Práctica 8

Ejercicio 1: ¿Explique claramente a qué se denomina excepción?

Es una condición inesperada o inusual que surge durante la ejecución del programa y no puede ser manejada en el contexto local. Condición anómala e inesperada que necesita ser controlada.

Ejercicio 2: ¿Qué debería proveer un lenguaje para el manejo de las excepciones? ¿Todos los lenguajes lo proveen?

Para que un lenguaje trate excepciones debe proveer mínimamente:

- Un modo de definirlas
- Una forma de reconocerlas
- Una forma de lanzarlas y capturarlas
- Una forma de manejarlas especificando el código y respuestas
- Un criterio de continuación

No, no todos los lenguajes lo proveen, por ejemplo C estándar no provee manejo de excepciones

Ejercicio 3: ¿Qué ocurre cuando un lenguaje no provee manejo de excepciones? ¿Se podría simular? Explique cómo lo haría

Cuando un lenguaje de programación no proporciona un mecanismo de manejo de excepciones, puede ser más difícil y menos estructurado manejar errores y situaciones excepcionales en el código. Sin un manejo adecuado de excepciones, los errores pueden propagarse sin control, lo que puede llevar a un comportamiento inesperado y a una mayor dificultad para depurar el código.

Aunque no se disponga de un mecanismo de manejo de excepciones incorporado en el lenguaje, es posible simularlo utilizando técnicas alternativas. Una forma común de simular el manejo de excepciones es mediante el uso de estructuras de control condicionales y funciones de retorno de errores.

Es importante tener en cuenta que este enfoque simulado del manejo de excepciones puede resultar más propenso a errores y puede llevar a un código más complicado y difícil de mantener en comparación con un lenguaje que ofrece un manejo de excepciones nativo. Además, la simulación no proporcionará todas las funcionalidades avanzadas de los mecanismos de manejo de excepciones, como la capacidad de capturar excepciones específicas o realizar acciones de limpieza antes de propagar el error.

Ejercicio 4: Cuando se termina de manejar la excepción, la acción que se toma luego es importante

01. ¿Qué modelos diferentes existen en este aspecto?

- **Reasunción**: cuando se produce una excepción, se maneja y al terminar de ser manejada se devuelve el control a la sentencia **siguiente** de donde se levantó la excepción.
    - **PL/1.**
- **Terminación**: cuando se produce una excepción, el bloque donde se levantó la excepción es **terminado** y se ejecuta el manejador asociado a la excepción.
    - **ADA**
    - **CLU**
    - **C++**
    - **Java**
    - **Python**
    - **PHP**

02. Dé ejemplos de lenguajes que utilizan cada uno de los modelos presentados anteriormente. Por cada uno responda respecto de la forma en que trabaja las excepciones.

a. ¿Cómo se define?

b. ¿Cómo se lanza?

c. ¿Cómo se maneja?

d. ¿Cuál es su criterio de continuación?

Primer lenguaje que incorpora excepciones – PL/1

- Utiliza el modelo de ejecución (criterio) de **reasunción**.
- Las excepciones se llaman **CONDITIONS**.
- Los manejadores de excepciones se declaran con la sentencia ON:
    - **ON CONDITION(Nombre_excepción) Manejador**
    - El manejador puede ser una instrucción o un bloque de instrucciones.
- Las excepciones son alcanzadas explícitamente con la palabra clave:
  Signal condition(Nombre_excepcion)
- PL/1 tiene excepciones predefinidas (las cuales tienen su manejador asociado). A dichas excepciones las llamamos Built-in exceptions.
    - Ejemplo: zerodivide que se levanta cuando hay una división por cero.
- Los manejadores son ligados de forma dinámica con las excepciones. Una excepción siempre estará ligada con el último manejador definido.
- El alcance de un manejador termina cuando finaliza la ejecución de la unidad donde fue declarado.

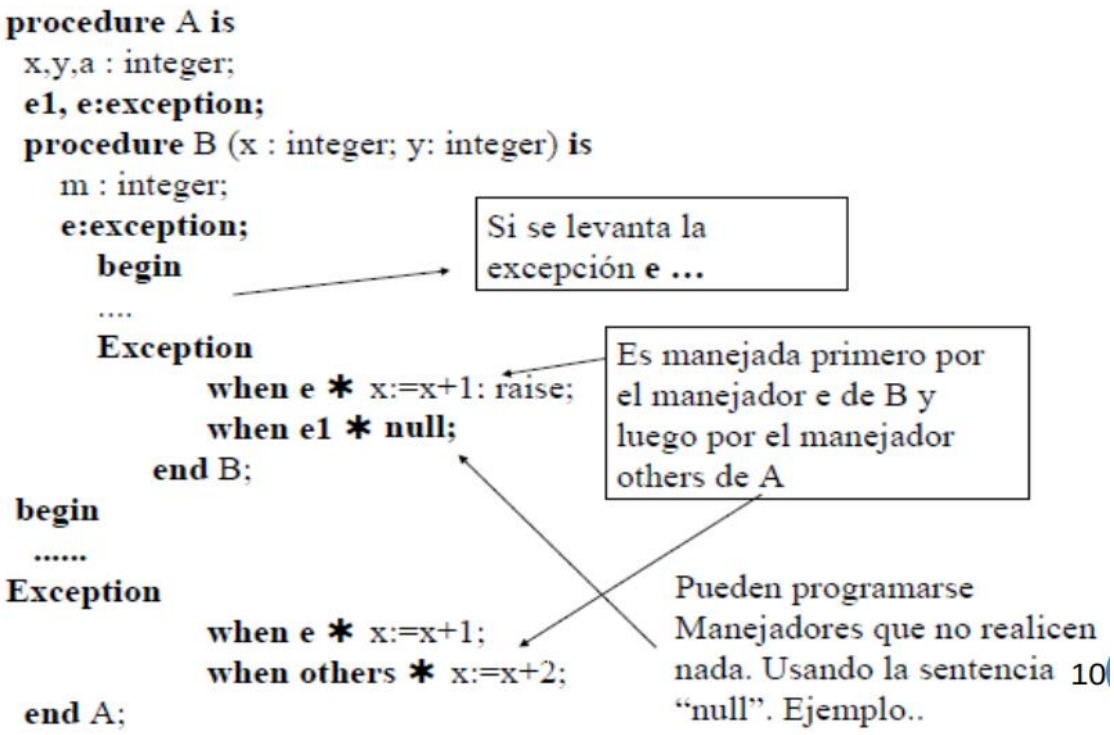
Excepciones en ADA

- Usa criterio de terminación.
- Cada vez que se produce una excepción, se termina el bloque dónde se levantó y se ejecuta el manejador asociado, y continúa luego.
- Las excepciones se definen en la zona donde definimos variables y poseen el mismo alcance → MiExcepcion: exception;
- Una excepción es alcanzada explícitamente con la palabra clave **raise**.
- Los manejadores se pueden encontrar en el final de cuatro diferentes unidades de programa: Bloque, procedimiento, paquete o tarea.

