Przypomnienie – dziedziczenie obiektów

Przykład (p6)

Zapis programu z wykorzystaniem modułu (Podstawy Delphi 2.1, 2.2, 2.3 – str11)

Przykład (p7) – przypomnienie zapis z wykorzystaniem rekordu

W pliku projektu (głównym) - **p1.dpr**:

```
uses modulT;

var t: Ttrojkat;

begin
writeln('Podaj dl bokow trojkata');
readln(t.a, t.b, t.c);
writeln('Pole=', pole(t));
readln;
//writeln ('Dodatkowo obliczenie obwodu: ', obwod(t));
//readln;
end.
```

W pliku modułu - **modulT.pas**:

```
unit modulT;

interface

type Ttrojkat= record
               a, b, c: real;
               end;

function pole(t: Ttrojkat): real;

implementation

function obwod(t: Ttrojkat): real;
begin
obwod:=t.a+t.b+t.c;
end;

function pole(t:Ttrojkat):real;
var p:real;
begin
p:=obwod(t)/2;
pole:=sqrt(p*(p-t.a)*(p-t.b)*(p-t.c));
end;

end.
```

Część interfejsowa zawiera elementy widoczne w pliku głównym.
Włączenie w tym pliku instrukcji:

```
writeln ('Dodatkowo obliczenie obwodu: ', obwod(t));
```

nie powiedzie się – procedura **obwod** dostępna tylko lokalnie w module.

Zmiana tego ograniczenia – w drodze dopisania zapowiedzi (nagłówka) procedury w sekcji interface. Sekcja w nowej postaci:

```
interface

type Ttrojkat= record
               a, b, c: real;
               end;
function pole(t: Ttrojkat): real;
function obwod(t: Ttrojkat): real;
```

Przykład (p8) **- zapis obiektowy**

Plik główny prawie identyczny:

```
uses modulT;

var t: Ttrojkat;

begin
writeln('Podaj dl bokow trojkata');
readln(t.a, t.b, t.c);
writeln('Pole=', t.pole);
readln;
end
```

Z uwagi na wykorzystanie obiektu wywołanie funkcji obliczającej pole jest postaci:

```
t.pole;
```

Plik modułu:

```
unit modulT;

interface

type Ttrojkat= object
               a, b, c: real;
               function obwod:real;
               function pole:real;
               end;

implementation

function Ttrojkat.obwod:real;
begin
obwod:=a+b+c;
end;
function Ttrojkat.pole:real;
var p:real;
begin
p:=obwod/2;
pole:=sqrt(p*(p-a)*(p-b)*(p-c));
end;
end.
```

Przy deklarowaniu obiektów w module obowiązuje zasada: w części interfejsowej tylko opis typu obiektowego – rozpisanie metod w części **implementation**.

W powyższym przykładzie cały obiekt jest dostępny na zewnątrz (np. w pliku głównym).
Np. można dodać wywołanie metody obwod:

```
writeln(Obwod=', t.obwod);
```

Wynika to z ogólnej zasady że wszystko to co zadeklarowano w części **interface** jest widoczne.

Jeżeli zachodzi potrzeba ukrycia pewnych szczegółów implementacji obiektu należy dodatkowo w zapisie deklaracji obiektu dodać dyrektywę **private**.

Opis dyrektyw **private** i **public** – (patrz Podstawy Delphi rozdział 3.3 – str 20)

Korekta przykładu – wyłączenie widoczności metody **obwod** poprzez dodanie dyrektyw.

Przykład (p9) zapis w trzech plikach.

