

```
class MyComplex {  
    double re;  
    double im;  
public:  
    MyComplex(double re, double im): re(re), im(im) {}  
    MyComplex(MyComplex &image): re(image.re), im(image.im) {}  
    MyComplex() {}  
    MyComplex &operator=(MyComplex &image) {  
        im = image.im;  
        re = image.re;  
        return *this;  
    }  
    double GetRe() const {return re;}  
    double GetIm() const {return im;}  
};
```

ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

ОСНОВНЫЕ КОНЦЕПЦИИ И МЕХАНИЗМЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

- **Объектная модель и ее развитие в современных языках программирования**
- **Иерархическое проектирование типов и программ**
- **Введение в обобщенное программирование**
- **Управляемые среды проектирования**
- **Основы компонентного программирования**
- **Демонстрационные программы, вопросы и упражнения для самостоятельной работы, примеры заданий практикума**

private
ASur



namespace
MString
VTable

УДК 681.3.068(075.8)
ББК 32.973.26-018.1я73
П94

Пышкин Е. В.

П94 Основные концепции и механизмы объектно-ориентированного программирования. — СПб.: БХВ-Петербург, 2005. — 640 с.: ил.

ISBN 5-94157-554-8

Рассматривается понятие объектной модели и анализируются механизмы управления вычислительным процессом, лежащие в основе объектно-ориентированного подхода: классы и интерфейсы, динамическое связывание, обработка исключений, пространства имен. Подробно рассматривается конструирование обобщенных типов и библиотека ввода-вывода применительно к программированию на C++. Содержится информация об управляемом коде, свойствах, делегатах, событиях, специализированных атрибутах, отражении, основах компонентной архитектуры.

Размещенные на компакт-диске демонстрационные программы разработаны с использованием современных языков программирования: C++, Java, C#, Visual Basic.

Для студентов и преподавателей технических вузов

УДК 681.3.068(075.8)
ББК 32.973.26-018.1я73

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Людмила Еремеевская</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Леонид Мирошенков</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн обложки	<i>Игоря Цырульникова</i>
Зав. производством	<i>Николай Тверских</i>

РЕЦЕНЗЕНТЫ:

Жабко А. П., доктор физ.-мат. наук, проф., зав. каф. теории управления СПбГУ
Евдокимов В. Е., к. т. н., доцент каф. информационно-измерительных технологий СПбГПУ

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 15.06.05.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 51,6.

Тираж 3000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-554-8

© Пышкин Е. В., 2005
© Оформление, издательство "БХВ-Петербург", 2005

Оглавление

Предисловие	1
Преподавание: исторический экскурс	1
Основная задача книги.....	3
Благодарности	4
Введение	6
Предпосылки	6
Организация данной книги	6
Исходные тексты программ и листинги программ	9
О соответствии стандарту C++	9
Одно замечание о терминологии	11
ЧАСТЬ I. ОСНОВАНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ	13
Глава 1. Развитие языков как развитие абстрактных моделей	15
1.1. Концепции проектирования и языки программирования	15
1.2. Абстрактные модели, лежащие в основе языков программирования	17
Абстрагирование в языках структурного императивного программирования.....	18
Другие абстрактные модели.....	19
1.3. Сущность объектно-ориентированного подхода	20
1.4. Вопросы и упражнения	25
Глава 2. Элементы объектной модели	26
2.1. Смысл абстрагирования как элемента объектной модели.....	27
2.2. Смысл инкапсуляции как элемента объектной модели.....	30
2.3. Отношения классов	35
Отношение обобщения	36
Отношение ассоциации	40
Отношение зависимости.....	46
Отношение реализации	47

2.4. Типы структурных иерархий.....	50
Структурная иерархия "is-part-of" (агрегирование как разновидность ассоциации)	50
Структурные иерархии "is-a" и "is-like-a"	51
2.5. Иерархии классов и модули: модель С++	58
2.6. Вопросы и упражнения	62

ЧАСТЬ II. КОНСТРУИРОВАНИЕ ТИПОВ..... 63

Глава 3. Класс как основной механизм абстракции..... 65

3.1. Структура и организация определения класса.....	65
Элементы-данные и элементы-действия	66
Определение класса	66
Управление доступом к членам класса	67
Класс <i>QueueInt</i> : реализация инкапсуляции.....	68
Порядок объявления открытых и закрытых членов класса	70
Важное замечание об инкапсуляции	71
3.2. Инициализация объектов. Конструкторы	72
Конструктор как инструмент гарантированной инициализации	72
3.3. Спецификация и реализация класса.....	75
Спецификация класса	76
Реализация класса.....	77
3.4. Альтернативные способы инициализации объектов.	
Перегрузка конструкторов.....	81
Понятие о конструкторе по умолчанию	83
Права доступа в связи с конструкторами	84
Конструкторы как неявные преобразования	84
3.5. Разрушение среды функционирования объектов. Деструкторы	87
3.6. Основные возможности в связи с определением членов класса	90
Константные члены класса.....	91
Константные данные.....	91
Константные функции.....	92
Статические константы, переменные и функции	92
Члены класса, требующие обязательную инициализацию.....	97
Инициализация статических членов класса.....	99
Понятие о константном указателе <i>this</i>	100
Встроенные функции-члены класса.....	105
3.7. Перегруженные функции и операции-члены класса	106
Перегрузка методов класса	106
Перегрузка арифметических операций	108
Перегружаемые операции-члены класса.....	108
Перегружаемые операции, не являющиеся членами класса.....	110
Перегружаемые операции-друзья класса	111
Перегрузка операций ввода-вывода.....	111
Перегрузка других операций	112
Несколько замечаний по поводу перегрузки операций <i>new</i> и <i>delete</i>	118
3.8. Дружественный доступ. Локальные и вложенные классы	119
Несколько дополнительных замечаний о дружественном доступе	120
Локальные и вложенные классы.....	121

