



Laboratory Work #05

Java Basic Syntax. Loop Statements. Iteration Algorithms



LEARN. GROW. SUCCEED.

© 2020-2021. Department: <Software of Information Systems and Technologies>
Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #05

Базовый синтаксис языка Java.

Циклические конструкции.

Итерационные алгоритмы

Цель работы

Изучить синтаксис циклических конструкций языка Java для программирования итерационных алгоритмов и закрепить их на примере разработки простейших интерактивных консольных Java-приложений.

Требования

- 1) Каждому студенту нужно просмотреть все задания и выполнить по одному понравившемуся заданию из каждого раздела.
- 2) Для вычислительных алгоритмов (решается самостоятельно) необходимо спроектировать блок-схему решения, которую необходимо поместить в отчёт или конспект.
- 3) Все алгоритмы должны быть решены с использованием итеративного подхода (однако, если есть желание, то можно также привести параллельно решение задания с использованием других подходов).
- 4) На базе спроектированных алгоритмов разработать простейшее интерактивное консольное приложение с использованием архитектурного шаблона проектирования **Model-View-Controller, MVC**.
- 5) Создаваемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны быть вложены в указанные стартовые пакеты: **by.bntu.fitr.poisit.nameofstudent.javalabs.lab05**.
- 6) Название пользовательских идентификаторов (имён пакетов, классов, локальных переменных, методов и т.д.) должны быть осмысленными (нести в себе логический смысл своего содержимого) и удовлетворять Java-соглашению по именованию.

- 7) При выполнении задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.
- 8) Также рекомендуется придерживаться **Single Responsibility Principle, SRP** (принципа единственной ответственности): у каждого пакета, класса или метода должна быть только одна ответственность (цель), т.е. должна быть только одна причина изменить в дальнейшем соответствующий блок кода.
- 9) Если логически не подразумевается или в задании иного не указано, то входными и выходными данными являются вещественные числа (числа с плавающей запятой).
- 10) Все задания необходимо решать используя только базовые операции (простые операторы), определённые над примитивными типами данных в языке программирования Java, условные и циклические конструкции (т.е. не нужно использовать массивы или любые другие контейнеры данных, операции над строковыми типами данных и т.д.).
- 11) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».
- 12) При проверке работоспособности приложения необходимо проверить все тестовые случаи.
- 13) Для генерирования случайных чисел воспользуйтесь методами объекта класса **java.util.Random**, а для реализации ввода данных с консоли (терминала) – соответствующими методами объекта класса **java.util.Scanner**.
- 14) Программа должна обязательно быть снабжена комментариями, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчиков, название бригады (если есть), номер группы и дату разработки. Исходный текст классов и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.
- 15) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем. Интерфейс программы и комментарии в коде должны быть на английском языке.
- 16) При разработке программ придерживайтесь соглашений по написанию кода на Java (**Java Code-Convention**) !!!

Задание А

- 1) Написать программу, которая подсчитывает количество цифр заданного натурального числа.
- 2) Написать программу, которая находит сумму (или произведение) цифр заданного натурального числа.
- 3) Написать программу, которая находит арифметическое и геометрическое среднее цифр заданного натурального числа.
- 4) Написать программу, которая подсчитывает количество цифр заданного натурального числа, которое кратно двум (или трём и т.д.).
- 5) Написать программу, которая подсчитывает количество только чётных (или только нечётных) цифр заданного натурального числа.

Задание В

- 1) Разработайте программу, которая проверяет, что все цифры, которые входят в заданное натуральное число, имеют одинаковую чётность, т.е. либо все чётные, либо все нечётные.
- 2) Разработайте программу, которая проверяет, что все цифры заданного натурального числа различны (или одинаковы).
- 3) Разработайте программу, которая проверяет, что среди цифр заданного натурального числа есть хотя бы одна пара совпадающих.
- 4) Разработайте программу, которая проверяет, что среди цифр заданного натурального числа хотя бы одно число чётное (или нечётное).
- 5) Разработайте программу, которая проверяет, что среди цифр заданного натурального числа преобладают чётные (или нечётные).

Задание С

- 1) Разработайте программу, которая проверяет, что цифры заданного натурального числа образуют возрастающую (или убывающую) последовательность. К примеру, число 9876 удовлетворяет условию, т.к. его цифры соответствуют следующему неравенству: $9 > 8 > 7 > 6$, т.е. идут в порядке убывания. Число 468 также удовлетворяет условию, т.к. его цифры соответствуют неравенству

$4 < 6 < 8$, т.е. идут в порядке возрастания. А вот числа 13243546, 192837 и 777 не удовлетворяют условию.

- 2) Разработайте программу, которая проверяет, что заданное натуральное число читается одинаково слева направо и справа налево (т.е. является палиндромом). К примеру, числа 1235321, 112211, 7 и 1221 – удовлетворяют условию, а числа 12345321, 1000 и 12 – нет.
- 3) Разработайте программу, которая переворачивает заданное целое число. При этом, если число отрицательное, то в результате реверсирования также должно получиться отрицательное число. Последние нули числа должны оставаться на своих местах при реверсировании. К примеру, число 1234567 реверсируется в следующее число 7654321, число -789 – в число -987, а число 125000 – в число 521000.

Задание D

- 1) Найти количество различных цифр у заданного натурального числа. К примеру, число 12345436 имеет шесть различных цифр, а число 121212 – только два.
- 2) Найти наибольшую цифру у заданного натурального числа. К примеру, в числе 18273645 максимальная цифра восемь, а в числе 777 – семь.
- 3) Проверить, является ли заданное натуральное число простым (*prime number*). Для справки: простые числа – это натуральные (целые положительные) числа, которые имеют ровно два различных натуральных делителя – единицу и самого себя (ресурс: https://en.wikipedia.org/wiki/Prime_number). К примеру, числа 2, 3, 5, 7, 11, 13, 17, 19 и т.д. являются простыми.
- 4) Задано число, содержащее от двух и более цифр. Между каждой парой соседних цифр заданного числа, вставить соответствующую цифру. Например, необходимо вставить цифру 7 в число 243 → 27473.
- 5) Задано натуральное число. Каждое вхождение наибольшей цифры, использованной в записи заданного числа, продублировать. Например, в числе 349291 максимальной цифрой является 9, следовательно, получим результирующее число → 34**99**2**99**1.

