OBJEKTOVÉ PROGRAMOVÁNÍ V TURBO PASCALU A TURBO VISION

ZÁKLADNÍ PRINCIPY OOP(OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ)

OBJEKT je jakýmsi rozířením datového typuu záznam. K popisu vlastností slouí ostatní datové typy, konstanty... a **METODY**-podprogramy.
Deklarováním typu jako OBJEKTU zavademe **TŘÍDU (class)** objektů pod jedním jménem. Zavedená proměnná typu OBJEKT se nazývá **INSTANCE**.

Př.:

```
pes=object
    barva:(černá,bílá);
    délka_srsti:real;
    procedure stekot;
    procedure hlídej;
end;
```

tímto je zavedena třída objektů nazvanou PES.

Konkrétní realizace objektu třídy - proměnná se zve instancí.

OOP umoňnuje logocky sdruovat související data a kód (zapouzdření(encapsulation))

DĚDIČNOST VLASTNOSTÍ (INHERITANCE)

Pokud máme nadefinován nějakakou třídu a chceme vytvořit novou třídu, která má stejné vlastnosti jako původní a navíc své zvlátní vlastnosti, lze vyuít dědičnosti. Vytvoříme novou třídu např. *CHRT* jako **Potomka (Decendant) Třídy PES** (viz. výe).

```
CHRT=object(pes)
    procedure zavod
END
```

Tímto jsme vytvořily třídu CHRT, která má vechny vlasntosti jako pes, ale navíc obdahuje metodu zavod. Třída PES je **předchůdcem (ancestor)** třídy CHRT.

POLYMORFISMUS OOP

Metody lze definovat jako **Statické** = překladač zná vechny okolnosti v době překladu a vygeneruje kód.
Pokud poadujeme různý kód metody v závislosti na zvolené třídě potomků, pak u vech (i předka) musí být metoda definována jako **Virtuální** - pouitím VIRTUAL.

Př.:
Nadefinujeme si novou třídu PES

```
pes=object
    barva:(černá,bílá);
    délka_srsti:real;
    procedure kresli_figuru;
    procedure stekot;
    procedure hlídej;
end;
```

Metoda *kresli_figuru* slouí k vykreslení figury psa.
Metoda *hlídej* simuluje pohyb psa kolem domu překreslováním, tudí obsahuje volání metody kresli_figuru.

```
procedure hlidej
begin
```

```
...
kresli_figuru;
...
end;
```

Pokud nyní vyvoláme metodu hlídej z třídy CHRT (potomek PES) pak bude vykreslen obecný pes a nikoliv chrt, protoe je to statická definice a dolo k **ČASNÉ VAZBĚ (EARLY BINDING)**.

Pomocí virtuální metody:

```
pes=object
    barva:(černá,bílá);
    délka_srsti:real;
    procedure kresli_figuru;virtual;
    procedure stekot;
    procedure hlídej;
end;
```

pak u potomků musíme uvést té tuto metodu jako virtuální:

```
CHRT=object(pes)
...
procedure kresli_figuru;virtual;
...
END;
```

Překladač vytvoří tabulku virtuálních metod. V kódu metody hlídej není přímý skok na kód metody *kresli_figuru*, ale volání je uskutečně odkazem do tabulky virtuálních metod. A při běhu programu se pozná, do tabulky které třídy se má sáhnout pro adresu metody *kresli_figuru* = POZDNÍ VAZBA (LATE BINDING)

OOP V TURBO PASCALU

DEFINICE TŘÍDY, VYTVOŘENÍ INSTANCE OBJEKTU TŘÍDY

Třída objektů je novým datovým typem:

```
type pes=object
       barva:tbarva;
       délka:integer;
       constructor init(abarva:tbarva;delka:integer);
       procedure kresli_figuru;virtual;
       procedure stekej;
       procedure hlídej;
     end;
```

tímto je definována třída objektů pes, mající 2 datové vlastnosti a 4 metody.

Definice třídy jako potomka:

```
chrt=object(pes)
   procedure kresli_figuru;virtual;
end;
```

Při zápisu metod se před jméno procedury/funkce píe jméno třídy oddělené tečkou.

```
procedure pes.stekej;
begin
...
end;
```

Vytvoření instance třídy=deklarace promměnné typu třídy:

```
var pes1:pes;
```

Proměnná pes1 je instancí proměnné třídy pes

KONSTRUKTORY

Pokud třída obsahuje aspoň jednu vyrtuální metodu, musí také existovat metoda začínající klíčovým slovem CONSTRUKTOR !! V opačném případě dojde k syntaktické chybě při překladu. Jde o zvlátní metodu, při jejím vyvolání je vytvořeno spojení mezi tabulkou virtuálních metod třídy objektů a aktuální instancí. Z toho vyplývá, e KONSTRUKTOR MUSÍ BÝT VYVOLÁN DŘÍV NE JINÁ VIRTUÁLNÍ METODA, jinak dojde k chybě.

Tedy konstruktor by měl obsahovat vechny akce související s inicializací objektu, tj. nastavení počátečních hodnot... pro správnou funkci programu.

PŘÍSTUP K VLASTNOSTEM OBJEKTU, ROZSAHY PLATNOSTI

Přístup pomocí tečkové notace:

```
pes1.stekej;
a:=pes1.delka;
```

Uvnitř metody kadého objektu jsou vechny jeho vlastnosti (datové i dalí metody) přístupné přímo - bez tečkové notace.

```
procedure pes.hlídej
begin
...
stekej;
a:=delka;
...
end;
```

Pro statické metody platí, e lze předefinovat metodu na vyí úrovni=může se liit počtem parametrů. NELZE U VIRTUÁLNÍCH METOD!-zde nelze měnit parametry, hlavička musí být pořád stejná!!

IMPLICITNÍ IDENTIFIKÁTOR SELF

SELF je automaticky deklarovaný ukazatel, který v kadém okamiku obsahuje asresu aktuálního umístění objektu. Přístup k vlastnostem objektu uvnitř tohoto objektu lze uskutečnit bez tečkové notace - SELF automaticky doplní název třídy.
Pouívá se zejména v případech kolizí identifikátorů, které nelze odstranit jiným způsobem.
Př.:

```
TYPE popis=record
         barva:tbarva;
         delka:integer;
     end;

pes1=object(pes)    {...viz výe}
   procedure nastav(co:popis);
end;

procedure pes1.nastav(co:popis);
begin
  with co do
  begin
    self.barva:=barva;  {jinak barva=barva ?? }
    self.delka:=delka;
  end;
end;
```

DYNAMICKÁ ALOKACE OBJEKTŮ, DESTRUKTORY

Dynamická alokace: NEW(UKAZATEL,[,KONSTRUKTOR]);
tedy: **ukazatel=new(typUkazatele [,konstruktor]);**
konstruktor se po provedení alokace automaticky provede, uvádí se jen pokud objekt obsahuje virtuální metody.
Uvolnění dynamicky alokovaného objektu
DISPOSE(UKAZATEL[,DESTRUKTOR]);
První parametr je odkaz na ruený objekt, druhý parametr je tzv. destruktor. Destruktor je zvlátní metoda, která se definuje pomocí klíčového slova **DESTRUCTOR**

```
př.: type ovcak=object(pes);
          ...
```

```
        destructor done;
        ...
        end;
```

Destruktor uvolňuje dynamicky alokovaný objekt. Nejdříve se provede kód destruktoru a pak se dealokuje objekt. Pokud tedy v kódu destruktoru neprovedeme uvolnění dynamicky alokovaných dat, zůstanou nepřístupná ale pořád v paměti(heapu)!

VZÁJEMNÁ PŘIŘADIDELNOST OBJEKTŮ

předek:=předek
potomek:=potomek
ředek:=potomek

PŘÍKLAD PROGRAMU S PEJSKY :)

/v PASCALU/
{program simuluje hlidani psa-pes beha kolem domu a po stisknuti klavesy po dokonceni objehu zasteka a simulace je ukoncena...probiha pro 3 druhy psu}

```
program Priklad_objektoveho_programovani;
uses crt,graph;
type tbarva=(BILA,CERNA,JINA);
     pes=object
         barva:tbarva;
         delka:integer;
         constructor init(abarva:tbarva;adelka:integer);
         procedure stekej;
         procedure kresli_figuru(x,y:integer);virtual;
         procedure hlidej;
     end;
     chrt=object(pes)
          procedure kresli_figuru(x,y:integer);virtual;
     end;
     ovcak=object(pes)
           procedure kresli_figuru(x,y:integer);virtual;
           destructor done;
     end;
     povcak=^ovcak;

constructor pes.init(abarva:tbarva;adelka:integer);
begin
  barva:=abarva;
  delka:=adelka;
end;

procedure pes.stekej;
begin
  write(#7);
  delay(50);
  write(#7);
end;

procedure pes.kresli_figuru(x,y:integer);
begin
  outtextxy(x,y,'PES');
end;

procedure pes.hlidej;
var pom:integer;
    zn:char;
    procedure kresli(x,y:integer);
    begin
      setColor(white);
      kresli_figuru(x,y);
      delay(2);
      setcolor(black);
      kresli_figuru(x,y);
    end;
begin
  bar(100,100,200,200);
  repeat
    for pom:=50 to 210 do kresli(pom,80);
    for pom:=80 to 210 do kresli(210,pom);
    for pom:=210 downto 50 do kresli(pom,210);
    for pom:=210 downto 80 do kresli(50,pom);
```

```
  until keyPressed;
  zn:=readKey;
  stekej;
end;

procedure chrt.kresli_figuru(x,y:integer);
begin
  outtextXY(x,y,'CHRT');
end;

procedure ovcak.kresli_figuru(x,y:integer);
begin
  outtextXY(x,y,'OVCAK');
end;

destructor ovcak.done;
begin
end;

var driver,mode:integer;
    pes1:pes;
    pes2:chrt;
    pes3:povcak;

BEGIN
  pes1.init(CERNA,1);
  pes2.init(BILA,1);
  pes3:=NEW(povcak, init(JINA, 2));
  driver:=detect;
  initGraph(driver, mode,'../BGI');
  pes1.hlidej;
  pes2.hlidej;
  pes3^.hlidej;
  dispose(pes3,done);
  closeGraph;
end.
```

ÚVOD DO TURBO VISION

CO JSOU TURBO VISION

Programová jednotka Turbo Vision je nástrojem pro usnadnění vývoje uivatelsky příjemných programů. Tato knihovna je psána s využitím principů OOP!
Turbo Vision pracují v textovém reimu ale výsledné programy se vzhledově podobají grafickám systémům - obsahují stejné prvky jako dialog-box, radio-button a dalí.
Vzhled programu napsanho v Turbo Vision je podobný jako prostředí např. Turbo Pascalu.

PRINCIP PRÁCE

Vechny objekty lze rozdělit na dvě skupiny: OBJEKTY PROVÁDĚJÍCÍ VÝSTUP NA OBRAZOVKU - **VIEWS**
"NĚMÉ OBJEKTY" PROVÁDĚJÍCÍ OSTATNÍ ČINNOST JAKO VÝPOČTY...
Mezi těmito skupinami by měla bát zachována přísná dělba práce-tj. aby se němé objekty nijak nesnaili komunikovat a naopak, aby views jen komunikovala, ale práci přenechala němým objektům.

Uivatel působí na program pomocí klávesnice a myi. Jejich pouití v určité oblasti je akcí, na kterou program reaguje.
Kadá akce způsobí vznik události!
Událost je datová struktura rozlující akci uivatele=bylo-li stisknuto tlačítko myi, nebo klávesa... =>**EVENT DRIVENT PROGRAMING=událostmi řízené programování**

Turbo Visions poskytují objekty, v metodách těchto objektů je určeno jejich chování. Uivatel si přizpůsobuje jejich chování vytvářením potomků předdefinovaných tříd.

