Милан Чабаркапа

УВОД У ОБЈЕКТНО-ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

У основи објектно-оријентисаног програмирања (даље ООП) лежи идеја обједињавања у једној структури података и акција над њима (које се у терминологији ООП називају методе). То значи да при оваквом приступу организацији података, за разлику од традиционалног, теснија је веза између података и акција. ООП омогућава ефикаснији рад програмера тиме што се креирају компактнији и лако прошириви програми. Иако ће неки појмови ООП-а можда на први поглед изгледати нејасни, напор да се овлада новим приступом програмирања вишеструко ће се исплатити.

ООП се базира на три основна појма: инкапсулација, наслеђивање и полиморфизам.

Инкапсулација је комбиновање података са процедурама и функцијама које оперишу са тим подацима. Као резултат се добија нови тип који се назива *објект*.

Наслеђивање је могућност коришћења већ дефинисаних објеката за грађење хијерархије објеката, извођењем из њих. Сваки „наследник“ наслеђује описе података „прародитеља“ и приступ методама њихове обраде.

Полиморфизам је могућност дефинисања јединственог по имену метода (процедуре или функције), применљивог истовремено ка свим објектима хијерархије наслеђивања, при чему сваки објект хијерархије може назначити специфичности реализације те акције над „њим самим“.

Објекти. Основни појмови

Формално посматрајући објекти су врло слични слоговима у PASCAL-у. *Објект* је структура чије су компоненте узајамно повезани подаци разних типова, као и процедуре и функције које користе те податке. Компоненте-*подаци* се називају поља објекта, а компоненте-*процедуре* и *функције* се називају методе. За означавање типа „објект“ у TURBO PASCAL-у се користи резервисана реч `object`.

Објектни тип се описује слично слогу у PASCAL-у:

```
type
  ImeObjekta=object
      PoljaPodataka;
      ZaglavljaMetoda;
    end;
```

Конкретна променљива, декларисана типом `ImeObjekta`, назива се *примерак* тог типа.

Непосредно у објектном типу се задају само заглавља метода (процедура или функција), а њихови потпуни описи се задају одвојено у одељку описа функција и процедура, само се тада име метода формира од имена објектног типа коме припада метод, симбола „тачка“ и имена процедуре или функције.

Заглавља метода при њиховој реализацији не морају садржати списак формалних параметара, подразумева се да њихови описи у објекту као да имају директиву `forward`.

На пример, објектни тип којим се описује тачка и основне операције над тачком може се описати на следећи начин:

```
type
  Point=object
      X,Y,Color:integer;
      procedure Create(PosX,PosY,C:integer);
      procedure SwitchOn;
      procedure SwitchOff;
      procedure Move(dx,dy:integer);
      function GetX:integer;
      function GetY:integer;
    end;
```

Поља `X,Y,Color` чувају информацију о положају тачке и како је тачка обојена. Међутим, објект не садржи само информацију (кординате тачке и њену боју) већ и методе којим се задају операције над тачком: креирање тачке (задавањем координата и боје) — `Create`, њено „паљење“ и „гашење“ — `SwitchOn` и `SwitchOff`, померање по екрану — `Move`, и детекцију тренутног положаја тачке — `GetX` и `GetY`:

```
procedure Point.Create(PosX,PosY,C:integer);
  begin
    X:=PosX; Y:=PosY;
    Color:=C
  end;
procedure Point.SwitchOn;
  begin
    PutPixel(X,Y,Color)
  end;
procedure Point.SwitchOff;
  begin
    PutPixel(X,Y,GetBkColor)
  end;
procedure Point.Move(dx,dy:integer);
  begin
    SwitchOff;
    X:=X+dx;Y:=Y+dy;
    SwitchOn
  end;
function Point.GetX:integer;
  begin
    GetX:=X
  end;
function Point.GetY:integer;
  begin
    GetY:=Y
  end;
```

Објектни приступ подразумева да у програму декларисани објект садржи исцрпне податке о одговарајућем „физичком“ објекту, да се методе колико је то могуће обрађају једна другој и пољима података свог објекта.

