TRATAMIENTO DE EXCEPCIONES

Encarnación Sánchez Gallego

ÍNDICE

Introducción

- **Una excepción es un error semántico que se produce en tiempo de ejecución**. Aunque un código sea correcto sintácticamente (es código Java válido y puede compilarse), es posible que durante su ejecución se produzcan errores inesperados, como por ejemplo:
  - Dividir por cero.
  - Intentar acceder a una posición de un array fuera de sus límites.
  - Al llamar al nextInt() de un Scanner el usuario no introduce un valor entero.
  - Intentar acceder a un fichero que no existe o que está en un disco duro corrupto.
  - Etc.

Introducción

- Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase ***Exception*** (las excepciones en Java son objetos) y se notifica el hecho al sistema de ejecución. Se dice que se ha lanzado una excepción ("***Throwing Exception***"). Existen también los errores internos que son objetos de la clase ***Error*** que no estudiaremos. Ambas clases ***Error*** y ***Exception*** son clases derivadas de la clase base ***Throwable***.
- Un método se dice que es capaz de tratar una excepción ("***Catch Exception***") si ha previsto el error que se ha producido y las operaciones a realizar para "recuperar" el programa de ese estado de error. No es suficiente capturar la excepción, si el error no se trata tan solo conseguiremos que el programa no se pare, pero el error puede provocar que los datos o la ejecución no sean correctos.
- Los programas escritos en Java también pueden lanzar excepciones explícitamente mediante la instrucción **throw**, lo que facilita la devolución de un "código de error" al método que invocó el método que causó el error.

Introducción

- ***Ejemplo 1***
- Como primer encuentro con las excepciones, vamos a ejecutar el siguiente programa. En él vamos a forzar una excepción al intentar dividir un número entre 0:

```
12  public class Ejemplos_excepciones {
13
14      public static void main(String[] args) {
15          int div, x, y;
16
17          x = 3;
18          y = 0;
19
20          div = x / y;
21
22          System.out.println("El resultado es " + div);
23      }
24
25  }
```

Salida:

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:20)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Introducción

- ***Ejemplo 2***

A continuación vamos a forzar una excepción de conversión, para ello vamos a intentar pasar a entero una cadena que no sólo lleva caracteres numéricos:

```
12    public class Ejemplos_excepciones {
13
14        public static void main(String[] args) {
15            String cadena = "56s";
16            int num;
17
18            num = Integer.parseInt(cadena);
19
20            System.out.println("El número es " + num);
21        }
22
23    }
```

Salida:

```
run:
Exception in thread "main" java.lang.NumberFormatException: For input string: "56s"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Introducción

- ***Ejemplo 3***

En este ejemplo vamos a forzar una excepción de límites del vector, para ello vamos a crear un vector e intentar acceder a una posición que no existe:

```
12    public class Ejemplos_excepciones {
13
14        public static void main(String[] args) {
15            int v[] = {1,2,3};
16            int elem;
17
18            elem = v[5];
19
20            System.out.println("El elemento es " + elem);
21
22        }
23
24    }
```

Salida:

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

2. Lanzar excepciones (throw)

- Un programador puede programar su código de forma que se lancen excepciones cuando se intente hacer algo incorrecto o inesperado (en ocasiones es recomendable). Por ejemplo, cuando los argumentos que se le pasan a un método no son correctos o no cumplen ciertos criterios. En POO lo veremos con los casos adecuados.

Cómo lanzar una excepción

Para lanzar la excepción se utiliza la palabra reservada **throw** seguido de un objeto de tipo *Exception* (o alguna de sus subclases como *ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException, etc.*). Como las excepciones son objetos, deben instanciarse con "new". Por lo tanto, **podemos lanzar una excepción genérica así**:

```
throw new Exception();
```

Esto es equivalente a primero instanciar el objeto Exception y luego lanzarlo:

```
Exception e = new Exception();
throw e;
```

2. Lanzar excepciones (throw)

- El constructor de Exception permite (opcionalmente) un argumento String para dar detalles sobre el problema. Si la excepción no se maneja y el programa se para, el mensaje de error se mostrará por la consola (esto es muy útil para depurar programas).

```
throw new Exception("La edad no puede ser negativa");
```

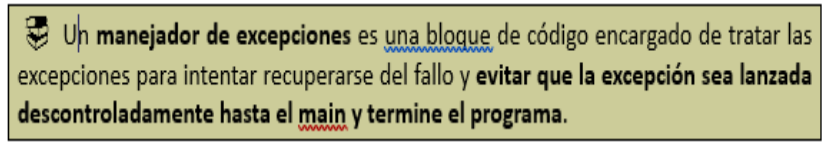
- En lugar de lanzar excepciones genéricas (Exception) también es posible lanzar excepciones específicas de Java como por ejemplo ArrayIndexOutOfBoundsException, *ArithmeticException, NumberFormatException, etc.* En Java todas las clases de excepciones heredan de Exception.

