



ORACLE

Academy



Java Foundations

8-3

Manejo de Excepciones

ORACLE
Academy



Objetivos

- En esta lección se abordan los siguientes objetivos:
 - Explicar el objetivo del manejo de excepciones
 - Manejar excepciones con un constructor try/catch
 - Describir excepciones comunes devueltas en Java



¿Qué Es una Excepción?



- Para comprender el manejo de excepciones, en primer lugar, debe comprender qué es una excepción
- Una excepción es un error que se produce durante la ejecución de un programa (tiempo de ejecución) que interrumpe el flujo normal del programa Java
- Sin embargo, puede manejar dichas condiciones en el programa y tomar las medidas correctivas necesarias para que el programa pueda continuar con su ejecución (manejo de excepciones)

¿Por qué se deben manejar excepciones?

- Si se produce una excepción mientras se está ejecutando el programa:
 - La ejecución del programa finaliza
 - Un rastreo de pila, con los detalles de la excepción, se imprime en la consola

Si no se manejan excepciones: Ejemplo

- En Java, el siguiente código devuelve una excepción porque no se puede dividir un entero por cero:

```
1 public class ExceptionHandling {  
2  
3     public static void main(String args[]) {  
4         int d = 0;  
5         int a = 10 / d;  La excepción se produce en esta sentencia  
6         System.out.print(a);  Esta sentencia no se ejecuta  
7     }//end method main  
8 }//end class ExceptionHandling
```

- Un rastreo de pila, con los detalles de la excepción, se imprime en la consola
- La ejecución del programa finaliza en la línea 4 y, por lo tanto, la sentencia de la línea 5 no se ha ejecutado

Si no se manejan excepciones

- Cuando Java encuentra un error o condición que evita que la ejecución continúe con normalidad, Java "devuelve" una excepción
- Si el programador no "atrapa" la excepción, el programa se bloquea
- La descripción de excepción y el rastreo de pila actual se imprimen en la consola

Gestión de excepciones

- Un método para tratar las excepciones es simplemente evitarlas en primer lugar
- Por ejemplo, evite una `ArithmeticException` mediante lógica condicional:
 - compruebe si la condición se producirá antes de poner en marcha la operación potencialmente peligrosa

```
int divisor = 0;
if(divisor == 0){
    System.out.println("Can't be zero!");
}
else {
    System.out.println(5 / divisor);
} //endif
```


Categorías de excepciones

- Las excepciones Java se dividen en dos categorías:
- Excepciones comprobadas:
 - El compilador comprueba y se hace cargo de las excepciones
 - Si las excepciones no se manejan en el programa, da un error de compilación
 - Ejemplos:
 - `FileNotFoundException`, `IOException`
- Excepciones no comprobadas:
 - El compilador no comprueba y no se hace cargo de las excepciones
 - Ejemplos:
 - `ArrayIndexOutOfBoundsException`,
`NullPointerException`, `ArithmeticException`

Ejercicio 1

- Cree un nuevo proyecto y agréguele el archivo `ExceptionEx1.java`
- Examine `ExceptionEx1.java`:
 - Ejecute el programa y observe la salida:
 - Se produce `ArrayIndexOutOfBoundsException`
 - ¿Se recomienda manejar la excepción para este programa?
 - Modifique el programa para calcular la suma de la matriz

Manejo de excepciones con el bloque try/catch

- Pero no todas las excepciones se pueden evitar porque no siempre se sabe si una operación determinada fallará antes de que se llame
- Otra estrategia consiste en utilizar el bloque try/catch para el manejo de excepciones

Descripción del bloque try/catch

- Para el código que es probable que produzca una excepción, puede escribir el código dentro de un bloque "try" especial
- Asocie los manejadores de excepciones con un bloque try proporcionando uno o más bloques catch después del bloque try
- Cada bloque catch maneja el tipo de excepción que indica su argumento
- El tipo de argumento ExceptionType declara el tipo de excepción



Control de flujo en los bloques try/catch: Correcto

- Si el bloque try se realiza correctamente, no se produce una excepción

```
try {  
    // risky code that is likely to cause  
    // an exception  
}  
catch(ExceptionType ex) {  
    // exception handling code  
}  
System.out.println("We made it");
```

1

2

En primer lugar, se ejecuta el bloque try, y, a continuación, se ejecuta el código después del bloque catch



