Exception Handling

10 print “hello EKON”;
20 Goto 10;

“Exceptions measure the stability of code before it has been written”

Kleiner
Kommunikation.....

EKON 16

maXbox

Agenda EKON

- What are Exceptions ?
- Prevent Exceptions (MemoryLeakReport)
- Log Exceptions (Global Exception Handler)
- Check Exceptions (Test Driven)
- Compiler Settings

Some Kind of wonderful ?

- One of the main purposes of exception handling is to allow you to remove error-checking code altogether and to separate error handling code from the main logic of your application.
- [DCC Fatal Error] Exception Exception: Compiler for personality "Delphi.Personality" and platform "Win32" missing or unavailable.

Types of Exceptions

Throwable

- Checked (Handled) Exceptions
- Runtime Exceptions (unchecked)
- Errors (system)
- Hidden, Deprecated or Silent

```
EStackOverflow = class(EExternal) end deprecated;
```

Kind of Exceptions

Some Structures

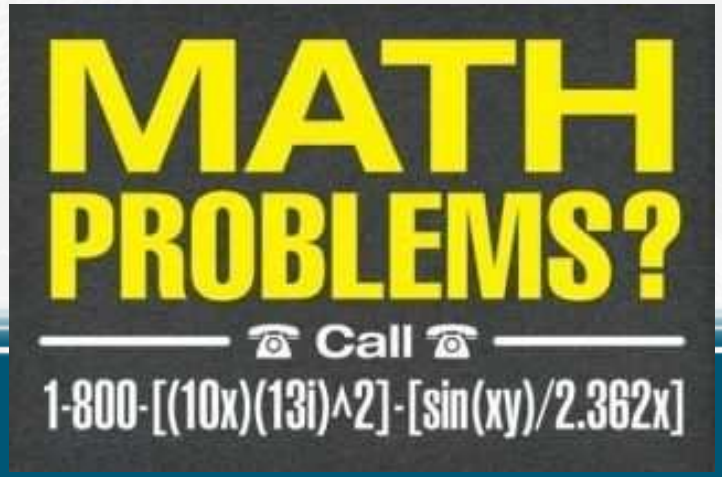
- try finally
- try except
- try except finally
- try except raise
- Interfaces or Packages (design & runtime)
- Static, Public, Private (inheritance or delegate)

Cause of Exceptions

Bad Coordination

- Inconsistence with Threads
- Access on objects with Null Pointer
- Bad Open/Close of Input/Output Streams or I/O Connections (resources)
- Check return values, states or preconditions
- Check break /exit in loops
- Access of constructor on non initialized vars

- Who the hell is general failure and why does he read on my disk ?!
- Die Allzweckausnahme!
- Don’t eat exceptions → silent



Top 5 Exception Concept

1. Memory Leak Report
- if maxform1.STATMemoryReport = true then
- ReportMemoryLeaksOnShutdown:= true; →Ex. 298_bitblt_animation3
2. Global Exception Handler Logger
3. Use Standard Exceptions
4. Beware of Silent Exceptions (Ignore but)
5. Safe Programming with Error Codes

Ex: 060_pas_datefind_exceptions2.txt

Memory Leaks I

- The Delphi compiler hides the fact that the string variable is a heap pointer to the structure but setting the memory in advance is advisable:

```
path: string;
setLength(path, 1025);
```

- Check against Buffer overflow

```
var
  Source, Dest: PChar;
  CopyLen: Integer;
begin
  Source:= aSource;
  Dest:= @FData[FBufferEnd];
  if BufferWriteSize < Count then
    raise EFIFOStream.Create('Buffer over-run.');
```

Memory Leaks II

- Avoid pointers as you can
- Ex. of the win32API:

```
pVinfo = ^TVinfo;
function TForm1.getvolInfo(const aDrive: pchar; info: pVinfo):
    boolean;
// refactoring from pointer to reference
function TReviews.getvolInfo(const aDrive: pchar; var info: TVinfo):
    boolean;
```

"Each pointer or reference should be checked to see if it is null. An error or an exception should occur if a parameter is invalid."

Global Exceptions

Although it's easy enough to catch errors (or exceptions) using "try / catch" blocks, some applications might benefit from having a global exception handler. For example, you may want your own global exception handler to handle "common" errors such as "divide by zero," "out of space," etc.

