Anton Vogelaar, Noud van Kruysbergen

Met klasse

Programmeercursus Pascal, deel 3: classes

Tot nu toe heb je in deze cursus kennis kunnen maken met de basisprincipes van het programmeren in Pascal en hoe je fouten in het programmaverloop kunt achterhalen en oplossen. In dit deel gaan we het hebben over classes, die je kunt gebruiken om sneller te programmeren.

Voor veel beginnende programmeurs zijn classes een vloek. Voor ervaren programmeurs zijn ze echter een zegen. Er hangt om classes een zweem van mysterie. Wat is het, wat kun je er mee, wat gebeurt er in een klasse? De korte versie: het is een snelle manier om code te hergebruiken. Maar laten we bij het begin beginnen.

Je hebt wel gemerkt dat programmeren een beetje lijkt op het oplossen van een puzzel. Je denkt al redelijk snel een oplossing te weten en gaat daar snel een programma voor maken [1]. Bij de eerste tests van het programma blijken een aantal dingen dan toch niet aan de gestelde eisen te voldoen en anders te reageren dan je had verwacht. Dan wordt het tijd om de code te gaan debuggen, zoals we in het tweede deel van deze cursus hebben omschreven [2]. Daarbij worden steeds kleine verbeteringen doorgevoerd tot het programma uiteindelijk voldoet aan je verwachtingen.

Als je dan een goed werkend programma hebt, heb je ook code geschreven die later wellicht weer bruikbaar zal zijn voor een volgend gelijksoortig probleem, wat je dan sneller kunt oplossen door deze code te hergebruiken. Een van de technieken die daarbij gebruikt wordt, is het gebruik van classes.

Hergebruik

Programmeren blijkt vaak een tijdrovende bezigheid, met name doordat het debuggen vaak drie tot vijf maal langer duurt dan het schrijven van de eerste versie van het programma. Door het programma op te delen in logische blokken en deze apart te verpakken en te documenteren, kunnen die worden hergebruikt zonder dat je ze nog hoeft te debuggen. Dat scheelt een heleboel tijd.

Als het om een algemeen bruikbare functionaliteit gaat, kun je die zelfs te koop aanbieden. Andersom kun je zelf ook functionaliteiten inkopen om een applicatie sneller te kunnen bouwen. Delphi en Lazarus worden een RAD-omgeving (Rapid Application Development) genoemd omdat er bij het installeren al zeer veel standaard classes worden meegeleverd. Denk maar aan de in het eerste deel van deze cursus gebruikte classes TForm en TButton, die te gebruiken zijn zonder ook maar een regel te programmeren. In dit deel van de cursus gaan we dieper in op structuren als types, arrays, records, classes en componenten.

Template

Om de werking van deze structuren te demonstreren, gebruiken we een template-applicatie die bestaat uit een Form, een Button en een Memo-object. Door op de button te drukken, verschijnt het resultaat in het memoveld. De broncode bij dit artikel kun je downloaden via de softlink, waar ook een torrent-bestand staat voor de gratis ontwikkelomgeving Lazarus.

We beginnen met het aanmaken van een template, zoals we dat in het eerste deel uitvoeriger hebben beschreven. Start Lazarus, ga naar 'Bestand / Nieuw' en klik in het verschijnende venster op 'Applicatie' en 'OK'. Selecteer het formulier Form1 en zet via de werkbalk 'Standard' een TButton-component en een TMemo-component op het formulier. Dubbelklik op de knop Button1 zodat Lazarus daar automatisch een OnClick-event voor implementeert. In de Broncode Bewerker komt de cursor meteen op de goede plek te staan. Type daar de volgende twee regels in:

```
Memo1.Clear;
Memo1.Lines.Add ('Hallo wereld');
```

