

ΜΑΤΗΜΑΤΙΧΕΣ ΜΟΔΕΛΙΣΜΟΙ

ΟΟΠ Β DELPHI

άγεωμέτρητοζ μηδείζ είσιτω

© Β. Ι. Ζενκιν, 2008 γ.



ναзад

закр.

ООП в Delphi. Delphi — это мощная, прекрасно организованная среда разработки для визуального программирования, основной компонент которой, Object Pascal, является объектно-ориентированным языком.

Рассмотрим реализацию принципов ООП в среде Delphi. Формат объявления класса

```
type Tclass_name = class( Tbase_class )  
    // поля, методы и свойства  
    ...  
end;
```

Здесь Tclass_name — имя класса (имена классов в Delphi принято начинать с буквы Т, но это не обязательно), Tbase_class — имя родительского для Tclass_name класса, *полями* называют инкапсулированные в классе данные, *методы* — процедуры и функции, *свойства (property)* — поля специального вида, регулирующие доступ к обычным полям. Класс может быть объявлен только в интерфейсной части модуля или самом начале области реализации, но не в разделе описаний подпрограмм.

Все классы в Object Pascal всегда порождены от единственного предка — класса TObject, который не имеет полей и свойств, но содержит методы самого общего назначения. При объявлении класса, непосредственного наследующего от TObject, его идентификатор может опускаться, т. е. можно писать: Tclass_name = *class* вместо: Tclass_name = *class*(TObject). В языке Object Pascal отсутствует множественное наследование — класс может иметь только одного непосредственного предка.

Дерево классов Object Pascal, имея в корне TObject, постепенно разрастается, включая всё новые и новые классы, каждый из которых обладает, кроме своих собственных, всеми возможностями своих предков. Из любого из этих классов программист может вывести свой собственный класс, добавив в него новые свойства и методы или переопределить родительские методы.

Для реализации графического интерфейса с пользователем Delphi использует библиотеку VCL (Visual Component Library — библиотека визуальных компонентов), которая содержит большое количество разнообразных классов, поддерживающих форму (окно) и различные её компоненты (командные кнопки, поля редактирования, меню и т. д.). Во время конструирования формы, по мере добавления на форму компонентов, Delphi автоматически вписывает в текст программы необходимые объявления.

При создании классов доступ к его данным извне ограничивается. Хотя в Object Pascal можно в некоторых случаях напрямую обратиться к полям объекта, это считается отступлением от принципов ООП, согласно которым это обращение должно осуществляться только с помощью методов класса и никак иначе. Для ограничения доступа к данным и методам задаются различные *области видимости*.



назад

закр.

В Delphi класс может содержать четыре секции, которые задают следующие области видимости:

1. *public* (общедоступные) — поля и методы этой секции доступны из любого модуля программы;
2. *published* (декларированные) — так же, как *public*, не ограничивает области видимости, используется только при разработке пользовательских компонент VCL Delphi, свойства, объявленные в этой секции, будут доступны в окне Object Inspector;
3. *protected* (защищённые) — поля и методы доступны только методам самого класса и его потомков, независимо от того в каком модуле они находятся;
4. *private* (личные) — минимально возможная область видимости, приватные элементы доступны только потомкам класса, причём только тем, которые размещены в том же модуле.

Внутри каждой секции вначале объявляются поля, а затем методы. Порядок следования секций произволен. По умолчанию, т.е. без явного задания ключевого слова, секция считается *published*.

Класс может содержать два специальных метода: конструктор (*constructor*) и деструктор (*destructor*). В любом конструкторе вначале вызывается конструктор родителя с использованием объявления (*inherited*), а затем, обычно, выполняются некоторые инициализирующие действия. Конструктор размещает объект в динамической памяти (т.е. до вызова конструктора объект не существует!) и автоматически объявляет указатель Self на выделенную объекту память. Деструктор уничтожает объект, возвращает память обратно в heap. Действия с объектом возможны только после вызова конструктора.

Методы базового класса можно изменять (перекрывать) в потомках, причём это можно сделать как *статически*, так и *динамически*. При статическом изменении объявляется процедура с таким же именем, что и в родительском классе, но выполняющая другие действия. При динамическом замещении метод родительского класса должен объявляться с директивой *virtual* или *dynamic*, а в классе-потомке — с директивой *override*. Компилятор создаст Таблицу виртуальных методов (ТВМ) или, соответственно, Таблицу динамических методов (ТДМ), в которые поместит точки входа соответствующих методов, и при каждом обращении к методам будет вставлять код для отыскания нужных процедур или функций в той или иной таблице.

