Инкапсуляция

Объединение в объекте кода и данных называется инкапсуляцией.

Объекты ***TPerson*** *и* ***TStudeat*** *написаны таким образом, что* ***нет необходимости в прямом обращении к их внутренним полям данных*****.**

```
TYPE
    Tperson = OBJECT
        Name  : STRING[30];
        Date  : STRING[10];
        Rate  : REAL;
        PROCEDURE Init(Nm,Dt:STRING; Rt:REAL);
        FUNCTION GetName : STRING;
        FUNCTION GetDate : STRING;
        FUNCTION GetRate : REAL;
        PROCEDURE ShowName;
        PROCEDURE ShowDate;
        PROCEDURE ShowRate;
    END;
```

Инкапсуляция

```
TYPE
    TStudent = OBJECT(TPerson)
        Ball     : REAL;
        PROCEDURE  Init(Nm,Dt:STRING;  Rt,Bl:REAL);
        FUNCTION GetBall    :   REAL;
        FUNCTION GetSum     :   REAL;
        PROCEDURE ShowBall;
        PROCEDURE ShowSum;
        PROCEDURE ShowAll;
    END;
```

Инкапсуляция

ИНКАПСУЛЯЦИЯ

Для экземпляра ***Student*** *типа* ***TStudent*** *можно использовать набор методов для косвенной работы с полями данных, например:*

```
WITH Student  DO
BEGIN
     Init   ('Петр Петров','25-06-1995', 40000,4.5);
     ShowAll;
END;
```

Обратите внимание, что ***доступ к полям объекта осуществляется только с помощью методов этого объекта*.**

Полиморфизм

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ

При использовании стандартных средств Турбо Паскаля очень трудно, если вообще возможно, создавать гибкие процедуры, которые работали бы с формальными параметрами переменных типов.

- *как это делает, к примеру процедура* ***WriteLn****, которая может выводить на экран или в файл данные типа* ***STRING****,* ***REAL****,* ***INTEGER****,* ***BOOLEAN*** *и родственные им.*

Эта проблема решается в ООП с помощью механизма наследования

- *если определен порожденный тип, то методы порождающего типа наследуются, однако, при желании они могут переопределяться.*

Для переопределения наследуемого метода просто описывается новый метод с тем же именем, что и наследуемый метод,
***но с другим телом и с другим множеством параметров*.**

Проиллюстрируем этот процесс на простом примере.

Ранее нами был определен тип ***TStudent***, являющийся потомком типа ***TPerson***:

```
TYPE
    TStudent = OBJECT(TPerson)
        Ball     : REAL;
        PROCEDURE Init(Nm,Dt:STRING; Rt,Bl:REAL);
        FUNCTION GetBall    : REAL;
        FUNCTION GetSum     : REAL;
        PROCEDURE ShowSum;
        PROCEDURE ShowBall;
        PROCEDURE ShowAll;
    END;
```

Переопределение методов

ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ

*Чем **TStudent** похож на **Tperson?***

- *Студент имеет все характеристики, которые используются для определения объекта **TPerson** (фамилию, дату выплат, ставку)*

