|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Ing. Guadalupe Lizeth Parrales Romay |
| *Asignatura:* | Programación Orientada a Objetos |
| *Grupo:* | 04 |
| *No de Práctica(s):* | 10 |
| *Integrante(s):* | Dávila Ortega Jesús Eduardo - No. Cuenta: 317199860  Díaz Hernández Marcos Bryan - No. Cuenta: 317027253  Pareja Ávila Emiliano - No. Cuenta: 317081345  Vázquez Zavala Oliver Alexis - No. Cuenta: 317202263 |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* | 12,14, 30, 37 |
| *Semestre:* | 2021-1 |
| *Fecha de entrega:* | 30 de enero del 2021 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_

**Objetivo de la práctica.**

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

**Introducción.**

Existen muchos tipos de errores que pueden estar presentes en un programa y no todos estos pueden ser detectados por el compilador, por tanto, durante el desarrollo pueden darse dos tipos de errores: los de tiempo de compilación y los de tiempo de ejecución. En general es preferible que los lenguajes de compilación, como Java, estén diseñados de tal manera que la compilación pueda detectar el máximo número posible de errores, es preferible que los errores de tiempo de ejecución se deban a situaciones inesperadas y no a descuidos del programador, errores de tiempo de ejecución siempre habrá, y su gestión a través de excepciones es fundamental en cualquier lenguaje de programación actual.

En general podemos clasificar a los errores en:

* Errores de Sintaxis: Un error de sintaxis se produce al escribir, incorrectamente, alguna parte del código fuente de un programa, de forma que, dicho error impedirá, tanto al compilador como al intérprete, traducir la instrucción que se intenta definir en dicho bloque.
* Errores en tiempo de ejecución: son aquellos que ocurren de manera inesperada: disco duro lleno, error de red, cast inválido, etc.
* Errores lógicos: Cuando el programa no da ningún error, pero los resultados no son los esperados. Puede ser porque el algoritmo ha sido incorrectamente implementado o porque el algoritmo estaba mal hecho.

Las excepciones son eventos que ocurren durante la ejecución de un programa y hacen que éste salga de su flujo normal de instrucciones, este mecanismo permite tratar los errores de una forma selecta, ya que separa el código para el tratamiento de errores del código normal del programa, se dice que una excepción es lanzada cuando se produce un error, y esta excepción puede ser capturada para tratar dicho error. En java tenemos dos tipos de excepciones, las verificadas, que deben ser capturadas o bien especificar que pueden ser lanzadas de forma obligatoria, de lo contrario obtendremos un error de compilación, y las no verificadas, las cuales dejan al desarrollador la decisión de si capturarlas o no, todas ellas descienden de la clase Throwable, la cual tiene dos descendientes directos:

-Error: Se refiere a errores graves en la máquina virtual de Java, como por ejemplo fallos al enlazar con alguna librería.

-Exception: Representa errores que no son críticos y por lo tanto pueden ser tratados y continuar la ejecución de la aplicación. La mayoría de los programas Java utilizan estas excepciones para el tratamiento de los errores que puedan ocurrir durante la ejecución del código.

Dentro de Exception, cabe destacar una subclase especial de excepciones denominada RuntimeException, de la cual derivarán todas aquellas excepciones referidas a los errores que comúnmente se pueden producir dentro de cualquier fragmento de código, como por ejemplo hacer una referencia a un puntero null, o acceder fuera de los límites de un array.

Cuando un fragmento de código sea susceptible de lanzar una excepción y queramos tratar el posible error producido o bien por ser una excepción de tipo verificada debamos capturarla, podremos hacerlo mediante la estructura try-catch-finally, que consta de tres bloques de código:

-Bloque try: Contiene el código regular de nuestro programa que puede producir una excepción en caso de error.

-Bloque catch: Contiene el código con el que trataremos el error en caso de producirse.

-Bloque finally: Este bloque contiene el código que se ejecutará al final tanto si se ha producido una excepción como si no lo ha hecho. Este bloque se utiliza para, por ejemplo, cerrar algún fichero que haya podido ser abierto dentro del código regular del programa, de manera que nos aseguremos que tanto si se ha producido un error como si no este fichero se cierre, este bloque no es obligatorio ponerlo.

Para el bloque catch además deberemos especificar el tipo o grupo de excepciones que tratamos en dicho bloque, pudiendo incluir varios bloques catch, cada uno de ellos para un tipo/grupo de excepciones distinto, la sintaxis del bloque try-catch-finally es:

*try {*

*// Código regular del programa*

*// Puede producir excepciones*

*} catch(TipoDeExcepcion1 e1) {*

*// Código que trata las excepciones de tipo*

*// TipoDeExcepcion1 o subclases de ella.*

*// Los datos sobre la excepción los encontraremos*

*// en el objeto e1.*

*} catch(TipoDeExcepcion2 e2) {*

*// Código que trata las excepciones de tipo*

*// TipoDeExcepcion2 o subclases de ella.*

*// Los datos sobre la excepción los encontraremos*

*// en el objeto e2.*

*...*

*} catch(TipoDeExcepcionN eN) {*

*// Código que trata las excepciones de tipo*

*// TipoDeExcepcionN o subclases de ella.*

*// Los datos sobre la excepción los encontraremos*

*// en el objeto eN.*

*} finally {*

*// Código de finalización (opcional)*

*}*

Cabe mencionar que cuando se tiene más de un bloque catch dentro del manejador de excepciones, se debe de tener cuidado en el orden de captura, ya que las excepciones se deben acomodar de las más específicas a las más generales, de lo contrario no se podrá hacer una compilación exitosa.

**Análisis de los ejemplos.**

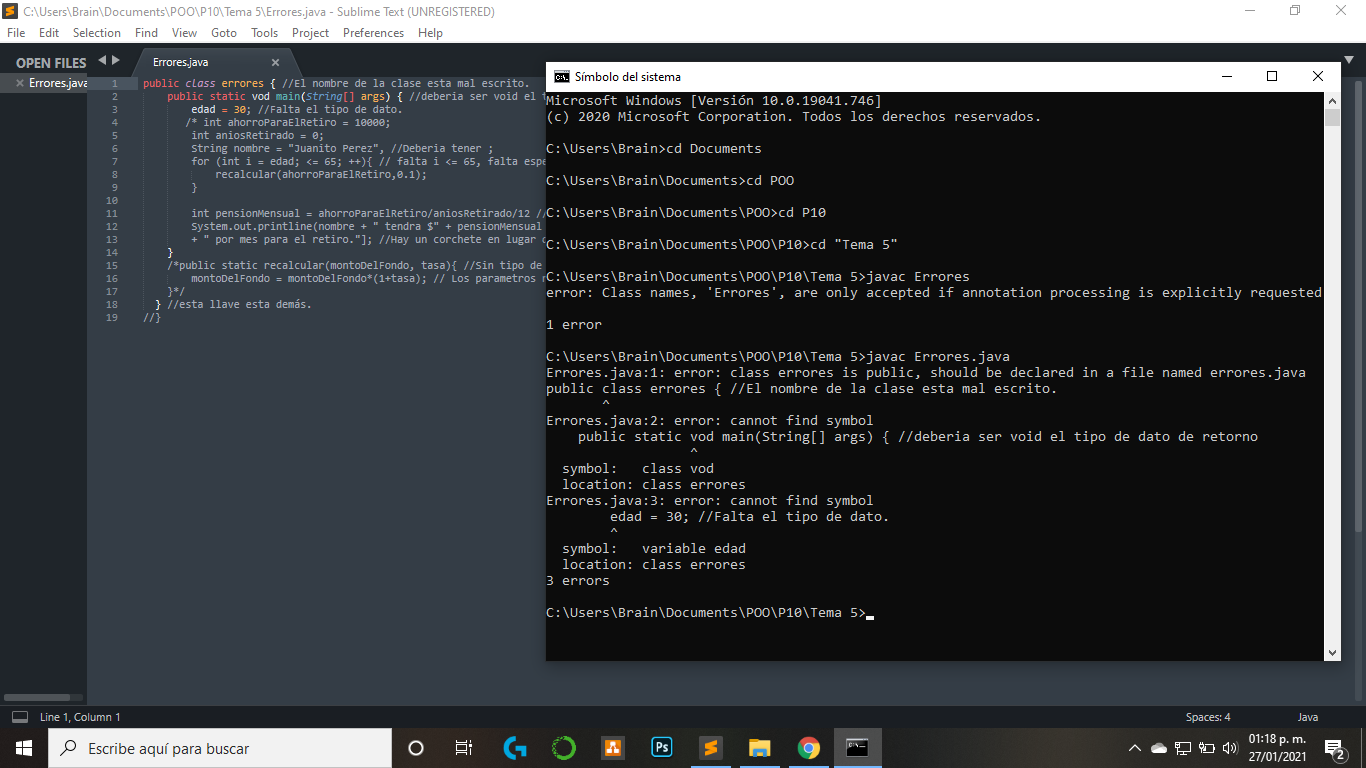
**Ejemplo Errores 1**

* **Análisis**

El primer ejemplo de la práctica fue Errores.java, el cual mostraba los errores de sintaxis más comunes que podemos cometer como programadores. Desde errores de punto y coma, hasta el nombre de la clase. Los errores de sintaxis son aquellos que ocurren cuando no se cumplen con las reglas de la escritura dentro del lenguaje de programación.

Las líneas escritas sirven para demostrar cómo el controlador de excepciones puede identificar los errores más comunes al momento de compilar, ya que estos no permiten que la compilación se lleve a cabo.

* **Ejecución del programa**

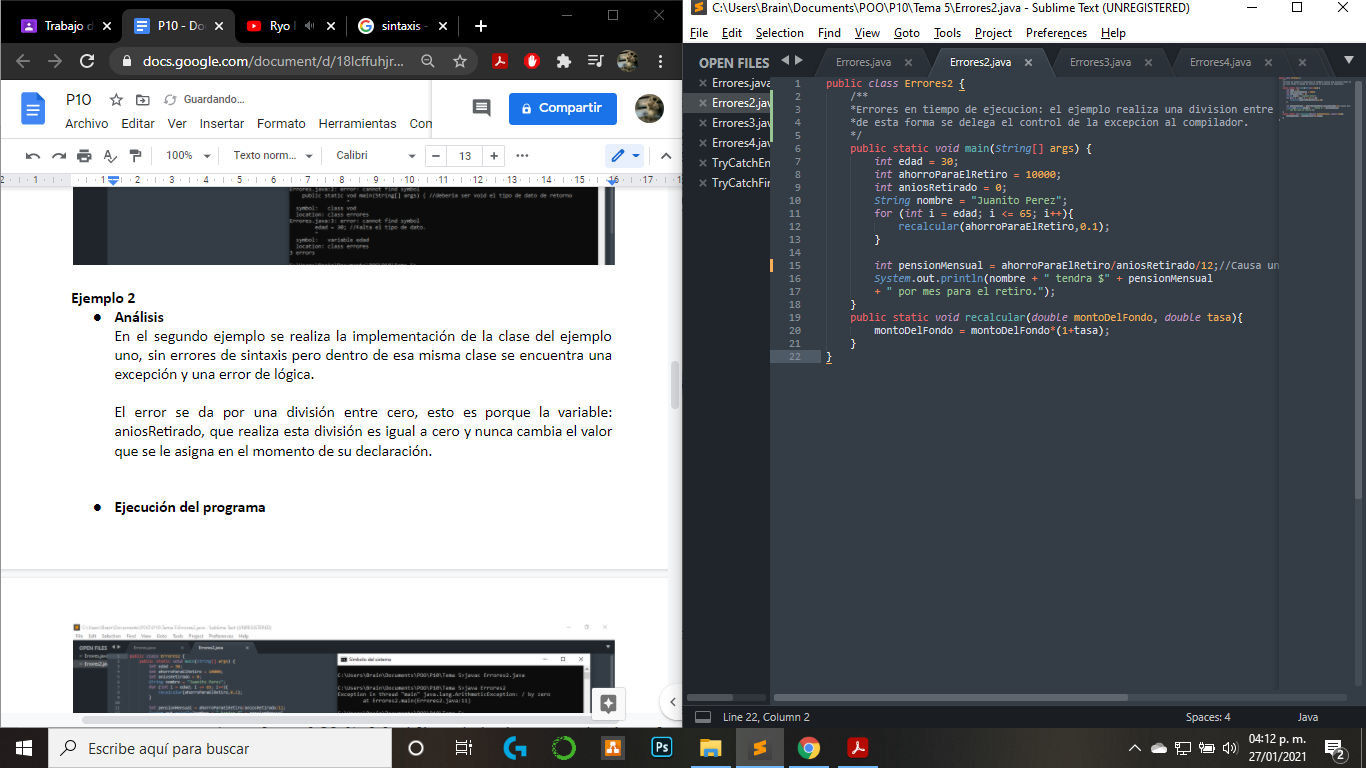
****

**Ejemplo Errores 2**

* **Análisis**

En el segundo ejemplo se realiza la implementación de la clase del ejemplo uno, sin errores de sintaxis pero dentro de esa misma clase se encuentra una excepción.

El error se da por una división entre cero, esto es porque la variable: aniosRetirado, que realiza esta división es igual a cero y nunca cambia el valor que se le asigna en el momento de su declaración.



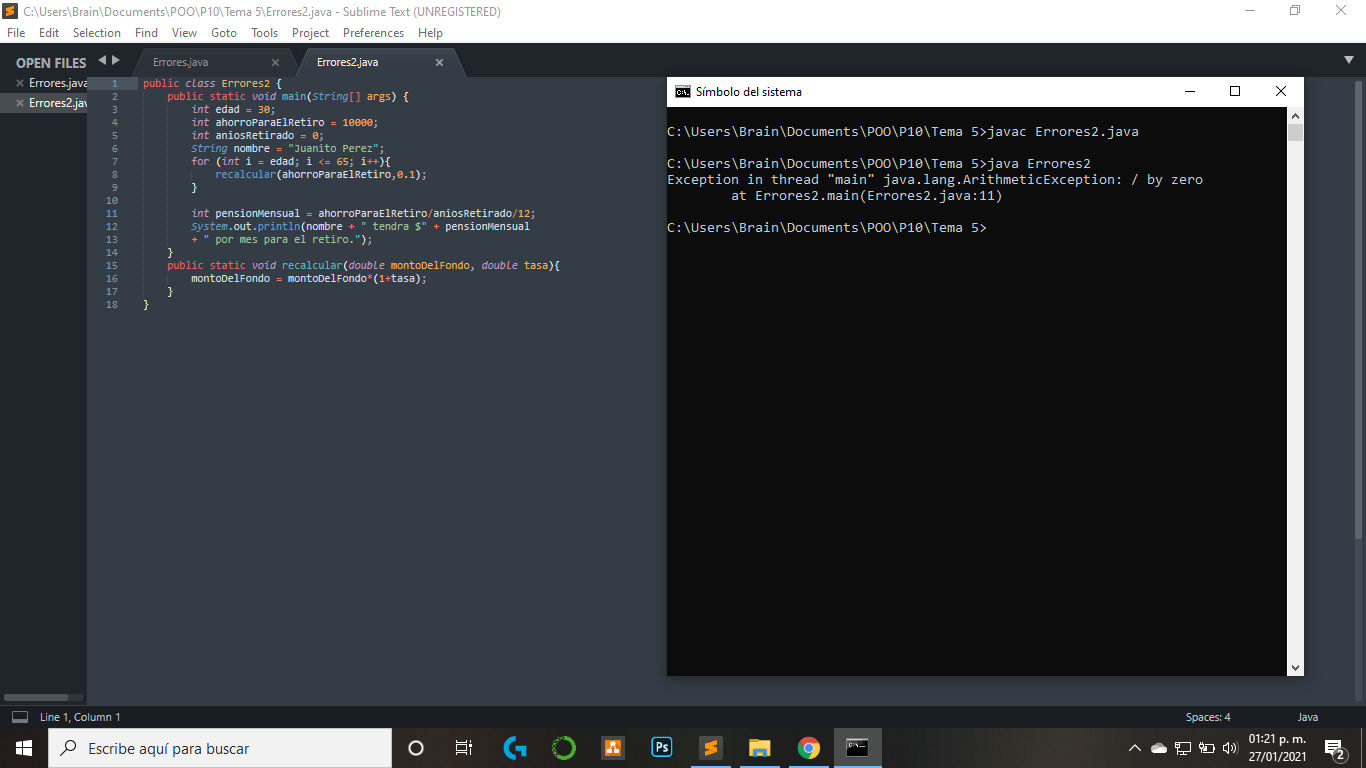
Variable causante de la excepción.

Para este caso la excepción no es controlada, es decir que al momento de compilar este programa se compila con éxito ya que solo se analizan las excepciones verificadas, lo que en este programa no aplica, ya que solo se ejecuta una división.