Задание E

- 1) Найти конкретное число Фибоначчи заданного его порядковым номером.
- 2) Найти все элементы последовательности чисел Фибоначчи до указанного пользователем порядкового номера.
- 3) Найти все элементы той части последовательности чисел Фибоначчи, значение последнего элемента которой не превосходит введённого пользователем значения.

- 4) Найти число трибоначчи по его порядковому номеру.

Числа трибоначчи – последовательность целых чисел $\{t_n\}$, заданного с помощью рекуррентного соотношения: $t_0 = 0, t_1 = 0, t_2 = 1, t_{n+3} = t_{n+2} + t_{n+1} + t_n$. Пример ряда трибоначчи: 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, ...

Условимся, что первые три числа ряда трибоначчи – это 0, 0 и 1, а если передаётся отрицательный индекс или ноль, то метод должен вернуть -1.

- 5) Найти простое число по его порядковому номеру. Для справки: простые числа – это натуральные (целые положительные) числа, которые имеют ровно два различных натуральных делителя – единицу и самого себя (ресурс: https://en.wikipedia.org/wiki/Prime_number). К примеру, последовательность простых чисел начинается с 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79 и т.д. Следовательно, если пользователь задаёт порядковый номер 10, то ему должно вернуться число 29 и т.д.
- 6) Найти все элементы последовательности простых чисел до указанного пользователем порядкового номера.
- 7) Найти все элементы той части последовательности простых чисел, значение последнего элемента которой не превосходит введённого пользователем значения.
- 8) Разработайте программу, которая проверяет, является ли данное число точной степенью двойки (или тройки и т.д.) или нет.
- 9) Найти все простые делители заданного натурального числа.
- 10) Найти НОД и НОК двух натуральных чисел.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Victor Ivanchenko









Графическое представление алгоритмов



Для общего представления решения задачи без привязки к конкретному языку программирования на практике используют блок-схемы. Они позволяют в графическом виде представить алгоритм решения задачи, который понятен не только разработчику, но даже домохозяйке.

Для графического представления алгоритмов решения задачи использую специальные унифицированные блоки, каждый из которых несёт в себе определённую смысловую нагрузку. Кратко каждый из наиболее востребованных блоков описывается в нижеприведённой таблице.

Таблица 1 – Наиболее часто используемые блоки

#	Shape (блок)	Description (описание)
1.		Блок начала/окончания выполнения программы
2.		Блок данных – используется для ввода, объявления и инициализации переменных программы
3.		Блок действия – используется для вычисления любых выражений программы
4.		Блок вызова процедур или функций – используется для обозначения вызова пользовательской функции или процедуры, код или реализация которой находится в другом файле
5.		Блок условия – задаёт соответствующие условия дальнейшего выбора хода выполнения кода программы
6.		Блок вывода данных – используется для обозначения выводимых данных или результата работы программы

Продолжение таблицы 1

7.		Блок соединитель на странице – используется в случае, если блок-схема алгоритма не может идти всё время сверху вниз и требуется её перенести на другую часть свободного места на том же листе
8.		Блок соединитель между страницами– используется в случае, если блок-схема алгоритма не помещается на одной странице и одну из её частей нужно перенести на другую страницу

Пример выполнения задания с использованием архитектурного шаблона проектирования MVC

Задание

Необходимо разработать наиболее эффективный алгоритм и написать программу для нахождения количества цифр заданного натурального числа, а также их сумму.

Решение

- 1) Перед написанием основного приложения разработаем алгоритм решения заданий. На рисунке 2 изображён алгоритм подсчёта количества цифр у заданного натурального числа, а на рисунке 3 – алгоритм подсчёта суммы цифр данного числа. Блок-схемы алгоритмов были построены в Microsoft Office Visio.
- 2) Далее согласно разработанным алгоритмам реализуем основную бизнес-логику приложения. Для чего опишем соответствующий класс **NumberWorker** (компонент *Model* согласно архитектурному шаблону проектирования MVC) и поместим в него два статических метода: **countQuantityOfNumberDigit** и **countSumOfNumberDigit**. Каждый статический метод будет принимать на вход по одному аргументу типа **long** – соответствующее натуральное число, и возвращать одно значение типа **int** – результат выполнения каждого метода:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab05.model.logic;
```

```
public class NumberWorker {
```

```
    public static final int DECIMAL_NOTATION_BASE = 10;
```

```
    public static int countQuantityOfNumberDigit(Long number) {
        int count = 0;
```

```
        while (number > 0) {
            count++;
            number /= DECIMAL_NOTATION_BASE;
        }
```

```
        return count;
    }
}
```



Обратите внимание, как **приятно читать** и **сопровождать** вышеописанный код. Рекомендуется следовать такому же стилю написания программного кода на Java

```

    public static int countSumOfNumberDigit(Long number) {
        int sum = 0;

        while (number > 0) {
            sum += number % DECIMAL_NOTATION_BASE;
            number /= DECIMAL_NOTATION_BASE;
        }

        return sum;
    }
}

```

3) Рассмотрим более подробно данный класс бизнес-логики:

Объявление соответствующего пакета для уникальности имени описываемого типа

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab05.model.Logic;
```

```
public class NumberWorker {
```

Объявление
класса

Объявление именованной константы
для повышения читабельности кода и
его дальнейшей лёгкой поддержки

```
    public static final int DECIMAL_NOTATION_BASE = 10;
```

Метод подсчёта количества цифр у
заданного натурального числа

Описание формального
параметра метода

```
    public static int countQuantityOfNumberDigit(Long number) {
```

