Это весьма непростой и спорный вопрос. В "чистых" язы-

ках, использующих подход OOP, статические методы не существу-

ют; все методы являются виртуальными. И сторонник "чистого"

подхода OOP мог бы сказать, что все методы в нашей иерархии

объектов должны быть виртуальными именно по той причине, что

виртуальные методы стоят на первом месте. Такой аргумент мож-

но было бы признать справедливым, но еще больше истины в том,

что делать все методы только виртуальными просто непрактично

- по крайней мере, тогда, когда Вы программируете на Turbo

Pascal.

На наш взгляд, имеются два убедительных довода в пользу

того, чтобы везде, где это только возможно, использовать ста-

тические методы. Во-первых, интеллектуальный компоновщик

Turbo Pascal не может отменять неиспользуемые виртуальные ме-

тоды, а только неиспользуемые статические методы; простой акт

ввода объекта данного типа приводит к тому, что в программу

должны быть скомпонованы все виртуальные методы этого объек-

та. И во-вторых, чем больше виртуальных методов имеет объект,

тем обширнее его таблица виртуальных методов VMT: объект,

имеющий 100 виртуальных методов, использовал бы более 400

байт пространства в сегменте данных. В системе Object

Professional нет ни одного объекта, который бы имел так много

виртуальных методов, но если бы все методы были виртуальными,

в ней бы имелся не один такой объект.

Полагая, что из практических соображений невозможно во

всех случаях применять только одни виртуальные методы, мы

стоим на тех позициях, что целесообразно создавать виртуаль-

ные методы только в следующих трех случаях:

а) если мы знаем, что он должен быть отменен объек-

том-потомком,

б) если метод предназначен для того, чтобы быть отменен-

ным, и для этой цели и существует,

в) если в самой природе метода заложена возможность то-

го, что желательно его отменить в объекте-потомке.

Указатели процедур против производных типов.

Еще один концептуально спорный вопрос. Рассмотрим слу-

чай, когда для некоторого объекта необходимо обеспечить

средство, позволяющее программисту передавать такую информа-

цию для объекта, которая не всегда бывает известна на момент

компилирования. Наглядным примером такого объекта является

"PickList" ("Список\_Подбора") в модуле OPPICK: он должен

обеспечить средство, которое предоставит Вам возможность "со-

общить" ему, какие элементы имеются в списке подбора.

Сторонник "чистого" метода мог бы сказать, что для реше-

ния этой проблемы следует обеспечить фиктивный виртуальный

метод, который, как предполагается, будет возвращать необхо-

димую информацию, и пусть потом программист создает производ-

ный тип, который отменяет этот метод. Но такой подход порож-

дает две проблемы. Первая заключается в том, что было бы

досадно, если бы КАЖДЫЙ раз, когда возникает потребность ис-

пользовать, например, объект PickList, пришлось создавать

производный тип, особенно в том случае, если все, что Вам

действительно требуется - это написать функцию для восстанов-

ления строки на основе номера элемента. Вторая проблема сос-

тоит в том, что при этом в большинстве случаев исключается

возможность использования одного и того же объекта PickList

для отображения на экране различных списков.

Мы предпочитаем принять компромиссное решение. В подоб-

ных случаях действительно имеются виртуальные методы, которые

Вы можете отменить, если захотите. Но помимо этого существуют

такие средства, которые с помощью указателя процедуры задают

ту подпрограмму, написанную пользователем, которая нужна для

выполнения операции. Реализация виртуального метода по умол-

чанию просто использует этот указатель процедуры, чтобы выз-

вать Вашу подпрограмму.

Конструкторы и деструкторы.

В системе Object Professional в соответствии с соглаше-

ниями фирмы Borland принято для конструкторов использовать

имя "Init" ("Начальный"), а для деструкторов - имя "Done"

("Законченный"). Большинство объектов также имеет конструктор

с именем "Load" ("Загрузка"), который используется для заг-

рузки объекта из потока (еще больше на потоках через момент

?) - в соответствии с другим соглашением фирмы Borland.

Для обеспечения унификации каждый объект в иерархии име-

ет деструктор с именем Done, даже если он и не нужен, и эти

деструкторы никогда не принимают никаких параметров. Что ка-

сается конструкторов, однако, то здесь существуют некоторые

варианты.

Конструктор Init всегда принимает столько параметров,

сколько возможно. Если в определенных обстоятельствах жела-

тельно передать больше параметров, существует второй конс-

труктор с именем "InitCustom" ("Начальная\_Настройка"), кото-

рый их принимает (дополнительно к параметрам, которые

переданы конструктору Init). В объекте, имеющем конструктор

InitCustom, конструктор Init обычно используется для вызова

конструктора InitCustom, который, в свою очередь, по умолча-

нию передает значения туда, где это необходимо.

Существует несколько объектов, которые имеют также и

другие конструкторы - например, "MemoFile" ("Файл\_Памяти")

(модуль OPMEMO) и PickList (модуль OPPICK) - но при этом в

большинстве случаев их имена начинаются с "Init", и они при-

нимают такие же параметры, как и конструктор Init.

Использование динамически распределяемой области.

Все объекты в библиотеке используют динамически распре-

деляемую область для хранения данных в тех случаях, когда мо-

жет изменяться размер данных, а также в тех случаях, когда

может изменяться количество элементов данных (например, спи-

сок полей с использованием указателей в экране ввода данных и

список элементов в меню). Такой подход обеспечивает эффектив-

ное использование имеющейся в Вашем распоряжении памяти, но

он также обладает одним возможным существенным недостатком:

он может вызвать в определенных случаях проблемы фрагментиро-

вания динамически распределяемой области.

