МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Чувашский государственный университет имени И.Н. Ульянова»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания к курсовой работе

Чебоксары
2015

УДК 3793.2я73
ББК 004.434(076.5)

Составитель И.А. Обломов

Объектно-ориентированное программирование: метод. указания к курсовой работе. Чебоксары: Изд-во Чуваш. ун-та, 2015. – 16 с.

Приведены рекомендации по выполнению курсовой работы, требования к содержанию отчета и основные правила по оформлению.

Для студентов III курса направления «Информатика и вычислительная техника».

Работа выполнена при финансовой поддержке программы «Кадры для регионов».

Утверждено Учебно-методическим советом университета

Ответственный редактор: канд. техн. наук, доцент А.Л. Симаков

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания к курсовой работе

Редактор *О.А. Хлебкова*
Компьютерная верстка и правка *Т.В. Калишовой*

Согласно закону №436-ФЗ от 29 декабря 2010 г.
данная продукция не подлежит маркировке

Подписано в печать 28.12.2015. Формат 60×84/16.
Бумага газетная. Печать офсетная. Гарнитура Time New Roman.
Усл. печ. л. 1.62. Уч-изд. л 1,14.Тираж 200 экз. Заказ № 1435.

Издательство Чувашского университета
Типография университета
428015 Чебоксары, Московский просп. 15

Общие указания

Курс «Объективно-ориентированное программирование» входит в ряд дисциплин, связанных с изучением специальных разделов программирования для ЭВМ, должен обеспечить формирование навыков постановки и решения задач в среде объективно-ориентированных программных систем, таких как Smalltalk [5], C++ [1], Visual C [4,9].

Целью курсовой работы является углубление и закрепление у студентов теоретических знаний, навыков практического применения основных принципов объективно-ориентированных систем: наследования, инкапсуляции, полиморфизма, а также формирование новых взглядов на процессы программирования с учетом абстракции данных.

В методических указаниях приведены правила оформления курсовых работ, которыми студенты должны руководствоваться в течение всего периода обучения в вузе, а также в дальнейшей практической деятельности.

В рамках указаний не рассмотрены вопросы, касающиеся технологии и отладки программ, поскольку они различны для каждой конкретной программной оболочки, работающей под управлением той или иной операционной системы.

Методические указания являются продолжением работы [7].

Основные определения

Объектно-ориентированное программирование (в дальнейшем сокращено до ООП) возникло относительно недавно, но, не смотря на малый срок, завоевало ведущие позиции в области программирования. Примером тому могут служить бурно развивающиеся ООП системы: Smalltalk, C++, Visual C, Delphi и др. [6,8]. Объектно-ориентированная парадигма предлагает новый подход к разработке программного обеспечения, предназначенного для решения большого класса задач. Под парадигмой следует понимать модели разработки и реализации программ. Различные модели приводят к различным приёмам программирования.

Фундаментальная концепция ООП состоит в *передаче сообщения объектам.* Для этого необходимо, чтобы объекты определялись вместе с сообщениями, на которые они будут реагировать. Это и есть главное отличие ООП от императивного (процедурного) программирования, в котором сначала опреде-

Если смысл задачи стал понятен, нужно попробовать её обобщить. Самый лучший способ для этого – сформулировать задачу как математическую, пользуясь такими понятиями, как векторы, матрицы, автоматы и т.д.

Очень полезно определить класс задачи: является ли она чисто вычислительной, поисковой, логической. Это поможет в дальнейшем выбрать адекватный метод или способ решения задачи.

В процессе анализа задачи должен быть выработан подход к решению задачи.

Спецификация на разрабатываемую программу

Спецификация формулирует в наиболее общем виде, что должна делать программа для переработки входных данных в выходные. Можно сказать, что спецификация описывает программу как «чёрный ящик», указывая, «что» должна делать программа, оставляя в стороне вопрос о том, «как» она это делает. Спецификация должна обладать свойствами полноты, точности и понятности. Для этого, формулируя задачу на естественном языке, нужно постараться и добиться, чтобы спецификация легко читалась.