- La lista de controladores de excepciones lleva el prefijo de la palabra clave exception.
- Cada controlador lleva el prefijo de la palabra clave when
- When others es para manejar una excepción que no definimos. Puede tener efector colaterales

```
Procedure prueba;
      e1, e3: Exception
Begin
.....
Exception
when e1 → Manejador1;
when others →Manejadorn;
End;
```

- Posee excepciones predefinidas:
    - Constraint_error, Program_error, Storage_error, Numeric_error, Name_error, Tasking_error.
- Propagación cuando se produce una excepción:
    - Se termina la unidad, bloque, paquete o tarea donde se levanta la excepción.
    - Si el manejador se encuentra en ese ámbito, se ejecuta.
    - Si el manejador NO se encuentra en dicho ámbito, la excepción es propagada dinámicamente. Se levanta en otro ámbito.
        - Esto provoca que se termine la unidad del otro ámbito.
    - Debemos tomar en cuenta el alcance, la excepción puede volverse anónima.
        - Si el manejador de la excepción no está en el ambito donde se declara la excepción:
            - Se propaga al ambito que llamó mi unidad,bloque,paquete o tarea.
            - Si dicho ámbito tiene para manejar, se maneja la excepción de forma **anónima**, es decir, usando la opción when others.
        - Ejemplo: en procedure prueba tenemos e3, supongamos que e3 no tiene manejador y se va a buscar al ámbito que llamó a mi procedure y dicho ámbito tiene manejadores.
            - En dicho ámbito, mi e3 va a entrar al caso donde sea when others, por eso decimos que e3 se vuelve anónima.
- Una excepción se puede levantar NUEVAMENTE usando sólo la palabra **raise.** (se propaga y queda como anónima)

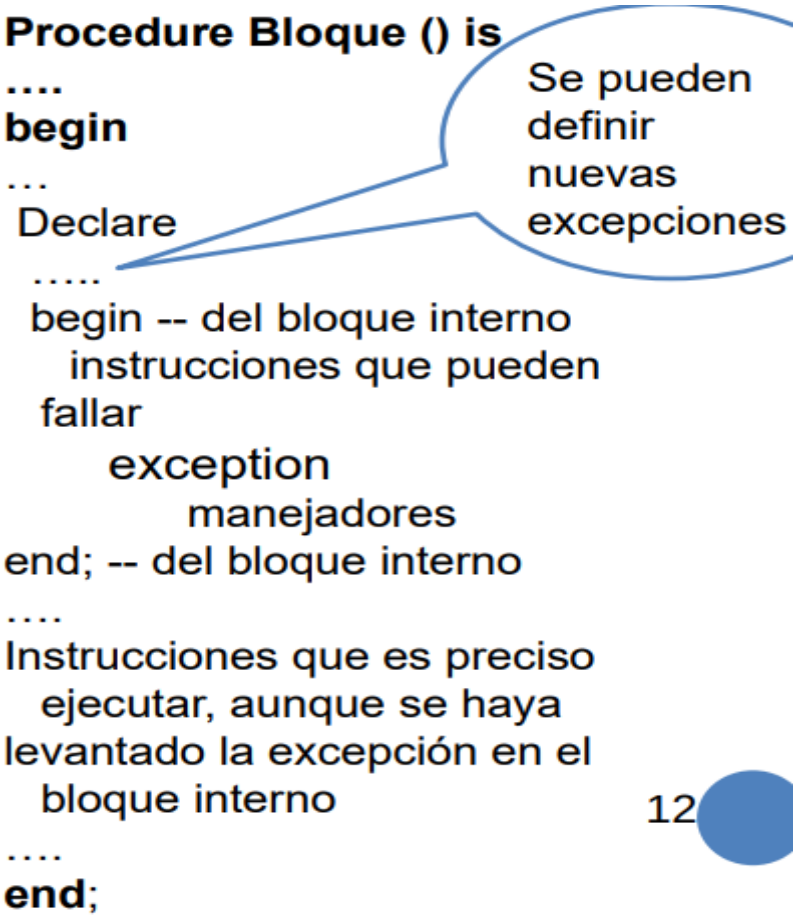


- Cuando hacemos raise dentro del procedure B, se termina el manejador en B y trata la excepción **e** (la **e** local al procedimiento B)que fue levantada de nuevo en el others del manejador en procedure A (sale anónima).
- Si no hay nadie que maneje la excepción, se termina el programa.

- Ejemplo en código bien escrito:

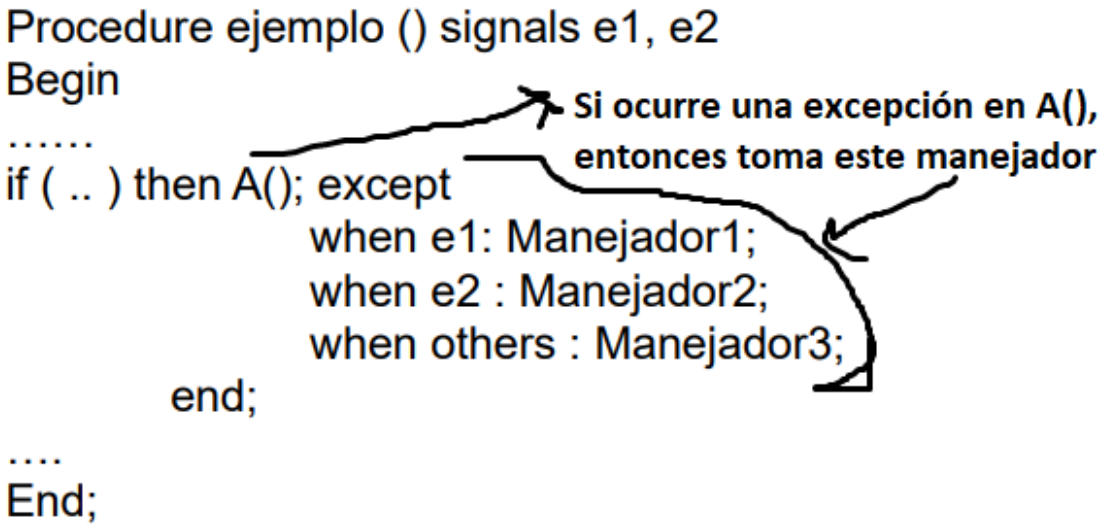
```
1  with Ada.Text_IO; use Ada.Text_IO;
2  procedure Main is
3      e : Exception;
4      dividendo : integer;
5      divisor : integer;
6  begin
7      dividendo := 15;
8      divisor := 0;
9      if divisor = 0 then
10         raise e;
11     end if;
12
13     Exception
14         when e => Put("Division por cero");
15         when others => Put("Otro error");
16
17 end Main;
```

  - Observar que la excepción posible se define en la parte de variables.
  - Como se llega de forma explícita a la excepción, se usa if.
- Tomamos en cuenta
  - La asociación entre excepción y manejador se dá por nombre.
  - Si se quiere continuar ejecutando las instrucciones de un bloque que lanzó una excepción, es necesario crear un bloque interno que contenga las instrucciones que pueden fallar junto al manejador de la excepción.



Excepciones en CLU

- Utiliza el criterio de terminación.
- Las excepciones SOLAMENTE pueden ser alcanzadas por procedimientos.
- Las excepciones están asociadas a sentencias.
    - Si se produce una excepción en un procedimiento, se va a buscar el manejador a la sentencia asociada.



