Java Pro 6.1

TEL-RAN
by Starta Institute

Введение в многопоточность

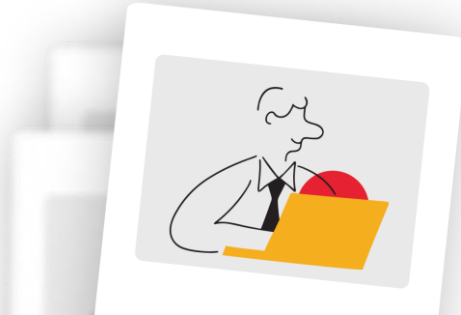
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

Повторение

TEL-RAN
by Starta Institute

1 Для чего нужен Stream API?

2 Какие типы операций есть в стримах и чем они отличаются?

Повторение

TEL-RAN
by Starta Institute

1 Для чего нужен Stream API?

2 Какие типы операций есть в стримах и чем они отличаются?

1 Для написания более короткого кода благодаря функциональному стилю: вместо набора циклов и условий происходит выполнения типовых методов Stream с учётом переданных в них лямбда-выражений.

2 Существуют промежуточные и терминальные операции. Промежуточные операции получают стрим, воздействуют на его элементы и на выходе возвращают новый стрим. Терминальные операции завершают стрим, превращая стрим в некоторый результат (коллекцию, строку, число и т.д.). Поток начинает выполнение в момент вызова терминальной операции.

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
System.out.println("Numbers:");
int sum = Stream.of(1, 2, 3, 4, 5).limit(3).peek(System.out::println).reduce(Integer::sum).get();
System.out.println("Sum: " + sum);
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
System.out.println("Numbers:");
int sum = Stream.of(1, 2, 3, 4, 5).limit(3).peek(System.out::println).reduce(Integer::sum).get();
System.out.println("Sum: " + sum);
```

Numbers:

1

2

3

Sum: 6

Повторение

TEL-RAN
by Starta Institute

Задача: Проверьте, содержит ли список строк хотя бы одну строку с длиной более 10 символов с использованием Stream API. Какие нужно поставить методы вместо *abc* и *xyz*

```
List<String> words = Stream.of("apple", "banana", "kiwi", "watermelon")
        .abc(s -> s.length() > 10)
        .xyz();
```

Повторение

TEL-RAN
by Starta Institute

Задача: Проверьте, содержит ли список строк хотя бы одну строку с длиной более 10 символов с использованием Stream API. Какие нужно поставить методы вместо abc и xyz

```
List<String> words = Stream.of("apple", "banana", "kiwi", "watermelon")
        .filter(s -> s.length() > 10)
        .toList();
```

Повторение

TEL-RAN
by Starta Institute

Задача: Выберите из списка три уникальных максимальных числа.

Какие нужно поставить методы вместо *xxx, yyy* и *zzz?*

```
Set<Integer> max3 = List.of(10, 55, 32, 23, 14, 55).stream()
        .xxx()
        .yyy(Comparator.reverseOrder())
        .limit(3)
        .zzz(Collectors.toSet());
```

Повторение

TEL-RAN
by Starta Institute

Задача: Выберите из списка три уникальных максимальных числа.

Какие нужно поставить методы вместо *abc* и *xyz*

```
Set<Integer> max3 = List.of(10, 55, 32, 23, 14, 55).stream()
        .distinct()
        .sorted(Comparator.reverseOrder())
        .limit(3)
        .collect(Collectors.toSet());
```

Повторение

TEL-RAN
by Starta Institute

Преобразуйте список строк в карту, где ключом является сама строка, а значением - длина строки с использованием Stream API.

```
List<String> strs = List.of("java", "stream", "api");
```

Повторение

TEL-RAN
by Starta Institute

Преобразуйте список строк в карту, где ключом является сама строка, а значением - длина строки с использованием Stream API.

```
List<String> strs = List.of("java", "stream", "api");
Map<String, Integer> strToLength = strs.stream().collect(Collectors.toMap(s -> s, String::length));
```

{java=4, stream=6, api=3}

Повторение

TEL-RAN
by Starta Institute

Объедините два списка строк в один список без повторений с использованием Stream API.

```
List<String> list1 = List.of("apple", "banana", "kiwi");
List<String> list2 = List.of("banana", "orange", "grape");
```

Повторение

TEL-RAN
by Starta Institute

Проверьте, все ли строки из списка содержат только буквы с использованием Stream API.

```
List<String> list1 = List.of("apple", "banana", "kiwi");
List<String> list2 = List.of("banana", "orange", "grape");
List<String> common = Stream.concat(list1.stream(), list2.stream()).distinct().toList();
```

[apple, banana, kiwi, orange, grape]

Повторение

TEL-RAN
by Starta Institute

Проверьте, все ли строки из списка содержат только буквы с использованием Stream API.

```
List<String> wordList = List.of("apple", "kiwi", "123", "orange");
boolean isAllAlphabetic = ???;
```

Повторение

TEL-RAN
by Starta Institute

Проверьте, все ли строки из списка содержат только буквы с использованием Stream API.

