Laboratory Work #06

Java One-Dimensional Arrays

LEARN. GROW. SUCCEED.

Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #06
Одномерные массивы в Java

Цель работы 

Научиться работать с одномерными массивами в Java и закрепить приобретённые навыки на примере разработки интерактивных консольных Java-приложений.

Требования 

1) Необходимо спроектировать и реализовать алгоритмы решения заданий и UML-диаграмму взаимодействия классов и объектов разрабатываемой программной системы с отображением всех связей (отношений) между классами и объектами.
2) Каждый студент выбирает по одному уникальному индивидуальному заданию.
3) Основные классы системы должны быть самодостаточными, т.е. не зависеть, к примеру, от консоли! Любые типы отношений между классами должны применяться обосновано и лишь тогда, когда это имеет смысл.
4) На базе спроектированной программной системы реализуйте простейшее интерактивное консольное приложение. Используйте при реализации архитектурный шаблона проектирования ***Model-View-Controller***, ***MVC***.
5) Создаваемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: ***by.bntu.fitr.nameofstudent.javalabs.lab06***.
6) Попытайтесь реализовать все средства инициализации при создании соответствующих бизнес-объектов программной системы.
7) При выполнения задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.
8) Если логически не подразумевается или в задании иного не указано, то входными и выходными данными являются вещественные числа (числа с плавающей запятой).

9) 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.
10) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».
11) Для задания размерности массива и конкретных его элементов необходимо использовать несколько способов (пользовательский ввод во время выполнения программы; генератор случайных чисел; ...).
12) Для генерирования случайных чисел воспользуйтесь методами объекта класса ***java.util.Random***, а для ввода пользовательских значений – соответствующими методами объекта класса ***java.util.Scanner***.
13) 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.
14) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем.
15) Интерфейс программы и комментарии должны быть на английском языке.
16) При проверки работоспособности приложения необходимо проверить все тестовые случаи.
17) При разработке программ придерживайтесь соглашений по написанию кода на *Java* (***Java Code-Convention***) !!!

Индивидуальное задание

1) В векторе, состоящем из n вещественных элементов, вычислить: сумму отрицательных элементов вектора и произведение элементов вектора, расположенных между максимальным и минимальным элементами.
2) В векторе, состоящем из n целых элементов, вычислить: произведение элементов вектора с четными номерами и сумму элементов вектора, расположенных между первым и последним нулевыми элементами.
3) В векторе, состоящем из n вещественных элементов, вычислить: максимальный элемент вектора и сумму элементов вектора, расположенных до последнего положительного элемента.
4) В векторе, состоящем из n целых элементов, вычислить: номер максимального элемента вектора и произведение элементов вектора, расположенных между первым и вторым нулевыми элементами.
5) В векторе, состоящем из n вещественных элементов, вычислить: максимальный по модулю элемент вектора и сумму элементов вектора, расположенных между первым и вторым положительными элементами.
6) В векторе, состоящем из n вещественных элементов, вычислить: номер максимального по модулю элемента вектора и сумму элементов вектора, расположенных после первого положительного элемента.
7) В векторе, состоящем из вещественных элементов, вычислить: количество элементов вектора, больших С и произведение элементов вектора, расположенных после максимального по модулю элемента.
8) В векторе, состоящем из к целых элементов, вычислить: количество положительных элементов вектора и сумму элементов вектора, расположенных после последнего элемента, равного нулю.
9) В векторе, состоящем из n вещественных элементов, вычислить: произведение отрицательных элементов вектора и сумму положительных элементов вектора, расположенных до максимального элемента.
10) В векторе, состоящем из n вещественных элементов, вычислить: минимальный элемент вектора и сумму элементов вектора, расположенных между первым и последним положительными элементами.

11) В векторе, состоящем из n вещественных элементов, вычислить: количество элементов вектора, лежащих в диапазоне от A до B и сумму элементов вектора, расположенных после максимального элемента.

12) В векторе, состоящем из n вещественных элементов, вычислить: произведение положительных элементов вектора и сумму элементов вектора, расположенных до минимального элемента.

