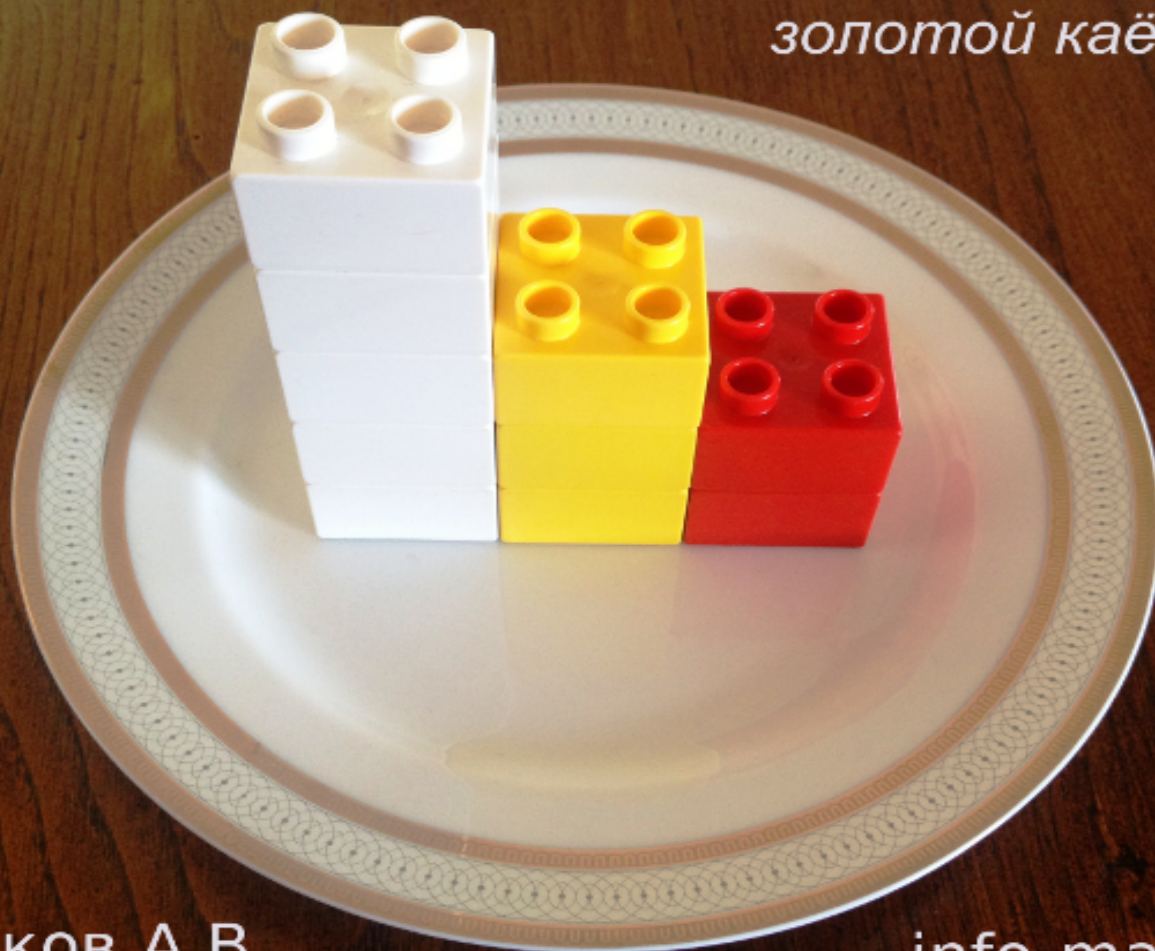


Объектно Ориентированное Программирование

*на блюдечке с
золотой каёмочкой*



Поляков А.В.

info-master.su

*О сложных приёмах программирования
простым и понятным языком*

*Начало вашего пути от "середнячка"
к профессионалу*

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	8
ЧАСТЬ 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ООП	10
1. ПАРАЛЛЕЛЬНЫЕ МИРЫ	11
1-1. Мир людей	11
1-2. Мир машин	13
1-3. Что такое ООП	15
1-3-1. Основной замысел ООП	15
1-3-2. Сторонники и противники ООП	19
1-3-3. Недостатки структурного программирования	20
1-3-3-1. Доступ к данным	21
1-3-3-2. Суровая действительность	22
1-3-4. Языки программирования	23
2. ОСНОВНЫЕ ПОНЯТИЯ ООП	24
2-1. Класс	24
2-2. Объект	24
2-3. Свойства объекта	25
2-4. Методы объекта	25
2-5. Прежде чем погрузиться в дебри...	26
2-6. Абстракция	27
2-7. Абстрактные типы данных	28
2-8. Сообщения	29
2-9. Инкапсуляция	30
2-10. Интерфейс	30
2-11. Наследование и повторное использование кода	31
2-12. Полиморфизм	32
2-13. Конструктор и деструктор	33
2-14. Виртуальные функции	33
2-15. Абстрактные классы	35
2-16. Шаблоны функций и классов	37
2-17. Прототип	38
2-18. “Суммируя впечатления вечера”	38
3. ОСНОВНЫЕ КОНЦЕПЦИИ ООП	39
3-1. Объекты с похожим поведением	39
3-2. Объектная декомпозиция	40
3-3. Модульность	41
3-4. Иерархия классов	42
3-5. Типизация	45