    -

- Las excepciones que un procedimiento puede lanzar se declaran en el encabezado de dicho procedimiento.

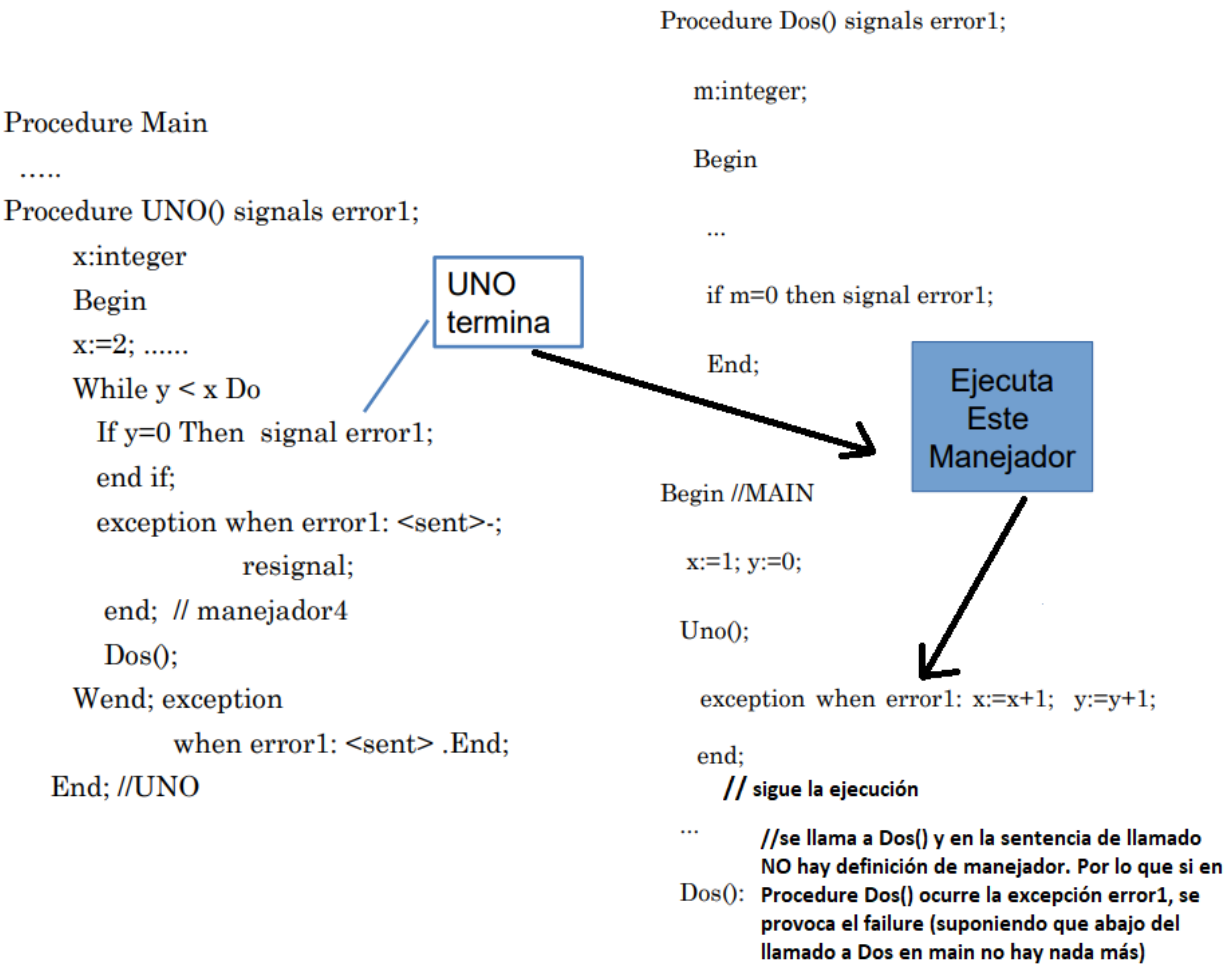
- Son alcanzadas explícitamente con la palabra clave **signal**.
- Los manejadores se colocan al lado de una sentencia simple o compleja.

```
<sentencia> except
when Nombre-Excepción: Manejador1;
when Nombre-Excepción : Manejador2;
.....
when others: Manejadorn;
end;
```

- CLU posee excepciones predefinidas.
- Se pueden pasar parámetros a los manejadores.
- Una excepción se puede levantar de nuevo UNA SOLA VEZ (no como ADA) utilizando resignal.
- La excepción se puede levantar en cualquier lugar del código.

Propagación cuando ocurre una excepción

- El procedimiento donde se levantó la excepción es TERMINADO, se devuelve el control al que llamó al procedimiento, justo en la sentencia donde se hace el llamado donde se debe encontrar al manejador.
  - Por esto asociamos manejadores con sentencias.
- Si el manejador está ahí, se ejecuta y luego se pasa el control a la sentencia siguiente a la que está ligada el manejador.
- Si el manejador no se encuentra en ese lugar, se propaga estáticamente la excepción. Es decir, se busca hacia abajo si hay algún manejador que tome la excepción.
  - Ocurre una sola propagación dinámica: cuando terminamos el procedimiento al principio.
- Si no hallamos ningún manejador en el procedimiento que hizo el llamado al procedimiento que lanzó la excepción, se levanta la excepción **failure** y termina todo el programa.



- Otro caso posible:
  - El if dentro de Procedure Uno no es ejecutado, se ejecutan las sentencias hasta que se llama a Dos() dentro de Procedure Uno.
    - La sentencia while tiene asociado un manejador.
      - Si ocurre una excepción en el procedure Dos, va a buscar ese manejador.

Excepciones en C++

- Criterio de terminación.
- Las excepciones pueden alcanzarse explícitamente a través de la sentencia Throw(nombre_excepcion)
  - Se pueden pasar parámetros al levantar la excepción.
    - Throw(Ayuda msg); msg es el parámetro.
- Tiene excepciones predefinidas.
- Los manejadores de excepciones se asocian a bloques.
  - Conjunto de instrucciones.
- Los bloques que pueden llegar a levantar excepciones van precedidos por la palabra clave **Try**.
- Cuando se finaliza el bloque Try se detallan los manejadores utilizando la palabra clave **Catch(NombreExcepcion)**.
- Cuando se levanta una excepción (Throw dentro del Try) el control del programa se transfiere al manejador correspondiente.

```
Try
{
….. /* Sentencias que pueden provocar una excepción*/
}
catch(NombreExcepción1)
{
….. /* Sentencias Manejador 1*/
}
….
catch(NombreExcepciónN)
{
….. /* Sentencias Manejador N*/
}
…
```

