

Control de excepciones en Java

1. Introducción

En este material, se introduce al manejo de errores en Java. Para esto Java provee una clase ya implementada llamada *Exception*.

2. ¿Para qué las excepciones?

Para tener control de los errores que se pueden producir al ingresar datos, o leer datos o transferir datos. Por ejemplo:

- Si al usuario le solicito que me indique la cantidad de hijos que tiene y ese dato lo voy a guardar en una variable numérica, me interesaría controlar que, si ingresa “ninguno” o un guion, al momento de asignar ese dato en una variable numérica se produciría un error, previendo la situación decido encapsular el error y controlar qué hará el programa en los casos de que el dato ingresado no sea el esperado.
- Si al usuario le solicito la edad, controlar que el dato ingresado sea un número entero, positivo y que esté entre un rango determinado de números, por ejemplo, mayor o igual a 18, o entre 50 y 70.
- Cuando tengo que hacer una división y quiero controlar de no dividir entre 0.
- Cuando quiero acceder a una base de datos y quiero controlar que la base de datos

E_xcepciones

Clase en Java:

- `Exception`

Finalidad:

- Para tener control sobre los errores.
- Para que mi programa no caiga.
- Para mejorar la calidad del flujo de datos y finalmente para mejorar la calidad e integridad de mi programa.

exista, que la conexión a la misma funcione, que tenga la tabla que estoy buscando, que tenga el o los registros buscados, etc.



Ejemplo 1 – controlar de no hacer una división entre 0

Se tiene el siguiente código:

```
2
3 import java.util.Scanner;
4
5 public class DividirCero {
6
7     public static void main(String[] args) {
8         // ejemplo controlar división entre cero
9         int a = 24;
10        System.out.print("Ingresar el b:");
11        Scanner ingreso = new Scanner(System.in);
12        int b = ingreso.nextInt();
13        System.out.println("la división entre a/b= " + a / b);
14    }
15 }
```

<terminated> DividirCero [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (15 abr. 2021 19:15:08 – 19:15:08)

Ingresar el b:3
la división entre a/b= 8

En donde tenemos una variable definida con el nombre a y está inicializada con el valor 24.

Además, por consola pedimos que se ingrese otro valor que cargaremos en la variable b.

Si ejecutamos el programa y el valor ingresado es 3, se logra hacer la división y se muestra en consola el resultado de la división.

Pero, si volvemos a ejecutar el programa y el valor ingresado es 0, se corta la ejecución del programa, porque se “lanza” una excepción.

Se puede observar en la siguiente imagen, la excepción que corta la ejecución del programa.

Se ingresó 0

Descripción de la excepción, un dato muy valioso cuando el programa es muy largo.

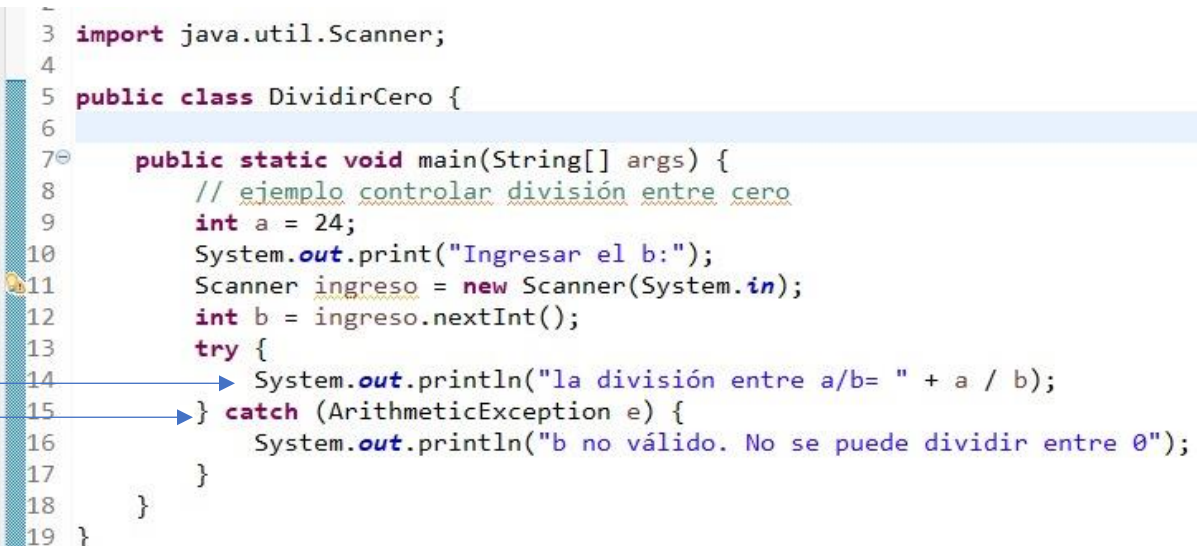


```
<terminated> DividirCero [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (15 abr. 2021 19:19:10 - 19:19:11)
Ingresar el b:0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at videos.DividirCero.main(DividirCero.java:13)
```