3-6. Параллелизм	46
3-7. Сохраняемость	48
3-8. Ещё раз об инкапсуляции и полиморфизме	48
3-9. Объектная модель	49
3-9-1. Основные элементы объектной модели	51
3-9-2. Отношения между классами и объектами	53
3-9-2-1. Ассоциация	53
2-2. Обобщение	56
3-9-2-3. Зависимость	57
3-10. Основные средства разработки классов	57
3-10-1. Наследование	57
3-10-2. Полиморфизм	58
3-10-2-1. Простой полиморфизм	58
3-10-2-2. Сложный полиморфизм	59
3-10-3. Композиция	60
3-10-3-1. Когда наследование применять не стоит	61
3-11. Дополнительные средства и приёмы разработки классов	63
3-11-1. Метаклассы	63
3-11-2. Делегирование методов	65
3-11-3. Контейнерные классы	65
3-11-4. Параметризованные классы	66
4. ООП - ВОПЛОЩЕНИЕ В ЖИЗНЬ	67
4-1. Поля данных	67
4-2. Методы	67
4-3. Наследование классов и интерфейсы	68
4-4. Реализация инкапсуляции и полиморфизма	69
4-4-1. Контроль доступа	69
4-4-2. Методы доступа	70
4-4-3. Свойства объекта	70
4-5. Основные составляющие объектно-ориентированных программ	71
4-5-1. Конструктор и деструктор	72
4-5-2. Интерфейс	73
4-5-3. Реализация	74
4-6. Состояние	74
4-7. Поведение	76
4-8. Идентичность	77
4-9. Жизненный цикл	78
4-10. Я не прощаюсь...	78
ЧАСТЬ 2. РЕАЛИЗАЦИЯ ООП В ОБЪЕКТ PASCAL	79

5. ОБЩИЕ СВЕДЕНИЯ	80
6. ОБЪЕКТЫ	83
6-1. Объявление	83
6-1-1. Область видимости private	86
6-1-2. Область видимости protected	87
6-2. Поля	89
6-3. Поля class или static	91
6-4. Конструкторы и деструкторы	93
6-5. Методы	96
6-5-1. Объявление	96
6-5-2. Вызов метода	98
6-5-2-1. Обычные статические методы	98
6-5-2-2. Виртуальные методы	101
6-5-2-3. Абстрактные методы	104
6-5-2-4. Методы class и static	107
6-5-2-5. VMT и раннее/позднее связывание	110
6-6. Видимость	111
6-7. Как средства разработки помогают программистам	112
7. КЛАССЫ	113
7-1. Описание класса	114
7-1-1. Список компонентов	116
7-1-2. Видимость компонентов класса	118
7-1-3. Предварительное объявление классов	121
7-2. Обычные и статические поля	123
7-2-1. Обычные поля/переменные	125
7-2-2. Поля/переменные класса	126
7-3. Создание экземпляра класса	128
7-4. Уничтожение класса	131
7-5. Методы	134
7-5-1. Объявление	134
7-5-2. Вызов	136
7-5-3. Виртуальные методы	137
7-5-3-1. Виртуальные методы в Delphi	141
7-5-3-2. Абстрактные методы	143
7-5-4. Методы class	143
7-5-5. Конструкторы и деструкторы class	146
7-5-6. Статические class-методы	148
7-5-7. Методы-сообщения	151
7-5-7-1. Реализация методов-сообщений	152

7-5-7-2. Отправка сообщений (Message dispatching)	152
7-5-7-3. Примеры методов-сообщений	154
7-5-8. Использование наследования	156
7-5-9. Идентификатор Self	158
7-6. Свойства	160
7-6-1. Определение	160
7-6-1-1. Спецификаторы доступа	161
7-6-1-2. Повторение пройденного	164
7-6-2. Индексированные свойства	165
7-6-3. Свойства-массивы	167
7-6-4. Свойства по умолчанию	172
7-6-5. Свойства Published	172
7-6-6. Хранение данных	173
7-6-7. Переопределение свойств	174
7-7. Class-свойства	175
7-8. Вложенные типы, константы и переменные	176
7-9. Операторы для работы с классами	178
7-9-1. Оператор is	178
7-9-2. Оператор as	179
7-10. TObject и TClass	180
7-11. Совместимость типов классов	180
7-12. Объектные типы	181
7-13. Ещё немного о конструкторах	182
7-14. Ещё немного о деструкторах	184
8. ИНТЕРФЕЙСЫ	186
8-1. Определение	186
8-2. Идентификация интерфейса: GUID	187
8-3. Реализация интерфейса	189
8-4. Наследование интерфейса	190
8-5. Делегирование интерфейса	192
8-6. Интерфейсы и COM	196
8-7. CORBA и другие интерфейсы	196
8-8. Подсчет ссылок	197
9. ОБОБЩЁННЫЕ ТИПЫ	199
9-1. Введение	199
9-2. Определение обобщённого типа	199
9-3. Специализация обобщённого типа	203
9-4. Ограничения обобщённого типа	205
9-5. Совместимость с Delphi	207
9-5-1. Элементы синтаксиса	207

9-5-2. Ограничения типа Record	208
9-5-3. Перегрузка типа	209
9-5-4. Вопросы пространства имен	210
9-5-5. Вопросы области видимости	210
9-6. Совместимость типов	211
9-7. Использование внутреннего default	214
9-8. Слово об области видимости	215
9-9. Перегрузка операторов и обобщённые типы	218
10. РАСШИРЕННЫЕ ЗАПИСИ	221
10-1. Определение	221
10-2. Перечисления расширенных записей	224
11. ПОМОЩНИКИ КЛАССОВ, ЗАПИСЕЙ И ТИПОВ	227
11-1. Определение	227
11-2. Ограничения для помощников по классам	229
11-3. Ограничения для помощников записи	230
11-4. Советы для помощников типов (простых)	231
11-5. Примечание по области и времени жизни для помощников по записям и типам	233
11-6. Наследование	236
11-7. Использование	237
12. КЛАССЫ OBJECTIVE-PASCAL	241
12-1. Введение	241
12-2. Объявление классов в Objective-Pascal	242
12-3. Формальное объявление	245
12-4. Выделение и освобождение памяти для экземпляров	247
12-5. Определение протокола	248
12-6. Категории	249
12-7. Область имен и идентификаторы	251
12-8. Селекторы	252
12-9. Тип id	252
12-10. Перечисление в классах Objective-C	253
13. КОГДА НЕ НУЖНО ИСПОЛЬЗОВАТЬ ООП	255
ЧАСТЬ 3. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ	256
14. КАК ЭТО ДЕЛАЕТСЯ	257
14-1. С чего начать	257
14-2. Пример программы	258
14-2-1. Объектная модель	258
14-2-2. Постановка задачи	259
14-2-3. Структура программы	260