W pliku projektu (głównym) - **p1.dpr**:

```
uses modulT2;

var t: TT2;

begin
writeln('Podaj dl bokow trojkata');
readln(t.a, t.b, t.c);
writeln('Pole=', t.pole);
readln;
end.
```

Plik modułu – modulT2:

```
unit modulT2;

interface

uses modulT;

type TT2= object(Ttrojkat)
          function ha:real;
          function hb:real;
          function hc:real;
          end;

implementation

function TT2.ha:real;
begin
ha:=2*pole/a;
end;
function TT2.hb:real;
begin
hb:=2*pole/b;
```

```
end;
function TT2.hc:real;
begin
hc:=2*pole/c;
end;

end.
```

Plik modułu – modulT:

```
unit modulT;

interface

type Ttrojkat= object
                  a, b, c: real;
               private
                  function obwod:real;
               public
                  function pole:real;
               end;

implementation

function Ttrojkat.obwod:real;
begin
obwod:=a+b+c;
end;
function Ttrojkat.pole:real;
var p:real;
begin
p:=obwod/2;
pole:=sqrt(p*(p-a)*(p-b)*(p-c));
end;

end.
```

Obiekty dynamiczne – class (patrz Podstawy Delphi – rozdział 4)

W Pascalu istnieją dwa typy klas:
- statyczne – w deklaracji słowo **object**
- dynamiczne – w deklaracji słowo **class**

Są one podobne w zakresie deklarowania typów i zmiennych zasad dostępu do składowych, organizacji dziedziczenia itp.

Najważniejsze różnice to:

- Dynamiczny charakter zmiennych typu klasowego.
- Konieczność inicjowania klas za pomocą konstruktora.
- Istnienie wspólnego typu macierzystego dla wszystkich klas: TObject.

Typ TObject (patrz Podstawy Delphi – 4.2)

Istnieje typ TObject, który jest przodkiem wszystkich klas. Dziedziczenie od tego typu jest domyślne i nie musi być wykazywane w deklaracji.

Predefiniowany typ TObject ma szereg składowych (same metody), które oczywiście są przekazywane do potomków i dlatego każda klasa ma od razu pewne wyposażenie.
Najważniejsze ze składowych – specjalizowana metoda konstruktor **Create**, druga specjalizowana metoda destruktor **Destroy** oraz metoda **Free**.

Dla klas można definiować własne konstruktory, jednak często wykorzystywany jest konstruktor standardowy dziedziczony z typu TObject.

Odmiennie niż dla obiektów statycznych deklarowanie zmiennej obiektowej nie powoduje rezerwacji obszaru pamięci dla tej zmiennej – rezerwowany jest tylko mały obszar wystarczający do zapisania wskaźnika (adresu pamięci).

Każda zmienna klasowa musi być zainicjowana (kreowana) w czasie wykonywania programu poprzez wywołanie konstruktora. Konstruktor rezerwuje odpowiednią pamięć i wpisuje adres zarezerwowanego obszaru w zmiennej obiektowej.

Zatem obowiązuje schemat postępowania:

```
type TObiekt = class
                 { wykaz składowych }
               end;

var ZmObiekt: TObiekt; {rezerwacja miejsca na wskaźnik }

begin

ZmObiekt:= TObiekt.Create; { rezerwacja miejsca na obiekt i wpisanie
                                  adresu do ZmObiekt }

{ wykorzystanie obiektu ZmObiekt }

ZmObiekt.Destroy;  { zwolnienie pamięci na obiekt }
end.
```

Wywołanie konstruktora powinno mieć postać instrukcji przypisania:

```
< zmienna obiektowa> := <typ obiektowy >.Create;
```

Do destrukcji obiektu można wykorzystać zamiennie metodę Free np.:

ZmObiekt.Free;

Przykład – przerobienie PObiekt – potem P8 i ew. p9

Zastosowanie metod obiektu TObject

Dodanie:

```
      Writeln(Ttrojkat.ClassName);              // wynik: Tc
      Writeln(Ttrojkat.InstanceSize);                 // wynik: 8
      Writeln(Ttrojkat.ClassParent.ClassName);        // wynik: TObject
      Writeln(Ttrojkat.InheritsFrom(TObject));        // wynik: TRUE
      Writeln(Ttrojkat.ClassNameIs('Ttrojkat'));            // wynik: TRUE
readln;
```

```
      Writeln(t.ClassName);               // wynik: Tc
      Writeln(t.InstanceSize);                  // wynik: 8
      Writeln(t.ClassParent.ClassName);         // wynik: TObject
      Writeln(t.InheritsFrom(TObject));         // wynik: TRUE
      Writeln(t.ClassNameIs('Ttrojkat'));       // wynik: TRUE
readln;
```