Порівняльний аналіз при викладенні навчального матеріалу як засіб інтелектуального розвитку учнів (на прикладі порівняння об'єктних моделей Turbo Pascal 7.0 і Delphi)

І.М. Лукаш

Проблема інтелектуального розвитку учнів в останнє десятиліття набуває все більшої актуальності. Суспільство вимагає від людини як високої компетентності в деякій професійній галузі, так і розвинутих інтелектуальних умінь, що є основою для подальшого самовдосконалення. Ті знання, які набуваються учнями в школі, в процесі подальшої навчальної і професійної діяльності повинні поглиблюватись і розширюватись завдяки сформованим інтелектуальним умінням виконувати розумові операції: аналіз, синтез, порівняння, класифікацію, узагальнення тощо. Тому одним з способів формування зазначених інтелектуальних умінь можна вважати моделювання подібної ситуації в процесі шкільного навчання, коли викладається новий матеріал, в якому є дуже багато схожого з попереднім, але це загальне представлене у більш розширеній і поглибленій формі. Засвоюючи такий матеріал, учні повинні будуть виконувати порівняння, виявляючи елементи подібності і відмінності, аналогію, поглиблений аналіз особливостей нового тощо.

З матеріалу шкільного курсу основ інформатики та обчислювальної техніки можна навести багато прикладів можливостей реалізації такого підходу. Це може бути перехід від вивчення навчальної алгоритмічної мови до вивчення реальної мови програмування, від роботи з простішим програмним забезпеченням (ПЗ) в курсі користувача до роботи з пакетами ПЗ професійного призначення тощо. З точки зору вирішення проблеми розвитку мислення учнів значну роль може відіграти використання об'єктно-орієнтовного підходу, вивченню якого в шкільній інформатиці ще не приділяється належної уваги. Існує кілька причин доцільності викладання основ об'єктно-орієнтовного підходу взагалі і в програмуванні (ООП) зокрема. Як і в повсякденному житті, так і при використанні ООП людина мислить однаковими категоріями – об'єктами зі своїми характерними ознаками і лінією поведінки [1]. Побудова окремого об'єкта, іншими словами створення абстракції, потребує від програміста виконання цілої низки розумових операцій: виділення суттєвих з точки зору галузі застосування рис об'єкта (аналіз); порівняння заданого об'єкта з іншими цього ж класу з метою визначення, які риси є загальними для класу, які є індивідуальною особливістю; побудова ієрархічної залежності об'єктів (класифікація) тощо. Досвід програмування показує, що програми, створенні на основі структурних компонентів об'єктів більш надійні у використанні і гнучкі при перебудові, ніж ті, які реалізовані засобами структурного програмування. Окрім цього, на ринку програмного продукту все більшої популярності набувають об'єктно-орієнтовні (ОО) середовища програмування, ОО бази даних, ОО експертні системи тощо. Зрозуміло, що знання з основ ООП будуть корисними як з

точки зору професійного їх застосування у майбутньому, так і засіб формування інтелектуальних умінь учнів сьогодні.

Цікавим і насиченим матеріалом з застосування ОО підходу в програмуванні є вивчення середовищ візуального програмування, зокрема середовища візуального програмування Delphi. Delphi є своєрідним продовженням, більш розвиненою формою поширеної у навчальному процесі в школах мови програмування Pascal. Багато подібного в реалізаціях ОО підходу існує в останній версії Object Pascal 7.0 (Turbo Pascal 7.0) і Delphi. В той же час ці засоби реалізації ООП в Delphi різноманітніші і досконаліші. Тому вивчення Delphi на основі порівняльного аналізу з Turbo Pascal 7.0 буде сприяти не тільки кращому засвоєнню навчального матеріалу, а й формуванню умінь виконання відповідних інтелектуальних операцій.

Основою ООП є об'єктна модель. Основні принципи формування об'єктів, що використовувалися в Turbo Pascal 7.0 залишаються і в Delphi, але внесені відповідні зміни щодо синтаксису їх оголошення і використання. Якщо в оголошеннях старих об'єктних типів застосовувалося ключове слово object, нові об'єктні типи визначаються за допомогою слова class. В Delphi прийнято ім'я об'єктного типу починати з літери "T", а ім'я поля – з літери "F". На відміну від старої моделі, де можна було працювати як з динамічними так і статичними екземплярами, в новій об'єктній моделі програміст працює тільки з динамічними екземплярами класів, тобто з тими, для яких виділяється пам'ять в heap – області.