Основная логика
метода

```
        int count = 0;
```

```
        while (number > 0) {
            count++;
            number /= DECIMAL_NOTATION_BASE;
        }
```

Локальная переменная
для хранения конечного
результата работы метода

Возврат
результата

```
        return count;
    }
```

Метод подсчёта суммы цифр заданного натурального числа

```
    public static int countSumOfNumberDigit(Long number) {
```

```
        int sum = 0;
```

```
        while (number > 0) {
            sum += number % DECIMAL_NOTATION_BASE;
            number /= DECIMAL_NOTATION_BASE;
        }
```

Описание формального
параметра метода

Основная логика метода

```
        return sum;
    }
```

Возврат результата

```
}
```

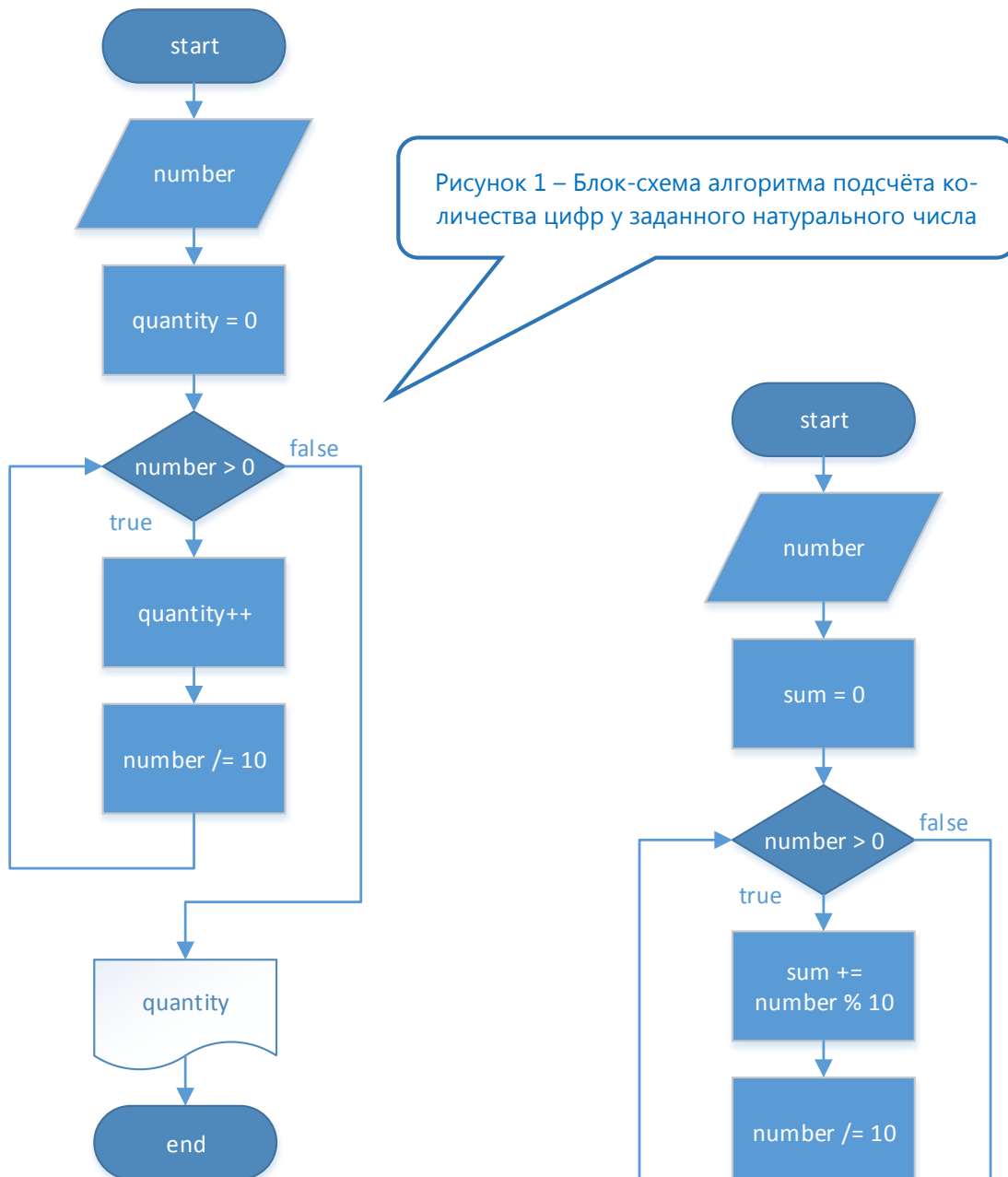
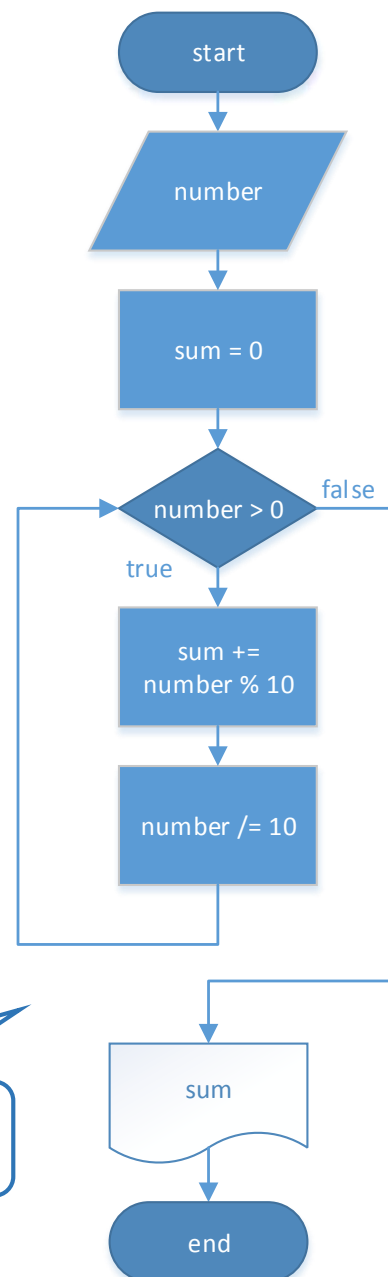


