



Masterclass

Exception

IT Academy - Ainoa Aran



Index

01

Exceptions

02

Jerarquía

03

Manejo de exceptions

04

Exceptions personalizadas

05

Exceptions más comunes

06

Buenas prácticas

01 Exceptions



¿Qué es una exception?



Una excepción es un evento “exceptional” que interrumpe el flujo normal de un programa.

Exceptions

¿Por qué ocurre una exception?

- Un usuario ha introducido datos no válidos.
- No se encuentra un archivo que se debe abrir.
- Se ha perdido una conexión de red o el dispositivo se ha quedado sin memoria.

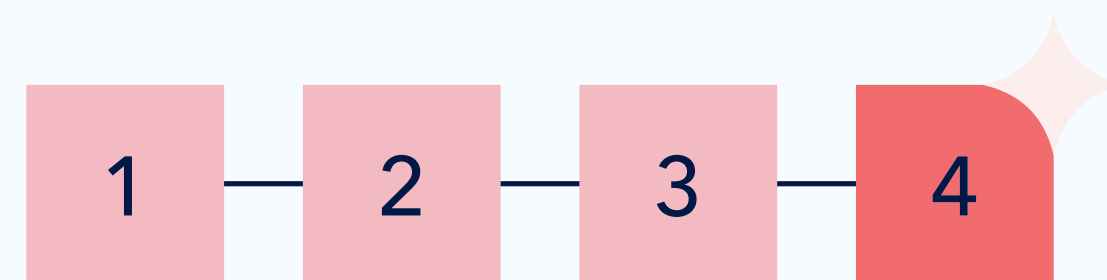
¿Por qué es importante manejarlas?

- Evitar que el programa falle y termine abruptamente.
- Programa más robusto y confiable.
- Informar del error: Proporcionar mensajes de error claros y útiles al usuario o al desarrollador.

