**1. Основные определения типа <объект>**

**Объект**- это структура данных, содержащая поля данных (подобно записи) различных типов и заголовки методов.

Синтаксис объявления объекта:   
*< ИмяПотомка > = Object(< ИмяПредка >)  
                        поле;  
                        ............  
                        поле;  
                        метод;  
                        .............  
                        метод;  
                      End;***Метод**- это процедура или функция, объявленные внутри объявления элемента типа объект и предназначенная в обычно для работы с полями этого объекта.   
Формат обьявления

*Procedure <ИмяМетода>(<Параметры, как у процедуры>);*Метод имеет доступ к полям данных объекта, не требуя передачи их ему в виде параметров.

Объявление метода внутри объявления объектного типа содержит только заголовок. Тело метода определяется вне объявления объекта. Его заголовок должен содержать имя объекта, которому принадлежит метод.

Методы подразделяются на статические и виртуальные. Виртуальный метод отличается от статического тем, что реализующий его код подсоединяется к вызову не в процессе компиляции, а в процессе выполнения, что достигается так называемым поздним связыванием. Это дает возможность строить иерархию объектов с одинаковыми названиями методов, реализуемыми, однако, различными кодами.

Синаксис объявления виртуального метода: *Procedure Метод(< параметры >); virtual;*

Кроме обычных процедур и функций Турбо-Паскаль 6.0 реализует два специальных типа методов: конструктор и деструктор.

**Конструктор** – це спеціальний вид процедури, що виконує установчі роботи по віртуалізації метода(забезпечення пізнього звязування). Всі типи, що мають вірт. методи, повинні мати конструктор. Конструктор завди викликається до першого виклику вірт. метода.   
Синтаксис: *constructor Init(< параметры >);*

Конструктор инициализирует объект установлением связи между объектом и специальной таблицей виртуальных методов, содержащей адреса кодов, реализующих виртуальные методы. Конструктор может также использоваться для инициализации полей данных объекта.

**Деструктор** - это специальный метод, освобождающий память кучи от динамических объектов. Он объявляется с использованием специально зарезервированного слова destructor.   
Синтаксис: *destructor Done;*

Деструктор може унаслідуватись. Крім того, якщо об’єкт містить віртуальні методи, то деструктор виконує пошук розміру об’єкта в табл. VMT і передає його процедурі Dispose.

**2. Основные свойства объектов**

Основными отличительными свойствами объекта являются:   
  
• инкапсуляция - объединение записей с процедурами и функциями, работающими с этими записями**; Інкапсуляція – обєднання в єдиній структурі полів і методів;**• наследование - задание объекта, затем использование его для построения иерархии порожденных объектов с наследованием доступа каждом из порожденных объектов к коду и данным предка;  Наслідування – дозволяє створити новий обєкт на базі вже існуючих, причому поля «нащадки» успадковуют поля та методи «батька» («нащадок» - похідний обєкт);  
• полиморфизм - задание одного имени действию, которое передается вверх и вниз по иерархии объектов, с реализацией этом действия способом, соответствующим каждому объекту в иерархии. ***Поліморфмізм – це існування декількох однакових за іменем, але різними за змістом методів для різних класів;***

Турбо-Паскаль разрешает сохранить потомку имя родительского метода, "перекрывая" его. Чтобы перекрыть родительский метод, нужно просто задать его с тем же именем, но с другим телом (кодом) и, если необходимо, с другим набором параметров. Такой метод делается виртуальным и к его объявлению добавляется слово virtual. Применение виртуальных методов налагает ограничения на процедуры инициализации, которые должны записываться с зарезервированным словом constructor и иметь общее имя Init.

Каждый отдельный экземпляр объекта должен инициализироваться с помощью отдельного вызова конструктора.

Для очистки и убирания динамически распределенных объектов существует специальная процедура - destructor Done.

Деструктор комбинирует шаг освобождения памяти в "куче" с некоторыми другими задачами. Метод деструктора может быть пустым, поскольку работу выполняет не только код тела, но и код, генерируемый Турбо-Паскалем в ответ на зарезервированное слово destructor.

**Директиви private, public.**

В рамках описання об’єкта директива private виконує туж роль що й implementation в модулях.