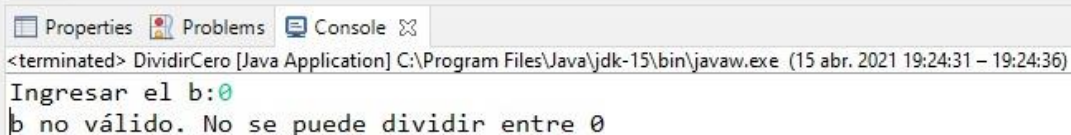
Ubicación de dónde se da la excepción: la línea 13 de la clase DividirCero.java en el método main.

Para que esto no suceda, decidimos controlar la excepción.

- En el bloque try ponga la línea que lanza la excepción. O sea, en este caso, cuando se hace la división.
- En el bloque catch, capturo la excepción y decido mostrar un mensaje porque el dato no es válido.



```
3 import java.util.Scanner;
4
5 public class DividirCero {
6
7     public static void main(String[] args) {
8         // ejemplo controlar división entre cero
9         int a = 24;
10        System.out.print("Ingresar el b:");
11        Scanner ingreso = new Scanner(System.in);
12        int b = ingreso.nextInt();
13        try {
14            System.out.println("la división entre a/b= " + a / b);
15        } catch (ArithmeticException e) {
16            System.out.println("b no válido. No se puede dividir entre 0");
17        }
18    }
19 }
```



```
<terminated> DividirCero [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (15 abr. 2021 19:24:31 - 19:24:36)
Ingresar el b:0
b no válido. No se puede dividir entre 0
```

Luego al ejecutarse el programa, ingresar 0, en lugar de cortarse la ejecución del programa, me aparece el mensaje porque se lanzó la excepción, el programa la capturó y ejecutó lo establecido en el bloque catch.

► Mira el video de este ejemplo.

En resumen: bloques **try** y **catch**.

En el primer bloque: **try**, irá todo código que pueda ser que lance una excepción. Luego, asociado al bloque **try**, irá uno o varios, bloques **catch**.

Dentro de los bloques **catch** es donde se realiza el manejo del error, es decir, donde implementamos lo que se quiere ejecutar cuando sucede un error y se lanza una excepción dentro del bloque **try**.

El **catch** es el que captura la excepción lanzada en el bloque **try**. Una vez lanzada una excepción dentro del **try**, inmediatamente se continúa la ejecución del programa en el **catch**, sin importar que queden instrucciones sin ejecutar dentro del **try**.

No puede existir un bloque **try** sin su bloque **catch** correspondiente y viceversa.

Palabras claves

try:

- Lo que intento hacer.
- En este bloque va el código que quiero controlar, la o las instrucciones que pueden ocasionar error.

catch

- Captura el error.
- En este bloque va lo que quiero que haga si se produce un error.



