**25.Объектно-ориентированные языки программирования. Свойства объекта.**

Поля предназначены для использования внутри класса. Однако класс должен каким-либо образом взаимодействовать с другими классами или программными элементами приложения. В подавляющем большинстве случаев класс должен выполнить с некоторыми данными определенные действия и представить результат.

Для получения и передачи данных в классе применяются свойства.

Свойства представляют собой атрибуты, которые составляют индивидуальность объекта и помогают описать его. Например, обычная кнопка в окне приложения обладает такими свойствами, как цвет, размеры, положение.

Так как свойство обеспечивает обмен данными с внешней средой, то для доступа к его значению используются специальные методы класса. Поэтому обычно свойство определяется двумя элементами: двумя методами, которые осуществляют его чтение/запись.

Свойства реализованы таких языках как C#,Delphi,java. В С++ свойств нет.

**26. Объектно-ориентированные языки программирования. Векторные свойства.**

**На примере Delphi**

Свойство может быть и векторным; в этом случае оно внешне выглядит как массив:

property APoints[Index : Integer]:TPoint read GetPoint write SetPoint;

На самом деле в классе может и не быть соответствующего поля — массива. Напомним, что вся обработка обращений к внутренним структурам класса может быть замаскирована.

Для векторного свойства необходимо описать не только тип элементов массива, но также имя и тип индекса. После ключевых слов read и write в этом случае должны стоять имена методов — использование здесь полей массивов недопустимо. Метод, читающий значение векторного свойства, должен быть описан как функция, возвращающая значение того же типа, что и элементы свойства, и имеющая единственный параметр того же типа и с тем же именем, что и индекс свойства:

function GetPoint(Index:Integer):TPoint;

Аналогично, метод, помещающий значения в такое свойство, должен первым параметром иметь индекс, а вторым — переменную нужного типа (которая может быть передана как по ссылке, так и по значению):

procedure SetPoint(Index:Integer; NewPoint:TPoint);

У векторных свойств есть еще одна важная особенность. Некоторые классы в Delphi (списки т-List, наборы строк TStrings) "построены" вокруг основного векторного свойства (см. гл. 7). Основной метод такого класса дает доступ к некоторому массиву, а все остальные методы являются как бы вспомогательными. Специально для облегчения работы в этом случае векторное свойство может быть описано с ключевым словом default:

type

TMyObject = class;

property Strings[Index: Integer]: string read Get write Put; default;

  end;

Если у объекта есть такое свойство, то можно его не упоминать, а ставить индекс в квадратных скобках сразу после имени объекта:

var AMyObject: TMyObject;

begin

...

AMyObject.Strings[1] := 'First'; {первый способ}

 AMyObject[2] := 'Second'; (второй способ}

...

end.

Будьте внимательны, применяя зарезервированное слово default, — как мы увидели, для обычных и векторных свойств оно употребляется в разных случаях и с различным синтаксисом.

**27.Полиморфизм. Статические, виртуальные, динамические методы.**

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

[Статический](http://msdn.microsoft.com/ru-ru/library/98f28cdx.aspx) класс в основном такой же, что и нестатический класс, но имеется одно отличие: нельзя создавать экземпляры статического класса. Другими словами, нельзя использовать ключевое слово [new](http://msdn.microsoft.com/ru-ru/library/51y09td4.aspx) для создания переменной типа класса. Поскольку нет переменной экземпляра, доступ к членам статического класса осуществляется с использованием самого имени класса. Например, если имеется статический класс, называемый UtilityClass, имеющий открытый метод, называемый MethodA, вызов метода выполняется, как показано в следующем примере.

C#

UtilityClass.MethodA();

Принципиально отличаются от статических виртуальные и динамические методы. Они должны быть объявлены путем добавления соответствующей директивы virtual или dynamic. Обе эти категории существовали и в прежних версиях языка Pascal. С точки зрения наследования методы этих двух видов одинаковы: они могут быть перекрыты в дочернем классе только одноименными методами, имеющими тот же тип.

Если задуматься над рассмотренным выше примером, становится ясно, что у компилятора нет возможности определить класс объекта, фактически переданного в процедуру showData. Нужен механизм, позволяющий определить это прямо во время выполнения. Такой механизм называется поздним связыванием (late binding).

Естественно, что этот механизм должен быть каким-то образом связан с передаваемым объектом. Для этого используются таблица виртуальных методов (Virtual Method Table, VMT) и таблица динамических методов (Dynamic Method Table, DMT).

Разница между виртуальными и динамическими методами заключается в особенности поиска адреса. Когда компилятор встречает обращение к виртуальному методу, он подставляет вместо прямого вызова по конкретному адресу код, который обращается к VMT и извлекает оттуда нужный адрес.

Такая таблица есть для каждого класса (объектного типа). В ней хранятся адреса всех виртуальных методов класса, независимо от того, унаследованы ли они от предка или перекрыты в данном классе. Отсюда и достоинства, и недостатки виртуальных методов: они вызываются сравнительно быстро, однако для хранения указателей на них в таблице VMT требуется большое количество памяти.

Динамические методы вызываются медленнее, но позволяют более экономно расходовать память. Каждому динамическому методу системой присваивается уникальный индекс. В таблице динамических методов класса хранятся индексы и адреса только тех динамических методов, которые описаны в данном классе. При вызове динамического метода происходит поиск в этой таблице; в случае неудачи просматриваются таблицы DMT всех классов-предков в порядке иерархии и, наконец, класс TObject, где имеется стандартный обработчик вызова динамических методов. Экономия памяти налицо.

**28.Абстрактные методы. Директива override.**

Для перекрытия и виртуальных, и динамических методов служит директива override, с помощью которой (и только с ней!) можно переопределять оба этих типа методов.

**Абстра́ктный ме́тод** (или **чистый виртуальный метод** (**pure virtual method** - часто неверно переводится как **чисто виртуальный метод**)) — в [объектно-ориентированном программировании](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), метод [класса](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), реализация для которого отсутствует. Класс, содержащий абстрактные методы, также принято называть [абстрактным](http://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81) (там же и пример). Абстрактные методы зачастую путают с [виртуальными](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4). Абстрактный метод подлежит определению в классах-наследниках, поэтому его можно отнести к виртуальным, но не каждый виртуальный метод является абстрактным.