

Las Excepciones

Las Excepciones

Las excepciones son un mecanismo usado para describir qué hacer cuando sucede algo inesperado.

Algo inesperado es un error de algún tipo, por ejemplo, un método que se invoca con argumentos no válidos, una conexión de red que falla o la solicitud de un usuario para abrir un archivo inexistente.

El siguiente ejemplo muestra un programa que pasa los argumentos por línea de comandos sin hacer ningún tipo de tratamiento de excepciones.

Funciona perfectamente si todos los argumentos son enteros, pero falla si alguno de los argumentos no es un entero.

```
public class AddArguments {  
  
    public static void main(String[] args) {  
        int sum = 0;  
        for(int i = 0; i < args.length; i++)  
            sum += Integer.parseInt(args[i]);  
        System.out.println("Sum: " + sum);  
    }  
}  
→ excepciones javac AddArguments.java  
→ excepciones java AddArguments 1 2 3 4 5 6 7 8 9  
Sum: 45  
→ excepciones java AddArguments 1 2 tres 4 5 6  
Exception in thread "main" java.lang.NumberFormatException: For input string: "  
tres"  
    at java.base/java.lang.NumberFormatException.forInputString(NumberForma  
tException.java:65)  
    at java.base/java.lang.Integer.parseInt(Integer.java:652)  
    at java.base/java.lang.Integer.parseInt(Integer.java:770)  
    at AddArguments.main(AddArguments.java:7)  
→ excepciones ■
```

La sentencia try-catch:

El lenguaje Java provee un mecanismo para resolver qué excepción será lanzada y cómo recuperarse de ella.

```
public class AddArguments2 {  
    public static void main(String[] args) {  
        try{  
            int sum = 0;  
            for(int i = 0; i < args.length; i++){  
                sum += Integer.parseInt(args[i]);  
            }  
            System.out.println("Sum: " + sum);  
        } catch (NumberFormatException nfe) {  
            System.err.println("Uno de los argumentos de línea de"  
                + "comandos no es un entero");  
        }  
    }  
}
```

→ **excepciones** javac AddArguments2.java

→ **excepciones** java AddArguments2 1 2 3 4 5 6 7 8 9

Sum: 45

→ **excepciones** java AddArguments2 1 2 tres 4 5 6

Uno de los argumentos de línea de comandos no es un entero

→ **excepciones** ■

La sentencia try-catch: Puede usarse en pequeños fragmentos de código. En el ejemplo se muestra cómo descartar los argumentos inválidos.

```
public class AddArguments {  
  
    public static void main(String[] args) {  
        int sum = 0;  
        for(int i = 0; i < args.length; i++)  
            try{  
                sum += Integer.parseInt(args[i]);  
            } catch (NumberFormatException nfe) {  
                System.err.println("[ " + args[i] + " ] no es un entero");  
            }  
        System.out.println("Sum: " + sum);  
    }  
}
```


→ **Semana12** java AddArguments 1 2 3 4 5 6 7 8 9

Sum: 45

→ **Semana12** java AddArguments 1 2 tres 4 5 6

[tres] no es un entero

Sum: 18

→ **Semana12** 

Uso de múltiples sentencias catch

Puede haber múltiples bloques catch después de un bloque try. Cada uno maneja un tipo de excepción diferente. El ejemplo a continuación lo muestra:

```
try {  
    //código que debería lanzar una o más excepciones  
} catch (MyException e1) {  
    //código a ejecutar si se lanza MyException  
} catch (MyOtherException e2) {  
    //código a ejecutar si se lanza MyOtherException  
} catch (Exception e3) {  
    //código a ejecutar si cualquier otra exception es lanzada.  
}
```

El orden de las cláusulas catch es relevante, una excepción lanzada desde el bloque try, será capturada por el primer catch que lo pueda hacer. Si **Exception** se pone en primer lugar, manejaría todas las excepciones, y las otras dos nunca se invocarían.

El mecanismo de Stack en las llamadas

Considere un caso en el cual el método `main()` llama a otro método llamado `primero()`, y éste llama a otro llamado `segundo()`. Si una excepción ocurre en `segundo()` y no es manejada allí, es lanzada hacia `primero()`. Si en `primero()` hay un `catch` para ese tipo de excepciones, la excepción es manejada y no avanza más. Sin embargo si en `primero()` no tiene un `catch` para ese tipo de excepciones, entonces el siguiente método en el stack de llamadas `main()` es verificado. Si la excepción no es manejada en el método `main()`, la excepción se despliega en la salida estándar y el programa finaliza su ejecución.

