МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Циклова комісія програмування***

**Робочий план**

з курсу «Java - програмування»

Практична робота №6

Солом’яного Ярослава Сергійовича

*ПІБ студента*

студента групи **2П-18**

|  |  |  |
| --- | --- | --- |
| **Вид роботи** | **Дата** | **Оцінка** |
| *Пром. контроль* |  |  |
| *Залік* |  |  |

Викладач Марченко С. В.

Черкаси-2021

Завдання №1. (Створення та завершення потоків) Напишіть програму, яка створить два потоки, кожен з яких спатиме випадкову кількість секунд, а потім буде виводити повідомлення, яке включатиме ID потоку.

Лістинг 1. Class Test(Ex\_1)

package Ex\_1;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
 public static void main(String[] args) {  
 ThreadOne threadOne = new ThreadOne();  
 ThreadTwo threadTwo = new ThreadTwo();  
 threadOne.start();  
 threadTwo.start();  
 printTimeStamp();  
 }  
 public static void printTimeStamp() {  
 final String sname = "Yaroslav Solomianyi";  
 Date currentDate = new Date();  
 SimpleDateFormat dateFormat = null;  
 dateFormat = new SimpleDateFormat();  
 System.out.println("Made by: " + sname + " on " + dateFormat.format(currentDate));  
 }  
}

Лістинг 2. Class ThreadOne (Ex\_1)

package Ex\_1;  
  
public class ThreadOne extends Thread{  
 public void run()  
 {  
 System.out.println("Поток 1 створено.");  
 try  
 {  
 int a = 0;  
 int b = 10000;  
 long randomTime = (a+(long)(Math.random()\*b));  
 System.out.println("Поток 1 спав: "+randomTime+" Секунд");  
 Thread.sleep(randomTime);  
 }  
 catch (InterruptedException e) {  
 System.out.println("Поток 1 перерваний");  
 }  
 System.out.println("Я поток 1, мій ID:"+getId());  
 System.out.println("Поток 1 завершено.");  
 }  
}

Лістинг 3. Class ThreadTwo(Ex\_1)

package Ex\_1;  
public class ThreadTwo extends Thread {  
 public void run() {  
 System.out.println("Поток 2 створено.");  
 try {  
 int a = 0;  
 int b = 10000;  
 long randomTime = (a+(long)(Math.random()\*b));  
 System.out.println("Поток 2 спав: "+randomTime+" Секунд");  
 Thread.sleep(randomTime);  
  
 } catch (InterruptedException e) {  
 System.out.println("Поток 2 перерваний");  
 }  
 System.out.println("Я поток 2, мій ID:" + getId());  
 System.out.println("Поток 2 завершено.");  
 }  
}

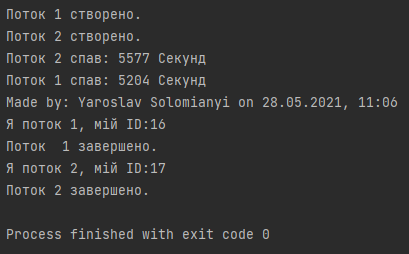


Рис 1. Результат виконання завдання №1.

Завдання №2. Потрібно створити власний клас на базі TimerTask, що дозволятиме виконувати виклик біпера через певні проміжки часу, наприклад, 1 секунду. Даний клас може бути окремим або внутрішнім (допоміжним) класом для класу, що інкапсулює роботу з біпером (запуск таймеру, відтворення звуку та ін.).