```
throw new NumberFormatException("...");
```

3. MANEJAR EXCEPCIONES (TRY – CATCH – FINALLY)

En Java se pueden manejar excepciones utilizando tres mecanismos llamados **manejadores de excepciones**. Existen tres y funcionan conjuntamente:

- Bloque **try** (intentar): código que podría lanzar una excepción.
- Bloque **catch** (capturar): código que manejará la excepción si es lanzada.
- Bloque **finally** (finalmente): código que se ejecuta tanto si hay excepción como si no.




```
try {
        // Instrucciones que podrían lanzar una excepción.
}
catch (TipoExcepción nombreVariable) {
        // Instrucciones que se ejecutan cuando 'try' lanza una excepción.
}
finally {
        // Instrucciones que se ejecutan tanto si hay excepción como si no.
}
```

3. MANEJAR EXCEPCIONES (TRY – CATCH – FINALLY)

- Vamos a producir una excepción y tratarla haciendo uso de los manejadores *try-catch:*

```
12  public class Ejemplos_excepciones {
13
14      public static void main(String[] args) {
15          int x = 1, y = 0;
16
17          try
18          {
19              int div = x / y;
20
21              System.out.println("La ejecución no llegará aquí.");
22          }
23          catch(ArithmeticException ex)
24          {
25              System.out.println("Has intentado dividir entre 0");
26          }
27
28          System.out.println("Fin del programa");
29      }
30  }
```

```
run:
Has intentado dividir entre 0
Fin del programa
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. PARTICULARIDADES DE LA CLÁUSULA CATCH

- Es importante entender que **el bloque catch solo capturará excepciones del tipo indicado.** Si se produce una excepción distinta no la capturará. Sin embargo **capturará excepciones heredadas del tipo indicado**. Por ejemplo, *catch (ArithmeticException e)* capturará cualquier tipo de excepción que herede de ArithmeticException. El caso más general es *catch (Exception e)* que capturará todo tipo de excepciones porque **en Java todas las excepciones heredan de Exception**.
- Sin embargo, es mejor utilizar excepciones lo más cercanas al tipo de error previsto, ya que lo que se pretende es recuperar el programa de alguna condición de error y si "se meten todas las excepciones en el mismo saco", seguramente habrá que averiguar después qué condición de error se produjo para poder dar una respuesta adecuada.
- **El objetivo de una cláusula *catch* es resolver la condición excepcional para que el programa pueda continuar como si el error nunca hubiera ocurrido.**

Cláusulas catch múltiples

```
try {
        // instrucciones que pueden producir distintos tipos de Excepciones
    }
    catch (TipoExcepción1 e1) {
        // instrucciones para manejar un TipoExcepción1
    }
    catch (TipoExcepción2 e2) {
        // instrucciones para manejar un TipoExcepción2
    }
    ...
    }
    catch (TipoExcepciónN eN) {
        // instrucciones para manejar un TipoExcepciónN
    }
    finally { // opcional
        // instrucciones que se ejecutarán tanto si hay excepción como si no
    }
```

```java
public class Ejemplos_excepciones {

    public static void main(String[] args) {
        int x, y, div, pos;
        int[] v = {1,2,3};
        Scanner in = new Scanner(System.in);

        try
        {
            System.out.print("Introduce el numerador: ");
            x = in.nextInt();

            System.out.print("Introduce el denominador: ");
            y = in.nextInt();

            div = x / y;

            System.out.println("La división es " + div );

            System.out.print("Introduce la posición del vector a consultar: ");
            pos = in.nextInt();

            System.out.println("El elemento es " + v[pos] );

        }
        catch(ArithmeticException ex)
        {
            System.out.println("División por cero: " + ex);
        }
        catch(ArrayIndexOutOfBoundsException ex)
        {
            System.out.println("Sobrepasado el tamaño del vector: " + ex);
        }

        System.out.println("Fin del programa");
    }
}
```

Pueden suceder tres cosas diferentes:

- El *try* se ejecuta sin excepciones, se ignoran los *catch* y se imprime “Fin del programa”.
- Se produce la excepción de división por cero (línea 29), el flujo de ejecución salta al 1er catch, se imprime el mensaje “División por cero...” y luego “Fin del programa”.
- Se produce la excepción de sobrepasar el vector (línea 36), el flujo de ejecución salta al 2º catch, se imprime el mensaje “Sobrepasado el tamaño del vector...” y “Fin del programa”.

