## §9. ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

### 9.1. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Как уже отмечалось выше (п. 4.1), в настоящее время растет популярность методологий, ориентированных на данные. В первую очередь, это объектно-ориентированное программирование.

Объектно-ориентированная методология проектирования программ основана на концепциях упрятывания информации и абстрактных типов данных. Такой подход рассматривает все такие ресурсы как данные, модули и системы в качестве объектов. Каждый *объект* содержит некоторую структуру данных (или тип данных), обрамленную набором процедур (методов), предназначенных для манипулирования этими данными. Используя эту методологию, программист может создать свой собственный абстрактный тип и отобразить проблемную область в эти созданные им абстракции вместо традиционного отображения проблемной области в предопределенные управляющие структуры и структуры данных языка программирования. Подобный подход является более естественным, чем методологии, ориентированные на обработку (на процесс), из-за возможности использовать в процессе программирования разнообразные виды абстракции типов данных. На этом пути программист может сконцентрироваться на проекте системы, не беспокоясь о деталях информационных объектов, используемых в системе.

Основные шаги разработки программы, предусмотренные даннойметодологией:

• определить проблему;

• развить неформальную стратегию, представляющую общую последовательность шагов, удовлетворяющую требованиям к будущей программе;

• формализовать стратегию

• идентифицировать объекты иих атрибуты;

• идентифицировать операции;

• установить интерфейсы;

• реализовать операции.

Большинство современных языков и систем программирования развивается в направлении все большего использования объектной методологии в создании программ. Характерными примерами являются универсальные языки Паскаль, СИ и даже Бейсик, в современных версиях которых появились средства объектно-ориентированного программирования. Так, начиная с версии 5.5, Турбо-Паскаль охватывает метод проектирования программ на основе объектно-ориентированного программирования.

### 9.2. ОСНОВЫ ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ В СИСТЕМЕ ТУРБО-ПАСКАЛЬ

**Объект** в ТурбоПаскале - это структура данных, содержащая поля данных различных типов и заголовки методов и обобщающая структуру «Запись» (record).

Синтаксис описания объекта:

<ИмяПотомка>=

оbjесt<ИмяПредка> поле;

поле;

…

метод;

…

метод;

end;

В отличие от записи полями объекта могут быть, кроме данных, еще и методы, обрабатывающие эти данные

**Метод** - это процедура или функция, объявленные внутри описания объекта. Синтаксис описания метода:

procedure <Заголовок>(<Параметр1>, <Параметр2>:integer),

Метод имеет доступ к полям данных объекта, не требуя передачи их ему в виде параметров

Объявление метода внутри объявления объектного типа содержит только заголовок. Тело метода определяется вне объявления объекта. Его заголовок должен содержать имя объекта, которому принадлежит метод. Например:

procedure <ТипОбъекта.Метод>

(<Параметр1>, <Параметр2> : integer);

begin

…

…

еnd;{Метод}

Методы подразделяют на статические и виртуальные. Виртуальный метод отличается от статического тем. что реализующий его код подсоединяется к исполняемой программе не в процессе компиляции, а в процессе выполнения, что достигается, так называемым, поздним связыванием. Это дает возможность строить иерархию объектов с одинаковыми названиями методов, реализуемыми, однако, различными кодами.

Синтаксис виртуального метода:

procedure <Метод> (<Параметр1>, <Параметр2> : integer): virtual;

Кроме обычных процедур и функций, ТурбоПаскаль 6.0 реализует два специальных типа методов: конструктор и деструктор.

**Конструктор** - это специальный метод, инициализирующий объект, содержащий виртуальные методы, он объявляется специально зарезервированным словом constructor. Конструктор инициализирует объект путем установления связи между объектом и специальной таблицей виртуальных методов, содержащей адреса кодов, реализующих виртуальные методы. Конструктор может также использоваться для инициализации полей данных объекта.

**Деструктор** - это специальный метод, освобождающий память «кучи» от динамических объектов. Он объявляется с использованием специально зарезервированного слова destructor.

Основными отличительными свойствами объекта являются

• инкапсуляция - объединение записей с процедурами и функциями, работающими с этими записями;

• наследование - задание объекта, затем использование его для построения иерархии порожденных объектов с наследованием доступа каждого из порожденных объектов к коду и данным предка;

• полиморфизм - задание одногоимени действию, которое передается вверх и вниз по иерархии объектов с реализацией этого действия способом, соответствующим каждому объекту в иерархии.

Рассмотрим смысл каждого из перечисленных свойствна примере построения на экране дисплея точек разных цветов (звездного неба).

**Инкапсуляция.** Основой решения задачи является задание положения (позиции) отдельной точки на экране, описываемого координатами Х и Y. Для задания координат подходит тип «запись»:

Pozition = record

X, Y : integer;

end;

Далее может быть необходимо задать значения координат (такая процедура носит название инициализации). Создадим соответствующую процедуру:

procedure Init(CoordX, CoordY : integer);

begin

X : *=* CoordX;

Y : = CoordY;

end;

Затем потребуется знание фактических значений координат. Для этого вводим две функции:

function GetX : integer;

begin

GetX: = X;

end;

function GetY: integer;

begin

GetY: = Y;

end;

По нашему замыслу процедура Init и функции GetX и GetY должны работать только с полями записи Pozition.

Введение объектов позволяет зафиксировать это положение, объявив и поля, и действия над ними в единой конструкции:

Pozition = object

X, Y: integer;

procedure Init(CoordX, CoordY : integer);

function GetX : integer;

function GetY : integer;

end;

Процедура Init и функции GetX и GetY являются методами объекта Pozition.

Для инициализации экземпляра типа Pozition достаточно вызвать его метод, как если бы он был полем записи:

var

FirstPozition : Pozition;

…

FirstPozition.Init(10,15);

Метод задается так же, как и процедура в модуле: внутри объекта записывается заголовок (как в секции Interface модуля); при этом все поля, используемые мето-дом, должны предшествовать его объявлению. Определение метода (расшифровка действий) происходит вне объявления объекта. Имя метода должно предваряться названием типа объекта, которому метод принадлежит, сопровождаемым точкой. Например,

procedure Pozition. Init(CoordX, CoordY: integer);

begin

X: = CoordX:

Y: = CoordY;

end;

Заметим, что имена формальных параметров метода не могут совпадать с именами полей данных объекта.

Также как модуль скрывает детали реализации процедур от пользователя, объект может скрывать свои поля и методы. Для этого используется ключевое слово private (личный). Личные поля и методы доступны только внутри метода. Объявление выглядит следующим образом:

type

ObjectName=object

поле;

…

поле;

метод;

метод;

private

поле;

…

поле;

метод;

…

метод;

end;

**Наследование**. Рассмотрим точку с координатами Х и Y. Ее можно сделать видимой или невидимой, ей можно задать цвет, ее можно переместить. Создадим объект с такими возможностями:

Point=object

X,Y : integer; "

procedure Init(CoordX, CoordY : integer);

function GetX : integer;

function GetY . integer;

Visible: Boolean;

Color: Word;

procedure Init(CoordX, CoordY : integer; InitCoIor : Word);

function Is Visible : Boolean;

procedure Show;{показывает точку}

procedure Blind; {стирает точку}

procedure Jump(NextX, NextY : integer);{nepeMeiuaer точку}

end;

Заметим, однако, что поля X.Y и методы GetX, GetY практически совпадают с соответствующими полями и методами объекта Pozition.