```
List<String> wordList = List.of("apple", "kiwi", "123", "orange");
boolean isAllAlphabetic = wordList.stream()
        .flatMapToInt(String::chars)
        .allMatch(Character::isAlphabetic);
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

```
Map<Integer, List<String>> groupedByLength = Stream.of("apple", "kiwi", "banana", "orange", "grape")
        .collect(Collectors.groupingBy(String::length));
System.out.println(groupedByLength);
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

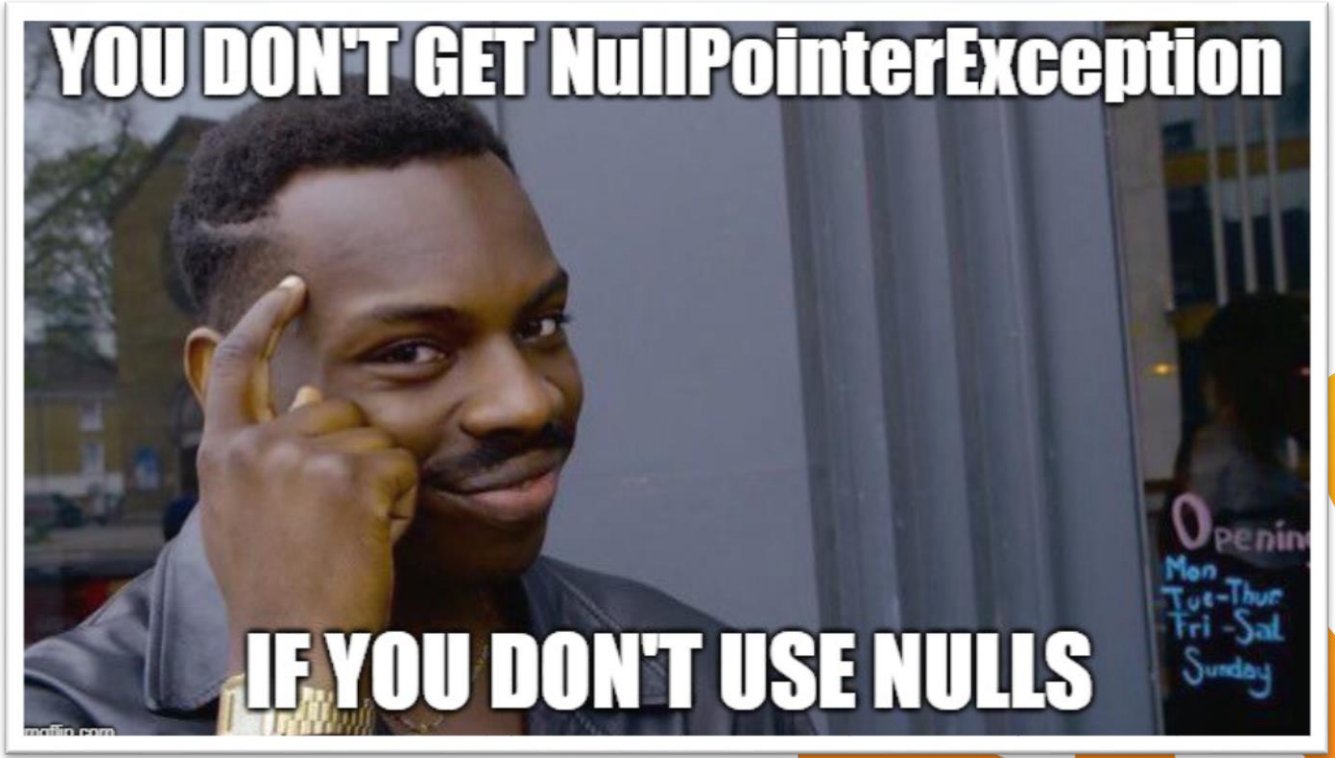
```
Map<Integer, List<String>> groupedByLength = Stream.of("apple", "kiwi", "banana", "orange", "grape")
        .collect(Collectors.groupingBy(String::length));
System.out.println(groupedByLength);
```

{4=[kiwi], 5=[apple, grape], 6=[banana, orange]}

Повторение

TEL-RAN
by Starta Institute

В чём прикол мема?



TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Выносить ребёнка за месяц
- Сколько ниточке ни виться
- Много потоков – много проблем

Проблема

TEL-RAN
by Starta Institute

Говорят, что девять женщин не могут выносить одного ребёнка за месяц. Развитие малыша является строго последовательным процессом, в котором все необходимые должны быть выполнены дин за другим. К таким процессам относится выполнения обычных программ – инструкция за инструкцией.
Но в реальной жизни огромное количество примеров параллельных процессов. Например, кассы в супермаркете, которые «берут» покупателей из очереди и обслуживают их одновременно.

Как организовать параллельные процессы в программе?

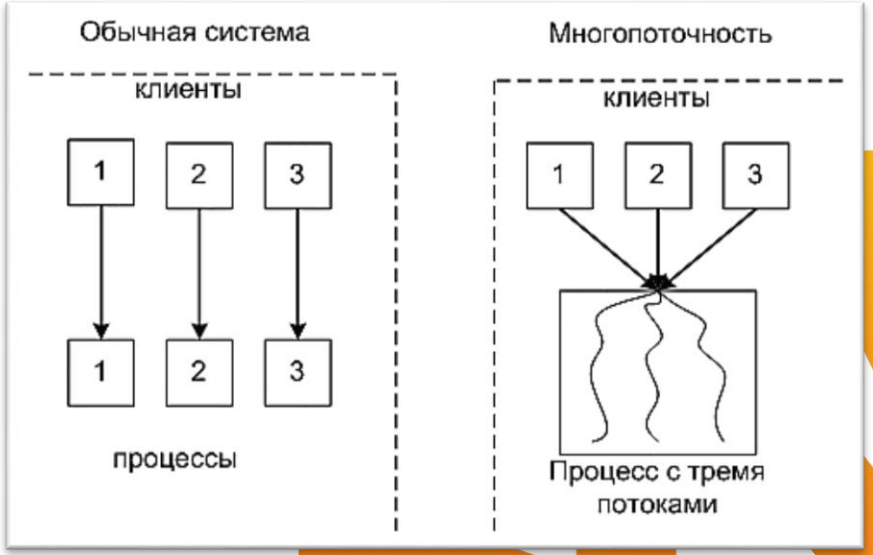
Выносить ребёнка за месяц

Процессы и потоки

TEL-RAN
by Starta Institute

Процесс или задача (process) – программа, находящаяся в режиме выполнения в операционной системе (ОС). Для выполнения Java-программы обычно запускается процесс работы JVM. С каждым процессом связывается его адресное пространство, которого он может читать и в которое он может писать данные, поэтому процессы не влияют друг на друга в ОС. Процессы могут запускать друг друга (взаимные запуски программ).

Поток или нить (thread) – это подпроцесс, порождённый программой. Он выполняется в рамках и с помощью ресурсов, выделенных процессу. Эти ресурсы делятся между всеми подпроцессами.



Выносить ребёнка за месяц

Преимущества использования потоков перед процессами