3.9. Копирование объектов класса	123
Проблемы копирования объектов и понятие поверхностного копирования.....	123
Реализация корректной семантики копирования объектов.....	126
Копирующий конструктор и перегрузка операции присваивания	126
Запрет копирования объектов данного класса	133
3.10. Вопросы и упражнения	135

Глава 4. Наследование и иерархии классов

4.1. Наследование как основная форма отношения обобщения	142
Можно ли обойтись без наследования	142
Какие преимущества получает программист, используя отношение наследования классов.....	144
4.2. Методы производного класса в отношении к методам базового класса	151
Переопределение функций-членов базового класса в производном классе	151
Инициализация объектов производных классов.....	153
Наследование конструкторов и операций	155
Порядок вызова конструкторов и деструкторов в связи с иерархией классов.....	156
4.3. Управление доступом к членам класса в связи с наследованием.....	158
Наследование как многократное использование интерфейса	163
4.4. Реализация наследования: модели, отличные от C++	164
Однокоренная иерархия классов	164
Соккрытие имен.....	165
Неизменные методы и классы	167
4.5. Вопросы и упражнения	168

Глава 5. Объекты классов и полиморфизм.....

5.1. Понятие о статическом и динамическом связывании	173
Статическое связывание. Установление типов объектов во время компиляции	174
Динамическое связывание. Установление типов объектов во время выполнения.....	177
5.2. Виртуальные функции — механизм реализации полиморфизма в C++.....	177
Общие сведения	178
Пример со студентами и аспирантами	178
Пример с геометрическими фигурами	181
Требования к сигнатуре виртуальной функции	184
Таблица виртуальных функций — основной элемент механизма реализации позднего связывания	185
Таблица виртуальных функций.....	185
Выбор корректной версии виртуальной функции	186
Инициализация указателя на таблицу виртуальных функций.....	187
Простой пример для иллюстрации затрат памяти на реализацию полиморфизма.....	188
Чисто виртуальные функции и абстрактные классы	192
Виртуальные функции: некоторые подробности.....	193
Разрушение среды функционирования полиморфных объектов.	
Виртуальные деструкторы.....	193
Вызов виртуальной функции из функции-члена класса.....	197

Можно ли в производном классе определить не виртуальную функцию вместо виртуальной функции базового класса?	200
Функции-не члены класса, работающие подобно виртуальным функциям.....	201
5.3. Динамическое связывание и приведение типов.....	204
RTTI (определение типа во время выполнения) и понижающее приведение типов	204
RTTI: проблемы использования	207
5.4. Несколько замечаний о преобразованиях типа в C++	211
Преобразование типов во время компиляции (преобразование <i>static_cast</i>).....	211
Преобразование типов во время выполнения (преобразование <i>dynamic_cast</i>).....	212
Преобразования для избавления от константности (<i>const_cast</i>).....	213
Преобразование "на свой страх и риск" (<i>reinterpret_cast</i>).....	213
5.5. Вопросы и упражнения	214

Глава 6. Обработка ошибок на основе использования механизма исключений

6.1. Варианты обработки ошибок, не связанные с использованием исключений.....	220
Обработка ошибки на месте	221
Использование возвращаемых значений	221
Использование глобальных переменных или переменных-членов класса, исполняющих роль индикаторов состояния приложения или объекта	224
Использование специально разработанных функций	228
Использование функций обратного вызова	229
6.2. Обработка исключений	234
Основная идея подхода	234
Что такое "исключение" и как оно образуется	236
Исключения следует перехватывать по ссылке	238
Преимущества встраивания в язык обработки исключительных ситуаций.....	241
6.3. Особенности обработки исключений в языке C++	242
Типы исключений.....	242
Использование встроенных типов языка	242
Использование типов, определяемых пользователем	244
Группирование исключений.....	249
Обработчики исключений	252
Порядок записи обработчиков	252
Повторная генерация исключения	252
Использование обработчика <i>catch(...)</i>	255
Когда исключение считается обработанным?	256
Исключения в связи с ошибкой выделения памяти.....	256
Исключения в конструкторах и деструкторах.....	258
Конструкторы и исключения	258
Деструкторы и исключения.....	264
Спецификация исключений.....	268
Неожидаемые и неперехваченные исключения	269
Стандартные исключения	271

6.4. Особенности обработки исключений в языках Java и C#.....	272
Java: обязательная спецификация и обработка исключений в Java.....	273
C#: необязательная обработка исключений.....	274
6.5. Вопросы и упражнения.....	274

ЧАСТЬ III. ОРГАНИЗАЦИЯ И ВЗАИМОДЕЙСТВИЕ ТИПОВ.....281

Глава 7. Множественное наследование и интерфейсы.....283

7.1. Множественное наследование для реализации суммы функциональностей.....	284
Обычное множественное наследование.....	284
Множественное наследование с общим базовым классом.....	292
7.2. Виртуальные базовые классы.....	299
7.3. Дискуссия о множественном наследовании. Понятие интерфейса.....	306
Реализация семантики наследования с интерфейсами на языке C++.....	309
Реализация семантики наследования с интерфейсами на языке C#.....	312
Конфликты имен и сигнатур при реализации интерфейсов.....	316
7.4. Роль интерфейсов в компонентном программировании.....	318
7.5. Вопросы и упражнения.....	319

Глава 8. Пространства имен в связи с модульностью и иерархией.....321

8.1. Области действия и пространства имен.....	321
Программный проект: модули и интерфейсы.....	322
8.2. Пространства имен в C++.....	323
Помещение программных объектов в пространство имен.....	324
Использование программных объектов, объявленных в пространстве имен.....	325
Пространство имен как общая среда реализации программного проекта и пространство имен как внешний интерфейс для пользователей.....	328
Пространства имен: технические подробности.....	329
Разрешение конфликтов имен.....	329
Псевдонимы пространств имен.....	330
Открытость пространств имен.....	330
Объединение пространств имен.....	331
Вложенные пространства имен.....	332
Пространства имен как механизм управления версиями.....	333
Стандартное пространство имен. Пространства имен и код, использующий библиотеки C.....	337
Проблемы пространств имен в C++.....	338
8.3. Пространства имен в Java.....	341
Именованые пакеты.....	341
Создание и использование пакета.....	343
Права доступа в связи с модульной организацией.....	346
8.4. Пространства имен в C#. Введение в архитектуру приложения .NET.....	346
8.5. Вопросы и упражнения.....	350