Турбо-Паскаль предоставляет возможность учесть эту ситуацию. Следует считать тип объекта Point порожденным типом Pozition, записав это следующим образом:

Point=object(Pozition)

Visible : Boolean;

Color : Word;

procedure Init(CoordX, CoordY : integer; InitColor : Word);

function Is Visible . Boolean;

procedure Show;

procedure Blind:

procedure Jump(NextX, NextY : integer);

end;

Объект Point теперь наследует свойства объекта Pozition. Поля X,Y явно не заданы в Point, но Point ими обладает благодаря наследованию, т.е. можно написать

Point.X:=17;

Смысл объектно-ориентированного программирования заключается в работе с полями объекта через его методы.

**Полиморфизм**. Предположим, надо создать объект «кружок». Очевидно, что новый объект должен иметь предком объект Point, обладая всеми его свойствами, кроме того. быть больше по размеру. Однако, ясно, что «высветить» точку и закрашенную окружность не удастся одними и теми же командами. Турбо-Паскаль разрешает сохранить потомку имя родительского метода, «перекрывая» его. Чтобы перекрыть родительский метод, надо просто задать его с тем же именем, но с другим телом (кодом) и, если необходимо, с другим набором параметров. Такой метод делается виртуальным и к его объявлению добавляется слово virtual. Применение виртуальных методов налагает ограничения на процедуры инициализации, которые должны записываться с зарезервированным словом constructor и иметь общее имя Init

Каждый отдельный экземпляр объекта должен инициализироваться с помощью отдельного вызова конструктора

Для очистки и удаления динамически распределенных объектов существует специальная процедура destructorDone.

Деструктор комбинирует шаг освобождения памяти в «куче» с некоторыми другими задачами. Метод деструктора может быть пустыу), поскольку работу выполняет не только код тела, но и код, генерируемый Турбо-Паскалем в ответ на зарезервированное слово destructor.

### 9.3. ОБОЛОЧКА TURBO-VISION

Объектно-ориентированный подход позволяет создавать объектные среды, библиотеки объектов для последующего использования при разработке программ. Особенно широкую известность получили объектные оболочки для разработки интерфейсов программ в некотором едином, ставшем стандартным, стиле.

Turbo-Vision - это оболочка для создания оконченных интерфейсов программ в среде Турбо-Паскаль версий 6.0, 7.0.

Объектно-ориентированная библиотека Turbo-Vision включает

• многократные перекрещивающиеся окна с изменяемыми размерами;

/

• выпадающие меню;

• поддержку мыши;

• встроенную установку цвета;

• кнопки, полосы скроллинга, окна ввода, зависимые и независимые кнопки;

• стандартную обработку клавиш и нажатий мышки и многое другое.

Стандартный отработанный интерфейс, создаваемый с помощью Turbo-Vision, имеет сверху строку меню, а снизу - строку статуса.

Текст, который появляется в строке статуса, обычно выводит сообщения о текущем состоянии программы, показывая доступные горячие клавиши или подсказки для команд, которые доступны пользователю в данный момент.

Когда «выпадает» меню. полоса подсветки пробегает по списку элементов меню при движении мыши или нажатии на клавиши курсора. Нажатием клавиши <Enter> или левой кнопки мыши выбирается подсвеченный элемент. Выбранный элемент меню посылает команду в определенную часть программы.

Программа обычно взаимодействует с пользователем через одно или более окон или диалоговых окон, которые появляются и исчезают на панели экрана в соответствии с командами от мышки или клавиатуры. Turbo-Vision предоставляет большой набор окон для ввода и отображения информации. Окна можно сделать со скроллингом, что позволяет выводить в окнах большие объемы данных. Скроллинг окна по информации выполняется передвижением по полосе скроллинга внизу окна, с правой стороны окна или обеим сторонам. Полоса скроллинга указывает относительную позицию окна в отображаемых данных.

Диалоговые окна часто содержат кнопки - подсвеченные слова - которые могут активизироваться с помощью мыши (или переходом через клавишу Tab и нажатием клавиши <Пробел>). Отображаемые слова, выбранные мышью, могут быть установлены для передачи команд в программу.

Простейшая программа на Turbo-Vision выполняет следующие действия:

• заполняет панель экрана полутеневым шаблоном;

• выводит на экран полосу меню (наверху) и строку статуса (внизу);

• устанавливает обработчик для событий, инициированных от клавиш и мыши;

• строит объект меню и соединяет его с полосой меню;

• строит диалоговое окно;

• связывает диалоговое окно с меню;

• ожидает действий пользователя через мышь или клавиатуру. Краеугольным объектом любой программы является объект TApplication. TApplication - это абстрактный тип объекта. Он ничего не делает. TApplication используется для создания порожденных типов от TApplication. которые содержат программный код. Например, в простейшей демонстрационной программе HELLO.PAS, выводящей в стандартном для Turbo-Vision виде надпись HELLO, WORLD! определяется порожденный тип объекта THelloApp:

PHelloApp = ^HelloApp;

THelloApp = object(TApplication)

procedure GreetingBox;

procedure HadleEvent(var Event: TEvent); virtual;

procedure InitMenuBar; virtual;

procedure InitStatusLine; virtual;

end;

Как показано здесь, полезно определить тип указателя на каждый тип определяемого объекта, поскольку большая часть работы над объектами производится через указатели. Полиморфизм, главным образом, работает через указатели.

THelloApp содержит гораздо больше, чем четыре содержащихся в нем метода; порожденный объект наследует все от его предка. В определении THelloApp определяется, чем новый объект отличается от его предка TApplication. Все, что не переопределяется, наследуется в неизменном виде от TApplication.

Четыре метода, определенные в THelloApp, завершают приложения:

1) метод **HadleEvent** определяет, что некоторые события произошли (и указывает это) и как на эти события отвечать другим прикладным функциям; метод HadleEvent, определенный в TApplication, работает с общими событиями, которые возникают внутри любой программы, но необходимо обеспечить обработку событий, специфичных для программы;

2) метод **InitMenuBar** устанавливает меню дляполосы Меню программы (TApplication содержит полосу меню, но не сами меню);

3) метод **InitStatusLine** устанавливает текст строки статуса внизу экрана; этот текст обычно отображает сообщение о текущем состоянии программы, показывая доступные горячие клавиши или напоминая пользователю о некоторых действиях;

4) метод **GreetingBox** вызывает диалоговое окно в ответ на элемент меню Greeting; GreetingBox вызывается из метода HadleEvent в ответ на событие, переключаемое выбором элемента меню Greeting; в более сложных программах можно использовать различные методы в ответ на каждый элемент меню, определяемый в начальном меню.

На верхнем уровне абстракции главная программа на Turbo-Vision выглядит аналогично Hello и содержит три метода:

var

HelloWorld: THelloApp;

begin

HelloWorld.Init;

HelloWorld Run;

HelloWorld.Done;

end;

Каждый из этих трех методов требует небольшого пояснения.

