

EXCEPCIONES

CYPLP

EXCEPCIONES

Los programadores a menudo escriben programas bajo la ***suposición optimista de que nada saldrá mal*** cuando se ejecute el programa. ***Pero esto no es así.***

desbordamiento
de pila



división por cero

acceder a zona de
memoria
no permitida

índices fuera de
rango

EXCEPCIONES

Estas cuestiones pueden ser atendidas por

- ***mecanismos del lenguaje*** (*construcciones especializadas, excepciones*)
- ***mecanismos de hardware*** (interrupciones)
- ***mecanismos del Sistema Operativo*** (*comunicación entre procesos (IPC), señales*)



EXCEPCIONES

¿Qué es una excepción?

Condición inesperada o inusual, que ocurre durante la ejecución del programa y no puede ser manejada en el contexto local.

Denota un comportamiento anómalo e indeseable que supuestamente ocurre raramente, pero es necesario controlarlo. Pero, suelen ocurrir con frecuencia.

EXCEPCIONES - CASO ARIADNE 5

- **Cohete Ariane 5 explotó a los 37 segundos por una excepción de software no manejada.**

4 June 1996

Ariane 5 rocket launched by the European Space Agency exploded just under forty seconds after its lift-off



Butrous Foundation

www.butrousfoundation.com

**Más de 10 años de fabricación
Mas de 7 millones de U\$S**

El software utilizado en el A5 fue heredado del A4. El A5 generó valores más grandes de lo que otro software podía manejar, lo que provocó una excepción de desbordamiento entero (integer overflow) durante la conversión de un número de punto flotante a un entero de 16 bits. La excepción no estaba controlada y se ejecutaron otras acciones

EXCEPCIONES

La **excepción** interrumpe el flujo normal de ejecución y ejecuta un **controlador de excepciones** registrado previamente.

Los **lenguajes** pueden proporcionar estas facilidades, pero **no todos funcionan igual**.

EXCEPCIONES

Para que un lenguaje trate excepciones debe proveer mínimamente:

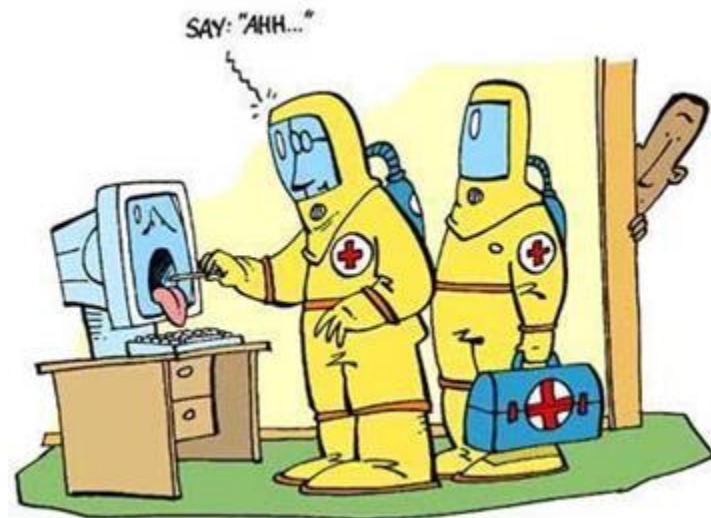
- Un modo de **definirlas**
- Una forma de **reconocerlas**
- Una forma de **lanzarlas y capturarlas**
- Una forma de **manejarlas** especificando el **código y respuestas**
- Un **criterio de continuación**

EXCEPCIONES

Controlador de excepciones/Manejador

Es una **sección de código** que se encarga de **manejar una excepción** en un programa.

Su **objetivo principal** es proporcionar una forma de **recuperarse de un error o falla**, permitiendo que el **programa continúe ejecutándose en lugar de detenerse abruptamente**.



EXCEPCIONES

Controlador de excepciones

Encargado de manejar la excepción.

Puede tomar distintas acciones según la situación:

- **Imprimir un mensaje de error**
- **Realizar acciones correctivas**
- **Lanzar otra excepción**
- **Finalizar la ejecución del programa**

Debe ser la solución menos perjudicial

EXCEPCIONES

Tipos de excepciones:

- **Implícitas: Definidas por el lenguaje (*built-in*)**
- **Explícitas: Definidas por el programador**

EXCEPCIONES

¿Qué se debe tener en cuenta un lenguaje que provee manejo de excepciones?

- **¿Cuáles son las excepciones que se pueden manejar?**
¿Cómo se definen?
- **¿Cómo se maneja una excepción y cuál es su ámbito?**
- **¿Cómo se lanza una excepción?**
- **¿Cómo especificar los controladores de excepciones que se han de ejecutar cuando se alcanza las excepciones?**
- **¿A dónde se cede el control cuando se termina de atender una excepción?**
- **¿Cómo se propagan las excepciones?**
- **¿Hay excepciones predefinidas?**
- **¿Hay situaciones no controladas que lleven a mayores fallos?**

Dependerá de cada lenguaje

EXCEPCIONES

Los lenguajes deben proveer instrucciones para:

- **Definición** de la excepción
- **Levantamiento** de una excepción
- **Manejador** de la excepción

EXCEPCIONES

Punto de retorno:

Después de atender a una excepción, el punto de retorno dependerá del flujo de ejecución del programa y de cómo se haya diseñado el manejo de excepciones en el código. También va a depender del lenguaje.

