Синхронизация потоков выполнения с помощью методов класса Thread.

1. Методы класса Thread

1.1 Конструкторы класса Thread

- Создание нового объекта Thread
  public Thread();
- Создание нового объекта Thread с указанием объекта, для которого будет активирован метод run().
  public Thread(Runnable target);
- Аналогично предыдущему конструктору, но также указывается имя объекта Thread
  public Thread(Runnable target, String name);
- Создание нового объекта Thread с указанием его имени
  public Thread(String name);
- Создание нового объекта Thread с указанием имени группы, к которой он принадлежит, и объекта, для которого будет активирован метод run().
  public Thread(ThreadGroup group, Runnable target);
- Аналогично предыдущему конструктору, но также указывается имя объекта Thread
  public Thread(ThreadGroup group, Runnable target, String name);
- Создание нового объекта Thread с указанием названия группы, к которой он принадлежит, и его названия.
  public Thread(ThreadGroup group, String name);

1.2. Методы класса Thread

- activeCount – определяет количество активных потоков в группе, к которой он принадлежит
  public static int activeCount();
- currentThread – определяет текущий активный поток
  public static Thread currentThread();
- getName – возвращает имя потока.
  public final String getName();
- getPriority возвращает текущий приоритет потока
  public final int getPriority();
- getThreadGroup – возвращает группу, к которой принадлежит поток
  public final ThreadGroup getThreadGroup();
- interrupt – прерывает выполнение потока
  public void interrupt();
- Interrupted – определяет, прерван ли поток
  public static boolean interrupted();
- isAlive – определяет, выполняется ли поток
  public final boolean isAlive();
- isDaemon – определяет, является ли поток демоном
  public final boolean isDaemon();
- join – ожидает полного выполнения потока или ожидает выполнения потока в течение времени, указанного в миллисекундах или наносекундах
  public final void join();
  public final void join(long millis);
  public final void join(long millis, int nanos);
- run – метод активируется для выполнения потока
  public void run();
- setDaemon – указывает, что созданный поток будет демоном

public final void setDaemon(boolean on);

- setName – присваивает имя потоку
  public final void setName(String name);
- yield – временно останавливает активный поток и позволяет выполнить другой поток
  public static void yield();
- setPriority – установка приоритета
  public final void setPriority(int newPriority);
- sleep – останавливает выполнение (»засыпает») потока на время, указанное в миллисекундах и наносекундах
  public static void sleep(long millis);
  public static void sleep(long millis, int nanos);
- start – активация потока
  public void start();
- stop – окончательная деактивация потоков
  public final void stop();