En el momento de la ejecución se realizan las operaciones y la división entre cero se ejecuta lo que resulta en una excepción aritmética, lo que significa que algo rompió la ejecución normal del programa, es decir la división entre cero.

Como esta excepción no se controla o identifica por parte del programador, es controlada por el manejador de excepciones pero este lo que hace es identificar el tipo de excepción y notificar que detuvo el correcto funcionamiento del programa.

* **Ejecución del programa**

****

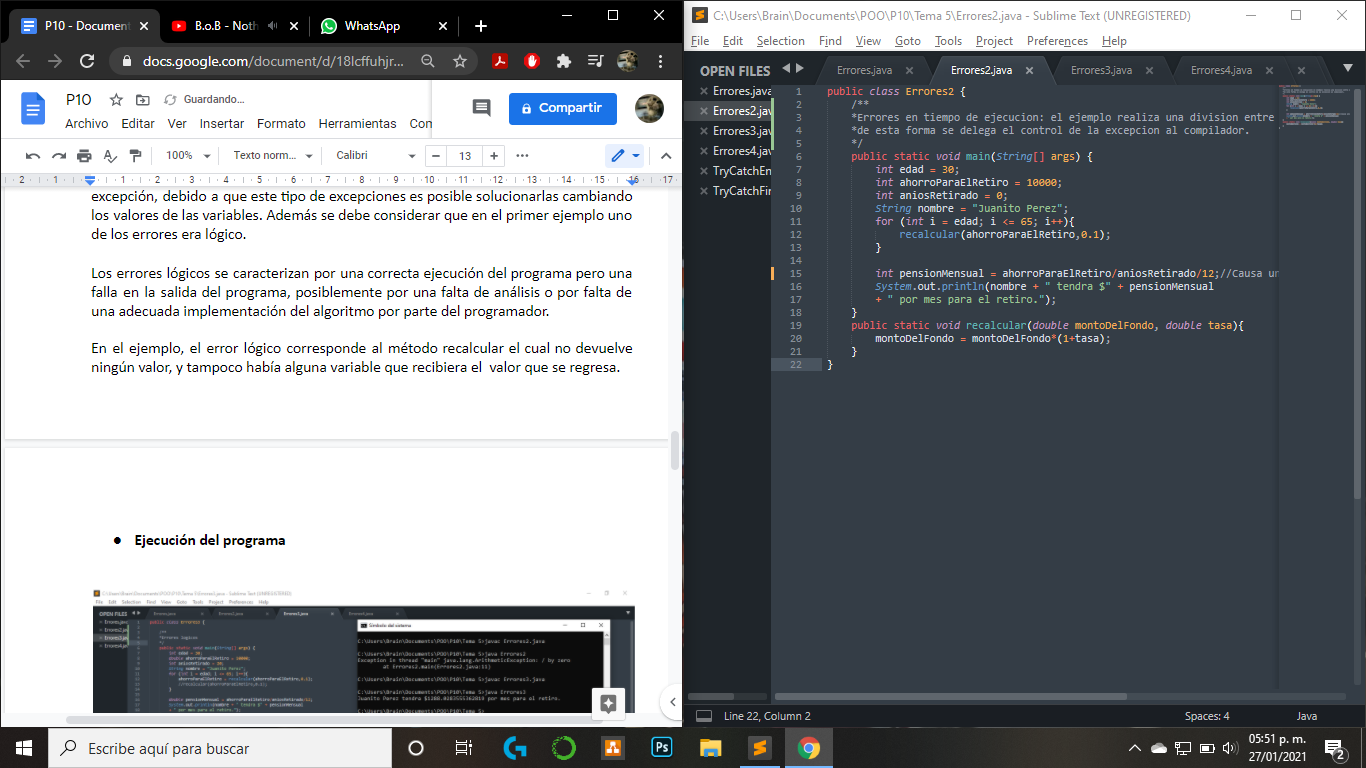
**Ejemplo Errores 3**

* **Análisis**

Continuando con la versión del primer ejemplo, en este ejemplo ya no tiene la excepción, debido a que este tipo de excepciones es posible solucionarlas cambiando los valores de las variables. Además se debe considerar que en el primer ejemplo uno de los errores era lógico.

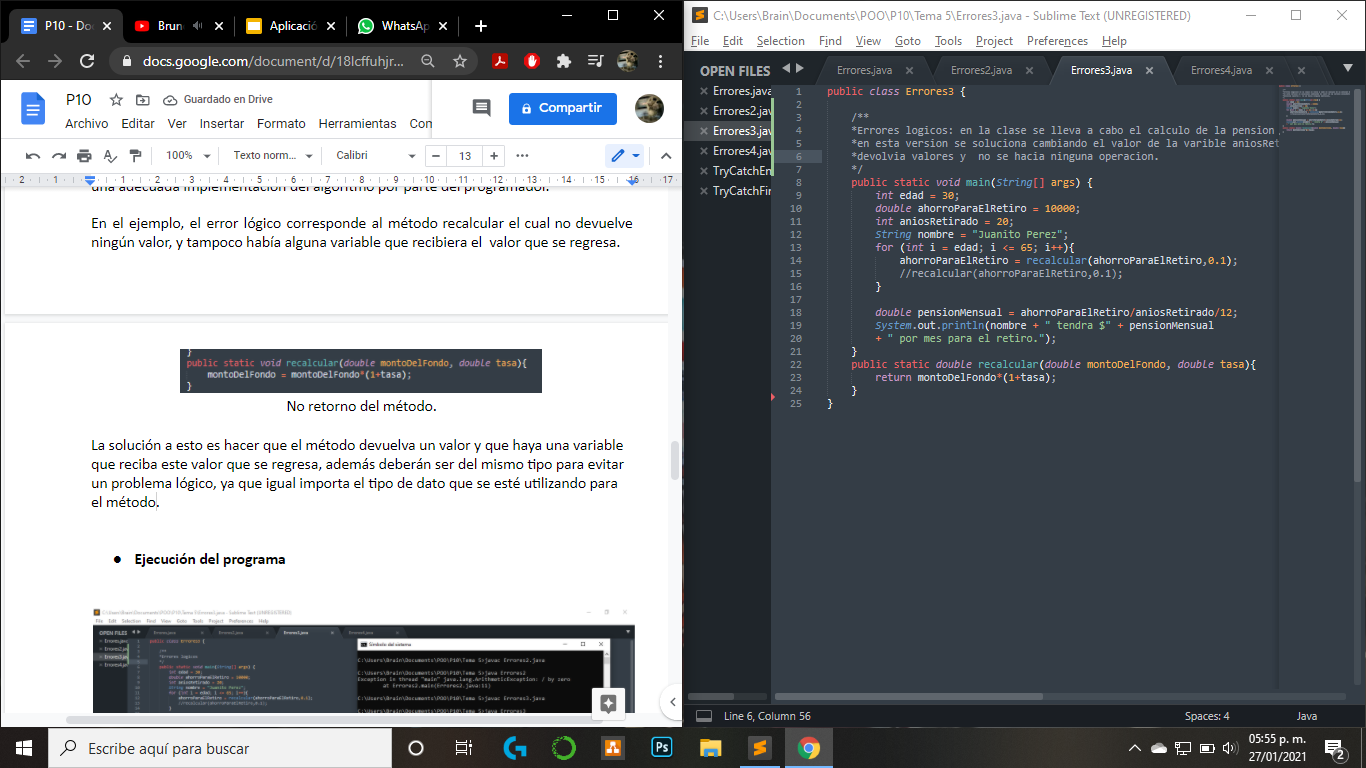
Los errores lógicos se caracterizan por una correcta ejecución del programa pero una falla en la salida del programa, posiblemente por una falta de análisis o por falta de una adecuada implementación del algoritmo por parte del programador.

En el ejemplo, el error lógico corresponde al método recalcular el cual no devuelve ningún valor, y tampoco había alguna variable que recibiera el valor que se regresa.



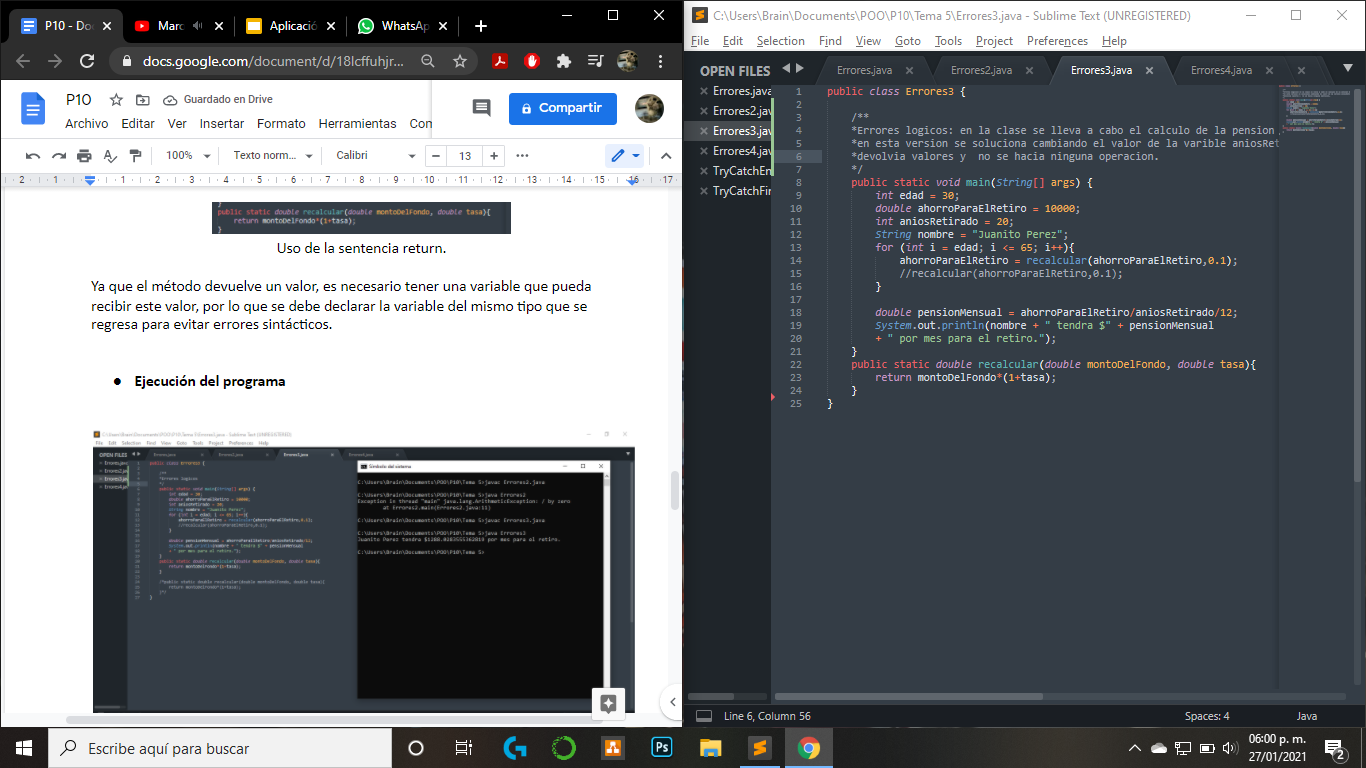
No retorno del método.

La solución a esto es hacer que el método devuelva un valor y que haya una variable que reciba este valor que se regresa, además deberán ser del mismo tipo para evitar un problema lógico, ya que igual importa el tipo de dato que se esté utilizando para el método. Normalmente se utiliza la sentencia return, de igual manera se podría utilizar un atributo estático para que cada método sea capaz de modificarlo.



Uso de la sentencia return.

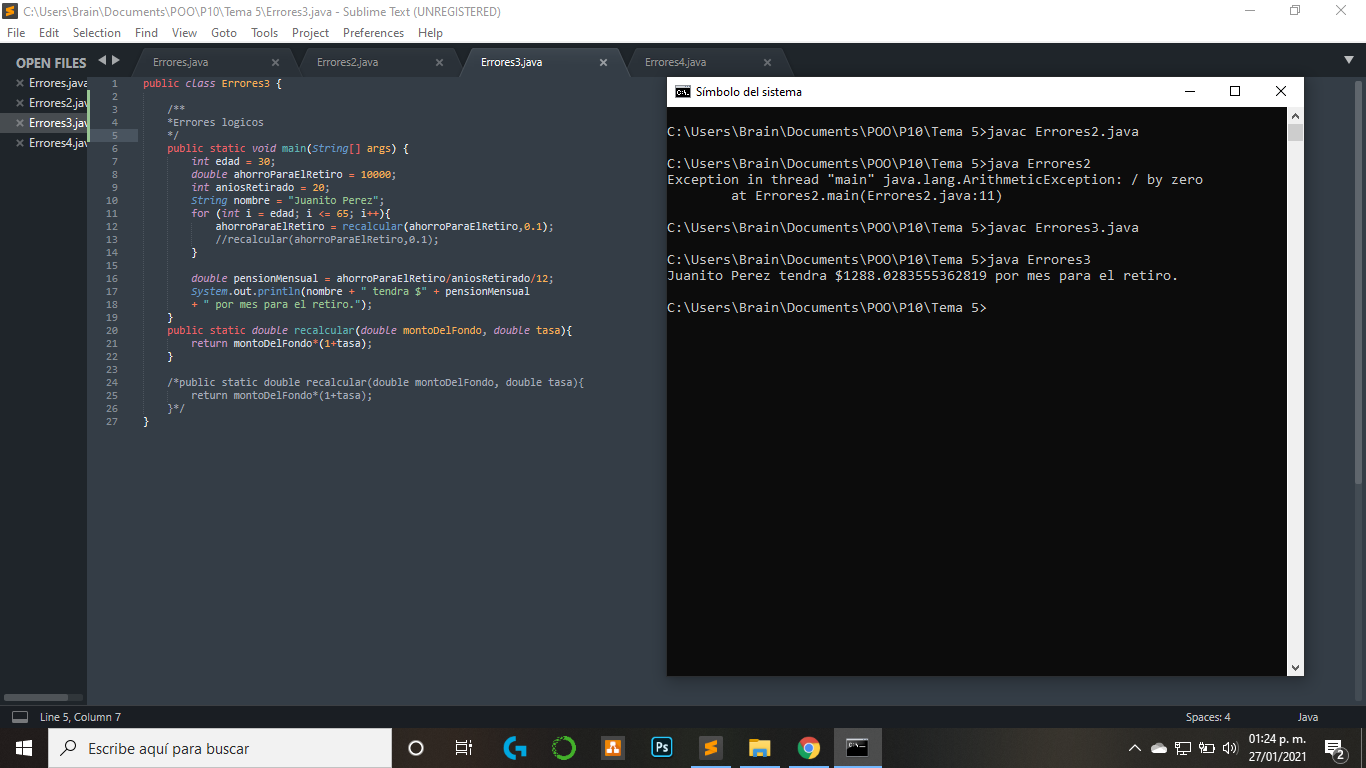
Ya que el método devuelve un valor, es necesario tener una variable que pueda recibir este valor, por lo que se debe declarar la variable del mismo tipo que se regresa para evitar errores sintácticos.



Variable que recibe el valor que regresa el método.

De esta forma se resuelve el problema lógico, ya que se obtiene el resultado esperado de la ejecución del programa. Este tipo de errores son los más comunes y más difíciles de resolver debido a que el compilador no notifica este tipo de errores ya que sintácticamente están bien, pero a nivel de resultados no cumplen con lo que espera el programador y es necesario hacer una depuración para saber qué hace cada una de las instrucciones y conocer donde está el error.