Se puede tener en cuenta:

- **Continuar la ejecución normal del programa**
- **Retornar a un estado anterior**
- **Propagar la excepción**
- **Terminar la ejecución del programa**

EXCEPCIONES

Continuar la ejecución normal del programa:

- Si después de manejar una excepción **el programa puede continuar** la ejecución del código restante sin problemas
- El **punto de retorno** será **definido por el lenguaje** (por ejemplo, el siguiente bloque de código después del bloque de manejo de excepciones o siguiente instrucción)

Retornar a un estado anterior:

- Cuando el manejo de excepciones puede requerir que el programa **regrese a un estado anterior o deshaga acciones** realizadas antes de que se produjera la excepción.

EXCEPCIONES

Propagar la excepción:

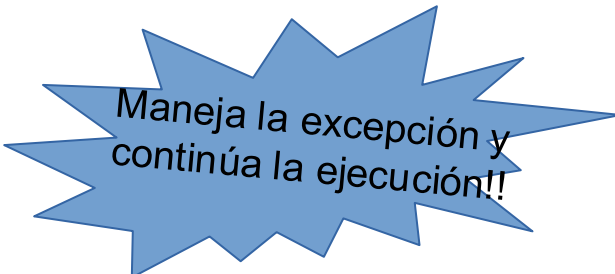
- En el controlador de excepciones que no puede manejar completamente la excepción, puede optar por propagarla a un nivel superior en la jerarquía de llamadas.
- El punto de retorno sería el controlador de excepciones en el nivel superior que pueda manejar la excepción o decidir cómo manejarla.

Terminar la ejecución del programa:

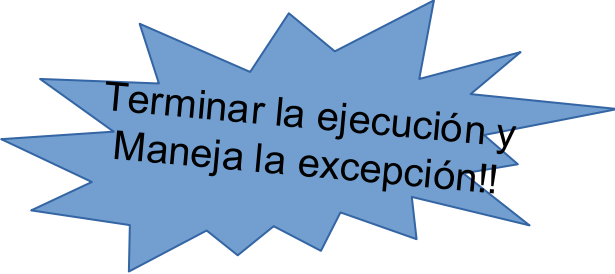
- En situaciones excepcionales o críticas, es posible que el controlador de excepciones determine que no se puede continuar ejecutando el programa de manera segura.
- El punto de retorno puede ser la finalización del programa o alguna acción específica de cierre antes de la finalización.

EXCEPCIONES – 2 MODELOS DE MANEJO DE EXCEPCIONES

- **Reasunción:** se refiere a la **posibilidad de retomar la ejecución normal del programa después de manejar una excepción**. El controlador de excepciones realiza las acciones necesarias para manejar la excepción (medidas correctivas) y luego **el programa continúa su ejecución a partir del punto donde se produjo la excepción**.
- **Terminación:** El controlador de excepciones realiza las acciones necesarias para manejar la excepción, pero no se retorna al punto donde se produjo la excepción (invocador), **continúa su ejecución a partir de la finalización del manejador**.



Maneja la excepción y continúa la ejecución!!



Terminar la ejecución y Maneja la excepción!!

Terminación es el más utilizado actualmente

EXCEPCIONES – 2 MODELOS DE MANEJO DE EXCEPCIONES

● Reasunción

- PL/1

Maneja y continúa!!

● Terminación

- ADA
- CLU
- C++
- Java
- Phyton
- PHP

Termina ymaneja!!



ALGUNOS LENGUAJES QUE INCORPORARON EL MANEJO DE EXCEPCIONES- PL/I

- Fue el primer lenguaje. que incorporó el manejo de excepciones.
- Utiliza el criterio de **Reasunción**. Cada vez que se produce la excepción, la maneja **el manejador y devuelve el control a la sentencia siguiente de dónde se levantó.**
- Las excepciones se llaman **CONDITIONS** en PL/I
- Los manejadores se declaran con la sentencia **ON**:
ON CONDITION(Nombre-excepción) Manejador
- El Manejador puede ser una instrucción o un bloque (entre **begin** y **end**)
- Las **exepciones** se lanzan explícitamente con la palabra clave **SIGNAL**:

SIGNAL CONDITION(Nombre-excepción)

EXCEPCIONES – PL1 – EJEMPLO

FORMATO EN PL/1

Prog Main

PROC UNO

Begin

...

ON CONDITION PEPE begin ... end;

..

If (condError) then

SIGNAL CONDITION PEPE

end

....

Begin

...

ON CONDITION PIPO begin ... end;

..

ON CONDITION PEPE begin ... end;

...

UNO;

...

If (condError) then

SIGNAL CONDITION PIPO

end

*****DECLARACION (ON....)
y MANEJADOR3 (BEGIN...END)**

***** INVOCACION (SIGNAL....)**

*****DECLARACION y MANEJADOR1**

*****DECLARACION y MANEJADOR2**

***** INVOCACION (SIGNAL....)**

EXCEPCIONES – PL/I

- Este lenguaje tiene una serie de **excepciones** ya **predefinidas** con su manejador asociado. Son las **Built-in exceptions**.

Por ej. **zerodivide**, se levanta cuando hay una división por cero.

