Manejo de Excepciones en Delphi

Excepciones

- Una **excepción** es una condición excepcional que se produce en una aplicación en tiempo de ejecución. Debe ser atendida para evitar que la aplicación aborte.
- **Ejemplos**
  - División por cero
  - Memoria insuficiente
  - Acceso a un puntero nulo
  - Error de conversión de tipos
- Se denomina **manejador** al conjunto de sentencias para el tratamiento de una excepción particular.

Palabras Clave

- El mecanismo de manejo de excepciones se basa en cuatro palabras clave
  - **try**: Delimita el bloque de sentencias a proteger.
  - **except**: Contiene sentencias para manejar las excepciones producidas.
  - **finally:** Bloque de sentencias que deben ejecutarse se produzcan o no excepciones.
  - **raise**: Sentencia utilizada para lanzar explícitamente una excepción.

El bloque try/except

- Delphi provee una construcción sencilla para proteger código con sentencias para el manejo de excepciones, el bloque ***try*/*except***.

- Cuando una sentencia dentro del bloque protegido lanza una excepción, el flujo de control pasa al código de manejo de excepciones. Al finalizar la ejecución del manejador, el bloque ***try*/*except*** termina y el control pasa a la sentencia siguiente al bloque.

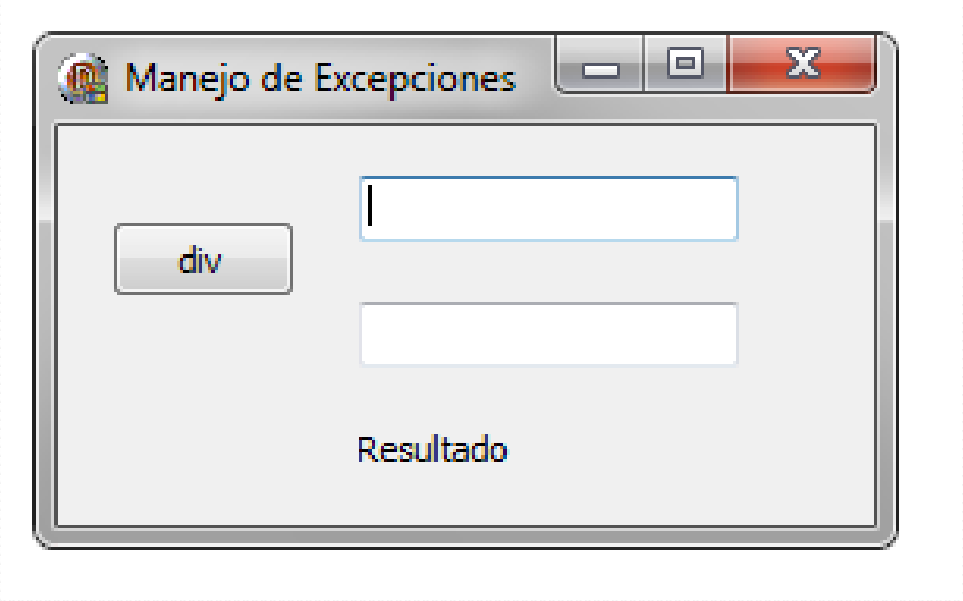
El bloque try/except

1 – La forma más simple de utilizar el bloque try/except es:

```
try
    //Sentencias
except
    //Sentencias
end;
```

El bloque try/except – Ejemplo 1

Hacer una aplicación que permita ingresar dos números enteros y calcule la división entera de los mismos utilizando la siguiente interfaz



El bloque try/except – Ejemplo 1

```
Procedure TForm1.Button1Click(Sender: TObject);
Var    A, B: Integer;
begin
    try
        A := StrToInt(Edit1.text);
        B := StrToInt(Edit2.text);
        Label1.Caption := IntToStr(A div B);
   except
      ShowMessage('Se ha encontrado un error desconocido.');
   end;
end;
```

El bloque try/except

2 – Para manejar excepciones de diferentes tipos, se puede escribir de la siguiente forma:

```
try
    // Sentencias
except
    on tipo_de_excepcion do
        // Sentencias
    on tipo_de_excepcion2 do
        // Sentencias
    on tipo_de_excepcion3 do
        // Sentencias
    else [opcional]
        // Sentencias
end;
```

