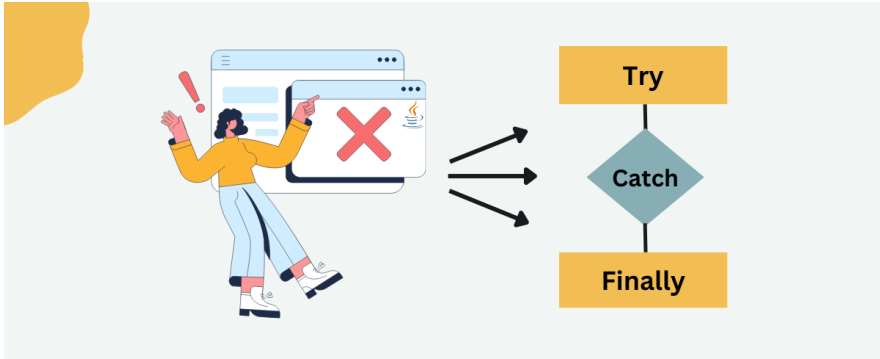
Excepciones en Java

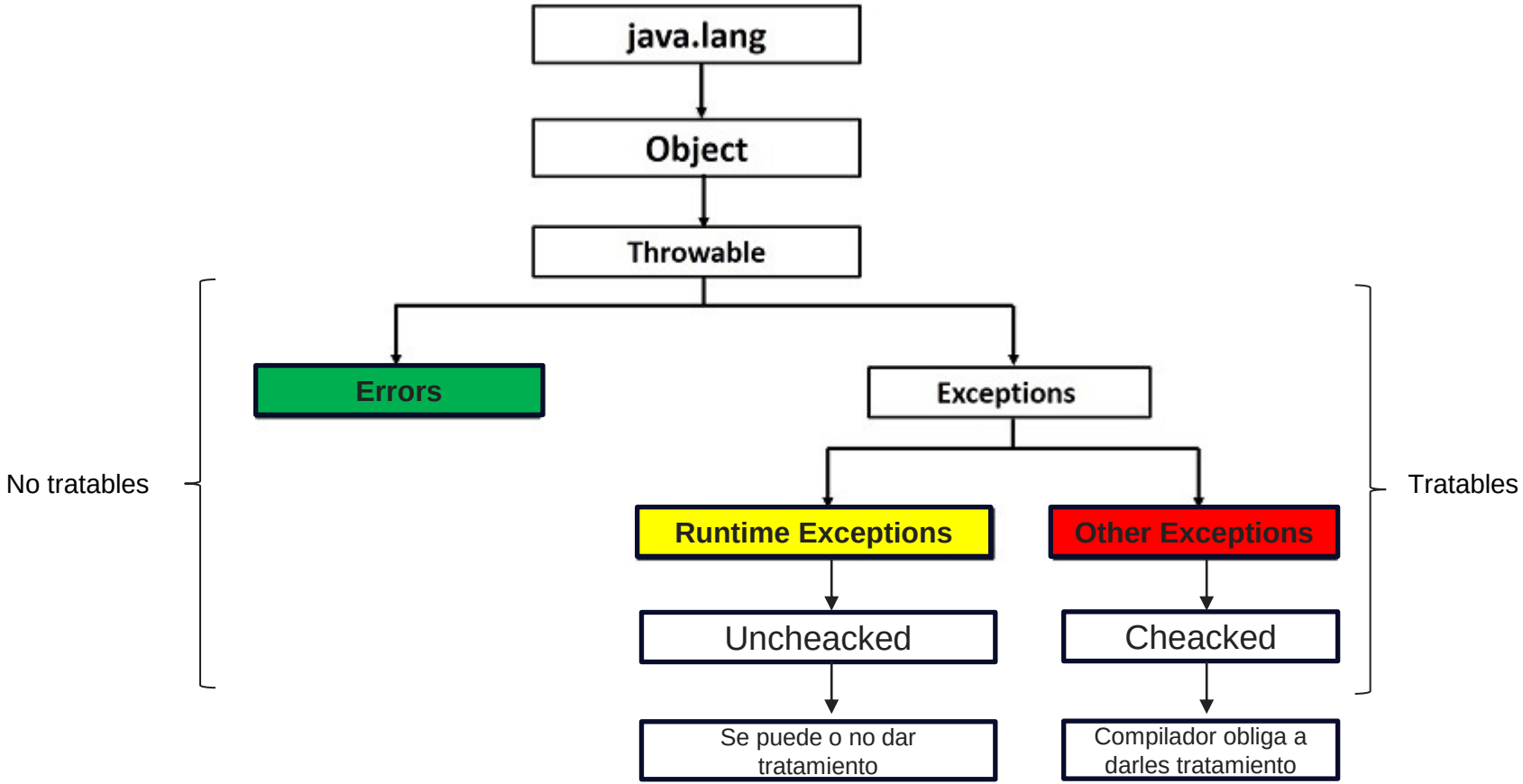
Natalia Reyes Altamirano

¿Qué es una excepción?

- Es una condición anormal ocurrida durante la ejecución de un programa. No necesariamente es un error, sino más bien un comportamiento no adecuado o no permitido.
- Este comportamiento interrumpe el flujo de ejecución de los programas y puede provocar la terminación del programa de forma inmediata.