TEL-RAN
by Starta Institute

1. Повышение производительности самой программы, т.к. есть возможность одновременно выполнять вычисления на разных процессорах или, например, делать вычисления и операции ввода/вывода.
2. Потоки намного легче процессов поскольку требуют меньше времени и ресурсов.
3. Переключение контекста между потоками намного быстрее, чем между процессами.
4. Намного проще добиться взаимодействия между потоками, чем между процессами благодаря общему адресному пространству. Потоки могут взаимодействовать друг с другом через основной «родительский» поток, из которого они стартованы.

Пример: текстовый редактор с тремя потоками может одновременно взаимодействовать с пользователем, форматировать текст и записывать на диск резервную копию.

Выносить ребёнка за месяц

Модель потока

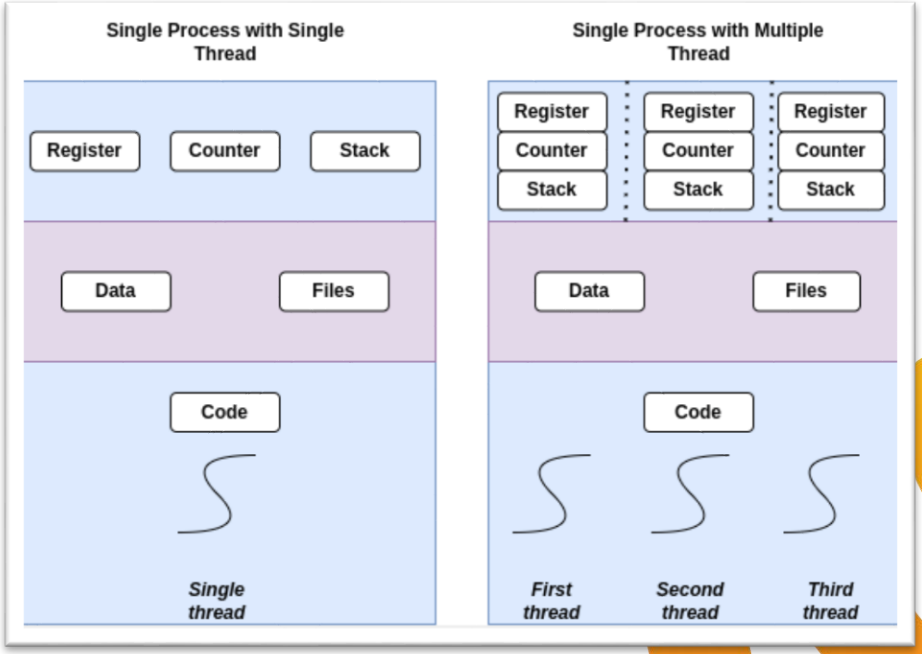
TEL-RAN
by Starta Institute

Потоки делят между собой элементы своего процесса:

- Адресное пространство
- Глобальные переменные
- Открытые файлы
- Таймеры
- Семафоры
- Статистическую информацию.

С каждым потоком связывается:

- Счетчик выполнения команд
- Регистры для текущих переменных
- Стек
- Состояние



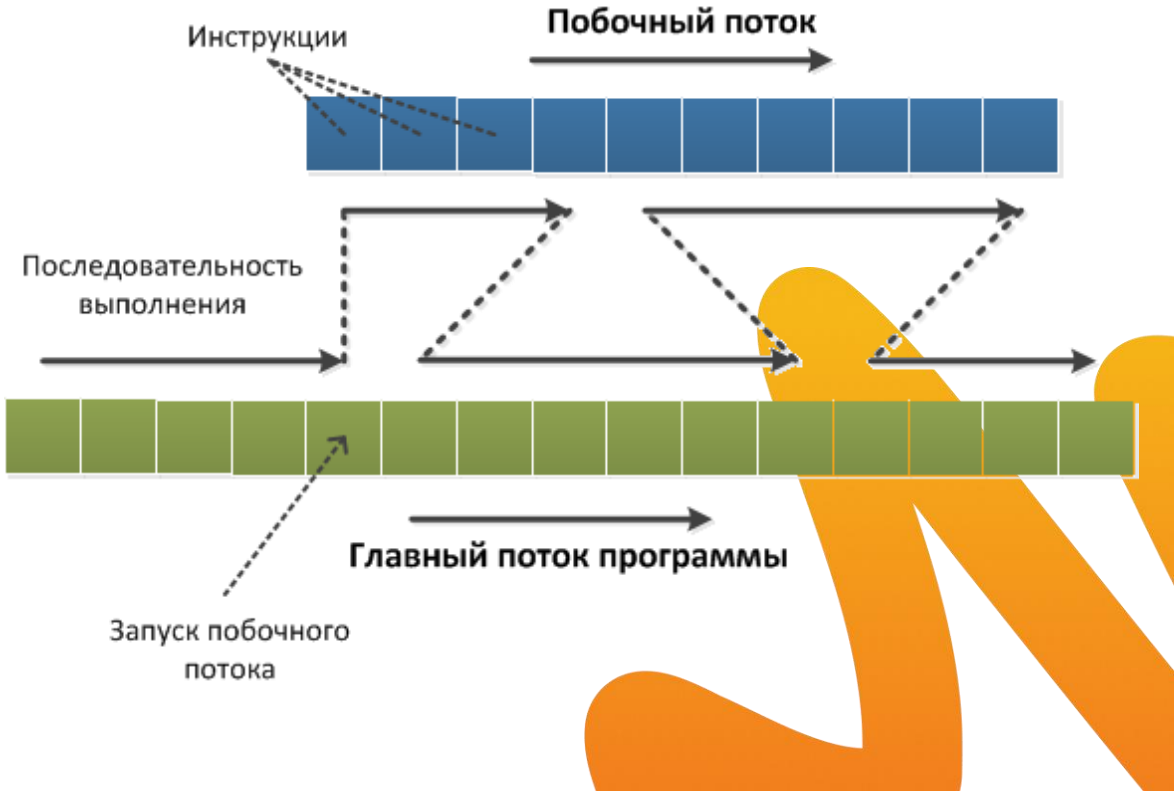
В POSIX и Windows есть поддержка потоков на уровне ядра.

Выносить ребёнка за месяц

Многопоточность

TEL-RAN
by Starta Institute

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.



Выносить ребёнка за месяц

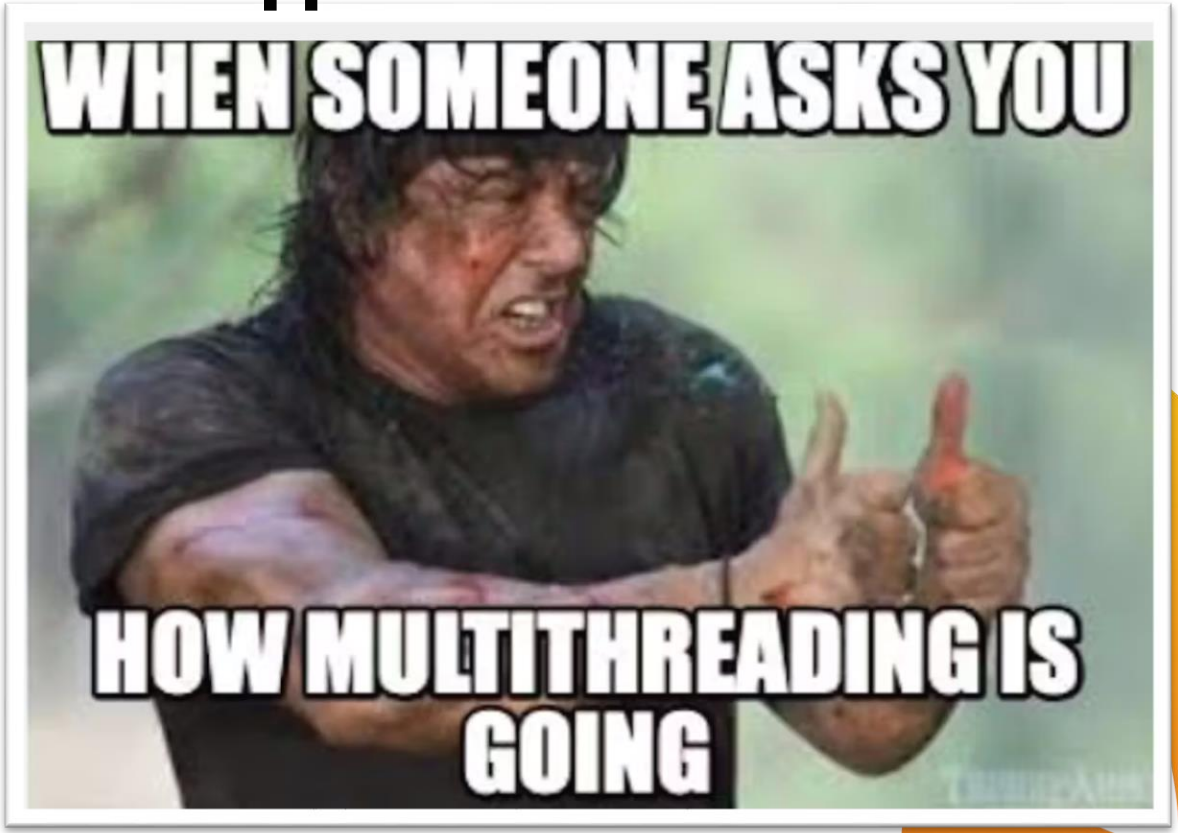
Многопоточность

TEL-RAN
by Starta Institute

Многопоточность (multithreading) – 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.

Первый шаг сделан

TEL-RAN
by Starta Institute



Сколько ниточке ни виться

Способы создания потоков

TEL-RAN
by Starta Institute

В Java существует два основных способа создания потоков:

1. Создать наследника класса *Thread.* В наследнике переопределяется метод *run()*, внутри которого помещается код, который нужно выполнять в отдельном потоке. Этот метод ограничивает разработчика необходимостью включать класс *Thread* в иерархию, где, возможно, ему не место.
2. Имплементировать интерфейс *Runnable*, и передать экземпляр имплементирующего класса в конструктор *Thread*. Имплементировать *Runnable* можно явно, в виде анонимного класса или в виде лямбда-выражения.

Сколько ниточке ни виться

Конструкторы Thread

TEL-RAN
by Starta Institute