14-2-4. Структура объекта	261
14-2-5. Пример программы с использованием ООП	261
14-3. Снова здорово - опять теория	265
14-3-1. Как создавались программы в доООПные времена	266
14-3-2. Как создаются программы сегодня	267
14-3-3. Про варианты использования	268
14-3-3-1. Действующие субъекты	269
14-3-3-2. Варианты использования	270
14-3-3-3. Сценарии	271
14-3-3-4. От вариантов использования к классам	271
14-4. Ещё один пример программы	272
14-4-1. Начало	273
14-4-2. Развитие	274
14-4-3. Построение	275
14-4-3-1. Ищем варианты использования	275
14-4-3-2. Описываем варианты использования	276
14-4-3-3. Создаём сценарии	278
14-4-3-4. Превращаем варианты использования в классы	281
14-4-3-5. Пишем код	283
15. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ И ЗАДАЧИ	285
15-1. Диалоговое окно в консольном приложении	285
15-2. Квадраты	289
15-3. Массив строк	291
ЗАКЛЮЧЕНИЕ	297
ИСТОЧНИКИ ИНФОРМАЦИИ	298
ОБ АВТОРЕ	299

ПРЕДИСЛОВИЕ

Эта книга является очередной в серии книг о программировании для начинающих. Я задумал эту серию с целью предоставить читателю набор книг, которые помогут ему с полного нуля стать программистом хорошего уровня.

Книги и видеокурсы по программированию, которые полностью готовы на текущий день, вы можете найти на моём сайте <https://info-master.su>.

До сих пор в качестве примеров для своих книг я использовал консольные приложения.

Конечно, я понимаю, что начинающему программисту как можно скорее хочется начать создавать программы с графическим интерфейсом. И, возможно, многие от меня ждали, что следующей книгой будет именно книга о программировании оконных приложений для Windows.

Но нет. Не на того напали)))

Я буду последователен. Поскольку глубоко убеждён, что сначала нужно хорошо изучить теорию, и только потом - начать создавать на практике современные качественные программы.

И я также глубоко убеждён, что у тех, кто прочитал только книгу [Основы программирования](#), ещё недостаточно знаний, чтобы создавать качественные программы с графическим интерфейсом. Я подчёркиваю - качественные.

И, поскольку свою миссию я вижу в том, чтобы, ни много ни мало, сдвинуть в нашей стране эпоху дилетантства в сторону профессионализма, я буду постепенно, шаг за шагом, от книги к книге, вести вас по пути профессионала. Разумеется, **если вы этого захотите**.

Как говорили древние мастера боевых искусств: "Прежде, чем научиться ходить - научись правильно стоять".

Да, это сложно, нудно, утомительно. Но без этого вас сбросит с пути малейший ветерок, не говоря уже о шторме.

Поэтому все примеры программ в книге, которую вы сейчас читаете, также будут консольными приложениями.

А теперь немного о содержании.

Объектно-ориентированное программирование (ООП) - это парадигма программирования, в которой концепциями являются понятия объектов и классов [1].

Пока на этом остановимся, так как терминологию мы будем изучать в соответствующих разделах.

Также скажу, что создание современных программ практически невозможно без ООП. Хотя у этого подхода есть и критики, которые утверждают, что без ООП вполне можно обойтись.

Да, можно. Также как можно, например, обойтись без зубной щётки. Но мы ведь привыкли к комфорту и чистоте. Не так ли?

Во всяком случае, практически все современные средства разработки, такие как Visual Studio (Visual C++, Visual C#, Visual Basic и др.), RAD Studio (Delphi, C++ Builder и др.), Lazarus и другие используют именно ООП.

Конечно, можно состряпать какую-нибудь программку и не вникая в суть ООП. Однако это любительский подход. Мы же с вами хотим [стать профессионалами](#). Поэтому ООП (хотя бы в общих чертах) мы просто обязаны изучить и понять.

Этим мы и будем заниматься в данном курсе.

ЧАСТЬ 1.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ООП

1. ПАРАЛЛЕЛЬНЫЕ МИРЫ

Этот раздел для тех, кто не любит сильно долго изучать нудную теорию. Я назвал его немного странно. Почему именно так - поймёте позже.

Из этого раздела вы быстро узнаете самые-самые основные сведения об ООП. Возможно, кому-то этого будет вполне достаточно. А для тех, кто захочет узнать больше, я подготовил следующие разделы...

1-1. Мир людей

Мы живём с вами в мире людей. Разумеется, есть ещё леса и поля, а также разные зверушки и бактерии. Однако мы воспринимаем мир как люди, а не как бактерии (хотя некоторые люди, пожалуй, недалеко ушли по уровню развития от бактерий).

Поэтому, что бы мы ни делали, мы стараемся, чтобы это было нам понятным с точки зрения восприятия окружающего нас мира. Отчасти поэтому создавать сложные программы может не каждый - мир программ слишком непонятен большинству людей.

Давайте возьмём совершенно обычный и простой предмет - фонарик на батарейках, и попробуем на его примере разобраться с азами объектно-ориентированного программирования. Сначала посмотрим, как создаётся фонарик в мире людей.

Как видно из рисунка 1-1, сначала создаётся **чертёж**, на котором разработчик определяет внешний вид, размеры и некоторые другие свойства фонарика. Если быть точным, то понадобится ещё электрическая схема (ведь фонарик на батарейках) и некоторая другая конструкторская документация. Но для упрощения будем считать, что для изготовления фонарика достаточно одного чертежа.

Далее, имея чертёж, мы можем сделать сколько угодно похожих друг на друга фонариков - **готовых изделий**. Однако ключевое слово здесь - "похожих". То есть не обязательно одинаковых. Например, фонарики могут отличаться

- Цветом.
- Типом элементов питания (обычные батарейки или аккумуляторы).
- Типом лампочки (светодиодная или лампа накаливания).
- Ну и другими **свойствами**.

