Тема:
История развития ООП. Базовые понятия ООП: объект, его свойства и методы, класс, интерфейс. Основные принципы ООП: инкапсуляция, наследование, полиморфизм.

Понятие абстрактных типов данных является ключевым в программировании.

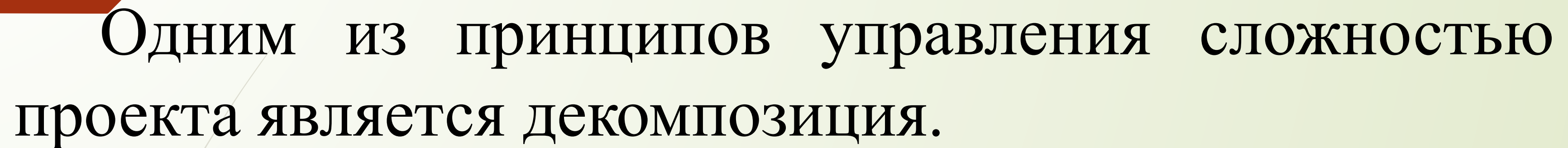
Абстракция подразумевает разделение и независимое рассмотрение интерфейса и реализации .

Абстракция данных предполагает определение и рассмотрение **абстрактных типов данных**(АТД) или, новых типов данных, введенных пользователем .

Абстрактный тип данных — это совокупность данных вместе с множеством операций, которые можно выполнять над этими данными .

Понятие объектно-ориентированного программирования

По определению авторитета в области объектно-ориентированных методов разработки программ Гради Буча «объектно-ориентированное программирование (ООП) – **это методология программирования, которая основана на представлении программы в виде совокупности объектов, каждый из которых является реализацией определенного класса (типа особого вида), а классы образуют иерархию на принципах наследуемости».**



Одним из принципов управления сложностью проекта является декомпозиция.

Гради Буч выделяет две разновидности декомпозиции: алгоритмическую (так он называет декомпозицию, поддерживаемую структурными методами) и объектно-ориентированную, отличие которых состоит, по его мнению, в следующем:

«Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение факторам, либо вызывающим действия, либо являющимся объектами приложения этих действий».

Объекты и классы

Объект — это часть окружающей нас реальности, т. е. он существует во времени и в пространстве (впервые понятие объекта в программировании введено в языке Simula).

Класс — это множество объектов, имеющих общую структуру и общее поведение. Класс — описание (абстракция), которое показывает, как построить существующую во времени и пространстве переменную этого класса, называемую **объектом.**

Определим теперь понятия **состояния, поведения и идентификации** объекта.

Состояние объекта объединяет все его поля данных (статический компонент, т.е. неизменный) и текущие значения каждого из этих полей (динамический компонент, т.е. обычно изменяющийся).

Поведение выражает динамику изменения состояний объекта и его реакцию на поступающие сообщения, т.е. как объект изменяет свои состояния и взаимодействует с другими объектами.

Идентификация (распознавание) объекта — это свойство, которое позволяет отличить объект от других объектов того же или других классов. Осуществляется идентификация посредством уникального имени (паспорта), которым наделяется объект в программе, впрочем как и любая другая переменная.

Базовые принципы ООП

К базовым принципам объектно-ориентированного стиля программирования относятся:

- **пакетирование или инкапсуляция;**
- **наследование;**
- **полиморфизм;**
- **передача сообщений.**

