

## Оглавление

Введение .....	2
Цель и задачи практикума, требования к результатам его выполнения .....	3
1 Структурная (процедурная) декомпозиция программы .....	4
1.1 Понятие процедурной и структурной декомпозиции .....	4
1.2 Правила изображения схем алгоритма и записи псевдокодов .....	6
1.3 Метод пошаговой детализации .....	10
1.4 Использование метода пошаговой детализации при структурной декомпозиции ...	12
Контрольные вопросы .....	18
2 Объектная декомпозиция .....	19
2.1 Понятие объектной декомпозиции .....	19
2.2 Построение диаграмм классов .....	21
2.3 Язык описания объектных разработок .....	25
2.4 Документирование объектных разработок в среде Delphi .....	29
2.5 Документирование объектных разработок на языке C++ .....	34
Контрольные вопросы .....	38
3 Порядок выполнения заданий практикума и требования к отчетам по каждому заданию .....	39
3.1 Задание 1. Turbo Delphi. Создание программной системы .....	39
3.2 Задание 2. C++. Создание программной системы с элементарным интерфейсом консольного режима .....	40
3.3 Задание 3. C++ Создание программной системы с Qt интерфейсом .....	41
Литература .....	42

## **ВВЕДЕНИЕ**

Умение разрабатывать программы и знание особенностей программирования на 2-3 языках программирования – обязательное требование к бакалаврам, обучающимся по направлению Информатика и вычислительная техника. Для получения навыков разработки программ различного назначения создания небольших программ, демонстрирующих знание различных механизмов и приемов, которые входят в лабораторный практикум по дисциплинам Основы программирования и Объектно-ориентированное программирование, недостаточно. Поэтому учебный план включает специальную практику – Практикум по программированию.

Во время выполнения практикума студенты должны научиться создавать - проектировать, отлаживать и тестировать – небольшие программные системы. Особое внимание при этом обращается на качество программных интерфейсов с пользователем.

Практикум проводится на втором семестре обучения бакалавров под контролем преподавателей.

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

## ЦЕЛЬ И ЗАДАЧИ ПРАКТИКУМА, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕГО ВЫПОЛНЕНИЯ

**Целью** практикума является получение навыков создания небольших программных систем с оконными и консольными интерфейсами.

**Задачами** практикума являются:

- более глубокое изучение средств реализации проектов программ на одном из изучаемых универсальных языках программирования высокого уровня;
- овладение методикой и получение практических навыков проектирования небольших программных систем при структурном и объектном подходах;
- воспитание внимания, аккуратности, систематичности, а также формирование интереса к изучаемой профессиональной деятельности.

Выполнение практикума должно способствовать формированию и развитию навыков и умений, обеспечивающих следующие компетенции:

- выделение объектов предметной области, обобщение их в классы, определение связей между классами (ПК-3);
- проектирование эргономичного обеспечения информационных систем (ПК-5);
- разработка и отладка компонентов аппаратно-программных комплексов с помощью современных автоматизированных средств проектирования (ПК-7);
- разработка проектной и эксплуатационной документации на программную и техническую продукцию (ПК-8);
- выполнение контроля разрабатываемых проектов и технической документации на соответствие стандартам и техническим требованиям (ПК-10);
- разработка интерфейсов «человек - ЭВМ» (ПК-12).

Практикум включает три задания, одно – на структурную декомпозицию и два – на объектную декомпозицию. По завершению каждого из них студент представляет текущий отчет и получает баллы за работу и защиту.

Итоговая оценка по практикуму проставляется по сумме полученных баллов.

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

# 1 СТРУКТУРНАЯ (ПРОЦЕДУРНАЯ) ДЕКОМПОЗИЦИЯ ПРОГРАММЫ

## 1.1 Понятие процедурной и структурной декомпозиции

При разработке сравнительно несложного программного обеспечения обычно используют процедурную декомпозицию.

*Процедурной декомпозицией* называют представление разрабатываемой программы в виде совокупности вызывающих друг друга подпрограмм. Каждая подпрограмма в этом случае выполняет некоторую операцию, а вся совокупность подпрограмм решает поставленную задачу.

Документирование программ, включающих подпрограммы, предполагает разработку *схем алгоритмов* подпрограмм и схемы структурной.

*Структурной* называют схему, отражающую *состав программного обеспечения и взаимодействие по управлению* его частей. При процедурной декомпозиции на структурной схеме в виде прямоугольников представляют основную программу и подпрограммы и в виде линий показывают передачи управления между ними в процессе выполнения программы.

*Структурная декомпозиция* – частный случай процедурной, при структурной декомпозиции запрещены «циклические» или обратные вызовы подпрограмм. Схема структурной декомпозиции иерархическая и напоминает дерево, перевернутое корнем вверх.

На верхнем уровне, в корне дерева находится основная программа, которая вызывает подпрограммы следующего уровня, а эти подпрограммы, в свою очередь, вызывают только подпрограммы более низких уровней. Вызов подпрограмм, расположенных в иерархии вызовов выше или на том же уровне, что и вызывающая подпрограмма, является циклическим и, следовательно, запрещен.

Структурная декомпозиция рекомендуется технологией структурного программирования – технологией разработки программного обеспечения, которая представляет собой набор рекомендаций по уменьшению количества ошибок в программах.

Рассмотрим пример структурной декомпозиции программы.

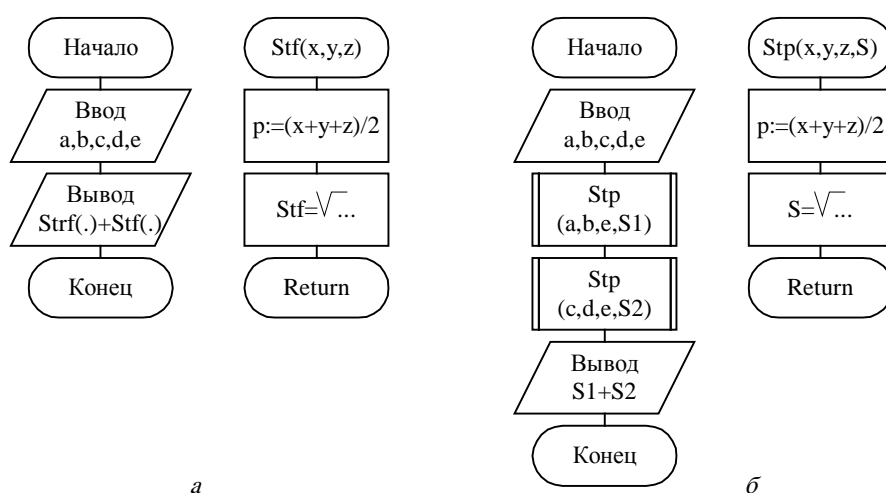
### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

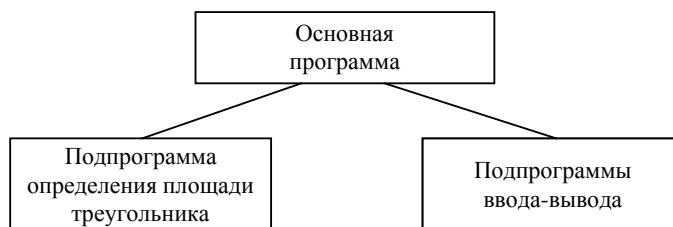
**Пример 1.** Разработать программу, которая определяет площадь четырехугольника по заданным длинам сторон и диагонали.

Будем считать площадь четырехугольника, как сумму площадей двух треугольников по формуле Герона. Вычисление площади треугольника оформим как подпрограмму. Исходные данные такой подпрограммы – длины сторон треугольника. Подпрограмма не должна менять их значения, поэтому длины можно передать как параметры-значения или параметры-константы. Результат работы этой подпрограммы – скалярное значение, значит, она может быть реализована как функция. Однако она может быть реализована и как процедура, которая возвращает результат через параметр-переменную. Схемы алгоритма проектируемой программы с использованием подпрограмм обоих типов приведены на рисунке 1.



**Рисунок 1** – Схемы алгоритмов программы определения площади четырехугольника с использованием функции (а) и процедуры (б)

Структурная схема программы определения площади четырехугольника приведена на рисунке 2.



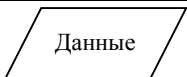


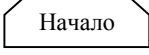






**Рисунок 2** - Схема структурная программы определения площади четырехугольника

## 1.2 Правила изображения схем алгоритма и записи псевдокодов

Изображение *схем алгоритмов* осуществляют согласно ГОСТ 19.701–90, в котором каждой группе действий ставится в соответствие блок особой формы. Некоторые, часто используемые обозначения приведены в таблице 1.

Таблица 1

Название блока	Обозначение	Назначение блока
1. Терминатор		Начало, завершение программы или подпрограммы
2. Процесс		Обработка данных (вычисления, пересылки и т.п.)
3. Данные		Операции ввода-вывода
4. Решение		Ветвления, выбор, итерационные и поисковые циклы
5. Подготовка		Счетные циклы
6. Граница цикла	 	Любые циклы
7. Предопределенный процесс		Вызов процедур
8. Соединитель		Маркировка разрывов линий
9. Комментарий		Пояснения к операциям

При разработке схемы алгоритма каждое действие обозначают соответствующим блоком, показывая их последовательность линиями, идущими слева направо и сверху вниз. Для удобства чтения схемы желательно, чтобы *линия входила в блок сверху, а выходила снизу*. Если направление линии отлично от стандартного, то в конце линии ставится стрелка, в противном случае стрелку можно не ставить.

Если схема алгоритма не умещается на листе, то используют специальные соединители. При переходе на другой лист или получении управления с другого листа в комментарии указывают номер листа, например, «с листа 3», «на лист 1».