Глава 9. Введение в обобщенное программирование.....352

9.1. Обобщенное программирование без использования шаблонных функций.....	352
Определение функции сортировки простыми обменами для сортировки массива целых чисел.....	353
Определение универсальной функции сортировки простыми обменами.....	353

9.2. Шаблонные функции	356
Определение шаблонной функции.....	356
Использование шаблонной функции. Инстанцирование.....	358
9.3. Шаблонные классы — основной инструмент параметрического полиморфизма	359
Определение и использование шаблонного класса	359
Параметры шаблонов	363
9.4. Контейнеры и итераторы	364
Проектирование специализированных контейнеров.....	364
Определение специализированного контейнерного класса.....	365
Итерируемые контейнеры.....	365
Использование итератора специализированного контейнера	367
Проектирование стандартных контейнеров	368
Обобщенное представление стандартного контейнера и итератора контейнера.....	369
Определение контейнерного класса в стиле STL.....	370
Обобщенные функции для обработки стандартных контейнеров.....	385
Стандартная библиотека шаблонов (STL).....	386
9.5. Контейнеры, построенные на основе однокоренной иерархии классов (модели Java, C#)	388
9.6. Иерархии шаблонных классов.....	390
Организация размещения данных в памяти	390
Постановка задачи	392
Разработка классов.....	392
9.7. Вопросы и упражнения	403

ЧАСТЬ IV. РАЗВИТИЕ МОДЕЛЕЙ.....405

Глава 10. Организация вычислительного процесса в управляемых средах.....407

10.1. Управляемый и неуправляемый код	407
Управляемый код в Java.....	409
Взаимодействие Java с неуправляемым кодом.....	412
Ограничения, накладываемые управляемым кодом	417
Управляемый код платформы Microsoft .NET.....	418
Код, безопасный по отношению к типам	422
Вместо резюме.....	423
10.2. Встроенные, размерные и ссылочные типы	424
Объекты и встроенные типы.....	428
10.3. Автоматическое удаление объектов (сборка мусора)	431

Глава 11. Развитие объектно-ориентированной модели управления типами.....435

11.1. Реализация полиморфизма	435
Java: полиморфизм по умолчанию	436
C#: спецификатор <i>new</i> , версии классов и виртуальные функции	438
11.2. Свойства.....	445
Поддержка свойств на уровне языка (на примере C#)	446
Особый случай: индексирование.....	447

11.3. Функции обратного вызова и делегаты	450
Понятие делегата и реализация в языке C#.....	451
11.4. Программирование, ориентированное на события	455
Обработка сообщений приложением Windows.....	456
Объектно-ориентированная архитектура приложения	457
Поддержка модели программирования, ориентированной на события, средствами языка	459
11.5. Специализированные атрибуты.....	464
Применение атрибутов, используемых компилятором	465
Использование атрибутов в период выполнения.....	468
Атрибуты, определяемые пользователем	471
11.6. Отражение (рефлексия)	474
Извлечение информации о типе.....	474
Создание экземпляров типов	476
Отражение методов класса.....	477
11.7. Интеграция кода и документации	478
Документирующие комментарии: модель Java	479
Документирующие комментарии: модель C#	481
Глава 12. Введение в компонентное программирование	483
12.1. Понятие о компонентной архитектуре	484
Соккрытие реализации.....	484
Повторное использование кода	485
Динамическая компоновка и совместимость интерфейсов.....	487
12.2. Реализация основных элементов компонентной архитектуры на базе COM... ..	487
Доступ к компоненту посредством запроса интерфейса.....	488
Идентификация интерфейсов, поддерживаемых компонентом	497
Управление временем жизни компонента	497
Явное выделение элементов архитектуры компонентного приложения	506
12.3. COM-сервер и COM-клиент: действующий макет.....	513
Реализация и использование DLL-компонента	513
Фабрики классов и регистрация компонентов.....	517
Преодоление языковой зависимости	531
Основные проблемы COM	539
12.4. Межязыковая и межплатформенная интеграция: проблемы и решения	542
12.5. Элементы компонентной архитектуры .NET Framework.....	545
Управляемые модули и сборки	546
Общая система типов .NET.....	551
Общезыковая спецификация	554
12.6. Вопросы и упражнения	557
Заключение	558
ПРИЛОЖЕНИЯ	559
Приложение 1. Примеры заданий практикума	561
Практикум. Задание № 1	561
Цель задания.....	561

Основные умения, которые должны быть продемонстрированы студентом	561
Постановка задачи	562
Оформление результатов выполнения упражнения.....	562
Практикум. Задание № 2	562
Цель задания.....	562
Основные умения, которые должны быть продемонстрированы студентом	562
Постановка задачи	563
Оформление результатов выполнения упражнения.....	563
Практикум. Задание № 3	564
Цель задания.....	564
Основные умения, которые должны быть продемонстрированы студентом	564
Постановка задачи	564
Оформление результатов выполнения упражнения.....	565
Практикум. Задание № 4	565
Цель задания.....	565
Основные умения, которые должны быть продемонстрированы студентом	565
Постановка задачи	565
Оформление результатов выполнения упражнения.....	566
Практикум. Задание № 5	566
Цель задания.....	566
Основные умения, которые должны быть продемонстрированы студентом	566
Постановка задачи	567
Рецензия на чужой проект.....	567
Обсуждение Вашего проекта	567
Подведение итогов.....	568
Практикум. Курсовой проект	568
Цель задания.....	568
Основные умения, которые должны быть продемонстрированы студентом	568
Постановка задачи	569
Отчетность по заданию и подведение итогов.....	571
Рекомендуемые источники.....	571
Приложение 2. Введение в потоковый ввод-вывод C++.....	573
П2.1. Иерархия классов ввода-вывода (заголовочный файл <code>iostream.h</code>)	574
Класс <code>ios</code> и производные классы.....	575
Класс для вывода <code>ostream</code>	578
Классы для вывода <code>ofstream</code> , <code>ostrstream</code> , <code>ostream_withassign</code>	578
Класс для ввода <code>istream</code>	578
Классы для ввода <code>ifstream</code> , <code>istrstream</code> , <code>istream_withassign</code>	579
Класс для ввода и вывода <code>iostream</code>	579
Классы для вывода <code>fstream</code> , <code>strstream</code> , <code>stdiostream</code>	579
П2.2. Вывод данных.....	579
Вывод данных встроенных типов	580
Вывод символов и строк.....	580
Буферизация	580
Форматирование вывода. Манипуляторы	582
Вывод для типов, не встроенных в C++	586