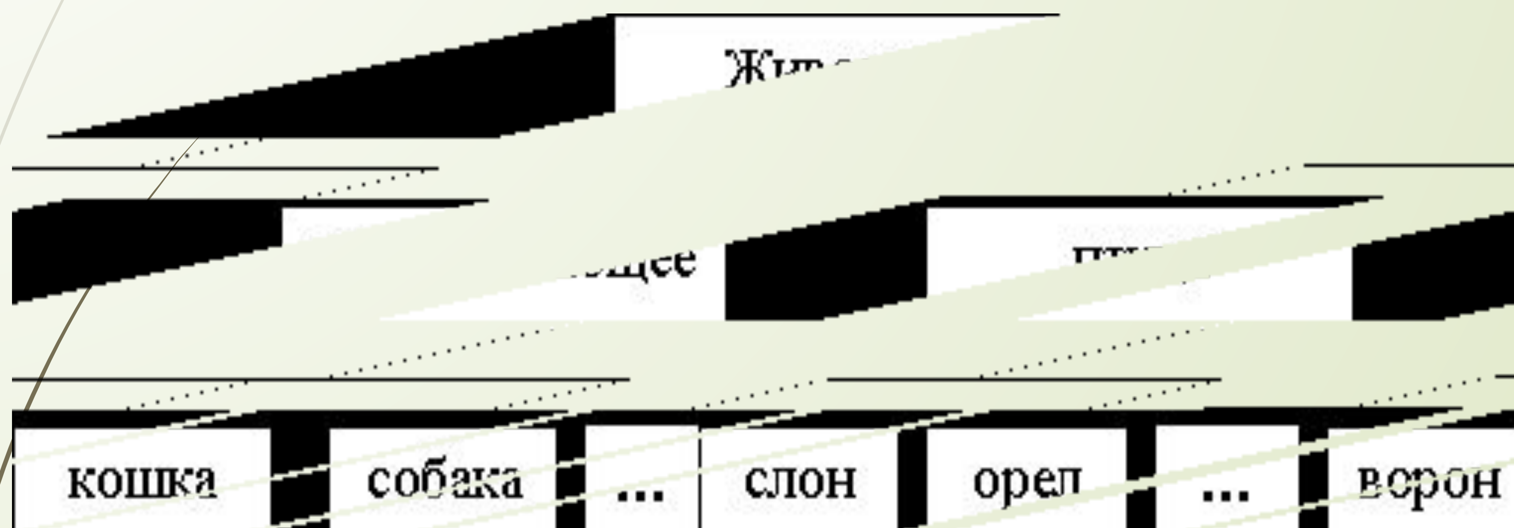
Пакетирование (инкапсуляция) предполагает соединение в одном объекте данных и функций, которые манипулируют этими данными. Доступ к некоторым данным внутри пакета может быть либо запрещен, либо ограничен.

Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте алгоритмы обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируется.

Наследование

И структурная, и объектно-ориентированная методологии преследуют цель построения иерархического дерева взаимосвязей между объектами (подзадачами). Но если структурная иерархия строится по простому принципу разделения целого на составные части,

то при создании объектно-ориентированной иерархии принимается другой взгляд на тот же исходный объект. В объектно-ориентированной иерархии непременно отражается наследование свойств родительских (вышележащих) типов объектов дочерним (нижележащим) типам объектов.



По Гради Бучу «наследование – это такое отношение между объектами, когда один объект повторяет структуру и поведение другого».

Наследование позволяет использовать библиотеки классов и развивать их (совершенствовать и модифицировать библиотечные классы) в конкретной программе.

Наследование позволяет создавать новые объекты, изменяя или дополняя свойства прежних.

Объект-наследник получает все поля и методы предка, но может добавить собственные поля, добавить собственные методы или перекрыть своими методами одноименные унаследованные методы.

Последовательное проведение в жизнь принципа «наследуй и изменяй» хорошо согласуется с поэтапным подходом к разработке крупных программных проектов и во многом стимулирует такой подход.

Когда вы строите новый класс, наследуя его из существующего класса, можно:

- добавить в новый класс новые компоненты-данные;
- добавить в новый класс новые компоненты-функции;
- заменить в новом классе наследуемые из старого класса компоненты-функции.

Полиморфизм позволяет использовать одни и те же функции для решения разных задач.

Полиморфизм выражается в том, что под одним именем скрываются различные действия, содержание которых зависит от типа объекта.

Полиморфизм – это свойство родственных объектов (т.е. объектов, имеющих одного общего родителя) решать схожие по смыслу проблемы разными способами.

Например, действие «бежать» свойственно большинству животных. Однако каждое из них (лев, слон, крокодил, черепаха) выполняет это действие различным образом.

Описание объектного типа

Класс или объект – это структура данных, которая содержит поля и методы.

Как всякая структура данных она начинается зарезервированным словом и закрывается оператором **end** .

Формальный синтаксис не сложен: описание объектного типа получается, если в описании записи заменить слово **record** на слово **object** или **class** и добавить объявление функций и процедур над полями.

```
<поле>;
<поле>;
….
<метод>;
 <метод>;
end ;
```

В Object Pascal существует специальное зарезервированное слово **class** для описания объектов, заимствованное из C++.

