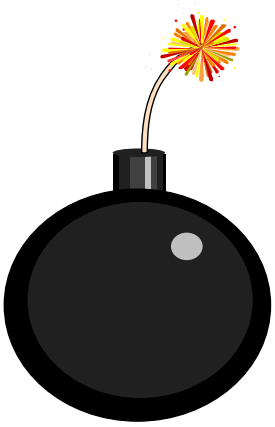
# Exceções

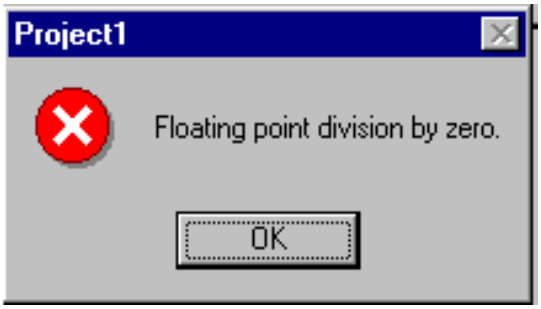
Fabiano Rodrigo Alves Nascimento



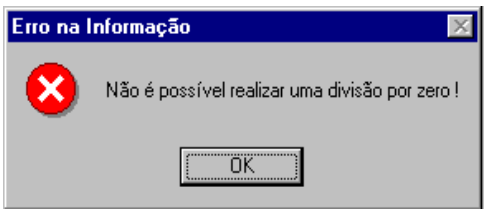
# ***Exceções***

## *1. Manipulando Exceções*

Quando um erro ocorre no aplicativo durante a execução, a aplicação chama uma *exceção* . Uma exceção é um objeto que contém informações sobre o erro ocorrido e onde ocorreu. Se não for especificado no código uma resposta ao erro, o Delphi mostrará uma mensagem descrevendo o erro :




No exemplo acima, poderia-se clicar no botão OK e continuar a executar a aplicação. Mas muitas vezes o erro ocorrido impede a continuação da operação, ou leva a perda de informações valiosas.
O Ideal seria que se pudesse tratar os erros ocorridos, evitando a perda de dados ou a necessidade de encerrar a aplicação. Além de tratar o erro , a rotina de tratamento de erros poderia enviar ao usuário uma mensagem em português , mais significativa :




O Delphi permite que se desenvolva aplicações robustas, ou seja, que tratam os erros ocorridos de forma que os dados não se percam . Para fazer uma aplicação robusta, é necessário que o código reconheça as excessões quando elas aparecerem, respondendo a elas.
Cria-se uma resposta a uma exceção em um bloco de código, chamado de *bloco protegido* porque está resguardado de erros que podem parar a aplicação ou perder dados. Um bloco protegido começa com a palavra reservada **try** e termina com a palavra reservada **end.** Entre essas palavras determina-se os comandos protegidos e a forma de reação aos erros.
Quando se define um bloco protegido, especifica-se respostas a exceções que podem ocorrer dentro deste bloco. Se a exceção ocorrer, o fluxo do programa pula para a resposta definida, e após executá-la, abandona o bloco.
A forma mais simples para responder a uma exceção é garantir que algum código limpo é executado. Este tipo de resposta não corrige o erro, mas garante que sua aplicação não termine de forma instável. Normalmente, usa-se este tipo de resposta para garantir a liberação de recursos alocados, mesmo que ocorra um erro.
Outra forma de se responder a uma exceção é tratando o erro , ou seja, oferecendo uma resposta específica para um erro específico. Ao tratar o erro destrui-se a exceção , permitindo que a aplicação continue a rodar.

