JAZYK OBJECT PASCAL

Karel Müller

1. Úvod

Objektově orientované programování (zkráceně jen objektové programování) je technika, při které se složité programové systémy strukturují jako množiny vzájemně komunikujících objektů. Každý objekt definuje svou vlastní datovou strukturu a algoritmy. Lze tím dosáhnout vysoké úrovně modularity a datových abstrakcí.

I když speciální jazykové rysy nejsou pro objektově orientované programování nezbytné, bez nich jsou objektově orientované programy málo průhledné a obtížně se modifikují. Mezi programovací jazyky, které podporují koncepci objektů, patří např. Simula-67, Smalltalk-80 a C++. Prvním jazykem pascalského typu, podporujícím objektové programování, byl Object Pascal, z něhož vychází též MPW Pascal 3.0. V obou případech se jedná o implementace na počítačích firmy Apple. Na počítačích IBM-PC je objektové programování podporováno systémem Turbo Pascal 5.5. V tomto článku jsou stručně popsány objektové rysy jazyka Object Pascal.

2. Koncepce objektů v Object Pascalu

Ve standardním Pascalu jsou zavedeny čtyři strukturované typy : pole, záznam, množina a soubor. Pátým strukturovaným typem v jazyku Object Pascal je typ objekt, který má tyto vlastnosti :

a) Objekty mají podobnou strukturu jako záznamy, tj. obsahují pevný počet pojmenovaných položek různých typů. Typem objektu jsou dále definovány tzv. metody, což jsou procedu-

ry a funkce, které realizují operace s objekty daného typu.

b) Libovolný typ objektu může dědit všechny vlastnosti jiného typu objektu. Dědí se jak položky, tak i metody, zděděné metody však mohou mít jinou implementaci. Jestliže typ T1 dědí vlastnosti typu T2, nazýváme jej následníkem typu T2 a typ T2 nazýváme předchůdcem typu T1.

c) Všechny objekty jsou dynamické, tzn. vznikají i zanikají provedením příkazů při běhu programu a identifikují se pomocí ukazatelů.

Vlastnosti typu objekt ukážeme podrobněji na příkladech.

3. Deklarace a použití typu objekt

Deklarace typu objekt může mít např. tento tvar :

```
type
   Retez = packed array (1..10) of char;
   Osoba =
      object
         Jmeno : Retez;
         Prijmeni : Retez;
         Rok narozeni : integer;
         procedure Init (jm,pr:Retez; r:integer);
         procedure Vystup;
      end;

procedure Osoba.Init;
begin
   Jmeno:=jm;
   Prijmeni:=pr;
   Rok narozeni:=r;
end;
```

```
procedure Osoba.Vystup;
begin
   writeln ('jmeno: ',Jmeno);
   writeln ('prijmeni: ',Prijmeni);
   writeln ('rok narozeni: ',Rok narozeni);
end;
```

Objekty typu Osoba obsahují položky Jmeno, Prijmeni a Rok narozeni a jsou pro ně definovány metody Init a Vystup. Definice metod jsou rozdělany do dvou částí : V popisu typu je uvedena hlavička metody (ve formě hlavičky procedury nebo funkce), algoritmus metody je definován mimo popis typu, a to stejným způsobem, jakým se definují procedury a funkce, předběžně deklarované pomocí direktivy forward. Identifikátory položek objektu jsou v definici metody přímo viditelné.

Jak bylo řečeno, proměnné typu objekt mohou být pouze dynamické. Vytvářejí se procedurou new a identifikují pomocí ukazatelů. Proměnná, jejíž hodnotou může být ukazatel na objekt typu T, se nazývá reference typu T. Reference R typu T se deklaruje v úseku deklarací proměnných zápisem R:T. Například, mají-li proměnné On a Ona být referencemi typu Osoba (tj. mají-li identifikovat objekty typu Osoba), zavedeme deklaraci

```
var On, Ona : Osoba;
```

Po deklaraci mají proměnné On a Ona nedefinované hodnoty. Ukazatele na objekty jim přiřadí procedura new :

```
new (On);
new (Ona);
```

Volání metody M na objekt, jehož reference je R, má tvar

R.M (seznam skutečných parametrů)

Příklad :

```
On.Init ('Jan       ', 'Novak      ', 1952);
```

Podobným způsobem se označuje přístup k položkám objektu. Např. položce Rok narozeni v objektu, na který ukazuje proměnná Ona,

můžeme přiřadit hodnotu příkazem

```
Ona.Rok narozeni:=1960;
```

Mezi proměnnými, které jsou referencemi téhož typu objekt, je dovoleno přiřazení. Příklad :

```
Ona:=On;
```

Po provedení tohoto příkazu budou obě proměnné referencovat tentýž objekt; obsah objektu, který byl předtím referencován proměnnou Ona, se příkazem nezmění.