Лістинг 4. Class MyTimerTask(ex\_2)

package Ex\_2;  
  
import java.awt.\*;  
import java.util.TimerTask;  
public class MyTimerTask extends TimerTask{  
 @Override  
 public void run() {  
 System.out.println("Я TimerTask, включив звук бипера.");  
 Toolkit.getDefaultToolkit().beep();  
 }  
}

Лістинг 4. Class Test(ex\_2)

package Ex\_2;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public class Test {  
 public static void main(String[] args) {  
 TimerTask timerTask = new MyTimerTask();  
 Timer timer = new Timer(true);  
 timer.schedule(timerTask,0,1000);  
 System.out.println("TimerTask почав своє виконання.");  
 try  
 {  
 Thread.sleep(10000);  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 timer.cancel();  
 System.out.println("TimerTask закончив своє виконання.");  
 printTimeStamp();  
 }  
 public static void printTimeStamp() {  
 final String sname = "Yaroslav Solomianyi";  
 Date currentDate = new Date();  
 SimpleDateFormat dateFormat = null;  
 dateFormat = new SimpleDateFormat();  
 System.out.println("Made by: " + sname + " on " + dateFormat.format(currentDate));  
 }  
}

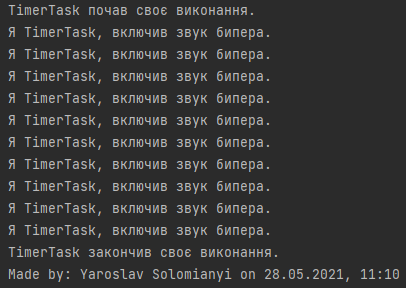


Рис 2. Результат виконання завдання №2.

Завдання №3. Ви реалізуєте контролер для пристрою, доступ до якого відбувається за допомогою наступного інтерфейсу:

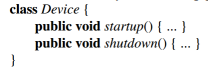


Рис 3.Class Device

Також присутні 2 датчики (нагріву та тиску), які можна використовувати для моніторингу пристрою.

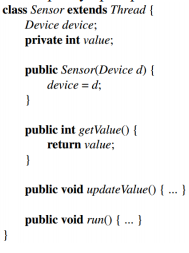


Рис 4.Class Sensor

Напишіть клас Controller, який може опитувати датчики в багатопоточному режимі. Реалізуйте його метод run() так, щоб він запускав пристрій, а потім стежив за новими значеннями датчиків. Контролер відключає пристрій, якщо температурний датчик повертає значення понад 70ºС, а датчик тиску – значення 100. Також завершіть метод run() у класі Sensor, який неперервно викликає updateValue() та сигналізує контролеру, якщо значення змінились.

Лістинг №5.Class Device(ex\_3)

package ex\_3;  
  
public class Device {  
 public void startup(String name){  
 System.*out*.println("Пристрій: | " + name + " | почав працювати");  
 }  
 public void shutdown(String name){  
 System.*out*.println("Пристрій: | " + name + " | закінчив працювати");  
 }  
}

Лістинг №6.Class Sensor(ex\_3)

package ex\_3;  
public class Sensor extends Thread {  
 Device device;  
 private int value;  
 private String name;  
  
 public Sensor(Device d,String name){  
 this.name = name;  
 device = d;  
 }  
 public int getValue() {  
 return value;  
 }  
 public void updateValue(String name){  
 int a = 0;  
 int b = 10;  
 int number = (int)((Math.*random*() \*(b-a))+a);  
 value += number;  
 System.*out*.println("Значення | "+ name + " | змінилось на : "+ number);  
 System.*out*.println(name + " | показує : " + value);  
 }  
  
 @Override  
 public void run()  
 {  
 device.startup(name);  
 while (!isInterrupted()) {  
 try {  
 Thread.*sleep*(3000);  
 updateValue(name);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 device.shutdown(name);  
 }  
}

Лістинг №7.Class Controller(ex\_3)