```
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.

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.

Сколько ниточке ни виться

Пример создания Thread

TEL-RAN
by Starta Institute

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

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

Сколько ниточке ни виться

Жизненный цикл потока

TEL-RAN
by Starta Institute

При выполнении программы объект *Thread* может находиться в одном из четырех основных состояний (вложенное перечисление Thread.State):

NEW - поток создан, но еще не запущен;

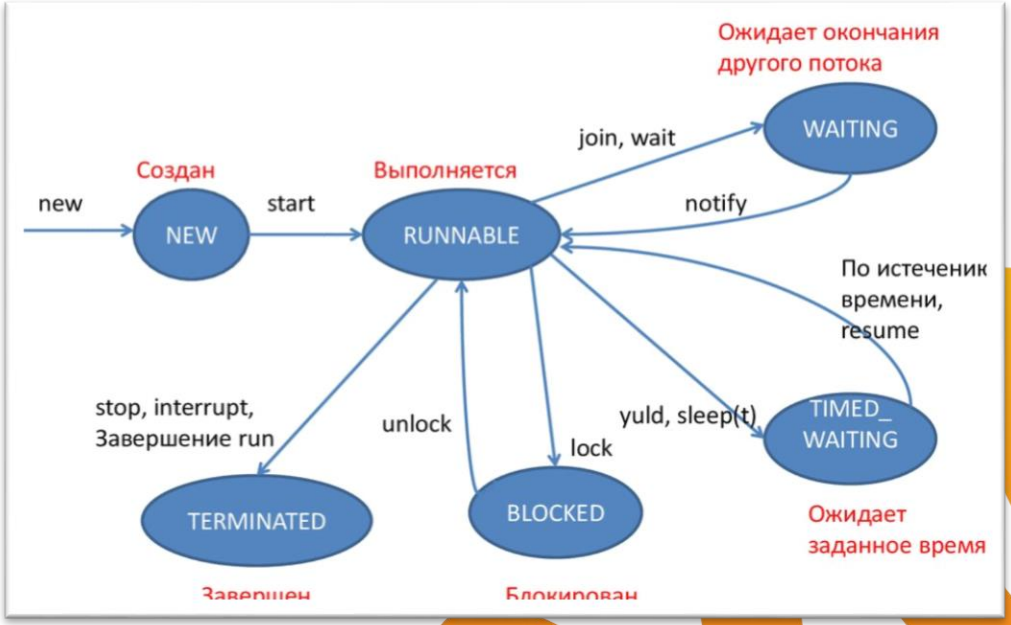
RUNNABLE - поток выполняется;

BLOCKED - поток блокирован;

WAITING - поток ждет окончания работы другого потока;

TIMED_WAITING - поток некоторое время ждет окончания другого потока;

TERMINATED - поток завершен.

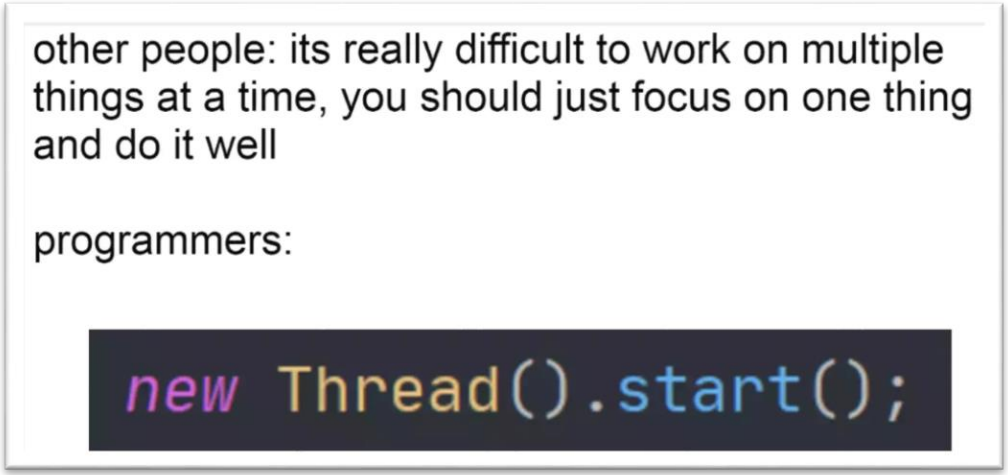


Сколько ниточке ни виться

Жизненный цикл потока

TEL-RAN
by Starta Institute

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()*.



Сколько ниточке ни виться

Методы Thread

TEL-RAN
by Starta Institute

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;

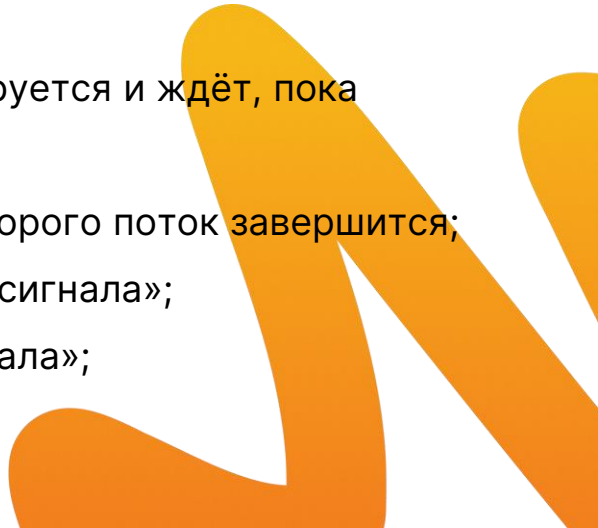
void join() - ожидание завершения потока (текущий поток блокируется и ждёт, пока закончится запущенный из него другой поток);

void join(millis) – то же, с таймаутом millis милисекунд, после которого поток завершится;

void notify() - «пробуждение» отдельного потока, ожидающего «сигнала»;

void notifyAll() - «пробуждение» всех потоков, ожидающих «сигнала»;

void run() – код, который нужно выполнить в потоке;



Сколько ниточке ни виться

Методы Thread

TEL-RAN
by Starta Institute

boolean isDaemon() - проверка, является ли поток фоновым (не требует, чтобы другие потоки ждали его завершения);

void setDaemon(bool) - определение потока как фонового. JVM прекращает работу, как только все не Daemon потоки завершаются.

void setPriority(int) - установка приоритета потока;

void start() - запуск потока. После этого метода поток начинает выполнения кода из метода *run()* в отдельном потоке.