**Init** - это необходимый вызов конструктора. Все объекты, содержащие виртуальные методы, должны быть созданы (через вызов их конструктора) до вызова любых других методов этого объекта. Все конструкторы Turbo-Vision имеют имя Init. Это удобное соглашение, которым мы рекомендуем пользоваться.

HelloWorld.Init устанавливает объекты главной программы для использования. Он очищает экран, устанавливает некоторые значения для важных переменных, заполняет панель экрана и выводит строку статуса и полосу меню. Он вызывает конструкторы многих других объектов, некоторые из которых никогда не видны, поскольку они вызываются внутри.

Почти вся работа программы в Turbo-Vision выполняется внутри метода главной программы **Run.** Определения метода Run здесь нет, поскольку Run наследуется от родительского объекта TApplication.

Этот метод состоит, главным образом, из цикла repeat...until, показанном здесь в псевдокоде;

repeat

<получить событие>;

<обработать событие>;

until Quit;

Это не точный код, а общая схема реализации Run. В сущности программа Turbo-Vision циклически выполняет две задачи: получение события (где событие - это, в сущности, «сделать что-нибудь») и обработку этого события. Одноиз событий вычисляется в некоторый вид команды выхода и цикл завершается.

Дестрактор **Done** в действительности очень прост: он уничтожает все объекты программы - полосу меню, строку статуса и панель экрана - и удаляет обработчик ошибок и драйверы Turbo-Vision. Метод Done должен отменять все действия, выполненные конструктором Init, а затем вызывать TApplication.Done, который обрабатывает все стандартные элементы.

Объектно-ориентированное программирование с использованием Turbo-Vision требует хорошего знания иерархии объектов и спецификаций их методов.

Помимо объектно-ориентированной оболочки Turbo-Vision для Турбо-Паскаля, существует аналогичная более развитая система для языка C++. Объектно-ориентированное программирование стало основой интерфейса системы Windows. Современные объектно-ориентированные системы типа Visual Basic, Delphi совершенно скрывают механизмы программирования и позволяют разрабатывать интерфейсы программ-приложений вообще без кодирования команд, пользуясь лишь средствами наглядного проектирования на экране дисплея.

### 9.4.\* СРЕДА ОБЪЕКТНОГО ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ DELPHI

\* Раздел о Delphi написан Е.А.Ереминым.

Созданный в 1970 г. Н. Виртом, язык программирования Паскаль не оставался чем-то неизменным. Хотя основная его часть сохранилась прежней, появился целый ряд усовершенствований и дополнений, которые были направлены на облегчение программирования задач все возрастающей сложности. Например, типа данных string не было в первоначальной версии языка, в то время как представить без него современный Паскаль практически невозможно.

Позднее были добавлены более сложные конструкции: Н. Вирт предложил структуру программы из модулей unit, для возможности реализации объектного программирования был введен тип object. В результате возможности Паскаля значительно возросли и он по-прежнему мог удовлетворить запросы любого самого изощренного системного программиста. Известна, например, высокая оценка языка, которую дал в одной из своих книг знаменитый американский программист Питер Нортон: «Лично я пользуюсь и языком Паскаль, и языком Си. Мои популярные сервисные программы «Утилиты Нортона» были написаны на Паскале, а затем переписаны на Си. Мне нравятся оба эти языка. Я считаю, что сам по себе Паскаль лучше, в нем меньше возможностей допустить ошибку...».

С появлением графической среды Windows положение существенно изменилось. Конечно, реализовать программу с современным оконным интерфейсом на Паскале было возможно, но для этого требовалось немалое мастерство и много времени.

Чтобы исправить это положение, в 1996 г. фирма Borland, известная своими разработками в области реализации языков программирования, выпустила компилятор нового поколения Delphi. Прежде всего, это мощный компилятор языка Паскаль. дополненного рядом существенно новых возможностей для программирования в среде Windows. Но не только. Delphi - это система, имеющая интерфейс качественно нового типа, позволяющий при составлении текста программы видеть те графические объекты, для которыхона пишется -так называемая, система визуального программирования.

Delphi является системой программирования очень высокого уровня. Она берет на себя значительную часть работы по управлению компьютером, что делает возможным в простых случаях обходиться без особых знаний о деталях ее работы.

В отличие от традиционных систем программирования, Delphi даже «сама» пишет значительную часть текста программы: описания объектов, заголовки процедур и многое другое. Программисту остается только вписать необходимые строчки, определяющие индивидуальное поведение программы, которые система не в состоянии предугадать. Но даже здесь Delphi во многих случаях сама указывает место, где надо разместить эти строки.

Вершиной автоматизации процесса программирования являются, так называемые, эксперты. Эксперт - это диалоговое окно, которое помогает пользователю описать, что он хочет видеть в своей программе. Например, *Эксперт проекта* спрашивает, необходимо ли вам системное меню и какого из предложенных типов ваш проект. Проанализировав введенные ответы. Delphi пишет код программы на Паскале. Отметим, что эксперты могут быть созданы самим пользователем.

Чтобы составить себе некоторое представление о работе в Delphi, проследим за реализацией в системе конкретной пробной программы, рекомендованной в качестве первого шага освоения Delphi в одном из руководств. Опустим для простоты описание несущественных сейчас деталей, касающихся вида экрана и объектов управления на нем. Сосредоточим свое внимание на сути процесса создания программы в среде Delphi.

Еще до того, как программист успел что-то сделать, система при запуске уже выполнила значительную работу. Во-первых, она создала для будущей программы стандартное окно (в Delphi его принято называть **формой**). Во-вторых, уже сгенерирован текст программы довольно значительной длины, который необходим для порождения этой формы на экране.

Теперь попробуем что-нибудь сделать сами. Выберем из предлагаемого широкого ассортимента стандартных объектов наиболее простой - кнопку. Найдем ее изображение в верхней части экрана (в Delphi ее называют **палитрой** **компонентов**) и щелкнем на нем мышкой. Тем самым мы дали системе знать, что нам нужна именно кнопка. Остается указать, в какое место формы поместить компонент. Подведем курсор «мыши» к требуемому месту и снова щелкнем - появится изображение кнопки. При необходимости с помощью мыши кнопку можно легко передвинуть или изменить ее размеры.

Заметим, что текст программы после появления кнопки увеличился - Delphi автоматически добавила необходимое описание.

Уже сейчас можно посмотреть на то, что получилось, в действии. Запустим программу и увидим на экране стандартное окно Windows, содержащее кнопку. В отличие от этапа проектирования, компоненты при выполнении программы «оживают»: если щелкнуть по этой кнопке мышкой, то будет видно, как она нажимается. (Правда, никакого эффекта пока нет, так как он еще не запрограммирован.)

Итак, мы уже запустили собственное Windows-приложение, хотя не набрали еще ни одной строчки текста с клавиатуры!