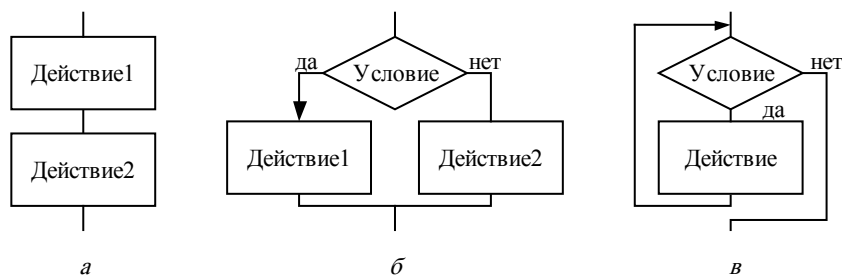
Доказано, что для записи любого, сколь угодно сложного алгоритма достаточно трех базовых управляющих структур:

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

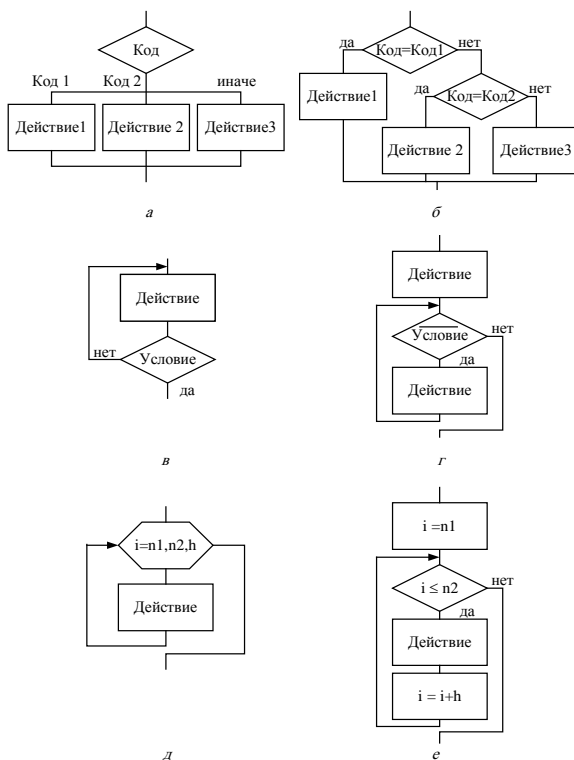
- *следование* – обозначает последовательное выполнение действий (см. рисунок 3, а);
- *ветвление* – соответствует выбору одного из двух вариантов действий (см. рисунок 3, б);
- *цикл-пока* – определяет повторение действий, пока не будет нарушено некоторое условие, выполнение которого проверяется в начале цикла (см. рисунок 3, в).



**Рисунок 3** – Базовые алгоритмические структуры: следование (а), ветвление (б) и цикл-пока (в)

Помимо базовых структур используют еще три дополнительные структуры, производные от базовых:

- *выбор* – обозначающий выбор одного варианта из нескольких в зависимости от значения некоторой величины (см. рисунок 4, а);



**Рисунок 4** - Дополнительные структуры и их реализация через базовые структуры: выбор (а-б), цикл-до (в-г) и цикл с заданным числом повторений (д-е)

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

- *цикл-до* – обозначающий повторение некоторых действий до выполнения заданного условия, проверка которого осуществляется после выполнения действий в цикле (см. рисунок 4, в);

- *цикл с заданным числом повторений (счетный цикл)* – обозначающий повторение некоторых действий указанное количество раз (см. рисунок 4, д).

На рисунке 4 (б, г и е) показано, как каждая из дополнительных структур может быть реализована через базовые структуры.

В том случае, если в схеме алгоритма отсутствуют другие варианты передачи управления, алгоритм называют *структурным*, подчеркивая, что он построен с учетом рекомендаций структурного программирования.

Помимо детальных схем алгоритма допустимы обобщенные схемы, на которых группы действий изображают в виде подпрограмм.

*Псевдокод* базируется на тех же основных структурах, что и структурные схемы алгоритма. Описать на псевдокоде неструктурный алгоритм нельзя.

Для каждой структуры используют свою форму описания. В литературе были предложены несколько вариантов псевдокодов. Один из них приведен в таблице 2.

Таблица 2

Структура	Псевдокод	Структура	Псевдокод
1. Следование	<действие 1> <действие 2>	4. Выбор	<b>Выбор</b> <код> <код1>: <действие 1> <код2>: <действие 2> ... <b>Все-выбор</b>
2. Ветвление	<b>Если</b> <условие> <b>то</b> <действие 1> <b>иначе</b> <действие 2> <b>Все-если</b>	5. Цикл с заданным количеством повторений	<b>Для</b> <индекс> = <n>,<k>,<h> <действие> <b>Все-цикл</b>
3. Цикл-пока	<b>Цикл-пока</b> <условие> <действие> <b>Все-цикл</b>	6. Цикл-до	<b>Выполнять</b> <действие> <b>До</b> <условие>

**Пример 2.** Разработать алгоритм определения наибольшего общего делителя двух натуральных чисел.

Существует несколько способов определения наибольшего общего делителя двух натуральных чисел. Самым простым из них является так называемый «алгоритм Евклида». Суть этого метода заключается в последовательной замене большего из чисел на разность большего и меньшего. Вычисления заканчиваются, когда числа становятся равны.



Например:

a)	<b>A</b>	<b>B</b>	б)	<b>A</b>	<b>B</b>
	225	125		13	4
	225-125=100	125		13-4=9	4
	100	125-100=25		9-4=5	4
	100-25=75	25		5-4=1	4
	75-25=50	25		1	4-1=3
	50-25=25	= 25		1	3-1=2
				1	2-1=1

Программа должна начинаться с ввода чисел. Заметим, что любой ввод данных пользователем должен сопровождаться запросом на ввод, чтобы пользователь знал, чего от него ждет программа после запуска. На схеме алгоритма и при записи псевдокодов этот запрос обычно не указывают.

В основе алгоритма лежит циклический процесс, количество повторений которого заранее неизвестно (итерационный). Условие выхода из цикла – получение одинаковых чисел. Поскольку нельзя исключить, что пользователь введет равные числа, проверку будем осуществлять на входе в цикл, т. е. имеет смысл использовать цикл-пока. Если числа не равны, то при каждом проходе цикла большее из чисел должно заменяться разностью большего и меньшего. Для реализации этой замены потребуется описать оба варианта, т.е. использовать ветвление с проверкой, какое из чисел больше. После выхода из цикла числа равны, поэтому можно выводить пользователю любое из двух полученных чисел. На рисунке 5 показана схема алгоритма, а ниже приведено его описание на псевдокоде.

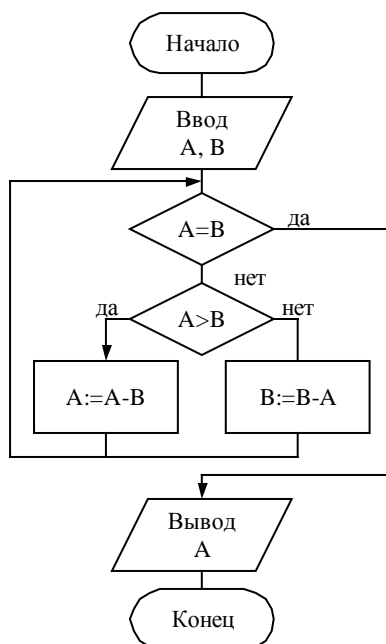


Рисунок 5 - Схема алгоритма Евклида

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

**Алгоритм Евклида:**

Ввести A,B

**цикл-пока** A≠B    **если** A>B        **то** A:=A-B    **иначе** B:=B-A    **все-если****все-цикл**

Вывести A

**Конец алгоритма**

Алгоритмы более сложных программ разрабатывают, используя специальный метод.

### 1.3 Метод пошаговой детализации

С использованием *метода пошаговой детализации* разработку алгоритмов выполняют поэтапно. На первом этапе описывают решение поставленной задачи «по крупному», выделяя подзадачи, которые необходимо решить. На следующем – аналогично описывают решение подзадач, формулируя уже подзадачи следующего уровня. Процесс продолжают, пока не доходят до подзадач, алгоритмы решения которых очевидны. При этом, описывая решение каждой задачи, желательно использовать не более одной-двух конструкций, таких как цикл или ветвление, чтобы четче представить себе структуру программы.

**Пример 3.** Разработать программу, которая с заданной точностью  $\varepsilon$  находит значение аргумента  $x$  по заданному значению функции  $y$  при известном значении  $n$ .

$$y = \frac{(x+1)^n + 1}{x}, \text{ где } n > 1, x > 0.$$

При  $n > 1$  данная функция является монотонно возрастающей. Следовательно, для нахождения значения  $x$  можно применить метод половинного деления. Суть метода заключается в следующем. Вначале определяют отрезок  $[x_1, x_2]$ , такой что  $f(x_1) \leq y \leq f(x_2)$ . Затем делят его пополам  $x_t = (x_1 + x_2)/2$  и определяют, в какой половине отрезка находится  $x$ , для чего сравнивают  $f(x_t)$  и  $y$ . Полученный отрезок опять делят пополам и так до тех пор, пока разность  $x_1$  и  $x_2$  не станет меньше заданного значения  $\varepsilon$ .

Для разработки алгоритма используем метод пошаговой детализации.

*Шаг 1.* Определяем общую структуру программы:

**Программа:**Ввести  $y, n, \varepsilon$ .

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Определить  $x$ .

Вывести  $x$ ,  $y$ .

**Конец.**

*Шаг 2.* Детализируем операцию определения  $x$ :

**Определить  $x$ :**

Определить  $x_1$ , такое что  $f(x_1) \leq y$ .

Определить  $x_2$ , такое что  $f(x_2) \geq y$ .

Определить  $x$  на интервале  $[x_1, x_2]$ .

**Все.**

*Шаг 3.* Детализируем операцию определения  $x_1$ . Значение  $x_1$  должно быть подобрано так, чтобы выполнялось условие  $f(x_1) \leq y$ . Известно, что  $x > 0$ , следовательно, можно взять некоторое значение  $x$ , например,  $x_1 = 1$ , и последовательно уменьшая его, например, в два раза определить значение  $x_1$ , удовлетворяющее данному условию:

**Определить  $x_1$ :**

$x_1 := 1$

**цикл-пока**  $f(x_1) > y$

$x_1 := x_1 / 2$

**все-цикл**

**Все.**

*Шаг 4.* Детализируем операцию определения  $x_2$ . Значение  $x_2$  находим аналогично  $x_1$ , но исходное значение будем увеличивать в два раза.

**Определить  $x_2$ :**

$x_2 := 1$

**цикл-пока**  $f(x_2) < y$

$x_2 := x_2 * 2$

**все-цикл**

**Все.**

*Шаг 5.* Детализируем операцию определения  $x$ . Определение  $x$  выполняется последовательным сокращением отрезка  $[x_1, x_2]$ .

