ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Н. И. Чулочникова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 2

МНОГОПОТОЧНОЕ ПРОГРАММИРОВАНИЕ

по курсу:

КРОССПЛАТФОРМЕННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1 Цели работы

Цели работы заключались в следующем:

- Ознакомиться с основами многопоточного программирования в Java.
- Изучить принципы взаимодействия и синхронизации потоков при помощи объектов-мониторов (synchronized, wait(), notify(), notifyAll()).
- Научиться реализовывать обмен сигналами между потоками и организовывать их согласованную работу.
- Разработать программу, в которой один поток выполняет роль таймера (хронометра), а другие потоки реагируют на его сигналы с заданной периодичностью, не вмешиваясь в код основного потока.

2 Задание

Работа выполнялась по варианту № 17.

Напишите программу, которая каждую секунду отображает на экране данные о времени, прошедшем от начала сессии, а другой её поток выводит сообщение каждые 5 секунд. Предусмотрите возможность ежесекундного оповещения потока, воспроизводящего сообщение, потоком, отсчитывающим время. Не внося изменений в код потока-"хронометра", добавьте ещё один поток, который выводит на экран другое сообщение каждые 7 секунд.

3 Листинг программы

Ниже представлен листинг полной программы, реализованной на Java.

Листинг 1 — Листинг программы

```
public class MultiThreadLab {
  public static void main(String[] args) {
    Object lock = new Object(); // общий объект синхронизации

    TimerThread timer = new TimerThread(lock);
    MessageThread message5 = new MessageThread(lock, 5, "Сообщение каждые 5 
секунд");
    MessageThread message7 = new MessageThread(lock, 7, "Сообщение каждые 7 
секунд");

    timer.start();
    message5.start();
    message7.start();
  }
}
```

```
// Поток-хронометр
class TimerThread extends Thread {
  private final Object lock;
  private int secondsPassed = 0;

  public TimerThread(Object lock) {
    this.lock = lock;
  }

  @Override
  public void run() {
    try {
      while (true) {
        Thread.sleep(1000);
        secondsPassed++;
        System.out.println("Прошло секунд: " + secondsPassed);
        synchronized (lock) {
          lock.notifyAll(); // оповещаем другие потоки
        }
      }
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}

// Поток, который выводит сообщение message каждые interval секунд
class MessageThread extends Thread {
  private final Object lock;
  private final int interval;
  private final String message;
  private int currentSeconds = 0;

  public MessageThread(Object lock, int interval, String message) {
    this.lock = lock;
    this.interval = interval;
    this.message = message;
  }

  @Override
  public void run() {
    try {
      while (true) {
        synchronized (lock) {
          lock.wait(); // ждём уведомления от хронометра
          currentSeconds++;
          if (currentSeconds % interval == 0) {
            System.out.println("\t" + message);
          }
        }
      }
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

4 Результаты работы программы

На рис. 1 изображен результат исполнения программы.

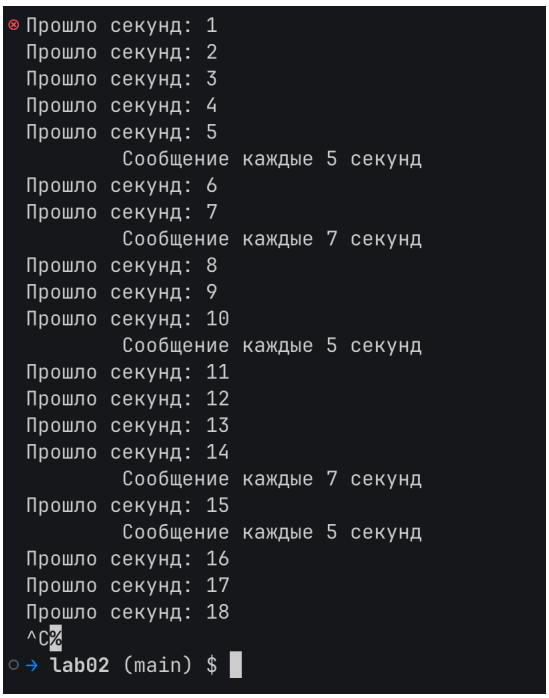

Рисунок 1 — Тестирование программы

5 Выводы

В ходе лабораторной работы была реализована многопоточная программа, демонстрирующая взаимодействие нескольких потоков через общий объект синхронизации. Поток-хронометр каждую секунду уведомлял остальные потоки о прошедшем времени, а дополнительные потоки выводили сообщения через определённые интервалы (5 и 7 секунд).

В процессе выполнения работы было изучено:

- использование ключевого слова synchronized для организации взаимного исключения при доступе к общим ресурсам;
- применение методов wait() и notifyAll() для синхронизации действий между потоками;
- механизм прерывания потоков и обработка исключения InterruptedException.

Полученная программа наглядно показала, как с помощью стандартных средств Java можно организовать координированную работу нескольких потоков, обеспечивая корректное взаимодействие и последовательность выполнения задач.