Для того чтобы спецификация была полной, в неё следует включать описание всех возможных входных и выходных данных, принятых ограничений и особых ситуаций.

Проектирование программы

Анализ условий задачи, позволяющий сформулировать спецификацию программы, одновременно приводить к возможности представления достаточно сложной задачи в виде подзадач. Средства ООП предлагают для этого богатый набор возможностей, подзадачи могу представлять собой некую иерархическую систему. В этом плане нет необходимости определять часто выполняемые фрагменты программы как самостоятельные программные единицы. Имеет смысл создать метод (функцию), обладающий наиболее общими свойствами, и определить его как метод суперкласса. Если возникнет необходимость в выполнении подобного метода объектами подклассов, необходимо осуществить ссылку к суперклассу, точнее сказать, к его свойствам, и наполнить его свойствами, присущими объектам конкретного класса. Например, можно сказать, что все объекты живой природы могут перемещаться. В то же время, объекты класса «Птицы» летают, объекты класса «Млекопи-

тающие» ходят, а объекты класса «Рыбы» плавают. Уместно будет отметить вопрос о «габаритах» метода. Если его размер не вписывается в один рукописный лист (занимает более 50 строк), то имеет смысл разить его на некоторое множество взаимосвязанных методов. Читаемость программы, а следовательно и понятность, от этого только выиграют.

Что касается данных, с которыми работает программа, то в ООП этот вопрос приводит к понятию абстракции данных. Это следует понимать так: любой объект содержит в себе как данные, так и операции для переработки этих данных. Весь процесс завершается, когда методы (функции) будут реализованы средствами конкретного языка ООП.

Проверка правильности решения задачи и планирование отладки

Современные методы проектирования программ предполагают не только уменьшение затрат времени, но и то, что программный продукт будет правильным и выдаст корректные результаты. Если есть сомнения относительно результатов, имеет смысл повторить весь процесс проектирования с точки зрения возможных упущений в логике программы и нарушений связей между модулями. Нахождение лексических и синтаксических ошибок в программе – задачи компилятора. Большую опасность представляют ошибки логического типа. Радикальным способом проверки правильности программы является тестирование. Рекомендуется следующая система тестов:

1. Типовые наборы исходных данных и известные для них правильные результаты.

2. Тесты для проверки различных путей в программе, т.е. управляющих структур, таких как операторы цикла, условные операторы. С помощью определённых наборов данных можно преднамеренно входить в цикл, выходить из него, проверять выполнение условных операторов и операторов варианта.

3. Тесты для проверки границ областей изменения данных.

4. Тесты для проверки реакции на недопустимые входные данные.

Хорошей поддержкой тестирования программы могут послужить отладочные средства, входящие в состав интегрированных систем программирования.

Отдельные программы могут содержать в себе графическую поддержку. Графический интерфейс должен быть простым, понятным, удобным для пользования. Нежелательна избыточность, которая, как правило, делает «картинку» излишне загруженной. В случае воспроизведения движения объекта (мультипликации) по экрану терминала следует стремиться создать впечатление движения в реальном времени. При моделировании игровых программ (шахматы, шашки, различные карточные игры и т.д.) попытаться максимально визуализировать игровую ситуацию.

В пояснительной записке к курсовой работе должны присутствовать результаты тестирования программы в целом.

Оформление курсовой работы

Правила оформления текста, рисунков и таблиц

Пояснительная записка к курсовой работе оформляется на листах формата А4 (210×297). Каждый лист должен иметь поля: левое -- 30 мм, правое -- не менее 10 мм, верхнее и нижнее – по 25 мм. Допускается представлять записку в машинописном виде.

Структурными элементами текста являются разделы, подразделы, пункты, подпункты и перечисления. Номера структурных элементов образуются по десятичному принципу: разделы нумеруются: 1,2,3 и т.д., номера подразделов составляются из двух чисел: 3.1, 3.2 и т.д. Разделы и подразделы должны иметь заголовок. Заголовки разделов записывают прописными буквами, а заголовки подразделов – строчными, кроме первого символа. Точка в конце названия не ставится. Каждый раздел рекомендуется начинать с нового листа.

