Дмитриева Марина Валерьевна

ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)

В статье объектно-ориентированная модель демонстрируется на концепции структур данных. Линейный цепной список рассматривается как объект, определяются его свойства и методы, описываются объекты – потомки, их взаимодействие с объектом – предком.

ПОНЯТИЕ ОБЪЕКТА

При использовании многих языков программирования предоставляются средства для укрупнения таких важных понятий как действия и данные.

Подпрограммы позволяют выделить группу действий таким образом, чтобы эту группу один раз в программе описать, а затем рассматривать ее как простое действие, используя каждый раз, когда в этом возникает необходимость. В языке Turbo Pascal понятие подпрограммы воплотилось в процедуры и функции, описание и использование которых подробно рассмотрено в предыдущих статьях.

...«абстрактный» означает, что при работе с объектом... не интересует внутренняя реализация объектов...

При обработке сложных данных необходимость в подобных средствах возникает не только в отношении действий, но и в отношении данных. Многие языки программирования предоставляют средства структурирования данных. Независимую часть можно выделить, объединив в единое целое данные (константы, переменные) и процедуры с функциями. Для этого можно использовать модули.

С каждым вновь определенным типом данных связывался набор операций, которые предусматриваются для обработки данных такого типа. Например, определяя стек как множество данных, вводились операции работы со стеком.

Тот факт, что новый тип данных должен связываться с набором операций для данного типа, привел к понятию абстрактного типа данных, широко используемого в современном программировании.

Важным моментом является тот факт, что обработка таких объектов должна осуществляться только с помощью введенных операций. Термин «абстрактный» означает, что при работе с объектом (с помощью заданного набора операций) нас не интересует внутренняя реализация объектов. В этом смысле объекты рассматриваются как «абстрактные».

Однако на первом этапе создания объекта, то есть при определении операций над объектами некоторого нового

типа необходимо иметь доступ к структуре объектов этого типа. После того как все операции над объектом определены, знание структуры объектов не является необходимым, так как работа с объектом должна обеспечиваться с помощью операций над объектом.

Концепция абстрактных типов данных подразумевает два уровня работы с объектом. На одном уровне определяются операции над объектом. В этом случае необходимо знание конкретной структуры объекта. Второй уровень – работа с объектом посредством введенных операций. В этом случае совсем необязательно (а иногда и невозможно) знать структуру объектов.

Многие языки предоставляют средства, с помощью которых можно воплотить идею абстрактных типов данных. В языке Turbo Pascal для поддержки идеи программирования с абстрактными типами данных имеется аппарат модулей (начиная с версии 4.0) и аппарат объектов (начиная с версии 5.5).

Объектно-ориентированное программирование (ООП) появилось в результате развития идей структурного программирования и абстрактных типов данных. В основе технологии ООП лежит понятие объекта. Объект обладает некоторым внешним видом или *свойствами* и характеризуется своим поведением или *методами*.

Свойства объекта отражены в значениях констант и переменных, а методы определяются с помощью процедур и функций. Свойства и методы объединены вместе, не существуют отдельно друг от друга. Удобно считать, что методы «охватывают» или «окружают» свойства объекта, не позволяя напрямую обращаться к ним и менять их значения. Свойства как бы заключены в капсулу объекта (инкапсулированы в нем). *Инкапсуляция* означает объединение в одном объекте данных (свойств) и действий над ними, то есть методов. Доступ к свойствам обеспечивается с помощью методов. Объект можно представить с помощью схемы на рис. 1.

Рис. 1. Свойства и методы объекта

Свойства заключены в ядре объекта, находятся под защитой от несанкционированного или случайного доступа. Методы являются некоторой внешней оболочкой и представляют программный интерфейс для доступа к свойствам объекта. Инкапсуляцию можно трактовать как расположение констант и переменных (определяющих свойства объекта) под защитой методов. С другой стороны свойство инкапсуляции позволяет скрыть детали реализации объекта. Важно знать методы, которые может выполнять объект. Итак, свойства инкапсулированы в объект и доступ к ним осуществляется только с помощью методов, предоставляемых объектом.