El bloque try/except – Ejemplo 2

```
Procedure TForm1.Button1Click(Sender: TObject);
var A, B: Integer;
Begin
Try
    A := StrToInt(Edit1.text);
    B := StrToInt(Edit2.text);
    Label1.Caption := IntToStr(A div B);
 Except
    on EDivByZero do begin
    //Se maneja la excepción de división por cero.
       ShowMessage('No se puede dividir por cero.');
       Label1.Caption := '0';
    end;
    …
```

Ejemplo 2 (Continuación)

```
  …
on EConvertError do begin
    // Se maneja la excepción de error en la conversión.
        ShowMessage('Uno de los operandos no es un nro.válido.');
        Label1.Caption := 'Error';
    end
    else
     // Se maneja cualquier otra excepción.
        ShowMessage('Error desconocido.');
   end;
end;
```

El bloque try/except

3 –

```
try
    // Sentencias
except
    on E : tipo_de_excepcion_1 do
        // Sentencias
    ...
    on E : tipo_de_excepcion_N do
        // Sentencias
    else [opcional]
        //Sentencias
end;
```

Permite acceder a las propiedades del objeto *Exception* creado automáticamente por Delphi cuando se produce la excepción.

Propiedades

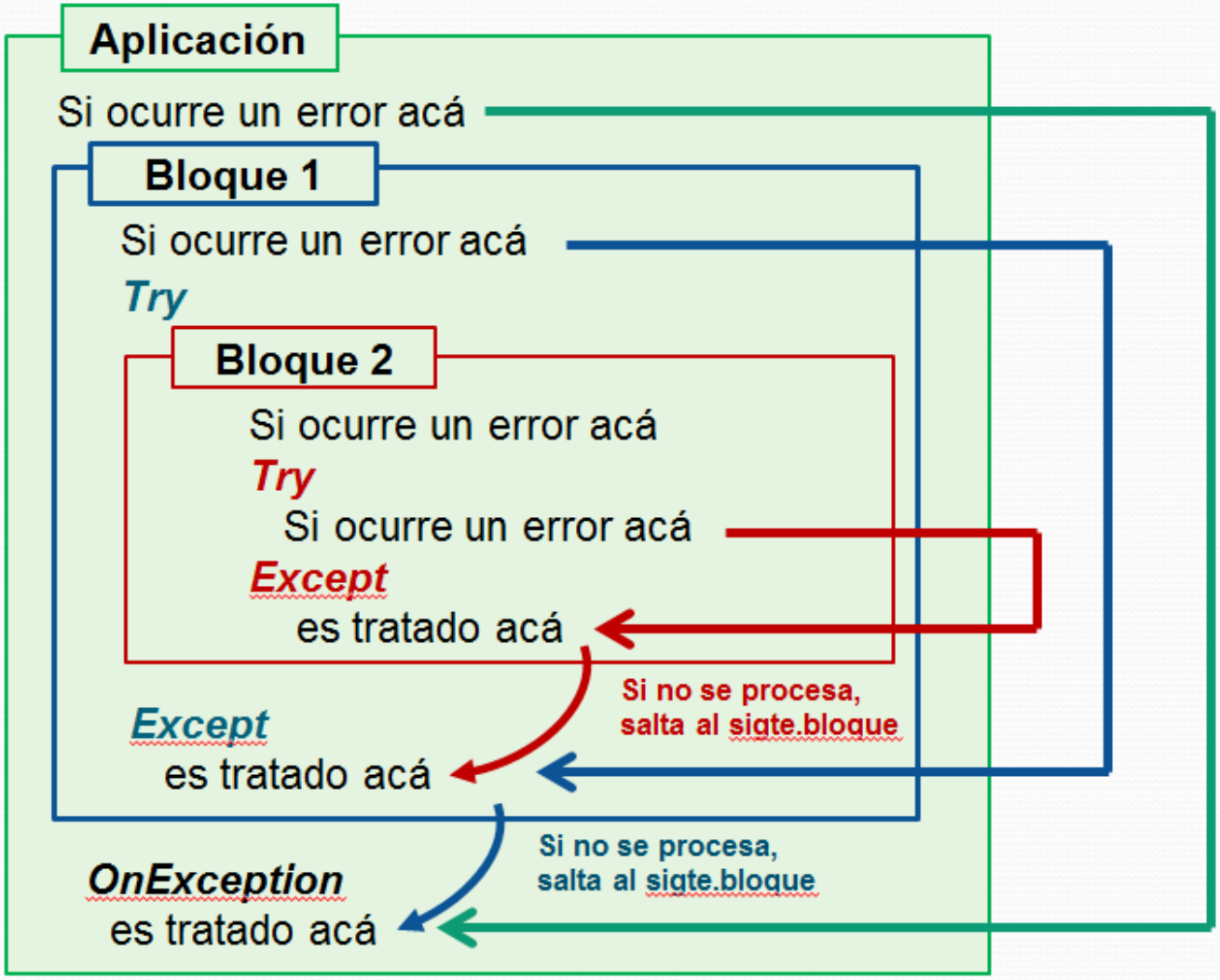
- *Message*
- *ClassName*
- *HelpContext*

```
Procedure TForm1.Button1Click(Sender: TObject);
var A, B: Integer;
Begin
Try
    A := StrToInt(Edit1.text);
    B := StrToInt(Edit2.text);
    Label1.Caption := IntToStr(A div B);
  Except
    on E : EDivByZero do begin
    // Se maneja la excepción de división por cero.
       ShowMessage('Se produjo un error de tipo: ' + E.ClassName);
       Label1.Caption := '0';
    end;
  …
```