- Java incorpora la gestión de excepciones, a través de ciertas clases especiales para controlar los errores en tiempos de ejecución

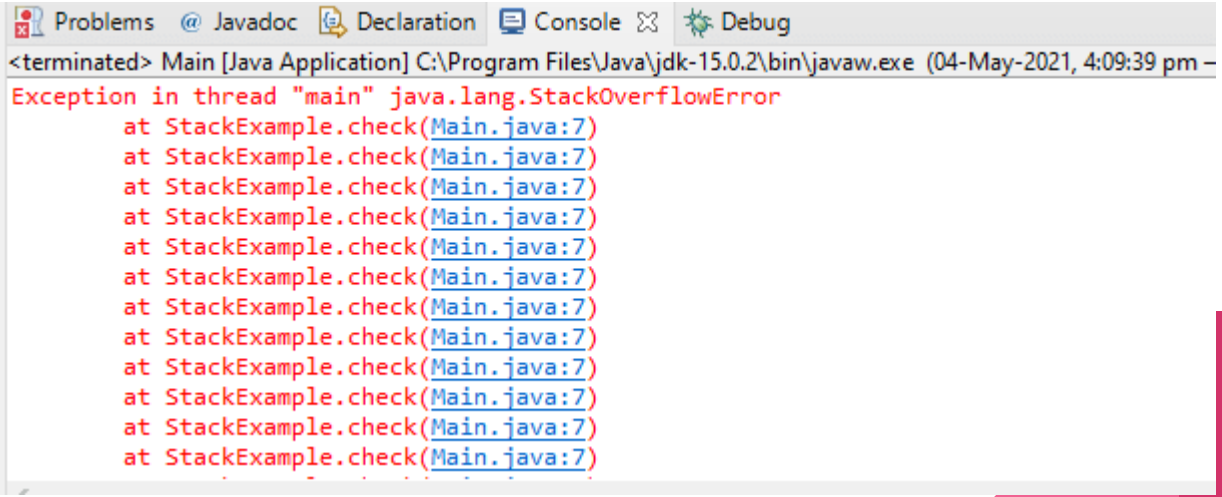


Tipos de excepciones



Errors

Cuando ocurre una falla de vinculación dinámica u otra falla grave en la máquina virtual de Java, la máquina virtual genera un archivo Error. Los programas simples normalmente no atrapan ni lanzan Error.



