5. Excepciones

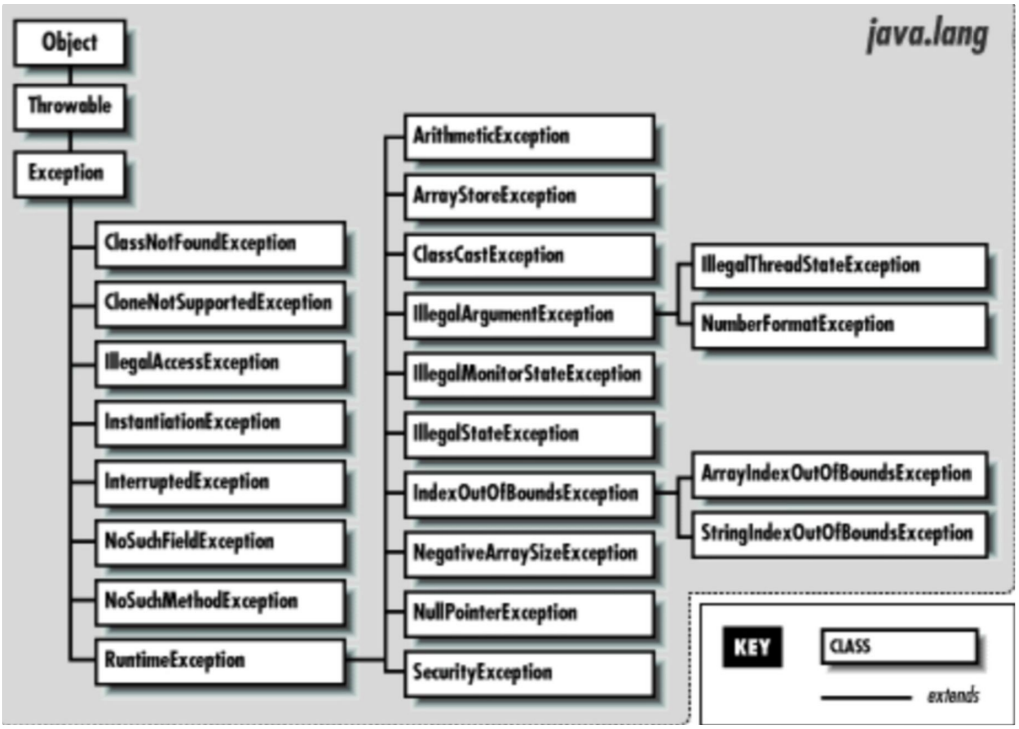
Date	Empty
Status	Not started
Type	Empty
Units	7. Uso avanzado de clases

Tratamiento de excepciones

Cuando se programa hay que tratar que nuestro código sea lo más robusto posible. Para ello hay que tener en cuenta que en determinada ocasiones puede fallar aun que en un primer instante este s compile perfectamente. Existen multitud de casos en los que esto puede pasar: intentar utilizar un objeto con valor nulo, intentar acceder a una posición de un array que no existe, intentar hacer una operación matemática que no tenga resultado, intentar acceder a un recurso que no existe, superar la memoria del ordenador, etc.... En la mayoría de los casos comentados, el programador puede crear un mecanísmo por el

cual se pueda actuar ante la situación para que el programa no pare de forma repentina. Es lo que se conoce como excepción.

Jerarquía y tipos de excepciones



Todas aquellas situaciones que extiendan de Exception son capturables (se debe crear un mecanismo de prevención de errores). Las principales excepciones son:

- ClassNotFoundExcepcion: producida cuando se intenta la carga de una clase que no se encuentra disponible.
- ArithmeticException: producida cuando se realiza una operación aritmética incorrecta
- ArrayStoreExceptio: producida cuando se utiliza de forma incorrecta el almacenamiento en un array
- ClassCastException: producida cuando un objeto de una clase se intenta castear a otro tipo incompatible
- IndexOutBoundException: producida cuando se intenta acceder a posiciones de un array que no existen
- NumberFormatException: producida al pasar una cadena de caracteres a números
- NullPointerExcepcion: producida cuando se utiliza un objeto que tiene el valor nulo
- IOEXception: producida cuando se hace un uso no correcto de dispositivos de entrada / salida. Pe filenotfound, eof,

Estas son las principales excepciones, existiendo más y cuyo tratamiento es muy recomendable. Incluso un programador puede crear sus propias excepciones según necesidad, extendiendo de la clase Exception como se verá más adelante

Captura o lanzamiento de excepciones

Para poder capturar excepciones se utiliza el bloque try / catch / finally (opcional)

```
public class Entrada { public static void main(String[] args)
{ String palabra = null; System.out.println(palabra.length
()); } }
```

En código anterior provoca un error al intentar acceder a una propiedad de un objeto que es nulo. El programa se para de porfía automática. Para tratar la excepción se utiliza el bloque try / catch

```
public class Entrada { public static void main(String[] args)
{ String palabra = null; try{ // parte donde aparece el códig
o que se quiere ejecutar y puede probocar fallo System.out.pr
intln(palabra.length()); } catch (NullPointerException e){ //
parte donde aparecen las ejecuciones que tendrán lugar cuando
se produzca una exceción de tipo nullpointer System.out.print
ln("objeto nulo"); } finally { // parte de código opcional qu
e será ejecutada siempre System.out.println("Siempre ejecutad
o"); } } }
```

