**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное учреждение высшего образования

**«Московский технический университет связи и информатики»**

**ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

Кафедра «МКиИТ»

**Информационные технологии и**

**программирование**

**Лабораторная работа по ИТП №7**

Выполнил: студент группы БВТ2205

Белов Егор Олегович

**Задание 1:**

import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.Semaphore;  
import java.util.concurrent.TimeUnit;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class firstExercise {  
 public static void main(String[] args) throws InterruptedException {  
 int[] array = new int[] {2, 4, 1, 5, 19, 1, 5, 1, 2, 5};  
  
 int[] threadsQuantity = *findQuantity*(array.length, 1);  
  
 ExecutorService executorService = Executors.*newFixedThreadPool*(threadsQuantity[1]);  
  
 AtomicInteger startIndex = new AtomicInteger(0);  
 AtomicInteger endIndex = new AtomicInteger(threadsQuantity[0]);  
 AtomicInteger result = new AtomicInteger(0);  
  
 Semaphore semaphore = new Semaphore(1);  
  
 for (int i = 0; i < threadsQuantity[1]; i++) {  
 executorService.execute(() -> {  
 try {  
 semaphore.acquire();  
  
 int indexStart = startIndex.get();  
 int indexEnd = endIndex.get();  
  
 startIndex.set(indexEnd);  
 endIndex.addAndGet(threadsQuantity[0]);  
  
 for (int k = indexStart; k < indexEnd; k++) {  
 result.addAndGet(array[k]);  
 }  
 } catch (InterruptedException e) {  
 System.*out*.println(e.getMessage());  
 } finally {  
 semaphore.release();  
 }  
 });  
 }  
  
 executorService.shutdown();  
 executorService.awaitTermination(Long.*MAX\_VALUE*, TimeUnit.*NANOSECONDS*);  
  
 System.*out*.println(result.get());  
 }  
  
 public static int[] findQuantity(int array, int threads) {  
 if (array % 2 != 0) {  
 return new int[] {array, threads};  
 }  
  
 return *findQuantity*(array / 2, threads \* 2);  
 }  
}

Создается массив array, содержащий числа.

Вызывается метод findQuantity, который определяет количество потоков для распараллеливания вычислений. Этот метод рекурсивно делит размер массива пополам, пока не достигнет максимального количества потоков.

Создается пул потоков executorService с использованием фиксированного количества потоков, рассчитанного на предыдущем шаге.

Создаются атомарные переменные startIndex, endIndex и result. Они используются для отслеживания текущих индексов в массиве и результата суммирования.

Создается семафор semaphore с изначальным количеством разрешений равным 1. Он используется для синхронизации доступа к общим ресурсам.

В цикле создаются и запускаются потоки, которые выполняют следующие действия: a. Поток захватывает семафор, чтобы получить доступ к общим ресурсам. b. Получает текущие значения startIndex и endIndex. c. Обновляет startIndex и endIndex для следующей итерации. d. Вычисляет сумму элементов массива от startIndex до endIndex. e. Результат суммирования добавляется к переменной result. f. Поток освобождает семафор.

После создания всех потоков вызывается метод shutdown для завершения пула потоков после завершения всех задач.

Вызывается метод awaitTermination, чтобы дождаться завершения выполнения всех потоков (используется значение Long.MAX\_VALUE для ожидания).

Выводится значение суммы из переменной result.

**Задание 2:**

import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class secondExercise {  
 public static void main(String[] args) throws InterruptedException {  
 int[][] array = new int[][] {{5, 4, 2}, {8}};  
  
 int[] threadsQuantity = *findQuantityValue*(array.length, 1);  
  
 ExecutorService executorService = Executors.*newFixedThreadPool*(threadsQuantity[1]);  
  
 int[] result = new int[threadsQuantity[1]];  
  
 for (int i = 0; i < threadsQuantity[1]; i++) {  
 int validI = i;  
  
 executorService.execute(() -> {  
 int calculation = 0;  
  
 for (int k = validI \* threadsQuantity[0]; k < (validI + 1) \* threadsQuantity[0]; k++) {  
 for (int j = 0; j < array[k].length; j++) {  
 calculation += array[k][j];  
 }  
 }  
  
 result[validI] = calculation;  
 });  
 }  
  
 executorService.shutdown();  
 executorService.awaitTermination(Long.*MAX\_VALUE*, TimeUnit.*NANOSECONDS*);  
  
 if (result.length == 1) {  
 System.*out*.println(result[0]);  
 return;  
 }  
  
 int itog = 0;  
  
 for (int numbers : result) {  
 itog += numbers;  
 }  
  
 System.*out*.println(itog);  
 }  
  
 public static int[] findQuantityValue(int array, int threads) {  
 if (array % 2 != 0) {  
 return new int[] {array, threads};  
 }  
  
 return *findQuantityValue*(array / 2, threads \* 2);  
 }  
}

Создается двумерный массив array, содержащий числовые значения.

Вызывается метод findQuantityValue, который определяет количество потоков для распараллеливания вычислений. Аналогично предыдущему коду, этот метод также рекурсивно делит размер массива пополам, пока не достигнет максимального количества потоков.

Создается пул потоков executorService с использованием фиксированного количества потоков, рассчитанного на предыдущем шаге.

Создается массив result размером, соответствующим количеству потоков, для сохранения результатов суммирования.

В цикле создаются и запускаются потоки, которые выполняют следующие действия: a. В потоке задается validI, чтобы сохранить правильное значение индекса потока. b. Для каждого потока вычисляются диапазоны индексов в массиве array для обработки. c. Вычисляется сумма элементов в заданном диапазоне и сохраняется в соответствующем элементе массива result.

