

# Refactoring mit Delphi 2006

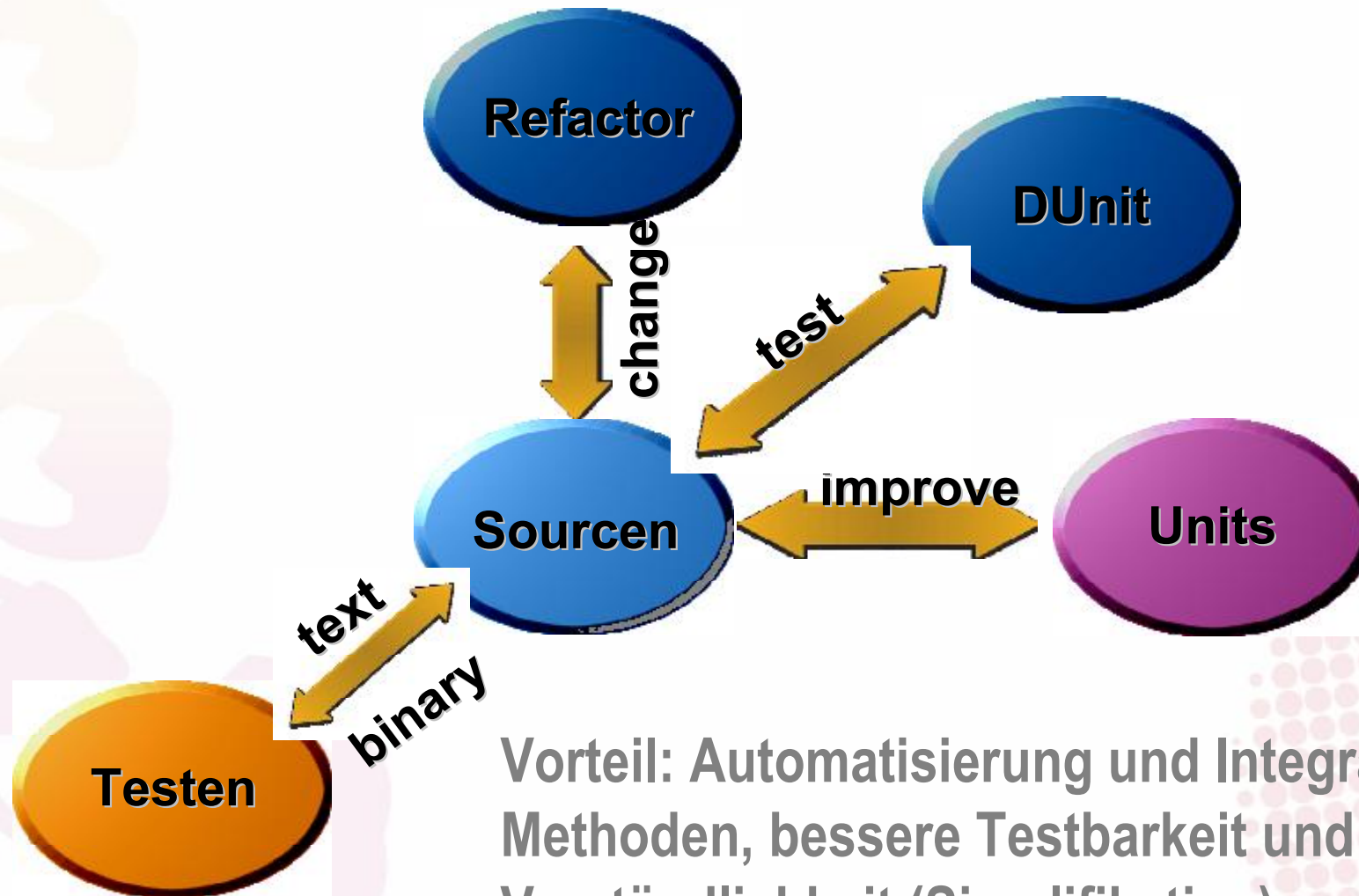


Als Refactoring betrachtet man allgemein ein Vereinfachen, Verbessern und Stabilisieren einer bestehenden Codestruktur, welche keine Änderung auf das „beobachtbare“ Verhalten der Applikation und dessen Ergonomie bewirken soll.

