

# Tratamiento de Excepciones

---

## Tratamiento de Excepciones

---

### 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.

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.

En el momento en que es lanzada una excepción, la máquina virtual Java recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada. Para ello, comienza examinando el método donde se ha

producido la excepción; si este método no es capaz de tratarla, examina el método desde el que se realizó la llamada al método donde se produjo la excepción y así sucesivamente hasta llegar al último de ellos. En caso de que ninguno de los métodos de la pila sea capaz de tratar la excepción, la máquina virtual Java muestra un mensaje de error y el programa termina.

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.

### Ejemplo 1: División por cero.

Como primer encuentro con las excepciones, vamos a ejecutar el siguiente programa:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author jose
 */
class Division {
    public static void main(String argumentos[]) {
        int i=5, j=0;
        int k=i/j; // División por cero
    }
}
```

Produce la siguiente salida al ser ejecutado:

```
Output - Division (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Division.main(Division.java:13)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

Lo que ha ocurrido es que la máquina virtual Java ha detectado una condición de error y ha creado un objeto de la clase `java.lang.ArithmeticException`. Como el método donde se ha producido la excepción no es capaz de tratarla, es tratada por la

máquina virtual Java, que muestra el mensaje de error anterior y finaliza la ejecución del programa.

Existe toda una jerarquía de clases derivada de la clase base *Exception*. Estas clases derivadas se ubican en dos grupos principales:

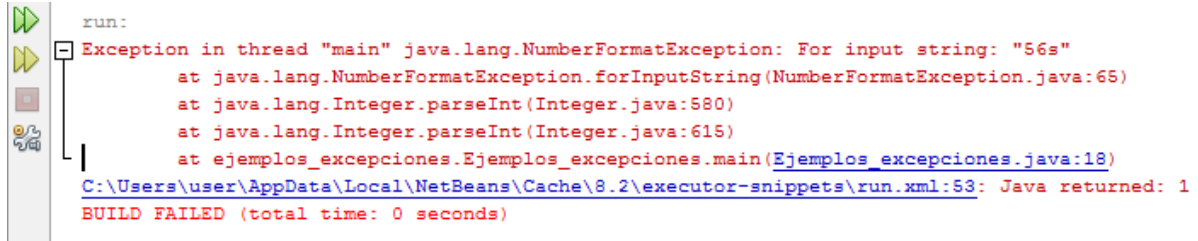
- Las excepciones en tiempo de ejecución ocurren cuando el programador no ha tenido cuidado al escribir su código. Por ejemplo, cuando se sobrepasa la dimensión de un array se lanza una excepción *ArrayIndexOutOfBoundsException*. Cuando se hace uso de una referencia a un objeto que no ha sido creado se lanza la excepción *NullPointerException*. Estas excepciones le indican al programador que tipos de fallos tiene el programa y que debe arreglarlo antes de proseguir.
- El segundo grupo de excepciones, es el más interesante, ya que indican que ha sucedido algo inesperado o fuera de control.

## Ejemplo 2: Excepción de conversión

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 }
```

Como se utilizan caracteres no numéricos la salida es la siguiente:



```
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)
```

Debido a que la cadena no tiene el formato adecuado ("56s" no representa un número válido), el método `Integer.parseInt(...)` no puede convertirla a un valor de tipo `int` y lanza la excepción *NumberFormatException*. La máquina virtual Java finaliza el programa en la línea 18 y muestra por pantalla la información sobre la excepción que se ha producido.

### Ejemplo 3: Excepción de objeto no inicializado

Habitualmente, cuando llamamos desde un objeto no inicializado, a una función miembro.