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();*

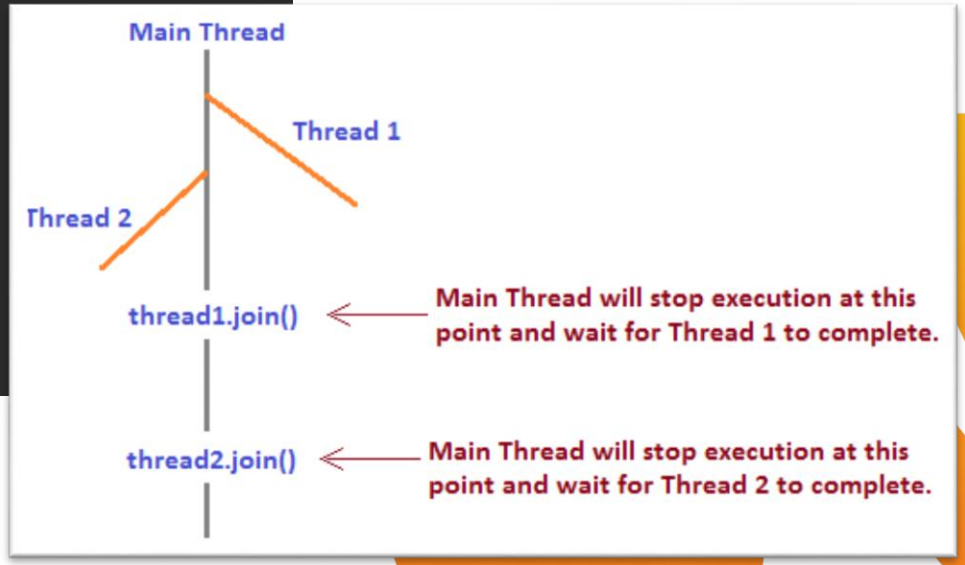
Сколько ниточке ни виться

Метод join()