Теперь сделаем нашу кнопку действующей. Вернемся в режим проектирования и дважды щелкнем мышкой по кнопке. После этого Delphi вынесет на первый план текст программы и установит маркер в то место, где мы должны набирать текст, тем самым как бы призывая описать действия по нажатию на кнопку. Введем строку

Form I .Color : = clAqua

Несколько слов по ее расшифровке. Свойство Color (цвет) объекта Forml (нашей формы) получает значение clAqua (буквы с1 указывают на цвет, a Aqua - это название одного из 16 стандартных цветов). Снова запустим программу и нажмем кнопку - форма поменяет свой цвет.

Описанный выше пример хорошо показывает стиль работы с Delphi, существенно отличающийся от традиционного стиля программирования. Конечно, реальные задачи посложнее, но работа протекает примерно так же.

На примере описанной выше программы видно, что многие компоненты Delphi имеют свое визуальное изображение. Замечательным достоинством системы является то, что размещение компонентов на экране, а также задание начальных значений их свойств (размера, цвета, вида бордюра и др.) Delphi позволяет осуществлять на этапе конструирования формы без написания какой-либо программы. Для этой цели предусмотрено специальное окно, называемое **инспектором объектов**, в котором перечислены все доступные в режиме проектирования свойства выделенного компонента и их текущие значения. Разумеется, любое из них при необходимости легко изменить, что немедленно скажется на внешнем виде объекта Например, если в окне Инспектора объектов изменить цвет, то система тут же перекрасит компонент. Иными словами, можно до запуска программы видеть, как будет выглядеть на экране проектируемая форма.

Такой способ работы с объектами, имеющими графическое представление, принято называть ***визуальным программированием.***

Визуализация процесса позволяет значительно быстрее увидеть результат своих усилий, делает его наглядным. Не последнюю роль при этом, по-видимому, играют эмоции и эстетические чувства - стремление красиво разместить объекты, подобрать их цвет и т.п. Замечено, что даже люди, которые не умеют рисовать, часто с интересом и удовольствием занимаются построением изображенийиз готовых элементов.

Какие еще перспективные черты заложены в систему Delphi?

Напомним, что Delphi работает в среде Windows и позволяет создавать для нее программные продукты. Первая (16-разрядная) версия Delphi прекрасно функционирует под любой разновидностью Windows, в то время как вторая (32-разрядная; более мощная, но достаточно хорошо совместимая с первой) требует более мощной операционной системы, например, Windows'95. Учитывая, что фирма «Microsoft» -главный разработчик системного программного обеспечения для персональных компьютеров - уделяет сейчас основное внимание именно развитию Windows, с этой точки зрения компилятор для Windows нельзя не признать перспективным.

Кроме того, Delphi является системой объектного программирования. Delphi позволяет не только использовать уже ставшие классическими объекты типа **object**, но и позволяет создавать новые, которые могут иметь графическое изображение и обладать свойствами стандартных элементов среды Windows. Такие объекты получили название **визуальных компонентов**; для их описания используется специально зарезервированное слово **class**. Помимо уже встречавшихся выше формы и кнопки, примерами визуальных компонентов могут служить меню, списки, поля редактирования, полосы прокрутки, таблицы и многое-многое другое. К Delphi прилагается целая библиотека стандартных визуальных компонентов - Visual Component Library. Очень важно, что библиотеку эту каждый пользователь может пополнить, добавляя в нее собственные компоненты.

Еще одной существенной чертой системы программирования Delphi является ее открытость - почти все имеющиеся в системе объекты реализованы на языке Паскаль и могут быть легко дополнены новыми. Например, если вас по какой-либо причине не устраивает стандартный редактор чисел, вы можете написать собственный и подключить его к системе.

Таким образом, среда Delphi содержит в себе все наиболее передовые черты системы программирования. Она является мощным и в то же время несложным в использовании инструментальным средством для создания приложений с современным интерфейсом (в том числе и обучающих программ). Из-за своего высокого уровня системы программирования типа Delphi даже получили специальное название - Среда быстрой разработки приложений, **RAD** (Rapid Application Development).

Если посмотреть на компоненты, входящие в стандартную библиотеку, то многие из них словно специально созданы для обучающих программ. Рассмотрим несколько примеров.

Возьмем радиокнопки - группу кнопок, названную так благодаря функциональному сходству с переключателями в радиоаппаратуре. Основной особенностью радиокнопок является то, что из всей группы в нажатом состоянии всегда находится только одна кнопка. В педагогической интерпретации это выглядит как наиболее примитивный метод опроса - выбор единственного правильного ответа из списка предложенных. Если же вам захочется иметь несколько правильных ответов на вопрос, возьмите другую разновидность кнопок - **Check Box** (обычно этот термин переводят как «кнопки с независимой фиксацией»). У таких кнопок состояние никак не зависит от окружающих кнопок.

Для тех. кому традиционное тестирование кажется устаревшим, найдутся другие компоненты, например: **Image** - образ, изображение. Помимо возможности разместить картинку на экране, этот полезный компонент обладает способностью «чувствовать на себе» щелчок мышки, что позволяет элементарно реализовать контроль вопросов типа «Найдите и укажите на карте остров Мадагаскар». Кроме того, во все. компоненты библиотеки Delphi заложена технология «**Drag and drop**-«Перенеси и оставь». Благодаря ей можно располагать объекты на экране определенным образом при выполнении задания «Составьте схему из батареи, амперметра, выключателя и резистора».

Наконец, если в компьютере есть устройство считывания с компакт-дисков или хотя бы набор звуковых файлов, в Delphi предусмотрен специальный компонент -проигрыватель, позволяющий включать в урок музыку или звуки.

Важным достоинством Delphi как инструментальной среды является то, что ее компилятор делает автономные ЕХЕ-файлы. Они будут работать в среде Windows даже на той ЭВМ, на которой нет системы Delphi. Это свойство выгодно отличает Delphi от других аналогичных сред, например, от Microsoft Visual Basic.

Завершая краткое знакомство с системой Delphi, рассмотрим некоторые наиболее важные базовые понятия, лежащие в ее основе.

Все объекты в Delphi характеризуются свойствами. **Свойство** - это атрибут объекта, определяющий то, как объект выглядит или как он может себя вести. Например, свойства, определяющие внешний вид кнопки: color - ее цвет, left и top -координаты левого верхнего угла, heigt и width - высота и ширина. В качестве примера свойств, определяющих, как кнопка может себя вести, опишем свойство «enabled» (от английского давать возможность, разрешать, разблокировать, включать). Оно может принимать значения TRUE или FALSE - в зависимости от этого кнопка либо будет фиксировать на себе щелчок мыши, либо нет.

Отметим, что свойство является своеобразным обобщением понятия поля объекта **object**, так как помимо имени и типа дополнительно содержит способы записи и чтения значения поля. Например, при выполнении строки Form I.Color *:=* clAqua происходит не просто запись значения, характеризующего цвет, в определенное место памяти, а вызывается метод Forml.SetColor(clAqua). Этот метод не только производит присвоение, но и перекрашивает форму.

Еще одно важное понятие Delphi - это событие. Термин «событие» заимствован из Windows, его значение проще всего объяснить на примерах. В частности, событиями являются воздействия пользователя на клавиатуру или мышь (нажатие на клавишу или ее отпускание, движение мыши и т.п.). Кроме того, событием являются любые изменения состояния экрана: создание окна, изменение его размеров и многие другие (таблица с описанием полного списка событии занимает несколько страниц).

