Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Разработка кроссплатформенных приложений»

Инструкция

по выполнению лабораторной работы
«Разработка, отладка и испытание программ с несколькими потоками»

Минск 2025 г.

Лабораторная работа № 14

Тема работы: «Разработка, отладка и испытание программ с несколькими потоками»

1 Цель работы

Сформировать умения создания и запуска нескольких потоков в программе.

2 Задание

Номер варианта соответствует вашему номеру по списку, если номер по списку больше последнего варианта, то необходимо начать подсчет варианта с начала (например, учащийся с номером по списку 13 должен выполнить 2 вариант).

Необходимо создать класс (алгоритм шифрования).
Класс содержит:

- поля для хранения ключевой информации;
- конструктора (конструкторы) для создания ключевой информации;
- методы шифровки и дешифровки информации.

Нужно создать несколько объектов этого класса и осуществить с помощью их шифрование данных. Каждый объект шифрует (дешифрует) в своем потоке.

Вариант 1.
Аффинная система шифрования Цезаря

Вариант 2.
Биграммный шифр Плейфера

Вариант 3.
Двойная таблица перестановки

Вариант 4.
Криптосистема RSA или Эль-Гамаля (на выбор)

Вариант 5.
Магический квадрат

Вариант 6.
Простая таблица перестановки

Вариант 7.
Система цезаря с ключевым словам

Вариант 8.
Система шифрования Вижинера

Вариант 9.
Система шифрования Цезаря

Вариант 10.
Шифр «двойной квадрат» Уинстона

Вариант 11.
Шифрующие таблицы Трисемуса

3 Оснащение работы

Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4 Основные теоретические сведения

Многопоточное программирование позволяет разделить представление и обработку информации на несколько «легковесных» процессов (light-weight processes), имеющих общий доступ как к методам различных объектов приложения, так и к их полям. Многопоточность незаменима в тех случаях, когда графический интерфейс должен реагировать на действия пользователя при выполнении определенной обработки информации. Потоки могут взаимодействовать друг с другом через основной «родительский» поток, из которого они стартованы.

В качестве примера можно привести некоторый поток, отвечающий за представление информации в интерфейсе, который ожидает завершения работы другого потока, загружающего файл, и одновременно отображает некоторую анимацию или обновляет прогресс-бар. Кроме того этот поток может остановить загружающий файл поток при нажатии кнопки «Отмена».

Создатели Java предоставили две возможности создания потоков: реализация (implementing) интерфейса *Runnable* и расширение(extending) класса *Thread*. Расширение класса - это путь наследования методов и переменных класса родителя. В этом случае можно наследоваться только от одного родительского класса *Thread*. Данное ограничение внутри Java можно преодолеть реализацией интерфейса *Runnable*, который является наиболее распространённым способом создания потоков.

Преимущества потоков перед процессами

- потоки намного легче процессов поскольку требуют меньше времени и ресурсов;
- переключение контекста между потоками намного быстрее, чем между процессами;

➢ намного проще добиться взаимодействия между потоками, чем между процессами.

Главный поток

Каждое java приложение имеет хотя бы один выполняющийся поток. Поток, с которого начинается выполнение программы, называется главным. После создания процесса, как правило, JVM начинает выполнение главного потока с метода main(). Затем, по мере необходимости, могут быть запущены дополнительные потоки. **Многопоточность** — это два и более потоков, выполняющихся одновременно в одной программе. Компьютер с одноядерным процессором может выполнять только один поток, разделяя процессорное время между различными процессами и потоками.

Класс Thread

В классе Thread определены семь перегруженных конструкторов, большое количество методов, предназначенных для работы с потоками, и три константы (приоритеты выполнения потока).

Конструкторы класса Thread

```
Thread();
Thread(Runnable target);
Thread(Runnable target, String name);
Thread(String name);
Thread(ThreadGroup group, Runnable target);
Thread(ThreadGroup group, Runnable target, String name);
Thread(ThreadGroup group, String name);
```

где :

- target – экземпляр класса реализующего интерфейс Runnable;
- name – имя создаваемого потока;
- group – группа к которой относится поток.