Розглянемо приклад описання динамічного об'єкта, що виводить на екран деяке текстове повідомлення, як в Turbo Pascal 7.0, так і в Delphi. Якщо в Turbo Pascal 7.0 заголовок структури динамічного об'єкта задається одним ідентифікатором, наприклад, DataOutput, а посилання на неї – за допомогою іншого ідентифікатора (DataOutputPrt), то в Delphi структура динамічного об'єкта іменується єдиним ідентифікатором (TDataOutput).

Оголошення динамічного об'єкта в Turbo Pascal 7.0:

```
Type
 DataOutputPrt=^DataOutput;
 DataOutput=object
    St:string;{Поле для збереження значення текстового рядка}
    Constructor Init(StInit:string);{Ініціалізація початкових значень екземпляра об'єкта}
    Procedure Show;Virtual; {Метод виведення на екран текстового рядка}
    Destructor Done;Virtual; {Звільнення пам'яті від екземпляра об'єкта}
    end;
```

Оголошення динамічного об'єкта в Delphi:

```
Type
  TDataOutput=class(TObject)
  FSt:string; {Поле для збереження значення текстового рядка}
  Constructor Create(StInit:string); {Ініціалізація початкових значень екземпляра об'єкта}
  Function GetSt:string;{Метод здійснення доступу до поля FSt}
  Destructor Destroy;override; {Звільнення пам'яті від екземпляра об'єкта}
  end;
```

Згідно з різними оголошеннями динамічних об’єктів змінна екземпляра об’єкта в Turbo Pascal 7.0 описується як вказівник на структуру об’єкта (Var EkzData:DataOutputPrt), а в Delphi - як змінна типу заданого класу (Var EkzData:TDataOutput).

Відмінності в програмних кодах конструктора і деструктора різних об’єктних моделей обумовлені особливостями відповідних середовищ програмування. Якщо об’єкт вважається коренем деякої ієрархічної структури об’єктів, то в Turbo Pascal 7.0 конструктор містить в собі тільки оператори присвоювання початкових значень полям екземпляра об’єкта:

```
Constructor DataOutput.Init(StInit:string);
 begin
 St:=Stinit;{Присвоювання початкового значення полю St}
 end;
```

В Delphi при описанні аналогічного об’єкта вважається, що він завжди буде нащадком від якогось стандартного, заданого в середовищі програмування класу, наприклад, Tobject. Таким чином перевизначення конструктора для створюваного об’єкта, окрім ініціалізації, передбачає в програмному коді звернення до конструктора батьківського класу:

```
Constructor TDataOutput.Create(StInit: string);
begin
inherited Create; {Виклик конструктора батьківського класу}
FSt:=StInit; { Присвоювання початкового значення полю FSt}
end;
```

В Delphi як і в Turbo Pascal 7.0 зберігається звернення до методів батьківського класу через зарезервоване слово Inherited [4]. У випадку, коли при створенні екземпляра об’єкта в Delphi не передбачається додаткових операцій, то конструктор Create можна не перевизначати, тобто не включати до оголошення типу об’єкта. Автоматично буде використаний конструктор батьківського класу.

Аналогічні особливості програмування стосуються і деструктора. В Turbo Pascal 7.0 деструктор обов’язково задається в структурі об’єкта, він звільняє пам’ять від екземпляра об’єкта навіть тоді, коли він пустий:

```
Destructor DataOutput.Done;
Begin
end;
```

В Delphi деструктор, що перевизначається, повинен містити звернення до деструктора батьківського класу:

```
Destructor TDataOutput.Destroy;
begin
inherited Destroy {Звернення до деструктора батьківського класу}
end;
```

Як і конструктор, в Delphi деструктор можна теж не перевизначати. Для ліквідації екземпляра об’єкта, що не містить в собі в якості полів інших об’єктів, допустимо використання методу Free, наприклад: EkzData.Free.

