Tema 11: Excepciones

Antonio J. Sierra

Índice

Introducción

- Condición anormal que surge en una secuencia de código durante la ejecución.
  - Error de ejecución.
- Si no se realiza gestión de excepciones, los errores se tienen que controlar y gestionar de forma manual, normalmente utilizando códigos de error.
  - Solución incómoda.
- La gestión de excepciones de Java evita estos problemas y, en el proceso, se realiza la gestión del errores en tiempo de ejecución con objetos.

Excepciones en Java

- Una excepción de Java es un objeto que describe una condición excepcional,
  - un error que se ha producido en un fragmento de código.
- Cuando surge una condición excepcional, se crea un objeto que representa la excepción y se envía al método que ha provocado la excepción.
- Ese método puede elegir gestionar la excepción él mismo o pasarla.
- En algún punto, la excepción es capturada y procesada.

Generación de excepciones

- Las excepciones pueden ser generadas por el intérprete de Java o pueden ser generadas por el propio código.
  - Las excepciones generadas manualmente se suelen utilizar para informar de alguna condición de error al método llamante.
- La gestión de excepciones de Java se gestiona mediante cinco palabras clave: **`try`**, **`catch`**, **`throw`**, **`throws`** y **`finally`**.

Funcionamiento

- Las sentencias del programa que se quieren controlar se incluyen en un bloque **`try`**; si se produce una excepción dentro de un bloque **`try`**, ésta es lanzada.
- El código puede capturar esta excepción, utilizando **`catch`**, y tratarla de una manera racional.
- Para genera una excepción manualmente se utiliza la palabra clave **`throw`**. Cualquier excepción que se lanza fuera de un método debe ser especificada como tal utilizando la sentencia **`throws`**.
- Cualquier código que se tenga que ejecutar antes de que termine un método se introduce en un bloque **`finally`**.

Sintaxis

```
try {
  //bloque de código
}catch (TipoExcepción1 exOb) {
  //gestor de excepciones para TipoExcepción1
}catch (TipoExcepción2 exOb) {
  //gestor de excepciones para TipoExcepción2
}finally {
 //Se ejecutará antes de que termine el bloque try
}
```

Tipos de Excepciones

Todas las excepciones son subclases de la clase **`Throwable`** (lanzable). Esta clase está en la parte superior de la jerarquía de clases de excepción. Inmediatamente debajo de **`Throwable`** hay dos subclases que dividen las excepciones en dos ramas distintas.

1. **`Exception`**
2. **`Error`**

Exception

- Una rama está encabezada por la clase **`Exception`**.
- Esta clase se utiliza para condiciones excepcionales que los programas de usuario deberían capturar.
- Esta es la clase de partida de las subclases que utilizaremos para crear nuestros propios tipos de excepción.
- Una subclase importante de **`Exception`** es **`RuntimeException`** (excepción durante la ejecución). Las excepciones de este tipo son definidas automáticamente por los programas e incluyen cosas del estilo de la división por cero o un índice inválido de una matriz.

Error

- La otra rama del árbol es la clase **`Error`**, que define las excepciones que no se suelen capturar en los programas en condiciones normales.
- Las excepciones de tipo **`Error`** las utiliza el intérprete de Java para iniciar lo que tiene que hacer con los errores que se producen en tiempo de ejecución.
- Un ejemplo de este tipo de errores puede ser el desbordamiento de una pila. Este capítulo no trata las excepciones de tipo **`Error`**, porque se crean habitualmente como respuesta a fallos catastróficos que no suelen ser gestionados por los programas.

Excepciones no capturadas (I)

```
class Exc0{
public static void main(String args[ ]) {
  int d = 0;
  int a = 42 /d;
}
```

- Salida

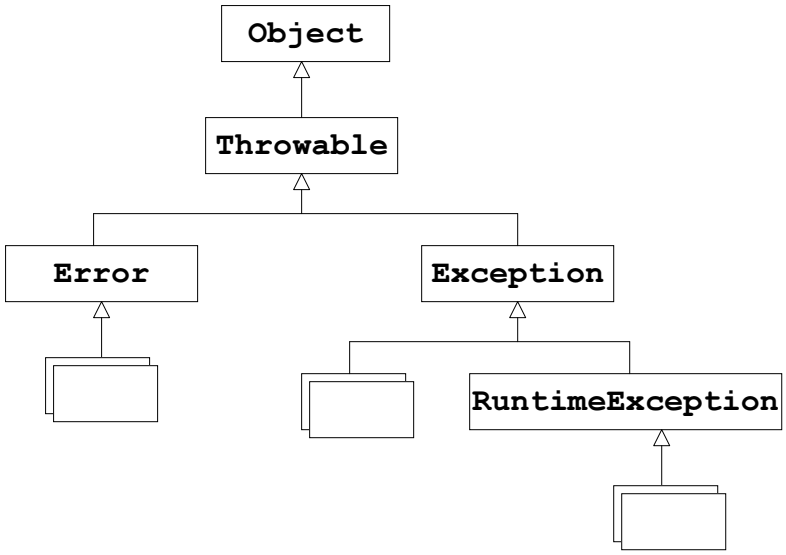
```
java.lang.AritmeticException: / by zero
at Exc0.main(Exc0.java:4)
```

