Práctica 8

1. ¿Explique claramente a qué se denomina excepción?  
   Una excepción es una situación anómala que se da en la ejecución de un programa y que se supone que ocurre con poca frecuencia.
2. ¿Qué debería proveer un lenguaje para el manejo de las excepciones? ¿Todos los lenguajes lo proveen?
   1. Un modo de definirlas
   2. Una forma de alcanzarlas, invocarlas
   3. Una forma de manejarlas
   4. Un criterio de continuación

No todos los lenguajes de programación proveen manejo de excepciones.

1. ¿Qué ocurre cuando un lenguaje no provee manejo de excepciones? ¿Se podría simular? Explique cómo lo haría  
   Cuando un lenguaje no maneja excepciones, o bien deben contemplarse los casos excepcionales dentro del programa (contemplar explícitamente que el divisor en una división sea distinto de cero, por ejemplo); o bien se aborta el programa que produce el error. Se podría simular haciendo todos los controles para manejar todos lo posibles casos excepciones que pudieran ocurrir.
2. 5) Cuando se termina de manejar la excepción, la acción que se toma luego es importante. Indique 01. ¿Qué modelos diferentes existen en este aspecto?  
   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?  
     
   Continuación:  
   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. • La excepciones son llamadas CONDITIONS • Los manejadores se declaran con la sentencia ON: ON CONDITION(Nombre-excepción) Manejador • El manejador puede ser una instrucción o un bloque • Las excepciones se alcanzan explícitamente con la palabra clave Signal condition (Nombre-excepción)  
     
   • PL/1  
   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.  
   • El alcance de un manejador termina cuando finaliza la ejecución de la unidad donde fue declarado  
     
   Terminación:  
   Criterio de Terminación. Cada vez que se produce una excepción, se termina el bloque dónde se levantó y se ejecuta el manejador asociado.  
     
   • ADA:

• Los manejadores pueden encontrarse en el final de cuatro diferentes unidades de programa: Bloque, Procedimiento, Paquete o Tarea.

• Se definen en la zona de definición de las variables y tienen el mismo alcance  
• Se alcanzan explícitamente con la palabra clave raise.  
•Tiene cuatro excepciones predefinidas : Constraint\_error, Program\_error, Storage\_error, Numeric\_Error, Name\_Error, Tasking\_error.  
• Propagación, al producirse una excepción:  
 ● Se termina la unidad, bloque, paquete o tarea dónde se alcanza la unidad.  
 ● Si el manejador se encuentra en ese ámbito, se ejecuta.  
 ● Si el manejador no se encuentra en ese lugar la excepción se propaga dinámicamente. Esto

significa que se vuelve a levantar en otro ámbito.  
 ● Siempre tener en cuenta el alcance, puede convertirse en anónima!  
• Una excepción se puede levantar nuevamente colocando solo la palabra raise.  
  
• CLU  
Solamente se pueden ser alcanzadas por los procedimientos.  
• Están asociadas a sentencias.  
• La excepciones que un procedimiento puede alcanzar se declaran en su encabezado.  
• Se alcanzan explícitamente con la palabra clave signal  
• Los manejadores se colocan al lado de una sentencia simple o compleja. Forma de definirlos: <sentencia> except  
when Nombre-Excepción: Manejador1;  
when Nombre-Excepción : Manejador2;  
...........

when others: Manejadorn;  
end;

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

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

• C++  
• Utiliza el criterio de Terminación.

• Las excepciones pueden alcanzarse explícitamente a través de la sentencia throw  
• Posee excepciones predefinidas  
• Los manejadores van asociados a bloques .  
• Los bloques que pueden llegar a levantar excepciones van precedidos por la palabra clave Try y al finalizar el bloque se detallan los manejadores utilizando la palabra clave Catch(NombreDeLaExcepción)  
• Al levantarse una excepción dentro del bloque Try el control se transfiere al manejador correspondiente.  
• Al finalizar la ejecución del manejador la ejecución continúa como si la unidad que provocó la excepción fue ejecutada normalmente.  
• Permite pasar parámetros al levantar la excepción. Ejemplo: Throw (Ayuda msg); Se está levantando la excepción Ayuda y se le pasa el parámetro msg.  
Ejemplo:  
…  
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 en su interface pueden listar las excepciones que ellas pueden alcanzar. void rutina ( ) throw (Ayuda, Zerodivide); ¿Qué sucede si la rutina …?

• alcanzó otra excepción que no está contemplada en el listado de la Interface? En este caso NO se propaga la excepción y una función especial se ejecuta automáticamente: unexpected(), que generalmente causa abort(), que provoca el final del programa. Unexpected puede ser redefinida por el programador.

• Colocó en su interface el listado de posibles excepciones a alcanzar En este caso Si se propaga la excepción. Si una excepción es repetidamente propagada y no machea con ningún manejador, entonces una función terminate() es ejecutada automáticamente.