Пример создания потока, который входит в группу, реализует интерфейс Runnable и имеет свое уникальное название :

```
Runnable    r  = new MyClassRunnable();
ThreadGroup tg = new ThreadGroup();
Thread      t  = new Thread(tg, r, "myThread");
```

Группы потоков удобно использовать, когда необходимо одинаково управлять несколькими потоками. Например, несколько потоков выводят данные на печать и необходимо прервать печать всех документов поставленных в очередь. В этом случае удобно применить команду ко всем потокам одновременно, а не к каждому потоку отдельно. Но это можно сделать, если потоки отнесены к одной группе.

Несмотря на то, что главный поток создаётся автоматически, им можно управлять. Для этого необходимо создать объект класса *Thread* вызовом метода *currentThread()*.

Методы класса Thread

Наиболее часто используемые методы класса *Thread* для управления потоками:

- long getId() - получение идентификатора потока;
- String getName() - получение имени потока;
- int getPriority() - получение приоритета потока;
- State getState() - определение состояния потока;
- void interrupt() - прерывание выполнения потока;
- boolean isAlive() - проверка, выполняется ли поток;
- boolean isDaemon() - проверка, является ли поток «daemon»;
- void join() - ожидание завершения потока;
- void join(millis) - ожидание millis милисекунд завершения потока;
- void notify() - «пробуждение» отдельного потока, ожидающего «сигнала»;
- void notifyAll() - «пробуждение» всех потоков, ожидающих «сигнала»;
- void run() - запуск потока, если поток был создан с использованием интерфейса Runnable;
- void setDaemon(bool) - определение «daemon» потока;
- void setPriority(int) - определение приоритета потока;
- void sleep(int) - приостановка потока на заданное время;
- void start() - запуск потока.
- void wait() - приостановка потока, пока другой поток не вызовет метод notify();
- void wait(millis) - приостановка потока на millis милисекунд или пока другой поток не вызовет метод notify();

Жизненный цикл потока

При выполнении программы объект Thread может находиться в одном из четырех основных состояний: «новый», «работоспособный», «неработоспособный» и «пассивный». При создании потока он получает состояние «новый» (NEW) и не выполняется. Для перевода потока из состояния «новый» в «работоспособный» (RUNNABLE) следует выполнить метод start(), вызывающий метод run().

Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления Thread.State :

- NEW — поток создан, но еще не запущен;
- RUNNABLE — поток выполняется;
- BLOCKED — поток блокирован;
- WAITING — поток ждет окончания работы другого потока;
- TIMED_WAITING — поток некоторое время ждет окончания другого потока;
- TERMINATED — поток завершен.

Пример использования Thread

В примере ChickenEgg рассматривается параллельная работа двух потоков (главный поток и поток Egg), в которых идет спор, «что было раньше, яйцо или курица?». Каждый поток высказывает свое мнение после небольшой задержки, формируемой методом ChickenEgg.getTimeSleep(). Побеждает тот поток, который последним говорит свое слово.

```
package example;

import java.util.Random;

class Egg extends Thread
{
    @Override
    public void run()
    {
        for(int i = 0; i < 5; i++) {
            try {
                // Приостанавливаем поток
                sleep(ChickenEgg.getTimeSleep());
                System.out.println("Яйцо");
            }catch(InterruptedException e){}
        }
    }
}
public class ChickenEgg
{
    public static int getTimeSleep()
    {
        final Random random = new Random();
        int tm = random.nextInt(1000);
        if (tm < 10)
            tm *= 100;
        else if (tm < 100)
            tm *= 10;
        return tm;
    }
    public static void main(String[] args)
    {
        Egg egg = new Egg (); // Создание потока
        System.out.println(
                  "Начинаем спор : кто появился
первым ?");

        egg.start(); // Запуск потока
```

```
            for(int i = 0; i < 5; i++) {
                try {
                    // Приостанавливаем поток

Thread.sleep(ChickenEgg.getTimeSleep());
                    System.out.println("Курица");
                }catch(InterruptedException e){}
            }
            if(egg.isAlive()) {
                // Сказало ли яйцо последнее слово?
                try {
                    //    Ждем,    пока    яйцо    закончит
высказываться
                    egg.join();
                } catch (InterruptedException e){}

                System.out.println("Первым появилось яйцо
!!!");
            } else {
                //если      оппонент      уже      закончил
высказываться
                System.out.println("Первой      появилась
курица !!!");
            }
            System.out.println("Спор закончен");
        }
    }
```

