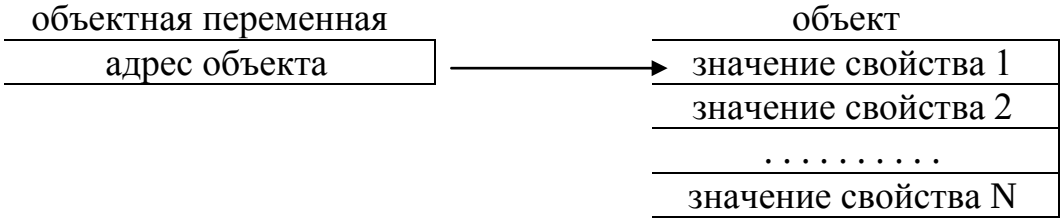
3. Создание и использование объектов. Объектные переменные

Для использования стандартных или собственных классов прежде всего необходимо объявить одну или несколько **объектных переменных.** Для каждой переменной вводится **уникальное имя** и указывается **принадлежность** к одному из известных **классов**. Это позволяет называть объектные переменные **переменными классового типа**.

В языках Java, C#, Delphi/Free Pascal объектные переменные являются **неявными, скрытыми указателями** на размещаемые в памяти объекты. Как известно, значениями указательных переменных являются адреса областей памяти. Техника работы с адресными переменными использовалась и в обычных языках (Си, Паскаль), а для объектных программ она является **основной.** В отличие от языка C++, где с адресными переменными можно делать практически все что угодно, в более поздних языках реализован только тот набор операций, который делает использование адресов более **безопасным.**

На наш взгляд, имеет смысл различать понятия «**объект**» и «**объектная переменная**». Под объектом будем понимать область памяти, динамически выделенную для хранения значений свойств объекта и некоторой дополнительной информации. Объектная переменная используется для **адресации** этой области и позволяет организовать доступ к находящейся в ней информации.



Адресный характер объектных переменных позволяет реализовать много интересных возможностей, прежде всего связанных с принципом **полиморфизма**, но в то же время увеличивает вероятность возникновения

ошибок времени выполнения и поэтому требует очень тщательной отладки программы.

Необходимо понимать, что **объявление** объектной переменной **НЕ означает** установку ее значения в некоторый адрес. Компилятор выделяет память только для самой объектной переменной, но НЕ для адресуемого с ее помощью объекта. **Установка адресного значения объектной переменной происходит только при создании объекта!** А для этого требуется вызов конструктора класса. Только после этого объектная переменная может использоваться для вызова методов адресуемого ею объекта. Этот вызов реализуется с использованием имени объектной переменной и имени метода.

Интересной (и перспективной, как будет ясно из дальнейшего) возможностью является объединение отдельных объектных переменных в единую структуру – массив или динамический список. Например, массив объектных переменных содержит указатели на размещенные в разных областях памяти объекты и позволяет обрабатывать набор объектов с помощью циклов.

Приведем примеры объявления и использования объектных переменных.

3.1. Язык Delphi/Free Pascal

Шаг 1. Объявление объектной переменной (ОП)**:** указывается имя одной или нескольких переменных и через двоеточие – имя класса

var ИмяОП : ИмяКласса;

Пример объявления переменных для управления объектами-окружностями (класс TCircle):

var MyCirc1, MyCirc2 : TCircle;

Шаг 2. Создание объекта с помощью конструктора:

Реализуется в виде оператора присваивания, в левой части которого указывается объявленная ранее объектная переменная, а в правой – имя класса и через точку – имя конструктора

ИмяОП := ИмяКласса.Конструктор(параметры);

Пример: создание двух объектов-окружностей – одна с параметрами по умолчанию, другая – с заданными в конструкторе параметрами:

```
MyCirc1 :=  TCircle.Create;
MyCirc2 :=  TCircle.Create (200, 100, 50);
```

Шаг 3. Вызов методов объекта:

ИмяОП.ИмяМетода(параметры);