Пример реализации нескольких методов синхронизации из класса Thread:

```
class Lab3 {
    static Thread1 primu = new Thread1();
    static Thread2 второй = new Thread2();
    static Thread3 третий = new Thread3();
    static Thread4 четвертый = new Thread4();
    static int counter = 0;
    static int counter2 = 0;
    static String info1 = "\nИмя";
    static String info2 = "\nФамилия";
    static String info3 = "\nКонкурентное и распределенное программирование";
    static String info4 = "\nГруппа";

    static class Thread1 extends Thread {
        @Override
        public void run() {
            System.out.println("");
            while (true) {
                if (!(primu.isInterrupted())) {
                    System.out.println("Запуск потока 1");
                    int p_term = 0;
                    int d_term = 0;
                    int p_term1 = 0;
                    int d_term2 = 0;
                    int n = 0;
                    for (int i = 200; i <= 300; i += 2) {
                        n++;
                        if (n == 1) {
                            p_term = i;
                        }
                        if (n == 2) {
                            d_term = i;
                        }
                        if (n == 3) {
                            p_term1 = i;
                        }
```

```
                        if (n == 4) {
                            d_term2 = i;
                            n = 0;
                            System.out.println(this + "(" + p_term + "*" + d_term +
"+" + p_term1 + "*" + d_term2 + ")=" +
                                    (p_term * d_term + p_term1 * d_term2));
                        }

                    }
try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                    }

                    primu.interrupt();
                }


                char отображение;

                if (второй.getPriority() == 9 && четвертый.getPriority() == 9 &&
первый.getPriority() == 9) {
                    for (int i = 0; i < info1.length(); i++) {
                        отображение = info1.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(отображение);
                    }
                    al_treilea.setPriority(9);
                    break;

                }
            }
        }
    }


    static class Thread2 extends Thread {
        @Override
        public void run() {
            System.out.println("");
            while (true) {
                if (!(второй.isInterrupted())) {
                    System.out.println("Starting Thread 2");
                    int p_term = 0;
                    int d_term = 0;
                    int p_term1 = 0;
                    int d_term2 = 0;
                    int n = 0;
                    for (int i = 106; i >= 6; i -= 2) {
                        n++;
                        if (n == 1) {
                            p_term = i;
```

```
                        }
                        if (n == 2) {
                            d_term = i;
                        }
                        if (n == 3) {
                            p_term1 = i;
                        }
                        if (n == 4) {
                            d_term2 = i;
                            n = 0;
                            System.out.println(this + "(" + p_term + "*" + d_term +
"+" + p_term1 + "*" + d_term2 + ")=" +
                                    (p_term * d_term + p_term1 * d_term2));

                        }
                    }


                    al_doilea.interrupt();

                }
                System.out.println();
                второй.setPriority(9);
                char отображение;
                if (второй.getPriority() == 9) {
                    for (int i = 0; i < info2.length(); i++) {
                        отображение = info2.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(отображение);
                    }
                    al_patrulea.setPriority(9);
                    break;
                }

            }
        }
    }

    static class Thread3 extends Thread {

        @Override
        public void run() {
            while (true) {
                if (!(al_treilea.isInterrupted())) {
                    System.out.println("Starting Thread 3");
                    for (int i = 234; i <= 1000; i += 1) {
                        System.out.print(al_treilea.getName() + ":" + i + " ");
                        counter++;
                        if (счетчик == 10) {
                            System.out.println(" ");
                            счетчик = 0;
                        }
```

```
                        if (i == 1000) {
                            System.out.println("\n ");

                        }
                    }
                    al_treilea.interrupt();
                }
                char отображение;
                if (второй.getPriority() == 9 && четвертый.getPriority() == 9 &&
первый.getPriority()== 9 && третий.getPriority() == 9) {
                    for (int i = 0; i < info3.length(); i++) {
                        отображение = info3.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(отображение);
                    }

                    break;


                }
            }
        }
    }

    static class Thread4 extends Thread {
        @Override
        public void run() {
            while (true) {
                if (!(al_patrulea.isInterrupted())) {
                    System.out.println("Starting Thread 4");
                    for (int i =1234; i >= 457; i -= 1) {
                        System.out.print(al_patrulea.getName() + ":" + i + " ");
                        counter2++;
                        if (counter2 == 10) {
                            System.out.println(" ");
                            counter2 = 0;
                        }
                    }

                    al_patrulea.interrupt();
                }
                char отображение;
                if (al_doilea.getPriority() == 9 && al_patrulea.getPriority() == 9) {
                    for (int i = 0; i < info4.length(); i++) {
                        отображение = info4.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(отображение);
                    }
                    первый.setPriority(9);
```

```
                    break;
                }
            }
        }
    }
    public static void main(String[] args) throws InterruptedException {
        первый.start();

        второй.start();
        третий.start();

        третий.setName("Th3");
       // третий.join(1000);
        четвертый.start();
        четвертый.setName("Th4");
    }
    }
```

Были использованы методы **setPriority(), interrupt(), join()**

Лабораторная работа № 3

Тема работы: ***Синхронизация потоков выполнения с использованием методов класса Thread.***

Цель работы:

Изучение механизмов синхронизации потоков выполнения в языке Java с использованием методов, предоставляемых классом `Thread`, для понимания того, как можно обеспечить правильное и упорядоченное выполнение параллельных процессов.

Цели работы:

1. Ознакомление с концепциями **потоков**, **конкуренции** и **синхронизации**.
2. Изучение методов класса `Thread` (например: `sleep()`, `join()`, `interrupt()`, `isAlive()`).
3. Реализация простых примеров синхронизации между двумя или более потоками выполнения.
4. Анализ поведения программы в ситуациях конкуренции без синхронизации и с синхронизацией.
5. Применение методов класса `Thread` для решения практических задач.
6. Формирование практических навыков конкурентного программирования и управления проблемами синхронизации.