Následující příklad deklarace typu ilustruje mechanismus dědění :

```
type
   Pracovník=
      object (Osoba)
         Odpracovano : integer; (počet odpracovaných hodin)
         procedure Init (jm,pr:Retez; r:integer); override;
         procedure Odpracoval (h:integer);
         function Vyplata (tarif:integer):integer;
      end;

procedure Pracovnik.Init;
begin
   inherited Init (jm,pr,r);
   Odpracovano:=0;
end;

procedure Pracovnik.Odpracoval (h:integer);
begin
   Odpracovano:=Odpracovano+h;
end;

function Pracovnik.Vyplata;
begin
   Vyplata:=Odpracovano*tarif;
   Odpracovano:=0;
end;
```

Typ Pracovnik je následníkem typu Osoba a dědí všechny položky a metody, definované tímto typem. Objekty typu Pracovnik tedy obsahují zděděné položky Jmeno, Prijmeni a Rok narozeni a dále novou položku Odpracovano. Metoda Vypis je zděděná beze změny, metoda Init má pro objekty typu Pracovnik jinou implementaci, což vyjadřuje slovo "override" za hlavičkou metody. První příkaz z příkazové části metody Pracovnik.Init je voláním původní implementace metody Init pro typ Osoba. Novými metodami pro typ Pracovnik jsou procedura Odpracoval a funkce Vyplata.

Příklady použití typu Pracovnik :

```
var Prac, DalsiPrac : Pracovnik;

new (Prac);
Prac.Init (...);
Prac.Vypis;
Prac.Odpracoval (hod);
writeln (Prac.Vyplata);
```

4. Kompatibilita typů objekt

Pravidla kompatibility vzhledem k přiřazení připouštějí, aby proměnné, která je referencí objektů typu T, byl přiřazen nejen ukazatel na objekt typu T, ale též ukazatel na objekt, jehož typ je následníkem typu T. Např. příkaz

```
On:=Prac;
```

je přípustný, zatímco příkaz

```
Prac:=On;
```

není dovolen.

I když proměnná, která je referencí typu T, obsahuje ukazatel na objekt, jehož typ je následníkem typu T, jsou prostřednictvím této proměnné přímo přístupné pouze položky a metody, definované typem T.

Příklad : Po provedení příkazu

```
On:=Prac;
```

proměnná On sice referencuje objekt typu Pracovnik, který obsahuje položku Odpracovano, zápis

```
On.Odpracovano
```

však není dovolen.

Souhrnně můžeme říci, že viditelnost identifikátorů položek a metod je statickou vlastností referencí, kterou neovlivňuje běh programu. Na druhé straně vazba identifikátoru metody na její implementaci má dynamický charakter. Znamená to, že když proměnná R, která je referencí typu T, referencuje objekt typu T1, který je následníkem typu T a má předefinovanou metodu M, pak zápisem R.M se volá implementace metody, kterou definuje typ T1.

Příklad : Po provedení příkazu

```
On:Prac;
```

se příkazem

```
On.Init (...);
```

vyvolá metoda Pracovnik.Init a nikoliv Osoba.Init.

Na příkladu proměnných On a Prac vysvětlíme ještě tzv. koerci typu a funkci Member. Jestliže proměnná On obsahuje ukazatel na objekt typu Pracovnik, lze její hodnotu přiřadit proměnné Prac příkazem

```
Prac:Pracovnik (On)
```

ve kterém zápis Pracovnik (On) se nazývá koerci typu Osoba na typ Pracovnik. Použitelnost koerce typu lze ověřit funkcí Member, jejíž zápis ve tvaru

```
Member (R,T)
```

má hodnotu true tehdy, když reference R obsahuje ukazatel na objekt typu T. Příklad :

```
if Member (On,Pracovnik) then Prac:=Pracovnik (On);
```

Koerci typu a funkci Member lze využít též při označení metod a položek, jejichž identifikátory nejsou jinak viditelné.

Příklad :

```
if Member (On,Pracovnik) then
   writeln (Pracovnik (On).Vyplata;
```

5. Hierarchie typů objekt

Mechanismus dědičnosti dovoluje vytvářet hierarchie typů objekt. Na vrcholu takové hierarchie je typ, který sdružuje všechny společné vlastnosti typů dané hierarchie. Jeho následníci mají další, vzájemně různé vlastnosti, jejich následníci představují další speciální případy, atd. Postupné rozšiřování takové hierarchie je přitom poměrně snadné a obvykle nevyžaduje změny v existujících definicích. Ukážeme si to na příkladu.

