Asignatura

Programación

UNIVERSAE
Instituto Superior de FP

Asignatura

Programación

UNIDAD 10

Control y manejo de excepciones

UNIVERSAE
Instituto Superior de FP

Excepciones

Los errores

- Errores de compilación
- Errores de ejecución

Error de compilación

```
int variable = "hola";
```

Type mismatch: cannot convert from String to int

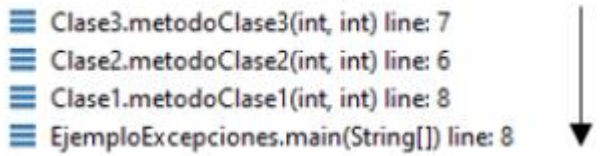
Error de ejecución

```
int variable = 10;

variable = 10/0;
```

Orden de ejecución

- Punto principal (MAIN)
- Llamadas a clases, funciones, librerías externas, etc.
- Cada llamada se acumula. **Pila de llamadas**.

Pila de llamadas



¿Que sucede si se produce un error?

- Se produce una excepción
- La excepción se propaga sobre la pila de llamadas hasta regresar al punto principal.

```
Empieza nuestro software
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Clase3.metodoClase3(Clase3.java:7)
        at Clase2.metodoClase2(Clase2.java:6)
        at Clase1.metodoClase1(Clase1.java:8)
        at EjemploExcepciones.main(EjemploExcepciones.java:8)
```

Tratamiento de excepciones

¿Como evitar las excepciones?

- Hacer un buen desarrollo de código.
- Prever situaciones que puedan producir un error
- Diseñar un tratamiento en caso inevitable

Control de excepciones

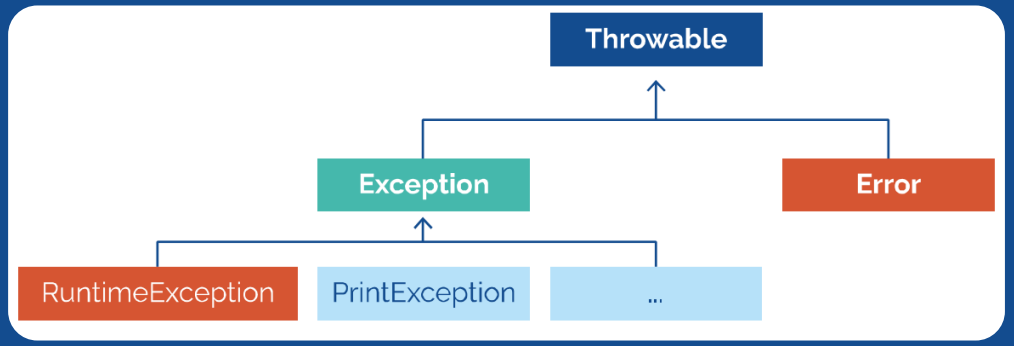
- Tener un listado de excepciones
- Poner un identificador (código) exclusivo
- Determinar el flujo de propagación
- Separar código correctivo del resto.

Beneficios

- Una aplicación más robusta
- Evitar cualquier anomalía
- Seguridad

Jerarquía

- Existe excepciones predefinidas del lenguaje de programación.
- Tienen un orden jerárquico
- La raíz es Throwable
- Se puede definir nuevas excepciones.



Excepción	Descripción error
FileNotFoundException	No encuentra un fichero
ClassNotFoundException	No existe o no encuentra una clase
EOFException	Se intenta acceder a una parte final de un fichero
ArrayIndexOutOfBoundException	Posición inexistente de un array
NumberFormatException	Número en otro tipo de formato no esperado
NullPointerException	Se intenta acceder a un objeto no inicializado
IOException	Error de entrada o salida
ArithmeticException	Operación aritmética incorrecta
OutOfMemoryError	No hay suficiente memoria para su ejecución
StackOverFlowError	Sobrecarga de la pila de ejecución