Type <имя типа объекта>= class
<поле>;
....
<метод>;
<метод>;
end ;

ObjectPascal поддерживает обе модели описания объектов.

Компонент объекта – либо поле, либо метод.

Поле содержит имя и тип данных.

Метод – это процедура или функция, объявленная внутри декларации объектного типа, в том числе и особые процедуры, создающие и уничтожающие объекты (конструкторы и деструкторы).

Объявление метода внутри описания объектного типа состоит только из заголовка. Это разновидность предварительного описания подпрограммы. Тело метода приводится вслед за объявлением объектного типа.

Пример.

Вводится объектный тип «предок», который имеет поле данных Name (имя) и может выполнять два действия:

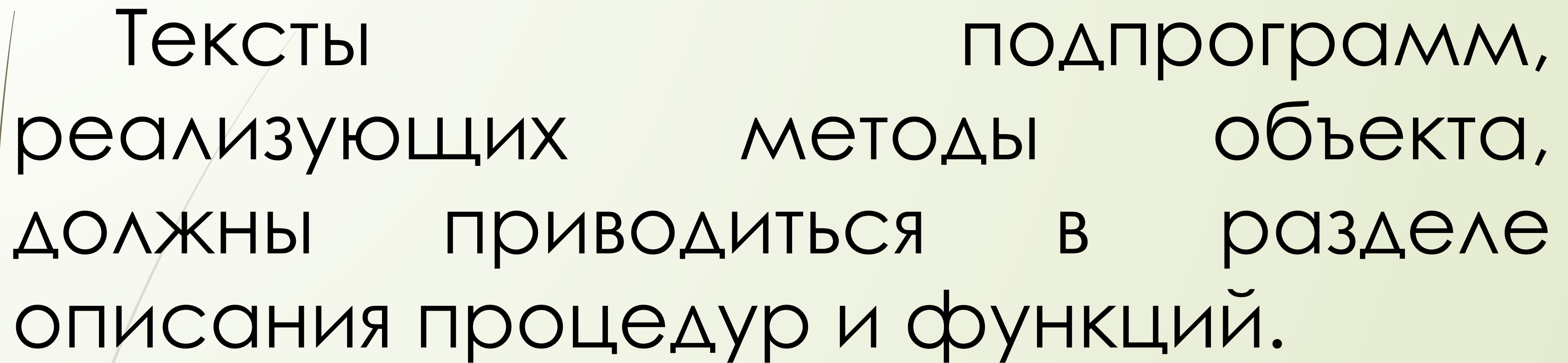
- провозглашать: «Я – предок!»;
- сообщать свое имя.

Type tPredoc = object Name : string ; {поле данных объекта}

Procedure Declaration ; {объявление методов объекта}

Procedure MyName ;

End ;



Тексты подпрограмм, реализующих методы объекта, должны приводиться в разделе описания процедур и функций.

Заголовки при описании реализации метода повторяют заголовки, приведенные в описании типа, но дополняются именем объекта, которое отделяется от имени процедуры точкой.

В нашем примере:

Procedure tPredoc.Declaration; {реализация метода объекта}

begin

writeln (‘Я – предок!’);

end ;

Procedure tPredoc.MyName; {реализация метода объекта}

begin

writeln(‘Я –’, Name);

End;

Внутри описания методов на поля и методы данного типа ссылаются просто по имени.

Так метод **MyName** использует поле **Name** без явного указания его принадлежности объекту так, если бы выполнялся неявный оператор **with <переменная_типа_объект> do** .

Под объектами понимают и переменные объектного типа – их называют **экземплярами.**

Как всякая переменная, экземпляр имеет имя и тип: их надо объявить.

…….{объявление объектного типа и описание его методов}

var v 1: tPredoc ; {объявление экземпляра объекта}

begin

v1. Name := ‘Петров Николай Иванович’;

v1.Declaration;

v1.MyName

end.

Использование поля данных объекта v1 не отличается по своему синтаксису от использования полей записей. Вызов методов экземпляра объекта означает, что указанный метод вызывается с данными объекта v1.

В результате на экран будут выведены строчки

Я – предок!

Я – Петров Николай Иванович

Аналогично записям, к полям переменных объектного типа разрешается обращаться как с помощью уточненных идентификаторов, так и с помощью оператора with .

