МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине

Шаблоны проектирования программного обеспечения

ВЫПОЛНИЛИ:

Студенты группы \_\_\_22-ИВТ-2\_\_\_\_\_\_\_\_

(шифр группы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Киселев А. Р.

(подпись) (фамилия, и.,о.)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Назаров А. И.

(подпись) (фамилия, и.,о.)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРОВЕРИЛ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жевнерчук Д. В.

(подпись) (фамилия, и.,о.)

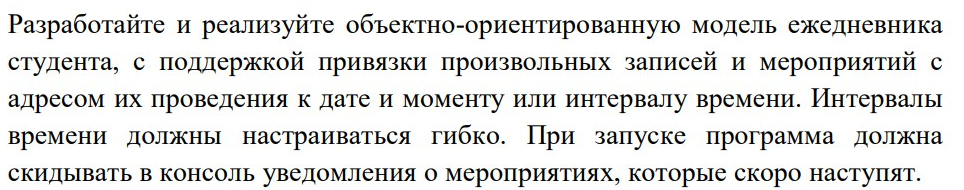
Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Нижний Новгород 2024

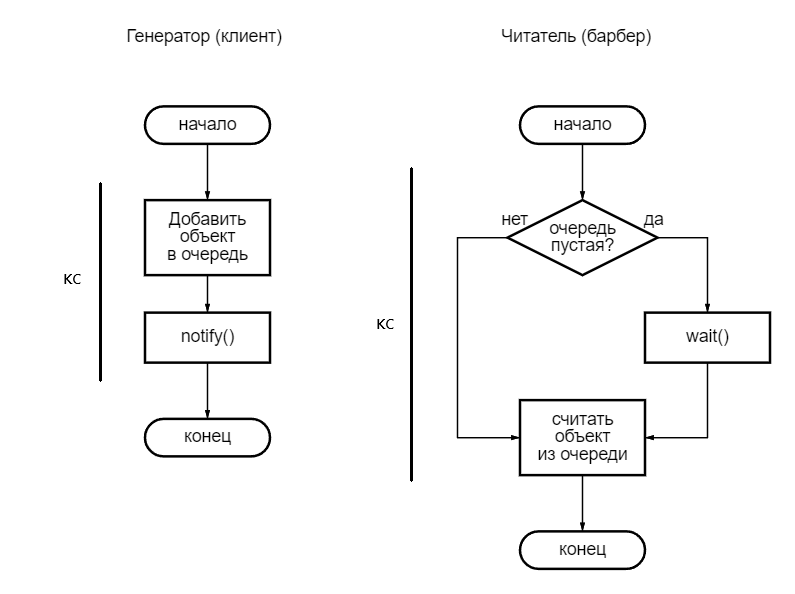
**Задание**

**Вариант 17**



**Описание проекта**

В проекте создается несколько потоков, базовыми классами которых являются: Генератор (Клиент) и Читатель (Барбер). Также можно заметить, что у каждого из потоков есть параметр *rate*, отвечающий за частоту работы каждого цикла, однако, несмотря на это, если Генератор ничего не сгенерировал, то каждый поток Читателя будет вставать в очередь до тех пор, пока для него не появится доступный объект, сгенерированный Генератором.



**Приложение 1**

**Программный код**

src/main/java/org/lab03/Event.java:  
package org.lab03;  
  
import org.springframework.stereotype.Component;  
  
import java.time.LocalDateTime;  
  
@Component  
public class Event {  
 private String name = "unknown";  
 private String place = "unknown";  
 private LocalDateTime dateTime = LocalDateTime.MIN;  
  
 public Event() {  
  
 }  
  
 public Event(String name, String place, LocalDateTime dateTime) {  
 this.name = name;  
 this.place = place;  
 this.dateTime = dateTime;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getPlace() {  
 return place;  
 }  
  
 public void setPlace(String place) {  
 this.place = place;  
 }  
  
 public LocalDateTime getDateTime() {  
 return dateTime;  
 }  
  
 public void setDateTime(LocalDateTime dateTime) {  
 this.dateTime = dateTime;  
 }  
  
 public String toString() {  
 return "Name: " + getName() + "; Place: " + getPlace() + "; DateTime: " + getDateTime();  
 }  
}  
  