П2.3. Ввод данных	589
Ввод данных встроенных типов	589
Ввод символов и строк	589
Форматированный ввод. Манипуляторы	590
Ввод для типов, не встроенных в C++	591
П2.4. Файловые потоки	593
Ввод данных из файлового потока <i>ifstream</i>	594
Вывод данных в файловый поток <i>ofstream</i>	596
Использование потока <i>fstream</i> для реализации ввода и вывода при работе с одним и тем же файлом	598
Манипуляторы в связи с файловыми потоками	600
П2.5. Поточковый ввод-вывод в пространстве имен стандартной библиотеки C++	600
Иерархия классов стандартной библиотеки ввода-вывода	601
Класс <i>ios_base</i>	601
Класс <i>basic_ios</i> и структура <i>char_traits</i>	604
Классы <i>basic_istream</i> и <i>istream</i>	606
Классы <i>basic_ostream</i> и <i>ostream</i>	608
Классы <i>basic_iostream</i> и <i>iostream</i>	609
Организация работы с файловыми потоками	609
Резюме	610
Приложение 3. Описание компакт-диска	611
Список источников	614
Предметный указатель	623

ГЛАВА 1



Развитие языков как развитие абстрактных моделей

Язык и мышление теснейшим образом связаны. Если язык обеднеет, обеднеет и мышление. Колоссальное значение имеет терминология, грамматические системы, способствующие самовыражению человека.

Д. С. Лихачев

Объектная модель представляет собой концептуальную основу объектно-ориентированного программирования, поэтому изучение основных элементов объектной модели и их взаимодействия позволяет лучше разобраться в принципах организации объектно-ориентированных программ, а также понять, в чем сходства и различия поддержки объектно-ориентированной парадигмы различными языками.

Прежде чем перейти к изучению элементов объектной модели как таковых, проанализируем вкратце процесс развития языков программирования. Это позволит нам выяснить концептуальные и методологические различия изобразительных средств, предоставляемых программисту языками программирования, и причины успешности и востребованности объектно-ориентированного подхода.

1.1. Концепции проектирования и языки программирования

Объектно-ориентированное программирование (ООП) является, вероятно, самой распространенной в современной проектной практике парадигмой программирования, которая поддерживается большинством современных языков программирования. При этом реализации концепций и механизмов, состав-

ляющих основание ООП, могут отличаться в разных языках. Поэтому изучение ООП предполагает не столько изучение какого-либо конкретного языка программирования, сколько лежащих в его основе концепций и механизмов как таковых. К самым важным вещам, которые должен знать профессиональный программный инженер, относится набор фундаментальных концепций, которые вновь и вновь используются в его работе и которые в первую очередь следует изучать [Meurer, 2001]. С одной стороны, такой подход к изучению программирования позволяет лучше разобраться в проблемах, возникающих перед разработчиками программного обеспечения (ПО), с другой стороны, позволяет лучше понять причины появления и успешного развития различных языков и технологий программирования.

Основная задача любой технологии заключается в том, чтобы предоставить разработчикам средства решения основных проблем проектирования. Широкое применение технологии (которая всегда является реализацией определенных концепций) доказывает ее успешность. В частности, это и означает, что данная технология (а значит, и лежащие в ее основе концепции, модели, методы проектирования) позволяет разработчикам успешно решать актуальные проблемы проектирования.

Вообще говоря, появление новых технологий программирования в большой степени обусловлено трансформацией проблем проектирования ПО с его растущей сложностью. В свою очередь, рост сложности программного обеспечения обусловлен развитием областей жизнедеятельности человека, где находят применение новейшие информационные и компьютерные технологии. Это становится возможным благодаря развитию средств проектирования и возможностей вычислительной техники, удешевлению аппаратных ресурсов и увеличению продуктивности разработчиков, использующих современные средства проектирования. С другой стороны, сейчас уже никто не удивляется тому, что новейшие технологии проектирования зачастую требуют более мощного аппаратного обеспечения для своей эффективной работы (хотя это иногда и связано с неэффективностью алгоритмов, используемых в ходе разработки программ), подталкивая пользователей к модернизации используемого оборудования и вычислительных средств. Таким образом, новое более мощное оборудование способствует более эффективному использованию программных средств, это порождает новые области применения и дает толчок к возникновению новых задач проектирования, требующих для своего решения новых технологий разработки программ [Пышкин, 2003-1]. Графически описанный процесс представлен на рис. 1.1.

Язык программирования — инструмент, вытекающий из концепций проектирования, инструмент, облекающий в синтаксическую форму основное содержание концепций, их суть. Поэтому изучение языка самого по себе вне связи с моделями проектирования, заложенными в основу языка, не влечет улуч-

шения качества проектирования и не обязательно способствует повышению квалификации разработчика.



Рис. 1.1. Прогресс информационных технологий и рост сложности задачи проектирования

Разумеется, нужно отдавать себе отчет в том, что *качество проектирования*, прежде всего, *определяется квалификацией и творческими способностями проектировщиков*, а не моделью программирования: выдающиеся проекты создают выдающиеся проектировщики [Brooks, 1995]. При этом технология проектирования помогает грамотному разработчику продемонстрировать свои умения и построить качественный программный продукт. Таким образом, *необходимо изучать не языки программирования сами по себе, а концепции, выражаемые этими языками* [Eckel, 2000-1].

