Om den här pappersbunten

Redan när jag började planera för en Delphi-kurs hade jag en tanke på att jag skulle ställa samman någon typ av papper som gav lite grundläggande information om hur man använder Delphi. Då det visade sig att intresset för kursen var betydligt mycket större än vad som fick plats bestämde jag mig för att försöka göra så att det pappret blev läsbart på egen hand. Jag är inte helt nöjd med resultatet, men allting är bättre än inget :-) Kompendiet behandlar det mest grundläggande när det gäller Delphi, dvs forms, units och programkomponenter. Dessutom innehåller det lite om grundläggande databashantering. Därutöver ges även en del tips på saker som kanske inte tillhör det mest grundläggande men som är bra att kunna, tex drag'n drop.
Observera att allt det som står här gäller för Delphi, det är möjligt (om än inte speciellt troligt) att Delphi2 (som jag i skrivandets stund bara har hunnit kasta ett hastigt öga på) gör vissa saker lite annorlunda. Jag förutsätter att läsarna har tillgång till NT-konton på DSV och är kapabla att hitta till **mickey/kurser/Delphi-Kv/** där det finns ett antal exempel upplagda.
Ni kommer säkert att hitta otydligheter, troligen också felaktigheter, jag beklagar detta, och skyller naturligtvis på tidsbrist. Om ni har frågor, förslag på förbättringar, felrapporter etc så ser jag gärna att ni skickar mig ett email, adressen är **h-bergst@dsv.su.se**

Henrik Bergström 1997

Allmänt

Innan vi kommer in på hur man faktiskt programmerar i Delphi är det dags för lite allmänt tröttsamt tugg som alla ändå kommer att strunta i (tro mig, jag struntar i det själv mestadels :-)

Läsbarhet

Delphi-program som man har hållit på med ett tag, har lätt en tendens att bli både röriga och svårlästa. De främsta anledningarna som jag ser är den "ad hoc"-programmering man vanligtvis tvingas till p.g.a. tidsbrist (samt det faktum att Delphi inte sorterar de procedurer som skapas utan bara lägger dom i den ordning de skapas). Det finns dock några saker man kan göra för att förbättra läsbarheten radikalt, dom borde vara välkända, men vi tar det väl en gång till:

- Namngivningskonventioner, bestäm er för ett sätt att namnge klasser, typer, variabler, units etc. Och viktigast av allt, HÅLL er till dem. En som ni snabbt kommer att lägga märke till är att i Delphi heter alla klasser och typer någonting på T, tex TEditBox, TColor etc.
- Kommentarer, använd kommentarer, det gör det hela så mycket enklare, både för er själva och för andra.
- Kodindragning, använd också kodindragning. Varje gång ni går in i en loop e.dyl., tryck en gång på tab. Det ökar läsbarheten något otroligt.

Dvs, det ska se ut något sånt här:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
   {En kommentar som berättar vad proceduren gör}
   indragen kod
end;
```

Tips

En sak till innan vi går över till att beskriva hur man egentligen programmerar i Delphi. Lägg **ALLTID** era nya projekt i ett nytt bibliotek. Default-biblioteket innehåller vanligen redan från början rätt många filer och att försöka hitta ett större projekt där för att lägga det någon annanstans eller föra över det till en diskett är INTE kul.

Units och programkomponenter

Units

Ett program i Delphi består av en eller flera units eller enheter. Dessa innehåller i sin tur de olika programkomponenter som bygger upp själva programmet, samt de procedurer som "styr" dessa

komponenter. Rätt använda gör units att det blir lättare för flera personer att jobba på samma projekt. För att detta ska fungera i praktiken krävs dock att man väljer namnen på allting på ett bra sätt. Kom ihåg att Pascal (och därmed även Delphi) inte tillåter NÅGRA namnkonflikter. De vanligaste problemen i början är flera formulär/units med samma namn (tex två st Form1) eller formulär och units som får samma namn. Ett enkelt sätt att komma tillrätta med det senare problemet är tex att döpa alla formulär till något som slutar på Form.

Anatomin hos en Unit

En unit består av rätt många delar, med undantag för rubriken är alla frivilliga.

