Свойства объектов, их отличие от полей, ограничение на запись и чтение. Инкапсуляция. Показать реализацию принципа на предложенном примере.

Свойства объектов. Инкапсуляция.

Классическое правило объектно-ориентированного программирования утверждает, что для обеспечения надежности нежелателен прямой доступ к полям объекта[5]: чтение и обновление их содержимого должно производиться посредством вызова соответствующих методов. Это правило называется **инкапсуляцией**[6].. В языке Object Pascal принцип инкапсуляции реализуется посредством **свойств** - виртуальных характеристик объектов, базирующихся на значениях конкретных полей.

Обратите внимание! Поля объекта хранят данные (т.е. занимают место в памяти), а свойства посредством вызова соответствующих методов позволяют определять и узнавать различные характеристики объектов, которые, в общем случае, могут и не храниться в памяти, а, например, вычислться по формулам.

Обычно свойство определяется полем соответствующего типа и двумя методами доступа – для чтения значения поля (функция) и изменения значения (процедура):

```
type
      TMyClass = class
        fValue : TSomeType;
        function GetValue: TSomeType;
        procedure SetValue(NewValue: TSomeType);
        property pValue: TSomeType read GetValue write SetValue;
      end;
```

В этом примере

`fValue` – поле для хранения значения свойства,
`GetValue` – метод для чтения значения свойства,
`SetValue` – метод для изменения значения свойства,
`pValue` – само свойство,
`TSomeType` – тип данного свойства.

В тексте программы использование свойства не отличается от использования обычного поля, но при компиляции вызов свойства для чтения или для записи заменяется вызовом соответствующих методов, например (`var MyObject : TMyClass`)[7]:

исходный текст	интерпретируется как
`MyObject.pValue:=AValue;`	`MyObject.SetValue(AValue);`
`AVariable:= MyObject.pValue;`	`AVariable:= MyObject.GetValue;`

В рамках методов `Get` и `Set` прописываются дополнительные операции, которые необходимо выполнить для реализации чтения или изменения свойства. Например, при изменении

[5] Об ограничении доступа к полям объектов см. раздел "Области видимости".
[6] Некоторые авторы под инкапсуляцией понимают наделение компьютерного объекта свойствами и методами, соответствующими характеристикам реального объекта (см. например фразу из Фаронова "Объект Screen класса TScreen инкапсулирует свойства и методы, упрощающие работу с дисплеем ПК...").
[7] Поэтому свойства не могут выступать в вызовах процедур и функций в качестве параметров-переменных. Т.е. оператор `Inc(fValue);` корректен, а оператор `Inc(pValue);` - ошибочен.

ширины формы надо не просто занести в соответствующую ячейку новое значение, но и выполнить операции по перерисовке на мониторе как самой формы, так и окружающей ее области.

Если же дополнительные операции не требуются, то при описании свойства ссылку на метод можно заменить ссылкой на поле. Рассмотрим следующее описание:

```
type
      TMyClass = class
        fValue : TSomeType;
        procedure SetValue(NewValue: TSomeType);
        property pValue: TSomeType read fValue write SetValue;
        function Correct(Value: TSomeType):boolean;
        procedure DoSomeThing;
      end;

...
procedure TmyClass.SetValue(NewValue: TSomeType);
begin
  if Correct(NewValue) then fValue:=NewValue;
  DoSomeThing;
end;
```

Здесь чтение свойства `pValue` означает просто чтение поля `fValue`. Зато при присвоении ему значения внутри `SetValue` будут вызваны сразу два метода.

Если свойство только читается или записывается, то в его описании присутствует только один, соответствующий, метод:

```
property pValue: TSomeType read GetValue;
```

В этом примере попытка изменить значение свойства `pValue` вызовет ошибку компиляции.

Часто количество свойств, основанных на группе полей может значительно превышать число самих полей. Например, описывая как объект точку на плоскости можно на основе двух полей – координат точки X и Y можно построить достаточно много свойств - характеристик точки: X и Y - декартовые координаты (для чтения и записи), ρ и φ - полярные координаты (для чтения и записи), K- номер координатной четверти (от 1 до 4, только для чтения) и т.п.

В заключении заметим, что в семействе компонент Delphi абсолютно все поля недоступны и представлены соответствующими свойствами.