Tipos

Checked

- Son errores leves
- La causa no debe de afectar a la estabilidad del programa
- Obligatorio realizar tratamiento
- Suelen ser del tipo *Exception*

Ejemplos

- ArithmeticException
- IOException
- DataFormatException

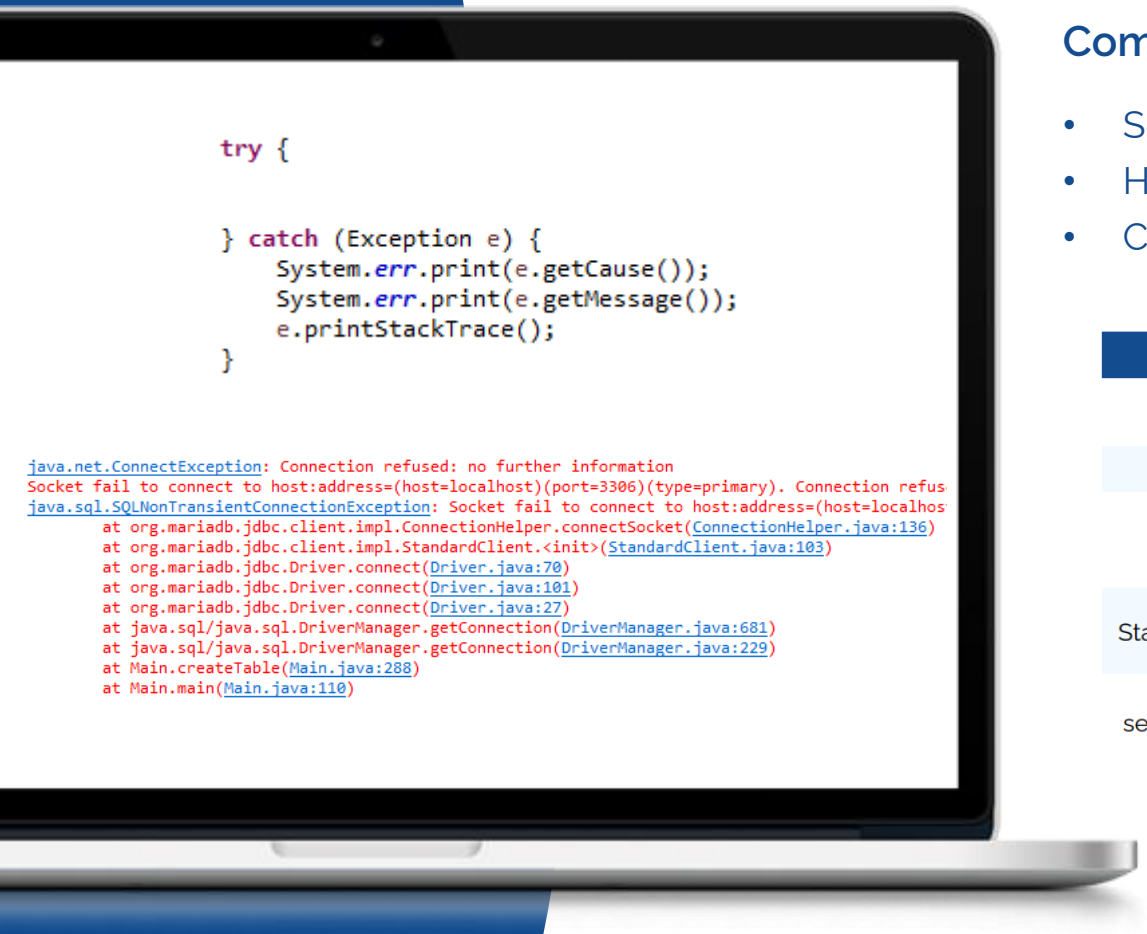
Unchecked

- Son errores graves.
- La causa puede afectar a la estabilidad del programa.
- No hay que hacer tratamiento.
- Suelen ser del tipo *Error* o *RuntimeException*

