Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра «Программное обеспечение вычислительной техники

и автоматизированных систем»

**Отчёт**

по лабораторной работе № 08

по дисциплине ***«Объектно-ориентированное программирование»***

тема: «Базовый синтаксис языка Java. Циклические конструкции. Итерационные алгоритмы»

Название бригады: «eNotes»

|  |  |  |
| --- | --- | --- |
| Исполнитель: |  | студенты группы 10702217  Тарсенко И.В.  Камадей А.В. |
| Преподаватель: |  | Иванченко Виктор Викторович |

2018-2019 учебный год

**ЛАБОРАТОРНАЯ РАБОТА № 08**

**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.

**Задание А**

1. Написать программу, которая подсчитывает количество только чётных (или нечётных) цифр заданного натурального числа.
2. Написать программу, которая находит сумму цифр заданного натурального числа.

**Задание B**

1. Разработайте программу, которая проверяет, что среди цифр заданного натурального числа хотя бы одно число чётное.
2. Разработайте программу, которая проверяет, что среди цифр заданного натурального числа преобладают чётные .

Задание C

1. Разработайте программу, которая переворачивает заданное 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.
2. Разработайте программу, которая проверяет, что заданное натуральное число читается одинаково слева направо и справа налево (т.е. является палиндромом). К примеру, число 1235321 – удовлетворяет условию, а число 12345321 – нет.

Задание D

1. 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 34992991.
2. Найти наибольшую цифру у заданного натурального числа. К примеру, в числе 18273645 максимальная цифра восемь, а в числе 777 – семь.

Задание E

1. Найти все элементы последовательности чисел Фибоначчи до указанного пользователем порядкового номера.
2. Найти число трибоначчи по его порядковому номеру.

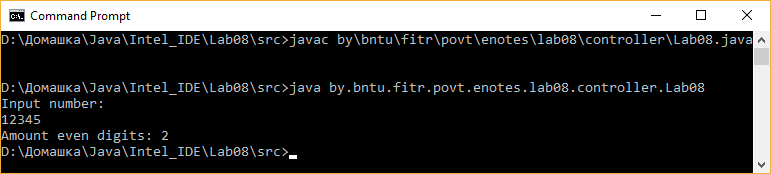
Числа трибоначчи – последовательность целых чисел *{tn}*, заданного с помощью рекуррентного соотношения: *t0 = 0, t1 = 0, t2 = 1, tn+3 = tn+2 + tn+1 + tn.* Пример ряда трибоначчи: *0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, …*

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

Требования к выполнению

1. Каждому участнику из команды нужно просмотреть все задания и выполнить по одному понравившемуся заданию из каждого раздела. Задания из соответствующих разделов внутри команды не должны повторяться.
2. Для некоторых 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 рекурсивного подхода).
4. На базе спроектированных алгоритмов разработать простейшее интерактивное консольное приложение с использование архитектурного 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.povt.nameofteam.javalabs.lab08****.*
6. При выполнения задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый и легко поддерживаемый и читаемый код.
7. 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.
8. Если логически не подразумевается или в задании иного не указано, то входными и выходными данными являются вещественные числа (числа с плавающей запятой).
9. Все задания необходимо решать используя только базовые операции (простые операторы), определённые над примитивными типами данных в языке программирования Java, и условные и циклические конструкции (т.е. не нужно использовать массивы или любые другие контейнеры данных, операции над строковыми типами данных и т.д.).
10. В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».
11. 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***.
12. 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.
13. Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем.
14. Интерфейс программы и комментарии должны быть на английском языке.
15. При проверки работоспособности приложения необходимо проверить все тестовые случаи.
16. При выполнении задания не рекомендуется использовать интегрированные средства разработки (*Integrated Development Environment, IDE*). Лучше задействовать любой текстовый редактор и основные компоненты Java (компилятор – ***javac***, утилиту для запуска *JVM* – ***java***).
17. При разработке программ придерживайтесь соглашений по написанию кода на *Java* (***Java Code-Convention****)* !!!