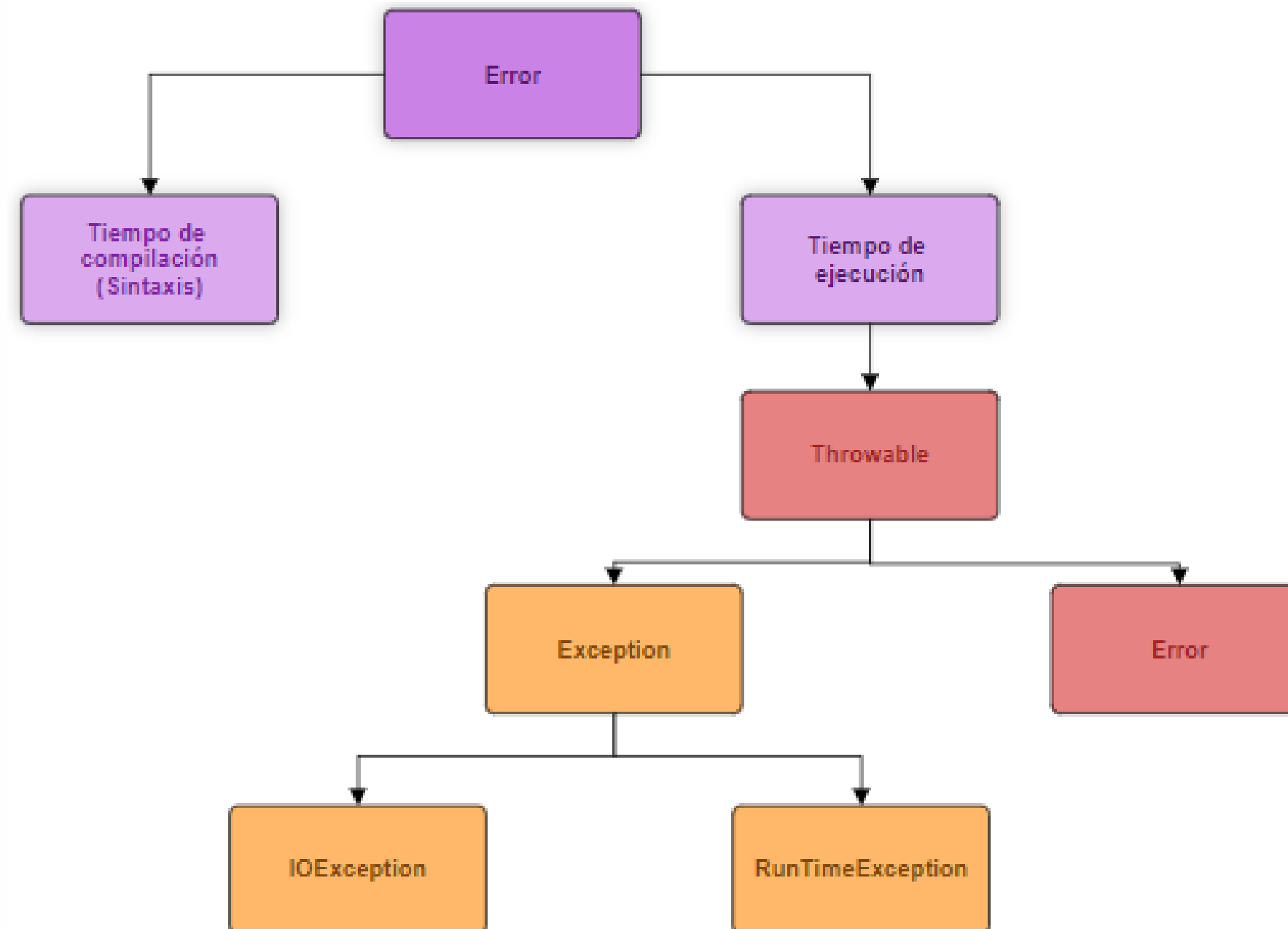


02

Jerarquía



Jerarquía



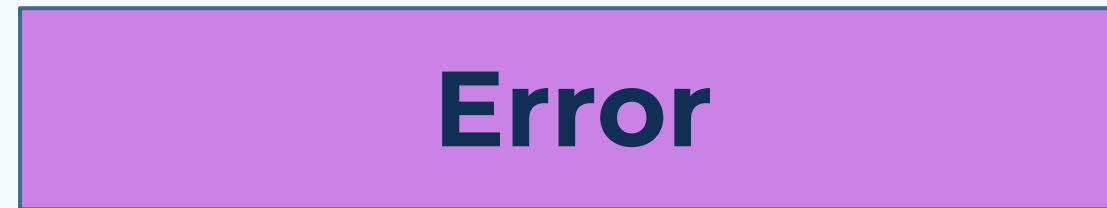
Jerarquía

Error

**Tiempo de
compilación
(Sintaxis)**

**Tiempo de
ejecución**

Throwable



Throwable

Todas las excepciones y errores heredan de la clase Throwable



Exception

Representa condiciones que podemos manejar.

IOException

RuntimeException



Error

Representa problemas graves que generalmente no se pueden manejar.

Jerarquía

Exception

Checked

IOException

I/O operations:

- Archivos no encontrados
- Permisos insuficientes
- Errores de red
- Memoria llena
- Errores de formato

Unchecked

RuntimeException

- Errores de programación
- Errores lógicos



Checked

IOException

- Errores en tiempo de compilación.
- Compilador obliga a manejarlas.
- Se deben a problemas externos.
- Requieren que el código las maneje explícitamente utilizando bloques `try-catch` o que se declare que las lanza mediante la cláusula `throws`.



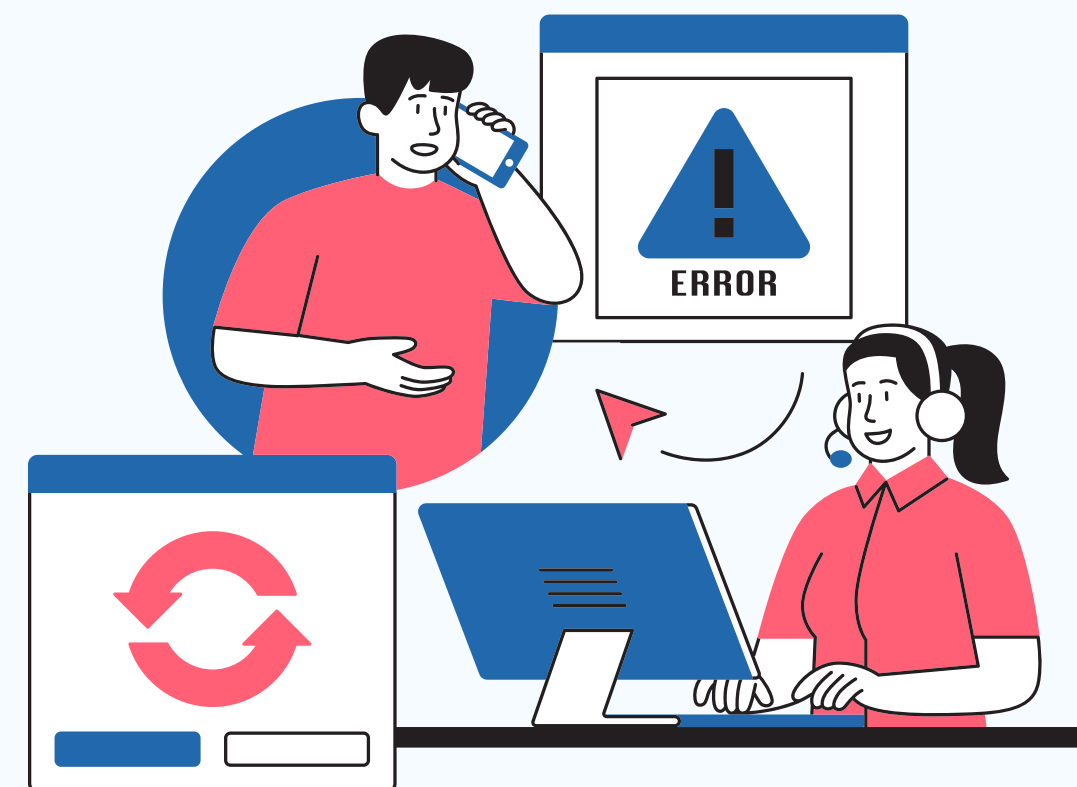
Unchecked

RuntimeException

- Ocurren durante la ejecución del programa.
- Compilador no nos obliga a manejarlas.
- Se debe a problemas de código.