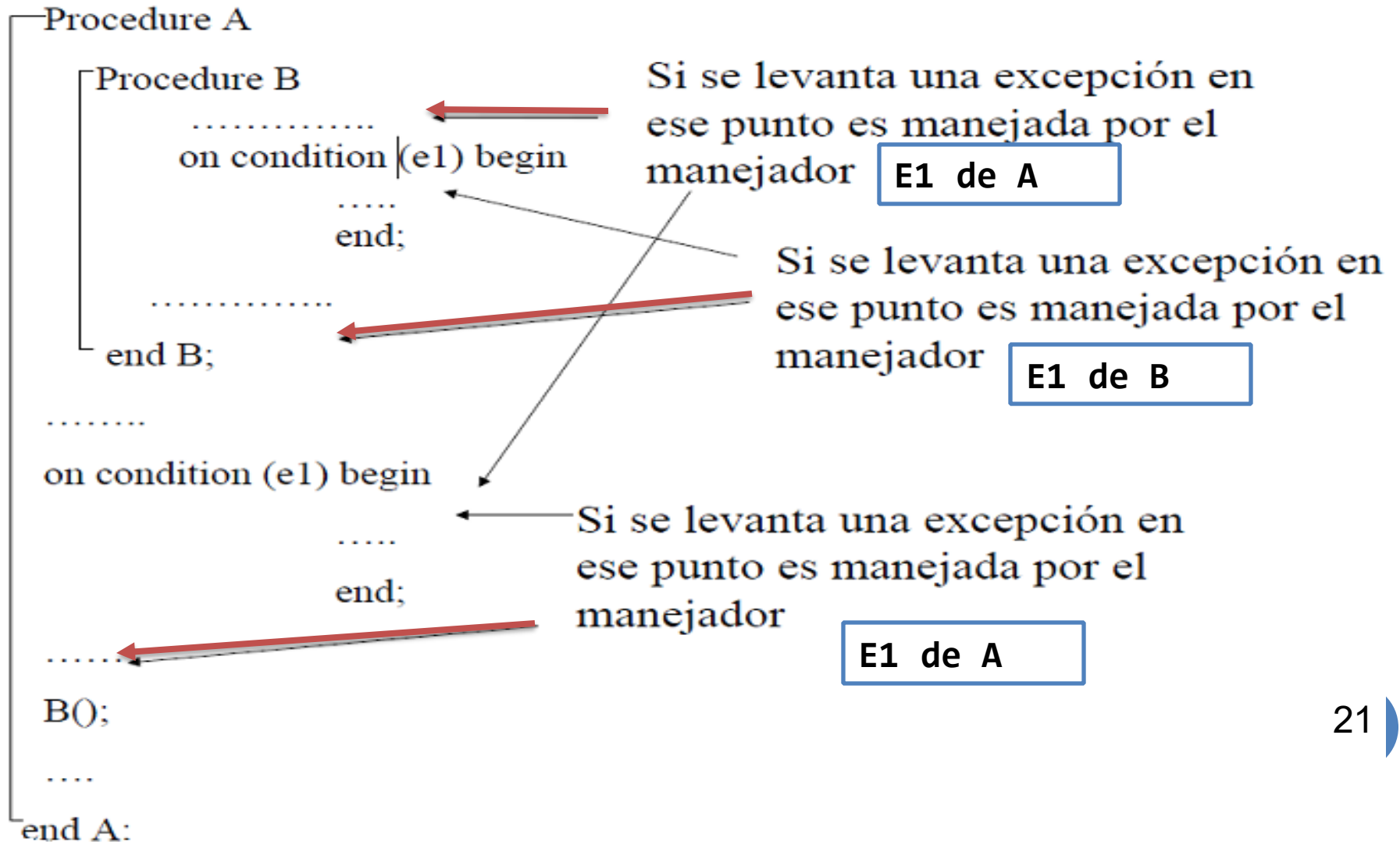
- Los manejadores se **ligan dinámicamente** con las **excepciones**. Una excepción siempre estará ligada con el **último manejador definido**.
(Manejo de pila de manejadores de excepciones)
- El alcance de un manejador termina cuando finaliza la ejecución de la unidad donde fue declarado.

Pila de
Manejadores

e1 (B)

e1 (A)

EXCEPCIONES – PLI EJEMPLO



EXCEPCIONES – ADA

- Aplica criterio de **Terminación**

Cada vez que se produce una excepción, se **termina el bloque donde se levantó y se ejecuta el manejador asociado, y continúa luego.**

- Las **excepciones se definen/declaran en la zona de definición de variables y tienen el mismo alcance en la unidad.**

Su formato para declarar es

MiExcepcion: exception;

EXCEPCIONES – ADA

- La lista de controladores de excepciones lleva el prefijo de la palabra clave **exception**.
- Cada controlador lleva el prefijo de la palabra clave **when** (con un formato específico), seguido de las acciones.
- Se puede utilizar **when others** para capturar cualquier excepción no especificada,
 - debe colocarse al final del bloque de manejo de excepciones,
 - posee efectos colaterales
- Las excepciones se levantan explícitamente con la palabra clave **raise**.
- Los manejadores pueden agregarse y encontrarse al final de diferentes unidades de programa:
Subprograma, Bloque, Procedimiento, Paquete o Tarea que maneja la excepción.

EXCEPCIONES – ADA

begin --this is a block with exception handlers

... statements ...

exception **when** *Help* => *handler for exception Help*

when *Constraint_Error* => *handler for exception*
Constraint_Error, which might be raised by a
division by zero

when others => *handler for any other exception that is not Help*
nor Constraint_Error

end;

Se debe ser cuidadoso
porque entrarían todas
las restantes
excepciones

EXCEPCIONES – ADA

- **Tiene varias excepciones predefinidas built-in:**
- **Constraint_Error:** cuando se intenta *violar una restricción impuesta en una declaración*. (Por ejemplo, *indexar más allá de los límites de un array* o *asignar a una variable un valor fuera del rango de su subtipo, zerodivide*)
- **Program_Error:** cuando se intenta *violar la estructura de control o regla del lenguaje*. (Por ejemplo, *una función termina sin devolver un valor*)
- **Storage_Error:** cuando se produce una **violación de memoria**, (Por ejemplo, *cuando se requiere más memoria de la disponible*)
- **Tasking_Error:** cuando hay *errores en la comunicación y manejo de tareas del sistema*. (Por ejemplo, *en concurrencia y la programación de tareas /threads*)
- **Name_Error:** cuando hay **error de nombre**. (Por ejemplo, *se produce cuando se intenta abrir un fichero que no existe*)

EXCEPCIONES – ADA

Propagación

- Si la unidad que genera la excepción **proporciona un manejador** para la misma, **el control se transfiere inmediatamente a ese manejador**:
 - se omiten las acciones que siguen al punto en el que se generó la excepción,
 - se ejecuta el manejador
 - luego el programa continúa su ejecución normalmente, desde la instrucción que sigue al manejador.
- Si la unidad no proporciona un manejador²⁶se busca por propagación

EXCEPCIONES – ADA

Propagación

Si la unidad que genera la excepción **no proporciona un manejador**, se debe buscar ese manejador dinámicamente.