Без обзира што су описи метода одвојени од дефиниције објектног типа, они имају приступ свим његовим пољима података.

Примерак објекта се декларише на следећи начин:

```
var
  TestPt:Point;
```

и с њим се оперише посредством његових метода:

```
    TestPt.Create(100,200,GetMaxColor);    {* zadaje tacku *}
    TestPt.SwitchOn;    {* pali tacku *}
    TestPt.Move(20,-10);    {* gasi tacku i pali na novoj poziciji *}
    TestPt.SwitchOff;    {* gasi tacku *}
```

У заглављу наредбе `with` може се ставити и примерак објектног типа, тако да се методама може приступати а да се не понавља име примерка објекта:

```
with TestPt do
  begin
    Create(100,200,GetMaxColor);    {* zadaje tacku *}
    SwitchOn;    {* pali tacku *}
    Move(20,-10);    {* gasi tacku i pali na novoj poziciji *}
    SwitchOff;    {* gasi tacku *}
  end;
```

Пољима објектног типа може се приступати не само коришћењем метода објекта, већ и непосредно као пољима обичног слога:

```
TestPt.X:=100;
TestPt.Y:=200;
TestPt.Color:=GetMaxColor;
```

Међутим, овим се одступа од објектно-оријентисаног стила по коме се све операције над информацијом задатом у објектном типу остварују посредством његових метода. Нарушавањем овог правила могу се изгубити преимућства објектног приступа програмирању.

При опису објеката треба водити рачуна о следећим захтевима:

- опис типа „објект“ може се креирати само у одељку за опис типова `type` основног програма или у одељцима модула;
- не смеју се описивати локални објекти у процедурама и функцијама;
- при опису објекта поља података се морају налазити испред описа метода;
- компоненте објекта не могу бити фајлови, а фајлови не могу садржати компоненте типа „објект“.

Следећи програм тестира објект `Point` тако што зависно од тога која је стрелица на тастатури притиснута помера тачку у одговарајућем смеру. Програм се прекида притиском на тастер ESC.

```
program DemoObj;
  uses crt,graph;
  type
    Point=object
        X,Y,Color:integer;
        procedure Create(PosX,PosY,C:integer);
        procedure SwitchOn;
```

```
      procedure SwitchOff;
      procedure Move(dx,dy:integer);
      function GetX:integer;
      function GetY:integer;
    end;
procedure Point.Create(PosX,PosY,C:integer);
  begin
    X:=PosX; Y:=PosY;
    Color:=C
  end;
procedure Point.SwitchOn;
  begin
    PutPixel(X,Y,Color)
  end;
procedure Point.SwitchOff;
  begin
    PutPixel(X,Y,GetBkColor)
  end;
procedure Point.Move(dx,dy:integer);
  begin
    SwitchOff;
    X:=X+dx;Y:=Y+dy;
    SwitchOn
  end;
function Point.GetX:integer;
  begin
    GetX:=X
  end;
function Point.GetY:integer;
  begin
    GetY:=Y
  end;
var
  TestPt:Point;
  GDriver,GMode,MaxC,i:integer;
  ch:char;
begin
  GDriver:=Detect;
  InitGraph(GDriver,GMode,'c:\tp60\bgi');
  if GraphResult<>0
    then
      begin
        writeln('Greska pri inicijalizaciji grafike');
        halt(1)
      end;
  ClearDevice;
  MaxC:=GetMaxColor;
  TestPt.Create(GetMaxX div 2,GetMaxY div 2,MaxC);
  TestPt.SwitchOn;
  repeat
    with TestPt do
      begin
        ch:=ReadKey;
        if ch=#0
          then
            begin
              ch:=ReadKey;
              case ch of
                #75:Move(-1,0); {* тачка лево *}
```

```
                #77:Move(+1,0); {* тачка десно *}
                #72:Move(0,-1); {* тачка горе *}
                #80:Move(0,+1) {* тачка доле *}
              end
            end
        end
      until ch=#27;
      CloseGraph;
    end.
```

Ако се програм тестира без `SwitchOff` у методи `Point.Move` оставља се траг на путањи тачке.

