Introducción a excepciones

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Cuándo usar excepciones

Cuando en nuestro código se produce un error por una situación excepcional, la forma de prevenirlo es usando excepciones. Para usar excepciones disponemos de los bloques try / catch. Lo que podemos hacer es literalmente lo que nos dicen:

- Intentar (lo que podría darnos problema)
- Atrapar (el problema)

Cuándo usar excepciones

```
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   int num1,num2, division;

   System.out.println("Primer número, debe ser un valor entero ");
   num1=scanner.nextInt();
   System.out.println(" Divisor, un valor entero ");
   num2=scanner.nextInt();
   division= num1/num2;
   System.out.println(division);
}
```

Si ejecutamos este código, aparentemente es correcto, pero si en el segundo número se ingresa 0, se generará una **excepción**.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.company.Main.main(Main.java:16)
```

Solución con excepciones

Para mostrar el error, utilizamos **System.err.println**, esto hará que el mensaje salga en otro color

```
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   int num1,num2, division;

   System.out.println("Primer número, debe ser un valor entero ");
   num1=scanner.nextInt();
   System.out.println(" Divisor, un valor entero ");
   num2=scanner.nextInt();
   try{
          division= num1/num2;
          System.out.println(division);}
   catch(ArithmeticException excepcion){
          System.err.println("Se intentó dividir por cero");
}
```

{código}

```
   try{
         division= num1/num2;
         System.out.println(division);}
   catch(ArithmeticException excepcion){
         System.err.println("Se intentó dividir por cero");
}
```

En el bloque **try** están las instrucciones que podrían generar un problema, en este caso, la división (si el divisor es 0).

El bloque **catch** "atrapa" la excepción, si se intenta dividir por cero, entonces, se captura esa excepción y, en este caso, se muestra el mensaje. Si se efectúa una división que no tenga inconvenientes, entonces, el catch no actúa.

ArithmeticException es el tipo de excepción que ocurrió, cuando ocurre, se crea un objeto en este caso **exception**.

Excepciones en detalle

```
   System.out.println(""Primer número, debe ser un valor entero "");
   num1=scanner.nextInt();
   System.out.println(" Divisor, un valor entero ");
   num2=scanner.nextInt();
   try{
         division= num1/num2;
         System.out.println(division);}
   catch(ArithmeticException excepcion){
         System.err.println("Se intentó dividir por cero");
}
```

Estamos protegiendo el código de la división por cero, pero aún puede haber errores inesperados. Estamos solicitando el ingreso de números enteros, pero podrían ingresar valores con decimales. Para proteger el código vamos a modificarlo.

Excepciones en detalle

```
try{
        System.out.println("Primer número, debe ser un valor entero ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, un valor entero ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

 catch(ArithmeticException excepcion){
        System.err.println("Se intentó dividir por cero");}
```

Ahora estamos protegiendo el código, pero no analizamos el error. Si se ingresa un número con decimales, nos saldrá otra **excepción**.

```
Exception in thread "main" java.util.InputMismatchException
```

Excepciones en detalle

```
try{
        System.out.println(""Primer número, debe ser un valor entero "");
        num1=scanner.nextInt();
        System.out.println(" Divisor, un valor entero ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

 catch(ArithmeticException excepcion){
        System.err.println(“Se intentó dividir por cero”);}
```

Ahora estamos protegiendo el código, pero no analizamos el error. Si se ingresa un número con decimales, nos saldrá otra **excepción**.

```
Exception in thread "main" java.util.InputMismatchException
```

Diferentes excepciones

Los bloques try / catch nos permiten utilizar más de un catch. De esa forma podemos tratar excepciones específicas primero y luego las más generales.
En el ejemplo anterior se pueden generar

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

```
Exception in thread "main" java.util.InputMismatchException
```

Entonces, vamos a adaptar el código para diferenciar la ocurrencia.

Diferenciando errores

```
try{
        System.out.println("Primer número, debe ser un valor entero ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, un valor entero ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

   catch(InputMismatchExcption excepcion){
        System.err.println("Se ingresó un tipo de dato incorrecto");
        }
   catch(ArithmeticException excepcion){
        System.err.println("Se intentó dividir por cero");
        }
```

El primer catch captura la excepción que ocurriría por un ingreso incorrecto y el segundo, la que ocurriría al intentar dividir por cero.

El bloque finally

A los bloques **try / catch** se le puede agregar el bloque **finally**, que es opcional es decir, no es obligatorio utilizarlo. El **finally** se ejecuta siempre, si no ocurre una excepción y no entra al catch, si se ejecuta el finally. Si ocurriera una excepción y el catch la “atrapa”, también se ejecuta el **finally**. Veamos cómo quedaría en el ejemplo de la siguiente página.

Diferenciando errores

```
try{
        System.out.println("Primer número, debe ser un valor entero ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, un valor entero ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

   catch(InputMismatchExcption excepcion){
        System.err.println("Se ingresó un tipo de dato incorrecto");
        }
   catch(ArithmeticException excepcion){
        System.err.println("Se intentó dividir por cero");
        }
   finally{
        System.out.println("Ha finalizado el ejemplo");}
```

El **finally** siempre se ejecuta y es opcional.


DigitalHouse>
Coding School