В ответ на любое событие в системе Windows передает управление обработчику события (event handler). **Обработчик события** - это программа, которая определяет реакцию объекта на это событие. Если пользователь не предусмотрел действия по данному событию. Windows обработает его сама; в частности, она может просто проигнорировать событие.

Благодаря такой идеологии программа на Delphi фактически представляет собой совокупность относительно самостоятельных обработчиков всевозможных событий.

Рассмотрим еще одно понятие - **метод.** Метод - это процедура или функция класса, определяющая поведение объекта. Метод приводит к выполнению определенной последовательности действий, часть из которых может быть связана с изменением внешнего вида объекта. Так, применение к форме Forml процедуры Close не просто удаляет ее из списка окон Windows, но и обеспечивает его исчезновение с экрана. Другой пример - метод SetFocus для формы переносит на нее фокус ввода, т.е. она становится активной и принимает набор с клавиатуры. Кроме того, метод выносит изображение активного окна на первый план и выделяет цветом его заголовок.

Отметим, что понятие метода в Delphi не отличается от введенного ранеепонятия при описании Object Pascal.

### 9.8. СИСТЕМА ОБЪЕКТНОГО ПРОГРАММИРОВАНИЯ SMALLTALK

Большой интерес представляют языки, изначально созданные для объектно-ориентированного программирования. Самым ярким примером является язык Smalltalk.

Система Smalltalk представляет собой совокупность взглядов, методов и средств проектирования и эксплуатации современных систем обработки информации, базирующихся на ряде новых концепций в понимании и модельном представлении вычислительного процесса. Авторами системы являются сотрудники исследовательской группы фирмы XEROX (США), работающие над этим проектом с 1970 г.

Smalltalk принадлежит к классу, так называемых, объектно-ориентированных языков программирования. Объект в Smalltalk содержит и процедурную, и понятийную части (объект=данные+процедуры), причем для представленияпонятийнойчасти используется аппарат абстрактных типов данных.

При решении задач на объектно-ориентированном языке используется абстрактная модель обработки данных

«объект-класс-сообщение».

Эта модель носит фундаментальный характер и легко может применяться в любой предметной области.

**Объекты и сообщения**. Основой объектно-ориентированного программирования (ООП) является **объект**. В качестве объектов могут выступать абстрактные данные (числа, символы, файлы и т.д.) или сущности моделируемой предметной области и их взаимосвязь (черепашка, цветок, собака, врач, треугольник и т.п.). Объекты взаимодействуют друг с другом, посылая и принимая сообщения.

Объект обладает собственной памятью для хранения информации и набором **методов** - операций для манипулирования этой информацией. Для активизации метода объекту посылается сообщение, являющееся аналогом обращения к функции в традиционных языках программирования.

В каждом сообщении должны быть указаны

• адресат-объект, которому посылается сообщение;

• имя сообщения;

• объекты-параметры, которыми манипулирует метод (необязательно).

Программа в Smalltalk - это последовательность сообщений, посылаемых различным объектам. Простые сообщения делятся на три вида: унарные, бинарные, ключевые, табл.3.8.

**Унарное** сообщение представляет собой конструкцию следующего вида:

<объект-адресат> <сообщение>.

Например:

'строка'размер.

**Бинарное** **сообщение** имеет следующий вид:

<объект-адресат> <арифметико-логическая операция> <объект-параметр>.

Например:

2+3

**Ключевое** **сообщение** представляет собой конструкцию, допускающую несколько параметров.

Например:

книга автор: 'Толстой' название: 'Война и мир'.

В случае композиции двух или более сообщений их обработка выполняется в соответствии со следующими правилами:

*Правило 1.* Унарное сообщение имеет более высокий приоритет, чем бинарное, а бинарное - более высокий приоритет, чем ключевое сообщение.

*Правило 2.* В рамках одного типа все сообщения имеют одинаковый приоритет. Сложное выражение выполняется слева направо.

*Правило 3.* Скобки меняют порядок вычисления.

Таблица 3.8.

**Некоторые примеры сообщений и их результатов**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Сообщение** | **Описание** | | **(Класс)** | | |
| 1+2  Бинарное сообщение | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | 1 (МалоеЦелое)  +  2 (МалоеЦелое)  3 (МалоеЦелое) | | |
| " привет' вПозиции: 5  Ключевое сообщение | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | ' привет' (Строка)  в Позиции:  5(МалоеЦелое)  е (Символ) | | |
| 5факториал  Унарное сообщение | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | 5(МалоеЦелое)  факториал  ------------------------  120(БольшоеПоложительное Целое) | | |
| 5между: 3 и:7  Ключевое сообщение | | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | 5(МалоеЦелое)  между: и:  3, 7 (МалоеЦелое)  истина |
| 1+2  Бинарное сообщение | | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | 1 (МалоеЦелое)  +  2 (МалоеЦелое)  3 (МалоеЦелое) |
| ' привет' вПозиции: 5  Ключевое сообщение | | объект-адресат:  имя сообщения:  объект-параметр:  результат: | | 'привет' (Строка)  в Позиции:  5(МалоеЦелое)  е (Символ) |
| 5факториал  Унарное сообщение | | объект-адресат:  имя сообщения:  объект-параметр;  результат. | | 5(МалоеЦелое)  факториал  -------------------------  120(БольшоеПоложнтельноеЦелое) |
| 5между: 3 и:7  Ключевое сообщение | | объект-адресат:  имя сообщения:  объект-параметр:  результат; | | 5(МалоеЦелое)  между: и:  3, 7 (МалоеЦелое)  истина |

**Классы объектов и методы.** Объект обладает свойствами, поведением и состоянием. Ему можно присвоить имя. Объекты с одинаковыми свойствами и поведением объединены в классы. Каждый объект входит в один класс и называется **экземпляром класса.**

Объекты из одного класса распознают одни и те же сообщения и имеют одинаковую структуру собственной памяти. Объект имеет собственную память - **переменные экземпляра**, где хранится информация о его свойствах и состоянии. Доступ к указанной информации имеет только сам объект.

С каждым объектом связан набор (протокол) сообщений, которые он понимает. Всякому сообщению из протокола соответствует реализующая его процедура, называемая методом.

Метод определяет реакцию объекта на данное сообщение, т.е. его поведение. Он состоит из операций над своими переменными экземпляра и из посылок сообщений другим объектам. В конечном счете объект возвращает ответ на посланное ему сообщение. Поскольку все объекты одного класса обладают одинаковым набором методов, последние хранятся в одном месте - в самом классе.

У каждого класса существует два типа методов:

• методы класса (используются, когда необходимо произвестикакие-либо действия с целым классом, например, добавить к классу новый экземпляр);

• методы экземпляра (сообщения к экземплярам данного класса).

Один из основных классов языка Smalltalk - класс «Величина». Экземплярами этого класса являются объекты, которые можно измерять, сравнивать, упорядочивать и вычислять. К этому классу , в частности, относятся символы, числа, дата и время. Эта группа классов имеет следующую иерархию.

