Объекты

- Процедурное структурированное программирование
- Язык программирования Фортран
- Организация данных
- Языки программирования:Алгола —60, Паскаля, Модулы —2, Си
- Развитие структуры типов данных
- Модульный подход
- Объектно-ориентированное программирование(ООП)
- Связь данных с обрабатывающими этих данных процедурами в единое целое – объект.
- Инкапсуляция(объединение) данных и алгоритмов их обработки
- Основные идеи ООП
- Преимущество ООП в полной мере проявляются лишь в разработке достаточно сложных программ.

Основные принципы ООП

Объектно-ориентированное программирование основано на трех важнейших принципах: **ИНКАПСУЛЯЦИЯ**, **НАСЛЕДОВАНИЕ** и **ПОЛИМОРФИЗМ**.

ИНКАПСУЛЯЦИЯ - есть объединение в единое целое данных и алгоритмов обработки этих данных.

Данные называются полями объекта, а алгоритмы-объектными методами. Легкость обмена объектами переноса их из одно программы в другую. ООП “провоцирует” разработку библиотек объектов.

Принцип “ наследования и изменения”.

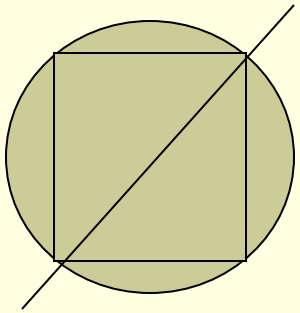
НАСЛЕДОВАНИЕ - есть свойство объектов порождать своих потомков.

ОБЪЕКТ-потомок автоматически наследует от родителя все поля и методы, может дополнять объекты новыми полями и заменять (перекрывать) методы родителя или дополнять их.

ПОЛИМОРФИЗМ - это свойство родственных объектов решать схожие по смыслу проблемы разными способами. Для изменения метода необходимо перекрыть его в потомке, т.е. Объявить в потомке одноименный метод и реализовать в нем нужные действия.

ВИРТУАЛИЗАЦИЯ.

Разработать программу, которая создает на экране ряд графических изображений (точки, окружности, линия, квадрат) и может перемещать эти изображения по экрану.



Для перемещения клавиши:

Home, End, PgUp, PgDn и клавиша **Tab** для выбора перемещаемого объекта. Выход из программы — клавиша **Esc.**

Техническая реакция библиотек - **CRT и GRAPH**

Создание объектов

Используются три зарезервированных слова: object, constructor, destructor и три стандартные директивы: **private, public** и **virtual.**

Object используется для описания объекта в разделе типов:

```
type
MyObject = object
                {Поля объекта}
                {Методы объекта}
end;
```

Если объект порождается от родителя:

```
type
MyDescendantObject = object(MyObject)
                                        …
end;
```

Иерархические деревья наследования объектов.

Создадим объект-родитель TGraphObject

```
type
TGraphObj = object
Private                    {Поля объекта будут скрыты от пользователя}
X,Y: Integer;              {Координаты реперной точки}
Color: Word;               {Цвет фигуры}
Public                     {Методы объекта будут доступны пользователю}
Constructor Init(aX,aY: Integer; aColor: Word);
                           {Создает экземпляр объекта}
Procedure Draw(aColor: Word); Virtual;
                           {Вычерчивает объект заданным цветом aCdlot}
Procedure Show;
                  {Показывает объект - вычерчивает его цветом Color}
Procedure Hide;
                    {Прячет объект - вычерчивает его цветом фона}
Procedure MoveTo(dX,dY: Integer);
                 {Перемещает объект в точку с координатами X+dX и
    Y+dY}
end;                       {Конец описания объекта TGraphObj}
```

Объявим абстрактный объект **TGraphObj**

Private –скрытие поле и методов “не видимых программисту”.
В примере он не может менять координаты точки (X,У).
Для изменения полей X и У предусмотрены методы Init и Move To.
TGraphObj следующие за public элементы объекты доступны в любой программной единице.
Объявление объекта **TGraphObj** без использования механизма **private...public:**

```
type
TGraphObj = object
X,Y: Integer;
Color: Word;
Constructor Init(aX,aY: Integer; aColor: Word);
Procedure Draw(aColor: Word); Virtual; Procedure Show;
Procedure Hide;
Procedure MoveTo(dX,dY: Integer);
end;
```

Для описания методов в ООП используются традиционные процедуры и функции, а так же особый вид процедур-конструкторы и деструкторы.