Динамические методы могут вообще не выполнять никаких действий и лишь перекрываться в потомках. Такие методы называют *абстрактными*. Они объявляются с директивой *abstract*. Классы, содержащие такие методы, также называют абстрактными. Такие классы инкапсулируют методы, реализация которых откладывается до объявления соответствующего класса-потомка. Конечно, невозможно создать объект абстрактного класса и вызвать неперекрытый абстрактный метод.



назад

закр.

Рассмотрим простой **пример**. Пусть для работы с графикой нам требуются различные геометрические фигуры. Основной иерархии классов для таких фигур традиционно служит *точка*. Точка на плоскости характеризуется своими координатами и цветом, а также способом представления — метод Show. Этот метод объявлен виртуальным, так как будет переопределяться в потомках. Каждый класс лучше задавать в отдельном модуле.

```
unit Point;
interface
uses Forms, Graphics, Windows;
//-----
type TOЧКА = class // предок;
    protected
        forma : TForm;    // форма, где рисуется точка;
        x, y   : integer;  // координаты точки;
        cvet   : TColor;   // её цвет;
    public
        constructor Init( xn, yn : integer;
                           color  : TColor;
                           F       : TForm
                           );
        procedure Show; virtual;
    end;
//-----
```

Реализация методов класса ТОЧКА

implementation

constructor ТОЧКА.Init(xn, yn : integer;
 color : TColor;
 F : TForm
);

begin

inherited Create;

x := xn;

y := yn;

cvet := color;

forma := F;

end;

//-----

procedure ТОЧКА.Show;

begin

with forma.canvas *do*

begin

 pen.Color := cvet;

 ellipse (x - 5, y - 5, x + 5, y + 5);

 pixels[x, y] := cvet;

end;

end;

end.

Из базового класса ТОЧКА выведем производный KVADRAT, который унаследует от родителя цвет и координаты точки, определяющий центр квадрата. Метод Show, разумеется, нужно переопределить.

```
unit Quad;  
interface  
uses Point, Forms, Graphics, Windows;  
//-----  
type KVADRAT = class( ТОЧКА ) // наследник ТОЧКА;  
protected  
side_quad : word;           // сторона квадрата;  
public  
constructor init(  xn, yn, a : integer;  
                  color      : TColor;  
                  F          : TForm  
                  );  
procedure show; override;  
end;
```


implementation

```
constructor KVADRAT.init( xn, yn, a : Integer;  
                           color      : TColor;  
                           F          : TForm  
                           );
```

begin

```
inherited init( xn, yn, color, F );
```

```
side_quad := a;
```

end;

```
procedure KVADRAT.show;
```

```
var half_side : Integer;
```

begin

```
half_side := side_quad div 2;
```

```
  with forma.canvas do
```

begin

```
    brush.Style := bsClear;
```

```
    pen.Color   := cvet;
```

```
    Rectangle( x - half_side, y - half_side,  
               x + half_side, y + half_side );
```

end;

end;

end.

Другой **пример**. Создадим заготовку для модели шахматной игры. Объекты шахматные фигуры, очевидно, имеют общие свойства: цвет, координаты положения на доске и др. В соответствии с этим создадим базовый класс

```
// файл figure.pas
unit figure;
interface
uses board;
//-----
// класс Шахматная фигура
//-----
Type Tfigure = class
    protected
        ID      : Tid;          // идентификатор фигуры;
        board    : Tboard;      // используется board;
        function can_move(      // правильность хода;
                           to_position: Tposition
                           ): Boolean; virtual; abstract;
    public
        procedure move( to_position: Tposition);
        constructor Init ( ident      : Tid;
                           chessBoard : Tboard );
        destructor delete;
end;
```

Этот абстрактный класс содержит поле ID — идентификатор фигуры, включающий её номер (№0 — фигура отсутствует на доске, №1 — король, №2 — ферзь, №3 — ладья и т. д.), координаты на шахматной доске, цвет, а также объект board — «шахматная доска» (объявления класса Tboard и всех типов вынесены в отдельный модуль board.pas, см. ниже). Методы класса

```
//-----  
implementation  
//-----  
// ход фигуры в позицию to_position  
//-----  
procedure Tfigure.move( to_position: Tposition);  
var tempID : Tid;  
begin  
    if ( can_move( to_position ) )           // если ход правиль-  
    then begin                                // ный, запись фигуры  
        tempID := ID;                        // удаляется с "доски"  
        tempID.number := 0;                  // (№0 — нет фигуры) и  
        board.entry_fig_board( tempID );    // фигура записывается  
        ID.pos := to_position;               // в другой позиции;  
        board.entry_fig_board( ID );  
    end;  
board.refresh ;  
end;
```

```
//-----  
// инициализация  
//-----  
constructor Tfigure.Init( ident: Tid; chessBoard: Tboard );  
begin  
  ID := ident;  
  board := chessBoard;  
end;  
//-----  
// фигура удаляется с доски  
//-----  
destructor Tfigure.delete;  
begin  
  ID.number := 0;  
  board.entry_fig_board( ID );  
  board.refresh;  
end;  
end. // конец файла figure.pas
```

