CONTROL Y MANEJO DE EXCEPCIONES.

Una de las peores cosas que puede suceder en un programa es que se detenga de forma abrupta, a causa de un error no previsto.

Esto puede tener graves consecuencias, como, por ejemplo, pérdidas de información irreparables, debido a que en el momento en el que dejó de funcionar se encontraran ficheros abiertos, accesos a bases de datos en curso y otras situaciones similares que llevan a estados imprecisos.

Lejos de ignorar la posibilidad de que ocurra un error, un buen programa debe ser capaz de detectarlo y realizar las acciones oportunas para solucionar los problemas que surjan.

En esta unidad se estudiará cómo tratar las situaciones anormales, que llamaremos excepciones, más comunes en los programas. Estas situaciones deben poder ser recuperadas o el programa debe ser capaz de detectar e informar del problema.

1. ¿QUÉ ES UNA EXCEPCIÓN? CONCEPTO.

Una situación anormal es una excepción. En la ejecución de un programa pueden darse situaciones anormales. Un programa debe ser capaz de tratar las posibles situaciones anormales de manera inteligente y, si es posible, recuperarse de ellas.

Ejemplo: ¿Qué pasa cuando en un vector queremos acceder a una posición mayor a la permitida?

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

    int[] v = new int[3];
    System.out.println(v[10]);
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at excepciones.ErrorArray.main(ErrorArray.java:8)
```

¿Conocemos otras situaciones problemáticas?

- Cambio de tipos. Asumir que estamos ante un número y no ser así, por ejemplo, que el usuario introduce texto en lugar de números.

```
String str = "2g2";
int value_int = Integer.valueOf(str);
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "2g2"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.valueOf(Integer.java:766)
        at excepciones.ErrorArray.main(ErrorArray.java:11)
```

- Al procesar una operación por ejemplo dividir entre 0 que produce una situación anómala, pues no hay una respuesta correcta al poder representar el valor “infinito”.