```
public static void main(String[] args) {
    String str;
    str.length();
    //...
}
```

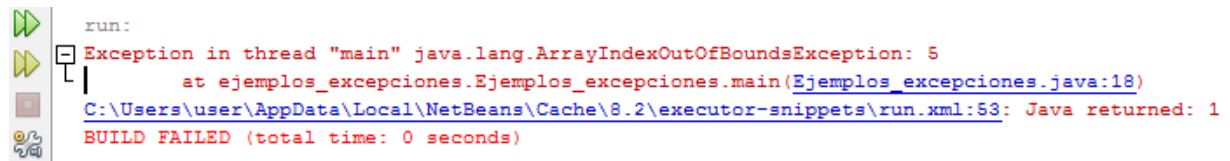
El compilador se queja con el siguiente mensaje "variable str might not have been initialized". En otras ocasiones, se lanza una excepción del tipo *NullPointerException*.

### Ejemplo 4: acceso a un vector

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 }
```

Siendo la 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)
```

Al intentar acceder a una posición que sobrepasa el tamaño del vector se produce una excepción de tipo *ArrayIndexOutOfBoundsException*. La máquina virtual de java finaliza el programa en la línea 18 y muestra el mensaje de error sobre la excepción que se ha producido.

## Excepciones estándar

---

### ¿Por qué lanzar excepciones?

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.

Esto es habitual en la Programación Orientada a Objetos (POO): Recordad que una clase debe ser la responsable de la lógica de sus objetos: asegurar que los datos sean válidos, además de controlar qué está permitido y no está permitido hacer. Por ejemplo si se instancia una clase con valores incorrectos como un objeto Persona con un DNI no válido, una edad negativa, una cuenta bancaria con saldo negativo, etc. En esos casos es conveniente que el constructor lance una excepción.

También puede ser apropiado lanzar excepciones en los setters si el valor no es válido, y en cualquier otro método en el que se intente hacer algo no permitido o que viole la integridad del objeto como por ejemplo retirar dinero de una cuenta sin saldo suficiente.

Téngase en cuenta que las excepciones pueden manejarse y controlarse sin que el programa se pare (se explicará cómo).

Es decir, lanzar una excepción no implica necesariamente que el programa terminará, es simplemente que es una forma de avisar de un error. Quien llame al método es responsable de manejar la excepción para que el programa no se pare.

### ¿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;
```

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("...");
```

En lugar de lanzar excepciones propias de Java (ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException, etc.) normalmente es preferible lanzar excepciones genéricas ‘Exception’ o mejor aún, utilizar nuestras propias excepciones.

## Indicar la excepción en la cabecera del método.

Es obligatorio indicar en la cabecera del método que puede lanzar excepciones. Para ello hay que añadir, a la derecha de la cabecera y antes de las llaves, la palabra reservada `throws` seguido del tipo de excepción que puede lanzar (si puede lanzar distintos tipos de excepciones deben indicarse todas separadas por comas).

Por ejemplo, veamos el método `setEdad(int edad)` de la clase `Persona`. Como hemos decidido que la edad de una persona no puede ser negativa, lanzaremos una excepción si `edad < 0`.

```
// Setter del atributo edad. Debe ser >= 0
public void setEdad(int edad) throws Exception {
    if (edad < 0)
        throw new Exception("Edad negativa no permitida");
    else
        this.edad = edad;
}
```

Hay que tener en cuenta que al lanzar una excepción se parará la ejecución de dicho método (no se ejecutará el resto del código del método) y se lanzará la excepción al método que lo llamó. Si por ejemplo desde la función `main` llamamos a `setEdad()`, como puede suceder que `setEdad()` lance una excepción, entonces en la práctica es posible que el `main` lance una excepción (no directamente con un `throw`, sino por la excepción que nos lanza `setEdad()`). Por lo tanto, también tenemos que especificar en el `main` que se puede lanzar una excepción:

```
public static void main(String[] args) throws Exception {
    Persona p = new Persona("44193900L", "Pepito", 27);
    ...
    ...
    p.setEdad(valor); // Puede lanzar una excepción
}
```

## Lanzando distintos tipos de excepciones



Un método puede lanzar distintos tipos de excepciones (si lo consideramos necesario). En tal caso hay que especificar todos los tipos posibles en la cabecera, separados por comas. Por ejemplo, imaginemos que el constructor de *Persona* toma como argumentos el dni y la edad, y queremos lanzar excepciones distintas según cada caso.