Например, в тексте программы вместо операторов

```
v1.Name := ‘Петров Николай Иванович’;
v1.Declaration ;
v1.MyName
```

возможно использование оператора with такого вида

```
with v1 do
begin
 Name:= ‘Петров Николай Иванович’;
Declaration ;
MyName
End ;
```

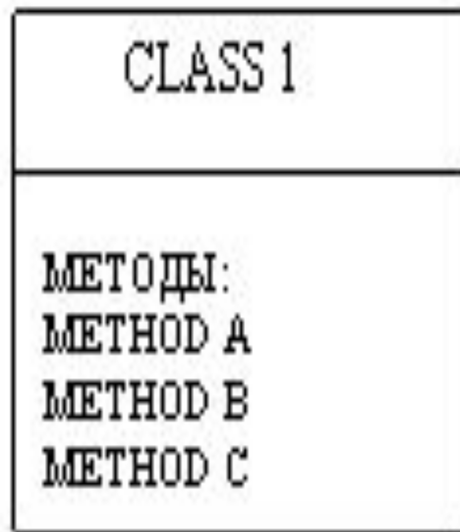
Более того, применение оператора **with** с объектными типами, также как и для записей не только возможно, но и рекомендуется.

Иерархия типов (наследование)

Типы можно выстроить в иерархию. Объект может наследовать компоненты из другого объектного типа. Наследующий объект — это потомок. Объект, которому наследуют — предок. Подчеркнем, что наследование относится только к типам, но не к экземплярам объектов.

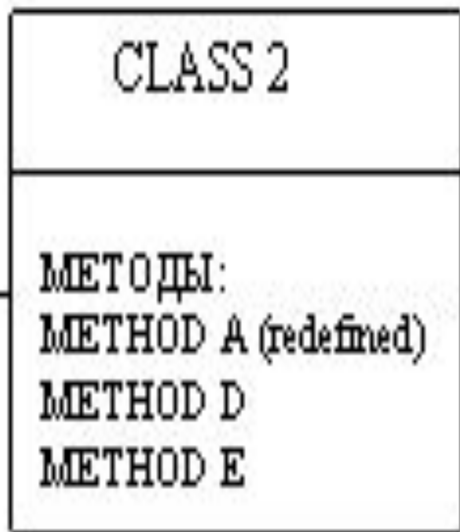
Если введен объектный тип (предок, родительский), а его надо дополнить полями или методами, то вводится новый тип, объявляется наследником (потомком, дочерним типом) первого и описываются только новые поля и методы. Потомок содержит все поля типа предка. Заметим, что поля и методы предка доступны потомку без специальных указаний. Если в описании потомка повторяются имена полей или методов предка, то новые описания переопределяют поля и методы предка.

Базовый класс для CLASS2



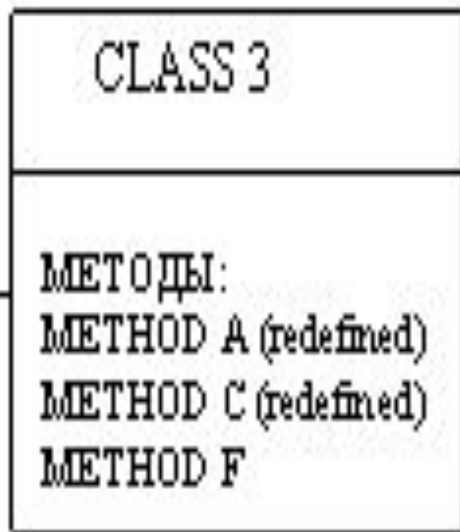
Методы: A,B,C

Производный от CLASS1 Базовый для CLASS3



Методы:
B,C из CLASS1
A,D,E из CLASS2

Производный от CLASS2



Методы:
B из CLASS1
D,E из CLASS2
A,C,F из CLASS3

ООП всегда начинается с базового класса. Это шаблон для базового объекта. Следующим этапом является определение нового класса, который называется производным и является расширением базового.

Производный класс может включать дополнительные методы, которые не существуют в базовом классе. Он может переопределять (redefined) методы (или даже удалять их целиком).

В производном классе не должны переопределяться все методы базового класса. Каждый новый объект наследует свойства базового класса, необходимо лишь определить те методы, которые являются новыми или были изменены. Все другие методы базового класса считаются частью из производного. Это удобно, т.к. когда метод изменяется в базовом классе, он автоматически изменяется во всех производных классах.

