**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: **«Визуальные средства разработки программных**

**приложений»**

на тему: **Разработка многопоточных приложений**

Выполнил: студент гр. ИТП-31

Дашкевич Д.А.

Принял: ассистент

Белявский Е. В.

Гомель 2020

**Цель**: изучить создание многопоточных приложений с использованием языка программирования *Java*.

**Ход работы**

**Вариант 7**

# **Задание:**

Разработать приложение, в котором выполняется следующий алгоритм: два параллельных потока циклически выполняют вызов функций (согласно варианту). Каждый поток использует свою функцию. Результаты работы каждый поток помещает в общую очередь конечной длины. Третий поток забирает из очереди результаты работы функций и отображает их на экране в произвольной форме. Схема взаимодействия потоков приведена на рисунке 1.

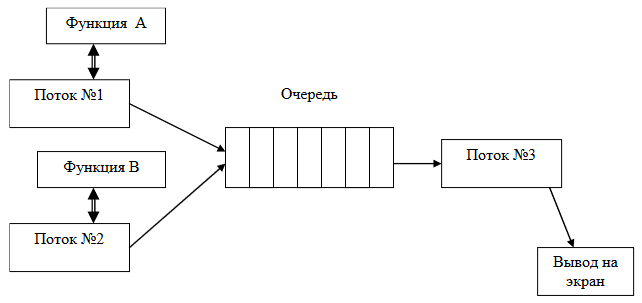


Рисунок 1 – Схема взаимодействия потоков

Очередь должна быть реализована с использованием объектов синхронизации. Потоки должны корректно завершаться при завершении приложения. Для реализации выбрать любые две функции из таблицы согласно варианту.

****

Рисунок 2 – Задание

**Результат работы программы:**

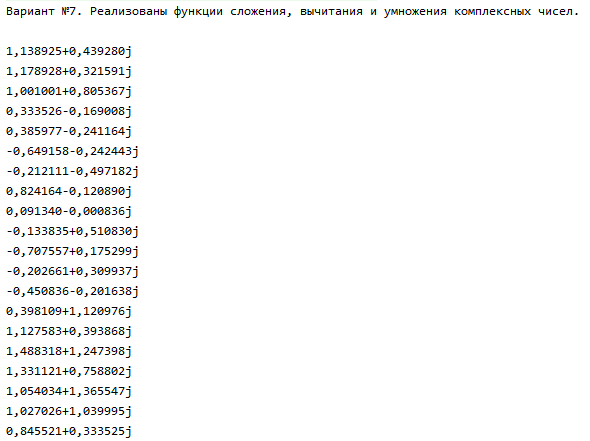


Рисунок 3 – Вывод в консоль результата работы программы

**Вывод:** в результате выполнения лабораторной работы были изучены потоки в *Java* и разработано многопоточное приложение программы с тремя потоками на основании схемы задания.

**Приложение А**

**Исходный код программы**

**Runner.java:**

**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Random;  
  
**public class** Runner {  
 **public static void** main(String[] args) {  
  
 System.***out***.println(**"Вариант №7. Реализованы функции сложения, вычитания и умножения комплексных чисел.\n"**);  
  
 **int** numberOfElements = 10;  
 **int** queueLength = 3;  
  
 List<Complex> inputList1 = **new** ArrayList<>(numberOfElements);  
 List<Complex> inputList2 = **new** ArrayList<>(numberOfElements);  
 Random random = **new** Random();  
 **for** (**int** i = 0; i < numberOfElements; i++) {  
 inputList1.add(**new** Complex(random.nextDouble(), random.nextDouble()));  
 inputList2.add(**new** Complex(random.nextDouble(), random.nextDouble()));  
 }  
  
 SynchronizedQueue<Complex> queue = **new** SynchronizedQueue<>(queueLength);  
  
 Thread input1 = **new** Thread(() -> {  
 **for** (**int** i = 0; i < numberOfElements; i++) {  
 **if** (!queue.put(inputList1.get(i).add(inputList2.get(i)))) {  
 i--;  
 }  
 }  
 });  
  
 Thread input2 = **new** Thread(() -> {  
 **for** (**int** i = 0; i < numberOfElements; i++) {  
 **if** (!queue.put(inputList1.get(i).sub(inputList2.get(i)))) {  
 i--;  
 }  
 }  
 });  
  
 Thread output = **new** Thread(() -> {  
 **for** (**int** i = 0; i < 2 \* numberOfElements; i++) {  
 **if** (!queue.isEmpty()) {  
 System.***out***.println(queue.get());  
 } **else** {  
 i--;  
 }  
 }  
 });  
  
 input1.start();  
 input2.start();  
 output.start();  
  
 **try** {  
 input1.join();  
 input2.join();  
 output.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 input1.interrupt();  
 input2.interrupt();  
 output.interrupt();  
 }  
 }  
}

**SynchronizedQueue.java:**

**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** SynchronizedQueue<T> {  
  
 **private final int maxLength**;  
 **private final** List<T> **queue**;  
  
 **public** SynchronizedQueue(**int** maxLength) {  
 **this**.**maxLength** = maxLength;  
 **queue** = **new** ArrayList<>(maxLength);  
 }  
  
 **public synchronized int** size() {  
 **return queue**.size();  
 }  
  
 **public synchronized boolean** isEmpty() {  
 **return** size() == 0;  
 }  
  
 **public synchronized boolean** isMaxLength() {  
 **return** size() == **maxLength**;  
 }  
  
 **public synchronized** T get() {  
 **return queue**.remove(0);  
 }  
  
 **public synchronized boolean** put(T item) {  
 **if** (isMaxLength()) {  
 **return false**;  
 }  
 **return queue**.add(item);  
 }  
}

**Complex.java:**

**import** java.util.Objects;  
  
**public final class** Complex {  
 **private final double real**;  
 **private final double img**;  
  
 **public** Complex(**double** real, **double** img) {  
 **this**.**real** = real;  
 **this**.**img** = img;  
 }  
  
 **public** Complex() {  
 **this**(0.0, 0.0);  
 }  
  
 **public** Complex add(Complex another) {  
 **return new** Complex(**real** + another.**real**, **img** + another.**img**);  
 }  
  
 **public** Complex sub(Complex another) {  
 **return new** Complex(**real** - another.**real**, **img** - another.**img**);  
 }  
  
 **public** Complex mul(Complex another) {  
 **return new** Complex(**real** \* another.**real** - **img** \* another.**img**,  
 **real** \* another.**img** + **img** \* another.**real**);  
 }  
  
 **public double** getReal() {  
 **return real**;  
 }  
  
 **public double** getImg() {  
 **return img**;  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 Complex complex = (Complex) o;  
 **return** Double.*compare*(complex.**real**, **real**) == 0 &&  
 Double.*compare*(complex.**img**, **img**) == 0;  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**real**, **img**);  
 }  
  
 @Override  
 **public** String toString() {  
 **return** String.*format*(**"%f%+fj"**, **real**, **img**);  
 }  
}