Объект характеризуется тремя основными свойствами: *инкапсуляция, наследование, полиморфизм.*

ОБЪЕКТЫ В ЯЗЫКЕ TURBO PASCAL

Для определения типа *объект* в языке Turbo Pascal используется слово **`object`**, за которым следует описание полей, представляющих данные, и перечислены заголовки процедур и функций, называемых методами (операциями), завершается описание словом **`end`**.

Предположим, требуется описать объект «линейный цепной список» (в дальнейшем просто «список»). Для того чтобы задать список, следует задать ссылку на его первый элемент. Эта ссылка будет представлять данные.

Определим три операции для работы со списком: инициализацию списка, вставку элемента в список и обход списка. При инициализации списка выполняются начальные действия по организации работы

Листинг 1. Описание объекта «Линейный список»

```
type elem_list = integer; {элемент списка}
     pelem= ^elem;
     elem=   record info: elem_list; next: pelem end;
     action= procedure (var z: elem_list);
     {описание объекта list}
     list= object
            first: pelem; {указатель на первый элемент}
            procedure init; {инициализация списка}
            procedure insert (n: elem_list);
                      {вставка элемента в список}
            procedure trav(p: action) {обход списка}
           end
```

со списком. Обход списка состоит в просмотре всех элементов списка с выполнением над ними одного и того же действия.

Такое описание объекта «список» представлено в листинге 1. В тексте программ полужирным начертанием выделяются конструкции, относящиеся к объекту, его свойствам и методам. Сами методы подробно не описываются, так как в предыдущих статьях подробное внимание уделялось реализации алгоритмов обработки списков.

Типы **elem_list**, **pelem**, **elem** – вспомогательные при описании объекта **list**. Объект **list** совмещает в себе данные (**first** – ссылка на первый элемент списка) и три метода работы с этим объектом.

При описании объекта в одном месте определены все свойства объекта, что облегчает понимание работы программы, ее отладку и модификацию. К данным объекта обращаются с помощью соответствующих методов. В рассматриваемом примере со списком связываются три метода: **init** (инициализация), **insert** (вставка в список нового элемента), **trav** (обход списка). Схематично объект **list** можно изобразить так, как представлено на рис. 2.

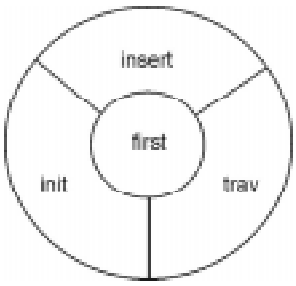


Рис. 2. Объект «Линейный цепной список»

Естественно, что все используемые методы должны быть описаны. При этом перед именем метода через точку записывается имя объекта, к которому относится данный метод. В листинге 2 представлено описание метода инициализации списка.

В листинге 3 описание метода добавления элемента в список.

Описание третьего метода (обход или итерация списка) приведено в листинге 4. Методу передается параметр процедурного типа, определяющий действие, которое осуществляется с каждым элементом списка.

При определении методов работы с объектом необходимо иметь доступ к структуре объектов. Знанием структуры

...объект "линейный цепной список"...

Листинг 2. Инициализация списка

```
{Инициализация списка}
procedure list.init;
  begin first := nil end;
```

Листинг 3. Вставка элемента в список

```
procedure list.insert (n: elem_list);
  var q: pelem;
begin new(q);
       if q = nil then begin writeln('Нехватка памяти'); exit
end;
      q^.info:= n;
      q^.next := first;
      first := q
end;
```

объекта **list** мы воспользовались, описывая методы **init**, **insert**, **trav**.

РАБОТА С ОБЪЕКТОМ С ПОМОЩЬЮ МЕТОДОВ

После того как методы работы с объектом определены, знание структуры объектов не является необходимым. Работу с объектом **list** можно организовать, используя лишь описанные методы.