13) В векторе, состоящем из и вещественных элементов, вычислить: сумму элементов вектора с нечетными номерами и сумму элементов вектора, расположенных между первым и последним отрицательными элементами.

14) В векторе, состоящем из n вещественных элементов, вычислить: номер минимального элемента вектора и сумму элементов вектора, расположенных между первым и вторым отрицательными элементами.

15) В векторе, состоящем из n вещественных элементов, вычислить: количество элементов вектора, равных 0 и сумму элементов вектора, расположенных после минимального элемента.

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

Что нужно запомнить (краткие тезисы)

Перед написанием очередного приложения, очень важно вспомнить (запомнить) следующие вещи о одномерных Java-массивах:

1. Массив (*array*) представляет собой упорядоченную совокупность ***однотипных*** данных (значений или переменных), доступ к которым осуществляется через общее имя и порядковый номер (индекс).
2. В Java есть два типа массива: ***одномерные*** (самые востребованные) и ***многомерные*** (двух-, трёхмерные и т.д.).
3. Одномерный массив – это список (вектор) однотипных связанных данных (значений), элементы которого располагаются в непрерывной памяти друг за другом, т.е. вместе.
4. Главное преимущество массива – это эффективное использование памяти и организация хранения элементов в ней (памяти). У одномерных массивов один из самых быстрых доступов к элементам. Доступ к элементам не зависит от количества самих элементов и выполняется за константное время – O(1).
5. Массив в Java – это ***объект***, ***класс*** которого ***создаётся «на лету»*** перед созданием объекта массива во время выполнения кода.
6. Память для массива в Java динамически распределяется с помощью оператора ***new***.
7. При создании массива его размер может задаваться любым целым числом из диапазона типа ***int*** или выражением, результат которых является любое целое число в диапазоне типа ***int***.
8. Если размер массива задан некорректно (отрицательным значением) или данное значение получается в результате выполнения соответствующего выражения, то во время выполнения программы выбрасывается исключительная ситуация ***NegativeArraySizeException***.
9. Объект массива может быть создан с нулевой длиной.
10. Массив может ***хранить*** как данные ***примитивных типов***, так и ***ссылочных типов*** (ссылки на объекты).
11. Доступ к элементам массива осуществляется через их порядковый номер – соответствующий ***индекс*** (индекс обозначает соответствующее положение конечного элемента в массиве).

12. Нумерация индекса элементов массива начинается ***с нуля***.
13. При доступе к элементу в качестве индекса можно задавать любое целое число в диапазоне типа ***int***.
14. Границы массива в Java строго контролируются. Если код (индекс) выходит за пределы массива или получается отрицательным, то при выполнении программы выбрасывается исключительная ситуация ***ArrayIndexOutOfBoundsException***.
15. Т.к. массив реализован в виде объекта, то в нём дополнительно содержится свойство только для чтения ***length***, которое хранит количество элементов массива.
16. Если необходимо обработать каждый элемент массива, то это удобно сделать с помощью циклов, особенно с помощью цикла ***for***, где счётчик соответствующего цикла комбинируется с индексом соответствующего элемента массива.
17. Начиная с JDK 5.0 в языке Java появилась ещё одна разновидности цикла – расширенный цикл ***for*** (или ***for-each***), который призван максимально облегчить (рационализировать) перебор элементов любого из контейнеров в языке Java, в том числе и элементов массива. Помните, с помощью расширенного цикла ***for-each*** невозможно изменить состояние контейнера. Для этого лучше воспользоваться обычным циклом со счётчиком ***for***.
18. Для получения дополнительного функционала при работе с массивами используют соответствующие статические методы утилитного (сервисного) класса ***java.util.Arrays***.
19. Если необходимо создать метод, который бы принимал неограниченное количество однотипных данных, то это можно сделать с помощью специального дополнительного спецификатора для параметра метода – многоточие «...». Этот спецификатор ставится после типа соответствующего параметра. С точки зрения языка Java – это неявное декларирование объекта массива, т.е. внутри метода работа с данным параметром будет похоже на то, что идёт работа с одномерным массивом. Но для пользователя будет казаться, что есть множество однотипных перегруженных версий данного метода. К примеру, статический метод, который должен искать максимальный элемент среди любого количества переданных извне целочисленных данных, в Java декларируется следующим образом: *public static int max(int... params) { ... }* .
20. В методе можно объявить только один параметр, который обозначает передачу неограниченного количества однотипных данных, и только самым последним в списке параметров метода.