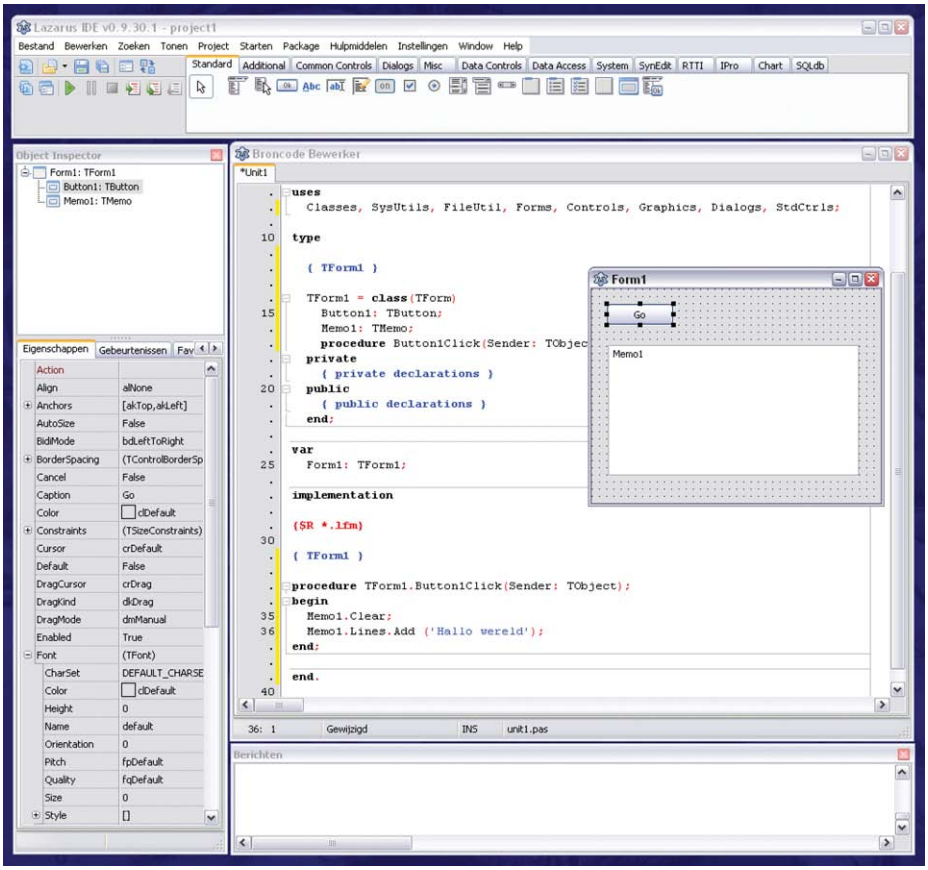
Het memoveld heeft automatisch de naam Memo1 gekregen, die we hier gebruiken. Deze regels zorgen er respectievelijk voor dat het memoveld leeg gemaakt wordt en dat daar vervolgens de tekst Hallo wereld in komt te staan. Ga links naar het tabblad 'Eigenschappen' en klik op 'Caption'. Vervang de standaardtekst Button1 dan door Go en druk op de Enter-toets. Compileer en run de applicatie vervolgens door op F9 te drukken.

Als het programma start, klik je op de Go-knop en verschijnt de tekst 'Hallo wereld' inderdaad in het memoveld. Je kunt het programma afsluiten door op het rode kruisje rechtsboven te klikken. Deze template blijven we in dit artikel gebruiken om de verschillende datatypen te illustreren.

Datatypen

Het moge bekend zijn dat data in een computer in binaire vorm opgeslagen worden en niet meer dan een enen en nullen zijn, bijvoorbeeld 11110000. Bij 32-bit besturingssysteem is het kleinste element 32-bits groot, oftewel 4 bytes. Bij een 64-bit systeem is dat 8 bytes. Hoe de exacte inhoud van die bits en bytes geïnterpreteerd moet worden, wordt bepaald door de afspraken die daarover gemaakt zijn, oftewel de typedefinities. Een aantal simpele types zijn in Lazarus al vastgelegd, waaronder de types Boolean, Integer, Double en String. De werking van deze simpele types wordt gedemonstreerd door in de template-applicatie de code van Button1Click als volgt te wijzigen.

```
Procedure TForm1.Button1Click (Sender : TObject);
Var I : Integer;
```