Приведем программу работы с только что описанными объектами типа **list**. В программе при переходе от типа к переменным появится так называемый экземпляр объекта. Описание **var l1, l2: list** приведет к появлению в программе двух экземпляров **l1** и **l2** объектов типа **list**. В программе будем использовать процедуру **out**, которая помещает значение своего параметра в стандартный файл вывода и которая будет использоваться в дальнейшем при обходе списка. Раздел операторов программы приведен в листинге 5.

Вызов **l1.init** означает применение метода **init** к экземпляру объекта **l1** и выполняет действия по инициализации списка. При выполнении **l1.insert(5)** в список будет добавлен элемент с информационным полем 5. После выполнения четырех вызовов

Листинг 4. Обход или итерация списка

```
procedure list.trav(p: action);
  var  q: pelem;
begin q := first;
      while q <> nil do
         begin p(q^.info);
               q:= q^.next
         end
 end
```

```
l1.insert(5);  l1.insert(15);
l1.insert(9); l1.insert(4)
```

будет построен список из четырех элементов. После применения к экземпляру объекта метода **trav** с параметром **out** в выходной файл поступят значения **4 9 15 5**, учитывая тот факт, что элементы добавляются в начало списка.

Листинг 5. Пример использования объекта «Линейный цепной список»

```
{раздел операторов программы}
begin  l1.init; l2.init;
       l1.insert(5); l1.insert(15); l1.insert(9); l1.insert(4);
       writeln('список после первого обхода');  l1.trav(out);
       writeln('Введите элементы списка');
       read(x);
       while x <> 0 do
         begin l2.insert(x); read(x) end;
       writeln('список после второго обхода'); l2.trav(out);
    end.
```

ОБЪЕКТЫ И МОДУЛИ

Технологию абстрактных типов данных, кроме объектно-ориентированного программирования, поддерживает аппарат модулей языка Turbo Pascal. Использование модулей яввляется одним из способов выделения независимых частей в программе. В системе Turbo Pascal в настоящее время стандартные процедуры и данные объединены в стандартные модули.

Пользователь может описывать и в дальнейшем использовать свои модули. Модуль состоит из четырех разделов:

– заголовок,
– интерфейсная часть,
– часть реализации,
– часть инициализации.

Все разделы модуля кроме инициализации являются обязательными. Заголовок модуля состоит из слова **Unit** и идентификатора, являющегося именем модуля. Модуль должен быть помещен в файл с именем модуля и расширением **.pas**.

Интерфейсная часть модуля начинается со слова **Interface**. Далее могут располагаться все те же разделы, что и при описании программы с той лишь разницей, что в интерфейсной части для процедур и функций указываются только заголовки. Через интерфейс осуществляется взаимодействие программ (модулей) с данным модулем.

Часть реализации начинается словом **Implementation** и содержит описание всех процедур и функций, заголовки которых встречаются в интерфейсной части. Она также может содержать локальные константы, типы, переменные, процедуры и функции.

Часть инициализации начинается словом **Begin** и содержит действия, которые необходимо выполнить при инициализации модуля. Заканчивается описание модуля словом **end** и точкой.

Как же взаимодействуют между собой модули и объекты? Естественно, по-види-

Листинг 6. Модуль, содержащий описание объекта

```
unit ou_list;
 interface
 type elem_list= integer; {элемент списка}
          pelem=^elem;
          elem= record info : elem_list; next : pelem  end;
          action= procedure (var z: elem_list);
          list = object
                   first : pelem;
                   procedure init;
                   procedure insert(n: elem_list);
                   procedure trav(p: action);
                  end;
  implementation
    {инициализация списка}
      procedure list.init;
      begin first:= nil end;
    {вставка элемента в список}
      procedure list.insert (n: elem_list);
     {обход списка}
     procedure list.trav(p: action);
       var  q: pelem;
     begin q := first;
           while q <> nil do
             begin p(q^.info); q := q^.next end
     end;
   end.
```