***Пример реализации*:**

```
public class Lab3PCD {
    static int size = 100;
    static int counter = 0;
    static int[] b = new int[size];
    static int[] a;

    static Straight first = new Straight();
    static Reverse second = new Reverse();
    static ReverseInterval third = new ReverseInterval();
```

```
    static StraightInterval fourth = new StraightInterval();

    static intpairone = 0;
    static intpairtwo = 0;

    static class Straight extends Thread {
int result = 0;
        @Override
        public void run(){
System.out.println("Starting Thread 1");
            for (inti = 0; i< counter; i+=4){
try{
Thread.sleep(100);
                }
                catch (InterruptedExceptione){
e.printStackTrace();
                }
                if ((i+4) >= counter){
                }
                else {
pairone = a[i] + a[i+1];
pairtwo = a[i+2] + a[i+3];
                    result = pairone + pairtwo;
System.out.println  ("Текущее  значение  для  потока  1:  " + pairone + " + " +
pairtwo + " = " + result);
                }
            }
            while(fourth.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("1: Name");
        }
    }

    static class Reverse extends Thread {
int result = 0;
        @Override
        public void run(){
System.out.println("Starting Thread 2");
            for (inti = counter-1; i>= 0; i-=4){
try{
Thread.sleep(100);
                }
                catch (InterruptedExceptione){
e.printStackTrace();
                }
               // if (a[i] <= 106 && a[i] >= 16) {
                    if ((i - 4) <= 0) {
                    }
                    else {
pairone = a[i] + a[i-1];
pairtwo = a[i-2] + a[i-3];
                        result = pairone + pairtwo;
```

```
System.out.println ("Текущее значение для потока 2: " + pairone + " + " +
pairtwo + " = " + result);
                    }

            }
System.out.println("2: Фамилия");
        }
    }

    public static class StraightIntervalextends  Thread {
        @Override
        public void run(){
System.out.println("Starting Thread 4 ");
            for (inti = 200; i<= 300; i++){
System.out.print(i + " ");
            }
System.out.println(" ");
            while(second.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("4: Группа");
        }

    }

    public static class ReverseInterval extends Thread {
        @Override
        public void run(){
System.out.println("Starting Thread 3 ");
            for (inti = 1000; i<= 1100; i++){
System.out.print(i + " ");
            }
System.out.println(" ");
            while(first.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("3: P C D");
        }
    }

    public static void main(String[] args) {
System.out.println("Печать массива: ");
        for (inti = 0; i< 100; i++) {
            b[i] = (int) Math.round((Math.random() * 100) + 15);
System.out.print (b[i] + " ");
            if (i == 50){
System.out.println (b[i] + " ");
            }
            if (i == 99)
```

```
System.out.println();
            if (b[i]%2==0){
                счетчик++;
            }
        }
        a = new int[counter+1];
int k = 0;
        for (inti = 0; i< 100; i++){
            if (b[i]%2==0){
                a[k] = b[i];
                k++;
            }
        }

first.start();
second.start();
third.start();
fourth.start();
try{
first.join();
второй.присоединиться();
third.join();
fourth.join();
        } catch (InterruptedException e) {
e.printStackTrace();
        }

    }
}
```

Результат выполнения:

Печать массива:
44 25 18 115 103 83 53 67 95 56 107 17 82 16 85 59 55 30 16 114 40 20 112 39 101 102 59 20 27 60 68 21 86 106 87 81 26 107 68 52 16 92 113 79 62 80 109 67 56 100 63 63 105 114 47 39 34 94 59 26 43 17 40 23 81 104 94 55 56 57 20 18 65 88 81 43 77 103 101 48 40 82 88 91 29 19 28 53 19 51 77 20 38 56 54 33 57 20 70 103 108
Начало темы 1
Начальная ветка 2
Начало темы 4
Начальная ветка 3
1000 1001 1002 1003 200 1004 201 202 203 1005 204 1006 205 1007 206 1008 207 1009 208 1010 209 1011 210 211 1012 212 213 1013 214 215 1014 216 217 1015 218 1016 219 1017 220 1018 221 1019 222 1020 223 1021 224 1022 225 1023 226 1024 227 1025 228 1026 229 1027 230 1028 231 1029 232 1030 233 1031 234 1032 235 1033 236 1034 237 1035 238 1036 239 1037 240 1038 241 1039 242 1040 243 1041 244 1042 245 1043 246 1044 247 248 1045 249 1046 250 1047 251 252 253 1048 254 255 256 1049 1050 257 258 259 260 1051 1052 1053 261 262 263 264 265 266 267 1054 268 1055 269 1056 270 1057 271 1058 272 1059 273 1060 274 275 276 277 278 1061 279 1062 280 1063 281 1064 1065 282 1066 283 1067 284 1068 285 1069 1070 286 1071 287 1072 288 1073 1074 289 290 1075 291 1076 292 293 294 295 1077 296 1078 297 1079 298 1080 299 300 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100
Текущее значение для потока 2: 178 + 74 = 252