03 Manejo de excepciones



Bloque try-catch-finally



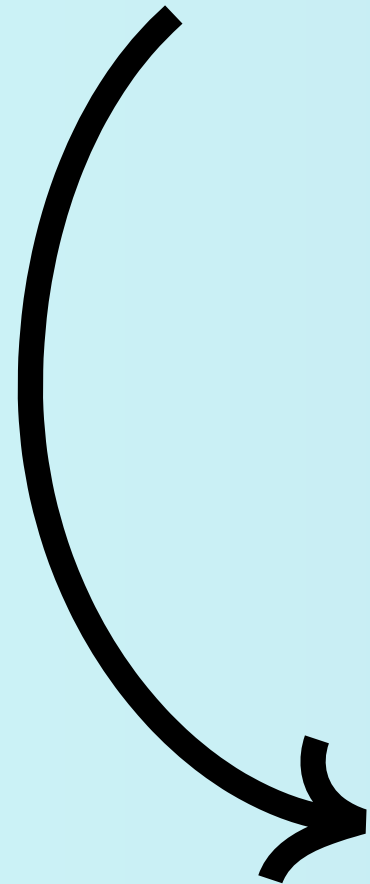
Contiene el código que puede generar una exception.

Se ejecuta si existe una exception dentro del bloque try.

Se ejecuta siempre, independientemente de si hay una exception o no.



```
try {  
    // Código que puede generar un error o excepción.  
    int operation = 10 / 0;  
} catch (ArithmeticException e) {  
    // Código que maneja la excepción  
    System.out.println("Error: divided by zero.");  
} finally {  
    // Bloque opcional que siempre se ejecuta  
    System.out.println("Optional block that is always executed.");  
}
```



```
Error: divided by zero.  
Optional block that is always executed.
```



```
try {
    System.out.print("Write the dividend number: ");
    int dividend = sc.nextInt();
    System.out.print("Write the divisor number: ");
    int divisor = sc.nextInt();
    int result = dividend / divisor;
    System.out.println("The result is: " + result);
} catch (InputMismatchException e) {
    System.out.println("Error: you must enter an integer number.");
} catch (ArithmeticException e) {
    System.out.println("Error: It cannot be divided by zero.");
} catch (Exception e) {
    System.out.println("Unexpected error: " + e.getMessage());
}
}
```

Manejo de excepciones

Throw

Lanza una excepción **dentro del cuerpo de un método** cuando ocurre una condición.

Throws

Se utiliza **en la declaración de un método** para indicar que puede producirse una excepción de ese tipo al llamar el método.

Throw

```
static void checkAge(int age) { 1 usage new *
    if (age < 18) {
        throw new ArithmeticException("Access denied: You must be at least 18 years old.");
    } else {
        System.out.println("Access granted - You are old enough!");
    }
}

public static void main(String[] args) { new *
    checkAge(15);
}
```

Throws

```
public static void readFile(String fileName) throws FileNotFoundException {
    FileReader fr = new FileReader(fileName);
    // Código para leer el archivo
    // ...
}

public static void main(String[] args) { new *
    try {
        readFile(fileName: "file.txt");
    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + e.getMessage());
    }
}
```

Bloque try-with-resources



The diagram illustrates the structure of a try-with-resources block. It consists of two light blue circles, one labeled 'Try' on the left and one labeled 'Catch' on the right. A large, dark blue arrow points from the 'Try' circle to the 'Catch' circle. Below each circle is a descriptive text block. The 'Try' block explains its role in automatically closing resources to prevent leaks. The 'Catch' block explains that it executes only if an exception occurs within the try block.

Try

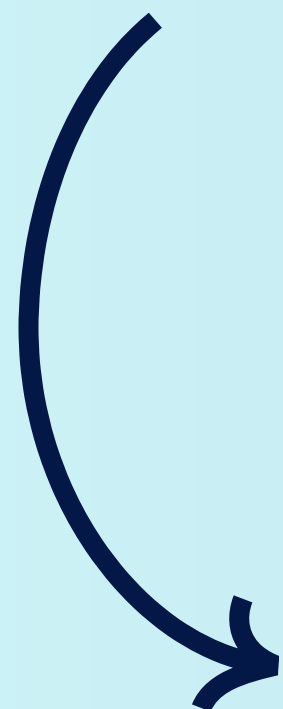
Este bloque se encarga de cerrar el recurso automáticamente al finalizar el bloque. Esto evita posibles fugas de recursos (como conexiones a bases de datos, archivos abiertos, etc.).