```
public Persona(String dni, int edad) throws InvalidDniException, InvalidEdadException {
    if (!dni.matches("[0-9]{8}[A-Z]")) {
        throw new InvalidDniException("DNI no válido: " + dni);
    }
    if (edad < 0) {
        throw new InvalidEdadException("Edad no válida: " + edad);
    }
    this.dni = dni;
    this.edad = edad;
}
```

Ojo, téngase en cuenta que *InvalidEdadException* e *InvalidDniException* no son tipos de excepciones de Java, sino excepciones propias que nosotros habríamos definido a medida para nuestro software.

## Las excepciones se lanzan de método a método (la patata caliente)

Obsérvese que el lanzamiento de excepciones se produce recursivamente a través de la secuencia de llamadas a métodos. Imaginemos que en el método A llamamos al método B, que llama al método C, etc. hasta llegar a E.

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$  (secuencia de llamadas de métodos)

Si el método E lanza una excepción, esta le llegará a D que a su vez se la lanzará a C, etc. recorriendo el camino inverso hasta llegar al método inicial A.

$A \leftarrow B \leftarrow C \leftarrow D \leftarrow E$  (secuencia de lanzamiento de excepciones)

Por lo tanto, como todos estos métodos pueden acabar lanzando una excepción, en sus cabeceras habrá que incluir el `throws Exception` (o el que corresponda según el tipo de excepción).

¡Ojo! No es recomendable dejar que las Excepciones de nuestro software lleguen de forma descontrolada hasta el main y terminen el programa. Lo ideal es manejar las excepciones como veremos en el siguiente apartado.

# Tratamiento

## Manejadores de excepciones

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.

Un manejador de excepciones es una bloque de código encargado de tratar las excepciones para intentar recuperarse del fallo y evitar que la excepción sea lanzada descontroladamente hasta el main() y termine el programa.

Siempre que se utilice un try es obligatorio utilizar al menos un catch. El finally es opcional.

```
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.  
}
```

El bloque try intentará ejecutar el código. Si se produce una excepción se abandona dicho bloque (no se ejecutarán las demás instrucciones del try) y se saltará al bloque catch. Lógicamente, si en el try no se produce ninguna excepción el bloque catch se ignora.

El bloque catch capturará las excepciones del tipo TipoExcepción, evitando que sea

lanzada al método que nos llamó. Aquí deberemos escribir las instrucciones que sean necesarias para manejar el error. Pueden especificarse varios bloques catch para distintos tipos de excepciones.

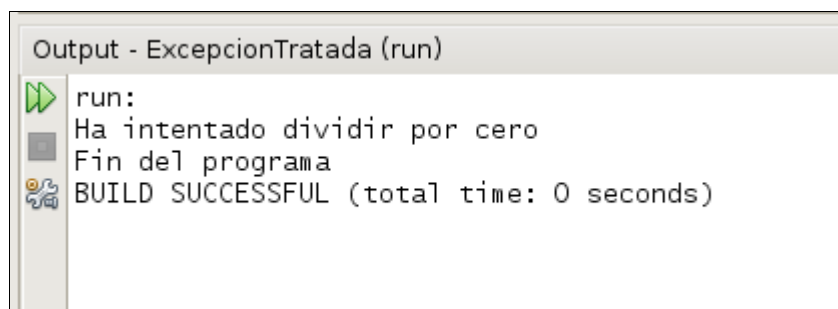
El bloque finally es opcional y se ejecutará tanto si se ha lanzado una excepción como si no.

Veamos un ejemplo sencillo.

Ejemplo:

```
class ExcepcionTratada {  
    public static void main(String argumentos[]) {  
        int i=5, j=0;  
        try {  
            int k=i/j;  
            System.out.println("Esto no se va a ejecutar.");  
        }  
  
        catch (ArithmeticException ex) {  
            System.out.println("Ha intentado dividir por cero");  
        }  
        System.out.println("Fin del programa");  
    }  
}
```

Salida:



```
Output - ExcepcionTratada (run)  
run:  
Ha intentado dividir por cero  
Fin del programa  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Al intentar dividir por cero se lanza automáticamente una *ArithmeticException*. Como esto sucede dentro del bloque try, la ejecución del programa pasa al primer bloque catch porque coincide con el tipo de excepción producida (*ArithmeticException*). Se ejecutará el

código del bloque catch y luego el programa continuará con normalidad.