Manejo de excepciones

Sirve para :

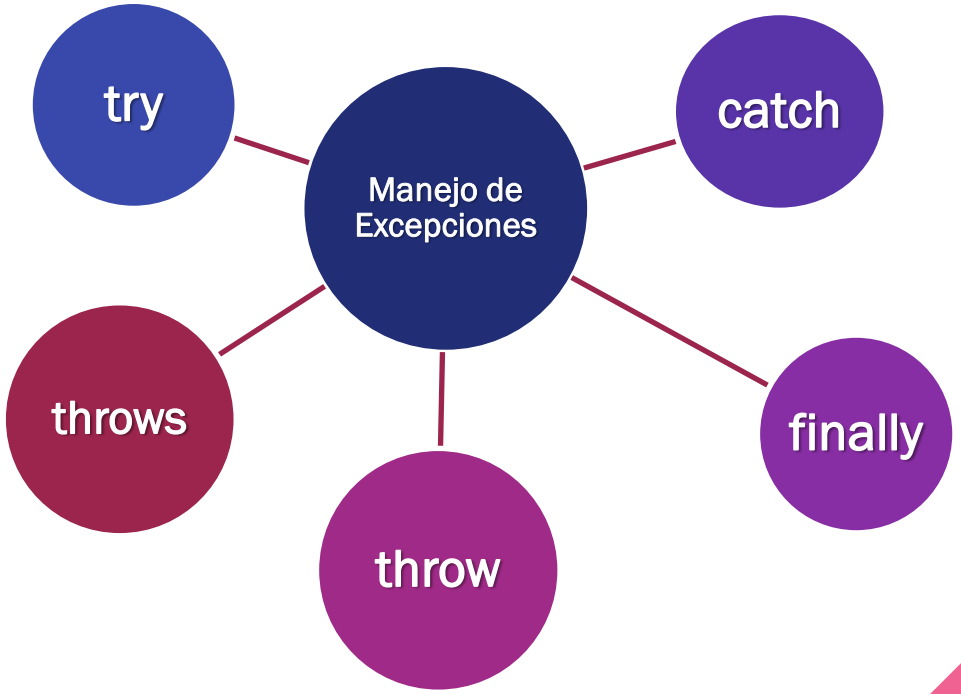
- Procesar solamente situaciones excepcionales donde un método no podría completar su tarea debido a que no posee el control.

- En proyectos grandes para manejar las excepciones de una manera uniforme en todo el proyecto.

- Simplifican los programas ya que se diferencia el código normal del código de tratamiento de errores.

- Se crean programas más robustos ya que en muchos casos si no se trata la excepción el programa no compila.

Palabras clave del manejo de excepciones



Bloque try

- Bloque de código que se quiere "intentar" ejecutar.
- Las instrucciones contenidas en un bloque try, lanzarán la excepción si ésta ocurre.

```
try{
    FileReader fichero = new FileReader("nombre del fichero");
```

```java
public static void main(String[] args) {

    String[] alumnos  = new String [6];

    alumnos[0] = "Juan";
    alumnos[1] = "Fernando";
    alumnos[2] = "Martín";
    alumnos[3] = "Marcos";
    alumnos[4] = "José";
    alumnos[5] = "Carlos";


        for (int i = 0; i<=alumnos.length;i++){

            System.out.println(alumnos[i]);
```

Al no utilizar try , al momento de correr la aplicación se va a generar una Excepción y como no la manejamos, se termina la ejecución del código

Console

```
<terminated> ViewFinal [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (20/11/2012 11:14:44)
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at Ejemplo1.main(Ejemplo1.java:22)
Juan
Fernando
Martín
Marcos
José
Carlos
```

```java
public static void main(String[] args) {

    String[] alumnos  = new String [6];

    alumnos[0] = "Juan";
    alumnos[1] = "Fernando";
    alumnos[2] = "Martín";
    alumnos[3] = "Marcos";
    alumnos[4] = "José";
    alumnos[5] = "Carlos";

    try {

            for (int i = 0; i<=alumnos.length;i++){

                System.out.println(alumnos[i]);
            }
```

De esta manera se puede controlar a la Excepción.