Catch

Se ejecuta si existe una exception dentro del bloque try.



```
public static void main(String[] args) { new *  
    try (BufferedReader br = new BufferedReader(new FileReader( fileName: "file.txt"))) {  
        String text;  
        while ((text = br.readLine()) != null) {  
            System.out.println(text);  
        }  
    } catch (IOException e) {  
        System.out.println("Error reading the text: " + e.getMessage());  
    }  
}
```



```
Error reading the text: file.txt (El sistema no puede encontrar el archivo  
especificado)
```

¿Porqué no siempre se utiliza try-catch?

Podemos controlar algunas exceptions con otros recursos, como bloque if-else.



```
public void processArray1(int[] array, int index) throws ArrayIndexOutOfBoundsException {  
    if (index < 0 || index >= array.length) {  
        throw new ArrayIndexOutOfBoundsException("Index out of bounds.");  
    } else {  
        int valueIndex = array[index];  
        System.out.println("Index value: " + index + ": " + valueIndex);  
    }  
}
```

¿Porqué no siempre se utiliza try-catch?

Podemos controlar algunas exceptions con otros recursos, como bloque if-else.



¿Y si no sé qué excepcion lanzar?

Es a base de prueba y error. Algunos comandos útiles:

- **e.printStackTrace();** : Recomendable solo a nivel local. Nos imprime la pila de error (en rojo) y donde esta alojado el constructor del error.
- **e.getMessage()** : Nos indicara que error sucede.
- **e.getClass().getName()** : Nos indica el nombre de la clase del error.

04

Exceptions personalizadas



¿Porqué crear exceptions personalizadas?

- **Mejor lectura:** Las excepciones personalizadas hacen que el código sea más claro y fácil de entender al proporcionar información específica sobre el error.
- **Manejo de Errores Específico:** capturar situaciones únicas o específicas en tu aplicación que no están bien representadas por las excepciones predefinidas.
- **Jerarquía propia:** puedes establecer convenciones y patrones de manejo de errores para tu aplicación al crear una jerarquía de excepciones personalizadas que sigan un diseño.

Crear exception personalizada

```
// Creamos una clase que extiende de Exception o RuntimeException.
public class myException extends Exception{ no usages new *
    //Constructor que recibe el mensaje personalizado.
    public myException(String message){ no usages new *
        super(message);
    }
}
```



```
public class noFundsAccountException extends Exception { 3
    public noFundsAccountException(String message) { 1 usage
        super(message);
    }
}
```

```
public void withdrawMoney(double amount) throws noFundsAccountException {
    if (amount > balance) {
        throw new noFundsAccountException("Insufficient balance: \n" +
            " You are trying to withdraw " + amount +
            " but your balance is " + balance);
    }
    balance -= amount;
}

public static void main(String[] args) { new *
    Banc account = new Banc( balance: 100);

    try {
        account.withdrawMoney( amount: 200);
    } catch (noFundsAccountException e) {
        System.out.println(e.getMessage());
    }
}
```

```
public class noFundsAccountException extends Exception { 3
    public noFundsAccountException(String message) { 1 usage
        super(message);
    }
}
```







```
public void withdrawMoney(double amount) throws noFundsAccountException {
    if (amount > balance) {
        throw new noFundsAccountException("Insufficient balance: \n" +
            "You are trying to withdraw " + amount +
```

○ ○ ○

Insufficient balance:
You are trying to withdraw 200.0 but your balance
is 100.0

```
try {
    account.withdrawMoney( amount: 200);
} catch (noFundsAccountException e) {
    System.out.println(e.getMessage());
}
}
```

