Java Excepciones

Octavio Robleto

octavio.robleto@gmail.com

https://octaviorobleto.com

Excepciones

- Excepciones, o sencillamente problemas. En la programación siempre se producen errores, más o menos graves, pero que hay que gestionar y tratar correctamente, ya que JAVA al presentar un error, este, interrumpe el flujo normal del programa.

```
char operador = '/';
int numero1, numero2;
int resultado = 0;

numero1 = 10;
numero2 = 0;
switch (operador) {
case '+':
    resultado = numero1 + numero2;
    break;
case '-':
    resultado = numero1 - numero2;
    break;
case '*':
    resultado = numero1 * numero2;
    break;
case '/':
    // aqui se presentara error al dividir por cero
    resultado = numero1 / numero2;
    break;
}
```

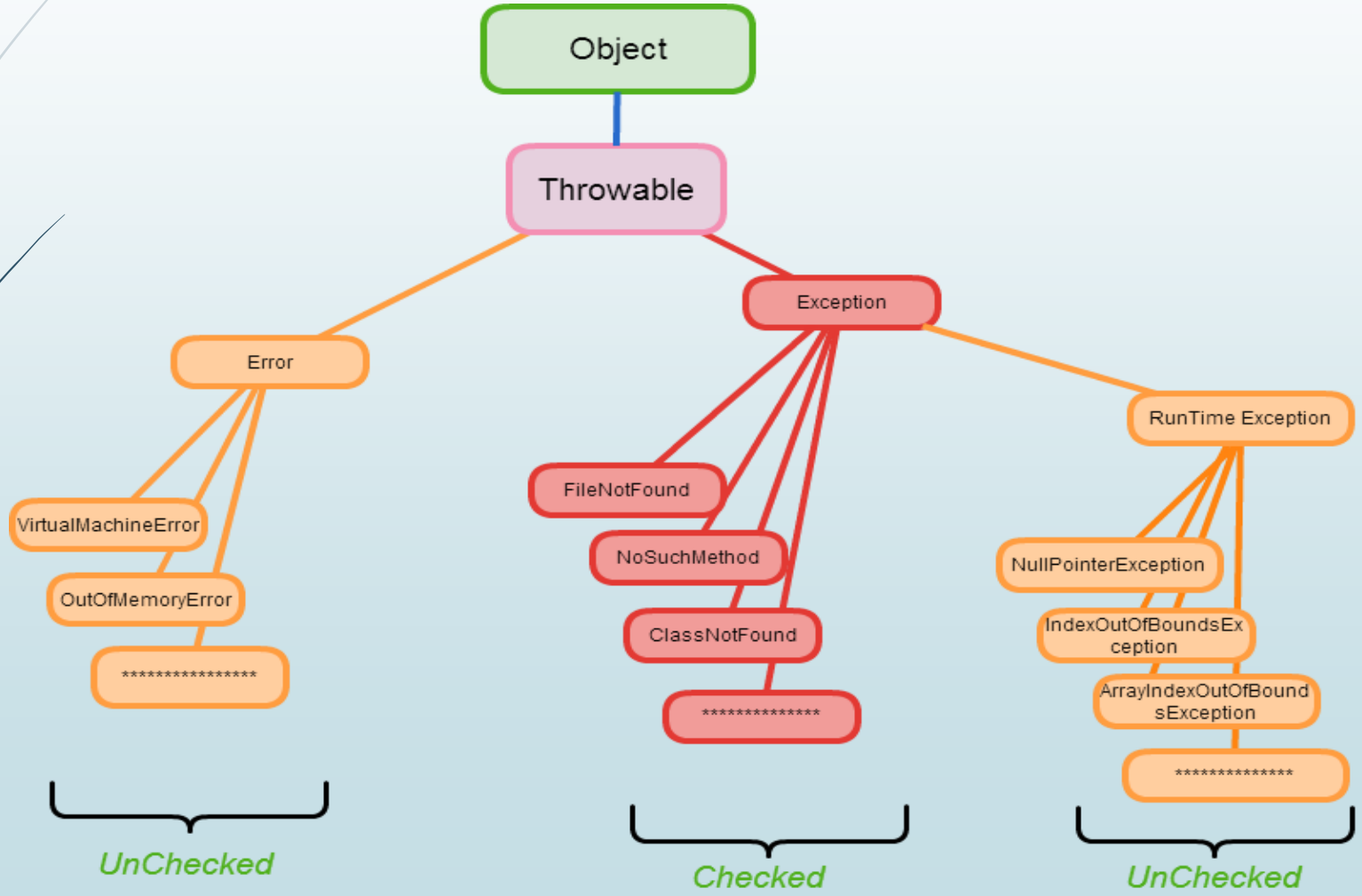
No se debe dividir por cero un entero

Bloques try, catch y finally

- Consiste en colocar las instrucciones que podrían provocar problemas dentro de un bloque try, y colocar a continuación uno o más bloques catch, de tal forma que si se provoca un error de un determinado tipo, lo que haremos será saltar al bloque catch capaz de gestionar ese tipo de error específico.
- El bloque catch contendrá el código necesario para gestionar ese tipo específico de error. Suponiendo que no se hubiesen provocado errores en el bloque try, nunca se ejecutarían los bloques catch. **(Pueden existir tantos catch se necesiten siempre y cuando sean de distinto tipo de excepción)**
- El bloque finally siempre se ejecutara.

```
// aqui se presentara error al dividir por cero
try {
    resultado = numero1 / numero2;
} catch (ArithmeticException e) {
    JOptionPane.showMessageDialog(null, "No se debe dividir por Cero");
} finally {
    System.out.println("Dividir numeros");
}
```

Tipos de Excepciones



Checked Exceptions

- Son las excepciones que tienen como superclase a la clase Exception. Necesitan ser capturadas, caso contrario no se podrá compilar el código.

```
try {
    FileReader archivo = new FileReader(new File("datos.txt"));
    BufferedReader lector = new BufferedReader(archivo);
    String linea = lector.readLine();
    while (linea != null) {
        System.out.println(linea);
        linea = lector.readLine();
    }
    lector.close();
    archivo.close();
} catch (FileNotFoundException e) {
    String mensaje = "Archivo no encontrado \n " + e.getMessage() + "\n " + e.toString();
    JOptionPane.showMessageDialog(null, mensaje, "Error", JOptionPane.ERROR_MESSAGE);