De eerste voorbeeldapplicatie in deze aflevering van de Pascal-cursus doet niets anders dan een tekst in een memoveld zetten.

```
    Function IntToBin (X : Integer) : String;
    (* Convert X as 16-bit Integer into a binary string *)
    Var Mask : Integer;
Begin
    Result := ''; Mask := %1000000000000000;
    Repeat
        If Mask And X <> 0 Then Result := Result + '1'
                           Else Result := Result + '0';
        Mask := Mask Shr 1;
    Until Mask = 0;
    End;
Begin
    I := 32 + 4 + 2;
    Memo1.Clear;
    Memo1.Lines.Add ('Een Integer is ' + IntToStr (SizeOf (I))
                                      + ' bytes groot');
    Memo1.Lines.Add ('De binaire inhoud van I is ' +
                                      IntToBin (I));
    Memo1.Lines.Add ('De waarde van integer I is ' +
                                      IntToStr (I));
End;
```

Het resultaat van dit programma laat zien hoeveel bytes een integer in Lazarus groot is, wat de binaire inhoud van integer I is en wat zijn waarde is. Conversieopdrachten als IntToStr en IntToBin herken je ongetwijfeld uit het tweede deel over het debuggen van je programma.

Eigen datatypen

Om de leesbaarheid van programma's te verbeteren en om te voorkomen dat bewerkingen tussen verschillende types kunnen worden uitgevoerd – te vergelijken met het optellen van 4 appels en 3 peren – declareren we onze eigen typen, de zogeheten user-types. In het onderstaande voorbeeld maken we een user-type aan voor kaartkleuren schoppen, harten, ruiten en klaveren. We declareren een variabele Kaart die een TKaart bevat en een Hand die een aantal van dergelijke kaarten bevat. De opgave is om de kaartkleuren in Kaart en Hand naar het memoveld te schrijven. Om de werking van de user-types te zien, wijzigen we de bovenstaande template-applicatie door het volgende:

```
Procedure TForm1.Button1Click (Sender : TObject);
(* Voeg TypInfo to aan Uses clause *)
Type    TKaart = (Schoppen, Harten, Ruiten, Klaveren);
Var     Kaart : TKaart;
        Hand : Set Of TKaart;
Begin
    Kaart := Klaveren;
    Hand := [Harten, Schoppen]; Hand := Hand + [Ruiten];
    Memo1.Clear;
    Memo1.Lines.Add ('Kaart bevat het ' + IntToStr
                (Integer (Kaart)) + ' e element van TKaart');
    Memo1.Lines.Add ('Kaart is ' + GetEnumName ( TypeInfo
                (TKaart), Integer (Kaart)));
    Memo1.Lines.Add ('Hand bevat de kaarten : ');
    If Schoppen
            In Hand Then Memo1.Lines.Add ('   Schoppen');
    If Harten In Hand Then Memo1.Lines.Add ('   Harten');
    If Ruiten In Hand Then Memo1.Lines.Add ('   Ruiten');
    If Klaveren In Hand Then Memo1.Lines.Add
                                    ('   Klaveren');
End;
```

Zoals op de tweede regel van de code staat, moet je TypInfo toevoegen aan de gebruikte bibliotheken. Als het programma loopt, moet de naam van het TKaart-element worden opgezocht waar Kaart naar verwijst. Kaart bevat namelijk het getal 3 omdat de waarde Klaveren het vierde element van TKaart is – begin met tellen vanaf 0. De slimme compiler heeft er bij ons 32-bit Windows-systeem een enkel 32-bit element van gemaakt. Het terugzoeken van de waarde 3 naar Klaveren gebeurt door de functie GetEnumName die in de bibliotheek TypInfo staat. Daarom moet bovenaan de Unit achter Uses ook de bibliotheek TypInfo worden opgenomen:

```
Uses Classes, SysUtils, ..., StdCtrls, TypInfo;
```

Als je de applicatie laat runnen en op de Go-knop klinkt, krijg je te zien om welke kaart het gaat en welke kaarten er in de hand zitten.