5. EL OBJETO Exception

- Toda excepción genera un objeto de la clase Exception (o uno más específico que hereda de Exception). Dicho objeto contendrá detalles sobre el el error producido. Puede ser interesante mostrar esta información para que la vea el usuario (que sepa qué ha sucedido) o el desarrollador (para depurar y corregir el código si es pertinente). En la cláusula catch tenemos acceso al objeto en caso de que queramos utilizarlo:
- Los dos métodos de Exception más útiles son:

getMessage() → Devuelve un String con un texto simple sobre el error.

printStackTrace() → Es el que más información proporciona. Indica qué tipo de Excepción se ha producido, el mensaje simple, y también toda la pila de llamadas. Esto es lo que hace Java por defecto cuando una excepción no se maneja y acaba parando el programa.

```
...
catch (Exception e){

        // Mostramos el mensaje de la excepción
        System.err.println("Error: " + e.getMessage());

        // Mostramos toda la información, mensaje y pila de llamadas
        e.printStackTrace();
}
...
```

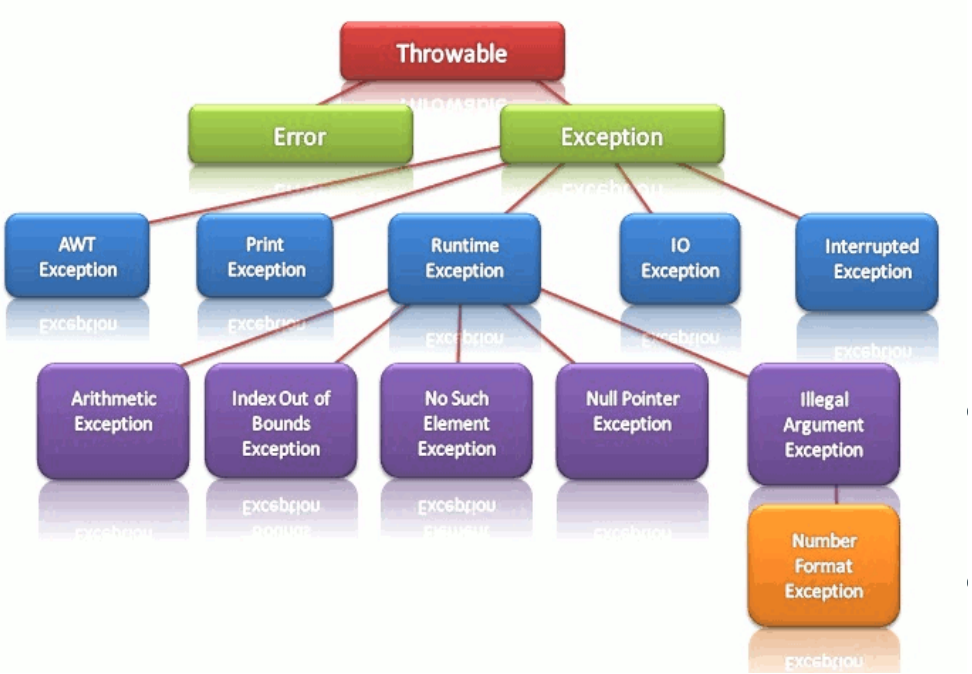
Los objetos de tipo Exception tienen sobrecargado el método toString() por lo que también es posible imprimirlos directamente mediante println().

```
...
catch (Exception e){

        // Mostramos el mensaje de la excepción
        System.out.println(e);

}
...
```

6. Jerarquía y tipos de Excepciones en Java



- Como *java.lang* es importado de forma implícita en todos los programas, la mayor parte de las excepciones derivadas de *RuntimeException* están disponibles de forma automática. Además **no es necesario incluirlas en ninguna cabecera de método mediante *throws***.

Las Excepciones pueden ser comprobadas y no comprobadas:

Excepciones comprobadas: aquellas que Java comprueba durante la compilación, antes de la ejecución del programa.

Excepciones no comprobadas: aquellas que Java no puede comprobar durante la compilación y se producirán durante la ejecución del programa.

6. Jerarquía y tipos de Excepciones en Java

Las **excepciones comprobadas** definidas en *java.lang* son:

Excepción	Significado
ClassNotFoundException	No se ha encontrado la clase.
CloneNotSupportedException	Intento de duplicado de un objeto que no implementa la interfaz clonable.
IllegalAccessException	Se ha denegado el acceso a una clase.
InstantiationException	Intento de crear un objeto de una clase abstracta o interfaz.
InterruptedException	Hilo interrumpido por otro hilo.
NoSuchFieldException	El campo solicitado no existe.
NoSuchMethodException	El método solicitado no existe.