  
src/main/java/org/lab03/EventHolder.java:  
package org.lab03;  
  
import java.util.LinkedList;  
  
public class EventHolder {  
 private final LinkedList<Event> queue = new LinkedList<>();  
  
 public synchronized void set(Event event) {  
 queue.push(event);  
 notify();  
 }  
  
 public synchronized Event get() throws InterruptedException {  
 if (queue.isEmpty()) {  
 wait();  
 }  
  
 return queue.poll();  
 }  
}  
  
  
src/main/java/org/lab03/Logger.java:  
package org.lab03;  
  
import org.lab03.iface.ILogger;  
import org.lab03.iface.IPrinter;  
  
import java.time.LocalDateTime;  
import java.util.HashSet;  
  
public class Logger implements ILogger {  
 private String format = "${time}:${level}:${message}";  
 private HashSet<IPrinter> subscribers = new HashSet<>();  
  
 public String getFormat() {  
 return format;  
 }  
  
 public void setFormat(String format) {  
 this.format = format;  
 }  
  
 public void subscribe(IPrinter printer) {  
 subscribers.add(printer);  
 }  
  
 public void unsubscribe(IPrinter printer) {  
 subscribers.remove(printer);  
 }  
  
 @Override  
 public void debug(String message) {  
 log("DEBUG", message);  
 }  
  
 @Override  
 public void info(String message) {  
 log("INFO", message);  
 }  
  
 @Override  
 public void warning(String message) {  
 log("WARNING", message);  
 }  
  
 private void log(String level, String message) {  
 String formatted = getFormat();  
  
 formatted = formatted.replace("${time}", LocalDateTime.now().toString());  
 formatted = formatted.replace("${level}", level);  
 formatted = formatted.replace("${message}", message);  
  
 for (IPrinter printer : subscribers) {  
 printer.handle(formatted);  
 }  
 }  
}  
  
  
src/main/java/org/lab03/Main.java:  
package org.lab03;  
  
import org.lab03.thread.Generator;  
import org.lab03.thread.Reader;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Main {  
 public static void main(String[] args) throws InterruptedException {  
 ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("app.xml");  
  
 Logger logger = ctx.getBean("logger", Logger.class);  
 logger.subscribe(System.out::println);  
  
 Thread gen1 = new Thread(new Generator(ctx, "Generator 1", 10));  
  
 Thread rd1 = new Thread(new Reader(ctx, "Reader 1", 200));  
 Thread rd2 = new Thread(new Reader(ctx, "Reader 2", 200));  
  
 gen1.start();  
  
 rd1.start();  
 rd2.start();  
  
 Thread.sleep(1000);  
  
 gen1.interrupt();  
  
 rd1.interrupt();  
 rd2.interrupt();  
  
 ctx.close();  
 }  
}  
  
src/main/java/org/lab03/Profiler.java:  
package org.lab03;  
  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;  
import org.lab03.iface.ILogger;  
import org.springframework.stereotype.Component;  
  
import java.time.LocalTime;  
  
@Aspect  
@Component  
public class Profiler {  
 private ILogger logger;  
  
 public Profiler(ILogger logger) {  
 this.logger = logger;  
 }  
  
 @Around("execution(\* org.lab03.Event.\*(..))")  
 private Object profileEvent(ProceedingJoinPoint jp) throws Throwable {  
 LocalTime time = LocalTime.now();  
  
 Object r = jp.proceed();  
  
 logger.debug("profileEvent( "  
 + jp.getSignature() + " -> "  
 + (LocalTime.now().toNanoOfDay() - time.toNanoOfDay() + " ns")  
 + ")");  
 return r;  
 }  
}  
  
  
src/main/java/org/lab03/iface/ILogger.java:  
package org.lab03.iface;  
  
public interface ILogger {  
 void debug(String message);  
 void info(String message);  
 void warning(String message);  
}  
  
  
src/main/java/org/lab03/iface/IPrinter.java:  
package org.lab03.iface;  
  
public interface IPrinter {  
 void handle(String message);  
}  
  
  
src/main/java/org/lab03/thread/AbstractThread.java:  
package org.lab03.thread;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
abstract public class AbstractThread implements Runnable {  
 protected final ClassPathXmlApplicationContext ctx;  
 protected final String name;  
 protected final long rate;  
  