*Символ.* Экземпляры класса «Символ» являются расширенным множеством символов во внутреннем коде со значениями от 0 до 255.

*Дата.* Экземпляры класса «Дата» представляют собой даты, такие как

ЯНВАРЬ 1, 1980.

*Время.* Экземплярыкласса «Время» представляют собой время, такое как 10 или

12:15.

*Число.* Система поддерживает три вида числа:

• целое;

• вещественное с плавающей точкой (если есть сопроцессор);

• дробь - рациональные целое.

Символ обозначается с помощью знака $, например,

$a,$9,$M.

В языке Smalltalk имеется 6 типов переменных:

• *экземплярные переменные,* существуют в течение всего жизненного цикла объекта экземпляра класса, в заголовке которого они описаны;

• *временные переменные,* описываются внутри метода и существуют только во время выполнения данного метода;

• *переменные класса,* декларируются в описании класса, доступны всем экземплярам данного класса;

• *глобальные переменные,* доступны всем экземплярам всех классов;

*•переменные пула,* декларируются в описании класса и доступныэкземплярамнекоторого подмножества классов;

• *псевдопеременные,* специально зарезервированные идентификаторы, указывающие на специальные объекты.

Для присвоения значения какой-либо переменной(исключая псевдопеременные)используется операция : : = (присваивание).

Экземпляры объектов можно сравнивать друг с другом.

Класс «Объект» содержит методы сравнения: *=,* ~=.Все классы, являющиеся подклассами этого класса, наследуют эти методы.Помимо этого, каждый класс может определить для себя другие методы сравнения. Например, классы из группы «Величина» имеют еще несколько методов: > < *<=* >=. Аналогичные методы определены для класса «Строка».

Результат сравнения объектов - экземпляр класса «Логический». Этот класс имеет два подкласса: Истина и Ложь. Экземплярами этих классов являются псевдопеременные истина и ложь соответственно. Задание новых экземпляров этих классов приводит к ошибке.

Классы «Истина» и «Ложь» имеют следующий набор методов:

и: - истина, если адресат и параметр - «истина», в противном случае -«ложь» (существует аналогичное бинарное сообщение - &)

или: - ложь, если адресат и параметр - «ложь», в противном случае -«истина» (существует аналогичное бинарное сообщение -|).

На основе этих методов реализованы условные конструкции. В общем виде условная конструкция имеет вид:

<условие>

если условие истинно, выполнить <выражение1> если условие ложно, выполнить <выражение2>.

На языке ООП эта конструкция записывается таким образом:

<объект-адресат класса Истина или Ложь>

если Истина:<объект-параметр1>

если Ложь: <объект-параметр2>.

В данном случае <объект-параметр> представляет собойблок сообщений.

Возможна конструкция

<условие> еслиИстина: <объект-параметр>,

а также

<условие> еслиЛожь: <объект-параметр>.

Блок сообщений -это группа сообщений, заключенная в квадратные скобки. Блоки - это объекты специального вида, вычисление которых (т.е. выполнение списка внутренних операторов) производится путем посылкик ним сообщения-значения.

Блок может иметь аргументы. В этом случае сообщение для его вычисления будет ключевым:

[блок] значение: <значение аргумента>

**Программирование на SmalTalk**. Язык Smalltalk обладает богатым набором циклических конструкций, отраженных в табл. 3.9.

Программирование на SmallTalk состоит не только в использовании предопределенных в языке объектов и их методов, но и в задании новых методов и объектов.

Вначале рассмотрим создание новых методов для уже существующих классов. Перед созданием нового метода необходимо ответить на следующие вопросы:

1) что будет объектом-адресатом для создаваемого метода;

2) каким будет объект-параметр;

3) что будет результатом, возвращаемым новым методом. Таким образом определяется сообщение, обращающееся к новому методу. Затем можно описать и добавить в соответствующий класс сам создаваемый метод по форме

имя Сообщения

"комментарии"

| локальные переменные сообщения |

^ возвращаемый результат.

Таблица 3.9

**Циклические конструкции языка Smalltalk**

|  |  |
| --- | --- |
| **Запись** | **Содержание** |
| <число> разПовторнть: [блок сообщении]  [блок условия] покаЛожь: [блок сообщений]  блок условия] покаИстина: [блок сообщений] <число1> до: <число2> через: <шаг>  выполнить:  [:<переменная> | блок сообщений]  <объект> выполнить:  [:<переменная> | блок сообщений] <объект> выбрать:  [:<переменная> | <условие>]  <объект> исключить:  [:<переменная> | <условие>]  <объект> собрать:  [:<переменная> : сообщение] | Повторить заданное <число> раз  Пока условие ложно, выполняются сообщения  Пока условие истинно, выполняются сообщения  Выполнить блок сообщений, пока значение  <переменной>, изменяющее свое значение  с заданным шагом, принадлежит  промежутку (число1, число2)  Значение <переменной> присваивается  последовательно элементам <объект>  Изменяет <объект>, удаляя элементы, не  удовлетворяющие условию  Изменяет <объект>, удаляя элементы,  удовлетворяющие условию  Заменяет каждый элемент <о6ъекта> на результат  выполненного сообщения |

В качестве примера рассмотрим определение метода поиска максимальногоиздвух целых чисел.

Объектом-адресатом будет целое число, следовательно сам метод будет принадлежать к классу «Целое». Параметром будет второе целое число, а возвращаемым результатом - максимальное из этих чисел.

Текст метода «макс» может быть следующим (комментарии приведены в кавычках):

макс: экзЦелое "имяметода с параметром"

| максимум | "список локальных переменных"

сам>экзЦелое "псевдопеременная "сам" означает объект-адресат"

еслиИстина: [максимум : = сам]

еслиЛожь : [максимум : = экзЦелое].

^Максимум.

**Наследование и полиморфизм.** Создание новых объектов происходит по принципам наследования и полиморфизма. Сами объекты подчиняются отношению наследования, т.е. могут быть представлены в виде иерархической структуры с помощью дерева.

Чтобы создать объект, классу посылается сообщение о создании нового экземпляра. Класс создает экземпляр с присущей всем объектам этого класса структурой -набором переменных экземпляра. Он их инициализирует и, если в этом есть необходимость, выдает созданный экземпляр в качестве ответа на сообщение.

Например:

х:= Массив новыйЭкземпляр: 10.

(создается новый объект с именем х, класса Массив, размера 10).

