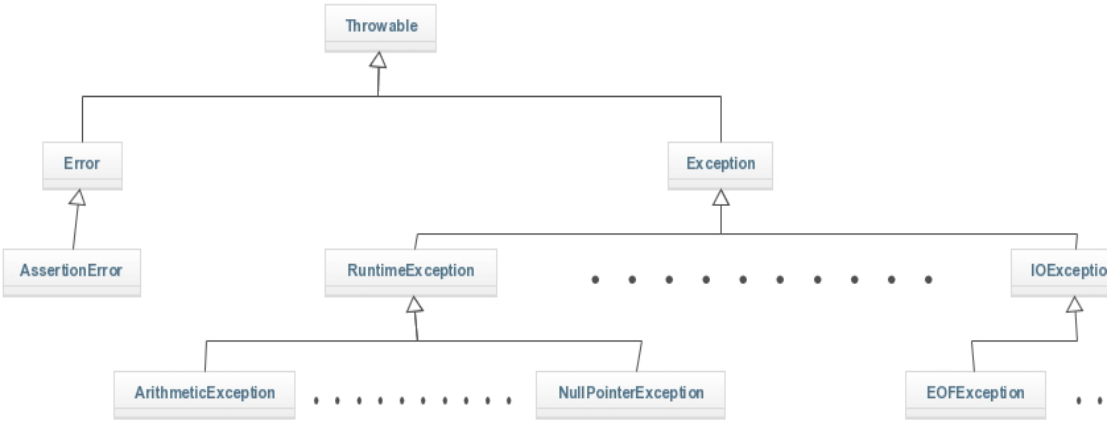
EXCEPCIONES DE JAVA

Excepción es una condición anormal que surge en una secuencia de código durante la ejecución de un programa. Cuando se produce se crea un objeto que representa la excepción y se le envía al método que lo ha provocado.

Son los errores que pueden aparecen en la ejecución del programa. Al no introducir correctamente los datos, dividir por cero, exceder el tamaño de un array, abrir un archivo que no existe, etc…

Las excepciones se pueden gestionar con palabras reservadas como: **try**, **cactch**, **finally**, **throw** y **throws**, para resolver y gestionar esta situación. El objetivo es que el programa no termine de forma abrupta.

Todas las excepciones se encuentran en la clase **Throwable** (se encuentra en java.lang)



Se dividen en dos grupos:

1. **Error**, por errores anormales de ejecución del programa, producidos por el sistema. Ej: **AWTError**, **LinkageError**,….

2. **Exception**, son errores producidos por la ejecución del programa. Estos errores pueden ser controlados y podemos darles una solución. Ej: **ArithmeticException**, **RuntimeException**, **Exception**,…

Algunos de los métodos que podemos utilizar con las excepciones son:

String toString(), devuelve una cadena con el mensaje de la excepción.

getClass(),devuelve la clase a la que pertenece la excepción.

String getMessage(), devuelve el mensaje con el que fue creada la excepción.

printStackTrace(), imprime el objeto desde el que es invocado y con la llamada a los métodos desde los que se ha producido la excepción.

Si prevemos un error, podremos controlarlo, para que el programa no rompa y gestiona nosotros los errores y seguir con nuestro trabajo.

Encerramos el código que puede dar el error entre las llaves del **try**

```
try
{
   Sentencias;
}
```

En el **catch** indicamos que hacemos si se produce el error.

```
catch(tipoerror  nombre )
{
  sentencias para el error;
}
```

Es opcional escribir el **finally**

```
finally
{
 Sentencias que se van a ejecutar con o sin error.
 Aquí debemos cerrar los canales para cerrar el programa.
}
```

Se pueden escribir tantos catch como errores distintos puedan aparecer. Cada catch solo controla un único tipo de error. Pasará por el primer error que corresponda al producido. Si tenemos un caso genérico y otros particulares deberíamos poner el genérico el último. El nombre que le damos al error (como identificador o variable) puede ser el mismo para todos los casos, lo mejor es que se identifiquen de alguna forma. Algunos errores son:

- ✓ **Exception**, es un error genérico. Incluye todos los tipos de errores.
- ✓ **NumberFormatException**, captura letras por números.
- ✓ **ArithmeticException**, realiza una división por cero.
- ✓ **InputMismatchException**, en la captura de datos no es correcta.
- ✓ **ArrayIndexOfBoundsException**, si sobrepasamos el tamaño del array.
- ✓ **ClassNotFoundException**, al tratar de utilizar una clase.
- ✓ **DataFormatException**, error en el formato de datos.
- ✓ **IllegalAccessException**, se intenta acceder a una clase a la que no se tiene permiso. Como puede ser ejecutar un método privado de otra clase.
- ✓ **IOException**, excepciones producidas al realizar tareas de entrada/salida por el programa. Como pueden ser: **EOFException**, **FileNotFoundException**, **MalformedURLexception**, **SocketException**, **UnknownHostException**, **UTFDataFormatException**.
- ✓ **NoSuchFieldException**, no se encuentra un determinado atributo.
- ✓ **NoSuchMethodException**, no se encuentra un determinado método.
- ✓ **RuntimeException**, producidos en tiempo de ejecución. Algunos pueden ser: **AritmeticException**, **ClassCastException**, **IndexOutOfBoundsException**, **NegativArraysSizeException**, **NullPointerException**.
- ✓ Etc…

Podemos tener más de un **try** dentro de nuestro programa ya sea en el programa principal o en los métodos desarrollados, pero cada try debe tener a continuación su **catch**. Un error dentro de un método se puede tratar ahí, o si no lo llevara al programa de llamada para evaluarlo y así hasta llegar al main. Solo se propagan los errores de la clase **RunTimeException**. La ventaja de la propagación consiste en no repetir la excepción en cada método y solo una vez.

