Excepciones

Programación Orientada a Objetos (CC2008) - 2026

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
      │     └─ IllegalArgumentException
      ├─ IOException
      │     ├─ FileNotFoundException
      │     └─ EOFException
      └─ SQLException
```

- `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`.

```
public class Propagacion {
    public static void main(String[] args) {
        try {
            metodo();
        } catch (Exception e) {
            System.out.println(
                "Excepción: " + e.getMessage()
            );
        }
    }

    static void metodo() {
        throw new RuntimeException(
            "Error en método 2"
        );
    }
}
```

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;
    }

    public String getCuenta() {
        return cuenta;
    }

    public double getMonto() {
        return monto;
    }
}
```

```
public class Cuenta {
    private double saldo = 1000;

    public void retirar(double monto) throws OperacionBancariaEx {
        if (monto > saldo) {
            throw new OperacionBancariaEx("123-456", monto, "Saldo insuficiente.");
        }

        saldo -= monto;
    }
}
```

```
try {
    Cuenta c = new Cuenta();
    c.retirar(2000);
} catch (OperacionBancariaException e) {
    System.out.println(e.getMessage());
```

```
    System.out.println("Cuenta: " + e.getCuenta());
    System.out.println("Monto: " + e.getMonto());
}
```

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.