“Refactoring is 'improving the design of code after it has been written’”

**Kleiner**  
K o m m u n i k a t i o n . . . . .

# Refactor Umfeld



Vorteil: Automatisierung und Integration von Methoden, bessere Testbarkeit und Verständlichkeit (Simplifikation)

# Refactoring: What's all about ?



**Never touch a running system ?!:**

//System.Get just reads the contents of memory locations:

Get(a, v);     or v:= M[a];

o.IntToStr;    or v:= IntToStr(o);

GetMem(p, Count \* SizeOf(Pointer)); or

GetMem(FDataPtr, MemSize);

If you can't see how to easily test or maintain your code, it's probably time to refactor!

- How to test?
- Refactor to change code
- Build Test Class and run test to confirm it's okay

# When and why Refactoring ?



After a Code Review

By change of a release

Practice with UML (Patterns or Profiles)

**Law of Demeter not passed**

**Bad Testability**

- Work on little steps at a time
- Test after each step
- Each method should do one thing
- Modify not only structure but also format

# Refactoring und DUnit

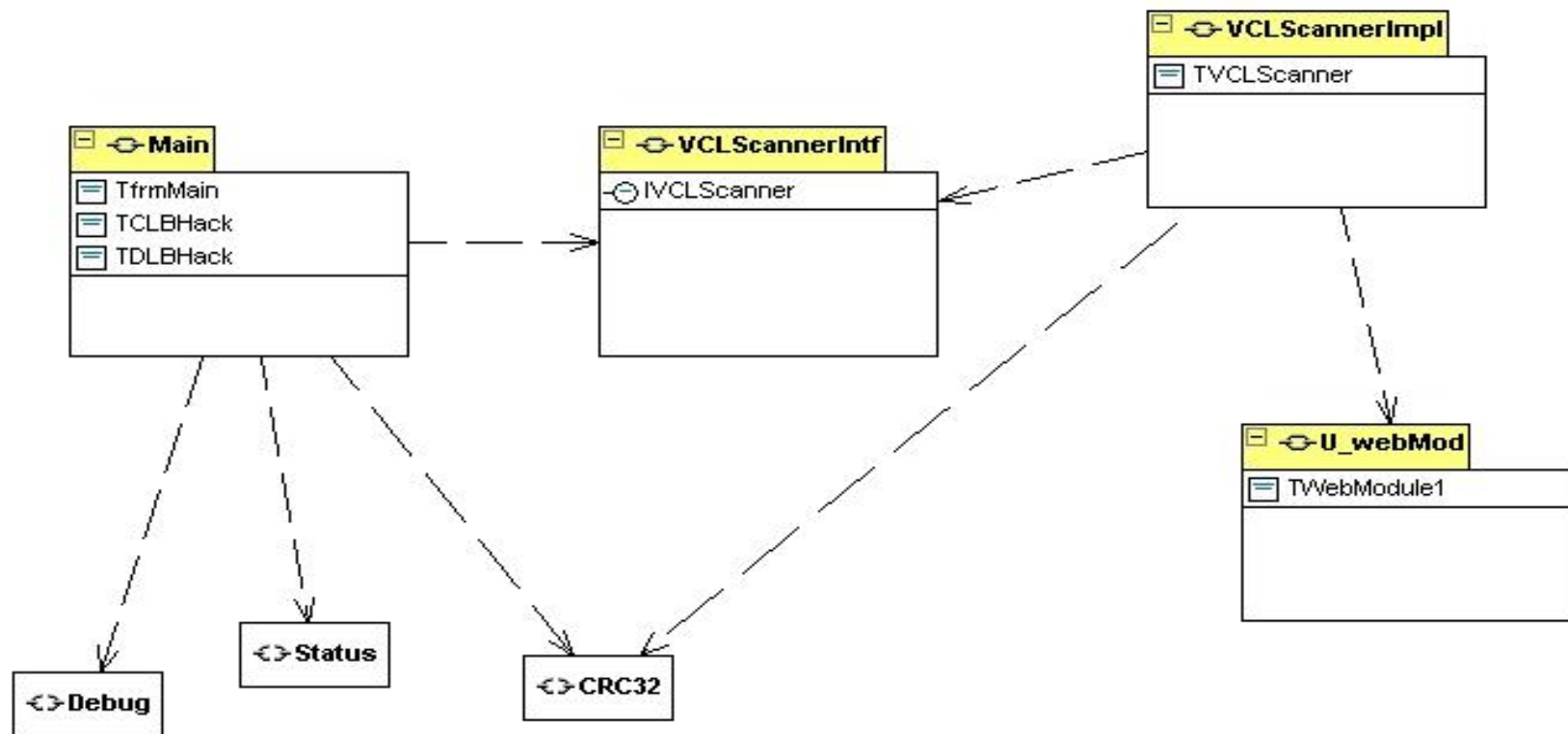


Das Testen mit DUnit kann an Refactoring koppeln und setzt auf drei Elementen auf:

- Das eigentliche Testobjekt, fixture genannt
- Der Testfall zum Objekt, action genannt
- Das Resultat nach dem Testfall, check genannt

Als einfaches Beispiel sei hier das fortlaufende Zählen von Zeichen erwähnt. Das Resultat kann Positiv, alphanumerisch oder ein unerwarteter Fehler sein. Der Check testet, ob das Resultat dem Referenzwert plus einem Zeichen entspricht (Bsp. folgt)

# Test Units in practice



# Law of Demeter



You should avoid:

- Large classes
- More than seven or eight variables
- More than fifty methods
- You probably need to break up the class

Components in (Strategy, Composite, Decorator)

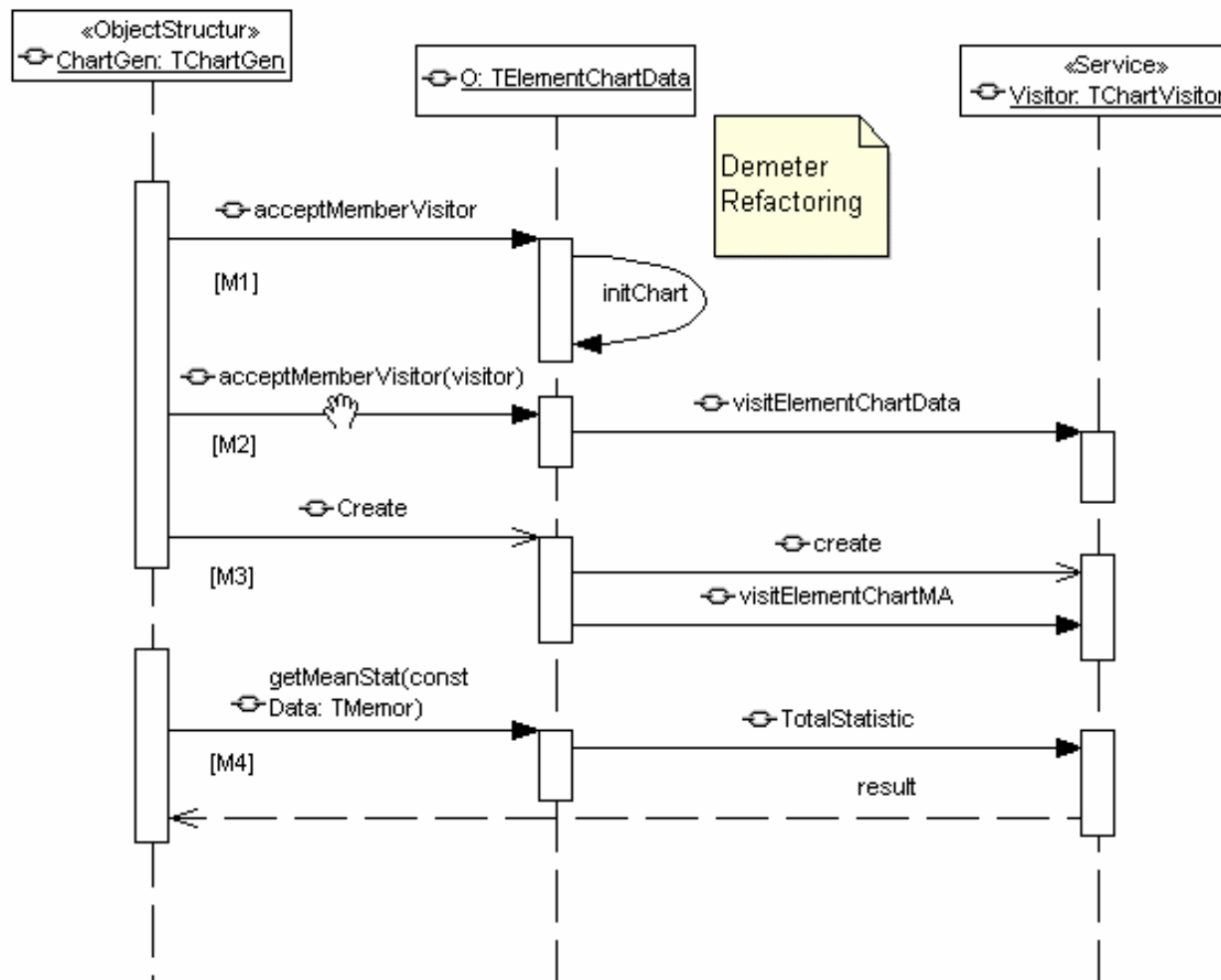
*Das Gesetz von Demeter (don't talk to strangers) besagt, dass ein Objekt O als Reaktion auf eine Nachricht m, weitere Nachrichten nur an die folgenden Objekte senden sollte:*

