UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Acreditación Institucional Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 8 DE MAYO-VIGENCIA 5 AÑOS

ACREDITACIÓN INSTITUCIONAL DE ALTA CALIDAD MULTICAMPUS

Res. MEN No. 03455 del 23 de enero de 2020

Vigencia por seis años

QS STARS RATED FOR EXCELLENCE

ISO 9001 Icontec SC4259-1

CERTIFIED IQNet MANAGEMENT SYSTEM

UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA
VIGILADA MINEDUCACIÓN - SNIES 1732

Faculty: Systems engineer
Course: Introduction of Programming
Topic: **TRY-CATCH-FINALLY y Uso de ficheros de texto para guardar información**

Socializer: Luis Fernando Castellanos Guarín
Email: Luis.castellanosg@usantoto.edu.co
Phone: 321-4582098

Topics

Files and Try-catch and finally

- Acerca de Try -Catch /About Try – Catch
- Ejemplos / examples
- Acerca de /About files
- Crear y escribir en ficheros/ Create and write to files
- Ejemplos / examples
- Abrir y leer desde ficheros / Open and read from files
- Ejemplos / examples
- Ejercicios / Exercises

¡Siempre hacia lo alto!

Try – catch – finally

Una **excepción** es una **situación anómala** que puede provocar que el programa no funcione de forma correcta o termine de forma inesperada. Ejemplos de situaciones que provocan una excepción:

- No hay memoria disponible para asignar
- Acceso a un elemento de un array fuera de rango
- Leer por teclado un dato de un tipo distinto al esperado
- Error al abrir un fichero
- División por cero
- Problemas de Hardware

Try – Catch- finally

Si la excepción no se trata, el manejador de excepciones realiza lo siguiente:

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

Ejemplos de código Java que provocan una excepción:

```
public static void main(String[] args) {
    int a = 4, b=0;
    System.out.println(a/b);
}
```

provoca:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

```
public static void main(String[] args) {
    int [] array = new int [5];
    array[5] = 1;
}
```

provoca:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Introduce un número entero: ");
    int n = sc.nextInt();
    System.out.print("Número introducido: " + n);
}
```

Se espera un int. Si por ejemplo se lee 4,5 provoca:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextInt(Scanner.java:2160)
    at java.util.Scanner.nextInt(Scanner.java:2119)
    at excepciones1.Excepciones1.main(Excepciones1.java:7)
```

El manejo de excepciones proporciona una **separación entre el código básico y el código que maneja los errores**, haciéndolo más legible.

Utilizando excepciones se consigue:

- Separar las instrucciones del programa de las del tratamiento de errores.
- Evitar llenar el código del programa de instrucciones de comprobación (if, switch, etc).
- El código es más simple de escribir ya que no se fuerza al programador a comprobar los errores constantemente.