Thanks to TApplication's ‘OnException’ event - which occurs when an unhandled exception occurs in your application, it only takes three (or so) easy steps get our own exception handler going:

Global Handling

1. Procedure AppOnException(sender: TObject;
   E: Exception);

```
if STATExceptionLog then
    Application.OnException:= @AppOnException;
```

Global Log

2. procedure TMaxForm1.AppOnException(sender: TObject;
E: Exception);

```
begin
  MAppOnException(sender, E);
end;

procedure MAppOnException(sender: TObject; E: Exception);
var
  FErrorLog: Text;
  FileNamePath, userName, Addr: string;
  userNameLen: dWord;
  mem: TMemoryStatus;
```

Log Exceptions

3. Writeln(FErrorLog+ Format('%s %s [%s] %s %s
   [%s]'+[DateTimeToStr(Now),'V:'+MBVERSION,
       UserName, ComputerName, E.Message,'at: ,Addr]));

```
try
  Append(FErrorlog);
except
  on EInOutError do Rewrite(FErrorLog);
end;
```

Use Standards

```
(CL.FindClass('Exception'),'EExternal');
(CL.FindClass('TOBJECT'),'EExternalException');
(CL.FindClass('TOBJECT'),'EIntError');
(CL.FindClass('TOBJECT'),'EDivByZero');
(CL.FindClass('TOBJECT'),'ERangeError');
(CL.FindClass('TOBJECT'),'EIntOverflow');
(CL.FindClass('EExternal'),'EMathError');
(CL.FindClass('TOBJECT'),'EInvalidOp');
(CL.FindClass('EMathError'),'EZeroDivide');
```

"Die andere Seite, sehr dunkel sie ist" - "Yoda, halt's Maul und iß Deinen Toast!"

Top Ten II

6. Name Exceptions by Source (ex. a:= LoadFile(path) or a:= *LoadString(path)*
7. Free resources after an exception (Shotgun Surgery (try... except.. finally) → pattern missed, see wish at the end))
8. Don't use Exceptions for Decisions (Array bounds checker) -> Ex.
9. Never mix Exceptions in try… with too many delegating methods) → twoexcept
10. Remote →(check remote exceptions, →cport)

Ex: 060_pas_datefind_exceptions2.txt

Testability

- Exception handling on designtime

```
{$IFDEF DEBUG}
  Application.OnException:= AppOnException;
{$ENDIF}
```

- Assert function

```
accObj:= TAccount.createAccount(FCustNo,
                                std_account);
assert(aTrans.checkOBJ(accObj), 'bad OBJ cond.');
```

- Logger

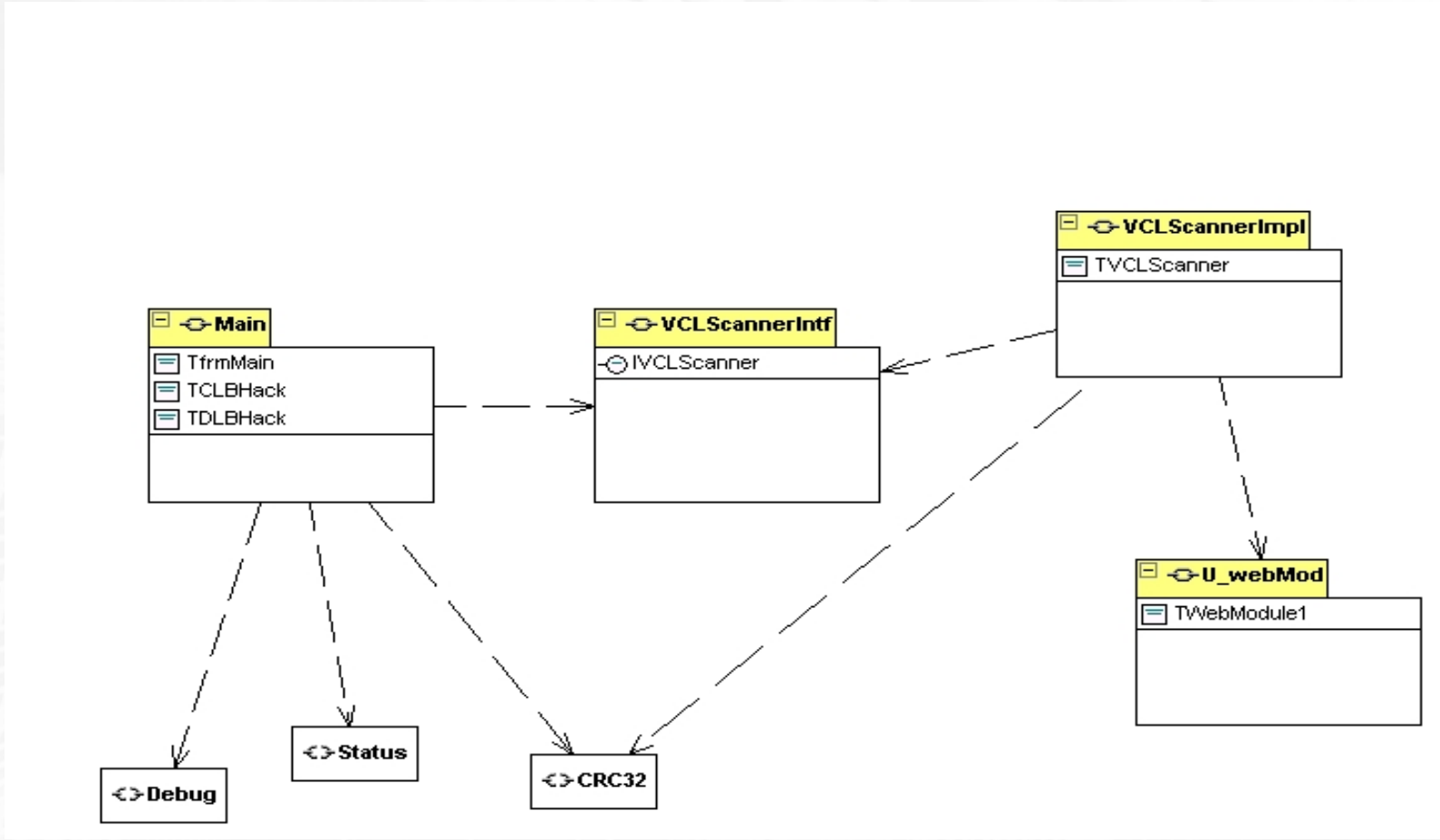
```
LogEvent('OnDataChange', Sender as TComponent);
LogEvent('BeforeOpen', DataSet);
```