Рисунок 2 – Блок-схема алгоритма подсчёта суммы цифр заданного натурального числа



- 4) Далее реализуем дополнительный утилитный функциональный класс *UserInput* для пользовательского ввода данных, который для своей работы будет использовать метод **nextLong()** объекта утилитного класса **Scanner** для ввода значения типа **long**:

```
package by.bntu.fitr.poisit.vikvik.javababs.Lab05.util;
```

```
import java.util.Scanner;
```

Импортируем полное имя класса **Scanner** для того, чтобы в программе можно было к нему обращаться по простому имени

```
public class UserInput {
```

```
    private static Scanner scanner = new Scanner(System.in);
```

```
    public static long input(String msg) {
        System.out.print(msg);
        return scanner.nextLong();
    }
```

Создаём объект класса **Scanner** используя статическое поле класса

```
}
```

Ожидаем ввод пользователем целого значения и возвращаем его из метода

- 5) Один из последних классов, который необходимо реализовать, это класс для вывода результирующих данных на консоль (компонент *View* согласно архитектурному шаблону проектирования *MVC*):

```
package by.bntu.fitr.poisit.vikvik.javababs.Lab05.view;
```

```
public class Printer {
    public static void print(String msg) {
        System.out.print(msg);
    }
}
```

Вывод содержимого параметра **msg** на системную консоль без перехода на новую строку

- 6) На заключительном этапе соберём из разработанных компонентов (классов) готовую программу. Для этого опишем класс *Lab05*, который будет нести роль контроллера согласно архитектурному шаблону проектирования *MVC*. В нём будет помещён стартовый статический метод **main(...)**:

```
package by.bntu.fitr.poisit.vikvik.javababs.Lab05.controller;
```

```
import by.bntu.fitr.poisit.vikvik.javababs.Lab05.model.Logic.NumberWorker;
```

```
import by.bntu.fitr.poisit.vikvik.javababs.Lab05.util.UserInput;
```

```
import by.bntu.fitr.poisit.vikvik.javababs.Lab05.view.Printer;
```

Импортируем соответствующие полные имена классов для того, чтобы в программе можно было к ним обращаться по простому имени

```

public class Lab05 {

    public static void main(String[] args) {

        Printer.print("\nThe program calculate quantity and sum of number
        digits.\n");

        Long number = UserInput.input("\nInput number: ");

        int quantity = NumberWorker.countQuantityOfNumberDigit(number);
        int sum = NumberWorker.countSumOfNumberDigit(number);

        Printer.print("\nResult:");
        Printer.print("\n\t quantity = " + quantity);
        Printer.print("\n\t sum = " + sum);

    }
}

```

Вызываем метод бизнес-логики для вычисления корней

Просим пользователя ввести соответствующее натуральное число

Выводим результат

7) В общем виде архитектура приложения представлена ниже на рисунке 3:

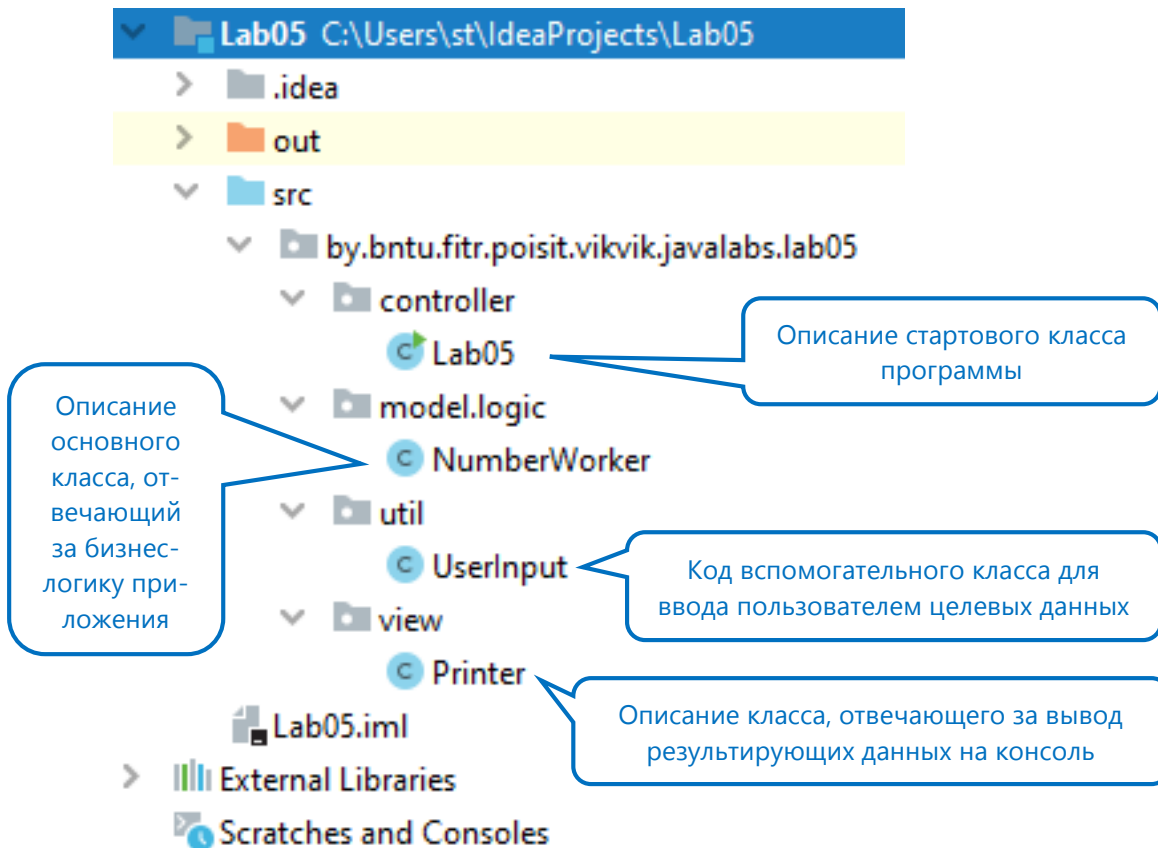


Рисунок 3 – Архитектура разработанной программы

Программа была написана и тестировалась с использованием среды разработки IntelliJ IDEA (Ultimate Edition).