TEL-RAN
by Starta Institute

Метод выполняет остановку текущего потока до тех пор, пока не завершится запускаемый из него дочерний поток.

```
public static void main(String[] args) {
    Thread thread1 = new Thread(() -> System.out.println("one"));
    Thread thread2 = new Thread(() -> System.out.println("two"));
    thread1.start();
    thread2.start();
    try {
        thread1.join();
        thread2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

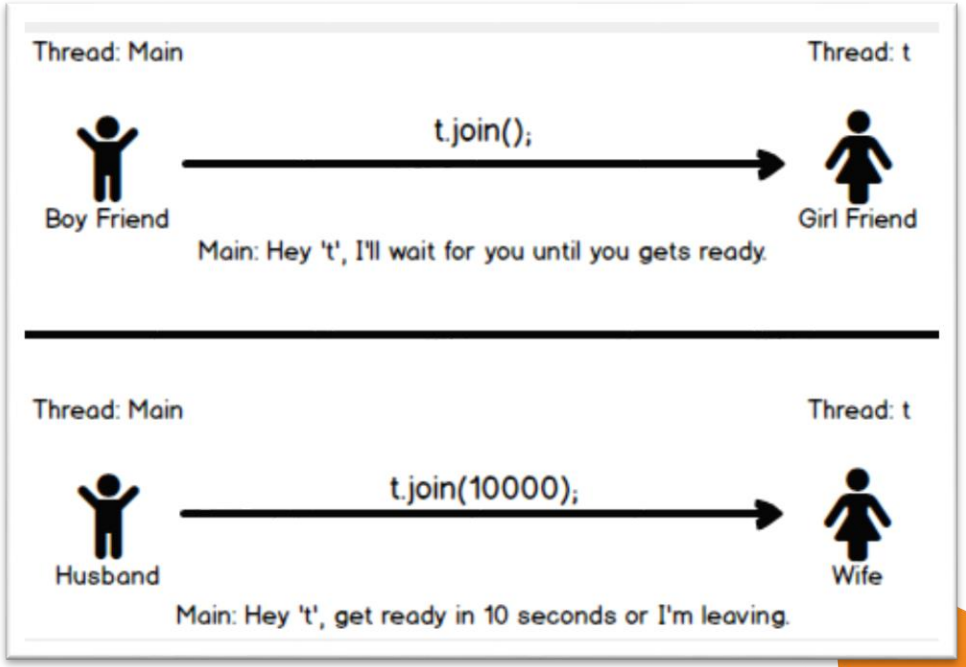


Сколько ниточке ни виться

Метод join()

TEL-RAN
by Starta Institute

У метода существует перегруженная версия, принимающая время timeout, после которого текущий поток перестаёт ждать и продолжает выполнять собственные мнструкции.



Сколько ниточке ни виться

Статические методы Thread

TEL-RAN
by Starta Institute

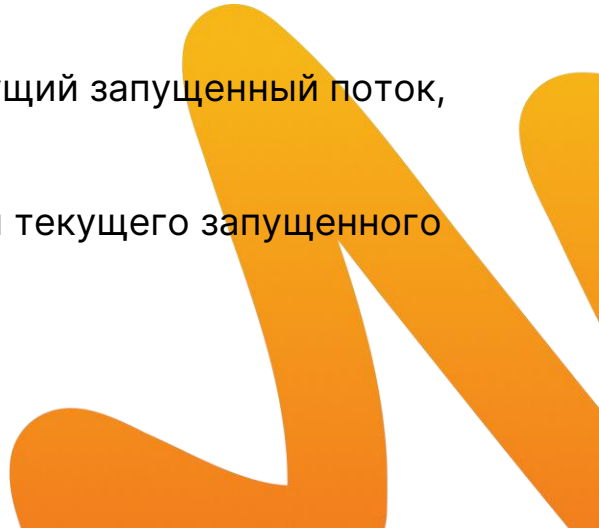
public static void yield() – заставляет текущий запущенный поток уступить место любым другим потокам с тем же приоритетом, которые ожидают планирования.

public static void sleep(long millisec) – блокирует текущий запущенный поток по крайней мере на указанное количество миллисекунд.

public static boolean holdsLock(Object x) – возвращает *true*, если текущий поток удерживает блокировку данного объекта.

public static Thread currentThread() – возвращает ссылку на текущий запущенный поток, который вызывает этот метод.

public static void dumpStack() – выводит отслеживание стека для текущего запущенного потока, что полезно при отладке многопоточного приложения.

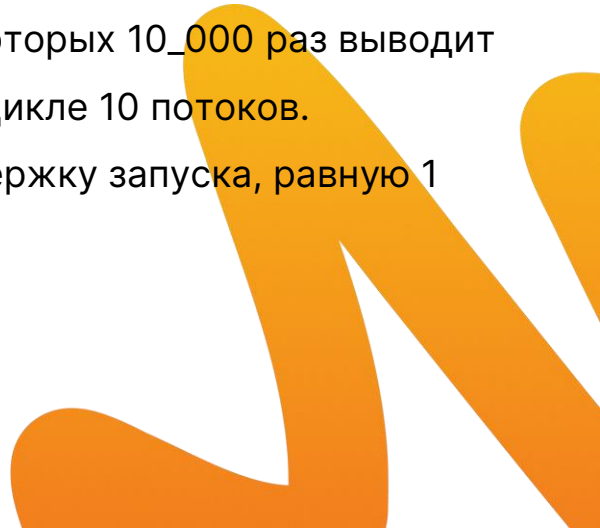


Задание

TEL-RAN
by Starta Institute

1 Напишите программу, которая определит, что появилось раньше – курица или яйцо. Для этого создайте два класса – Курица и Яйцо, которые будут наследоваться от Thread и выводить своё мнение в консоль после случайно заданной задержки. Задержка устанавливается методом *Thread.sleep()*. В основной программе запустите оба потока. Чьё слово будет первым, тот и победил.

2 Используя лямбда-выражения, создайте потоки, каждый из которых 10_000 раз выводит в консоль число – свой номер по порядку запуска. Запустите в цикле 10 потоков. Выводятся ли потоки по порядку? Что будет если добавить задержку запуска, равную 1 мс?



Идём далее

TEL-RAN
by Starta Institute

Проблема

TEL-RAN
by Starta Institute

К сожалению, Java не гарантирует порядок выполнения JVM, поэтому работа потоков является конкурентной (*concurrent*).

Возможно, стоит задать приоритет потокам, и тогда они будут выполняться по порядку?

Много потоков – много проблем

Приоритет потоков

TEL-RAN
by Starta Institute

Каждый поток Java имеет приоритет, который помогает операционной системе определять порядок, в котором планируются потоки.

Приоритеты потоков Java находятся в диапазоне от *MIN_PRIORITY* (константа 1) до *MAX_PRIORITY* (константа 10). По умолчанию каждому потоку устанавливается приоритет *NORM_PRIORITY* (константа 5).