Arrays en records

Een programma wordt overzichtelijker als je elementen kunt groeperen. Elementen van het zelfde type kun je samenvoegen in een array. Zo betekent

```
Var Waarde1, Waarde2, Waarde3 : Integer;
```

hetzelfde als

```
Var Waarden : Array [1..3] Of Integer;
```

Het resultaat is niet overweldigend, maar doet gewoon wat het doen moet en kan uitgebreid worden.

Een ander voorbeeld van de mogelijkheden: laat een getal in zijn binaire vorm zien.

Je kunt ook eigen datatypes maken. Hier zijn de kaartkleuren schoppen, harten, ruiten en klaveren in één type samengevoegd en met een index te benaderen.

Dit wordt gedemonstreerd in de onderstaande applicatie:

```
Procedure TForm1.Button1Click (Sender : TObject);
Var Waarde1, Waarde2, Waarde3  : Integer;
    Waarden                    : Array [1..3] Of Integer;
    I                          : Integer;
    S                          : String;
Begin
    Waarde1     := 17; Waarde2 := 25; Waarde3 := 49;
    Waarden [1] := 17;
    Waarden [2] := Waarde1 + 1;
    Waarden [3] := Waarde2 + Waarde3;
    Memo1.Clear;
    Memo1.Lines.Add (Format ('Waarde1 is %d', [Waarde1]));
    Memo1.Lines.Add (Format ('Waarde2 is %d', [Waarde2]));
    Memo1.Lines.Add (Format ('Waarde3 is %d', [Waarde3]));
    S := '';
    For I := 1 To Length (Waarden) Do S := S + IntToStr
                                        (Waarden [I]) + ' ';
    Memo1.Lines.Add (Format ('Waarden bevat %s', [S]));
End;
```

In de uitvoer van dit programma zie je achtereenvolgens de waarden voor de variabelen Waarde1 tot en met Waarde3 en welke drie getallen er in de array Waarden zitten.

Bij de uitvoer gebruiken we hier de opdracht Format uit de SysUtils-bibliotheek. Daarmee kun je de uitvoer bijvoorbeeld beter uitlijnen. Dat heeft hier nog niet zoveel voordelen, maar in het volgende voorbeeld wordt de uitwerking daarvan duidelijker.

Een array wordt dus gebruikt voor variabelen van hetzelfde type. Een record is een datatype dat je daarentegen voor het opslaan van verschillende typen kunt gebruiken. Omdat een array zelf ook een type is, kan een record ook arrays bevatten.

We gaan aan de slag met het volgende voorbeeld: we hebben een tandem met twee fietsers en een passagier op de bagagedrager. Van iedere reiziger willen we de naam, de leeftijd en het gewicht bewaren. Om de applicatie overzichtelijk te houden, zetten we al deze gegevens in een record van het type TTandem. De gegevens van de afzonderlijke reizigers bewaren we in een record van het type TPersoon. Vervolgens willen we al deze gegevens in kolommen laten weergeven.

Voor het weergeven in kolommen willen we voor het memoveld een lettertype hebben waarvan alle letters even breed zijn. Dat doe je bij de Object Inspector, waar je op het Eigenschappen-tabblad van Memo1 bij 'Font' een andere lettertype kiest dan het standaard ingesteld (TFont). In ons geval gaan we voor het lettertype Courier New en stellen we de lettergrootte in op 9 in plaats van de standaard 10, maar dat kan afhankelijk van je besturingssysteem en de geïnstalleerde fonts een ander lettertype of –grootte zijn. Vervolgens moet je de inhoud van de Button1Click-event in de template-applicatie als volgt aanpassen:

```
Procedure TForm1.Button1Click (Sender : TObject);
Type    TPersoon = Record
                     Naam      : String;
                     Leeftijd  : Integer;
                     Gewicht   : Double;
                   End;
        TTandem = Array [1..3] Of TPersoon;
Var     Tandem  : TTandem;
Begin
    Tandem [1].Naam := 'Piet'; Tandem [1].Leeftijd := 24;
    Tandem [1].Gewicht := 75.8;
    Tandem [2].Naam := 'Marie'; Tandem [2].Leeftijd := 26;
    Tandem [2].Gewicht := 55.3;
    With Tandem [3] Do
    Begin
        Naam := 'Jan'; Leeftijd := 45; Gewicht := 147.5;
    End;
    Memo1.Clear;
    Memo1.Lines.Add ('Op de tandem zitten:');
    Memo1.Lines.Add (Format ('%-10s %-8s %-8s %-8s',
                     ['', 'Persoon1', 'Persoon2', 'Persoon3']));
    Memo1.Lines.Add (Format ('%-10s %-8s %-8s %-8s',
                ['Naam', Tandem [1].Naam, Tandem [2].Naam,
                                          Tandem [3].Naam]));
    Memo1.Lines.Add (Format ('%-10s %-8d %-8d %-8d',
            ['Leeftijd', Tandem [1].Leeftijd, Tandem [2].Leeftijd,
                                          Tandem [3].Leeftijd]));
    Memo1.Lines.Add (Format ('%-10s %-8.1f %-8.1f %-8.1f',
          ['Gewicht', Tandem [1].Gewicht, Tandem [2].Gewicht,
                                          Tandem [3].Gewicht]));
End;
```