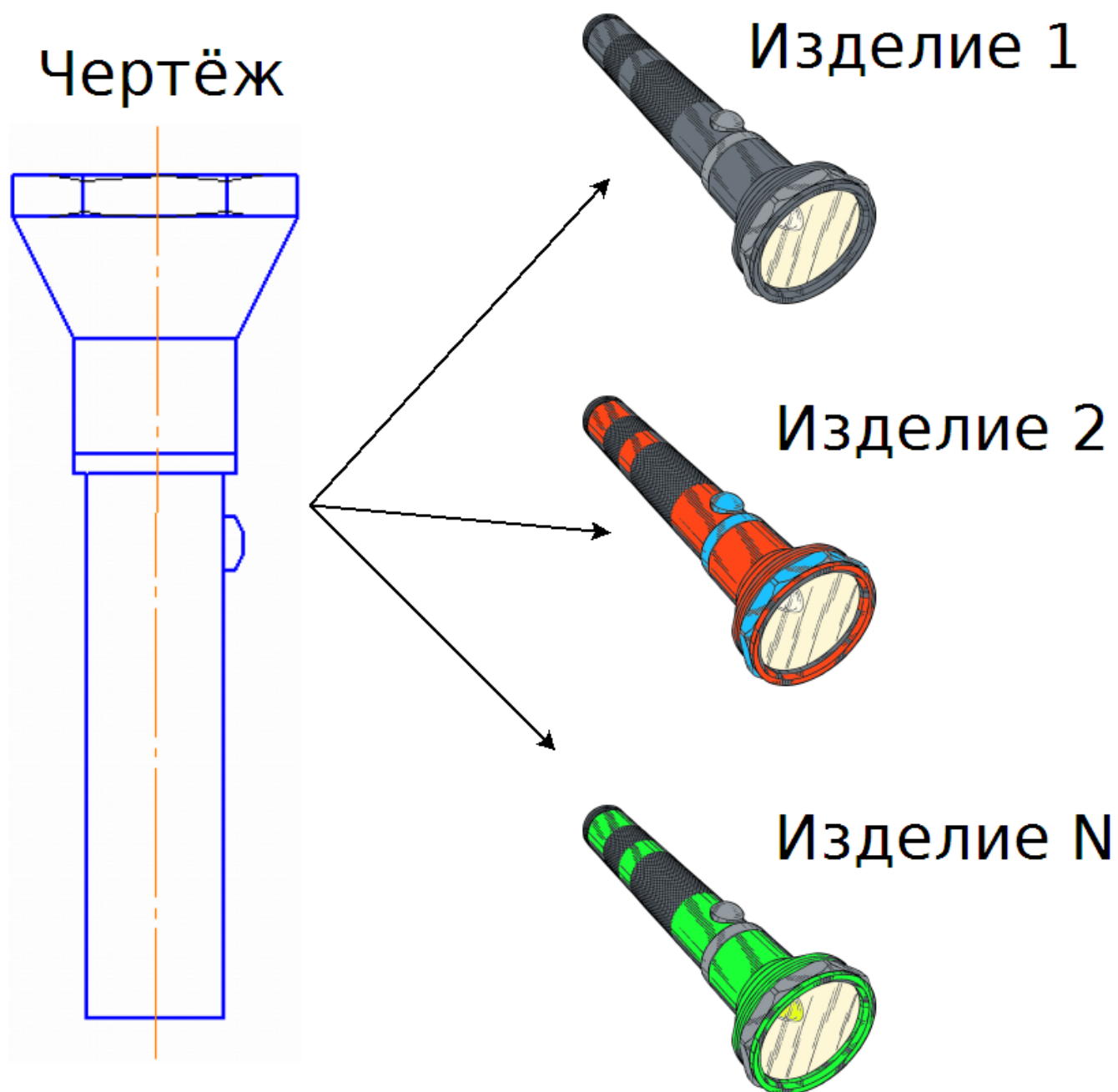


Рис. 1-1. Фонарик в нашем мире (мире людей).

По своим **функциям** одинаковые фонарики, как правило, не отличаются. В простейшем случае у фонарика только две функции: ВКЛЮЧИТЬ и ВЫКЛЮЧИТЬ лампу.

Но вот тем, как эти функции **работают**, фонарики отличаться могут. Например, в старой модели фонарика была обычная кнопка с контактами, которые напрямую включали лампу, а в ходе работы над усовершенствованием фонарика инженеры решили добавить микроконтроллер, и кнопка уже подаёт сигнал на микроконтроллер, который управляет лампой.

При этом чертёж фонарика остался прежним, а его электрическая схема **изменилась**. Это важный вопрос, по мере прочтения книги вы поймёте, почему.

Ну а теперь переместимся в параллельный мир...

1-2. Мир машин

Мир умных машин немыслим без программирования. Ум машины как раз таки определяется качеством программы, которая управляет этой машиной. И у такой умной машины не может быть простой программы. Программы умных машин не просто сложные, а очень-очень сложные.

Создавать сложные программы тоже очень сложно. Поэтому на протяжении всей истории развития программирования постоянно создаются новые средства разработки, языки программирования и способы написания программ, которые позволяют упростить такое сложное дело, как программирование. Именно так и появилось ООП. Но в историю его создания погрузимся чуть позже, а пока посмотрите на рис. 1-2.

Итак, если мы захотим создать **виртуальный** фонарик в **виртуальном** мире с помощью ООП, то это будет выглядеть примерно так, как показано на рис. 1-2.

Как видите, по сравнению с миром людей мало что изменилось. По сути изменились только названия, а всё остальное осталось прежним.

В этом и заключается главный замысел ООП - **стереть границы между параллельными мирами (миром людей и миром программ)**, чтобы программисту было проще переносить объекты настоящего мира в мир виртуальный.

И ещё одно важное замечание по этому рисунку: объекты и классы - это основные понятия ООП, на которых, можно сказать, держится этот способ разработки программ.

А теперь чуть более глубоко погрузимся в теорию...