- **Se termina la unidad** (bloque, paquete, subprograma o tarea) dónde se produce la excepción.
- **Si el manejador no se encuentra en ese lugar la excepción se propaga dinámicamente (quién lo llamó). Esto significa que se vuelve a levantar en otro ámbito.**
- **Siempre tener en cuenta el alcance, puede convertirse en anónima. Al propagarse a otras unidades la variable excepción declarada ya no está en el alcance y quedará sin nombre y entrará por when others**

EXCEPCIONES – ADA

Uso del Raise:

- La utilidad de **raise** es poder lanzar excepciones que pueden ser definidas por el programador.
- una excepción se puede levantar nuevamente colocando solo la palabra raise.
- Para qué? el manejador podría realizar algunas acciones de recuperación y luego utilizar raise para volver a lanzar la excepción y permitir que se propague más arriba en la jerarquía de manejo de excepciones.

EXCEPCIONES – ADA

```
with Ada.Text_IO;
```

```
procedure Excepciones_1 is
```

```
  I: Integer;
```

```
begin
```

```
  I := 0;
```

```
  I := 4 / I; -- elevará una excepción Constraint_Error
```

```
  Ada.Text_IO.Put_Line ("Resultado: " & Integer'Image (I));
```

```
end Excepciones_1;
```

excepción built-in (predefinidas)

Resultado: raised CONSTRAINT_ERROR : excepciones.adb:7 divide by zero

```
with Ada.Text_IO;
```

```
procedure Excepciones_2 is
```

```
  I: Integer;
```

```
begin
```

```
  I := 0;
```

```
  I := 4 / I; -- elevará una excepción Constraint_Error
```

```
  Ada.Text_IO.Put_Line ("Resultado: " & Integer'Image (I));
```

```
exception
```

```
  when Constraint_Error =>
```

```
    Ada.Text_IO.Put_Line ("Intento de dividir por 0");
```

```
end Excepciones_2;
```

excepción del programador

Resultado: Intento de dividir por 0

EXCEPCIONES – ADA

procedure A is

x,y,a : integer;

e1, e:exception;

procedure B (x : integer; y: integer) is

m : integer;

e:exception;

begin

....

Exception

when e (*) x:=x+1: raise;

when e1 (*) null;

end B;

begin

.....

Exception

when e (*) x:=x+1;

when others (*) x:=x+2;

end A;

Si se levanta la
excepción e ...

Si levanta la
excepcion e1

Es manejada primero por
el manejador e de B y
luego por el manejador
others de A

Por el
raise
busca
la
excepci
on sin
nombre
afuera

Pueden programarse
Manejadores que no realicen
nada. Usando la sentencia
“null”. Ejemplo..

EXCEPCIONES – ADA – EJEMPLO

Prog MAIN

e: exception

*****DECLARACION**

...

Begin

...

.. if (condError) then

raise e ***** INVOCACION**

**** si fuera sólo (raise) levanta una excepción anónima afuera**

endif

....

Exception

when e : begin ...

...

... (raise)

******MANEJADORES**

***** este raise levanta la misma excepción que está manejando, pero como es por terminación Saldría por propagación y terminaría dando error**

End

When others Begin

...

..

End

..... •

End //(de main)

EXCEPCIONES – ADA

Tener en cuenta :

- La **asociación** de la **excepción** con el **manejador** es **determinístico** (*variable* ↔ *manejador de esa variable*). **Asigno mismo nombre a ambos**
- Si se deseara **continuar ejecutando las instrucciones de un bloque** donde se lanza una excepción, es preciso **“crear un bloque más interno”**. Se usa **Declare** para **amar una unidad**, como se muestra en el ejemplo y luego **agregar instrucciones restantes abajo**. (simular reasunción)

Procedure Bloque () is

```
....  
begin
```

```
...
```

```
Declare
```

```
.....
```

```
.....
```

```
begin
```

```
    exception
```

```
        Manejadores
```

```
        .....
```

```
    end; -- del bloque interno
```

```
....
```

En esta sección se deben colocar Instrucciones que es preciso ejecutar, aunque se haya levantado la excepción en el bloque interno

```
.....  
end;
```

instrucción declare se utiliza para declarar variables locales, instrucciones, excepciones dentro de un bloque, y armar una unidad interna

bloque interno para manejar las instrucciones que pueden fallar

EXCEPCIONES – C++

- Utiliza el criterio de **Terminación**.
- Cuenta con **excepciones** estandar predefinidas.
- **Try** para **indicar los bloques donde pueden llegar a levantarse excepciones**. Los mismos van precedidos por esta palabra clave **Try**
- **Catch** se utilizada para especificar los manejadores.
Catch(NombreDeLaExcepción)
 - las cláusulas **catch** deben estar después del **bloque try** y antes de cualquier código que esté fuera del **bloque try**.
 - Los **manejadores** van asociados a bloques { }
- **Throw** Se utiliza para **lanzar explícitamente una excepción**

EXCEPCIONES – C++

```
// ...código previo...
```

```
try
```

```
{
```

```
    // bloque de código a comprobar
```

```
}
```

Sentencias que pueden provocar una excepción.