1.2. Абстрактные модели, лежащие в основе языков программирования

Языки программирования — это всегда некоторые модели (виртуальные вычислительные машины), позволяющие наиболее эффективно использовать возможности вычислительных средств, существенные для конкретных областей применения. По сути дела, при написании программ ориентируются не на вычислительную машину как таковую, а на некоторую абстрактную модель вычислительного устройства [Minsky, 1967]. Качество абстрактной модели, ее соответствие содержанию решаемых задач во многом определяет качество и эффективность проектирования с использованием этой модели. По меткому замечанию Эккеля, компьютеры не столько машины, сколько средства усиления мысли [Eckel, 2000-1].

Сложность задач, которые возможно решить с применением тех или иных концепций проектирования (читай, выражающих их языков), непосредственно связана с уровнем абстракции как при постановке задачи, так и в ходе ее решения. Абстракция является, вероятно, самым мощным интеллектуальным средством познания из тех, что имеются в распоряжении человека [Dahl, Dijkstra, Hoare, 1972]. Со способностью к абстракции связана и способность человека к творчеству. Кроме того, обнаружение общих абстракций и механизмов значительно облегчает понимание сложных систем [Booch, 1994].

Так, языки ассемблера позволяют избежать необходимости программирования в машинном коде, и в этом смысле ассемблеры являются абстракцией вычислительной машины, для которой были спроектированы [Eckel, 2000-1]. Кстати, и сама вычислительная машина, являясь иерархической системой, может рассматриваться на разных уровнях абстракции [Booch, 1994]. В подобной интерпретации языки, основанные на методологии процедурного программирования, можно считать абстракциями ассемблеров.

Таким образом, язык программирования — не только выразитель концепций проектирования, но и уровня абстракции при представлении задач и технических средств, используемых для решения этих задач. *Развитие языков программирования — это, прежде всего, развитие абстрактных моделей, облегчающих и систематизирующих проектирование.*

Абстрагирование в языках структурного императивного программирования

Методология структурного императивного программирования воплощает подход, характеризующийся принципом последовательного изменения состояния вычислителя пошаговым образом с поддержкой концепции структурного программирования. Императивное программирование является первой методологией программирования, поддержанной на аппаратном уровне и ориентированной на класс архитектур фон Неймана [Одинцов, 2002].

Примерами языков-выразителей концепций структурного императивного программирования могут служить языки Fortran, Pascal, C, PL/1 (в последнем, впрочем, был реализован и ряд других важных концепций, например, обработка исключений). Все перечисленные языки основаны на парадигме процедурного программирования, основанной на представлении программы в виде иерархии процедур и функций. В принципе, понятия процедурного и структурного программирования не следует отождествлять, поскольку использование процедурной модели совместимо и с объектно-ориентированным программированием, и с функциональным программированием, и с параллельным программированием [Легалов, 2000]. Развитие структурного про-

граммирования позволило реализовать или развить ряд важных методов проектирования:

- ❑ функционально-иерархическая декомпозиция;
- ❑ структурная организация данных;
- ❑ повторное использование проектных решений;
- ❑ технология тестирования программного обеспечения.

Основная проблема процедурных языков заключается в том, что достигаемый ими уровень абстракции все еще требует от программиста мышления в большей мере в терминах вычислительной машины (пусть даже и виртуальной), чем в терминах задачи, которую ему приходится решать. Качество проектирования определяется в итоге тем, насколько удачно программисту удалось установить соответствие между пространством понятий, характерных для решаемой задачи, и набором изобразительных средств языка, не всегда позволяющих адекватно отобразить эти понятия в рамках машинной модели. Абстрагирование, достигаемое посредством использования процедур, хорошо подходит для описания абстрактных действий, но не предоставляет адекватных языковых средств для описания абстрактных объектов.

Кинг указывает также на проблемы реального проектирования, возникающие в ходе разработки программ на основе функционально-иерархической декомпозиции [King, 1997]:

- ❑ Функциональную модель трудно развивать: эволюция системы, учет новых требований может породить плохо управляемую архитектуру приложения.
- ❑ Функционально-ориентированный код трудно обобщать для многократного использования.

Резюме: языки императивного программирования, основанные на процедурной парадигме, реализуют систематический подход к разработке программного обеспечения, который во многих случаях основан на концепции структурного программирования. При этом задача программиста — перейти от представления о решаемой задаче в терминах, присущих этой задаче, к терминам, присущим модели вычислительной машины.

Другие абстрактные модели

В историческом плане развитие абстрактных моделей не всегда означало переход на более высокие уровни абстракции представления вычислительной машины как исполнительного устройства. Специфические особенности некоторых классов задач способствовали появлению альтернативных моделей.

Примерами альтернатив моделированию вычислительной машины могут служить концепции, реализованные в языках функционального и логического программирования. Примером языка функционального программирования может служить язык LISP, основанный на модели, которая в компактной форме характеризуется постулатом "все задачи, в конечном счете, могут быть сведены к работе со списками". Классическим примером языка логического программирования является язык Prolog, основывающийся на принципе: "все задачи могут быть сведены к цепочке логических рассуждений" [Eckel, 2000-1]. В частности, эти языки нашли применение в задачах искусственного интеллекта и прикладной математики.

Каждый из подобных подходов оказывался приемлемым или даже лучшим для определенного класса задач, но не обладал достаточной универсальностью. Во многих случаях более успешными оказывались языки общего назначения. Даже в области искусственного интеллекта многие коммерческие образцы систем, основанных на знаниях, были разработаны на универсальных языках. В разных источниках приводятся примеры систем, спроектированных полностью или частично на языках Pascal, Fortran, C, PL/1 (см., например, [Forsyth, 1984], [Попов и др., 1990]).