Пример выполнения практического задания с использованием одномерных массивов в Java

Задание

Дан целочисленный вектор размера N. Необходимо проверить, чередуются ли в нём чётные и нечётные числа.

Решение

1) Т.к. в основе задания лежит разработка вычислительного алгоритма, то спроектируем данный алгоритм и представим его графическое представление (или блок-схему):

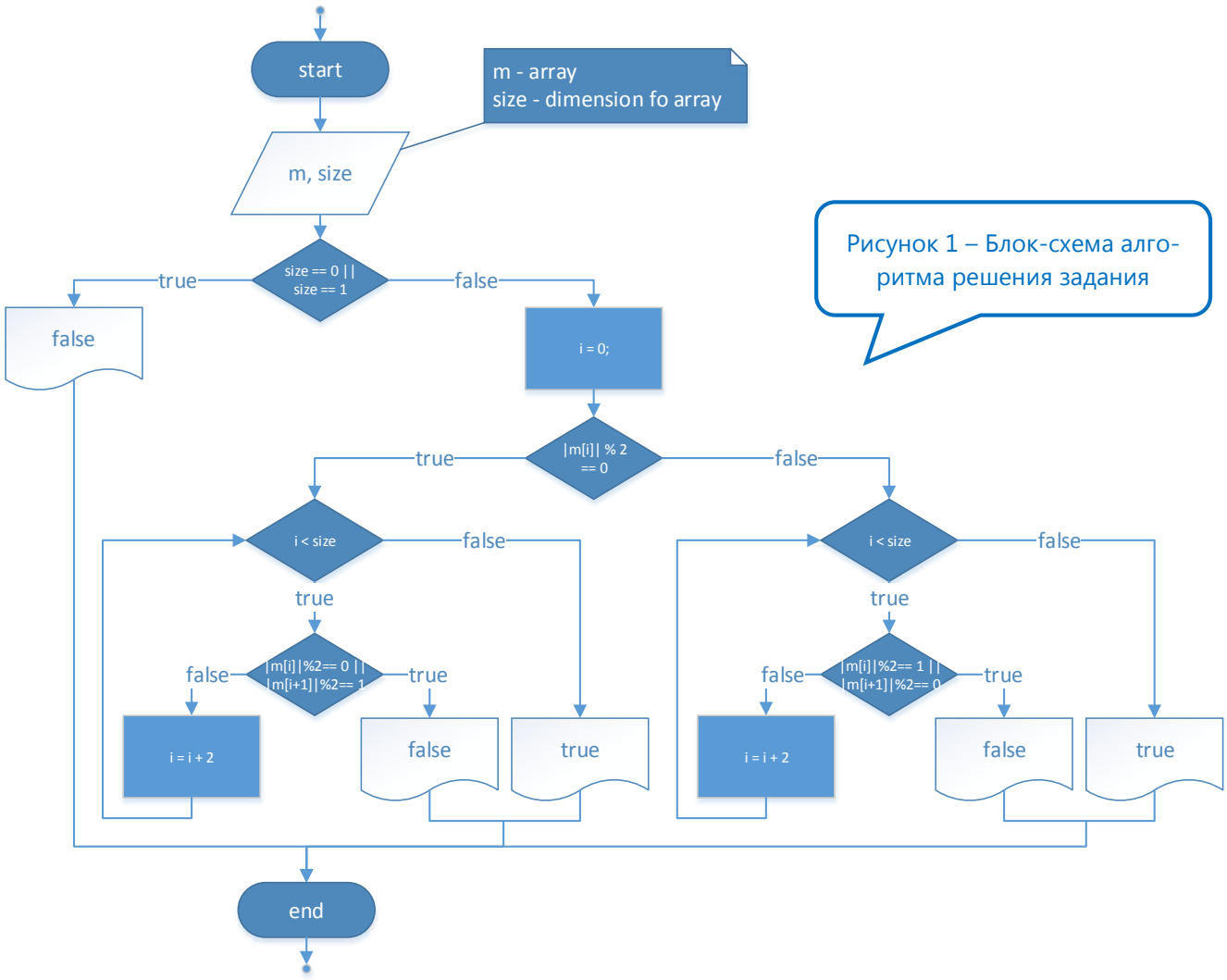

Рисунок 1 – Блок-схема алгоритма решения задания

2) Реализуем основной класс бизнес-логики программы для работы с целочисленными массивами *ArrayWorker*. Данный класс содержит один статически метод, который будет осуществлять целевую проверку: *checkEvenOddNumbersInterchange(int[ ] array)*. Он принимает на вход один параметр – ссылочную переменную, которая ссылается на соответствующий массив, и возвращает булевское значение, которое подтверждает, чередуются ли в целевом массиве чётные и нечётные числа:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab06.model.logic;

import static java.lang.Math.abs;

public class ArrayWorker {
     public static boolean checkEvenOddNumbersInterchange(int[] array) {

         if (array.length == 0 || array.length == 1) {
             return false;
         }

         if (array[0] % 2 == 0) {
             for (int i = 1; i < array.length; i += 2) {
                 if (abs(array[i]) % 2 == 0 || abs(array[i + 1]) % 2 == 1) {
                    return false;
                 }
             }
         } else {
             for (int i = 1; i < array.length; i += 2) {
                 if (abs(array[i]) % 2 == 1 || abs(array[i + 1]) % 2 == 0) {
                    return false;
                 }
             }
         }
         return true;
     }
}
```

3) Опишем класс *ArrayInitializer,* который будет инициализировать элементы массива случайными значениями из указанного промежутка *[min, max]*:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab06.util;

import java.util.Random;

public class ArrayInitializer {
```

Метод для «рандомной» инициализации элементов массива из промежутка [min, max]

```
public static void rndInit(int[] array, int min, int max) {

