

# 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.

# Un ejemplo

El siguiente ejemplo muestra un programa al que se le pasan 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);  
    }  
} //Veamos como se comporta en la línea de comandos...
```

# Bloque try-catch

## `try-catch`

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

El siguiente programa captura la excepción para el primer argumento no entero de la línea de comandos y genera un mensaje de alerta. NO realiza la suma de los enteros válidos.

Luego veremos otro ejemplo, donde el programa captura la excepción para cada argumento no entero de la línea de comandos y genera un mensaje de alerta. Realiza además, la suma de todos los enteros válidos.

# Bloque try-catch

```
public class AddArguments {  
    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("One of the command-line arguments" +  
                               + "is not an Integer");  
        }  
    }  
} //Veamos como se comporta en la línea de comandos...
```



# Manejo refinado

La sentencia try-catch puede usarse en pequeños segmentos de código.

```
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("One of the command-line arguments " +  
                    "[" + args[i] + "] is not an Integer");  
            }  
        }  
        System.out.println("Sum: " + sum);  
    }  
}
```

# Múltiples catch

Puede haber múltiples bloques catch después de un bloque try. Cada uno maneja un tipo de excepción diferente.

```
try {  
    // code that might throw a particular exception  
} catch (MyExceptionType myExcept) {  
    // code to execute if a MyExceptionType exception is thrown  
} catch (MyOtherExceptionType myOtherExcept) {  
    // code to execute if a MyOtherExceptionType exception is thrown  
} catch (Exception otherExcept) {  
    // code to execute if a general Exception exception is thrown  
}
```

# El orden es importante

- Si 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.

```
try {  
    // code that might throw a particular exception  
} catch (Exception otherExcept) {  
    // code to execute if a general Exception exception is thrown  
} catch (MyExceptionType myExcept) {  
    // code to execute if a MyExceptionType exception is thrown  
} catch (MyOtherExceptionType myOtherExcept) {  
    // code to execute if a MyOtherExceptionType exception is thrown
```



# Pila de llamadas

Considere un caso en el cual:

- El método `main()` llama a otro método llamado `primero()`.
- `primero()` 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.
- 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.

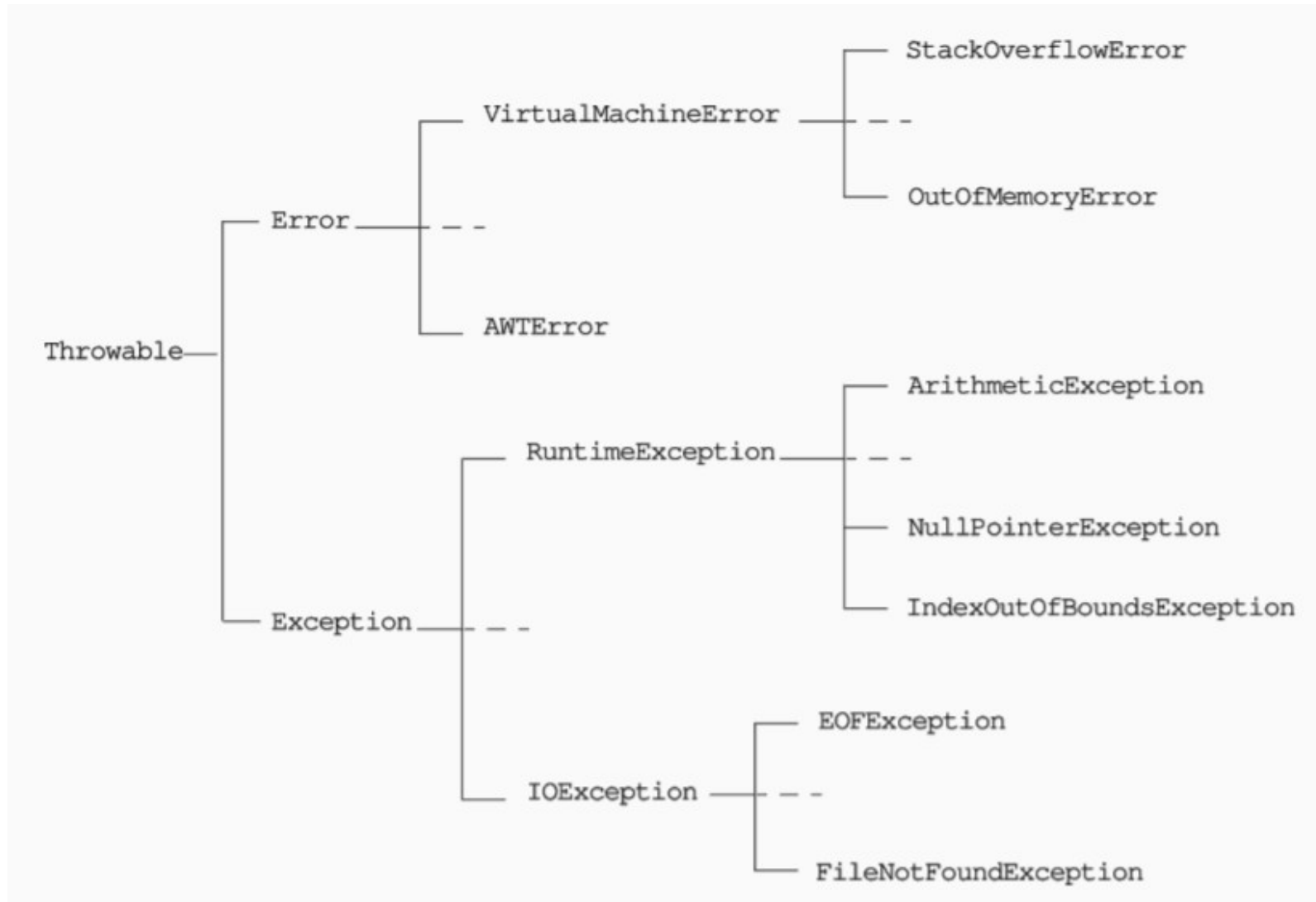
```
public class HelloWorld {  
    public static void primero(String [] greetings ){  
        segundo(greetings);  
    }  
    public static void segundo(String [] greetings ){  
        int i = 0;  
        while (i < 4) {  
            System.out.println (greetings[i]);  
            i++;  
        }  
    }  
    public static void main (String[] args) {  
        String greetings [] = {"Hello world!", "No, I mean it!", "HELLO WORLD!!"};  
        primero(greetings);  
    }  
}
```