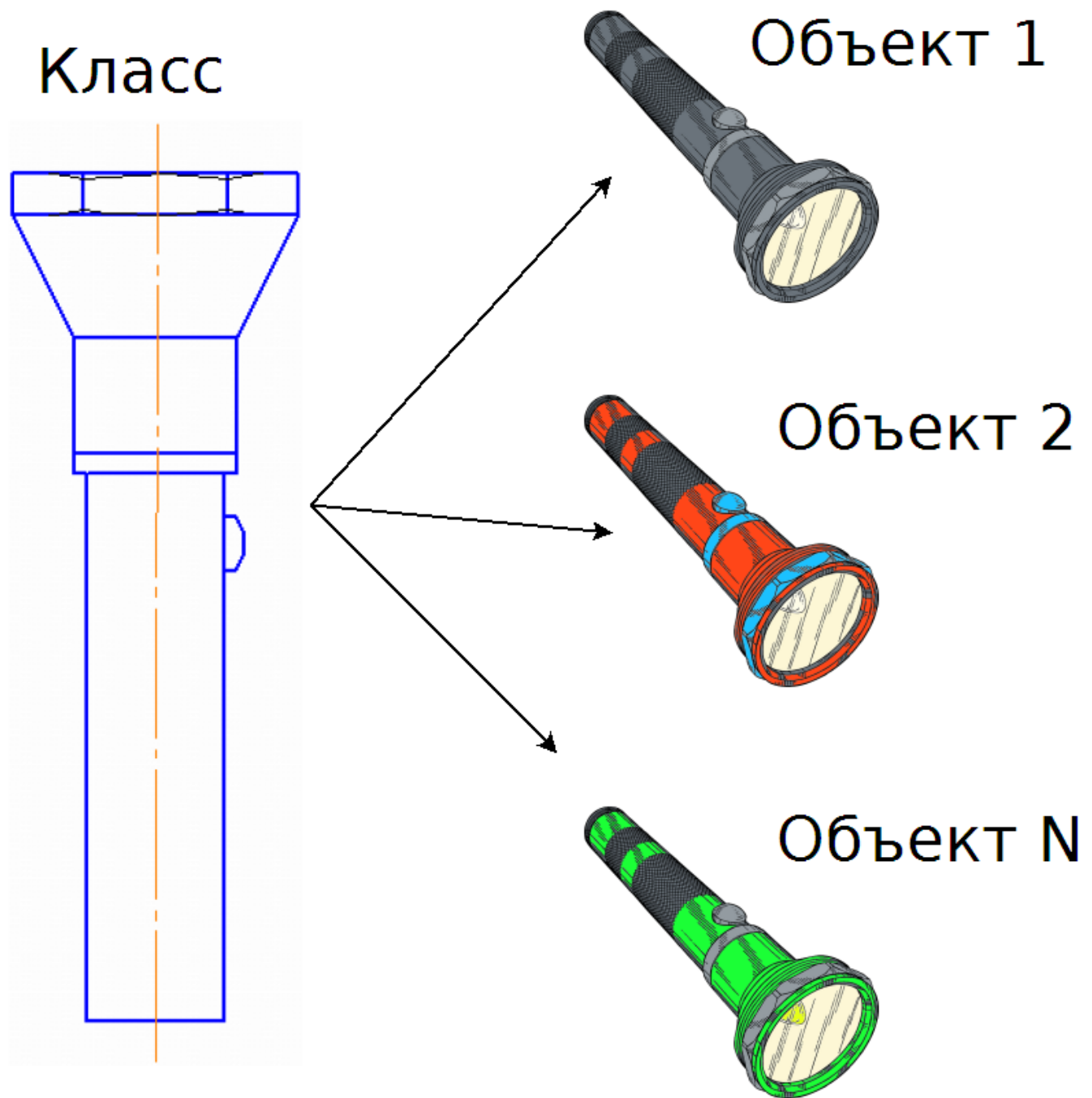


Рис. 1-2. Фонарик в параллельном мире (мире машин).

1-3. Что такое ООП

Объектно-ориентированное программирование (ООП) - это **парадигма программирования**, в которой **концепциями** являются понятия объектов и классов [1].

Парадигма программирования - это совокупность идей и понятий, которые определяют стиль написания компьютерных программ (подход к программированию). Это способ создания **концепций**, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Концепция (от латинского *conceptio* — понимание, система) - определённый способ понимания, трактовки каких-либо явлений, основная точка зрения, руководящая идея для описания этих явлений.

В нашем случае **концепция** - это некий принцип (замысел) написания программного кода.

А теперь попробуем дать определение ООП более простыми словами (потому что лично у меня такие заимствованные из иностранных языков слова, как "парадигма" и "концепция" вызывают жгучее желание попросить автора говорить по-русски, чтобы его могли понять не только профессора и кандидаты каких-нибудь наук). Итак,

Объектно-ориентированное программирование (ООП) - это способ написания компьютерных программ, основанный на использовании объектов и классов, позволяющий сблизить мир людей с миром программ. То есть ООП даёт такой способ написания исходных кодов, при котором объекты реального мира становятся объектами в программе.

Здесь ещё осталось непонятным, что такое объекты и классы. Но это мы уже изучим в следующих разделах.

1-3-1. Основной замысел ООП

То, что мы изучали и делали в книге [Основы программирования](#), называется **процедурным программированием**.

Процедурное программирование в своё время было прорывом. Однако требования к программам ужесточались. Программы становились всё сложнее. Создавать программы с использованием только лишь подпрограмм становилось всё утомительнее.

Придумали разбивать программы на модули. Это немного помогло. Но оставалась большая неразрешимая задача: программа была “монолитной”. Иногда, чтобы внести небольшие изменения, приходилось переделывать чуть ли не всю программу.

Замысел ООП заключался в том, чтобы разбить программу на несколько модулей. Но не как попало, а по неким функциональным признакам. Впрочем, это можно было делать и без ООП.

Однако чего нельзя (или почти нельзя) делать без ООП, так это использовать объекты и классы (подробнее об этом позже).

Объекты и классы позволяют более просто работать со сложными структурами данных.