мому, каждый объект описывать в отдельном модуле. Причем, в интерфейсную часть модуля помещается описание объекта (набор свойств и методов), а в часть реализации описание этих методов. Поместим в файл **ou_list.pas** описание модуля **ou_list**, который содержит описание объекта **list**. В интерфейсной части модуля описание вспомогательных типов **elem_list**, **elem**, **action** и объекта **list**. В части реализации приведено описание рассмотренных методов, часть инициализации отсутствует. Листинг 6 содержит описание модуля, в котором располагается описание объекта.

Программа, хранящаяся в листинге 7, использует объект **list**, описанный в модуле **ou_list**. Используемые модули в программе задаются с помощью предложения **uses ou_list**.

В программе используются два экземпляра **l1** и **l2** объекта **list**, которые сначала инициализируются, затем дополняются элементами и к построенным объектам применяется процедура обхода. Программа с используемым модулем имеет вид, как в листинге 7. Экземпляр объекта **l2** формировался по последовательности ненулевых чисел. Число ноль считалось признаком конца последовательности.

СВОЙСТВА ОБЪЕКТОВ

Между двумя объектами можно установить наследственную связь. Один из объектов объявляется потомком другого, который, в свою очередь, может быть предком для следующего объекта.

Потомок наследует все атрибуты своего предка и его методы, поэтому описывать эти методы второй раз в объекте – потомке нет необходимости. Объект – потомок может содержать кроме методов объекта предка свои собственные методы, которые, естественно, надо описывать.

НАСЛЕДОВАНИЯ ОБЪЕКТОВ

Например, объект «упорядоченный список» может быть потомком объекта «список» и наследовать его методы. Кро-

Потомок наследует все атрибуты своего предка...

Листинг 7. Программа, использующая модуль с объектом «Линейный цепной список»

```
program P5;
 uses ou_list;
var l1,l2: list; x: elem_list;
{$F+}
procedure out (var z: elem_list);
 begin write (z,' ') end;
{$F-}
{раздел операторов программы}
begin l1.init; l2.init;
      l1.insert(5); l1.insert(15); l1.insert(9); l1.insert(4);
      writeln('список после первого обхода'); l1.trav(out);
      writeln('Введите элементы списка');
      read(x);
      while x <> 0 do  begin  l2.insert(x); read(x) end;
      writeln('список после второго обхода'); l2.trav(out);
end.
```

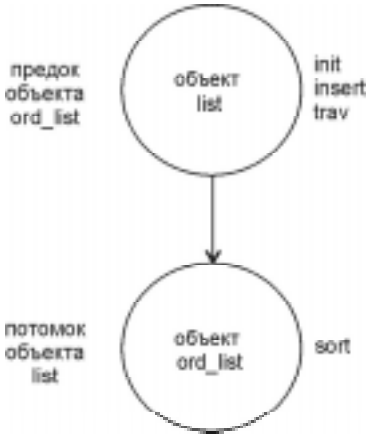


Рис. 3. Наследование объектов

ме того, объект «упорядоченный список» может содержать свои собственные методы, например метод сортировки списка. Пусть объект **ord_list** («упорядоченный список») описан как потомок объекта **list** («список»). В этом случае объект **list** является предком объекта **ord_list** и при описании объекта **ord_list** он указывается в скобках после слова **object**. Объект **ord_list** можно описать так, как в листинге 8.

Листинг 8. Описание потомка объекта с собственным методом

```
type ord_list = object (list)
                  procedure sort
              end
```

Обект **ord_list** наследует атрибуты (**first**) и методы (**init**, **insert**, **trav**) обьекта **list** и имеет свой собственный метод **sort**. На рис. 3 приведена схема связи объектов. Для каждого из объектов указаны описанные методы.