Try – Catch- finally

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:**

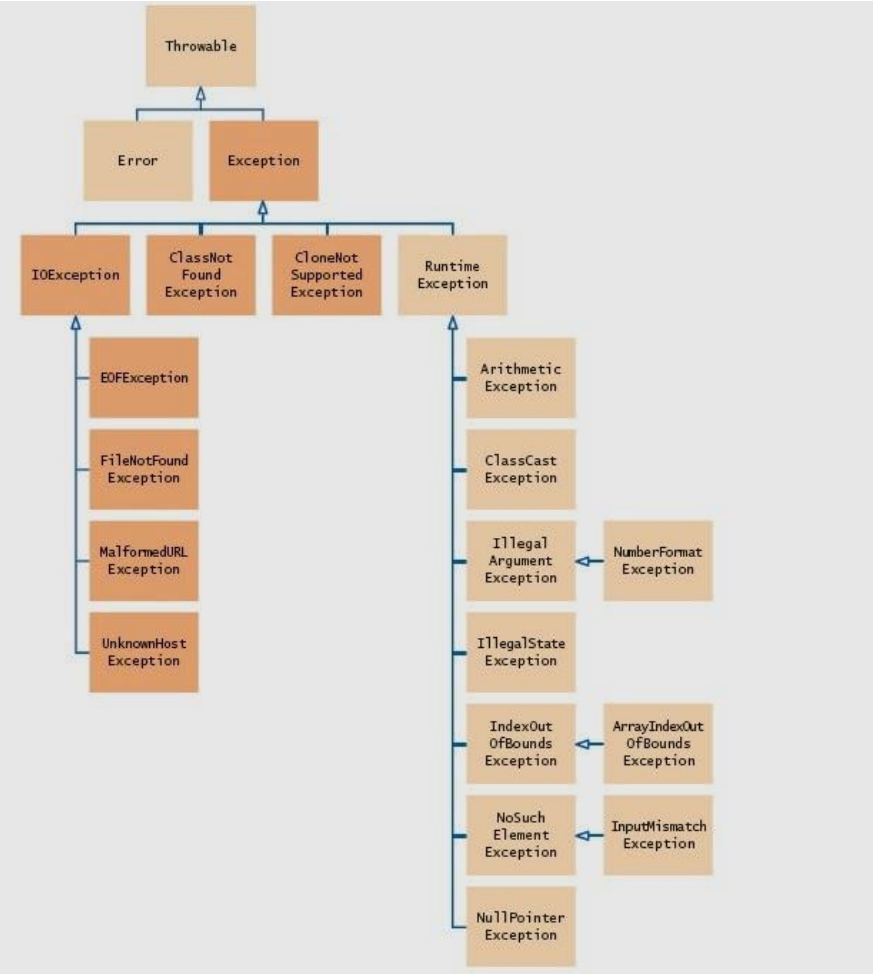
La clase **Error** está relacionada con errores de la máquina virtual de Java. Generalmente estos errores no dependen del programador por lo que no nos debemos preocupar por tratarlos, por ejemplo OutOfMemoryError, StackOverflowError, errores de hardware, etc.

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.

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



Try – Catch- finally

Esquema general del uso de TRY – Catch – finally

```
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
}
```

Try – Catch- finally

Bloque try:
se encuentran las **instrucciones que pueden lanzar una excepción.**
Solamente se pueden capturar las excepciones lanzadas dentro de un bloque try.
Una excepción se puede lanzar de forma automática o mediante la palabra reservada **throw**.
Cuando se lanza la excepción se transfiere la ejecución del programa desde el punto donde **se lanza** la excepción a otro punto donde **se captura** la excepción.

Bloque catch:

Es el bloque de código donde se captura la excepción. El bloque catch es el **manejador** o **handler** de la excepción. Aquí se decide qué hacer con la excepción capturada. Puede haber varios bloques catch relacionados con un bloque try.
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.

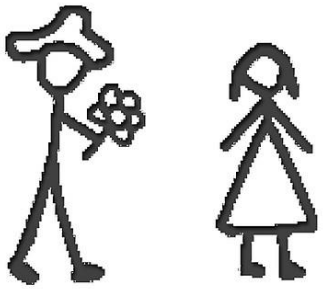
Bloque finally: Es opcional.

Debe aparecer a continuación de los bloques catch.
También puede aparecer a continuación de un bloque try si no hay bloques catch.
La ejecución de sus instrucciones queda garantizada independientemente de que el bloque try acabe o no su ejecución incluso en estos casos:
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.**
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.

Try – Catch- finally

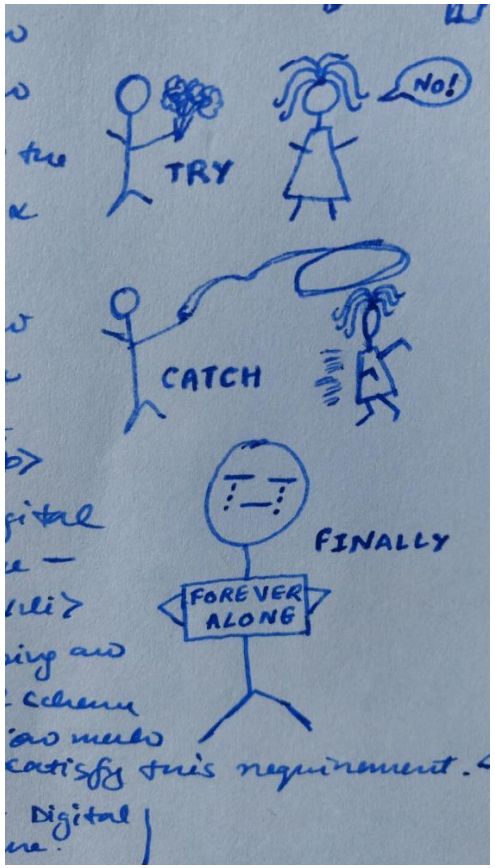
Algo de memes para explicar mejor:

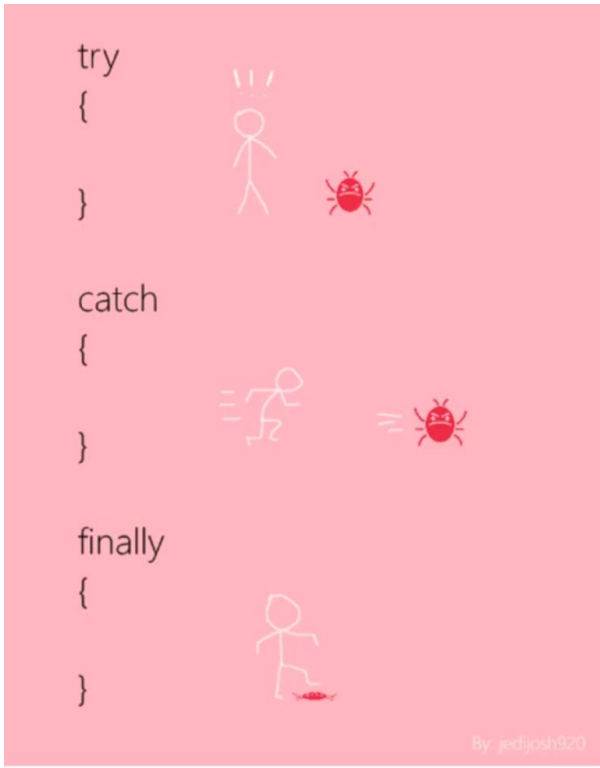
```
try{
    girl.approach();
```



```
}catch (FriendzoneException e){
```

```
}
```





¡Siempre hacia lo alto!

Try – Catch- finally

Ejemplos:

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        do{
            try{
                System.out.print("Introduce un número entero > 0: ");
                n = sc.nextInt();
                System.out.println("Número introducido: " + n);
            }catch(InputMismatchException e){
                sc.nextLine();
                n = 0;
                System.err.println("ERROR: Debe introducir un número entero " + e.toString());
            }
        }while(n<=0);
    }
}
```

Try – Catch- finally

Ejemplo de tratamiento de excepciones con varios catch:

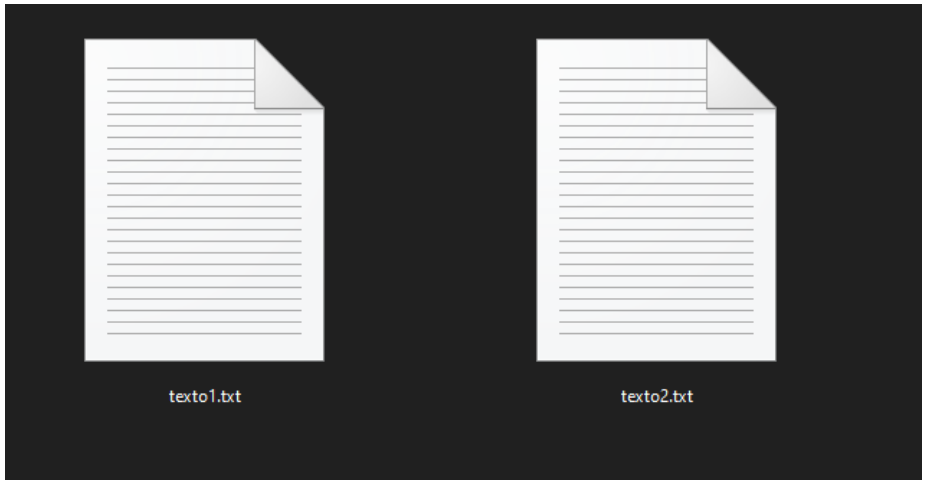
En este programa se controla el error producido si se introduce un dato no entero y si se intenta acceder a una posición fuera del array.