Возможно создание нового экземпляра в результате выполнения некоторых сообщений. Например:

' Привет,', 'мартышка'

(создается новый экземпляр класса «Строка», имеющий значение 'Привет, мартышка )

1/2

(объект-адресат и объект-параметр - экземпляры класса «Целое», а результат -экземпляр класса «Дробь»)

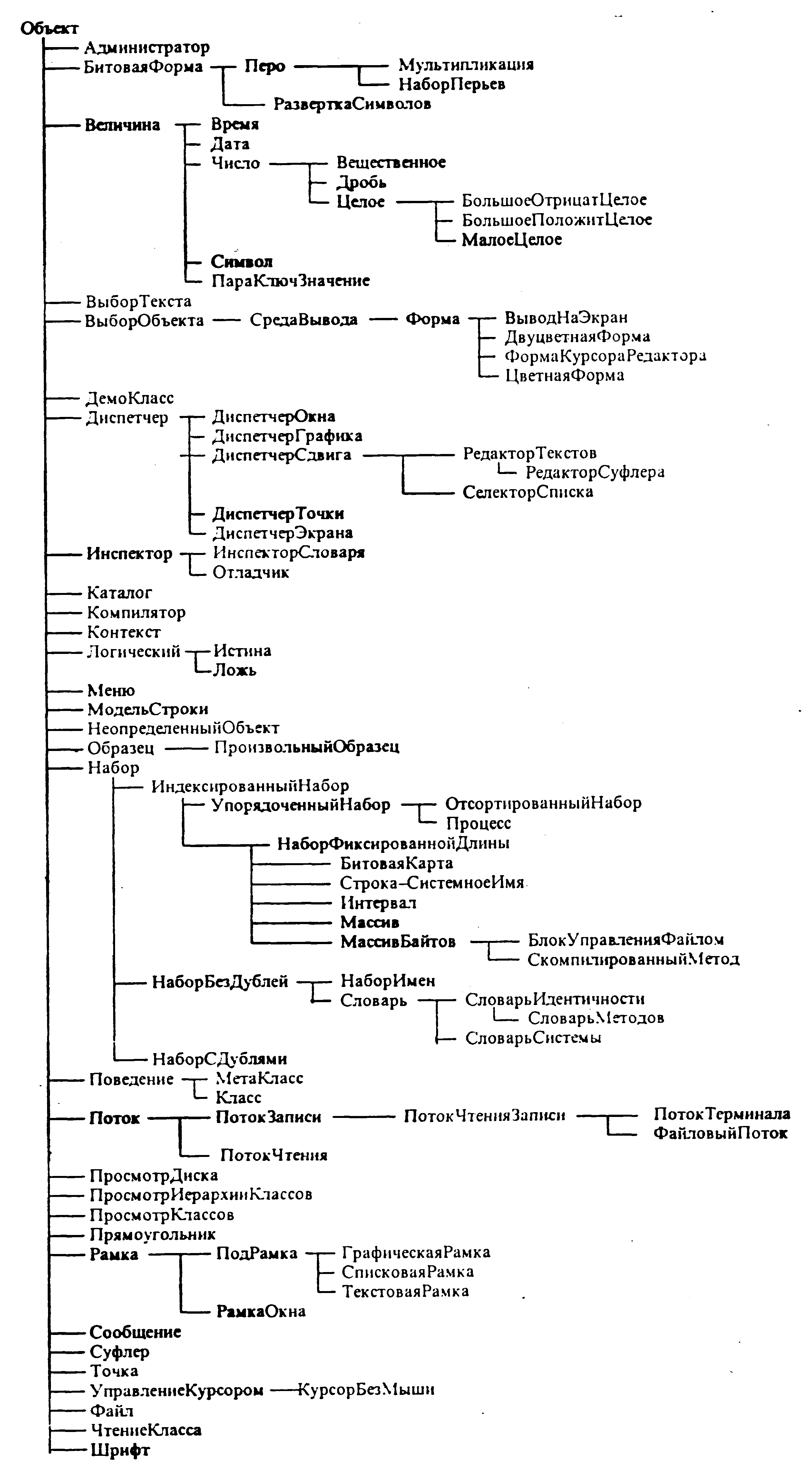
1>2 (результат класса «Ложь»)

Каждый класс имеет одного предка, называемого **суперклассом**. Класс может иметь одного или нескольких потомков, называемых **подклассами**.

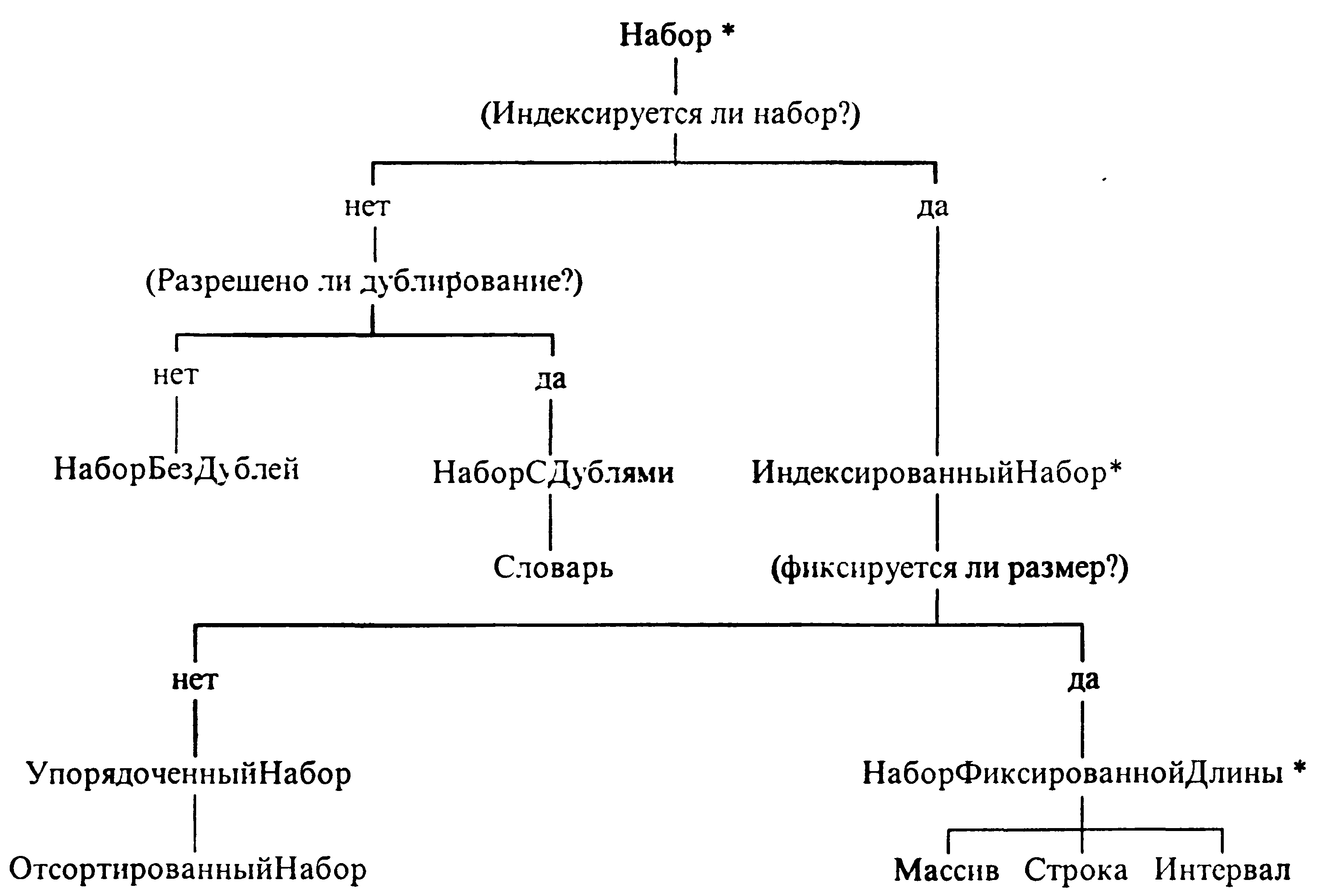
Класс «Объект» не имеет суперкласса и является корнем дерева иерархии классов.