В книгах об ООП часто для примера проводят аналогию с автомобилем. Не будем отходить от традиций)))



Рис. 1-3. Инопланетянин в автомобиле

Итак, допустим у вас есть автомобиль. Вы ездили на нём какое-то время, но ситуации на дорогах изменились, и мощности двигателя вам стало не хватать.

Если ваш автомобиль “сделан” по принципу процедурного программирования, то вам придётся менять автомобиль целиком. Потому что такой автомобиль - это “монолит” - единый неразборный узел.

А вот если ваш автомобиль сделан по принципу ООП, то вы сможете легко заменить не только двигатель, но и любой другой узел.

Справедливости ради надо сказать, что “заменить двигатель” можно и в процедурном программировании. Однако это будет замена “через выхлопную трубу”.

В ООП же это будет замена через открытие (снятие) капота. Тоже, конечно, непросто. Но намного проще, чем через выхлопную трубу)))

В общем, ООП было придумано изначально именно для того, чтобы упростить жизнь программистам - позволить им разбивать программы на крупные объекты, и при необходимости менять эти объекты (или изменять их характеристики) “лёгким движением руки”.

В процедурном программировании выполнение программы, как правило, линейно. То есть выполняется некоторая последовательность инструкций от “А” до “Я” - от ввода данных до вывода результата.

В ООП такая линейность встречается редко. В ООП основной упор делается на структуру и смысл данных.

То есть данные в программе представлены не как попало, **НЕ** в виде неупорядоченного набора переменных, а в виде классов и объектов (подробнее об этом дальше).

С некоторым подобием объектов вы уже знакомы - это записи в [Паскале](#) или структуры в [C++](#). Эти типы данных существенно упрощают жизнь программисту. Однако они лишены некоторых важных свойств, которые имеются у объектов и классов. Поэтому, используя только записи (структуры), нельзя воплотить в жизнь все замыслы ООП.

Но, если вы помните, что такое записи, то вы также помните, что они позволяют структурировать данные.

Структурирование (упорядочивание) данных - это одна из задач ООП.

Создав какой-то объект, вы можете многократно использовать его в своей программе (а при желании - и в других программах). При этом вы **будете работать с объектом, а не с набором переменных**.

При этом программист часто даже не знает, как работают те или иные функции объекта. Потому что многие из этих функций просто скрыты от программиста.

Если вспомнить наш автомобиль, то, чтобы успешно ездить на нём, вам не надо знать принципы работы двигателя внутреннего сгорания. Принципы работы автомобиля скрыты от вас. Поэтому вам достаточно изучить только назначение приборов и педалей. Да научиться крутить руль. И всё - поехали.

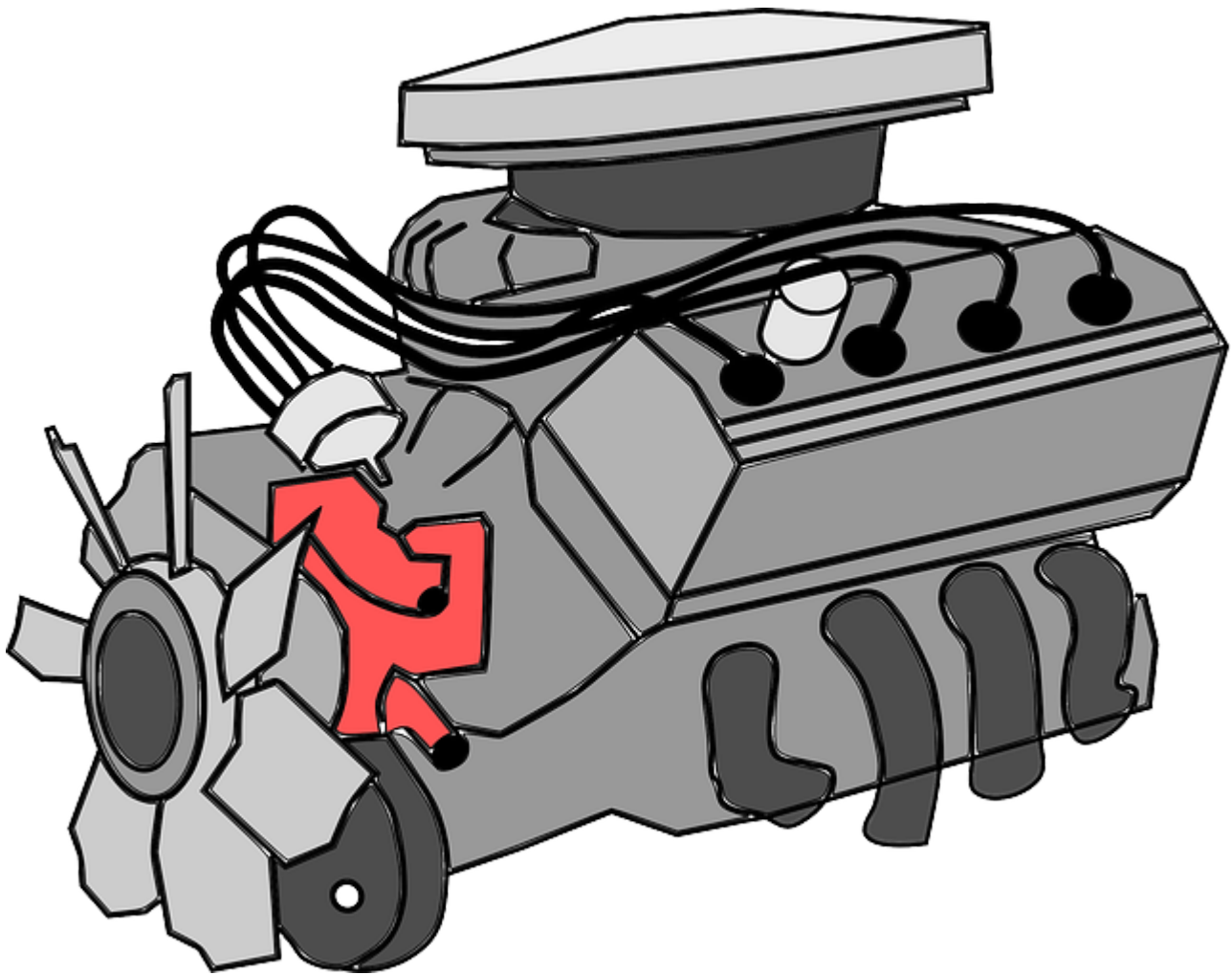


Рис. 1-4. Двигатель.