Ejemplos

- OutOfMemoryError
- NoClassDefFoundError
- VirtualMachineError

Miembros de una excepción



Composición

- Son clases
- Heredan de alguna excepción predefinida de la jerarquía
- Contienen atributos y métodos

Método	Descripción
Throwable getCause()	Devuelve un objeto Throwable con los datos de la causistica de la excepción.
String getMessage()	Devuelve el mensaje de la excepción.
printStackTrace()	Imprime todo el flujo de llamadas a los métodos hasta donde se ha producido el error.
StackTraceElement[] getStackTrace()	Obtiene una colección de StackTraceElement con cada flujo de excepcion.
setStackTrace(StackTraceElement[])	Establece una colección de StackTraceElement para indicar el flujo de la excepcion.

Gestión de excepciones. Captura

Sintaxis

```
try {
        // Código que produce un error
} catch (Tipo excepción) {
        // Código correctivo
} finally {
        // Código que siempre se ejecuta
}
```

```java
try {
    System.out.println("- Inicio del programa");

    int numero;

    System.out.println("Escribe un número");
    Scanner entrada = new Scanner(System.in);
    numero = Integer.valueOf(entrada.nextLine());
    System.out.println("El valor introducido es: "+numero);

} catch (NumberFormatException e) {
    System.err.println("El valor introducido no es un número:\n"
                       +"Mensaje : "+e.getMessage() );
} catch (Exception e) {
    System.err.println("Se ha producido un error:\n"
                       +"Causa : "+e.getCause() +"\n"
                       +"Mensaje : "+e.getMessage() );
} finally {
    System.out.println("- Fin del programa");
}
```

```
<terminated> EjemploCaptura (1) [Java Application] E:\Software\Eclips
- Inicio del programa
Escribe un número
es una cadena de texto
El valor introducido no es un número:
Mensaje : For input string: "es una cadena de texto"
- Fin del programa
```

Gestión de excepciones. Propagación

¿Hay que capturar todas las excepciones?

- No. Uso de la propagación.
- Utilizar *throws (tipo excepción)* en la cabecera del método

¿Cuándo propagar?

- Tenemos una clase o método raíz para capturar la excepción
- La funcionalidad es reutilizada en múltiples partes del programa
- Se desarrollan métodos de interfaces

```
public static void main(String[] args) {
    try {
        dividir(0,0);
    } catch (ArithmeticException e) {
        System.out.println("Se ha producido el siguiente error: "+e.getMessage());
        System.out.println("En: "+e.getClass());
        for (StackTraceElement traceElement : e.getStackTrace())
            System.out.println("\tat " + traceElement);
    }
}


public static int dividir(int num1, int num2)
        throws ArithmeticException {

    return num1/num2;

}
```

Gestión de excepciones. Lanzamiento

- Llamar explícitamente a excepción
- Uso de *throw new (tipo excepción)* en la parte de código correspondiente.

```
public static int calculadora(int operacion, int param, int param2)
        throws IllegalArgumentException {

    switch (operacion) {
        case 0:
                return param + param2;
        case 1:
                return param - param2;
        case 2:
                return param * param2;
        case 3:
                return param / param2;
        default:
                throw new IllegalArgumentException("Operacion no reconocida");
    }


}
```

Ejercicio de ampliación

- El código de las imágenes ¿Hacen lo mismo?
- ¿Que diferencia existe entre utilizar throws o no utilizarlo?
- ¿Qué diferencia hay entre el uso del throws y throw?
- ¿Qué código es mejor?

```
public static int calculadora(int operacion, int param, int param2)
        throws IllegalArgumentException {

    switch (operacion) {
        case 0:
                return param + param2;
        case 1:
                return param - param2;
        case 2:
                return param * param2;
        case 3:
                return param / param2;
        default:
                throw new IllegalArgumentException("Operacion no reconocida");
    }


}
```