La cláusula finally

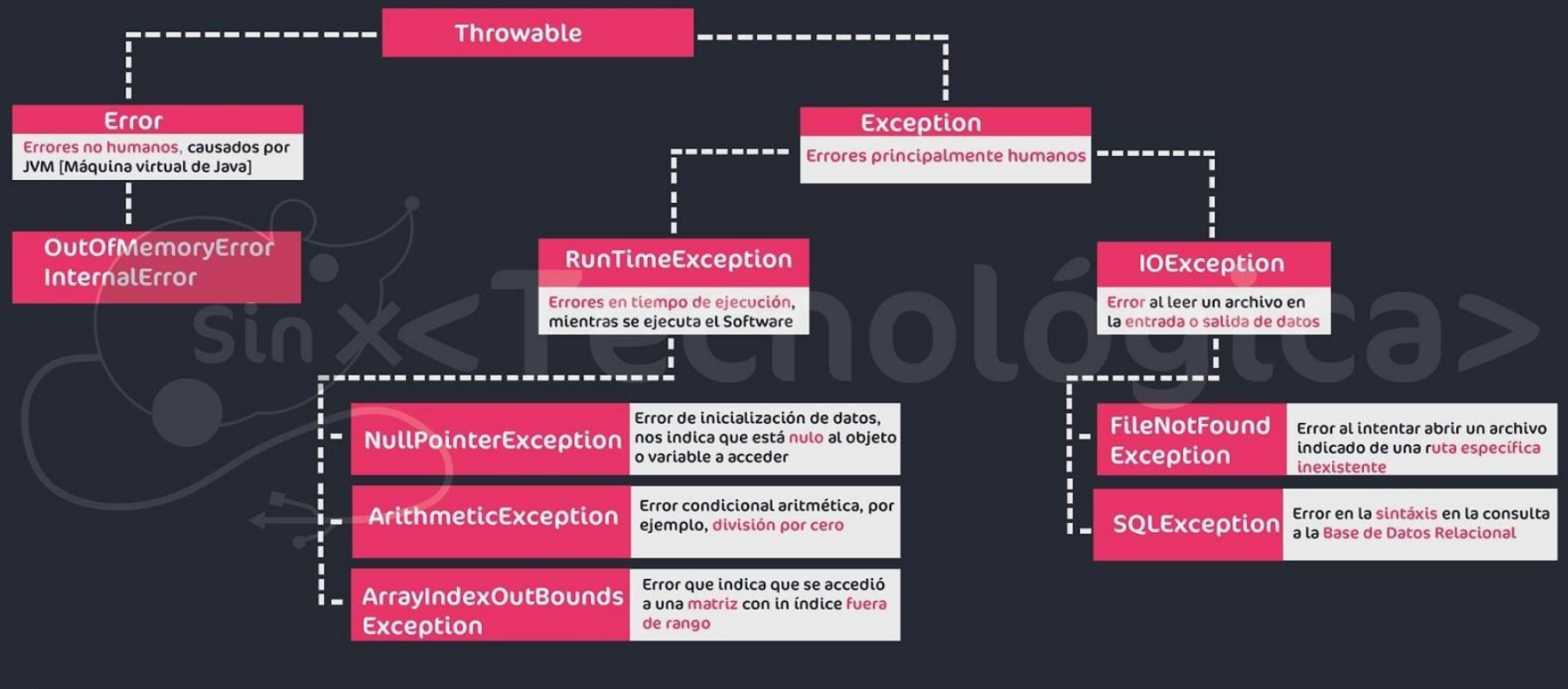
La cláusula finally define un bloque de código que siempre se ejecuta sin importar si alguna excepción fue atrapada:

```
try {  
    abrirGrifo();  
    regarCesped();  
} catch (brokenPipeException e) {  
    logProblem(e);  
} finally {  
    cerrarGrifo();  
}
```

El grifo es cerrado sin importar si una excepción ocurre mientras se está abriendo el grifo se riega el césped. Incluso si se incluye una sentencia return en el código correspondiente al bloque try, el código en la cláusula finally se ejecuta antes del return.

Las categorías de Excepción

Errores y Excepciones en Java



Las categorías de Excepción

La clase `java.lang.Throwable` actúa como la clase padre para todos los objetos que pueden ser lanzados y atrapados usando mecanismos de manejo de excepciones.

Los métodos definidos en la clase `Throwable` devuelven el mensaje de error asociado con la excepción y despliegan la traza del stack mostrando dónde ocurrió la excepción.

Hay tres subclases claves de `Throwable`:

- **Error:** Indica un problema severo del cual recuperarse puede ser difícil o imposible. Por ejemplo, quedarse sin memoria.
- **RuntimeException:** Indica un problema de diseño o implementación. Algo que nunca debería haber ocurrido si el programa funcionara adecuadamente. Una `NullPointerException`, por ejemplo.
- **IOException:** Excepciones que indican una dificultad en tiempo de ejecución causada con frecuencia por los efectos del entorno. Por ejemplo, intentar abrir un archivo no encontrado.

No debería usarse nunca la clase `Throwable`.