Также и в ООП. Вы можете с такой же лёгкостью использовать какой-то объект или класс, созданный другим программистом. И даже не подозревать, что этот добрый программист потратил на создание этого класса множество часов и написал сотни, а то и тысячи строк исходного кода.

В процедурном же программировании, прежде чем использовать чужой модуль, вам пришлось бы досконально разобраться в том, как он работает.

Это то же самое, как если бы вам пришлось полностью изучить конструкцию автомобиля, прежде чем получить права.

А теперь представьте, что у вас был бензиновый двигатель, а новую машину вы купили с дизелем. И всё - ваши права уже не действительны. Потому что принципы работы дизеля вы не знаете...

Ну пока всё. Это, так сказать, основной замысел ООП “в первом приближении”. Дальше мы будем разбирать всё более подробно. Но сначала ещё немного “лирики”...

1-3-2. Сторонники и противники ООП

Как у любого хорошего дела, у замысла ООП есть сторонники и противники.

Если хотите знать моё мнение, то я считаю, что для создания прикладных программ лучше ООП пока ничего не придумали.

В то же время есть языки программирования, которые просто не поддерживают ООП (например, ассемблер). И есть задачи, для решения которых ООП не только не поможет, но и помешает.

Поэтому надо просто включать мозг - и выбирать язык программирования, технологии программирования и парадигмы программирования, которые наиболее подходят к вашей задаче, и которые вы наиболее хорошо знаете.

Противников у ООП немало. И некоторые из них доказали, что какой-либо ощутимой разницы в производительности программиста между использованием ООП и процедурным программированием не существует.

Однако, по моему скромному мнению, все эти доказательства из разряда **“Сократ смертен. Все кошки смертны. Следовательно, Сократ - кошка”**.

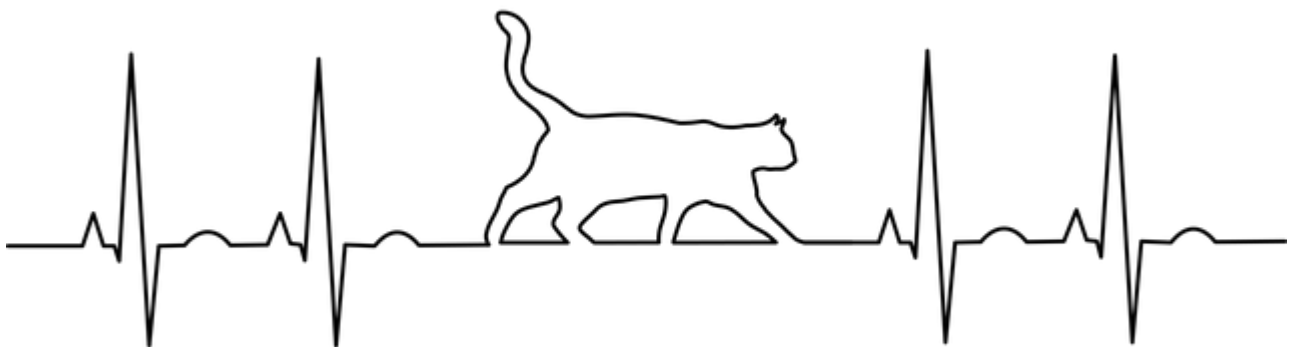


Рис. 1-5. Кошка.

Я ещё могу допустить, что это справедливо при создании консольных приложений. Но как без ООП создавать программы в визуальных средствах разработки? Интересно было бы посмотреть.

В общем, гневных ругательств в адрес ООП поступает достаточно. Но, несмотря на это, ООП используется очень широко в программировании. И, я уверен, что будет использоваться ещё долго.

Доводы против использования ООП вы можете найти в источнике [1]. Я же не настолько умен, чтобы заниматься сравнением и анализом разных парадигм программирования. И не настолько глуп, чтобы приводить в этой книге чужие мысли.

1-3-3. Недостатки структурного программирования

Как я уже говорил, задачи программирования постоянно усложняются. При разработке программного обеспечения очень часто бывает, что задача оказывается сложнее, чем казалось, и сроки сдачи проекта переносятся. Всё новые и новые программисты привлекаются для работы, а расходы резко увеличиваются. Окончание работы вновь переносится, и в итоге проект тихо умирает.

Как бы умело ни применялся структурный подход, он не позволяет в достаточной степени упростить большие сложные программы.

Процедурно-ориентированные языки имеют два основных недостатка:

1. Доступ функций к глобальным данным неограничен.
2. Разделение данных и функций (а это основа структурного подхода) плохо отображает картину нашего [мира людей](#).

1-3-3-1. Доступ к данным

В процедурной программе, написанной, к примеру, на языке C, существует два типа данных.

1. **Локальные данные.** Находятся внутри подпрограммы и предназначены для использования только этой подпрограммой. Из другой подпрограммы или даже из основной программы доступ к этим данным напрямую невозможен.
2. **Глобальные данные.** Если требуется использовать одни и те же данные совместно несколькими подпрограммами, то данные должны быть объявлены как глобальные. Любая подпрограмма имеет доступ к глобальным данным.

Большие программы обычно содержат очень много функций и процедур, а также глобальных переменных. Отсюда вытекает первый недостаток процедурного подхода: количество возможных связей между глобальными переменными и подпрограммами может быть очень большим. То есть **одну и ту же глобальную переменную может изменить любая подпрограмма**.