Конструкторы предназначены для создания конкретного экземпляра объекта, т.е. “шаблон”, по которому можно создать сколько угодно рабочих экземпляров данных объектного типа.

Зарезервированное слово constructor используется вместо procedure.

Virtual.

Конструктор Init объекта **TGraphObj.** Процедура **Draw** реализовывается в потомках объекта **TGraphObj** по разному. Для визуализации точки - вызвать процедуру **PutPixel**, для вычерчивания линии –процедура **Line** и т.д.

Draw определена как виртуальная(“ воображаемая”).

Абстрактный объект **TGraphObj** процедуры **Draw .**

Любой поток TGraphObj должен иметь собственный метод Draw .

При трансляции объекта таблица виртуальных методов (ТВМ).

ТВМ объекта **TGraphObj** – адрес метода **Draw.**

TGraphObj с родительского метода **Draw.**

В методе **Show,** вызывается **Draw** с цветом **aColor, Color;** в методе **Hide** вызывается **Draw** со значением цвета **GetBkColor.**

Если потоком **TGraphObj** (например, **TLine**), хочет переместить себя на экране, он обращается к родительскому методу **Move To** .

Draw , MoveTo в рамках объекта **Tline, TBM, Draw .**

Если бы перемещалась окружность, ТВМ содержала бы адрес метода **Draw** объекта **TCircle**.

```
type
TGraphQbj = Object . . .
end;
Constructor TGraphObj.Init;  begin
X := aX; Y := aY; Color := aColor end;
Procedure TGraphObj.Draw; begin
{процедура в родительском объекте
    ничего не делает, поэтому
    экземпляры TGraphObj не способны
    отображать себя на экране. Чтобы
    потомки объекта TGraphObj были
    способны отображать себя, они
    должны перекрывать ztot метод).
end;
Procedure TGraphObj.Show;
begin
    Draw(Color) end;
Procedure TGraphObj.Hide;
begin      Draw(GetBkColor)      end;
Procedure TGraphObj.MoveTo;
Begin      Hide;
X := X+dX; У := Y+dY;
Show end;
     Begin end;
```

```
Constructor TGraphQbj.Init;
var
   x,y: integer;
    {Ошибка!}
Color: Word;
    {Ошибка!}
begin
end;
```

Сообщение о двойном определении переменных *X, Y* и *Color*

- Создадим простейшего потомка **TGraphObj** -объект **Tpoint**, с помощью будет:

```
type
TPoint = object (TGraphObj)
    Procedure Draw(aColor); Virtual; end;
Procedure TPoint.Draw;
begin
PutPixel(X,Y,Color)   {Показываем цветом Color пиксель с
    координатами X                  и Y}
end;
```

MoveTo, TGraphObj. MoveTo, TPoint.Draw, Init объекта **TPoint**

Если вызвать**TPoint. Draw** до вызова Init, его ТВМ не будет ~~содержать правильного адреса~~.