Для решения задач логического управления в настоящее время активно развивается подход, основанный на использовании теории автоматов — автоматное программирование [Шалыто, 1998]. Модели представления данных и поведения системы, положенные в основу этого подхода, фокусируются на абстракциях *состояния* (определяющего статику объекта управления) и *события* (определяющего динамику процесса управления). Автоматное программирование находит применение не только при решении задач управления, но и в таких областях, как синтаксический анализ и трансляция, моделирование систем управления, проектирование программируемых логических контроллеров и др.

Приведенные примеры показывают, что *развитие абстрактных моделей связано не только с переходом на более высокие уровни абстракции при описании вычислительной машины, а также и с развитием других концепций абстрагирования, ориентированных на различные классы задач.*

1.3. Сущность объектно-ориентированного подхода

Сущность объектно-ориентированного подхода заключается, прежде всего, в двух моментах.

С одной стороны, он предоставляет разработчику инструмент, который позволяет описать задачу и существенную часть реализации проекта в терми-

нах, характеризующих предметную область, а не компьютерную модель. По замечанию Кинга, объектно-ориентированный анализ начинается с исследования предметов реального мира, являющихся частью решаемой задачи [King, 1997]. Хотя применение объектно-ориентированной парадигмы само по себе не обеспечивает качества проектирования, оно способствует ясному представлению данных и присущих им свойств и действий (поведения) на уровне программного кода. При этом программные структуры на формальном языке соответствуют ментальной модели описания задачи и способов ее решения на языке, естественном для данной предметной области. Эта особенность отличает объектно-ориентированные языки от языков императивного программирования, нацеленных на моделирование действий, выполняемых вычислительной машиной (рис. 1.2).



Рис. 1.2. Абстрагирование посредством императивных и объектно-ориентированных языков

С другой стороны, объектно-ориентированное проектирование предоставляет разработчикам гибкий мощный универсальный инструмент, не связанный с каким-то определенным классом задач. По замечанию Буча, "объектный подход зарекомендовал себя как унифицирующая идея всей компьютерной науки, применяемая не только в программировании, но также в проектировании интерфейса пользователя, баз данных и даже архитектуры компьютеров"

[Booch, 1994]. При этом использование объектно-ориентированного подхода оправдывает себя не для всех классов задач. Например, для реализации основных вычислительных алгоритмов применение объектно-ориентированного программирования едва ли позволит получить более наглядный и более эффективный код, хотя, разумеется, эти задачи можно использовать в качестве примеров, иллюстрирующих отличия в управлении данными в процедурных и объектно-ориентированных языках [Давыдов, 2001].

Другим классическим примером области, в которой, как раньше считалось, объектная модель вряд ли обеспечивает какие-либо существенные преимущества при проектировании и использовании кода, является синтаксический анализ и трансляция. Однако в ряде случаев объектно-компонентная организация компилятора может быть разумным решением. Здесь можно выделить, по крайней мере, два момента.

Во-первых, разработка и использование элементов транслятора как независимых компонентов (классов) обусловлена развитием интегрированных сред проектирования. Это связано с тем, что реализация различных средств поддержки программирования требует возможности обращения к отдельным компонентам транслятора. Например, подсветка служебных слов и литеральных констант, быстрая навигация по тексту требует вызова лексического анализатора, частичная трансляция фрагментов исходного текста в процессе его набора и информирование программиста о совершаемых ошибках при записи синтаксических конструкций языка требует вызова синтаксического анализатора и т. д. Одной из известных реализаций подобной схемы построения транслятора является так называемая обобщенная инфраструктура компилятора (Common Compiler Infrastructure), проектируемая как часть Microsoft .NET Framework в подразделении Microsoft Research (рис. 1.3). Данная инфраструктура представляет собой библиотеку классов для создания собственных трансляторов языков .NET.

Во-вторых, структурирование компилятора позволяет облегчить его перенос на другую вычислительную архитектуру. Отделяя кодогенерирующие функции от остальной части компилятора, можно существенно упростить создание семейства компиляторов для различных целевых архитектур [Franz, 1998]. Справедливости ради, отметим, что структурирование и компонентная организация компилятора не обязательно предполагают использование объектно-ориентированных технологий.

Во многих случаях наилучшие результаты проектирования обеспечиваются совместным применением множества моделей программирования. Собственно, язык C++ и создавался как язык, поддерживающий множество парадигм проектирования, предоставляя программистам мощный и гибкий механизм для конструирования абстрактных типов, который подходил бы для большинства современных вычислительных машин [Stroustrup, 1998]. C++ под-

держивает много стилей программирования и сочетает эффективность и гибкость языка С для системного программирования с возможностями объектно-ориентированной организации кода и данных. Таким образом, многие специалисты соглашаются с тем, что преимущества объектно-ориентированной технологии можно реализовать быстрее и в более полном объеме, если использовать ее в сочетании с традиционными методами [Лекарев, 1997].