```
Unit Unit1;                 {Rubrik, namnet på Uniten}

interface                   {Här börjar deklarationen av enhetens interface som består av flera delar}
uses unit2, unit3;          {denna enhet använder sig av två andra}
type
 mintyp=(nr1,nr2);          {Denna typ är en del av enhetens interface och alltså tillgänglig utåt}
var
 n:integer;                 {dito för denna variabel}

 function Hej:string;       {dito för denna funktion}

implementation              {Här börjar själva implementeringen}
 function Hej:string;       {Här är implementeringen av funktionen Hej som deklarerades ovan}
 begin
   Hej:='Hejsan';
 end;

initialization              {Här läggs allt som ska göras innan man kan börja använda enheten}
 n:=10;                     {Denna kod kommer enbart att exekveras EN gång, första gången enheten
                             anropas på något sätt}

end.                        {Obs .}
```

Den vanligaste formen av units är den som innehåller ett formulär (form) och dess komponenter. Formuläret kan ses som en av de där plattorna man använde att bygga lego på.

Tips

Så fort ni har skapat en ny unit, spara. Det namn ni då ger filen kommer då automatiskt att ges till den nya enheten. Naturligtvis sker detta även om ni sparar senare, men projektet kan då direkt hitta er nya unit, och ni blir direkt uppmärksamma på om några namnkonflikter har uppstått.

Programkomponenter

Om ett formulär är bottenplattan du ska bygga upp ditt program på, är programkomponenterna de legobitar-du bygger upp det av. Om du vill ha en ruta där användaren kan mata in text placerar du bara ut en sådan (en EditBox ) på formuläret. De olika komponenterna du har tillgång till ligger under ett fliksystem. För att få reda på mer om en komponent, klicka på den i fliksystemet, eller klicka på en som redan ligger på formuläret och tryck på F1.

Properties

Properties, eller egenskaper som dom skulle kunna heta på svenska är det som bestämmer hur en program-komponent ska se ut (och i viss mån uppföra sig). Det är genom att sätta olika egenskaper som ni bestämmer saker som hur stora komponenterna ska vara, vilka fonter som ska användas etc.
För att titta på en komponents egenskaper, eller ändra på dem klicka på komponenten på formuläret (även formuläret har naturligtvis egenskaper) och titta i ”Object Inspector”, det avlånga fönstret till vänster. Object Inspectorn har två flikar, dels ”Properties” och dels det vi ska berätta om nedan, ”Events”. För att få reda på mer om en viss egenskap, klicka på dess namn och tryck på F1.

Events

Events, eller händelser, är i stort sett allt ni gör i Windows. Om ni klickar på en knapp händer det någonting med den knappen, nämligen en OnClick-händelse. Nästan all programkod ni kommer att skriva, åtminstone i början, kommer att ligga under olika events. Events, precis som egenskaper (properties) finns under ”Object Inspector”.

Exempel

Nog teori, nu programmerar vi :-)

Skapa ett tomt formulär (om du inte redan har ett) och placera ut en EditBox  och en Button  på det.

Klicka på EditBoxen  på formuläret och sedan på egenskapen Text i ”Object Inspector”.
Radera texten som står där, den kommer då också att försvinna från själva EditBoxen.

Klicka på knappen  och sätt dess Caption-egenskap till något lämpligt, typ ”Klicka här”.
Dubbelklicka sedan på knappen. Ett textfönster med titeln UNIT1.PAS ska komma upp. Skriv av följande kod mellan begin och end

```
Form1.caption:=edit1.text;
```

Tryck på F9 för att köra programmet. Varje gång du trycker på knappen kommer den händelse som då inträffar (en OnClick-händelse) att köra den kod du just skrev och sätta titeln på formuläret till det som för tillfället står i textrutan.
Så enkelt är det att programmera Delphi.

Sammankoppling av formulär

För att visa ett formulär på skärmen använder du proceduren show (eller showmodal om du vill ha fönstret modalt). Dvs någonstans, tex under en knapps OnClick-händelse skriver du tex

```
Form1.show
```

Punktnotationen i kodsnutten ovan är något som kommer att användas ofta. Det koden ovan säger är att ta objektet Form1 (vårt formulär) och applicera proceduren show på den.
För att detta ska fungera krävs dock att den unit som det andra formuläret finns i finns inskriven under Uses-delen i den Unit där det första formuläret finns i. Exemplet nedan använder sig av två formulär Form1 och Form2 som ligger i Unit1 respektive Unit2.