Vlastní aplikační program je také objekt! Je pro něj předdefinována třída **TAPPLICATION**
Jména vech předdefinovaných tříd v Turbo Vision začínají písmenem T, vechny ukazatele začínají P.

PRVNÍ PROGRAM V TURBO VISION (TV1.PAS)

```
USES APP;
```

```
type tMyApp=object(Tapplication)
     end;
var myapp:tmyapp;
begin
  myapp.init;
  myapp.run;
  myapp.done;
end.
```

Tento program pouze vykreslí pracovní plochu a jediná událost na kterou reaguje je kliknutí na nápis ALT-X Exit ve **status line (stavový řádek)**, nebo stisk ALT+X, co provede ukončení programu.

VYTVOŘENÍ MENU A STAVOVÉHO ŘÁDKU APLIKACE (TV2.PAS)

Vytvoříme program databáze PC komponentů. Zatím jen ukáeme tvorbu různých menu a stavového řádku, jedinou opravdovou funkcí je ukončení programu.

```
USES app, views, objects, menus, drivers, dialogs;

const cmSesNab=100;
      cmUkNab=101;
      cmZrnab=102;
      cmIMonitory=103;
      cmIDisky=104;
      cmIKlavesnice=105;
      cmOMonitory=106;
      cmOdisky=107;
      cmOKlavesnice=108;
      cmMore=109;
type pDealer=^Tdealer;
     tdealer=object(Tapplication)
             procedure initMenuBar;virtual;
             procedure initStatusLine;virtual;
     end;

procedure tDealer.initMenuBar;
var r:tRect;
begin
  getExtent(r);
  r.b.y:=r.a.y+1;
  menubar:=new(pMENUbar, init(r, newmenu(
    NEWSUBmenu('~N~abidka',hcNOcontext,NEWMENU(
      newItem('~S~estav','',0,cmsesnab, hcNocontext,
      newItem('~U~kaz','',0,cmuknab, hcNocontext,
      newItem('~Z~rus','',0,cmzrnab, hcNocontext,
      newLine(
      newItem('~K~onec','ALT-X',kbALTX,cmQuit, hcNocontext,
      nil)))))
      ),
    newSubMenu('Vstupy',hcNOcontext,newMenu(
      newItem('Monitory','',0,cmImonitory, hcNocontext,
      newItem('Disky','',0,cmIdisky, hcNocontext,
      newItem('Klavesnice','',0,cmIklavesnice, hcNocontext,
      nil)))),
    newSubMenu('Vystupy',hcNOcontext, newMenu(
      newItem('Monitory','',0,cmOmonitory, hcNocontext,
      newItem('Disky','',0,cmOdisky, hcNocontext,
      newItem('Klavesnice','',0,cmOklavesnice, hcNocontext,
      nil)))),
    NIL))))
    ));
  end;

Procedure tDealer.initStatusLine;
var r:trect;
begin
  getExtent(r);
  r.a.y:=r.b.y-1;
  statusLine:=new(pStatusLine, Init(r,
    newStatusDef(0,$FFFF,
    newStatusKey('~ALT-X~ Exit',kbAltX,cmQuit,
    newStatusKey('~F1~ Help',kbF1,cmHelp,
    newStatusKey('~F2~ Nabidka',kbF2,cmSEsNab,
    newStatusKey('~F10~ Menu',kbF10,cmMenu,
```

```
    nil)))),
    nil)
  ));
end;

VAR a:tdealer;
BEGIN
  a.init;
  a.run;
  a.done;
END.
```

Vzhled menu a stavové řádky lze ovlivnit předefinováním metody *InitMenuBar* a *InitStatusLine* standartního objektu **TApplication**. Tyto metody jsou předefinovávy v potomkovy *TDealer*.

PŘÍKAZOVÉ KONSTANTY CMXXXX

Kadá akce způsobí vznik události. **Událost je datová struktura typu RECORD se jménem typu TEVENT, která obsahuje informace o dění a je předávána jestnotlivým view.**
Vybere-li uivatel nějakou poloku v menu, znamená to té vznik události. Informace o tom, jaký příkaz byl vybrán a jaká akce se má provést je reprezentován hodnotou typu word, která je jednoznačně svázána s kadou polokou menu. Programátor je ve svém programu definuje jako konstanty, začínající **CM (od COMMAND)**.
V Turbo Vision jsou defonovány konstanty pro standartní akce jako *cmQuit* atd. Jsou tedy rezervovány hodnoty **0..99** a **256..999**.
Ostatní hodnoty má programátor k dispozici. (tedy 100..255 a 1000..65535)

HLAVNÍ MENU PROGRAMU - METODA INITMENUBAR

Hlavní menu vytvoříme předefinováním metody *initMenuBar třídy TApplication*, to jest vytvořením potomka.
Tato metoda je předdefinována jako virtuální, tudí potomek musí být té virtuální!!

OBJEKT TRECT

TRect je jednoduchý objekt, jeho datovými položkami jsou dva objekty typu **TPoint**, kde **TPoint** je objekt obsahující dvě datové položkytypu **X,Y**. TPoint představuje souřadnice nějakého bodu na obrazovce. **TRect je zdednodueně definován:**

```
Type TRect=Object
     A,B:TPoint;
     procedure Assign(XA,YA, XB,YB:integer);
     procedure Move(ADX,ADY:integer);
     Procedure Grow(ADX,ADY:integer);
     ...
     END;
```

TRect se pouívá k určení polohy a velikosti něajkého view na obrazovce. **A** - souřadnice levého horního rohu, **B** - souřadnice pravého dolního rohu
Metoda Assign - umoňuje zadání vech souřadnic
Metoda Move - zvětí *X-ové* položky**A** a **B** o hodnotu **ADX** a *Y-ové* o hotnotu **ADY**->posune view v určeném směru
Metoda Grow - zvětí pouze poloku **B** - změní se velikost view

ZJITĚNÍ VELIKOSTI VIEW

Metoda GetExtent - vrací velikost view do parametru typu *TRect* tak, e **A** je nulováno a B je nastaveno na velikost v jednotlivých směrech.
V programu jsme nejdříve zjistili velikost celé plochy obrazovky a poté rozsah **Y** změnili na jediný řádek->menu je na nejvrchnějím řádku obrazovky.

PROMĚNNÁ MENUBAR

Ukazatel na hlavní menu aplikace je uloen v globální proměnné **MenuBar** typu **PMenuView** (ukazatel na **TmenuView** co je předek typů menu TMenuBar a TMenuBox)
Pokud je hodnota této proměnné NIL, neobsahuje ádné menu.
Musíme tedy její hodnotu definovat a to pomocí procedur a funkcí:

PROCEDURY A FUNKCE PRO VYTVÁŘENÍ MENU

Turbo Vision poskytuje prostředky pro vytváření dvou typů menu:

horizontální menu - objekty třídy **TMenuBar**
vertikální menu - objekty třídy **TMenuBox**

Tyto objekty se zpravidla vytvářejí dynamicky, proto je tvoříme voláním **NEW**, kde druhým parametrem je konstruktor pro přísluný typ menu.

- **TVAR KONSTRUKTORŮ**

```
constructor TMenuBar.INIT(var Bounds:TRect;AMenu:Pmenu);

constructor TMenuBox.INIT(var Bounds:TRect;Amenu:PMenu;AParentMenu:PMenuView);
```

- **Bounds** vymezuje oblast ve které bude menu vykresleno
- **AMenu** je seznam poloek menu (typu TMenuItem)
- **AParentMenu** je ukazatel na menu vlastnící definované menu a umoňuje vytvářet hiearchickou strukturu pull-down menu. Pokud jde o samostatné menu, bude tento parametr **NIL**

Seznam poloek menu, předávaný jako pako parametr AMenu vytvoříme pomocí funkcí:

```
Function NewMenu(Items:PMenuItem):Pmenu;

Fuction NewItem(Name,Param:TmenuStr;KeyCode:word;Command:word;AHELPCtx:word;next:pMenuItem):pMenuItem;

Function NewLine(Next:pMenuItem):PmenuItem;

Function NewSubMenu(Name:TmenuSTR;AHelpCtx:word;SubMenu:Pmenu;Next:PmenuItem):pMenuItem;
```

Parametrem funkce **NewMenu** je seznam poloek typu *TMenuItem*, který se vytvoří libovolným zřetězením funkcí **newLine, newItem a NewSubMenu**. Toto zřetězení je moné pomocí parametru **NEXT**.
Pro ukončení menu, vloíme NEXT=NIL a doplníme pravé závorky.

Parametr NAME - řetězec, představuje jméno položkyv menu. Pomocí tildy (~) lze označit znak pro zkrácené vybírání.
Parametr AHelpCTX - souvisí s vytvářením kontextově závislé nápovědy. Pokud není poadována, uvede se na tomto místě konstatna hcNoContext, která má hodnotu 0
Parametr Param - řetězec, který se objeví v menu vedle jména (NAME). Pouívá se pro vypsání jména klávesy jejím stisem lze provést alternativní výběr akce. Klávesa je určena parametrem **KeyCode**, na jeho místo se dosadí předdefinovaná konstanta, její jméno začíná **kb** (kbF1)...
Parametr SubMenu - odkaz na seznam poloek. Vytvoří se opakováním výe popsaných postupů.

NEZÁVISLÉ MENU, PROMĚNNÁ DESKTOP A METODA EXECVIEW

Pokud nechceme vytvářet hlavní menu pomocí MenuBar a chceme vytvořit menu jiné úrovně, vytvoříme menu stejným postupem (viz.výe) a **uloíme jej do vlastní proměnné.**
Toto menu pak v programu spustíme voláním metody:

```
DeskTop^.ExecView(p:Pview):word;
```

p - parametr, za něj dosadíme odkaz na vytvořené menu

Menu se objeví na obrazovce a bude tam tak dlouho, dokud nevybereme některou z jeho poloek. Pak zmizí a funkce **ExecView** vrátí hodnotu Command příslunou vybrané poloce. **DeskTop** - globální proměnná, obsahuje ukazatel na desktop aplikace. Je potomkem třídy TGroup - obsahuje jednotlivá view, které jsou do ní vkládány pomocí metody Insert a odebírány pomocí metody Delete. DeskTop je tedy **Group**, do něho vloená view se automaticky vykreslái na obrazovce. **Metoda ExecView** - obdoba metod *Insert* a *Delete*, provede vloení view určeného parametrem P do TGroup a převede jej do modálního stavu - čeká se a se view provede. Provedení je vybrání položkya je realizováno metodou **Execute**

STAVOVÁ ŘÁDKA APLIKACE - METODA INITSTATUSLINE

Pouívá se k výpisu krátkých informací, lze zde zobrazit poloky, které lze vyvolat stiskem horké klávesy nebo kliknutím myí...
Definice stavové řádky se provádí přepsáním metody InitStatusLine objekty TApplication

PROMĚNNÁ STATUSLINE

Obsahuje odkaz na hlavní stavovou řádku programu. Je typu *PStatusLine* a pokud je její hodnota NIL pak program nemá stavovou řádku. Vytvoření stavové řádky tedy znamená nadefinování hodnot této proměnné.