```
int div = 5 / 0;
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at excepciones.ErrorArray.main(ErrorArray.java:13)
```

- No hay memoria disponible para asignar
- Leer por teclado un dato de un tipo distinto al esperado
- Error al abrir un fichero
- Problemas de Hardware

¿Qué realiza Java (y su manejador de excepciones) cuando suceden dichas situaciones anómalas?

- Muestra la descripción de la excepción.
- Muestra la traza de la pila de llamadas.
- Provoca el final del programa.

SOLUCIÓN:

- El control y manejo de excepciones permite la construcción de programas robustos y con capacidad de recuperación ante errores.
- Además, separan la lógica de manejo de errores de la lógica de ejecución normal de los programas, haciéndolo más legible.

¿Qué ocurre, en más detalle, cuando se produce una excepción?

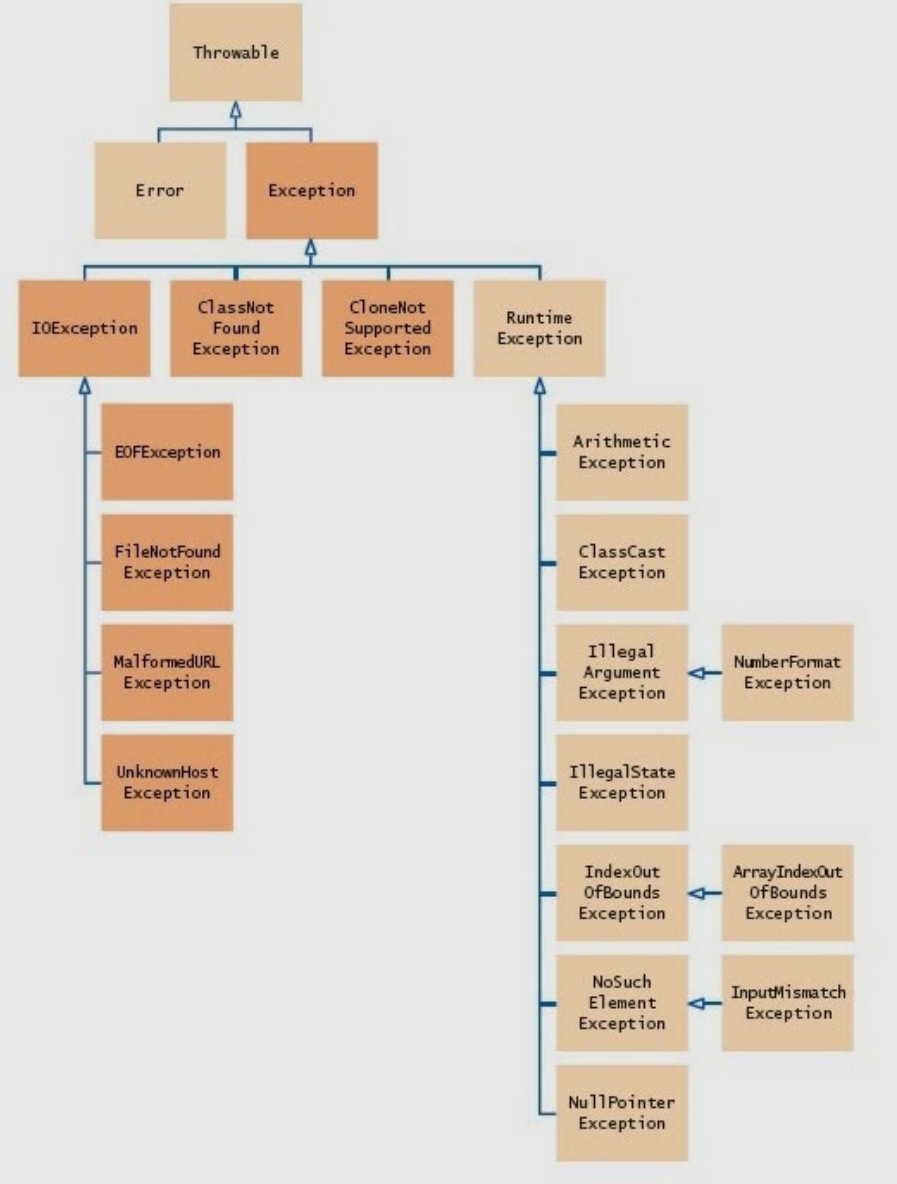
Cuando ocurre una excepción:

1. La Máquina Virtual Java crea un objeto excepción y lo lanza. El objeto excepción creado contiene información sobre el error. La ejecución normal del programa se detiene.
2. El sistema busca en el método donde se ha producido la excepción un manejador de excepciones que capture ese objeto y trate la excepción.
3. Si el método no contiene un manejador para la excepción se busca en el método que llamó a este y así sucesivamente en toda la pila de llamadas.
4. Cuando se encuentra un manejador apropiado se le pasa la excepción. Un manejador de excepciones es considerado apropiado si el tipo de objeto excepción lanzado es compatible al tipo que puede manejar.
5. Si no se encuentra un manejador adecuado la Máquina Virtual Java muestra el error y acaba el programa.

Así pues, Java tiene definida una jerarquía de clases para controlar una serie de excepciones predefinidas y permite la creación de nuevos tipos de excepciones.

2. JERARQUÍA DE EXCEPCIONES.

Todas las excepciones lanzadas automáticamente en un programa Java son objetos de la clase Throwable o de alguna de sus clases derivadas. La clase Throwable deriva directamente de Object y tiene dos clases derivadas directas: Error y Exception.



2.1. Clase ERROR

La clase Error está relacionada con errores de la máquina virtual de Java. Generalmente estos errores no dependen del programador por lo que no nos debemos preocupar por tratarlos, por ejemplo, OutOfMemoryError, StackOverflowError, errores de hardware, etc. este tipo de errores no se pueden tratar.