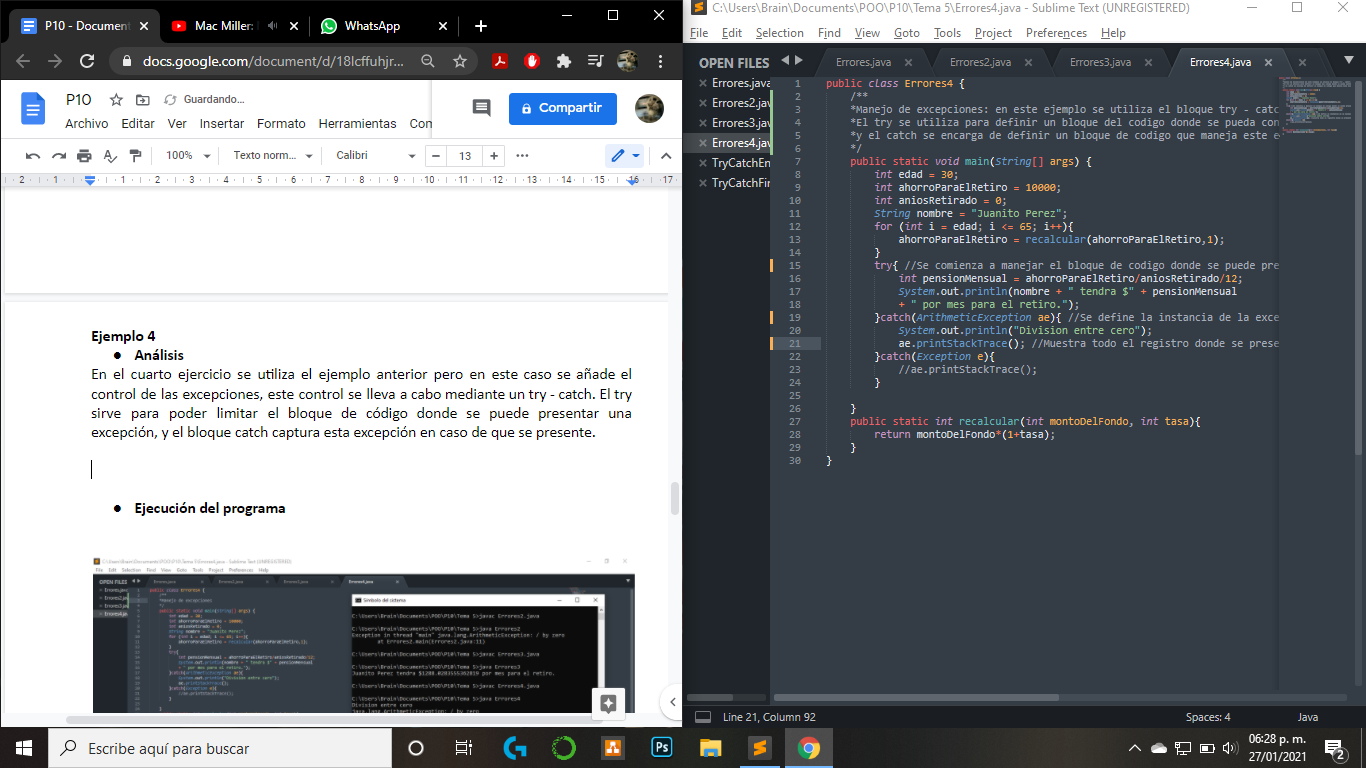
* **Ejecución del programa**

****

**Ejemplo Errores 4**

* **Análisis**

En el cuarto ejercicio se utiliza el ejemplo anterior pero en este caso se añade el control de las excepciones, este control se lleva a cabo mediante un try - catch. El try sirve para poder limitar el bloque de código donde se puede presentar una excepción, y el bloque catch captura esta excepción en caso de que se presente.

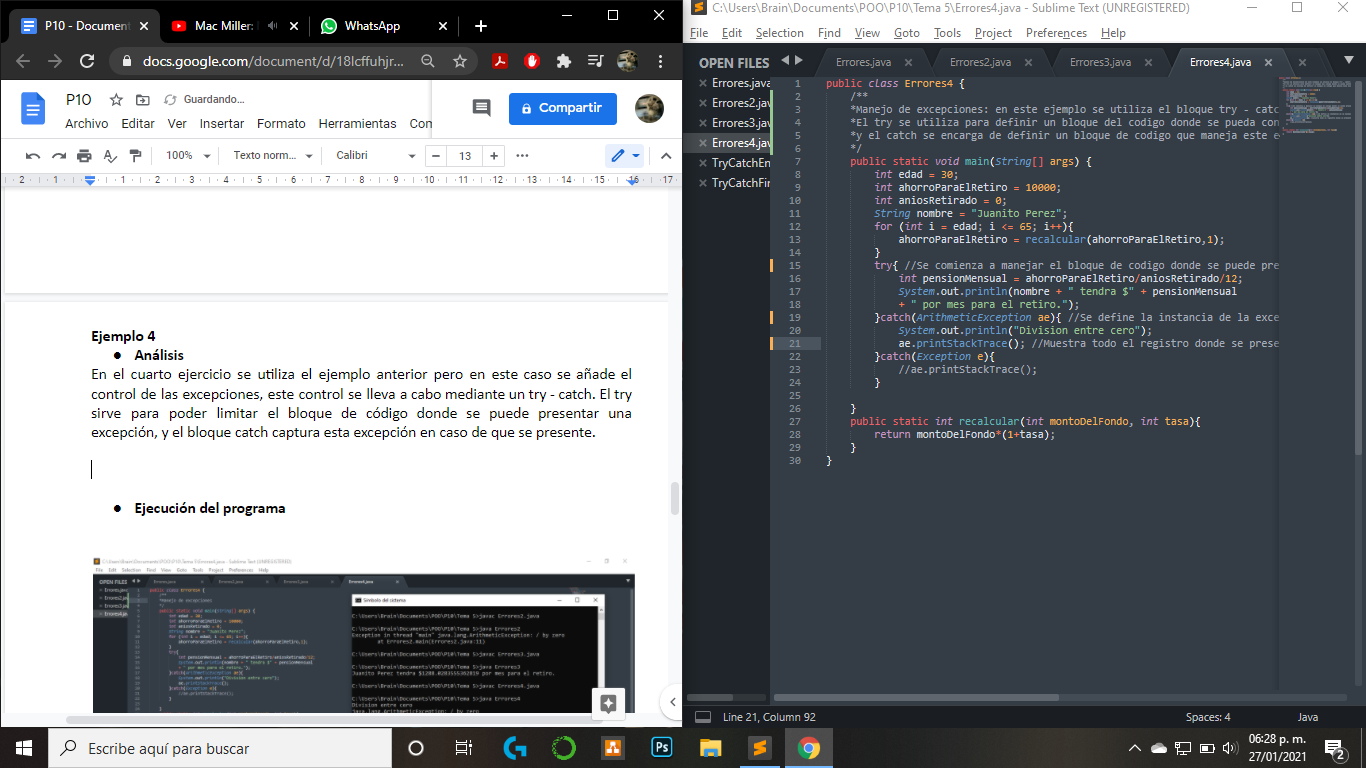


Manejo de la excepción.

Para el caso del ejemplo dos se presentaba una excepción por una división entre cero, en este caso se prova a propósito la excepción para poder capturarla, una vez que ocurre. Las excepciones se caracterizan por contener la información del error que las produjo, debido a que heredan de la Exception, que a la vez hereda de Throwable.

Debido a lo anterior es posible definir el tipo de excepción que se puede presentar, en este caso es de tipo Arithmetic, ya que se origina por un error en la división entre cero. De acuerdo al tipo es posible el realizar el manejo de excepciones en base a que tan específicas sean.

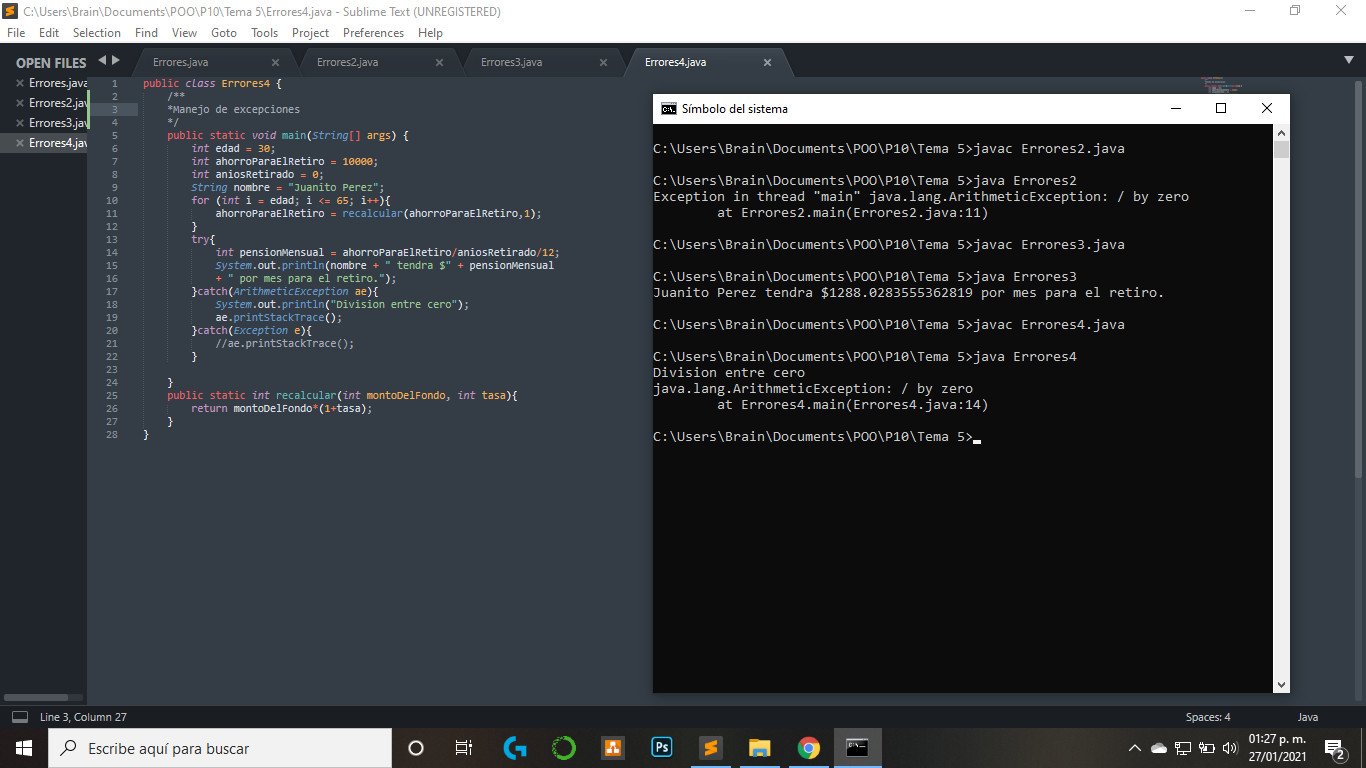
Existe un método que permite ver el registro de donde sucede la excepción mediante el método printStackTrace() que pertenece a la clase Throwable.



printStackTrace()

El bloque try - catch nos permite saber dónde se originan estos errores inesperados, y de igual forma nos indica que tipo de datos son lo que dan origen a las excepciones.

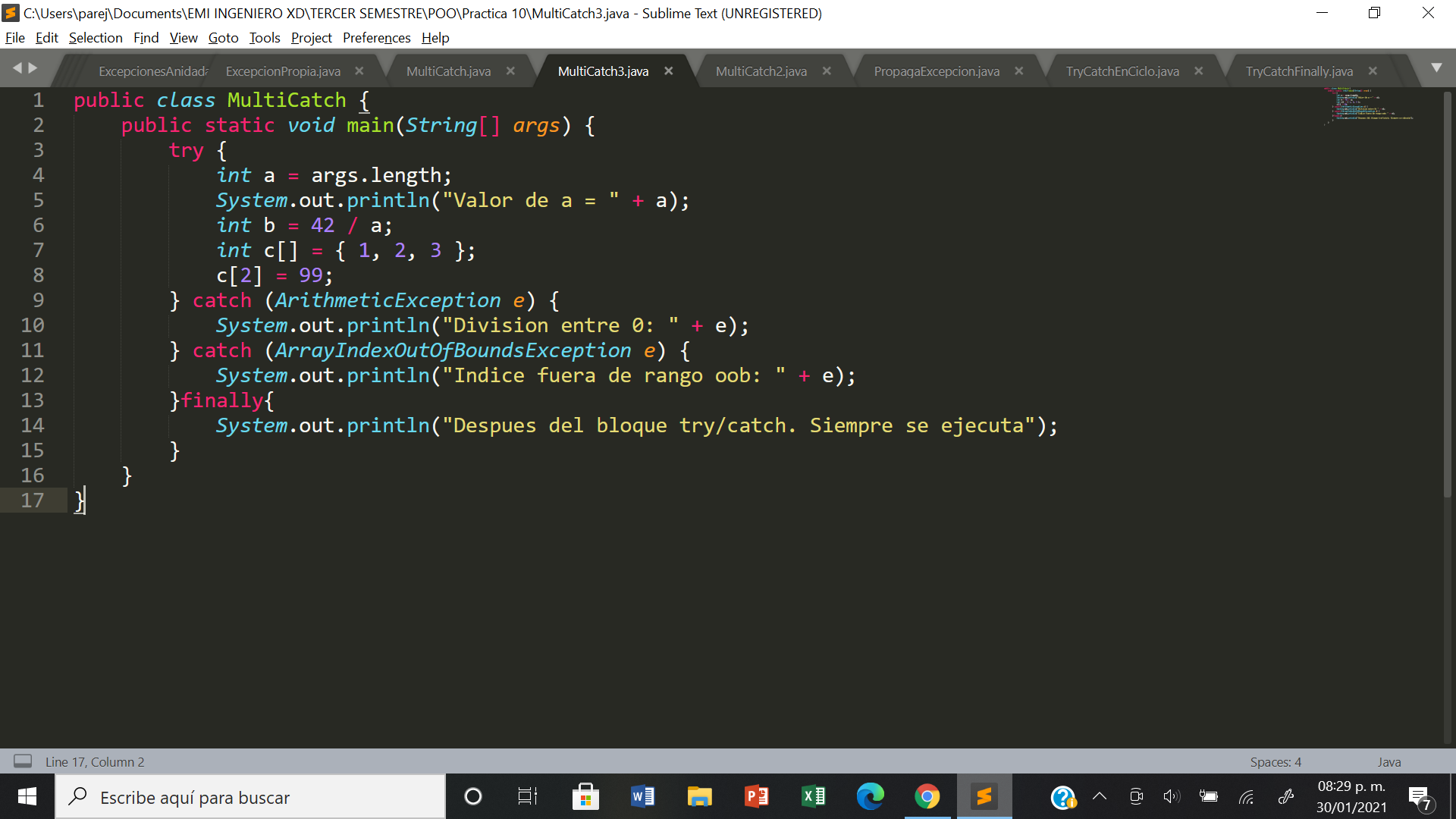
* **Ejecución del programa**

****

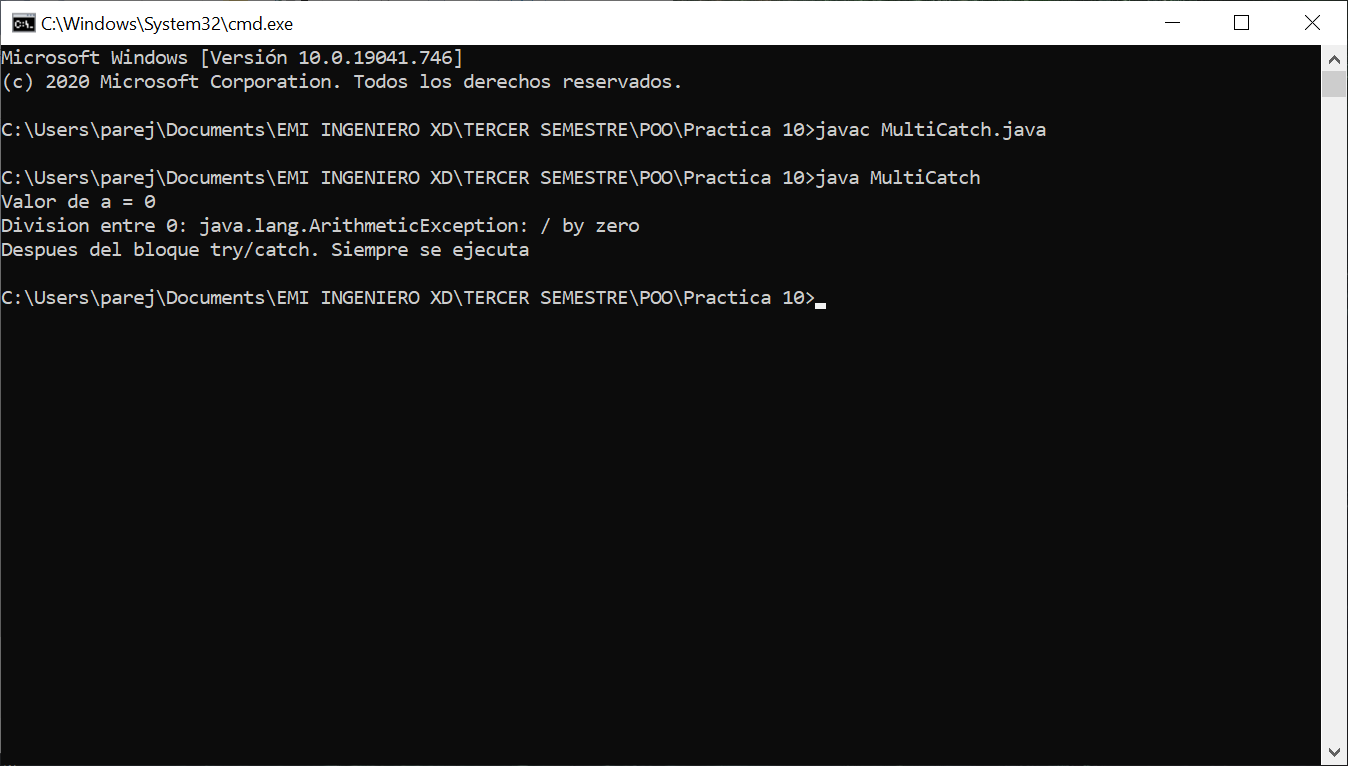
**Ejemplo Multicatch**

* **Análisis**

En el main, en un bloque try, se obtiene la longitud del tamaño de cadenas que se le pasan al método main por consola en una variable entera llamada a, y después en otro entero se almacena el resultado de la división de 42 entre a, en caso de que no se pasen cadenas, entonces el valor de a será 0, y si se hace una división entre 0 se dará una indeterminación, después se crea un arreglo de 3 elementos y se modifica el tercer elemento de este arreglo. En el bloque catch, primero se captura una excepción de la clase ArithmeticException, y esta se dará cuando en el bloque try exista una división entre cero, y después se imprime un mensaje a pantalla junto con el contenido de la excepción que se capturó, después hay otro bloque catch, que capturará una excepción de la clase ArrayIndexOutOfBoundsException, y esta se dará si se intenta obtener o modificar algún elemento en el arreglo en una posición que no exista, en caso de que se dé este error, se imprime a pantalla un mensaje junto con el contenido de la excepción que se capturó. Y después de estos dos bloques catch, hay un bloque finally, el cual siempre se ejecutará exista o no un error en el try. En este caso no pasaremos cadenas al main para comprobar que funciona el código