• Colocó en su interface una lista vacía (throw())? Significa que NINGUNA excepción será propagada.  
  
• 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.  
  
• 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 como los mostrados arriba.  
  
• ¿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”  
  
  
• PHP  
• Modelo de Terminación

• Una excepción puede ser lanzada (thrown), y atrapada ("catched")

• 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.

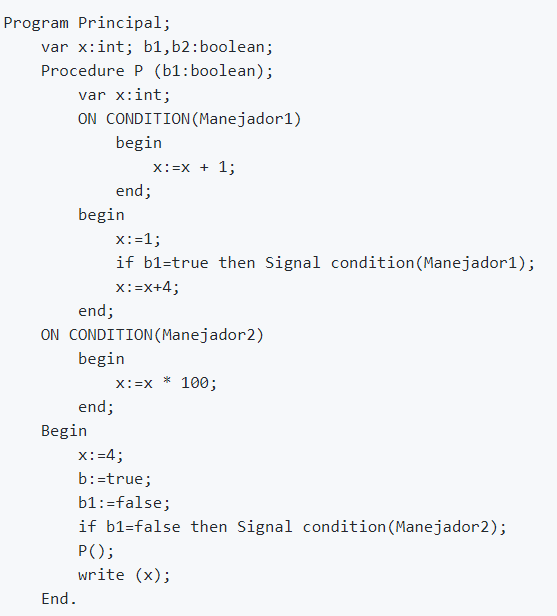
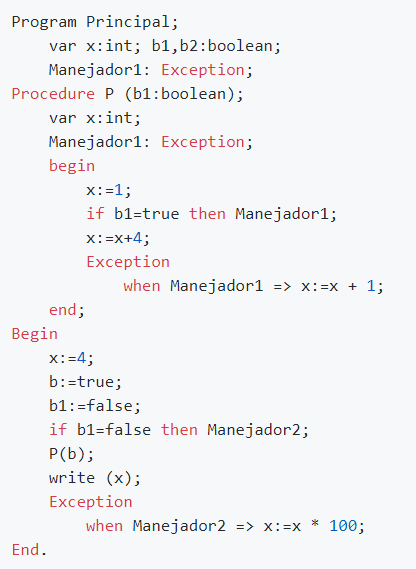
• Se puede utilizar un bloque finally después de los bloques catch  
  
• El objeto lanzado debe ser una instancia de la clase Exception o de una subclase de Exception. Intentar lanzar un objeto que no lo es resultará en un Error Fatal de PHP.  
• Cuando una excepción es lanzada, el código siguiente 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().

03. ¿Cuál de esos modelos es más inseguro y por qué?  
El modelo por reasunción es más inseguro dado que cuando surge una excepcion no termina, sigue ejecutando donde se quedó y puede que eso acarree errores en la ejecución.

6) Sea el siguiente programa escrito en Pascal, ¿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.  
  
Esta usando un manejo de excepciones por reasunción ya que una vez que se ejecuta la excepción sigue con la ejecución normalmente. Si lo que se quisiera es que maneje la excepción por terminación, entonces si Error es true debe terminar la ejecución.

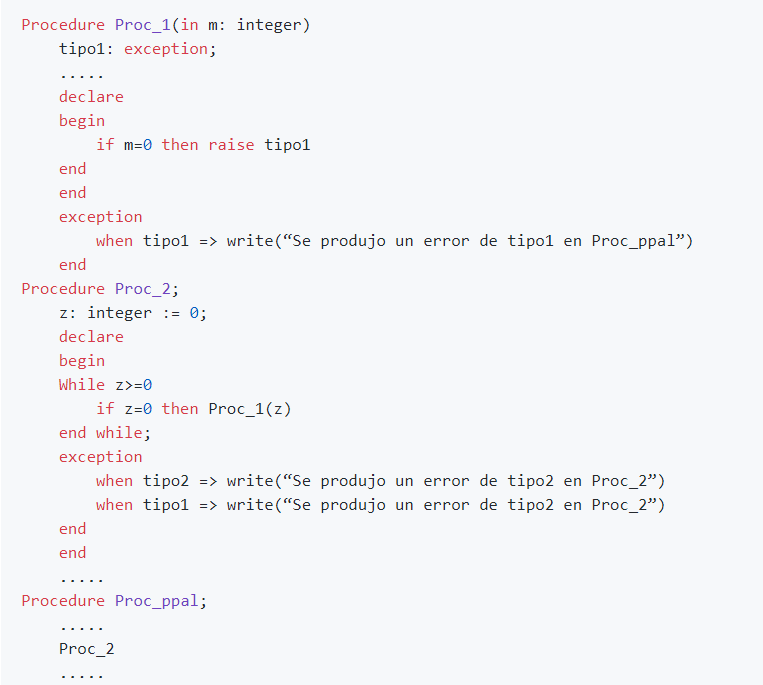
7) Sea el siguiente programa escrito en pascal:

1. 2.