Ejemplo 3 (Continuación)

```
...
    on E : EConvertError do begin
    // Se maneja la excepción de error en la conversión.
        ShowMessage('El mensaje de error es: ' + E.Message);
        Label1.Caption := 'Error';
    end;
    on E : Exception do
    // Se maneja cualquier otra excepción.
    ShowMessage('Error desconocido: ' + E.ClassName +
                        ', con mensaje de error: ' + E.Message);
end;
```

Flujo de Control de las Excepciones

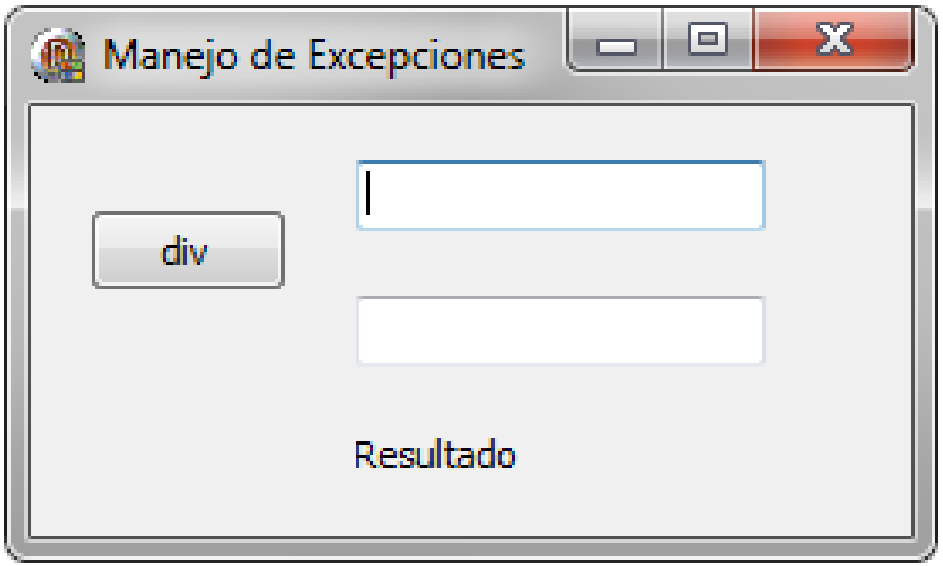


Flujo de Control de las Excepciones

- Cuando se produce una excepción en el bloque ***try***:
- 1 – Si en el bloque ***except*** hay un conjunto de sentencias sin la especificación del tipo de excepción, se ejecutan dichas sentencias.
- 2 – Si hay uno o más manejadores, se ejecuta aquél que coincida con la excepción. Una excepción coincide con un manejador cuando la excepción lanzada es la misma que la excepción declarada en el manejador o es subclase de ella.
- 3 – Si no hay manejadores pero hay cláusula ***else***, se ejecutan las sentencias de la cláusula ***else***.
- 4 – Si la excepción no es manejada la misma se propaga: se continúa la búsqueda en otro bloque ***try/except*** que contenga al bloque ***try/except*** actual. Si la rutina que lanzó la excepción no provee un manejador para ella, la rutina termina su ejecución y la excepción se propaga a la rutina invocadora, como si la hubiera lanzado ella misma.
- 5 – Si no se encuentra ningún manejador, se ejecuta el manejador por defecto de Delphi, el cual imprime un cartel de error.

Flujo de Control de las Excepciones

Analicemos el flujo de control de las excepciones volviendo al ejemplo anterior.



Flujo de Control de las Excepciones

```
procedure TForm1.Button1Click(Sender: TObject);