PROCEDURY A FUNKCE PRO VYTVÁŘENÍ STAVOVÉ ŘÁDKY

Stavová řádka se tvoří dynamicky. Pouívá se tedy konstruktor:

```
constructor Init(var Bounds:TRect;ADefs:PstatusDef);
```

Bounds - umístění stavového řádku na obrazovce
ADefs - odkaz na seznam poloek typy TStataus Def

Typ TStatusDef je definován:

```
Type TStatusDef=record
     next:PstatusDef;
     min,max:word;
     Items:PStatusItem;
     end;
```

Next - ukazatel na dalího člena v seznamu
Min, Max - určují rozsah help kontextů pro ně je tato položka platná
Items - seznam poloek typu TStatusItem, definuje aktuální obsah stavové řádky

Typ TStatusItem je definován:

```
Type TStatusItem = record
     next:PstatusItem;
     text:Pstring;
     KeyCode:word;
     Commnand:word;
     End;
```

Next - ukazatel na dalí prvek seznamu
Text - text poloky
KeyCode - kombinace kláves pro vyvolání akce přísluné poloky, existují předdefinované konstanty kb... (kbAltX)
Command - příkaz který se má vygenerovat, je-li položka vybrána, příkazy jsou konstanty cmXXXX
Seznam, jeho odkaz se předává do konstruktoru jako parametr ADefs se vytvoří zřetězeným voláním funkcí:

```
Function NewStatusDef(AMin,AMax:word;AItems:PstatusItem;ANext:PsatusDef):PstatusDEf;
Function NewStatusKey(AText:String;AKeyCode:word;Acommand:word;ANext:PsatusItem):PstatusItem;
```

Zřetězení umoňují parametry ANext!

UDÁLOSTMI ŘÍZENÉ PROGRAMOVÁNÍ

DATOVÝ TYP TEVENT

Typ TEvent obsahuje informace o tom, o jaký typ události se jedná a některé dalí doplňkové informace, které jsou pro jednotlivé události odliné. **Je definován takto:**

```
Type TEvent=record
     what:word;
     case word of
           evNothong:();
           evMouse:(
                 Buttons:Byte;
                 Double:Boolean;
                 Where:TPoint;
                   )
           evKeyDown:(
                 case Integer of
                       0:(KeyCode:Word);
                       1:(CharCode:Char;
                          ScanCode:Byte));
           evMessage:(
                 Command:word;
                 Case word of
                       0:(InfoPtr:Pointer);
                       1:(InfoLong:LongInt);
                       2:(InfoWord:Word);
```

```
                    3:(InfoInt:Integer);
                    4:(InfoByte:Byte);
                    5:(InfoChar:Char));
  End;
```

Události jsou předdefinovány pomocí konstant **evXXXX**. Turbo Vision rozliují čtyři typy událostí. **Konstanta EvMessage** označuje událost, která je vyvolána nějakým objektem aplikace.

PRÁZDNÁ UDÁLOST EVNOTHING

Pokud položka **what** datové struktury *TEvent* obsahuje konstantu **evNothing**, znamená to, e ostatní položky neobsahují ádnou informaci. Prázdná událost vznikne z neprázdné tak, e některý objekt zpracuje událost a nastaví poloku what na evNothing, aby u na ni nemohl reagovat ádný dalí objekt.

UDÁLOSTI OD MYI EVMOUSE

Lze rozdělit na čtyři podtypy označené konstantou
evMouseDown - stisknutí tlačítka
evMouseUp - uvolnění tlačítka
evMouseMove - změna pozice
evMouseAuto - periodicky generovaná událost při drení stisknutého tlačítka
evMouse je tedy maskou pro vechny tyto události a dovoluje nám zjistit jestli lo o akci od myi, nezávisle na jejím konkrétním charakteru:

```
if Event.What and evMouse <>0 then ...
```

UDÁLOST KLÁVESNICE EVKEYDOWN

Vzniká po stisku nějaké klávesy, doplňující informací je kód stisknuté klávesy. (KeyCode nebo CharCode, ScanCode)

UDÁLOST GENEROVANÁ OBJEKTEM EVMESSAGE

Má podtypy:
evCommand - událost spojená s vybráním položky v menu nebo stavovém řádku **evBroadcast** - událost ířící se zvlátním způsobem **Uivatelsky definované události**

ÍŘENÍ UDÁLOSTÍ V APLIKACI

KDE UDÁLOSTI VZNIKAJÍ

Pro zjiování události by měla existovat nějaká smyčka, která kontroluje zda k nějaké události dolo či nikoliv a zároveň by na ni reagovala. Takovou smyčku není třeba vytvářet. V Turbo Vision existuje a to v objektu TApplication, konkrétně v metodě **RUN**. **Kód Run je:**

```
var E:TEvent;
E.What:=evNothing;
repeat
  if E.What evNothing then EventError(E);
  GetEvent(E);
  HandleEvent(E);
until EndState<>Continue;
```

METODA GETEVENT

Zde dochází k testování uivatelského vstupu. Pokud k nějakému dojde, naplní se **parametr E:TEvent**.
TApplication.GetEvent je jediné místo, zabívající se fyzickým vstupem, proto ji musíme předefinovat, chceme-li změnit způsob zacházení se vstupním zařízením.

ÍŘENÍ UDÁLOSTÍ, METODA HANDLEEVENT

Pokud v GetEvent vznikne událost, je uloena do struktury E typu TEvent a předána metodě **HandleEvent.** Úkolem metody **TApplication.HandleEvent** je předání události objektu, kterému patří a její případné zpracování, pokud se zjistí, e událost

patří aplikaci (potomku TApplication)

ÍŘENÍ UDÁLOSTÍ OD MYI

Adresát události vzniklé stisknutím tlačítka myi je view nacházející se pod kurzorem myi v době vzniku události. Jednotlivá subview modálního view jsou procházena v obráceném pořadí ne byla vloena. (poslední vloené bude zkoumáno první) Toto uspořádání se zve z-order.

ZAMĚŘENÉ UDÁLOSTI

Jako *Focused* události se íří události z klávesnice a události typu příkaz (vybrání položkyz menu). Adresátem je vdy zaměřené view=aktivní. Pokud po prohledání vech subview se neúspěně vrátí událost k modálnímu view (od kterého začalo prohledávání) je zavolána metoda EventError a událost zaniká. **EventError** je prázdná, lze ji předefinovat.

UDÁLOST TYPU BROADCAST

Události typu evBroadcast a uivatelsky definované. Není přesně určen adresát, proto jsou události předány vem subview modálního view. Pouívají se k vzájemné komunikaci mezi objekty.

ZPRACOVÁNÍ UDÁLOSTÍ

Automatickou reakci na standardní události zabezpečuje metoda HandleEvent. Např. okno reaguje na událost typu broadcast, e se má zavřít, změnit velikost a podobně. HandleEvent lze předefinovat, ale ztratíme automatické zpracování těchto událostí.

Po zpracování události je třeba ji vyčistit. Pomocí metody **ClearEvent** se nastaví

```
Event.What:=evNothing
```

a do **event.Info se zapíe @Self** - označení události jako zpracované.

DOPLNĚNÍ PROGRAMU ABY REAGOVAL NA PŘÍKAZY (TV3.PAS)

```
uses app, views, objects, menus, drivers, dialogs;

const cmSesNab=100;
      cmUkNab=101;
      cmZrnab=102;
      cmIMonitory=103;
      cmIDisky=104;
      cmIKlavesnice=105;
      cmOMonitory=106;
      cmOdisky=107;
      cmOKlavesnice=108;
      cmMore=109;
type pDealer=^Tdealer;
     tdealer=object(Tapplication)
             procedure initMenuBar;virtual;
             procedure initStatusLine;virtual;
             procedure HandleEvent(var event:TEvent);virtual;
             procedure SestavNabidku;
             Procedure UkazNabidku;
             Procedure ZrusNabidku;
             procedure Vstup(ceho:string);
             procedure vystup(ceho:string);
     end;

procedure tDealer.initMenuBar;
var r:tRect;
begin
  getExtent(r);
  r.b.y:=r.a.y+1;
  menubar:=new(pMENUbar, init(r, newmenu(
    NEWSUBmenu('~N~abidka',hcNOcontext,NEWMENU(
      newItem('~S~estav','',0,cmsesnab, hcNocontext,
      newItem('~U~kaz','',0,cmuknab, hcNocontext,
      newItem('~Z~rus','',0,cmzrnab, hcNocontext,
      newLine(
      newItem('~K~onec','ALT-X',kbALTX,cmQuit, hcNocontext,
      nil)))))
      ),
```

```
    newSubMenu('Vstupy',hcNOcontext,newMenu(
      newItem('Monitory','',0,cmImonitory, hcNocontext,
      newItem('Disky','',0,cmIdisky, hcNocontext,
      newItem('Klavesnice','',0,cmIklavesnice, hcNocontext,
      nil)))),
    newSubMenu('Vystupy',hcNOcontext, newMenu(
      newItem('Monitory','',0,cmOmonitory, hcNocontext,
      newItem('Disky','',0,cmOdisky, hcNocontext,
      newItem('Klavesnice','',0,cmOklavesnice, hcNocontext,
      nil)))),
    NIL))))
    ));
  end;

Procedure tDealer.initStatusLine;
var r:trect;
begin
  getExtent(r);
  r.a.y:=r.b.y-1;
  statusLine:=new(pStatusLine, Init(r,
    newStatusDef(0,$FFFF,
    newStatusKey('~ALT-X~ Exit',kbAltX,cmQuit,
    newStatusKey('~F1~ Help',kbF1,cmHelp,
    newStatusKey('~F2~ Nabidka',kbF2,cmSEsNab,
    newStatusKey('~F10~ Menu',kbF10,cmMenu,
    nil)))),
    nil)
  ));
end;

Procedure TDealer.HandleEvent(var Event:TEvent);
begin
  Tapplication.handleEvent(Event);
  case event.command of
    cmSesNab         :sestavNabidku;
    cmUkNab          :UkazNabidku;
    cmZrNab          :ZrusNabidku;
    cmImonitory      :Vstup('Monitory');
    cmIdisky         :Vstup('Disky');
    cmIKlavesnice    :Vstup('Klavesnice');
    cmOmonitory      :Vystup('Monitory');
    cmODisky         :Vystup('Disky');
    cmOKlavesnice    :Vystup('Klavesnice');
  else exit;
  end;
  ClearEvent(event);
end;

Procedure Tdealer.sestavNabidku;
begin
end;

Procedure Tdealer.zrusNabidku;
begin
end;

Procedure Tdealer.ukazNabidku;
begin
end;

Procedure Tdealer.vstup(ceho:string);
begin
end;

Procedure Tdealer.vystup(ceho:string);
begin
end;


VAR a:tdealer;
BEGIN
  a.init;
  a.run;
  a.done;
END.
```

Odezva na vechny události typu příkaz je umístěna v **Tdealer.handleEvent**. Pokud nebude na obrazovce ádný modální dialog-box, pak budou vechny příkazy zpracovávány tímto objektem (potomek TApplication)

POZNÁMKY KE ZPRACOVÁNÍ UDÁLOSTÍ

MASKOVÁNÍ UDÁLOSTÍ

Kadé view obsahuje poloku **EventMask**, její jednotlivé bity korespondují s bity položky TEvent.What. Nastavení bitu do 1 znamené, e view bude danou událost zpracovávat, 0 - ignorace události

ZMĚNA POŘADÍ PŘEDÁVÁNÍ ZAMĚŘENÝCH UDÁLOSTÍ

Kadé view má poloku **OPTION**. Pokud v této poloce nastavíme bit **ofPreProcess** - pak toto view dostane zaměřenou událost dříve ne zaměřené view **ofPostProcess** - dostane událost a po zaměřeném view. View zjituje v jaké fázi událost dostává pomocí položky**PHASE** svého vlastníka (GROUP ve které je vloeno). **PHASE nabývá hotnoty: phFocused, phPreProcess, phPostProcess**

ZÁKAZ A POVOLENÍ PŘÍKAZŮ

Příkazy s hodnotou 0..255 mohou být zakázány a následně povoleny.
Zakázaný příkaz je uivateli nedostupný (edá položka v menu).
Provádí se pomocí metod: **TView.DisableCommands** a **TView.EnableCommands**.
Obě mají jediný parametr **TCommandSet** - mnoina příkazů. které se mají zakázat či povolit.