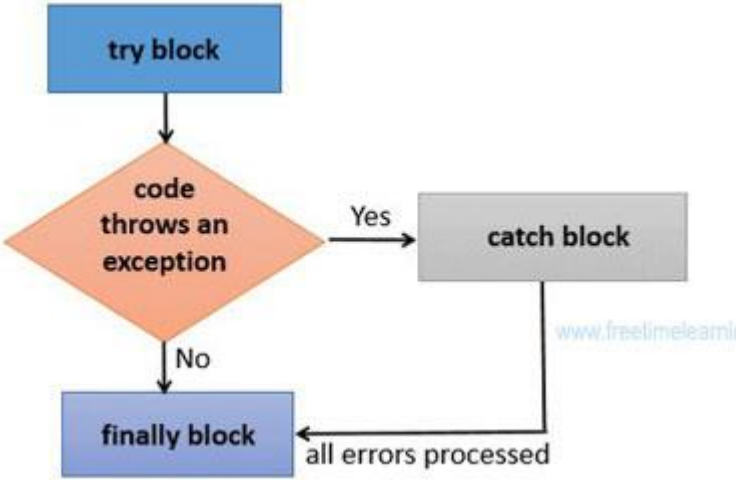
Bloque catch

- Para capturar las instrucciones a realizar cuando se detecta el error.
- Se debe escribir inmediatamente después del bloque try.
- Puede haber varios bloques catch para un bloque try, y cada bloque catch manejará un tipo de excepción.
- Es decir, la sentencia “Catch” atrapa la excepción lanzada y la maneja.

```
try{
    FileReader fichero = new FileReader("nombre del fichero");
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Bloque finally

- Bloque finally siempre se ejecuta independientemente de si se da o no una excepción.
- Es opcional.




```
Resource resource = null;
try {
    resource = new Resource();
    resource.someAction();
}
catch (Exception e) {
    System.out.println("Exception caught");
}
finally {
    resource.close();
}
```

Crear excepciones personalizadas

- Normalmente se crean nuevas excepciones cuando las clases que posee Java no contemplan el manejo de una situación anómala particular.
- Para crear las excepciones propias se hereda de la clase Exception o de una de sus subclases.

En el siguiente ejemplo el texto de la excepción se puede recuperar con el método **getMessage()** definido en la clase **Throwable**.