## *2. A Estrutura Try..Finally..End*

O código precisa garantir que , mesmo ocorrendo um erro, os recursos alocados sejam desalocados. Entre estes recursos estão : arquivos, memória, recursos do windows, objetos.
Para garantir a desalocação dos recursos , usamos a estrutura abaixo :

```
{aloca os recursos}
try
    {comandos que usam os recursos}
finally
    { libera os recursos }
end;
```

A aplicação sempre executará os comandos inseridos na parte *finally* do bloco, mesmo que uma exceção ocorra. Quando um erro ocorre no bloco protegido, o programa pula para a parte *finally* , chamada de código limpo, que é executado. Mesmo que não ocorra um erro, estes comandos são executados.
No código a seguir, foi alocada memória e gerado um erro, ao tentar-se a divisão por 0 (zero). Apesar do erro, o programa libera a memória alocada:

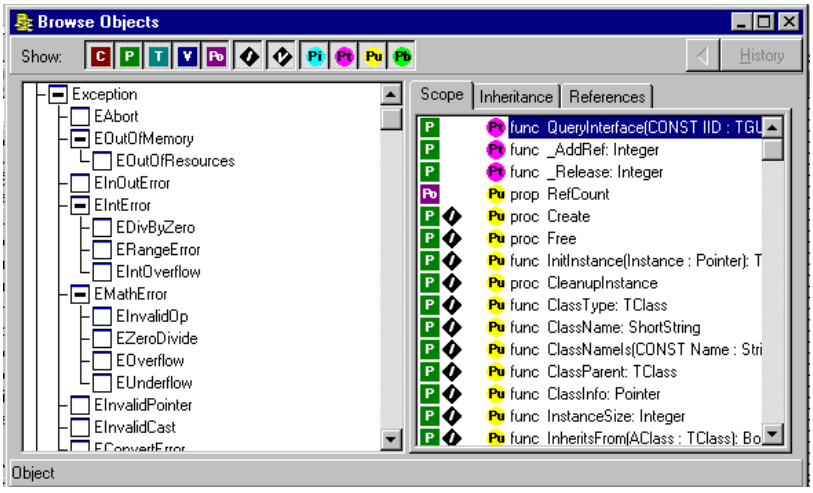
```
Procedure Tform1.Button1click (Sender : Tcomponent );
Var
Ponteiro : Pointer;
Inteiro, Dividendo : Integer;
Begin
      Dividendo:= 0;
      GetMem(Ponteiro, 1024);
      Try
            Inteiro: = 10 div dividendo;
       Finally
            FreeMem(Ponteiro, 1024);
        End;
End;
```

## 3. *Exceções da RTL*

Quando se escreve código chama rotinas da biblioteca de run-time ( RTL, run-time library), como funções matemáticas ou de manipulação de arquivos, os erros aparecem na forma de exceções. Por padrão, a RTL manda uma mensagem para o usuário, mas pode-se tratar estes erros de outra forma.
As exceções geradas pela RTL são definidas na unit *SysUtils* , e todas descendem da classe mais geral Exception, que provê a string que aparece no quando de mensagem da exceção.
Há sete tipos de exceções geradas pela RTL :

- ✓ Exceções de entrada e saída;
- ✓ Exceções de memória heap;
- ✓ Exceções matemáticas envolvendo inteiros;
- ✓ Exceções matemáticas envolvendo pontos flutuantes;
- ✓ Exceções de typecast;
- ✓ Exceções de conversão;
- ✓ Exceções de Hardware;

Observa-se estas exceções com mais detalhes através do OBJECTBROWSER , ativado através do comando *View/Browser* :



✓ *Exceções de Entrada e Saída*

Ocorrem quando RTL tenta acessar arquivos ou dispositivos de entrada e saída.
A Unit *SysUtils* define uma exceção genérica de entrada e saída chama *EinOutError*, que contém um atributo chamado *ErrorCode* que indica o erro ocorrido.

✓ *Exceções de Memória Heap*

Ocorrem quando se tenta alocar ou acessar memória dinâmica. São definidos dois tipos de exceção : *EoutofMemory* (indica que não há memória suficiente) e *EinvalidePointer* (Ponteiro Inválido).

✓ *Exceções Matemáticas para Inteiros*

Ocorrem quando se realiza operações com números inteiros. A Unit *SysUtils* define uma exceção genérica chamada *EintError*, com três tipos de exceção derivadas : *EdividByZero* (Divisão por zero), *ErangeError* (Número fora da região), e *EintOverFlow* (Estouro na operação com inteiro).