При выполнении программы в консоль было выведено следующее сообщение.

```
Начинаем спор : кто появился первым ?
Курица
Курица
Яйцо
Курица
Яйцо
Яйцо
Курица
Курица
Яйцо
Яйцо
Первым появилось яйцо !!!
Спор закончен
```

Невозможно точно предсказать, какой поток закончит высказываться последним. При следующем запуске «победитель» может измениться. Это происходит вследствии так называемого «асинхронного выполнения кода». Асинхронность обеспечивает независимость выполнения потоков. Или, другими словами, параллельные потоки независимы друг от друга, за исключением случаев, когда бизнес-логика зависимости выполнения потоков определяется предусмотренными для этого средств языка.

Интерфейс Runnable

Интерфейс *Runnable* содержит только один метод *run()* :

```
interface Runnable
{
    void run();
}
```

Метод run() выполняется при запуске потока. После определения объекта Runnable он передается в один из конструкторов класса Thread.

Пример класса RunnableExample, реализующего интерфейс Runnable

```
package example;

class MyThread implements Runnable
{
    Thread thread;
    MyThread() {
        thread = new Thread(this, "Дополнительный
поток");
        System.out.println("Создан дополнительный
поток " +

thread);
        thread.start();
    }
    @Override
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println(
                        "\tдополнительный поток: "
+ i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println(
                    "\tдополнительный поток
прерван");
        }
```

```
            System.out.println(
                                "\tдополнительный       поток
завершён");
        }
    }
    public class RunnableExample
    {
        public static void main(String[] args)
        {
            new MyThread();
            try {
                for (int i = 5; i > 0; i--) {
                    System.out.println("Главный поток: "
+ i);
                    Thread.sleep(1000);
                }
            } catch (InterruptedException e) {
                System.out.println("Главный          поток
прерван");
            }
            System.out.println("Главный поток завершён");
        }
    }
```

При выполнении программы в консоль было выведено следующее сообщение.

```
    Создан дополнительный поток Thread[Дополнительный
поток,5,main]
    Главный поток: 5
        дополнительный поток: 5
        дополнительный поток: 4
    Главный поток: 4
        дополнительный поток: 3
        дополнительный поток: 2
    Главный поток: 3
        дополнительный поток: 1
        дополнительный поток завершён
    Главный поток: 2
    Главный поток: 1
    Главный поток завершён
```

5 Порядок выполнения работы

1. Выделить ключевые моменты задачи;
2. Построить алгоритм и теоритическую объектную модель решения задачи;
3. Запрограммировать полученный алгоритм и объектную модель;
4. Провести тестирование полученной программы.

6 Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

_____

_____

7. Контрольные вопросы и задания

1. Что представляет собой поток?
2. В чем отличие потока от процесса?
3. Какими способами можно создавать потоки?
4. Расскажите о жизненном цикле потока.
5. Для чего используется интерфейс Runnable?

8. Рекомендуемая литература

1. **Урванов, Ф. В.** Java. Состояние языка и его перспективы. / Ф.В. Урванов. - Санкт-Петербург : БХВ-Петербург, 2023. - 368 с.
2. **Копец Дэвид.** Классические задачи Computer Science на языке Java. - Санкт-Петербург : Питер, 2022. - 288 с
3. **Васильев А. Н.** Java. Объектно-ориентированное программирование: Учебное пособие / А.Н. Васильев. - Санкт-Петербург : Питер, 2021. - 400 с.
4. **Эванс Бенджамин**. Java для опытных разработчиков. 2-е изд. — (Серия «Библиотека программиста»). - Санкт-Петербург : Питер, 2024. - 736 с.
5. **Лой Марк.** Программируем на Java. 5-е межд. изд. . - Санкт-Петербург : Питер, 2023. - 544 с.
6. **Гетц Брайан**. Java Concurrency на практике. - Санкт-Петербург : Питер, 2021. - 464 с.