Пример для объектов-окружностей

```
MyCirc1.Show;  //  отображение окружности
MyCirc1.MoveTo(100, 100);  //  перемещение окружности
Length := 6.28*MyCirc.GetR;   //  подсчет длины окружности
```

Пример объявления и использования **массива** объектных переменных-указателей на окружности:

```
var   ArrOfCircs : array [1.. 100]  of  TCircle;
for  i := 1  to  100  do  ArrOfCircs[ i ] :=  TCircle.Create (параметры);
for  i := 1  to  100  do  ArrOfCircs[ i ].Show;  // отображение всех окр-тей
```

Пример для объектов класса **Студент**:

Шаг 1:

```
var   Stud : TStudent;        //  объявление объектной переменной
```

Шаг 2:

```
Stud := TStudent.Create('Козин');          // создание объекта-студента
```

Шаг 3:(консольный вариант)

```
Stud.AddOcenky(5);                          // добавление оценок
Stud.AddOcenky(3);
Stud.AddOcenky(2);
Stud.SetOcenky(3, 4);                  // изменение третьей оценки
```

```
Writeln (Stud.GetKol);                  // вывод числа оценок
Writeln (Stud.SredBall);                // вывод среднего балла
Writeln (Stud.GetOcenky(2));            // вывод второй оценки
Stud.SetFam ('Иванов');                 // изменение фамилии
```

Пример для **МАССИВА** объектов класса Студент:

```
var   Gruppa : array [1..20]  of  TStudent;  // объявление массива ссылок
Gruppa[1] := TStudent.Create('Иванов');   // создание объектов-студентов
Gruppa[2] := TStudent.Create('Петров');
. . . . . .
Gruppa[ i ].AddOcenky(5);      // добавление оценок студенту  i
. . . . . .
for   i := 1  to  20   do
    Writeln (Gruppa[ i ].SredBall);    // вывод среднего балла всех студентов
Writeln (Gruppa[ i ].GetKol);              // вывод числа оценок студента  i
Writeln (Gruppa[ i ].GetOcenky( j ));      // вывод оценки j студента i
```

3.2. Языки Java и C#

Шаг 1. Объявление: указывается имя класса и через пробел(ы) – имя одной или нескольких переменных

ИмяКласса ИмяОП;

Пример для класса окружностей:

```
Circle   MyCirc1,  MyCirc2;
```

Шаг 2. Создание с помощью конструктора: реализуется в виде оператора присваивания, слева – имя объектной переменной, справа – специальное ключевое слово **new** и затем – имя конструктора (возможно - с параметрами), совпадающее с именем класса

ИмяОП = new Конструктор( );

Пример для класса окружностей:

MyCirc1 = **new** Circle( ); // вызов конструктора без параметров

MyCirc2 = **new** Circle(200, 100, 50); // конструктор с параметрами

Оба языка позволяют шаги 1 и 2 **объединить вместе**, т.е. в одной конструкции выполнить объявление объектной переменной и ее инициализацию (создание объекта):

ИмяКласса ИмяОП = new Конструктор( );

Circle MyCirc = **new** Circle(100, 100, 100);

Шаг 3. Вызов методов

ИмяОП.ИмяМетода(параметры);

Пример для объектов-окружностей

MyCirc1.Show( ); // отображение окружности

MyCirc1.MoveTo(100, 100); // перемещение окружности

Length = 6.28*MyCirc.GetR( ); // подсчет длины окружности

Пример объявления и использования **массива** объектных переменных-указателей на окружности **(важно:** в этих языках **массивы** сами являются **объектами** и поэтому должны создаваться **динамически** с помощью операции **new**):

Circle[ ] Circs; // объявление массива

Circs = **new** Circle[100]; // создание массива на 100 указателей

Circs[i] = MyCirc1; // занесение в массив существующего объекта

Circs[i] = **new** Circle (100, 100, 50); // создание и занесение нового

Пример для объектов класса **Студент**:

Student Stud; // объявление объектной переменной

Stud = **new** Student('Козин'); // создание объекта-студента

Stud.AddOcenky(5); // добавление оценок

```
Stud.AddOcenky(3);
Stud.AddOcenky(2);
Stud.SetOcenky(3, 4);                  // изменение третьей оценки
Console.WriteLine (Stud.GetKol);       // вывод числа оценок
Console.WriteLine (Stud.SredBall);     // вывод среднего балла
Console.WriteLine (Stud.GetOcenky(2)); // вывод второй оценки
Stud.SetFam ('Иванов');                // изменение фамилии
```

Пример для **МАССИВА** объектов класса Студент:

Шаг 1:

```
Student [ ]   Gruppa;  // объявление массива ссылок
```

Шаг 2: создание объекта-массива и заполнение его ссылками

```
Gruppa = new  Student [20] ;
Gruppa[0] =  new   Student ('Иванов');   // создание объектов-студентов
Gruppa[1] =  new   Student ('Петров');
. . . . . .
```

Шаг 3:

```
Gruppa[ i ].AddOcenky(5);      // добавление оценок студенту  i
. . . . . .
Console.WriteLine (Gruppa[ i ].GetKol);   // вывод числа оценок студента  i
Console.WriteLine (Gruppa[ i ].GetOcenky( j ));  // вывод оценки j студента i
```

3.3. Язык C++

Работа с объектами в этом языке немного отличается от других языков. Наравне с динамически создаваемыми объектами можно использовать

статические объекты, для доступа к которым не требуется использовать указательные переменные.

Для статического создания объекта достаточно объявить одну или несколько переменных соответствующего классового типа:

```
Circle   Circ1, Circ2;
```

При обработке этой строки компилятор включает в создаваемый код обращение к конструктору без параметров, поэтому объекты создаются с заранее заданными значениями свойств. При необходимости вместо конструктора без параметров можно вызвать конструктор с параметрами, например, следующим образом:

```
Circle  Circ3 (50, 50, 20);
```

При обработке этой строки компилятор анализирует входные значения (их количество и тип) и включает код для вызова подходящего конструктора. После такого неявного создания объектов можно эти объекты использовать, обращаясь к их методам:

```
Circ1.Show ( );                // отобразить объект Circ1
Circ3.MoveTo (100, 100);   // переместить объект Circ3
int   r = Circ3.GetR ( );      //  запросить радиус объекта Circ3
```

Для динамического создания объектов, прежде всего, необходимо объявить ссылочную переменную:

```
Circle  *pCirc1;
```

При обработке этой строки компилятор выделяет память только для хранения адреса будущего объекта. После этого записывается строка для динамического создания объекта, причем для выделения памяти используется стандартная функция **new,** а для инициализации выделенной области явно вызывается один из конструкторов класса:

```
pCirc1 = new  Circle ( );
```

При необходимости объявление объектной переменной и вызов конструктора можно объединить в одной строке:

```
Circle  *pCirc1 = new  Circle ( );
```

Вызов методов динамических объектов выполняется следующим образом:

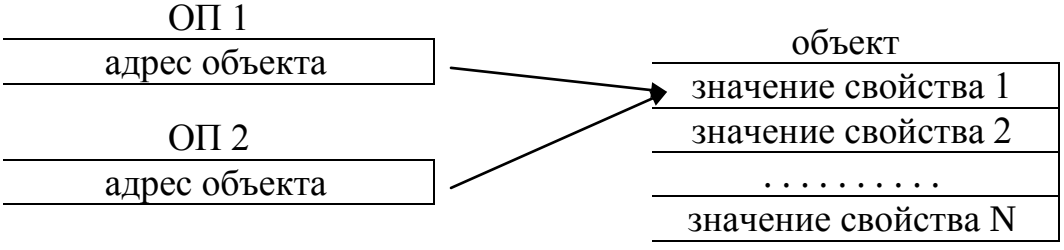
```
pCircle -> Show ( );    // отобразить
pCircle -> MoveTo (200, 200);  // переместить
int  Rad = pCirc2 -> GetRad ( );  // получить радиус
```