*Чем **TStudent** отличается от **Tperson?***

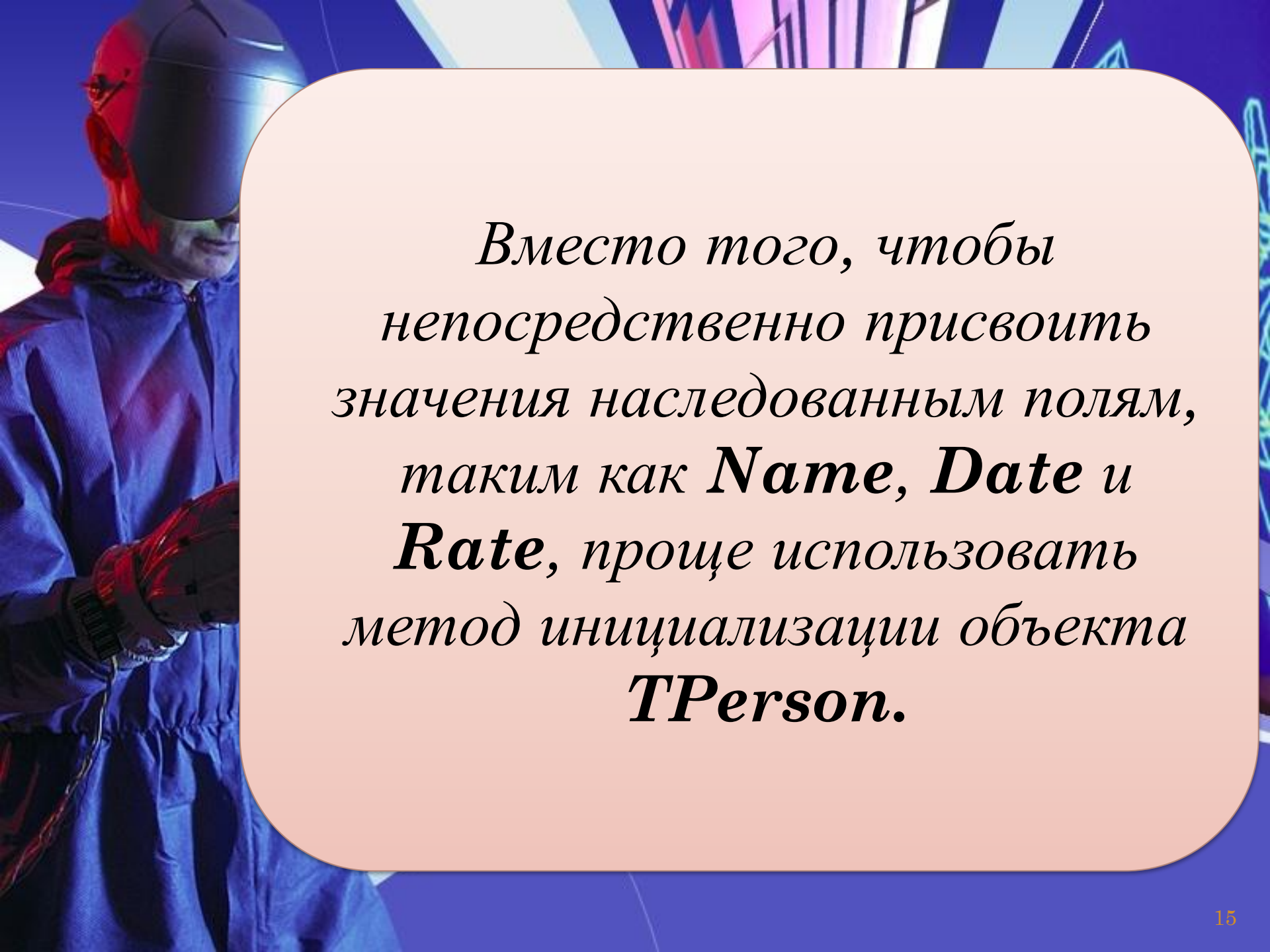
- *Для объекта **TStudent** требуется еще и поле среднего балла **Ball.***

Т.к. ***TStudent*** определяет новое поле ***Ball***, его инициализация требует нового метода ***Init***, который инициализирует и средний балл, и наследованные поля.

- *Процедура **Init** наследуется от предка, но в данном случае ее необходимо переопределить:*

```
PROCEDURE TStudent.Init(Nm,Dt:STRING; Rt,Bl:REAL);
    BEGIN
        TPerson.Init(Nm,Dt,Rt) ;
        Ball:=Bl;
    END;
```

Переопределение методов



Вместо того, чтобы непосредственно присвоить значения наследованным полям, таким как ***Name***, ***Date*** *и* ***Rate***, *проще использовать метод инициализации объекта* ***TPerson.***

Синтаксис вызова наследуемого метода следующий:

Предок.Метод;

где

Предок — это идентификатор типа родительского объекта;
Метод — идентификатор метода этого типа.

Необходимо обратить внимание на то, что вызов переопределяемого метода не является обязательным.

- В общем случае ***TPerson.Init*** выполняет важную, но скрытую инициализацию.

