	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M.C. Jorge Rommel Santiago Arce
Asignatura:	Programación Orientada a Objetos
Grupo:	2
No de Práctica(s):	10
Integrante(s):	Cruz Rangel Leonardo Said Ibañez Guzman Osvaldo Maya Ortega Maria Fernanda van der Werff Vargas Pieter Alexander
No. de Equipo de cómputo empleado:	45-48
No. de Lista o Brigada:	7
Semestre:	2020-1
Fecha de entrega:	28/10/2019
Observaciones:	

CALIFICACIÓN: _____

Práctica 10: “Excepciones y errores”

Objetivos:
Identificar bloques de código propensos a generar errores y aplicar técnicas adecuadas para el manejo de situaciones excepcionales en tiempo de ejecución.

Desarrollo:

Ejercicio 2. El Error StackOverflow :
Este tipo de errores aparece cuando el programa por sí solo no tiene un fin, en este caso no tendrá un fin el programa ya que se vuelve recursivo por el método doNotCodeThis así que el programa manda este error a pantalla.

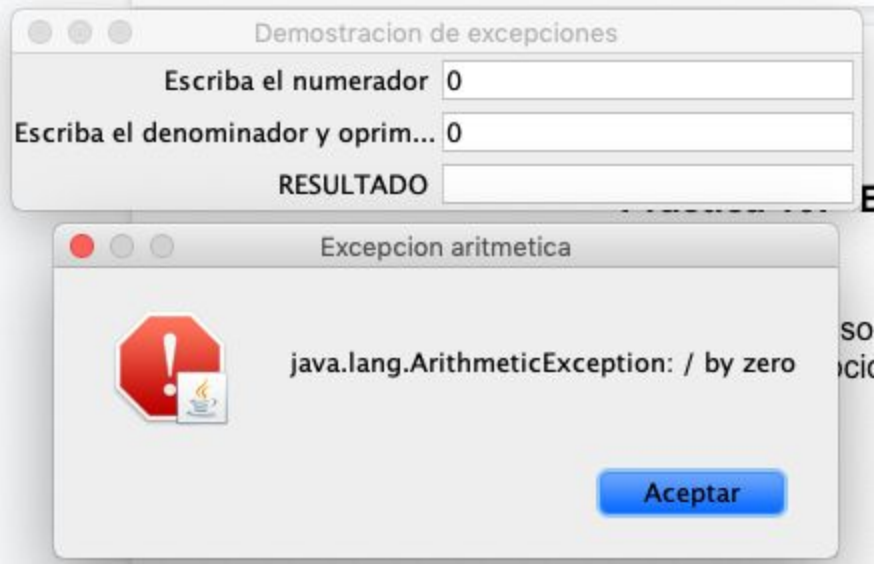
```java
public class Error{
    public static void main(String[] args){
        doNotCodeThis(1);
    }

    public static void doNotCodeThis(int num){
        doNotCodeThis(1);
    }
}
```

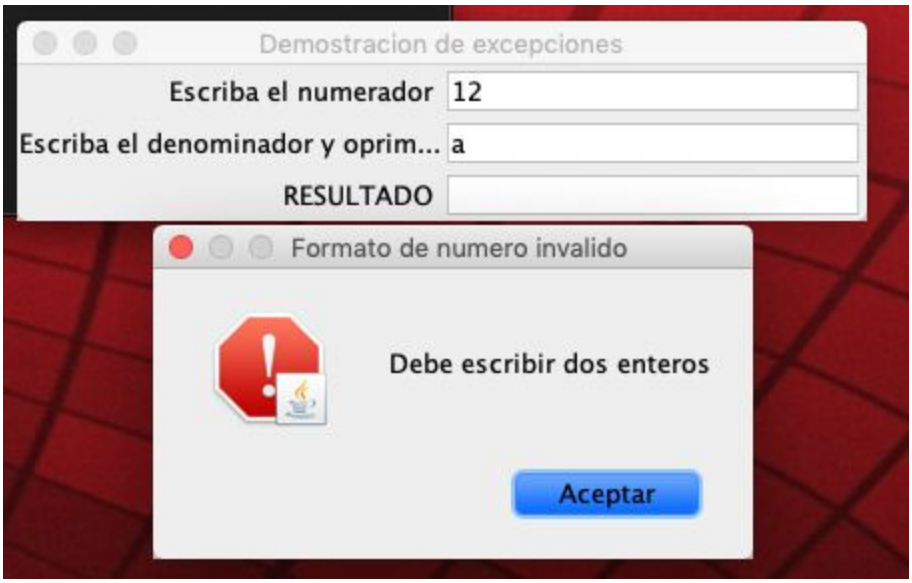
```
Georgia46:desktop poo02alu19$ java Error
Exception in thread "main" java.lang.StackOverflowError
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
        at Error.doNotCodeThis(Error.java:7)
```

Ejercicio 3. Try catch:

En este código se hacen las divisiones según sean los numeradores y denominadores dados por el usuario. Cuando ingresamos como denominador el número cero, nos muestra el mensaje siguiente:



Y en el caso de que se ingrese alguna letra:



Lo anterior se debe al uso de la instrucción **try-catch()** en la cual se intenta hacer la división de los números y en el caso de obtener(cachar) una excepción aritmética se muestre cierto mensaje. O si se tiene alguna división con un carácter no numérico y la excepción *NumberFormatException* se muestra en el bloque la cadena dada.