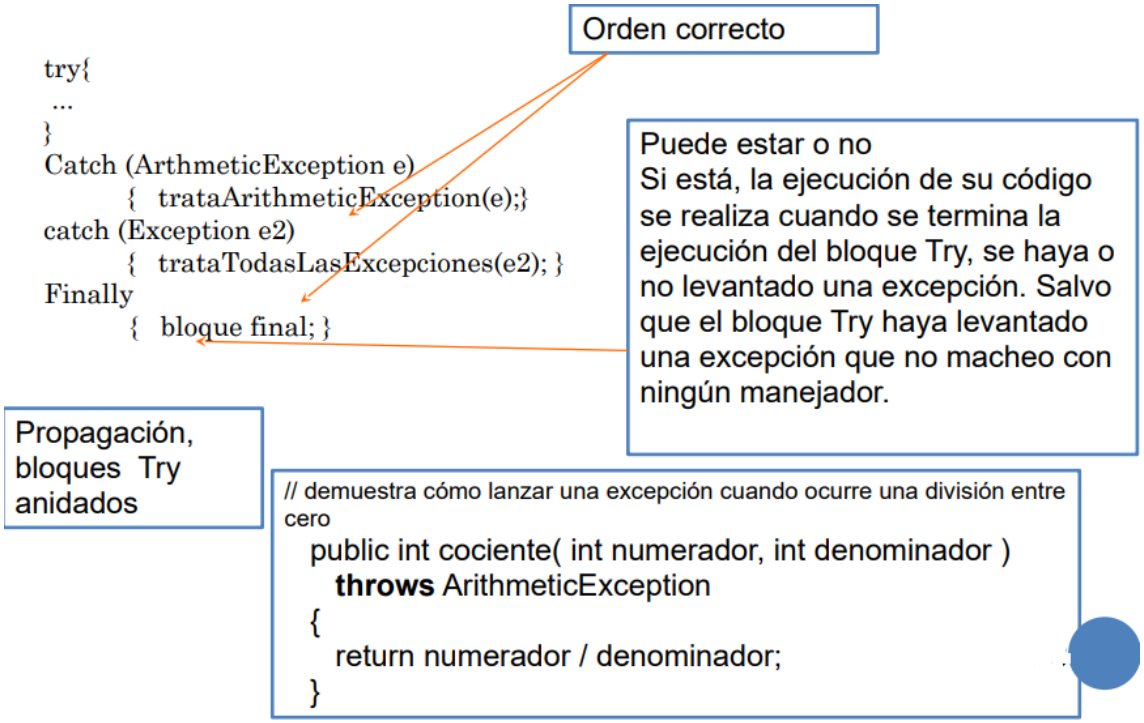
Las excepciones se pueden levantar nuevamente colocando **Throw.**

- Las rutinas pueden listar las excepciones que ellas pueden alcanzar.
  - Void rutina() throw (Ayuda,ZeroDivide);
- ¿Qué sucede si la rutina…?
  - Alcanzó otra excepción no contemplada en el listado.
    - No se propaga la excepción y la función especial unexpected() es ejecutada automáticamente, dicha función normalmente causa abort() que provoca el final del programa.
      - Unexpected puede ser redefinida por el programador.
  - Colocó en su interface el listado de posibles excepciones alcanzables
    - Se puede propagar la excepción. Pero si una excepción se propaga repetidamente y no encuentra manejador, entonces la función terminate() se ejecuta de forma automática.
  - Colocó en su interface una lista vacía -> throw()
    - Ninguna excepción será propagada.

Excepciones en Java

- Igual que C++, las excepciones son objetos alcanzables y manejables por manejadores adicionados al bloque donde se produce la excepción.
- Cada excepción se representa por una instancia de la clase Throwable o de una de sus hijas, Error y Exception.
- La gestión de excepciones usa cinco palabras clave:
  - Try
  - Catch
  - Throw
  - Throws: especificación de qué excepción puede enviarse desde un método (posibles excepciones que puede lanzar)

- Finally: bloque de código que se ejecuta siempre
- Fases del tratamiento:
  - Detectar e informar el error (dos formas):
    - Se lanza la excepción usando throw.
    - Un método detecta una condición anormal que le impide continuar la ejecución y lanza un objeto Excepción.
  - Se recoge el error y se trata (dos formas):
    - Captura de Excepciones -> usando try-catch.
    - Un método recibe un objeto Excepción que le dice que otro método no terminó correctamente su ejecución, entonces se decide como proseguir en función del tipo de error.



Excepciones en Python

- Manejadas con Try except.

```
>>> while True:
...     try:
...         x = int(input("Por favor ingrese un número: "))
...         break
...     except ValueError:
...         print("Oops! No era válido. Intente nuevamente...")
```

- El Try funciona así:
  - Se ejecuta el bloque try.
  - Si no ocurre ninguna excepción, el bloque except es salteado y se termina la ejecución del bloque try.
  - Si ocurre una excepción en el bloque try, el resto del bloque try es salteado. Luego, si el tipo de excepción coincide con alguna excepción nombrada luego

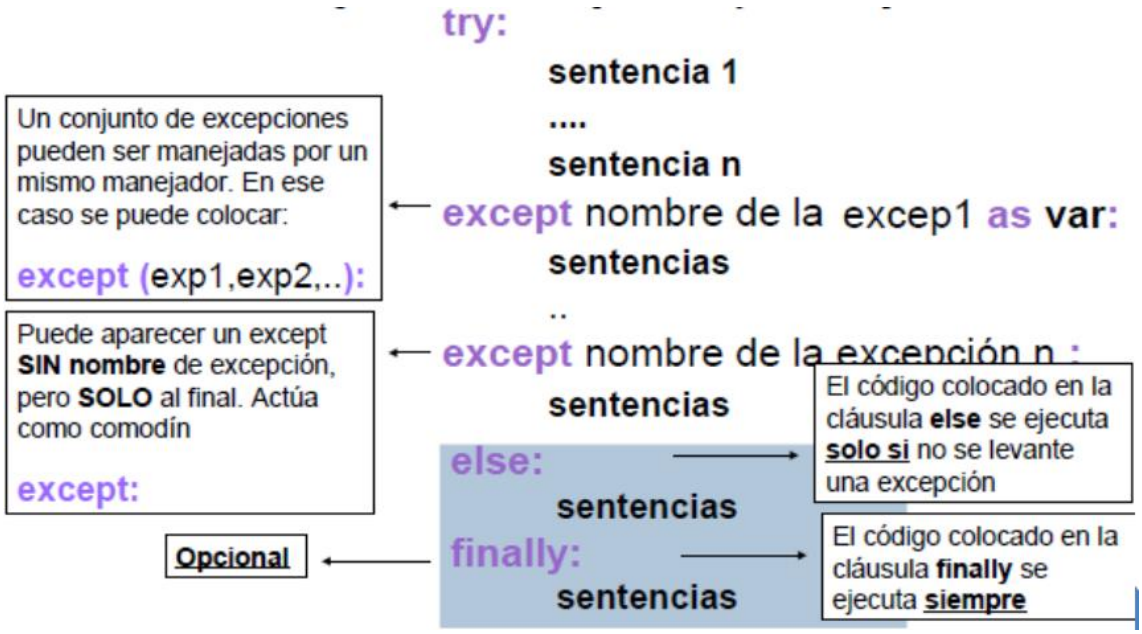
de la palabra reservada except, se ejecuta el bloque except que corresponda, y la ejecución del programa continúa.