При вызове переопределяемого метода необходимо быть уверенным в том, что порожденный тип объекта учитывает особенности функционирования родителя.

Кроме того, любое изменение в родительском Методе автоматически оказывает влияние на все порожденные.

- Каждый порожденный тип объекта ***TPerson*** имеет свой метод ***GetSum***, т.к. расчет производится в каждом случае по разному.
  - Метод ***TStudent.GetSum***
  - должен учитывать средний балл студента.
  - Метод ***TTeacher.GetSum*** должен учитывать количество лекционных часов, часовую ставку и премиальные выплаты.
  - Метод ***TStaff.GetSum*** должен учитывать только размер премиальных выплат.

Переопределение методов

```
UNIT Persons;

INTERFACE

TYPE
  TPerson = OBJECT
  PRIVATE
    Name: STRING[30];
    Date: STRING[10]
    Rate: REAL;
  PUBLIC
    PROCEDURE Init(Nm,Dt:STRING; Rt:REAL);
    PROCEDURE ShowAll;
  END;

  TStudent = OBJECT(TPerson)
  PRIVATE
    Ball: REAL;
  PUBLIC
    PROCEDURE Init(Nm,Dt:STRING;                    Rt,Bl:REAL);
    FUNCTION GetSum : REAL;
    PROCEDURE ShowSum;
    PROCEDURE ShowAll;
  END;
```

Примерный текст интерфейсной части модуля Persons


```
TStaff = OBJECT(TPerson)
PRIVATE
```

Метод ***TStudent.GetSum***, в котором учитывается средний балл:

```
FUNCTION TStudent.GetSum  :  REAL;
 BEGIN
    GetSum := Rate*Ball;
 END;
```

В методе ***TStaff.GetSum*** к ставке прибавляется размер премиальных выплат :

```
FUNCTION TStaff.GetSum  :  REAL;
 BEGIN
     GetSum := Rate+Bonus;
 END;
```

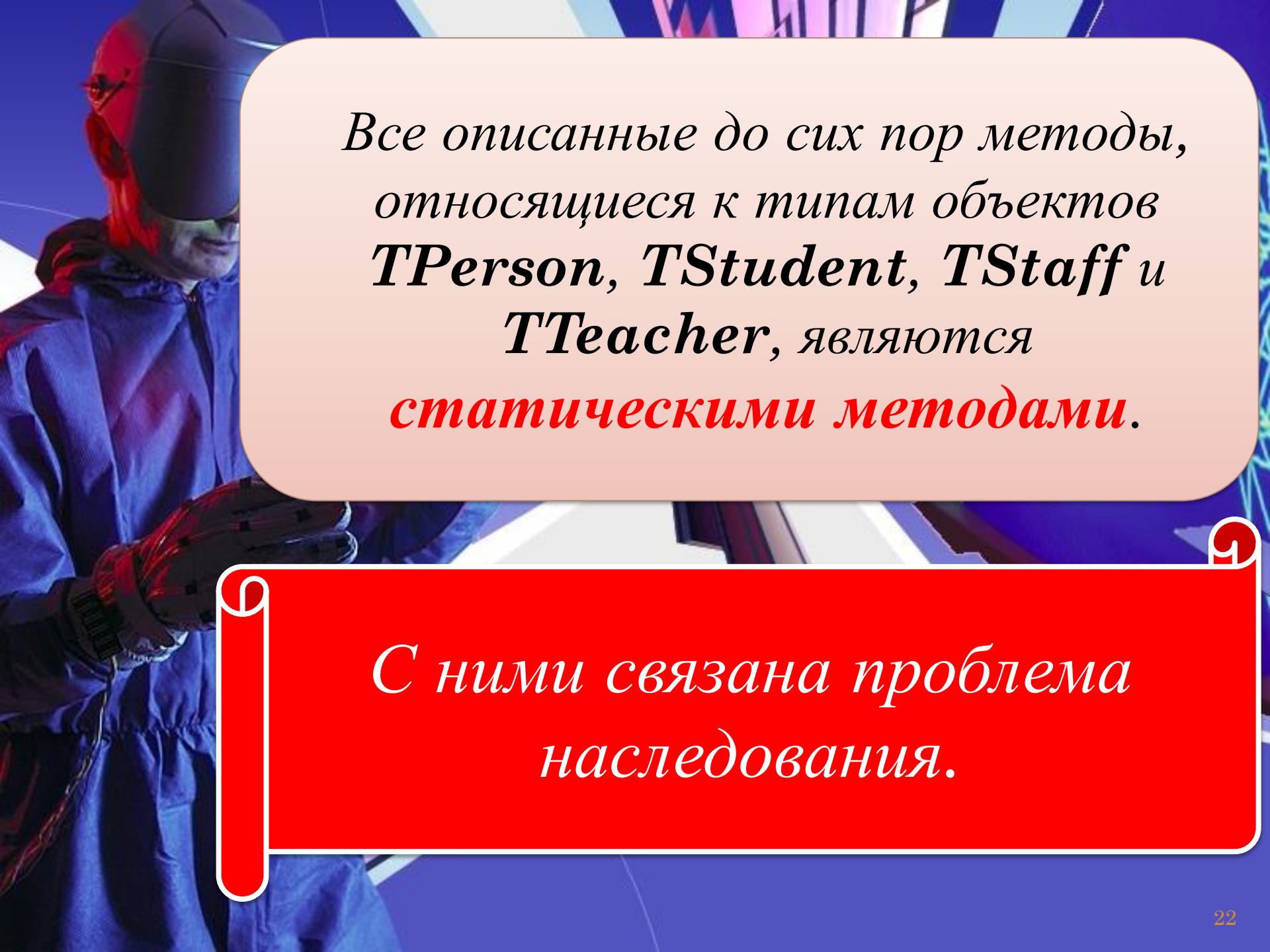
Метод ***TTeacher.GetSum*** вызывает ***TStaff.GetSum*** и добавляет к этому значению размер часовой ставки, умноженный на количество часов :

```
FUNCTION TTeacher.GetSum  :  REAL;
 BEGIN
   GetSum:= TStaff.GetSum+Hours*HourRate;
 END;
```

