**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Кафедра інженерії програмного забезпечення**



Лабораторна робота №1.3

з дисципліни: «Конструювання Програмного Забезпечення»

на тему:

«**Дослідження роботи багатопотокових застосунків**»

Виконав: студент ПІ-321

Потапенко О.В.

Перевірили: Безкоровайна Ю.М.

Київ 2020

**ЗМІСТ**

1. **Теоретичні відомості** ………………….……………………………………...3
2. **Лістинг проекту** ………………………………………..…………….………..3
   1. Репозиторій проекту………………………………………………….…….3
   2. Лістинг проекту та коментарі до класів…………….….………….………3
3. **Результат роботи програми**…………………………………..…..…………..7
   1. Результат роботи програми в консолі……………………….……….…….7
   2. Результат роботи програми у файлі………………………….…….………10
4. **Посилання**………………………………………………………….…….……..10

**ДОСЛІДЖЕННЯ РОБОТИ БАГАТОПОТОКОВИХ ЗАСТОСУНКІВ**

## Теоретичні відомості

Java підтримує однопотокові, а також багатопотокові операції. Однопотокова програма має єдину точку входу (main() метод) і єдину точку виходу. Багатопотокова програма має початкову точку входу (метод main ()), за якою слідує безліч точок входу та виходу, які запускаються одночасно з main (). Термін "паралельність" означає виконання одночасно декількох потоків.

Java має вбудовану підтримку паралельного програмування шляхом одночасного запуску декількох потоків в рамках однієї програми. Протягом життєвого циклу потоку існує лише одна точка виконання. Потік сам по собі не є програмою, оскільки він не може працювати самостійно. Натомість він працює в рамках програми.

В рамках наступного проекту було вирішено класичну задачу багатопотокового програмування “Виробник – споживач” за наступною структурою:

* клас, що виконує роль «Постачальника» (наприклад, Сервіс оплати) – об;єкт класу створює дані, що є спільними з об’єктом класу «Споживач». «Постачальник» створює дані в окремому потоці;
* клас, що виконує роль «Споживача» (наприклад, Сервіс доставки) – об’єкт класу

оброблює дані, що є спільними з об’єктом класу «Постачальник», та зберігає їх у файл.

«Споживач» оброблює дані в окремому потоці;

* клас, об’єкт якого містить та оброблює спільні дані (наприклад, транспортна накладна:

дата, час відвантаження, назва та адреса вантажовідправника, назва та адрес

вантажоодержувача, вантаж (товар, кількість, загальна вага), адрес доставляння), для

чого застосовується синхронізація;

* клас, який створює об’єкти «Постачальника», «Споживача» та один об’єкт класу зі

спільними даними (наприклад, Склад товарів).

* Застосування організує взаємодію між потоками наступним чином: потік «Споживача» не може отримати спільні дані доти вони не будуть створені потоком «Постачальника», а потік «Постачальника» не може створювати нові дані, що є спільними, доти потік «Споживача» не обробить «старі» спільні дані

В ході створення проекту було використано наступні інструменти багатопотокового програмування Java: наслідування класу Thread, використання синхронізації, методів wait(), Thread.sleep(), notify(), notifyAll() та інші

## Лістинг проекту

### 2.1 Репозиторій проекту

Посилання на репозиторій з лістингом класу:

https://github.com/TapeNau/constr1.2/blob/main/ImmutableRGB.java

### 2.2 Лістинг проекту та коментарі до класів

**Goods.java**

Клас, що відповідає за збереження інформації про товар: назва товару, кількість.

public class Goods {  
 private String name;  
 private int amount;  
  
 public Goods(String name, int amount) {  
 this.name = name;  
 this.amount = amount;  
 }  
  
 public String getName(){  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getAmount(){  
 return amount;  
 }  
  
 public void setAmount(int amount) {  
 this.amount = amount;  
 }  
}

**Storage.java**

Клас, який створює об’єксти «Постачальника», «Споживача» та об’єкс класу «Накладна»(Invoice) зі спільними данними.