### 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

Se pueden especificar varias cláusulas catch, tantas como queramos, para que cada una capture un tipo diferente de excepción.

```
try {  
    // instrucciones  
}  
catch (TipoExcepción1  
    // instrucciones  
}  
catch (TipoExcepción2  
    // instrucciones  
}  
...
```

```
}  
catch (TipoExcepciónN  
    // instrucciones  
}  
finally { // opcional  
    // instrucciones  
}
```

Cuando se lanza una excepción dentro del try, se comprueba cada sentencia catch en orden y se ejecuta la primera cuyo tipo coincida con la excepción lanzada. Los demás bloques catch serán ignorados. Luego se ejecutará el bloque finally (si se ha definido) y el programa continuará su ejecución después del bloque try-catch-finally.

Si el tipo excepción producida no coincide con ninguno de los catch, entonces la excepción será lanzada al método que nos llamó.

## Ejercicio práctico

Vamos a ejecutar distintas instrucciones dentro del bloque try y manejaremos las excepciones de división por cero y de si sobrepasamos el tamaño del vector.

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         int x, y, div, pos;
18         int[] v = {1,2,3};
19         Scanner in = new Scanner(System.in);
20
21         try
22         {
23             System.out.print("Introduce el numerador: ");
24             x = in.nextInt();
25
26             System.out.print("Introduce el denominador: ");
27             y = in.nextInt();
28
29             div = x / y;
30
31             System.out.println("La división es " + div );
32
33             System.out.print("Introduce la posición del vector a consultar: ");
34             pos = in.nextInt();
35
36             System.out.println("El elemento es " + v[pos] );
37
38         }
39         catch(ArithmeticException ex)
40         {
41             System.out.println("División por cero: " + ex);
42         }
43         catch(ArrayIndexOutOfBoundsException ex)
44         {
45             System.out.println("Sobrepasado el tamaño del vector: " + ex);
46         }
47
48         System.out.println("Fin del programa");
49     }
50 }
```

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”.

## Tratamiento de Excepciones

- 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”.



# El objeto Exception

## 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 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
```

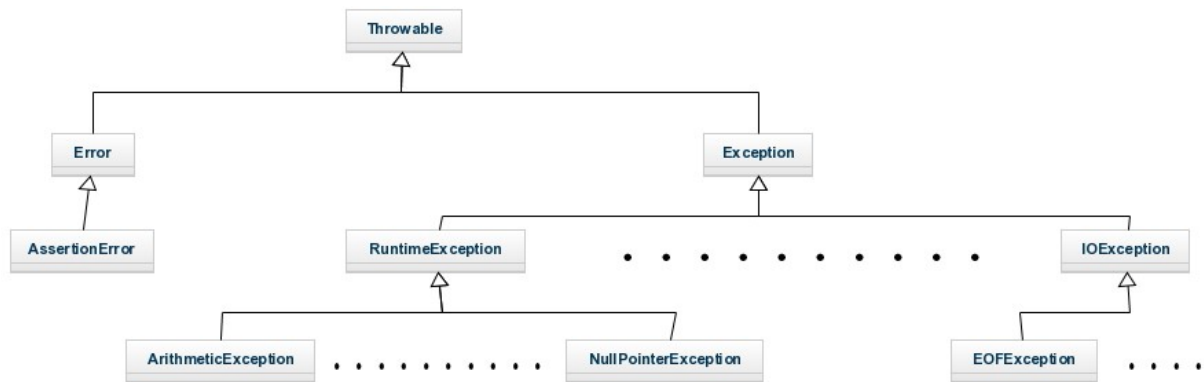
## Tratamiento de Excepciones

```
        System.out.println(e);  
    }  
    ...
```

# Jerarquía de Excepciones

## JERARQUÍA Y TIPOS DE EXCEPCIONES JAVA

La clase Exception hereda de Throwable, y a su vez, todas las excepciones heredan de Exception.



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.

Las excepciones comprobadas definidas en java.lang son:

- 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.

Las subclases de `RuntimeException` no comprobadas son:

- `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 el estado actual del hilo.
- `IndexOutOfBoundsException` 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.
- `SecurityException` Intento de violación de seguridad.
- `StringIndexOutOfBoundsException` Intento de sobrepasar el límite de una cadena.
- `TypeNotPresentException` Tipo no encontrado.
- `UnsupportedOperationException` Operación no admitida.