package ex\_3;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Controller {  
 public static void main(String[] args) {  
 Device Tempery = new Device();  
 Device Presure = new Device();  
 Sensor temperature = new Sensor(Tempery,"Датчик температури");  
 Sensor presure = new Sensor(Presure, "Датчик тиску");  
 temperature.start();  
 presure.start();  
 boolean isActive = true;  
 while (isActive) {  
 try {  
 Thread.*sleep*(1500);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 if (temperature.getValue() > 70) {  
 System.*out*.println("Датчик температури на критичному рівні. Датчик вимкнувся.");  
 temperature.interrupt();  
 presure.interrupt();  
 isActive = false;  
 }  
 if (presure.getValue() > 100) {  
 System.*out*.println("Датчик тиску на критичному рівні. Датчик вимкнувся.");  
 presure.interrupt();  
 temperature.interrupt();  
 isActive = false;  
 }  
 }  
 try {  
 Thread.*sleep*(10000);  
 System.*out*.println("Пристрої вимкнуті. Контролер завершив роботу");  
 *printTimeStamp*();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 public static void printTimeStamp() {  
 final String sname = "Yaroslav Solomianyi";  
 Date currentDate = new Date();  
 SimpleDateFormat dateFormat = null;  
 dateFormat = new SimpleDateFormat();  
 System.*out*.println("Made by: " + sname + " on " + dateFormat.format(currentDate));  
 }  
}

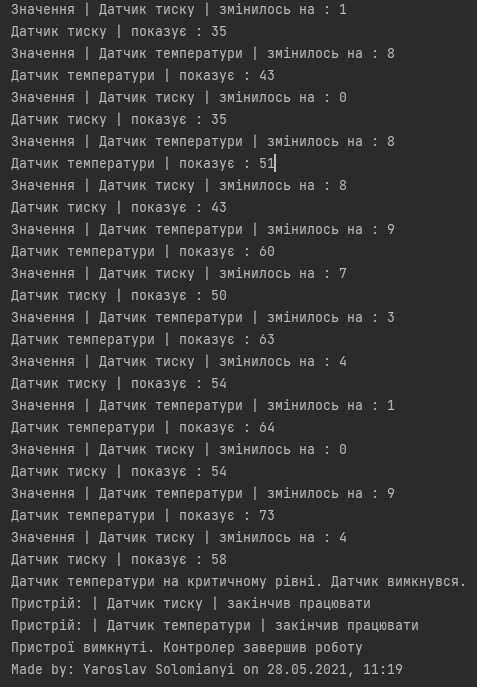


Рис 5. Результат роботи програми №3

Завдання 4. (CountdownLatch) Змоделюйте роботу додатку, який запускає N потоків, що перевірять зовнішні системи та відзвітують про це засуву (latch) в головному, запускаючому класі. Як тільки всі служби верифіковані та перевірені, завантаження додатку продовжиться. Нехай потрібно зібрати звіти для додатку, що перевірятиме показники здоров’я людини. Для цього створіть абстрактний базовий клас, який буде батьківським для усіх перевірок зовнішніх служб (мережі, бази даних, кешу тощо).

Зокрема, перевірка надходження даних з мережі може описуватись класом NetworkHealthChecker:



Рис 6. Class NetworkHealthChecker

Головний клас, який контролює запуск додатку, має ініціалізувати засув та очікувати, поки всі служби не будуть перевірені.

Лістинг №8. Class Test(ex\_4)

package Ex\_4;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
 public static void main(String[] args)  
 {  
 boolean result = false;  
 try {  
 result = ApplicationStartupUtil.checkExternalServices();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 System.out.println("External services validation completed !! Result was : "+ result);  
 printTimeStamp();  
 }  
 public static void printTimeStamp() {  
 final String sname = "Yaroslav Solomianyi";  
 Date currentDate = new Date();  
 SimpleDateFormat dateFormat = null;  
 dateFormat = new SimpleDateFormat();  
 System.out.println("Made by: " + sname + " on " + dateFormat.format(currentDate));  
 }  
}

Лістинг №8. Class BaseHealthChecker(ex\_4)

package Ex\_4;  
import java.util.concurrent.CountDownLatch;  
  
public abstract class BaseHealthChecker implements Runnable {  
  
 private CountDownLatch \_latch;  
 private String \_serviceName;  
 private boolean \_serviceUp;  
  
 public BaseHealthChecker(String serviceName, CountDownLatch latch)  
 {  
 super();  
 this.\_latch = latch;  
 this.\_serviceName = serviceName;  
 this.\_serviceUp = false;  
 }  
  