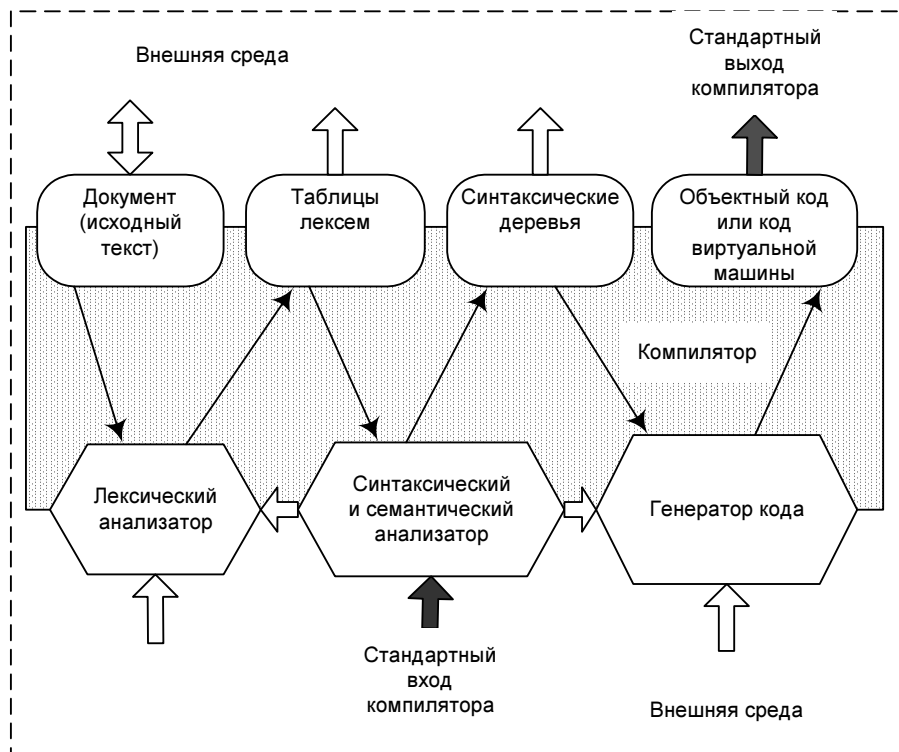


Рис. 1.3. Компонентная организация компилятора

Будучи составной частью множества технологий проектирования, объектно-ориентированная модель программирования поддерживается многими языками (Simula-67, Ada, Smalltalk, Object Pascal, C++, Eiffel, Java, C# и др.). На рис. 1.4 графически представлена генеалогия наиболее распространенных процедурных, объектных и объектно-ориентированных языков. Рисунок концептивно представляет развитие основополагающих концепций программирования в связи с развитием языков программирования.

Подведем итог. *Объектно-ориентированное программирование улучшает проектирование, фокусируясь на данных как более стабильном элементе вычислительной системы. Объектно-ориентированный подход концентрирует*

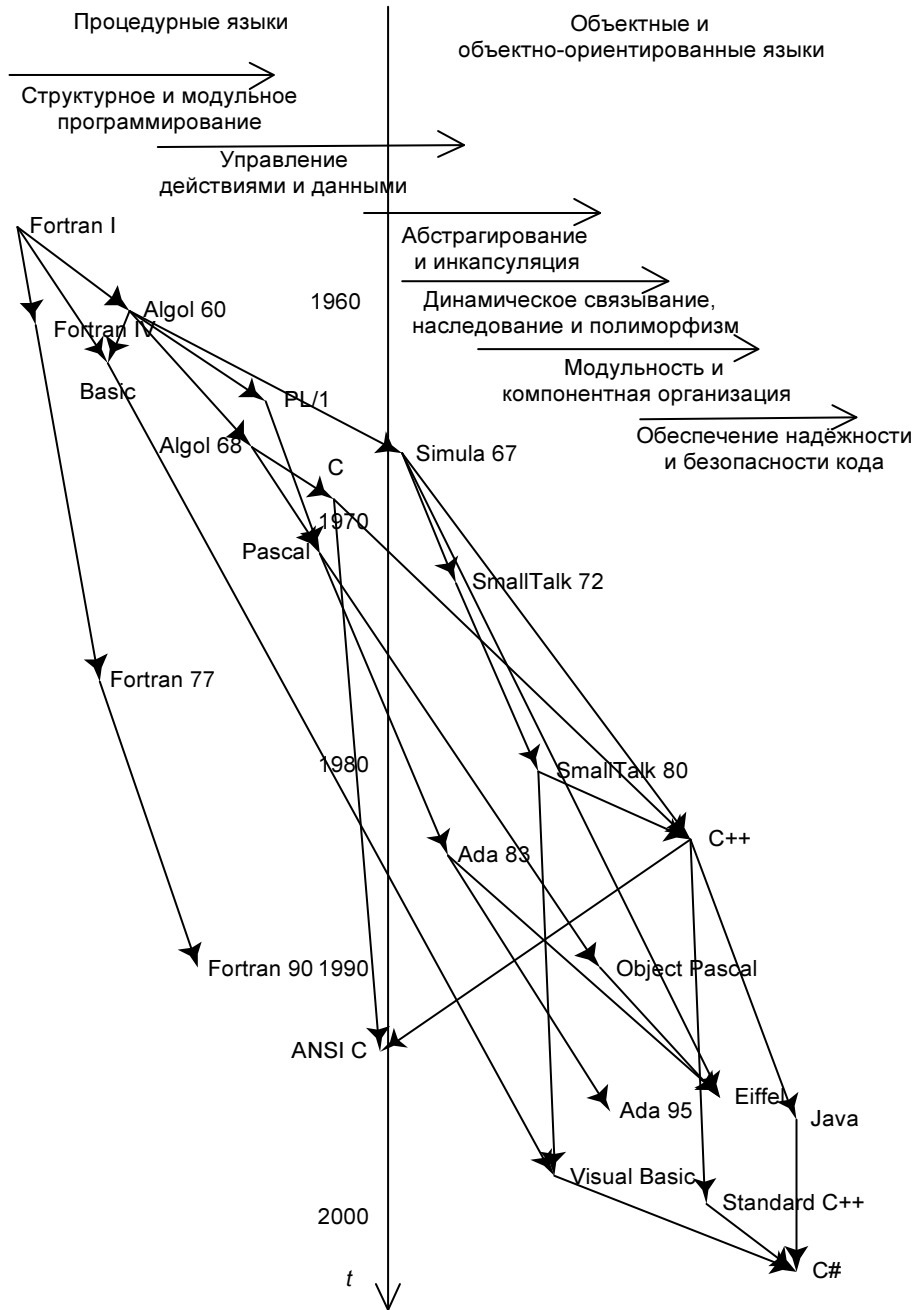


Рис. 1.4. Генеалогия распространенных языков программирования и развитие основных концепций

ется на разработке кода, нацеленного на повторное использование. Объектно-ориентированная модель обеспечивает лучшую масштабируемость проектов. Именно поэтому большинство успешных современных технологий проектирования программных систем предполагает преимущественное или исключительное использование объектно-ориентированной парадигмы.