Excepciones no capturadas (II)

```
class Exc1 {
  static void subrutina() {
      int d = 0;
      int a = 10/d;
  }
  public static void main (String args[]){
      Exc1.subrutina();
  }
}
```

- Salida

```
java.lang.ArithmeticException: / by zero
        at Exc1.subrutina(Compiled Code)
        at Exc1.main(Compiled Code)
```



Descripción de una excepción: **`try`** y **`catch`**.

- Para protegerse de los errores de ejecución y poder gestionarlos, hay que escribir el código que se quiere controlar dentro de un bloque **`try`**.
- Inmediatamente después del bloque **`try`** es necesario incluir la cláusula **`catch`** que especifica el tipo de excepción que se desea capturar.

Ejemplo I

```
class Exc2 {
   public static void main (String args[]){
        int d, a;

        try{    //controla un bloque de código.
                d = 0;
                a = 42/d;
                System.out.println("Esto no se imprime.");
        }catch(ArithmeticException e){ //captura div por cero
                System.out.println("División por cero.");
        }
        System.out.println("Después de catch.");
   }
}
```

Este programa genera la siguiente salida:

```
División por cero.
Después de catch.
```

Ejemplo II

```
import java.util.Random;

class Exc3 {
   public static void main (String args[]){
        int  a =0, b =0, c =0;
        Random r = new Random();

        for (int i=0; i<32000;i++){
                try{
                        b = r.nextInt();
                        c = r.nextInt();
                        a = 12345/(b/c);
                }catch(ArithmeticException e){
                        System.out.println("División por cero.");
                        a = 0;//asigna cero y continua
                }
                System.out.println("a: "+a);
        }
   }
}
```

Descripción de una excepción

- La clase **`Throwable`** sobrescribe el método **`toString()`** definido por la clase **`Object`**, devolviendo una cadena que contiene la descripción de una excepción.
- Se puede mostrar esta descripción simplemente pasando la excepción como argumento de una sentencia **`println()`**.

```
Excepción: java.lang.ArithmeticException: / by zero
```

Cláusulas catch múltiples.

- La misma secuencia de código puede activar más de un tipo de excepción. Para gestionar este tipo de situaciones , se pueden especificar dos o más cláusulas **`catch`**, cada una para capturar un tipo distinto de excepción.
- Cuando se lanza una excepción, se inspeccionan por orden las sentencias **`catch`** y se ejecuta la primera que coincide con el tipo de excepción que se ha producido.
- Después de ejecutar la sentencia **`catch`**, no se ejecuta ninguna de las restantes, continuando la ejecución al final del bloque **`try`**/**`catch`**.

Ejemplo

```
class Exc4 {
    public static void main (String args[]){
     try{
      int  a = args.length;
      System.out.println("a: " + a);
      int b = 42 / a;
      int c[] = {1};
      c[42] = 99;
    }catch(ArithmeticException e){
      System.out.println("División por cero. " + e);
    }catch(ArrayIndexOutOfBoundsException e){
      System.out.println("Índice fuera de límites. " + e);
    }
      System.out.println("Después del bloque try/catch. " );
    }
}
```

- Salida

```
[30]-adriano:Mon> java Exc4
a: 0
División por cero. java.lang.ArithmeticException: / by zero
Después del bloque try/catch.
[31]-adriano:Mon> java Exc4 jsjs
a: 1
Índice fuera de límites. java.lang.ArrayIndexOutOfBoundsException: 42
Después del bloque try/catch.
[32]-adriano:Mon>
```

Ejemplo II

```
/* Este programa contiene un error
    Una subclase debe estar antes que su superclase
    en una serie de sentencias catch. Si no es así,
    existirá código que no es alcanzable y se producirá
    un error en tiempo de compilación.
*/
class Exc5 {
    public static void main (String args[]){
          try{
                    int  a = 0;
                    int  b = 42/a;
          }catch(Exception e){
                    System.out.println("Sentencia catch para cualq. excepc." + e);
          }

          /*Esta sentencia catch nunca se ejecutará ya que la
          ArithmeticException es una subclase de Exception */

          catch(ArithmeticException e){ //ERROR
                    System.out.println("Esto nunca se alcanza ");
          }
    }
}
```

