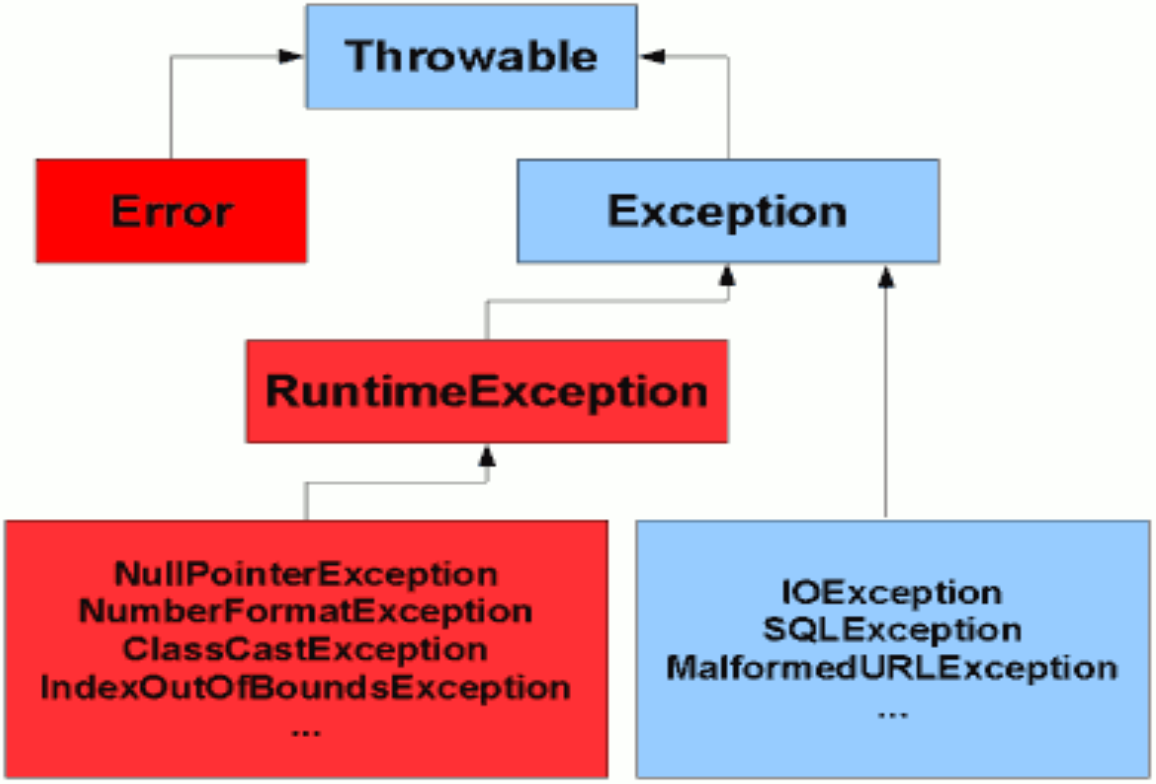
06 Excepciones

EXCEPCIONES

- En java cuando se produce un error en un método se “lanza” un error de tipo **Throwable**.
- Cualquier método que halla llamado al método que lanza el error, podrá “***capturar***” la excepción y tomar las medidas necesarias.
- Una vez lanzada la excepción el programa se interrumpe.
- La clase **Throwable** tiene las siguientes características:
  - Contiene una instancia del estado de la pila en el momento en el que se creo el objeto.
  - Almacena un mensaje que permite detallar el error que se produjo.
  - Puede contener la causa del error que provoco el actual error y también es de tipo **Throwable.**

EXCEPCIONES



EXCEPCIONES

Error: Es una subclase de **Throwable** que indica problemas graves que una aplicación no debería intentar resolver: "*Memoria Agotada*" o "*Error interno de la JVM*".

Exception: Clase que indican situaciones que una aplicación debería tratar de forma razonable. Los dos tipos principales son:

- **RuntimeException**: Se usan en caso de errores del programador del tipo acceso a una posición de array no correcta o división por 0 de un numero.
- **IOException**: Se usan en caso de errores que no pueden ser evitados por el programador y van muchas veces relacionados

EXCEPCIONES

Manejo de Excepciones

El manejo de errores en Java se realiza mediante *Excepciones:*

en algún punto del programa ocurre un error, y no sabemos exactamente qué hacer con él. paramos la ejecución y *enviamos el problema* a otro sitio, al que nos invocó, para que intente tratar el error.

A su vez, este objeto que nos invocó puede hacer lo mismo, o tratar el error, o *enviar el problema* al que le invocó a él. En una última instancia está la JVM.