Написання програмного коду інших методів різних об'єктних моделей залежить від принципів роботи майбутньої програми. Наприклад, в Turbo Pascal 7.0 виведення текстового рядка будемо здійснювати прямо на екран, починаючи з позиції (10,10). Тому програмний код методу Show буде містити відповідні оператори:

```
Procedure DataOutput.Show;
 begin
 GotoXY(10,10);writeln(St);
 end;
```

В Delphi передбачимо виведення текстового рядка, наприклад, в компоненті Edit. Для цього відповідним методом необхідно реалізувати непрямий доступ до поля екземпляра об'єкта, що зберігає значення заданого текстового рядка:

```
Function TDataOutput.GetSt:string;
begin
GetSt:=FSt
end;
```

Удосконалення в оголошенні нової об'єктної моделі спричинили зміни в синтаксисі звернення до полів і методів об'єктів. Раніше (в Turbo Pascal 7.0) для роботи з динамічними екземплярами об'єктів, ініціалізованими завдяки сполученню використання звернення до конструктора і функції New, програміст повинен був застосовувати звернення "за адресою" (^):

```
Begin {Приклад програмного коду головної програми}
New(EkzData);{Відведення в пам'яті місця для екземпляра об'єкта}
EkzData^.Init('ООП');{або  New(EkzData,Init('ООП'));} {Ініціалізація екземпляра об'єкта}
EkzData^.Show;{Виведення текстового рядка на екран}
Dispose(EkzData,Done);{Вилучення екземпляра об'єкта з пам'яті}
end.
```

В Delphi такий доступ припускається автоматично [3,4,5]. Наступний приклад програмного коду ілюструє використання нової об'єктної моделі:

```
procedure TForm1.Button1Click(Sender:TObject);
begin {Дії відбуваються після натиснення відповідної кнопки, передбаченої в проекті}
EkzData:=TDataOutput.Create('OOP');{Створення екземпляра об'єкта}
Edit1.Text:=EkzData.GetSt;{Виведення значення текстового рядка в компоненті Edit}
EkzData.Destroy;{Вилучення з пам'яті екземпляра об'єкта}
end;
```

Можливість використання як тип поля тип об'єкт – є новою особливістю Delphi порівняно з Turbo Pascal 7.0 [5]. Наприклад, клас TData повинен містити як поля класи TNumber і TMonth. У зв'язку з цим класи TNumber і TMonth оголошуються окремо, а їх заголовки включаються до структури класу TData:

```
type
 TNumber=class(TObject)
 FN:string;
 end;{Оголошення об'єкта TNumber }
 TMonth=class(TObject)
 FS:string;
 end;{ Оголошення об'єкта TMonth }
 TData=class(TObject)
 FNumber:TNumber;
 FMonth:TMonth;
 Constructor Create;
 Destructor Destroy; override;
 end;{ Оголошення об'єкта TData }
```

При цьому, як показує фрагмент наступного програмного коду, в конструкторі об’єкта TData необхідно створити екземпляри об’єктів TNumber і TMonth для відповідних полів, а в деструкторі відповідно звільнити від них пам’ять:

```
Constructor TData.Create;
 Begin {Програмний код конструктора}
 Inherited Create;
 FNumber:=TNumber.Create;
 FMonth:=TMonth.Create;
end;
```

```
Destructor TData.Destroy;
Begin { Програмний код деструктора}
FNumber.Free;
FMonth.Free;
Inherited Destroy;
End;
```

Удосконалення об’єктної моделі торкнулися і загальної її структури. Крім полів і методів в Delphi введений новий елемент – властивості. Якщо поля розглядаються як атрибути об’єкта, де зберігаються відповідні данні, що можуть бути зчитані або змінені в процесі роботи програми, то властивості набувають деяких значень після виконання спеціальних дій, які стосуються зчитування або запису даних полів, а також їх модифікації за допомогою відповідних методів. Можна вважати, що властивості є одним з засобів запобігання прямого звернення до полів об’єкта. При створенні візуальних компонентів властивості, описані в структурі об’єкта після слова published, висвітлюються в інспекторі об’єктів, тим самим передбачається можливість встановлення відповідних значень до початку роботи програми. Різні прийоми оформлення оголошення властивостей дозволяють реалізовувати навчальне завдання різними способами, оцінювати переваги того чи іншого прийому, що сприяє розвиткові гнучкості, критичності мислення учнів.

Один з можливих форматів властивостей має структуру:

Property <ім’я>:<тип> read < ім’я1> write < ім’я2> default <значення> , де

<ім’я> - ім’я властивості;

<тип> - тип властивості, повинен збігатися з типом відповідного поля або результатом метода - функції;

< ім’я1> - ім’я поля, з якого буде зчитане значення, або метода, дія якого призведе до зчитування даних з цього поля;

< ім’я2> - ім’я методу, який встановлює нове значення відповідному полю в процесі роботи програми;

<значення> - значення, яке присвоюється властивості за замовченням.