} catch (IOException e) {
    JOptionPane.showMessageDialog(null, "No existe información en el archivo", "Error",
            JOptionPane.ERROR_MESSAGE);
}
```

Unchecked Exceptions

- Son las excepciones que tienen como superclase a la clase RuntimeException. No hay necesidad de capturarlas, es decir que no se necesita utilizar el bloque try/catch/finally, pero al saltar una excepción de este tipo, como todas las excepciones corta el flujo de ejecución.
- Las excepciones de tipo Error son excepciones en las que el sistema no puede hacer nada con ellas, son clasificadas como errores irreversibles y que en su mayoría provienen desde la JVM, como por ejemplo: IOError, NoClassDefFoundError, NoSuchMethodError, OutOfMemor y Error y VirtualMachineError por mencionar algunos de los errores.

```
<terminated> ExcepcionesConError [Java Application] C:\Program Files\Java\jdk1.8.0_201\bin\javaw.exe (19 jun. 2020 18:57:55)
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.curso.java.principal.ExcepcionesConError.main(ExcepcionesConError.java:35)
```

Comando throws

- Podemos dejar que la máquina virtual de Java se encargue de las excepciones no verificas agregando la palabra clave throws y el nombre de la excepción en el método main en este caso, luego si ocurre un error se detiene el programa y nos informa la excepción lanzada

```
public static void main(String[] args) throws FileNotFoundException, IOException {

    FileReader fr = new FileReader(new File("datos.txt"));
    BufferedReader br = new BufferedReader(fr);
    String linea = br.readLine();
    while (linea != null) {
        System.out.println(linea);
        linea = br.readLine();

        br.close();
        fr.close();
    }
    System.out.println("Fin del Programa");
}
```

Métodos

- **toString():** que como ya conocemos pertenece a la clase "Object" que esta redefinido para cada una de sus clases y devuelve una cadena de caracteres.
- **getClass():** método que está heredado también de la clase "Object" y que lo que hace es devolver la clase del objeto sobre el que se invoca.
- **getMessage():** es un método que pertenece a la clase "Throwable" y que nos devuelve una una cadena de caracteres que contiene el mensaje original con el que fue creado el objeto.
- **printStackTrace():** es un método que pertenece a la clase "Throwable", que imprime a la salida estándar de errores por consola.