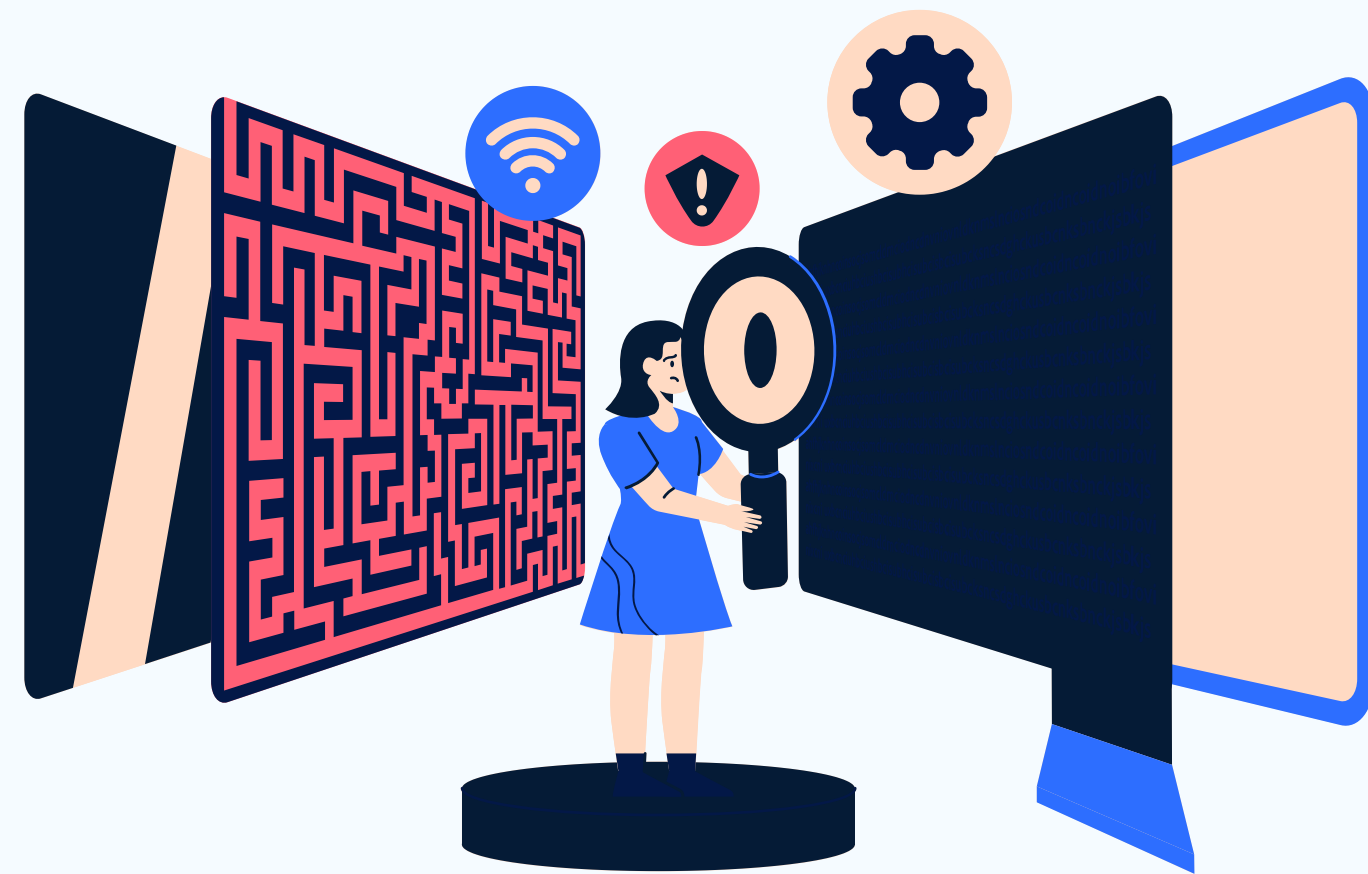
¿Cuándo crear exceptions personalizadas o una exception existente?

	Exception existente	Crear personalizada
El error ya está bien cubierto por una excepción estándar (como NullPointerException, IOException).		
Quieres que el error sea más fácil de identificar en el contexto de tu aplicación.		
Complejidad de código		

*Hay que evitar la proliferación excesiva de exception personalizadas, ya que podría afectar al rendimiento y la mantenibilidad de la aplicación.

05

Exceptions más comunes





Checked

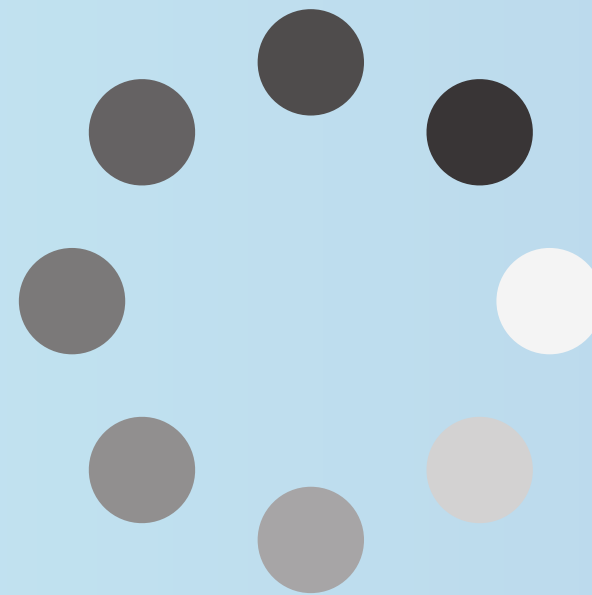
- FileNotFoundException
- SQLException
- IOException
- ClassNotFoundException

Unchecked

- ArrayIndexOutOfBoundsException
- IllegalArgumentException
- NullPointerException
- ArithmeticException