Класс Tboard предназначен для поддержки графики и интерфейса с шахматными фигурами и, помимо прочего, содержит поле fig_board — массив 8×8 идентификаторов фигур.

```
// файл board.pas
unit board;
interface
uses Forms, Graphics, Types;

Type Tletter   = ( _a, _b, _c, _d, _e, _f, _g, _h );
Type Tposition = record
    x : Tletter;      // координаты шахматных
    y : 1..8;         // фигур на доске;
end;

type Tfigcolor = ( black, white ); // цвет фигур;
Type Tid       = record // идентификатор фигуры;
    pos : Tposition; // положение;
    color : Tfigcolor; // цвет фигуры;
    number: 0..16; // № фигуры
    // 0 — отсутствует, 1 — король, 2 — ферзь,...
end;
```

Фигура «делает ход», посылая сообщение объекту `board` отметить её положение в этом массиве. Поэтому, кроме конструктора и деструктора этот класс имеет метод *move*, который осуществляет «ход» фигуры — посылает сообщение стереть запись своего текущего положения на доске и записать новые координаты в массив `fig_board`, предварительно проверив допустимость данного хода. Правильность хода проверяет логическая функция `can_move`. Поскольку эта функция индивидуальна для каждой фигуры, она объявлена абстрактной и перекрывается в соответствующих классах.

Далее приводятся только заголовки класса `Tboard` и его методов.

Type Tboard = class

protected

Forma : TForm; *// форма, куда всё рисуется;*

whiteCell : Tcolor; *// цвета клетки*

blackCell : Tcolor; *// шахматной доски;*

hCell : word; *// длина стороны клетки;*

fig_board : array [1..8, 1..8] of Tid;

function get_color_cell(i, j: word): boolean;

procedure draw;

function search_rect(pos: Tposition): TRect;

procedure show_figures;

// рисование фигур:

procedure draw_queen(pos : TPosition; color : Tfigcolor);

procedure draw_rook(pos : TPosition; color : Tfigcolor);

// здесь могут быть другие фигуры...

public

function search_pos(X, Y: word): Tposition;

procedure entry_fig_board(ID: Tid);

procedure refresh;

constructor init(white_cell, black_cell: Tcolor;
h_cell : word; form : TForm);

end;

Из класса Tfigure выведем классы-потомки Tqueen (ферзь) и Trook (ладья). Приведём листинги лишь для класса Tqueen, для другого программный код аналогичен.

```
// файл queen.pas
unit queen;
interface
uses figure, board;
//-----
Type Tqueen = class( Tfigure ) // класс Ферзь
    protected
    function can_move(
                                to_position: Tposition
                                ): Boolean; override;

    public
    constructor Init( ident: Tid; chessBoard: Tboard );
    destructor delete;
end;
```


implementation

```
//-----  
// может ли ферзь идти в to_position  
//-----  
function Tqueen.can_move( to_position: Tposition ): Boolean;  
begin  
  result := ( ID.pos.x = to_position.x )  
             OR ( ID.pos.y = to_position.y )  
             OR ( Abs( ID.pos.y - to_position.y ) =  
                 Abs( ord(ID.pos.x) - ord(to_position.x) ) );  
end;  
//-----  
constructor Tqueen.Init( ident: Tid; chessBoard: Tboard );  
begin  
  inherited Init( ident, chessBoard );  
end;  
//-----  
destructor Tqueen.delete;  
begin  
  inherited delete;  
end;
```

Теперь, при желании, легко можно продолжить добавлять другие фигуры, снова наследуя от базового класса Tfigure. **Программа в действии.**

Список литературы

1. В. И. Зенкин. Практический курс математического и компьютерного моделирования. Учеб. пособие. Калининград: изд. РГУ им. Канта, 2006.
2. В. В. Фаронов. Delphi 3. Учебный курс. М.: Нолидж, 1998.
3. Т. Бадд. Объектно-ориентированное программирование в действии. СПб, 1998.
4. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. М.-СПб., 1998.
5. Д. Тейлор, Дж. Мишель, Дж. Пенман, Т. Гоггин, Дж. Шемитц. Delphi 3: библиотека программиста. Спб, 1996.
6. Д. Кнут. Искусство программирования для ЭВМ. М., Т. 1, Т. 2: Получисленные алгоритмы, 1977; Т. 3: Сортировка и поиск, 1978.



назад

закр.