```
unit Unit1;

interface

uses
  Unit2,       {Här har vi lagt till Unit2 så att man senare kan skriva tex Form2.Show}
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs;
```

Ibland vill man kunna hoppa fram och tillbaka mellan två formulär, för att det ska fungera måste man placera namnen på de Units där formulären ingår på lite olika platser. Om man bara skriver Unit1 respektive Unit2 under Uses i de båda enheterna (units) kommer man att drabbas av “Error 68: Circular unit reference (Unit1)”. Dvs, man drabbas av cirkelreferens. Lösningen är enkel men inte direkt uppenbar, man placerar bara minst en av dessa Uses Unit?; under raden **implementation** i aktuell modul. Då klarar Delphi av det hela. För att få lite ordning på detta förvirrande resonemang, titta på exemplet i biblioteket circ.

Kommunikation mellan Units

Det absolut enklaste sättet att lösa kommunikation mellan units (eller formulär) i början är naturligtvis globala variabler. För att få tillgång till sådana skapar ni helt enkelt en ny Unit som ni lämpligen döper till just Global. Sedan är det bara att deklarera de variabler du vill ha tillgång till i Global, samt naturligtvis

skriva till Global under Uses i alla Units du vill ska ha tillgång till de globala data. Kom dock ihåg att alla de vanliga invändningarna mot användande av globala variabler gäller även i Delphi.
Ett mer praktiskt sätt är dock att sätta de variabler du behöver ha som parametrar till en Unit under rubriken **var** i dess **interface**-del.

```
unit Unit2;

interface

var
  test:string;          {Test ingår i Unit2s interface och kan kommas åt utifrån med hjälp av
                         unit2.test}
```

Vill man ha lite mer kontroll kan man naturligtvis använda sig av speciella gränssnittsprocedurer.

Filerna i ett Delphi-projekt

Ett Delphi-projekt består av många filer, följande är de vanligaste filnamnsändelserna och en kort beskrivning av vad filen innnehåller.

Extension	Beskrivning
.dpr	Projektets källkods-fil
.dfm	En formulär-fil
.pas	Källkoden till en Unit
.opt	Projektets "options"-fil
.dsk	"Desktop status"-fil
.dcu	En kompilerad Unit
.res	En resurs-fil, tex en muspekare.

Delade events

Om man har flera komponenter som ska reagera på ungefär samma sätt är det onödigt att skriva in koden en gång för varje komponent. Risken är dessutom stor att man råkar ut för "klipp och klistra"-fel, eller att man glömmer att göra ändringar på alla ställen. I stället bör man använda sig av sk delade-händelser.

Hur man gör

För att skapa en delad händelse gör man så här:
1. Markera alla komponenter som ska vara med och dela på händelsen.
2. Dubbelklicka på händelsetypen i "Object Inspector"
3. Skriv händelsehanteraren.

Följande kod, som är saxad rakt ur Delphis hjälp tittar efter om det objekt (komponent) som instansierade Sender var Button1 och i så fall sätter den titeln (caption) på en dialogruta (AboutBox) till "About" + applikationens titel. Låt er inte luras av det faktum att proceduren heter Button1Click, Delphi ger även delade events namn efter den komponent som har det lägsta numret.
För den som undrar så är är "Application.Title" naturligtvis titeln på hela applikationen. Man kan ge sin applikation en titel genom att välja Options|Project och därefter Application. Om man inte har satt någon titel så används projektets namn, dvs namnet på filen med extensionen .DPR.

```
procedure TMain1.Button1Click(Sender: TObject);
begin
if Sender = Button1 then
  AboutBox.Caption := 'About ' + Application.Title
else AboutBox.Caption := ;
AboutBox.ShowModal;
end;
```

Följande kod är associerad till ett antal editboxar och ändrar titeln (caption) på formuläret till det som står i den editbox du för tillfället ändrar texten (Edit1Change) i. Anledningen till typkonverteringen är att Sender är ett objekt av typen TObject, en abstrakt basklass som vanligen används för att bygga upp nya klasser. Denna klass har ingen text associerad med sig, men eftersom vi vet att Sender bara kommer att instansieras med editboxar, som ju har ett textfält, så kan vi lugnt göra den konverteringen.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
   form1.caption:=TEdit(Sender).text;
end;
```

Ett lika godtagbart alternativ är att skriva så här:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
   with Sender as TEdit do form1.caption:=text;
end;
```

Detta skrivsätt är speciellt praktiskt när man ska göra många saker med Sender och inte vill behöva göra typkonverteringen varje gång.

IS, AS & WITH

För att detta med delade events ska kunna användas effektivt är det viktigt att känna till de två operatorerna IS och AS samt kommandot WITH.
IS är en boolsk operator som tittar efter om klassinstansen till vänster är av klasstypen till höger.
Ex:
Button1 IS TButton är sant
Button1 IS TObject är sant
Button1 IS TBitmap är falskt, detta kommer överhuvudtaget inte att gå igenom kompilatorn.

AS används för att omvandla en typ till en annan
Ex (sender är av typen TObject)
sender AS TButton fungerar om sender är en TButton eller någon sub-klass till TButton.

WITH används ofta för att slippa göra typomvandling med AS på varje rad.
Ex