Ejemplo 2 – controlar de la edad ingresada este en un rango determinado

```
3 import java.util.Scanner;
4
5 public class RangoEdad {
6
7     public static void main(String[] args) {
8         // control de la edad habilitada, están habilitados entre 50 y 70
9         try {
10             int edad = LeerEdad();
11             System.out.println("Puede pasar tiene " + edad + " años");
12         } catch (ValidarException e) {
13             System.out.println(e);
14         }
15         System.out.println("fin del programa");
16     }
17
18     public static int leerEdad() throws ValidarException {
19         Scanner sc = new Scanner(System.in);
20         System.out.print("Ingrese la edad: ");
21         int edad = sc.nextInt();
22         if (edad < 50 || edad > 70) {
23             throw new ValidarException("El valor " + edad + " esta fuera de rango")
24         }
25         return edad;
26     }
27 }
```

Lo que hicimos fue lo siguiente:

1. Primero que nada, debemos generar la lógica de nuestra función para que la misma, pueda saber cuándo es necesario **lanzar la excepción**.

Hicimos la función leerEdad() para controlar el rango de números válidos y si el número ingreso no está en el rango se lanzará la excepción.

Para lanzar una excepción cuando el dato no es válido, debemos siempre colocar la palabra reservada **"throw"** seguido del constructor esta.

Por este motivo, luego del throw, es necesario llamar al constructor de la clase (que se llamara luego del new). En este caso se está llamando al constructor de la clase ValidarException pasándole como parámetro un String. Este corresponde al mensaje de error que deseamos asociarle a esta excepción.

2. Por otro lado, es necesario indicarle al método en el que se lanza la excepción, que esta excepción puede llegar a ser lanzada. Para esto, en la línea de la firma del método, luego de la definición de los parámetros de entrada, se puede ver la palabra “throws” seguido por el nombre de la excepción: ValidarException.

```
17  
18 public static int leerEdad() throws ValidarException {
```

3. Luego, hicimos la clase que extiende o hereda de la clase de Java Exception, y le definimos el constructor que recibe el String con el mensaje acorde a la excepción lanzada.

```
2  
3 public class ValidarException extends Exception {  
4     public ValidarException (String msg) {  
5         super(msg);  
6     }  
7 }
```

4. Finalmente, en el método main, implementamos el bloque try y catch. En el try hacemos referencia a la función que lee la edad y si el dato no es el esperado lanza la excepción, y si esto pasa, se captura la excepción en el bloque catch, y en este caso, el mensaje se imprime en consola.

```
9     try {  
10         int edad = LeerEdad();  
11         System.out.println("Puede pasar tiene " + edad + " años");  
12     } catch (ValidarException e) {  
13         System.out.println(e);  
14     }
```

```
Ingrese la edad: 45  
videos.ValidarException: El valor 45 esta fuera de rango  
fin del programa
```



Mira el video de este ejemplo.



Ejemplo 3 – manejo de múltiples excepciones

Dentro de un mismo método es posible desear lanzar tipos distintos de excepciones, de acuerdo a nuestro flujo de ejecución de programa. Es decir, queremos manejar distintos casos inválidos, con distintas clases de excepciones.

Veamos ahora un ejemplo de cómo poder hacer esto:

Supongamos que queremos tener una excepción cuando el año es negativo y otra distinta cuando el año es de mayor que 2017.

Tendremos ahora dos excepciones distintas, como se muestra a continuación:

```
package programa;

public class FechaFuturaException extends Exception {

    public FechaFuturaException(){
        super("La fecha es posterior al año actual!");
    }

}

package programa;

public class FechaNegativaException extends Exception {

    public FechaNegativaException(){
        super("La fecha es negativa!");
    }

}
```

En la implementación de *calcularEdad* para tomar en cuenta estos casos:

```
public static int calcularEdad(int añoNacimiento) throws FechaNegativaException, FechaFuturaException {

    if (añoNacimiento<0){
        throw new FechaNegativaException();
    }

    if (añoNacimiento > 2017) {
        throw new FechaFuturaException();
    }

    int añoActual = 2017;
    return añoActual - añoNacimiento;
}
```

Ahora, si el parámetro de entrada *añoNacimiento* es menor a 0, se lanza la excepción *FechaNegativaException*, si el *añoNacimiento* es mayor a 2017 se lanza la excepción *FechaFuturaException*.

Por lo tanto, se debe declarar esto en el *throws* del método. Puede verse, que se separa el listado de las excepciones que lanza *calcularEdad* con una coma *FechaFuturaException, FechaNegativaException*.