Sentencias try anidadas

- Las sentencias **`try`** pueden estar anidadas (una sentencia **`try`** puede estar dentro del bloque de otra sentencia **`try)`**.
- Cada vez que se entra en una sentencia **`try`** se almacena en una *pila el contexto* de esa excepción.
- Si la sentencia **`try`** interna no tiene un gestor **`catch`** para una excepción determinada, se examina la pila buscando los gestores **`catch`** de siguientes sentencias **`try`**.
  - Esto continúa hasta que se encuentra la sentencia **`catch`** adecuada o se agotan las sentencias **`try`** anidadas. Si no se encuentra la sentencia **`catch`** buscada, el intérprete de Java gestionará esta excepción.

Ejemplo

```
class Exc6 {
    public static void main (String args[]){
            try{
                        int  a = args.length;

                        /* Si no hay ningún argumento en la línea de comandos
                        la siguiente sentencia genera un excepción. Div por 0
                        */
                        int b = 42/a;
                        System.out.println("a = " + a);

                        try{ //bloque try anidado
                                    /* Si se utiliza un argumento en la lín. coman.,
                                    la siguiente sentencia genera una excepc. Div. 0
                                    */
                                    if (a ==1) a = a/(a-a);

                                    /*Si se le pasan dos argumento en L. C.*/
                                    if ( a==2){
                                                int c[] = {1};
                                                c[42] = 99; //genera excep. fuera de limites
                                    }
                        }catch(ArrayIndexOutOfBoundsException e){
                                    System.out.println("Índice fuera de limites. " + e);
                        }
            }catch(ArithmeticException e){ //ERROR
                        System.out.println("División por cero: " + e);
            }
    }
}
```

Ejecución

```
[34]-adriano:Mon> java Exc6
División por cero: java.lang.ArithmeticException: / by zero
[35]-adriano:Mon> java Exc6 uno
a = 1
División por cero: java.lang.ArithmeticException: / by zero
[36]-adriano:Mon> java Exc6 uno dos
a = 2
Índice fuera de limites.
   java.lang.ArrayIndexOutOfBoundsException: 42
[37]-adriano:Mon>
```

Con el mismo resultado

```
/*Las sentencias try pueden estar implícitamente anidadas
  a través de llamadas a métodos*/
class Exc7 {
    static void anidado (int a){
           try{
                      if (a ==1) a = a/(a-a);

                      if ( a==2){
                                 int c[] = {1};
                                 c[42] = 99;
                      }
           }catch(ArrayIndexOutOfBoundsException e){
                      System.out.println("Índice fuera de limites. " + e);
           }

    }

    public static void main (String args[]){
           try{
                      int  a = args.length;
                      int b = 42/a;
                      System.out.println("a = " + a);
                      anidado(a);
           }catch(ArithmeticException e){ //ERROR
                      System.out.println("División por cero: " + e);
           }
    }
}
```

throw

- Se ha trabajado con las excepciones generadas por el intérprete de Java.
- Sin embargo, utilizando la sentencia **`throw`** es posible hacer que el programa lance una excepción de manera explícita. La forma general de la sentencia **`throw`** es la siguiente:

```
throw Instanciathrowable;
```

- *Instanciathrowable* tiene que ser un objeto de tipo *Throwable* o una subclase de **`Throwable`**. Los tipos simples, como **`int`** o **`char`**, y las clases que no son **`Throwable`**, como **`String`** y **`Object`** no se pueden utilizar como excepciones.

Formas de obtener un objeto **`Throwable`**

1. Utilizando un parámetros en la cláusula **`catch`**
2. Creando uno con el operador **`new`**

Ejemplo

```
class Exc8 {
   static void demo (){
        try{
                throw new NullPointerException("demo");
        }catch(NullPointerException e){
                System.out.println("Captura dentro de demo" + e);
                throw e; //vuelve a lanzar la excepción
        }

   }

   public static void main (String args[]){
        try{
                demo();
        }catch(NullPointerException e){
                System.out.println("Nueva captura: " + e);
        }
   }
}
```

Con sus constructores

- Todas las excepciones que están en el núcleo de Java tiene dos construtores:
- 1. Uno sin parámetros y
- 2. Otro que tiene un parámetros de tipo cadena.
- El operador **`new`** se utiliza para construir una instancia de **`NullPointerException`**.

```
throw new NullPointerException("demo");
```

- Cuando se utiliza la segunda forma, el argumento especifica una cadena que describe la excepción. Esta cadena se imprime cuando se utiliza este objeto como argumento de **`println()`**.