```
public class TemperaturaNoValidaException extends Exception {
    public TemperaturaNoValidaException() {
        super("La temperatura no puede ser inferior a -273º Celsius");
    }
}
```

Throw

- Se utiliza para lanzar una excepción.
- Se puede emplear para indicar que ha ocurrido una excepción, particularmente cuando se quiere lanzar una excepción propia.
- Throw se utiliza junto con un tipo de exception.

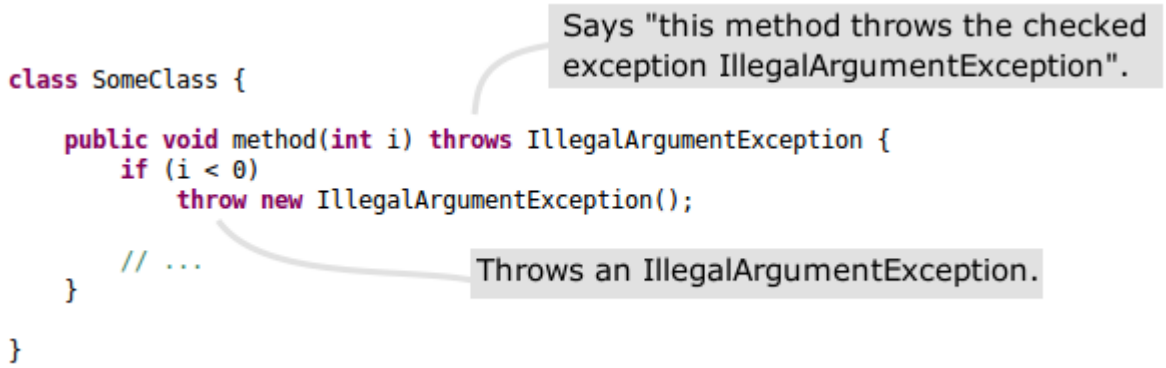
```java
public class ThrowAnException {
    public static void main (String[] args) {
        try {
            int result=divideInt(10,5);
            System.out.format("10 divided by 5 is %d%n", result);
            divideInt(10,0);
            System.out.format("10 divided by 0 is %d%n", result);
        }
        catch (IllegalArgumentException ex) {
            System.out.println(ex.getMessage());
        }
    }
    public static int divideInt(int i, int j) {
        if (j == 0) {
            throw new IllegalArgumentException("Divisor cannot be zero!");
        }
        return i/j;
    }
}
```

Throws

- Dicha sentencia obliga a quién invoque al método, usar un bloque try / catch o propagarlo con otro throw.
- Cuando un método utiliza "throws" para declarar que puede lanzar una excepción verificada, está indicando que la responsabilidad de manejar esa excepción recae en el código que llama a ese método. Es decir, el método que llama debe estar preparado para capturar y manejar la excepción o bien propagarla a su vez utilizando otra declaración "throws" en su propia firma de método.

Diferencia entre Throw y Throws

- Throw" se usa para lanzar una excepción explícitamente dentro de un bloque de código, mientras que “Throws" se utiliza en la firma de un método para indicar que ese método podría lanzar una o varias excepciones verificadas y que su manejo es responsabilidad del código que lo invoca.



Multicatch

- Permite capturar múltiples tipos de excepciones en un solo bloque de captura. Antes de esta característica, cada tipo de excepción requería su propio bloque "catch".
- Las excepciones capturadas deben ser independientes en la jerarquía de herencia de excepciones. Es decir, no deben tener una relación de subclase-superclase entre ellas.

Sintaxis Multicatch

- ExcepcionTipo1", "ExcepcionTipo2", ..., "ExcepcionTipoN" son los tipos de excepciones que se desean capturar.
- “Variable“ (en el ejemplo e) es el nombre de la variable que se utilizará para referirse a la excepción capturada en el bloque "catch". Puedes usar esta variable para acceder a la información relacionada con la excepción, como el mensaje de error o la causa de la excepción.

```
    } catch (NumberFormatException |NullPointerException | ArithmeticException e) {
    System.out.println("Exception "+e);
    e.printStackTrace();
}catch(Exception e){

}
```

Try With Resourses

- Facilta la gestión de recursos que deben ser cerrados explícitamente después de su uso, como archivos, conexiones de bases de datos o conexiones de red.
- Esta estructura se utiliza junto con la declaración try y permite que los recursos sean automáticamente cerrados al finalizar el bloque de código, incluso si ocurre una excepción.

Sintaxis Try with resourses

- Se utiliza la declaración try junto con la declaración de recursos dentro de paréntesis. Los recursos declarados deben implementar la interfaz AutoCloseable o java.io.Closeable, que define el método close() para cerrar el recurso.
- Cuando el bloque de código dentro del try finaliza, los recursos declarados se cierran automáticamente, ya sea que el bloque termine normalmente o se produzca una excepción. Esto garantiza que los recursos se liberen adecuadamente y no se generen pérdidas de memoria o errores por no cerrarlos correctamente.

```
try (Resource1 con1 = new Resource1();
                Resource2 con2 = new Resource2();
                     .
                     .
                     .
                 ResourceN conN = new ResourceN()) {
        } catch (Exception e) {

                 }
```

Interface AutoCloseable