```
WITH sender AS TButton do begin
        caption:='Här får knappen en ny text';
        caption:='Här får knappen en ny text igen';
end;
```

Databashantering

Database Desktop

Database Desktop är det program man använder för att skapa databaser. Det kan antingen startas direkt från programhanteraren eller inifrån Delphi (under Tools).

Att skapa ett alias

Alias är en sorts fejkad sökväg som visar var en databas är belägen. Fördelen med att använda dem är åtminstone i teorin att det blir en smula lättare att flytta saker och ting mellan datorer. Delphi är dock såpass uselt på att hantera vad som händer om det blir det minsta fel någonstans i någon sökväg att den fördelen kan tas med en nypa salt.
För att göra ett alias gör följande:

1. Välj File|Aliases... (i Database Desktop)
2. Klicka på New
3. Skriv in ett NYTT namn
4. Ge sökvägen till aliaset
5. Klicka på Keep New

6. Klicka på Ok och svara Ja på frågan om det är ok att spar aliaset, om man inte gör det existerar enbart aliaset tills man går ur Database Desktop.

Att skapa en tabell

Innan man skapar en tabell är det praktiskt att sätta arbetsbiblioteket, lämpligen till det alias vi precis har skapat.

1. Välj File|New|Table
2. Välj vilken typ av databas du vill använda dig av och klicka på OK.
   (Härifrån förutsätter jag att ni valde en vanlig Paradox-databas, annars kan vissa mindre saker vara annorlunda)
3. Skriv in namnet på det fält du vill skapa och tryck på tab.
4. Välj vilken typ av fält du vill använda mha mellanslagstangenten, höger musknapp eller lämpligt kortkommando och tryck på tab.
5. Om fälttypen har variabel storlek (tex om det är ett Alpha-fält) så får du nu skriva in hur stort fältet ska vara.
6. Om fältet ska vara en nyckel markerar du nästa fält (genom att trycka på någon tangent).

OBS! SAMTLIGA nyckelfält MÅSTE stå överst

7. Klicka på Save As och spara tabellen.

Om du vill lägga in lite data i tabellen innan du tar in den i Delphi klicka på ikonen längst till höger på verktygsraden. 

Tabellen du skapar kommer automatiskt att sorteras efter nyckeln (nycklarna).
Ett kommando som kan vara bra att nämna här är Ctrl-Delete som tar bort en hel rad.

Att visa en tabell i Delphi

Att visa en tabell i Delphi är lätt, klicka på fliken ”Data Controls” och välj den första komponenten (DBGrid).
Placera ut din tabell på formuläret genom att klicka och dra på vanligt sätt.
Klicka sedan på fliken ”Data Access” och placera ut en DataSource och en Table på ditt formulär.
Klicka på DataSource-komponenten och välj egenskapen DataSet sätt den sedan till Table1 genom att klicka på den lilla pilen.
Klicka sedan på din Table-komponent (Table1) och välj egenskapen DatabaseName. Välj en databas att jobba med, lämpligen det alias du skapade tidigare.
Välj sedan egenskapen TableName och välj vilken tabell du vill jobba med, lämpligen den du skapade förut.
Sätt också egenskapen Active till true. Detta gör att databaskomponenterna kan börja jobba mot databasen.
Klicka på din DBGrid och under egenskapen DataSource väljer du DataSource1.
Klart, du har en fin liten databas du kan titta på och editera.

Att använda SQL i Delphi

Att göra utsökningar mha SQL i Delphi är inte svårare än att visa en vanlig tabell. Plocka bort Table-komponenten från formuläret och sätt dit en Query-komponent i stället och ändra på följande egenskaper:
DataSource1: Dataset = Query1
Query1: DatabaseName = DBDEMOS
Klicka sedan på egenskapen SQL och skriv av följande:

```
Select Name, Population
From Country
where Area<2000000;
```

Klicka på OK för att gå ut och sätt egenskapen Active till true. Tadaa, vi har en lista över de länder i Sydamerika som har en yta mindre än 2 miljoner km^2. (Tror jag i alla fall :-)

Lägga till någonting i en databas inifrån Delphi

Följande kodsnutt lägger till ett element sist i en databas. Jag tror jag låter kommentarerna tala för sig själva.