Es interesante conocer los distintos tipos de excepciones, pero no es necesario sabérselas de memoria ni entender exactamente cuándo se produce cada una.

Aunque las excepciones que incorpora Java, gestionan la mayoría de los errores más comunes, es probable que el programador prefiera crear sus propios tipos de excepciones para gestionar situaciones específicas de sus aplicaciones. Tan solo hay que definir una subclase de `Exception`, que naturalmente es subclase de `Throwable`.

# Throws

## aviso

Si un método es capaz de causar una excepción que él mismo no puede gestionar, ha de especificarse el comportamiento de manera que los métodos que llamen a ese primer método puedan protegerse contra esa excepción. Para ello se incluye una cláusula `throws` en la declaración de método.

Una cláusula `throws` da un listado de los tipos de excepciones que el método puede lanzar. Esto es necesario para todas las excepciones, excepto las del tipo `Error` o `RuntimeException`, o cualquiera de sus subclases. Si no se declaran se provoca un error de compilación. Esta es la forma general de una declaración de método que incluye una sentencia `throws`:

```
tipo nombre_método (argumentos) throws lista_excepciones
{
    // cuerpo del método
}
```

Las excepciones de la lista se separan por comas y son las excepciones que el método puede lanzar.

Por ejemplo:

```
// este programa contiene un error y no se compilará.
class ThrowsDemo{
    public static void main(String args[]){
        throwuno();
    }

    static void throwuno() {
        System.out.println("Estoy en throwuno");
        throw new IlegalAccesoExcepcion("Demostración");
    }
}
```

```
}  
}
```

Para que este programa funcione se deben realizar dos cambios:

1. El método `throwuno` debe declarar la excepción que puede lanzar.
2. En el programa principal debe haber un bloque `try/catch`.

```
/**  
 *  
 * @author jose  
 */  
// este programa ahora es correcto.  
  
// este programa ahora es correcto.  
class PreubaThrows{  
  
    public static void main(String args[]) {  
        try {  
            throwuno();  
        }  
        catch (IllegalAccessException e){  
            System.out.println("Excepción capturada." + e);  
        }  
    }  
  
    static void throwuno() throws IllegalAccessException {  
        System.out.println("Estoy en throwuno");  
        throw new IllegalAccessException("Demostración");  
    }  
}
```

La salida de este programa será:

```
run:  
Estoy en throwuno  
Excepción capturada.java.lang.IllegalAccessException: Demostración  
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 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`.

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

Siendo la salida:

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

Como se puede observar la excepción se lanza cuando el número es mayor que 10 y será tratada por la nueva clase creada que hereda de Exception. Además se sobrescribe el método `toString` que es el encargado de mostrar el mensaje asociado a la excepción.

Otro ejemplo:

Estudiaremos una situación en la que el usuario introduce un valor fuera de un determinado intervalo, el programa lanza un excepción, que vamos a llamar `ExcepcionIntervalo`.

```
public class ExcepcionIntervalo extends Exception{
    public ExcepcionIntervalo(String msg){
        super(msg);
    }
}
```

Para crear y lanzar una excepción propia tenemos que definir la clase *ExcepcionIntervalo* derivada de la clase base `Exception`.

La definición de la clase es muy simple. Se le pasa un `String msg`, que contiene un mensaje, en el único parámetro que tiene el constructor de la clase derivada y éste se lo



pasa a la clase base mediante `super`.

El método que puede lanzar una excepción

La función miembro que lanza una excepción tiene la declaración habitual que cualquier otro método pero se le añade a continuación la palabra reservada *throws* seguido de la excepción o excepciones que puede lanzar.

```
public static void rango(int num, int min, int max) throws ExcepcionIntervalo{
    if((num < min)|| (num > max)){
        throw new ExcepcionIntervalo("Número fuera del intervalo");
    }
}
```

Cuando `num` es menor a `min` o mayor que `max` se lanza (*throw*) una excepción, un objeto de la clase *ExcepcionIntervalo*. Dicho objeto se crea llamando al constructor de dicha clase y pasándole un string que contiene el mensaje "Número fuera del intervalo".

Captura de las excepciones

En el siguiente programa, añadimos la llamada a la función *rango* que verifica si el número está dentro del intervalo dado, y el bloque `catch` que captura la excepción que puede lanzar dicha función si el número no están en el intervalo especificado.