2 Ventajas:

- Código limpio de tratamiento de errores.
- Se garantiza que “alguien” tratará el error.

EXCEPCIONES

Objeto Exception

Cuando lanzamos una excepción (es decir, cuando delegamos el problema al código que nos ha invocado), se crea un objeto que contiene toda la información de lo que ha ocurrido.

Con la referencia a este objeto, se pone en marcha el mecanismo de gestión de excepciones, que intenta localizar el trozo de código que va a tratar el error.

A este trozo de código se le llama *exception handler*.

EXCEPCIONES

LANZANDO EXCEPCIONES

Imaginemos que nuestro método recibe como argumento una referencia a un objeto (***t***).

Cuando ***t*** nos llega *null*, no sabemos qué hacer!!

Así que decidimos abdicar de esa responsabilidad de tratar el error lanzando una excepción:

if(t == null) ***throw new NullPointerException();***

Cuando se ejecuta un *throw*, se crea un objeto que representa la condición del error.

Es muy importante el nombre de la clase retornada (en el ejemplo *NullPointerException*)

que nos dice el tipo del error.

everis
an NTT DATA Company

EXCEPCIONES

LANZANDO EXCEPCIONES

Los objetos Exception se crean como todos los objetos en Java, utilizando **new**.

Además del constructor por defecto, se le puede pasar una cadena de caracteres con información sobre la excepción.

```
throw new NullPointerException("t = null");
```

EXCEPCIONES

Exception handler

Después de lanzar la excepción, el sistema intenta encontrar “alguien” (*exception* handler) que la trate, siguiendo por la lista de invocaciones de métodos que se realizaron hasta el método que lanzó la excepción, en orden inverso. Si no se encuentra ninguno, se llega hasta la JVM, que trata el error y el programa termina.

Los *exception handlers* pueden estar especializados en tratar tipos de excepciones, por tanto, lo que se intenta localizar en realidad es un manipulador de excepción que trate el tipo de excepción que se ha lanzado.

EXCEPCIONES

TIPOS DE EXCEPCIONES

Excepciones *checked*: son aquellas a las que nos podemos anticipar y de las que nos podemos recuperar.

Excepciones *Error:* condiciones excepcionales externas a la aplicación y que no podemos preveer, como que se produzca un fallo en el sistema de ficheros al intentar abrir un fichero. Son indicadas por la clase *Error*.

Excepciones *Runtime Exceptions*: son excepciones que tampoco podemos preveer, suelen indicar un *bug* en la aplicación. Es decir, no intervienen factores externos a la aplicación como en el caso anterior. Son indicadas por la clase *RuntimeException*.

EXCEPCIONES

TRATAMIENTO DE EXCEPCIONES

En Java, cada método debe informar al cliente de las excepciones *checked* que el mismo puede lanzar durante su ejecución. De este modo los invocadores de este método, saben a qué atenerse y podrán capturar las excepciones adecuadamente o simplemente pasarlas al método que a su vez les invocó.

Esto se realiza en la declaración del método:

void dividir() throws NullPointerException,ArithmeticException { //...

Si en vez de esto, la declaración del método es ***void sumar()***, significa que el método no genera excepciones, a no ser que sean *RuntimeException,* que por su naturaleza no son previsibles.

En tiempo de compilación, el compilador se quejará si detecta que el código fuente de tu método invoca otros métodos con *throw* en su declaración. Así que es obligatorio manipular esas excepciones dentro del método, o bien indicarlo en la declaración del método para que sean tratadas a su vez por los invocadores del mismo.

EXCEPCIONES

TRATAMIENTO DE EXCEPCIONES

Un método que invoque a métodos que pueden lanzar excepciones *checked* (tienen en su cabecera *throws*) tienen que obligatoriamente hacer una de las dos cosas siguientes:

- pasar la excepción al método que le ha invocado a él (poniendo *throws* también en su cabecera con el mismo tipo de Exception).