**Определить  $x$ :**

**цикл-пока**  $x_2 - x_1 > \epsilon$

Сократить отрезок  $[x_1, x_2]$ .

**все-цикл**

**Все.**

*Шаг 6.* Детализируем операцию сокращения интервала определения  $x$ . Сокращение отрезка достигается делением пополам и отбрасыванием половины, не удовлетворяющей условию  $f(x_1) \leq y \leq f(x_2)$

**Сократить интервал определения  $x$ .**

```

xt:=(x1+x2)/2
если f(xt)>y
    то    x2:=xt
    иначе x1:=xt
все-если

```

**Все.**

Таким образом, за шесть шагов мы разработали весь алгоритм. Полностью он выглядит следующим образом.

**Программа:**

```

Ввести y,n,eps.
x1:=1
цикл-пока f(x1)>y
    x1:=x1/2
все-цикл
x2:=1
цикл-пока f(x2)<y
    x2:=x2/2
все-цикл
цикл-пока x2-x1>eps
    xt:=(x1+x2)/2
    если f(xt)>y
        то    x2:=xt
        иначе x1:=xt
    все-если
все-цикл
Вывести xt,y.

```

**Конец.**

При разработке алгоритма методом пошаговой детализации мы использовали псевдокод, но можно было использовать и схемы алгоритма.

Достоинством метода является то, что в процессе разработки на каждом шаге мы решали одну достаточно простую задачу. Использование метода, таким образом, существенно облегчает разработку алгоритмов.

#### 1.4 Использование метода пошаговой детализации при структурной декомпозиции

Метод пошаговой детализации позволяет не только разрабатывать алгоритмы, но и осуществлять процедурную декомпозицию, т. е. выделять подпрограммы, соблюдая все требования, предъявляемые структурным программированием.

**Пример 4.** Разработать программу исследования элементарных функций, которая для функций

$$y=\sin x, \quad y=\cos x, \quad y=\operatorname{tg} x, \quad y=\ln x, \quad y=e^x$$

выполняет следующие действия на заданном отрезке:

- строит таблицу значений функции с заданным шагом;
- определяет корни функции;
- определяет максимум и минимум функции.

Программы такого рода обычно взаимодействуют с пользователем через меню. В данной программе будет использовано два меню: меню функций, в котором пользователь будет выбирать функцию, и меню операций, в котором пользователь будет выбирать вид обработки. Так, меню функций (см. рисунок 6, а) должно выводиться на экран при запуске программы.

После ввода номера функции на экране должно появиться меню операций (см. рисунок 6, б). Введя номер операции, пользователь должен увидеть на экране запрос на ввод данных, необходимых для выполнения этой операции (см. рисунок 6, в-д). Задав данные, он должен получить на экране результаты. После анализа результатов необходимо любую клавишу, чтобы вернуться в меню операций.

Далее пользователь может выбрать другую операцию, а может вернуться в меню функций, если закажет операцию 4 (выход).

Программа исследования функций:

1 - sin x  
2 - cos x  
3 - ln x  
4 - e<sup>x</sup>  
5 - выход

Введите номер функции: \_

а

Список операций:

1 - получение таблицы значений;  
2 - определение корней;  
3 - определение экстремумов;  
4 - выход.

Определите операцию: \_

б

Введите интервал: \_  
Введите шаг: \_

**Таблица значений функции.**

x=...	y=...
x=...	y=...
x=...	y=...

Нажмите любую клавишу.

в

Введите интервал: \_  
Введите погрешность: \_

**Корень x=...**  
**значение функции в корне y=...**

Нажмите любую клавишу.

г

Введите интервал: \_  
Введите погрешность: \_

**Минимум y=... при x=...**  
**Максимум y=... при x=...**

Нажмите любую клавишу.

д

**Рисунок 6** - Состояния интерфейса программы исследования функций: меню функций (а), меню операций (б), оформление операций (в-д)

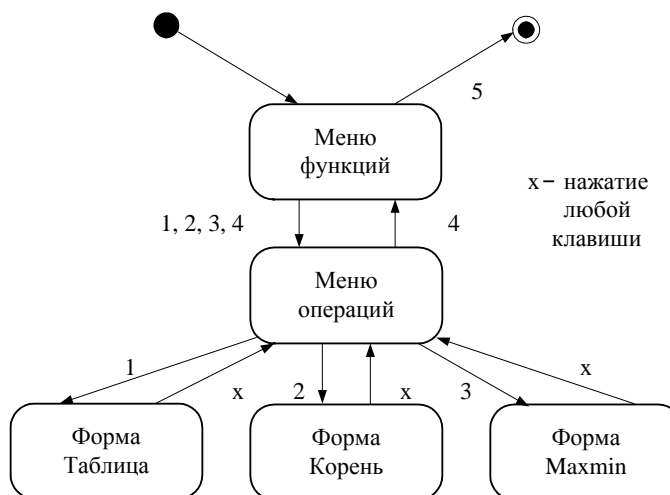
## [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Вернувшись в меню функций, пользователь может выбрать другую функцию, а может выйти из программы.

На рисунке 7 представлен граф состояний интерфейса пользователя в нотации UML (см. раздел 2.3), который показывает, в какой последовательности и по нажатию каких клавиш осуществляется переключение форм (экранов) интерфейса.



**Рисунок 7** - Диаграмма (или граф) состояний интерфейса программы исследования функции

При разработке алгоритма методом пошаговой детализации будем использовать псевдокод. Начинаем разработку алгоритма «сверху», т. е. с реализации меню функций.

Меню функций работает следующим образом. При запуске программы выводим меню на экран. Затем вводим номер функции и, если номер функции не равен 5, то передаем управление меню операций, сообщая ему номер выбранной функции. Когда меню операций завершит свою работу, то на экран вновь необходимо вывести меню функций и опять ввести номер функции:

#### **Программа исследования элементарных функций:**

ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ.

Ввести Номер\_функции.

**Цикл-пока** Номер\_функции  $\neq$  5

    Вызвать МЕНЮ\_ОПЕРАЦИЙ (Номер\_функции)

    ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ.

    Ввести Номер\_функции.

**Все-цикл.**

**Конец.**

На этом шаге определились две подпрограммы: ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ и МЕНЮ\_ОПЕРАЦИЙ (Номер\_функции). Подпрограмма ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ будет иметь линейную структуру:

**Подпрограмма ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ:**

ОЧИСТИТЬ\_ЭКРАН.

Вывести «Программа исследования функций.»

Вывести « $1 - \sin x$ ;».

Вывести « $2 - \cos x$ ;».

Вывести « $3 - \ln x$ ;».

Вывести « $4 - e^x$ ;».

Вывести «5 – Выход.».

**Конец подпрограммы.**

Подпрограмма МЕНЮ\_ОПЕРАЦИЙ (Номер\_функции) также должна реализовать меню:

**Подпрограмма МЕНЮ\_ОПЕРАЦИЙ (Номер\_функции):**

ВЫВЕСТИ\_МЕНЮ\_ОПЕРАЦИЙ.

Ввести Номер\_операции.

**Цикл-пока** Номер\_операции  $\leq 4$

**Выбор** Номер\_операции:

1: Вызвать ТАБЛИЦА (Номер\_функции);

2: Вызвать КОРЕНЬ (Номер\_функции);

3: Вызвать МАКСМИН (Номер\_функции);

**Все-выбор.**

ВЫВЕСТИ\_МЕНЮ\_ОПЕРАЦИЙ. .

Ввести Номер\_операции.

**Все-цикл.**

**Конец подпрограммы.**

Подпрограмма ВЫВЕСТИ\_МЕНЮ\_ОПЕРАЦИЙ похожа на подпрограмму ВЫВЕСТИ\_МЕНЮ\_ФУНКЦИЙ:

**Подпрограмма ВЫВЕСТИ\_МЕНЮ\_ОПЕРАЦИЙ:**

ОЧИСТИТЬ\_ЭКРАН.

Вывести «Список операций:».

Вывести «1 – получение таблицы значений;».

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Вывести «2 – определение корней;».

Вывести «3 – определение экстремумов;».

Вывести «4 – выход.».

Вывести «Определите операцию: ».

#### **Конец подпрограммы.**

Подпрограмма ТАБЛИЦА(Номер\_функции) должна вводить дополнительные данные о границах интервала и значении шага:

#### **Подпрограмма ТАБЛИЦА(Номер\_функции):**

ОЧИСТИТЬ\_ЭКРАН.

Вывести «Введите границы интервала»

Ввести A, B.

Вывести «Введите шаг».

Ввести h.

x=A

#### **Цикл-пока $x \leq B$**

**Если** ФУНКЦИЯ(Номер\_функции, x, y)

**то** Вывести «x=», x, « y=», y

**иначе** «x=», x, «значение функции не определено»

**Все-если**

x=x+h

#### **Все-цикл.**

ОЖИДАТЬ\_НАЖАНИЯ\_КЛАВИШИ.

#### **Конец подпрограммы.**

Подпрограммы КОРЕНЬ (Номер\_функции) и МАКСМИН (Номер\_функции) определяются аналогично. Естественно они также обращаются к подпрограмме ФУНКЦИЯ для определения конкретного значения функции в точке.