# Demeter konkret



1. **[M1]** an Objekt O selbst  
Bsp.: `self.initChart(vdata);`
2. **[M2]** an Objekte, die als Parameter in der Nachricht m vorkommen  
Bsp.: `O.acceptmemberVisitor(visitor)  
visitor.visitElementChartData;`
3. **[M3]** an Objekte, die O als Reaktion auf m erstellt  
Bsp.: `visitor:= TChartVisitor.create(cData, madata);`
4. **[M4]** an Objekte, auf die O direkt mit einem Member zugreifen kann  
Bsp.: `O.Ctnr:= visitor.TotalStatistic`

# Demeter Test as SEQ



# Testability – We should avoid:



Bad Naming (no naming convention)

Duplicated Code (side effects)

Long Methods (to much code)

Temporary Fields (confusion)

Long Parameter List (Object is missing)

Data Classes (no methods)

- Large Class
- Class with too many delegating methods
- Coupled classes

# Refactoring Process



The act of serialize the process:

- Build unit test
- Refactor and test the code
- Check with Pascal Analyzer or another tool
- Building the code
- Running all unit tests
- Generating the documentation
- Deploying to a target machine
- Performing a “smoke test” (just compile)

# Was aus der Krypto Praxis



- 1
- 11
- 21
- 1211
- 111221
- 312211

Versuchen Sie nun diese “optische” Reihe fortzusetzen, suchen Sie nach einer Systematik oder Logik?

```

function runString(Vshow: string): string;
var i: byte;
Rword, tmpStr: string;
cntr, nCount: integer;
begin
cntr:=1; nCount:=0;
Rword:=""; //initialize
tmpStr:=Vshow; // input last result
for i:= 1 to length(tmpStr) do begin
  if i= length(tmpstr) then begin
    if (tmpStr[i-1]=tmpStr[i]) then cntr:= cntr +1;
    if cntr = 1 then nCount:= cntr
    Rword:= Rword + intToStr(ncount) + tmpStr[i]
  end else
    if (tmpStr[i]=tmpStr[i+1]) then begin
      cntr:= cntr +1;
      nCount:= cntr;
    end else begin
      if cntr = 1 then cntr:=1 else cntr:=1; //reinit counter!
      Rword:= Rword + intToStr(ncount) + tmpStr[i] //+ last char(tmpStr)
    end;
  end; // end for loop
result:=Rword;
end;

```

# Before R.



# After R.



```
function charCounter(instr: string): string;
var i, cntr: integer;
    Rword: string;
begin
cntr:= 1;
Rword:=' ';
for i:= 1 to length(instr) do begin
//last number in line
if i= length(instr) then
concatChars()
else
if (instr[i]=instr[i+1]) then cntr:= cntr +1
else begin
concatChars()
//reinit counter!
cntr:= 1;
end;
end; //for
result:= Rword;
end;
```

# Testfunction



```
function teststring(vstring: string): integer;  
var a, b, i: integer;  
begin  
  for i:= 1 to length(vstring) do begin  
    a:= (ord(vstring[i]))  
    b:= b + a;  
  end  
  result:= b  
end;
```

Check with a reference value:

```
if teststring(charCounter(RUN2)) = 578 then ...  
  //CheckEquals(578, teststring(charCounter(RUN2)))
```

