Принцип наследования. Перекрытие полей и методов. Области видимости. Показать реализацию принципа на предложенном примере.

Принцип наследования.

Вторым основным принципом объектно-ориентированного программирования является **наследование**. Смысл его в том, что если вы желаете создать новый класс, лишь немногим отличающийся от уже описанного, то нет необходимости переписывания заново существующих полей и методов. Достаточно просто объявить, что новый класс порождается от существующего

```
type  TNewClass = class (TOldClass),
```

т.е. является **потомком** или **дочерним классом** старого класса, называемого **предком** или **родительским классом**. При этом к новому классу автоматически (по умолчанию) будут добавлены все поля, методы и свойства старого класса. Поэтому, при описании потомка достаточно просто указать новые, дополнительные поля, методы и свойства.

Если в описании нового класса имя предка не указано (как в примерах выше) то новый класс порождается от класса `TObject`, ближними или далекими потомками которого являются все остальные классы. Кстати, методы `Create`, `Free` и `Destroy` наследуются именно от класса `TObject`.

Если имеет место совпадение имен новых и унаследованных полей и методов, то говорят, что они **перекрываются**. Рассмотрим пример перекрытия полей и статических методов:

```
type
      T1 = class
        a : integer;
        procedure SetA(Value:integer);
      end;

      T2 = class (T1)
        a : real;
        procedure SetA(Value: real);
      end;

...
procedure T1.SetA(Value:integer);
begin
  a:=Value;
end;

procedure T2.SetA(Value:real);
begin
  a:=0;
  inherited SetA(Round(Value))
end;
```

В этом примере разные методы с именем `SetA` присваивают значения разным полям с именем `a`. В классе потомке одноименные поля и методы не наследуются, а перекрываются.

Однако, если перекрытое поле предка недоступно вообще, то перекрытый метод доступен при указании зарезервированного слова `inherited`[8].

Области видимости (доступности).

При описании класса важно соблюсти разумный компромисс. С одной стороны, требуется скрыть ряд внутренних методов и полей, одни из которых бесполезны пользователю класса и только усложняют его интерфейс, а доступ к другим полям нужно организовать через систему проверок или свойств (инкапсуляция).

С другой стороны, если слишком ограничивать возможного пользователя класса, то данный класс может стать ему неинтересен.

В языке Object Pascal применяют следующие виды доступа к полям, методам и свойствам:

- **`Public`** (общие). Поля, методы и свойства из этой секции не имеют ограничений на видимость. Они доступны из других функций и методов объектов как в данном модуле, так и в других модулях, ссылающихся на него.
- **`Protected`** (защищенные). Поля, методы и свойства этой секции доступны всем функциям и методом данного модуля. В других модулях доступны только в классах, порожденных от данного класса.
- **`Private`** (личные). Наибольшее ограничение доступности. Поля, методы и свойства из этой секции доступны только в данном модуле и недоступны из других модулей.

Кроме основных трех областей видимости существуют еще две:

- **`Published`** (опубликованные). Аналогично `public`, только свойства из данной секции доступны еще и из среды визуального программирования (Object Inspector).
- **`Automated`**. Появилась в последних версиях Delphi. Используется для создания объектов Автоматизации (COM-технологии).

В классе-потомке можно переопределить область видимость метода, не понижая ее.

```
type
      T1 = class
        private
          fNumber : integer;
        protected
          property Number:integer read fNumber;
      end;

      T2 = class (T1)
        published
          property Number;
      end;
```

Здесь переводом свойства `Number` в группу опубликованных свойств, была повышена его видимость. Повысить видимость элементов секции `private` нельзя.

[8] В данном примере вызов перекрытого метода меняет значение не данного, а перекрытого поля. Поэтому значение нового поля останется равным нулю.