Сокращать слова в тексте нельзя, кроме общепринятых сокращений и случая, когда составляется перечень принятых сокращений на отдельной странице перед списком литературы.

Рисунки нумеруются либо в пределах всей работы, либо в пределах раздела. В последнем случае номером рисунка может быть, например, 3.1, что означает первый из рисунков в третьем разделе. То же относится к таблицам.

Рисунки и таблицы должны иметь тематический заголовок, располагаемый сверху. Рисунки могут иметь подрисуночный текст. Номер рисунка располагается ниже подрисуночного текста.

Номера страниц проставляются в правом верхнем углу страницы. Первой страницей отчёта является титульный лист, номер на котором не ставится. Лист должен иметь заголовок «ОГЛАВЛЕНИЕ», он не нумеруется как раздел.

Оформление текста программы

Если для вычислительной машины оформление текста программы не имеет никакого значения, то для программиста этот вопрос очень важен: если текст оформлен как следует, затраты труда на отладку программы и сопровождение могут быть существенно снижены.

Требования по оформлению текста программы включают в себя:

- правила размещения текста, т.е. разбивки программы на строки и относительного расположения содержимого строк;
- использование мнемонических имен для программных объектов и объединение всех объектов в некоторую систему;
- правила по составлению и размещению комментариев.

Следует помнить, что любая программа является коммерческим продуктом, поэтому правила оформления текста программы являются обязательными.

Размещение операторов относительно друг друга – дело вкуса программиста, единственное требование – удобство чтения программы.

Мнемонические имена объектов программы должны иметь определённый смысл, как можно точнее отражать внутреннее содержание объекта. Например, идентификатор String в достаточной степени отражает смысл объекта строка, а идентификатор Integer – объект класса целых чисел.

Любая сложная программа должна сопровождаться комментариями, основная цель которых – сделать программу « самодокументированной», т.е. обеспечить возможность работы с её текстом без привлечения каких-либо дополнительных сопровождающих документов.

Структура курсовой работы

В ходе выполнения курсовой работы следует придерживаться следующей структуры:

1. Введение. В данном разделе, помимо основных сведений, должна быть точно определена цель курсовой работы.

2. Постановка задачи. Исходя из полученного задания на курсовой проект, здесь необходимо выполнить его конкретизацию и выбор предметной области.

3. Выбор метода, способа решения поставленной задачи и обоснование выбора. В этом разделе необходимо определить возможные способы, описать их положительные и отрицательные стороны.

4. Теоретический раздел. В данном разделе должно присутствовать подробное описание выбранного метода.

5. Выводы по работе.

6. Список использованной литературы. В разделе использованной литературы должны присутствовать только официальные издания, можно делать ссылки на сайты Интернета.

7. Оглавление.

8. Приложение. В качестве приложения приводятся тексты программ на исходном языке.

