**Основы ООП в Delphi**

*Предпосылки возникновения ООП*

Рост сложности кода:

* Следование/ветвление/цикл
* Процедуры
* Модули
* …

Рост сложности данных:

* Скалярные типы
* Агрегированные (составные) типы
* …

ООП – парадигма (идея)

Программа состоит из объектов, которые обмениваются сообщениями

*Эволюция ООП*

Type

TButton = record

X, Y, W, H: Integer;

…

End;

Procedure SetPos(var Button: TButton; X, Y: Integer);

Type

TListBox = record

X, Y, W, H: Integer;

Items: array of String;

…

End;

Procedure SetPos(var ListBox: TListBox; X, Y: Integer);

…

Procedure Button\_SetPos(var Button: TButton; X, Y: Integer);

Procedure ListBox\_SetPos(var ListBox: TListBox; X, Y: Integer);

Но вот идея:

Type

TControl = record

X, Y, W, H: Integer;

Visible: Boolean;

End;

TButton = record

ControlData: TControl;

…

End;

TListBox = record

ControlData: TControl;

…

End;

Procedure Control\_SetPos(var Control: TControl; X, Y: Integer);

Control\_SetPos(Button.ControlData, 100, 200);

Button\_Draw(Button)

«Три кита» ООП:

* Инкапсуляция
* Наследование
* Полиморфизм

Инкапсуляция

Объединение данных и кода, который их обрабатывает

Наследование

Возможность создавать новые типы данных путем расширения существующих

Полиморфизм

Возможность использовать различные типы данных взаимозаменяемо

Type

TSetPosProc = procedure (var Control; …);

TDrawProc = procedure (var Control);

…

TButton = record

ControlData: TControl;

Caption: String;

SetPos: TSetPosProc;

Draw: TDrawProc;

End;

MyButton.SetPos(MyButton, 100, 200);

MyButton.Draw(MyButton);

Было:

Control\_SetPos(Button.ControlData, 100, 200);

Button\_Draw(Button)

ООП не привязан к ЯП:

* Писать в ООП-стиле можно на любом языке программирования.
* Но проще, если в ЯП есть поддержка ООП (на уровне синтаксиса)

Type

TControl = class

X, Y, W, H: Integer;

Visible: Boolean;

Procedure SetPos(X, Y:Integer);

Procedure Draw;

…

End;

Все объекты в Delphi выделяются в динамической памяти.

* Переменная хранит адрес объекта

В теле каждого метода доступна переменная Self.

* Содержит адрес объекта, у которого был вызван метод.
* (Кроме методов класса)

**Наследование**

Класс может наследоваться от другого класса

* При этом наследуемый класс будет содержать все члены родительского.
* Но сможет добавить новые…
* … или изменить видимость существующих
* Если не указать родительский класс, им будет TObject

Type

TShape = class

FColor: TColor;

End;

TRectangle = class(TShape)

FWidth, FHeight: Real;

Procedure Draw;

End;

(TRectangle теперь как TShape, но + FWidth, FHeight и Procedure Draw;

Членам класса могут задаваться различные уровни видимости:

* Private
* Protected
* Public
* Published

Private

* Внутри методов этого же класса

Protected

* Внутри методов этого же класса…
* … или наследуемых от него.

Public

* Везде.

В Delphi есть некоторые особенности:

* Private-члены доступны всем внутри модуля, в котором объявлен класс
* Есть published-видимость (используется для интеграции со средой);
* По умолчанию – public/published

*Члены класса*

Члены класса:

* Поля
* Методы
* Свойства/события

Могут принадлежать:

* Экземплярам класса (объектам)
* Самому классу

Реализация метода

Определяющее описание (реализация) метода:

* По синтаксису аналогично обычным подпрограммам/функциям
* Но используют составное имя

Procedure TMyClass.SomeMethod(…);

Begin

…

End;

*Обращение к унаследованным методам*

Для обращения к унаследованным методам используется ключевое слово inherited.

// Вызов унаследованного метода Draw

inherited Draw(…);

// Вызов метода с тем же именем,

// Что и вызывающий, и с теми же параметрами

inherited

Методы могут быть:

* Статическими
* Виртуальными
* Динамическими
* Методами класса

Особо выделяют:

* Конструкторы
* Деструкторы

*Конструкторы*

Вводятся ключевым словом constructor. (Крч для инициализации, при создании)

* Могут иметь произвольные имена, но общепринятое – Create

Создание объекта:

Var

MyObj: TShape;

…

MyObj := TShape.Create(…);

*Деструкторы*

Вводятся ключевым словом destructor. (Крч для free памяти при удалении)

* Могут иметь произвольные имена, но общепринятое – Destroy. (Напрямую не вызывают Destroy (FPen.Free; inherited Destroy), т.к. FPen может быть nil???) (FPen – пример)

Поля создаваемого объекта инициализируются нулями.

Если конструктор выбрасывает исключение, создание объекта прерывается.

* Деструктор при этом будет вызван…
* … но объект при этом всё еще инициализирован не полностью (некоторые поля содержат нули)

У класса TObject реализован метод Free:

Procedure TObject.Free;

Begin

If Self <> nil then

Destroy;

End;

*Процедура FreeAndNil*

Частая ошибка – двойное освобождение памяти.

* Например, разрушение уже разрушенного объекта

В модуле SysUtils объявлена процедура FreeAndNil:

* Вызывает метод Free;
* Записывает в переменную-объект nil

**Свойства**

Свойство – способ доступа к внутреннему состоянию объекта, имитирующий поведение переменной.

* Внутреннее состояние объекта задается значениями его полей.

Свойство может быть реализовано.

* Прямым доступом к какому-либо полю
* В виде метода, вызываемого при обращении к свойству

Property Color: TColor read GetColor write SetColor;

Для прогера:  
MyColorClass.Color := clRed 🡪 вызовется SetColor

*Свойства-массивы*

Property Items[Index: Integer]: TObject read GetObject write SetObject

*Индексированные свойства*

Property Left: LongInt index 0 read GetCoordinate write SetCoordinate