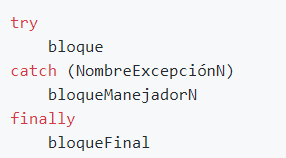
1. Implemente este ejercicio en PL/1 utilizando manejo de excepciones
2. ¿Podría implementarlo en ADA utilizando manejo de excepciones? En caso afirmativo, realícelo.

8) Sean los siguientes, procedimientos de un programa escrito en CLU:

1. Analizar el ejemplo y decir qué manejadores ejecuta y en qué valores quedan las variables. JUSTIFIQUE LA RESPUESTA.
2. ¿Podría simular un efecto parecido en ADA? En caso de poder, explique cómo.
3. En Proc\_ppal si surge una excepción en ..... el programa aborta y finaliza su ejecución. En Proc\_2, si surge una excepción de tipo1 en ....., se aborta el módulo y en la siguiente línea se maneja la excepción correspondiente; si no fuera de tipo1, finaliza el módulo y continua la ejecución en Proc\_ppal. En Proc\_1 hay declarado un signals de tipo1, lo cual hace posible que se pueda levantar una excepción con un signal tipo1 y que esta sea manejada abajo del llamado al procedimiento. De no poder manejarse en ese except se busca estáticamente en el while, si no es posible falla el programa completo.
4. Se puede usando un declare. La idea es encapsular el procesamiento dentro del declare y aislar las excepciones(en el modulo que llama la excepcion), por lo que si surge una se mata el modulo (porque no hay forma de resolverla en el declare) y se busca una forma de manejarla en el modulo que la llamo que, convenientemente, va a tener la parte de su procesamiento involucrada de llamar a otro modulo dentro de otro declare. Dentro de ese declare, van a estar las excepciones que puede atender el modulo al que se invoca. Fuera del declare va a estar el resto del procesamiento. De esta forma, cuando surge una excepcion en Proc\_1, no es manejada en el modulo en donde se produjo, termina dicho modulo y se busca una forma de resolverla en el modulo que llamo a ese modulo en donde se produjo la excepcion.

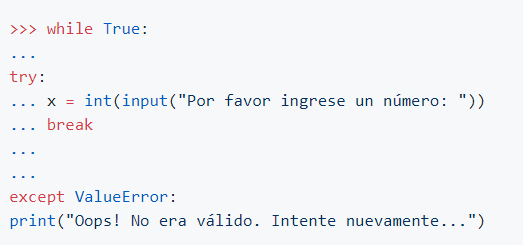


9) Indique diferencias y similitudes entre Phyton y Java con respecto al manejo de excepciones.

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. Diferencias: TODAS las excepciones que puedan ser alcanzadas explícitamente por la rutina y son chequeadas por el compilador, DEBEN ser listadas en la interface de la misma o ser manejadas por un manejador. La justificación de esta obligatoriedad es que el usuario de la rutina DEBE conocer qué excepciones puede alcanzar la misma. Uso de FINALLY

donde finally 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. Salvo que el bloque Try haya levantado una excepción que no macheo con ningún manejador.

Python

Se manejan a través de bloques try except

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 como los mostrados arriba.

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

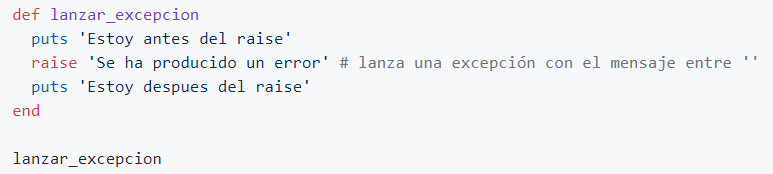
Busca estáticamente: Analiza si ese try está contenido dentro de otro y si ese otro tiene un manejador para esa excepción. Sino...

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”

10) Qué modelo de excepciones implementa Ruby?. Qué instrucciones específicas provee el

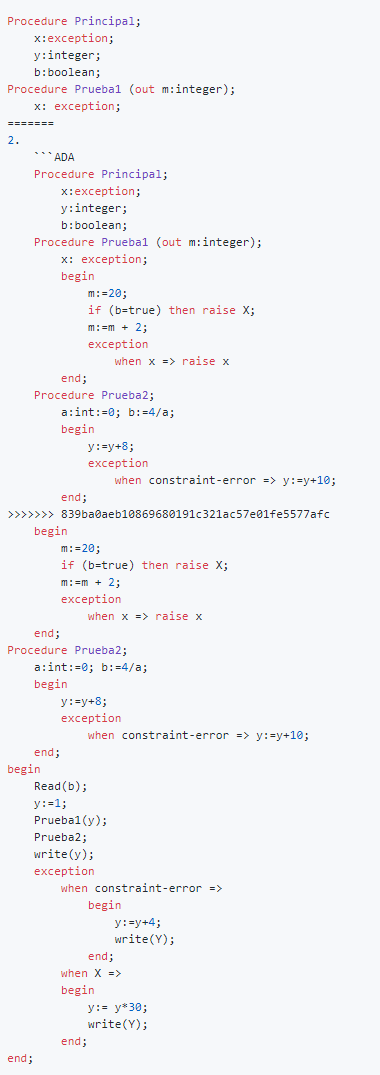
lenguaje para manejo de excepciones y cómo se comportan cada una de ellas?  
  