`throws`

- Si un método es capaz de provocar una excepción que no maneja él mismo, debe especificar este comportamiento para que los métodos que lo llaman puedan protegerse frente a esta excepción.
  - **`throws`** en la declaración del método.
- **`throws`** lista los tipos de excepciones que un método puede lanzar. Esto es necesario para todas las excepciones, excepto para las de tipo **`Error`** o **`RuntimeException`**, o cualquiera de sus subclases.

Sintaxis para `throws`

```
Tipo nombre_del_método( lista_de_parámetros) throws lista_de_excepciones
{
    //cuerpo del método
}
```

- *`lista_de_excepciones`* es la lista de excepciones, separadas por comas, que un método puede lanzar

Ejemplo (Parte I)

```
class Exc9 {
   static void ThrowsDemo (){
        System.out.println("Dentro de ThrowsDemo");
        throw new IllegalAccessException("demo");
   }

   public static void main (String args[]){
        ThrowsDemo ();
   }
}
/*
C:\>javac Throw.java
Throw.java:4: unreported exception java.lang.IllegalAccessException; must be ca
ght or declared to be thrown
        throw new IllegalAccessException("demo");
        ^
1 error

C:\>
*/
```

Ejemplo (Parte II)

```
class Exc10 {
   static void ThrowsDemo () throws IllegalAccessException{
        System.out.println("Dentro de ThrowsDemo");
        throw new IllegalAccessException("demo");
   }

   public static void main (String args[]){
        try{
                ThrowsDemo ();
        }catch(IllegalAccessException e){
         System.out.println("Captura "+e);
        }
   }
}
```

El resultado:

```
Dentro de ThrowsDemo
Captura java.lang.IllegalAccessException: demo
```

`finally`

- Para cerrar descriptores de archivo y liberar cualquier otro recurso que se hubiese asignado al comienzo de un método con la intención de liberarlo antes de terminar.
- La cláusula **`finally`** es opcional.
- Sin embargo, cada sentencia **`try`** necesita, al menos, una cláusual **`catch`** o **`finally`**.
- **`finally`** crea un bloque de codigo que se ejecutará después del bloque **`try`**/**`catch`** y antes del siguiente bloque **`try`**/**`catch`**.

Su funcionamiento…

- El bloque **`finally`** se ejecutará tanto si se lanza una excepción como si no lo hace.
  - Si se lanza una excepción, el bloque **`finally`** se ejecutará incluso aunque no haya ninguna sentencia **`catch`** que capture dicha excepción.
- Siempre que un método vaya a devolver el control al método llamante desde un bloque **`try`** /**catch**, mediante una excepción no capturada, o una sentencia **`return`** explícita, se ejecuta la cláusula **`finally`** justo antes del final del método.

Ejemplo

```
class Exc11 {
 //lanza una excepción fuera del método
 static void metodoA () {
  try{
        System.out.println("Dentro de metodoA");
        throw new RuntimeException("demo");
  }finally{
       System.out.println("Sentencia finally metodoA");
  }
 }

 //ejecuta la sentencia return dentor del bloque try
    static void metodoB () {
    try{
     System.out.println("Dentro de metodoB");
           return;
    }finally{
      System.out.println("Sentencia finally metodoB");
    }
 }

 //ejecuta un bloque try normalmente
 static void metodoC () {
   try{
    System.out.println("Dentro de metodoC");
   }finally{
    System.out.println("Sentencia finally metodoC");
   }
 }

 public static void main (String args[]){
   try{
    metodoA ();
   }catch(Exception e){
     System.out.println("Excepción Capturada "+e);
   }
    metodoB ();
    metodoC ();
 }
}
```

```
Dentro de metodoA
Sentencia finally metodoA
Excepción Capturada java.lang.RuntimeException: demo
Dentro de metodoB
Sentencia finally metodoB
Dentro de metodoC
Sentencia finally metodoC
```

```
class MiExcepcion extends Exception {
    private int detalle;

    MiExcepcion(int a){
           detalle = a;
    }

    public String toString(){
           return "MiExcepcion[" + detalle + "]";
    }
}

class Exc12 {
    static void calcula (int a) throws MiExcepcion{
           System.out.println("Ejecuta calcula (" + a + ")" );
           if(a>10)
                  throw new MiExcepcion(a);
           System.out.println("Finalizacion normal");
    }
    public static void main (String args[]){
           try{
                  calcula (1);
                  calcula (20);
           }catch(MiExcepcion e){
            System.out.println("Excepción Capturada "+e);
           }
    }
}
/*
Ejecuta calcula (1)
Finalizacion normal
Ejecuta calcula (20)
Excepción Capturada MiExcepcion[20]
*/
```