Throw

```
catch( tipo ) // Ha ocurrido un suceso en el try que se ha terminado  
//la ejecución del bloque y catch recoge y analiza lo sucedido
```

```
{
```

```
    // bloque de código que analiza lo que ha lanzado el try
```

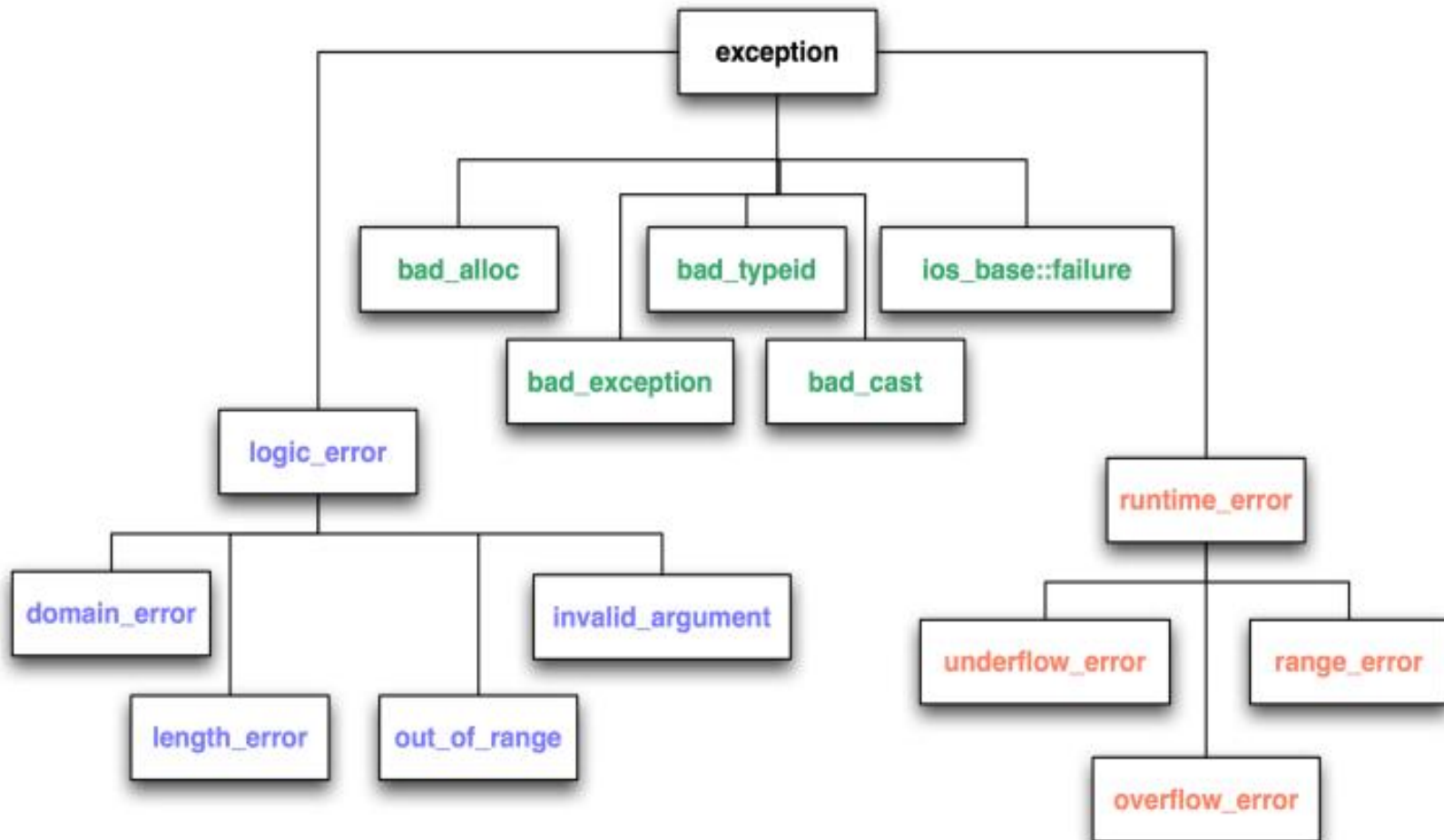
```
}
```

```
// ...código posterior...
```

Sentencias del Manejador

EXCEPCIONES – C++

Excepciones estándar. Predefinidas



EXCEPCIONES – C++

```
// Demostración de los comandos try, throw y catch
#include <iostream>
```

```
// Función: main
```

```
// Recibe: void
```

```
// Devuelve: int
```

```
// En la función principal se tratarán los comandos try, throw y catch
```

```
int main(void)
```

```
{
```

```
    try // Se intenta hacer el siguiente código
```

```
    {
```

```
        // Aquí puede ir más código...
```

```
        throw 125; //...aunque directamente en este caso se lanza una excepción.
```

```
    }
```

```
    catch(int) // Se captura con un catch de enteros (podría usarse long o char, por ejemplo)
```

```
    {
```

```
        std::cout << "Ha surgido una excepción de tipo entero" << std::endl; // y se muestra por pantalla
```

```
    }
```

```
    std::cin.get(); // Y el programa finaliza.
```

```
    return 0;
```

```
}
```

**Excepción a levantar
de tipo entero**

EXCEPCIONES – C++

Funcionamiento:

- El **bloque try** que contiene código que **puede lanzar una excepción**.
- Si se **lanza** una excepción el **control se transfiere** inmediatamente a la cláusula **catch**.
- Si la **excepción coincide** con el **tipo especificado** en la cláusula **catch**, se **ejecuta el bloque de código** de esa cláusula catch.
- Si la **excepción se maneja exitosamente**, la ejecución **continúa después del bloque try-catch**.
- Si **no se encuentra una catch correspondiente** o **no se maneja la excepción**, la excepción puede **propagarse hacia bloques try-catch externos**.
- Sinó puede resultar en una **finalización abrupta del programa**.