Потоки с более высоким приоритетом более важны для программы, и в первую очередь им должно выделяться процессорное время. Однако приоритеты потоков не могут гарантировать порядок, в котором выполняются потоки, и очень сильно зависят от платформы.

Задание

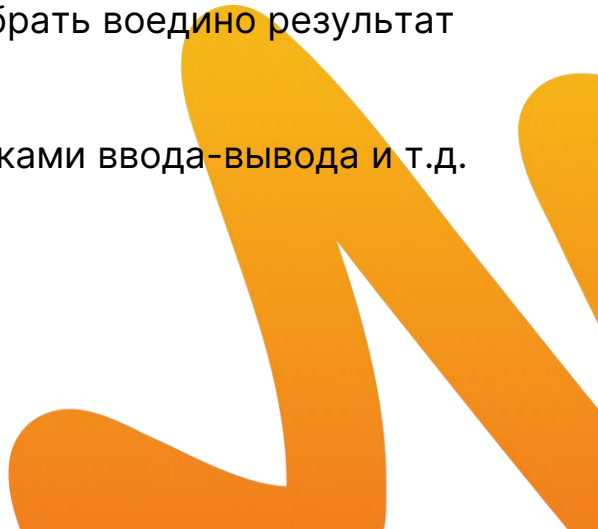
Допишите предыдущую программу (10 потоков) так, чтобы у потоков был установлен приоритет в порядке запуска. Задать потоку приоритет можно с помощью метода setPriority(int).

Проблема

TEL-RAN
by Starta Institute

Часто множество потоков работают с одним и тем же ресурсом. Например, мы можем

- читать файл несколькими потоками ввода-вывода параллельно, чтобы ускорить процесс получения данных;
- производить вычисления полученной информации в программе (параллельные вычисления), когда работаем с большим массивом данных и порядок обработки не важен. Каждому потоку нужно дать чать данных, а потом собрать воедино результат вычислений.
- записывать результат вычислений в файл несколькими потоками ввода-вывода и т.д.



Проблема

TEL-RAN
by Starta Institute

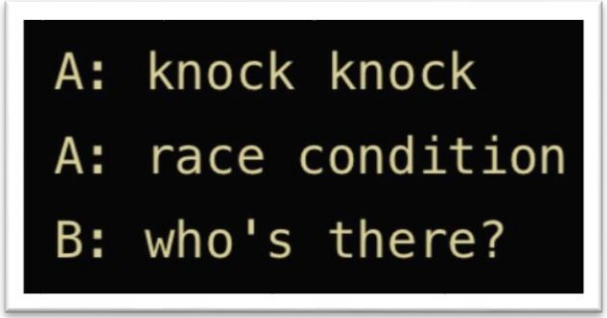
Во всех этих случаях потоки конкурируют за доступ к ресурсу и находятся в состоянии гонки (**race condition**). Это приводит к сильной неопределённости в программе.

Например, один поток вычисляет и записывает значение переменной. Несколько других потоков – используют переменную для своих вычислений. Если первый поток не успел записать данные, то остальные потоки получат ошибку при вычислении.

Проблема

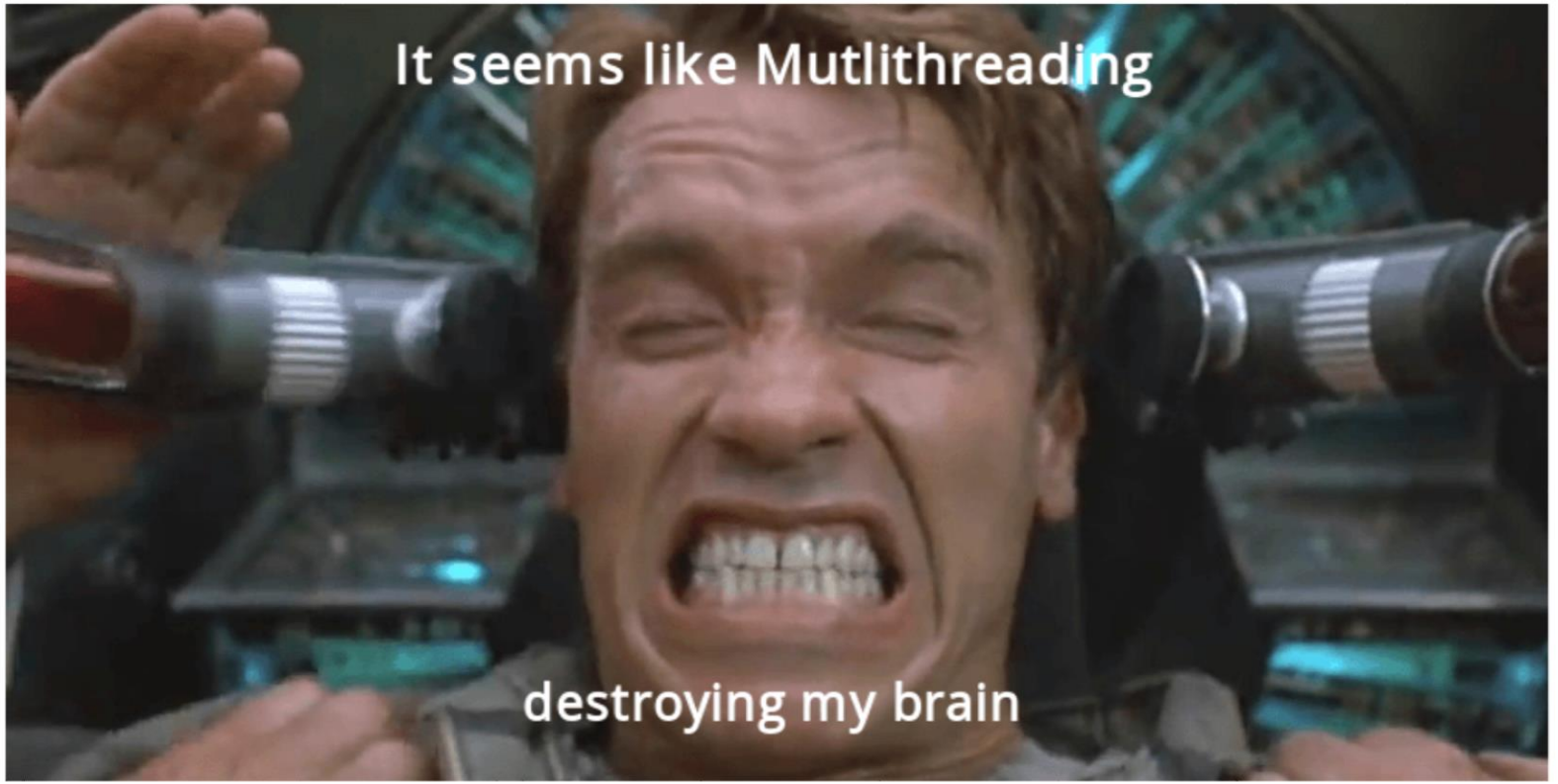
TEL-RAN
by Starta Institute

Race Condition (состояние гонки) – ошибка проектирования многопоточного приложения, при которой работа приложения зависит от того, в каком порядке выполняются части кода.



Идём далее

TEL-RAN
by Starta Institute



Проблема

TEL-RAN
by Starta Institute

Например, представь, что программа отвечает за работу робота, который готовит еду:

Поток-0 достает яйца из холодильника.

Поток-1 включает плиту.

Поток-2 достает сковородку и ставит на плиту.

Поток-3 зажигает огонь на плите.

Поток-4 выливает на сковороду масло.

Поток-5 разбивает яйца и выливает их на сковороду.

Поток-6 выбрасывает скорлупу в мусорное ведро.

Поток-7 снимает готовую яичницу с огня.

Поток-8 выкладывает яичницу в тарелку.

Поток-9 моет посуду.

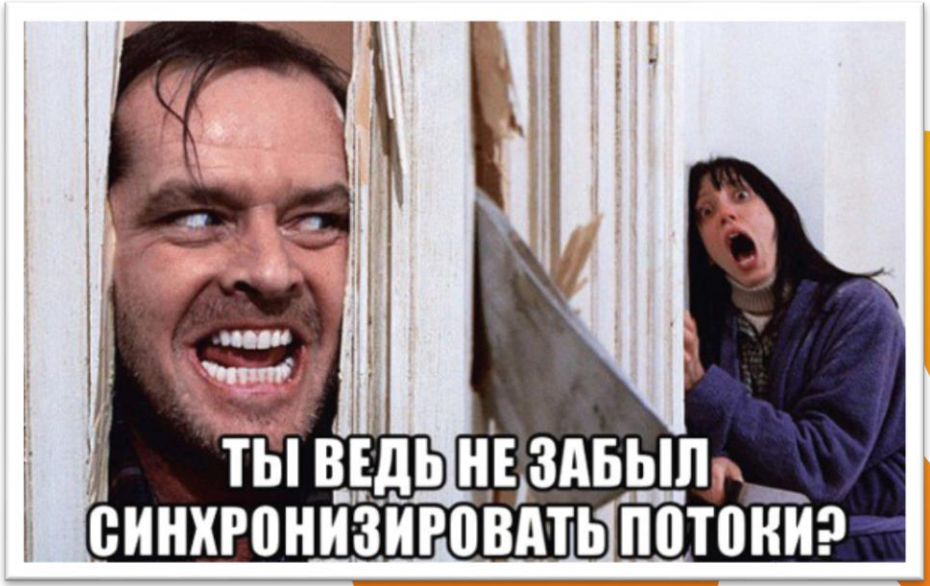
Что произойдёт, если потоки будут выполняться не по порядку? Как избежать такой проблемы?

Много потоков – много проблем

Синхронизация потоков

TEL-RAN
by Starta Institute

Проблемы с использованием общих ресурсов в многопоточном приложении решаются **синхронизацией потоков** (блокировкой ресурсов). Механизм синхронизации обеспечивает последовательный доступ к ресурсам. Выполнение потока не будет продолжено, пока блокировка интересующего поток ресурса не освободится. Для блокировки ресурса используется ключевое слово *synchronized.*



Много потоков – много проблем

Синхронизация потоков

TEL-RAN by Starta Institute

Синхронизированным может быть либо отдельный метод либо блок кода.