Var A,B : integer;
begin
  try
    try
      A := StrToInt(edit1.text);
      B := StrToInt(edit2.text);
      label1.Caption := IntToStr(A div B);
    except
      on EConvertError do begin
        A := 0;
        B := 0;
        showmessage('Dentro del manejador interno.');
        label1.Caption := IntToStr(A div B);
      end;
  ...
```

Flujo de Control de las Excepciones

```
...
      on EDivByZero do begin
        showMessage('EDivByZero en el bloque try/except interno.');
        label1.Caption := 'Error';
      end;
    end;
  except
    on EDivByZero do begin
      showMessage('EDivByZero en el bloque try/except externo.');
      label1.Caption := '0';
    end;
  end;
end;
```

Flujo de Control de las Excepciones

```
function f1(n:string):real;
var res:real;
begin
  try
     res:= 1/strToFloat(n);
  except
  on EConvertError do
     res:=1;
  end;
  f1:= res;
end;
```

```
procedure proc1;
var   res:real;
begin
  try
     res:=f1('0');
  except
     on EZeroDivide do
          res:=0;
  end;
  ShowMessage(FloatToStr(res));
end;
```

Cuando la excepción es manejada, el flujo de control pasa a la sentencia que sigue después del bloque try/except que la haya manejado. El control nunca vuelve a la sentencia que originó la excepción.

El bloque try/finally

- Bajo circunstancias normales, el programador puede asegurar que una aplicación libera los recursos alocados.
- Como una excepción pasa el control a un manejador fuera del bloque protegido donde ocurre el error, es necesario asegurarse de que la aplicación libere los recursos en todos los casos, sin importar si el código lanza o no una excepción.

El bloque try/finally - Sintaxis

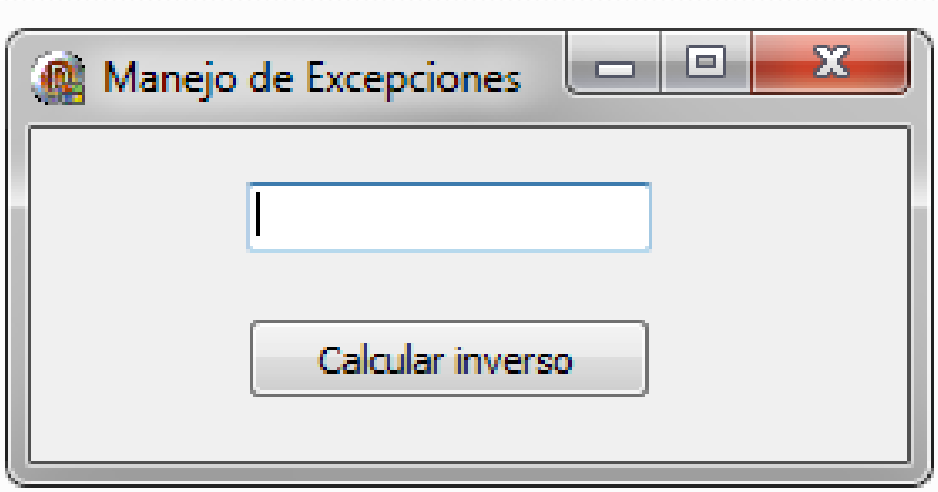
```
Try
    //Sentencias
Finally
    //Sentencias
End;
```

