|  |  |
| --- | --- |
| 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 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 |  |  | Р.В. Баканов | |
|  | (Группа) |  | (Подпись, дата) | | (И.О. Фамилия) |
|  |  |  |  | |  |
| Преподаватель |  |  |  | | П.В. Степанов |
|  |  |  | (Подпись, дата) | | (И.О. Фамилия) |

Москва, 2024

**Цель работы:** освоить базовые принципы работы с потоками в Java.

**Вариант: 2.**

**Задание 1:** 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.

Код решения приведен в листингах 1 и 2.

Листинг 1 — Задание 1 (1)

public class Main {  
 public static void main(String[] args) throws Exception {  
 var robot = new Robot();  
  
 robot.walk();  
  
 Thread.*sleep*(5000);  
  
 robot.stopWalking();  
 }  
}

Листинг 2 – Задание 1 (2)

import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class Robot {  
 private final Leg \_leftLeg = new Leg("LEFT");  
 private final Leg \_rightLeg = new Leg("RIGHT");  
 private ExecutorService \_walkExecutorService = null;  
  
 public void walk() throws Exception {  
 \_walkExecutorService = Executors.*newFixedThreadPool*(2);  
  
 \_walkExecutorService.submit(\_leftLeg);  
 Thread.*sleep*(\_leftLeg.getActionExecutionTimeMs());  
 \_walkExecutorService.submit(\_rightLeg);  
 }  
  
 public void stopWalking() {  
 \_walkExecutorService.shutdownNow();  
 }  
  
 public static class Leg implements Callable<Object> {  
 private final String \_completionMessage;  
 private final long \_actionExecutionTimeMs;  
 private final long \_idleTimeMs;  
  
 public Leg(String completionMessage) {  
 \_completionMessage = completionMessage;  
 \_actionExecutionTimeMs = 500;  
 \_idleTimeMs = 500;  
 }  
  
 public long getActionExecutionTimeMs() {  
 return \_actionExecutionTimeMs;  
 }  
  
 @Override  
 public Object call() throws Exception {  
 move();  
  
 return null;  
 }  
  
 private void move() throws Exception {  
 while (true) {  
 Thread.*sleep*(\_actionExecutionTimeMs);  
  
 System.*out*.println(\_completionMessage);  
  
 Thread.*sleep*(\_idleTimeMs);  
 }  
 }  
 }  
}

**Задание 2:** 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.

Код решения приведен в листингах 3 и 4.

Листинг 3 — Задание 2 (1)

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 var store = new Store();  
 Scanner sc = new Scanner(System.*in*);  
  
 while (true) {  
 var rawCommand = sc.nextLine();  
 var command = *parseCommand*(rawCommand);  
  
 if (command == null) {  
 System.*out*.println("Invalid command");  
  
 continue;  
 }  
  
 command.execute(store);  
 }  
 }  
  
 private static ICommand parseCommand(String input) {  
 var inputElements = input.split(" ");  
  
 if (inputElements.length < 2) {  
 return null;  
 }  
  
 return switch (inputElements[0]) {  
 case "p" -> *parsePlaceCommand*(inputElements);  
 case "b" -> *parsePurchaseCommand*(inputElements);  
 default -> null;  
 };  
 }  
  
 private static PlaceCommand parsePlaceCommand(String[] inputElements) {  
 try {  
 var placingTimeInSeconds = Integer.*parseInt*(inputElements[1]);  
  
 return new PlaceCommand(placingTimeInSeconds \* 1000L);  
 } catch (NumberFormatException e) {  
 return null;  
 }  
 }  
  
 private static PurchaseCommand parsePurchaseCommand(String[] inputElements) {  
 try {  
 var itemId = Integer.*parseInt*(inputElements[1]);  
  
 return new PurchaseCommand(itemId);  
 } catch (NumberFormatException e) {  
 return null;  
 }  
 }  
  
 public interface ICommand {  
 void execute(Store store);  
 }  
  
 public static class PlaceCommand implements ICommand {  
 private final long \_placingTimeMs;  
  
 public PlaceCommand(long placingTimeMs) {  
 \_placingTimeMs = placingTimeMs;  
 }  
  
 @Override  
 public void execute(Store store) {  
 store.placeItem(\_placingTimeMs);  
 }  
 }  
  
 public static class PurchaseCommand implements ICommand {  
 private final int \_itemId;  
  
 public PurchaseCommand(int itemId) {  
 \_itemId = itemId;  
 }  
  
 @Override  
 public void execute(Store store) {  
 store.purchaseItem(\_itemId);  
 }  
 }  
}

Листинг 4 – Задание 2 (2)

import java.util.Date;  
import java.util.HashMap;  
import java.util.concurrent.Callable;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class Store {  
 private int \_itemIdSequence = 1;  
  
 private final ExecutorService \_storeExecutorService = Executors.*newFixedThreadPool*(3);  
 private final HashMap<Integer, Item> \_items = new HashMap<>();  
  
 public void placeItem(long placingTimeMs) {  
 var item = new Item(\_itemIdSequence++, placingTimeMs);  
 \_items.put(item.\_id, item);  
  
 \_storeExecutorService.submit(item);  
 }  
  
 public void purchaseItem(int itemId) {  
 var item = \_items.get(itemId);  
  
 item.purchase();  
 }  
  
 private class Item implements Callable<Object> {  
 private final int \_id;  
 private final long \_placingTimeMs;  
 private Date \_expectedPlacingEndDate = null;  
 private boolean \_isPlaced = false;  
 private boolean \_isPurchased = false;  
  
 public Item(int id, long placingTimeMs) {  
 \_id = id;  
 \_placingTimeMs = placingTimeMs;  
 }  
  
 @Override  
 public Object call() throws Exception {  
 place();  
  
 return null;  
 }  
  
 public void purchase() {  
 if (!\_isPlaced) {  
 System.*out*.printf("Item %d is not placed yet. Remaining time (ms): %d\n", \_id, \_expectedPlacingEndDate.getTime() - System.currentTimeMillis());  
  
 return;  
 }  
  
 if (\_isPurchased) {  
 System.out.printf("Item %d is already purchased\n", \_id);  
  
 return;  
 }  
  
 \_isPurchased = true;  
  
 System.out.printf("Item %d is successfully purchased\n", \_id);  
 }  
  
 private void place() throws Exception {  
 System.out.printf("Item %d is being placed\n", \_id);  
  
 \_expectedPlacingEndDate = new Date(System.currentTimeMillis() + \_placingTimeMs);  
  
 Thread.sleep(\_placingTimeMs);  
  
 \_isPlaced = true;  
  
 System.out.printf("Item %d is placed\n", \_id);  
 }  
 }  
}

**Вывод:** в результате выполнения лабораторной работы были разработаны несколько программ в соответствие с требованиями задания.