Компиляция программы и тестирование всех возможных случаев (условий) выполнения с использованием инструментария JDK представлены на соответствующих рисунках 4, 5, 6 и 7, а их аналогичная компиляция и запуск в интегрированной среде разработки IntelliJ IDEA – на рисунках 8, 9 и 10.

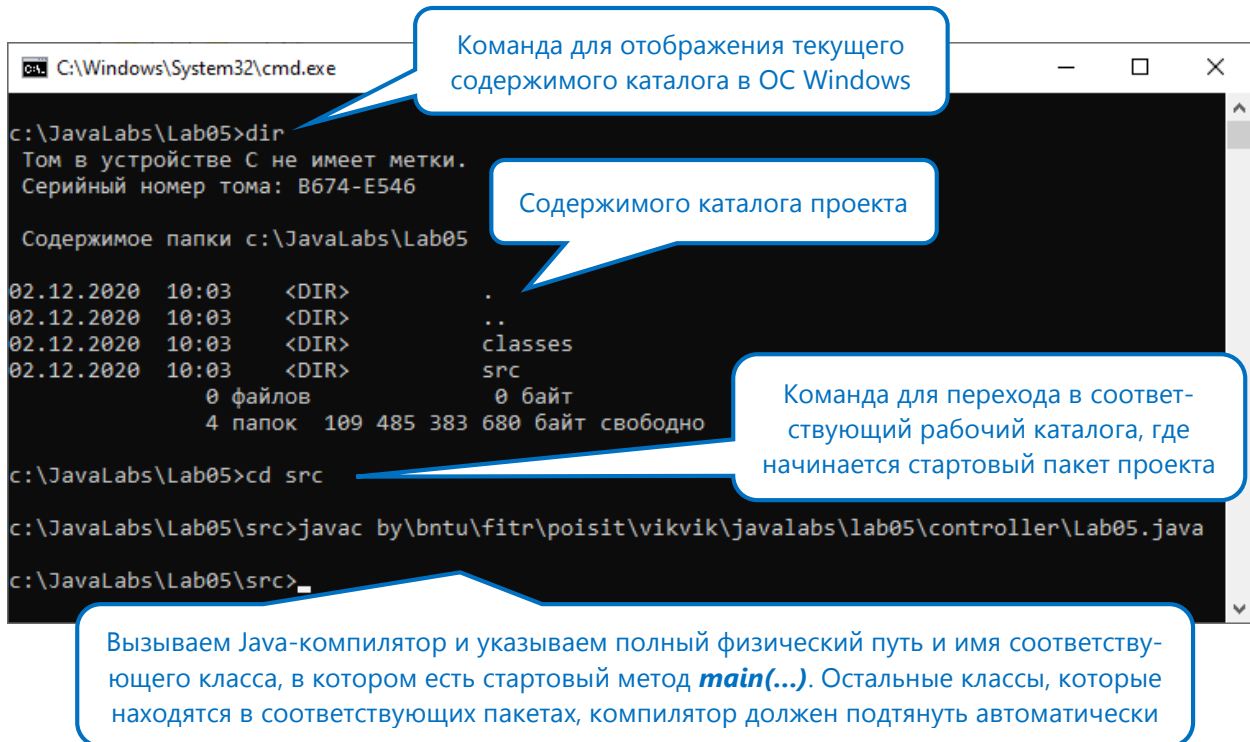


Рисунок 4 – Содержимое рабочего каталога приложения и его компиляция

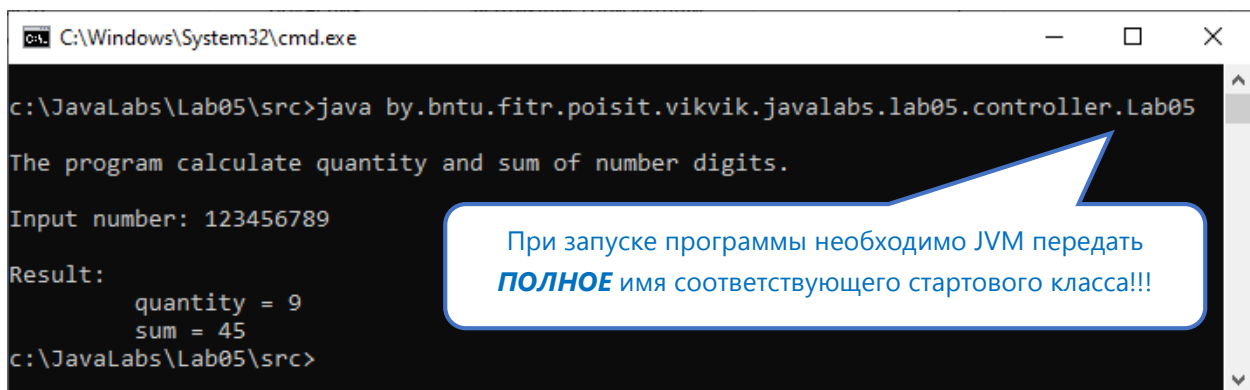
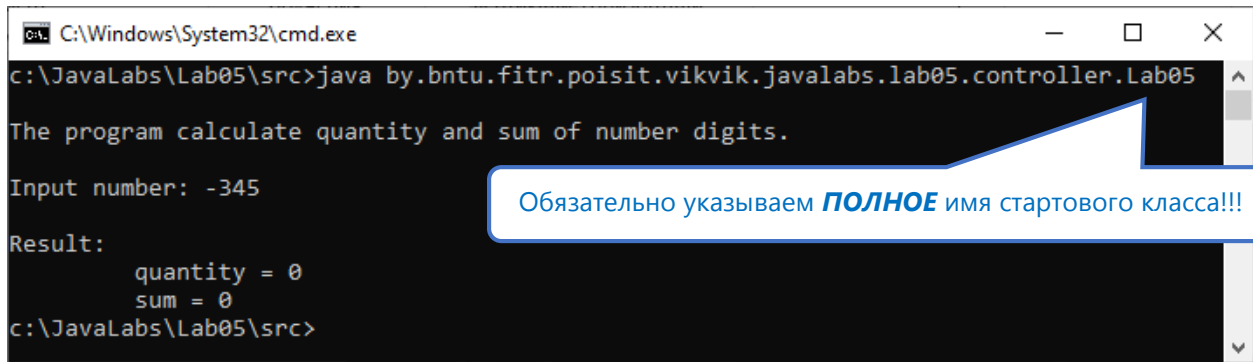