Подпрограмма ФУНКЦИЯ будет возвращать логическое значение: true – если значение функции определено, и false – в противном случае. Посчитанное значение исследуемой функции подпрограмма будет возвращать через параметр-переменную y:

#### **Подпрограмма ФУНКЦИЯ(Номер\_функции, x, y):**

ФУНКЦИЯ=true

**Выбор** Номер\_функции:

1: y=sin(x);

#### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



```

2:  $y = \cos(x)$ ;
3: Если  $x > 0$ 
    то  $y = \ln(x)$ 
    иначе ФУНКЦИЯ=false

```

**Все-если**

```

4:  $y = \exp(x)$ 

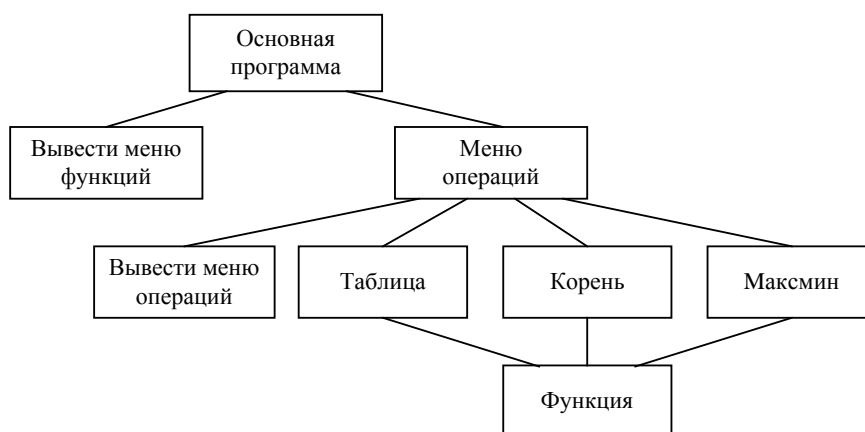
```

**Все-выбор**

**Конец подпрограммы.**

Таким образом, выполняя пошаговую детализацию программы, мы осуществили ее декомпозицию на основную программу и семь подпрограмм (подпрограммы ОЧИСТИТЬ\_ЭКРАН и ОЖИДАТЬ\_НАЖАНИЯ\_КЛАВИШИ являются стандартными процедурами библиотеки управления экраном в текстовом режиме и их можно не учитывать).

Результат процедурной декомпозиции представим в виде структурной схемы (см. рисунок 8).



**Рисунок 8** - Схема структурная программы исследования функций

Использование метода пошаговой детализации, таким образом, существенно облегчает процедурную декомпозицию.

### Контрольные вопросы

1. Что такое «процедурная декомпозиция»?

[Ответ.](#)

2. Что отражает схема структурная при процедурной декомпозиции?

[Ответ.](#)

3. Какой вид процедурной декомпозиции называют структурным? Каковы ее достоинства?

[Ответ.](#)

4. Какие правила используют при изображении схем алгоритмов?

[Ответ.](#)

5. Назовите три базовые управляющие структуры.

[Ответ.](#)

6. Что такое «псевдокод»?

[Ответ.](#)

7. В чем заключается метод пошаговой детализации?

[Ответ.](#)

8. Для чего используется метод пошаговой детализации? Продемонстрируйте работу метода на своем варианте.

[Ответ.](#)

9. С каких элементов следует начинать проектирование программы и почему?

[Ответ.](#)

10. Какая информация представляется в виде диаграммы состояний интерфейса?

[Ответ.](#)

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

## 2 ОБЪЕКТНАЯ ДЕКОМПОЗИЦИЯ

### 2.1 Понятие объектной декомпозиции

*Объектной декомпозицией* называют процесс представления предметной области задачи в виде совокупности функциональных элементов – *объектов*, обменивающихся в процессе выполнения программы входными воздействиями – *сообщениями*.

Каждый выделяемый объект предметной области должен уметь выполнять некоторые действия, зависящие от полученных сообщений и параметров самого объекта.

Совокупность значений параметров объекта называют его *состоянием*, а совокупность реакций на получаемые сообщения – *поведением*. Параметры состояния и элементы поведения объектов определяются условием задачи.

В процессе решения задачи объект, получив некоторое сообщение, выполняет заранее определенные действия, например, может изменить собственное состояние, выполнить некоторые вычисления, нарисовать окно или график и, в свою очередь, сформировать сообщения другим объектам. Таким образом, *процессом решения задачи управляет последовательность сообщений*. Передавая эти сообщения от объекта к объекту, система выполняет необходимые действия.

Различие процедурной и объектной декомпозиции предметной области задачи рассмотрим на примере разработки программы исследования элементарных функций, приведенной в примере 4.

**Пример 5.** Разработать программу исследования элементарных функций, которая для функций  $y=\sin x$ ,  $y=\cos x$ ,  $y=\operatorname{tg} x$ ,  $y=\ln x$ ,  $y=e^x$  выполняет следующие действия на заданном отрезке:

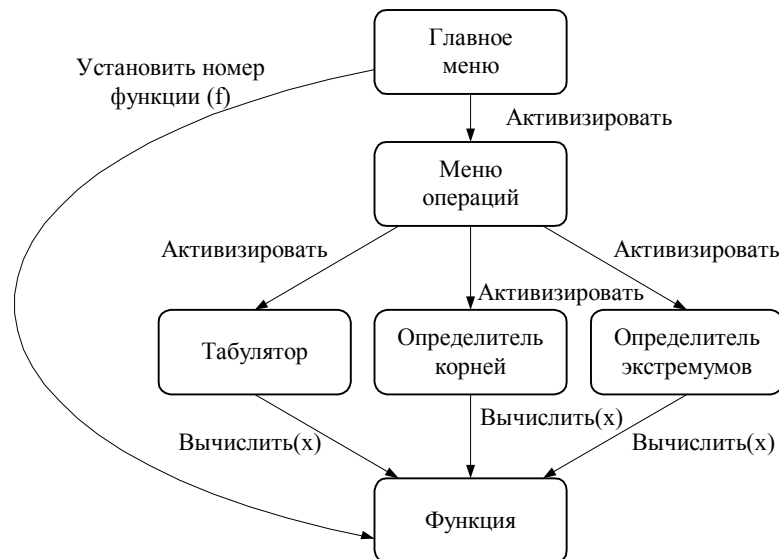
- строит таблицу значений функции с заданным шагом;
- определяет корни функции;
- определяет максимум и минимум функции.

В основе объектной декомпозиции также лежит граф состояний интерфейса (см. рисунки 6–7). Будем считать, что каждое состояние интерфейса – это состояние некоторого функционального элемента системы, т. е. объекта. Состояний интерфейса пять, соответственно получаем пять объектов. Назовем эти объекты следующим образом: Главное ме-

ню, Меню операций, Табулятор, Определитель корней, Определитель экстремумов. Эти объекты передают управление друг другу, генерируя сообщение Активизировать.

Кроме этого можно выделить еще один объект Функция, который должен обеспечивать вычисление выбранной функции по заданному аргументу. Номер функции сообщается данному объекту Главным меню, после того, как пользователь осуществит выбор.

Результат объектной декомпозиции изображают в виде *диаграммы объектов* (см. рисунок 9).



**Рисунок 9** - Диаграмма объектов

Согласно определению схемы структурной, как схемы, отражающей *состав программного обеспечения и взаимодействие по управлению* его частей, диаграмма объектов является также схемой структурной сравнительно небольшой программы при объектной декомпозиции. Для сложных программных систем схема структурная отражает взаимодействие между более сложными частями программы – подсистемами, и обычно выглядит так же, как при структурной декомпозиции.

Полная характеристика объекта включает идентифицирующее условное имя, а также перечень и описание параметров состояния и аспектов поведения.

Так состояние объекта Функция характеризуется единственным параметром: номером функции, который передает ему Главное меню. А поведение включает реакции на два типа сообщений: получив номер функции, объект должен сохранить его, изменив свое состояние, а получив запрос на вычисление значения функции, сопровождающийся определенным значением аргумента – вернуть значение функции в заданной точке.

Таким образом, при выполнении объектной декомпозиции определяют и описывают множество объектов предметной области и множество сообщений, которое формирует и получает каждый объект.

## 2.2 Построение диаграмм классов

Для реализации объектов используют специальные типы – классы. *Класс* – это структурный тип данных, который включает описание полей данных, а также процедур и функций, работающих с этими полями данных.

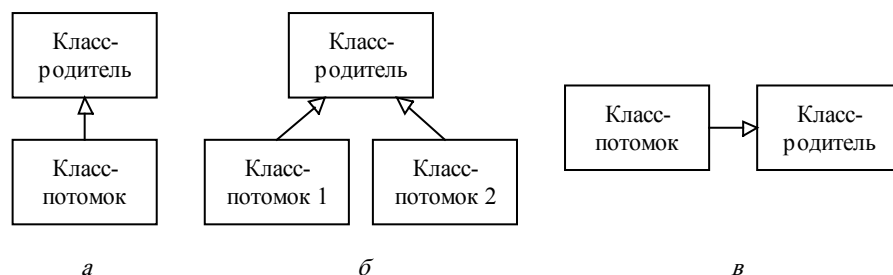
Одним из наиболее значимых достоинств ООП является то, что большинство классов для реализации объектов не приходится разрабатывать «с нуля». Обычно классы строят на базе уже существующих, используя механизмы, реализующие определенное отношение существующего и строящего классов между собой: наследование, композицию, наполнение, полиморфное наследование.

*Наследованием* называют отношение между классами, при котором один класс строится на базе второго посредством добавления полей и определения новых методов. При этом исходный класс, на базе которого выполняется построение, называют *родительским* или *базовым*, а строящейся класс – *потомком* или *производным* классом.

При наследовании поля и методы родительского класса повторно не определяют, предусмотренный механизм наследования позволяет использовать эти компоненты класса, специально этого не оговаривая.

Отношения между различными классами проекта принято иллюстрировать *диаграммой отношений классов* или просто *диаграммой классов*. Если на диаграмме отношений классов показано только отношение наследования, то такую диаграмму называют *иерархией классов*.

На диаграмме классов согласно UML (см. раздел 2.3) наследование изображают линией со стрелкой, направленной к классу-родителю (см. рисунок 10).



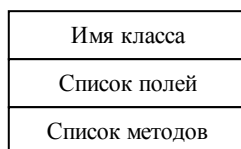
**Рисунок 10** - Примеры иерархий классов: с одним потомком (а, в) и с двумя потомками (б)

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Кроме отношения между классами на диаграмме классов целесообразно бывает указывать поля и методы каждого или только строящегося класса, так как это позволяет уточнить структуру разрабатываемых классов. Поля и методы перечисляют в специальных секциях уточненного обозначения класса (см. рисунок 11).



**Рисунок 11** - Условное обозначение класса с указанием полей и методов

*Композицией* называют такое отношение между классами, когда один является частью второго. Физически, композиция реализуется включением в класс фиксированного количества полей, являющихся объектами другого класса. Такие поля обычно называют *объектными*.

На диаграмме классов композицию изображают линией с ромбом на конце, подходящем к более сложному классу (см. рисунок 12).



