Excepciones

Semestre 02, 2025

Excepciones

Una excepción es un evento anómalo que ocurre durante la ejecución del programa y que interrumpe su flujo normal.

Cuando ocurre una excepción:

1. Java crea un objeto de tipo Exception.
2. Ese objeto se lanza (throw).
3. El sistema busca un bloque `catch` que pueda manejarlo.
4. Si no lo encuentra, la excepción se propaga hacia los métodos superiores.

Jerarquía de excepciones en Java

```
Throwable
├─ Error
│  ├─ OutOfMemoryError
│  ├─ StackOverflowError
│  └─ ...
└─ Exception
   ├─ RuntimeException
   │  ├─ NullPointerException
   │  ├─ ArithmeticException
```

```
20  |  └─ IllegalArgumentException
21
22  ├─ IOException
23
24  |  ├─ FileNotFoundException
25
26  |  └─ EOFException
27
28  └─ SQLException
29
```

- `Throwable` es la superclase base de todos los errores y excepciones.
- `Error` representa fallos graves del sistema (no deben capturarse).
- `Exception` representa situaciones que un programa puede prever y manejar.
- `RuntimeException` y sus subclases son no chequeadas.
- Las demás (`IOException`, `SQLException`, etc.) son chequeadas.

Tipos de excepciones

Chequeadas

Ejemplos: `IOException`, `SQLException`

- Obligan a manejarse con `try/catch` o declararse con `throws`.
- Representan errores previsibles y recuperables, como fallos de archivo o conexión.

No chequeadas

Ejemplos: `NullPointerException`, `ArithmeticException`

- Heredan de `RuntimeException`.
- Indican errores de programación o mal uso de la API.
- No requieren manejo obligatorio por el compilador.

Errores

Ejemplos: `OutOfMemoryError`, `StackOverflowError`

- Son condiciones graves del sistema.

- No se deben capturar, pues representan fallos fuera del control del programa.

Propagación de excepciones

Cuando ocurre una excepción, Java busca un manejador (`catch`) adecuado en el método actual.

Si no lo encuentra, la excepción se propaga al método que lo invocó, y así sucesivamente, hasta el método `main`.

```
1
2  public class Propagacion {
3
4  public static void main(String[] args) {
5
6  try {
7
8  metodo();
9
10 } catch (Exception e) {
11
12 System.out.println(
13
14 "Excepción: " + e.getMessage()
15
16 );
17
18 }
19
20 }
21
22
23
24 static void metodo() {
25
26 throw new RuntimeException(
27
28 "Error en método 2"
29
30 );
31
32 }
```

```
}
```

Salida:

```
Excepción manejada en main: Error en método 2
```

Declarar excepciones con `throws`

Cuando un método puede lanzar una excepción chequeada, debe declararlo con `throws`.

```
public void leerArchivo() throws IOException {
Files.readString(Path.of("archivo.txt"));
}
```

Esto no maneja la excepción, solo la propaga al método que lo llame.

```
public void procesar() {
try {
leerArchivo();
} catch (IOException e) {
System.out.println("Error: " + e.getMessage());
}
}
```

Captura

```
try {
Files.readString(Path.of("data.txt"));
} catch (FileNotFoundException e) {
System.out.println("Archivo no encontrado");
} catch (IOException e) {
System.out.println("Error de I/O");
}
```

- Los `catch` deben ir del más específico al más general.
- Puedes capturar varios tipos con multi-catch:

```
catch (FileNotFoundException | AccessDeniedException e) {
System.out.println("Error de acceso a archivo");
}
```

Excepciones de I/O

Las clases en el paquete `java.io` y `java.nio.file` pueden lanzar IOException o sus subclases:

```
BufferedReader br;


try {
```

```
br = new BufferedReader(new FileReader("datos.txt"))

String linea = br.readLine();

System.out.println(linea);

} catch (FileNotFoundException e) {

System.out.println("Archivo no encontrado");

} catch (IOException e) {

System.out.println("Error de lectura");

}

```

Ejemplos comunes:

- `FileNotFoundException`: el archivo no existe o no se puede abrir.
- `EOFException`: fin de archivo inesperado.
- `IOException`: error general de entrada/salida.

finally

Siempre se ejecuta, haya o no excepción.

```

FileReader fr = null;



try {

fr = new FileReader("data.txt");

} catch (IOException e) {

System.out.println("Error de lectura");

```

```
} finally {

if (fr != null) {

try {

fr.close();

} catch (IOException ignore) {}

}

}

```

Excepciones personalizadas

En Java puedes definir tus propias excepciones para representar errores específicos del dominio de tu aplicación.

Esto mejora la claridad del código y permite manejar cada tipo de error de manera más controlada.

Para crear una excepción, extiende `Exception` (chequeada) o `RuntimeException` (no chequeada):

```

// Excepción chequeada

// debe manejarse o declararse con throws



public class EdadInvalidaEx extends Exception {

public EdadInvalidaEx(String mensaje) {

super(mensaje);

}

}

```

```
// Excepción no chequeada
// no requiere manejo obligatorio


public class SaldoInsuficienteEx extends RuntimeException {
public SaldoInsuficienteEx(String mensaje) {
super(mensaje);
}
}

```

Se puede lanzar la excepción desde cualquier método usando `throw`.

```
class Persona {
private int edad;


public void setEdad(int e) throws EdadInvalidaEx {
if (e < 0 || e > 120) {
throw new EdadInvalidaEx("Edad inválida");
}


this.edad = e;
}
}

```

En este ejemplo, el método obliga a quien lo use a manejar o propagar la excepción.

```

try {

Persona p = new Persona();

p.setEdad(-5);

} catch (EdadInvalidaException e) {

System.out.println("Error: " + e.getMessage());

}



Se puede incluir campos adicionales para dar más contexto sobre el error:


```java[]

public class OperacionBancariaEx extends Exception {

private final String cuenta;

private final double monto;


public OperacionBancariaEx(String cta, double monto, String msg) {

super(msg);

this.cuenta = cta;

this.monto = monto;

}


```
```

```
43  public String getCuenta() {
44
45  return cuenta;
46
47  }
48
49
50
51  public double getMonto() {
52
53  return monto;
54
55  }
56
57  }
58
```

```

1
2   public class Cuenta {
3
4   private double saldo = 1000;
5
6
7
8   public void retirar(double monto) throws OperacionBancariaEx {
9
10  if (monto > saldo) {
11
12  throw new OperacionBancariaEx("123-456", monto, "Saldo insuficiente.");
13
14  }
15
16
17
18  saldo -= monto;
19
20  }
21
22  }
23
```

```

1
2   try {
3
```

```
4   Cuenta c = new Cuenta();
5
6   c.retirar(2000);
7
8   } catch (OperacionBancariaException e) {
9
10  System.out.println(e.getMessage());
11
12  System.out.println("Cuenta: " + e.getCuenta());
13
14  System.out.println("Monto: " + e.getMonto());
15
16  }
17
```

Buenas prácticas

- Usar excepciones personalizadas solo si aportan semántica o contexto adicional.
- Incluir información relevante (por ejemplo, ID del usuario o valor inválido).
- `Exception` para casos recuperables, `RuntimeException` para errores de lógica o precondiciones.
- Evitar excepciones genéricas con mensajes vagos como “Error del sistema”.
- Nombres claros: `ProductoNoEncontradoException`, `SaldoInsuficienteException`, etc.