		Random random = new Random();

		for (int i = 0; i < array.length; i++) {
			array[i] = random.nextInt(max - min + 1) + min;
		}
	}
}
```

4) Реализуем стандартный класс для отображения данных с использованием системной консоли:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab06.view;

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

5) На заключительном этапе соберём из разработанных компонентов (классов) готовую программу, которая будет проверять все варианты работы бизнес-логики программы. Для этого опишем тестовый класс *Lab06*. В нём будет описан стартовый статический метод ***main(...)***:

```
package by.bntu.fitr.poisit.vikvik.javalabs.lab06.controller;

import java.util.Arrays;

import by.bntu.fitr.poisit.vikvik.javalabs.lab06.model.logic.ArrayWorker;
import by.bntu.fitr.poisit.vikvik.javalabs.lab06.util.ArrayInitializer;
import by.bntu.fitr.poisit.vikvik.javalabs.lab06.view.Printer;

public class Lab06 {
    public static void main(String[] args) {

        // example 1
        int size = 0;
        int[] array = new int[size];
        boolean result = ArrayWorker.checkEvenOddNumbersInterchange(array);
        Printer.print("\nArray: " + Arrays.toString(array));
        Printer.print("\nResult: " + result);

        // example 2
        size = 1;
```

Проверка работы логики программы при нулевом массиве

Проверка работы логики программы в случае, если массив состоит из одного элемента

```
            array = new int[size];
            result = ArrayWorker.checkEvenOddNumbersInterchange(array);
            Printer.print("\nArray: " + Arrays.toString(array));
            Printer.print("\nResult: " + result);

            // example 3
            size = 10;
            array = new int[size];
            ArrayInitializer.rndInit(array, -10, 10);
            result = ArrayWorker.checkEvenOddNumbersInterchange(array);
            Printer.print("\nArray: " + Arrays.toString(array));
            Printer.print("\nResult: " + result);

    }
}
```

Проверка работы логики программы над массивами, значение элементов которых были сгенерированы случайным образом