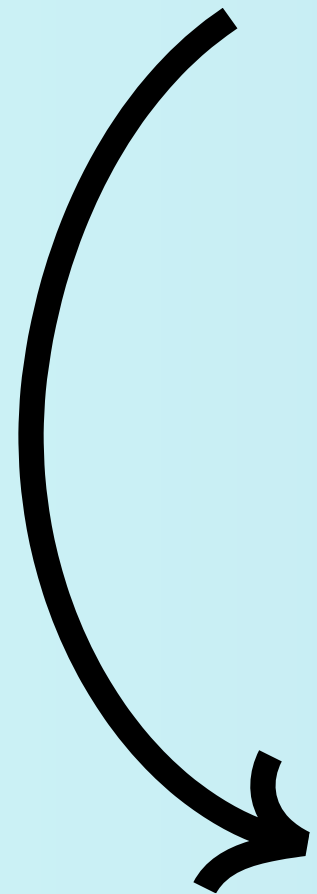
```
public static void main(String[] args) {  
    int[] number = {1, 2, 3};  
    System.out.println(number[3]);  
}
```



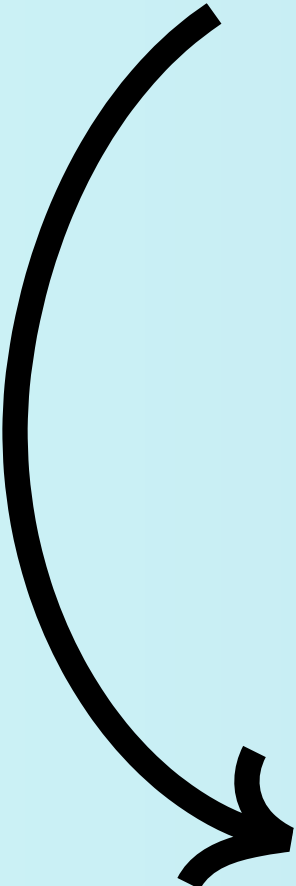


Unchecked

```
public static void main(String[] args) {  
    int[] number = {1, 2, 3};  
    System.out.println(number[3]);  
}
```



```
Exception in thread "main" java.lang  
    .ArrayIndexOutOfBoundsException Create breakpoint : Index 3 out of  
    bounds for length 3  
    at org.example.exception.main(exception.java:10)
```

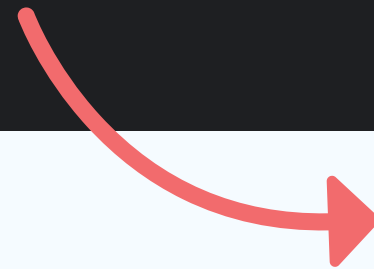


```
public static void main(String[] args) {  
    int[] number = {1, 2, 3};  
    try {  
        System.out.println(number[3]);  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("Error: The index of the array is out of bounds.");  
    }  
}
```

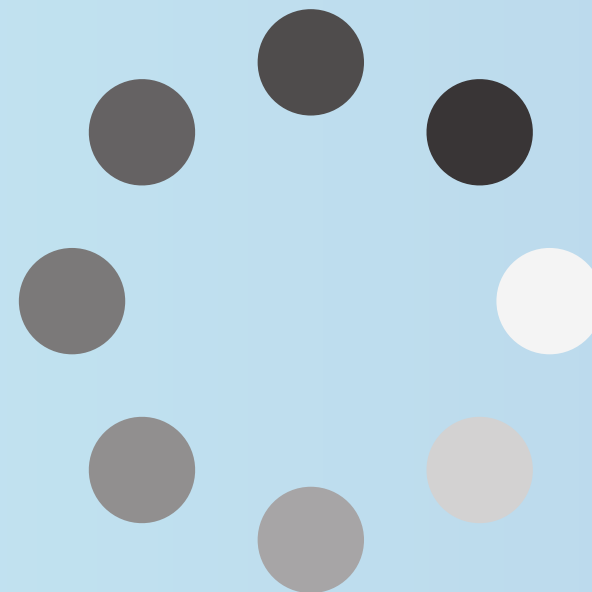
```
Error: The index of the array is out of bounds.
```




```
public static void main(String[] args) { new *  
    File file = new File( pathname: "/user/path");  
    file.createNewFile();  
}
```



Compilador nos avisa





```
public static void main(String[] args) { new *  
    File file = new File(absoluteName: "/user/path");  
    file.createNewFile()  
}
```



No se ejecuta

```
C:\Users\ainoa\Desktop\IT ACADEMY\test\src\main\java\org\example\exception.java:11:31  
java: unreported exception java.io.IOException; must be caught or declared to be thrown
```



```
Exception in thread "main" java.io.IOException Create breakpoint : El sistema no p  
encontrar la ruta especificada  
    at java.base/java.io.WinNTFileSystem.createFileExclusively0(Native Method)  
    at java.base/java.io.WinNTFileSystem.createFileExclusively(WinNTFileSystem  
        .java:569)  
    at java.base/java.io.File.createNewFile(File.java:1045)  
    at org.example.exception.main(exception.java:11)
```