- Si ocurre una excepción que no coincide con ninguna excepción nombrada en el except, ésta se pasa a declaraciones try de más afuera, si no se encuentra un manejador, significará que es una excepción no manejada, por lo que la ejecución se frena con un mensaje de error.

```
1  dividendo = 15
2  divisor = 0
3  try:
4      resul = dividendo / divisor
5  except(ValueError):
6      print("OOPS")
7
```

```
Traceback (most recent call last):
  File "<string>", line 4, in <module>
ZeroDivisionError: division by zero
>
```

- Estructura de manejo de excepciones



- ¿Y si una excepción no encuentra un manejador en su bloque try except?
  - Búsqueda estática:
    - Analiza si ese try está contenido dentro de otro y si este otro tiene un manejador para la excepción.
  - Búsqueda dinámica:
    - Analiza quién llamó y busca allí.
  - Si no se halla manejador -> corta el proceso largando el mensaje estándar de error.
  - Las excepciones se pueden levantar explícitamente con “raise”

Excepciones en PHP

- Modelo de terminación
- Una excepción puede ser lanzada (thrown) y atrapada (catched).
- El código está dentro de un bloque try.
- Cada bloque try debe tener **al menos** un bloque catch correspondiente.
- Las excepciones pueden ser lanzadas o relanzadas dentro de un bloque catch.
- Se puede utilizar un bloque finally después de los bloques catch.

Como proceder:

- El objeto lanzado DEBE ser una instancia de la clase Exception o de una subclase de la misma. Intentar lanzar un objeto que no corresponda es un error fatal para PHP.
- Cuando una excepción se lanza, el código siguiente a la declaración no se ejecuta. PHP tratará de hallar el primer bloque catch que coincida. Si una excepción no es capturada entonces se emite un error fatal de PHP.
    - Con el mensaje “Excepción no capturada”
        - A menos que se haya definido un gestor con set_exception_handler()

```php
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('División por cero.');
    }
    return 1/$x;
}
try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Primer finally.\n";
}
try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Segundo finally.\n";
}
// Continuar ejecución
echo 'Hola Mundo\n';
?>
```

El resultado del ejemplo sería:

```
0.2
Primer finally.
Excepción capturada: División por cero.
Segundo finally.
Hola Mundo
```

03. ¿Cuál de esos modelos es más inseguro y por qué?

El modelo por reasuncion es mas inseguro dado que cuando surge una excepcion no termina, sigue ejecutando donde se quedo y puede que eso acarree errores en la ejecución.

Ejercicio 5: La propagación de los errores, cuando no se encuentra ningún manejador asociado, no se implementa igual en todos los lenguajes. Realice la comparación entre el modelo de Java, Python y PL/1, respecto a este tema. Defina la forma en que se implementa en un lenguaje conocido por Ud.

Resuelto en el punto 4.

En JavaScript, cuando no se encuentra ningún manejador asociado a una excepción, la excepción se propaga hacia arriba en la pila de llamadas hasta encontrar un bloque try-catch correspondiente o, en su defecto, se produce una falla no capturada que puede resultar en la terminación del programa.

Ejercicio 6: Sea el siguiente programa escrito en Pascal

```
...
Procedure Manejador;
Begin ... end;
Procedure P(X:Proc);
begin
....
if Error then X;
....
end;
Procedure A;
begin
....
P(Manejador);
….
end;
....
```

¿Qué modelo de manejo de excepciones está simulando? ¿Qué necesitaría el programa para que encuadre con los lenguajes que no utilizan este modelo? Justifique la respuesta.

El modelo de manejo de excepciones que está simulando es reasunción, ya que una vez que se maneja la excepción se continua con la sentencia siguiente de donde se levantó la excepción, es decir, no se termina el bloque. Para que encuadre con los lenguajes que no utilizan este modelo Procedure P debería no tener más código luego del if.

Ejercicio 7: Sea el siguiente programa escrito en Pascal:

```
Program Principal;
var x:int;  b1,b2:boolean;
 Procedure P (b1:boolean);
      var x:int;
      Procedure Manejador1
                        begin
                  x:=x + 1;
                 end;
          begin
          x:=1;
          if b1=true then Manejador1;
          x:=x+4;
          end;
          Procedure Manejador2;
            begin
                  x:=x * 100;
          end;
```

```
Begin
        x:=4;
        b2:=true;
        b1:=false;
        if b1=false then Manejador2;
        P(b);
        write (x);
End.
```

a) Implemente este ejercicio en PL/1 utilizando manejo de excepciones

```
Prog Principal;
DCL x INTEGER;
DCL b1,b2 BOOLEAN;
    PROC P (b1 BOOLEAN);
    DCL x INTEGER;
    BEGIN
        ON CONDITION Manejador1 BEGIN SET x = ARITH(x + 1); END;
        SET X = 1;
        IF b1=true THEN
            SIGNAL CONDITION Manejador1
        SET x = ARITH(x+4);
    END;

BEGIN
    ON CONDITION Manejador2 BEGIN SET x = ARITH(x * 100); END;
    SET X = 4;
    SET b2 = true;
    SET b1 = false;
    IF b1=false THEN SIGNAL CONDITION Manejador2
    CALL P(b1);
    WRITE (x);
END.
```

b) ¿Podría implementarlo en JAVA utilizando manejo de excepciones? En caso afirmativo, realícelo.

Si podria, pero la diferencia esta en que Java utiliza la terminación y hay partes del código que no se ejecutarían si hay una excepción, por ejemplo en PROC P `SET x = ARITH(x+4);` si hay una excepción en Java, eso no se ejecutaría, ya que se terminaría el bloque. Una manera de solucionar esto seria haciendo uso de “finally” en donde debe estar el código que el programador desee que se ejecute siempre.

```java
public class Principal {
    public static void P (boolean b1){
        int x;
        try{
            x = 1;
            if (b1){
                throw new Manejador1Exception();
            }
        }
        catch (Manejador1Exception e){
            x = x + 1;
        }
        finally{
            x = x + 4;
        }

    }

    public static void main(String[] args) {
        int x = 4,
        boolean b1 = true, b2 = false;
        try {
            if (!b1){
                throw new Manejador2Exception();
            }
        }
        catch (Manejador2Excepction e){
            x = x * 100;
        }
        finally{
            P(b1);
            System.out.println(x);
        }
    }
}

public class Manejador1Exception extends Exception{
    public Manejador1Exception(){
        super();
    }
}

public class Manejador2Exception extends Exception{
    public Manejador2Exception(){
        super();
    }
}
```

Ejercicio 8: Sean los siguientes, procedimientos de un programa escrito en JAVA

```
public static void main (String[] argos){
        Double array_doubles[]= new double[500];
        for (int i=0; i<500; i++){
                array_doubles[i]=7*i;
         }
        for (int i=0 ; i<600 ; i=i+25){
             try{
                    system.out.println("El elemento en "+ i + " es " + acceso_por_indice (array_doubles,i));
             }
             catch(ArrayIndexOutOfBoundsException e){
                    system.out.println(e.tostring());
             }
             catch(Exception a){
                    system.out.println(a.tostring());
             }
             finally{
                    system.out.println("sentencia finally");
             }
        }
}

Public static double acceso_por_indice (double [] v, int indice) throws Exception; ArrayIndexOutOfBoundsException{
          if ((indice>=0) && (indice<v.length)){
               Return v[indice];
          }
          else{
               if (indice<0){
                 // caso excepcional
                 Throw new ArrayIndexOutOfBoundsException(" el índice" + indice + " es un número negativo");
```

```
                }
                else{
                  // caso excepcional
                  Throw new Exception(" el indice" + indice + " no es una posición válida");
                  }
          }
}
```

a. Analizar el ejemplo y decir qué manejadores ejecuta y en qué valores quedan las variables. JUSTIFIQUE LA RESPUESTA.