Рисунок 5 – Результат работы программы для подсчёта количества цифр в числе и суммы этих цифр (ввод пользователем верного значения)



```

C:\Windows\System32\cmd.exe
c:\JavaLabs\Lab05\src>java by.bntu.fitr.poisit.vikvik.javab05.controller.Lab05

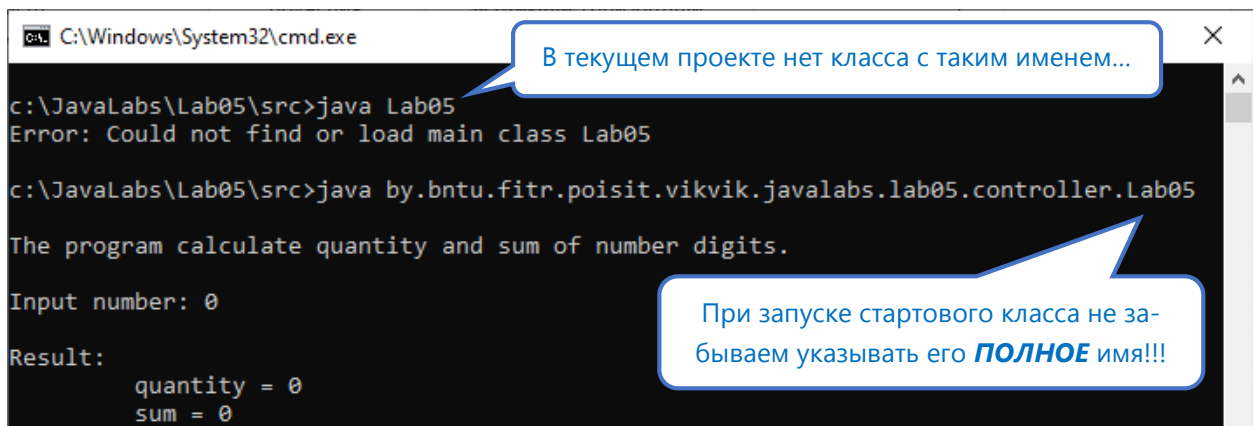
The program calculate quantity and sum of number digits.

Input number: -345

Result:
    quantity = 0
    sum = 0
c:\JavaLabs\Lab05\src>
  
```

Обязательно указываем **ПОЛНОЕ** имя стартового класса!!!

Рисунок 6 – Результат работы программы при вводе пользователем отрицательного числа (отрицательное число не является натуральным)



```

C:\Windows\System32\cmd.exe
c:\JavaLabs\Lab05\src>java Lab05
Error: Could not find or load main class Lab05

c:\JavaLabs\Lab05\src>java by.bntu.fitr.poisit.vikvik.javab05.controller.Lab05

The program calculate quantity and sum of number digits.

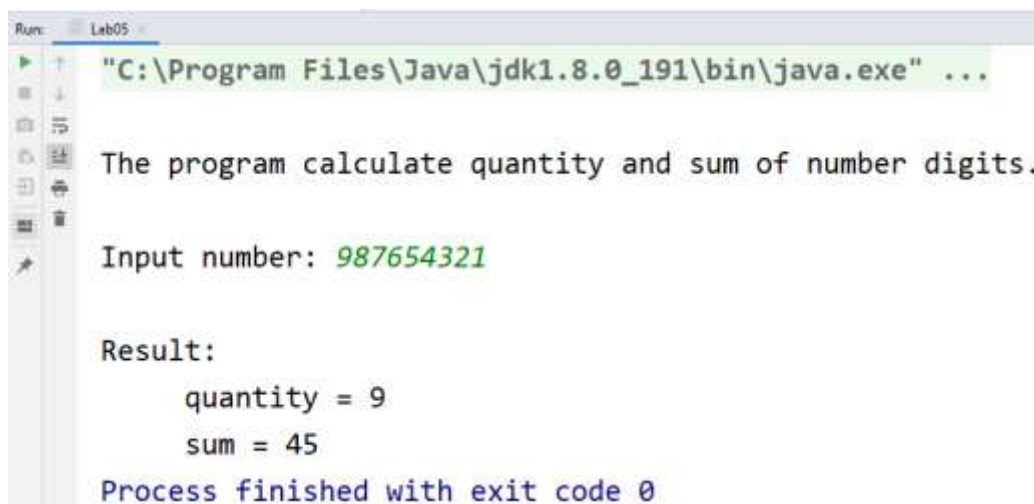
Input number: 0

Result:
    quantity = 0
    sum = 0
  
```

В текущем проекте нет класса с таким именем...

При запуске стартового класса не забываем указывать его **ПОЛНОЕ** имя!!!