Зрозуміло, що властивості необов’язково містити всі зазначені операції. Властивість може тільки зчитувати або змінювати значення поля. Поряд з командою default може використовуватися, наприклад, команда stored, поряд з якою повинно стояти одне з слів false або true . Якщо поточне значення властивості не співпадає із значенням за замовченням і stored встановлене як false, то це поточне значення властивості не зберігається.

Розглянемо приклад створення власного об’єкта – прямокутника з фіксованими параметрами довжини і ширини, який повинен переміщуватися на поверхні форми після натиснення відповідної кнопки. Для імітації

переміщення об’єкта будемо спочатку малювати об’єкт кольором малюнка, потім кольором фону, і в новій позиції знов кольором малюнка. Тобто в процесі роботи програми передбачається зміна значень координат і кольору. Тому до структури об’єкта TRectangle введемо відповідні поля (FX, FY, FColor), властивості (CoordX, CoordY, ColorR) і методи (SetX, SetY, SetColor), які будуть підтримувати роботу програми.

. . . . . .

```
TRectangle=class(TObject) {Оголошення класу TRectangle}
 private
 FX:integer;{Поле координати X верхнього лівого кута прямокутника}
 FY:integer;{ Поле координати Y верхнього лівого кута прямокутника }
 FColor:TColor;{Поле кольору малювання прямокутника}
 procedure SetX(NewX:integer);{Метод доступу до поля FX }
 procedure SetY(NewY:integer);{ Метод доступу до поля FY }
 procedure SetColor(NewColor:TColor);{ Метод доступу до поля FColor }
 public
 Constructor Create(X,Y:integer;Col:TColor);{Конструктор}
 Destructor Destroy;override;{Деструктор}
 Procedure Show;{Метод малювання прямокутника}
 property CoordX:integer read FX write SetX default 10;{Властивості для задання і зміни}
 property CoordY:integer read FY write SetY default 10;{значень відповідних полів}
 property ColorR:TColor read FColor write SetColor default clRed;
 end;
var
 Form1: TForm1;{Прямокутник буде малюватися на канві форми Form1}
  Rec:TRectangle;{Описання екземпляра об’єкта}
```

Реалізація описаних об’єкті методів матиме вигляд:

```
implementation
{$R *.DFM}
 procedure TRectangle.SetX(NewX:integer);
 begin  FX:=NewX;  end;
 procedure TRectangle.SetY(NewY:integer);
 begin  FY:=NewY;  end;
 procedure TRectangle.SetColor(NewColor:TColor);
 begin  FColor:=NewColor;  end;
 Constructor TRectangle.Create(X,Y:integer;Col:TColor);
 Begin  inherited Create;  FX:=X; FY:=Y; FColor:=Col;  end;
 Destructor TRectangle.Destroy;
 Begin  inherited Destroy;  end;
```

При малюванні прямокутника замість прямого звернення до полів використовуємо звернення до відповідних властивостей.

```
 Procedure TRectangle.Show;
 begin
 Form1.Canvas.Pen.Color:=ColorR;
 Form1.Canvas.MoveTo(CoordX,CoordY);
 Form1.Canvas.LineTo(CoordX+50,CoordY); {30,50 – сторони прямокутника}
 Form1.Canvas.LineTo(CoordX+50,CoordY+30);
 Form1.Canvas.LineTo(CoordX,CoordY+30);
 Form1.Canvas.LineTo(CoordX,CoordY);
```

```
 end;
```

Виконувана частина проекту матиме вигляд:

```
procedure TForm1.Button1Click(Sender: TObject);
var i,j:integer;
begin
 Rec:=TRectangle.Create(10,10,clRed);{Створюємо екземпляр об'єкта TRectangle }
  For i:=1 to 20 do {20 разів будемо переміщувати об'єкт по канві форми}
  begin
  Rec.Show;{намалюємо прямокутник встановленим раніше червоним кольором}
  Rec.SetColor(clBtnFace);{Змінимо колір малюнку на колір фону}
  Rec.Show;{намалюємо прямокутник кольором фону}
 Rec.SetX(Rec.CoordX+10);Rec.SetY(Rec.CoordY);{Встановимо значення нових координат}
 Rec.SetColor(clRed);{Змінимо колір малюнка}
  end;
Rec.Destroy;{Вилучення екземпляра об'єкта з пам'яті}
end;
```

.. . .