In de uitvoer van het programma is dan te zien wie er op de tandem zitten en hoe oud ze zijn en wat ze wegen. Op zich niet verrassend, omdat dat precies de data zijn die je hard in de code hebt ingevoerd. Voor het invoeren van de eerste twee personen hebben we de velden helemaal uitgeschreven in de vorm van Tandem [1].Naam, maar bij de derde tandem hebben we een andere constructie gebruikt waarbij alleen rechtstreeks het betreffende recordveld Naam hoeft te worden gebruikt. Bij uitgebreidere records kan dat aardig schelen in het typewerk en de overzichtelijkheid.

In dit voorbeeld gebruiken we het Format-commando om de resultaten overzichtelijk weer te geven. Het aantal mogelijkheden is dusdanig groot dat we daar hier niet verder op ingaan, maar bij de helpbestanden van Lazarus en Free Pascal is daar genoeg over te vinden.

Records met procedures

De complexiteit van de gebruikte datatypes wordt steeds groter. Om de code nog beter te hergebruiken, kun je de functies en procedures die betrekking hebben op de data van een record ook in het record zelf zetten. In dit geval doen we dat bijvoorbeeld met de functies TotaalGewicht en GemiddeldeLeeftijd. Als we het record TTandem in een bibliotheek bewaren, is bij later hergebruik alle functionaliteit beschikbaar. Om functies in records te kunnen gebruiken, moet de Lazarus-compiler in de Delphi-modus worden gezet. Dat gebeurt op regel 2 in de broncode:

```
{$mode delphi}{$H+}
```

We verdelen het voorbeeldprogramma in drie stukken. Het eerste deel is de typedeclaratie waarin TPersoon en TTandem worden gedeclareerd.

```
Implementation
{$R *.lfm}
Type    TPersoon  = Record
                      Naam     : String;
                      Leeftijd : Integer;
                      Gewicht  : Double;
                    End;
        TTandem   = Record
                      Persons  : Array [1..3] Of TPersoon;
                      Function GemiddeldeLeeftijd : Integer;
                      Function TotaalGewicht      : Double;
                    End;
Var     Tandem    : TTandem;
```

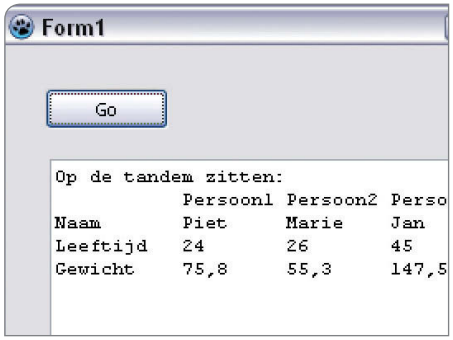
In het tweede deel worden de procedures en functies van het record, methods genoemd, geïmplementeerd. Iedere record-method begint met de naam van het record gevolgd door een punt en de method-naam.

```
Function TTandem.GemiddeldeLeeftijd : Integer;
Begin
    Result := (Persons [1].Leeftijd + Persons [2].Leeftijd +
                                      Persons [3].Leeftijd) Div 3;
End;

Function TTandem.TotaalGewicht : Double;
Begin
    Result := Persons [1].Gewicht + Persons [2].Gewicht +
                                    Persons [3].Gewicht;
End;
```