Процесс наследования может быть продолжен. Класс, который произведен от базового, может сам стать базовым для других производных классов. Таким образом, ОО программы создают иерархию классов.

Наследование дочерними типами информационных полей и методов их родительских типов выполняется по следующим правилам.

Правило 1. Информационные поля и методы родительского типа наследуются всеми его дочерними типами независимо от числа промежуточных уровней иерархии.

Правило 2. Доступ к полям и методам родительских типов в рамках описания любых дочерних типов выполняется так, как будто бы они описаны в самом дочернем типе.

Правило 3. Ни в одном дочернем типе не могут быть использованы идентификаторы полей родительских типов.

Правило 4. Дочерний тип может доопределить произвольное число собственных методов и информационных полей.

Правило 5. Любое изменение текста в родительском методе автоматически оказывает влияние на все методы порожденных дочерних типов, которые его вызывают.

Правило 6. В противоположность информационным полям идентификаторы методов в дочерних типах могут совпадать с именами методов в родительских типах. В этом случае говорят, что дочерний метод перекрывает (подавляет) одноименный родительский метод. В рамках дочернего типа, при указании имени такого метода, будет вызываться именно дочерний метод, а не родительский.

В дополнение к введенному нами типу предка tPredoc можно ввести типы потомков:

type tSon= object(tPredoc) {Тип, наследующий tPredoc }

procedure Declaration; {перекрытие методов предка}

procedure My Name(Predoc : tPredoc);

end ;

type tGrandSon=object(tSon) {Тип, наследующий tSon}

procedure Declaration ; {перекрытие методов предка}

end ;

Имя типа предка приводится в скобках после слова **object.**

Мы породили наследственную иерархию из трех типов: tSon («сын») наследник типу tPredoc, а тип tGrandSon (“внук”) типу tSon. Тип tSon переопределяет методы Declaration и MyName, но наследует поле Name.

Тип tGrandSon переопределяет только метод Declaration и наследует от общего предка поле Name, а от своего непосредственного предка (типа tSon) переопределенный метод Declaration .

Давайте разберемся, что именно мы хотим изменить в родительских методах. Дело в том, что «сын» должен провозглашать несколько иначе, чем его предок, а именно сообщить ‘Я – отец!’

```
procedure tSon.Declaration ; {реализация методов объектов — потомков}
begin
writeln (' Я — отец !');
end;
```

А называя свое имя, “сын” должен сообщить следующие сведения:
Я <фамилия имя отчество >
Я – сын <фамилия имя отчество своего предка>

```
procedure tSon .My Name ( predoc : tPredoc );
begin
inherited My Name ; {вызов метода непосредственного предка}
writeln ('Я — сын ', predoc.Name, ‘ а ’ );
End;
```

В нашем примере потомок tSon из метода MyName вызывает одноименный метод непосредственного предка типа tPredoc.

Такой вызов обеспечивается директивой **inherited**, после которой указан вызываемый метод непосредственного предка.

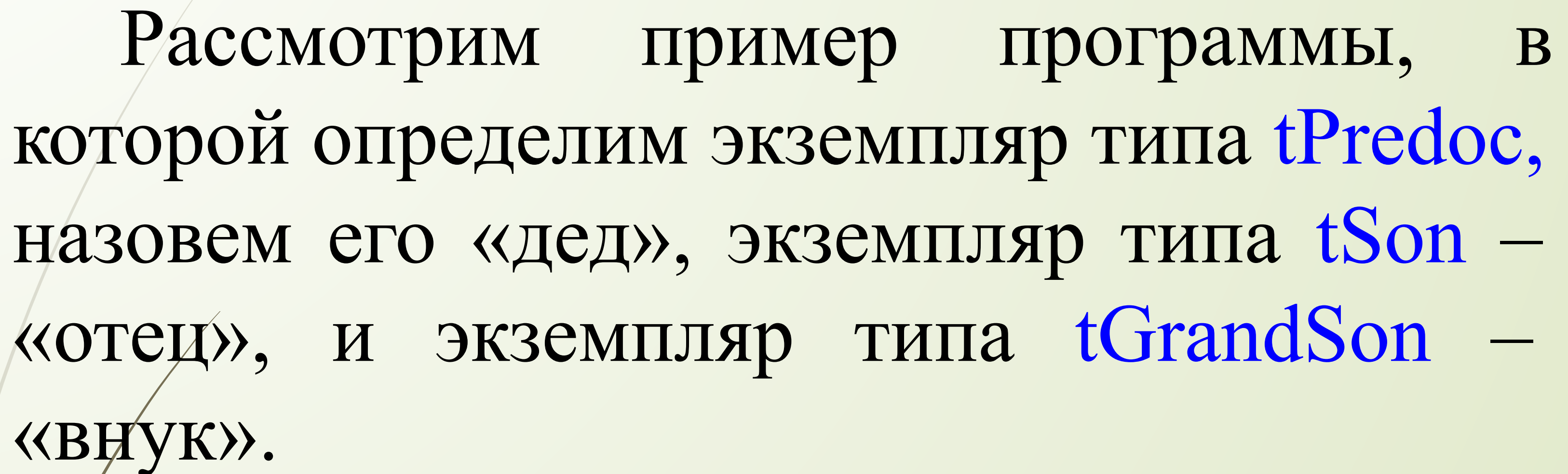
Если возникает необходимость вызвать метод отдаленного предка в каком-нибудь дочернем типе на любом уровне иерархии, то это можно сделать с помощью уточненного идентификатора, т.е. указать явно имя типа родительского объекта и через точку – имя его метода:

TPredoc.MyName;

Теперь давайте разберемся с «внуком». Метод, в котором «внук» называет свое имя, в точности такой же, как и у его непосредственного предка (типа tSon), поэтому нет необходимости этот метод переопределять, этот метод лучше автоматически наследовать и пользоваться им как своим собственным.

А вот в методе Declaration нужно провозгласить ‘Я – внук!’, поэтому метод придется переопределить.

```
Procedure tGrandSon.Declaration;
begin
writeln (' Я — внук !');
End;
```



Рассмотрим пример программы, в которой определим экземпляр типа tPredoc, назовем его «дед», экземпляр типа tSon – «отец», и экземпляр типа tGrandSon – «внук».

Потребуем от них, чтобы они представились.

Пример программы с использованием ООП

{заголовок программы}

…………………

{раздел описания типов, в том числе и объектных типов tPredoc , tSon , tGrandSon}

{Обратите внимание! Экземпляры объектных типов можно описать как типизированные константы, что мы для примера и сделали ниже}

const ded : tPredoc = (Name : 'Петров Николай Иванович');

otec : tSon = (Name : 'Петров Сергей Николаевич');

vnuk : tGrandSon = (Name : 'Петров Олег Сергеевич');

{раздел описания процедур и функций, где обязательно должны быть написаны все объявленные в объектных типах методы}

```
begin
ded.Declaration; {вызов методов общего предка}
ded.My Name;
writeln;
otec.Declaration;
otec.MyName(ded); { вызов методов объекта otec типа tSon}
writeln;
vnuk.Declaration; { вызов методов объекта vnuk типа tGrandSon}
vnuk.MyName (otec);
end .
```

Наша программа выведет на экран:

Пример вывода на экран результата

Я —предок!
Я —Петров Николай Иванович
Я —отец!
Я —Петров Сергей Николаевич
Я —сын Петров Николай Ивановича
Я —внук!
Я —Петров Олег Сергеевич
Я —сын Петров Сергей Николаевича

Обратите внимание, что в заголовке процедуры tSon.

MyName в качестве параметра приведен тип данных tPredoc, а при использовании этой процедуры ей передаются переменные как типа tPredoc, так и типа tSon.

Это возможно, так как предок совместим по типу со своими потомками. Обратное несправедливо.