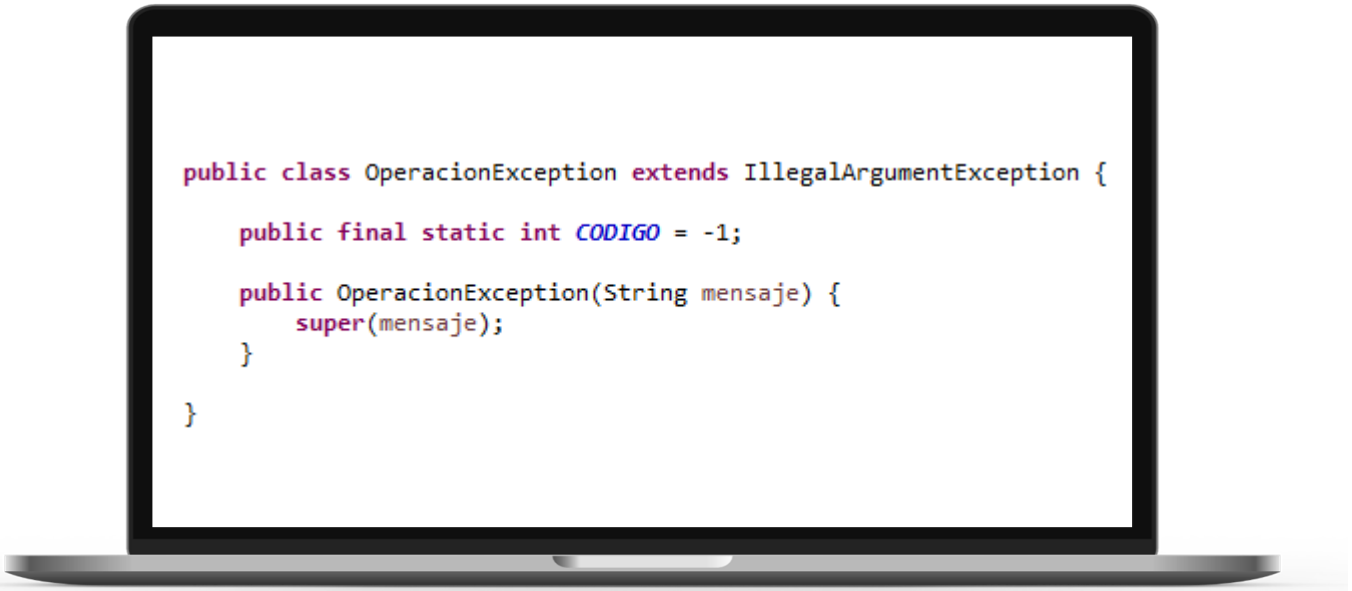
```
public static int calculadora(int operacion, int param, int param2)  {

    switch (operacion) {
        case 0:
                return param + param2;
        case 1:
                return param - param2;
        case 2:
                return param * param2;
        case 3:
                return param / param2;
        default:
                throw new OperacionException("Operacion no reconocida");
    }

}
```

Crear nuevas excepciones

- Nueva clase
- Debe de heredar de Throwable o cualquier de sus hijas
- Debe llamar al constructor padre con super()
- Puede definir sus propios atributos y métodos

Nomenclatura

- Si heredan de Exception. *nombreException*
- Si heredan de Error. *nombreError*



Recomendaciones

- No abusar de excepciones.
- Emplear correctamente los bloques try/catch
- Usar finally para objetos de entrada/salida o conexiones
- Reutilizar excepciones existentes.
- Evitar tratar excepciones genéricas

Resumen

1. Excepciones
2. Tratamiento de excepciones
3. Jerarquía
4. Tipos
5. Miembros de una excepción
6. Gestión de excepciones. Captura
7. Gestión de excepciones. Propagación
8. Gestión de excepciones. Lanzamiento
9. Ejercicio de ampliación
10. Crear nuevas excepciones
11. Recomendaciones

UNIVERSAE

CHANGE YOUR WAY