МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»

ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра компьютерной инженерии и моделирования

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

**«СРЕДСТВА СИНХРОНИЗАЦИИ ПОТОКОВ»**

Лабораторная работа

по дисциплине

«Параллельные и распределенные вычисления»

студента 4 курса группы ПИ-б-о-182(2)

Змитрович Никита Сергеевич

направления подготовки 09.03.04 «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| Научный руководитель  старший преподаватель кафедры компьютерной инженерии и моделирования | \_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) | Чабанов В.В. |

Симферополь, 2021

**Цель:**

1. Убедиться, что, в случае совместного использования ресурс несколькими потоками необходима синхронизация
2. Изучить средства синхронизации потоков на примере ключевого слова synchronized;
3. Реализовать приложение выполняющее многопоточные вычисления;

**Постановка задачи:**

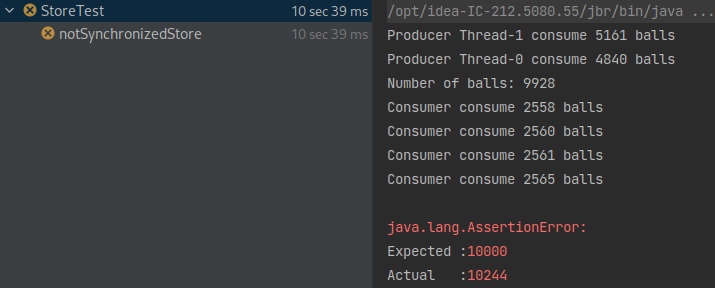
Реализуйте приложение, имитирующее добавление и изъятие шаров в/из корзины несколькими потоками. В начальный момент времени все шары хранятся на складе их общее количество задано и равно 10 000. Два потока (далее поставщики) берут со склада шары и добавляют их в корзину, три других потока (далее потребители) забирают шары из корзины. Каждый поставщик, за один раз, может взять со склада произвольное число шаров в диапазоне [0...100], при этом их число на складе уменьшается соответственно. Каждый потребитель, за один раз, берёт из корзины только один шар. После того, как на складе и в корзине больше не останется шаров, на экране должно отобразиться количество шаров, изъятых каждым потребителем и их суммарное значение.

# Выполнение работы

## Задание 1.

Данная лабораторная была выполнена при помощи встроенного ключевого слова synchronized доступного на платформе JVM. Его суть состоит в том, что JVM использует монитор, известный также как внутренняя блокировка для предоставления синхронизации. Данный монитор прикреплён к объекту, таким образом доступ к данному объекту имеет лишь один поток в один и тот же момент.

Код программы представлен ниже в приложении.

Рис 1 - Результат работы программы без синхронизации

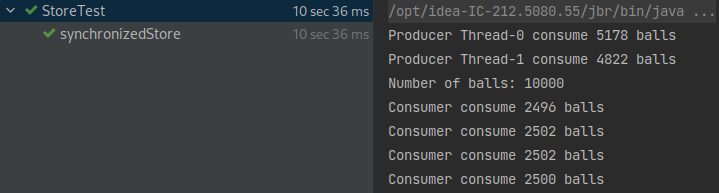


Рис 2 — Результат работы с синхронизацией

**Вывод:**

Изучили базовые средства синхронизации потоков на платформе JVM. Сравнили точность выполнения вычислений при условии использования средств синхронизации потоков и без.

# Приложение

public class SynchronizedStore implements Store { @Volatile int numberOfBalls = 10\_000; @Override public synchronized int getNumberOfBalls() { return numberOfBalls; } @Override public synchronized void consumeBalls(int consumingBalls) { if (consumingBalls <= numberOfBalls) { numberOfBalls -= consumingBalls; } else { throw new IllegalArgumentException(consumingBalls + ">" + numberOfBalls); } } @Override public synchronized void setNumberOfBalls(int numberOfBalls) { this.numberOfBalls = numberOfBalls; } @Override public synchronized void produceBalls(int producingBalls) { numberOfBalls += producingBalls; }}

class NotSynchronizedStore( override var numberOfBalls: Int = 10\_000) : Store { override fun consumeBalls(consumingBalls: Int) { if (consumingBalls <= numberOfBalls) { numberOfBalls -= consumingBalls } } override fun produceBalls(producingBalls: Int) { numberOfBalls += producingBalls }}

class Consumer( private val store: Store) : Thread() { var internalCounter = 0 override fun run() { while (store.numberOfBalls > 0) { store.consumeBalls(1) internalCounter++; sleep(1L) } *println*("Consumer consume $internalCounter balls") }}

class Producer( private val firstStore: Store, private val secondStore: Store) : Thread() { var internalCounter = 0 override fun run() { while (firstStore.numberOfBalls > 0) { val consumingBalls = *min*(Random.nextInt(1, 100), firstStore.numberOfBalls) firstStore.consumeBalls(consumingBalls) internalCounter += consumingBalls secondStore.produceBalls(consumingBalls) sleep(1L) } *println*("Producer ${this.*name*} consume $internalCounter balls") }