EXCEPCIONES – C++

```
void f() {  
    try { g(); } catch(Excepcion &ex) {  
        cerr << ex.queHaPasado();  
    }  
    // sigue...  
}
```

f() captura el tipo de
excepciones que puede
lanzar h()

Si **no se encuentra** una
cláusula **catch correspondiente**
la excepción se **propaga**
hacia bloques
try-catch externos.

```
void g() {  
    h();  
    // sigue...  
}
```

g() no la captura...

```
void h() {  
    if (algo_fallar)  
        throw Excepcion("¡Mecachis!");  
    // sigue...  
}
```

h() puede lanzar
una excepción...

EXCEPCIONES – C++

Permite pasar **parámetros** al levantar la excepción.

throw MiExcepcion(parametro1, , parametroN);

Ejemplo: *levantar excepción y pasar un parámetro mensaje usando una excepción estándar.*

```
#include <iostream>
#include <stdexcept>

int main() {
    try {
        throw std::runtime_error("Se produjo un error.");
    }
    catch (const std::exception& excepcion) {
        std::cout << "Excepción capturada: " << excepcion.what() << std::endl;
    }
    return 0;
}
```

Salida: Excepción capturada: Se produjo un error

EXCEPCIONES – C++

- **permite que el programador especifique de manera precisa la intención de una rutina, al especificar tanto el comportamiento normal esperado (*los datos que puede aceptar y devolver*) como sus comportamientos anormales pasándole los parámetros al throw.**

Ejemplo

void rutina () throw (Ayuda, Zerodivide);

rutina se declara como una **función void** que no devuelve ningún valor y que puede **lanzar 2 tipos de excepciones: *Ayuda y Zerodivide*.**

EXCEPCIONES – C++

void rutina () throw (Ayuda, Zerodivide);

Qué sucede en este caso si la rutina....?

- Si lanzó otra excepción que no está contemplada en el listado de la Interface

En este caso no se propaga la excepción y una función especial **unexpected()** se ejecuta automáticamente, que eventualmente causa **abort()** que provoca el final del programa.

Unexpected() puede ser redefinida por el programador.

EXCEPCIONES – C++

Qué sucede si la rutina

- Si colocó en su interface el listado de posibles excepciones a alcanzar
En este caso Si se propaga la excepción.

Si una excepción se propaga repetidamente y nunca se encuentra un manejador coincidente, llama automáticamente a una función especial llamada **terminate()**.

terminate() puede ser redefinido por el programador. Su comportamiento predeterminado, eventualmente aborta la ejecución del programa

EXCEPCIONES – C++

Qué sucede si la rutina

- Si no se proporciona ninguna lista **throw**
Significa que CUALQUIER excepción puede ser propagada.
- Si se colocó en su interface una lista vacía **throw()**
Significa que **NINGUNA** excepción será propagada.

EXCEPCIONES – CLU

- Utiliza el criterio de **Terminación**.
- Solamente pueden **ser lanzadas** por los **procedimientos**.
si una instrucción genera una excepción, el procedimiento que contiene la instrucción retorna anormalmente al generar la excepción. Un procedimiento no puede manejar una excepción generada por su ejecución, quien llama al procedimiento debe encargarse de manejarla.
- Las excepciones que un procedimiento puede lanzar se declaran en su encabezado.

EXCEPCIONES – CLU

- Se lanzan explícitamente con la palabra clave **signal**
- Los **manejadores** se **colocan** al lado de una **sentencia** simple o compleja y llevan la palabra clave **when** . Forma de definirlos:

<instrucción> except <lista_de_controladores> end

donde <instrucción> puede ser cualquier instrucción (compuesta) del lenguaje. Si la ejecución de una invocación de procedimiento dentro de <instrucción> genera una excepción, el control se transfiere a <lista_de_controladores>.

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

EXCEPCIONES – CLU

- Posee excepciones **predefinidas** con su manejador asociado. Por ejemplo, **failure**
- Se pueden **pasar parámetros** a los manejadores.
- Una excepción se puede **volver a levantar una sola vez** utilizando **resignal**
- Una excepción **se puede levantar en cualquier lugar del código**

```
Procedure ejemplo () signals e1, e2
Begin
.....
if ( .. ) then A(); except
                    when e1: Manejador1;
                    when e2 : Manejador2;
                    when others : Manejador3;
end;

....
End;
```

EXCEPCIONES – CLU

Si no encuentra el manejador es por Propagación al producirse una excepción:

- Se termina el procedimiento donde se levantó la excepción y devuelve el control al llamante inmediato donde **se debe encontrar el** manejador.
- Si el manejador **se encuentra en ese ámbito, se ejecuta y luego se** pasa el control a la sentencia siguiente a la que está ligado dicho manejador.
- Si el manejador **no se encuentra en ese lugar la excepción se propaga estáticamente** en las sentencias asociadas. Esto significa que el proceso se repite para las sentencias incluidas estáticamente.
- En caso de no encontrar ningún manejador en el procedimiento que hizo la llamada se levanta una excepción **failure** y devuelve el control, terminando todo el programa

EXCEPCIONES – CLU

Procedure Main

.....

Procedure UNO() signals error1;

x:integer

Begin

x:=2;

While y < x Do

If y=0 Then signal error1;

end if;

exception when error1: <sent>-;

resignal;

end; // manejador4

Dos();

Wend; exception

when error1: <sent> .End;

End; //UNO

UNO
termina

Procedure Dos() signals error1;

m:integer;

Begin

...

if m=0 then signal error1;

End;

Begin //MAIN

x:=1; y:=0;

Uno();

exception when error1: x:=x+1; y:=y+1;

end;

...