 @Override  
 public void run() {  
 try {  
 verifyService();  
 \_serviceUp = true;  
 } catch (Throwable t) {  
 t.printStackTrace(System.err);  
 \_serviceUp = false;  
 } finally {  
 if(\_latch != null) {  
 \_latch.countDown();  
 }  
 }  
 }  
  
 public String getServiceName() {  
 return \_serviceName;  
 }  
  
 public boolean isServiceUp() {  
 return \_serviceUp;  
 }  
  
 public abstract void verifyService();  
}

Лістинг №9. Class DataBaseHealthChecker(ex\_4)

package Ex\_4;  
import java.util.concurrent.CountDownLatch;  
  
public class DatabaseHealthChecker extends BaseHealthChecker  
{  
 public DatabaseHealthChecker (CountDownLatch latch)  
 {  
 super("Database Service", latch);  
 }  
  
 @Override  
 public void verifyService()  
 {  
 System.out.println("Checking " + this.getServiceName());  
 try {  
 Thread.sleep(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.out.println(this.getServiceName() + " is UP");  
 }  
}

Лістинг №10. Class CacheBaseHealthChecker(ex\_4)

package Ex\_4;  
import java.util.concurrent.CountDownLatch;  
  
public class CacheHealthChecker extends BaseHealthChecker  
{  
 public CacheHealthChecker (CountDownLatch latch)  
 {  
 super("Cache Service", latch);  
 }  
  
 @Override  
 public void verifyService()  
 {  
 System.out.println("Checking " + this.getServiceName());  
 try {  
 Thread.sleep(5000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.out.println(this.getServiceName() + " is UP");  
 }  
}

Лістинг №11. Class NetworkHealthChecker(ex\_4)

package Ex\_4;  
import java.util.concurrent.CountDownLatch;  
  
public class NetworkHealthChecker extends BaseHealthChecker  
{  
 public NetworkHealthChecker (CountDownLatch latch)  
 {  
 super("Network Service", latch);  
 }  
  
 @Override  
 public void verifyService()  
 {  
 System.out.println("Checking " + this.getServiceName());  
 try  
 {  
 Thread.sleep(7000);  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 System.out.println(this.getServiceName() + " is UP");  
 }  
}

Лістинг №11. Class ApplicationStartUtil(ex\_4)

package Ex\_4;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.CountDownLatch;  
import java.util.concurrent.Executor;  
import java.util.concurrent.Executors;  
  
public class ApplicationStartupUtil  
{  
 private static List<BaseHealthChecker> \_services;  
 private static CountDownLatch \_latch;  
  
 private ApplicationStartupUtil()  
 {  
 }  
  
 private final static ApplicationStartupUtil INSTANCE = new ApplicationStartupUtil();  
  
 public static ApplicationStartupUtil getInstance()  
 {  
 return INSTANCE;  
 }  
  
 public static boolean checkExternalServices() throws Exception  
 {  
 \_latch = new CountDownLatch(3);  
 \_services = new ArrayList<BaseHealthChecker>();  
 \_services.add(new NetworkHealthChecker(\_latch));  
 \_services.add(new CacheHealthChecker(\_latch));  
 \_services.add(new DatabaseHealthChecker(\_latch));  
  
 Executor executor = Executors.newFixedThreadPool(\_services.size());  
  
 for(final BaseHealthChecker v : \_services)  
 {  
 executor.execute(v);  
 }  
  
 \_latch.await();  
  
 for(final BaseHealthChecker v : \_services)  
 {  
 if( ! v.isServiceUp())  
 {  
 return false;  
 }  
 }  
 return true;  
 }  
}

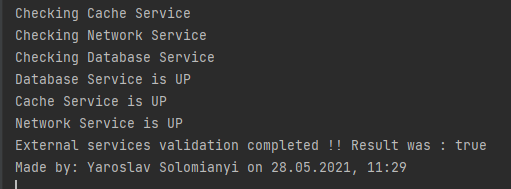


Рис 7. Результат виконання завдання №4.