int = 0 - "El elemento en 0 es 0" – "sentencia finally"
int = 25 - "El elemento en 25 es 175" – "sentencia finally"
int = 50 - "El elemento en 50 es 350" – "sentencia finally"
…
int = 475- "El elemento en 475 es 3325" – "sentencia finally"
int = 500 - "El índice 500 no es una posición valida" – "sentencia finally"
int = 525 - "El índice 525 no es una posición valida" – "sentencia finally"
int = 550 - "El índice 550 no es una posición valida" – "sentencia finally"
int = 575 - "El índice 575 no es una posición valida" – "sentencia finally"

En los que se ejecuta un manejador, se ejecuta el manejador del catch(Exception a)

b. La excepción se propaga o se maneja en el mismo método? ¿Qué instrucción se agrega para poder propagarla y que lleve información?

La excepción se propaga dinámicamente del método "acceso_por_indice" al "main". Para propagarla se agrega la declaración "throws Exception, ArrayIndexOutOfBoundsException" en la firma del método "acceso_por_indice". Esto

indica que el método puede lanzar esas excepciones y que cualquier llamada a ese método debe manejar o propagar esas excepciones. En este caso, el método “main” captura y maneja las excepciones utilizando bloques try-catch.

c. **Como modificaría el método “acceso_por_indice” para que maneje él mismo la excepción.**

Agregando un try-catch

```
public static double acceso_por_indice(double[] v, int indice) {
    try {
        if ((indice >= 0) && (indice < v.length)) {
            return v[indice];
        } else {
            if (indice < 0) {
                throw new ArrayIndexOutOfBoundsException("El índice " +
indice + " es un número negativo");
            } else {
                throw new Exception("El índice " + indice + " no es una
posición válida");
            }
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(e.toString());
        return 0.0;
    } catch (Exception a) {
        System.out.println(a.toString());
        return 0.0;
    }
}
```

Ejercicio 9: Indique diferencias y similitudes entre Phyton y Java con respecto al manejo de excepciones.

El manejo de excepciones en Python y Java comparte algunos conceptos fundamentales, pero también hay diferencias significativas. A continuación, se presentan las similitudes y diferencias clave entre Python y Java en cuanto al manejo de excepciones:

<u>Similitudes:</u>

1. Estructura try: Ambos lenguajes utilizan una estructura try para capturar y manejar excepciones. Se puede colocar código potencialmente problemático dentro de un bloque try, y las excepciones se capturan y manejan en bloques catch o except correspondientes.
2. Tipos de excepciones: Tanto Python como Java tienen una jerarquía de clases de excepciones predefinidas. Ambos lenguajes proporcionan una serie de excepciones estándar, como NullPointerException, IOException, entre otras, que se utilizan para capturar errores comunes.

3. Bloque finally: Ambos lenguajes permiten el uso de un bloque finally opcional, que se ejecuta siempre, ya sea que se produzca o no una excepción. Se utiliza para realizar limpieza de recursos u otras acciones necesarias, independientemente de si se lanzó o no una excepción.

Diferencias:

1. Declaración de excepciones: En Java, se requiere que los métodos declaren las excepciones específicas que pueden lanzar utilizando la palabra clave throws en su firma. En Python, no se requiere una declaración explícita de excepciones en la firma de un método.
2. Tipos de excepciones: En Java, las excepciones se dividen en dos categorías: excepciones comprobadas (checked exceptions) y excepciones no comprobadas (unchecked exceptions). Las excepciones comprobadas deben ser declaradas o manejadas explícitamente en el código. En Python, todas las excepciones son consideradas excepciones no comprobadas y no requieren una declaración explícita o manejo.
3. Bloque except: En Python, se utiliza la palabra clave except para capturar excepciones específicas y manejarlas en un bloque de código correspondiente. En Java, se utiliza la palabra clave catch para capturar y manejar excepciones.

Ejercicio 10: ¿Qué modelo de excepciones implementa Ruby?. ¿Qué instrucciones específicas provee el lenguaje para manejo de excepciones y cómo se comportan cada una de ellas?

Proporciona una serie de instrucciones específicas para el manejo de excepciones que permiten capturar, lanzar y manejar errores en el código. A continuación, se describen las instrucciones principales y su comportamiento en Ruby:

1. begin-rescue-end: Esta es la estructura básica utilizada para capturar y manejar excepciones en Ruby. El código problemático se coloca dentro del bloque begin, y las excepciones se capturan y manejan en el bloque rescue. Por ejemplo:

   ```ruby
   begin
     # Código problemático
   rescate ExceptionType
     # Manejo de la excepción
   end
   ```

   ExceptionType puede ser una clase de excepción específica o el tipo

   StandardError para capturar cualquier excepción estándar.

2. raise: Esta instrucción se utiliza para lanzar una excepción explícitamente en un punto determinado del código. Puede lanzar una excepción predefinida o una excepción personalizada. Por ejemplo:

   ```ruby
   raise ExceptionType, "Mensaje de error"
   ```

   ExceptionType es la clase de excepción que se lanza.

   "Mensaje de error" es un mensaje opcional que proporciona información adicional sobre la excepción.

3. rescue: Esta instrucción se utiliza dentro de un bloque begin-rescue para capturar excepciones específicas. Puede capturar excepciones individuales o agrupar múltiples excepciones. Por ejemplo:

```
begin
  # Código problemático
rescue ExceptionType1
  # Manejo de la excepción de tipo ExceptionType1
rescue ExceptionType2, ExceptionType3
  # Manejo de las excepciones de tipo ExceptionType2 y
ExceptionType3
end
```

   Se pueden proporcionar múltiples cláusulas rescue para capturar diferentes tipos de excepciones.

4. ensure: Esta instrucción se utiliza dentro de un bloque begin-rescue para ejecutar código que se debe ejecutar siempre, independientemente de si se produce o no una excepción. Por ejemplo:

```
begin
  # Código problemático
rescue ExceptionType
  # Manejo de la excepción
ensure
  # Código que se ejecuta siempre
end
```

   El bloque ensure se ejecutará incluso si no hay una coincidencia de excepción en el bloque rescue correspondiente.

5. Excepciones predefinidas: Ruby proporciona una serie de excepciones predefinidas que se pueden capturar y manejar según sea necesario. Algunas de las excepciones comunes incluyen StandardError (base de las excepciones estándar), TypeError, ArgumentError, ZeroDivisionError, entre otras. Estas excepciones se pueden capturar y manejar individualmente utilizando la cláusula rescue.

Además de estas instrucciones y conceptos principales, Ruby también ofrece funcionalidades adicionales como la definición de excepciones personalizadas mediante la creación de clases, el acceso a la traza de la pila de excepciones (backtrace), y la posibilidad de propagar excepciones a través de múltiples niveles de llamadas de métodos.