Обсуждая взаимодействие объектов и модулей, мы уже говорили о том, что разумно, по-видимому, каждый объект описывать в отдельном модуле. Модуль **ou_lord**, содержащий описание потомка объекта «линейный список», представлен в листинге 9.

Листинг 9. Описание объекта-потомка в отдельном модуле

```
unit ou_lord;
interface
 uses ou_list; {Модуль, содержащий описание предка объекта}
 type   ord_list= object (list)
                    procedure sort;
                  end;
implementation
    procedure ord_list.sort;
     var p,q: pelem; r: elem_list;
    begin
      p:=first;
      while p <> nil do
             begin q:= p^.next;
                   while q <> nil do
                     begin if p^.info > q^.info
                           then begin r:=  p^.info;
                                    p^.info := q^.info;
                                    q^.info:=r
                                end;
                                q:=  q^.next;
                     end;
                     p:= p^.next
                end
       end;
end.
```

Листинг 10. Программа, использующая два объекта, заданных в модулях

```
program P10;
 uses ou_list, ou_lord;
    var   l1: list;
          l2: ord_list;
          x: elem_list;
    {$F+}
    procedure out (var z: elem_list);
    begin write (z,' ') end;
    {$F-}
    {раздел операторов программы}
    begin l1.init; l2.init; writeln;
          writeln('введите элементы списка');
          read(x);
          while x<> 0 do
            begin  l1.insert(x); l2.insert(x); read(x)  end; {1}
          writeln;   writeln('вывод неупорядоченного списка');
          l1.trav(out);  writeln; l2.sort;
          writeln('вывод упорядоченного списка'); l2.trav(out);
     end.
```

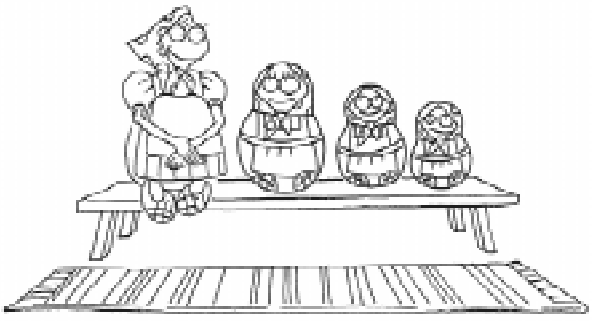
Программа с именем **P10**, которая использует экземпляр объекта-предка и экземпляр объекта-потомка, может быть описана так, как в листинге 10.

В точке {1} раздела операторов программы построены два списка, элементы которых неупорядочены, так как при добавлении элемента в список использовался метод **insert** объекта **list**, осуществляющий вставку элемента в начало списка. Оба списка состоят из одинаковых элементов. После применения к экземпляру объекта **l2** метода **sort**, осуществляющего сортировку списка, элементы в списке будут упорядочены по возрастанию информационных полей. При применении метода **trav** к экземпляру объекта **l2** элементы списка будут упорядочены и затем с помощью метода **trav** выведены в стандартный файл вывода. К экземпляру объекта **l2** в программе трижды применялись методы предка, которые он наследовал, и один раз собственный метод.

Заметим, что использование конструкции **l1.sort** ошибочно, так как экземпляр объекта **l1** методом **sort** не обладает.

ПОЛИМОРФИЗМ

Обратимся к третьему свойству объекта – свойству *полиморфизма*. По-прежнему рассматриваются два объекта – список и упорядоченный список, причем последний объект – потомок объекта список. Напомним, что метод добавления элемента в список, описанный в объекте **list**, добавляет элемент в начало списка. Если же мы рассматриваем упорядоченный список, то удобно добавлять элемент в упорядоченный список так, чтобы не нарушалась упорядоченность. Другими словами, нам бы хотелось в объекте – потомке изменить один метод, а именно **insert**,



...удобно добавлять элемент в упорядоченный список так, чтобы не нарушалась упорядоченность...