import java.util.ArrayList;  
  
public class Storage {  
 private final Invoice invoice;  
  
 public ArrayList<Goods> getGoodsInvoice(){  
 return this.invoice.getGoodsList();  
 }  
  
 public void setGoodsInvoice(ArrayList<Goods> list) {  
 this.invoice.setGoodsList(list);  
 }  
  
 public Storage() {  
 this.invoice = new Invoice();  
 }  
  
 public void createProcess(String p1name, String c1name) {  
 Producer p1 = new Producer(invoice, p1name);  
 Consumer c1 = new Consumer(invoice, c1name);  
 p1.start();  
 c1.start();  
 }  
}

**Invoice.java**

Клас – накладна, що містить та оброблює спільні дані: дата та час відвантаження, список товарів. Використовує синхронізацію для обробки спільних даних.

import java.util.ArrayList;  
import java.util.Date;  
  
public class Invoice {  
 private ArrayList <Goods> list;  
 private Goods contents;  
 private boolean available = false;  
 private final Date date = new Date();  
  
 public Date getDate() { return date; }  
  
 public ArrayList <Goods> getGoodsList(){  
 return list;  
 }  
  
 public void setGoodsList(ArrayList <Goods> list) {  
 this.list = list;  
 }  
  
 public synchronized void put(Goods value) {  
 while (available) {  
 try {  
 wait();  
 }  
 catch (InterruptedException e) {  
 }  
 }  
 contents = value;  
 available = true;  
  
 //затримка для корректної синхронізації  
 try {  
 Thread.*sleep*(50);  
 }catch (InterruptedException e) {}  
  
 notifyAll();  
 }  
  
 public synchronized Goods get() {  
 while (!available) {  
 try {  
 wait();  
 }  
 catch (InterruptedException e) {  
 }  
 }  
 available = false;  
 //затримка для корректної синхронізації  
 try {  
 Thread.*sleep*(50);  
 }catch (InterruptedException e) {}  
  
 notifyAll();  
 return contents;  
 }  
}

**Producer.java**

Клас, що виконує роль постачальника. Створює дані, що є спільними з об’єктом класу Споживача в окремому потоці. Наслідує клас Thread та містить назву організації – постачальника.

import java.lang.Thread;  
  
class Producer extends Thread {  
 private Invoice invoice;  
 private String name;  
  
 public Producer(Invoice c, String name) {  
 invoice = c;  
 this.name = name;  
 }  
  
 public void run() {  
 for (int i = 0; i < invoice.getGoodsList().size(); i++) {  
 Goods temp = invoice.getGoodsList().get(i);  
 invoice.put(temp);  
 System.*out*.println(invoice.getDate().toString() + " --- Поставщик " + this.name + " привіз товар: " + temp.getName() + " в кількості: " + temp.getAmount());  
 try {  
 *sleep*((int)(Math.*random*() \* 100));  
 } catch (InterruptedException e) { }  
 }  
 }  
}

**Consumer.java**

Клас, що виконує роль споживача. Створює дані, що є спільними з об’єктом класу Постачальника в окремому потоці, а також зберігає дані у файл. Наслідує клас Thread та містить назву організації – споживача, реалізує серіалізацію інформації у файл.

import java.io.\*;  
  
class Consumer extends Thread {  
 private Invoice invoice;  
 private String name;  
  
 public Consumer(Invoice c, String name) {  
 invoice = c;  
 this.name = name;  
 }  
  
 public void run() {  
 for (int i = 0; i < invoice.getGoodsList().size(); i++) {  
 Goods value = invoice.get();  
 String text = invoice.getDate().toString() + " --- Покупець " + this.name + " забрав товар: " + value.getName() + " в кількості: " + value.getAmount();  
 this.Serialize(text);  
 System.*out*.println(text);  
 }  
 this.Serialize("---");  
 }  
  
 public void Serialize (String text) {  
 try(FileWriter fw = new FileWriter("myfile.txt", true);  
 BufferedWriter bw = new BufferedWriter(fw);  
 PrintWriter out = new PrintWriter(bw)) {  
 out.println(text);  
 } catch (IOException e) { }  
 }  
}

**MultiThreading.java**

Головна точка входу програми, містить дані для 5 спроб запуску.