После создания всех потоков вызывается метод shutdown для завершения пула потоков после завершения всех задач.

Вызывается метод awaitTermination, чтобы дождаться завершения выполнения всех потоков (используется значение Long.MAX\_VALUE для ожидания).

Если длина массива result равна 1, это означает, что был использован только один поток, и его результат просто выводится.

Иначе, выполняется добавление всех элементов массива result для получения общей суммы, которая выводится на экран.

**Задание 3:**

import java.util.ArrayList;  
import java.util.concurrent.\*;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class thirdExercise {  
 public static void main(String[] args) {  
 // Думаю ничего не будет, если я поменяю килограммы на тонны :3  
 Good good1 = new Good("Engine", 55, 1);  
 Good good2 = new Good("Fuel", 25, 1);  
 Good good3 = new Good("Radar", 20, 1);  
 Good good4 = new Good("Bullets", 40, 1);  
 Good good5 = new Good("Carbon-plates", 65, 1);  
 Good good6 = new Good("Tracks", 80, 1);  
 Good good7 = new Good("Optical sight", 5, 1);  
 Good good8 = new Good("Tracks", 80, 1);  
 Good good9 = new Good("Electron-optical complex of active protection", 5, 1);  
  
 Good[] warehouse1 = new Good[] {good1, good2, good3, good4, good5, good6, good7, good8, good9};  
 ArrayList<Good> warehouse2 = new ArrayList<>();  
  
 LoaderRealization loaderRealization = new LoaderRealization(warehouse1, warehouse2);  
  
 loaderRealization.run();  
 }  
  
  
 public interface Loader {  
 Good getGood();  
 }  
  
  
 public static class LoaderRealization implements Loader {  
 private ExecutorService executorService;  
 private Good[] warehouse1;  
 private ArrayList<Good> warehouse2;  
 private int index = 0;  
  
  
 public LoaderRealization(Good[] warehouse1, ArrayList<Good> warehouse2) {  
 this.warehouse1 = warehouse1;  
 this.warehouse2 = warehouse2;  
 }  
  
 public void run() {  
 executorService = Executors.*newFixedThreadPool*(3);  
  
 Semaphore semaphore = new Semaphore(1);  
  
 AtomicInteger resultWeight = new AtomicInteger(0);  
  
 for (int i = 0; i < 3; i++) {  
 executorService.execute(() -> {  
 try {  
 semaphore.acquire();  
  
 for (int j = 0; j < warehouse1.length; j++) {  
 Good good = getGood();  
  
 if (good == null) {  
 break;  
 }  
  
 if (resultWeight.get() + good.weight <= 150) {  
 warehouse2.add(good);  
 good.downgradeQuantity();  
  
 resultWeight.addAndGet(good.weight);  
 }  
 }  
 } catch (InterruptedException e) {  
 System.*out*.println(e.getMessage());  
 } finally {  
 semaphore.release();  
 }  
 });  
 }  
  
 executorService.shutdown();  
 try {  
 executorService.awaitTermination(Long.*MAX\_VALUE*, TimeUnit.*NANOSECONDS*);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
  
 for (Good good : warehouse2) {  
 System.*out*.println(good.name);  
 }  
  
 System.*out*.println("\n" + resultWeight.get());  
 }  
  
 @Override  
 public synchronized Good getGood() {  
 if (warehouse1[index].quantity > 0) {  
 Good good = warehouse1[index];  
 good.downgradeQuantity();  
 return good;  
 }  
  
 if (index + 1 < warehouse1.length) {  
 index += 1;  
 return warehouse1[index];  
 }  
  
 return null;  
 }  
 }  
  
  
 public static class Good {  
 String name;  
 int weight;  
 int quantity;  
  
 public Good(String name, int weight, int quantity) {  
 this.name = name;  
 this.weight = weight;  
 this.quantity = quantity;  
 }  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public void downgradeQuantity() {  
 quantity -= 1;  
 }  
 }  
}

Создаются объекты класса Good, представляющие товары, с их названиями, весом и начальным количеством.

Создается массив warehouse1, который содержит объекты **Good**, представляющие товары на складе 1.

Создается ArrayList warehouse2, который будет использоваться в качестве склада 2.

Создается экземпляр класса LoaderRealization, который принимает warehouse1 и warehouse2 в качестве аргументов конструктора.

Вызывается метод run() у loaderRealization.

Внутри класса LoaderRealization:

Создается пул потоков executorService с использованием фиксированного количества потоков (3).

Создается объект Semaphore с начальным значением 1. Семафор используется для ограничения доступа к критическому участку кода одновременно несколькими потоками.

Создается объект AtomicInteger с начальным значением 0, который будет использоваться для хранения общего веса товаров на складе 2.

В цикле создаются и запускаются три потока, каждый из которых выполняет следующие действия: a. Поток запрашивает доступ к семафору методом semaphore.acquire(), чтобы получить разрешение на доступ к критическому участку кода. b. Поток обходит все элементы массива warehouse1 и получает товары из метода getGood(). c. Если товар доступен, его вес сравнивается с общим весом (хранящимся в объекте AtomicInteger resultWeight). Если сумма веса и веса товара не превышает 150, товар добавляется в warehouse2, его количество уменьшается на 1, а вес товара добавляется к общему весу. d. По завершении обработки товаров поток освобождает семафор, вызывая semaphore.release().

После создания всех потоков вызывается executorService.shutdown() для завершения пула потоков после завершения всех задач.

Затем вызывается executorService.awaitTermination(Long.MAX\_VALUE, TimeUnit.NANOSECONDS), чтобы дождаться завершения выполнения всех потоков.

В цикле for-each выводятся названия товаров на складе 2.

Выводится общий вес товаров, который хранится в resultWeight.