El bloque try/finally

- En lugar de ejecutarse cuando ocurre una excepción, el bloque ***finally*** es ejecutado siempre después del bloque ***try***, aun en los casos en los que el bloque ***try*** no termina su ejecución debido a una excepción.
- En el bloque ***finally*** se coloca el código para liberar recursos, o cualquier otra actividad que deba realizarse en todos los casos, aún en presencia de una excepción.
- El bloque ***finally*** no captura la excepción, la cual se propagará.

El bloque try/finally – Ejemplo 4

Hacer una aplicación que permita ingresar un número real y calcular su inverso.



El bloque try/finally – Ejemplo 4

```
procedure TForm1.Button1Click(Sender: TObject);
var  numero: real;
Begin
  numero := 0;
  try
    numero := 1 / (strToFloat(Edit1.text));
    ShowMessage( 'inverso= ' + FloatToStr(numero) );
  finally
    if numero = 0 then  ShowMessage( 'La variable "numero"  tiene' +
                                     '  el valor por defecto: 0.' )
                   else   ShowMessage( 'numero=' + Edit1.text );
  end;    //del bloque try-finally
end;
```

El bloque try/finally – Ejemplo 5

- Los bloques try/except y try/finally se pueden anidar de forma arbitraria:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  numero: real;
begin
  numero := 0;
  try
    try
          numero := 1 / (strToFloat(Edit1.text));
        ShowMessage('inverso= ' + FloatToStr(numero));
    finally
   …
```

Ejemplo 5 (Continuación)

```
...
if numero = 0 then begin
      ShowMessage( 'La variable "numero" tiene el valor por
                              defecto: 0.' );
    end
     else
      ShowMessage('numero=' + FloatToStr(numero));
end;   //del bloque finally
except
 on E: EZeroDivide do
   ShowMessage('No se puede dividir por cero.');
end;
end;
```

La sentencia *raise*

- Las excepciones que surgen de errores en tiempo de ejecución son transformadas automáticamente por Delphi en objetos *Exception*, pero el programador también puede lanzar excepciones explícitamente mediante la sentencia ***raise***.

La sentencia *raise* - Sintaxis

- La sentencia ***raise*** puede usarse sin parámetros dentro de un bloque ***except***:

```
Try
        // Sentencias
Except
        // Sentencias
         Raise;
End;
```

Usada de ésta forma, la sentencia ***raise*** vuelve a lanzar la excepción que haya ocurrido en el bloque ***try.***

Excepciones definidas por el usuario

- **Sintaxis**

```
Type
    MiExcepcion = class(Exception);
```

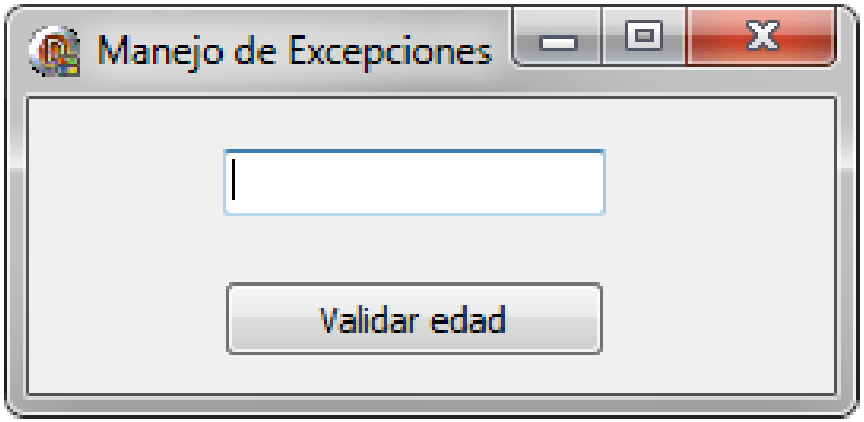
Nombre de la excepción definida por el usuario

- Para provocar la excepción utilice por ejemplo

raise MiExcepcion.Create('Error');

Creación de excepciones - Ejemplo 6

- Desarrolle una aplicación que permita ingresar un número entero correspondiente a la edad de una persona y determine si está en condiciones de jubilarse, es decir, si la edad es mayor o igual a 65 años.
- Si la edad es menor que 18, se debe lanzar la excepción ***EEdadMinima***.



Creación de excepciones - Ejemplo 6