En el caso de Ruby, encaja más con el enfoque de "Reasunción" cuando se produce una excepción. En Ruby, cuando se lanza una excepción y se captura en un bloque rescue, una vez que se maneja la excepción, el control vuelve a la sentencia siguiente de donde se levantó la excepción.

Ejercicio 11: Indique el mecanismo de excepciones de JavaScript.

JavaScript utiliza un mecanismo de excepciones similar a otros lenguajes de programación. Proporciona una estructura de control try-catch-finally para capturar y manejar excepciones. A continuación se describe el mecanismo de excepciones de JavaScript:

1. try-catch-finally: Esta estructura se utiliza para capturar y manejar excepciones en JavaScript. El código problemático se coloca dentro del bloque try, y las excepciones se capturan y manejan en el bloque catch. El bloque finally es opcional y se utiliza para ejecutar código que siempre debe ejecutarse, independientemente de si se produce una excepción o no. Por ejemplo:

   ```
   try {
     // Código problemático
   } catch (error) {
     // Manejo de la excepción
   } finally {
     // Código que se ejecuta siempre
   }
   ```

   El bloque catch captura la excepción y la asigna a la variable error (que puede tener cualquier nombre que elijas).
   El bloque finally se ejecutará incluso si no hay una coincidencia de excepción en el bloque catch correspondiente.

2. throw: Esta instrucción se utiliza para lanzar una excepción explícitamente en un punto determinado del código. Puede lanzar una excepción predefinida o una excepción personalizada. Por ejemplo:

   ```
   throw new Error('Mensaje de error');
   ```

   Error es una de las excepciones predefinidas en JavaScript, pero también puedes crear tus propias excepciones personalizadas.

3. Excepciones predefinidas: JavaScript proporciona un conjunto de excepciones predefinidas, como Error, TypeError, ReferenceError, SyntaxError, entre otras. Estas excepciones se pueden capturar y manejar utilizando la estructura try-catch.

4. Error object: En JavaScript, todas las excepciones son objetos que heredan de la clase Error. Estos objetos contienen información sobre el error, como el mensaje de error, la traza de la pila y otros detalles relevantes.

Es importante tener en cuenta que JavaScript también proporciona otros mecanismos adicionales para manejar excepciones, como el uso de bloques catch anidados para capturar excepciones específicas y el acceso a la traza de la pila a través de la propiedad stack del objeto Error. Además, las excepciones en JavaScript pueden propagarse a través de las llamadas de función hasta que se capturen o lleguen al ámbito global, donde pueden generar un error no capturado que detiene la ejecución del programa.

Ejercicio 12: Sea el siguiente programa escrito en PYTHON::Indique el camino de ejecución.

```
#!/usr/bin/env python
#calc.py

def dividir():
      x = a / b
      print (("Resultado"), (x))
```

```
while True:
      try:
        a = int(input("Ingresa el primer numero: \n"))
        b = int(input("Ingresa el segundo numero: \n"))
        dividir()
        break
      except ZeroDivisionError:
        print ("No se permite dividir por cero")
      finally:
        print ("Vuelve a probar")
```

a) Describa qué caminos ejecuta para diferentes valores de ingreso

Si se ingresa 4 y 2: “Resultado 2”
Si se ingresa 4 y 0: “No se permite dividir por cero” – “Vuelve a probrar”
Si se ingresa x e y: se produce un ValueError al intentar convertir el valor ingresado a entero, el programa lanzará la excepción sin capturarla y mostrará el rastreo de la pila (traceback) junto con un mensaje de error estándar. En este caso, el bloque finally no se ejecutará y el programa se detendrá.

b) Agregar el uso de una excepción anónima

```
def dividir():
    x = a / b
    print("Resultado:", x)

while True:
    try:
        a = int(input("Ingresa el primer numero: \n"))
        b = int(input("Ingresa el segundo numero: \n"))
        dividir()
        break
    except ZeroDivisionError:
        print("No se permite dividir por cero")
    except Exception:
        print("Ha ocurrido una excepcion")
    finally:
        print("Vuelve a probar")
```

Ejercicio 13: Sea el siguiente código escrito en JAVA

```
public class ExcepcionUno extends Exception {
    public ExcepcionUno(){
        super(); // constructor por defecto de Exception
    }

    public ExcepcionUno( String cadena ){
        super( cadena ); // constructor param. de Exception
    }
}

public class ExcepcionDos extends Exception {
    public ExcepcionDos(){
        super(); // constructor por defecto de Exception
```

```
    }
    public ExcepcionDos( String cadena ){
        super( cadena ); // constructor param. de Exception
    }
}

public class ExcepcionTres extends Exception {
    public ExcepcionTres(){
        super(); // constructor por defecto de Exception
    }
    public ExcepcionTres( String cadena ){
        super( cadena ); // constructor param. de Exception
    }
}

public class Lanzadora {
    public void lanzaSiNegativo(int param) throws ExcepcionUno {
        if (param < 0)
            throw new ExcepcionUno("Numero negativo");
    }
    public void lanzaSimayor100(int param) throws ExcepcionDos {
        if (param >100 and param<125)
            throw new ExcepcionDos("Numero mayor100");
    }
    public void lanzaSimayor125(int param) throws ExcepcionTres {
        if (param >= 125)
            throw new ExcepcionTres("Numero mayor125");
    }
}

import java.io.FileInputStream;
import java.io.IOException;
public class Excepciones {

    public static void main(String[] args) {
        // Para leer un fichero
        Lanzadora lanza = new Lanzadora();
        FileInputStream entrada = null;
        int leo;
        try {
            entrada = new FileInputStream("fich.txt");
            while ((leo = entrada.read()) != -1){
                if (leo < 0)
                    lanza.lanzaSiNegativo(leo);
                else if (leo > 100)
                    lanza.lanzaSimayor100(leo);
            }
            entrada.close();
            System.out.println("Todo fue bien");
```

```
        }
        catch (ExcepcionUno e) { // Personalizada
            System.out.println("Excepcion: " + e.getMessage());
        }
        catch (ExcepcionDos e) { // Personalizada
            System.out.println("Excepcion: " + e.getMessage());
        }
        catch (IOException e) { // Estándar
            System.out.println("Excepcion: " + e.getMessage());
        }
        finally {
            if (entrada != null)
                try {
                    entrada.close(); // Siempre queda cerrado
                }
                catch (Exception e) {
                    System.out.println("Excepcion: " + e.getMessage());
                }
            System.out.println("Fichero cerrado.");
        }
    }
}
```

a) Indique cómo se ejecuta el código. Debe quedar en claro los caminos posibles de ejecución, cuales son los manejadores que se ejecutan y cómo se buscan los mismos y si en algún caso se produce algún error.