UIVATELSKY DEFINOVANÉ UDÁLOSTI

Nejvyích est bitů **TEvent.What** lze vyuít pro zavedení nových kategorií událostí. Pak je s nimi zacházeno jako s typem evBroadcast. Případně lze nastavit typ událostí, nastavením bitů v proměnné **PositionalEvents** nebo **FocusedEvents** (poziční nebo zaměřená událost)

KOMUNIKACE MEZI OBJEKTY

Vzájemná komunikace se pouívá zřídka a značí patně navrenou strukruru programu! K zasílání zpráv slouí funcke message:

```
Function Message(receiver:PView;What,Command:Word;InfoPtr:Pointer):pointer;
```

Receiver - označení příjemce zprávy
What - druh zprávy
Command - příkaz
InfoPtr - pomacná informace

Je-li událost úspně zpracována, funkce vrátí ukazatel na objekt, který ji zpracoval.

METODA TAPPLICATION.IDLE

Je vyvolána metodou **tApplication.GetEvent**, kdykoliv není k dispozici ádná událost. Její tělo je prázdné. Lze ji proto předefinovat a vyuít např. ke kontrole heapu... je spoutěna pravidelně, ani by zpomalovala běh programu, protoe se spoutí, kdy se ádná práce nevyaduje.

ZOBRAZITELNÉ OBJEKTY - VIEWS

CO JSOU TO VIEWS

Označujeme tak kadý objekt, který mue být zobrazen na obrazovce. Je to jediná cesta pro výstup na obrazovku v Turbo Vision. View jsou vechny třídy objektů odvozené od třídy TView. TView je abstraktní objekt, předek pro vechna view. Obsahuje vlastnosti pro vechna view společné.

Hiearchie Objektů: - hiearchický strom objektů

ZÁKLADNÍ FUNKCE VIEW

View zabírá určitou pravoúhlou plochu a jeho úkolem je zpracovávat vechny události, které vzniknou na této ploe.
Musí poskytovat funkce:

- Vykreslení obsahu celé své plochy nebo její části v okamiku poadavku. Způsob vykreslení je určen metodou **DRAW**
- Zpracování jemu určených událostí - reakce na události je určena metodou **HandleEvent**

DATOVÉ POLOŽKYTŘÍDY TVIEW

POLOŽKYORIGIN A SIZE

Kadé view si musí pamatovat, kde je na obrazovce umístěno. Tato informace je umístěna ve dvou datových polokách typu TPoint.
položka **Origin** udává polohu horního levého rohu vzhledem k vlastníkovi view.
položka **Size** udává souřadnice pravého dolního rohu vzhledem k k Origin
Souřadnice ba textové obrazovce jsou chápány jako hranice mezi jednotlivými znaky a nikoliv jako znaky samotné.
Levý horní roh má souřadnice (0,0) a (0,0,0,0) je jeden bod.
Hranice mezi prvním a druhým znakem má souřadnici jedna, a (0,0,1,1) je plocha ohraničující jeden, levý horní, znak na obrazovce.

POLOŽKA OPTIONS

TView obsahuje pět bitově orientovaných datových poloek, ovlivňujících nebo informujících o chování view v různých situacích.
Poloky:
EventMask - určuje na které události bude view reagovat a které bude ignorovat

Options:	**Bit 0 - ofSelectable**	uivatel může vybrat view pomocí myi nebo TAB
	Bit 1 - ofTopSelect	po vybrání je view přesunuto nad ostatní
	Bit 2 - ofFirstClick	view je vybráno a reaguje na první kliknutí
	Bit 3 - ofFramed	view má viditelný rámeček
	Bit 4 - ofPreProcess	
	Bit 5 - ofPostProcess	
	Bit 6 - ofBuffered	vykreslení view se děje pomocí bufferu - po prvním vykreslení je uloeno do bufferu, a rychleji se poté vykresluje.Pouze při dostatku paměti.
	Bit 7 - ofTileable	okno může být uspořádáno systémem dladic. Jinak zůstane na svém míste. Uspořádání se děje metodami - **TDeskTop.Tile, TDesktop.Cascade**
	Bit 8 - ofCenterX	centrování ve směru x vzhledem k nějaké group
	Bit 9 - ofCenterY	
	Bit 10 - ofCentered	

POLOŽKA GROWMODE

Určuje jak se bude měnit velikost view, změní-li se velikost vlastníka (TGroup)

Bit 0	**gfGrowLoX**	levá strana view má konstantní vzdálenost od levé strany vlastníka
Bit 1	**gfGrowLoY**	horní strana view má konst. vzdálenost od horní strany vlastníka
Bit 2	**gfGrowHiX**	Pravá strana ...
Bit 3	**ghGrowHiY**	Dolní strana ...
Bit 4		Okno si udruje stálou relativí velikost vzhledem k vlastníkovi.

První čtyři bity lze nastavit najednou pomocí masky **gfGrowAll**

POLOŽKA DRAGMODE

Určuje chování view při pokusu o jeho taení pomocí myi.

Bit 0 **dmDragMove** Okno lze táhnout po kliknutí na horní okraj rámečku.
Bit 1 **dmDragGrow** Můeme měnit velikost view
Bit 4 **dmLimitLoX** levá strana view nesmí opustit plochu vlastníka
Bit 5 **dmLimitLoY** horní
BIt 6 **dmLimitHiX** pravá
Bit 7 **dmLimitHiY** dolní

Poslední čtyři bity lze nastavit najednou pomocí masky **dmLimitAll**

POLOŽKA STATE

Je určena pouze ke čtení, ke změně obsahu se pouívá metoda **SetState**.

```
procedure TView.SetState(AState:Wors;Enable:Boolean);virtual;
```

AState - bitová maska, odpovídající jednotlivým bitům položkyState
Enable - Určuje mají-li bity nastavené maskou být nastaveny či nulovány. Metoda je často přepisována kvůli implementaci vlastního chování. V odvozené metodě je vdy nutno volat původní metodu. Nová metoda tedy vypadá přiblině.:

```
Procedure MyView.SetState(AState:word;Enable:Boolean);
Begin
  TView.SetState(AState,enable);
  ---vlastní akce na základě hodnoty AState;---
End;
```

Obsah položky**State** není jednoznačný, ale mění se během činnosti. Jednotlivé bity:

Bit 0	**sfVisible**	určuje, je-li view viditelné v rámci vlastníka. Nastavuje se metodami **TView.Show, TView.Hide**
Bit 1	**sfCursorVis**	nastavuje viditelnost kurzoru view, metodami **TView.ShowCursor, TView.HideCursor**
Bit 2	**sfCursorIns**	kurzor blokový a nulový, ve tvaru podtrítka. Metodami **TView.BlockCursor, TView.MormalCursor**
Bit 3	**sfShadow**	určuje stíňování view
Bit 4	**sfActive**	nastaven, jedná-li se o aktivní okno nebo subview aktivního okna
Bit 5	**sfSelected**	nastaveno, je-li view právě vybrané subview v rámci svého vlastníka
BIt 6	**sfFocused**	nastaveno u zaměřeného view
Bit 7	**sfDragging**	je jedničkový při taení okna
Bit 8	**sfDisabled**	jedničkový při zakázání view
Bit 9	**sfModal**	nastaven pokud je view v modálním stavu
Bit 10	**sfExposed**	určuje, mue-li být view viditelné (je-li vlastněno aplikací). Vyuíván metodou **TView.Exposed**

VIEW TŘÍDY TGROUP

Sloitější view jako okna nebo dialog-boxy jsou tvořeny pomocí objektů jejich společným předkem je třída TGroup. Tato view jsou sloena z jednoduích view - jsou jejich vlastníky. Vlastněná view zveme **SubView**.
Typ TGroup musí obsahovat prostředky pro mechanismus vastnění a správu subview. Mezi tyto prostředky patří datové položky**Current** a **Last**, které jsou ukazateli na aktuální a poslední poloku v řetězci subView.
Dalím prostředkem je položka **Owner**, popisující vlastníka, má funkci ukazatele na vlastníka.

Proces vytváření group nazýváme vkládáním view do group. Provádíme jej pomocí metod:

```
procedure TGroup.Insert(P:PVIew);

procedure TGroup.Delete(P:PView);

function TGroup.ExecView(P:PView):word;
```

Metoda INSERT vloí view určené parametrem P do skupiny
metoda DELERE vyjme view určené parametrem P ze skupiny
(Z-order)

metoda ExecView se vyuívá pro sputění view v modálním stavu. Volá metodu **Execute** daného view a čeká na její dokončení a její návratovou hodnotu převezme jako svoji návratovou hodnotu. Při neúspěchu vrací *cmCancel*

Modální Stav - vechny události jsou směrovány do view v tomto stavu. ádný ovládací prvek mimo toto view nefunguje, dostupné jsou jen subview tohoto view. Modální reim lze zruit stiskem Esc

BAREVNOST VIEW, BAREVNÉ PALETY

Typ TView má metodu nazvanou **GetPalette**, která vrací ukazatel na string, který je chápán jako pole čísel proměnné délky. Toto pole se zve barevná paleta daného view a určuje jak bude view barevně vykresleno.
Jednotlivá čísla nejsou kódy byrev, ale indexy do barevné palety vlastníka, která obsahuje opět index...a do kořene - objektu TApplication.
Barevná paleta objektů TApplication je pole skutečných textových atributů.
Např.: **TScroller** (vkládá se do objektu TWindow a slouí jako zobrazitel textu) - má barevnou paletu o dvou barvách - barva normálního textu a barva zvýrazněného textu. Tyto hodnoty slouí jako indexy do osmimístné palety objektu TWindow, její jednotlivé položky mají význam např. barvy rámečku pasivního, aktivního okna, barvu ikony...

Jak změnit barvu objektu?

- Přepsáním přísluné položkyv paletě objektu TApplication - tím se změní vzhled vech view, kvůli mapování
- Přepsáním metody *GetPallete* v následníku nějakéhio view. Tím změníme pouze mapování a ne absolutní barvy.