```
public class GenerarError {

    GenerarError newObj = new GenerarError();

    public GenerarError() {
        System.out.println("esto va a petar!");
    }

}
```

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

    GenerarError stackObjet = new GenerarError();

}
```

```
Exception in thread "main" java.lang.StackOverflowError
        at excepciones.GenerarError.<init>(GenerarError.java:5)
        at excepciones.GenerarError.<init>(GenerarError.java:5)
```

2.2. Clase Exception

En la clase Exception se encuentran las excepciones que se pueden lanzar en una aplicación. Tiene varias subclases entre ellas:

- RuntimeException: excepciones lanzadas durante la ejecución del programa. Por ejemplo: ArithmeticExcepcion, NullPointerException,ArrayIndexOutOfBoundsException, etc. Pertenecen al paquete java.lang.
- IOException: excepciones lanzadas al ejecutar una operación de entrada-salida. Pertenecen al paquete java.io.

```
try {
    BufferedReader br = Files.newBufferedReader(Paths.get("filename.txt"));
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

- ClassNotFoundException: excepción lanzada cuando una aplicación intenta cargar una clase, pero no se encuentra el fichero .class correspondiente.

```
java.lang.ClassNotFoundException: org.joda.time.ReadablePartial
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1858)
at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1709)
```

3. TRATAMIENTO Y MANEJO DE EXCEPCIONES

Como se ha explicado hasta ahora, el manejo de excepciones consiste detectar situaciones anormales. ¿Pero como podemos capturar/recuperar/tratar una excepción?

3.1. CAPTURA DE EXCEPCIÓN: BLOQUE TRY – CATH – FINALLY.

Un programa que trate las excepciones debe realizar los siguientes pasos:

1. Se intenta (try) ejecutar un bloque de código.
2. Si se produce una circunstancia excepcional se lanza (throw) una excepción. En caso contrario el programa sigue su curso normal.
3. Si se ha lanzado una excepción, la ejecución del programa es desviada al manejador de excepciones donde la excepción se captura (catch) y se decide qué hacer al respecto.

```
try{
    //Instrucciones que se intentan ejecutar, si se produce una
    //situación inesperada se lanza una excepción
}
catch(tipoExcepcion e){
    //Instrucciones para tratar esta excepción
}
catch(otroTipoExcepcion e){
    //Instrucciones para tratar esta excepción
}
    //Se pueden escribir tantos bloques catch como sean necesarios
finally{
    // instrucciones que se ejecutarán siempre después de un bloque try
    // se haya producido o no una excepción
}
```

Bloque try:

En el bloque try se encuentran las instrucciones correspondientes al código/algoritmo que puntualmente puede llevar a lanzar una excepción.

Bloque catch:

Es el bloque de código donde se captura la excepción. El bloque catch es el manejador (*handler*) de la excepción. Es decir, donde se decide qué hacer con la excepción capturada. Puede haber varios bloques catch relacionados con un bloque try.
¡Ojo! Una vez finalizado un bloque catch la ejecución no vuelve al punto donde se lanzó la excepción. La ejecución continúa por la primera instrucción a continuación de los bloques catch.

Bloque finally:

Es opcional. Debe aparecer a continuación de los bloques catch o a continuación de un bloque try si no hay bloques catch.
La ejecución de sus instrucciones queda garantizada independientemente de que se produzca una interrupción o se llegue o no a capturar. Incluído:

- Aunque el bloque try tenga una sentencia return, continue o break, se ejecutará el bloque finally
- Cuando se haya lanzado una excepción que ha sido capturada por un bloque catch. El finally se ejecuta después del catch correspondiente.
- Si se ha lanzado una excepción que no ha sido capturada, se ejecuta el finally antes de acabar el programa.

¿Cuál es su uso? Un bloque finally se usa para dejar un estado consistente después de ejecutar el bloque try. Un ejemplo de uso de bloques finally puede ser cuando estamos tratando con ficheros y se produce una excepción. Podemos escribir un bloque finally para cerrar el fichero. Este bloque se ejecutará siempre y se liberarán los recursos ocupados por el fichero.

Por lo tanto:

- Un bloque try puede estar seguido de varios bloques catch, tantos como excepciones diferentes queramos manejar.
- La excepción es capturada por el bloque catch cuyo argumento coincida con el tipo de objeto lanzado.
- La búsqueda de coincidencia se realiza sucesivamente sobre los bloques catch en el orden en que aparecen en el código hasta que aparece la primera concordancia.
- Cuando acaba la ejecución de un catch de este bloque, el programa continúa después del último de los catch que sigan al bloque try que lanzó la excepción.

Por este motivo, es importante el orden en que se coloquen los bloques catch.
Las excepciones más genéricas se deben capturar al final

3.2. ANIDAMIENTO DE BLOQUES TRY-CATCH.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n1, n2;
    try {
         System.out.print("Introduce un número: ");
         n1 = sc.nextInt();
         try {
              System.out.print("Introduce otro número: ");
              n2 = sc.nextInt();
              System.out.println(n1 + " / " + n2 + " = " + n1/(double)n2);
         }catch (InputMismatchException e) {
                sc.nextLine();
                n2 = 0;
                System.out.println("Debe introducir un número");
         }catch (ArithmeticException e) {
                sc.nextLine();
                n2 = 0;
                System.out.println("No se puede dividir por cero");
         }
    }catch (InputMismatchException e) {
            sc.nextLine();
            n1 = 0;
            System.out.println("Debe introducir un número");
    }
}
```