* **Ejecución del programa**



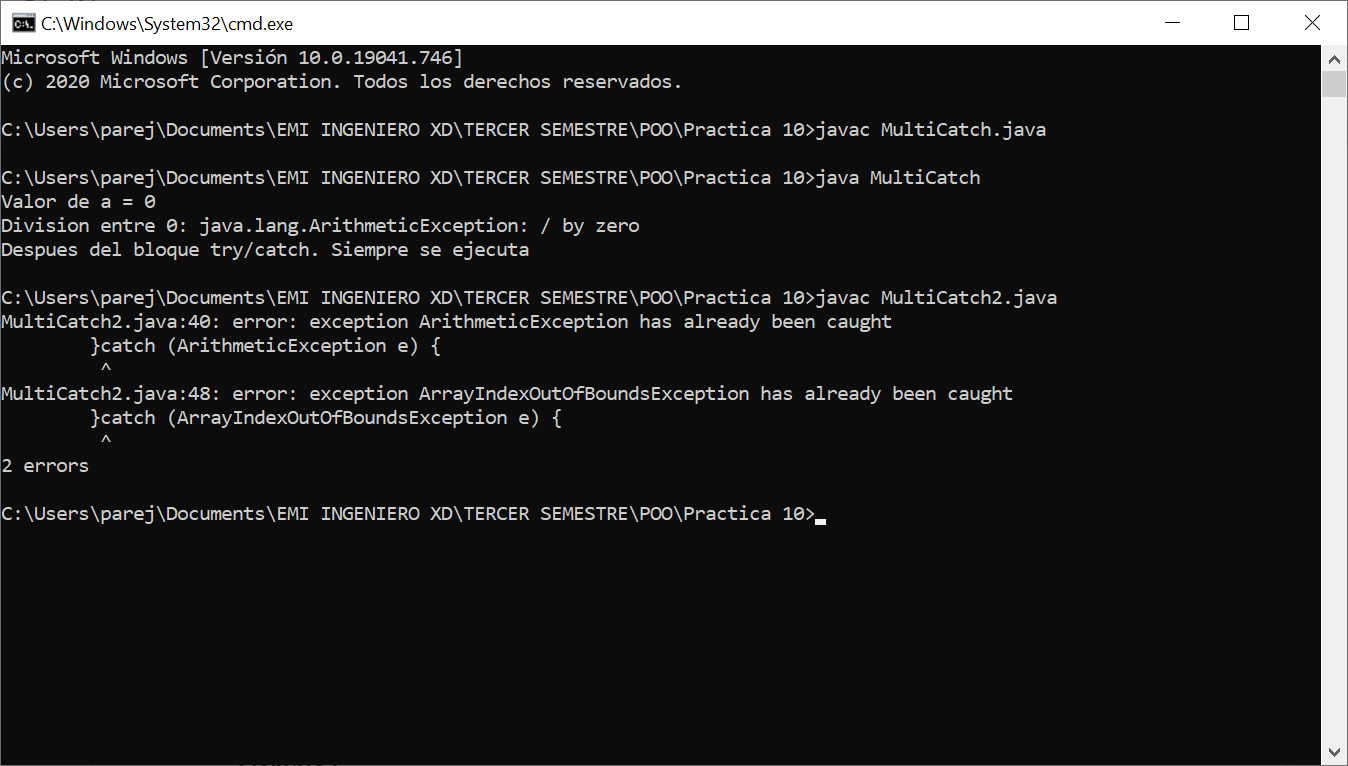
**Ejemplo MultiCatch2**

* **Análisis**

El ejemplo es el mismo que el anterior en el bloque try, sin embargo se hacen algunos cambios en el manejo de excepciones, y se comentó una excepción de la clase Exception, debido a que si no se comentara, habría un error con el manejo de excepciones, ya que las excepciones en un programa se manejan de la más específica hasta la más general, que en este caso son las de la clase Exception, después en otro bloque catch se captura una excepción de la clase RuntimeException, la cual capturará cualquier error que se dé en tiempo de compilación, aunque este tipo de excepción también es genérica, por tanto también generará un error, por el concepto visto anteriormente. Y finalmente en los dos últimos bloques catch se atrapan excepciones más específicas que las anteriores como una ArithmeticException y ArrayIndexOutOfBoundsException, las cuales se atraparán en caso de que exista una división entre cero, o que se trate de ingresar a una posición en el arreglo que no exista y finalmente hay un bloque finally el cual siempre se ejecutará y que imprimirá un mensaje. Al compilar el programa habrá un error, ya que el manejo de excepciones al capturar una excepción genérica antes que una específica es un error que se genera en tiempo de compilación.



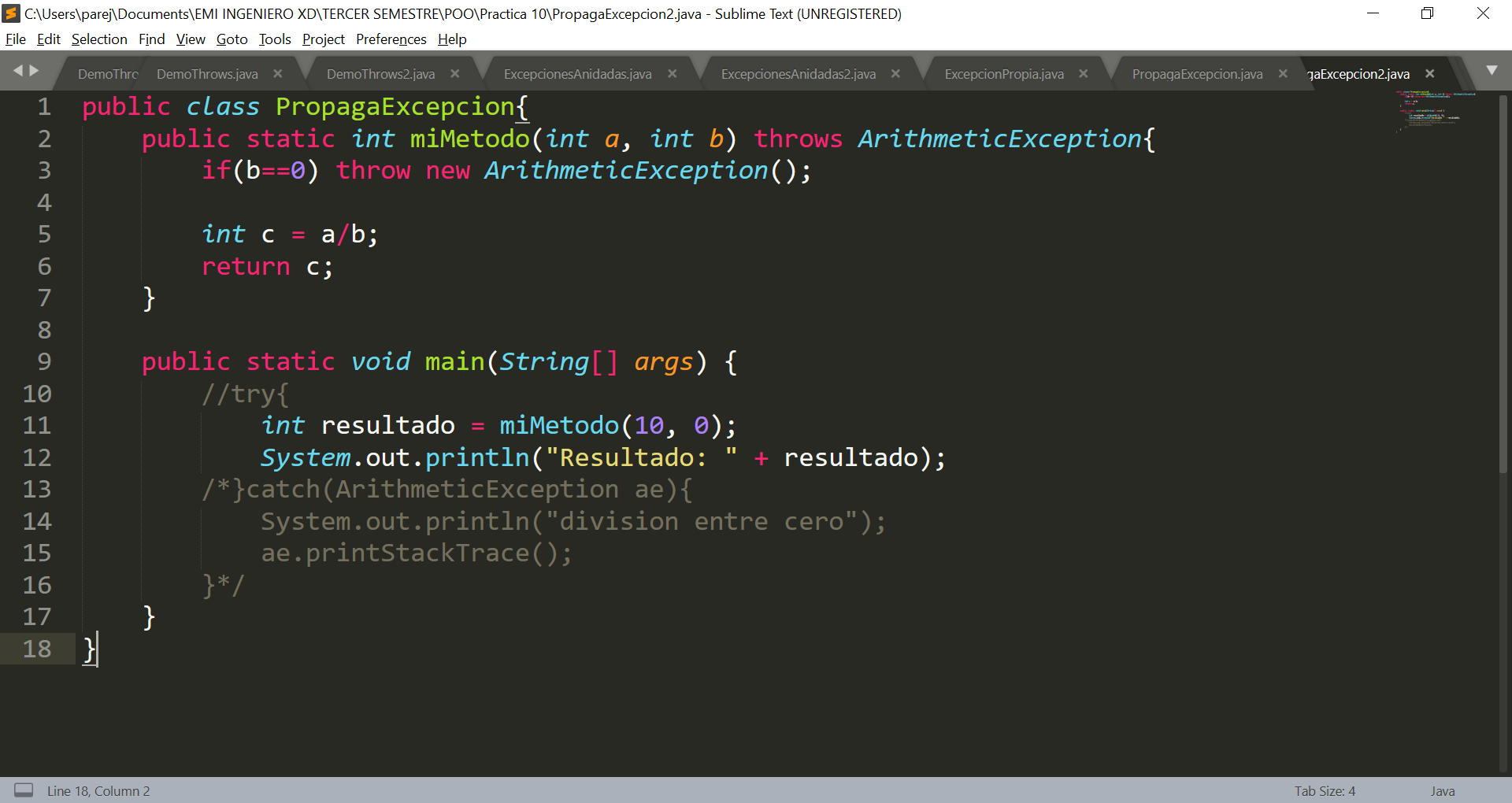
* **Ejecución del programa**



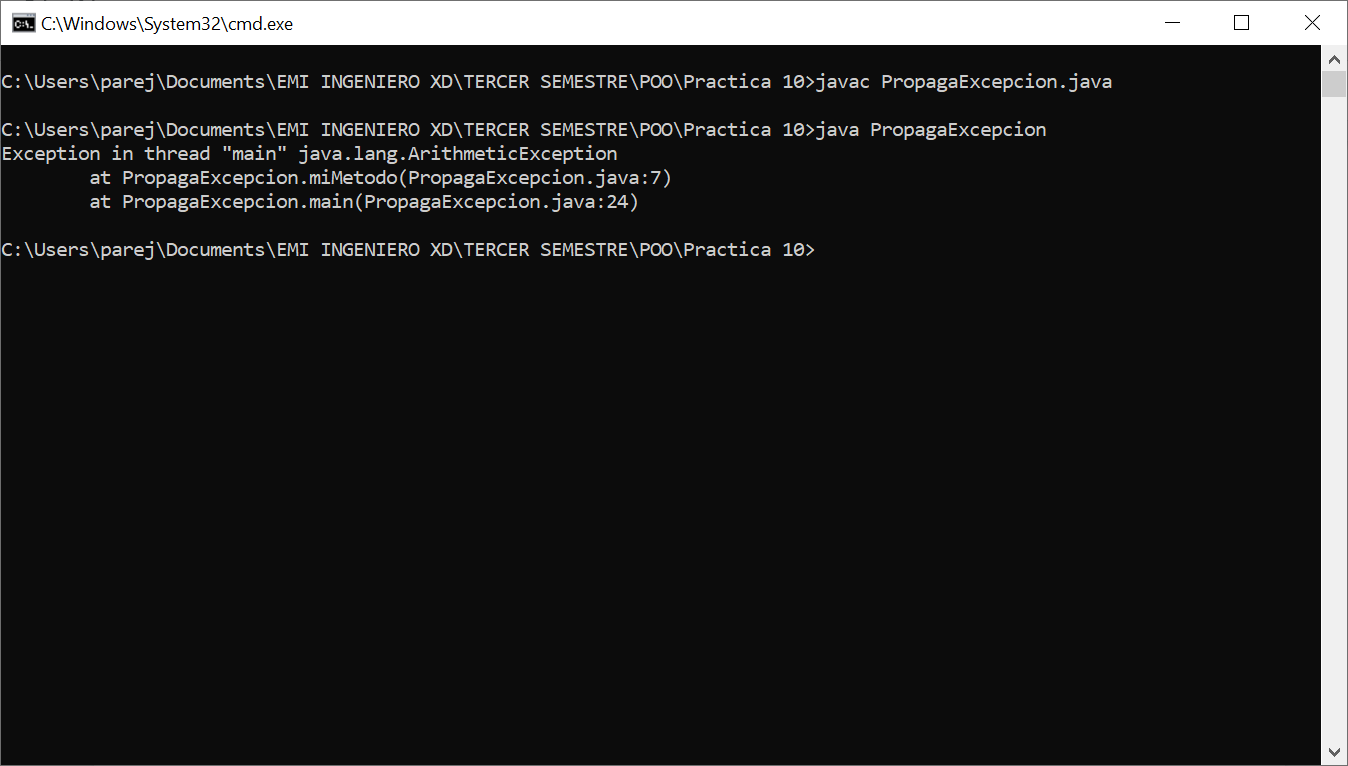
**Ejemplo PropagaExcepcion**

* **Análisis previo**

En la clase PropagaExcepcion se declara un método llamado miMetodo con modificador de acceso estático y de caracter público, el cual es de tipo entero, y recibe dos parámetros enteros, a y b. Y este arrojará una excepción de la clase ArithmeticException, dentro de este método en una estructura de control de flujo if, se verifica si el valor de b es igual a 0, en caso afirmativo se lanza una excepción de la clase ArithmeticException que creamos con la palabra reservada new, y después se hace una división que se almacena en otro entero llamado c, del valor de a entre b, y se regresa como valor de retorno el valor de c. Después en el main, una variable entera llamada resultado se iguala al valor de retorno del método miMetodo al cual se le pasan como parámetros un 10 y un 0, respectivamente, en este caso al ser el segundo parámetro un 0, entonces este método lanzará una excepción de la clase ArithmeticException, pero como dentro del main no hay ningún bloque catch que atrape dicha excepción, habrá un error al momento de compilar, ya que todas las excepciones que se generen en un programa deben ser capturadas.