✓ *Exceções de Pontos Flutuantes*

Ocorrem quando se realiza operações com dados do tipo real. A Unit *SysUtils* define uma exceção genérica chamada *EMathError*, com as seguintes exceções derivadas : *EinvalidOp* ( Processador encontrou uma instrução indefinida), *EZeroDivide* (Divisão por zero), *EOverFlow* , *EUnderFlow*.

✓ *Exceções de TypeCast*

Ocorrem quando você tenta atribuir um objeto a um tipo inválido usando o operados **as** . Gera-se a exceção *EinvalidCast* quando isto ocorre.

✓ *Exceções de Conversão*

Ocorrem quando se convertem dados de um tipo para outro. A Unit *SysUtils* define uma exceção chamada *EconvertError* quando isto ocorre, avisando que a conversão não pode ser feita.

✓ *Exceções de Hardware*

Podem ocorrer em dois tipos de situação: quando o processador detecta uma falha, ou quando a aplicação gera uma interrupção intencional.
A Unit *SysUtils* define uma exceção genérica chamada *EprocessorException*, com as exceções derivada : *Efault* (Exceção Base), *EGPFault* (Falha Geral de Proteção), *EstackedFault* (acesso ilegal ao seguimento stack do processador), *EpageFault* (o gerenciador de memória do windows não está conseguindo realizar o qrquivo de swap), *EinvalidOpCode* ( o processador encontra uma instrução indefinida), *EbreakPoint* (a aplicação gerou uma interrupção breakpoint), *Esinglestep* (a aplicação gera uma interrupção single-step).

## *4. A Estrutura Try..Except..End*

Um tratamento de exceção é um código que trata erros que ocorrem dentro de blocos protegidos. Para definir um tratamento de exceção, utiliza-se a seguinte construção :

```
try
    {comandos que se deseja proteger}
finally
    { comandos de tratamento de erros }
end;
```

A aplicação irá executar os comandos na parte *except* somente se ocorrer um erro. Se na parte *try* chamar uma rotina que não trata erros, e um erro ocorrer, ao voltar para este bloco a parte *except* será executada. Uma vez que a aplicação localiza um tratamento para a exceção ocorrida, os comandos são executados, e o objeto exceção é destruído. A execução continua até o fim do bloco.
Dentro da parte *except* define-se um código a ser executado para manipular tipos específicos de exceção. Por exemplo, o código abaixo trata o erro de divisão por zero, através da exceção *Edivbyzero* :

```
Function Divisao ( soma, numero : Integer) : integer;
Begin
Try
    Result:=  soma div numero;
Except
     On Edivbyzero  do  Result:= 0;
End;
```

A palavra reservada *ON* define respostas para uma exceção. *On* está sempre junto de *do,* para manipular a exceção. Para ler informações específicas sobre o erro ocorrido, usa-se uma variação da estrutura *on..do,* que provê uma variável temporária que engloba a exceção. Nesse caso, pode-se criar um quadro próprio de mensagens contendo a string da mensagem da exceção :

```
Try
  {comandos}
Except
     On E : EInvalidOperation do
      Messagedlg ( 'Ignorando a Exceção : ' + E.Message, mtinformation , [mbok] , 0 );
End;
```

Onde a variável temporária *E* é do tipo *EinvalidOperation*.
Pode-se prover um tratamento padrão de erro para tratar exceções que não tem tratamentos especificados. Para isto, adicione uma prate *else* na parte *except* do bloco :

```
Try
  {comandos}
Except
     On EPrimeiroTipo do
      {código especificado para o primeiro tipo de erro}
     On EsegundoTipo  do
      {código especificado para o segundo tipo de erro}
     else
      {código padrão de tratamento de erros }
End;
```

Além disso, pode-se utilizar as exceções genéricas para tratar um erro, em vez de uma exceção específica. Por exemplo , se desejar-se tratar um erro relacionado a uma operação com inteiros, mas não sabe exatamente o erro, poderá utilizar a exceção *Einterror* , que é a exceção genérica da qual se derivam outras exceções relacionadas a inteiros:

```
Try
   {Comandos}
except
     on EintError do
     {Código de tratamento de erros}
end;
```