Далі розглянемо наступні способи удосконалення описання структури об'єкта. Якщо група полів об'єкта має однаковий тип, то її можна записати як єдине поле типу масив (FCoord:array [0..1] of integer) і здійснювати доступ до окремих елементів через вказання їх індексів (FCoord[0], FCoord[1]) за допомогою методу-функції: function GetCoord(Ind:integer):integer;

```
TRectangle=class(TObject)
  private
  FCoord:array [0..1] of integer;
  FColor:TColor;
  function GetCoord(Ind:integer):integer;
  procedure SetCoord(Ind,NewValue:integer);
  procedure SetColor(NewColor:TColor);
  public
  Constructor Create(X,Y:integer;Col:TColor);
  Destructor Destroy;override;
  Procedure Show;
  property CoordX:integer index 0 read GetCoord write SetCoord default 10;
  property CoordY:integer index 1 read GetCoord write SetCoord default 10;
  property ColorR:TColor read FColor write SetColor default clRed;
  end;
```

В методах GetCoord, SetCoord розподіл доступу до елементів поля FCoord відбувається в залежності від значення вхідного параметра індексу Ind цілого типу. Наприклад:

```
procedure
TRectangle.SetCoord(Ind,NewValue:integer);
  begin
  case Ind of
  0: FCoord[0]:=NewValue;
  1: FCoord[1]:=NewValue;
  end;
  end;
```

```
Function
TRectangle.GetCoord(Ind:integer):integer;
  begin
  case Ind of
  0: GetCoord:=FCoord[0];
  1: GetCoord:=FCoord[1];
  end;
 end;
```

Цей же індекс указується і в виконавчій частині проекту при застосуванні заданих методів: Rec.SetCoord(0,Rec.CoordX+10);Rec.SetCoord(1,Rec.CoordY); .

Аналогічно змінюється і оголошення відповідних властивостей об'єкта. Перед специфікаціями зчитування і запису даних слово index і ціла константа поряд вказують, з яким елементом поля FCoord пов'язується задана властивість. Наприклад, рядок програмного коду в описанні структури об'єкта – прямокутника (property CoordX:integer index 0 read GetCoord write SetCoord) означає, що значення за допомогою методу GetCoord буде зчитано з першого елемента поля FCoord.

При описанні структури об'єкта існує можливість запису у вигляді масиву не тільки полів однакового типу, але й властивостей, що дозволяє значно скоротити кількість рядків програмного коду. Тобто замість двох рядків в описанні прямокутника з попереднього прикладу:

```
property CoordX:integer index 0 read GetCoord write SetCoord;
property CoordY:integer index 1 read GetCoord write SetCoord;
```

можна записати один:

```
property Coordinates[index:integer]:integer read GetCoord write SetCoord; .
```

При цьому слід зазначити, що index не повинен бути обов'язково цілого типу, індексів може бути кілька.

У зв'язку зі зміною в оголошенні властивостей, змінюється і запис програмного коду звернення до них в методі TRectangle.Show:

```
Form1.Canvas.LineTo(Coordinates[0]+50,Coordinates[1]);
```

та у виконавчій частині проекту:

```
Rec.SetCoord(0,Rec.Coordinates[0]+10);Rec.SetCoord(1,Rec.Coordinates[1]); .
```

Таким чином, знання наведених прийомів оголошення і використання властивостей дозволятимуть учням більш раціонально будувати свої програми. Як і кожний створюваний людиною в реальному житті проект, розглянутий приклад проекту, що реалізує переміщення об'єкта прямокутника на поверхні форми, містить можливості для його подальшого удосконалення. Це стосується, наприклад, введення до структури об'єкта параметрів: висоти, ширини прямокутника, кроку переміщення тощо; передбачення їх змін в процесі роботи програми. Завдання, пов'язані з виконанням змін в наданому проекті, вимагаючи самостійної роботи учнів, сприяють не тільки кращому засвоєнню матеріалу, але й розвитку, розкриттю творчої особистості.