Check Exceptions

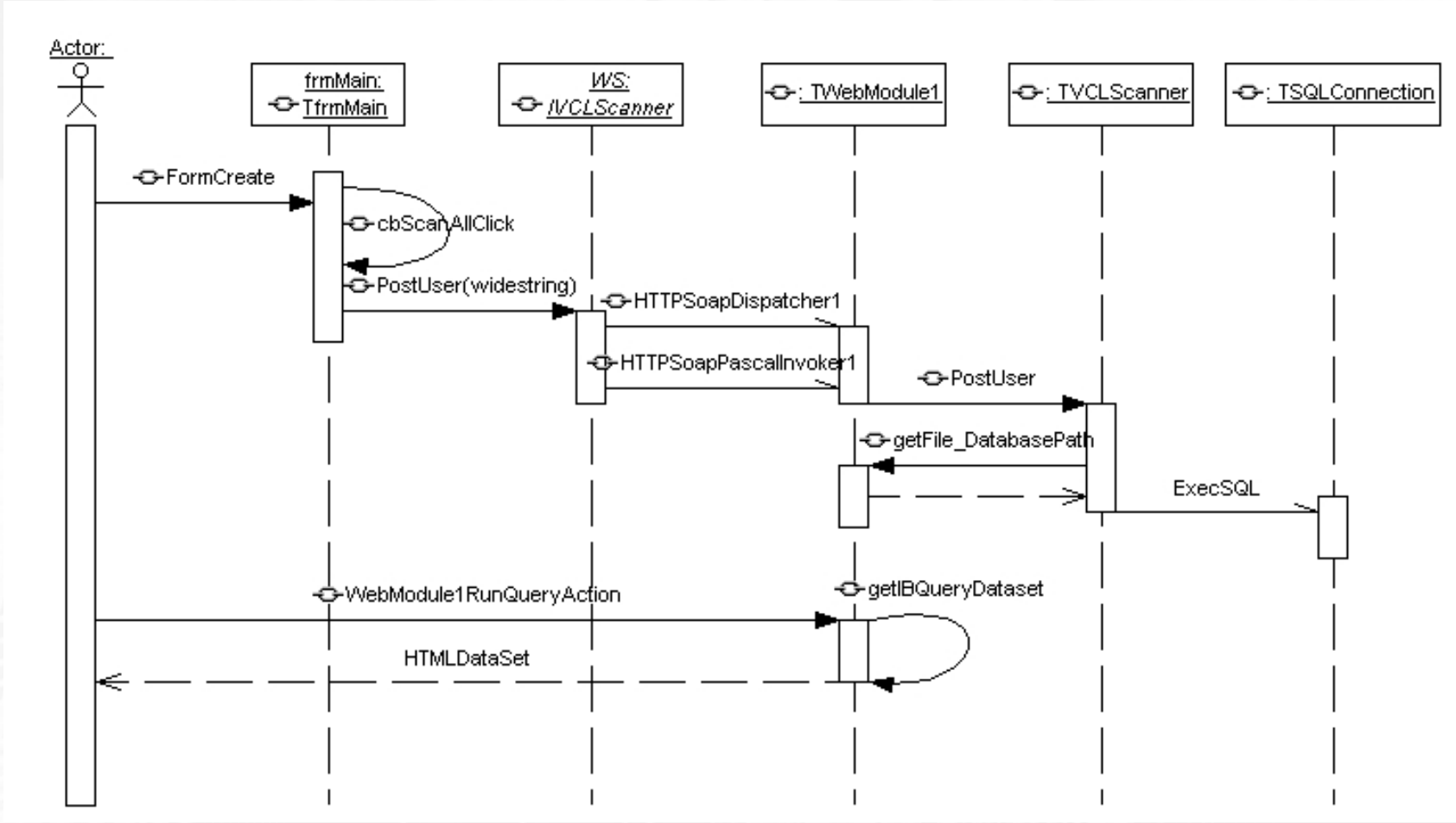
- Check Complexity

```
function IsInteger(TestThis: String): Boolean;
begin
  try
    StrToInt(TestThis);
  except
    on E: ConvertError do
      result:= False;
  else
    result:= True;
  end;
end;
```