**Рисунок 12** - Примеры диаграмм классов, изображающих композицию: с одним объектным полем (а) и несколькими объектными полями различных типов (б)

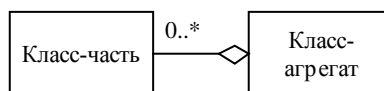
*Наполнением* называют такое отношение между классами, при котором точное количество объектов одного класса, включаемых в другой класс, не ограничено и может меняться от 0 до достаточно больших значений. Физически наполнение реализуется с использованием указателей на объекты. В отличие от объектного поля, которое включает в класс точно указанное количество объектов (1 или более – при использовании массива объектов или нескольких объектных полей) конкретного класса, использование указателей позволяет включить 0 или более объектов, если они собраны в массив или списковую (линейную или нелинейную) структуру.

На диаграмме классов наполнение изображают аналогично композиции, но ромб не закрашивают, показывая более слабую связь объектов классов (см. рисунок 13).

## [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



**Рисунок 13** - Пример диаграммы классов с наполнением

Для определения отношения имеющегося и строящегося классов необходимо выполнить анализ структуры объектов предметной области, полученных в результате объектной декомпозиции.

Если объекты предметной области слишком сложны, чтобы ставить им в соответствие некий простой класс, то процесс декомпозиции можно продолжить, выделяя внутри сложных объектов более простые.

При этом возможны следующие варианты.

1. Внутри объекта можно выделить объект близкого назначения, но с более простой структурой и/или поведением – класс для реализации данного объекта следует строить на базе более простого, используя наследование. Если при этом объекты строящего класса отличаются от объектов базового класса некоторыми аспектами поведения, то следует изменить поведение объектов строящего класса при наследовании, используя полиморфное наследование с переопределение методов.

2. Внутри объекта можно выделить определенное количество объектов более простой структуры со своим поведением – класс конструируется объединением объектов других классов с добавлением полей и методов строящегося класса – композиция классов.

3. Внутри объекта можно выделить заранее не предсказуемое количество объектов более простой структуры со своим поведением – класс конструируется с возможностью динамического подключения объектов других классов (наполнение) и добавлением полей и методов строящегося класса.

Рассмотрим два примера.

**Пример 6.** Разработать класс для реализации объекта «Текст», который должен:

- для каждого слова некоторой последовательности слов хранить его атрибуты (тип шрифта и размер букв);
- определять количество слов в тексте;
- добавлять слова после слов с указанными номерами;
- удалять заданные слова из текста;
- менять местами слова с заданными номерами;
- заменять одно заданное слово на другое во всем тексте;
- позволять переопределять атрибуты слова с заданным номером.

#### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Итак, реализуемый объект должен оперировать с некоторыми внутренними объектами «Словами», для которых можно определить собственное состояние и поведение. Естественно создать специальный класс TWord для реализации «Слов». Класс TText для реализации «Текста» может быть построен как с использованием композиции, так и с использованием наполнения.

В первом случае он должен включать массив объектов класса TWord. Максимальное количество элементов массива должно быть определено заранее и, следовательно, ограничено (см. рисунок 14, а). При выполнении операций удаления и вставки придется сдвигать и раздвигать элементы массива.

Во втором случае класс TText должен включать список объектов класса TWord (см. рисунок 14, б). Ограничения предыдущей реализации будут сняты, но реализация со списком имеет несколько большую трудоемкость, и, кроме того, при обращении к слову по номеру придется каждый раз последовательно отсчитывать нужный элемент. Выбор конкретного варианта реализации зависит от условий решаемой задачи.

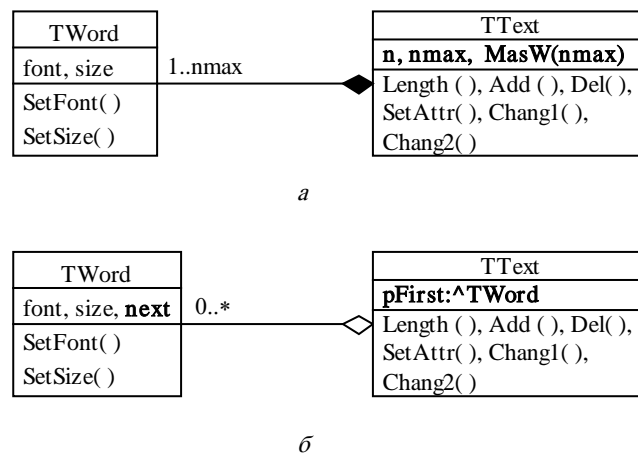


Рисунок 14 - Диаграммы классов для реализации объекта Текст: с композицией (а) и с наполнением (б)

**Пример 7.** Разработать классы для реализации объектов Табулятор, Определитель корней и Определитель экстремумов из примера 3.

Объекты Табулятор, Определитель корней и Определитель экстремумов отвечают за реализацию методов решения некоторых частных задач при исследовании функций. Они имеют много общего. Попробуем определить это общее.

Любой объект, получив управление должен ввести диапазон изменения аргумента [a, b], решить подзадачу, вывести результаты, а затем вернуть управление меню операций. Общее поведение и поля объектов опишем в классе TMethod. Основной метод этого класса Run должен реализовывать общее поведение и обеспечивать возможность изменения элементов этого поведения (решения конкретных подзадач) для объектов классов, которые

#### [Оглавление](#)

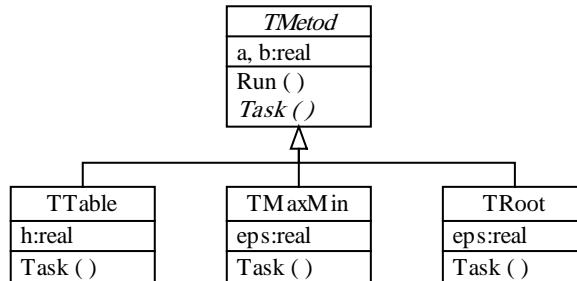
Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



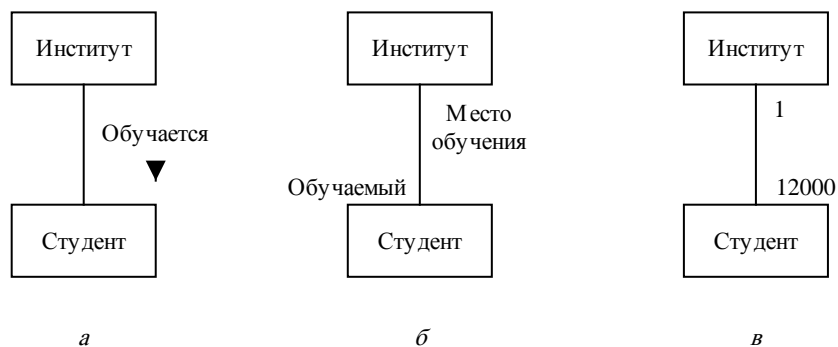
будут от него наследоваться. Решение конкретной подзадачи реализуем как внутренний метод Task, вызываемый из метода Run. Этот внутренний метод для класса TMethod определять не будем (абстрактный метод).

Классы для реализации разрабатываемых объектов наследуем от TMethod, переопределяя метод Task и добавляя недостающие поля (см. рисунок 15).



**Рисунок 15** - Иерархия классов для реализации объектов Табулятор, Определитель корней и Определитель экстремумов

При разработке программ с использованием объектного подхода часто строят диаграммы классов, на которых показывают также отношение *ассоциации*, которое отражает взаимодействие объектов классов между собой в процессе работы программы. Поскольку наполнение и композиция также предполагают взаимодействие, ассоциацию используют либо на ранних этапах разработки, либо, если отношение классов не подходит под определение композиции или агрегации. Ассоциация может иметь имя, каждый элемент ассоциации может иметь имя и указание множественности (см. рисунок 16).



**Рисунок 16** - Обозначение ассоциации:

*а* – с указанием имени ассоциации и ее направления; *б* – с указанием имен ролей;  
*в* – с указанием множественности

## 2.3 Язык описания объектных разработок

С середины 90-х годов для документирования объектных разработок все шире применяют язык UML (Unified Modeling Language – Унифицированный Язык Моделирова-

[Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

ния). В настоящее время он фактически признан стандартным средством описания проектов, создаваемых с использованием объектно-ориентированного подхода. Создателями этого языка являются ведущие специалисты в этой области: Гради Буч, Ивар Якобсон и Джеймс Рамбо, которые использовали в своем языке все лучшее, что появилось в подходах различных авторов в предыдущие годы.

Модель проекта программного обеспечения по замыслу авторов языка может включать большое количество диаграмм различных типов, но использующих единую систему обозначений. Среди диаграмм наиболее часто используемыми являются:

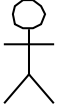

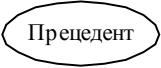
- *диаграммы вариантов использования* или прецедентов (uses case diagrams) – показывают основные функции системы для каждого типа пользователей – применяются на этапе анализа требований и построения спецификаций;
- *диаграммы классов* (class diagrams): контекстные, описания интерфейсов и реализации – демонстрируют отношения классов между собой – используются соответственно на этапе анализа, этапе проектирования и этапе реализации;
- *диаграммы деятельности* (activity diagrams) – представляет собой схему потоков управления для решения некоторой задачи по отдельным действиям, допускает наличие параллельных и/или альтернативных действий – применяются при анализе потоков действий одного или взаимодействующих вариантов использования;
- *диаграммы взаимодействия* (interaction diagrams) двух альтернативных типов:
  - а) *диаграммы последовательности действий* (sequence diagrams) – отображает упорядоченное по времени взаимодействие объектов в процессе выполнения варианта использования – применяются на стадии анализа для выявления ответственности каждого класса,
  - б) *диаграммы кооперации* (collaboration diagrams) – предоставляют ту же информацию, что и диаграммы последовательности действий, но в другой форме;
- *диаграммы состояний объекта* (statechart diagrams) – показывают состояния объекта и условия переходов из одного состояния в другое – используются для проектирования объектов с большим количеством состояний;
- *диаграммы пакетов* (package diagrams) – демонстрируют связи наборов классов, объединенных в пакеты, между собой – может использоваться вместо диаграммы классов на этапе анализа и/или физического проектирования программного обеспечения;
- *диаграммы компонентов* (component diagrams) – показывают, из каких программных компонентов состоит система и как эти компоненты связаны между собой;

- *диаграммы размещения* (deployment diagrams) – позволяет связать программные и аппаратные компоненты системы – в основном используется при проектировании распределенных программных систем.