Рисунок 7 – Результат работы программы при вводе пользователем значения ноль (ноль не является натуральным числом)



```

Run: Lab05
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...

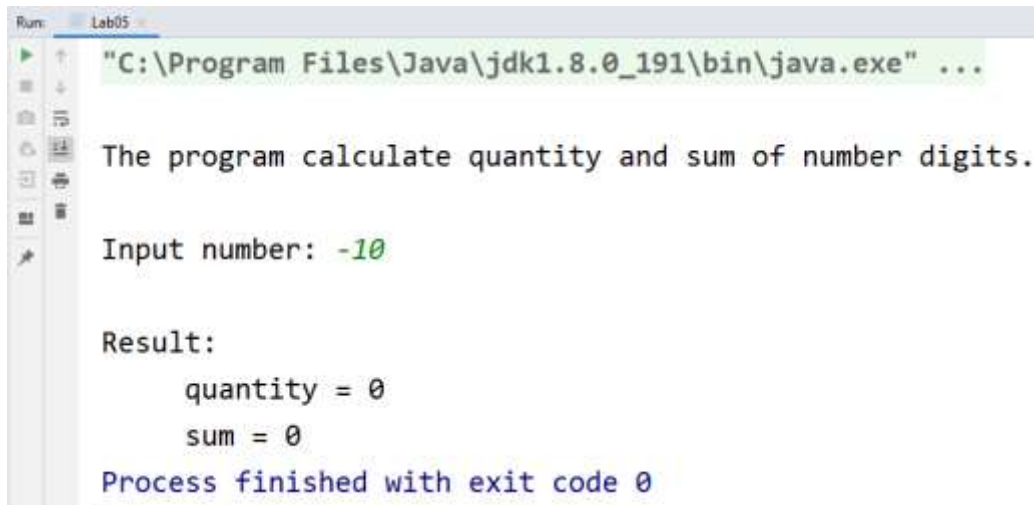
The program calculate quantity and sum of number digits.

Input number: 987654321

Result:
    quantity = 9
    sum = 45

Process finished with exit code 0
  
```

Рисунок 8 – Результат работы программы для подсчёта количества цифр в числе и суммы этих цифр (ввод пользователем верного значения)



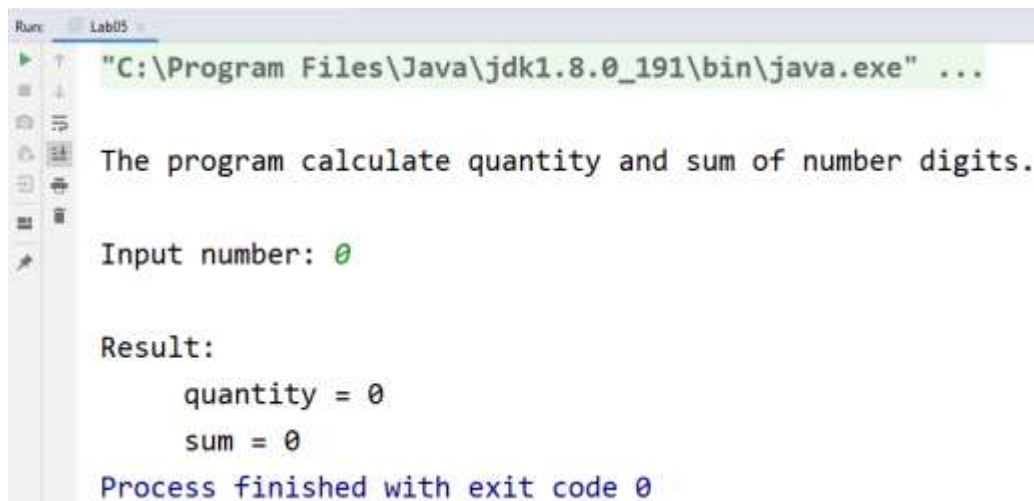
```
Run: Lab05
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...

The program calculate quantity and sum of number digits.

Input number: -10

Result:
    quantity = 0
    sum = 0
Process finished with exit code 0
```

Рисунок 9 – Результат работы программы при вводе пользователем отрицательного числа (отрицательное число не является натуральным)



```
Run: Lab05
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...

The program calculate quantity and sum of number digits.

Input number: 0

Result:
    quantity = 0
    sum = 0
Process finished with exit code 0
```

Рисунок 10 – Результат работы программы при вводе пользователем значения ноль (ноль не является натуральным числом)

Контрольные вопросы



- 1) Какие алгоритмы называются итерационными (итеративными) и где они применяются?
- 2) Что такое итерация?
- 3) Какие разновидности циклов существуют в мире программирования и когда они используются?
- 4) Какие в языке Java существуют разновидности циклических конструкций (сложных операторов циклов)?
- 5) Опишите синтаксис самой универсальной циклической конструкции с предусловием **while** и её обозначение в виде блок-схемы? Как она работает? Когда данная конструкция применяется?
- 6) Чем является выражение внутри скобок после ключевого слова **while** – инициализацией, условием или обновлением?
- 7) Опишите синтаксис циклической конструкции с постусловием **do-while** и её обозначение в виде блок-схемы? Как она работает? Когда данная конструкция применяется?
- 8) Опишите синтаксис циклической конструкции с предусловием (или со счётчиком) **for** и её обозначение в виде блок-схемы? Когда данная конструкция применяется?
- 9) Какие блоки в цикле **for** являются не обязательными? Если данные блоки пустые, то что подразумевается по умолчанию?
- 10) Опишите последовательность выполнения каждого блока цикла **for**, т.е. что за чем выполняется или не выполняется.
- 11) Обязательно ли инициализировать и(или) использовать переменную-счётчик цикла **for** внутри самого цикла?
- 12) Какая разновидность цикла **for** появилась в Java начиная с версии JDK 5.0? В чём предназначение данного цикла?
- 13) Опишите синтаксис расширенного цикла **for** (в простонародий его обычно называют *foreach*). Как он работает и где его обычно используют?
- 14) Какая из циклических конструкций в Java выполняется немного быстрее остальных?