После рассмотрения особенностей объявления объектных переменных надо вернуться к **общим** вопросам использования таких переменных. Корректная работа с объектными переменными требует соблюдения ряда **правил**, связанных с адресным характером этих переменных. Для объектных переменных определены **две основные операции**:

- **присваивание** значения одной переменной другой
- **сравнение** значений двух переменных

Особенности использования операции присваивания:

- обе переменные должны относиться к **одному классу** (далее, при изучении полиморфизма, это требование будет **ослаблено**)
- переменная в правой части должна иметь **некоторое значение** (как известно, это относится к любым операциям присваивания)
- присваивание выполняется на уровне **адресных значений** объектных переменных, т.е. после выполнения операции **<u>ОБЕ переменные будут адресовать одну и ту же область памяти</u>**; никакого копирования объекта НЕ происходит!



- переменной **любого** классового типа можно присвоить **нулевое** (**пустое**) адресное значение, для чего используются

зарезервированные слова **null** (языки Java, C#, C++) или **nil** (Delphi/Free Pascal); после этого объектную переменную **НЕЛЬЗЯ** использовать для вызова методов

Примеры для графических объектов.

Delphi/Free Pascal

```
var   Circ1, Circ2 :  TCircle;
      Rect1, Rect2 :  TRectangle;
. . . . .
   Circ1  :=  TCircle.Create(200, 100, 50);
   Circ2  :=  Circ1;  // обе переменные адресуют одну и ту же окружность
   Circ1  :=  nil;  // а теперь Circ1 ничего не адресует
   Rect2 := Rect1; // ОШИБКА: переменная Rect1 не установлена!
   Rect1 := Circ2; // ОШИБКА: переменные РАЗНЫХ классовых типов!
   Rect2 :=  nil;  //  а так можно
   Rect2.Show;   //  ОШИБКА:  Rect2   НИЧЕГО не адресует!
```

Java и C#

```
   Circle   Circ1  =  new  Circle(200, 100, 50);
   Circle  Circ2 = Circ1; // обе переменные адресуют одну и ту же окр-ть
   Circ1  =  null;  // а теперь Circ1 ничего не адресует
   Rectangle   Rect1,  Rect2;  // только объявили, объекты не созданы
   Rect1.Show( ); // ОШИБКА: переменная Rect1 НЕ инициализирована
   Rect1 = Circ2; // ОШИБКА: переменные РАЗНЫХ классовых типов!
   Rect2 =  null;  //  а так можно
   Rect2.Show( );   //  ОШИБКА:  Rect2   НИЧЕГО не адресует!
```

Особенности использования операции сравнения:

- две объектные переменные можно сравнивать только на **равенство** или **неравенство** (операции сравнения типа «больше-меньше» не разрешены)

- сравнивать можно переменные **одного и того же** классового типа
- сравнение выполняется **на уровне адресных значений** переменных, но не на уровне самих адресуемых объектов
- объектную переменную **любого** классового типа можно сравнивать с **нулевым** (пустым) адресным значением, используя для этого ключевые слова **nil** или **null**

Примеры для графических объектов.

Delphi

```
if (Circ1 = Circ2) then . . .  // проверяется условие: адресуют ли обе
                                // переменные одну и ту же окружность
if (Rect1 <> nil) then . . . // проверка на нулевой адрес
if (Circ1 > Circ2) then . . . // ОШИБКА: не имеет смысла для адресов
```

Java и C#

```
if (Circ1 == Circ2) . . . // сравнение на равенство адресов
while (Rect1 != null) { … } // повторение цикла, пока Rect1
                            // адресует прямоугольники
```

В заключение надо отметить, что поскольку объекты создаются динамически в процессе выполнения программы, возможна ситуация **исчерпания** свободной динамически распределяемой памяти (кучи). Вероятность возникновения этой ситуации зависит от многих факторов, в первую очередь – от байтового размера создаваемых объектов и объема свободной оперативной памяти. Возникновение такой особой ситуации можно отследить с помощью специального механизма **исключений** (**exception**), который рассматривается в разделе 17.