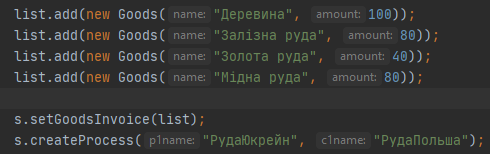
import java.util.ArrayList;  
  
public class MultiThreading {  
 public static void main(String[] args)  
 {  
 ArrayList<Goods> list = new ArrayList<Goods>();  
 Storage s = new Storage();  
  
// list.add(new Goods("Деревина", 100));  
// list.add(new Goods("Залізна руда", 80));  
// list.add(new Goods("Золота руда", 40));  
// list.add(new Goods("Мідна руда", 80));  
//  
// s.setGoodsInvoice(list);  
// s.createProcess("РудаЮкрейн", "РудаПольша");  
  
// list.add(new Goods("Двері", 20));  
// list.add(new Goods("Вікна", 25));  
// list.add(new Goods("Стільці", 25));  
// list.add(new Goods("Столи", 10));  
// list.add(new Goods("Шафи", 12));  
//  
// s.setGoodsInvoice(list);  
// s.createProcess("МебліЮкрейн", "МебліПольша");  
  
// list.add(new Goods("Телевізори", 18));  
// list.add(new Goods("Проектори", 70));  
// list.add(new Goods("Програвачі", 55));  
//  
// s.setGoodsInvoice(list);  
// s.createProcess("ТехнікаЮкрейн", "ТехнікаПольша");  
  
// list.add(new Goods("Ковбаса", 500));  
// list.add(new Goods("Сири", 700));  
// list.add(new Goods("Вино", 220));  
//  
// s.setGoodsInvoice(list);  
// s.createProcess("Продукти Київ", "Продукти Біла Церква");  
  
 list.add(new Goods("Кільця", 112));  
 list.add(new Goods("Медальйони", 227));  
 list.add(new Goods("Браслети", 79));  
 list.add(new Goods("Намисто", 140));  
  
 s.setGoodsInvoice(list);  
 s.createProcess("Коштовності Ужгород", "Коштовності Київ");  
 }  
}