Что бы создать объект-линию, необходимо ввести два новых поля для хранения координат второго конца.

```
type
TLine = object (TGraphObj)
dX,dY: Integer;    {Приращения координат второго конца}

Constructor Init (X1,Y1,X2,Y2: Integer; aColor: Word);

Procedure Draw(aColor: Word); Virtual; end;

Constructor TLine.Init;
    {Вызывает унаследованный конструктор TGraphObj для инициации полей X, Y и Color. Затем инициирует поля dX и dY}
Begin
{Вызываем унаследованный конс-тор}
Inherited Init
(X1,Y1,aColor) {Иниуиируем поля dX и dY}
dX := X2-X1;
dY := Y2-Y1
end;
Procedure Draw;
begin
SetColor (Color) ;
    {Устанавливаем цвет Color}
Line (X, Y,X+dX, Y+dY)
    {Вычерчиваем линию}
end;
```

- В конструкторе **TLine.Init** для инициации полей X, У и Color
- Конструктор **TGraph.Init** для **inherited** .
- **Inherited Init (X1, Y1,aColor);**
- **TGraphObj. Init (X1, Y1,aColor);**
- **dX и dY, Tline dX и dY.**
- X,Y в родительском методе **TGraph.MoveTo.**
- Реализовать объект **Tcircle**
- Окружность

```
type
TCircle = object(TGraphObj)
R: Integer;  {Радиус}
 Constructor Init(aX,aY,aR: Integer;
    aColor: Word);
Procedure Draw(aColor: Virtual);
end;
Constructor TCircle.Init;
begin
Inherited Init(aX,aY,aColor);
R := aR end;
Procedure TCircle.Draw;
begin
SetColor (aColor) ; {Устанавливаем
    цвет          Color}
    .
Circle(X,Y,R)
          {Вычерчиваем окружность}
end;
```

Объект TRect

удобней породить не от **TGraphObj**, а от **Tline, Init**

```
type
TRect = object(TLine)
    Procedure Draw(aColor:
    Word); end;
Procedure TRect.Draw;
 begin SetColor(aColor);
Rectangle (X,Y,X+dX,Y+dY)
    {Вычерчиваем прямоугольник}
    end;
```

Чтобы описания графических объектов не мешали созданию основной программы, оформим эти описания в отдельном модуле **GraphObj:**

```
Unit GraphObj; Interface
{Интерфейсная часть модуля
    содержит только
объявления объектов}
type
TGraphObj = object
end;
TPoint = object(TGraphObj)
    end;
TLine = object(TGraphObj)
end;
TCircle = object(TGraphObj)
end;
TRect = object(TLine)
end; Implementation

{Исполняемая часть содержит
Описания всех объектных методов}

Uses Graph;
Constructor TGraphObj.Init;
end.
```

- Инкапсуляция
- Инициация(**Init**)
- Выполнение работы(**Run**)
- Завершение(**Done**)
- **TPoint** и по одному экземпляру **TLine, TCircle, TRect**
- **TGraphApp** в модуле **GraphApp**

```
Unit GraphApp;
Interface
type
TGraphApp = object
Procedure Init;
Procedure Run;
Destructor Done; end;
Implementation Procedure
    TGraphApp.Init;
end;
•
end.
```

```
Program Graph_0bjects;
Uses GraphApp;
var
App: TGraphApp; begin
App.Init; App.Run; App.Done end.
```

В ней мы создаем единственный экземпляр **App** объекта — программы **TGrahpApp** и обращаемся к трем его методам.

```
var
App: TGraphApp;
```

Получив это указание, компилятор зарезервирует нужный объем памяти для размещения всех полей объекта **TGraphApp.**

```
App.Init;
App.Run;
App.Done;
```

Переменные объектного типа могут быть статическими или динамическими, т.е. располагаться в сегменте данных (статистические) или в куче (динамические)

```
Program Graph_0bjects;
Uses GraphApp; type
PGraphApp =:^TGraphApp;
Var App: PGraphApp;
begin
App := New(PGraphApp,Init)
App^ .Run;
App^.Done end;
```