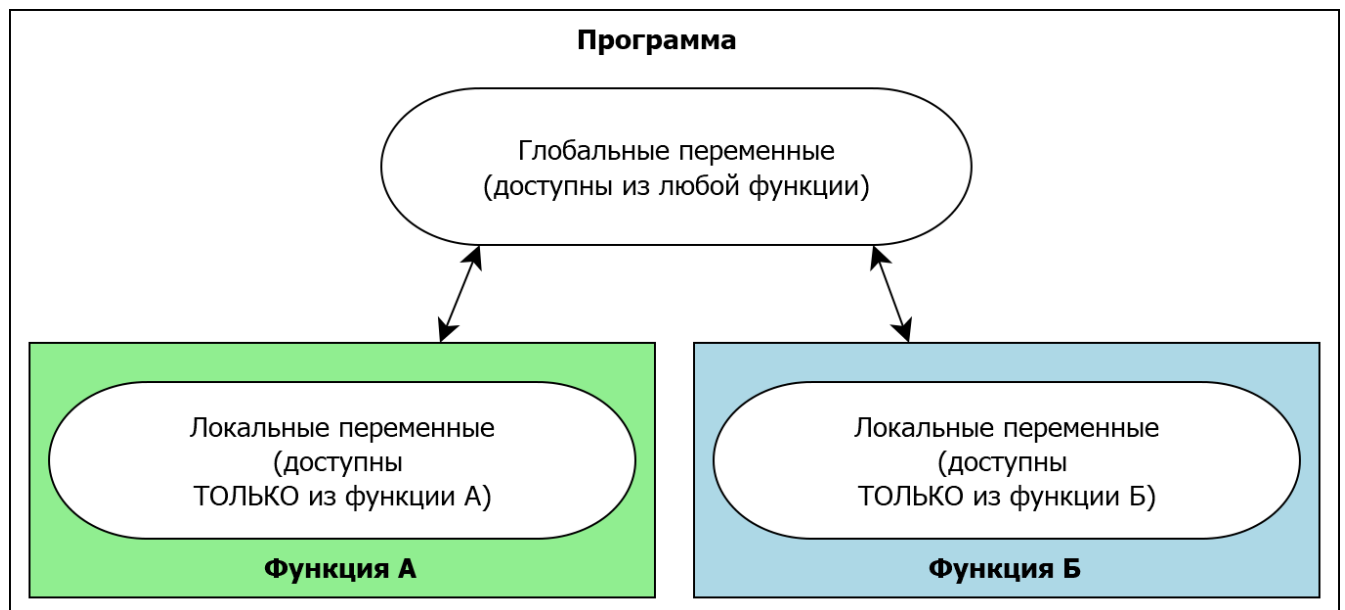


Рис. 1-6. Доступ к данным в процедурном программировании.

Это приводит к тому, что

1. Структура программы сильно усложняется и становится трудной для понимания.
2. Если потребуется внести изменения в программу, то это может потребовать очень больших трудозатрат, так как изменение структуры глобальных данных может потребовать переписывания всех подпрограмм, которые используют эти данные.
3. Когда изменения вносятся в глобальные данные больших программ, бывает крайне сложно определить, какие функции необходимо исправить. И тем более невозможно сделать это быстро. Даже в том случае, когда это удастся сделать, из-за многочисленных связей между подпрограммами и данными исправленные функции начинают "глючить" и неправильно работать с другими глобальными данными. Так что любое изменение влечет за собой далеко идущие неприятные последствия.

Эти неприятности сильно усложняют работу программиста.

1-3-3-2. Суровая действительность

Второй, более важный, сложный и требующий решения вопрос процедурного подхода заключается в том, что такой подход очень слабо отражает положение дел в [мире людей](#).

В нашем мире мы имеем дело с физическими объектами, такими, например, как люди или машины. Эти объекты нельзя отнести ни к данным, ни к функциям, потому что вещи в мире людей представляют собой совокупность свойств и поведения. То есть каждая вещь обладает какими-то свойствами (размер, цвет, запах...) и что-то делает в этом мире (издаёт звуки, запахи, перемещается в пространстве, просто лежит где-то в углу...).

Свойства - это особенности объекта (например, наибольшая скорость автомобиля).

Поведение - это некоторое действие объекта в ответ на внешнее воздействие. Например, отклик автомобиля на нажатие педали тормоза.

Поведение похоже на работу функции - если вы вызываете функцию, то она выполняет какие-то действия. Но в процедурном подходе подпрограммы отделены от данных (то есть **поведение отделено от свойств**), поэтому ни отдельно взятые данные, ни отдельно взятые функции не способны достоверно представить объекты мира людей в исходных кодах программы. И это очень сильно усложняет создание больших программ.

Если продолжить сравнение с автомобилем, то при процедурном подходе, пытаясь увеличить скорость, нам бы пришлось нажимать на педаль газа (вызывать функцию ускорения), а скорость смотреть не на спидометре, куда приходят данные непосредственно с датчика скорости автомобиля, а на навигаторе, который получает данные о скорости через спутник.

Может, сравнение не очень удачное и/или слишком сложное, но, надеюсь, что прочитав следующие разделы, вы его поймёте.

1-3-4. Языки программирования

Почти все современные языки программирования (за исключением, пожалуй, разных ассемблеров) поддерживают ООП.

Как ООП сделано в определённом языке программирования - см. в документации на этот язык.

В этой книге все примеры будут на языке Паскаль (с описанием некоторых особенностей различных реализаций Паскаля, таких как Object Pascal, [Delphi](#), [FreePascal](#)/[Lazarus](#)).

А теперь пора переходить к терминологии и более глубоко погружаться в теорию...

2. ОСНОВНЫЕ ПОНЯТИЯ ООП

**На этом ознакомительный
фрагмент заканчивается.
Полную версию книги вы можете
приобрести здесь:**

<https://info-master.su/books/oop/>

ОБ АВТОРЕ

На всякий случай представляюсь (вдруг кому интересно).



Это я Поляков Андрей Валерьевич (можно просто Андрей))).

По образованию я инженер. Много лет отработал в области автоматизации (и продолжаю работать).

Но есть у меня и другие интересы. Я пишу книги, создаю обучающие курсы. А также немного (на уровне любопытства) занимаюсь инфо-бизнесом.

Все ссылки на мои контакты можно найти здесь:
<http://info-master.su/contact.php>