Наслеђивање и преклапање

Важно својство објектних типова је да омогућавају да се при креирању новог објектног типа искористи претходно дефинисан објектни тип. Нека је, на пример, потребно креирати објектни тип који управља простом геометријском фигуром – кругом – на екрану монитора. Очигледно је да је структура информација за дефинисање круга врло слична опису структуре тачке: X и Y – дефинишу положај центра круга и `Color` – боју круга. Потребно је још додати поље за чување полупречника круга.

Објектно-оријентисани стил омогућава дефинисање новог објекта као *потомка* другог претходно дефинисаног типа. Тиме нови тип аутоматски располаже са свим пољима и методама претходно уведеног типа, који се назива *претком* или *родитељским типом*. У том случају се у дефиницији типа-потомка мора навести (у заградама после службене речи `object`) име родитељског типа:

```
type
  ImeObjektaNaslednika=object (ImeObjektaRoditelja)
      NovaPoljaObjektaNaslednika;
      NoveMetodeObjektaNaslednika;
    end;
```

На пример, може се описати објект на следећи начин:

```
type
  Circle=object (Point)
      Radius:integer
    end;
```

Задавање родитељског типа означава да су у објектном типу `Circle` имплицитно присутна сва поља и методе из типа `Point`.

Ако се декларише:

```
var
  Circle1:Circle;
```

може се приступати пољима и методама на следећи начин:

```
Circle1.Create(150,300);
Circle1.Radius:=50;
```

Описано својство објектних типова назива се *наслеђивање* и широко се користи у објектно-оријентисаном програмирању. Треба имати на уму да један тип може бити предак произвољном броју типова-потомака, док сваки објектни тип може наследити поља и методе само једног типа-родитеља, који се задаје у овалним заградама.

Тип потомак може, са своје стране, бити предак другом објектном типу (типовима). Тако се може дефинисати фигура „прстен“ која се састоји из два концентрична круга:

```
type
  Ring=object (Circle)
      Radius2:integer
    end;
```

Тип `Ring` наслеђује поље `Radius` од непосредног претка `Circle`, а поља и методе из типа `Point` који му је индиректни предак. Дужина оваквог ланца наслеђивања није ограничена.

Методи којим располаже `Point` нису довољни за операције са кругом. Док се методи `GetX` и `GetY` могу искористити у типу потомку да би вратиле координате центра, методе `SwitchOn` и `SwitchOff` не обезбеђују цртање круга. Зато, заједно са новим пољем `Radius` потпун опис типа `Circle` мора садржати сопствене методе за цртање, уклањање и померање круга по екрану. Најпростије решење је да се у нови тип уведу такве методе са новим именима. Међутим, објектно-оријентисани приступ дозвољава дефинисање нових метода *са старим* именима, *преклапајући* на тај начин родитељске типове:

```
type
  Circle=object (Point)
      Radius:integer;
      procedure Create(PosX,PosY,C,R:integer);
      procedure SwitchOn;
      procedure SwitchOff;
      procedure Move(dx,dy:integer);
      function GetR:integer;
    end;
procedure Circle.Create(PosX,PosY,C,R:integer);
  begin
    Point.Create(PosX,PosY,C);
    Radius:=R;
  end;
procedure Circle.SwitchOn;
  begin
    Graph.SetColor(Color);    {* koristi proceduru modula Graph *}
    Graph.Circle(X,Y,Radius);  {* ukazuje da se radi o proceduri modula Graph *}
  end;
procedure Circle.SwitchOff;
  begin
    Graph.SetColor(GetBkColor);
    Graph.Circle(X,Y,Radius)
  end;
procedure Circle.Move(dx,dy:integer);
  begin
    SwitchOff;
    X:=X+dx;Y:=Y+dy;
    SwitchOn
  end;
function Circle.GetR:integer;
  begin
    GetR:=Radius
  end;
```