Текущее значение для потока 1: 62 + 138 = 200
Текущее значение для потока 2: 94 + 48 = 142
Текущее значение для потока 1: 46 + 130 = 176
Текущее значение для потока 1: 60 + 214 = 274
Текущее значение для потока 2: 170 + 88 = 258
Текущее значение для потока 2: 106 + 154 = 182
Текущее значение для потока 1: 80 + 154 = 234
Текущее значение для потока 1: 132 + 66 = 252
Текущее значение для потока 2: 198 + 66 = 264
Текущее значение для потока 1: 128 + 214 = 342
Текущее значение для потока 2: 128 + 214 = 342
Текущее значение для потока 2: 136 + 154 = 290
Текущее значение для потока 1: 156 + 154 = 304
Текущее значение для потока 1: 68 + 94 = 212
Текущее значение для потока 2: 68 + 94 = 162
Текущее значение для потока 1: 192 + 128 = 188
Текущее значение для потока 2: 192 + 128 = 320
Текущее значение для потока 1: 136 + 122 = 258
Текущее значение для потока 2: 136 + 122 = 254
Текущее значение для потока 1: 116 + 58 = 174
Текущее значение для потока 2: 154 + 46 = 200
Текущее значение для потока 1: 110 + 90 = 200
2: Фамилия
4: Группа
1: Имя
3: P C D

***Задачи для решения**:*

Задача:

Напишите программу, которая создает четыре потока выполнения. Все потоки будут считывать данные из диапазона, указанного в задании. Пары потоков 1-2 и 3-4 имеют общие задачи. Первый поток Th1 выполнит: Задачу 1 из таблицы 3, второй поток Th2 отобразит Задачу 2 из таблицы 3, третий поток Th3 отобразит Задачу 3 из таблицы 3, четвертый поток Th4 отобразит Задачу 4 из таблицы 3.

После завершения выполнения задач потоков выполнения поток Th2 отобразит имя студента, выполнившего данную лабораторную работу, Th4 отобразит группу, Th1 отобразит фамилию студента, Th3 отобразит название дисциплины (полное). Буквы текста будут появляться на экране с интервалом в 100 миллисекунд.

Один член команды выполняет реализацию и синхронизацию потоков 1 - 2. Второй член команды выполняет реализацию и синхронизацию потоков 3 – 4. Каждый член команды использует по 2 разных метода синхронизации. **Методы класса Thread не используются**

Таблица 3 Условия для выполнения задачи проблемы в соответствии с вариантами

	Задача 1	Задача 2	Задача 3	Задача 4
1	Суммы четных чисел по два, начиная поиск и суммирование с первого элемента	Суммы четных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти с начала интервал [100,500]	Пройти с конца интервал [300,700]
2	Сумма позиций четных чисел по два, начиная поиск и суммирование с первого элемента	Сумма позиций четных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти с начала интервал [120, 690]	Пройти с конца интервал [1000, 1567]
3	Суммы нечетных чисел по два, начиная поиск и суммирование с первого элемента	Суммы нечетных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти с начала интервала [0, 798]	Пройти с конца интервал [1456, 2111]
4	Суммы позиций нечетных чисел по два, начиная поиск и суммирование с первого элемента	Суммы позиций нечетных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти с начала интервал [234, 987]	Пройти с конца интервал [123, 890]
5	Суммы произведений чисел с четными позициями по два, начиная с поиск и суммирование с первым элементом	Суммы произведений чисел с четными позициями по два, начиная с поиск и суммирование с последним элементом	Пройти с начала интервала [567, 1002]	Пройти с конца интервал [567, 1100]
6	Суммы произведений чисел на нечетных позициях по два, начиная с поиск и суммирование с первым элементом	Суммы произведений чисел на нечетных позициях по два, начиная с поиск и суммирование с последним элементом	Пройти с начала интервал [654, 1278]	Пройти с конца интервал [123, 908]
7	Суммы произведений четных чисел по два, начиная поиск и суммирование с первого элемента	Суммы произведений четных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти с начала интервал [234, 1000]	Пройти с конца интервал [456,1234]
8	Суммы произведений нечетных чисел по два, начиная с поиск и суммирование с первым элементом	Суммы произведений нечетных чисел по два, начиная с поиск и суммирование с последним элементом	Пройти с начала интервала [126, 987]	Пройти с конца интервал [213, 899]
9	Разница произведений чисел позиций по два, начиная с поиск и	Разница между числами в позициях появляется по два, начиная с поиска и	Пройти с начала интервал	Пройти с конца интервал