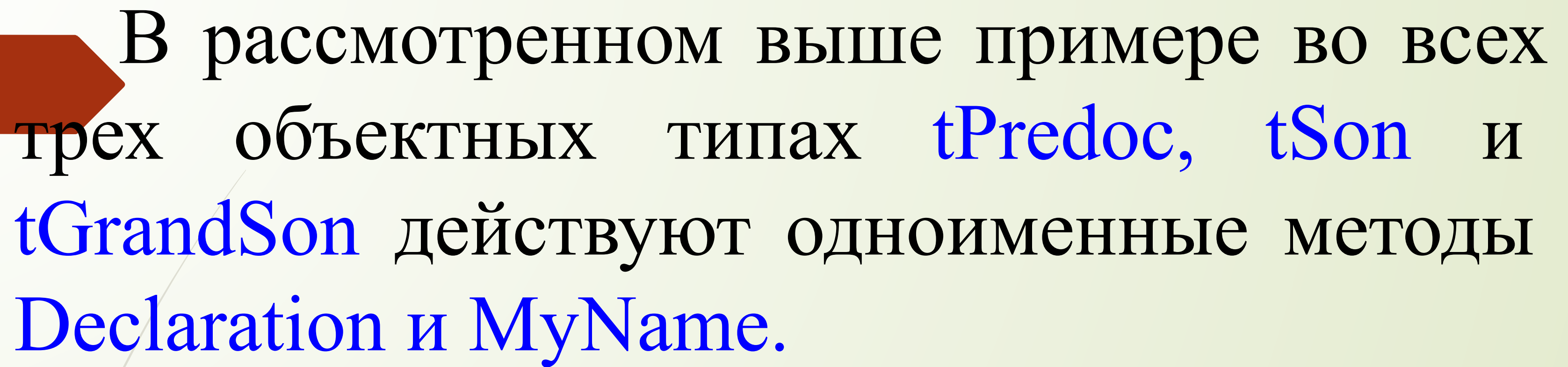
Если мы заменим в заголовке процедуры tSon. MyName при описании параметров тип tPredoc на tSon, компилятор укажет на несовместимость типов при использовании переменной ded в строке otec. MyName (ded)

Полиморфизм и виртуальные методы

Полиморфизм – это свойство родственных объектов (т.е. объектов, имеющих одного родителя) решать схожие по смыслу проблемы разными способами.

Два или более класса, которые являются производными одного и того же базового класса, называются полиморфными.

Это означает, что они могут иметь общие характеристики, но так же обладать собственными свойствами.



В рассмотренном выше примере во всех трех объектных типах tPredoc, tSon и tGrandSon действуют одноименные методы Declaration и MyName.

Но в объектном типе tSon метод MyName выполняется несколько иначе, чем у его предка.

А все три одноименных метода Declaration для каждого объекта выполняются по своему.

Методы объектов бывают статическими, виртуальными и динамическими.

Статические методы включаются в код программы при компиляции. Это означает, что до использования программы определено, какая процедура будет вызвана в данной точке. Компилятор определяет, какого типа объект используется при данном вызове, и подставляет метод этого объекта.

Объекты разных типов могут иметь одноименные статические методы. В этом случае нужный метод определяется по типу экземпляра объекта.

Виртуальные методы

в отличие от статических, подключаются к основному коду на этапе выполнения программы.

Виртуальные методы дают возможность определить тип и конкретизировать экземпляр объекта в процессе исполнения, а затем вызвать методы этого объекта.

Описание виртуального метода отличается от описания обычного метода добавлением после заголовка метода служебного слова **virtual**.

procedure Method (список параметров); virtual;

Использование виртуальных методов в иерархии типов объектов имеет определенные ограничения:

- если метод объявлен как виртуальный, то в типе потомка его нельзя перекрыть статическим методом;
- объекты, имеющие виртуальные методы, инициализируются специальными процедурами, которые, в сущности, также являются виртуальными и носят название **constructor**;
- списки переменных, типы функций в заголовках перекрывающих друг друга виртуальных процедур и функций должны совпадать полностью;

Конструктор – это специальный метод, который инициализирует объект, содержащий виртуальные методы.

Заголовок конструктора выглядит так:

constructor Method (список параметров);

Зарезервированное слово **constructor** заменяет слова procedure и virtual .

Основное и особенное назначение конструктора – установление связей с таблицей виртуальных методов (VMT) – структурой, содержащей ссылки на виртуальные методы.

Таким образом, конструктор инициализирует объект установкой связи между объектом и VMT с адресами кодов виртуальных методов. При инициализации и происходит позднее связывание.

Упомянув о конструкторе, следует сказать и о**деструкторе**. Его роль противоположна: выполнить действия, завершающие работу с объектом, закрыть все файлы, очистить динамическую память, очистить экран и т.д.

Заголовок деструктора выглядит таким образом:

destructor Done;

Основное назначение деструкторов – уничтожение VMT данного объекта. Часто деструктор не выполняет других действий и представляет собой пустую процедуру.

```
destructor Done ;
begin
End;
```