Algunas excepciones comunes

NullPointerException: Intento de acceder a un objeto usando una variable que no referencia a ningún objeto. Un ejemplo es cuando la variable no ha sido inicializada o cuando ningún objeto ha sido instanciado.

```
Empleado emp = null;  
System.out.println(emp.getName());
```

FileNotFoundException: Cuando se intenta leer un archivo que no existe.

NumberFormatException: Es un intento de analizar una cadena de caracteres como un número (entero o de punto flotante) que tiene un formato ilegal de número.

ArithmeticException: Este es un intento de dividir entre cero en una operación entre enteros.

```
int i=0;  
int j=12/i;
```

etc.

Regla de Manejo o Declaración

Java requiere que si alguna excepción verificada (subclase de `Exception` pero no subclase de `RuntimeException`) ocurriera en un punto cualquiera del código, el método que contiene ese punto debe definir explícitamente la acción que se tomará si el problema se origina.

- Manejar la excepción usando un bloque `try-catch-finally`.
- Declarar las Excepciones que el método puede lanzar. No se maneja la excepción y esta es derivada al método que lo lanzó

```
void trouble() throws IOException{ ... }
```

La sobrescritura de métodos que lanzan excepciones

Un método sobrescrito solo puede lanzar excepciones que son de la misma clase o una subclase de la excepción lanzada en la superclase.

Si un método de la superclase lanza una `IOException`, entonces un método que lo sobrescriba lanzará la misma o una `FileNotFoundException`, pero no una `Exception`.

```
public class TestA {  
    public void metodoA() throws IOException {  
        //  
    }  
}  
  
public class TestB1 extends TestA {  
    public void metodoA() throws EOFException {  
        //  
    }  
}  
  
public class TestB2 extends TestA {  
    public void metodoA() throws Exception {  
        //  
    }  
}
```

La class `TestB1` compila porque `EOFException` es subclase de `IOException`.

La clase `TestB2` falla al compilar porque `Exception` es superclase de `IOException`.

La creación de excepciones

Las excepciones definidas por el usuario se crean extendiendo la clase Exception.
Las clases Exception contienen lo mismo que una clase regular.

```
public class ServerTimeoutException extends Exception {  
    private int port;  
  
    public ServerTimeoutException (String message, int port) {  
        super(message);  
        this.port = port;  
    }  
  
    public int getPort() {  
        return this.port;  
    }  
}
```

Lanzar una excepción definida por el usuario

Consideremos un programa cliente-servidor. El código del cliente intenta conectarse al servidor, si luego de 5 segundos no obtiene respuesta se lanza la excepción `ServerTimedOutException` :

```
public void connectMe (String serverName) throws ServerTimedOutException {
    boolean successful;
    int portToConect = 80;

    successful = open(serverName, portToConect);

    if (!successful ) {
        throw new ServerTimedOutException("No se puede conectar" , portToConect);
    }
}
```

Manejar una excepción definida por el usuario

```
public void findServer() {  
    try {  
        connectMe(defaultServer);  
    } catch (ServerTimeoutException e) {  
        System.out.println("Server timed out, trying alternative");  
        try {  
            connectMe(alternativeServer);  
        } catch (ServerTimeoutException e1) {  
            System.out.println("Error: " + e1.getMessage() + " connecting to port " + e1.getPort());  
        }  
    }  
}
```


¿Cuál será el resultado de la ejecución del método main?

```
public class HiloEjecucion{
    private boolean[] datos = new boolean[3];

    public String metodo(int i, boolean valor){
        String salida = "";

        try{
            salida += this.getDatos(i);
            this.setDatos(i,valor);
            salida += " OK ";
        }
        catch (Exception e){
            salida += "Excepcion ";
        }
        finally{
            salida += " Finally ";
        }
        salida += " -- ";
        return salida;
    }

    public static void main(String [] args){
        HiloEjecucion hilo = new HiloEjecucion();
        System.out.println(hilo.metodo(0, true));
        System.out.println(hilo.metodo(3, false));
    }
}
```

¿Cuál será el resultado de la ejecución del método main?

```
public class HiloEjecucion{
    private boolean[] datos = new boolean[3];

    public String metodo(int i, boolean valor){
        String salida = "";

        try{
            salida += this.getDatos(i);
            this.setDatos(i,valor);
            salida += " OK ";
        }
        catch (Exception e){
            salida += "Excepcion ";
        }
        finally{
            salida += " Finally ";
        }
        salida += " -- ";
        return salida;
    }

    public static void main(String [] args){
        HiloEjecucion hilo = new HiloEjecucion();
        System.out.println(hilo.metodo(0, true));
        System.out.println(hilo.metodo(3, false));
    }
}
```

| | | | |
|-----------|---------|---------|----|
| false | OK | Finally | -- |
| Excepcion | Finally | -- | |