Ejecución

1. Se instancia un objeto de la clase Lanzadora, que es la que lanza las excepciones.
2. Se inicializa un objeto de la clase FileInputStream como null.
3. Se declara una variable leo de tipo int.
4. Se inicia un bloque try para manejar posibles excepciones.
    a. Se instancia “entrada” para leer el archivo “fich.txt”
    b. Se inicia un bucle while que lee el archivo mientras haya contenido por leer.
        i. Si el valor leido es negativo, se llama a el método de “lanzaSiNegativo” que lanza una instancia de la excepción “ExcepcionUno” con el mensaje "Numero negativo".
        ii. Si el valor leído es mayor que 100, se verifica si es menor que 125. Si se cumple esta condición, se llama a el método de “lanzaSimayor100” que lanza una instancia de la excepcion “ExcepcionDos” con el mensaje "Numero mayor100".
    c. Si no ocurre ninguna excepción en la lectura, se cierra el archivo de entrada y se muestra el mensaje "Todo fue bien".
5. Si ocurre alguna excepción de tipo ExcepcionUno, se captura en el bloque catch (ExcepcionUno e) y se muestra el mensaje "Excepcion: " seguido del mensaje de la excepción (que sería Numero negativo).

6. Si ocurre alguna excepción de tipo ExcepcionDos, se captura en el bloque catch (ExcepcionDos e) y se muestra el mensaje "Excepcion: " seguido del mensaje de la excepción (que sería Numero mayor100).
7. Si ocurre alguna excepción de tipo IOException, se captura en el bloque catch (IOException e) y se muestra el mensaje "Excepcion: " seguido del mensaje de la excepción.
8. En el bloque finally:
    a. Se verifica si la variable entrada no es nula y se cierra el archivo en caso de que esté abierto.
        i. Si ocurre alguna excepción al cerrar el archivo, se muestra el mensaje "Excepcion: " seguido del mensaje de la excepción.
    b. Se muestra el mensaje "Fichero cerrado".

Posibles caminos:

1. No ocurre ninguna excepción en la lectura del archivo:
    a. Se cierra el archivo y se imprime “Todo fue bien” y “Fichero cerrado”.
2. Ocurre una excepción en la lectura del archivo debido a que el valor leído es negativo:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionUno”.
    b. Se captura la excepción (en el catch), e imprime “Excepción: Numero negativo”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
3. Ocurre una excepción en la lectura del archivo debido a que el valor leído es mayor a 100 y menor a 125:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionDos”.
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: Numero mayor100”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
4. Ocurre una excepción en la lectura del archivo de tipo “IOException”.
    a. Se lanza una Excepción de tipo “IOException”.
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: (mensaje de IOException no se cual es)”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
5. Ocurre una excepción en el cierre del archivo en el bloque finally (es decir, el cierro del archivo si ocurrió una excepción en la lectura):
    a. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: (mensaje de Excepción no se cual es)”.
    b. Se muestra el mensaje “Fichero cerrado”.

Ejercicio 14. Dado el siguiente código en Java. Indique todos los posibles caminos de resolución, de acuerdo a los números que vaya leyendo del archivo.

```
class ExcepcionE1 extends Exception {
    public ExcepcionE1(){
        super(); // constructor por defecto de Exception
    }
    public ExcepcionE1( String cadena ){
```

```
        super( cadena ); // constructor param. de Exception
    }
}

class ExcepcionE2 extends Exception {
    public ExcepcionE2(){
        super(); // constructor por defecto de Exception
    }
    public ExcepcionE2( String cadena ){
        super( cadena ); // constructor param. de Exception
    }
}

// Esta clase lanzará la excepción
public class Evaluacion {
    void Evalua( int edad ) throws ExcepcionE1, ExcepcionE2 {
        if ( edad < 18 )
            throw new ExcepcionE1( "Es una persona menor de edad" );
        else if ( edad> 70 )
            throw new ExcepcionE2( "Es persona mayor de edad" );
    }
    void Segmenta( int edad ) throws ExcepcionE1, ExcepcionE2 {
        if ( edad < 35 )
            throw new ExcepcionE1( "Es una persona joven" );
    }
}

class AnalisisEdadPoblacion{

    public static void main( String[] args ){
        // Para leer un fichero
        Evaluacion Invoca = new Evaluacion();
        FileInputSt ream entrada = null;
        int leo;
        try{
            entrada = new FileInputStream( "fich.txt" );
            while ( ( leo = entrada.read() ) != -1 ) {
                try {
                    if (leo<0) {
                        throw new ExcepcionE1( "Edad inválida" );
                    }
                    else{
                        if (leo>120){
                            throw new ExcepcionE1( "Edad inválida" );
                        }
                    }
                    invoca.evalua (leo);
                    invoca.segmenta( leo );
```

```
                    System.out.println( "Es persona adulta, Todo fue
bien" );
                }
                catch ( ExcepcionE2 e ){
                    System.out.println( "Excepcion: " + e.getMessage() );
                }
                catch ( ExcepcionE1 e ){
                    System.out.println( "Excepcion: " + e.getMessage() );
                }
            }
        }
        catch (FileNotFoundException e1) {
            System.out.println("No se encontró el archivo");
        }
        catch (IOException e) {
            System.out.println("Problema para leer los datos");
        }
        finally {
            if (entrada != null)
                try {
                    entrada.close();
                }
                catch (Exception e) {
                    System.out.println("Excepcion: " + e.getMessage());
                }
            System.out.println("Fichero cerrado.");
        }
    }
}
```

Posibles caminos:

1. No ocurre ninguna excepción en la lectura del archivo y en la lectura de las edades:
    a. Se cierra el archivo y se imprime “Es persona adulta, Todo fue bien” y “Fichero cerrado”.
2. Ocurre una excepción en la lectura de las edades debido a que la edad leída es negativa:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionE1”.
    b. Se captura la excepción (en el catch), e imprime “Excepción: Edad inválida”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
3. Ocurre una excepción en la lectura de las edades debido a que la edad leída es mayor a 120:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionE1”.
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: Edad inválida”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.

4. Ocurre una excepción en la lectura de las edades debido a que la edad es menor a 18:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionE1” (en realidad la lanza el método evalua del objeto “invoca” y esta excepción se propaga).
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: Es una persona menor de edad”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
5. Ocurre una excepción en la lectura de las edades debido a que la edad es mayor a 70:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionE2” (en realidad la lanza el método evalua del objeto “invoca” y esta excepción se propaga).
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: Es persona mayor de edad”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
6. Ocurre una excepción en la lectura de las edades debido a que la edad es menor a 35:
    a. En el bloque try, se lanza una Excepción de tipo “ExcepcionE1” (en realidad la lanza el método segmenta del objeto “invoca” y esta excepción se propaga).
    b. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: Es una persona joven”.
    c. Se cierra el archivo y se muestra el mensaje “Fichero cerrado”.
7. Ocurre una excepción “FileNotFoundException” en la lectura del archivo, ya que no se encuentra el archivo:
    a. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “No se encontró el archivo".
    b. Se muestra el mensaje “Fichero cerrado”.
8. Ocurre una excepción “IOException” en la lectura del archivo:
    a. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Problema para leer los datos".
    b. Se muestra el mensaje “Fichero cerrado”.
9. Ocurre una excepción en el cierre del archivo en el bloque finally:
    a. Se captura la excepción (en el catch), el manejador controla la excepción e imprime “Excepción: (mensaje de Excepción no se cual es)”.
    b. Se muestra el mensaje “Fichero cerrado”.