Cabe aclarar que la ejecución de la sentencia *throw new ClaseDeExcepcion()* le da fin a la ejecución de un método, es decir, al lanzar una excepción se termina la ejecución y no se continúa con las sentencias posteriores que se encuentran dentro del bloque. Sino que se va a continuar en el bloque *catch* que la capture.

Y en el método main:

```
public static void main(String[] args) {  
    int año = -456;  
    try{  
        int edad = calcularEdad(año);  
    } catch (FechaFuturaException ex) {  
        // manejamos el error de fecha futura como queremos!  
        System.out.println(ex.getMessage());  
    } catch (FechaNegativaException ex) {  
        // manejamos el error de fecha negativa como queremos!  
        System.out.println(ex.getMessage());  
    }  
}
```

Ahora realizamos dos *catch* en cadena, uno para capturar las excepciones *FechaFuturaException* y otro para capturar *FechaNegativaException*.

Dentro de cada bloque *catch*, debemos implementar el código que queramos ejecutar cuando una excepción sea lanzada dentro del bloque del *try*.

Cuando ocurra una excepción *FechaFuturaException*, se ejecuta:

```
} catch (FechaFuturaException ex) {  
    // manejamos el error de fecha futura como queremos!  
    System.out.println(ex.getMessage());  
}
```


Cuando ocurra una excepción `FechaNegativaException`, se ejecuta:

```
} catch (FechaNegativaException ex) {  
  
    // manejamos el error de fecha negativa como queremos!  
    System.out.println(ex.getMessage());  
}
```

En el ejemplo, se manejan de igual manera ambas excepciones, pero no necesariamente tiene que ser así, podría haber distintas sentencias a ejecutar dependiendo si ocurrió una excepción u otra.

Una vez ejecutado el bloque de `catch` el programa continua su ejecución con sentencias posteriores a los `catch`.

El programa (con `main` y método `calcularEdad`) quedó de la siguiente manera:

```
package programa;  
  
public class Programa {  
  
    public static void main(String[] args) {  
  
        int año = -456;  
  
        try{  
            int edad = calcularEdad(año);  
        } catch (FechaFuturaException ex) {  
  
            // manejamos el error de fecha futura como queremos!  
            System.out.println(ex.getMessage());  
        } catch (FechaNegativaException ex) {  
  
            // manejamos el error de fecha negativa como queremos!  
            System.out.println(ex.getMessage());  
        }  
    }  
  
    public static int calcularEdad(int añoNacimiento) throws FechaNegativaException, FechaFuturaException {  
  
        if (añoNacimiento<0){  
            throw new FechaNegativaException();  
        }  
  
        if (añoNacimiento>2017) {  
            throw new FechaFuturaException();  
        }  
  
        int añoActual = 2017;  
        return añoActual - añoNacimiento;  
    }  
}
```

Como vemos, en este caso no existe diferencia alguna en el manejo de ambas excepciones, por lo tanto, podríamos con un sólo *catch* representar lo mismo:

```
public static void main(String[] args){  
    int año = -456;  
  
    try{  
        int edad = calcularEdad(año);  
    }  
    catch(Exception ex){  
        System.out.println(ex.getMessage());  
    }  
}
```

En vez de utilizar el catch de la clase hija de *Exception* (como ser: *FechaNegativaException* y *FechaFuturaException*), la utilizamos a esta misma.

Así nos aseguramos que toda excepción que sea su hija va a ser capturada por el catch. Y al ser toda excepción hija de la clase *Exception* este catch va a capturar cualquier excepción lanzada dentro del bloque try, tanto de tipo *FechaFuturaException*, *FechaNegativaException*, así como de cualquier otro tipo de error que suceda.

Palabras claves

throw:

- Se pone cuando se quiere lanzar la excepción, indicando el constructor de la excepción que se quiere instanciar.

throws

- Se agrega en la firma del método para indicar que desde el mismo se puede lanzar una excepción.

extends Exception

- Se agrega en la firma de la clase que queremos que herede de la clase *Exception* de Java.