Объектно-ориентированное программирование в Pascal

Содержание

Pascal – один из самых известных существующих языков программирования, изучаемых в школьной программе. Это – «база», с которой рекомендуется начинать всем, кто заинтересован в разработке программного обеспечения.

Паскаль относится к нескольким классам одновременно. Он выступает в виде императивного, структурированного и компилируемого ЯП. Появился в 1970 году, но по сей день не забывается.

На Pascal можно программировать, используя несколько парадигм. Одна из них – это ООП. Такой подход к написанию контента обрел огромную популярность у разработчиков. В данной статье будет рассказано более подробно об объектно-ориентированном программировании, а также об объектах и классах Паскаля.

Ключевые термины

Но перед тем, как углубленно рассматривать упомянутые компоненты и концепцию создания софта, стоит выучить базовые термины. Без них продвинуться в выбранном направлении не получится.

Каждый разработчик, планирующий использовать в своей деятельности парадигму ООП, должен запомнить следующие понятия:

1. Алгоритм – инструкции и правила, которые помогают решать поставленную изначально перед программистом задачу.
2. Аргументы – значения, передаваемые в функции или существующие команды.
3. Символ – единица отображения информации. Выражается одной буквенной или символьной записи.
4. Объект (Object) – комбинация связанных переменных, констант, структур информации, которые могут выбираться и обрабатываться совместно. Позже этому компоненту будет уделено больше внимания.
5. Класс – набор связанных между собой объектов, к которых есть какие-либо общие свойства.
6. Код – письменный набор инструкций, который написан с использованием правил выбранного языка разработки. Помогает формировать итоговое приложение.
7. Константа – значение, которое постоянно остается неизменной. Корректировке оно не подлежит даже при выполнении тех или иных команд в приложении.
8. Тип данных – классификация данных определенного типа.
9. Массив – группы, а также списки схожих типов значений. Они предварительно проходят процедуру группировки.
10. Фреймворк – готовые блоки кода. Необходимы для того, чтобы упрощать процедуру программирования.
11. Петля – носит название «цикл». Последовательность инструкций, которая выполняется в приложении раз за разом. Происходит это до тех пор, пока система не получит команду на остановку. Прекращение работы цикла также осуществляется, если достигнуто заранее установленное условие.
12. Итерация – один проход через тот или иной набор операций, работающих с исходным кодом утилиты.
13. Ключевое слово – специальные слова, которые зарезервированы синтаксисом языка. Нужны для того, чтобы обозначать определенные инструменты, функции и задачи.
14. Операнд – объект, которым через операторы осуществляется манипулирование.
15. Оператор – объект, умеющий управлять операндами.

16. Переменная – место хранения временных данных в программе. Они могут корректироваться, сохраняться, а также отображаться на экране при необходимости.
17. Указатели – переменные, содержащие адреса мест в памяти.

Это – «база», с которой предстоит познакомиться всем будущим разработчикам. Далее внимание будет сфокусировано на объектах и классах в Pascal. А еще – на особенностях объектно-ориентированного программирования.

ООП – это…

Объектно-ориентированное программирование – подход к написанию контента как к моделированию информационных объектов. На основном уровне здесь будет решаться ключевая задача структурной разработки: структурирование. Этот прием позволяет улучшить управляемость самим процессом моделирования. Как следствие – способствует более грамотному и простому подходу к реализации крупных проектов.

В основе ООП заложено программирование «через объекты». Здесь предполагается минимизация избыточных данных и их целостность. Парадигма базируется на образах и разнообразных элементах. Логика и соответствующие связи здесь второстепенны.

Абстрактный тип данных

Абстрактный тип данных в разработке программного обеспечения – одно из ключевых понятий. Абстракция предусматривает разделение и независимое рассмотрение интерфейсной составляющей и непосредственной реализации.

Абстрактный тип данных – совокупность данных вместе с множеством операций, который можно выполнить над соответствующей информации. Пример – просмотр телевизора:

1. Пусть телевизор носит название модуля или объекта.
2. Соответствующий объект обладает пользовательским интерфейсом (совокупностью кнопок), а также воспроизведением изображения и звука.

3. Переключение программ осуществляется за счет нажатия на определенные «клавиши». О физических процессах, происходящих при смене канала, человек не задумывается. Об этом хорошо осведомлены специалисты.
4. При выборе телевизора покупателя будут интересовать параметры объекта: стоимость, качество «картинки», звук. То, что находится непосредственно внутри, особой роли для обычного клиента не играет.