- implementar los bloques *try – catch* convirtiéndose así en manipuladores de la excepción.

everis
an NTT DATA Company

EXCEPCIONES

TRY CATCH

Si queremos capturar excepciones, debemos construir bloques try – catch:

```
try {
  // Código que puede generar excepciones
} catch(Type1 id1) {
 // Manipulando excepciones de Type1
} catch(Type2 id2) {
  // Manipulando excepciones de Type2
}
```

EXCEPCIONES

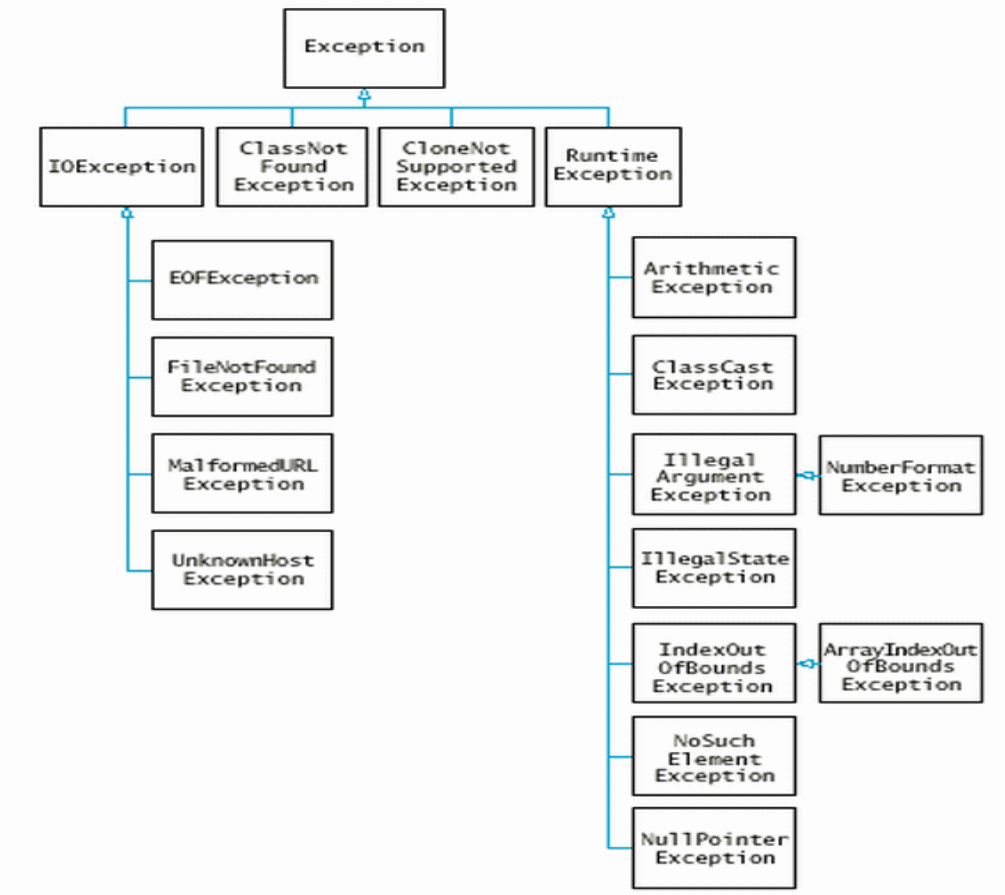
RESUMEN

- Una excepción es un evento que ocurre durante la ejecución de un método, que “rompe” el flujo normal de las instrucciones del mismo.
- Cuando ocurre el error, el método crea un objeto con información sobre ese error.
- Una vez creado el objeto, el método ejecuta un *throw exception*, delegando la manipulación de la excepción a otro método del *call stack*.
- Manejar la excepción es conocido como *“capturar la excepción”.*

everis
an NTT DATA Company

EXCEPCIONES

CLASES DE EXCEPTION



EXCEPCIONES

Métodos útiles del objeto Exception

String getMessage( ): Retorna un String que es una breve descripción de la excepción.

void printStackTrace( ): Pinta por error estandar la pila de ejecución del momento en que ocurrió la excepción.

Class **getClass( )**: Retorna un objeto que representa la clase de excepción.

EXCEPCIONES

Relanzando excepciones

Algunas veces es útil relanzar (*rethrow*) excepciones aunque ya hayan sido capturadas:

```
try {
//…..
}
catch(Exception e) {
  System.err.println("An exception was thrown");
  throw e;
// También podríamos haber lanzado otra distinta, por ejemplo: //throw new NullPointerException();
}
```

everis
an NTT DATA Company

EXCEPCIONES

Finally

Podemos querer ejecutar siempre un bloque de código fuente tanto si ha ocurrido alguna excepción como si no. Por ejemplo, en ese bloque podríamos cerrar sesiones o conexiones a bases de datos:

```
try {
  // The guarded region: Dangerous activities that might throw A, B, or C
} catch (A a1) {
  // Handler for situation A
} catch (B b1) {
  // Handler for situation B
} catch (C c1) {
  // Handler for situation C
} finally {
  // Activities that happen every time
}
```