```
public static void main(String[] args) {
    // TODO code application logic here
    Scanner stdin = new Scanner(System.in);
    String str1, str2, respuesta;
    int num, min, max;

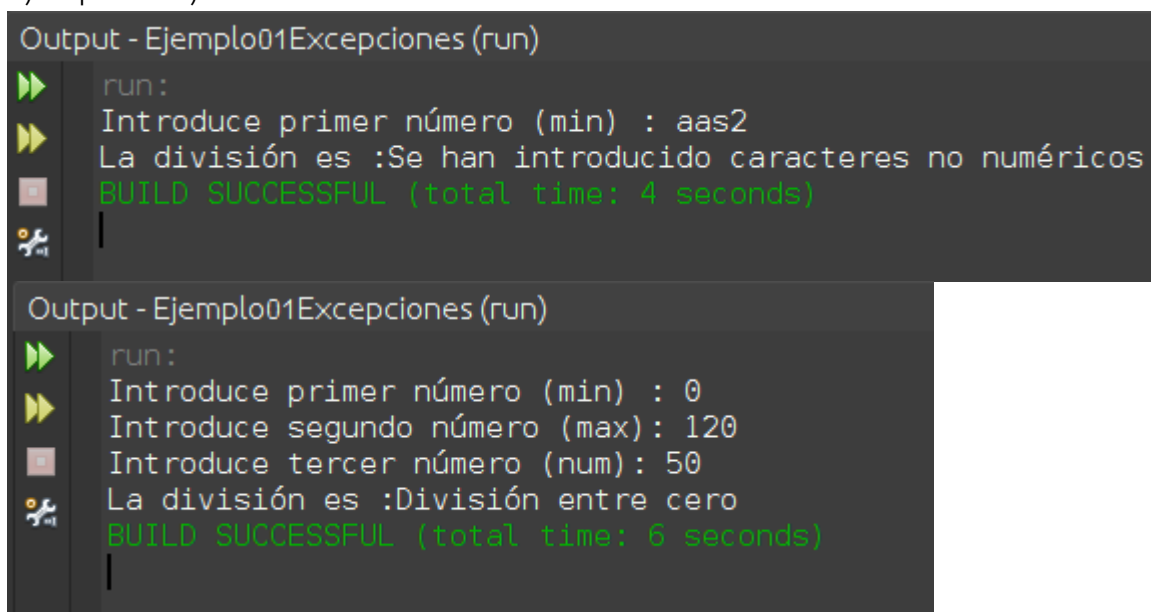
    try{
        // primer numero
        System.out.print("Introduce primer número (min) : ");
        str1=stdin.nextLine();
        min=Integer.parseInt(str1);
        // segundo número
```

## Tratamiento de Excepciones

```
        System.out.print("Introduce segundo número (max): ");
        str2=stdin.nextLine();
        max=Integer.parseInt(str2);
        // tercer número
        System.out.print("Introduce tercer número (num): ");
        str2=stdin.nextLine();
        num=Integer.parseInt(str2);

        rango(num, min, max);
        // Hacemos una operación de ejemplo
        num=(max/min)*num;
        // lo convertimos a String
        respuesta=String.valueOf(num);
    }
    catch(NumberFormatException ex){
        respuesta="Se han introducido caracteres no numéricos";
    }
    catch(ArithmeticException ex){
        respuesta="División entre cero";
    }
    catch(ExcepcionIntervalo ex){
        respuesta=ex.getMessage();
    }
    System.out.println("La división es :"+respuesta);
}
```

Ejemplo de ejecución:



```
Output - Ejemplo01Excepciones (run)
run:
Introduce primer número (min) : aas2
La división es :Se han introducido caracteres no numéricos
BUILD SUCCESSFUL (total time: 4 seconds)

Output - Ejemplo01Excepciones (run)
run:
Introduce primer número (min) : 0
Introduce segundo número (max): 120
Introduce tercer número (num): 50
La división es :División entre cero
BUILD SUCCESSFUL (total time: 6 seconds)
```

Output - Ejemplo01Excepciones (run)