Далее пример можно расширить. Кто-то считает, что хорошо разбирается в устройстве техники. Он начинает «модернизировать» его. Это иногда приводит к локальным (промежуточным) успехам, но конечный результат почти всегда оказывается провальным. Из-за этого «потенциально нежелательные действия» приходится запрещать. В программировании это происходит за счет запрета доступа или скрытия внутренних элементов. Каждый объект получает право самостоятельного распоряжения функциями и операциями. Игнорирование соответствующего принципа приводит к нарушению всей системы. Часто влечет за собой полное разрушение приложения. Принцип абстракции обязывает применять на деле механизмы скрытия, направленные на предотвращение корректировки внутренних компонентов случайным образом.

Об объектах и классах

В разработке базовыми блоками, если говорить об ООП, выступают объекты и классы. Первый «компонент»:

1. Может содержать нечто ощущаемое или воображаемое. То, что обладает хорошо улавливаемым поведением.
2. Объект удается потрогать или увидеть.
3. Представлен осязаемой сущностью, которая четко проявляет собственное поведение.

Object – часть окружающей человека реальности. Он существует во времени и пространстве. Формальное определение дать здесь трудно. Это можно сделать через определенные свойства. Objects имеют состояние, поведение, могут быть идентифицированы в системе однозначно (обладают уникальными именами).

Класс – множество объектов, обладающих общей структурой и поведением. Класс – описание (или абстракция), которая демонстрирует

то, как построить уже существующую во времени и пространстве переменную соответствующего класса (object).

В классе описывается поведение «компонентов» приложения. Тут же прописывается положенная структура. Стоит обратить внимание на несколько ключевых определений:

1. Состояние объекта. Объединяет все его поля данных и текущие значения каждого поля.
2. Поведение – отражает динамику корректировки состояний «элемента» и его реакцию на поступающие сообщения. То, как ведет себя «компонент», а также как взаимодействует с остальными составляющими кода.
3. Идентификация – это распознавание. Свойство, позволяющее определить элемент из числа других компонентов или в пределах того же класса. Происходит при помощи уникального имени (паспорта), которое устанавливается написанным контентом.

Объектно-ориентированная парадигма, реализовываемая в Паскале, позволяет представить приложения в виде набора определенных компонентов, которые взаимодействуют друг с другом. А классы помогают составлять описания соответствующих «элементов».

Принципы

В объектно-ориентированном программировании выделяют несколько ключевых принципов структурирования. Обратить внимание стоит на:

- проектирование – инкапсуляцию;
- наследование;
- полиморфизм;
- передачу сообщений.

Все это требует в Паскале и иных ЯП, базирующихся на объектно-ориентированной разработке, более детального изучения.

Инкапсуляция

Инкапсуляция – это соединение в одном объекте данных и функций, которые управляют соответствующими материалами. Доступ к некоторым

электронным сведениям в пределах пакета может оказаться под запретом. Еще один вариант – ограничение.

Объект – это совокупность всех своих свойств, а также их текущих значений. Также характеризуется связью допустимых для задействованного объекта действий. Объединение в единое целое в одном «компоненте» как «материального», так и возможных манипуляций – это и есть инкапсуляция.

В пределах ООП различают поля и методы. Первый компонент – это данные, второй – действующие алгоритмы.

За счет инкапсуляции удается максимально изолировать объект от внешнего окружения. Она позволяет повысить надежность создаваемого программного обеспечения. Связано это с тем, что локализованные в объекте алгоритмы будут обмениваться электронными материалами с контентом относительно небольшими объемами. Результат – замена или корректировка алгоритмов и данных, инкапсулированных в object, не влечет за собой негативных последствий для исходного кода. Инкапсуляция обеспечивает простоту обмена объектами, перенос их из одного софта в другой.

Наследование

Методология ООП предусматривает построение древа иерархии, которое будет отражать взаимосвязи между составляющими (подзадачами) приложения. Здесь осуществляется наследование свойств родительских (вышележащих) типов объектов дочерними (нижележащими).

Наследование – отношение между objects, когда один компонент будет повторять структуру и поведение другого. В жизни данная составляющая встречается повсеместно. Пример – млекопитающие и птицы. Они наследуют признаки живых организмов.

У родителей обычно нет конкретного экземпляра объекта. Пример – нет живого организма, который бы сам по себе носил имя «птица» или «млекопитающее». Подобные типы носят название абстрактных.

Конкретные экземпляры есть у нижних уровней иерархии в ООП. Пример – это «Крокодил Гена». Он относится к классу «крокодилы». Еще один вариант – известный из мультфильмов Матроскин. Его можно отнести к классу «кошачьи».

Наследование позволяет:

1. Применять библиотеки классов. Допускается их дальнейшее развитие в конкретной программе.
2. Создавать new objects путем изменения или дополнения свойства прежних. Наследник получит все поля и методы родителя, но может предусматривать и собственные.
3. Решить проблемы модификации свойств.
4. Придает гибкость всей парадигме ООП.