Příklad předefinování palety:

```
Procedure TMyScroller.GetPallete:PPAllete;
const
  CMyScroller=#1#7;
  PMyScroller:string[Length(CMScroller)]=CMyScroller;
BEGIn
  GetPallete:=@PMyScroller;
End;
```

VYUŽITÍ KONKRÉTNÍCH TYPŮ VIEW

DOPLNĚNÍ PROGRAMU (TV4.PAS)

Program bude umět přidávat položkydo jednotlivých nabídek.

```
uses app, views, objects, menus, drivers, dialogs;

const cmSesNab=100;
      cmUkNab=101;
      cmZrnab=102;
      cmIMonitory=103;
      cmIDisky=104;
      cmIKlavesnice=105;
      cmOMonitory=106;
      cmOdisky=107;
      cmOKlavesnice=108;
      cmMore=109;
type PVstupDialog=^TVstupDialog;
     TVstupDialog=Object(TDialog)
       Pol:array [1..7,1..2] of PinputLine;
       constructor init(var Bounds:TRect;ATitle:TTitleStr);
       destructor done;virtual;
       procedure handleEvent(var event:Tevent);virtual;
     end;
     TVstupDialogRec=record
       IL:Array [1..7] of
       record
         Na:String[50];
```

```
        Ce:String[8];
      end;
    end;
    pDealer=^Tdealer;
    tdealer=object(Tapplication)
            procedure initMenuBar;virtual;
            procedure initStatusLine;virtual;
            procedure HandleEvent(var event:TEvent);virtual;
            procedure SestavNabidku;
            Procedure UkazNabidku;
            Procedure ZrusNabidku;
            procedure Vstup(ceho:string);
            procedure vystup(ceho:string);
    end;

Const NVstupDialogRec:TvstupDialogRec=(
      IL:((Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''))
      );


procedure tDealer.initMenuBar;
var r:tRect;
begin
  getExtent(r);
  r.b.y:=r.a.y+1;
  menubar:=new(pMENUbar, init(r, newmenu(
    NEWSUBmenu('~N~abidka',hcNOcontext,NEWMENU(
      newItem('~S~estav','',0,cmsesnab, hcNocontext,
      newItem('~U~kaz','',0,cmuknab, hcNocontext,
      newItem('~Z~rus','',0,cmzrnab, hcNocontext,
      newLine(
      newItem('~K~onec','ALT-X',kbALTX,cmQuit, hcNocontext,
      nil)))))
      ),
    newSubMenu('Vstupy',hcNOcontext,newMenu(
      newItem('Monitory','',0,cmImonitory, hcNocontext,
      newItem('Disky','',0,cmIdisky, hcNocontext,
      newItem('Klavesnice','',0,cmIklavesnice, hcNocontext,
      nil)))),
    newSubMenu('Vystupy',hcNOcontext, newMenu(
      newItem('Monitory','',0,cmOmonitory, hcNocontext,
      newItem('Disky','',0,cmOdisky, hcNocontext,
      newItem('Klavesnice','',0,cmOklavesnice, hcNocontext,
      nil)))),
    NIL))))
    ));
  end;

Procedure tDealer.initStatusLine;
var r:trect;
begin
  getExtent(r);
  r.a.y:=r.b.y-1;
  statusLine:=new(pStatusLine, Init(r,
    newStatusDef(0,$FFFF,
    newStatusKey('~ALT-X~ Exit',kbAltX,cmQuit,
    newStatusKey('~F1~ Help',kbF1,cmHelp,
    newStatusKey('~F2~ Nabidka',kbF2,cmSEsNab,
    newStatusKey('~F10~ Menu',kbF10,cmMenu,
    nil)))),
    nil)
  ));
end;

Procedure TDealer.HandleEvent(var Event:TEvent);
begin
  Tapplication.handleEvent(Event);
  case event.command of
    cmSesNab         :sestavNabidku;
    cmUkNab          :UkazNabidku;
    cmZrNab          :ZrusNabidku;
    cmImonitory      :Vstup('Monitory');
    cmIdisky         :Vstup('Disky');
    cmIKlavesnice    :Vstup('Klavesnice');
    cmOmonitory      :Vystup('Monitory');
    cmODisky         :Vystup('Disky');
    cmOKlavesnice    :Vystup('Klavesnice');
  else exit;
  end;
```

```
  ClearEvent(event);
end;

Procedure Tdealer.sestavNabidku;
begin
end;

Procedure Tdealer.zrusNabidku;
begin
end;

Procedure Tdealer.ukazNabidku;
begin
end;

Procedure Tdealer.vstup(ceho:string);
var r:TRect;
    okno:PVstupDialog;
begin
  r.assign(5,5,65,18);
  okno:=new(PVstupDialog,Init(r,Ceho));
  desktop^.execView(okno);
  Dispose(okno,done);
end;

Constructor TVstupDialog.Init(var Bounds:Trect;ATitle:TTitleStr);
Var R:Trect;
    i,j:byte;
    pom:PButton;
    Code,info:integer;
BEGIN
  TDialog.Init(Bounds, ATitle);
  GetExtent(r);
  R.Assign(R.A.X+2, R.A.Y+10, R.A.X+17,R.A.Y+12);
  pom:=New(PButton, init(R,'~M~ore',cmMore, bfNormal));
  Insert(pom);
  pom:=New(PButton, init(R,'~O~k',cmOk, bfNormal));
  Insert(pom);
  R.Move(16,0);
  pom:=New(PButton, init(R,'~C~ancel',cmCancel, bfNormal));
  Insert(pom);

  GetExtent(R);
  Bounds.Copy(R);
  R.Assign(R.A.X+2, R.A.Y+8, R.B.X-10, R.A.Y+9);

  Bounds.assign(bounds.B.X-9, Bounds.A.Y+8, bounds.B.X-2, Bounds.A.Y+9);
  For i:=1 to 7 do
  BEGIN
    Pol[i,1]:=New(PInputLine,Init(R,50));
    Pol[i,2]:=New(PInputLine,Init(Bounds,8));
    R.Move(0,-1);
  END;
  For i:=7 Downto 1 do
  BEGIN
    Insert(pol[i,1]);
    Insert(pol[i,2]);
  END;
  POL[7,1]^.select;
  Insert(New(PLabel,Init(R,' ~N~ zev', Pol[7,1])));
  Insert(New(PLabel,Init(Bounds,' ~C~ena', Pol[7,2])));
end;

Destructor TVstupDialog.Done;
var i,j:byte;
Begin
  for i:=1 to 6 do
    for j:=1 to 2 do
      Dispose(pol[i,j],Done);
  TDialog.done;
end;

Procedure TVstupDialog.HandleEvent(var Event:TEvent);
  Procedure zapis;
  var data:TVstupDialogRec;
  begin
    data:=NVstupDialogRec;
    GetData(Data);
    setData(NVstupDialogRec);
  End;
  function Najdi_select:Boolean;
```

```
  var i,j:byte;
  Begin
    i:=1;j:=1;
    while (i<7) and not(pol[i,j]^.GetState(sfSelected))
    do if j=2 then
    begin
      j:=1;
      inc(i);
    end else inc(j);
    Najdi_select:=(i<7);
  end;
Begin
  if (event.what=evBroadCast) and (event.Command=cmDefault) then
  begin
    if najdi_select then selectNext(false)
    else TDialog.handleEvent(Event);
  end else
  begin
    if (event.what=evCommand) then
    begin
      if (event.command=cmMore) or (event.command=cmOk) then
      begin
        zapis;
        pol[7,1]^.select;
      end;
    end;
    TDialog.HandleEvent(event);
    end;
    ClearEvent(Event);
  end;


Procedure Tdealer.vystup(ceho:string);
begin
end;


VAR a:tdealer;
BEGIN
  a.init;
  a.run;
  a.done;
END.
```

VYTVOŘENÍ OBJEKTU TYPU TGROUP

Budeme rozebírat konstruktor objektu TVstupDialog, metodu Init.
Nejdříve zavoláme konstruktor předka - onjektu TDialog, který provede standardní inicializace, neboli vytvoří přísluné okénko, a jako titulek mu dá string, který dostane jako druhý parametr.
Pak následuje vlastní vytváření group - posloupnost volání **New** pro vytvoření subview, Insert pro vloení tohoto subview do group a manipulace s proměnnými typu TRect, které obsahují souřadnice jednotlivých subview.

OBJEKTY TŘÍDY TBUTTON

Jsou koncová view - nemůou vlastnit subview, pouívají se jako součást dialog-boxů. Slouí zde jako prvek generující událost typu příkaz. Konstruktor objektu vypadá následovně:

```
constructor TButton.Init(var Bounds:TRect;ATitle:TTitleStr;Acommand:word;AFlags:Byte);
```

Bounds - souřadnice buttonu na obrazovce
ACommand - příkaz, který má být při stisku generován
AFlags - příznaky ovlivňující vzhled a chování buttonu -*bfNormal* - normální button *bfDefault* - stisk Enter = stisk tohoto default buttonu *blLeftJust* - zarovnání titulku v buttonu doleva.

OBJEKTY TŘÍDY TINPUTLINE

```
constructor TInputLine.Init(var Bounds:TRect;AMaxLen:Integer);
```

Bounds - souřadnice vstupní řádky na obrazovce
AMaxLen - maximální délka editovaného řetězce
Pro zápis/čtení řetězce slouí dvě metody:

```
Procedure TInputLine.GetData(var Rec);virtual;

procedure TInputLine.SetData(var Rec);virtual;
```

OBJEKTY TŘÍDY TLABEL

Jedná se o statický text, který může být zaměřen. Obsahuje datovou položky**Link** typu PView a ukazuje na view, s ním je daný TLabel spojen.

```
constructor TLabel.Init(vat Bounds:Trect;AText:String;ALink:PView);
```

Bounds - souřadnice labelu na obrazovce
AText - text labelu
ALink - ukazatel na připojené view

ZRUENÍ OBJEKTU TGROUP

Protoe v naem objektu máme pevné odkazy na dynamicky alokované subview, musíme před zruením objektu uvolnit tato subview, protoe jinak bychom na ně ztratili odkazy a nebyli schopni uvolnění provést. Nejvhodnější místo na toto uvolnění jsou destruktory.

MODIFIKACE CHOVÁNÍ VIEW

Poud chceme nae view jinak uzpůsobit, uděláme to předefinováním metody **HandleEvent**

STREAMY - VSTUPNÍ A VÝSTUPNÍ OPERACE

Speciální prostředek, pro uchování objektů mimo uivatelský program (např. datové soubory), a prostředek pro správu vnějších pamětí. Streamy dokáí zvládnout libovolnou regulérní třídu = potomka třídy TObject Prostředky pro uloení dat do souborů nebo EMS paměti poskytují typy: TDosStrem, TBufStream, TEMSStream . Kadý objekt, který má být uloen do streamu, musí být zaregistrován. Stream je potřeba před pouitím otevřít, lze sekvenčně zapisovat, nebo pomocí speciálních funkcí přistupovat na vybrané místo souboru a po ukončení práce zavřít soubor.

OTEVŘENÍ STREAMU

Otevření souboru je jeho inicializace pro následující pouití. Je tedy logické, e v případě objektů tato úloha připadne konstruktoru. Syntaxe konstruktoru je odliná pro jednotlivé typy.

Typ **TDosStream** odpovídá souboru uloenému na disku. Konstruktor má tvar:

```
constructor TDosStream.Init(FileName:FnameStr;Mode:word);
```

FileName - jméno přísluiného souboru
Mode - způsob přístupu k danému souboru, předdefinovány konstanty

Typ **TBufStream** je diskový soubor, do kterého se přístup provádí přes buffer v operační paměti. Je vhodný při velkém mnostvím přístupů k objektům malé velikosti.

```
constructor TBufStream.Init(filename:FnameStr;mode, size:word);
```

Size - udává velikost bufferu v paměti

Typ **TEmsStream** slouí k ukládání objektů do expand paměti, můeme ho tedy vyuít ke zrychlení přistupu k často vyuívanému diskovému streamu tím, e tento stream zkopírujeme do streamu typu TEmsStream.

```
constructor TEmsStream.Init(MinSize:longint);
```

MinSize - minimální velikost streamu v bytech

KONSTANTY MÓDU OTEVŘENÍ STREAMU

Parametr **Mode** lze zadat pomocí předdefinovaných konstant:
stCreate - vytvoř nový diskový soubor
stOpenRead - otevři existující soubor pouze pro čtení
stOpenWrite - otevři existující soubor pouze pro zápis
stOpen - otevři existující soubor pro zápis i čtení

SEKVENČNÍ PŘÍSTUP KE STREAMU

Pro čtení a zápis objektů do a ze streamu se vyuívají metody **PUT, GET**

```
function TStream.Get:PObject;
```

Načte následující objekt ze streamum zavolá jeho konstruktor **Load** viz níe a vrátí ulazatel na tento nově vytvořený objekt.

```
procedure TStream.Put(P:PObject);
```

zapíe objekt na aktuální pozici ve streamu.

Čtené a zapisované objekty musí být registrované pro pouití ve streamu. Při této registraci je kadé třídě přiřazeno jednoznačné číslo, kreré je při zápisu objektu uloeno před vlastní posloupnost bytů.