```
run :  
Introduce primer número (min) : 20  
Introduce segundo número (max): 30  
Introduce tercer número (num): 50  
La división es :Número fuera del intervalo  
BUILD SUCCESSFUL (total time: 10 seconds)
```

## Ejercicio propuesto

---

Hay otra alternativa para el ejercicio que realiza la operación (esta operación solo es un ejemplo):

- Definir un método denominada *calcular*, que devuelva el resultado de la operación, cuando se le pasa los strings obtenidos de los respectivos controles de edición.
- El método *calcular*, convierte los Strings en números enteros, verifica el rango, calcula y devuelve el resultado.
- El método *calcular* puede lanzar, throws, tres tipos de excepciones. En el cuerpo de la función se crea e instancia (new), y se lanza (throw), explícitamente un objeto de la clase *ExcepcionIntervalo*, definida por el usuario, e implícitamente se crea y se lanza objetos de las clases *NumberFormatException* y *ArithmeticException* definidas en el lenguaje Java.
- La sentencia que llama al método *calcular* dentro del bloque try puede producir alguna de las tres excepciones que es capturada por el correspondiente bloque catch.

```
public class Ejemplo02Excepciones {  
    public static void main(String[] args) {  
  
        Scanner stdin = new Scanner(System.in);  
        String str1, str2, str3, respuesta;  
  
        try{  
            // primer numero  
            System.out.print("Introduce primer número (min) : ");  
            str1=stdin.nextLine();  
  
            // segundo número  
            System.out.print("Introduce segundo número (max): ");  
            str2=stdin.nextLine();  
  
            // tercer número  
            System.out.print("Introduce tercer número (num): ");  
            str3=stdin.nextLine();  

```

```
        respuesta = calcular(str3, str1, str2);
    }
    catch(NumberFormatException ex){
        respuesta="Se han introducido caracteres no numéricos";
    }
    catch(ArithmeticException ex){
        respuesta="División entre cero";
    }
    catch(ExcepcionIntervalo ex){
        respuesta=ex.getMessage();
    }
    System.out.println("El resultado es :"+respuesta);
}