При построении нового класса, наследуя его из уже существующего класса, можно:

- добавить в новый класс новые компоненты-информацию;
- добавить в новый класс новые функции;
- осуществить замену в новом классе наследуемые из старого компоненты-функции.

Все это значительно упрощает процедуру программирования.

Полиморфизм

Полиморфизм даем возможность использования одних и тех же функций для решения тех или иных задач. Выражен в том, что под одним и тем же именем скрываются разного рода действия. Их содержание напрямую зависит от типа object.

Полиморфизм – свойство родственных «элементов» решать схожие по смыслу проблемы посредством самых разных способов. Пример – «бежать» могут почти все животные. Но слон, лев и черепаха делают это по-разному.

При ООП поведенческие свойства базируются на наборе входящих методов и описания, которое когда-либо соответствующий класс выполнял. Программист сможет таким образом давать потомкам свойства и характеристики, которые отсутствовали у «родителей». Для изменения метода требуется перекрыть его в наследнике (объявить одноименный метод), а затем реализовать необходимые манипуляции.

В приведенном примере с бегом действие «бежать» — это полиморфическое действие. Многообразие форм его проявления – непосредственный полиморфизм.

Объектный тип

Класс или объект – это структура данных содержащая в себе поля и методы. Она начинается при помощи ключа, а заканчивается оператором end. Формальный синтаксис достаточно простой – описание объектного типа осуществляется, если в описании заменить record на object или класс (class). Далее – требуется добавить объявление функций и процедур над полями:

```
Type <имя типа объекта>= object
   <поле>;
   <поле>;
   ….
   <метод>;
   <метод>;
end ;
```

В ObjectPascal есть ключевое слово class. Оно помогает описывать objects, которые заимствованы из C++:

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

У ObjectPascal существуют две модели, помогающие описывать objects. Компонент объекта – поле или метод. Поле включает в себя имя и тип данных. Метод – процедура или функция, объявленная в пределах декларации объектного типа. Сюда можно отнести особые процедуры при помощи которых создаются и уничтожаются «компоненты» программы.

Объявление метода внутри описания объектного типа включает в себя только заголовок. Это – предварительное описание подпрограммы. Тело метода приводится после объявленного объектного типа:

```
Type tPredoc = object Name : string ; {поле данных объекта}
   Procedure Declaration ; {объявление методов объекта}
   Procedure MyName ;
End ;
```

Стоит запомнить следующее:

1. Тексты подпрограмм, отвечающих за реализацию методов объекта, приводятся в разделе описания процедур и функций.
2. Заголовки при описании реализации метода будут повторять заголовки, которые приведены в описании типа. Они дополняются именем object, которое отделяется точкой.

```
Procedure tPredoc.Declaration ; {реализация метода объекта}
begin
   writeln ('Я - предок!');
end ;
Procedure tPredoc.MyName ; {реализация метода объекта}
begin
   writeln('Я -', Name);
end;
```

3. В пределах описания методов на поля и методы соответствующего типа будет происходить ссылка просто по имени. Метод MyName из примера использует поле Name без явного указания его принадлежности к «компоненту» так, как если бы применялся неявный оператор with…do.
4. Под objects можно понимать и переменные объектного типа. Они носят название экземпляров. Каждая переменная и экземпляр имеет имя и тип. Они предусматривают предварительное объявление, как и класс.

```
……{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.
```

5. При использовании поля данных «элемента» v1 будут применяться те же алгоритмы, что и в случае с работой с полями записи. Вызов методов экземпляра указывает на то, что соответствующий метод вызывается данными object v1. На экране появится такой результат:

```
Я - предок!
Я - Петров Николай Иванович
```

.

К полям переменных объектного типа можно обращаться при помощи уточненных идентификаторов. Еще один вариант – применение оператора with.