3.3. CAPTURA GLOBAL DE TODAS LAS EXCEPCIONES – CAPTURA GENÉRICA.

Si no es necesario tratar excepciones concretas de forma específica se puede poner un bloque catch de una clase base que las capture todas y las trate de forma general. Esto se conoce como captura genérica de excepciones usando la clase "Exception" desde la que heredan el resto.

```
public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   int [] array = {4,2,6,7};
   int n;
   boolean repetir = false;
   do{
       try{
            repetir = false;
            System.out.print("Introduce un número entero > 0 y < " + array.length + " ");
            n = sc.nextInt();
            System.out.println("Valor en la posición " + n + ": " + array[n]);
       }catch(InputMismatchException e){
             sc.nextLine();
             n = 0;
             System.out.println("Debe introducir un número entero ");
             repetir = true;
       }catch(IndexOutOfBoundsException e){
            System.out.println("Debe introducir un número entero > 0 y < " + array.length + " ");
            repetir = true;
       }catch(Exception e){ //resto de excepciones de tipo Exception y derivadas
             System.out.println("Error inesperado " + e.toString());
             repetir = true;
       }
   }while(repetir);
}
```

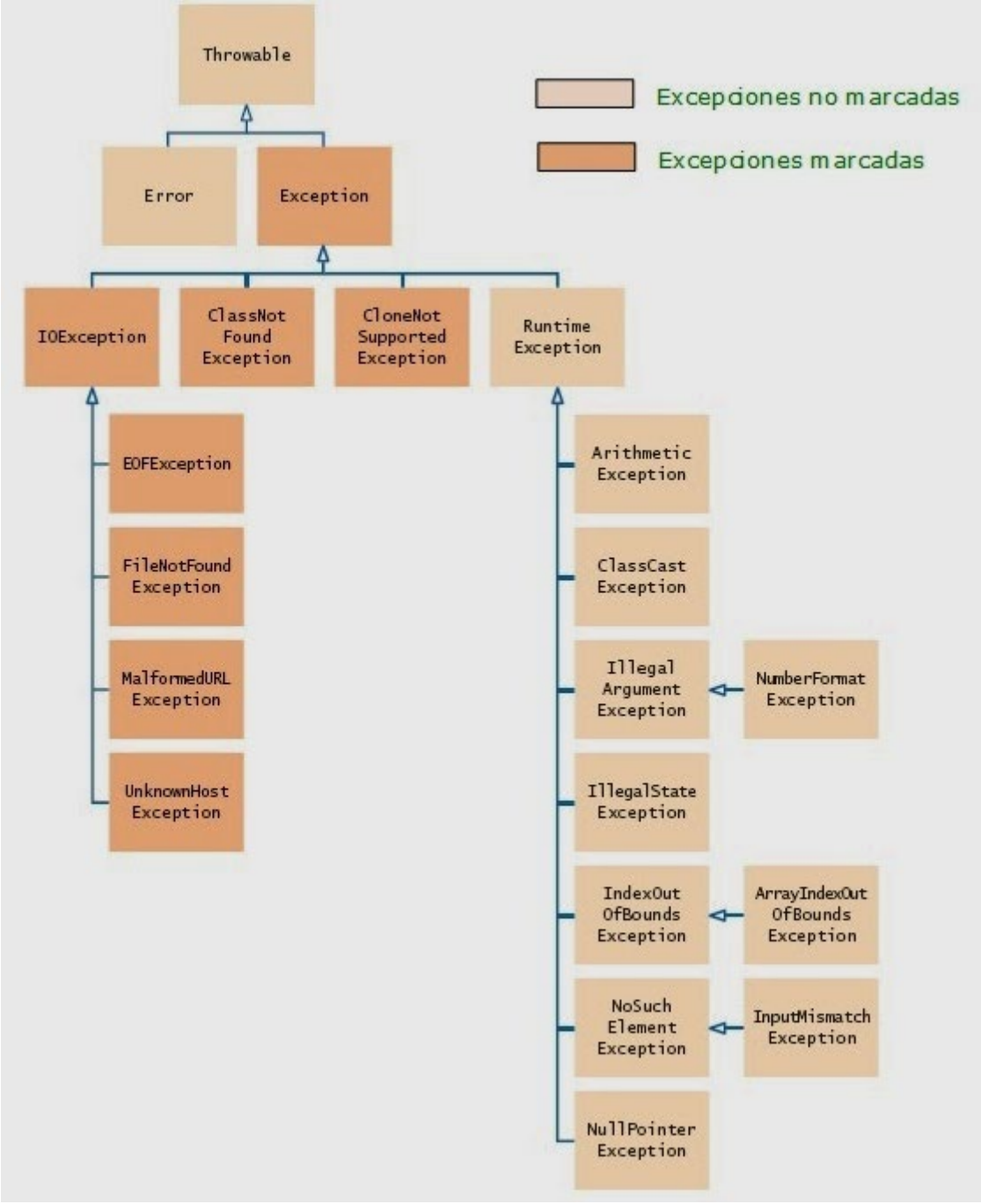
NOTA: Recordar que Exception es un objeto. Que hereda de Throwdable y posteriormente Object. Por lo que contamos con diversos métodos para ver en que consiste y trazar la Excepción.