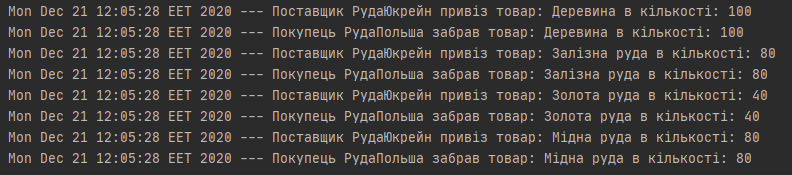
## Результат роботи програми

### 3.1 Результат роботи програми в консолі

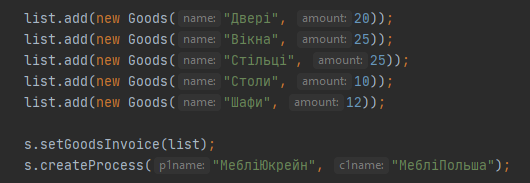


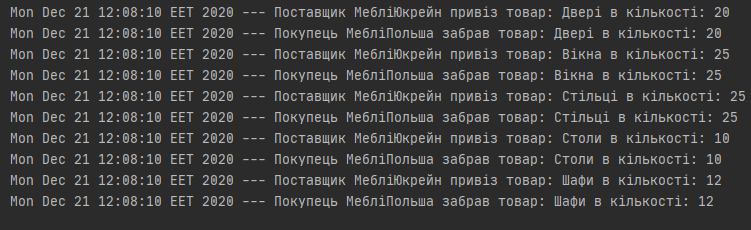
**1 спроба**



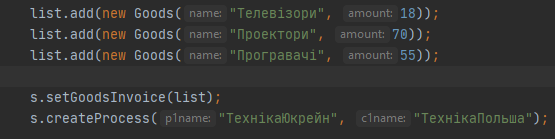


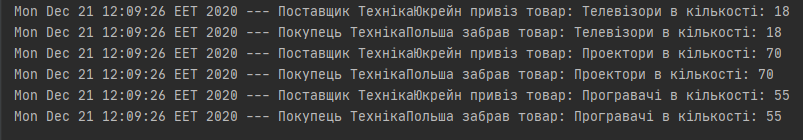
1. **спроба**



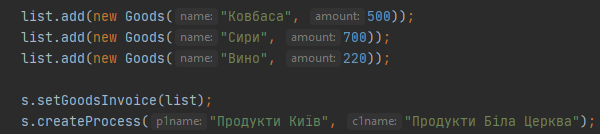


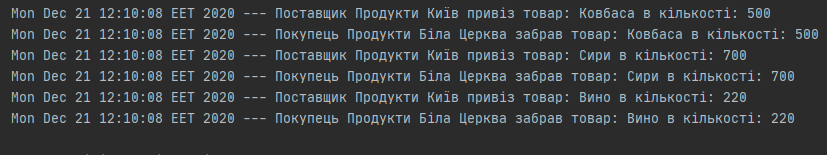
**3 спроба**



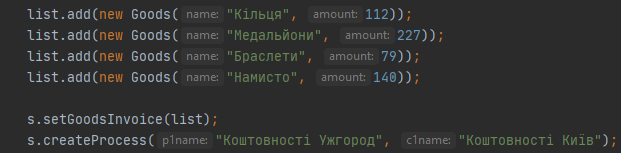


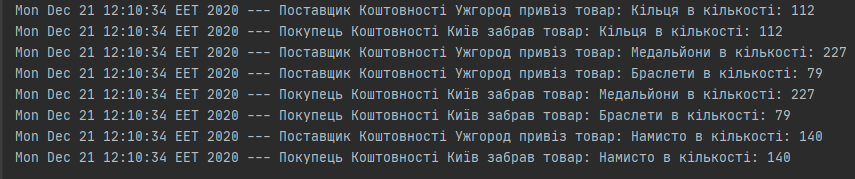
**4 спроба**



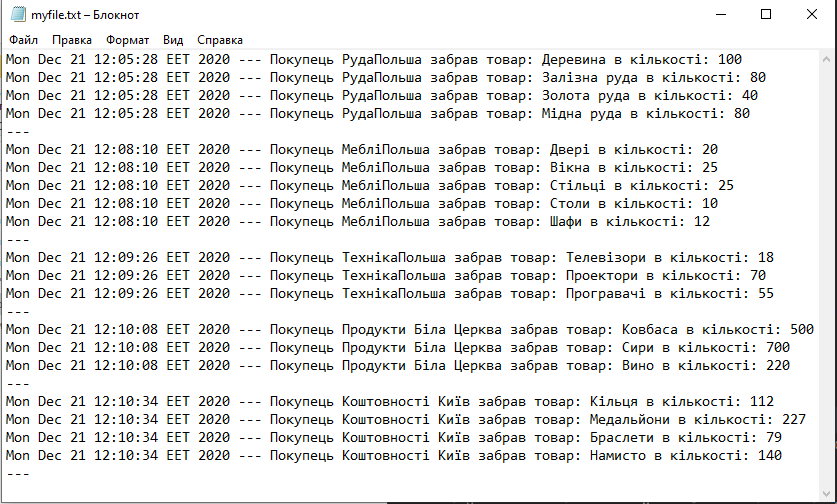


**5 спроба**





### 3.2 Результат роботи програми у файлі



## Посилання

Власний проект:

<https://github.com/TapeNau/constr1.2/blob/main/ImmutableRGB.java>

Конвенції щодо стилю програмування Java:

<https://www.magnumblog.space/java/131-translating-java-code-conventions>

Офіційні конвенції від Oracle:

<https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

Клас, який було досліджено:

<https://github.com/boredgus/SD-lw1.2/blob/master/src/Student.java>