Ejercicio 4. Try catch - finally:

De manera general el **try - catch** sirve para mostrar Excepciones o errores , por lo que lo que esté dentro del *TRY* si llegara a tener algún error se ejecutará el código que se encuentre en el *CATCH*, además existe *FINALLY* que sin importar si existe un error o no estas lineas de código se ejecutarán.

```
Granada47:Desktop poo02alu37$ javac Excepciones.java
Granada47:Desktop poo02alu37$ java Excepciones
El metodo lanzarExcepcion
La excepcion se manejo en el metodo lanzarExcepcion
Finalmente se ejecuto en lanzarExcepcion
La excepcion se manejo en main
El metodo noLanzaExcepcion
Finalmente se ejecuto en noLanzaExcepcion
Fin del metodo noLanzaExcepcion
```

Ejercicio 5. Tracing:

Al compilar el archivo *TraceExceptions.java* la terminal nos regresa 20 errores, que se pueden arreglar haciendo 3 cambios al archivo.

El primer error se encuentra en la primera línea del archivo, donde se omitió la palabra reservada *class* para declarar el nombre de la clase.

```
public TraceExceptions{
```

El segundo error se encuentra en la línea 30, con la omisión de los paréntesis en la declaración del método *metodo3*.

```
public static void metodo3 throws Exception{
```

El último error lo encontramos dentro del ciclo *for* de la línea 13. Donde se está creando un objeto con una clase que no es adecuada para instanciarlo con el arreglo. Se debe intercambiar la clase por *StackTraceElement*.

```
for(int i = 0; i < elementosRastreo.length; i++){
    TraceExceptions elementoActual = elementosRastreo[i];
```

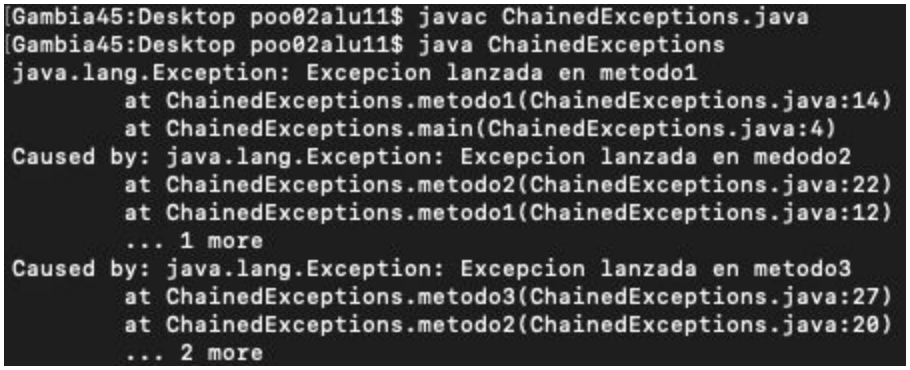
Se hace un llamado a 3 métodos, dentro de una instrucción *try* dentro del método main. El tercer método devuelve una excepción de nombre “La excepcion se lanzo en metodo3”. Al cachar la excepción se crea un arreglo de *StackTraceElement* que mediante un ciclo for imprime todos los llamados que se hicieron hasta llegar a esa línea.

```
Granada47:Desktop poo02alu37$ javac TraceExceptions.java
Granada47:Desktop poo02alu37$ java TraceExceptions
La excepcion se lanzo en metodo3


Rastreo de pila proveniente de getStacTrace:
Clase           Archivo                 Linea   Metodo
TraceExceptions TraceExceptions.java    31      metodo3
TraceExceptions TraceExceptions.java    27      metodo2
TraceExceptions TraceExceptions.java    23      metodo1
TraceExceptions TraceExceptions.java    4       main
```

Ejercicio 6. Propagación:

La clase *ChainedExceptions* tiene un método principal que intenta llamar a *metodo1*, sin embargo, se lanza la excepción y *metodo1* intenta llamar al *metodo2*, se lanza la excepción y *metodo2* intenta llamar al *metodo3*, se lanza la excepción del *metodo2* y también la del *metodo3*. Entonces la salida final del programa nos muestra la pila de excepciones.



Conclusiones:

Cruz Rangel Leonardo Said:

En la programación se producen muchos tipos de errores, producidos por el programador o por el usuario, éstos pueden causar que el programa termine su ejecución bruscamente, situación que no se quiere, por lo que es necesario tratar algunos de los errores que se producen. En java, los bloques try-catch son de utilidad para mantener el flujo del programa posterior a la presencia de alguna excepción.

Ibañez Guzman Osvaldo:

Cuando uno programa es muy importante el manejo de los errores ya que así se pueden evitar muchos conflictos tanto en la lógica como en la ejecución, además que siempre que se cree un programa se debe pensar en todo tipo de usuarios y que el programa debe ser lo mejor posible.

En esta práctica se vio claramente algunos errores que se pueden manejar y cómo es que estos errores resultan.

Maya Ortega María Fernanda (づ｡◕‿‿◕｡)づ

Al codificar soluciones muchas veces el usuario puede proporcionar datos erróneos o con un formato imprevisto, lo que puede llegar a generar excepciones las cuales en un buen programa deben ser manejadas para que el programa no se detenga. Es por eso, que en esta práctica vimos diferentes formas de manejar errores y excepciones en distintos códigos.

van der Werff Vargas Pieter Alexander:
Para evitar que un programa se detenga es posible anticipar los errores que puedan ocurrir a la hora de ejecución, de manera que haya código que resuelva qué hacer en caso de que se genere una excepción por algún error en particular. Esto es útil cuando se sabe que un usuario va a manipular el programa y se quiere evitar que realice acciones que sean “ilegales” para la máquina virtual de java y puedan detener la ejecución.