In het derde en laatste deel worden de methods van de GUI (grafische interface) geïmplementeerd. In ons geval is dat het object TForm1 met daarin het OnButtonClick-event van Button1. In

Als de data-elementen geen afzonderlijke namen krijgen maar in een array komen te staan, merk je daar in de uitvoer niets van. Met arrays zijn veel makkelijker bewerkingen op meerdere getallen uit te voeren.



Deze uitvoer is gemaakt met records van persoonsgegevens.

deze method wordt het record Tandem geladen met data en het Memo-object gevuld met gegevens uit de record-methods GemiddeldeLeeftijd en TotaalGewicht. Om de leesbaarheid van de code te bevorderen, hebben de methods begrijpelijke namen en worden ze uit leesbaarheidsoverwegingen geschreven in het zogeheten kamelenschrift, waarbij ieder woord begint met een hoofdletter.

```
Procedure TForm1.Button1Click (Sender : TObject);
Begin
    Memo1.Clear;
    (* definieer eerst de namen, leeftijden en gewichten! *)
    Memo1.Lines.Add ('Statistische gegevens van Tandem
                                                zijn:');
    Memo1.Lines.Add (Format ('De gemiddelde leeftijd is :
                        %d jaar', [Tandem.GemiddeldeLeeftijd]));
    Memo1.Lines.Add (Format ('Het totale gewicht :
                            %.1f kg', [Tandem.TotaalGewicht]));
End;
```

In de uitvoer staat dan de gemiddelde leeftijd van de reizigers en hun totale gewicht.

Om functies in procedures te kunnen gebruiken, moet je wel Free Pascal 2.6.0 hebben, de oudere versie 2.4.4 kan dat nog niet. Om te controleren met welke versie je werkt, ga je naar 'Instellingen / Options', waar bij 'Compilerpad' iets als '…\fpc\2.6.0\bin\i386-win32\fpc.exe' moet staan. Staat daar 2.4.2 in plaats van 2.6.0, dan moet je Free Pascal eerst updaten. Hoe je dat doet, staat bij de softlink.

Classes

Dan zijn we in de opbouw van dit artikel nu aanbeland bij waar het eigenlijk om begonnen was: de classes. We hebben tot nu toe gezien dat je met records en hun methods heel veel kunt doen en dat dit redelijk makkelijk werkt. Het voorbeeld ging over één tandem, maar dat zouden we ook kunnen uitbreiden naar voor het beheer van een toertocht met 100 tandems. Dan moet je beginnen met

```
Var Tandems : Array [1..100] Of TTandem;
```

Daarmee worden er 100 records in het geheugen gezet. Voor het datadeel is dat prima omdat alle records ook verschillende data bevatten. Maar ook de methods GemiddeldeLeeftijd en TotaalGewicht worden elk 100 maal in het geheugen opgeslagen, en dat zijn dan niet minder dan telkens 99 kopieën van hetzelfde. Dat moet beter kunnen door de code maar één

Door functies binnen records te gebruiken, heb je alle functionaliteiten altijd ter beschikking. Met classes gaat dat nog efficiënter.

Cursus Pascal

Het doel van deze minicursus is dat je de basisbeginselen leert van het programmeren in Pascal met de ontwikkelomgeving Lazarus en dat je een aantal eenvoudige programma's hebt kunnen maken met behulp van componenten.

Deze cursus is geschreven in samenwerking met de Stichting Ondersteuning Programmeertaal Pascal.

keer in het geheugen te zetten en dan door alle records te laten gebruiken. Dat is nou net het concept van classes. Een class bevat de structuur voor de variabelen en de code voor de methods. Van deze class worden dan klonen gemaakt, die objecten worden genoemd. Ieder object heeft dan ruimte voor de variabelen en pointers die naar de methods in de class verwijzen. Een kloon wordt in het geheugen gemaakt met de method Create en weer uit het geheugen verwijderd met de method Free of Destroy.