* **Ejecución del programa**

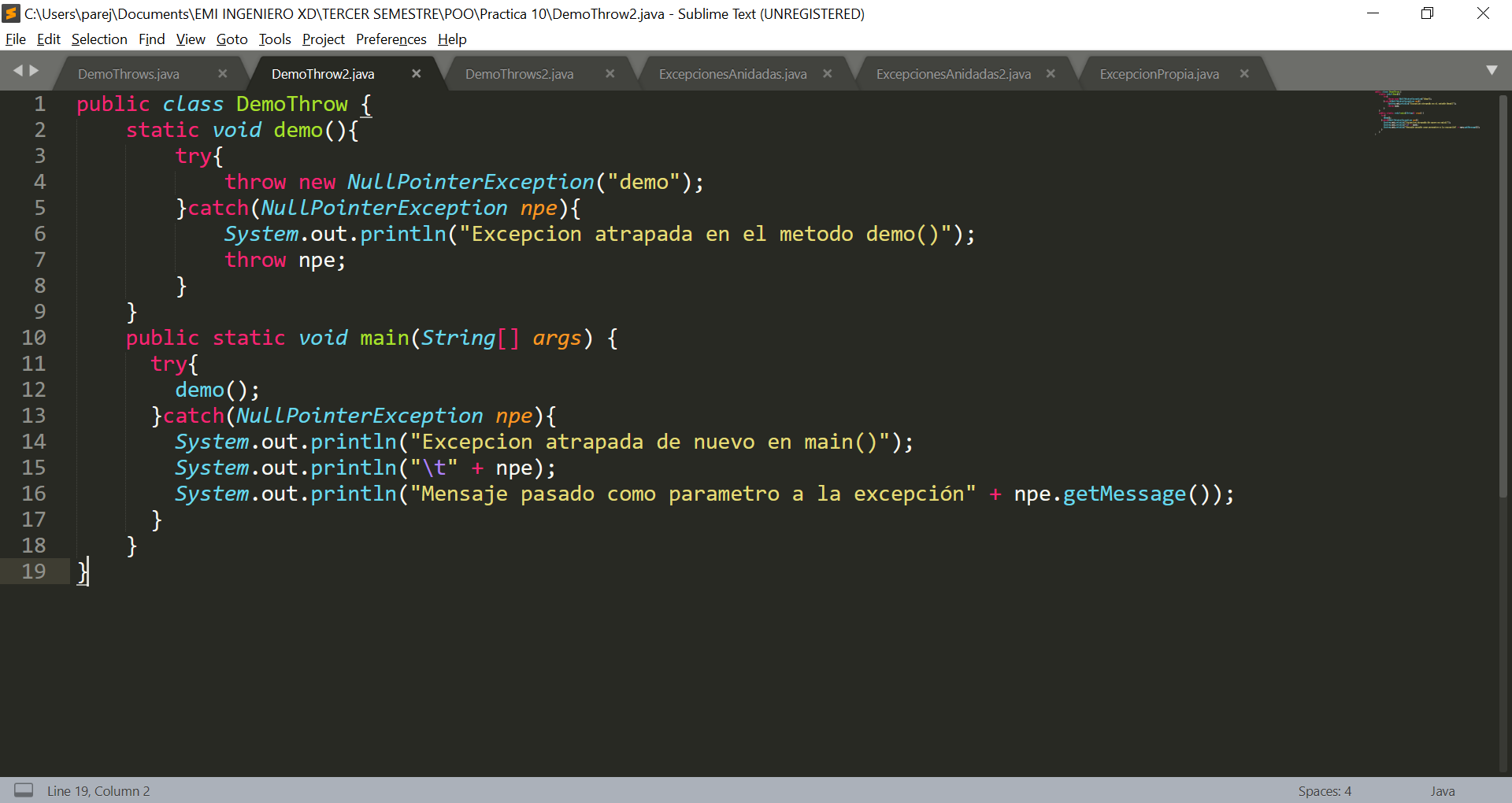


**Ejemplo DemoThrow**

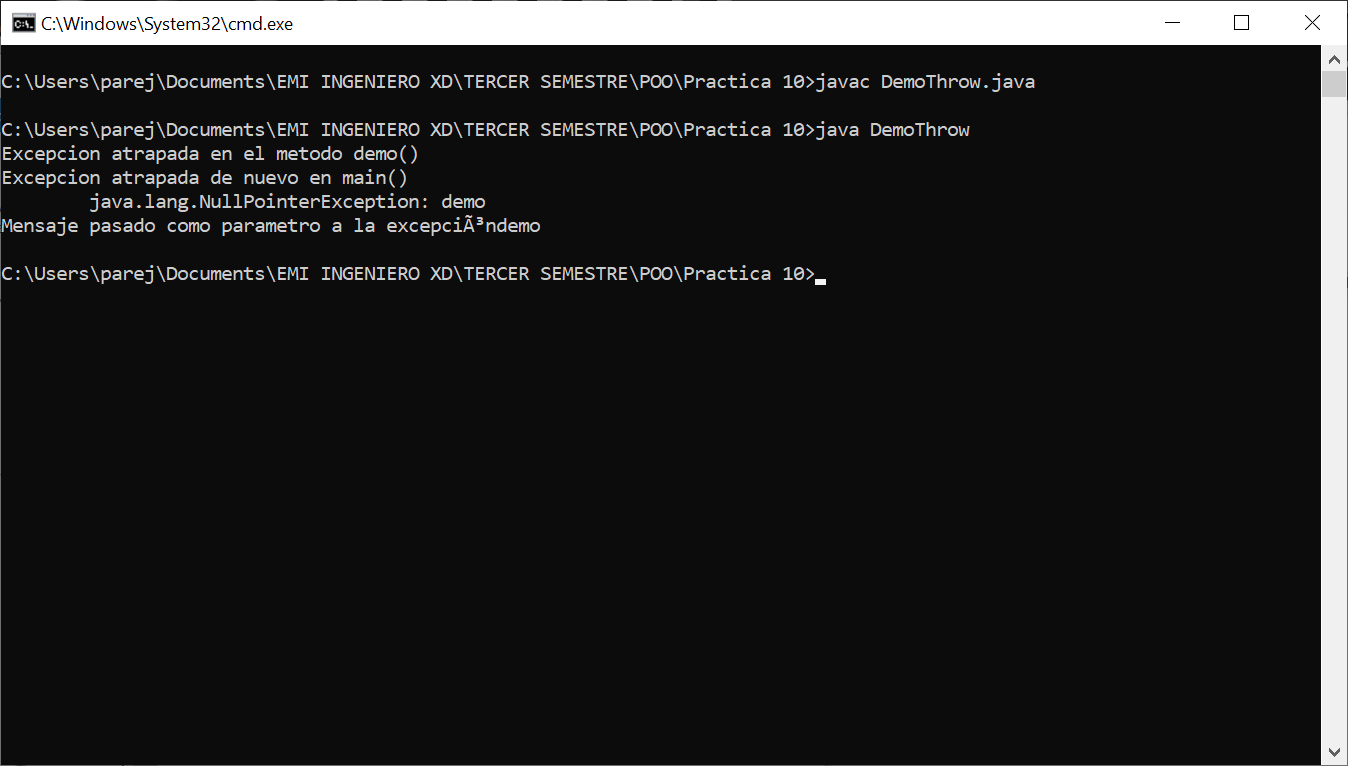
* **Análisis previo**

En la clase DemoThrow, se crea define un método de tipo void y estático llamado demo el cual no recibe parámetros, dentro de este en el blqoue try, se arroja una excepción de la clase NullPointerException que creamos con la palabra reservada new y se le pasa como parámetro el mensaje demo, después esta excepción la atrapamos en el bloque catch, que captura excepciones de la clase NullPointerException, y se imprime a pantalla que esta excepción se atrapó en el método demo de la clase, sin embargo esta excepción se vuelve a lanzar, ya que utilizamos la palabra reservada throw, para indicar que la excepción capturada en el bloque catch se volvió a lanzar.

En el main, en el bloque try, solo se hace una invocación del método demo, y no se le pasan parámetros, y como este método obligatoriamente lanzará una excepción de la clase NullPointerException, esta se capturará en el bloque catch, y dentro de este bloque se indica que la excepción ahora fue capturada en el main, se imprime el contenido de la excepción capturada, y además el mensaje el cual se le pasó como parámetro, entonces intuyendo la salida del programa, primero se indicará que la excepción se capturó en el método demo y después se indicará que la excepción se capturó en el main



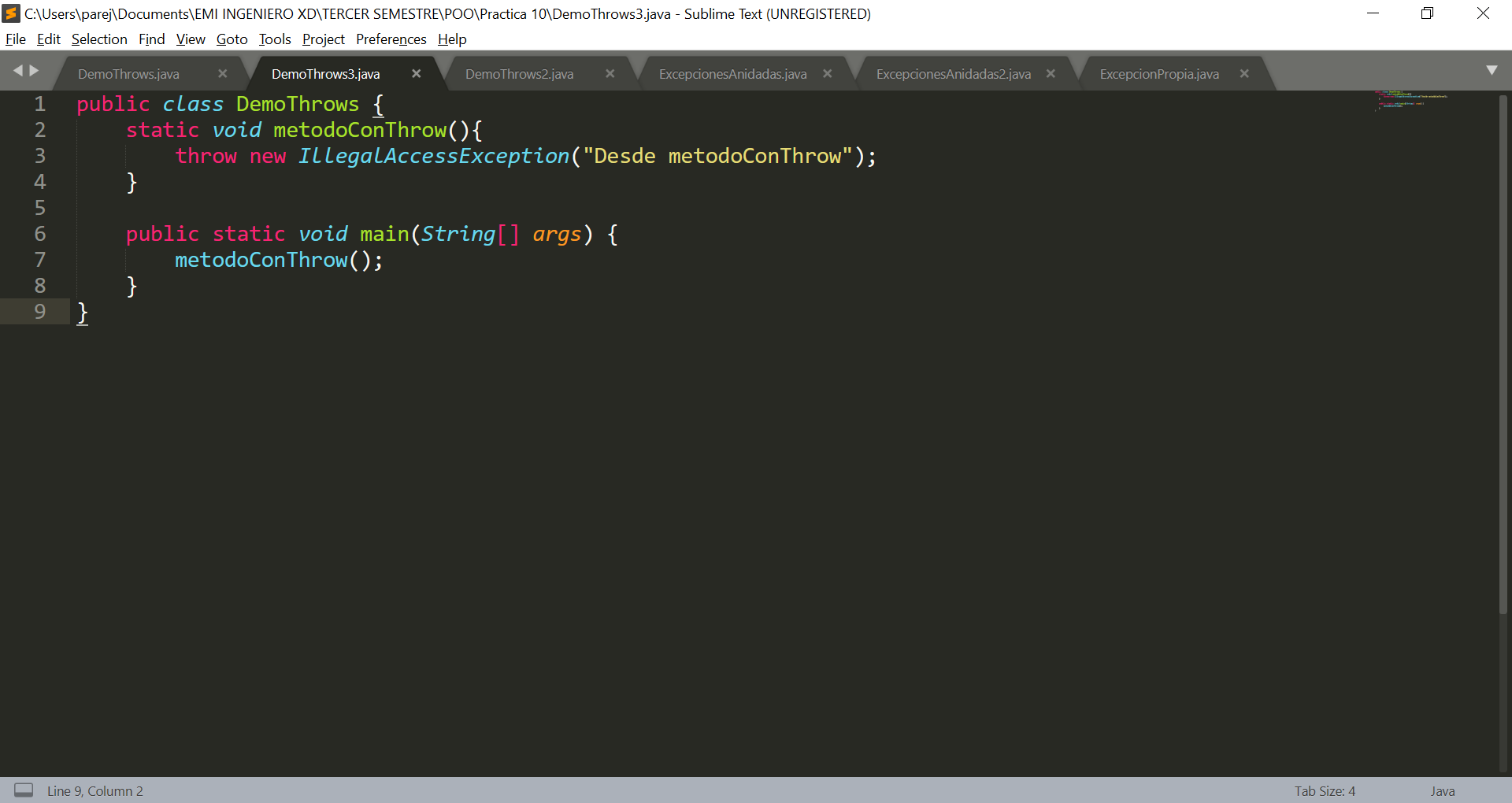
* **Ejecución del programa**



**Ejemplo DemoThrows**

* **Análisis previo**

En este código, en la clase DemoThrows se crea un método de tipo void estático, llamado metodoConThrow, en el cual solo se lanza una instancia de una excepción de la clase IllegalAccessException, a la cual se le pasa un mensaje como parámetro que dice "Desde metodoConThrow", y esta instancia la generamos con el uso de la palabra reservada new. Después en el main, solo se invoca al método metodoConThrow, el cual lanzará obligatoriamente una excepción de la clase IllegalAccessException, pero como dentro del main no hay bloques try catch, entonces este generará un error al momento de compilar, ya que todas las excepciones que se generen obligatoriamente también deben ser capturadas en un bloque catch.

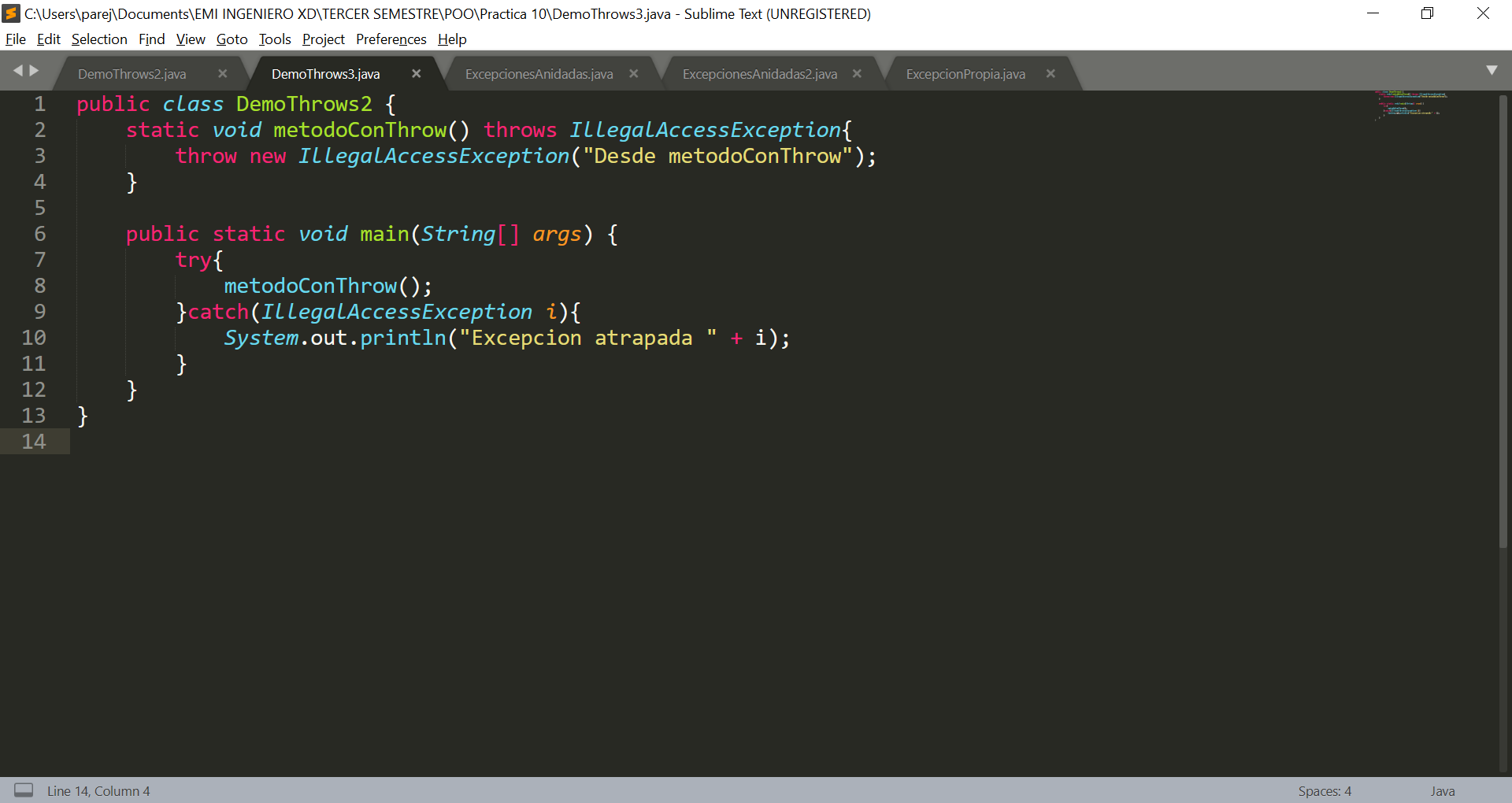


* **Ejecución del programa**

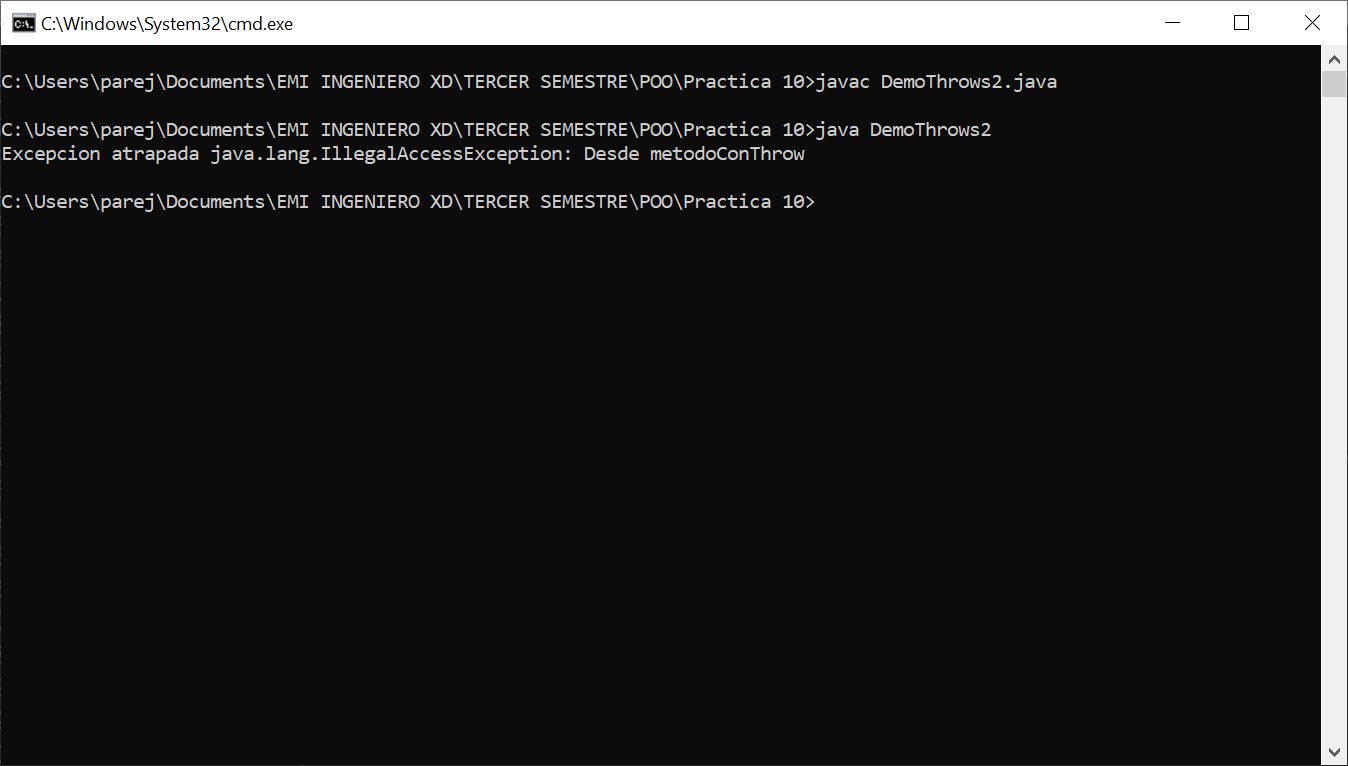
**Ejemplo DemoThrows2**

* **Análisis previo**

Este código es el mismo que el anterior en el método metodoConThrow, solo que ahora en su definición se indica que este método arrojará una excepción de la clase IllegalAccessException. La diferencia del ejemplo anterior es que ahora en el main, tenemos un bloque try-catch, que permitirá atrapar la excepción que lanza el método metodoConThrow, entonces en el método try se invoca a este método, y en el bloque catch se atrapa la excepción, y se imprime a pantalla el contenido de la excepción atrapada. Entonces ahora no tendremos errores en tiempo de compilación, y tendremos una salida donde solo se imprimirá el contenido de la excepción atrapada en el main