Una excepción es un tipo especial de objeto de la clase Exception. Para tratar el problema hace falta raise; de no ser así, el programa termina y avisa del error. Lo que hará raise (lanzar), será lanzar una "excepción" para manejar el error.  
Ruby tiene una serie de clases, Exception y sus hijas, que ayudan a manejar los errores que pueden ocurrir. La siguiente figura enseña la jerarquía en Ruby:  
  


11) Indique el mecanismo de excepciones de javascript.

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.

12) Sea el siguiente programa escrito en ADA:

1. Indique el camino de ejecución.
2. Agregar el uso de una excepción anónima
3. El Read(b) indica que el usuario determina el valor de verdad de dicha variable, por lo tanto tenemos dos caminos de ejecucion a evaluar. En principio, el camino de ejecucion no difiere en nada ya que los valores de las variables son iguales en los dos casos. Se envia y a Prueba1, pero como el parametro que recibe dicha funcion es de tipo OUT, esto no tiene mucho sentido. En dicha funcion, se le asigna a m(el parametro y que pasamos desde el main) 20 y aca es cuando la cosa puede variar. Si b es true, entonces va a entrar en el if y va a levantar la excepcion X. Ante este escenario se va a intentar manejar la excepcion localmente pero va a ser imposible dado que no hay un manejador definido para dicho error. Ada utiliza la propagacion dinamica, por lo que ese modulo va a finalizar y se buscara una forma de resolver la excepcion en el main. En este caso, es posible resolverla, por lo que se le asigna 500 a y y se imprime dicho valor y se continua la ejecucion en Main. Luego de Prueba 1 se ejecuta Prueba2 y ocurre un contraint\_error dado que se intenta asigna 4/0 a b. En este caso, se procede a abortar el modulo y a intentar manejar la excepcion en el Main. En este caso, la excepcion es manejada exitosamente y se le asigna 504 a y y se iprime dicho valor. Luego se

continua la ejecucion en Main y se imprime el valor de y y termina el programa. Por el contrario, si b es false, entonces no entra al if y se le suma 2 a m y finaliza el modulo, dejando a y con 22. Luego se ejecuta Prueba2 y ocurre lo mismo que se b fuera true, solo que, en este caso, cuando se resuelva la excepcion y queda con 26 y se imprime dicho valor. Cuando se retoma la ejecucion en Main, imprime 26.

b.

En este caso, cuando la excepcion x es levantada en prueba 1 se resuelve en ese mismo modulo y se relevanta y finaliza el modulo Prueba1. En ese momento la excepcion ya es anonima dado que el modulo que contenia la declaracion de la excepcion x ya no existe y no es compatible con otro modulo que tenga declarada esa excepcion ya que, a pesar de llamarse igual, son distintas entre si por lo que cuando intenta resolverse en Main no puede, ya que hay otra excepcion x declarada alli.

13) Sea el siguiente código escrito en CLU

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.

b) La ejecución del manejador para error1 modifica siempre la variable x de UNO? En caso negativo indique cómo haría para lograrlo. Justifique la respuesta.

1. Primero va a ejecutarse UNO(). Despues entra en el while y salta la excepcion. Termina uno y se sigue en la proxima linea a la llamada de uno, la cual es ...... Si en esa linea surge un problema, el programa termina ya que es la subrutina mas externa y no hay nadie que la llamo. Si no hay errores, luego se ejecuta Dos(). En dos, si se produce una excepcion de tipo error1, es manejada por el proceso main, pero el resignal no se puede llevar a cabo y termina ya que no hay nadie que llamo al main. Si no surgen excepciones durante los ...., si m es 0 se levanta una excepcion de tipo error1 y es manejada por el main, pero el resignal no se puede llevar a cabo y termina ya que no hay nadie que llamo al main. Si no, termina dos y termina el programa.
2. No, modifica siempre la del main. Para lograrlo la idea es pasar la variable a traves del signal y modificar esa variable recibida, de esa manera se modificaría la variable de UNO.

13) Dado el siguiente código en Ada. Marque con una cruz sólo los caminos de ejecución correctos en los casos en que se produzca o levante una excepción.