Een ander verschil tussen records enerzijds en classes en objects anderzijds is dat records gebruik maken van de stack, oftewel het beperkte geheugen dat aan een applicatie is toegewezen, terwijl classes de heap gebruiken. De heap is dynamisch geheugen dat door het besturingssysteem bij iedere aanroep van de method Create aan de applicatie beschikbaar wordt gesteld.

Om het voorbeeldprogramma dat met records werkt om te zetten naar een versie met classes, zijn slechts minimale aanpassingen nodig: verander in de typedefinitie van TTandem het woord Record in Class:

```
Type    TPersoon  = Record
                      Naam     : String;
                      Leeftijd : Integer;
                      Gewicht  : Double;
                    End;
        TTandem   = Class
                      Persons  : Array [1..3] Of TPersoon;
                      Function GemiddeldeLeeftijd : Integer;
                      Function TotaalGewicht      : Double;
                    End;
Var     Tandem    : TTandem;
```

en kloon bij de OnClick-event van Button1 de class TTandem met Tandem := TTandem.Create. De variabele Tandem verwijst nu als pointer naar een object dat gekloond is van TTandem. Het commando Tandem.Free geeft het gereserveerde geheugen terug aan het besturingssysteem.

```
Procedure TForm1.Button1Click (Sender : TObject);
Begin
    Tandem := TTandem.Create;
    With Tandem.Persons [1] Do
    Begin
        Naam := 'Piet'; Leeftijd := 24; Gewicht := 75.8;
    End;
    ...
    Memo1.Clear;
    (* definieer eerst de namen, leeftijden en gewichten! *)
    Memo1.Lines.Add ('Statistische gegevens van Tandem
                                                zijn:');
    Memo1.Lines.Add (Format ('De gemiddelde leeftijd is :
                        %d jaar', [Tandem.GemiddeldeLeeftijd]));
    Memo1.Lines.Add (Format ('Het totale gewicht :
                            %.1f kg', [Tandem.TotaalGewicht]));
    Tandem.Free;
End;
```

De uitvoer van dit programma is precies hetzelfde als bij het gebruik van records in plaats van classes. Met classes wordt je programma dus overzichtelijker en gebruikt het minder geheugen. Bij het uitvoeren van de functies moet wel steeds naar de betreffende functie in de class worden verwezen om vervolgens te laten uitvoeren, dus dat duurt dan wat langer. Afhankelijk van het doel van een applicatie kun je als programmeur dan ook kiezen voor een snel maar groter programma (met records) of een compact maar wat trager programma (met classes).

Componenten

Nu je weet wat classes zijn, wil je die ook gaan (her)gebruiken natuurlijk. Binnen Lazarus kan dat grafisch via de standaard Componentenbalk of als opsomming via 'Tonen / Componenten'. Dit zijn classes waaraan een pictogram gekoppeld is. Deze pictogrammen kunnen op een Form of DataModule worden gezet, waardoor er van deze classes nog sneller een kloon kan worden gemaakt. Zeker voor visuele componenten als TButton en TMemo is dit erg handig omdat je op deze manier met de muis de GUI in elkaar kunt knutselen en eigenschappen als Top, Left, Width en Height automatisch worden ingesteld. Dus zelfs bij het eerste deel van deze cursus heb je onbewust al gebruikt gemaakt van classes!

In het volgende deel van deze cursus gaan we kijken naar de multimediale mogelijkheden van de programmeertaal Pascal en de ontwikkelomgeving Lazarus. (nkr)

Softlink 1206138 c't

Literatuur

[1] Detlef Overbeek, Noud van Kruysbergen, "Hallo Wereld!", Programmeercursus Pascal, deel 1: de basis, c't 4/2012, p.102
[2] Siegfried Zuhr, Noud van Kruysbergen, Zoek de vout, Programmeercursus Pascal, deel 2: het debuggen, c't 5/2012, p.138