```
Type     // se define la nueva excepción
           EEdadMinima = class (Exception);
 const    EDAD_MINIMA = 18;
           EDAD_JUBILACION = 65;
…
procedure TForm1.Button1Click(Sender: TObject);
Var   edad: Integer;
begin
  try
     edad := StrToInt(Edit1.Text);
     if sePuedeJubilar(edad)  then  ShowMessage('Puede jubilarse')
  …
```

Ejemplo 6 (Continuación)

```
  …
     else
        ShowMessage('El empleado aún no está en edad de
                                  jubilarse.');
  except
    on E : EConvertError do
      showMessage('El número ingresado no es válido.');
    // se define un manejador para la excepción nueva
    on E : EEdadMinima do
      showMessage(E.Message);
  end;
end;
  …
```

Ejemplo 6 (Continuación 2)

```
  ...

function sePuedeJubilar(edad: integer): boolean;
begin
  if edad < EDAD_MINIMA then
      // se lanza explícitamente la excepción.
      raise EEdadMinima.Create('Edad laboral inválida');

  sePuedeJubilar := (edad >= EDAD_JUBILACION);
end;
```

La sentencia *raise* - Sintaxis

- La sentencia ***raise*** puede invocarse con un parámetro, que típicamente es un objeto *Exception*. En esta forma la sentencia ***raise*** puede usarse en cualquier parte:

```
Raise Exception.Create('Error');
```

- La sentencia *raise* no retorna el control a la instrucción siguiente como lo hace una sentencia normal, sino que pasa el control al bloque ***except*** asociado al bloque ***try*** donde se haya ejecutado, si lo hay. La invocación de ***raise*** se comporta exactamente igual que una excepción lanzada automáticamente por Delphi, y sigue el mismo flujo de control.

La sentencia *raise*

- Para crear una excepción de una clase particular se debe invocar algún constructor de dicha clase.

Mensaje a recuperar usando la propiedad **Message**

- Ejemplo
  - **Raise** Exception.Create('Error');

- En Delphi los constructores por convención se llaman *Create*. Sin embargo, una clase puede tener más de un constructor, cada uno con un identificador arbitrario.

Otros constructores de la clase *Exception*

- ***CreateFmt*** es similar a *Create* pero permite dar formato especial al el mensaje de error. Tiene dos argumentos, un String que contiene la plantilla del mensaje, al estilo de la función *printf* del lenguaje C, y un arreglo de valores constantes que contiene los Strings que van a ser insertados en la plantilla. Los especificadores de formato para *CreateFmt* son los mismos que los disponibles para la función *Format*.

```
Raise Exception.CreateFmt('%s no es un número válido.',
                          [Edit1.text]);
```

Otros constructores de la clase *Exception*

- ***CreateHelp*** es como *Create*, pero provee una forma de usar ayuda contextual al manejar excepciones. Recibe un segundo argumento, un ID de ayuda contextual (HelpContext ID). Este valor se puede usar como parámetro de *MessageDlg* dentro de un manejador, por ejemplo:

```
Try
  Raise Exception.CreateHelp('Error. Use F1 para más información.', 10);
Except
  On E : Exception do
   MessageDlg(E.Message, mtError, [mbOk, mbHelp], E.HelpContext);
End;
```

Algunas excepciones predefinidas

Nombre	Descripción
EConvertError	Error de conversión de tipos.
EAccessViolation	Acceso a un puntero nulo.
EDivByZero	División entera por cero.
EZeroDivide	División flotante por cero.
EInvalidCast	Error de casting de tipos.
ERangeError	Valor fuera de rango.