6. Jerarquía y tipos de Excepciones en Java

Las **subclases de *RuntimeException* no comprobadas son:**

Excepción	Significado
AritmeticException	Error aritmético como división entre cero.
ArrayIndexOutOfBoundsException	Índice de la matriz fuera de su límite.
ArrayStoreException	Asignación a una matriz de tipo incompatible.
ClassCastException	Conversión invalida.
IllegalArgumentException	Uso inválido de un argumento al llamar a un método.
IllegalMonitorStateException	Operación de monitor inválida, como esperar un hilo no bloqueado.
IllegalStateException	El entorno o aplicación están en un estado incorrecto.
IllegalThreadStateException	La operación solicitada es incompatible con ell estado actual del hilo.
IndexOutOfBoundException	Algún tipo de índice está fuera de su rango o de su límite.
NegativeArraySizeException	La matriz tiene un tamaño negativo.
NullPointerException	Uso incorrecto de una referencia NULL.
NumberFormatException	Conversión incorrecta de una cadena a un formato numérico.

6. Jerarquía y tipos de Excepciones en Java

SecurityException	Intento de violación de seguridad.
StringIndexOutBounds	Intento de sobrepasar el límite de una cadena.
TypeNotPresentException	Tipo no encontrado.
UnsupportedOperationException	Operación no admitida.

7. Definir excepciones propias

- Cuando desarrollamos software, sobre todo al desarrollar nuestras propias clases, es habitual que se puedan producir excepciones que no estén definidas dentro del lenguaje Java. Para crear una excepción propia tenemos que definir una clase derivada de la clase base *Exception*.

El uso de esta nueva excepción es el mismo que hemos visto.

Ejemplo

Vamos a ver un ejemplo sencillo de definición y uso de un nuevo tipo de excepción llamada ExcepcionPropia que utilizaremos cuando a se le pase a 'método' un valor superior a 10.

El main y el método que lanza la excepción:

```
14  public class Ejemplos_excepciones {
15
16      public static void main(String[] args) {
17          try
18          {
19              metodo(1);
20              metodo(20);
21          }
22          catch (ExcepcionPropia e)
23          {
24              System.out.println("capturada :" + e);
25          }
26      }
27
28      static void metodo(int n) throws ExcepcionPropia
29      {
30          System.out.println("Llamado por metodo(" + n +")");
31
32          if (n > 10)
33              throw new ExcepcionPropia(n);
34
35          System.out.println("Finalización normal");
36      }
37  }
```

La definición de la nueva excepción.

```
12    public class ExcepcionPropia  extends Exception
13    {
          private int num;
15
16        ExcepcionPropia(int n)
17        {
18            this.num = n;
19        }
          public String toString()
21        {
22            return "Excepcion Propia[" + this.num + "]";
23        }
24
25    }
```

- Salida:

```
run:
Llamado por metodo(1)
Finalización normal
Llamado por metodo(20)
capturada :Excepcion Propia[20]
BUILD SUCCESSFUL (total time: 0 seconds)
```

1. Ejemplo

1. Crear una clase que provoque una excepción, ejemplo

```
public class Clase1
{
        public static void main(String arg[])
         {
              int [] array = new int[4];
              array[-2] = 8;
        }
}
```

2. ¿Qué ocurre al ejecutar?

```
 Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
                at Try1.main(Try1.java:6)
```

```
public class Try2
{
        public static void main(String arg[])
        {
                int [] array = new int[20];
                try
                 {
                      array[-3] = 24;
                }
                catch(ArrayIndexOutOfBoundsException excepcion)
                 {
                      System.out.println(" Error de índice en un array");
                }
        }
}
```

1. Practica

1. Modifica la clase anterior y fuerza una división por 0:
    1. ¿Qué error aparece?
    2. Escribe el código try catch para controlarlo.

1. Práctica

1. Modifica la clase anterior y fuerza una división por 0:
    1. ¿Qué error aparece?
    2. Escribe el código try catch para controlarlo.

```
public class Clase1
{
        public static void main(String arg[])
        {
           int [] array = new int[20];
           try
            {
            //array[-3] = 24;
               int b = 0;
               int a = 23/b;
           } catch(ArrayIndexOutOfBoundsException excepcion)
           {
               System.out.println(" Error de índice en un array");
           } catch(ArithmeticException excepcion)
            {
               System.out.println(" Error de índice en un array");
             }
}}
```

Ejemplo

Una excepción se puede capturar de distintas maneras

1. Dentro de un bloque de código **try … catch** (como hemo hemos visto hasta ahora)

2. “lanzando” la excepción al bloque de código padre
   1. Por comportamiento no esperado del programa
   2. De forma intencionada

```
public class ExcepcionApp3 {
    public static void main(String[] args) {
    […]
    try{
        […]
        rango(numerador, denominador);
        […]
    }catch(ArrayIndexOutOfBoundsException ex){
            respuesta=“error";
    }catch(ExcepcionIntervalo ex){
        respuesta="División entre cero";
    }
    System.out.println(respuesta);}
```

```
static void rango(int num, int den) throws
        ArrayIndexOutOfBoundsException ,
ExcepcionIntervalo{
    if((num>100)||(den<-5)){
        throw new ExcepcionIntervalo(“Nos fuera de
rango");
} }}
```