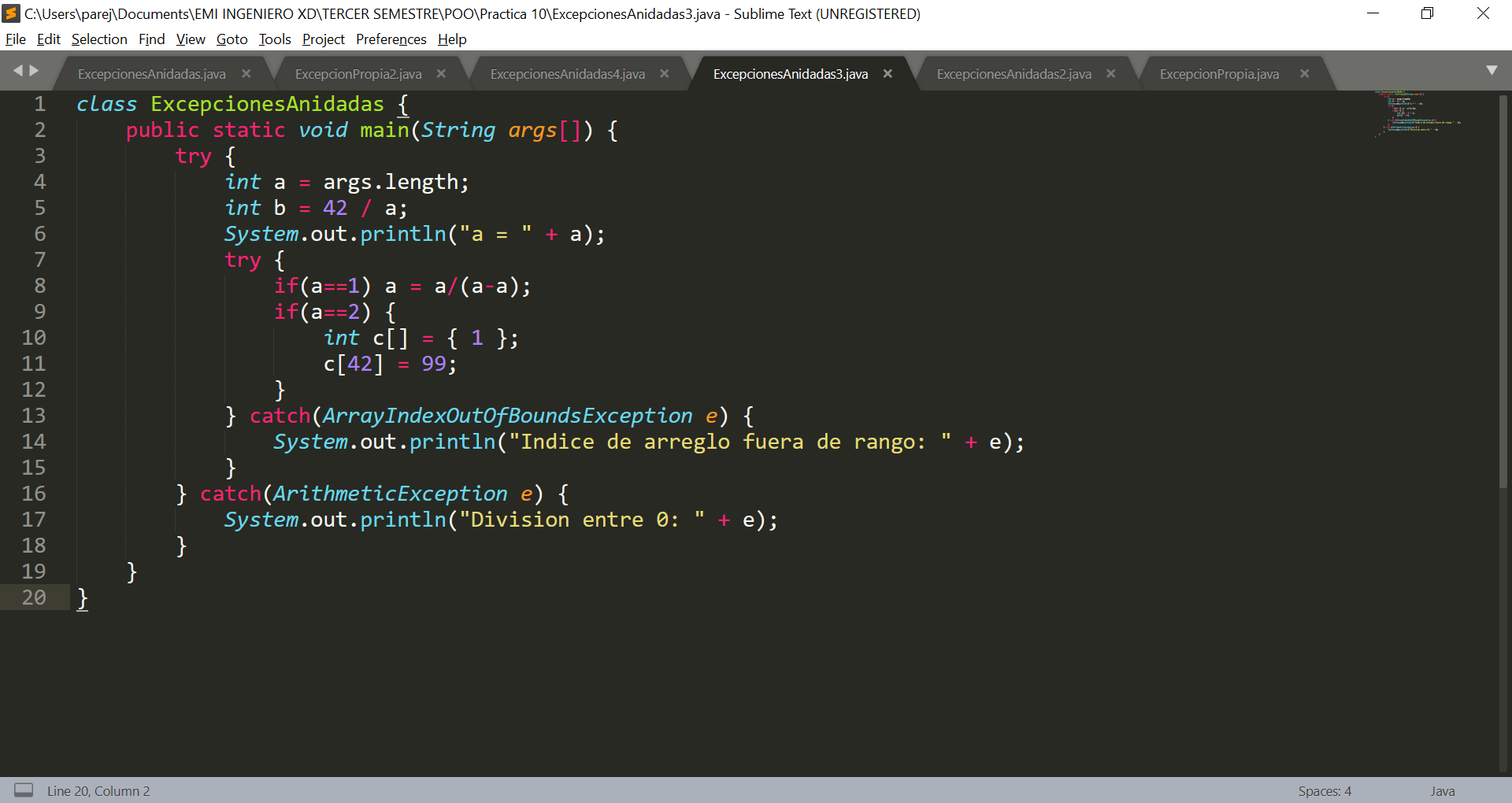
* **Ejecución del programa**



**Ejemplo ExcepcionesAnidadas**

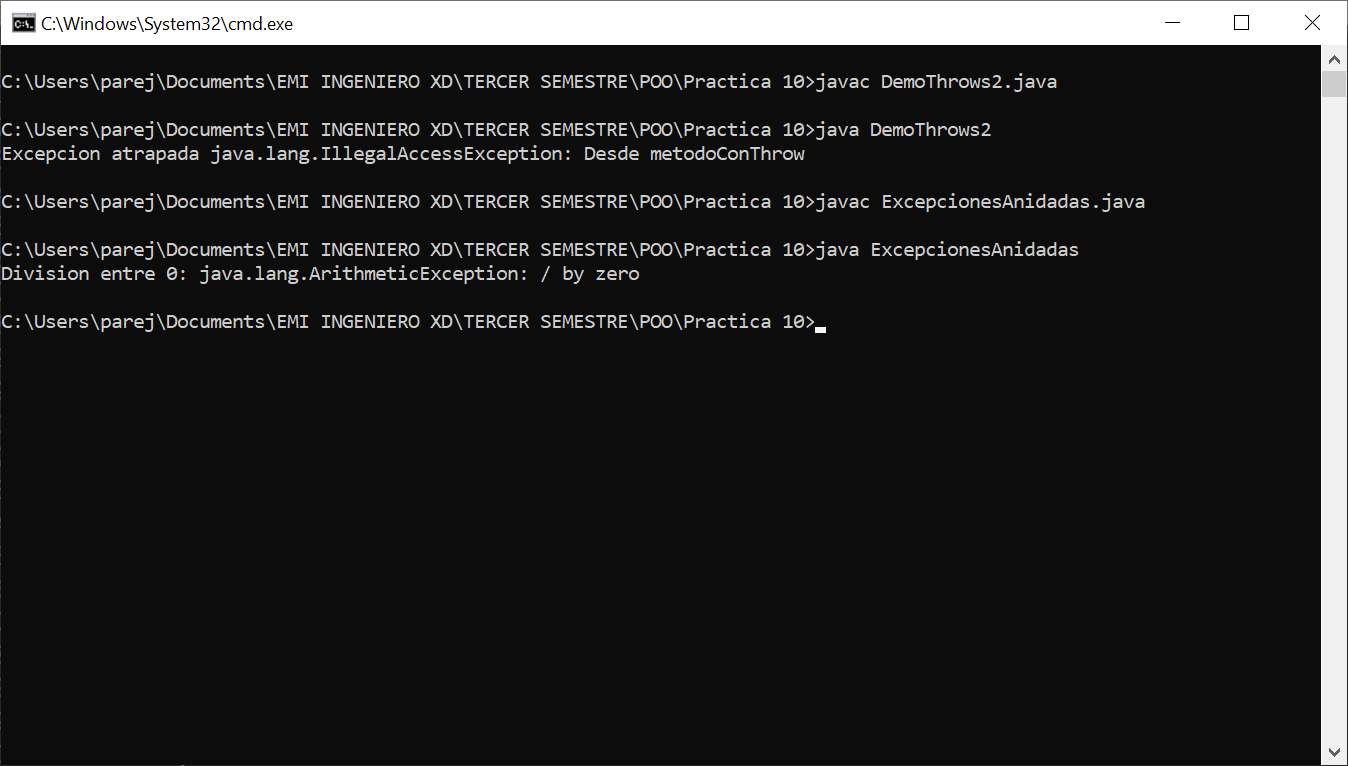
* **Análisis previo**

En este código se puede ver el uso de un doble bloque try catch el cual nos permite encerrar un bloque de código que puede generar dos tipos de excepciones en este caso se van a generar una excepción de tipo aritmética gracias a la división entre cero y por otro lado se generará una excepción de tipo índice fuera de los límites.



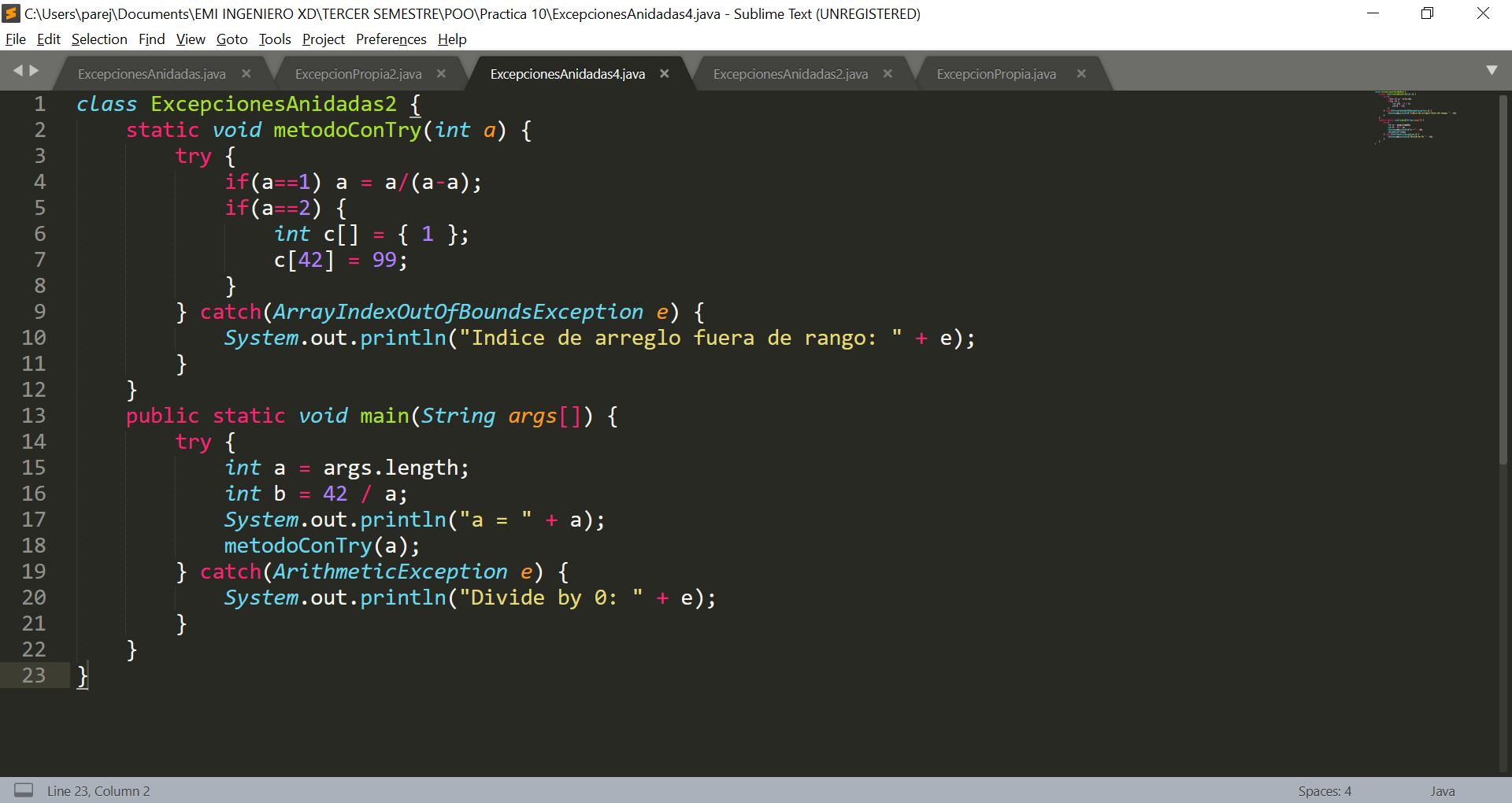
* **Ejecución del programa**

Al ejecutar el programa podemos observar como este captura la excepción de tipo aritmética a pesar de estar en lo más superior de lo anidado.

**Ejemplo ExcepcionesAnidadas2**

* **Análisis previo**

En este código se hace una prueba con un método el cual solo tiene un bloque try-catch el cual solo maneja una de las dos excepciones generadas, pero en el método Main hay un bloque try-catch el cual abarca también la invocación al método con dos excepciones, mediante este código se hace la prueba para ver si el anidado funciona aun cuando los dos bloques se ubican en distintos métodos