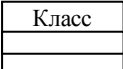

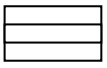


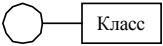

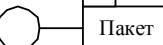

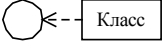
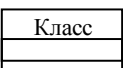
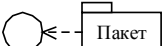

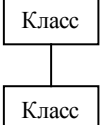
При разработке этих диаграмм используют условные обозначения, представленные в таблицах 3 - 7.

Кроме того, спецификация может включать формализованные и неформализованные текстовые описания, комментарии и словари.

**Таблица 3 - Условные обозначения вариантов использования**

№	Компонент модели	Условное обозначение	№	Компонент модели	Условное обозначение
1	Действующее лицо		3	Связь	
2	Вариант использования или прецедент		4	Связи «Расширение» и «Использование»	

**Таблица 4 - Условные обозначения диаграмм классов и пакетов**

№	Компонент модели	Условное обозначение	№	Компонент модели	Условное обозначение
1	2	3	4	5	6
1	Класс со скрытыми секциями		16	Обобщение	
2	Класс с раскрытыми секциями		17	Интерфейс	
3	Класс (пиктограмма)		18	Реализация интерфейса (раскрытая)	
4	Граничный класс		19	Реализация интерфейса классом	
5	Управляющий класс		20	Реализация интерфейса пакетом	
6	Класс-сущность		21	Использование интерфейса классом	
7	Активный класс		22	Использование интерфейса пакетом	
8	Абстрактный класс		23	Двунаправленная ассоциация	

### [Оглавление](#)

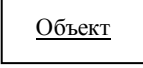







Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Таблица 4 (окончание)

1	2	3	4	5	6
9	Видимость атрибутов класса	+ Общий - Скрытый # Защищенный	24	Однонаправленная ассоциация	
10	Абстрактная операция класса		25	Агрегация	
11	Параметризованный класс		26	Композиция	
12	Настроенный класс		27	Отношение ассоциации класса	
13	Пакет со скрытой секцией		28	Зависимость классов	
14	Пакет с раскрытой секцией		29	Зависимость пакетов	
15	Пакет (пиктограмма)		30	Примечание	

Таблица 5 - Условные обозначения диаграмм последовательностей действий





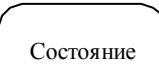

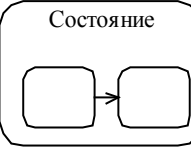
№	Компонент модели	Условное обозначение	№	Компонент модели	Условное обозначение
1	Объект		5	Возврат управления из процедуры	
2	Вызов процедуры или вложенного потока управления		6	Линия жизни	
3	Простой поток управления		7	Фокус управления	
4	Асинхронный поток сообщений		8	Разрушение объекта	

### [Оглавление](#)


Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

**Таблица 6** - Условные обозначения диаграмм состояний и деятельности

№	Компонент модели	Условное обозначение	№	Компонент модели	Условное обозначение
1	Начало		5	Переход	
2	Конец		6	Линейки синхронизации	
3	Деятельность		7	Состояние	
4	Выбор		8	Составное состояние	

**Таблица 7** - Условные обозначения диаграмм размещения и компоновки

№	Компонент модели	Условное обозначение	№	Компонент модели	Условное обозначение
1	Программный компонент		4	База данных	
2	Текстовый файл		5	Таблица	
3	DLL		6	Узел	

## 2.4 Документирование объектных разработок в среде Delphi

Документация простейших объектных разработок в среде Delphi должна включать:

- эскизы форм приложения с указанием типов визуальных интерфейсных компонентов;
- диаграммы состояний интерфейса в целом;
- диаграмму объектов приложения;
- диаграмму состояний каждой формы;
- диаграмму классов для каждой формы;
- диаграмму классов предметной области программы;
- диаграмма последовательностей действий для каждой функции, доступной пользователю.

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Для сложных методов возможно добавление схем алгоритмов или диаграмм деятельности. Информация, предоставляемая этими схемами примерно одинакова, но диаграммы деятельности при необходимости позволяют уточнить, какой объект выполняет то или другое действие.

**Пример 8.** В качестве примера рассмотрим описание разработки приложения «Записная книжка». Приложение должно обеспечивать ввод имени, фамилии и телефона, а также поиск абонента по имени и/или фамилии.

Эскиз главной формы приложения приведен на рисунке 17.

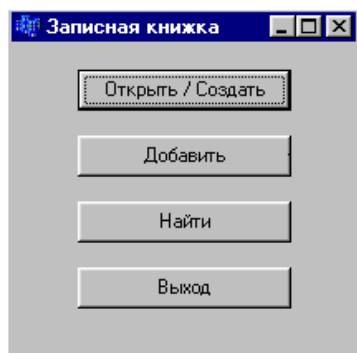


Рисунок 17 - Внешний вид главного окна приложения

При нажатии кнопок приложение должно переходить в режимы добавления записей и поиска телефонов (см. рисунок 18)

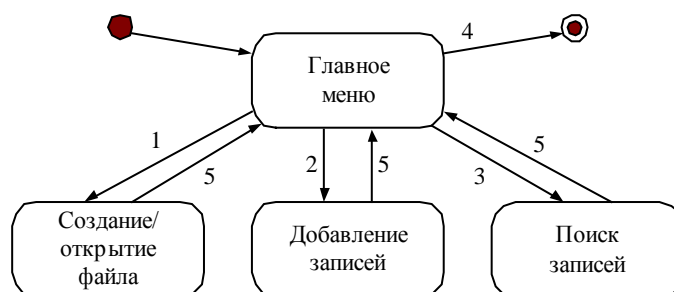
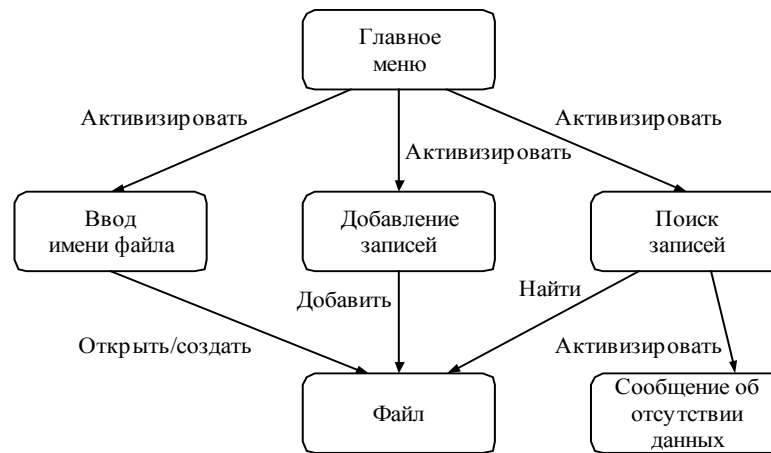


Рисунок 18 - Диаграмма состояний интерфейса приложения

Если каждый режим реализован на своей форме, то получаем диаграмму объектов интерфейса, представленную на рисунке 19. (Если файл записей реализовывать как объект, то объектная декомпозиция будет включать объект Файл).



**Рисунок 19** - Объектная декомпозиция приложения

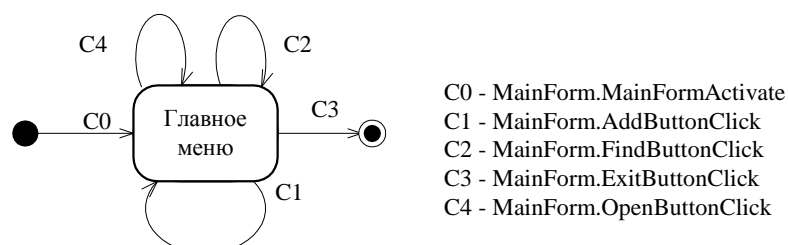
### 1. Проектирование объекта Главная форма

Проектирование объекта Главная форма включает уточнение внешнего вида с указанием компонентов экранной формы (см. рисунок 20).



**Рисунок 20** - Экранная форма Главное меню

Для объекта формы прорабатываем диаграмму состояний и привязываем изменение состояния к событиям Delphi (см. рисунок 21).



**Рисунок 21** - Диаграмма состояний интерфейса формы Главное меню

Далее проектируем диаграмму классов формы (см. рисунок 22).

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

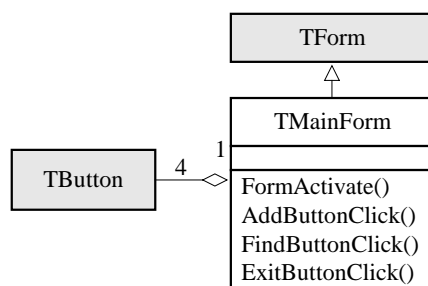


Рисунок 22 - Диаграмма классов для TMainForm

## 2. Проектирование вспомогательных форм

Проектирование вспомогательных форм выполняем аналогично. Так проектирование формы Поиск записей показано на рисунках 23–25.