java.lang.Object
  java.lang.Throwable
    java.lang.Exception

4. EXCEPCIONES MARCADAS Y NO MARCADAS (checked y unchecked)

Como hemos visto hay diversas excepciones definidas en el paquete estándar de Java. Pero, es importante diferenciar dos tipos:

- Excepciones no marcadas (unchecked) son aquellas que *no estamos obligados a tratar*. Pertenecen a la clase Error y a sus clases derivadas y a la clase RuntimeException y sus clases derivadas.
- Excepciones marcadas (checked) son aquellas *que estamos obligados a tratar*. Son todas las clases derivadas de la clase Exception excepto las derivadas de RuntimeException.



5. PROPAGAR EXCEPCIONES.

Un método puede lanzar una excepción ya sea porque no será tratada en la misma sino en donde es llamada o porque genera una excepción que obligatoriamente debe ser capturada. Para ello disponemos de la palabra reservada throws seguida del nombre de las excepciones (Clases) separadas por comas.
A continuación, veremos la situación que puede darse:

1. Captura dentro del método donde se produce la excepción.

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

    dividir(1,0);

}

public static int dividir (int num1, int num2){
    int div = 0;
    try {
        div = num1/num2;
    }catch (Exception e) {
        System.out.println("Tratado dentro de la función dividir");
        System.out.println("Excepcion por dividir entre 0");
        System.out.println("----------");
        e.printStackTrace();
    }

    return div;
}
```

2. El método donde se produce la excepción no la captura sino que la lanza al bloque que lo llama.

```
public static void main(String[] args) {
    try {
        dividir(1,0);
    }catch (Exception e) {
        System.out.println("Tratado dentro del main pues fue lanzada y no tratada en la función dividir");
        System.out.println("Excepcion por dividir entre 0");
        System.out.println("----------");
        e.printStackTrace();
    }