Тексты методов GetSum

Необходимо помнить, что хотя методы могут быть переопределены, поля данных переопределяться не могут.

После того как поле данных в иерархии объекта определено, никакой дочерний тип не может определить поле данных с таким же именем.



Все описанные до сих пор методы, относящиеся к типам объектов ***TPerson***, ***TStudent***, ***TStaff*** *и* ***TTeacher***, *являются* ***статическими методами***.

С ними связана проблема наследования.

- Рассмотрим процедуру ***ShowSum***
- Для объекта типа ***TStaff*** она имеет вид:

```
PROCEDURE TStaff.ShowSum;
BEGIN
WriteLn(GetSum);
END;
```

- Как ни странно, для объекта типа ***TTeacher*** эта процедура имеет тот же самый вид:

```
PROCEDURE TTeacher.ShowSum;
BEGIN
  WriteLn(GetSum);
END;
```

Наследование статических методов

НАСЛЕДОВАНИЕ СТАТИЧЕСКИХ МЕТОДОВ

- *Есть ли необходимость выполнять в процедуре **ShowSum** какие-либо дополнительные действия?*
  - *По сравнению с типом **TStaff** не изменилось ничего, кроме копирования процедуры и подстановки квалификатора **TTeacher** перед идентификатором **ShowSum**.*
- *Возникает резонный вопрос, нет ли здесь логической ошибки?*
  - *Поскольку методы одинаковы, нет нужды помещать **ShowSum** в **TStaff** и **TTeacher**.*

Именно здесь и возникает проблема, связанная со статическими методами

Проблема заключается в следующем:

Пока копия метода ***ShowSum*** не будет помещена в область действия ***TTeacher*** для подавления метода ***ShowSum*** объекта ***TStaff***, метод не будет работать правильно, если он будет вызываться из объекта типа ***TTeacher***.

- Если ***TTeacher*** запускает метод ***ShowSum*** объекта ***TStaff***, то и функция ***GetSum***, используемая в методе, будет принадлежать объекту ***TStaff***, и зарплата будет рассчитана неправильно, без учета количества часов.