Овај опис објектног типа `Circle` садржи следеће елементе:

- поља X, Y, `Color` наслеђена од родитељског типа `Point`;
- сопствено поље `Radius`;
- метод `Circle.Create` који иницијализује поља објекта `Circle`. Овај метод преклапа унутар типа `Circle` од `Point` наслеђени метод `Point.Create`. Може се приметити да се за иницијализацију поља X, Y, `Color` позива метод `Point.Create` који је доступан (наслеђивањем) типу `Circle`;
- методе `Circle.SwitchOn`, `Circle.SwitchOff` и `Circle.Move` цртају, уклањају и померају круг. Ове методе потпуно преклапају истоимене методе из `Point`, што је разумљиво јер цртање круга се разликује од цртања тачке. Пошто стандардна процедура за цртање круга има такође назив `Circle`, ради једнозначне идентификације користи се сложено име `Graph.Circle` (према модулу `Graph` у коме се налази процедура);
- нови метод `GetR` ради приступа текућој вредности полупречника круга;
- два наслеђена метода `GetX` и `GetY` за добијање текућих координата центра круга.

Приметимо: могу се преклапати само методе; поља родитељског типа безусловно наслеђује тип-потомак и не могу се преклапати (то јест имена поља потомка не смеју бити иста као поља типа-претка). Осим тога, нови метод у типу-потомку може имати параметре различите од истоименог метода из типа-претка.

Анализирајмо понашање објекта типа `Circle` позивањем његових метода у следећем програму који је скоро идентичан програму у коме смо пратили понашање тачке:

```
program DemoObjCircle;
  uses crt,graph;
    type
    Point=object
        X,Y,Color:integer;
        procedure Create(PosX,PosY,C:integer);
        procedure SwitchOn;
        procedure SwitchOff;
        procedure Move(dx,dy:integer);
        function GetX:integer;
        function GetY:integer;
      end;
        {* opis metoda iz Point koje su vec navedene u tekstu *}
...
  type
    Circle=object (Point)
        Radius:integer;
        procedure Create(PosX,PosY,C,R:integer);
        procedure SwitchOn;
        procedure SwitchOff;
        procedure Move(dx,dy:integer);
        function GetR:integer;
      end;
        {* opis metoda iz Circle koje su vec navedene u tekstu *}
...
  var
    TestCirc:Circle;
    GDriver,GMode,MaxC,i:integer;
    ch:char;
```

```
begin
  GDriver:=Detect;
  InitGraph(GDriver,GMode,'c:\tp60\bgi');
  if GraphResult<>0
    then
      begin
        writeln('Greska pri inicijalizaciji grafike');
        halt(1)
      end;
  ClearDevice;
  MaxC:=GetMaxColor;
  TestCirc.Create(GetMaxX div 2,GetMaxY div 2,MaxC,50);
  TestCirc.SwitchOn;
  repeat
    with TestCirc do
      begin
        ch:=ReadKey;
        if ch=#0
          then
            begin
              ch:=ReadKey;
              case ch of
                #75:Move(-10,0);     {* krug levo *}
                #77:Move(+10,0);     {* krug desno *}
                #72:Move(0,-10);     {* krug gore *}
                #80:Move(0,+10)     {* krug dole *}
            end
          end
      end
  until ch=#27;
  CloseGraph;
end.
```

Програмом се црта и брише круг уз померање зависно од тога која је стрелица притиснута на тастатури рачунара (лево, десно, горе, доле).