# Refactoring Techniken



| Einheit   | Refactoring Funktion                | Beschreibung  |
|-----------|-------------------------------------|---|
| Package   | Rename Package                      | Umbenennen eines Packages   |
| Package   | Move Package                        | Verschieben eines Packages  |
| Class     | Extract Superclass                  | Aus Methoden, Eigenschaften eine Oberklasse erzeugen und verwenden          |
| Class     | Introduce Parameter                 | Ersetzen eines Ausdrucks durch einen Methodenparameter                      |
| Class     | Extract Method                      | Heraustrennen einer Codepassage   |
| Interface | Extract Interface                   | Aus Methoden ein Interface erzeugen   |
| Interface | Use Interface                       | Erzeuge Referenzen auf Klasse mit Referenz auf implementierte Schnittstelle |
| Component | Replace Inheritance with Delegation | Ersetze vererbte Methoden durch Delegation in innere Klasse                 |
| Class     | Encapsulate Fields                  | Getter- und Setter einbauen   |
| Modell    | Safe Delete                         | Löschen einer Klasse mit Referenzen   |

# Consider the Options



The tool should provide options. Whenever you rename certain entities, a tool should automatically replace all relevant identifiers in code in order to keep code consistent

## Each refactoring operation has its own set of constraints!

- How to treat extracted code in original method – you can choose between
- Leave – leaves the original text unchanged, activates the new method.
- Select – leaves the original text unchanged and selects it (does not activate the new method).
- Comment – puts comment braces around the original text and activates new method.
- Remove – removes the original code and activates new method.
- copy the vars to the new method,
- add the new class to the same unit as the source method's class (if any).

# Constraints in Delphi



- If an error results from a refactoring, the engine cannot apply the change. For example, you cannot rename an identifier to a name that already exists in the same declaration scope. If you still want to rename your identifier, you need to rename the identifier that already has the target name first, then refresh the refactoring.
- You can also redo the refactoring and select a new name. The refactoring engine traverses parent scopes, searching for an identifier with the same name. If the engine finds an identifier with the same name, it issues a warning.

# Refactoring with D9



**If you only select by install win32 refactor won't work!!**

- Delphi 2005 provides following refactoring operations:
- Symbol Rename (Delphi, C#)
- Extract Method (Delphi)
- Declare Variable and Field (Delphi)
- Sync Edit Mode (Delphi, C#)
- Find References (Delphi, C#)
- Extract Resource string (Delphi)
- Find unit /import Namespace(Delphi)
- Undo (Delphi, C#)
- Examples...

# Refactoring in D9 SP3

## Corrections done



- When extracting methods involving DWords, they are getting redeclared as Integers.
- When the parameter has the same name as the type, 'Find local Reference' and 'rename parameter' fail
- Rename refactoring fails on overloaded procedures with at least one parameter of the same name.
- After using ExtractMethod, dragging a code snippet from the Tools palette causes a stack overflow.
- Rename for components with event handlers doesn't work for VCL and VCL.Net apps after you save the project

# Refactoring with D10



Delphi 2006 provides following new refactoring operations:

- **New!** Introduce Variable refactoring d D #
- **New!** Introduce Field refactoring d D #
- **New!** Inline Variable refactoring d D #
- **New!** Change Parameters refactoring d D #
- **New!** Safe Delete refactoring d D #
- **New!** Push Members Up / Down refactoring d D #
- **New!** Pull Members Up refactoring d D #
- **New!** Extract Superclass refactoring d D #
- **New!** Extract Interface refactoring d D #
- **New!** Move Members refactoring d D #
- Examples...

# Optimizations



Some tips and hints to optimize your code:

1. one statement and short var
2. assert call, IfThen()
3. use snippets
4. naming convention
5. \$IF-directive
6. Const instead of var parameters
7. extract code from dfm
8. comment your code
9. uses list



# Optimization – one and var

- one statement and with speed

An easy one states that only one statement should exist for every source line, like next and put a var lifetime as short as possible

```
begin ..
```

```
  I := 5; CallProc(I); //bad
```

```
//better
```

```
begin ..
```

```
  I := 5;
```

```
  CallProc(I);
```

```
var Rec: TMyRec;
```

```
begin ..
```

```
  with Rec do begin
```

```
    .. title := 'Hello Mars';
```

```
  end;
```

```
end;
```

# Optimization – assert and IfThen()



## assert call

Use assert calls as much, but don't forget the `$C` - setting is active or not. This means that identifiers used in the Assert procedure call, will be registered in your tests, but when compiled by Delphi, this code line will be (should) stripped out if `$C-` is defined.

```
procedure MyProc(p: pointer);  
begin
```

```
    Assert(p <> NIL, “”);
```

```
nMax:= IfThen(nA > nB, nA, nB) // no if/then/else
```

# Optimization – use code snippets



```
Procedure CopyRecord(const SourceTable, DestTable : TTable);
var
  i: Word;
begin
  DestTable.Append;
  For i:= 0 to SourceTable.FieldCount - 1 do
    DestTable.Fields[i].Assign(SourceTable.Fields[i]);
  DestTable.Post;
end;
```

# Optimization – naming convention



TBitBtn;bitn  
TButton;btn  
TCheckBox;chk  
TComboBox;cbo  
TRadioButton;rb  
TRadioGroup;rg .....