Module Tests as Checksum



Test on a SEQ Diagram



Compiler & runtime checks

Controls what run-time checking code is generated. If such a check fails, a run-time error is generated. → ex. Stack overflow

- Range checking
  Checks the results of enumeration and subset type operations like array or string lists within bounds
- I/O checking
  Checks the result of I/O operations
- Integer overflow checking
  Checks the result of integer operations (no buffer overrun)
- Missing: Object method call checking
  Check the validity of the method pointer prior to calling it (more later on).

Range Checking

```
{$R+}
   SetLength(Arr,2);
   Arr[1]:= 123;
   Arr[2]:= 234;
   Arr[3]:= 345;
 {$R-}
```

Delphi (sysutils.pas) throws the ERangeError exception →
ex. snippet

I/O Errors

The $I compiler directive covers two purposes! Firstly to include a file of code into a unit. Secondly, to control if exceptions are thrown when an API I/O error occurs.
{$I+} default generates the EInOutError exception when an IO error occurs.
{$I-} does not generate an exception. Instead, it is the responsibility of the program to check the IO operation by using the IOResult routine.

```
 {$i-}
   reset(f,4);
   blockread(f,dims,1);
 {$i+}
   if ioresult<>0 then begin
```

Overflow Checks

When overflow checking is turned on (the $Q compiler directive), the compiler inserts code to check that CPU flag and raise an exception if it is set. → ex.
The CPU doesn't actually know whether it's added signed or unsigned values. It always sets the same overflow flag, no matter of types A and B. The difference is in what code the compiler generates to check that flag.
In Delphi ASM-Code a call@IntOver is placed.

Silent Exception, but

EStackOverflow = class(EExternal) → ex. webRobot
end deprecated;

```
{$Q+} // Raise an exception to log
    try
      b1:= 255;
      inc(b1);
      showmessage(inttostr(b1));
//show silent exception with or without
    except
      on E: Exception do begin
      //ShowHelpException2(E);
      LogOnException(NIL, E);
    end;
end;
```

Exception Process Steps

The act of serialize the process:

- Use Assert {$C+} as a debugging check to test that conditions implicit assumed to be true are never violated (pre- and postconditions). → ex.
- Create your own exception from Exception class
- Building the code with try except finally
- Running all unit tests with exceptions!
- Deploying to a target machine with global log
- Performing a “smoke test” (compile/memleak)

Let‘s practice

- function IsDate(source: TEdit): Boolean;
- begin
- try
- StrToDate(TEdit(source).Text);
- except
- on EConvertError do
- result:= false;
- else
- result:= true;
- end;
- end;

Is this runtime error or checked exception handling ?

Structure Wish

The structure to handle exceptions could be improved. Even in XE3, it's either try/except or try/finally, but some cases need a more fine-tuned structure:

```
try
  IdTCPClient1.Connect;
    ShowMessage('ok');
  IdTCPClient1.Disconnect;
except
  ShowMessage('failed connecting');
finally //things to run whatever the outcome
  ShowMessage('this message displayed every time');
end;
```

Exceptional Links:

- Exceptional Tools:
  http://www.madexcept.com/
- http://www.modelmakertools.com/
- Report Pascal Analyzer:
  http://www.softwareschule.ch/download/pascal_analyzer.pdf
- *Compiler Options*:
- http://www.softwareschule.ch/download/Compileroptions_EKON13.pdf
- Example of code with maXbox
  http://www.chami.com/tips/delphi/011497D.html
- Model View in Together:
  *www.softwareschule.ch/download/**delphi**2007_modelview.pdf*

Q&A

max@kleiner.com
www.softwareschule.ch