## 5. *Exceções da VCL*

Quando se escreve código chama rotinas da biblioteca de run-time ( RTL, run-time library), como funções matemáticas ou de manipulação de arquivos, os erros aparecem na forma de exceções. Por padrão, a RTL manda uma mensagem para o usuário, mas pode-se tratar estes erros de outra forma.
As exceções geradas pela RTL são definidas na unit *SysUtils* , e todas descendem da classe mais geral Exception, que provê a string que aparece no quando de mensagem da exceção.
Há sete tipos de exceções geradas pela RTL :

- ✓ *EDBEditError*
  Diz respeito a erros ocorridos com o uso do componente TDBEdit.

- ✓ *EDataBaseError*
  Gerado sempre que ocorre um erro no Banco de Dados

- ✓ *EDBEngineError*
  Gerado sempre ocorre um erro no BDE.
- ✓ *EPrinter*
  Gerado quando ocorre um erro com a impressora
- ✓ *EDDEError*
  Gerado a aplicação não consegue encontrar um servidor DDE
- ✓ *EMCIDeviceError*
  Gerado quando ocorre um erro ao tentarmos acessar um dispositivo multimídia.

## *6. Exceções Silenciosas*

Pode-se definir exceções que não mostrem um quadro de mensagem para o usuário quando aparecem. São chamadas exceções Sileciosas. O caminho mais curto para criar esta exceção é através da procedure *Abort* . Esta procedure automaticamente gera uma exceção do tipo *Eabort* , que abortará a operação sem mostrar uma mensagem.
O exemplo abaixo aborta a operação de inclusão de itens em um Listbox quando tentamos inserir o terceiro elemento :

```
Begin
   For i := 1  to 10 do
        Begin
        Listbox.Items.Add ( Inttostr ( i ) );
        If i = 3  then
                Abort;
         End;
End;
```

## *7. Definindo suas Próprias Exceções*

Além das exceções geradas em tempo de execução e daquelas geradas pelo uso incorreto de componentes, pode-se acrescentar no código tratamento para suas próprias exceções.
Para gerar uma exceção, é chamada a palavra reservada *raise* seguida por uma instância do objeto *Exception* :

```
If Edit1.Text = ‘’  then
    Raise exception.Create (‘Dados Incompletos !’ );
```

## *8. Manipulando Exceções Globais*

O Delphi oferece um evento chamado *OnException* , ligado à a classe *Tapplication* , que permite manipular qualquer exceção que ocorra em seu programa, mesmo que não sabendo em que parte do programa ela ocorreu.
Inicialmente, deve-se criar manualmente, como um método de Tform1, a chamada para o método que tratará os erros :

```
Tform1 =  Class (Tform)
Procedure Trata_Erros (Sender : Tobject ; E : Exception );
End;
```

Depois de declarar a procedure, deve-se atribuí-la ao evento *OnException* de *Tapplication*. A atribuição é feita no evento *OnCreate* do formulário principal :

```
Procedure Tform1.FormCreate ( Sender : Tobject );
Begin
Application.OnException := Trata_Erros;
End;
```

O método Trata_Erros será agora chamado quando qualquer exceção ocorrer. Pode-se determinar mensagens para erros específicos, ou mensagens gerais :

```
Procedure Tform1.Trata_Erros (Sender : Tobject; E : Exception);
Begin
If  E is EdatabaseError then
      Showmessage('Erro no banco de dados')
Else
       Showmessage( 'Há erros no programa');
End;
```

## *9. Traduzindo as Mensagens de Erro do Delphi*

Se possuir um manipular de arquivos de recursos .RES (como o Resource WorkShop) , pode-se alterar as mensagens as mensagens de erros oferecidas pelo Delphi, traduzindo-as . Os arquivos de recursos do Delphi ficam no diretório \Arquivos de Programas\Borland\Delphi 3\Lib .