}

public static int dividir (int num1, int num2) throws ArithmeticException{
    int div = num1/num2;
    return div;
}
```

La posibilidad que no tratar una excepción y lanzar al método superior en la pila de llamadas, puede suceder que en ningún instante se trate/capture dicha excepción con un bloque “catch”. En ese caso y recordando como trata JVM las excepciones la ejecución se detendría y aparecerá un mensaje de error con la excepción no
tratada”

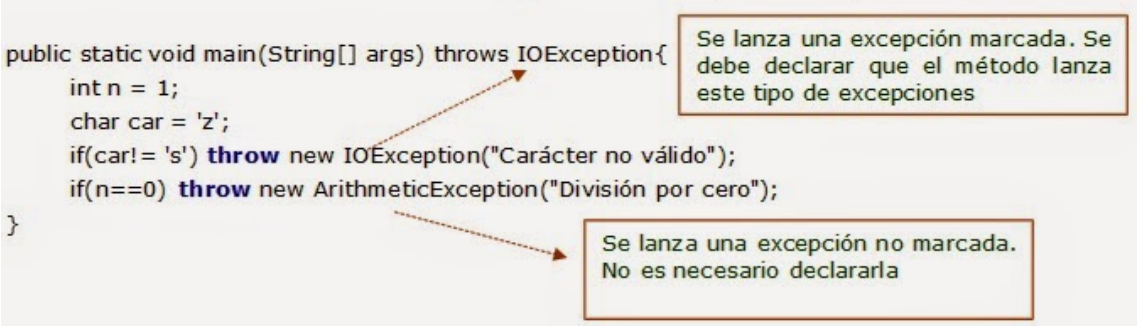
```
public static void main(String[] args) {
    dividir(1,0);
}

public static int dividir (int num1, int num2) throws ArithmeticException{
    int div = num1/num2;
    return div;
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at excepciones.ErrorArray.dividir(ErrorArray.java:16)
        at excepciones.ErrorArray.main(ErrorArray.java:12)
```

6. LANZAR EXCEPCIONES

Java permite al programador lanzar excepciones mediante la palabra reservada *throw*.
throw *objetoExcepcion*;

La excepción que se lanza es un objeto, por lo que hay que crearlo como cualquier otro objeto mediante new.



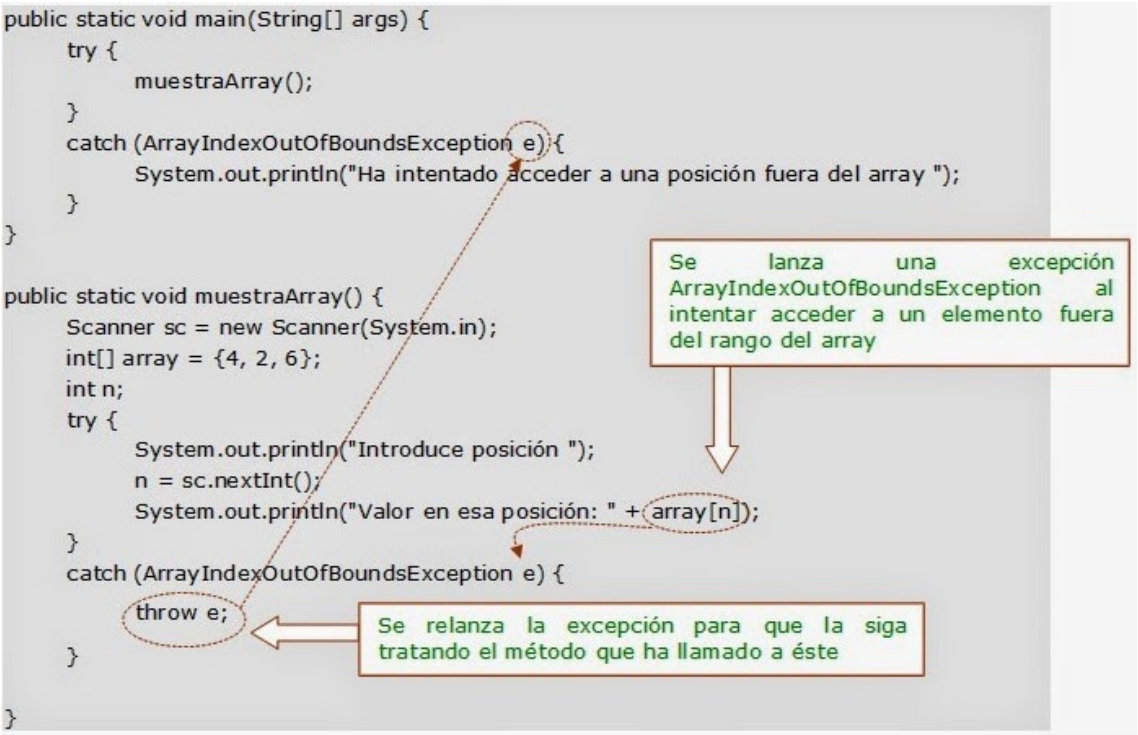
En el ejemplo que estamos siguiendo en esta Unidad:

```
public static void main(String[] args) {
    try {
        dividir(1,0);
    } catch (Exception e) {    // Captura genérica.
        System.out.println("capturada!");
        e.printStackTrace();
    }

}

public static int dividir (int num1, int num2){
    if (num2 == 0) {
        System.out.println("Lanzada excepción");
        throw new ArithmeticException("Num2 no puede ser 0");
    }else {
        return num1/num2;
    }
}
```

Es importante comprender que el flujo secuencial de ejecución del programa se interrumpe inmediatamente después de utilizar la sentencia throw y, por lo tanto, nunca se llegará a la sentencia siguiente, si la hay, ya que el control sale del bloque try en ese punto y pasa a un manejador catch cuyo tipo coincida con el del objeto.

7. RELANZAR EXCEPCIONES

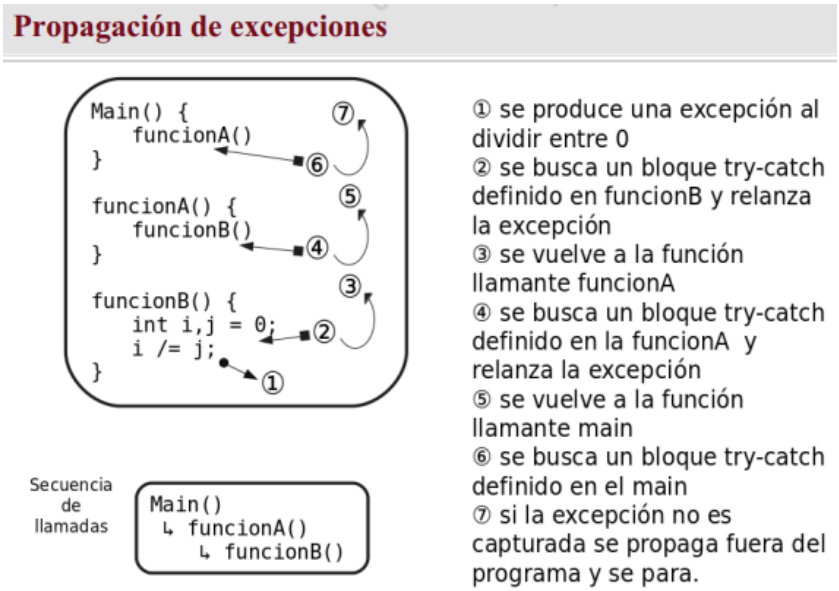
Si se ha capturado una excepción es posible desde el bloque catch relanzar la excepción (el mismo objeto recibido en el bloque catch) utilizando la instrucción throw objetoExcepción.



En nuestro ejemplo: Capturamos la excepción y podemos tratarla total o parcialmente, pero la novedad es que además la relanzamos para que el método que nos llamo pueda también actuar en consecuencia.

```java
public static void main(String[] args) {
    try {
        previoDividir(1,0);
    } catch (Exception e) {   // Captura genérica.
        System.out.println("capturada!");
        e.printStackTrace();
    }