Чаще всего проблемы фрагментирования динамически распре-

деляемой области могут возникнуть при использовании программ,

которые создают и разрушают множество объектов без какого-ли-

бо определенного порядка (лучшим примером таких программ яв-

ляется DESKPOP), и особенно это относится к таким программам,

которые позволяют пользователю "распахивать" окна во весь эк-

ран и изменять их размеры. Мы много размышляли над этим воп-

росом, надеясь найти способ, чтобы избежать всей этой сово-

купности проблем, но пришли к заключению, что все возможные

решения являются либо чересчур неуклюжими, либо слишком на-

вязчивыми, либо слишком рискованными. Единственное разумное

решение этой проблемы мы видим в том, чтобы фирма Borland

обеспечила программу управления динамически распределяемой

областью, с помощью которой можно осуществлять очистку памяти

более или менее доступно удобно и понятно.

Теперь перейдем к менее пессимистичному и более общему

вопросу. Модуль OPROOT, который используется большинством

других модулей, содержащих объекты, устанавливает функцию

ошибки динамически распределяемой области, которая "отвеча-

ет" за то, чтобы все аварийно завершившиеся вызовы New или

GetMem возвращали "пустое" значение ("Nil"). Если необходимо,

Вы можете отменить этот режим путем установки собственной

функции ошибки динамически распределяемой области, но прежде

чем это сделать, необходимо убедиться, что Вы ясно себе

представляете возможные последствия. Если это только возмож-

но, мы рекомендуем Вам писать свою программу так, чтобы она

работала при этих условиях. Единственное, что Вы можете сде-

лать, чтобы облегчить свою работу, это использовать функции

GetMemCheck и FreeMemCheck в модуле OPROOT всякий раз, когда

Вам необходимо разместить память или аннулировать размещение

памяти из динамически распределяемой области; это то, что мо-

гут делать все Ваши объекты. Подробные сведения об использо-

вании этих функций приведены в документации, описывающей мо-

дуль OPROOT.

Обработка ошибок.

Все объекты более высокого уровня в библиотеке построены

так, чтобы выдавать сообщения обо всех ошибках, которые

встречаются после того, как объект был введен в центральную

программу обработки ошибок. Эта программа обработки ошибок

передает как код ошибки, так и (обычно) принятое по умолчанию

сообщение об ошибках, которое позволяет программе обработки

ошибок вывести на экран либо типовое сообщение об ошибках,

либо сообщение, модифицированное в соответствии с потребнос-

тями пользователя. Ошибки, которые встречаются в конструкто-

рах (прежде чем имеется возможность указать программу обра-

ботки ошибок), хранятся в глобальной переменной InitStatus,

объявленной в модуле OPROOT. Более подробные сведения, касаю-

щиеся способов обработки ошибок, приведены в документации,

описывающей объект "CommandWindow" ("Окно\_Команд") (модуль

OPWINDOW) и в приложении D ("Коды ошибок").

Обработка команд и клавиатуры.

Все объекты в библиотеке, которые действуют на основе

команд, вводимых с клавиатуры, используют объект с именем

"CommandProcessor" ("Командный\_Процессор") для того, чтобы

преобразовать нажатия на клавиши в команды. Для того, чтобы

это было возможно, соответствующий объект передает объекту

CommandProcessor таблицу команд, представляющую собой массив

особого формата, который устанавливает зависимости между пос-

ледовательностями клавиш и командами. Каждая команда в библи-

отеке соответствует уникальному коду. Использование кодов ко-

манд и таблиц команд не только позволяет программе во время

своего выполнения модифицировать и добавлять назначения кла-

виш, но также и создавать отдельные программы конфигурирова-

ния, которые позволяют пользователю модифицировать назначе-

ния. Подробные сведения, касающиеся обработки команд,

приведены в документации, описывающей модуль OPCMD.

Объект CommandProcessor также реализует некоторые прог-

раммные средства обращения к подпрограммам ("hook"), причем

наиболее важное из них обеспечивает возможность указать, ка-

кая подпрограмма должна быть вызвана в тот момент, когда

пользователь нажмет на соответствующую клавишу. Это средство

позволяет выполнять "фоновые задачи" в то время, пока ввод

команды с клавиатуры еще не произведен.

Цвета экрана и объекты ColorSet.

В системе Object Professional все объекты, которые вы-

полняют экранный ввод/вывод ("I/O" - от "input/output"), раз-

работаны так, чтобы облегчить написание программ, автомати-

ческ учитывающих различия между цветными и монохромными

системами. Поэтому когда объекту необходимо указать атрибуты

отображения информации на экране, этот объект запрашивает два

атрибута: один для цветных систем и другой - для монохромных.

После этого объект будет использовать простую подпрограмму в

модуле OPCRT для того, чтобы выбрать соответствующий атрибута

отображения в данной ситуации.

Эти объекты также разработаны таким образом, что позво-

ляют, чтобы выбранные Вами цветовые характеристики изображе-

ния были сосредоточены в единой структуре данных в объекте с

именем ColorSet, который может быть многократно использован в

течение всего времени работы программы. Когда Вы вводите объ-

ект, использующий множество различных атрибутов отображения,

то Вы просто передаете ему имя Вашего объекта ColorSet. Опи-

сание объекта ColorSet и более подробные сведения о том, для

чего он используется, приведены в документации, описывающей

модуль OPCRT.