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

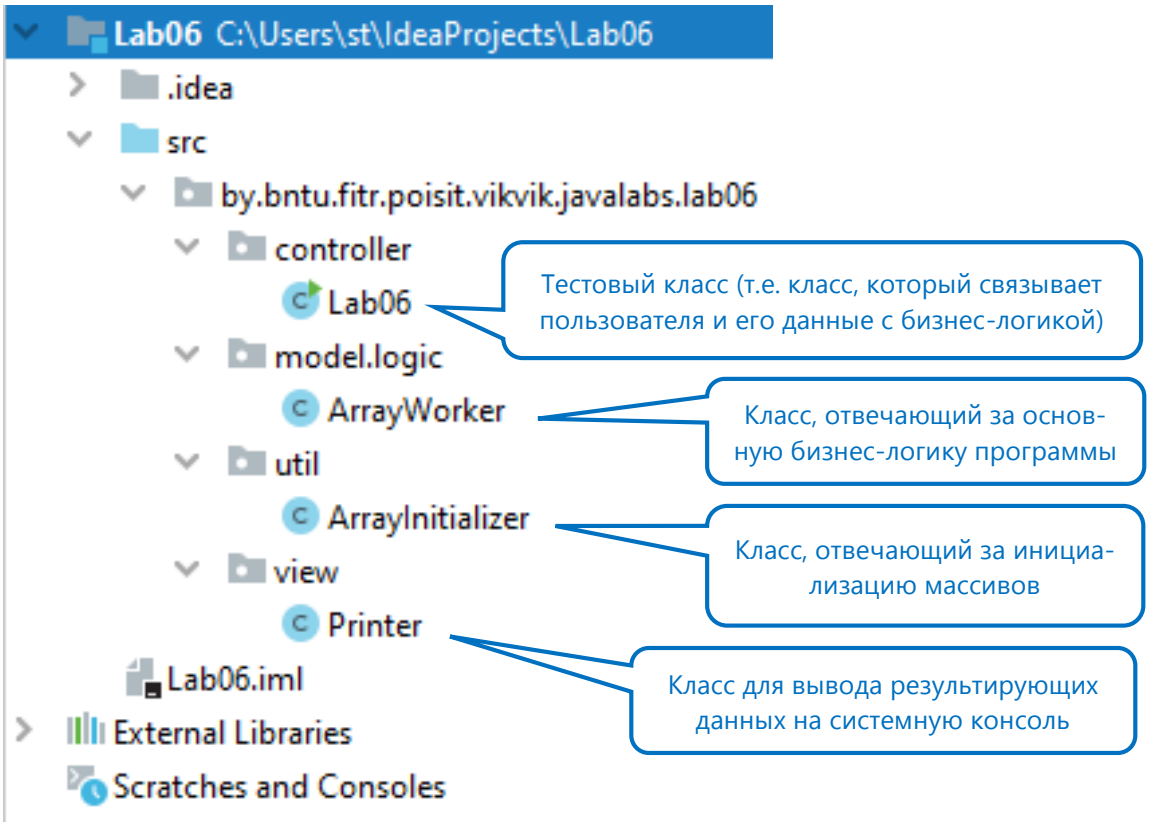

Рисунок 2 – Архитектура разработанного приложения

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 3 и 4, а их аналогичная компиляция и запуск в интегрированной среде разработки IntelliJ IDEA – на рисунке 5.