```
public class HelloWorld {  
    public static void primero(String [] greetings ){segundo(greetings);}   
    public static void segundo(String [] greetings ){  
        int i = 0;  
        while (i < 4) {  
            try{  
                System.out.println (greetings[i]);  
            } catch (ArrayIndexOutOfBoundsException aiobe){  
                System.err.println("Fuera de los límites del arreglo");  
            }  
            i++;  
        }  
    }  
    public static void main (String[] args) {  
        String greetings [] = {"Hello world!", "No, I mean it!", "HELLO WORLD!!"};  
        primero(greetings);  
    }  
}
```

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

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int i = 0;  
        String[] greetings = {"Hello world!", "No, I mean it!", "HELLO WORLD!!"};  
        try {  
            while (i < 4) {  
                System.out.println(greetings[i]);  
                i++;  
            }  
        } catch (ArrayIndexOutOfBoundsException e){  
            System.out.println("Re-setting Index Value");  
            i = 4;  
        } finally {System.out.println("This is always printed");}  
    }  
}
```

# Categorías





# Categorías

La clase `java.lang.Throwable` actúa como super clase 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 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. Por ejemplo, intentar abrir un archivo no encontrado.

# Nos podemos encontrar con

- **NullPointerException:** Intento de acceder a un objeto usando una variable que no referencia a ningún objeto. Por ejemplo 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.

# Manejo / 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{ ... }
```

# Lanzar una excepción en Java

```
throw new TipoDeExcepcion("Mensaje de error opcional");
```

## Puntos Clave al Lanzar Excepciones:

throw vs. throws vs catch:

- throw se usa dentro de un método para lanzar una instancia de excepción.
- throws se usa en la firma del método para declarar qué tipos de excepciones verificadas puede lanzar ese método.
- Catch se usa para capturar una excepcion lanzada por throw y verificada por throws.