Пример объявления иерархии классов

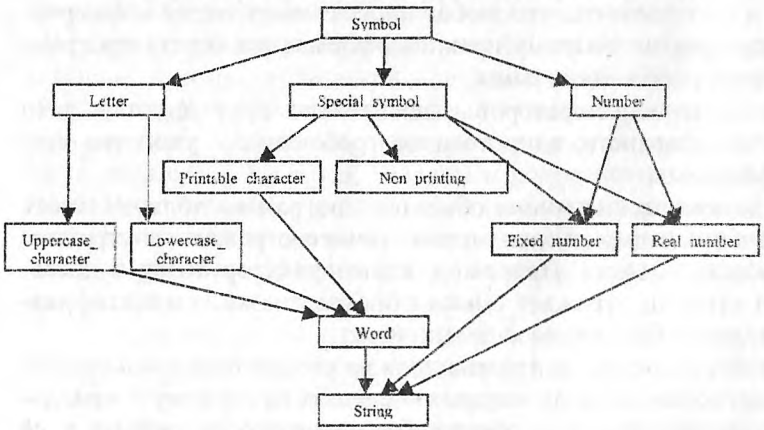

Рис. 1. Пример иерархии классов для обработки символьных строк

Фрагмент объявления классов для данной иерархии приведен ниже.

```
// class Symbol
class Symbol
{
```

```
protected:
    wchar_t symbol;
public:
    Symbol(){};
    Symbol(wchar_t);
    virtual void Out();
    friend ostream &operator <<(ostream &, const Symbol &);
};

// class Letter
class Letter :public Symbol
{
    wchar_t letter;          // symbols a, b, c, ... A, B, C, etc
public:
    Letter(){};
    Letter(wchar_t ch);
    void Out();
    friend ostream &operator <<(ostream &, const Letter &);
};

// class Special_Symbol
class Special_Symbol :public Symbol
{
protected:
    wchar_t sp_symbol;
public:
    Special_Symbol(){};
    Special_Symbol(wchar_t);
    void Out();
    friend ostream &operator <<(ostream &, const Special_Symbol
&);
};

// class Printable_character
class Printable_character :virtual public Special_Symbol
{
    wchar_t print_character;
public:
    Printable_character(){};
```

```
    Printable_character(wchar_t);
    void Out();
    friend ostream &operator <<(ostream &, const Printa-
ble_character &);
};

// class Non_printing _character
class Non_printing_character :virtual public Special_Symbol
{
    wchar_t non_print_character;
public:  Non_printing_character(){};
    Non_printing_character(wchar_t);
    friend ostream &operator <<(ostream &, const

// class Uppercase_character
class Uppercase_character :virtual public Letter
{
    wchar_t uppercase;
public:
    Uppercase_character(){};
    Uppercase_character(wchar_t);
    void Out();
    friend ostream &operator <<(ostream &, const Upper-
case_character &);
};
void Uppercase_character::Out()
{
    cout << " Прописной символ: " << uppercase << endl;
}

// Lowercase_character
class Lowercase_character :virtual public Letter
{
    wchar_t lowercase;
public:
    Lowercase_character(){};
    Lowercase_character(wchar_t);
    void Out();
```

```
    friend ostream &operator <<(ostream &, const Lower-
case_character &);
};

// class Word
class Word :public     Uppercase_character, public Lower-
case_character, public Printable_character,
{
protected:
    wchar_t *word;
public:
    Word(){};
    Word(wchar_t *);
    void Out();
friend ostream &operator <<(ostream &, const Word &);
};
```

Приведенная иерархия взаимосвязанных классов, предназначенная для хранения одиночных символов и отдельных слов, носит описательный характер. Реализация методов класса зависит от конкретной цели. Несмотря на определенную условность, эта иерархия позволяет применить и продемонстрировать основные принципы объектно-ориентированного программирования, в частности, инкапсуляцию, наследование, в том числе и множественное, полиморфизм.

Класс String наследует все свойства базовых классов, что позволяет хранить в строке произвольные символы. Кроме того, желательно предусмотреть методы для работы с потоками, позволяющими считывать одиночные символы из потоков и записывать символы в потоки. Таким образом, можно осуществить доступ к текстовым файлам через потоки.

Для хранения числовых символов предусмотрено три класса: Number, Fixed_number и Real_number. Класс Number представляет числовой символ в общем случае, класс Fixed_number обеспечивает хранение и методы обработки целых чисел, а класс Real_number – вещественных.

Их приблизительное описание приведено ниже.

```
// class Number
```

```
class Number :public Symbol
{
protected:
    wchar_t number;
public:
    Number(){};
    Number(wchar_t);
    virtual Number operator +(const Number &);
    virtual bool operator <(const Number &);
    friend ostream &operator <<(ostream &, const Number &);
};
// class Fixed_number
class Fixed_number :virtual public Number
{
protected:
    int fixed;
public:
    Fixed_number():Number('0'),fixed(0){};
    Fixed_number(int);
    Fixed_number operator +(const Fixed_number &);
    bool operator <(const Fixed_number &);
    friend ostream &operator <<(ostream &, const Fixed_number
&);
};
// class Real_number
class Real_number :virtual public Number
{
protected:
    float real;
public:
    Real_number():Number('0'),real(0){};
    Real_number(float);
    Real_number operator +(const Real_number &);
    bool operator <(const Real_number &);
    friend ostream &operator <<(ostream &, const Real_number &);
};
```