Всякий класснаследует переменные экземпляра и методы своего суперкласса. Кроме того, он может содержать новые переменные экземпляра и методы, может переопределять у наследованные.

Знание иерархии классов важно дляпонимания процессов выдачи ответана сообщение. При получении сообщения объект ищет в протоколе методов экземпляра своего класса метод с именем, совпадающим с именем сообщения. Если такого метода нет, он ищется в списке методов суперкласса и так далее. Когда метод обнаружен, он выполняется и выдается ответ. Если нигде, вплоть до корня дерева, метод не будет обнаружен, выдается сообщение об ошибке. Дерево предопределенных классов языка SmallTalk приведено ниже, рис. 3.18.



*Рис. 3.18.* Дерево предопределенных классов языка Smalltalk



*Рис. 3.19.* Основные подклассы класса «Набор»(\* - абстрактные классы, экземпляры которых не создаются)

В системе SmallTalk существует два типа объектов:

1) объекты с поименованными переменными (переменные с именами);

2) объекты с индексируемыми переменными (состоят из N объектов; при описании класса, которому принадлежит такой объект, указывается только одна именованная переменная, остальные N-1 индексируются и доступ к ним осуществляется посылкой сообщения "вПозиции: индекс", а не указанием имен).

Объекты с индексируемыми переменными также могут иметь поименованные переменные экземпляра.

**Набор** - это группа связанных между собой объектов. Классы наборов определяют различные структуры данных для хранения произвольных объектов. Основные подклассы класса «Набор» представлены на рис.3.19.

**Полиморфизм** проявляется в том, что одно и то же сообщение может посылаться разным объектам, и понимают они его каждый по-своему. Это увеличивает наглядность программ, поскольку не надо беспокоиться о случайном совпадении имен сообщений, как это имеет место с именами функций в традиционных языках.

Классы также являются объектами. Таким образом, классам тоже можно посылать сообщения. Поскольку сообщение, посылаемое экземпляру, анализируется этим классом, то логично предположить, что сообщение, посылаемое классу, анализировалось его классом. Мы получили единый механизм вызова сообщений, обеспечивающий ясность программ.

Класс класса называется **метаклассом.** Классам присваиваются имена, а метаклассам - нет. Для каждого класса существует один метакласс (рис. 3.20).

Между метаклассами тоже существует иерархия, образованная тем же отношением «суперкласс - подкласс». Иерархия классов и иерархия метаклассов изоморфны друг другу.

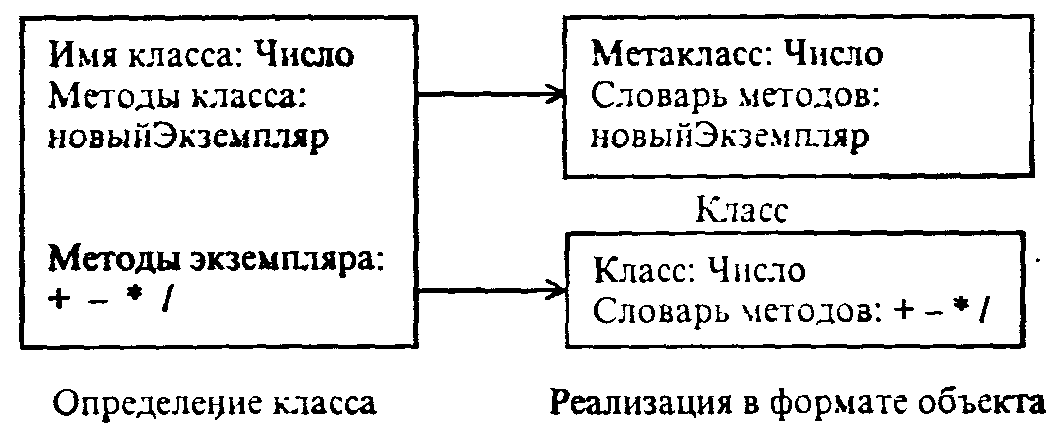


Рис. 3.20. Классы и метаклассы

В ряде случаев необходимо работать с объектами, обладающими свойствами пересечения некоторых классов, но не включения, т.е.:

Класс А не является подклассом класса В

Класс В не является подклассом класса А ПересечениеА и В не пусто.

В таком случае создается специальный суперкласс обоих классов, описывающий их общую часть. Этот суперкласс называют абстрактным классом. Он не создает своих экземпляров, а только служит для наследования общих методов.

**Рекурсии.** Как и в других языках, ориентированных на обработку символов и нашедших применение в создании систем искусственного интеллекта, в SmallTalk важную роль играет рекурсия. Рассмотрим пример определения рекурсивного метода, вычисляющего числа Фибоначчи:

Фибоначчи

"выдать N чисел Фибоначчи, где N - объект-адресат"

сам<3

еслиИстина: [^1]

еслиЛожь : [^ (сам - 1) фибоначчи + (сам - 2) фибоначчи ]

Классическим примером задачи, требующей рекурсии, является задача о Ханойских башнях. Имеется три штырька. На штырьке №1 расположены кольца разного диаметра в порядке убывания их размера снизу вверх. Требуется переложить кольца со штырька №1 на штырек .№2, пользуясь штырьком .№3 как промежуточным, таким образом, чтобы кольца на штырьке №2 располагались в порядке убывания их диаметра снизу вверх и чтобы в процессе перекладывания кольцо большего размера никогда не накладывалось на кольцо меньшего диаметра.

Данная задача решается рекурсивно.

а) Базис.

Если кольцо одно, то надо просто переложить это кольцо с х на у.

б) Рекурсия.

Если число колец равно *k,* то надо переложить со штырька х на штырек z *k-*1 колец, затем переложить самое большое *k-e* кольцо на у и переложить все остальные *k-*1 колец со штырька z на у.

Объект-адресат в нашем случае - целое число (количество колец). (Следовательно, данный метод будет принадлежать классу целых чисел.)

Объекты-параметры - символы '1', '2', '3' (информация: с какого штырькана какой надо переложить кольцо, а какой использовать как дополнительный).

Результат - надписи на экране о последовательности выполнения даннойзадачи**.**

Следовательно, вызов метода будет осуществляться следующим образом:

<число> ханойС: '1' на: '2' через:'3'

Реализация метода ханойС: х на: у через: z может быть следующей:

ханойС: х на: у через: z

"головоломка 'Ханойские башни""

сам=1

еслиИстина: [СистемнаяИнформация поместитьВсеПоследующие:

'Переложить со штырька',х,'на штырек',у;символВК.]

еслиЛожь: [(сам - 1) ханойС: х на: z через: у

СистемнаяИнформация поместитьВсеПоследующие:

'Переложить со штырька',х,'на штырек',у;символВК.]

(сам - 1) ханойС: z на: у через: х]