NÁHODNÝ PŘÍSTUP KE STREAMU

```
function TStream.GetPos:longint;virtual;
function TStream.GetSize:longint;virtual;
Procedure TStream.Seek(pos:LongInt);virtual;
```

GetPos vrací aktuální pozici ve streamu v bytech od začátku streamu.
GetSize vrací aktuální délku streamu v bytech
Seek nastaví aktuální pozici (místo, na kterém se bude provádět přítí čtení nebo zápis) na *pos* bytů od začátku.

Tyto metody jsou definovány jako abstraktní, pro konkrétní realizaxi streamu musí být přepsány. Způsob realizace stramu toti závisí na fyzickém způsobu realizace streamu.

PŘÍPRAVA OBJEKTŮ PRO POUITÍ VE STREAMU

Jestlie chceme uloit nějaký objekt do streamu, musí tento objekt splňovat určité poadavky, take definujeme-li nový typ objektů, musíme splnění těchto poadavků zajistit.

METODY LOAD A STORE

U kadého objektu musíme přesně určit jak se má na stream ukládat a jak se má ze streamu čist. Toto lze učinit pomocí metod Load a Store, kdy při odvození nového objektu stačí k metodám předka čtení a zápis do nově přidaných poloek.
Schematicky lze tento postup zapsat takto:

```
type naslednik=object(predek)
 nova_polozka:jeji_typ;
 constructor Load(var S:Tstream);
 procedure strore(var S:TStream);
end;
constructor nasledovnik.Load(var S:TStream);
begin
  predek.Load(S);
  S.Read(nova_polozka,sizeOf(jeji_typ));
end;
Procedure naslednik.store(var S:Stream);
begin
  Predek.Store(S);
  S.Write(nova_polozka,sizeOf(Jeji_typ));
end;
```

V kadé metodě je nejprve třeba vyvolat metodu předka, která zařídí uloení/načtení jeho poloek. Poté pomocí metod *Read, Write* provedeme uloení/načtení poloek, které jsme do objektu přidali.
Metody *Load, Store* obvykle nepouíváme přímo, ale jsou volány metodami *Put, Get*

REGISTRACE OBJEKTU PRO POUITÍ VE STREAMU

Je třeba vyplnit registrační záznam objektu a předat jej proceduře **RegisterType**
Regisrační záznam:

```
Type TStreamRec= record
        objType:word;
        VmtLink:word;
        Load:pointer;
        Store:pointer;
        Next:word;
End;
```

Názvy záznamů začínají písmenem **R**.
ObjType - číselný identifikátor k rozlení objektů ve streamu. Tato čísla přiděluje programátor, moné hodnoty jsou 100..65535. Musí být zajitěna jejich bezkonfliknost a jednoznačnost.
VmtLink - zajiuje napojení na tabulku virtuálních metod VMT, její inicializace se provádí:

```
RMyObject.VmtLink:=Ofs(TypeOf(TMyObject)^);
```

Load, Store - ukazatele na metody stejného jména daného objektu, jejich inicializace:

```
RMyObject:=@TmyObject.Load;
RmyObject:=@TmyObject.Store;
```

Next - ukazatel na dalí v řetězci registračních záznamů, plněno automaticky v proceduře REGISTERTYPE.
Po naplnění poloek registračního záznamu je nutné tento záznam předat proceduře RegisterType. Toto předání je nutné provést před prvním pokusem uloit nebo načíst objekt ze streamu. Např.:

```
type TMyApplication = object(TApplication)
  constructor Init;
end;
Constructor TMyApplication.Init;
begin
  TApplication.Init;
```

```
  RegisterType(RMyObject);
End;
```

ZPRACOVÁNÍ CHYB PŘI PRÁCI SE STREAMY

Třída *TStream* definuje metodu **Error**, která je vyvolána pokadé, kdy dojde při práci se streamem k chybě. Tato metoda nastavuje pooku Status třídy TStream na jednu z následujících hodnot.
stOK - bez chyby
stError - chyba přístupu
stInitError - neúspěná inicializace
stReadError - pokus o čtení za koncem streamu
stWriteError - do streamu nelze zapisovat (málo paměti)
stGetError - pokus o Get nezaregistrovaného objektu
stPutError - pokus i Put nezaregistrovaného objektu

položka **ErrorInfo** pak obsahuje číselný identifikátor objektu v případě, e status má hodnotu stGetError. Je-li status stPutError pak, ErrorInfo obsahuje ofset tabulky VMT daného objektu.

UKLÁDÁNÍ POLOEK TYPU ODKAZ NA OBJEKT

Obsahuje-li objekt poloke, která je odkazem na nějaké subview, nelze ji ukládat běným způsobem. Musíme pouít speciální metodu **TGroup.GetSubViewPtr, TGroup.PutSubViewPtr**

```
constuctor nasednik.load(var S:TStream);
begin
...
  GetSubViewPrt(S,ukazatel);
end;
Procedure naslednik.store(var S:TStream);
begin
...
  PutSubViewPtr(S,ukazatel);
End;
```

PŘÍKLAD POUITÍ STREAMU (TV5.PAS)

*** Nefunguje, jsou zde chyby, ktere zatim nevím jak opravit... ***

V příkladu pouijeme stream pro uloení databáze výrobků. Kadý výrobek bude popsán svým názvem a cenou. Pro uchování těchto intormací je zavedena nová třída objektů *TPart*

```
uses app, views, objects, menus, drivers, dialogs;

const cmSesNab=100;
      cmUkNab=101;
      cmZrnab=102;
      cmIMonitory=103;
      cmIDisky=104;
      cmIKlavesnice=105;
      cmOMonitory=106;
      cmOdisky=107;
      cmOKlavesnice=108;
      cmMore=111;
      cmOprav=112;
      cmDelete=113;

type PVstupDialog=^TVstupDialog;

     TVstupDialog=Object(TDialog)
       PracFile:PbufStream;
       Pol:array [1..7,1..2] of PinputLine;
       constructor init(var Bounds:TRect;ATitle:TTitleStr);
       destructor done;virtual;
       procedure handleEvent(var event:Tevent);virtual;

     End;
```

```
     TVstupDialogRec=record
       IL:Array [1..7] of
       record
         Na:String[50];
         Ce:String[8];
       end;
     end;

     pDealer=^Tdealer;

     tdealer=object(Tapplication)
             procedure initMenuBar;virtual;
             procedure initStatusLine;virtual;
             procedure HandleEvent(var event:TEvent);virtual;
             procedure SestavNabidku;
             Procedure UkazNabidku;
             Procedure ZrusNabidku;
             procedure Vstup(ceho:string);
             procedure vystup(ceho:string);
             constructor init;
     end;



     PPart=^TPart;

     Tpart=object(TObject)
       Price:real;
       Name:string[50];
       constructor init(Aname:string;Aprice:real);
       constructor load(var s:tstream);
       procedure store(var s:tstream);virtual;
       procedure setname(aname:string);
       procedure setPrice(aprice:real);
       function getprice:real;
       function getname:string;
       function getString:string;
     end;

Const NVstupDialogRec:TvstupDialogRec=(
      IL:((Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''))
      );
      Rpart:tstreamRec=(
        objtype:100;
        vmtLink:ofs(typeOf(Tpart)^);
        load:@Tpart.load;
        Store:@Tpart.store);
        sizeOfTpart=56;

Constructor Tdealer.init;
begin
  TApplication.init;
  registerType(RPart);
end;


procedure tDealer.initMenuBar;
var r:tRect;
begin
  getExtent(r);
  r.b.y:=r.a.y+1;
  menubar:=new(pMENUbar, init(r, newmenu(
    NEWSUBmenu('~N~abidka',hcNOcontext,NEWMENU(
      newItem('~S~estav','',0,cmsesnab, hcNocontext,
      newItem('~U~kaz','',0,cmuknab, hcNocontext,
      newItem('~Z~rus','',0,cmzrnab, hcNocontext,
      newLine(
      newItem('~K~onec','ALT-X',kbALTX,cmQuit, hcNocontext,
      nil)))))
      ),
    newSubMenu('Vstupy',hcNOcontext,newMenu(
      newItem('Monitory','',0,cmImonitory, hcNocontext,
      newItem('Disky','',0,cmIdisky, hcNocontext,
      newItem('Klavesnice','',0,cmIklavesnice, hcNocontext,
      nil)))),
    newSubMenu('Vystupy',hcNOcontext, newMenu(
      newItem('Monitory','',0,cmOmonitory, hcNocontext,
      newItem('Disky','',0,cmOdisky, hcNocontext,
```

```
      newItem('Klavesnice','',0,cmOklavesnice, hcNocontext,
      nil)))),
    NIL))))
    ));
  end;

Procedure tDealer.initStatusLine;
var r:trect;
begin
  getExtent(r);
  r.a.y:=r.b.y-1;
  statusLine:=new(pStatusLine, Init(r,
    newStatusDef(0,$FFFF,
    newStatusKey('~ALT-X~ Exit',kbAltX,cmQuit,
    newStatusKey('~F1~ Help',kbF1,cmHelp,
    newStatusKey('~F2~ Nabidka',kbF2,cmSEsNab,
    newStatusKey('~F10~ Menu',kbF10,cmMenu,
    nil)))),
    nil)
  ));
end;

Procedure TDealer.HandleEvent(var Event:TEvent);
begin
  Tapplication.handleEvent(Event);
  case event.command of
    cmSesNab          :sestavNabidku;
    cmUkNab           :UkazNabidku;
    cmZrNab           :ZrusNabidku;
    cmImonitory       :Vstup('Monitory');
    cmIdisky          :Vstup('Disky');
    cmIKlavesnice     :Vstup('Klavesnice');
    cmOmonitory       :Vystup('Monitory');
    cmODisky          :Vystup('Disky');
    cmOKlavesnice     :Vystup('Klavesnice');
  else exit;
  end;
  ClearEvent(event);
end;

Procedure Tdealer.sestavNabidku;
begin
end;

Procedure Tdealer.zrusNabidku;
begin
end;

Procedure Tdealer.ukazNabidku;
begin
end;

Procedure Tdealer.vstup(ceho:string);
var r:TRect;
    okno:PVstupDialog;
    path:string;
begin
  r.assign(5,3,73,18);
  okno:=new(PVstupDialog,Init(r,True));
  path:=ceho;
  if length(path)>8 then
    system.delete(path,8,length(path)-8);
  path:=concat(path,'.dta');
  okno^.natahni(path,ceho);
  desktop^.execView(okno);
  Dispose(okno,done);
end;

Constructor TVstupDialog.Init(var Bounds:Trect;ATitle:TTitleStr);
Var R:Trect;
    i,j:byte;
    pom:PButton;
    Code,info:integer;
    path:string;
BEGIN
  TDialog.Init(Bounds, ATitle);
  GetExtent(r);
  R.Assign(R.A.X+2, R.A.Y+10, R.A.X+17,R.A.Y+12);
  pom:=New(PButton, init(R,'~M~ore',cmMore, bfNormal));
  Insert(pom);
  R.Move(16,0);
```

```
  pom:=New(PButton, init(R,'~O~k',cmOk, bfNormal));
  Insert(pom);
  R.Move(16,0);
  pom:=New(PButton, init(R,'~C~ancel',cmCancel, bfNormal));
  Insert(pom);

  GetExtent(R);
  Bounds.Copy(R);
  R.Assign(R.A.X+2, R.A.Y+8, R.B.X-10, R.A.Y+9);

  Bounds.assign(bounds.B.X-9, Bounds.A.Y+8, bounds.B.X-2, Bounds.A.Y+9);
  For i:=1 to 7 do
  BEGIN
    Pol[i,1]:=New(PInputLine,Init(R,50));
    Pol[i,2]:=New(PInputLine,Init(Bounds,8));
    R.Move(0,-1);
  END;
  For i:=7 Downto 1 do
  BEGIN
    Insert(pol[i,1]);
    Insert(pol[i,2]);
  END;
  POL[7,1]^.select;
  Insert(New(PLabel,Init(R,' ~N~ázev', Pol[7,1])));
  Insert(New(PLabel,Init(Bounds,' ~C~ena', Pol[7,2])));

  if length(aTitle>8) then
     system.delete(atitle,8,length(atitle)-8);
  path:=concat(atitle,'.dta');
  pracFile:=new(pBufStream,init(path, stOpen, 512));
  with PracFile^ do
    if status stOk then
    begin
      Reset;
      init(path, stCreate, 512);
    end else seek(getsize);
end;

Destructor TVstupDialog.Done;
var i,j:byte;
Begin
  dispose(pracfile, done);
  for i:=1 to 6 do
    for j:=1 to 2 do
      Dispose(pol[i,j],Done);
  TDialog.done;
end;

Procedure TVstupDialog.HandleEvent(var Event:TEvent);
  Procedure zapis;
  var data:TVstupDialogRec;
  prac:PPart;
  num:real;
  code:integer;
  begin
    data:=NVstupDialogRec;
    GetData(Data);
    setData(NVstupDialogRec);
    i:=1;
    with data do
      while (i<7) and (il[i].na'') do
      begin
        val(il[i].ce,num,code);
        prac:=new(ppart, init(il[i].na,num));
        prac^.store(pracfile^);
        dispose(prac,done);
        ind(i);
    end;
  End;
  begin
  end;

  constructor Tpart.init(aname:string;aprice:real);
  begin
    tobject.init;
    name:=aname;
    price:=aprice;
  end;

  function Tpart.getprice:real;
  begin
```

```
    getPrice:=Price;
  end;

  function TPart.getName:string;
  begin
    getName:=name;
  end;

  Procedure TPart.setName(aname:string);
  begin
    name:=aname;
  end;

  Procedure Tpart.setPrice(aprice:real);
  begin
    price:=aPrice;
  end;

  function tpart.getstring:string;
  var a,b:string;
      l,hm:byte;
  begin
    str(price:8:2,A);
    b:=getname;
    hm:=length(b);
    if hm<50 then l:=hm to 49 do b:=concat(b,' ')
    else delete(b, 51, hm-50);
    getstring:=concat(b,' ',a,'Kcs');
  end;

  constructor Tpart.load(var s:tstream);
  begin
    s.read(name,sizeof(name));
    s.read(price, sizeOf(price));
  end;

  Procedure tPart.store(var s:tstream);
  begin
    s.write(name, sizeOf(name));
    s.write(price, sizeOf(price));
  end;

  function Najdi_select:Boolean;
  var i,j:byte;
  Begin
    i:=1;j:=1;
    while (i<7) and not(pol[i,j]^.GetState(sfSelected))
    do if j=2 then
    begin
      j:=1;
      inc(i);
    end else inc(j);
    Najdi_select:=(i<7);
  end;
Begin
  if (event.what=evBroadCast) and (event.Command=cmDefault) then
  begin
    if najdi_select then selectNext(false)
    else TDialog.handleEvent(Event);
  end else
  begin
    if (event.what=evCommand) then
    begin
      if (event.command=cmMore) or (event.command=cmOk) then
      begin
        zapis;
        pol[7,1]^.select;
      end;
    end;
    TDialog.HandleEvent(event);
    end;
    ClearEvent(Event);
  end;


Procedure Tdealer.vystup(ceho:string);
begin
end;


VAR a:tdealer;
```

```
BEGIN
  a.init;
  a.run;
  a.done;
END.
```