В классе Real_number в конструкторе Real_number::Real_number(float re):Number(re), real(number){}; содержится некорректное преобразование типа float в wchar_t, что

может привести к потере точности вычислений. Числовые классы содержат примеры перегруженных операторов арифметических действий и сравнения. Это позволит выполнять обозначенные действия над символами, представляющие числовые величины. Дальнейшее расширение реализации этих классов может содержать в себе диапазоны изменения величин конкретного типа. Например, объекты типа Letter могут принимать значения в диапазоне ‘a’, ‘b’,’c’, …’z’ и ‘A’, ‘B’, ‘C’, etc, а класса Number могут изменяться в диапазоне от 0 до 9. Более сложные комбинации числовых величин могут содержать специальные символы (например, унарный минус ‘-‘, символ точки ‘.’, символ e для экспоненциальной формы записи вещественных чисел), принадлежащие классу Special_symbol.

Варианты индивидуальных заданий

1. Модель электронной картотеки.
2. Электронный калькулятор.
3. Электронные часы с семисегментным индикатором.
4. Стрелочные часы.
5. Игровая программа «Ханойская башня».
6. Игровая программа «Пятнадцать».
7. Модель справочной службы «Аэрофлота».
8. Лексический анализатор языка Си.
9. Модель автомата с магазинной памятью (разбор снизу вверх).
10. Модель автомата с магазинной памятью (разбор сверху вниз).
11. Лексический анализатор языка Пролог.
12. Лексический анализатор языка Лисп.
13. Модель работы коммивояжера.
14. Модель электронного справочника.
15. Модель склада продовольственных товаров.
16. Модель детерминированного конечного автомата.
17. Модель функционирования ЭВМ.
18. Модель работы недетерминированного конечного автомата.
19. Модель работы диспетчера производства.
20. Телефонный справочник.
21. Модель «записная книжка».
22. Эмулятор ассемблера машины PDP 11.
23. Транслятор машинных команд IDM PC в команды машины PDP 11.
24. Модель отладчика языка Паскаль.
25. Модель отладчика языка Си.
26. Модель редакционно-издательской системы: верстка журналов, книг.
27. Планирование расписаний учебных занятий.
28. Модель экспертной системы.
29. Модель банкомата.
30. Программа «Переводчик».

31. Модель работы многопроцессорной ЭВМ.
32. Модель работы ЭВМ конвейерного типа.
33. Модели логических устройств.
34. Модели работы регулировщика.
35. Схема работы метрополитена.
36. Модель работы городской справочной службы 09.
37. Схема работы городской транспортной сети.
38. Модель обучающей системы.
39. Игровая программа «Поиск пути в лабиринте».
40. Семестровый табель успеваемости студентов группы.
41. Моделирование электронных схем.
42. Игровая программа «Шашки».
43. Модель работы раскройщика материала.
44. Модель «Фоторобот».
45. Игровая программа «Карточная игра».

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Белецкий Я. Турбо С++: Новая разработка / пер. с польск. – М.: Машиностроение, 1994.
2. Дьюхарст С., Старк К. Программирование на С++ / пер. с англ. – Киев: ДиаСофт, 1993.
3. Строуструп Б. Язык программирования С++ / пер. с англ. – М.: Радио и связь, 1991.
4. Франка П. С++: учеб. курс.– СПб.: Питер, 2001.
5. Smalltalk/V DOS. Object-Oriented Programming System. Digitalk inc.5 Hutton Center Drive, Santa Ana, California 92707 USA.
6. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. – СПб.: Питер, 2002. – 464 с.
7. Объектно-ориентированное программирование: метод. указания к курсовой работе / сост. И.А. Обломов. – Чебоксары: Изд-во Чуваш. ун-та, 2002. – 12 с.
8. Щупак Ю.А. Win32 API. Эффективная разработка приложений. – СПб.: Питер, 2007. – 572 с.
9. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. – СПб.: Питер, 2008. – 928 с.

Оглавление