* **Ejecución del programa**

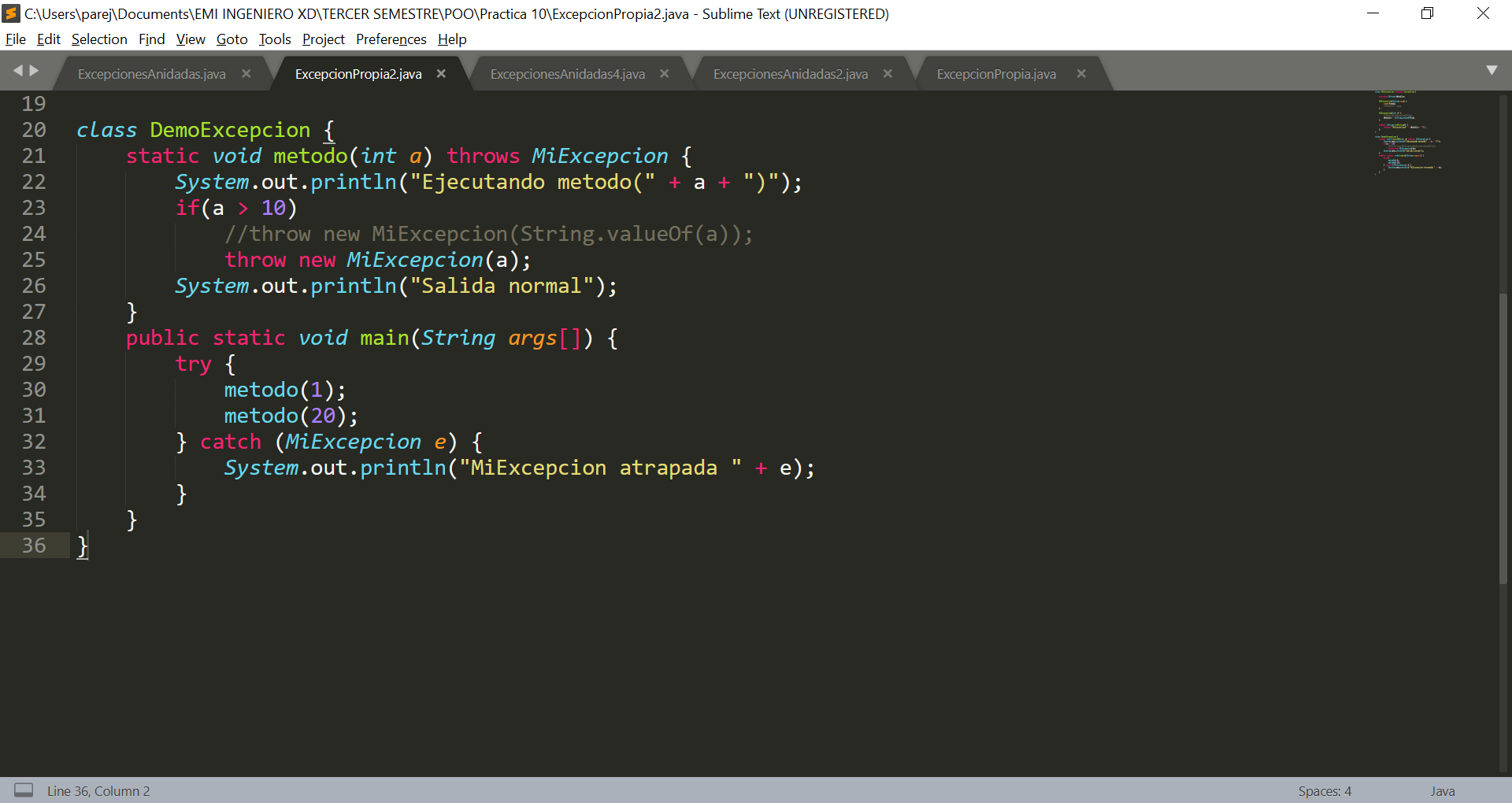
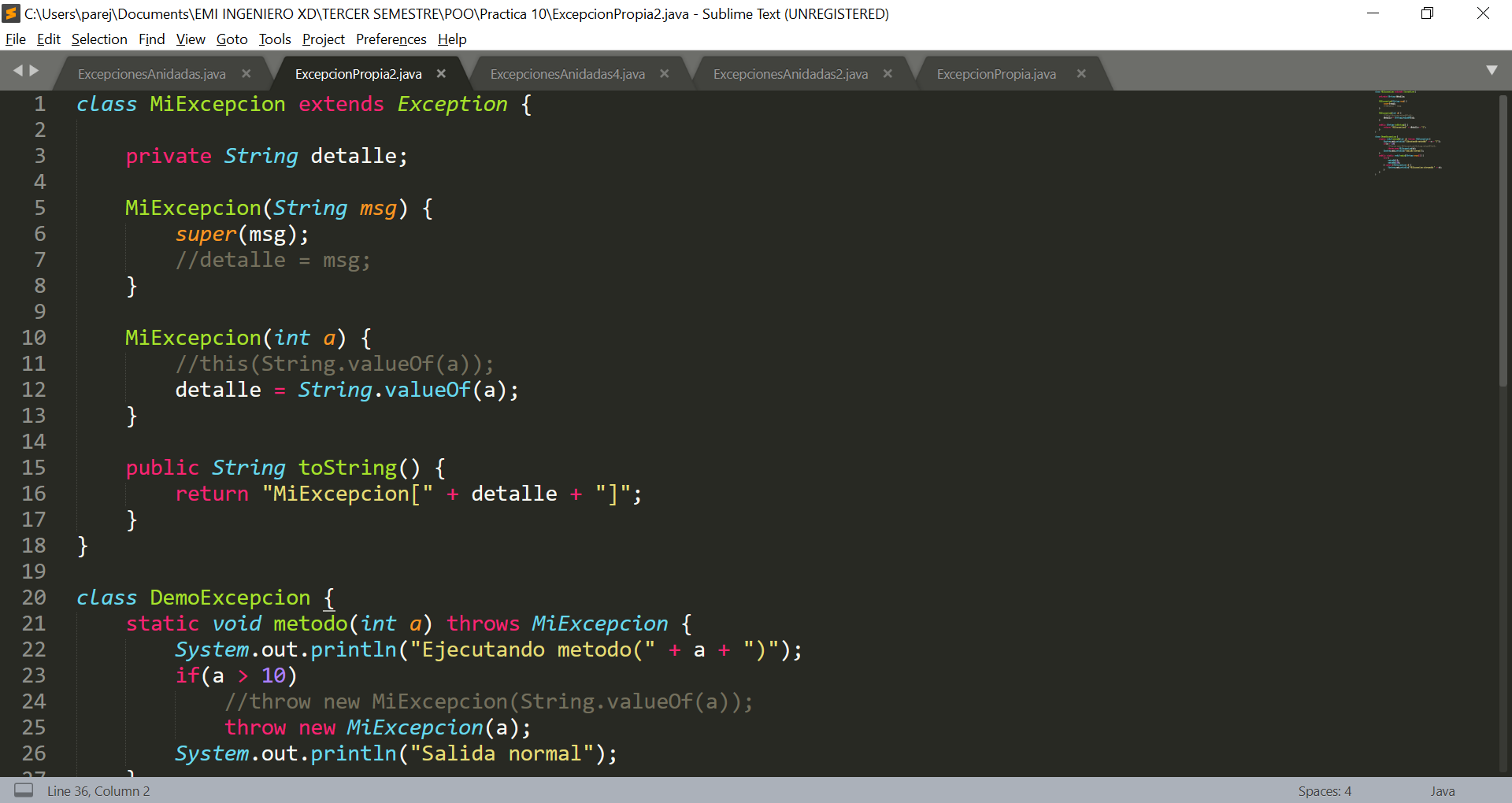
Para probar el código se generan las excepciones aritmética y índice fuera del límite, lo cual al momento de correr el programa no lanza el mensaje de que generó la excepción lo cual preconfiguramos anteriormente en el código, más en este caso no se alcanza a llegar a la excepción de tipo índice fuera de límites debido a que se lanzó antes la excepción de tipo aritmética.



**Ejemplo ExcepcionPropia**

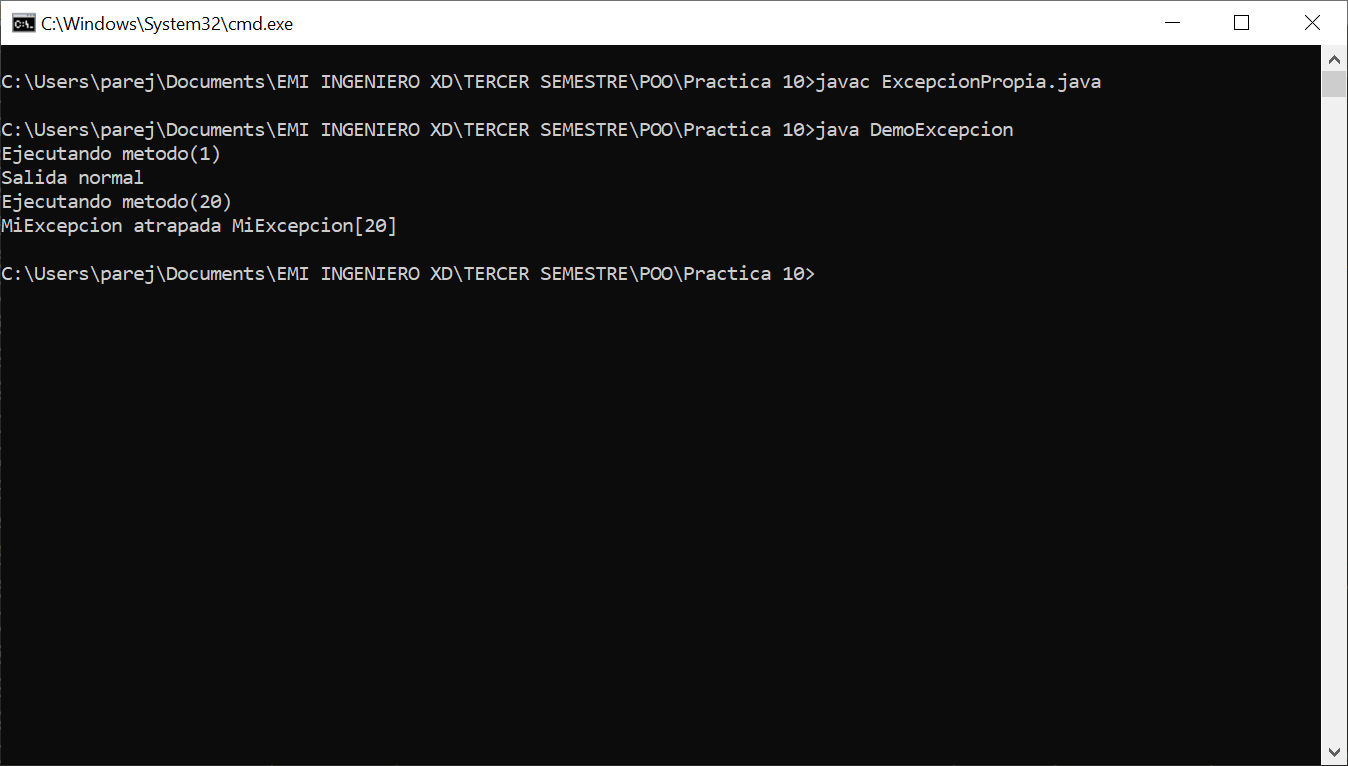
* **Análisis previo**

En este código se crea una clase la cual se extiende de la clase excepción para poder crear una excepción personalizada a el caso, para eso se crean dos métodos constructores los cuales varían dependiendo del argumento que se les pasa, para probar estas excepciones se creó la clase DemoExcepcion en la cual hay un método el cual puede lanzar una excepción de tipo Miexception, para generar esta excepción hacemos un error en un método, ya que en la clase principal se manda a llamar al método el cual genera una excepción pasando como argumento un número, el cual se le busca sacar un número entero un valor como si fuera una cadena lo cual provoca una excepción siempre y cuando el número que se pase como argumento sea menor a 10 se generará dicha excepción.



* **Ejecución del programa**

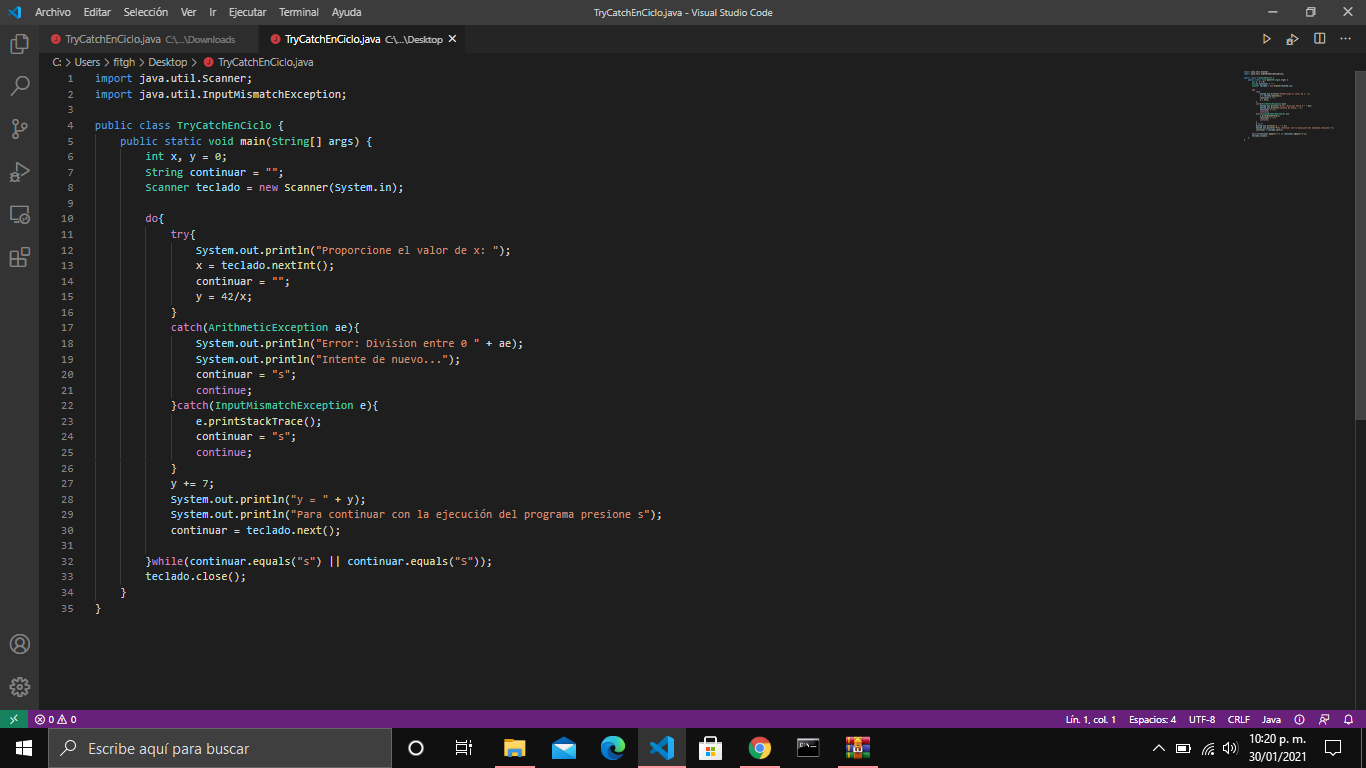
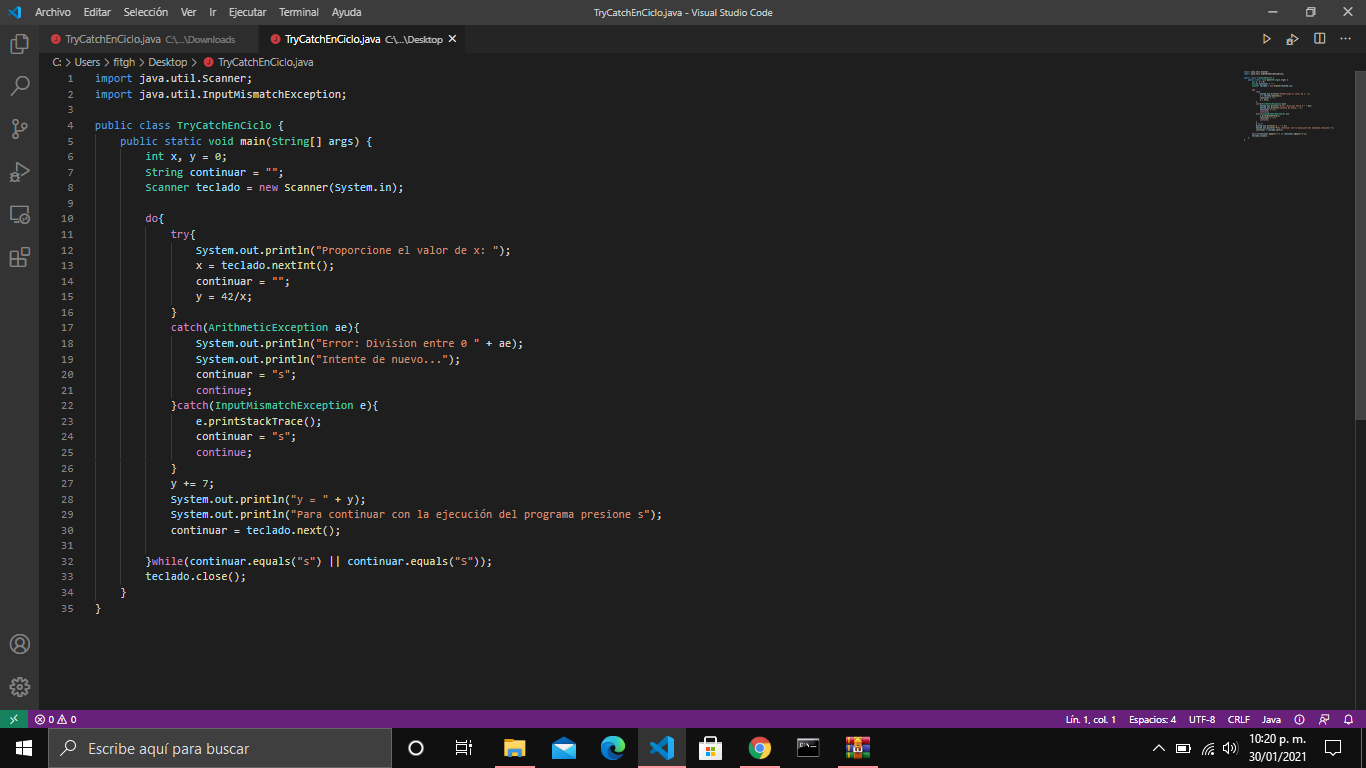
Para probar el funcionamiento de atrapar excepciones se mandó a llamar dos veces al método puede generar excepciones en la primera llamada se pasa como argumento un número uno lo cual no provoca una excepción mientras que en el segundo se manda a llamar con un argumento de valor de 20 lo cual genera una excepción pero esta es atrapada y nos manda un mensaje de que se capturó correctamente la excepción