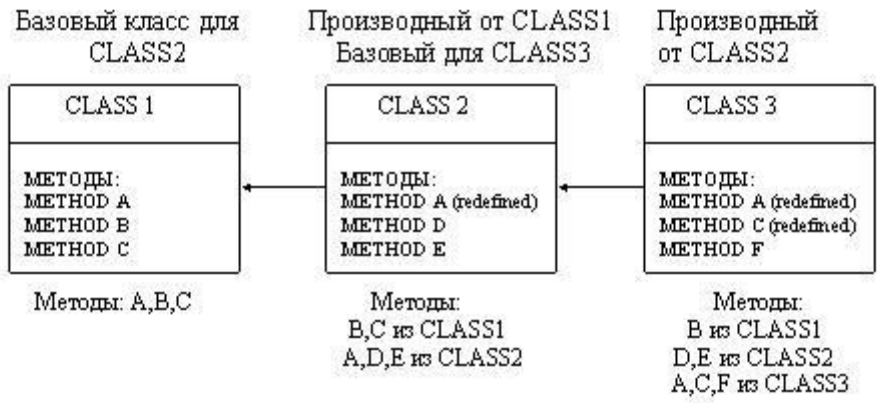
```
with v1 do
begin
   Name:= 'Петров Николай Иванович';
   Declaration ;
   MyName
End ;

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

Иерархия типов

Типы предусматривают построение в иерархию. Объект способен наследовать компоненты из другого объектного типа. Наследующий object – это потоком. Объект, которому наследуют – предок или родитель. Стоит обратить внимание на то, что наследование относится только к типам. К экземплярам никакого отношения не имеет.

Если был введен объектный тип (родительский), но его нужно дополнить молями или методами, нужно ввести новый тип. Он объявляется в качестве наследника (дочернего типа). Предусматривает описание полей и методов, которые считаются новыми.

Потомок включает в себя все поля типа предка (особенности класса). Поля и методы родителя наследнику доступны без специальных инструкций. Если в описании дочернего класса повторяются имена «родителя», то новые описания будут переопределять поля и методы «наследодателя».



Объектно-ориентированное программирование всегда начинается с базового класса. Это – шаблон для базового объекта. Следующий этап – определение нового класса, который носит название производного. Он выступает расширением «базы».

Производный класс может включать в себя дополнительные методы, отсутствующие у базового. Он иногда переопределяет методы или даже полностью удаляет их. Все методы базового класса в производном не переопределяются. Сюда записывается лишь то, что является новым или откорректированным.

Наследование может продолжаться. Класс, который произведен от базового, может выступать в виде «базы» для других производных. Так программы образовывают иерархии классов:

Здесь стоит запомнить следующие правила:

1. Информационные поля и методы родителя наследуются всеми его дочерними типами. Число промежуточных уровней иерархии не важно.
2. Доступ к полям и методам родительских типов в рамках описания наследников выполняет так, словно они описаны в дочерних типах.
3. Идентификаторы родительского класса не могут встречаться в дочерних.
4. Наследники могут доопределять произвольное количество собственных методов. Сюда же относят информационные поля.
5. Любые корректировки текста в родительских методах – это автоматическое влияние на все методы у наследников.

Также важно, что в дочерних типах идентификаторы методов могут совпадать с именами методов у «родителей». Тогда целесообразно говорить о том, что наследник перекрывает (подавляет) одноименный родительский метод.

Полиморфизм и виртуальные методы

Говоря о классах и objects в Паскале, стоит обратить внимание на полиморфизм и виртуальные methods. Полиморфизм – свойство родственных объектов (произошедших от одного и того же родителя) решать схожие по смыслу проблемы разного рода способами.

Два и более класса, выступающие в виде производных одного и того же базового класса, носят название полиморфными. Это значит, что они могут обладать общими характеристиками. А вот свойства у таких классов различаются.

В ООП поведенческие свойства объекта определяются набором включенных в его пределы методов. При помощи корректировки алгоритмов программист сможет придавать потомкам отсутствующие у родителя новые свойства.

Статический «подход»

Methods в программировании бывают статическими. Они включены в исходный код приложения в момент компиляции. Это указывает на то, что еще до начала работы приложения определено, какая процедура вызывается к заданной точке. Компилятор определит, какого типа объект задействован, а затем подставит соответствующий method.

Объекты разных типов могут иметь одноименные статические «приемы». Тогда method будет определен при помощи типа экземпляра объекта. Статическое перекрытие – первый шаг полиморфизма. Но одинаковые имена являются лишь удобством в разработке, а не ключевым принципом.

Виртуальные методы

Подключаются в момент выполнения программы. Позволяют определять тип и конкретизировать экземпляр объекта во время исполнения. Далее – вызывать методы этого самого объекта.

Данный механизм является принципиально новым. Он носит название позднего связывания. Отвечает за обеспечение полиморфизма – разного способа поведения для разных, но однородных в плане наследования объектов.

Для того, чтобы использовать виртуальный метод, нужно использовать ключевое слово virtual:

```
procedure Method ( список параметров ); virtual;
```

Иерархия типов объектов предусматривает определенные ограничения, связанные с virtual methods:

- в типе потомка не получится осуществить перекрытие при помощи статического «подхода»;
- объекты, которые работают с виртуальными методами, будут инициализироваться специальными процедурами – конструкторами (constructors);
- списки переменных, типы функций в заголовках перекрывающих друг друга виртуальных процедур, а также функции должны полностью совпадать.

Конструктор обычно выполняет роль инициализатора объектного экземпляра. А еще – связывает virtual methods.

Справка PascalABC.NET
https://pascalabc.net/downloads/pabcnethelp/index.htm?page=LangGuide/Classes/constrdestr.html