POUITÍ NĚKTERÝCH DALÍCH OBJEKTŮ TV

Máme hotovou část programu, která nám umoní zadávat data jednotlivých výrobků a uchovávat je v příslunych souborech na disku. Nyní vytvoříme prostředky pro prohlíení těchto dat.

TŘÍDA TLISTBOX A TSCROLLBAR

Třída **TListViewer**, která je předkem třídy TListBox, poskytuje prostředky pro zobrazení a správu obecných zřetězených seznamů poloek. Zobrazení tohoto seznamu mue být uivatelem řízeno pomocí jednoho nebo dvou skrolovacích sloupků - objektů třídy **TScrollBar**.
Nejprve tedy vytvoříme TScrollBar a ukazatel na něj se předá jako parametr konstruktoru objektu TListViewer. Tado dvě vytvořená view mezi sebou automaticky komunikují. Programátor tedy nemusí volat metody **TScrollBarr.SetParams, TScrollBarr.SetValue** a dalí, pokud nechce dosáhnout nějakého speciálního chování.
TListViewer definuje abstraktní metodu **GetText**, která určuje způsob, kterým budou jednotlivé oploky zobrazovaného seznamu vypsány na obrazovku. Pro pouitív programu je tato abstraktní metoda přepsána a tak je definován konkrétní postup vypsání kadé poloky.

TV samotné definují nsledníka TListViewer nazvaného **TListBox**, která implementuje asi nejobvyklejí typ seznamu, seznam řetězců. TListBox vypisuje položkyv jednom sloupci a je řízen jedním vertikálním skrolovacím sloupcem.
Jediné co lze v tomto standardnímu typu změnit je metoda **GetText**.
Vlastní zobrazovaný seznam je představován polokou **TListBox.List**, která je typu **PCollection**

DYNAMICKÁ POLYMORFNÍ POLE - TŘÍDA TCOLLECTION

Třída TCollection a její následníci **TSortedCollection a TStringCollection** jsou ze skupiny němých objektů. Nemohou se tedy být zobrazovány na obrazovce, ale jsou implementovány s cílem usnadnit řeení programátorské problematiky.

DYNAMICKÁ PROMĚNNÁ VELIKOSTI POLE

Vytvoření objektu TCollection:

```
constructor TCollection.Init(ALimit,ADelta:integer);
```

ALimit - pro kolik poloek se má na počátku vyhradit místo
ADelta - o kolik se rozsah zvětí je-li počet poloek překročen

Vkládání poloek:

```
Procedure TCollection.Insert(Item:Pointer);
```

Maximální počet poloek obsaených v TCollection je určen proměnnou **MaxCollectionSize**

ITERÁTORY POLE TCOLLECTION

Kromě dynamicky měnitelné velikosti a polymorfismu poskytuje TCollection navíc prostředky pro řeení dalího obvykle programového úkolu týkajícího se polí. Je to vykonání nějaké činnosti nade vemi položkami pole a najití první nebo poslední položkysplňukící určitou podmínku. Tyto problémy je mono jednodue řeit pomocí "iteračních" metod **ForEach, FirstThat,**

LastThat

METODA FOREACH

```
procedure TCollection.ForEach(Action:pointer);
```

Vykoná postupně akci určenou parametrem *Action*. Pouívá se následujícím způsobem:

```
Procedure Moje_Procedura(kol:PCollection);
 procedure AKCE(item:pointer);far;
   begin
     delej_neco_s(item);
   end;
 kol^.forEach(@Akce);
end;
```

Procedura její adresa je předávána jako parametr Action musí být obyčejná procedura - nesmí být metodou ádného objektu!! Dále musí být lokální vzhledem k proceduře, která ji volá. Musí být typu far.

METODY FIRSTTHAT A LASTTHAT

```
function TCollection.FIrstThat(Test:pointer):pointer;

function TCollection.LastThat(test:pointer);pointer;
```

Test - adresa funkce s návratovou hodnotou typu Boolean, vrací True pokud položka splňuje poadovanou podmínku
Způsob zápisu:

```
Procedure MojeProcedura(kol:PCollection);
var Prvni, posledni:pointer;
 function Podminka(pol:pointre):Boolean;far;
 begin
   podminka:=delej_neco_s(pol);
 end;
Begin
  Prvni:=firstThat(@podminka);
  posledni:=lastThat(@podminka);
end;
```

USPOŘÁDANÁ POLE - TSORTEDCOLLECTION

Třída TSortedCollection se pouívá jako pole, jeho položkyjsou uspořádány podle určitého klíče. Způsob uspořádání definujeme přepsáním metod:

```
function TSortedCollection.KeyOf(Item:pointer):pointer;virtual;
```

Má jediný parametr - ukazatel na poloku pole, jako náratovou hodnotu předává ukazatel na tu poloku pole, podle které se má provádět třídění.

```
function TSortedCollection.Compare(key1,key2:pointer):iteger;Virtual;
```

Parametry jsou dva ukazatele na klíče poloek pole. Vrací hodnotu 0 v případě rovnosti obou klíčů. -1 v případě e první klíč je mení ne druhý a 1 pokud je druhý klíč mení ne první.

```
Function TSortedCollection.Search(key:pointer;var index:integer):boolean;virtual;
```

Tato metoda byhldí poloku s klíčem **key**. Pokud je nalezena vrací True a v proměnné index nalezenou hodnotu poloky.

TStringCollection je přímý následník třídy TSortedCollection, poskytuje vechny její monosti. položkynejsou objekty ale stringy.

DOPLNĚNÍ PROGRAMU (TV6.PAS)