Ordinary types start with "T"

Exception types start with "E"

Pointer types start with "P"

Interface types start with "I"

Class fields by properties (read/write) start with "F"

Method pointers start with "On/Before/After"

# Optimization - \$IF-directive and const



In Delphi 6, the new \$IF-directive was introduced. The \$IF-directive is followed by an expression, that evaluates to TRUE or FALSE. Differentiate your code!

Avoid var parameters that are used, but never set in the subprogram they belong to. You may omit the var keyword, or change it to a const parameter.

Declare with the const directive, resulting in better performance since the compiler can assume that the parameter will not be changed.

```
procedure MyHexMaxProc(const i : integer); //not var
begin ..
  if i = 5 then // !! I is not set
    begin .. end;
end;
```

by the way: use not more than 3 parameters in a method, so the compiler can build it with fastcall directives, means registers are used instead of stack!

# Optimization - extract code from dfm



- Connection:= TSQLConnection.Create(NIL);
- with Connection do begin
- ConnectionName:= 'VCLScanner';
- DriverName:= 'INTERBASE';
- LibraryName:= 'dbexpint.dll';
- VendorLib:= 'GDS32.DLL';
- GetDriverFunc:= 'getSQLDriverINTERBASE';
- Params.Add('User\_Name=SYSDBA');
- Params.Add('Password=masterkey');
- with TWebModule1.create(NIL) do begin
- getFile\_DataBasePath;
- Params.Add(dbPath);
- end

# Optimization - comment your code



1. Connection := TSQLConnection.Create(nil);
2. with Connection do begin ...
3. //after connection, create dataset to it
4. DataSet := TSQLDataSet.Create(nil);
5. with DataSet do begin ...
6. SqlConnection := Connection;
7. //build the command string
8. CommandText :=
9. Format('insert into KINGS values("%d", "%s", "%s", "%s", "%s", "%s")', [10, Email, FName, LName, logdate]);

# Optimization – uses list



We know the linker is a smart one, but nevertheless init and finals are executed. Removing unused uses references has multiple benefits:

- less code to maintain, no need to bother about code that's not used
- code from initialization and finalization sections in unused units is not linked in and run, reducing the size of the EXE
- compilation runs smoother and quicker

When you drop a VCL component on a Delphi form the Delphi IDE automatically adds the unit or units required by the component to the interface section uses statement. This is done to ensure that the form file (DFM/XFM) can locate the code needed to stream the form and components. Even if you later remove the component, the units are not deleted from the uses statement.

# Expand your Refactoring



**You must refactor to make it compatible with Unit Testing or you must create a Unit Test Class before refactor !!**

**Refactoring in UML Diagrams (e. g. state -> superstate)**

**[Code Patterns]**

**[Syncedit]**

**[Macro recorder]**

**Tools practice:**

**„As far as I'm concerned, Castalia or Pascal Analyzer is a must have for any Delphi developer - I know I'd be lost without it now.“**

# Refactoring Links:



- Delphi 9...\BDS\3.0\Help.pdf (1656 p.) ab S. 85
- Delphi 9 Tools: <http://www.modelmakertools.com/>
- Report Pascal Analyzer:  
[http://www.softwareschule.ch/download/pascal\\_analyzer.pdf](http://www.softwareschule.ch/download/pascal_analyzer.pdf)
- *Refactoring* Martin Fowler (1999, Addison-Wesley)
- Changing the structure of code without changing the functionality
- <http://www.refactoring.com>
- Discussion site on code smells  
<http://c2.com/cgi/wiki?CodeSmell>
- Castalia3 - <http://www.delphi-expert.com/castalia2/>
- [http://www.informatics.sussex.ac.uk/courses/sde/Course-Outline/Lecture\\_slides/15/slides4.pdf](http://www.informatics.sussex.ac.uk/courses/sde/Course-Outline/Lecture_slides/15/slides4.pdf)



Q&A  
max@kleiner.com