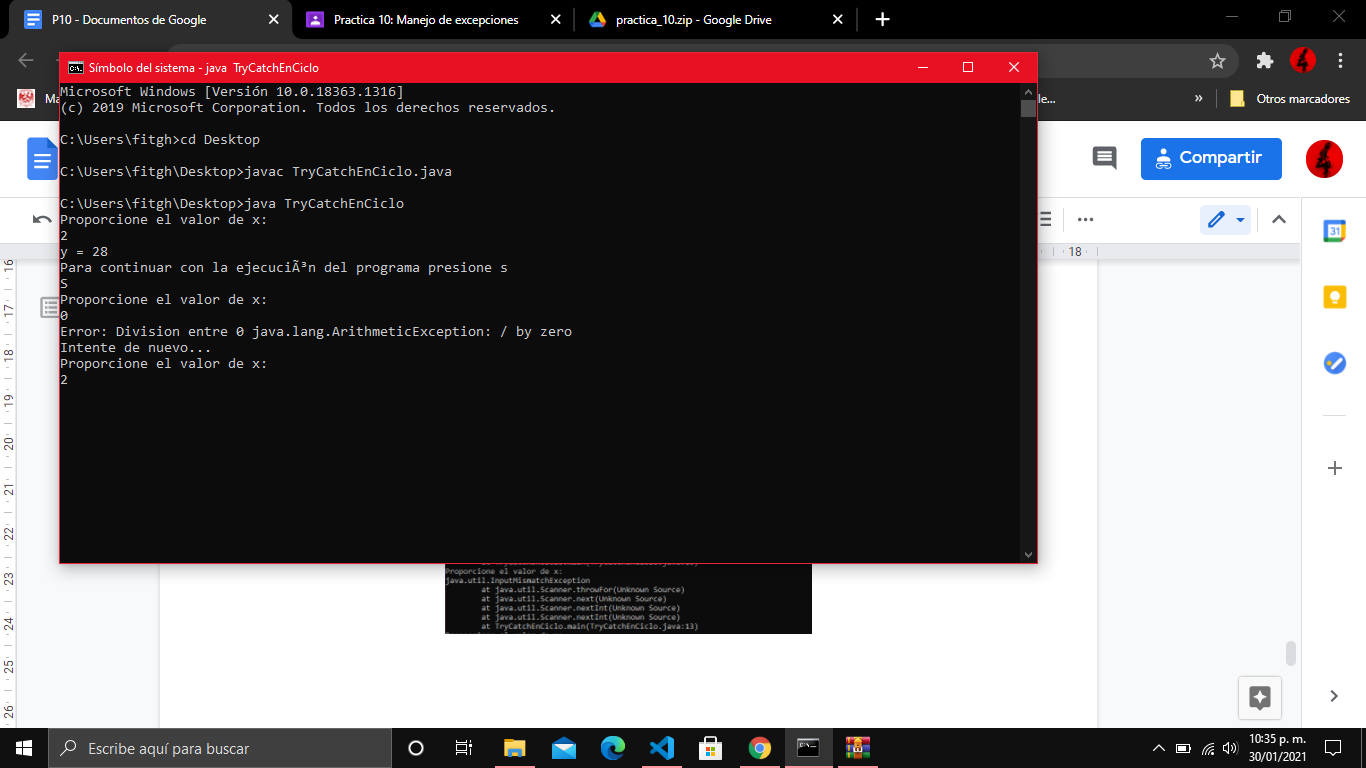
**Ejemplo TryCatchEnCiclo**

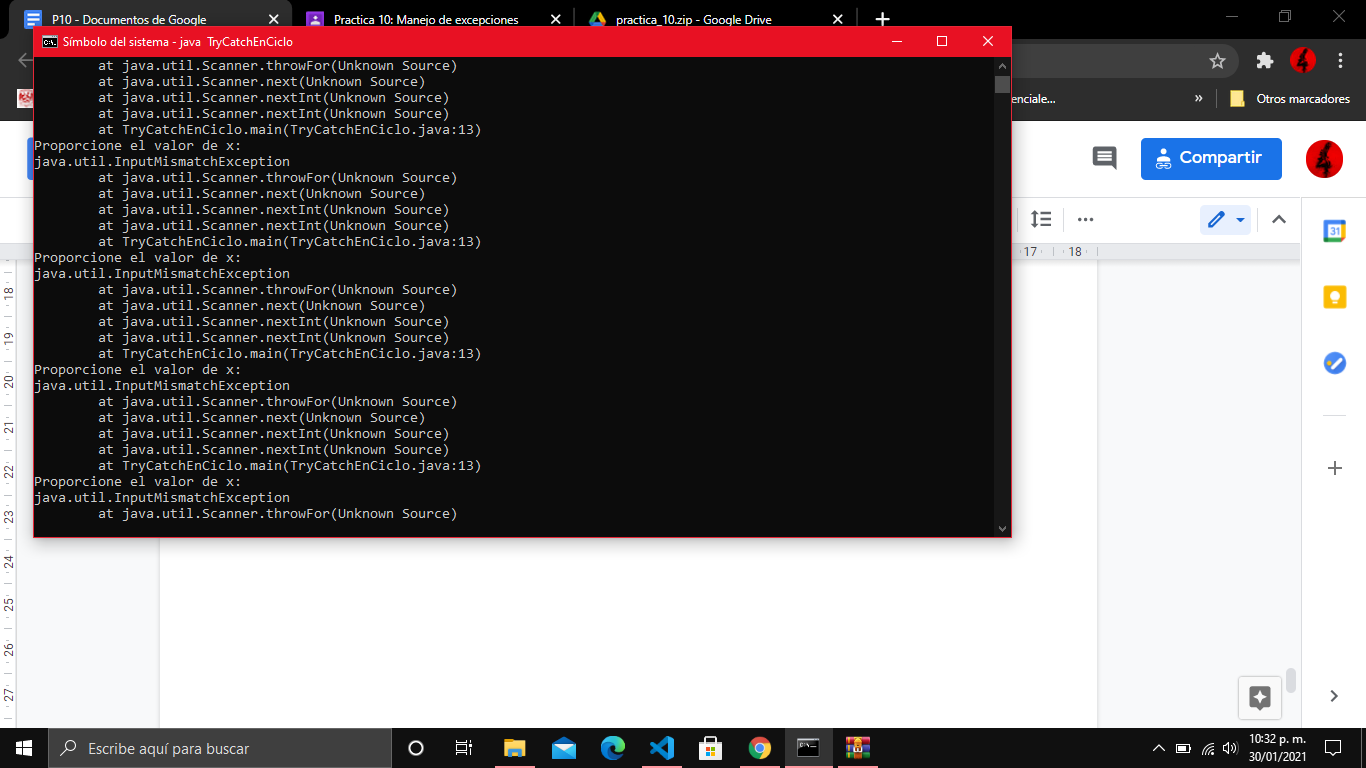
* **Análisis previo**

En este código hay definido un bloque try-catch en una estructura de repetición (bucle do while), dentro del bloque try se pide por consola un valor de tipo entero “x” que se utilizará como divisor del número 42, cuyo resultado sumado a 7 será el valor asignado a la variable de tipo entera “y”, esta última se imprime en consola y se le pregunta al usuario si desea volver a realizar la operación anterior (“ingresar s o S”), en este caso pueden presentarse dos excepciones, una generada por ingresar el valor de “x” igual a 0 y tratar de realizar la división entre 0 resultando en una excepción del tipo “ArithmeticException” y otra del tipo “InputMismatchException”, esta se dará si el valor que introduzca para el valor de la variable x no es un número, ya sea que el usuario ingrese un caracter o una cadena para la variable x, ambas excepciones son atrapadas, en caso de presentarse en un bloque catch determinado, imprimiendo en pantalla el mensaje del error generado, en caso de ingresar un 0: "Error: Division entre 0 " y en caso de ingresar un dato distinto a un número usando el método “printStackTrace()” se imprime la pila de errores que se generaron con la excepción, en ambos casos se ejecuta la sentencia continue para que se vuelva a repetir el ciclo do-while, en cada iteración del bucle se solicita ingresar el valor de “x” y en caso de presentarse las excepciones antes mencionadas son atrapadas, hasta que el usuario decida no realizar la operación de nuevo.

* **Ejecución del programa**

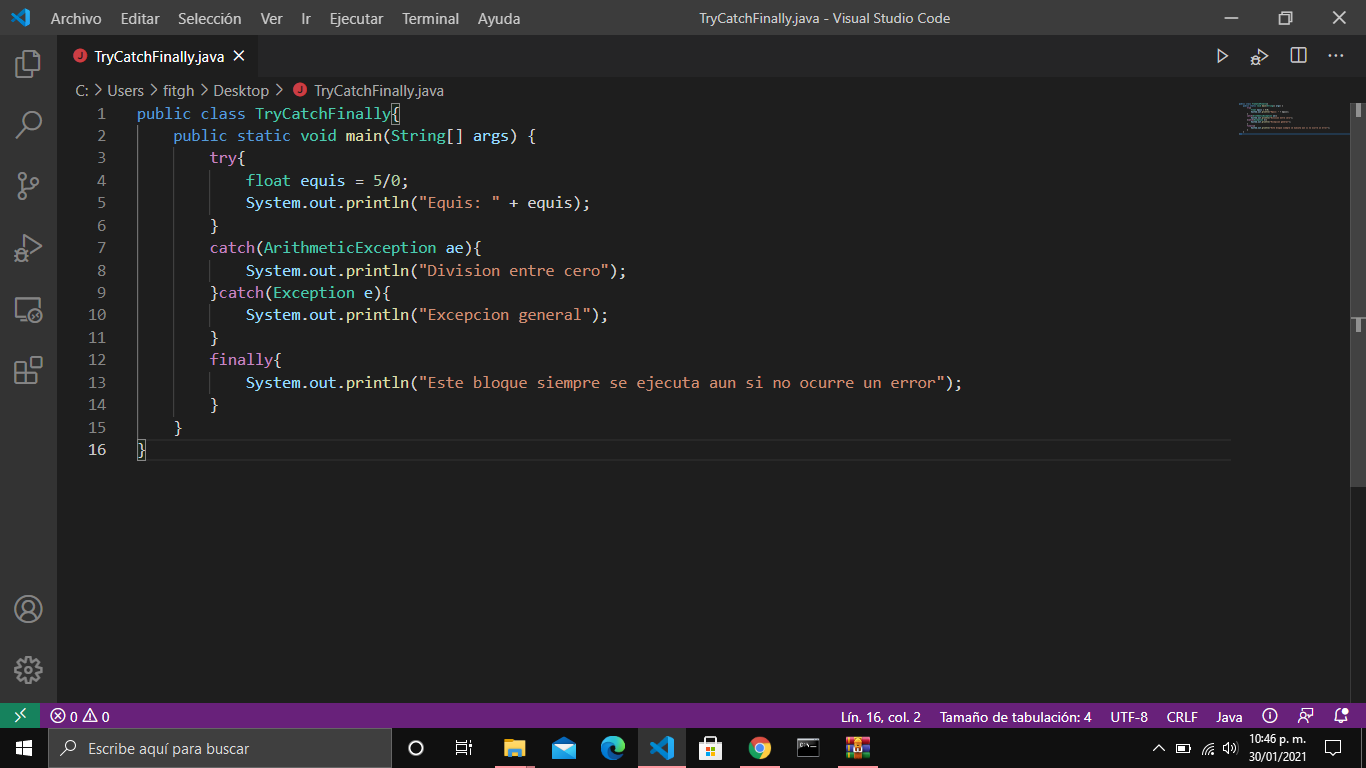
****

****

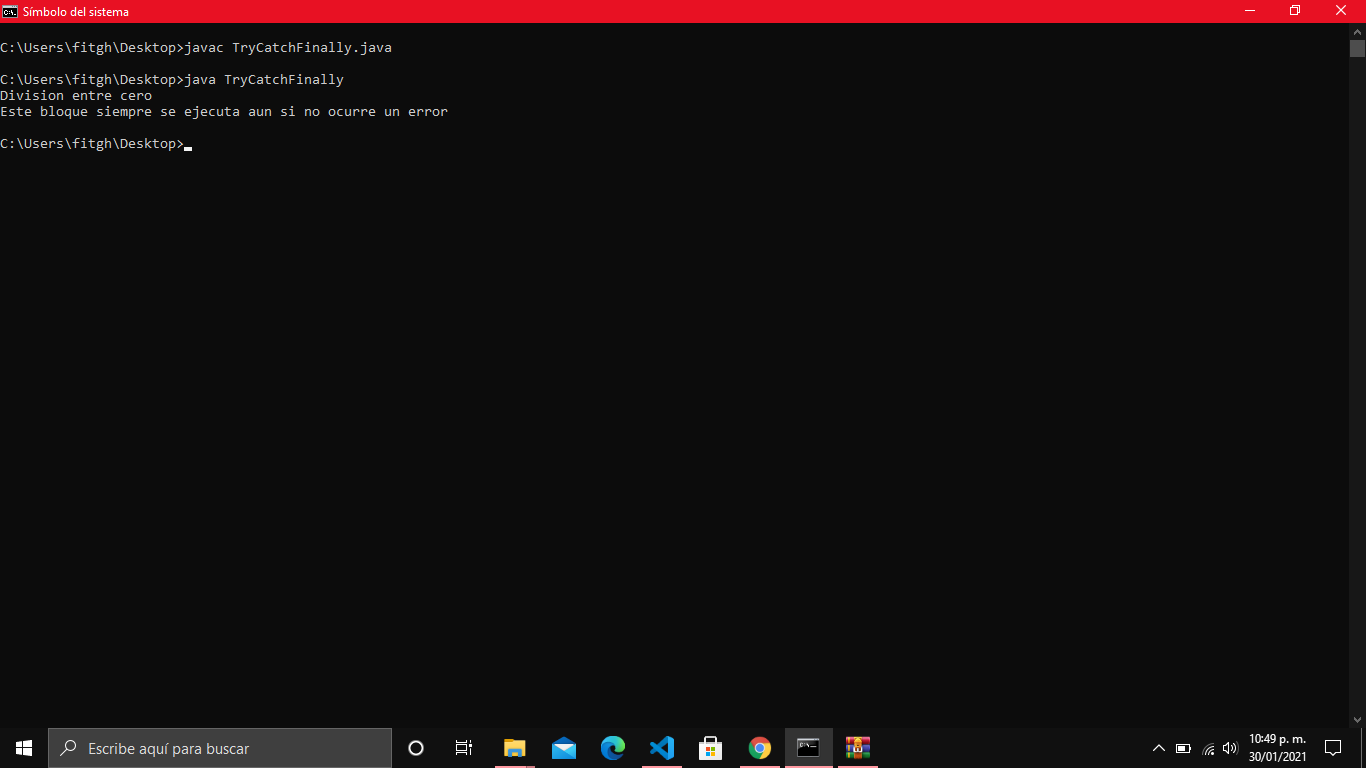
**Ejemplo TryCatchFinally**

* **Análisis previo**

En este programa inicialmente se divide un número entre cero en un bloque try, lo cual generará una indeterminación, hay definidos dos bloques catch el primero para una excepción del tipo “ArithmeticException” y otra para “Exception, este caso al compilar y ejecutar el programa, el error generado por la división entre cero será captado por el bloque con la excepción más específica en este caso “ArithmeticException” y se ejecutarán las instrucciones de este bloque catch (se imprime “División entre cero”), por último se ejecutarán las instrucciones del bloque finally, ya que este bloque siempre se ejecuta aún sin presentarse errores, en este caso se imprime en pantalla "Este bloque siempre se ejecuta aun si no ocurre un error".



* **Ejecución del programa**

****

**Conclusiones**

* Dávila Ortega Jesús Eduardo:

En el desarrollo de la práctica aprendí cómo entender los tipos de excepciones que pueden llegar a aparecer en el desarrollo de un programa y cómo poder llegar a tratar con ellas de una manera eficiente, logrando el correcto funcionamiento de un programa gracias a un buen manejo de excepciones.

* Díaz Hernández Marcos Bryan:

Al desarrollar la documentación de cada uno de los ejemplos, pude comprender las condiciones necesarias para establecer qué tipo de excepción se puede presentar en una determinada sección del código y cómo controlarlas, además de analizar la creación de una excepción propia. Por lo anterior considero que el objetivo de la práctica se pudo cumplir.

* Pareja Ávila Emiliano:

Pude identificar las diferentes tipos de excepciones que existen en Java, y se que para poder utilizarlas en un código, debo de establecer jerarquía entre ellas, ir de las más específicas a las más generales. Puedo identificar los errores que se pueden generar en un programa, y capturar los errores que se puedan generar con sus excepciones que corresponden al tipo de error. Puedo generar varios bloques catch, para poder atrapar todos los tipos de excepciones que se puedan generar en un bloque try, y creo que una de las cosas más importantes que se manejaron en la práctica y que ahora se hacer, es el generar mis propias excepciones que puedo definir a partir de condiciones generadas por mí, entonces se cumplieron los objetivos de la práctica.

* Vázquez Zavala Oliver Alexis:

Esta práctica me ha dejado de aprendizaje los tipos de errores que pueden presentarse al desarrollar un programa, los cuales son importantes conocer para en caso de presentarse en algún programa poder solucionarlos, también aprendí que son y los tipos de excepciones que existen en Java, así como el manejo que estas tienen mediante las estructuras try-catch-finally, además de que con esta práctica pude conocer que bloques de código están más expuestos a producir algún error y el tratamiento que deben de tener para evitar problemas posteriores.

**Bibliografía:**

* Andes, U. (2021). Manejo de las Excepciones · Fundamentos de Programación. Recuperado (28 enero 2021) de: <https://universidad-de-los-andes.gitbooks.io/fundamentos-de-programacion/content/Nivel4/5_ManejoDeLasExcepciones.html>
* Tratamiento de Errores. (2021). Recuperado (28 enero 2021), de <https://mastermoviles.gitbook.io/introduccion-a-java-y-eclipse/tratamiento-de-errores>