Всередині об’єкта можна створити приховані від користувача поля/методи. Приватні поля/методи доступні в тому ж модулі, де описаний об’єкт.

Type person = object

{Загальнодоступні поля}

{загальнодоступні методи}

Private

{приватні поля}

{приватні методи}

End;

Public – дозволяє розташувати приватні поля/методи вище загальнодоступних.

**Віртуальні методи**

Procedure <імя>(<параметр>); virual;

При використанні статичних методів втрачаються надбання поліморфізму, тому що наперед невідомо який об’єкт та якого типу буде переданий як параметр і яку однойменну пр./ф-цію треба викликати.

При віртуалізації метода потрібно дотримуватись таких правил:

1)Якщо породжуючий об’єкт описував метод як віртуальний, то і всі похідні об’єкти, що породжують метод з тим самим іменем повинні описуватись теж як віртуальні.

2)Якщо пере означується вірт.метод, то його заголовок у новій реалізації в довільному типу не може бути змінений.

3)В описуванні об’єкта обов’язково мусить бути присутній спец.метод, що ініціалізує об’єкт. В оголошенні метода замість “Procedure” вживається “Constructor”.

Якщо для об’єкта визначений хоча б один вірт. метод , то для нього в сегменті даних програми буде створена таблиця VMT – virtual method table:1)розмір типу об’єкта;

2)адреси кодів віртуальних пр./ф.

**Модуль** - це програма, яка має спеціальну структуру, містить свої внутрішні описування та виконавчу частину, але сама організація роботи модуля може бути схованою, оскільки модуль розподіляється на окремі частини, не всі з котрих є доступними для користувача.

**Структура модуля:**

UNIT <ідентифікатор модуля>{заголовок}

INTERFACE

[USES <список модулів>;]{інтерфейсна секція}

<відкриті описування>;

IMPLEMENTATION

[USES <список модулів>;]{секція реалізації}

<закриті описування>;

[begin <операторна частина>]{секція ініціалізації}

end.

a) <ідентифікатор модуля> - визначає ім'я, за яким буде виконуватись звернення до модуля, повинен співпадати з ім'ям файлу, що містить модуль.

б) у інтерфейсній частині - описуються ті константи,

змінні, типи, заголовки процедур, заголовки функцій

які є глобальними, тобто доступні програмі чи модулю, що використовує наданий.

в) у секції реалізації - містяться локальні для наданого модуля описування:

константи, помітки, типи, змінні, процедури, їх тіла функції тобто невидимі для інших модулів чи програм. Але те, що описано у інтерфейсній секції, є видимим для секції реалізації.

г) секція ініціалізації - містить 'код ініціалізації' чи є пустою (END.)

Не пуста секція:

begin <оператори> END.

Як правило містить оператори, що дозволяють ініціалізувати (визначити початкові значення) структур даних, які використовує наданий модуль чи котрі він робить доступними для інших модулів. Наприклад - відкрити файл, визначити початкові значення масиву та т.і.).

**Перехресні посилання на модулі**

Можна робити перехресні посилання, як правило, на процедури чи функції

unit u1 unit u2

interface interface

implementation implementation

uses u2; uses u1;

: :

end. end.

Такі перехресні посилання можуть виконуватись тільки у секції реалізації, також треба слідкувати, щоб описання з'являлись до їх використання. У цьому випадку інтерфейсні секції можуть бути скомпоновані за загальним правилом (у прикладі - у будь-якому порядку, оскільки інтерфейсні секції не залежать одна від одної).

***Приклад***

а)

**program ww;**

uses crt,display,error;

begin

writexy(1,1,'лівий верхній кут');

writexy(100,100,'правий нижній кут'); {на 80х25 неможливо}

end.

б)

**unit display**

interface

procedure writexy(x,y:integer;message:string);

implementation

uses crt,error;

procedure writexy;

begin

if (x in [1..80] and y in [1..25] then

begin goto(x,y);

write(message);

end;

else

showerror('невірні координати');

end;

end.

в)

**unit error;**

interface

procedure showerror(msg:string);

implementation

uses display;

procedure showerror;

begin

writexy(1,25,'помилка: '+msg);

end;

end.