	суммирование с первым элементом	суммирования с последним элементом	[222, 999]	[3333, 3999]
10	Разница произведений чисел на нечетных позициях по два, начиная с поиск и суммирование с первым элементом	Разница произведений чисел на нечетных позициях по два, начиная с поиск и суммирование с последним элементом	Пройти с начала интервал [11,548]	Пройти с конца интервал [1234, 678]
11	Разница произведений четных чисел по два, начиная поиск и суммирование с первым элементом	Суммы произведений четных чисел по два, начиная поиск и суммирование с последнего элемента	Пройти интервал от начала [385, 1024]	Пройти интервал от конца [1000, 408]
12	Суммы произведений нечетных чисел по два, начиная с поиск и суммирование с первым элементом	Суммы произведений нечетных чисел по два, начиная с поиск и суммирование с последним элементом	Пройти с начала интервала [1111, 1748]	Пройти с конца интервал [2000, 1478]

Критерии оценки:

1. Создание и инициализация потоков для выполнения задач.
2. Выбор методов синхронизации потоков.
3. Синхронизация потоков с помощью подходящих методов.
4. Создание графического интерфейса программы.
5. Корректность кода — проверка правильности кода , отсутствие ошибок в работе .
6. Соблюдение инструкций и требований — проверка правильности требований задачи, таких как количество нитей выполнения.
7. Оптимизация кода - оценка эффективности кода в использовании ресурсов и избегание кода.
8. Соблюдение сроков сдачи - оценка баллов за пунктуальность, если работа была сдана в установленный срок.
9. Оценка знаний - объяснения, данные о процессе выполнения работы, что может включать описание основных функций и использованной логики.
10. Использование ИИ студентом.

Для получения оценки 5-6 обязательны критерии 1,2,3,5,8,9

Для получения оценки 7–8 обязательны критерии 1, 2, 3, 4, 5, 6, 8, 9.

Для получения оценки 9–10 обязательны критерии 1–9.

Если был использован критерий 10, оценка снижается на 2 балла, только если студент разъяснил работу кода. В противном случае лабораторная работа не принимается.

***Контрольные вопросы*:**

1. Каковы основные методы класса `Thread` и какие функции они предоставляют?
2. В чем разница между методами `start()` и `run()`?
3. Как работает метод `join()` и в каких ситуациях он используется?
4. Какое влияние оказывает метод `sleep()` на поток выполнения?

5. Как можно прервать поток выполнения с помощью `interrupt()`?
6. Какова роль метода `isAlive()` и как его можно использовать для контроля выполнения?
7. Какие проблемы могут возникнуть, если несколько потоков одновременно обращаются к одному и тому же ресурсу без синхронизации?
8. В чем разница между синхронизацией с помощью методов класса `Thread` и использованием ключевого слова `synchronized`?
9. Почему синхронизация потоков выполнения важна в многопоточных приложениях?

Список литературы рекомендуется:

1. Оанча, М. – *Программирование на Java. Практическое руководство*, Издательство Polirom, Яссы, 2019.
2. Молдован, Д. – *Структуры данных и объектно-ориентированное программирование на Java*, Издательство Университета «Бабеш-Бойяй», Клуж-Напока, 2017.
3. Пырву, Д. – *Продвинутые техники программирования на Java*, издательство MatrixRom, Бухарест, 2016.
4. Хорстманн, К., Корнелл, Г. – *Java. Библиотека профессионала. Том 1, Основы*, 11-е изд., СПб.: Питер, 2022.
5. Эккель, Б. – *Философия Java*, 4-е изд., СПб.: Питер, 2018.
6. Шилдт, Г. – *Java. Полное руководство*, 11-е изд., Москва: Вильямс, 2021.
7. Герберт Шильдт – *Java: Полное руководство*, 12-е изд., McGraw-Hill, 2022.
8. Кей С. Хорстманн – *Core Java, Volume I–Fundamentals*, 12-е изд., Pearson, 2021.
9. Брайан Гоэц и др. – *Java Concurrency in Practice*, Addison-Wesley, 2006.