1.4. Вопросы и упражнения

1. Каким образом в рамках процедурного программирования решается вопрос о разграничении доступа к функциям и данным?
2. Какие средства структурного программирования обеспечивают построение моделей данных, присущих предметной области? Как связываются данные и функции для обработки этих данных?
3. Какие преимущества с точки зрения представления данных обеспечивает объектно-ориентированное программирование?
4. Напишите процедурную программу, обеспечивающую вычисление задаваемых в некотором исходном файле арифметических выражений, оперирующих комплексными данными. Выражения могут содержать операции сложения, вычитания, умножения и деления. Синтаксис представления комплексного числа продумайте самостоятельно.
5. Как Вы понимаете упомянутый в *разд. 1.3* термин "масштабируемость программных систем". Обсудите этот термин с коллегами или преподавателем.
6. Почему обеспечение возможности повторного использования является важной задачей проектирования? В связи с этим, как Вы могли бы пояснить смысл тезиса Мейерса "*Программируйте в будущем времени*" [Meyers, 1996]?

ГЛАВА 2



Элементы объектной модели



Александр Скрябин. "Гирлянды", оп. 73 No. 1

По замечанию Буча, каждый стиль программирования имеет свою концептуальную базу. Для объектно-ориентированного стиля концептуальная база — это *объектная модель*.

Объектную модель составляют четыре главных элемента:

- абстрагирование (abstraction);
- инкапсуляция (encapsulation);
- модульность (modularity);
- иерархия (hierarchy).

Абстрагирование позволяет выделить существенные характеристики некоторого объекта, отличающие его от всех других видов объектов. Абстракция четко определяет концептуальные границы объекта с точки зрения наблюдателя.

Инкапсуляция — это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция также служит для того, чтобы изолировать внешнее поведение объекта от его внутреннего устройства.

Иерархия — это упорядочение абстракций, средство классификации объектов, систематизация связей между объектами.

Модульность — это представление системы в виде совокупности обособленных сегментов, связь между которыми обеспечивается посредством связей между классами, определяемыми в этих сегментах.

Теперь рассмотрим назначение и взаимодействие этих элементов более подробно.

2.1. Смысл абстрагирования как элемента объектной модели

Абстрагирование является одним из основных методов, используемых для решения сложных задач. Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя [Booch, 1994]. В этом определении все элементы важны. Например, любой автомобиль имеет корпус и колеса, причем корпуса автомобилей могут весьма существенно отличаться друг от друга. Тем не менее мы вряд ли спутаем автомобиль с телегой, которая, вообще говоря, тоже имеет корпус и колеса, т. е. абстракции автомобиля и телеги отличаются друг от друга, несмотря на то, что они имеют некоторые общие свойства.

Аналогично понятиям теории абстрактных систем (см. [Mesarovic, Takahara, 1989]), под абстрактными моделями, используемыми в объектно-ориентированном проектировании, понимается не что-то воображаемое или не имеющее отношения к действительности, а модели, характеризующие наиболее общими свойствами, актуальными для практических случаев.

Объектно-ориентированное программирование характеризуется наличием двух основных видов абстракций:

- Тип данных объектной природы (класс) — определяемое программистом расширение исходных типов языка.
- Экземпляр класса (объект) — переменная класса. Объектов в сложной системе обычно гораздо больше, чем классов. Объект обладает состоянием, поведением и идентичностью.

Состояние объекта характеризуется набором его свойств (атрибутов) и текущими значениями каждого из этих свойств (автомобиль заведен, фары включены).

Состояние объекта — результат его поведения, т. е. выполнения в определенной последовательности характерных для него действий (при включении переключателя указателя поворотов лампочка указателя начинает мигать).

Идентичность — это такое свойство (или набор свойств) объекта, которое позволяет отличить его от всех прочих объектов того же типа. Идентичность

не обязательно связана с именованием или состоянием объекта: два совершенно одинаковых автомобиля, стоящих в одном и том же гараже, все же являются разными объектами. Свойством, отвечающим за идентичность, в этом случае, может считаться VIN автомобиля (Vehicle Identification Number — идентификационный номер автотранспортного средства).

В качестве простейшего примера абстракции, интуитивно понятного всем, Эккель в своих книгах (см., например, [Eckel, 2000-1]) приводит электрическую лампочку, для которой определены, как минимум, два присущих ей состояния: включена (светит) и выключена (не светит) (рис. 2.1).

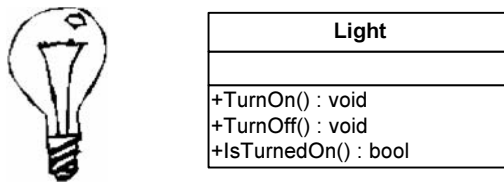


Рис. 2.1. Абстракция электрической лампы

Процесс включения и выключения лампочки описывает поведение объекта. Включению лампы на уровне абстрактной модели соответствует функция `TurnOn()`, выключению — функция `TurnOff()`. Для того чтобы узнать, светится лампа или нет, нам достаточно посмотреть на нее. На уровне абстрактной модели этот процесс представлен специальной функцией `IsTurnedOn()`, позволяющей установить текущее состояние модельного объекта. На диаграмме класса знак "плюс" рядом с именем каждой функции показывает, что эти функции составляют открытый интерфейс класса.

Все лампочки обладают примерно таким поведением, хотя их реализации могут отличаться. Принцип работы вольфрамовых, галогенных, газовых ламп различен, однако любая из них распознается нами как осветительное устройство, которое можно включить и выключить. Какие физические и химические процессы участвуют в ходе перехода лампы из выключенного состояния во включенное, конкретного потребителя может не интересовать. Ему может быть достаточно представления о том, где и как правильно использовать это устройство. Конструируя абстрактную модель, программист в первую очередь пытается выразить наиболее общие свойства моделируемых объектов.

Объявление на языке C++, которое соответствует данной модели, может выглядеть следующим образом:

```
class Light { // Класс Light
public:
    void TurnOn();
```