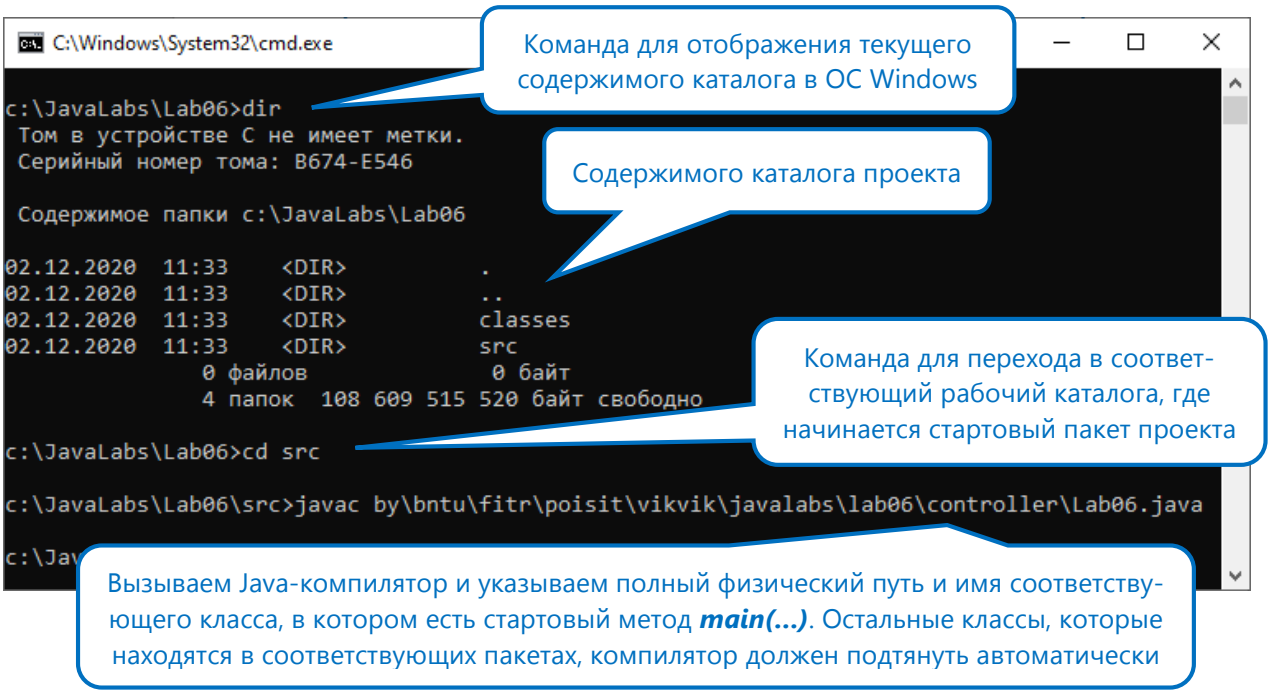

Рисунок 3 – Содержимое рабочего каталога приложения и его компиляция

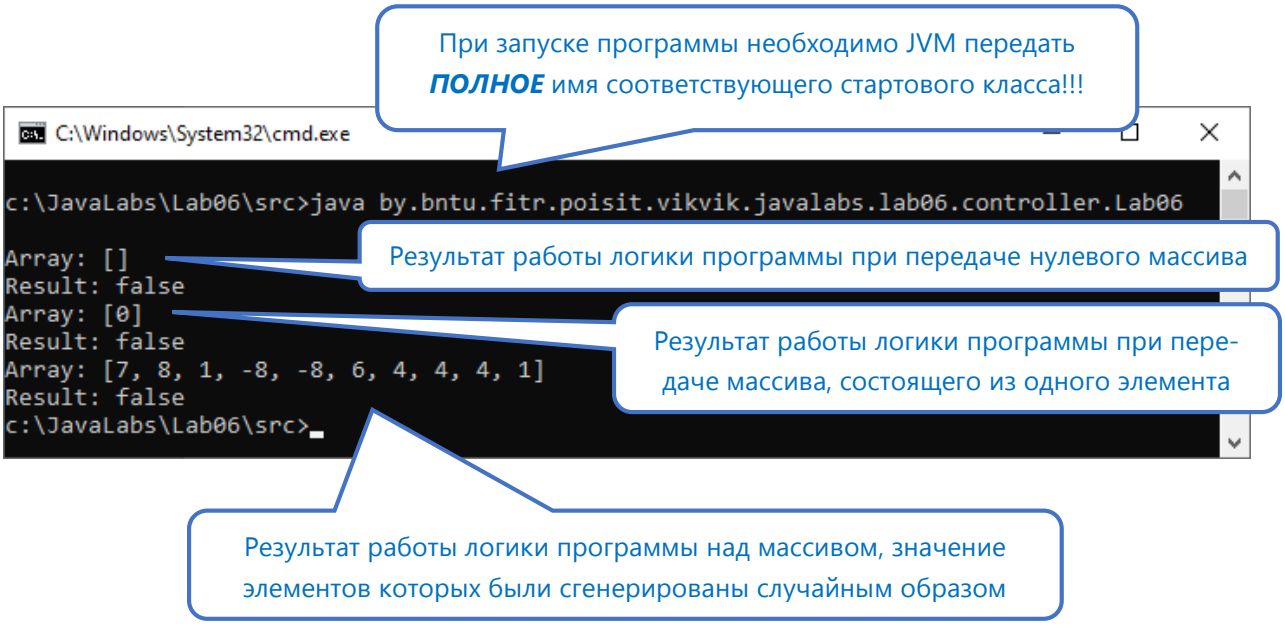

Рисунок 4 – Результат работы программы для всех тестовых случаем, полученный с использованием нативных средств JDK

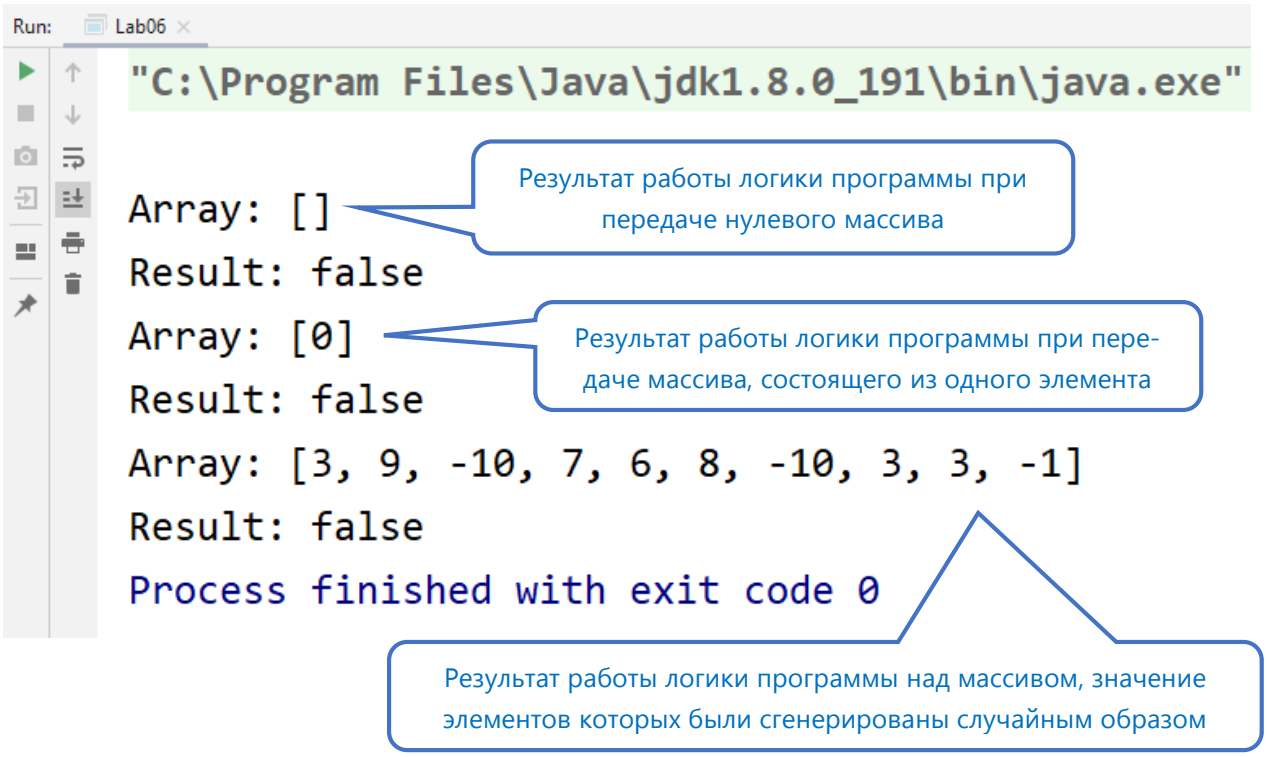

Рисунок 5 – Результат работы программы для всех тестовых случаем в IDE Eclipse

Как можно улучшить вышеприведённый код?

1) У вышеизложенного варианта реализации задания есть серьёзная ошибка, которая может привести к краху всей программы. Попробуйте найти и устранить данную ошибку.
2) Можно ли улучшить алгоритм логики решения задания?

