EJEMPLOS DE EXCEPCIONES NO VERIFICADAS

Vamos a ver algunos ejemplos de las excepciones no verificadas más frecuentes en Java. Estos ejemplos te servirán de ayuda para comprender mejor el concepto de excepciones no verificadas y el concepto de excepción como tal.

Antes de mencionar la lista de excepciones no verificadas más comunes, recordemos que en Java todo son objetos (salvo las variables primitivas por supuesto) y las *excepciones* no son la "excepción" a esta regla. Las excepciones también son objetos de clases que se encuentran distribuidas a través de distintos paquetes de la API de Java.

Cuando se lanza una excepción dentro de un método, lo que hace no es otra cosa más que crear un objeto de tipo Exception o de alguna de sus subclases.

En cada uno de los ejemplos siguientes, mostraremos el nombre completo de la clase Exception, una descripción de cuando ocurren y un ejemplo que muestre en qué tipo de código es producida esa excepción.

ArithmeticException

De esta excepción hemos hablado ya varias veces durante esta sección y ha sido nuestro ejemplo principal. En esta ocasión solo vamos a puntualizar para que quede fijo el conocimiento.

Nombre de la clase: java.lang.ArithmeticException.
Esta clase se produce cuando se intenta dividir un número entero entre cero. Ejemplo:

```
class ArithmeticExDemo

{

   public static void main(String args[])

   {

      try{

         int num1=30, num2=0;

         int output=num1/num2;

         System.out.println ("Resultado = " +output);

      }

      catch(ArithmeticException e){

         System.out.println ("Arithmetic Exception: No puedes dividir un entero entr
e cero");

      }

   }
```

En este ejemplo se ha dividido un entero entre un cero y por esta razón se produce la excepción ArithmeticException.

ArrayIndexOutofBoundsException

Esta excepción ya la hemos mencionado también y está relacionado con errores de acceso a los elementos de un arreglo. La clase ArrayIndexOutOfBoundsException hereda de la clase IndexOutOfBoundsException.

Nombre de la clase: java.lang.ArrayIndexOutOfBoundsException

Esta excepción ocurre cuando la referencia al elemento de un array no existe. O, dicho de otro modo, se está intentando acceder al elemento de un array en un índice que no existe. Por ejemplo, si un arreglo tiene solo cinco elementos y nosotros tratamos de desplegar el séptimo elemento, entonces se lanzará una excepción de este tipo. Ejemplo:

```
class ArrayIndexExceptionDemo

{

   public static void main(String args[])

   {

      try{

        int a[]=new int[10];

        //El arreglo solo tiene 10 elementos y se intenta acceder al elemento con el
índice 11

        //cuando en realidad este arreglo solo tiene hasta el índice 9 (empieza a co
ntar desde cero).

        a[11] = 26;

      }

      catch(ArrayIndexOutOfBoundsException e){

         System.out.println ("ArrayIndexOutOfBounds");

      }

   }

}
```

En el ejemplo anterior, el arreglo está inicializado para almacenar sólo 10 elementos con índices del 0 al 9. La excepción se lanza porque intentamos acceder a un elemento con el índice 11 el cual está fuera del rango de índices.

NumberFormatException

Nombre de la clase: java.lang.NumberFormatException

Un objeto de esta clase de excepción es creado cuando intentamos convertir un String a cualquier variable de tipo numérico (int, float, double, etc) pero con un formato equivocado.

Desde luego que a un String se le puede hacer "casting" a un tipo numérico, siempre y cuando esa cadena de texto represente fielmente a un número y no tenga ningún otro caracter que no sea un número o un punto decimal. Por ejemplo, en la sentencia:

`int numero = integer.parseInt("Java");` , se lanzará una excepción de este tipo porque la cadena "Java" no puede ser convertida a un tipo numérico int.

Este error no genera errores de compilación. El compilador de java comprueba que el argumento del método parseInt sea un String, hasta ese momento el código es válido y no es sino hasta el momento de la ejecución, cuando se puede corroborar que existe el error.

Ejemplo:

```
class NumberFormatExceptionDemo

{

   public static void main(String args[])

   {

      try{

     int numero = Integer.parseInt ("Java") ;

     System.out.println(numero);

      }catch(NumberFormatException e){

      System.out.println("Una excepción Number Format ha ocurrido");

       }

   }

}
```

Una forma válida de utilizar el método parseInt sería la siguiente:

```
int numero = Integer.parseInt ("36") ;
```

StringIndexOutOfBoundException

Esta clase es similar a la clase ArrayIndexOutOfBoundsException y también hereda de la clase IndexOutOfBoundsException.
A final de cuentas los Strings son una especie de arreglos, de hecho, son como arreglos de variables tipo char. Los Strings comparten con los arreglos el hecho de que cada uno de sus caracteres tiene un índice el cuál inicia en 0. De esta manera, la cadena "java" tiene el caracter 'j' en el índice 0, el caracter 'a' en el índice 1, el caracter 'v' en el índice 2 y así sucesivamente.