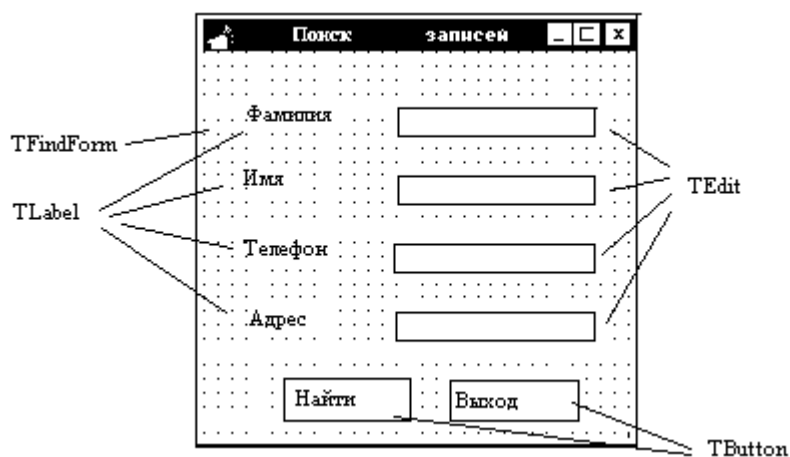


Рисунок 23 - Экранная форма Поиск записей

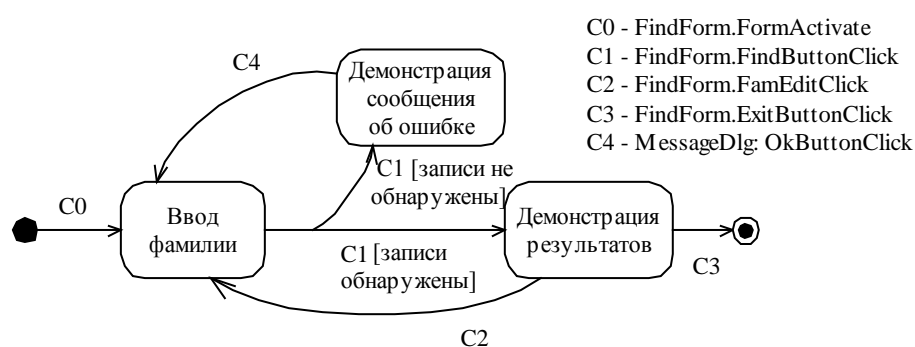
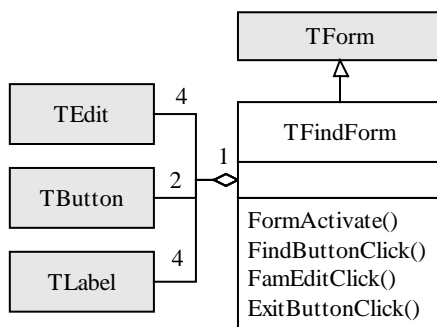


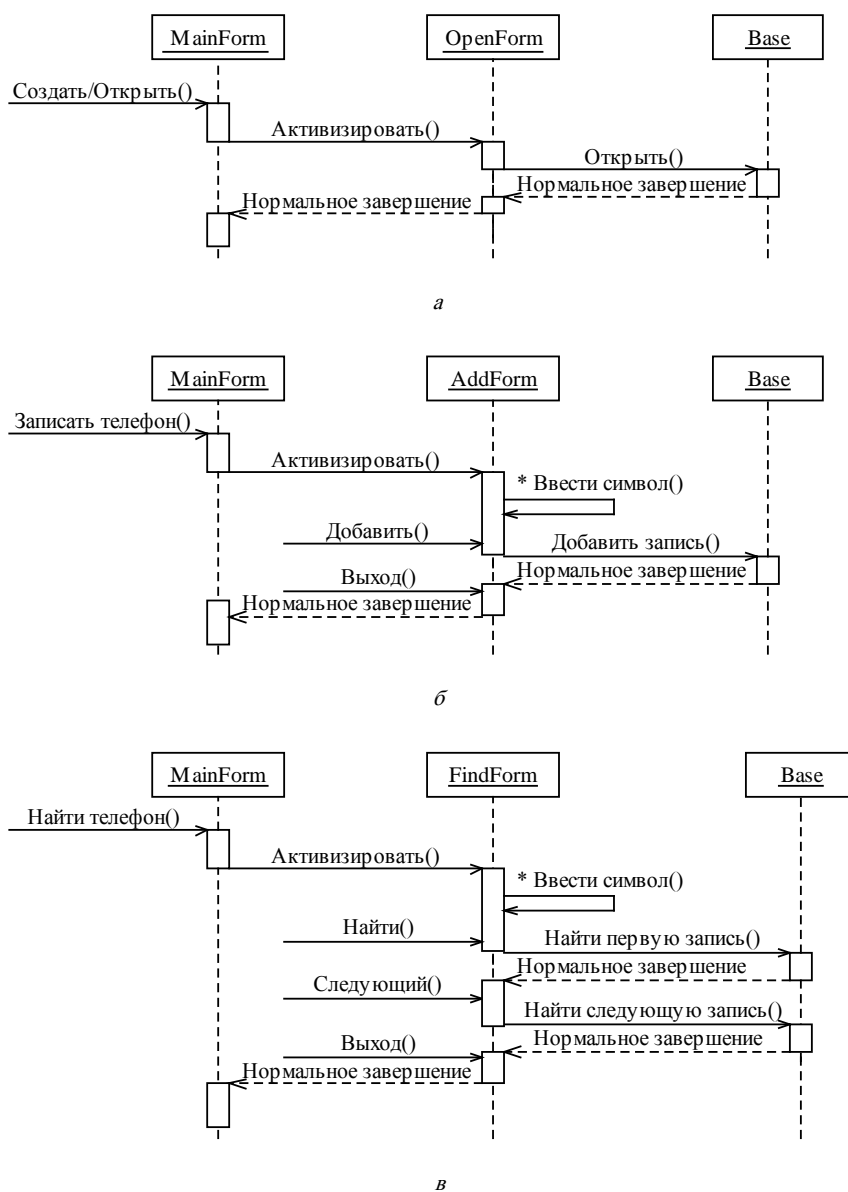
Рисунок 24 - Диаграмма состояний формы Поиск записей





**Рисунок 25** - Диаграмма класса TFindForm

Взаимодействие между объектами приложения покажем с помощью диаграммы последовательностей действий (см. рисунок 26).



**Рисунок 26** - Диаграммы последовательностей действий при выполнении функций:

*а* – создать/открыть книжку; *б* – записать телефон; *в* – найти телефон

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

Для метода, обрабатывающего событие FindButtonClick(), приведем диаграмму деятельности с уточнением ответственности объектов (см. рисунок 27, а) и схему алгоритма (см. рисунок 27, б). Сравнение показывает очевидное сходство этих схем, хотя, как уже упоминалось ранее, диаграмма деятельности более информативна. В качестве упражнения сравните эту диаграмму также с диаграммой состояний формы FindForm (см. рисунок 24) и диаграммой последовательностей действий для функции Найти (см. рисунок 26, в). Хорошо видно, что данные диаграммы уточняют друг друга.

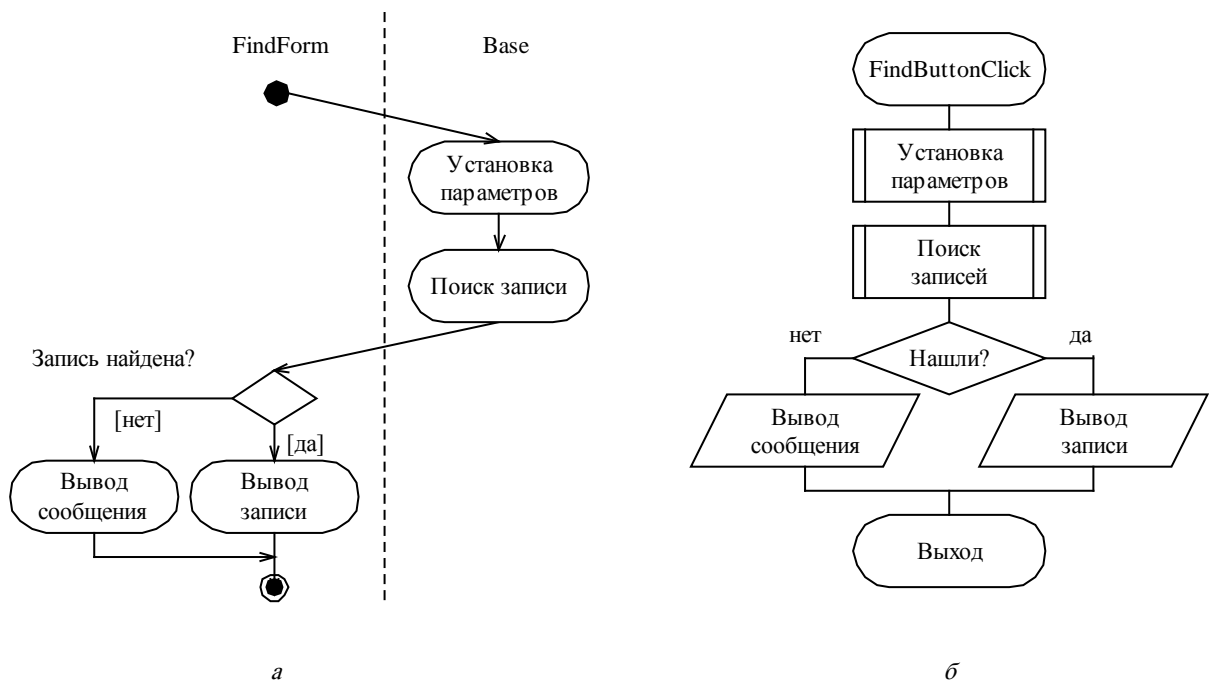


Рисунок 27 - Диаграмма активностей (а) и схема алгоритма (б) метода FindButtonClick()

## 2.5 Документирование объектных разработок на языке C++

В качестве примера рассмотрим программу создания движущихся изображений

**Пример 9.** В качестве примера рассмотрим программу создания движущихся изображений. Пусть на экране в окне программы вращаются две фигуры: Линия и Квадрат (см. рисунок 28). Единственная кнопка должна иметь название Завершить и соответственно должна закрывать окно и завершать приложение.

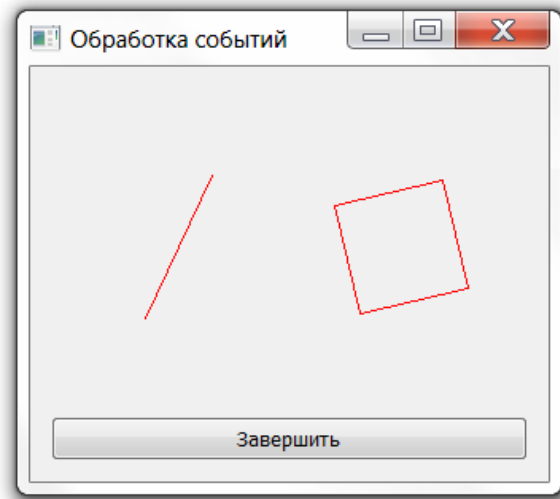
Проектирование программы начнем с разработки диаграммы объектов. Анализ показывает, что приложение может состоять из 6 объектов:

- Окна приложения,
- Холста, на котором будет выполнено рисование,
- Кнопки завершения приложения,

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



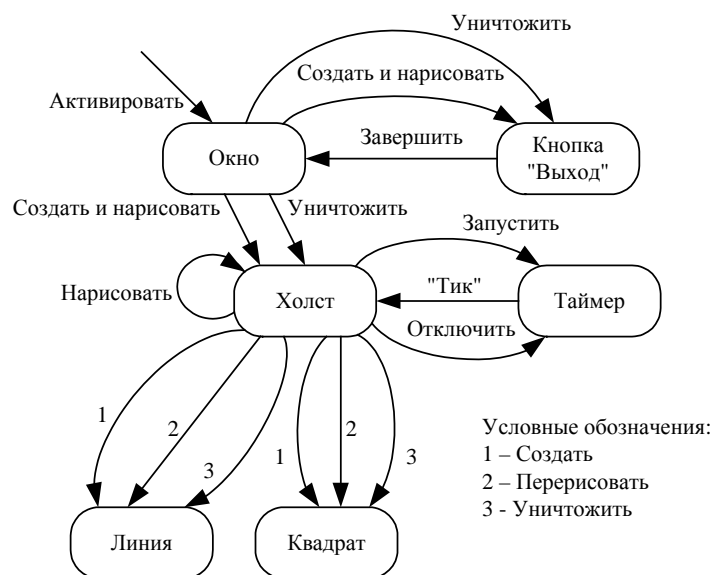
**Рисунок 28** - Внешний вид окна приложения во время работы программы

- фигур Линия и Квадрат,
- Таймера.