Від способу реалізації об'єктної моделі залежить характер взаємодії об'єктів різних класів. Для того, щоб об'єкт одного класу міг скористатися методом іншого класу в Turbo Pascal 7.0 необхідно було обов'язково створювати екземпляри об'єктів зазначених класів і здійснювати доступ до необхідного метода через відповідний екземпляр об'єкта [3,4]. Наприклад, оголосимо два типи об'єктів cl та cl1:

```
Type
  cl=object
    function xxx:string;
    end;

  cl1=object
    function yyy:string;
    end;
```

Метод xxx типу об’єкта cl передбачає присвоювання імені функції деякого текстового рядка. В методі yyy типу об’єкта здійснюється звернення до методу xxx через екземпляр с об’єкта cl.

```
var   c:cl; c1:cl1;{задання екземплярів об’єктів}
function cl.xxx:string;{Описання методу xxx об’єктного типу cl}
begin
xxx:='OOP';
end;
function cl1.yyy:string;{ Описання методу yyy об’єктного типу cl1 }
begin
yyy:=c.xxx;{Звернення до методу xxx через екземпляр c }
end;
Begin {головна програма}
writeln(c1.yyy);
end.
```

Результатом виконання головної програми буде виведення на екран текстового рядка 'OOP'.

Введення в об’єктній моделі Delphi нового типу методів – методів класів (class function або class procedure) виключає в подібній ситуації необхідність створення екземпляра об’єкта відповідного класу, доступ до методу якого здійснюється тільки через вказування його імені. Методи, оголошені за допомогою команди class, задають поведінку об’єктного типу таким чином, ніби тип є деяким параметром, що не пов’язаний з екземпляром того чи іншого об’єкта. Екземпляра об’єкта взагалі може не бути. Тому в методах класів не можна використовувати як змінні поля і властивості об’єкта [5]. У зв’язку з тим, що об’єкт c12 класу cl1 (var c12:cl1;) звертається до метода xxx іншого класу cl, метод xxx слід описати як class function.

```
Type
cl=class(TObject)                    cl1=class(TObject)
   class function xxx:string;           function yyy:string;
  end;                                  end;
```

Реалізація методів класів об’єктів:

```
class function cl.xxx:string;        function cl1.yyy:string;
begin                                begin
xxx:='OOP';                          yyy:=cl.xxx;{звернення до методу за його іменем}
end;                                 end;
```

За результатом роботи виконавчої частини програми в об’єкті Edit1 буде виведений той же самий текстовий рядок 'OOP'.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
c12:=cl1.create;
Edit1.Text:=c12.yyy;
end;
```

Окрім команди class, до в Delphi методів можуть застосовуватися також і інші команди (Virtual, Dynamic, Message, Abstract, Override). Характер дії команд, Virtual, Dynamic, що призначені для реалізації пізнього зв’язування, однаковий

для Delphi та Turbo Pascal 7.0. Нові, введенні в Delphi команди, виконують власні спеціальні дії. Команда Message означає, що виклик метода визначається механізмом обробки повідомлень. Слово Abstract вказує на те, що робота методу в даному класі не має сенсу, але метод може бути перевизначений і використаний класами - нащадками. Команда Override використовується для перевизначення віртуальних та динамічних методів [5].

Порівняно з Turbo Pascal 7.0, де для означення зони видимості елементів об'єкта (полів, методів, властивостей) вже використовуються команди private (елементи об'єкта досяжні тільки в модулі, в якому він оголошений) та public (елементи об'єкта досяжні в тих модулях, в яких видний сам об'єкт) [3], в Delphi наряду з ними використовуються додаткові команди protected та published. Команда protected порівняно з private послаблює захист елементів об'єкта. Зовні модуля, де визначений клас, його елементи, розміщені в зоні protected, досяжні тільки в класах – нащадках. Команда published навпаки підсилює відкритість зони видимості елементів об'єкта, надаючи можливість доступу до них через зовнішні програми, наприклад, для властивостей візуальних компонентів - через інспектор об'єктів [5].

Розглянуті особливості описання об'єктних моделей в Delphi є основою для створення власних візуальних компонентів, які відповідали б творчим задумам особистості, сприяли її інтелектуальному розвитку.

Література

1. Агафонов В.Н. Объектно-ориентированное программирование и абстрактные типы данных // Программирование. -№6, 1990. – С.27-32.
2. Бадд Т. Объектно-ориентированное программирование в действии / Перев. с англ. – СПб.: Питер, 1997. – 464 с.
3. Марченко А.И. Программирование в среде Borland Pascal 7.0. – К.: ТОО "ВЕК", "ЮНИОР", 1996. – 480 с.
4. Епашников А.М., Епашников В.А. Программирование в среде TURBO Pascal 7.0. – М.: Диалог – МИФИ, 1997.- 288с.
5. Миллер Т., Пауэл Д. Использование Delphi 3.- К.: Диалектика,1997.-768 с.