Результаты выполнения основного задания

Рисунок 1 – результат работы программы потщета количества четных цифр в числе

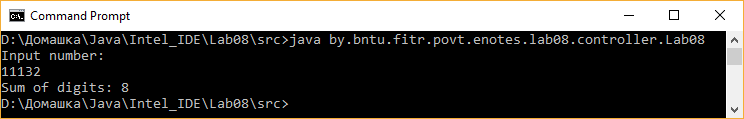


Рисунок 2 – результат работы программы подсчета суммы цифр числа

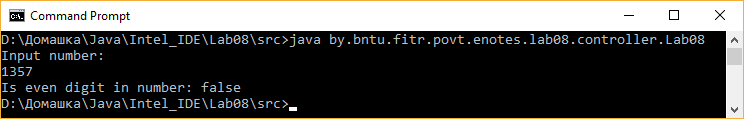


Рисунок 3.1 – результат работы программы проверки есть ли четная цифра в числе

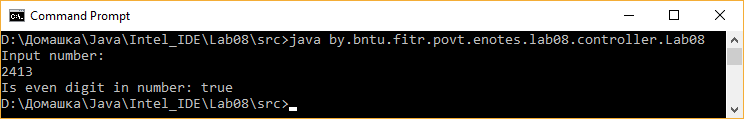


Рисунок 3.2 – результат работы программы проверки есть ли четная цифра в числе

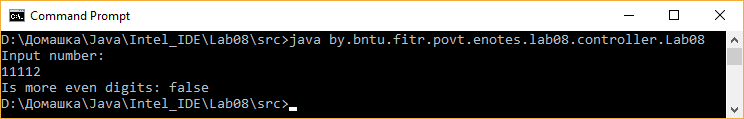


Рисунок 4.1 – результат работы программы проверки: количетво четных цифр больше нечетных

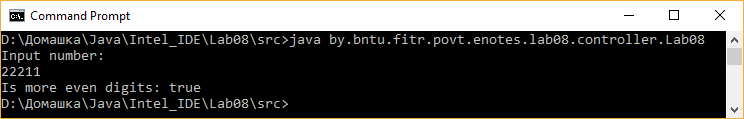


Рисунок 4.2 – результат работы программы проверки: количетво четных цифр больше нечетных

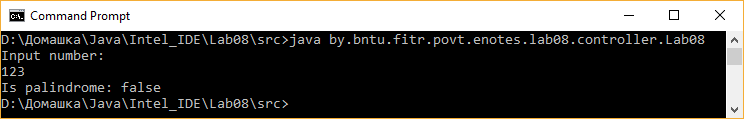


Рисунок 5.1 – результат работы программы проверки на палиндром



Рисунок 5.2 – результат работы программы проверки на палиндром

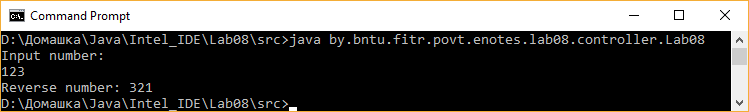


Рисунок 6.1 – результат работы программы реверса числа

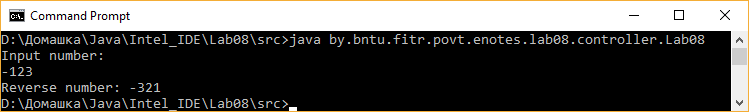


Рисунок 6.2 – результат работы программы реверса отрицательного числа

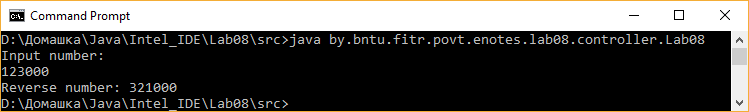


Рисунок 6.3 – результат работы программы реверса числа оканчивающегося нулями

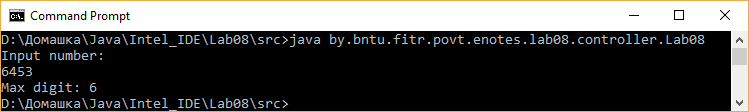


Рисунок 7 – результат работы программы поиска максимальной цифре в числе

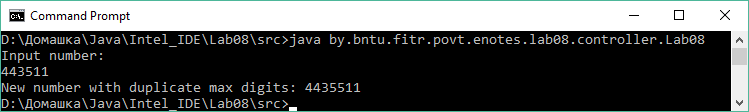


Рисунок 8 – результат работы программы удвоения максимальной цифры в числе

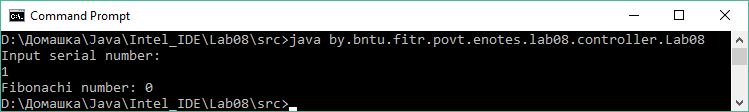


Рисунок 9.1 – результат работы программы поиска числа Фибоначчи по номеру

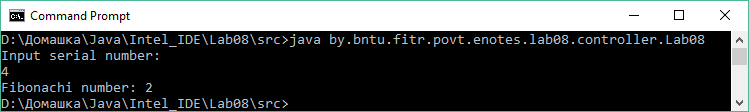


Рисунок 9.2 – результат работы программы поиска числа Фибоначчи по номеру

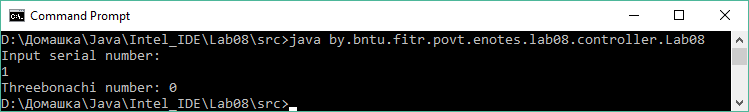


Рисунок 9.1 – результат работы программы поиска числа Трибоначчи по номеру

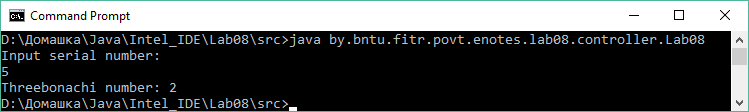


Рисунок 9.2 – результат работы программы поиска числа Трибоначчи по номеру

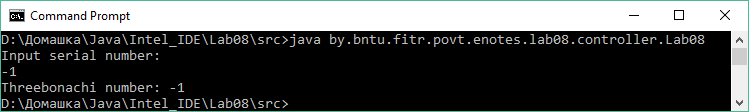


Рисунок 9.3 – результат работы программы поиска числа Трибоначчи по номеру

Что мы узнали нового в процессе выполнения лабораторной работы (выводы)

1) Если создать бесконечный цикл и не использовать ни одного break, то компилятор будет выдавать ошибку unreachable statement, если после цикла писать какие-нибудь команды.

2) В интегрированной среде Intellij IDEA при написании «значение».fori – получается цикл for от 0 до «значения».

3) При написании циклов условие должно выводить только значение boolean(true, false). Числа – 1, 0 и т.д. компилятор не принимает.

4) В цикле for-each мы обращаемся не непосредственно к массиву а присваиваем к переменной значение массива и работаем с этой переменной, т.е мы не можем изменить значение массива.

5) For используется, если мы знаем количество итераций, while удобней использовать, если мы не знаем количество итераций.

6) Если нам необходимо завершить первый цикл из вложенного, можно использовать метку.

*ПРИЛОЖЕНИЕ А*

Листинг исходного кода класса(ов)

Класс NumberLogic

***package by.bntu.fitr.povt.enotes.lab08.model.logic;  
  
public class NumberLogic {  
 public static final int DECIMAL\_BASE = 10;  
  
 public static int countDigitsAmount(int number) {  
 int count = 0;  
 while (number > 0) {  
 count ++;  
 number /= DECIMAL\_BASE;  
 }  
 return count;  
 }  
  
 public static boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
  
 public static int countEvenDigits(int number) {  
 int count = 0;  
 while (number > 0) {  
 count += isEven(number % DECIMAL\_BASE) ? 1 : 0;  
 number /= DECIMAL\_BASE;  
 }  
 return count;  
 }  
  
 public static int countSumDigits(int number) {  
 int sum = 0;  
 while (number > 0) {  
 sum += number % DECIMAL\_BASE;  
 number /= DECIMAL\_BASE;  
 }  
 return sum;  
 }  
  
 public static boolean isEvenDigitInNumber(int number) {  
 boolean result = false;  
 while (number > 0) {  
 if (isEven(number % DECIMAL\_BASE)) {  
 result = true;  
 break;  
 }  
 number /= DECIMAL\_BASE;  
 }  
 return result;  
 }  
  
 public static boolean isMoreEvens(int number) {  
 int evenCounter = 0;  
 int oddCounter = 0;  
 while (number > 0) {  
 if (isEven(number % DECIMAL\_BASE)) {  
 evenCounter++;  
 } else {  
 oddCounter++;  
 }  
 number /= DECIMAL\_BASE;  
 }  
 return evenCounter > oddCounter;  
 }  
  
 public static boolean isPalindrome(int number) {  
 int seveNumber = number;  
 int reverseNumber = 0;  
 while (number > 0) {  
 reverseNumber = reverseNumber \* DECIMAL\_BASE + number % DECIMAL\_BASE;  
 number /= DECIMAL\_BASE;  
 }  
 return reverseNumber == seveNumber;  
 }  
  
 public static int reverseNumber(int number) {  
 int reverseNumber = 0;  
 int digitCounter = 0;  
 boolean positive = true;  
 if (number < 0) {  
 positive = false;  
 number = Math.abs(number);  
 }  
 while (number > 0) {  
 digitCounter++;  
 reverseNumber = reverseNumber \* DECIMAL\_BASE + number % DECIMAL\_BASE;  
 number /= DECIMAL\_BASE;  
 }  
 reverseNumber \*= (int)Math.pow(10,digitCounter - countDigitsAmount(reverseNumber));  
 if (!positive) {  
 reverseNumber \*= -1;  
 }  
 return reverseNumber;  
 }  
  
 public static int definiteMaxDigit(int number) {  
 int maxDigit = 0;  
 while (number > 0) {  
 maxDigit = Math.max(number % DECIMAL\_BASE, maxDigit);  
 number /= DECIMAL\_BASE;  
 }  
 return maxDigit;  
 }  
  
 public static int duplicateMaxDigit(int number) {  
 int maxDigit = definiteMaxDigit(number);  
 int newNumber = 0;  
 while (number > 0) {  
 int lastDigit = number % DECIMAL\_BASE;  
 newNumber = newNumber \* DECIMAL\_BASE + lastDigit;  
 if(lastDigit == maxDigit){  
 newNumber = newNumber \* DECIMAL\_BASE + lastDigit;  
 }  
 number /= DECIMAL\_BASE;  
 }  
 return reverseNumber(newNumber);  
 }  
}***

**Класс FibonachiLogic**

***package by.bntu.fitr.povt.enotes.lab08.model.logic;  
  
public class FibonachiLogic {  
  
 public static final long FIRST\_FIBONACHI\_NUMBER = 0;  
 public static final long SECOND\_FIBONACHI\_NUMBER = 1;  
 public static final long FIRST\_THREEBONACHI\_NUMBER = 0;  
 public static final long SECOND\_THREEBONACHI\_NUMBER = 0;  
 public static final long THIRD\_THREEBONACHI\_NUMBER = 1;  
 public static final long FIRST\_DEFAULT\_SERIAL\_NUMBER = 2;  
 public static final long SECOND\_DEFAULT\_SERIAL\_NUMBER = 3;  
  
 public static long countFibonachiNumber(int serialNumber) {  
 if (serialNumber <= 0){  
 return -1;  
 }  
 long value1 = FIRST\_FIBONACHI\_NUMBER;  
 long value2 = SECOND\_FIBONACHI\_NUMBER;  
 long result = value1;  
 if (serialNumber == FIRST\_DEFAULT\_SERIAL\_NUMBER) {  
 result = value2;  
 }  
 while (serialNumber > FIRST\_DEFAULT\_SERIAL\_NUMBER) {  
 long newValue = value1 + value2;  
 value1 = value2;  
 value2 = newValue;  
 result = newValue;  
 serialNumber--;  
 }  
 return result;  
 }  
  
 public static long countThreebonachiNumber(int serialNumber) {  
 if (serialNumber <= 0){  
 return -1;  
 }  
 long value1 = FIRST\_THREEBONACHI\_NUMBER;  
 long value2 = SECOND\_THREEBONACHI\_NUMBER;  
 long value3 = THIRD\_THREEBONACHI\_NUMBER;  
 long result = value1;  
 if (serialNumber == FIRST\_DEFAULT\_SERIAL\_NUMBER) {  
 result = value2;  
 } else if (serialNumber == SECOND\_DEFAULT\_SERIAL\_NUMBER) {  
 result = value3;  
 }  
 while (serialNumber > SECOND\_DEFAULT\_SERIAL\_NUMBER) {  
 long newValue = value1 + value2 + value3;  
 value1 = value2;  
 value2 = value3;  
 value3 = newValue;  
 result = newValue;  
 serialNumber--;  
 }  
 return result;  
 }  
}***