```
procedure TForm1.NyClick(Sender: TObject);
```

```
var n:integer;
begin
   with Table1 do begin
      Last;                                  {Flyttar oss längst ner i databasen}
      n:=FieldByName('Nr').AsInteger;        {Spara undan värdet}

      Append;                                {Lägger till en ny post i databasen}
      FieldByName('Nr').AsInteger:=n+1;      {Ge den nya posten ett nummer}
      FieldByName('Namn').AsString:='Dummy'; {Ge den nya posten ett namn}

      {Nu ska vi bara titta efter om vi är säkra på att vi vill lägga till Dummy}
      if MessageDlg('Är du säker på att du vill lägga till en person ?',
            mtConfirmation, [mbYes, mbNo],0)
      =mrYes
      then Post                              {Spara ändringarna i databasen}
      else Cancel;                           {Plocka bort ändringarna ur databasen}
   end; {With}
end;
```

Ta bort någonting ur en databas inifrån Delphi

Följande lilla kodsnutt tar bort den record som för tillfället är vald i en databas.

```
procedure TForm1.BortClick(Sender: TObject);
begin
   {Titta efter om vi verkligen vill ta bort en person}
   with Table1 do begin
      if MessageDlg('Är du säker på att du vill ta bort '+FieldByName('Namn').asString+' ?',
            mtConfirmation, [mbYes, mbNo],0)
      =mrYes
      then Delete;          {Om användaren svarade Ja, ta bort posten}
   end; {With}
end;
```

Prestandan hos dina databasapplikationer

Om ni ska gå många steg i en databas (tex mha funktionen next) är det smart att ”slå av” dess datasource innan ni gör det. Ta tex följande kodrader saxade ur ett av de exempelprogram som finns upplagda på NT-systemet (DB_fort).

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var t:TDateTime;
begin
   t:=time;
   while DBGrid1.Fields[0].asinteger<1300 do Table1.next;
   t:=time-t;
   messagedlg('Det tog tiden:'+timetostr(t),mtInformation,[mbOk],0);
end;
```

```
procedure TForm1.BitBtn3Click(Sender: TObject);
var t:TDateTime;
begin
   t:=time;
   DataSource1.enabled:=false;
   while Table1.Fields[0].asinteger<1300 do Table1.next;
   DataSource1.enabled:=true;
```

```
    t:=time-t;
    messagedlg('Det tog tiden:'+timetostr(t),mtInformation,[mbOk],0);
end;
```

Båda använde sig av kunddatabasen från DBDEMOS och hade att söka igenom ca 300 poster innan dom gick ur loopen. Den första varianten tog drygt två sekunder på sig, den andra mindre än en. Detta i en pytte-databas. Tänk själva vilka förbättringar det kan röra sig om när det gäller STORA datamängder.

Drag'n Drop

Muspekaren

Först en liten sak som står förbaskat dåligt förklarat i hjälpen, hur man laddar in egna muspekare.
Börja med att skapa en muspekare, detta görs lättast mha "Image Editor" som finns under Tools-menyn.
Lägg inte muspekaren i en .cur-fil utan i en .res, titta sedan på exemplet nedan.

```
{$R d:\temp\test.res}        {Kompilatordirektiv som specificerar filnamnet där resursfilen finns}
procedure TForm1.FormCreate(Sender: TObject);
begin
    screen.cursors[1]:=loadcursor(hInstance, 'Cursor_1');          {Ladda in muspekaren som heter
                                                                    "Cursor_1" i den array av muspekare
                                                                    som finns, värdena från 0 och neråt
                                                                    är upptagna redan så ta ett tal >0}
    cursor:=1;                          {Välj vilken markör som ska visas}
end;
```

Att få Drag'n drop att fungera smärtfritt

Liksom på de flesta andra ställena i det här kompendiet ska vi ta och visa hur detta går till med hjälp av ett enkelt exempel. Vi behöver ett formulär med en knapp. Meningen är att man ska kunna dra omkring knappen, och klicka på den för att få reda på vilka kordinater den ligger på för tillfället. Anledningen till att vi inte bara skriver ut detta automatiskt kommer att uppenbaras.
Vi börjar med den enkla delen, att få knappen att skriva ut sina kordinater. Det gör den om ni skriver in följande kod under händelsehanteraren OnClick

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    form1.caption:='x:'+IntToStr(button1.left)+' y:'+IntToStr(button1.top);
end;
```

Sätt sedan knappens egenskap DragMode till dmAutomatic, då går det att börja dra den så fort man trycker ner vänster musknapp. Om du nu kör programmet kommer du att upptäcka att formuläret inte vill ta emot knappen, dvs muspekaren ser ut som en förbudsskylt. För att råda bot på detta skriv in följande under formulärets OnDragOver-hanterare:

```
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
    accept:=source is Tbutton;          {Om det objekt (source) som för tillfället dras över formuläret är en
                                         Tbutton så ska vi acceptera att det släpps}
end;
```

För att sedan få knappen att faktiskt ändra läge skriv till följande kod under dess händelse OnEndDrag

```
procedure TForm1.Button1EndDrag(Sender, Target: TObject; X, Y: Integer);
begin
    button1.left:=x;
    Button1.top:=y;
end;
```

Jättefint, det är bara det att nu fungerar det inte att klicka på knappen, anledningen är att vi tidigare satte dragmode till dmAutomatic, vi gav då upp alla möjligheter till att hantera OnClick. Sätt tillbaka knappens dragmode till dmManual och testa igen. Nu går det åter att klicka på den, men vi kan inte dra den. Lösningen är att lägga in följande kod under knappens OnMouseDown-händelse:

```
procedure TForm1.Button1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
    button1.begindrag(true);
end;
```

Om ni nu testar det så kommer ni att märka att det nu åter inte går att klicka på knappen. Anledningen är att vi har satt använt button1.begindrag(true), detta kommando gör att så fort användaren trycker ner musknappen tror knappen att det är dags att dras. Ändra det till button1.begindrag(false). Hmm, katten också, det går fortfarande inte riktigt som vi tänkt oss. Ändra koden under knappens OnEndDrag till:

```
procedure TForm1.Button1EndDrag(Sender, Target: TObject; X, Y: Integer);
begin
    if target is TForm then begin
      button1.left:=x;
      Button1.top:=y;
    end;
end;
```

Pust, nu ska det hela fungera tillfredställande. Skillnaden ligger helt enkelt i att vi nu tittar efter var vi egentligen försöker släppa objektet. Om man kombinerar ihop detta med lite snyggt användande av muspekare så kan man relativt lätt göra mycket snygga drag'n drop operationer. Kom bara ihåg att om du vill kunna klicka på något som också ska kunnas dras så får du inte sätta dragmode till dmAutomatic.
Hela exemplet finns i biblioteket dragdrop.

Klasser

Här ska vi bara prata lite snabbt om hur man skapar egna klasser, för att verkligen kunna utnyttja denna möjlighet krävs en hygglig dos objektorientering. Något som är speciellt trevligt i Delphi, men som vi tyvärr i praktiken inte kan utnyttja är att man kan ta en existerande komponent och göra om den. Dvs om du tex vill ha en bildkomponent som visar ett bildspel, kan du ta den existerande komponenten och göra om den samt lägga till den på verktygsraden. Sedan är det bara att klicka och dra precis som med alla andra komponenter. Tyvärr kan inte vi göra det här i skolan eftersom vi inte får lägga till saker där det skulle vara nödvändigt (och tur är nog det), så jag har inte haft tillfälle att titta alltför mycket på det. Den som är intresserad hänvisas till "Delphi Componet Writer's Guide".

Klasser i Delphi fungerar nästan precis som i C++ vilket dom flesta som läser detta borde ha åtminstone testat, så vi nöjer oss med att berätta hur man enklast gör en klass.
Välj File|New Component
Skriv in namnet du vill att din nya klass ska få och sätt "Ancestor type" till TComponent. "Palett page" har att göra med det jag pratade om ovan, så det kan ni göra hur ni vill med. Personligen föredrar jag att lämna den blank.
En ny Unit som innehåller ett skelett till din klass kommer då att skapas, och det är "bara" att skriva in koden för den. För ett exempel titta i biblioteket STACK.

En liten sak som är lite konfunderande för den som är van vid tex C++ är att om man vill skapa ett nytt objekt av en klass-typ så måste anropet till konstruktorn göras via en klass-referens. Dvs om C är ett objekt av en klass CKlass med konstruktorn Create så gäller:

```
C.Create;              {Fungerar ej}
C:=CKlass.Create;      {Fungerar}
```

Det som är riktigt irriterande är att den första varianten inte ger kompileringsfel utan bara en massa problem när man ska börja använda klassen.

Ytterligare en sak som kan ställa till problem för den som är van vid C++ är Pascals (och Delphis) totala vägran att acceptera någon typ av namnkonflikter. Dvs det går inte att ha flera procedurer eller funktioner med samma namn.

Appendix 1 Datatyper

Uppräkningsbara datatyper

Gemensamt för alla uppräkningsbara datatyper är funktionerna Ord, Pred, Succ, Low och High.

Heltalsdatatyper

Typnamn	Område	Anmärkning
Integer	-32768 .. 32767	Området kan variera beroende på vilken variant av Object-Pascal man använder (och operativsystem)
Shortint	-128 .. 127	Signed 8-bit
Smallint	-32768 .. 32767	Signed 16-bit
Longint	-2147483648 .. 2147483647	Signed 32-bit
Byte	0 .. 255	Unsigned 8-bit
Word	0 .. 65535	Unsigned 16-bit
Cardinal	0 .. 65535	Området kan variera beroende på vilken variant av Object-Pascal man använder (och operativsystem)

"Logiska" datatyper

Namn	Storlek	Anmärkning
Boolean	1 byte	Kan bara anta värdena 0 (falskt) och 1 (sant)
ByteBool	1 byte	0 falskt, alla andra värden sant
WordBool	2 bytes	0 falskt, alla andra värden sant
LongBool	4 bytes	0 falskt, alla andra värden sant

I normalfallet kommer ni bara att behöva använda boolean, de andra är till för att ge kompatibilitet med andra språk (tex C) och med själva Windows.

Dessutom finns typen Char som håller ett tecken och egendefinierade datatyper. De sistnämnda är av två typer, uppräknade, tex namn=(Kalle, Nisse, Johan); och "subrange" som definierar en delmängd av en annan uppräkningsbar datatyp, tex MinInt=0..99;

Flyttalsdatatyper

Namn	Område	Signifikanta siffror	Storlek (bytes)
Real	2.9 * 10-39 .. 1.7 * 1038	11-12	6
Single	1.5 * 10-45 .. 3.4 * 1038	7-8	4
Double	5.0 * 10-324 .. 1.7 * 10308	15-16	8
Extended	3.4 * 10-4932 .. 1.1 * 104932	19-20	10
Comp	$-2^{63}+1$ .. $2^{63}-1$	19-20	8

Strängtyper

Typen string definierar en sträng med max 255 tecken. Dvs, varje string ni deklarerar tar upp 256 bytes. Om man vet att man inte behöver allt detta utrymme kan man minska maxlängden, ex Namn = string[25];
Om man vill ha tag på ett speciellt tecken i en sträng gör man det med tex namn[1]. Observera att själva texten i en Pascal-sträng börjar i position 1, position noll innehåller strängens längd. Namn som vi definierade ovan är alltså 26 bytes stor och inte 25.

Det finns dock en annan strängtyp i Delphi också, men den är väl gömd, om man inte vet om att den måste finnas där, eller råkar snubbla över den av misstag så är det lätt att missa den. Den heter Pchar och fungerar på samma sätt som strängar i tex C, dvs den har dynamisk längd, och är ”null-terminerad” (ja det ska faktiskt vara null och inte nil :-) dvs den avslutas med det tecken som har ascii-koden 0. Den här typen av strängar är nödvändig tex för att kunna överföra parametrar till funktioner skrivna i andra språk (läs C/C++).

Strukturer

Förutom de vanliga strukturtyperna som alltid finns i Pascal och som ni förhoppningsvis känner till sen tidigare (array, file, record och set) finns det i Delphi två ytterligare object-typer och klassrefererande-typer.

Pekare

Pekare fungerar precis som vanligt i Pascal. Dvs:
En pekartyp deklareras mha ^
Ex:

```
type
IntPekarTyp = ^integer;
```

En Pekarinstans kan sedan deklareras mha pekartypen (eller direkt, men det är fult)
Ex:

```
var
IntPekare : IntPekarTyp;
```

Minne till en pekar-variabel deklareras mha funktionen New
Ex:

```
New(IntPekare);
```

För att referera till pekaren använder man ^
Ex:

```
IntPekare^ := 10;
```

När man är klar med minnet pekaren använder är det bra om man lämnar tillbaka det mha Dispose
Ex:

```
Dispose(IntPekare);
```

Dessutom kan man gör en del andra saker...
För att få adressen till en variabel kan man använda operatorn @
Ex:

```
{X är en integer}
IntPekare = @X;
```

För att peka på en specifik adress i minnet använder man funktionen Ptr
För att deklarera ett dynamiskt minnesområde av en viss storlek använder man funktionen GetMem.

Detta är dock överkurs, så vi får hänvisa till hjälpen eller någon lämplig bok.

Procedurtyper och Objektrefererande typer

Förutom ovannämnda typer som är dom ni i normalfallet kommer att använda själva finns det även procedurtyper och objektrefererande typer. Procedurtyper kan ni strunta helt i under mycket lång tid framöver. Objekttyper kan eventuellt vara till nytta någon gång, men även de är en bra bit in på överkurskapitlet.

Typkonvertering

Det enklaste sättet att göra typkonverteringar i Delphi om man vill vara säker på att slippa ”Error 61: Invalid type cast” är att använda de funktioner som finns för det. Dessa har namn av typen “FråntypToTilltyp”. En stor fördel med att använda dessa funktioner är att det gör det mycket lättare att se vad som faktiskt sker. Värt att notera är att heltal heter “int” och flyttal “float”. Anledningen är säkerligen

att det finns flera olika typer av dem båda, men när man har ägnat en kvart åt att leta efter en funktion som man tycker borde heta “IntToReal” är man inte så glad. Man kan tex göra så här.

```
A:integer;
B:real;

A:=10;
B:=IntToFloat(A);
```

En speciell funktion som förtjänar lite eget utrymme är StrToIntDef, som namnet antyder konverterar den en sträng till en integer. Till skillnad från den vanliga StrToInt fungerar dock denna även om strängen inte innehåller ett tal, i sådana fall används ett default-värde.

```
S1, S2: string;
I1, I2: integer;

S1:=‘Ingen integer här inte’;
S2:=‘100’;

I1:=StrToIntDef(S1, -1);          {I1 sätts till -1}
I2:=StrToIntDef(S2, -1);          {I2 sätts till 100}
```

Naturligtvis fungerar även Pascals vanliga typkonvertering dvs nytyp(variabel av annantyp).

Appendix 2 Nyttigheter

Typer

Förutom de ovan uppräknade datatyperna finns det en mängd fördefinierade. Jag ska här nämna några som är praktiska att känna till. För mer information hänvisas till hjälpen.

Namn	**Ex på värden**	**Beskrivning**
TColor	clBlack	Kan även skrivas som ett hex-tal
TCursor	crDefault	Pekartyper
TDateTime		Används för att lagra datum och tids-värden
TFont		
TModalResult	mrNone, mrOk	Används för att skicka tillbaka information om hur användaren avslutade ett modalt fönster.
TObject		Abstrakt datatyp som används när man inte vet vilken typ av objekt man ska hantera.

Procedurer och funktioner

Delphi innehåller bokstavligt talat hundratals funktioner, att hitta bland dessa är speciellt i början mycket snårigt. Här är några som kan vara nyttiga alternativt roliga att känna till. Jag har inte tagit med några funktioner som ligger som metoder till någon program-komponent tex, dessa kan ni läsa om i hjälpen.

Namn	**Beskrvning**
ExtractFileName(...)	Plockar ut själva filnamnet ur en sträng som också innehåller sökvägen
FileExists(...)	Tittar efter om filen finns.
FindComponent(string)	Returnerar en referens till den komponent vars namn står som parameter
MessageBeep(integer)	Ljud
Messagedlg(...)	Systemdialog
Random(Integer)	Ger ett slumptal < Integer, dvs Random(2) ger antingen 0 eller 1. Initializeras med proceduren Randomize
StrToIntDef(‘sträng’,integer)	Som StrToInt, men med ett deafult-värde om strängen inte innhåller ett heltal, gör att man slipper avbrott i sådana fall

Appendix 3 Översikt över projekten i kurskatalogen

Följande projekt finns i kurskatalogen mickey:/kurser/Delphi-kv i skrivandets stund. För en uppdaterad lista titta i filen mickey:/kurser/Delphi-kv/filer.txt.

Katalog	**Beskrivning**
circ	Exempel på hur man löser problemet med cirkulär-referens.
DB/DB_fort	Exempel som visar nyttan av att stänga av en datasource
DB/db_ny_bo	Ett första databasexempel som visar hur man lägger till och tar bort saker i en tabell
DB/DBAPP	Enkelt personregister som sammanfogar lite olika delar till något som åtminstone liknar en helhet
DB/KursDB	De tabeller som används av de olika databasapplikationerna.
delaeven	Exempel på en delad händelse
dragdrop	Den slutliga koden till Drag’n Drop-exemplet ovan
SKRIV	Enkel textredigerare avsedd som ett första exempel på ett fungerande Delphi-program, skriven på under 15 minuter
STACK	Exempel på en enkel klass som helt oväntat hanterar stackar, visar hur en enkel klass är uppbyggd, hur man skapar en instans av den och hur den kan användas.