*** nefunguje - chyby :((***

```
uses app, views, objects, menus, drivers, dialogs;

const cmSesNab=100;
      cmUkNab=101;
      cmZrnab=102;
      cmIMonitory=103;
      cmIDisky=104;
      cmIKlavesnice=105;
      cmOMonitory=106;
      cmOdisky=107;
      cmOKlavesnice=108;
      cmMore=111;
      cmOprav=112;
      cmDelete=113;

type PVstupDialog=^TVstupDialog;

     TVstupDialog=Object(TDialog)
       PracFile:PbufStream;
       Pol:array [1..7,1..2] of PinputLine;
       constructor init(var Bounds:TRect;ATitle:TTitleStr);
       destructor done;virtual;
       procedure handleEvent(var event:Tevent);virtual;

     End;

     PMyListBox=^TMyListBox;
     TMyListBox=Object(TListBox)
       kol:PCollection;
       constructor init(var bounds:Trect;AnumCols:word;AScrollBar:PScrollBar;APath:string);
       destructor done;virtual;
       function GetText(item:integer;maxLen:integer):String;virtual;
     end;

     PTitulText=^TTitulText;
     TTitulText=Object(TstaticText)
       function GetPalette:ppalette;virtual;
     end;

     PVystupDialog=^TVystupDialog;
     TvystupDialog=Object(TDialog)
       list:pMyListBox;
       Titul:PtitulText;
       Obs:Boolean;
       Scrolbar:Pscrollbar;
       Constructor init(var bounds:Trect);
       destructor done;virtual;
       procedure natahni(path,name:String);
     end;

     TVstupDialogRec=record
       IL:Array [1..7] of
       record
         Na:String[50];
         Ce:String[8];
       end;
     end;

     pDealer=^Tdealer;

     tdealer=object(Tapplication)
             procedure initMenuBar;virtual;
             procedure initStatusLine;virtual;
             procedure HandleEvent(var event:TEvent);virtual;
             procedure SestavNabidku;
             Procedure UkazNabidku;
             Procedure ZrusNabidku;
             procedure Vstup(ceho:string);
             procedure vystup(ceho:string);
             constructor init;
     end;
```

```
     PPart=^TPart;

     Tpart=object(TObject)
       Price:real;
       Name:string[50];
       constructor init(Aname:string;Aprice:real);
       constructor load(var s:tstream);
       procedure store(var s:tstream);virtual;
       procedure setname(aname:string);
       procedure setPrice(aprice:real);
       function getprice:real;
       function getname:string;
       function getString:string;
     end;

Const NVstupDialogRec:TvstupDialogRec=(
      IL:((Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''),(Na:'';Ce:''),(Na:'';Ce:''),
          (Na:'';Ce:''))
      );
      Rpart:tstreamRec=(
        objtype:100;
        vmtLink:ofs(typeOf(Tpart)^);
        load:@Tpart.load;
        Store:@Tpart.store);
        sizeOfTpart=56;

Constructor Tdealer.init;
begin
  TApplication.init;
  registerType(RPart);
end;

procedure tDealer.initMenuBar;
var r:tRect;
begin
  getExtent(r);
  r.b.y:=r.a.y+1;
  menubar:=new(pMENUbar, init(r, newmenu(
    NEWSUBmenu('~N~abidka',hcNOcontext,NEWMENU(
      newItem('~S~estav','',0,cmsesnab, hcNocontext,
      newItem('~U~kaz','',0,cmuknab, hcNocontext,
      newItem('~Z~rus','',0,cmzrnab, hcNocontext,
      newLine(
      newItem('~K~onec','ALT-X',kbALTX,cmQuit, hcNocontext,
      nil)))))
      ),
    newSubMenu('Vstupy',hcNOcontext,newMenu(
      newItem('Monitory','',0,cmImonitory, hcNocontext,
      newItem('Disky','',0,cmIdisky, hcNocontext,
      newItem('Klavesnice','',0,cmIklavesnice, hcNocontext,
      nil)))),
    newSubMenu('Vystupy',hcNOcontext, newMenu(
      newItem('Monitory','',0,cmOmonitory, hcNocontext,
      newItem('Disky','',0,cmOdisky, hcNocontext,
      newItem('Klavesnice','',0,cmOklavesnice, hcNocontext,
      nil)))),
    NIL))))
    ));
  end;

Procedure tDealer.initStatusLine;
var r:trect;
begin
  getExtent(r);
  r.a.y:=r.b.y-1;
  statusLine:=new(pStatusLine, Init(r,
    newStatusDef(0,$FFFF,
    newStatusKey('~ALT-X~ Exit',kbAltX,cmQuit,
    newStatusKey('~F1~ Help',kbF1,cmHelp,
    newStatusKey('~F2~ Nabidka',kbF2,cmSEsNab,
    newStatusKey('~F10~ Menu',kbF10,cmMenu,
    nil)))),
    nil)
  ));
end;

Procedure TDealer.HandleEvent(var Event:TEvent);
begin
  Tapplication.handleEvent(Event);
```

```
  case event.command of
    cmSesNab         :sestavNabidku;
    cmUkNab          :UkazNabidku;
    cmZrNab          :ZrusNabidku;
    cmImonitory      :Vstup('Monitory');
    cmIdisky         :Vstup('Disky');
    cmIKlavesnice    :Vstup('Klavesnice');
    cmOmonitory      :Vystup('Monitory');
    cmODisky         :Vystup('Disky');
    cmOKlavesnice    :Vystup('Klavesnice');
  else exit;
  end;
  ClearEvent(event);
end;

Procedure Tdealer.sestavNabidku;
begin
end;

Procedure Tdealer.zrusNabidku;
begin
end;

Procedure Tdealer.ukazNabidku;
begin
end;

Procedure Tdealer.vstup(ceho:string);
var r:TRect;
    okno:PVstupDialog;
    path:string;
begin
  r.assign(5,3,73,18);
  okno:=new(PVstupDialog,Init(r,True));
  path:=ceho;
  if length(path)>8 then
    system.delete(path,8,length(path)-8);
  path:=concat(path,'.dta');
  okno^.natahni(path,ceho);
  desktop^.execView(okno);
  Dispose(okno,done);
end;


Constructor TVstupDialog.Init(var Bounds:Trect;ATitle:TTitleStr);
Var R:Trect;
    i,j:byte;
    pom:PButton;
    Code,info:integer;
    path:string;
BEGIN
  TDialog.Init(Bounds, ATitle);
  GetExtent(r);
  R.Assign(R.A.X+2, R.A.Y+10, R.A.X+17,R.A.Y+12);
  pom:=New(PButton, init(R,'~M~ore',cmMore, bfNormal));
  Insert(pom);
  R.Move(16,0);
  pom:=New(PButton, init(R,'~O~k',cmOk, bfNormal));
  Insert(pom);
  R.Move(16,0);
  pom:=New(PButton, init(R,'~C~ancel',cmCancel, bfNormal));
  Insert(pom);

  GetExtent(R);
  Bounds.Copy(R);
  R.Assign(R.A.X+2, R.A.Y+8, R.B.X-10, R.A.Y+9);

  Bounds.assign(bounds.B.X-9, Bounds.A.Y+8, bounds.B.X-2, Bounds.A.Y+9);
  For i:=1 to 7 do
  BEGIN
    Pol[i,1]:=New(PInputLine,Init(R,50));
    Pol[i,2]:=New(PInputLine,Init(Bounds,8));
    R.Move(0,-1);
  END;
  For i:=7 Downto 1 do
  BEGIN
    Insert(pol[i,1]);
    Insert(pol[i,2]);
  END;
  POL[7,1]^.select;
  Insert(New(PLabel,Init(R,' ~N~ zev', Pol[7,1])));
```

```
  Insert(New(PLabel,Init(Bounds,' ~C~ena', Pol[7,2])));

  if length(aTitle) > 8 then
     system.delete(atitle,8,length(atitle)-8);
  path:=concat(atitle,'.dta');
  pracFile:=new(pBufStream,init(path, stOpen, 512));
  with PracFile^ do
    if status=stOk then
    begin
      Reset;
      init(path, stCreate, 512);
    end else seek(getsize);
end;

Destructor TVstupDialog.Done;
var i,j:byte;
Begin
  dispose(pracfile, done);
  for i:=1 to 6 do
    for j:=1 to 2 do
      Dispose(pol[i,j],Done);
  TDialog.done;
end;

Procedure TVstupDialog.HandleEvent(var Event:TEvent);
  Procedure zapis;
  var data:TVstupDialogRec;
  prac:PPart;
  num:real;
  code:integer;
  i:byte;
  begin
    data:=NVstupDialogRec;
    GetData(Data);
    setData(NVstupDialogRec);
    i:=1;
    with data do
      while (i<7) and (il[i].na) do
      begin
        val(il[i].ce,num,code);
        prac:=new(ppart, init(il[i].na,num));
        prac^.store(pracfile^);
        dispose(prac,done);
        inc(i);
    end;
  End;
  begin
  end;

  constructor Tpart.init(aname:string;aprice:real);
  begin
    tobject.init;
    name:=aname;
    price:=aprice;
  end;

  function Tpart.getprice:real;
  begin
    getPrice:=Price;
  end;

  function TPart.getName:string;
  begin
    getName:=name;
  end;

  Procedure TPart.setName(aname:string);
  begin
    name:=aname;
  end;

  Procedure Tpart.setPrice(aprice:real);
  begin
    price:=aPrice;
  end;

  function tpart.getstring:string;
  var a,b:string;
      l,hm:byte;
  begin
    str(price:8:2,A);
```

```
    b:=getname;
    hm:=length(b);
    if hm<50 then for l:=hm to 49 do b:=concat(b,' ')
    else delete(b, 51, hm-50);
    getstring:=concat(b,' ',a,'Kcs');
  end;

  constructor Tpart.load(var s:tstream);
  begin
    s.read(name,sizeof(name));
    s.read(price, sizeOf(price));
  end;

  Procedure tPart.store(var s:tstream);
  begin
    s.write(name, sizeOf(name));
    s.write(price, sizeOf(price));
  end;

  function Najdi_select:Boolean;
  var i,j:byte;
  Begin
    i:=1;j:=1;
    while (i<7) and not(pol[i,j]^.GetState(sfSelected))
    do if j=2 then
    begin
      j:=1;
      inc(i);
    end else inc(j);
    Najdi_select:=(i<7);
  end;
Begin
  if (event.what=evBroadCast) and (event.Command=cmDefault) then
  begin
    if najdi_select then selectNext(false)
    else TDialog.handleEvent(Event);
  end else
  begin
    if (event.what=evCommand) then
    begin
      if (event.command=cmMore) or (event.command=cmOk) then
      begin
        zapis;
        pol[7,1]^.select;
      end;
    end;
    TDialog.HandleEvent(event);
    end;
    ClearEvent(Event);
  end;


Procedure Tdealer.vystup(ceho:string);
var R:Trect;
    okno:pvystupDialog;
    path:string;
begin
  R.Assign(5,2,73,18);
  Okno:=new(pvystupDialog, init(R));
  path:=Ceho;
  if length(path)>8 then
    system.delete(path,8,length(path)-8);
  Path:=concat(path, '.dta');
  okno^.natahni(path,ceho);
  desktop^.execview(okno);
  dispose(okno,done);
End;

constructor TmyListBox.init(var bound:Trect;aNumCols:word;Ascrollbar:pascrollbar;
                            APath:strin);
var pom:ppart;
    dl:searchRec;
    soub:PDosstream;
Begin
  TlistBox.init(bounds, aNumcols, ascrollBar);
  findfirst(aPath, archive, Dl);
  id Doserror=0 then
    doub:=new(PdosStream, init(APath, stOpen))
  else soub:=new(PDosStream, init(Apath, stCreate));
  Kol:=new(Pcollection, init(10,1));
  while soub^.getpos < soub^.getsize do
```

```
  begin
    pom:=new(ppart, load(soub^));
    kol^.insert(pom);
  end;
  Dispose(soub,done);
  newList(kol);
end;

Destructor TmyListBox.done;
begin
  newList(nil);
  Tlistbox.done;
end;

function tMyListBox.getTExt(item:integer;maxLen:integer):string;
begin
  gettext:=ppart(list^.At(item))^.getstring);
end;

Constructor TVystupdialog.Init(var Bounds:tRect);
var R:TRect
    krok:=word;
begin
  TDialog.init(bounds, '');
  obs:=false;
  getExtend(R);
  R.b.y:=r.a.y+1;
  r.a.x:=r.a.x.+10;
  r.b.x:=r.a.x;
  titul:=new(PtitulText,init(R,'');
  getExtend(r);
  R.assign(r.b.x-1,r.a.y+2, r.b.x,r.b.y-5);
  scrollbar:=new(pscrollBar,Init(r));
  insert(scrollBar);
  GetExtend(R);
  R.grow(-11,-1);
  r.a.y:=r.b.y-2;
  insert(new(pbutton,init(R,'~O~k',cmOk,Bfdefault)));
end;

destructor TVystupDialog.done;
begin
  if Obs then list^.newlist(nil);
  Tdialog.done;
end;

Procedure TVystupDialog.natahni(path,name:string);
var a:string;
    r:Trect;
begin
  if obs then
  begin
    delete(list);
    list^.newlist(Nil);
    dispose(list,done);
  end;
  obs:=true;
  getextent(R);
  R.assign(r.a.x+1,r.a.y+1,r.b.x-1,r.b.y-4);
  list:=new(Pmylistbox, init(R,1,scrollBar, Path));
  insert(list);
  delete(titul);
  dispose(titul, done);
  a:=concat('Evidovane',Name,'');
  getExtent(R);
  r.b.y:=r.a.y+1;
  r.a.x:=(r.b.x-r.a.x)div 2 -length(a) div 2;
  r.b.x:=r.a.x+length(a);
  titul:=new(pTitulText,Init(r,a));
  insert(titul);
  drawview;
end;

function TtitulText.getpalette:ppalette;
const cTitulText=#2;
      ptitultext:string[length(cTitulText)]=CtitulText;
begin
  getpalette:=@PtitulText;
end;
```

```
VAR a:tdealer;
BEGIN
  a.init;
  a.run;
  a.done;
END.
```

DOKONČENÍ UKÁZKOVÉHO PROGRAMU