Choked

```
try {  
    File file = new File( pathname: "/user/path");  
    file.createNewFile();  
}catch (IOException e){  
    System.out.println("Error to create a new file."  
        + e.getMessage());  
}
```



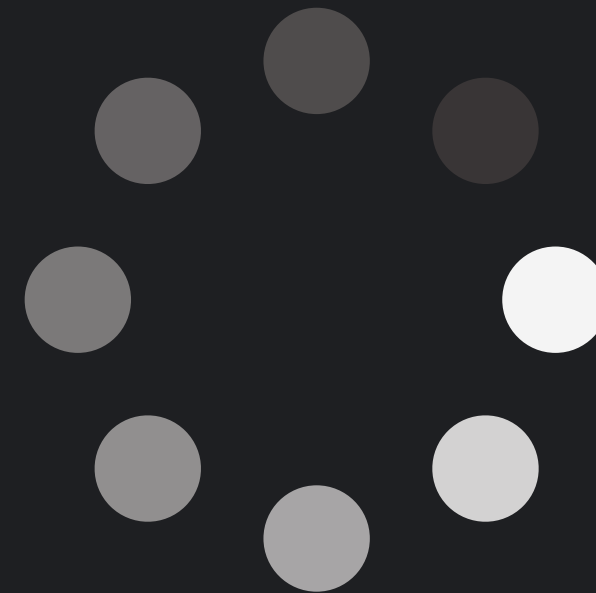
Error to create a new file: El sistema no puede encontrar la ruta especificada



```
public class Contact { 2 usages
    private String name; no usages
    private Address address; 1 usage

    // Constructor...Getters and setters...


    public String getCity() { 1 usage
        return address.getCity();
    }
}
```




```
public static void main(String[] args) { new *
    Contact contact1 = new Contact();
    System.out.println("City of contact 1: " + contact1.getCity());
}
```

Unchecked

```
public static void main(String[] args) {  
    new *  
    Contact contact1 = new Contact();  
    System.out.println("City of contact 1: " + contact1.getCity());  
}
```



```
Exception in thread "main" java.lang.NullPointerException Create breakpoint  
Cannot invoke "org.example.Address.getCity()" because "this.address" is  
null  
    at org.example.Contact.getCity(Contact.java:16)  
    at org.example.exception.main(exception.java:18)
```



```
public class Contact { 2 usages
    private String name; no usages
    private Address address; 2 usages

    // Constructor...Getters and setters...

    public String getCity() { 1 usage
        if (address != null) {
            return address.getCity();
        } else {
            return "Address not specified";
        }
    }
}
```

```
public static void main(String[] args) { new *
    Contact contact1 = new Contact();
    System.out.println("City of contact 1: " + contact1.getCity());
}
```

```
public class Contact { 2 usages
    private String name; no usages
    private Address address; 2 usages

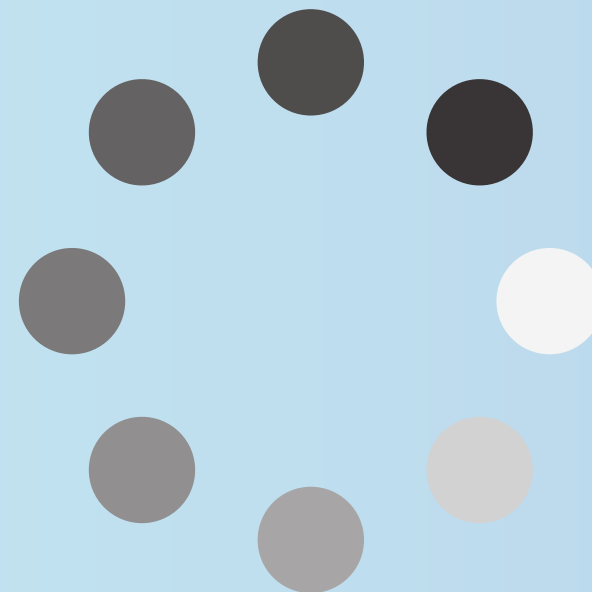
    // Constructor...Getters and setters...
}
```

```
City of contact 1: Address not specified
```

```
public static void main(String[] args) { new *
    Contact contact1 = new Contact();
    System.out.println("City of contact 1: " + contact1.getCity());
}
```



```
public class exception { new *  
    public static void main(String[] args) { new *  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Introduce your age: ");  
        int age = sc.nextInt();  
    }  
}
```



