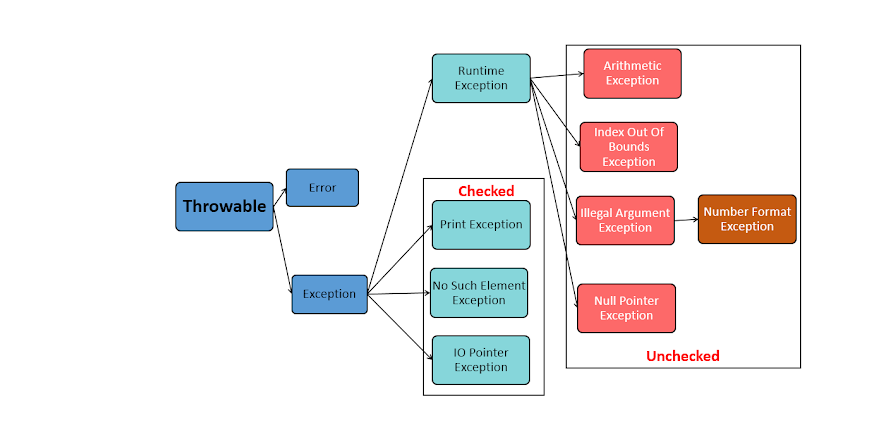
Checked Exception => Excepciones previstas. Cuestiones conocidas por el negocio.

**Actualizacion Checked vs Unchecked**

El lenguaje Java utiliza un [mecanismo de excepciones](https://picodotdev.github.io/blog-bitix/2017/10/las-excepciones-para-gestionar-errores-con-java/), las excepciones son objetos que se lanzan cuando se produce una condición de error. Todas las excepciones en Java heredan de [Throwable](https://docs.oracle.com/javase/10/docs/api/java/lang/Throwable.html) subdividiéndose en [Error](https://docs.oracle.com/javase/10/docs/api/java/lang/Error.html) y [Exception](https://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html), las primeras son condiciones de error del sistema y las segundas condiciones de error del programa. A su vez las Exception pueden ser **checked** si heredan de esta y son aquellas que el compilador fuerza a que sean capturadas no pudiendo ignorarse, han de capturarse en una construcción try catch o declarar que el método puede lanzar la excepción no capturada. Las excepciones **uncheked** heredan de [RuntimeException](https://docs.oracle.com/javase/10/docs/api/java/lang/RuntimeException.html) que heredan a su vez de Exception pero tienen la particularidad de que no es necesario capturarlas ni declararlas como que se pueden lanzar debido a que se consideran condiciones de error en la programación como un acceso a un array fuera de rango que produce un [ArrayIndexOutOfBounds](https://docs.oracle.com/javase/10/docs/api/java/lang/ArrayIndexOutOfBoundsException.html), el conocido [NullPointerException](https://docs.oracle.com/javase/10/docs/api/java/lang/NullPointerException.html) cuando se utiliza una referencia nula, otro es [ArithmeticException](https://docs.oracle.com/javase/10/docs/api/java/lang/ArithmeticException.html) que se produce al dividir por 0.

Algunas ventajas de las excepciones son:

* Separar el código que gestiona los errores del código con el caso principal del programa.
* Propagar errores hacia arriba en la pila de llamadas.
* Agrupar y diferenciar entre diferentes tipos de errores.



**Checked exception:**

Hereda de la clase Exception. Debe ser capturada, sino el código no compila. Sí voy a tener la obligación de manejarlas. Son detectadas desde el Compiler.

Ejemplos de checked exceptions son IOException y SQLException, que serán presentadas en los próximos temas del curso

Cuando creo una Excepcion por convención su nombre debería terminar con la palabra

**Exception (a nivel nomenclatura).** Creo una clase que herede de **java.lang.Exception.**

**Codigo Eclipse:**

****

**Unchecked exception**:

Hereda de la clase **RuntimeException**. Son eventos excepcionales que no estoy obligados a tratarlos. Ocurren en tiempo de ejecución. No son detectadas cuando compilamos el programa.