Dos();

exception when error1: resignal; end;

... End; //MAIN

Ejecuta
Este
Manejador

Excepciones – CLU – Ejemplo

Procedure Main

Error1 : exception;

x, y: integer;

Procedure UNO() signals error1; //en el encabezado se deben definir los manejadores que puede levantar

x:integer

Begin

x:=2;

While y < x Do

If y=0 Then signal error1; //se levanta la excepcion error

end if; exception when error1 -> y:=y+7; x:=x+2; resignal; end; // manejador4

Dos();

y:=y+1;

Wend; exception when error1 -> y:=x+3; x:=x+3; Resignal; .End; //manejador3

End; //UNO

Procedure Dos() signals error1;

m:integer;

Begin

...

if m=0 then signal error1; //se levanta la excepcion error

End;

Begin //MAIN

x:=1; y:=0;

Uno(); exception when error1 -> x:=x+1; y:=y+1; end; // manejador1

...

Dos(): exception when error1 → resignal; end; // manejador2

...

End; //MAIN

EXCEPCIONES - JAVA

- Al igual que C++ las excepciones son objetos que pueden ser alcanzados y manejados por manejadores adicionados al **bloque** donde se produjo la excepción.
- Cada excepción está representada por una instancia de la clase **Throwable** o de una de sus subclases (Error y Exception)
- La gestión de excepciones se lleva a cabo mediante cinco palabras clave: **try, catch, throw, throws, finally**.
- Se debe especificar mediante la cláusula **throws** cualquier excepción que se envía desde un método.
- Se debe poner cualquier código que el programador desee que se ejecute siempre, en el método **finally**.⁵⁰



FASES DEL TRATAMIENTO DE EXCEPCIONES

- **Detectar e informar del error:**
 - Lanzamiento de Excepciones → throw
 - Un método detecta una condición anormal que le impide continuar con su ejecución y finaliza “lanzando” un objeto Excepción.
- **Recoger el error y tratarlo:**
 - Captura de Excepciones → bloque try-catch
 - Un método recibe un objeto Excepción que le indica que otro método no ha terminado correctamente su ejecución y decide actuar en función del tipo de error.



EXCEPCIONES - JAVA

Orden correcto

```
try{
  ...
}
Catch (ArithmeticException e)
    { trataArithmeticException(e);}
catch (Exception e2)
    { trataTodasLasExcepciones(e2); }
Finally
    { bloque final; }
```

Puede estar o no
Si está, la ejecución de su código se realiza cuando se termina la ejecución del bloque Try, se haya o no levantado una excepción.

Propagación,
bloques Try
anidados

// demuestra cómo lanzar una excepción cuando ocurre una división entre cero

```
public int cociente( int numerador, int denominador )
    throws ArithmeticException
{
    return numerador / denominador;
}
```

52

Veamos un ejemplo en [compilador online \(java\)](#)

EXCEPCIONES - PHYTON

- Se manejan a través de bloques try except

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

La declaración try funciona de la siguiente manera:

- Primero, se ejecuta el *bloque try* (el código entre las declaración try y except).
- Si no ocurre ninguna excepción, el *bloque except* se saltea y termina la ejecución de la declaración try.
- Si ocurre una excepción durante la ejecución del *bloque try*, el resto del bloque se saltea. Luego, si su tipo coincide con la excepción nombrada luego de la palabra reservada except, se ejecuta el *bloque except*, y la ejecución continúa luego de la declaración try.
- Si ocurre una excepción que no coincide con la excepción nombrada en el except, esta se pasa a declaraciones try de más afuera; si no se encuentra nada que la maneje, es una *excepción no manejada*, y la ejecución se frena con un mensaje de error.

EXCEPCIONES - PHYTON

- ◆ Presenta la siguiente estructura para manejo de excepciones:
try:

sentencia 1

....

sentencia n

← **except** nombre de la excep1 **as** var:
sentencias

..

← **except** nombre de la excención n :
sentencias

else:
sentencias

finally:
sentencias

Un conjunto de excepciones pueden ser manejadas por un mismo manejador. En ese caso se puede colocar:

except (exp1,exp2,...):

Puede aparecer un except **SIN nombre** de excepción, pero **SOLO** al final. Actúa como comodín

except:

Opcional

El código colocado en la cláusula **else** se ejecuta **solo si** no se levante una excepción

El código colocado en la cláusula **finally** se ejecuta **siempre**

EXCEPCIONES - PYTHON

¿Qué sucede cuando una excepción no encuentra un manejador en su bloque “try except”?

- **Busca estáticamente**

Analiza si ese try está contenido dentro de otro y si ese otro tiene un manejador para esa excepción. Sino...

- **Busca dinámicamente**

Analiza quién lo llamó y busca allí

- **Si no se encuentra un manejador, se corta el proceso y larga el mensaje standard de error**

- **Levanta excepciones explícitamente con “raise”**

EXCEPCIONES - PHYTON

• Ej. con else:

```
PruebaConElse.py - E:\Python\Clases\Pruebas\Excepciones\PruebaConElse.py
File Edit Format Run Options Windows Help
dic={1:'juan',4:'pedro',5:'helena'}
y=8
print y
try:
    for x in range(1,6):
        print dic[x]
except (KeyError):
    dic[x]='nuevo'
else:
    print 'Se ejecutó el bloque try except en forma correcta'
print dic
Ln: 12 Col: 0
```

Se levanta la excepción..