Podemos salir del programa con **System.exit(0)**. Rompe totalmente la ejecución.

Se puede escribir código fuera de los bloques pero al final de los catch o finally no entre medias.

Veamos un ejemplo:

```
import java.io.*;
import java.util.*;

public class excepcion
{
 public static void main(String [] args)
 {
    Scanner dato = new Scanner(System.in);
    int numero;
    int []tabla = new int[8];
    try{
        System.out.println("INTRODUCE UN N\u00D9MERO ");
        numero = dato.nextInt();
        for(int a=0; a<numero;a++)
          {
            System.out.println("VALOR DEL ARRAY "+tabla[a]);
          }
       }
    catch(InputMismatchException a)
      {    System.out.println("ERROR EN LA CAPTURA DE DATOS ");   }
    catch(ArrayIndexOutOfBoundsException taLa)
      {    System.out.println("EXCESO DEL ARRAY");   }
    catch(Exception a)
      {    System.out.println("ERROR GENERICO ");   }
    finally
      {    System.out.println("FIN DE PROGRAMA ");     }
 }
}
```

Podemos tener try anidados, se terminan el más interno y después los externos. Se heredan las excepciones.

Podemos crear nuestras propias excepciones con una subclase y que herede de la clase **Exception**.

Los errores también se pueden controlar en la cabecera de los métodos. A esto llaman lanzar un error. Un método puede lanzar más de un error debemos escribir al final del método la palabra **throws** seguido de la excepción o excepciones separadas por comas.

tipo nombre_metodo(argumentos) throws excepcion1[, excepcion2,…]

Podemos crear nuestras propias clases de excepciones, con **throw**.

class nombre extends Exception

Dentro del código del programa escribiremos throw new Excepción(“mensaje”);

Ejemplo a realizar:

Tenemos un array cargado en memoria y pedimos al usuario que nos dé una posición del array para obtener el elemento guardado. La posición tecleada puede que no exista.

Ejemplo de lanzar una excepción al programa principal producida en un método.

```
import java.io.*;
public class lanzarExcepcion
{
 public static void main(String [] args) throws IOException
 {
   int num;
   System.out.println("DAME UN NUMERO ");
   try{
          num = pedirNumero();
          System.out.println("SU NUMERO ES " +num);
   }
   catch(NumberFormatException e)
   {        System.out.println("HA INSERTADO UN CARACTER NO NUMERICO ");    }
 }

  static int pedirNumero() throws IOException
  {
          InputStreamReader xx = new InputStreamReader(System.in);
          BufferedReader dato = new BufferedReader(xx);
          int num=0;
          num = Integer.parseInt(dato.readLine());
          return num;
  }
}
```

Desde el método controlamos la excepción, pero también es pasada al programa principal para que la evalúe de alguna manera. En este caso termina el programa viendo los resultados finales.

Es una modificación del ejercicio anterior, donde contamos los números capturados y la suma de todos ellos.

```
import java.io.*;
public class lanzarExcepcionb
{
 public static void main(String [] args) throws IOException
 {
   int num, suma=0, cnt=0;
   boolean seguir=true;
   do{
       System.out.println("DAME UN NUMERO ");
       try{
            num = pedirNumero();
            suma+=num;
            cnt++;
        }
        catch(NumberFormatException e)
        {  seguir = false;  }// al producirse un error en el método nos salimos
     }while(seguir==true);
     System.out.println("SE HAN CAPTURADO "+cnt +" NUMEROS.");
    System.out.println("LA SUMA DE TODOS ELLOS HA SIDO "+suma);
 }

  static int pedirNumero() throws IOException
  {
          InputStreamReader xx = new InputStreamReader(System.in);
          BufferedReader dato = new BufferedReader(xx);
          int num=0;
          try{
            num = Integer.parseInt(dato.readLine());
          }
   catch(NumberFormatException e)
          {
            System.out.println("HA INSERTADO UN CARACTER NO NUMERICO ");
                  throw e;
          }
          return num;
  }
}
```

También podemos invocar a una excepcion con:

Throw new nombreExcpcion(“mensaje a enviar”);

Aquí tenemos un ejemplo para controlar la excepción desde donde fue invocada. Esta se produce en un método en el que podemos poner en la cabecera la excepción o excepciones que se pueden producir y el error se gestionara desde donde se llamo al método correspondiente.

```
import java.io.*;

public class lanzarExcepcion
{
 public static void main(String [] args) throws IOException
 {
   int num;
   System.out.println("DAME UN NUMERO ");
   try{
            num = pedirNumero();
                System.out.println("SU NUMERO ES " +num);
   }
   catch(NumberFormatException e)
   {
            System.out.println("HA INSERTADO UN CARACTER NO NUMERICO ");
   }
 }

  static int pedirNumero() throws IOException
  {
          InputStreamReader xx = new InputStreamReader(System.in);
          BufferedReader dato = new BufferedReader(xx);
          int num=0;
          num = Integer.parseInt(dato.readLine());
          return num;
  }
}
```

Podemos utilizar las palabras **throws** o **throw**. Veamos su diferencia:

throws, la palabra reservada o cláusula permite lanzar un método. Por lo tanto, tiene que ir declarada en el método.

throw, esta palabra reservada nos permite lanzar una excepción propia.

Puedes ver ejemplos en:

https://www.webferrol.com/diferencias-entre-throw-y-throws/

http://dis.um.es/~bmoros/Tutorial/parte9/cap9-3.html

https://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html