- 15) Может ли условное выражение в цикле содержать сразу значение истины? Если да, то для чего это обычно может использоваться?
- 16) А значение других типов данных в качестве условия цикла можно использовать в Java?
- 17) Что такое бесконечный цикл (условно бесконечный цикл)?
- 18) Когда обычно применяется бесконечный цикл?
- 19) Приведите примеры реализации бесконечного цикла с использованием циклических конструкций, которые есть в Java.
- 20) Что такое недостижимый код в Java? Является ли он правильным с точки зрения синтаксиса языка Java, т.е. будет ли данный код компилироваться?
- 21) Отличительная особенность циклов Java с аналогичными циклами в других языках программирования (к примеру, в C/C++/C# и Python)?
- 22) Приведите примеры использования циклов для реализации следующих последовательностей чисел:
 - a. 1 2 3 4 5 6 7 8 9 10 ... ;
 - b. 10 9 8 7 6 5 4 3 2 1 ... ;
 - c. 10 12 14 16 18 20 ... ;
 - d. 95 85 75 65 55 45 35 25 15 5;
 - e. ... а что ещё можно?
- 23) Какова роль использования оператора **break** в теле цикла?
- 24) Какова роль использования оператора **continue** в теле цикла? Что про данные оператор говорит соглашение по разработке на Java? Как можно избавиться в коде от оператора **continue**?
- 25) Какова роль использования пустого оператора в теле цикла?
- 26) Что такое метка и для чего обычно на используется в языках программирования?
- 27) С каким оператором обычно используется метка в языке C/C++/C#?
- 28) В каких языках программирования злоупотребляют применения меток?
- 29) Для чего обычно в Java используют метки и с какими операторами?
- 30) Если вы собираетесь использовать вложенные циклы для вывода элементов матрицы в виде строк и столбцов, какой из циклов будет печатать строки: внутренний или внешний?

7 смертных грехов программирования



Рой Леман, разработчик ПО

1. Сначала написать, потом подумать. Вы получили требования к товару, пробежались по ним, запустили свою любимую IDE и принялись за работу. Легко, не правда ли? Стоп! Вы уверены, что поняли требования до конца? Я не сомневаюсь в вашем умении читать. **Но учили ли вы все пограничные случаи?** Продумали, как будете тестировать систему? **Набросали алгоритм, который собираетесь использовать?** Завтра вы этого и не вспомните!
2. Изобретать колесо. Итак, вам нужно создать шаблон проектирования *Producer-Consumer*. Вы знаете, как это сделать, еще с университетской скамьи... Легко, не правда ли? Стоп! Не важно, с каким языком вы работаете, уже **существуют готовые шаблоны**, или модули, или открытые исходники. Используйте их. Или по крайней мере изучите их перед тем, как создавать свои.
3. Бояться прикасаться к коду. Итак, у вас есть задание добавить несколько функций к 20 000-линейному файлу (О, нет! За что?) Вы радостно беретесь за работу и вдруг замечаете огрехи в исходных функциях – нет пограничного случая или проверки на нулевой показатель. Это находится за пределами сферы вашей ответственности. Так? Стоп! **Если вы видите небезопасный код – исправьте его.** Вы еще хлебнете на этих ошибках, даже если код написан не вами!
4. Быть безразличным к тому, чем занимается ваша компания. Вы программист (автоматизированный тестировщик и т.д.), верно? Написание кодов – это здорово, вы не изучали маркетинг или продажи, с чего вам интересоваться тем, что не имеет к вам отношения? А следовало бы! Как можно создать продукт, не понимая, чем занимается компания? Как сделать так, чтобы продукт

удовлетворял потребности клиента? Никак! **Изучите дело, будьте в курсе всех вопросов компании, а не только тех, которые касаются непосредственно вас.** Это важно! В какой-то момент это даже может повлиять на ваше повышение.

5. Не следить за новыми трендами. Вы занимаетесь программированием уже 10 лет и подыскиваете работенку. Перед этим вы работали старшим разработчиком C++ в крупной корпорации – за многое отвечали и имеете отличные рекомендации. Вы вроде знаете, что такое *DevOps*, но на практике никогда не сталкивались с этими практиками и с C++14? На вашем предыдущем месте работы в ходу был C++98... Не так уж важно, не так ли? Нет, не так! **Никто не похвалит вас за владение технологиями 15-летней давности! Если вы не учитесь в свободное время, чтобы соответствовать запросам работодателя, ваша кандидатура будет отвергнута!**
6. Не обладать коммуникативными навыками. Вы разработчик, к чему вам уметь общаться с людьми! Вам платят за умение общаться с компьютером, а не коллегами. Сиди себе, пиши качественные коды и добьешься успеха, верно? Не верно! Ваше неумение кратко и четко изложить суть дела вышестоящим – самая большая головная боль для менеджера. Очевидно, что это не единственный параметр, по которому вас оценивают, но все же – **грамотное предоставление информации в дружественной манере повысит доверие со стороны коллег** и вот тогда вы добьетесь успеха.
7. Не иметь целей. Вам нравится ваша работа, вы прекрасно владеете технологиями *Deep Learning*. Передовые технологии, прекрасные коллеги... Вы могли бы работать так вечно. Но – вы не будете. Все когда-нибудь заканчивается, иногда резко и неожиданно. Если у вас не будет карьерных целей, вы можете оказаться на задворках, выполняя работу и получая зарплату, которые вас не достойны. Так что **думайте наперед** – где бы вы хотели оказаться через 10 лет? В какой роли вы себя видите? Программистом? Функциональным тестировщиком или тестировщиком в автоматизации? Научным работником? Разработчиком? Менеджером по продукции? Вице-президентом? Техническим директором? Исполнительным директором? **Вам решать!**

Source: <https://www.kv.by/post/1053298-7-smertnyh-grehov-programmirovaniya>