Первые три объекта – интерфейсные. Окно, будучи главным, активируется при запуске приложения. Оно должно создавать Кнопку и Холст, отвечать за их рисование и освобождение их памяти. Назначение Кнопки – посылать сигнал закрытия Окну. По этому сигналу объект Окна должен уничтожаться, в процессе чего должен освобождать память сам и посылать соответствующие сигналы своим виджетам Кнопке и Холсту.

Холст будет отвечать за создание, перерисовку и уничтожение фигур. Он также должен запускать Таймер, обрабатывать сигналы от него и отключать Таймер при своем уничтожении.

Диаграмма объектов приложения показана на рисунке



**Рисунок 29** – Диаграмма объектов приложения

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



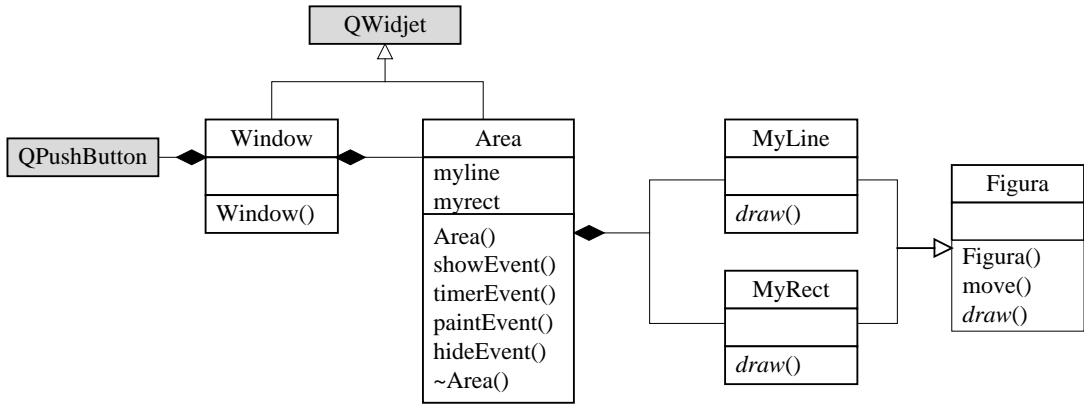


Рисунок 31 – Диаграмма классов приложения

Особый интерес для C++ представляет компоновка программы. В нашем случае целесообразно использовать отдельные модули для описания классов Window и Area, а также классов предметной области. Диаграмма компоновки программы представлена на рисунке 32.

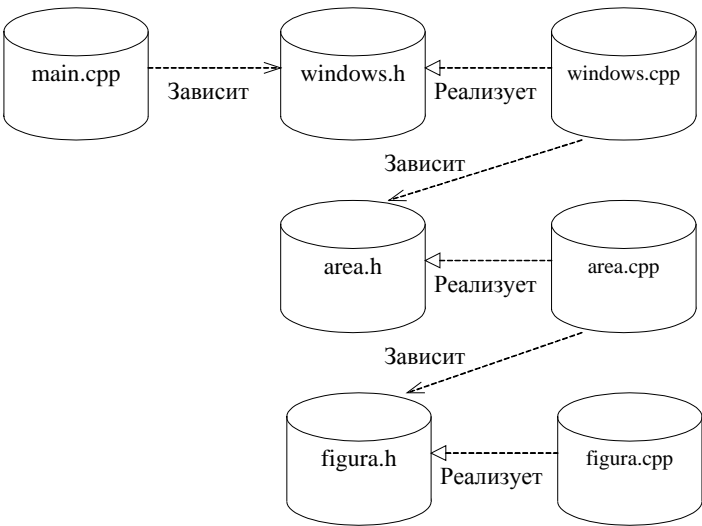


Рисунок 32 – Диаграмма компоновки приложения

### Контрольные вопросы

1. Что понимают под объектной декомпозицией? Чем объектная декомпозиция отличается от процедурной?

[Ответ.](#)

2. Чем характеризуется объект предметной области? Как объект предметной области представляют в программе?

[Ответ.](#)

3. Что показывает диаграмма классов? Какие механизмы используют для построения новых классов из уже существующих?

[Ответ.](#)

4. Как определяют, какой механизм использовать при разработке нового класса?

[Ответ.](#)

5. Какое отношение между классами называют ассоциацией?

[Ответ.](#)

6. Как называют язык документирования объектных разработок? Кто его авторы?

[Ответ.](#)

7. Назовите основные диаграммы UML. Для чего их используют?

[Ответ.](#)

8. Поясните по вашему отчету, какие диаграммы вы использовали и почему?

### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

### **3 ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАНИЙ ПРАКТИКУМА И ТРЕБОВАНИЯ К ОТЧЕТАМ ПО КАЖДОМУ ЗАДАНИЮ**

#### **3.1 Задание 1. Turbo Delphi. Создание программной системы**

Текст задания по номеру варианта приведен в специальном файле.

##### **Порядок выполнения задания**

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Для заданного варианта программной системы продумать и разработать формы интерфейса. Построить диаграмму переходов состояний интерфейса.
3. Выполнить объектную декомпозицию, разработать диаграмму объектов.
4. Разработать диаграмму классов приложения с использованием основных механизмов проектирования классов.
5. Разработать диаграмму последовательностей для одной из операций.
6. В среде программирования Turbo Delphi создать новый проект и описать классы.
7. Отладить программу.
8. Продемонстрировать работу программы преподавателю.
9. Составить отчет по заданию практики.
10. Защитить задание преподавателю.

##### **Требования к отчету**

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема домашнего задания;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.

Кроме того отчет по практике должен содержать:

- 1) текст задания;
- 2) формы приложения, диаграммы состояний интерфейса для двух форм;
- 3) диаграмму объектов;
- 4) диаграмму классов, диаграмму последовательностей для одной из операций;
- 5) текст программы;
- 6) результаты тестирования программы;

#### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»

7) выводы.

### **3.2 Задание 2. С++. Создание программной системы с элементарным интерфейсом консольного режима**

Текст задания по номеру варианта приведен в специальном файле.

#### **Порядок выполнения задания**

1. Прочитать и проанализировать задание.
2. Для заданного варианта программной системы продумать и разработать формы интерфейса. Построить диаграмму переходов состояний интерфейса.
3. Выполнить структурную декомпозицию методом пошаговой детализации. Разработать структурную схему и схемы алгоритмов.
4. В среде программирования Visual Studio создать новый проект приложения и записать реализацию программы.
5. Отладить программу.
6. Продемонстрировать работу программы преподавателю.
7. Составить отчет по заданию практики.
8. Защитить задание преподавателю.

#### **Требования к отчету**

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

- а) наименование факультета и кафедры;
- б) название дисциплины;
- в) номер и тема домашнего задания;
- г) фамилия преподавателя, ведущего занятия;
- д) фамилия, имя и номер группы студента;
- е) номер варианта задания.

Кроме того отчет по практике должен содержать:

- 1) текст задания;
- 2) формы приложения, диаграмму состояний интерфейса;
- 3) описание алгоритма методом пошаговой детализации, схему структурную и схемы алгоритмов;
- 4) текст программы;
- 5) результаты тестирования программы;
- 6) выводы.

#### [Оглавление](#)

Г.С. Иванова, Т.Н. Ничушкина

«Документирование программ при структурной и объектной декомпозиции»



### 3.3 Задание 3. С++ Создание программной системы с Qt интерфейсом

Текст задания по номеру варианта приведен в специальном файле. Интерфейс реализовать с использованием библиотеки Qt.

#### **Порядок выполнения задания**

Порядок выполнения задания и требования к отчету совпадают с указанными для задания 1. Кроме того, выводы отчета должны содержать сравнительный анализ реализаций в Turbo Delphi и с использованием библиотеки Qt.

## ЛИТЕРАТУРА

1. Подбельский В.В. Язык C++: Учеб. пособие. М.: Финансы и статистика, 2006. 560 с.
2. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование. Учеб. для вузов. М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. Гл. 5. С 218-245.
3. Иванова Г.С., Ничушкина Т.Н., Самарев Р.С. . C++. Часть 1. Средства процедурного программирования Microsoft Visual C++ 2008: Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2010. 126 с.
4. Иванова Г.С., Ничушкина Т.Н. C++. Часть 2. Объектно-ориентированное программирование на языке C++ в среде Visual Studio 2008: Учебное пособие. М.: МГТУ им. Н.Э. Баумана, 2012. 161 с.
5. Иванова Г.С. . C++. Часть 3. Создание пользовательских интерфейсов в программах на C++ с использованием библиотеки Qt: Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2010. 52 с.
6. Иванова Г.С. Создание многооконных приложений с использованием библиотеки Qt: Учебное пособие по дисциплине «Практикум по программированию». – М.: МГТУ им. Н.Э. Баумана, 2013. 24 с.
7. Иванова Г.С., Ничушкина Т.Н. Интерфейсные компоненты Visual Components Library: Методические указания по выполнению лабораторных работ и домашних заданий в среде Turbo DELPHI 2006. М.: МГТУ им. Н.Э. Баумана, 2011. 35 с.