Control de flujo en los bloques try/catch: Fallo

- Si el bloque try falla, se produce una excepción

```
try {  
  1 // risky code that is likely to cause  
    // an exception  
}  
2 catch(ExceptionType ex) {  
    // exception handling code  
}  
3 System.out.println("We made it");
```

Se ejecuta el bloque try, se produce una excepción y el resto del bloque try no se ejecuta

Se ejecuta el bloque catch y, a continuación, se ejecuta el resto del código



Control de flujo en los bloques try/catch: Ejemplo

```
1 public static void main(String args[]) {
2     int a = 100, res;
3     try{
4         System.out.println("Enter the value for b");
5         Scanner console = new Scanner(System.in);
6         int b = console.nextInt();
7         System.out.println("Enter the value for c");
8         int c = console.nextInt();
9         res = 10 / (b - c);
10        System.out.println(" The result is " + res);
11    }
12    catch(Exception e){
13        String errMsg = e.getMessage();
14        System.out.println(errMsg);
15    } //end try catch
16    System.out.println("After catch block");
17 } //end method main
```




Ejemplos de excepciones

- `java.lang.ArrayIndexOutOfBoundsException`
 - Intenta acceder a un índice de matriz no existente
- `java.lang.NullPointerException`
 - Intenta utilizar una referencia de objeto que no se instanciaba
- `java.io.IOException`
 - Operaciones de E/S fallidas o interrumpidas

Descripción de las excepciones comunes

- Excepciones no comprobadas debido a un error de programación:
 - Ejemplo:
 - Excepción `ArrayIndexOutOfBoundsException`

```
01  int[] intArray = new int[5];  
02  intArray[5] = 27;
```

- Rastreo de pila:

```
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 5  
        at TestErrors.main(TestErrors.java:17)  
)
```

Identificación de NullPointerException

- Una excepción no comprobada se devuelve cuando una aplicación intenta utilizar un valor nulo cuando se necesita un objeto
- Son los siguientes:
 - Llamar al método de instancia de un objeto nulo
 - Acceso o modificación del campo de un objeto nulo

Llamar al
método length
de un objeto
nulo

```
public static void main(String[] args) {  
  
    String name = null;  
    System.out.print("Length of the string" + name.length());  
  
} //end method main
```

Identificación de IOException

```
public static void main(String[] args) {  
  
    try {  
        File testFile = new File("//testFile.txt");  
        testFile.createNewFile();  
        System.out.println("testFile exists:"  
                             + testFile.exists());  
    }  
    catch (IOException e) {  
        System.out.println(e);  
    } //end try catch  
} //end method main
```

Prácticas recomendadas para el manejo de excepciones

- Intente ser lo más específico posible con el tipo de error que está tratando de detectar
- Esto permitirá al programa proporcionar información específica acerca de lo que ha fallado
- Detectar una excepción genérica suele ser demasiado impreciso para ser útil, pero se puede realizar como último recurso

```
catch (Exception e) {  
    System.out.println(e);  
}
```

Ejemplo de práctica no recomendada

```
public static void main(String[] args) {  
  
    try {  
        File testFile = new File("//testFile.txt");  
        testFile.createNewFile();  
        System.out.println("testFile exists:"  
                             + testFile.exists());  
    }  
    catch (Exception e) {  
        System.out.println("Error Creating File");  
    }  
} //end try catch  
} //end method main
```

Obtención de cualquier excepción

¿No se está procesando
la clase de excepción?

Una práctica algo mejor

```
public static void main(String[] args) {  
    try {  
        File testFile = new File("//testFile.txt");  
        testFile.createNewFile();  
        System.out.println("testFile exists:"  
                             + testFile.exists());  
    }  
    catch (IOException e) {  
        System.out.println(e);  
    }  
} //end try catch  
} //end method main
```

Obtención de una excepción específica

Se llama a toString() en este objeto

Ejercicio 2

- Agregue los archivos `Calculator.java` y `ShoppingCart.java` al proyecto creado para el ejercicio 1
- Examine `Calculator.java` y `ShoppingCart.java`
- Modifique los programas para implantar el manejo de excepciones:
 - `Calculator.java`:
 - Identificar la excepción que puede producirse
 - Cambiar la firma del método `divide` para indicar que devuelve una excepción
 - `ShoppingCart.java`:
 - Obtener la excepción en la clase que llama al método `divide`

Resumen

- En esta lección, debe haber aprendido lo siguiente:
 - Explicar el objetivo del manejo de excepciones
 - Manejar excepciones con un constructor try/catch
 - Describir excepciones comunes devueltas en Java





ORACLE

Academy