Předpokládejme, že máme navrhnout typy, reprezentující grafické elementy, jako je bod, kružnice, úsečka pod. Každý typ bude obsahovat souřadnice referenčního bodu (pro kružnici to bude střed, pro úsečku jeden krajní bod, atd.), další informace budou závislé na druhu elementu (např. typ, reprezentující kružnici, bude obsahovat velikost poloměru). Pro všechny elementy mají být definovány operace Vykresli a Presun.

Základem hierarchie požadovaných typů bude typ Graficky_element, který definujeme takto :

```
type
   Graficky_element =
      object
         X, Y : intefer;
         procedure Vykresli;
         procedure Nove XY (noveX, noveY : integer);
         procedure Presun (noveX, noveY : intefer);
      end;
```

Implementace metody Vykresli bude různá pro různé následníky typu Grafický_element, přesto jí však musíme definovat i pro tento typ :

```
procedure Graficky_element.Vykresli;
begin end;
```

Metoda NoveXY přepočte veškeré souřadnice elementu tak, aby odpovídaly změně X na nove X a změně Y na noveY. Pro typ Gra-

ficky_element uvedeme její nejjednodušší implementaci :

```
procedure Graficky_element.NoveXY;
begin
   X:=noveX;
   Y:=noveY;
end;
```

Metoda Presun smaže element z obrazovky tím, že jej vykreslí v barvě pozadí, pak přepočte souřadnice a nakonec jej znovu vykreslí v aktuální barvě "inkoustu" :

```
procedure Graficky_elemnt.Presun;
var b:integer;
begin
   b:=GetColor;
   SetColor (GetBkColor);
   Vykresli;
   NoveXY (noveX, noveX );
   SetColor (b);
   Vykresli;
end;
```

V této definici předpokládáme, že barvy jsou kódovány celými čísly, hodnotou funkce GetColor je aktuální barva, funkce GetBkColor je aktuální barva pozadí a procedura SetColor nastavuje aktuální barvu.

Typy, reprezentující jednotlivé grafické elementy, deklarujeme jako následníky typu Graficky_element:

```
type
   Bod =
     object (Graficky_element)
        procedure Vykresli; override;
     end;

   Kruznice =
     object (Gravicky_element)
        R : integer;
        procedure Vykresli; override;
     end;
```

```
Usecka =
  object (Graficky_element)
    X2, Y2 : integer;
    procedure Vykresli; override;
    procedure NoveXY (noveX, noveY : integer); override;
  end;

procedure Bod.Vykresli;
begin
   PutPixel(X,Y)
end;

procedure Kruznice.Vykresli;
begin
   DrawCircle (X,Y,R)
end;

procedure Usecka.Vykresli;
begin
   DrawLine (X,Y,X2,Y2)
end;

procedure Usecka.NoveXY;
begin
   X2:=X2+noveX-X; Y2:=Y2+NoveY-Y
   X:=noveX; Y:=NoveY
end;
```

Na závěr si ještě všimněme, jaké výhody má použití objektů oproti klasickému přístupu při řešení uvedené úlohy. Bez použití objektů bychom typ, reprezentující grafické elementy, definovali jako záznam s variantní částí :

```
type
   DruhElementu = (Bod, Kruznice, Usecka);
   Graficky_element =
      record
         X, Y : integer;
         case Druh: Druh_elementu of
```

```
            Bod : ();
            Kruznice : (R:integer);
            Usecka : (X2,Y2:integer);
    end;
    Ukazatel_na_element =   Graficky_element;
```

Zatímco použití objektů umožňuje postupné vytváření hierarchie typů objekt a přidání typu, reprezentujícího nový element, nevyžaduje změnu v již napsaných definicích, variantní záznam obsahuje informace o všech případech, a proto každé rozšíření znamená jeho změnu. To platí i o procedurách, realizujících jednotlivé operace. Jako příklad uveďme proceduru Vykresli, která při reprezentaci grafických elementů bude mít tuto deklaraci :

```
    procedure Vykresli (p:Ukazatel_na_element);
    begin
       with p   do
         case Druh of
          Bod:      PuPixel (X,Y);
          Kruznice: DrawCircle (X,Y,R);
          Usecka:   DrawLine (X,Y,X1,Y2);
         end
    end;
```

Přibude-li nový grafický element, musíme i tuto proceduru upravit, zatím co při použití objektů stačí definovat novou implementaci metody Vykresli.

Literatura :

/1/ L. Tesler : Object Pascal Report. In:Structured Language World, Vol.9, No.3, str. 10-14. 1985

/2/ K. Müller, K. Richta, J.Polák : Modulární a objektové programování. In:SofSEM 88, str. 221-254, 1988

Karel Müller
ČVUT FEL
Praha 2