Листинг 11. Описание модуля с объектом, обладающим методом вставки элемента в упорядоченный список

```
unit ou_sort;
interface
 uses ou_list;
 type  sort_list= object (list)
                    procedure insert(n:elem_list);
                  end;
implementation
    procedure sort_list.insert(n: elem_list);
     var q: pelem;  cur: ^pelem;
    begin
      new(q);  if q = nil then begin writeln('нет памяти'); exit end;
       cur := @first;
       while cur^ <> nil do
              if cur^^.info >=n    then break
              else cur := @(cur^^.next);
        q^.info :=  n;
        q^.next := cur^;
        cur^ := q
     end;
  end.
```

но так, чтобы имя метода было бы тем же самым. Возможность использования в разных объектах методов с одинаковым именем называется полиморфизмом. Какой конкретно метод применить, зависит от объекта.

ПЕРЕОПРЕДЕЛЕНИЕ СТАТИЧЕСКИХ МЕТОДОВ

Все рассмотренные ранее методы являются статическими. В объекте-потомке статический метод предка можно переопределить. В рассматриваемом примере переопределяется метод вставки элемента в список.

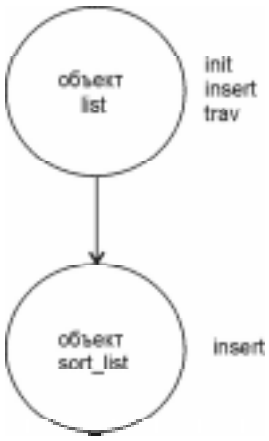


Рис. 4. Переопределение статического метода

Поясним сказанное примером. Пусть объект **list** описан, как и ранее, и хранится в модуле **ou_list**. Пусть в файле с именем **ou_sort** хранится модуль с описанием объекта **sort_list** (упорядоченный список). В листинге 11 приведено описание модуля.

На рис. 4 приведена схема связи двух объектов. В объекте потомке переопределен метод **insert**.

Напишем программу, которая использует экземпляр **l1** объекта **list** и экземпляр **l2** объекта **sort_list**. В теле цикла при выполнении **l1.insert(x)** элемент вставляется в начало списка, при выполнении **l2.insert(x)** элемент вставляется в упорядоченный список, не нарушая упорядоченности. Текст программы в листинге 12.

При определении потомка метод **insert** был переопределен. В общем случае при переопределении статического метода число и тип параметров могут быть различными для методов предка и потомка.

Листинг 12. Программа работы с объектом предком и потомком с переопределенным методом

```
program P12;
 uses ou_list, ou_sort;
    var  l1: list;  l2: sort_list;  x: elem_list;
    {$F+}
    procedure out (var z: elem_list);
    begin write (z,' ') end;
    {$F-}
    {раздел операторов программы}
    begin  l1.init; l2.init;  writeln;
      writeln('введите элементы списка');  read(x);
      while x <> 0 do
          begin   l1.insert(x);  l2.insert(x);  read(x) end; writeln;
      writeln('вывод неупорядоченного списка'); l1.trav(out); writeln;
      writeln('вывод упорядоченного списка');   l2.trav(out);
    end.
```

ЗАДАЧИ

1. Добавьте в рассмотренный объект `list` методы переворота списка удаления элемента с заданным информационным полем.

2. Создайте объект «уникальный список», все элементы которого различны, и упорядочены по возрастанию. Объект «уникальный список» должен быть потомком объекта «упорядоченный список».

3. Создайте объект «бинарное дерево», описав методы инициализации, построения бинарного дерева по линейной скобочной записи и обхода дерева.

4. Создайте объект «бинарное дерево поиска», который должен являться потомком объекта «бинарное дерево», добавьте собственный метод вставки элемента в упорядоченное дерево.

5. Создайте объект «уникальное дерево поиска», элементы которого различны и который должен являться потомком объекта «бинарное дерево поиска».

Дмитриева Марина Валерьевна, доцент кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета.