Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационные системы и анализ данных

Центр программной инженерии

ОТЧЁТ

к лабораторной работе №2

по дисциплине

|  |
| --- |
| «Технологии разработки программных комплексов» |
| Распараллеливание потоков |

наименование темы

Вариант №28

Выполнил cтудент ИСТБ-21-2 Н.В. Зимин

номер группы подпись И. О. Фамилия

дата

Проверил Доцент З.А. Бахвалова

Должность подпись И. О. Фамилия

дата

Иркутск – 2024

# Содержание

[1 Постановка задачи 3](#_Toc164936456)

[2 UML-диаграмма классов 4](#_Toc164936457)

[3 Результаты тестирования работы программы 6](#_Toc164936458)

[4 Исходный код 7](#_Toc164936459)

[5 Примеры использования потоков 9](#_Toc164936460)

[Список использованных источников 10](#_Toc164936461)

# 1 Постановка задачи

Все варианты заданий подразумевают использование промежуточных буферов, представляющих собой динамические массивы. Максимальный размер буферов - N чисел. N определяется для каждого варианта. Потоки, помещающие числа в буферы, следят за переполнение буферов. Потоки, извлекающие числа из буферов, могут производить данную операцию в произвольный момент времени вне зависимости от того, заполнен ли буфер полностью или нет.

Выбранный язык программирования должен поддерживать объектно-ориентированную парадигму. Выбор остального окружения производится студентом самостоятельно.

Индивидуальное задание:

Значения констант и реализуемые потоками функции:

N = 2;

Первый поток – генерирует в первый буфер 30 случайных чисел из интервала от -1 до 1;

Второй поток – извлекает числа из буфера, начиная с первого и возводит их в квадрат. Результат выводится на экран.

# 2 UML-диаграмма классов

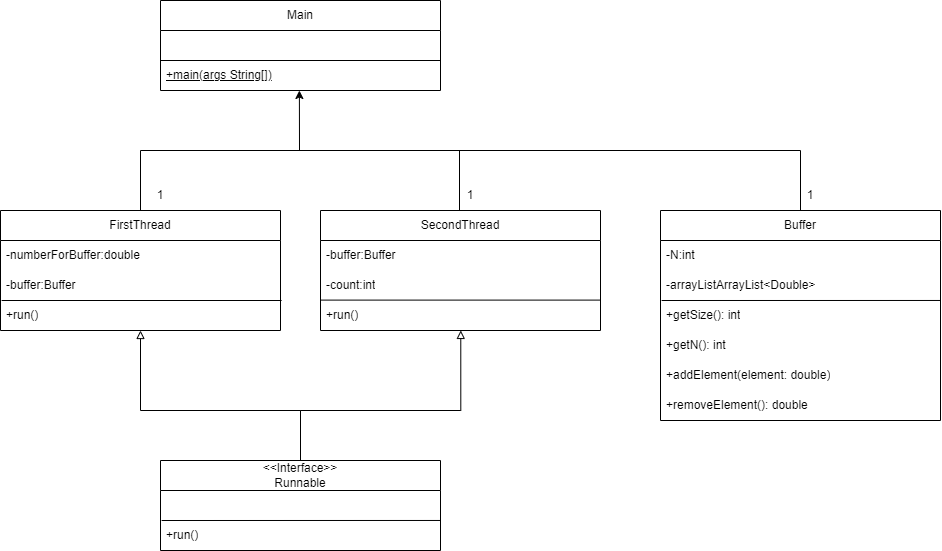


Рисунок 1 – Диаграмма классов программы

1. Main – начальный класс, точка входа в программу.
2. Menu – класс, реализующий интерфейс программы в консоли.
   1. public static void mainMenu() – метод, отображающий главное меню программы и реализующий взаимодействие с объектом класса DataAnalysis.
3. FirstThread – класс, наследующий интерфейс Runnable и реализующий первый поток, добавляющий числа в буфер.
   1. private double numberForBuffer – поле класса double, реализующее рандомное число для буфера.
   2. private Buffer buffer – поле класса Buffer, реализующее буфер, в который поток добавляет числа.
   3. public void run() – метод интерфейса Runnable, реализующий работу потока и генерирующий числа в диапазоне от -1 до 1 для буфера buffer.
4. SecondThread – класс, наследующий интерфейс Runnable и реализующий второй поток, извлекающий числа из буфера, возводящий их в квадрат и выводящий результат на экран.
   1. private double numberFromBuffer – поле класса double, реализующее извлекаемое из буфера число.
   2. private Buffer buffer – поле класса Buffer, реализующее буфер, из которого поток извлекает числа.
   3. public void run() – метод интерфейса Runnable, реализующий работу потока и извлекающий из буфера число для возведения в квадрат и вывода результата на экран.
5. Buffer – класс, реализующий буфер с которым взаимодействуют потоки.
   1. private int N – поле класса int, реализующее ограничение вместимости буфера.
   2. private ArrayList<Double> arrayList – поле класса ArrayList<>, реализующее динамический массив в который добавляются числа.
   3. public int getSize() – метод, возвращающий текущее количество элементов в массиве arrayList.
   4. public int getN() – метод, возвращающий ограничение вместимости буфера в виде поля N.
   5. public void addElement(double element) – метод, добавляющий в массив arrayList новый элемент.
   6. public double removeElement() – метод, возвращающий первый элемент массива arrayList и удаляющий его из массива.
6. Runnable – интерфейс, функционал которого реализуется классами FirstThread и SecondThread.
   1. public abstract void run() – метод, реализуемый классами-наследниками.

# 3 Результаты тестирования работы программы

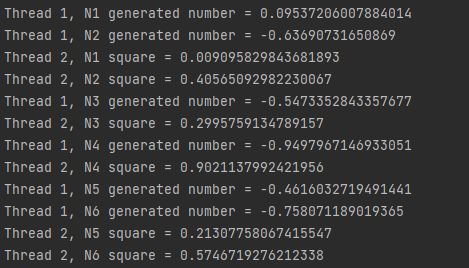


Рисунок 2 – результат работы программы

# 4 Исходный код

Код класса Main

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Buffer buffer = new Buffer();  //создание и запуск первого потока  FirstThread firstThread = new FirstThread(buffer);  Thread thread1 = new Thread(firstThread);  thread1.start();  //создание и запуск второго потока  SecondThread secondThread = new SecondThread(buffer);  Thread thread2 = new Thread(secondThread);  thread2.start();  }  } |

Код класса FirstThread

import java.util.concurrent.ThreadLocalRandom;

public class FirstThread implements Runnable{

private double numberForBuffer;

private Buffer buffer;

public FirstThread(Buffer buffer) {

this.buffer = buffer;

}

@Override

public void run() {

for (int i = 0; i < 30; i++) {

numberForBuffer = ThreadLocalRandom.current().nextDouble(-1, 1);

synchronized (buffer)

{

while (buffer.getSize() >= buffer.getN())

try {

buffer.wait();

} catch (InterruptedException e) {}

buffer.addElement(numberForBuffer);

System.out.println("Thread 1, N" + (i+1) + " generated number = " + numberForBuffer);

buffer.notify();

}

}

}

}

Код класса SecondThread

public class SecondThread implements Runnable{

private Buffer buffer;

private int count = 1;

public SecondThread(Buffer buffer) {

this.buffer = buffer;

}

@Override

public void run() {

while (true) {

synchronized (buffer) {

while(buffer.getSize() == 0) {

try {

buffer.wait();

} catch (InterruptedException e) {}

}

System.out.println("Thread 2, N" + count++ + " square = " + Math.pow(buffer.removeElement(), 2) );

buffer.notify();

}

}

}

}

Код класса Buffer

import java.util.ArrayList;

public class Buffer {

private int N = 2;

private ArrayList<Double> arrayList = new ArrayList<Double>();

public int getSize()

{

return arrayList.size();

}

public int getN() {

return N;

}

public void addElement(double element)

{

arrayList.add(element);

}

public double removeElement()

{

double returnElement = arrayList.get(0);

arrayList.remove(0);

return returnElement;

}

}

Ссылка на полный проект <https://github.com/CrusaderKleptoman/trainThread.git>

# 5 Примеры использования потоков

GIMPS (Great Internet Mersenne Prime Search) — широкомасштабный проект добровольных вычислений по поиску простых чисел Мерсенна.

Как следует из названия, целью проекта GIMPS является поиск новых простых чисел Мерсенна. Самое большое известное на данный момент простое число было найдено в рамках проекта GIMPS 7 декабря 2018 года и состоит из 24 862 048 десятичных цифр. Более того, 15 предыдущих рекордов также были установлены участниками GIMPS. Причина кроется в наличии эффективного (детерминированного) критерия их простоты, носящего имя Люка — Лемера. Для поиска простых чисел Мерсенна сервер GIMPS раздаёт клиентам простые «экспоненты» p для проверки числа на простоту тестом Люка — Лемера.

# Список использованных источников

1. 1. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Н.Новгород: ННГУ, 2000, 121 с. (2 изд. 2003).
2. ИНТУИТ. Параллельное программирование URL: https://www.intuit.ru/studies/courses/1110/153/info
3. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. — СПб: БХВПетербург, 2002. — 608 с.