public static String calcular(String str3, String str1, String str2)throws E
    String respuesta;
    int num=Integer.parseInt(str3);
    int min=Integer.parseInt(str1);
    int max=Integer.parseInt(str2);
    if((num < min)||(num > max)){
        throw new ExcepcionIntervalo("Número fuera del intervalo");
    }
    num=(max/min)*num;
    respuesta=String.valueOf(num);
    return (respuesta);
}
}
```

## Ejercicios

---

### IMPORTANTE

En estos ejercicios es fundamental hacer varias pruebas para comprobar y comprender qué sucede en cada caso (según el tipo de excepción, cuando no hay excepciones, etc.).

A no ser que se indique lo contrario, al lanzar una excepción deberás incluir un mensaje breve sobre el error (`new Exception("...")`), y cuando captures excepciones deberás mostrar la pila de llamadas (`printStackTrace()`).

Ejercicio 1: Implementa un programa que pida al usuario un valor entero `num` utilizando un `nextInt()` (de `Scanner`) y luego muestre por pantalla el mensaje “Valor introducido: ...”. Se deberá tratar la excepción `InputMismatchException` que lanza `nextInt()` cuando no se introduce un entero válido. En tal caso se mostrará el mensaje “Valor introducido incorrecto”.

Ejercicio 2: Implementa un programa que pida dos valores `int num1` y `num2` utilizando un `nextInt()` (de `Scanner`), calcule `num1/num2` y muestre el resultado por pantalla. Se deberán tratar de forma independiente las dos posibles excepciones, `InputMismatchException` y `ArithmeticException`, mostrando en cada caso un mensaje de error diferente en cada caso.

Ejercicio 3: Implementa un programa que cree un vector tipo `double` de tamaño 5 y luego, utilizando un bucle, pida cinco valores por teclado y los introduzca en el vector. Tendrás que manejar la/las posibles excepciones y seguir pidiendo valores hasta rellenar completamente el vector.

Ejercicio 4: Implementa un programa que cree un vector de enteros de tamaño `N` (número aleatorio entre 1 y 100) con valores aleatorios entre 1 y 10. Luego se le preguntará al usuario qué posición del vector quiere mostrar por pantalla, repitiéndose una y otra vez hasta que se introduzca un valor negativo. Maneja todas las posibles excepciones.

Ejercicio 5: Implementa un programa con tres funciones:

- `void imprimePositivo(int p)`: Imprime el valor `p`. Lanza una ‘Exception’ si `p < 0`
- `void imprimeNegativo(int n)`: Imprime el valor `n`. Lanza una ‘Exception’ si `p >= 0`

La función `main` para realizar pruebas. Puedes llamar a ambas funciones varias veces con

distintos valores, hacer un bucle para pedir valores por teclado y pasarlos a las funciones, etc. Maneja las posibles excepciones.

Ejercicio 6: Implementa una clase Gato con los atributos nombre y edad, un constructor con parámetros, los getters y setters, además de un método imprimir() para mostrar los datos de un gato.

- El nombre de un gato debe tener al menos 3 caracteres
- La edad no puede ser negativa.

Por ello, tanto en el constructor como en los setters, deberás comprobar que los valores sean válidos y lanzar una 'Exception' si no lo son.

Luego, haz una clase principal con main para hacer pruebas: instancia varios objetos Gato y utiliza sus setters, probando distintos valores (algunos válidos y otros incorrectos). Maneja las excepciones.

Ejercicio 7: Crea una copia del programa anterior y modifica el main para hacer lo siguiente:

- Crea un ArrayList<Gato>. Luego, utilizando un bucle, pide al usuario que introduzca los datos de 5 gatos: pide los datos, instancia el objeto y guárdalo en el ArrayList. Por último, imprime la información de los gatos.
- Maneja las posibles excepciones de modo que en el ArrayList solo almacenemos objetos Gato válidos y el bucle se repita hasta crear y almacenar correctamente 5 gatos.

Ejercicio 8: Vamos a mejorar el software del caso práctico del Banco de otra unidad añadiendo excepciones y alguna cosa más. Crea un nuevo proyecto y copia tu propio código de Banco. Realiza los siguientes cambios:

- Crea una nueva clase CuentaException que herede de Exception. La utilizaremos para lanzar excepciones relacionadas con cuentas bancarias.
- Crea una nueva clase AvisarHaciendaException que herede de Exception. La utilizaremos para lanzar una excepción cuando haya que avisar a hacienda.
- Modifica la clase CuentaBancaria:
  - Los movimientos deberán almacenarse en un ArrayList en lugar de en un vector. Ya no será necesario limitar a 100 el n.º de movimientos.
  - No se mostrará ningún tipo de mensaje de error. En su lugar, se lanzarán excepciones.

## Tratamiento de Excepciones

- Cuando se intente realizar algo incorrecto o no permitido se lanzará una excepción CuentaExcepción (deberá incluir un mensaje breve sobre el error producido).
- Cuando haya que avisar a hacienda se lanzará la excepción AvisarHaciendaException, que contendrá información sobre el titular, el iban y la operación realizada. Recuerda que aunque se avise a hacienda la operación debe realizarse de todos modos.
- Modifica la clase principal que contiene el main para manejar todas las posibles excepciones (no solo las de la clase CuentaBancaria), mostrando los mensajes de error oportunos y los printStackTrace().

## Actividad

Escribe un programa para realizar el pedido de pizzas a domicilio, con al menos los siguientes datos:

- Nombre del cliente.
- Apellidos del cliente.
- DNI del cliente.
- Nombre de la pizza:
  - Provenzal.
  - Barbacoa.
  - Mediterránea.
- Cantidad (numero entre 1 y 10).
- Extras (si o no).
  - Cantidad extra ingrediente 1 (numero entre 1 y 5).
  - Cantidad extra ingrediente 2 (numero entre 1 y 5).
  - Cantidad extra ingrediente 3 (numero entre 1 y 5).



Capturar todo tipo de posibles errores en la entrada de datos, de modo que una vez terminado de rellenar el formulario se muestren los datos introducidos a modo de resumen. Si ocurre un error, se muestra el error cometido y se rellena de nuevo el formulario.