Unchecked Exception

-- NullPointerException

-- ArrayIndexOutofBound

-- IllegalArgument Exception

-- ClassCastException

-- IllegalStateException

-- ConcurrentModificationException

Checked Exception

-- FileNotFoundException

-- ParseException

-- ClassNotFoundException

-- CloneNotSupportException

-- SQLException

**Codigo Eclipse:**



**Reutilizacion de Excepciones:**

Con la reutilización de excepciones creamos una excepción que puede recibir por parámetro por ejemplo un string el cual cambie de manera dinámica. Es decir cuando se llame a la excepcion.

**Codigo Netbeans:**



**Codigo Eclipse:**

****

**Manejo de capas con excepciones**

Como manejamos las excepciones cuando la excepción es llamada por un atributo objeto de otro objeto. Supongamos que en base al ejercicio previo creamos una clase llamada **Negocio**.

**Codigo Netbeans:**

****

**Excepciones construidas por el usuario(negocio)/programador**: Es muy común en Java modelar errores de negocio a través de la confección de excepciones

Ahora armamos un paquete llamado “Excepciones” dentro de Clase5 – Excepciones. Y creamos la clase “NoHayMasPasajesException”

**Codificar el siguiente codigo** (codigo Clase5 – Excepciones.Excepciones.NoHayMasPasajes):

package Excepciones;

public class NoHayMasPasajesException extends Exception {

private int cantidadDePasajes;

private String vuelo;

public NoHayMasPasajesException(){}

public NoHayMasPasajesException(int c, String v) {

setCantidadDePasajes(c);

setVuelo(v);

}

public int getCantidadDePasajes() {

return cantidadDePasajes;

}

public void setCantidadDePasajes(int cantidadDePasajes) {

this.cantidadDePasajes = cantidadDePasajes;

}

public String getVuelo() {

return vuelo;

}

public void setVuelo(String vuelo) {

this.vuelo = vuelo;

}

}

Esta clase es una excepción armada por nosotros. La lanzaremos en forma manual cuando no se puede realizar la venta de un pasaje porque no hay mas lugar en el vuelo.

No conviene hacerla heredar de **RuntimeException** porque las RuntimeExceptions pueden no ser capturadas y esta al ser una excepción de negocio debe ser capturada para tomar las medidas correctivas.

**Keywords “throws” y “throw”**

**throws**

Se utiliza para declarar que un método puede lanzar una excepción.

**throw**

Se utiliza para lanzar de forma manual una excepción. Se envia una instancia de una clase que hereda de Exception.

Armamos el siguiente codigo dentro del paquete “Excepciones”. Creamos la clase “Vuelo”:

package Excepciones;

public class Vuelo {

private String nombre;

private int lugaresDisponibles;

//el siguiente metodo controla que haya disponibilidad

//si no hay disponibilidad, lanza una excepcion.

public void venderPasajes(int c) throws NoHayMasPasajesException{

if(lugaresDisponibles < c){

throw new NoHayMasPasajesException(c, nombre);

} else{

lugaresDisponibles -=c; //descuenta lugares disponibles.

}

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public int getLugaresDisponibles() {

return lugaresDisponibles;

}

public void setLugaresDisponibles(int lugaresDisponibles) {

this.lugaresDisponibles = lugaresDisponibles;

}

}

Un método puede lanzar varias excepciones, solo hay que poner el nombre de cada una luego del throws. Por ejemplo

public void venderPasajes(int c) throws NoHayMasPasajesException, CancelacionExcepction {

}

**Armamos el siguiente codigo** (dentro del mismo paquete Excepciones, clase Programa):

package Excepciones;

public class Programa {

public static void main(String[] args) {

//Instancia vuelos

Vuelo v1 = new Vuelo("SXR-1214", 100);

Vuelo v2 = new Vuelo("ABC-4321", 100);

try {

// Vender pasajes

v1.venderPasajes(50);

v2.venderPasajes(25);

v1.venderPasajes(20);

v2.venderPasajes(50);

v1.venderPasajes(50); // esta linea lanza excepcion

v2.venderPasajes(10);

} catch (NoHayMasPasajesException e) {

System.out.println("El vuelo " + e.getVuelo() + " no tiene disponibles " + e.getCantidadDePasajes() + " pasajes.");

}

}

}

**TRY-WITH-RESOURCES**

Partimos de la siguiente lectura

****

**Codigo Eclipse:  
**

**Ejercicio Integrador Excepciones**



Codigo **Clase 5 – ProyectoIntegrador.**

****