 public AbstractThread(ClassPathXmlApplicationContext ctx, String name, long rate) {  
 this.ctx = ctx;  
 this.name = name;  
 this.rate = rate;  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 tick();  
 Thread.sleep(1000 / rate);  
 } catch (InterruptedException e) {  
 break;  
 }  
 }  
 }  
  
 abstract protected void tick() throws InterruptedException;  
}  
  
  
src/main/java/org/lab03/thread/Generator.java:  
package org.lab03.thread;  
  
import org.lab03.Event;  
import org.lab03.EventHolder;  
import org.lab03.Logger;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.time.LocalDateTime;  
import java.util.Random;  
  
public class Generator extends AbstractThread {  
 public Generator(ClassPathXmlApplicationContext ctx, String name, long rate) {  
 super(ctx, name, rate);  
 }  
  
 @Override  
 protected void tick() {  
 Random random = new Random();  
 Logger logger = ctx.getBean("logger", Logger.class);  
 EventHolder holder = ctx.getBean("holder", EventHolder.class);  
  
 Event event = ctx.getBean("event", Event.class);  
  
 event.setName("Зачет");  
 event.setPlace("Ауд. " + random.nextInt(6999));  
 event.setDateTime(LocalDateTime.now().plusDays(random.nextInt(20)));  
  
 // logger.info(name + ": " + event);  
  
 holder.set(event);  
 }  
}  
  
  
src/main/java/org/lab03/thread/Reader.java:  
package org.lab03.thread;  
  
import org.lab03.Event;  
import org.lab03.EventHolder;  
import org.lab03.Logger;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Reader extends AbstractThread {  
  
 public Reader(ClassPathXmlApplicationContext ctx, String name, long rate) {  
 super(ctx, name, rate);  
 }  
  
 @Override  
 protected void tick() throws InterruptedException {  
 Logger logger = ctx.getBean("logger", Logger.class);  
 EventHolder holder = ctx.getBean("holder", EventHolder.class);  
  
 Event event = holder.get();  
  
 if (event == null) {  
 logger.warning(name + ": event is null");  
 return;  
 }  
  
 logger.info(name + ": " + event);  
 }  
}

**Приложение 2**

**Результаты тестирования**

2024-05-26T17:41:37.683871100 - INFO - Reader 1: Name: Зачет; Place: Ауд. 1025; DateTime: 2024-06-11T17:41:37.679869600

2024-05-26T17:41:37.784720100 - INFO - Reader 2: Name: Зачет; Place: Ауд. 4068; DateTime: 2024-06-04T17:41:37.784720100

2024-05-26T17:41:37.894640200 - INFO - Reader 1: Name: Зачет; Place: Ауд. 5842; DateTime: 2024-06-11T17:41:37.894640200

2024-05-26T17:41:38.006162700 - INFO - Reader 2: Name: Зачет; Place: Ауд. 2453; DateTime: 2024-06-01T17:41:38.006162700

2024-05-26T17:41:38.114144700 - INFO - Reader 1: Name: Зачет; Place: Ауд. 1065; DateTime: 2024-06-07T17:41:38.114144700

2024-05-26T17:41:38.223365800 - INFO - Reader 2: Name: Зачет; Place: Ауд. 3626; DateTime: 2024-06-12T17:41:38.223365800

2024-05-26T17:41:38.333810300 - INFO - Reader 1: Name: Зачет; Place: Ауд. 641; DateTime: 2024-05-29T17:41:38.333810300

2024-05-26T17:41:38.441975800 - INFO - Reader 2: Name: Зачет; Place: Ауд. 5190; DateTime: 2024-05-26T17:41:38.441975800

2024-05-26T17:41:38.549384400 - INFO - Reader 1: Name: Зачет; Place: Ауд. 368; DateTime: 2024-06-06T17:41:38.549384400

2024-05-26T17:41:38.659961900 - INFO - Reader 2: Name: Зачет; Place: Ауд. 6734; DateTime: 2024-06-03T17:41:38.659961900

Process finished with exit code 0