|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ и системы  
 управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных**

**Отчет**

**по лабораторной работе №8**

**Название:** Потоки

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | И.Л. Баришпол |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**Задания**

2. Реализовать многопоточное приложение “Робот”. Надо написать робота, который умеет ходить. За движение каждой его ноги отвечает отдельный поток. Шаг выражается в выводе в консоль LEFT или RIGHT.

package lab8

import java.util.concurrent.Semaphore

/\*\*\*

\* 2. Реализовать многопоточное приложение “Робот”. Надо написать робота, который умеет ходить.

\* За движение каждой его ноги отвечает отдельный поток. Шаг выражается в выводе в консоль LEFT или RIGHT.

\*/

class Robot {

private val leftLeg = Semaphore(1)

private val rightLeg = Semaphore(0)

@Throws(InterruptedException::class)

fun walk() {

val leftLegThread = Thread {

try {

while (true) {

leftLeg.acquire()

println("LEFT")

rightLeg.release()

}

} catch (e: InterruptedException) {

Thread.currentThread().interrupt()

}

}

val rightLegThread = Thread {

try {

while (true) {

rightLeg.acquire()

println("RIGHT")

leftLeg.release()

}

} catch (e: InterruptedException) {

Thread.currentThread().interrupt()

}

}

leftLegThread.start()

rightLegThread.start()

leftLegThread.join()

rightLegThread.join()

}

}

fun main() {

val r = Robot()

r.walk()

}

3. Реализовать многопоточное приложение “Магазин”. Вся цепочка: производитель-магазин-покупатель. Пока производитель не поставит на склад продукт, покупатель не может его забрать. Реализовать приход товара от производителя в магазин случайным числом. В том случае, если товара в магазине не хватает– вывести сообщение.

package lab8

/\*\*\*

\* 3. Реализовать многопоточное приложение “Магазин”. Вся цепочка: производитель-магазин-покупатель.

\* Пока производитель не поставит на склад продукт, покупатель не может его забрать.

\* Реализовать приход товара от производителя в магазин случайным числом.

\* В том случае, если товара в магазине не хватает – вывести сообщение.

\*/

import java.util.\*

import java.util.concurrent.locks.ReentrantLock

class Store(private val capacity: Int) {

private val inventory = HashMap<String, Int>()

private val lock = ReentrantLock()

fun addProduct(name: String, quantity: Int) {

lock.lock()

try {

inventory[name] = inventory.getOrDefault(name, 0) + quantity

println("Added $quantity $name to inventory")

} finally {

lock.unlock()

}

}

fun removeProduct(name: String, quantity: Int): Boolean {

lock.lock()

return try {

if (inventory.getOrDefault(name, 0) >= quantity) {

inventory[name] = inventory[name]!! - quantity

println("Removed $quantity $name from inventory")

true

} else {

println("Not enough $name in inventory")

false

}

} finally {

lock.unlock()

}

}

fun checkInventory(name: String): Int {

lock.lock()

try {

return inventory.getOrDefault(name, 0)

} finally {

lock.unlock()

}

}

}

class Manufacturer(private val store: Store) : Thread() {

override fun run() {

while (true) {

val random = Random().nextInt(10) + 1 // Generate a random number between 1 and 10

sleep(random \* 1000L) // Sleep for that number of seconds

store.addProduct("Product", random) // Add the product to the store

}

}

}

class Shop(private val store: Store) : Thread() {

override fun run() {

while (true) {

sleep(5000L) // Sleep for 5 seconds

if (!store.removeProduct("Product", 1)) { // Attempt to remove one product from the store

println("Cannot pick up product - not enough in inventory")

}

}

}

}

class Buyer(private val store: Store) : Thread() {

override fun run() {

while (true) {

sleep(2000L) // Sleep for 2 seconds

if (store.removeProduct("Product", 1)) { // Attempt to remove one product from the store

println("Bought 1 product")

} else {

println("Cannot buy product - not enough in inventory")

}

}

}

}

fun main() {

val store = Store(10) // Create a store with capacity for 10 products

val manufacturer = Manufacturer(store)

val shop = Shop(store)

val buyer = Buyer(store)

manufacturer.start()

shop.start()

buyer.start()

}

**Вывод:** в ходе выполнения данной лабораторной работы были освоены основы работы с потоками в языке программирования Kotlin.