Cláusula **opcional**, que se ejecuta **SOLO** si **NO** se levanta excepción en el bloque try except

```
>>>
8
juan
(1: 'juan', 2: 'nuevo', 4: 'pedro', 5: 'helena')
```

→ **NO se ejecuta** el bloque **else**

```
PruebaConElseOtro.py - E:\Python\Clases\Pruebas\Excepciones\PruebaConElseOtro.py
File Edit Format Run Options Windows Help
dic={1:'juan',4:'pedro',5:'helena'}
y=0
print y
try:
    for x in (1,4,5):
        print dic[x]
except (KeyError):
    dic[x]='nuevo'
else:
    print 'Se ejecutó el bloque try except en forma correcta'
print dic
Ln: 5 Col: 19
```

NO se levanta la excepción..

```
>>>
8
juan
pedro
helena
Se ejecutó el bloque try except en forma correcta
{1: 'juan', 4: 'pedro', 5: 'helena'}
```

→ **SI se ejecuta** el bloque **else**

EXCEPCIONES EN PHP

- Modelo de Terminación
- Una excepción puede ser lanzada (thrown), y atrapada ("caught")
- El código esta 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

EXCEPCIONES EN PHP

- El objeto lanzado debe ser una instancia de la clase `Exception` o de una subclase de `Exception`. Intentar lanzar un objeto que no lo es resultará en un Error Fatal de PHP.
- Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque `catch` coincidente. Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "Uncaught Exception ..." ("Excepción No Capturada"), a menos que se haya definido un gestor con `set_exception_handler()`.

EXCEPCIONES EN 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";
}

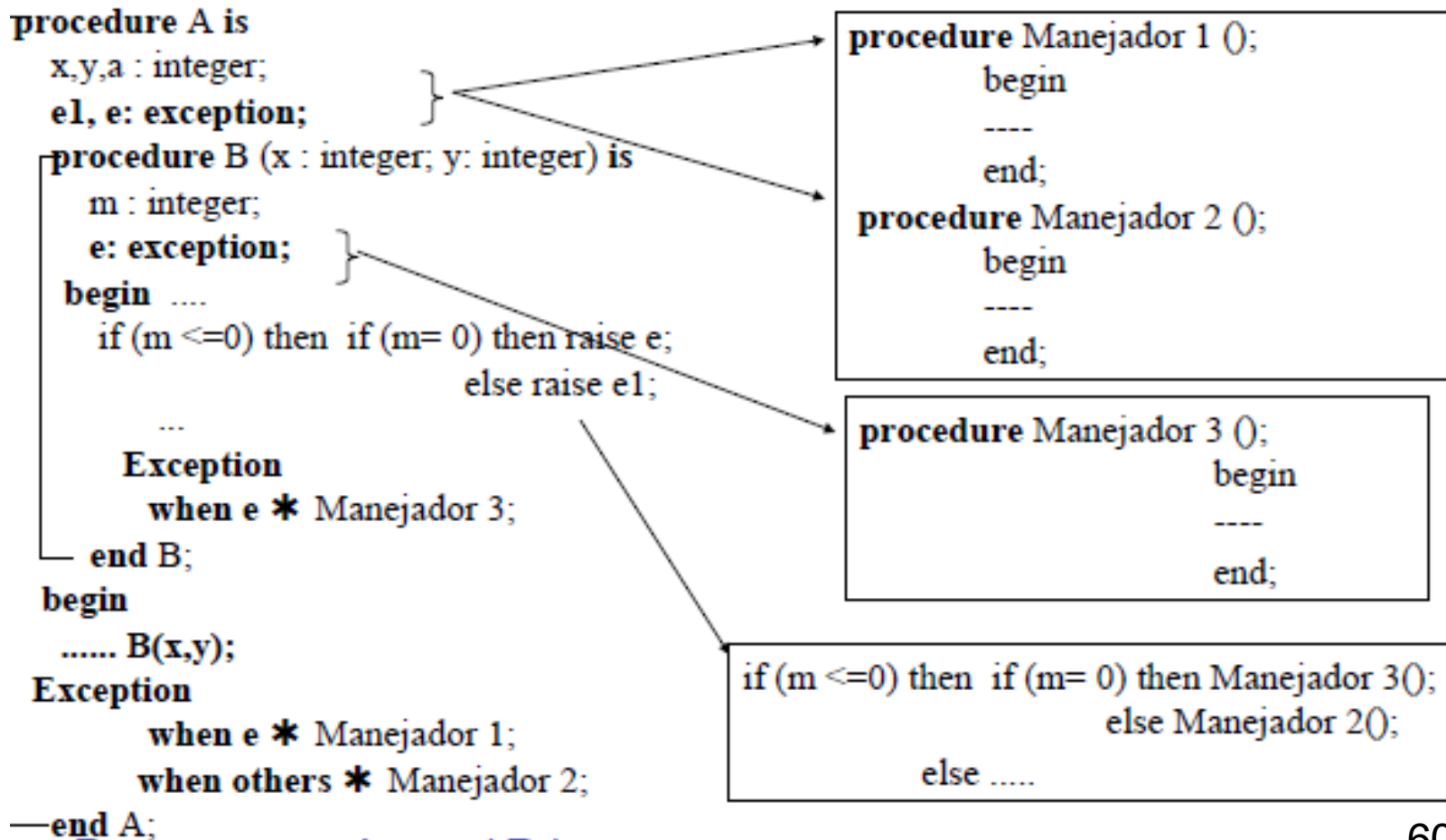
try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ', $e->getMessage(), "\n";
} finally {
    echo "Segundo finally.\n";
}

// Continuar ejecución
echo 'Hola Mundo\n';
?>
```

El resultado del ejemplo sería:

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

LENGUAJES QUE NO PROVEEN MANEJO DE EXCEPCIONES - SIMULAR!!



DUDAS?

- Cual modelo les parece mas seguro?
- Les parece útil el manejo de excepciones? Lo creen necesario?
- El modelo de terminación tiene diferentes comportamientos