}

public static int previoDividir(int num1, int num2) {
    int div=0;
    try {
        div = dividir(1,0);
    } catch (Exception e) {   // Paso la bola!!!
        System.out.println("Pasando excepcion!");
        throw e;
    }
    return div;
}

public static int dividir (int num1, int num2){
    if (num2 == 0) {
        System.out.println("Lanzada excepción");
        throw new ArithmeticException("Num2 no puede ser 0");
    }else {
        return num1/num2;
    }
}
```

RESUMEN PROPAGACIÓN DE EXCEPCIONES:



8. CREAR NUESTRAS PROPIAS EXCEPCIONES

Aunque Java proporciona una gran cantidad de excepciones, en algunas ocasiones necesitaremos crear excepciones propias. Normalmente crearemos excepciones propias cuando queramos manejar excepciones no contempladas por la librería estándar de Java.

Por ejemplo, vamos a crear un tipo de excepción llamado ValorNoValido que se lanzará cuando el valor utilizado en una determinada operación no sea correcto.

Para ello creamos una clase que herede de "Excepcion":

```
public class ValorNoValido extends Exception{
    public ValorNoValido(){ }
    public ValorNoValido(String cadena){
            super(cadena); //Llama al constructor de Exception y le pasa el contenido de cadena
    }
}
```

```
public static void main(String[] args) {
    try {
         double x = leerValor();
         System.out.println("Raiz cuadrada de " + x + " = " + Math.sqrt(x));
    }catch (ValorNoValido e) {
             System.out.println(e.getMessage());
    }
}

public static double leerValor() throws ValorNoValido {
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce número > 0 ");
        double n = sc.nextDouble();
        if (n <= 0) {
           throw new ValorNoValido("El número debe ser positivo");
        }
        return n;
}
```

Normalmente las excepciones propias provienen / heredan de la clase Exception, pero es posible que también extiendan RuntimeException o alguna otra excepción más precisa.

8.1. EJEMPLO EXCEPCIÓN A LA HORA DE PONER NOTAS DE LOS ALUMNOS.

Condicionantes:

- No puede ser menor que 0
- No puede ser 0 pues la nota mínima es 1
- No puede ser mayor que 10

1. Creamos nuestra clase “ExceptionNotas” que hereda de “Exception”

```
public class ExcepcionNotas extends Exception{

    public static final int ID_ERROR_ZERO = 0;
    public static final int ID_ERROR_NEGATIVO = 1;
    public static final int ID_ERROR_MAYOR_10 = 2;

    private int id_error;

    public ExcepcionNotas(int id_error) {
        this.id_error = id_error;
    }

    @Override
    public String toString() {
        String msg = "";
        switch (this.id_error) {
        case ID_ERROR_ZERO:
            msg = "Excepción producida porque no se puede poner una 0 al alumno, tiene que ser al menos un 1";
            break;
        case ID_ERROR_NEGATIVO:
            msg = "Excepción producida porque no se puede poner una nota negativa al alumno tiene que ser un número positivo";
            break;

        case ID_ERROR_MAYOR_10:
            msg = "Excepción producida porque no se puede poner una nota superior a 10, el cúal es la nota máxima.";
            break;

        default:
            msg = "Se ha produce Excepcion, pero es una Excepcion cuyo ID es desconocido";
            break;
        }

        return "ExcepcionPonerNotas [id_error=" + id_error +" msg: "+msg+"]";
    }

}
```

DEBE EXTENDER DE "EXCEPTION"

EL USUARIO AL GENERAR LA EXCEPCION DEBE PROPORCIONA UN CÓDIGO

USAMOS EL toString que hereda de la clase Object para formatear una mensaje personalizado para cada código de excepción

2. Cuando observamos que debemos lanzar nuestra Excepción, creamos un objeto de la clase que previamente hemos creado con la palabra reservada **throw** y al lanzar la Excepción también es necesario propagarlo y para eso utilizamos la palabra reservada **throws.**

```
public static void main(String[] args) throws ExcepcionNotas {

    Scanner sc = new Scanner(System.in);
    System.out.println("Introduzca la nota de Manuel");
    int nota = sc.nextInt();
    if(nota<0) {
        throw new ExcepcionNotas(ExcepcionNotas.ID_ERROR_NEGATIVO);
    }
    if(nota==0) {
        throw new ExcepcionNotas(ExcepcionNotas.ID_ERROR_ZERO);
    }
    if(nota>0) {
        throw new ExcepcionNotas(ExcepcionNotas.ID_ERROR_MAYOR_10);
    }


}
```

PROPAGAMOS EXCEPCIÓN

LANZAMOS EXCEPCIÓN