



Laboratory Work #04

# Java Basic Syntax. Conditional Statements. Branching 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](mailto:ivanvikvik@bntu.by) / Minsk

# ЛАБОРАТОРНАЯ РАБОТА #04

## Базовый синтаксис языка Java.

### Условные конструкции.

### Разветвляющиеся алгоритмы

#### Цель работы

Изучить синтаксис условной конструкции и оператора множественного выбора языка Java для реализаций разветвляющихся алгоритмов (ветвлений) и закрепить их на примере разработки простейших интерактивных консольных Java-приложений.

#### Требования

- 1) Необходимо выполнить весь блок основных заданий, по одному соответствующему заданию – из индивидуальных заданий А и В. Дополнительное задание выполняется на Ваше усмотрение.
- 2) Для каждого вычислительного алгоритма необходимо спроектировать блок-схему решения, которую необходимо поместить в отчёт.
- 3) На базе спроектированных алгоритмов разработать простейшее консольное интерактивное приложение с использованием архитектурного шаблона проектирования **Model-View-Controller, MVC**.
- 4) Создаваемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: ***by.bntu.fitr.poisit.nameofstudent.java-labs.lab04***.
- 5) При выполнении задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.

- 6) 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.
- 7) Если логически не подразумевается или в задании иного не указано, то входными и выходными данными являются вещественные числа (числа с плавающей запятой).
- 8) Все задания необходимо решать используя только базовые операции (простые операторы), определённые над примитивными типами данных в языке программирования Java, и условные конструкции (т.е. не нужно использовать циклические конструкции, массивы, строковые данные и операции над ними и т.д.).
- 9) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».
- 10) 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**.
- 11) Программа должна обязательно быть снабжена комментариями, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчика, название бригады (если есть), номер группы и дату разработки. Исходный текст классов и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.
- 12) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем. Интерфейс программы и комментарии в коде должны быть на английском языке.
- 13) При проверки работоспособности приложения необходимо проверить все тестовые случаи.
- 14) При выполнении задания не рекомендуется использовать интегрированные средства разработки (*Integrated Development Environment, IDE*). Следует задействовать любой текстовый редактор и основные компоненты JDK (компилятор – **javac**, утилиту для запуска JVM – **java**).
- 15) При разработке программ придерживайтесь соглашений по написанию кода на Java (**Java Code-Convention**) !!!

## Основное задание

- 1) В молодом возрасте дракон каждый год отращивает по три головы, но после того, как ему исполнится 200 лет – только по две, а после 300 лет – лишь по одной. Разработайте программу, которая высчитывала бы, сколько голов и глаз у дракона, которому N лет. Считать, что при рождении у дракона имеется уже три головы.
- 2) Напишите программу «*The Greatest*», которая определяет какое из четырёх (или пяти, или шести и т.д.) введённых пользователем значений наибольшее (наименьшее). Предусмотреть возможность равенства всех значений.
- 3) Напишите программу, которая определяла бы, является ли заданное число кратным соответствующим числам 2, 3, 5, 7, 11, 13, 17 и 19.

## Индивидуальное задание А

- 1) Даны три стороны в виде вещественных чисел. Напишите программу, которая определяет, являются ли данные стороны сторонами треугольника.
- 2) Напишите программу, которая бы определяла, является ли введённая буква гласной или согласной. Постарайтесь сделать данное задание несколькими способами (чем больше, тем лучше). К примеру: с использованием базовых операций, конструкции ***if-else***, конструкции ***switch*** и т.д. Ограничения, которые указаны в требованиях, можно игнорировать для данного задания.
- 3) Напишите программу «*Mood Sensor*» (эмулировать датчика настроения), которая «залазит» в душу пользователя и определяет его настроение в текущий момент времени. Приложение будет генерировать случайное число, в зависимости от значения которого на экран выводится одно из псевдографических «лиц», которое и будет отображать настроение пользователя.
- 4) Напишите программу, которая бы эмулировала игру «*Dice*» (игра в кости). Суть игры заключается в броске двух шестигранных кубиков (костей) и подсчёта общей суммы очков, которые выпали на первой и второй костей.
- 5) Напишите программу – симулятор пирожков с «сюрпризом». Программа должна выводить пирожок и один из пяти (можно больше) различных «сюрпризов», который бы выбирался случайным образом.

## Индивидуальное задание В

- 1) Дано целое число в диапазоне от 1 до 7, соответствующее дню недели (1 - Monday, 2 - Thursday, 3 - Wednesday и т. д.). Напишите программу, которая согласно номеру выводит название соответствующего дня недели. Если целое число не лежит в указанном диапазоне то вывести соответствующее сообщение.
- 2) Дано целое число  $M$ , которое ассоциируется с десятибалльной оценкой знаний. Необходимо написать приложение, которое выводит строку-описание оценки, соответствующей числу  $M$  (0-1 - «very bad» («очень плохо»), 2-3 - «unsatisfactory» or «poor» («неудовлетворительно»), 4 - «satisfactory» («удовлетворительно») и т.д. Если  $M$  не лежит в указанном диапазоне то вывести соответствующее сообщение.
- 3) Мастям игральных карт присвоены порядковые номера: 1 - пики, 2 - трефы, 3 - бубны, 4 - червы. Достоинству карт, старших десятки, присвоены номера: 11 - валет, 12 - дама, 13 - король, 14 - туз. Даны два целых числа: первое - достоинство карты (6 ... 14) и второе - масть карты (1 ... 4). Написать программу, которая выводит название соответствующей карты вида «шестерка бубен», «дама червей», «туз треф» и т. п.
- 4) Дано целое число в диапазоне от 1 до 120, определяющее возраст (в годах). Написать программу, которая выводит строку-описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: 20 - «двадцать лет», 32 - «тридцать два года», 41 - «сорок один год» и т.д.
- 5) Даны два неотрицательных целых числа: годы и месяц (целое число в диапазоне от 1 до 12, где 1 - January, 2 - February, 3 - March и т. д.). Напишите программу, которая определяет количество дней в этом месяце для невисокосного года. Если числа не лежат в указанном диапазоне то вывести соответствующее сообщение.
- 6) Дан номер месяца - целое число в диапазоне от 1 до 12 (1 - January, 2 - February, 3 - March и т. д.). Напишите программу, которая согласно числу месяца выводит название соответствующего времени года («winter», «spring», «summer», «autumn»). Если целое число не лежит в указанном диапазоне то вывести соответствующее сообщение.

- 7) Даны следующие арифметические действия: сложение, вычитание, умножение, деление и остаток от деления. Напишите программу, которая от пользователя принимает операцию и два вещественных числа, а затем выполняет указанное действие над этими числами и выводит результат. Если значение чисел не удовлетворяет указанной операции (к примеру, деление на ноль), то программа должна вывести соответствующее сообщение.
- 8) Даны два целых числа, соответствующие дню и месяцу и определяющие правильную дату. Написать программу, которая выводит знак Зодиака, соответствующий этой дате: «Водолей» (20.1-18.2), «Рыбы» (19.2-20.3), «Овен» (21.3-19.4), «Телец» (20.4-20.5), «Близнецы» (21.5-21.6), «Рак» (22.6-22.7), «Лев» (23.7-22.8), «Дева» (23.8-22.9), «Весы» (23.9-22.10), «Скорпион» (23.10-22.11), «Стрелец» (23.11-21.12) и «Козерог» (22.12-19.1).
- 9) Дано натуральное число  $N$  ( $0 \leq N \leq 999$ ). Напишите интерактивную программу, которая организует диалог с пользователем и записывает число английскими (русскими) словами. Ниже приведён рекомендуемый вид экрана во время работы программы. Ввод пользователя выделен полужирным шрифтом. Если  $N$  не лежит в указанном диапазоне то вывести соответствующее сообщение.

\*\*\* The name of the number as a string \*\*\*

Input number: **1** (N)

The number as a string: one

\*\*\* The name of the number as a string \*\*\*

Input number: **12**

The number as a string: twelve

\*\*\* The name of the number as a string \*\*\*

Input number: **62**

The number as a string: sixty-two

\*\*\* Название числа в виде строки \*\*\*

Введите число: **1** (N)

Число в виде строки: один

\*\*\* Название числа в виде строки \*\*\*

Введите число: **12**

Число в виде строки: двенадцать

\*\*\* Название числа в виде строки \*\*\*

Введите число: **62**

Число в виде строки: шестьдесят два

\*\*\* Название числа в виде строки \*\*\*

Введите число: **289**

Число в виде строки: двести восемьдесят девять

\*\*\* Название числа в виде строки \*\*\*

Введите число: **473**

Число в виде строки: четыреста семьдесят три

\*\*\* Название числа в виде строки \*\*\*

Введите число: **995**

Число в виде строки: девятьсот девяносто пять

## Дополнительное задание

Заданы три целых числа, которые задают некоторую дату по Григорианскому календарю ([https://ru.wikipedia.org/wiki/Григорианский\\_календарь](https://ru.wikipedia.org/wiki/Григорианский_календарь)). Определить дату следующего дня. Запрещается использовать типы стандартной библиотеки языка для работы с датой и временем (можно сделать второй вариант решения задачи с использованием новой библиотеки работы со временем в Java, которая появилась с JDK 8.0). Также необходимо учесть то, что по Григорианскому календарю (используется в настоящем момент) високосный год определяется следующим образом:

- годы, кратные 4 – високосные (например, 2008, 2012, 2016);
- годы, кратные 4 и 100 – невисокосные (например, 1700, 1800, 1900);
- годы, кратные 4, 100 и 400 – високосные (например, 1600, 2000, 2400).

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

*Victor Ivanchenko*



## Что нужно запомнить (краткие тезисы)

1. В Java (как и во многих C-подобных языках программирования: C/C++/C# JavaScript, ...) имеются две конструкции одной из основных алгоритмических структур организации кода – ветвления. Это условный оператор *if-else* и оператор множественного выбора *switch-case-default*.
2. Отличие операций (простых операторов) от конструкций (сложных операторов) заключается в том, что операции возвращают результат своего выполнения, а конструкции используются для управления ходом выполнения программы. Следовательно, условные конструкции ***if-else*** и ***switch-case-default*** являются управляющими конструкциями и никогда не могут стоять справа от оператора присваивания, т.к. они не возвращают результат.
3. Классическая условная конструкция ***if-else*** покрывает в программировании абсолютно все ветвления (условные алгоритмы) и является одной из самых распространённых конструкций в различных языках программирования.
4. Если необходимо покрыть только одно условие, то применяется сокращённая версия условной конструкции, где используется только ключевое слово ***if***.
5. Если необходимо покрыть более двух условий, то применяются вложенные конструкции ***if-else*** друг в друга.
6. Для описания кода с вложенными условными конструкциями существует два стиля форматирования.
7. Согласно синтаксису языка Java в условном операторе после ключевых слов ***if*** и ***else*** допускается применение только одного оператора или выражения. Если при выполнении условия требуется выполнять несколько операторов (выражений), то используется ***составной оператор – { }*** («фигурные скобки»).
8. Оператор множественного выбора ***switch-case-default*** является «синтаксическим сахаром» в языке Java и используется только тогда, когда необходимо сравнивать на строгое равенство значение соответствующей переменной с набором константных значений.
9. Обычно оператор множественного выбора ***switch-case-default*** используется для компактности и читабельности соответствующего кода.
10. В качестве типа переменной-селектора в операторе множественного выбора ***switch-case-default*** могут выступать следующие типы: *byte*, *short*, *char*, *int*, *Byte*, *Short*, *Character*, *Integer*, *enum* (с версии JDK 5.0) и тип *String* (с версии JDK 7.0).









## 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

В качестве примера решим самое любимое занятие школьников всех времён и народов – квадратное уравнение.

### Задание

Разработать интерактивную программу «*Quadric Equation*» («Квадратное уравнение») для решения квадратных уравнений вида:  $ax^2 + bx + c = 0$ . Программа должна запрашивать у пользователя соответствующие параметры  $a$ ,  $b$  и  $c$ , в зависимости от вычисленного дискриминанта  $D$ , выдавать соответствующий результат. В случае отрицательного дискриминанта программа должна выводить сообщение о том, что действительных корней нет.

### Решение

1) Перед написанием основного приложения разработаем алгоритм решения. Будем полагать, что коэффициенты уравнения  $a$ ,  $b$  и  $c$  представляют собой вещественные числа. Простейший случай предполагает, что все коэффициенты отличны от нуля.

Вербальное описание алгоритма (словесная последовательность действий) состоит из следующих шагов:

- a) передаём соответствующие коэффициенты  $a$ ,  $b$  и  $c$ ;
- b) проверяем правильность данных, в частности, проверяем коэффициент  $a$ , который не должен быть равен нулю, иначе, во-первых, у нас получится не квадратное уравнение и, во-вторых, может произойти деление на ноль. Примем, что если коэффициент  $a$  равен нулю, то осуществляется вывод соответствующего сообщения и переход к пункту g;
- c) вычисляем дискриминант;
- d) если дискриминант больше нуля, то вычисляем два корня, возвращаем их, а затем переходим к пункту g;
- e) если дискриминант равен нулю, то вычисляем один корень, возвращаем его, а затем переходим к пункту g;

- f) если дискриминант меньше нуля, то возвращаем сообщение о том, что действительных корней нет;
- g) завершаем выполнение алгоритма.

Блок-схема работы алгоритма решения задачи была построена в Microsoft Office Visio 2013 и приведена ниже на рисунке 1.

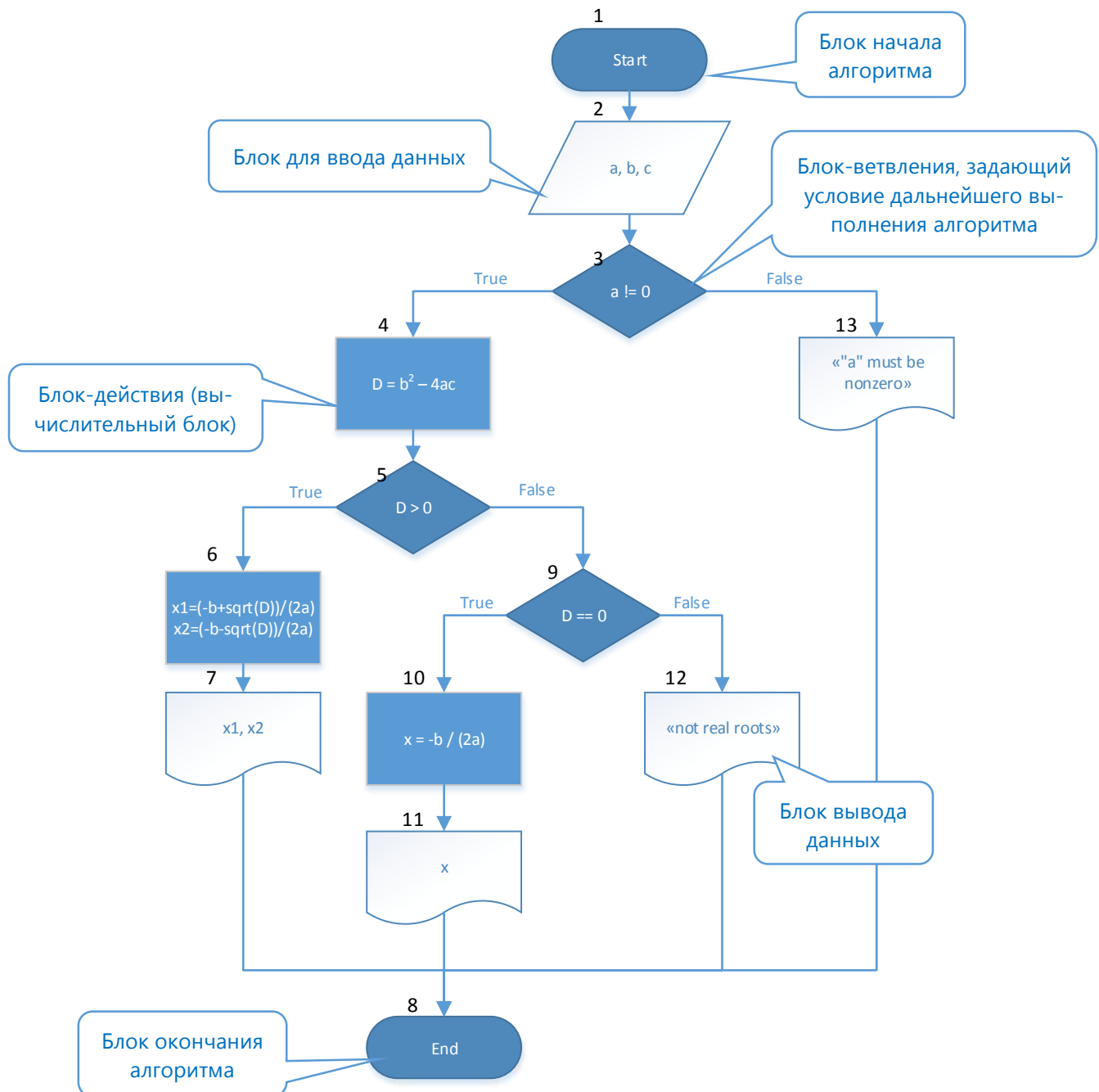


Рисунок 2 – Блок-схема алгоритма решения *квадратного уравнения*

2) Теперь реализуем основную бизнес-логику приложения согласно разработанному алгоритму в соответствующем статическом методе *solve(...)* функционального класса *QuadraticEquation* (компонент *Model* согласно архитектурному шаблону проектирования MVC). Статический метод будет принимать на вход три параметра типа *double*: соответствующие коэффициенты *a*, *b* и *c*, и возвращать одно значение типа *String* – результат нахождения корней квадратного уравнения:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab04.model.logic;
```

```
public class QuadraticEquation {
```

Получаем соответствующие коэффициенты

```
    public static String solve(double a, double b, double c) {
```

```
        String result = "Error: coefficient 'a' must be nonzero!";
```

```
        if (a != 0) {
```

Вычисляем дискриминант

```
            double D = b * b - 4 * a * c;
```

```
            if (D > 0) {
```

Вычисляем корни уравнения

```
                double x1 = (-b + Math.sqrt(D)) / (2 * a);
```

```
                double x2 = (-b - Math.sqrt(D)) / (2 * a);
```

```
                result = "There are two roots of equation: " + "x1 = " + x1 + "; x2 = " + x2;
```

```
            } else if (D == 0) {
```

Вычисляем корень уравнения

```
                double x = -b / 2 / a;
```

```
                result = "There is only one root of equation: " + "x = " + x;
```

```
            } else {
```

```
                result = "There aren't real roots of equation";
```

```
            }
```

```
        }
```

Возвращаем результат

```
        return result;
```

```
    }
```

```
}
```

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



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

```
package by.bntu.fitr.poisit.vikvik.javalabs.Lab04.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 double input(String msg) {
        System.out.print(msg);
        return scanner.nextDouble();
    }
```

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

Ожидаем ввод пользователем целого значения и возвращаем его из метода

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

```
package by.bntu.fitr.poisit.vikvik.javalabs.Lab04.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

- 5) На заключительном этапе соберём из разработанных компонентов (классов) готовую программу. Для этого опишем класс *Lab04*, который будет нести роль контроллера согласно архитектурному шаблону проектирования MVC. В нём будет помещён стартовый статический метод **main(...)**:

```
package by.bntu.fitr.poisit.vikvik.javalabs.Lab04.controller;
```

```
import by.bntu.fitr.poisit.vikvik.javalabs.Lab04.model.Logic.QuadraticEquation;
import by.bntu.fitr.poisit.vikvik.javalabs.Lab04.util.UserInput;
import by.bntu.fitr.poisit.vikvik.javalabs.Lab04.view.Printer;
```

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

```

public class Lab04 {

    public static void main(String[] args) {
        Printer.print("\nThe program solves quadratic equation:  $ax^2+bx+c = 0$ ");

        double a = UserInput.input("\nInput a: ");
        double b = UserInput.input("\nInput b: ");
        double c = UserInput.input("\nInput c: ");

        String result = QuadraticEquation.solve(a, b, c);

        Printer.print("\n" + result);
    }
}

```

Просим пользователя ввести соответствующую коэффициенты уравнения

Вызываем метод бизнес-логики для вычисления корней

Выводим результат

6) В общем виде архитектура приложения представлена ниже на рисунке 2:

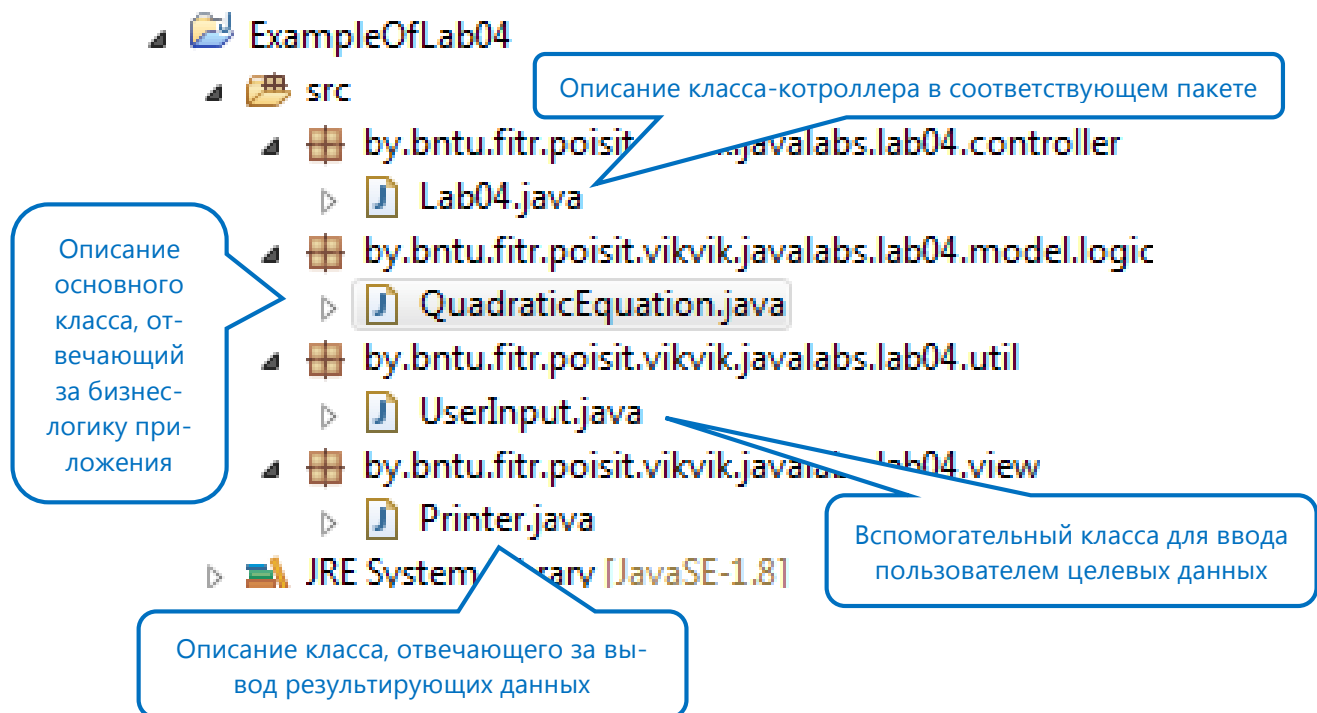


Рисунок 2 – Архитектура разработанной программы

Программа была написана и тестировалась с использованием среды разработки Eclipse Java EE IDE for Web Developers (version Neon.1.a). (Кстати, чтобы просмотреть список горячих клавиш в Eclipse необходимо нажать комбинацию клавиш Ctrl+Shift+L).

Компиляция программы и тестирование всех возможных случаев (условий) выполнения с использованием инструментария JDK представлены на соответствующих рисунках 3, 4, 5, 6 и 7, а их аналогичная компиляция и запуск в интегрированной среде разработки Eclipse IDE – на рисунках 8, 9, 10 и 11.

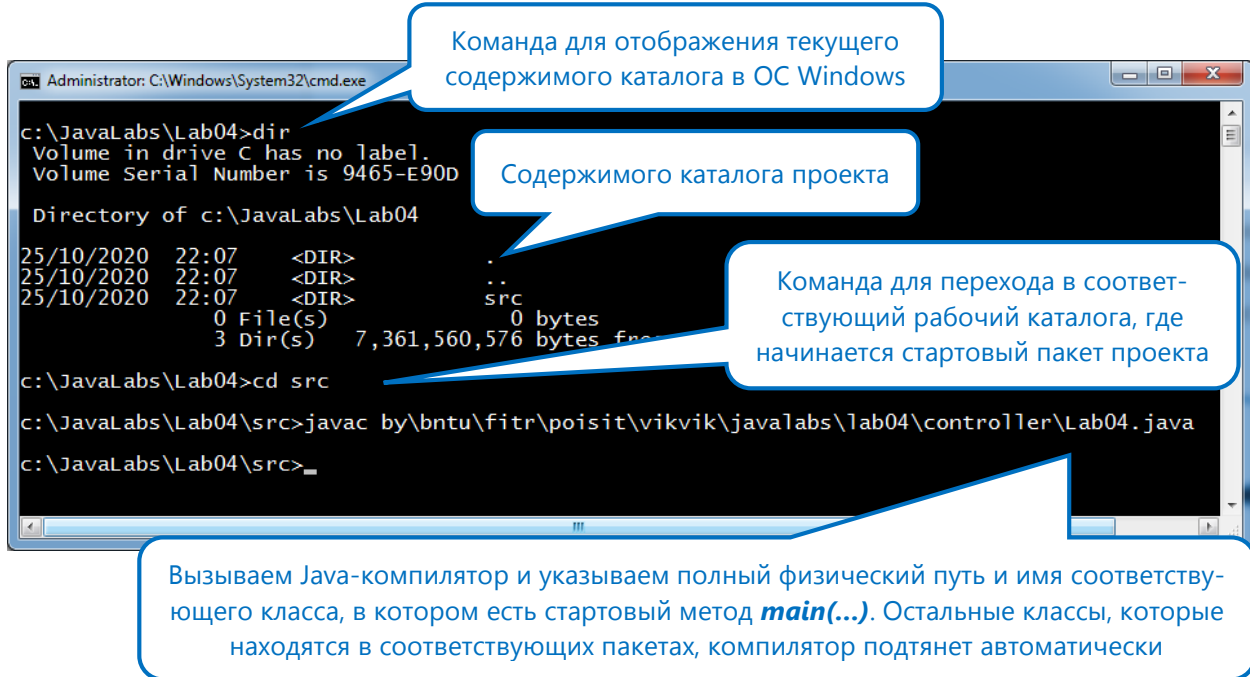


Рисунок 3 – Содержимое рабочего каталога приложения и его компиляция

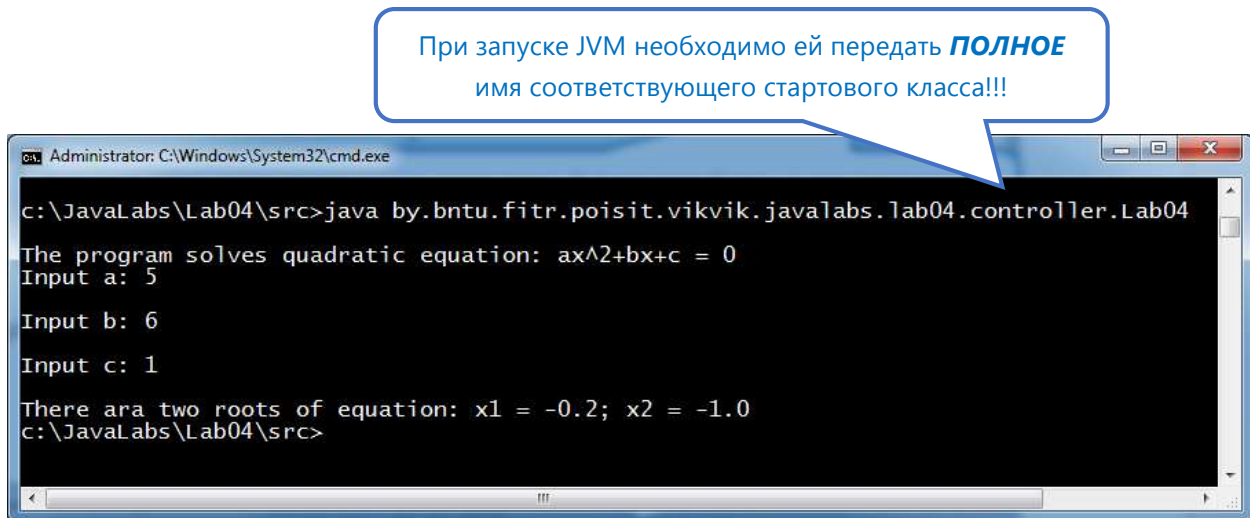


Рисунок 4 – Результат работы программы для решения *квадратного уравнения* при положительном дискриминанте



```
Administrator: C:\Windows\System32\cmd.exe

c:\JavaLabs\Lab04\src>java by.bntu.fitr.poisit.vikvik.javaabs.lab04.controller.Lab04

The program solves quadratic equation:  $ax^2+bx+c = 0$ 
Input a: 1
Input b: 2
Input c: 1
There is only one root of equation:  $x = -1.0$ 
c:\JavaLabs\Lab04\src>
```

Указываем **ПОЛНОЕ** имя стартового класса!!!

Рисунок 5 – Результат работы программы для решения *квадратного уравнения* при дискриминанте равном нулю

```
Administrator: C:\Windows\System32\cmd.exe

c:\JavaLabs\Lab04\src>java by.bntu.fitr.poisit.vikvik.javaabs.lab04.controller.Lab04

The program solves quadratic equation:  $ax^2+bx+c = 0$ 
Input a: 1
Input b: 2
Input c: 3
There aren't real roots of equation
c:\JavaLabs\Lab04\src>
```

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

Рисунок 6 – Результат работы программы для решения *квадратного уравнения* при отрицательном дискриминанте

```
Administrator: C:\Windows\System32\cmd.exe

c:\JavaLabs\Lab04\src>java Lab04
Error: Could not find or load main class Lab04

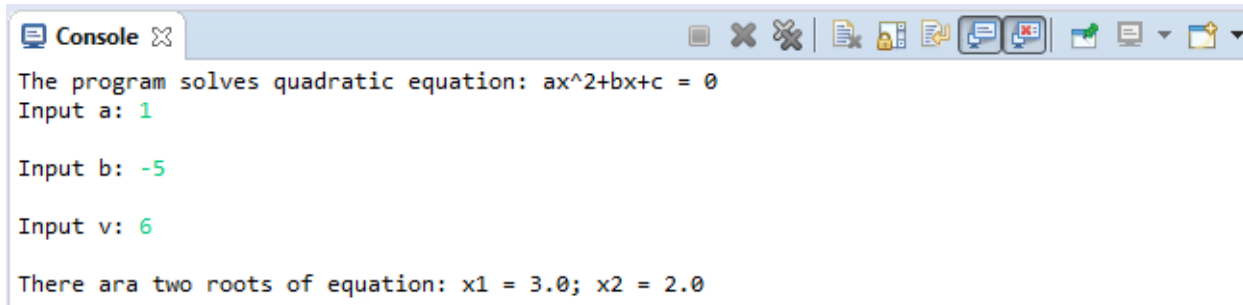
c:\JavaLabs\Lab04\src>java by.bntu.fitr.poisit.vikvik.javaabs.lab04.controller.Lab04

The program solves quadratic equation:  $ax^2+bx+c = 0$ 
Input a: 0
Input b: 1
Input c: 2
Error: coefficient 'a' must be nonzero!
c:\JavaLabs\Lab04\src>
```

В текущем проекте нет класса с таким именем...

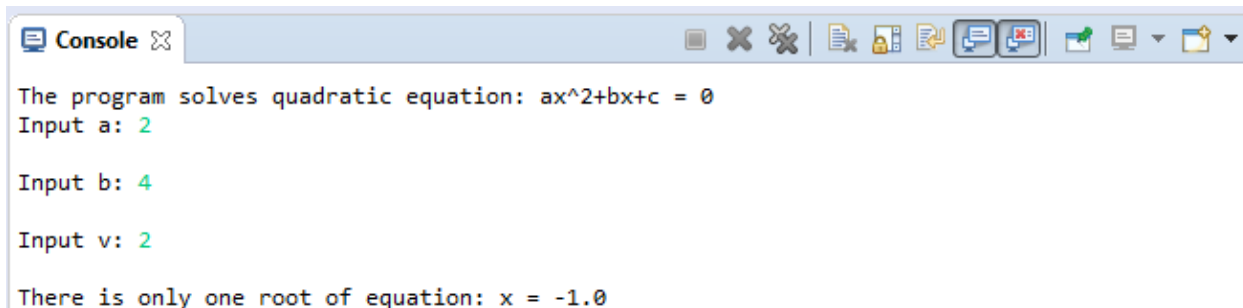
При запуске стартового класса не забываем указывать его **ПОЛНОЕ** имя!!!

Рисунок 7 – Результат работы программы для решения *квадратного уравнения* в случае, если пользователь ввёл в качестве значения коэффициента *a* ноль



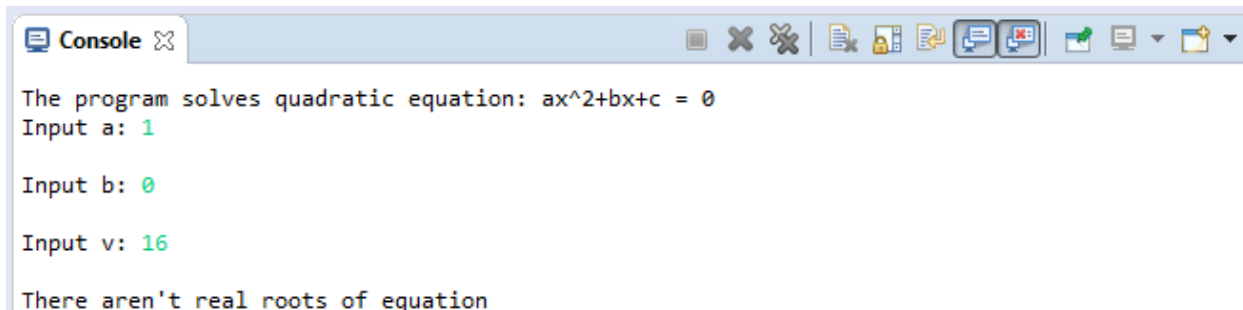
```
Console
The program solves quadratic equation: ax^2+bx+c = 0
Input a: 1
Input b: -5
Input c: 6
There are two roots of equation: x1 = 3.0; x2 = 2.0
```

Рисунок 8 – Результат работы программы для решения *квадратного уравнения* при положительном дискриминанте



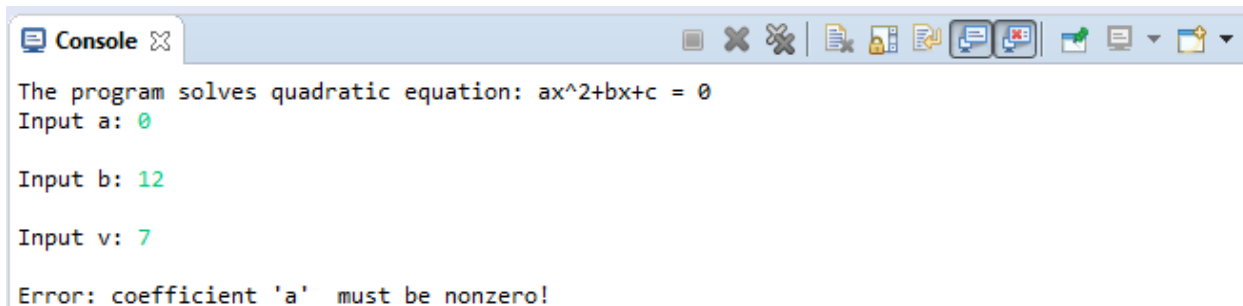
```
Console
The program solves quadratic equation: ax^2+bx+c = 0
Input a: 2
Input b: 4
Input c: 2
There is only one root of equation: x = -1.0
```

Рисунок 9 – Результат работы программы для решения *квадратного уравнения* при дискриминанте равном нулю



```
Console
The program solves quadratic equation: ax^2+bx+c = 0
Input a: 1
Input b: 0
Input c: 16
There aren't real roots of equation
```

Рисунок 10 – Результат работы программы для решения *квадратного уравнения* при отрицательном дискриминанте



```
Console
The program solves quadratic equation: ax^2+bx+c = 0
Input a: 0
Input b: 12
Input c: 7
Error: coefficient 'a' must be nonzero!
```

Рисунок 11 – Результат работы программы для решения *квадратного уравнения* в случае, если пользователь ввёл в качестве значения коэффициента *a* ноль

## Контрольные вопросы



- 1) Для чего в языках программирования нужны управляющие конструкции (операторы)? Чем они управляют?
- 2) Какие существуют в языках программирования фундаментальные типы управляющих конструкций, которые используются при построении любой компьютерной программы? Какие алгоритмы они покрывают?
- 3) Почему необходимо в начале проектировать систему (к примеру, разрабатывать алгоритмы решения задачи), а затем её реализовывать (преступить к непосредственному написанию кода программы)?
- 4) Опишите основные (базовые) элементы блок-схемы для графического представления алгоритмов решения задач?
- 5) Какие алгоритмы называются разветвляющимися и где они применяются?
- 6) Какие в языке Java существуют разновидности условных конструкций?
- 7) Как в Java реализуется механизм истинности-ложности?
- 8) Могут ли различные значения (объекты) выступать в качестве условия (или условного выражения) истинности-ложности в Java? Какое при этом используется правило?
- 9) Синтаксис универсальной условной конструкции *if-else* и её обозначение в виде блок-схемы? Как она работает? Когда данная конструкция применяется?
- 10) Какой логике обычно следуют разработчики при размещении соответствующих инструкций (последовательности операторов) для выполнения после ключевых слов *if* и *else*?
- 11) Какую разновидность условной конструкции *if-else* необходимо использовать, если необходимо выполнить только одно действие при определённом условии (истинности или ложности условия)? Как выглядит она с помощью блок-схемы? Какое используется условие, чтобы данная конструкция была в коде более читабельна?
- 12) Какой в Java используется подход для реализации выполнения кода, в котором задаётся больше двух условий выполнения? Как выглядит данный подход с помощью блок-схемы?
- 13) Какие есть способы форматирования вышеописанного подхода? Какой из этих способов лучше использовать согласно соглашению?

- 14) Каков синтаксис оператора множественного выбора *switch-case-default* и его обозначение в виде блок-схемы? Как он работает?
- 15) Как происходит каскадное выполнение *case*-ов в операторе множественного выбора *switch*?
- 16) Зачем нужна группировка *case*-ов в условном операторе множественного выбора?
- 17) Для чего нужно в условном операторе множественного выбора ключевое слово *default* и особенность его использования?
- 18) Какие типы данных можно использовать для переменной в конструкции *switch-case-default*? Что поменялось в синтаксисе конструкции с выходом различных версий JDK?
- 19) Чем условная конструкция множественного выбора *switch-case-default* будет отличаться от универсальной классической условной конструкции *if-else*? Когда они применяются в коде?
- 20) В каких случаях употребление условной (или «тернарной») операции будет лучше, чем условной конструкции *if-else*, а в каком случае наоборот?

## 7 смертных грехов программирования



**Джон Парселл (John Purcell), создатель онлайн курсов по Java и др. языкам программирования и технологиям ([www.caveofprogramming.com](http://www.caveofprogramming.com))**

1. Использовать «Пробел» вместо «Tab». Всегда, всегда используйте **«Tab»**, а не «Пробел».
2. Использовать «Tab» вместо «Пробела». Всегда, всегда используйте **«Пробел»**, а не «Tab».
3. Не использовать автоформатирование. Забудьте про весь мусор вроде табов и пробелов, **используйте автоформатирование** в своем коде и людям не придется видеть ваши странные скобки и отступы.
4. Использовать интегрированную среду разработки (IDE) с ее автоформатированием и цветными клавишами. Все коды должны быть написаны в vi или Emacs, что подтверждает безупречность ваших навыков программирования.
5. Не использовать IDE. Никто не хочет платить за время, которое вы тратите на набор текста, если это можно сделать в один клик, или за прокручивание вверх-вниз с помощью заумной комбинации клавиш из LISP.
6. Не учить C и C++. Два этих языка жизненно необходимы любому программисту. Думаете, Java так же хорош? Отлично, создайте мне систему управления гоночными автомобилями в режиме реального времени на Java, и я вам поверю.

7. Учить С и С++ в то время, которое вы могли бы использовать на что-то более современное, например, на Java. Признайте – все таблицы, написанные на С или С++, изживают себя в течение 5 лет. И в таком случае в программном обеспечении есть серьезные ошибки, которые Java просто не позволил бы вам совершить.

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