Конспект №1

**Интерфейс Delphi 7**

**Понятие интерфейса**

Из предыдущих глав вы уже знаете, что собой представляет объект. Представьте, что получится, если из объекта убрать поля и код всех методов. Останется лишь интерфейс - заголовки методов и описания свойств. Схематично понятие интерфейса можно представить в виде формулы:

Интерфейс = Объект - Реализация

В отличие от объекта интерфейс сам ничего "не помнит" и ничего "не умеет делать"; он является всего лишь "разъемом" для работы с объектом. Объект может поддерживать много интерфейсов и выступать в разных ролях в зависимости от того, через какой интерфейс вы его используете. Совершенно различные по структуре объекты, поддерживающие один и тот же интерфейс, являются взаимозаменяемыми. Не важно, есть у объектов общий предок или нет. В данном случае интерфейс служит их дополнительным общим предком.

**Описание интерфейса**

В языке Delphi интерфейсы описываются в секции type глобального блока. Описание начинается с ключевого слова interface и заканчивается ключевым словом end. По форме объявления интерфейсы похожи на обычные классы, но в отличие от классов:

интерфейсы не могут содержать поля;

интерфейсы не могут содержать конструкторы и деструкторы;

все атрибуты интерфейсов являются общедоступными (public);

все методы интерфейсов являются абстрактными (virtual, abstract).

Приведем пример интерфейса и сразу заметим, что интерфейсам принято давать имена, начинающиеся с буквы I (от англ. Interface):

|  |
| --- |
| type  ITextReader = interface  // Методы  function NextLine: Boolean;  // Свойства  property Active: Boolean;  property ItemCount: Integer;  property Items[Index: Integer]: string;  property EndOfFile: Boolean;  end; |

Интерфейс ITextReader предназначен для считывания табличных данных из текстовых источников. В главе 3 мы уже создавали объекты, которые умеют это делать, поэтому назначение методов и свойств должно быть вам понятно. Непонятно пока другое - зачем вообще нужен интерфейс для доступа к табличным данным, если уже есть готовый класс TTextReader с требуемой функциональностью.

Объяснение состоит в следующем. Не определив интерфейс ITextReader, невозможно разместить класс TTextReader в DLL-библиотеке и обеспечить доступ к нему из EXE-программы. Создавая DLL-библиотеку, мы с помощью оператора uses должны включить модуль ReadersUnit в проект библиотеки. Создавая EXE-программу, мы должны включить модуль ReadersUnit и в нее, чтобы воспользоваться описанием класса TTextReader. Но тогда весь программный код класса попадет внутрь EXE-файла, а это именно то, от чего мы хотим избавиться. Решение проблемы обеспечивается введением понятия интерфейса.

Чтобы вам было легче разобраться с интерфейсом ITextReader, мы привели его незаконченный вариант. Компиляция интерфейса в таком виде приведет к ошибкам: для свойств не указаны методы чтения и записи. Полное описание интерфейса выглядит так:

|  |
| --- |
| type  ITextReader = interface  // Методы  function NextLine: Boolean;  procedure SetActive(const Active: Boolean);  function GetActive: Boolean;  function GetItemCount: Integer;  function GetItem(Index: Integer): string;  function GetEndOfFile: Boolean;  // Свойства  property Active: Boolean read GetActive write SetActive;  property Items[Index: Integer]: string read GetItem; default;  property ItemCount: Integer read GetItemCount;  property EndOfFile: Boolean read GetEndOfFile;  end; |

Поскольку интерфейс не может содержать поля, все его свойства отображены на его методы.

Конспект №2

**Формы записи алгоритма**

**Словесный способ записи алгоритма**

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например. Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел (алгоритм Эвклида).

Словесный способ не имеет широкого распространения, так как такие описания:

строго не формализуемы;

страдают многословностью записей;

допускают неоднозначность толкования отдельных предписаний.

2. Наибольшее распространение благодаря своей наглядности получил графический способ записи алгоритмов. При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. В таблице приведены наиболее часто употребляемые символы.



Блок "процесс" применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок "решение" используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок "модификация" используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок "предопределенный процесс" используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

**Псевдокод.**

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.

Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же, как и в формальных языках, есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

**Литература.**

[**http://www.interface.ru/home.asp?artId=5859**](http://www.interface.ru/home.asp?artId=5859)

[**https://cs.petrsu.ru/studies/filatova\_information/CMD\_1996566\_M/my\_files/Inform/Algorithm/al-3.htm**](https://cs.petrsu.ru/studies/filatova_information/CMD_1996566_M/my_files/Inform/Algorithm/al-3.htm)