Наследование статических методов

НАСЛЕДОВАНИЕ СТАТИЧЕСКИХ МЕТОДОВ

Это объясняется способом, которым компилятор осуществляет вызов методов.

- *Как и при вызове любой процедуры, компилятор замещает ссылки на* ***TStaff.GetSum*** *и* ***TStaff.ShowSum*** *в исходном коде на их адреса в сегменте кода.*
- *Поскольку тип* ***TTeacher*** *является потомком типа* ***TStaff****, то сначала в сегмент кода будет скомпилирована функция* ***TStaff.GetSum****.*
- *Затем будет скомпилирована процедура* ***TStaff.ShowSum****, вызывающая* ***TStaff.GetSum***

При вызове код ***TStaff.ShowSum*** в свою очередь вызывает ***TStaff.GetSum***, что и составляет проблему.

Фактически, наследуется следующая процедура:

```
PROCEDURE TStaff.ShowSum;
  BEGIN
     WriteLn(GetSum);
  END;
```

Метод объекта ***TStaff*** ничего не знает о существовании объекта ***TTeacher***.

Таким образом, метод ShowSum нельзя наследовать.

Вместо этого он должен быть переопределен своей второй копией, вызывающей уже правильный метод.

Наследование статических методов

Вызывая методы, компилятор работает так:

Наследование статических методов

Наследование статических методов

Если статический наследуемый метод найден и используется, то необходимо помнить, что вызываемый метод является в точности таким, каким он был определен и скомпилирован для родительского типа.

- ***Если родительский метод вызывает другие методы, то вызываемые методы будут также родительскими методами, даже если дочерний объект содержит методы, которые переопределяют родительские.***

Наследование статических методов

Статические методы являются таковыми в том же смысле, в каком статической является статическая переменная: компилятор размещает ее и определяет все ссылки на нее во время компиляции.

- Сами по себе статические методы могут быть мощным инструментом для составления сложных программ.
- Но иногда при их использовании возникают проблемы, подобные описанной ранее.

Выход заключается в том, что метод должен быть динамическим, а ссылки на него должны определяться во время выполнения.

- Чтобы это стало возможным, Турбо Паскаль предоставляет механизмы поддержки так называемых виртуальных методов.

Виртуальные методы предоставляют чрезвычайно мощный инструмент для обобщения, называемый полиморфизмом.

Полиморфизм является способом присвоения действию имени, которое используется всеми объектами иерархии, причем каждый объект иерархии использует это действие определенным образом.

Описанная ранее простая иерархия геометрических фигур является хорошим примером полиморфизма в действии, предоставляемого с помощью виртуальных методов.

- Каждый тип объекта в иерархии представляет отдельный тип фигуры на экране.
- Если возникнет необходимость определить объекты для представления на экране других типов фигур, таких как прямоугольник, треугольник и т.д., можно написать метод для

Особым для каждого типа объекта является способ отображения самого себя на экране.

- Можно отобразить на экране любой тип фигуры, но механизм рисования каждой является сугубо индивидуальным.
- Одно слово "отобразить" используется для вывода многих фигур.
- То же самое в примере с платежной ведомостью — функция ***GetSum*** вычисляет размер выплат для различных типов лиц.

Это и есть полиморфизм, а виртуальные методы реализуют его в Турбо Паскале.

- Различие между вызовом статического метода и динамического метода
  - в случае статического метода компилятору заранее известна связь объекта с методом, и он устанавливает ее на этапе компиляции.
  - в случае динамического метода компилятор как бы откладывает решение до момента выполнения программы.

Раннее и позднее связывание

Раннее и позднее связывание

Процесс, с помощью которого вызовы статических методов связываются компилятором во время компиляции в один метод, называется
ранним связыванием.

Раннее и позднее связывание

При раннем связывании вызывающий и вызываемый методы связываются при первой же возможности

- т.е. во время компиляции

При позднем связывании вызывающий и вызываемый методы не могут связываться во время компиляции, поэтому включается механизм, позволяющий осуществить связывание позднее

- когда вызов действительно произойдет

Раннее и позднее связывание