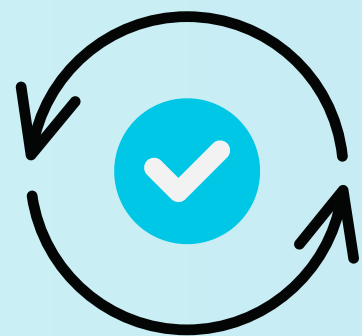
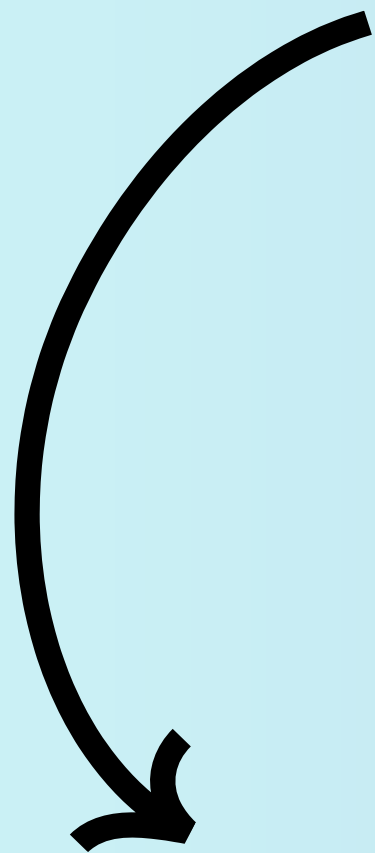
Unchecked

```
public class exception { new *  
    public static void main(String[] args) { new *  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Introduce your age: ");  
        int age = sc.nextInt();  
    }  
}
```



```
Exception in thread "main" java.util.InputMismatchException Create breakpoint  
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)  
    at java.base/java.util.Scanner.next(Scanner.java:1619)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)  
    at org.example.exception.main(exception.java:9)
```





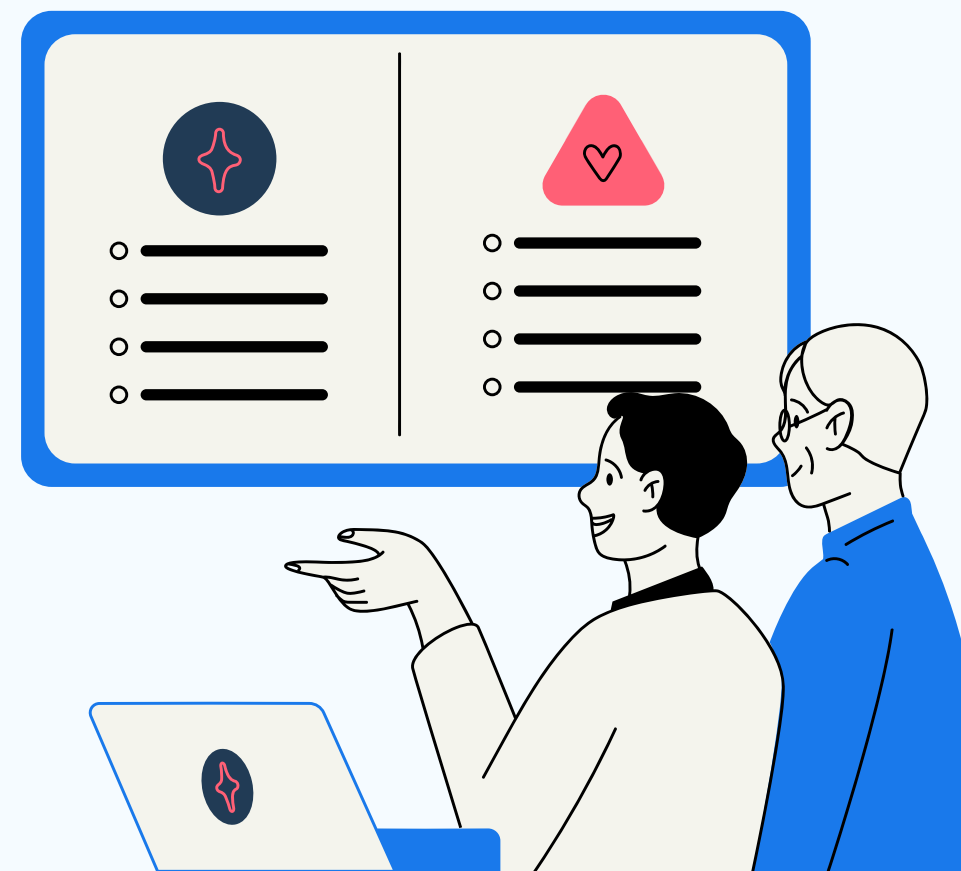
```
public class exception { new *
    public static void main(String[] args) { new *
        Scanner sc = new Scanner(System.in);
        boolean answer = false;
        do{
            try {
                System.out.println("Introduce your age: ");
                int age = sc.nextInt();
                answer=true;
            }catch(InputMismatchException e){
                System.out.println("Introduce a valid number, please.");
                sc.nextLine();
            }
        }while(!answer);
    }
}
```

```
Introduce your age:
no
Introduce a valid number, please.
Introduce your age:
|
```



06

Buenas prácticas





Buenas prácticas



- Usa excepciones específicas en lugar de genéricas.
- Lanza excepciones con mensajes claros.
- Cierre recursos apropiadamente.
- No abusar exceptions personalizadas.



IT ACADEMY - Java & Spring Framework

Ainoa Aran