Вариант модуля **GraphApp**

```
Unit GraphApp; Interface
Uses GraphObj;
const
NPoints = 100;
    {Количество точек}
type
{Объект-программа}
TGraphApp = object
Points: array [1..NPoints] of
    TPoint; {Массив точек}
Line: TLine;     {Линия}
Rect: TRect;
    {Прямоугольник}
Circ: TCirele;       {Окружность} .
    ActiveObj: Integer;
                {Активный объект}
Procedure. Init;
```

```
Procedure Run; Procedure Done;
Procedure ShowAll;
Procedure MoveActiveObj (dX,dY: Integer); end;
Implementation Uses Graph, CRT;
Procedure TGraphApp.Init;  {инициирует графический режим
    работы экрана.  Создает и отображает NPoints экземпляров
    объекта TPoint, а также экземпляры
    объектов TLine,  TCircle и TRect}
 var
D,R,Err,k: Integer; begin
{Инициируем графику}

D := Detect;        {Режим автоматического определения
                     типа графического адаптера}
InitGraph (D, R, ' \tp\bgi ' ) ; {Инициируем графический режим.
                                  Текстовая строка должна задавать путь
    к каталогу с графическими драйверами}
Err := GraphResult;  {Проверяем успех инициации графики}
if Err<>O then begin
GraphErrorMsg(Err) ; Halt end;
```

```
{Создаем точки}
for k := 1 to NPoints do
Points [k].Init (Random (GetMaxX) , Random (GetMaxY) ,
    Random ( 15 ) +1) ;
{Создаем другие объекты}
Line. Init (GetMaxX div 3, GetMaxY div 3,2*GetMaxX div 3,
    2*GetMaxY div  3, LightRed) ;
Circ. Init (GetMaxX div 2, GetMaxY div 2, GetMaxY div 5, White);
Rect.Init (2*GetMaxX div 5,2*GetMaxY div 5, 3* GetMaxX div 5,
3*GetMaxY div 5, Yellow);
ShowAll;
    {Показываем все графические объекты}
ActiveObj := 1       {Первым перемещаем прямоугольник} end;
    {TGraphApp.Init}
Procedure TGraphApp.Run ;
{Выбирает объект с помощью Tab и перемещает его по экрану}
```

```
var
Stop: Boolean;
    {Признак нажатия Esc}
const
D = 5;
    {Шаг смещения фигур}
begin
Stop := False;
{Цикл опроса клавиатуры}
repeat
case ReadKey of    {Читаем код #27: Stop := True;
# 9: begin        .....
inc(ActiveObj);
 if ActiveObj>3
   then . ActiveObj := 3 end; #0:
case ReadKey of
```

#71: MoveAdtiveObj(-D,-D) MoveActiveObj(0,-D) MoveActiveObj(D,-D) MoveActiveObj(-D, 0) MoveActiveObj(D, 0) MoveActiveObj(-D, MoveActiveObj(0, MoveActiveObj(D,нажатой клавиши} {Нажата Esc} {Нажата Tab}

#72 #73 #75 #77 #79 #80 #81

end end;

ShowAll; Until Stop end; { TGraphApp .Run}D) D) D)

{Влево и вверх} {Вверх} {Вправо и вверх}

{Влево} {Вправо}

{Влево и вниз} {Вниз}

{Вправо и вниз}-

Destructor TGraphApp.Done; {Закрывает графический режим}

begin

CloseGraph end; { TGraphApp .Done}

Procedure TGraphApp. ShowAll; {Показывает все графические объекты}

```
var
k: Integer;
begin
for k i:= 1 to NPoints do Points[k].Show;
Line.Show;
Rect.Show;
Circ.Show end;
Procedure TGraphApp.MoveActiveObj; {Перемещает активный
    графический объект} begin

case ActiveObj of
1: Rect.MoveTo(dX,dY);
2: Circ.MoveTo(dX,dY);
3: Line.MoveTo(dX,dY)
end
end;
end.
```

- Конструктор осуществляет настройку ТВМ, деструктор не связан с какими-то специфичными действиями : **destructor** и **procedure** –синонимы.

- Процедуру разрушающую экземпляр объекта, принято называть деструктором.
  Он прекращает работу с объектом и освобождают выделенную для него динамическую память.

- Формулистика ООП - введение лишь шести зарезервированные слова, необходимые: **object, constructor** и **virtual**.

- Мощный инструмент создания программного обеспечения.