Una excepción del tipo *StringIndexOutOfBoundsException* se lanza cuando se intenta invocar un *elemento* de un String mediante un índice que no está dentro del rango de ese String.
Para obtener un caracter particular de un objeto String, utilizamos el método charAt(int indice) de la clase String; donde el parámetro "indice" indica un número entero que debemos pasar como argumento. Este número entero debe encontrarse dentro del rango de índices de ese String o se lanzará la excepción *StringIndexOutOfBoundsException.*

El método charAt no es el único ejemplo en el cual se puede generar esta excepción. Cualquier método de la clase String que acceda a los elementos de un String mediante sus índices, es propenso a lanzar excepciones de este tipo. Los métodos *subSequence* y *subString* podrían ser otros ejemplos, pero no son los únicos.

```
class StringIndexExDemo

{

   public static void main(String args[])

   {

      try{

         String str="pasos sencillos para aprender programación";

         System.out.println(str.length()); // esto imprime 42 lo que indica que tene
mos hasta el índice 41



         c = str.charAt(50); //Esto produce una excepcion

         System.out.println(c);



      }catch(StringIndexOutOfBoundsException e){

          System.out.println("StringIndexOutOfBoundsException!!");

      }

   }

}
```

En el ejemplo anterior, imprimimos en consola el tamaño de la cadena str. El resultado sera 42 y eso nos indica que nuestra cadena str tiene hasta el índice 41 comenzando desde cero.

En la línea

```
c = str.charAt(50);
```

se está intentando acceder a un índice que el String no tiene y en consecuencia será lanzará una excepción de este tipo.

NullPointerException

Nombre de la clase: java.lang.NullPointerException

En el código...

```
Object objeto = new Object();
```

No debemos olvidar que la porción de código a la izquierda del símbolo igual es la *referencia al objeto*. La porción de código a la derecha del símbolo igual es *la creación del objeto mismo* (aunque no es la única manera de asignar un objeto a una referencia).

Los objetos se almacenan en memoria y las referencias ***apuntan*** a esa porción de memoria donde se encuentra el objeto.

Cuando una referencia se iguala a null, indica que está apuntando a un objeto nulo, aunque es más propio decir que no está apuntando a ningún objeto en realidad. En ciertas ocasiones esta práctica puede ser válida para diversos fines, pero mientras la referencia no se inicialice, no será posible realizar acciones sobre esos objetos que están representados por sus referencias.

En ocasiones los programadores inicializan una referencia a null o simplemente no la inicializan.

Escribir `Object objeto = null;` es lo mismo que decir `Object objeto;`

En algún otro momento apropiado, el programador puede inicializar sus referencias, asignándoles un objeto y entonces se podrán realizar acciones sobre ellos.

El siguiente ejemplo intenta ejecutar el método length() sobre una referencia a un objeto String no inicializado.

```
class NullPointerExDemo

{

   public static void main(String args[])

   {

      try{

         //referencia String apunta a un objeto null

         String str=null;

         //se intenta imprimir la dimensión de un objeto String no inicializado

         System.out.println (str.length());

      }catch(NullPointerException e){

         System.out.println("NullPointerException..");

      }

   }

}
```

En este ejemplo, se intenta acceder al método length (el cual mide la dimensión de una cadena de texto) de la clase String pero la referencia "str" no está inicializada y por tanto, no hay ningún objeto que medir. Esto provocará la excepcion NullPointerException.

Hay muchas otras excepciones no verificadas que puedes encontrar como subclases de la clase RuntimeException en la documentación de la API de java en la siguiente dirección:

https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html

En la sección *Direct Known Subclasses* están todas las subclases directas de la clase RuntimeException pero aun así no son las únicas excepciones no verificadas, algunas de estas subclases también tienen subclases y así sucesivamente y todas estas subclases son también excepciones no verificadas.

Sin embargo, los ejemplos aquí mencionados son los más usuales con los que se enfrentan los programadores cada día. Cuando te surja alguna excepción que no conozcas, siempre está la documentación oficial de Java para consultar su información, además de muchísimos otros recursos y foros en la web con respuestas de todo tipo.

Por último, en esta clase escrita, recordar primeramente que el compilador de java no exige que declaremos (throws) ni que atrapemos (try - catch) las excepciones no verificadas.

En los códigos de ejemplo hemos encerrado los códigos que generan alguna excepción en bloques try - catch con fines demostrativos. Esto haría que una aplicación continue ejecutándose a pesar de producirse la excepción; sin embargo, no significa que sin estos bloques try - catch la excepción no se produzca, sí se produce, la diferencia es que sin los bloques try-catch, la aplicación se cierra en cuanto se produce la excepción.

Aunque puede parecer conveniente utilizar bloques try-catch para atrapar excepciones no verificadas y así impedir el cierre de las aplicaciones, esto no siempre es conveniente.

En algunas ocasiones lo mejor es adecuar nuestro código para evitar que dichas excepciones se produzcan. Por ejemplo

En lugar del código:

```
class NullPointerExDemo

{

   public static void main(String args[])

   {

      try{

         //referencia String apunta a un objeto null

         String str=null;

         //se intenta imprimir la dimensión de un objeto String no inicializado

         System.out.println (str.length());

      }catch(NullPointerException e){

         System.out.println("NullPointerException..");

      }

   }

}
```

Sería mejor validar desde el inicio que el objeto no sea nulo de la siguiente manera:

```
class NullPointerExDemo

{

   public static void main(String args[])

   {

      String str=null;



      //valida que el objeto no sea nulo

      if( str != null ) {



          System.out.println(str.length());

      }



   }

}
```

por supuesto esto es solo un ejemplo (algo absurdo) y obviamente nunca se va a cumplir la condición establecida por la estructura if. Pero de esta manera nos aseguramos de que no se produzcan excepciones NullPointerException sin necesidad de usar bloques try-catch.

Evitar los bloques try-catch en las excepciones no verificadas no es una regla estricta; en ciertas ocasiones puede ser conveniente atraparlas con algún fin específico (por ejemplo, mostrar warnings en una consola de errores en nuestra aplicación).

En la clase siguiente nos dedicaremos a las excepciones verificadas, las cuales si nos exigen ser declaradas o atrapadas.