```
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){ nextInt() puede lanzar
            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;
      }
   }while(repetir);
}
```

array[n] puede lanzar

Archivos de texto para uso en JAVA

Un fichero de texto (.txt) está formado por secuencias de caracteres, organizados en líneas de igual o distinta longitud.
Todos los datos que aparecen en estos ficheros están formados por caracteres.



Create and write to files

Para crear archivos necesitaremos la librería FILE y una de las siguientes opciones:

- **Opción 1:** La primera estrategia es usando las clases **FileWriter** y **BufferWriter**, donde usamos el método **write** que permite escribir cadenas o arreglos de caracteres.

- **Opción 2:** La segunda forma es usando PrintWriter que te permite hacer más o menos lo mismo, pero de una forma más resumida y con la posibilidad de escribir otros tipos de datos sobre el archivo.

Create and write to files

Para crear archivos necesitaremos la librería FILE y la Opción 1: La primera estrategia es usando las clases **FileWriter** y **BufferWriter**, donde usamos el método **write** que permite escribir cadenas o arreglos de caracteres.

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;

public class CrearArchivo {

    public static void main(String ars[]){
        try {
            String ruta = "/ruta/filename.txt";
            String contenido = "Contenido de ejemplo";

            File file = new File(ruta);

            // Si el archivo no existe es creado
            if (!file.exists()) {
                file.createNewFile();
            }

            FileWriter fw = new FileWriter(file);
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write(contenido);
            bw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

La escritura se hace en estas sentencias:

```
FileWriter fw = new FileWriter(file);
BufferedWriter bw = new BufferedWriter(fw);
bw.write(contenido);
bw.close();
```

La clase **FileWriter** debe crearse con una referencia a una clase **File** que contiene los detalles del archivo que será creado.

El contenido del texto se crea con la función **bw.write(contenido)** de **BufferedWriter** y dependerá del texto a agregar el carácter de salto de línea.

Create and write to files

Para crear archivos necesitaremos la librería FILE y la Opción 2: La segunda forma es usando PrintWriter que te permite hacer más o menos lo mismo, pero de una forma más resumida y con la posibilidad de escribir otros tipos de datos sobre el archivo (*a mi parecer la más sencilla*)

```
import java.io.PrintWriter;

public class CrearArchivo {
    public static void main(String ars[]){
        try {
            PrintWriter writer = new PrintWriter("/ruta/filename.txt", "UTF-8");
            writer.println("Primera línea");
            writer.println("Segunda línea");
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

En esta segunda opción, la clase **PrintWriter** puede recibir una cadena con la ruta del archivo o una instancia de **File**, el segundo argumento es el tipo de encoding que será usado (**«UTF-8»**), que será útil si necesitas guardar texto con caracteres especiales como **acentos** y **eñes**.

El trabajo de escritura se hace con el método **println**, que permite escribir otros tipos de datos como enteros, de punto flotante, booleanos y caracteres.

Open and read from files

Para leer un fichero desde un programa en Java y mostrar su contenido por pantalla.
También hay varias formas:

- La primera opción utilizaremos las clases **BufferedReader** (lee texto de una entrada de caracteres.) y **FileReader** (para leer archivos).
- La segunda es usando la clase **Scanner**.

Para la primera opción, utilizaremos un método al que llamaremos **muestraContenido**, cuyo propósito es sacar el texto por pantalla. Tendrá como parámetro de entrada la ruta de nuestro archivo.

```
void muestraContenido(String archivo) throws FileNotFoundException, IOException {
		String cadena;
		FileReader f = new FileReader(archivo);
		BufferedReader b = new BufferedReader(f);
		while((cadena = b.readLine())!=null) {
			System.out.println(cadena);
		}
		b.close();
}
```

Open and read from files

Siguiendo con la primera opción, se lanzan las excepciones *FileNotFoundException* y *IOException* en caso de que no se encuentre el archivo o haya un error en la lectura. El ejemplo completo de uso sería el siguiente:

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Main {

    public static void muestraContenido(String archivo) throws FileNotFoundException, IOException {
        String cadena;
        FileReader f = new FileReader(archivo);
        BufferedReader b = new BufferedReader(f);
        while((cadena = b.readLine())!=null) {
            System.out.println(cadena);
        }
        b.close();
    }

    public static void main(String[] args) throws IOException {
        muestraContenido("/home/mario/archivo.txt");
    }

}
```

Open and read from files

Con la segunda opción (para mi es la más sencilla) es usando la librería **Scanner**, el ejemplo completo de uso sería el siguiente:

```
import java.io.File;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(new File("/ruta/filename.txt"));
            while (input.hasNextLine()) {
                String line = input.nextLine();
                System.out.println(line);
            }
            input.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

To practice!

¡Siempre hacia lo alto!

P2Txx: programmer_txt

create a JAVA software that:

Create a txt file in C: \ named "programmer.txt" where the programmer's personal data is stored, for each line:

- Name:
- Biography:
- E-mail:
- Phone:

Construir un programa en JAVA que,

Cree un archivo txt en C:\ con nombre “programador.txt” donde queden guardados los datos personales del programador, por cada línea :

- Nombre:
- Biografía:
- Email:
- Teléfono:

P2Txx: names_txt

create a JAVA software that:

ask the user to enter N amount of names of people and these names are saved in a file called "names.txt", the names must be saved one on each line.

Construir un programa en JAVA que,

le solicite al usuario que ingrese N cantidad de nombres de personas y dichos nombres se guarden en un archivo denominado “names.txt”, los nombres deben ser guardados uno en cada línea.

P2Txx: namesV2_txt

create a JAVA software that:

Load in a vector the names of the file named “names.txt” (created in the previous exercise) and visualize them on the screen, the software should show a menu where it gives the user the option to enter new names or delete existing ones.

Construir un programa en JAVA que,

Cargue en un vector los nombres del archivo denominado “names.txt” (creado en el ejercicio anterior) y visualícelos en pantalla, el software debe mostrar un menú donde le de la opción al usuario de ingresar nuevos nombre o borrar los existentes.

P2Txx: even_numbers_text

Create a JAVA software that:

Show the user a menu with the following options:

1. Create an array with the even numbers existing between two numbers (the user must indicate what those numbers are), the result of the array must be saved in a file whose name will be chosen by the user.
2. Display the contents of the created text file on the screen.
3. Exit the program.

Construir un programa en JAVA que,

Muestre al usuario un menú con las siguientes opciones:

1. Crear un array con los números pares existentes entre dos números (el usuario debe indicar cuales son esos números), el resultado del array debe ser guardado en un archivo cuyo nombre lo elegirá el usuario.
2. Mostrar por pantalla el contenido del archivo de texto creado.
3. Salir del Programa.

P2Txx: lower_case_text

Create a JAVA software that:

Ask the user to type the name of the file by keyboard and also type a text to be saved in the file.
After creating the file with the information entered, the software should display the text of the file on the screen but varying between upper and lower case.

• For example, if I write:

"welcome to tunja, where you will never want to go"

"*WeLcOmE tO tUnJa, WhErE yOu NeVeR wAnT tO gO"*

Construir un programa en JAVA que,

Solicite al usuario que escriba el nombre del fichero por teclado y también escriba un texto que se guardara en el archivo.
Después de crear el archivo con la información introducida, el software deberá mostrar por pantalla el texto del fichero pero variando entre mayúsculas y minúsculas.

• Por ejemplo, si escribo:

"bienvenidos a tunja, donde nunca se querrá ir"

"BiEnVeNiDoS a TuNjA, DoNdE nUnCa Se QuErRá Ir".


PROYECTO
CRUD
PERSISTENCIA
MATRICES
OPERACIONES ARREGLOS
ARREGLOS
MÉTODOS
CICLOS
CONCEPTOS BÁSICOS


GOOD JOB

UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

¡Siempre hacia lo alto!

USTATUNJA.EDU.CO

@santotomastunja