Контрольные вопросы 

1) Что такое массив?
2) Зачем нужно при проектировании и реализации программы использовать массивы?
3) Какие существуют типы массивов в Java?
4) Чем массивы в Java отличаются от массивов в C/C++/C#?
5) Опишите особенности Java-массивов и его свойства.
6) Почему в Java массивы занимают особое место по сравнению с другими контейнерами? Опишите основные преимущества и недостатки Java-массивов по сравнению с существующими библиотечными контейнерами: эффективность, типизация и возможность хранить примитивы.
7) Каково время доступа к элементам массива?
8) Почему нумерация элементов массива начинается с нуля?
9) Какие существуют нотации в Java объявления переменной типа массива?
10) Способы объявления, конструирования и инициализации массивов в Java?
11) Если массив – это объект, а все объекты создаются на базе описанного класса, то где хранятся классы-массивов?
12) Как передать массив в качестве входного параметра в метод?
13) Как возвратить из метода значение в виде массива?
14) Как инициализировать элементы одномерного массива во время объявления ссылочной переменной?
15) Как инициализировать элементы одномерного массива, который создаётся «на лету» в виде анонимного объекта и либо передаётся в качестве параметра в вызываемый метод, либо возвращается как результат работы метода?
16) Массивы и примитивные типы данных.
17) Массивы и ссылочные типы данных.
18) Что физически содержит (хранит) внутри себя массив?
19) Какие могут возникнуть исключения при работе с массивами и на что они указывают?
20) Какие существуют способы копирования (клонирования) массивов (элементарных и ссылочных типов)?
21) Что такое поверхностное и глубокое копирование объектов?
22) Что такое поверхностное и глубокое копирование массивов?

23) Для чего используется утилитный класс ***java.util.Arrays***?
24) Как сравнить два массива?
25) Как отсортировать элементы массивов?
26) Как произвести поиск элемента в массиве?
27) Как объявить метод с переменным количество передаваемых параметров? Как реализована логика данного метода?
28) Каковы основные правила объявления методов с переменным количеством параметров?
29) Как использовать расширенный цикл ***for*** (ещё его иногда называют *for-each*), который появился с JDK 5.0?
30) Чего нельзя сделать с помощью расширенного цикла ***for*** в *Java*?

Как стать хорошим разработчиком?

Равичадран Джейв, Full stack-разработчик структуры программного обеспечения

1. ***Полюбите ошибки.*** Внимательно просматривайте каждое сообщение об ошибке. Станьте истинным фанатом ошибок.
2. ***Показывайте свой код другим людям.*** Расскажите им, как вы написали эту программу – и не просто то, какими соображениями вы пользовались при ее создании, а тот путь, который вы прошли при написании самого ПО.
3. Разрабатывайте программное обеспечение и ***пишите коды для собственного удовольствия***. Если вам становится скучно за написанием кода, то лучше займитесь другим любимым делом: посмотрите фильм, погуляйте немного, встретьтесь с друзьями – в общем, ***займитесь чем-нибудь, что приносит вам истинное удовольствие***.
4. ***Следите за здоровьем.*** Какое-то время, пока мой компьютер компилировал, я поднимал гантели и ходил с ними из одной комнаты в другую, а потом выходил в коридор и шел обратно. Кодовая база тогда состояла из 15 000 строк – и это действительно большая цифра для компиляторов Clipper и Turbo C, так как им необходимо было сгенерировать объектный код для компоновщика, а с оперативной памятью в 256 Mb этот процесс проходил крайне медленно. Поэтому процесс поднимания и хождения с гантелями растягивался иногда на 30 минут, но именно это и помогало мне оставаться в хорошей физической форме!
5. Всегда ***следите за выходящими новинками*** в сфере разработки программного обеспечения, особенно за наиболее популярными экземплярами. Постарайтесь определить для себя, что

именно в них кажется вам скучным или бесполезным – именно это понимание и станет отправной точкой в вашей работе.

Source: www.kv.by/post/1055398-kak-stat-velikim-horoshim-razrabotchikom?utm_source=weekly_letter