Otros ejemplos sería

```
public class Entrada { public static void main(String[] args)
{ String[] palabras = new String[]{"uno","dos","tres"}; try{
for(int i=0;i<=3;i++){ System.out.println(palabras[i]); } } c
atch (ArrayIndexOutOfBoundsException e){ System.out.println("
posición inaccesible"); } finally { System.out.println("Siemp
re ejecutado"); } } }
```

```
public class Entrada { public static void main(String[] args)
```

```
{ try{ Int division = 4/0 } catch (ArithmeticException e){ Sy
stem.out.println("no se puede hacer"); } finally { System.ou
t.println("Siempre ejecutado"); } } }
```

Con la jerarquía de antes, todos las excepciones se podrían juntar en un solo tipo:

```
public class Entrada { public static void main(String[] args)
{ String palabra = null; String[] palabras = new String[]{"un
o","dos","tres"}; try{ System.out.println(palabra.length());
int division = 4/0; for(int i=0;i<=3;i++){ System.out.println
(palabras[i]); } } catch (Exception e){ System.out.println("E
rror producido"); } finally { System.out.println("Siempre eje
cutado"); } } }
```

Lo malo que tiene esto es que solo se detectaría el primer error, ya que al detectar el error producido en la línea

```
System.out.println(palabra.length());
```

El resto no los podría ejecutar. Para ello lo que se hace es un pila de excepciones, pudiendo tratar cada una de ellas de forma individual

```
public class Entrada { public static void main(String[] args)
{ String palabra = null; String[] palabras = new String[]{"un
o","dos","tres"}; try{ System.out.println(palabra.length());
int division = 4/0; for(int i=0;i<=3;i++){ System.out.println
(palabras[i]); } } catch (NullPointerException e){ System.ou
t.println("objeto nulo"); } catch (ArithmeticException e){ Sy
stem.out.println("operación incorrecta"); } catch (ArrayIndex
OutOfBoundsException e){ System.out.println("recorrido incorr
ecto"); } finally { System.out.println("Siempre ejecutado");
} } }
```

En este ejemplo se está haciendo todo desde el main, pero se podría hacer desde cualquier método ubicado en una clase.

```
package github.excepciones; public class TratamientoEx { publ
ic void excecionNull() { try { String palabra = null; System.
```

```
out.println(palabra.length()); } catch (NullPointerException
e) { System.out.println("Objeto nulo"); } } public void excec
ionArray() { String[] palabras = new String[]{"uno", "dos", "
tres"}; try { for (int i = 0; i <= 3; i++) { System.out.print
ln(palabras[i]); } } catch (ArrayIndexOutOfBoundsException e)
{ System.out.println("posición incorrecta"); } } public void
excecionOperacion() { try { int division = 4 / 0; } catch (Ar
ithmeticException e) { System.out.println("operación incorrec
ta"); } } }
```

La cual puede ser llamada desde el main

```
package github.excepciones; public class EntradaEx { public s
tatic void main(String[] args) { TratamientoEx tratamientoEx
= new TratamientoEx(); tratamientoEx.excecionOperacion(); tra
tamientoEx.excecionNull(); tratamientoEx.excecionArray(); } }
```

Otra forma de tratar las excepciones es utilizar el lanzamiento de las mismas por parte del método donde se produce la excepción. Para ellos en el método donde se ejecutan las líneas que piden fallar se acompaña la palabra reservada throws y el nombre de la Exception correspondiente

```
package github.excepciones; public class LanzarEx { public vo
id excecionNull() throws NullPointerException { String palabr
a = null; System.out.println(palabra.length()); } public void
excecionArray() throws ArrayIndexOutOfBoundsException { Strin
g[] palabras = new String[]{"uno", "dos", "tres"}; for (int i
= 0; i <= 3; i++) { System.out.println(palabras[i]); } } publ
ic void excecionOperacion() throws ArithmeticException { int
division = 4 / 0; } }
```

Esto no evita que se ejecute un bloque try / catch, sino que obliga a la clase donde se llamen estos métodos a capturar la excepción

```
package github.excepciones; public class EntradaTh { public s
tatic void main(String[] args) { LanzarEx lanzarEx = new Lanz
arEx(); try { lanzarEx.excecionOperacion(); } catch (Arithmet
icException e) { System.out.println("operación incorrecta");
} try { lanzarEx.excecionNull(); } catch (NullPointerExceptio
```

```
n e) { System.out.println("objeto nulo"); } try { lanzarEx.ex
cecionArray(); } catch (ArrayIndexOutOfBoundsException e) { S
ystem.out.println("posición incorrecta"); } } }
```

Existe la posibilidad de en la clase donde se utilizan los métodos que han utilizado un throws, también poner la palabra throws en la firma, postergando así la captura de la excepción. Lo malo que tiene esto es que si la excepción pasa a ser lanzada en un método main el programa no será capaz de manejar la excepción

Creación de excepciones personalizadas

Del mismo modo que muchas de las operaciones que se realizan en java pueden producir excepciones propias del lenguaje, un programador puede crear las suyas propias y ser ejecutadas cuando se quiera. Para ello simplemente hay que crear una clase que herede de Exception sobreescribiendo el constructor con un mensaje que será el que se

```
package github.excepciones; public class MiExcepcion extends
Exception { public MiExcepcion(String message) { super(messag
e); } }
```

Cuando esta excepción quiera ser lanzada hay que utilizar la palabra reservada throw

```
package github.excepciones; public class LanzarEx { public vo
id lanzarMiExcepcion() throws MiExcepcion { throw new MiExcep
cion("Excepción personalizada lanzada"); } }
```

Y tratada en el método main

```
package github.excepciones; public class EntradaTh { public s
tatic void main(String[] args) { LanzarEx lanzarEx = new Lanz
arEx(); try { lanzarEx.lanzarMiExcepcion(); } catch (MiExcepc
ion miExcepcion) { System.out.println(miExcepcion.getMessage
()); } } }
```