```
public void println(String x) {
    if (getClass() == PrintStream.class) {
        writeln(String.valueOf(x));
    } else {
        synchronized (this) {
            print(x);
            newLine();
        }
    }
}
```

```
public synchronized boolean isDestroyed() {
    return destroyed;
}
```

Монитор синхронизируемого метода – текущий объект. Если метод статический, то текущий класс

Монитором блока кода выступает текущий объект. JVM будет хранить признак доступности этого объекта

synchronized говорит о том, что при выполнении блока кода или метода один из потоков завладеет нужным ресурсом (**монитором**) и начнёт его использовать. В это время у ресурса выставляется признак «занято». Все остальные потоки, претендующие на ресурс, будут ждать освобождения ресурса, периодически проверяя признак занятости.

Много потоков – много проблем

Monitor. Mutex. Semaphore

TEL-RAN
by Starta Institute

Семафор - это средство синхронизации доступа к ресурсу. Ограничивает количество потоков, которые могут войти в заданный участок кода. Использует счетчик потоков, который указывает, сколько потоков одновременно могут получать доступ к общему ресурсу.

Мьютекс - поле для синхронизации потоков. Есть у каждого объекта в Java. Это простейший *Семафор*, который может находиться в одном из двух состояний: *true* или *false*.

Монитор - это дополнительная надстройка над Мьютексом. Блокирует объект именно монитор.

Когда один тред заходит внутрь synchronized блока кода, JVM тут же блокирует Мьютекс синхронизированного объекта. Больше ни один тред не сможет зайти в этот блок, пока текущий тред его не покинет.

Много потоков – много проблем

Не требуется синхронизировать

Недостатком использования *synchronized* является то, что другие потоки вынуждены ждать, пока нужный объект или метод освободится. Это создает **bottle neck** (узкое место) в программе, отчего скорость работы может пострадать.

Любая синхронизация снижает скорость выполнения программы, поэтому синхронизировать нужно не всё:

- *final* поля класса инициализируются в его конструкторе – соответственно, корректное значение *final* полей будет видно всем потокам без синхронизации. По этой причине неизменяемые классы прекрасно подходят для многопоточности.
- многие классы уже синхронизированы или имеют соответствующие аналоги. Например, *StringBuilder* имеет синхронизированный аналог *StringBuffer*.

Задание

TEL-RAN
by Starta Institute

Создайте класс MyDate, имеющий поля год, месяц и день. Создайте класс Today. Представьте, что класс Today получает сегодняшнюю дату от сервера точного времени по сети, т.е. с задержкой. Смоделируйте эту ситуацию в методе getTodayDate(). Создайте экземпляр Today, который выполнит getTodayDate() в отдельном потоке. Создайте и запустите два потока, которые берут текущее значение даты из Today и прибавляют случайное значение к году.

Попробуйте синхронизировать работу метода getTodayDate().

Идём далее

TEL-RAN
by Starta Institute

Проблема

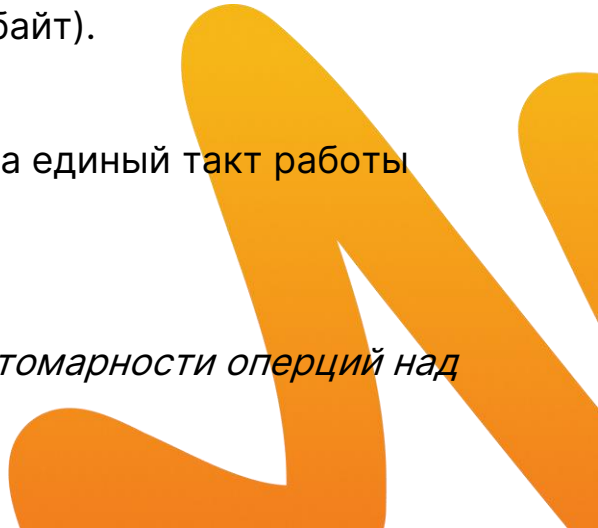
TEL-RAN
by Starta Institute

При взаимодействии с переменной каждый поток хранит ее значение в *своем стеке*. Может возникнуть ситуация, что один поток изменит значение общей переменной, а второй поток будет продолжать работать с ее старым значением из своего *кэша*.

Также, в отличие от других примитивных типов данных, операции чтения и записи *long* и *double* не являются **атомарными** из-за их большого размера (8 байт).

Атомарная операция – простейшая операция, выполняющаяся за единый такт работы процессора.

Как решить две проблемы – взаимодействия стеков потоков и атомарности операций над переменными?



Много потоков – много проблем

Модификатор поля volatile

Эти две проблемы решает модификатор *volatile*:

- Операции чтения и записи *volatile* переменной являются атомарными.
- Переменная не будет помещаться в кэш: результат записи значения в *volatile* переменную одним потоком будет виден всем другим потокам, которые используют эту переменную для чтения.

```
public volatile long x;
public volatile double y;
```

Задание

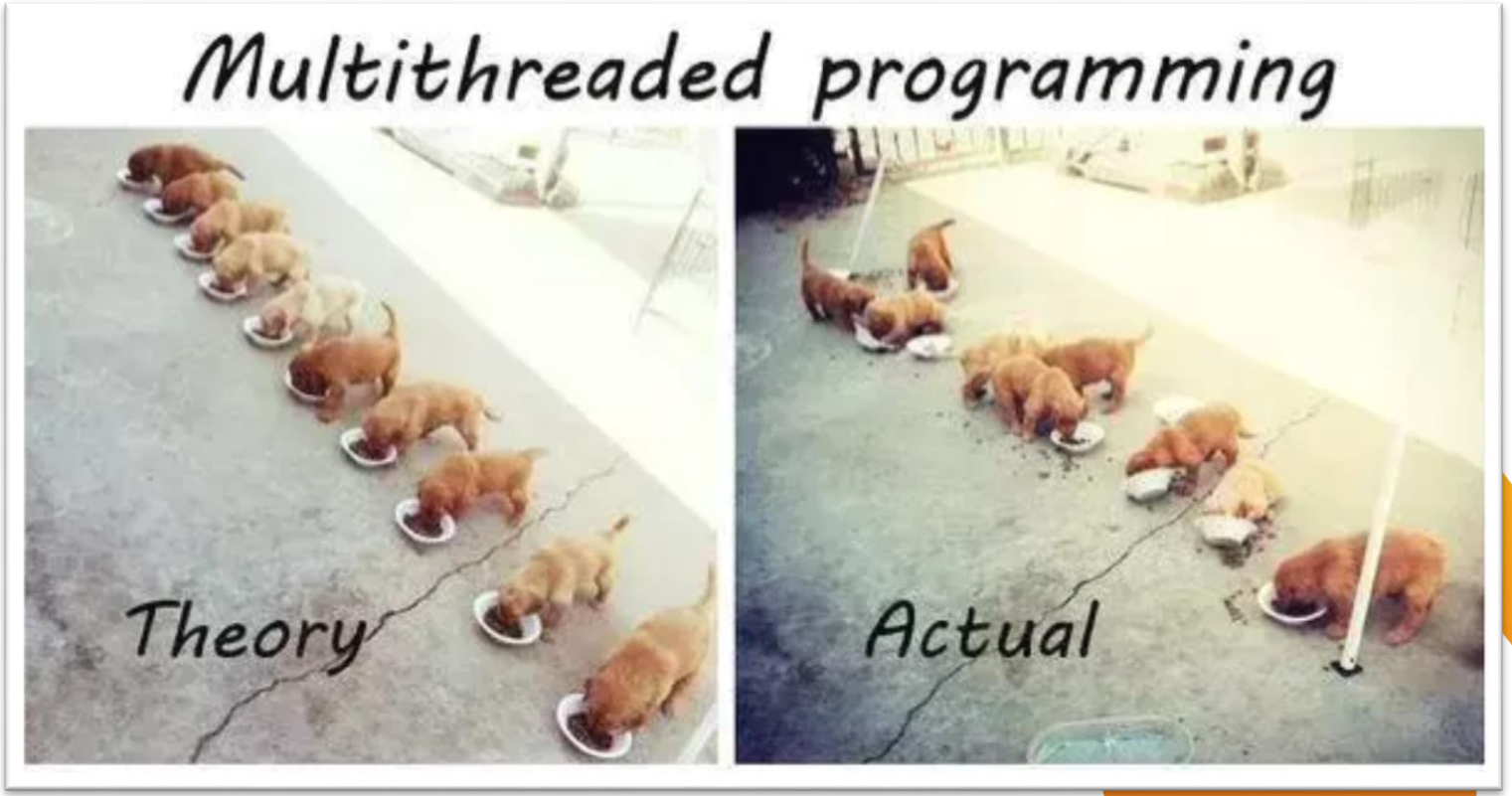
Создайте класс с единственным публичным полем long count.

Создайте экземпляр класса. В цикле запустите потоки, которые увеличивают значение поля на 1 и выводят его в консоль.

Добавьте volatile полю count и повторите программу.

Идём далее

TEL-RAN
by Starta Institute



Проблема

TEL-RAN
by Starta Institute

В реальной жизни есть ситуации, которые мы привыкли называть «порочный круг».

Например,

Ты не можешь устроиться на работу, так как на работу берут только с опытом;

Ты не можешь получить опыт работы, из-за того что не работаешь.

В программировании такие ситуации тоже возникают.

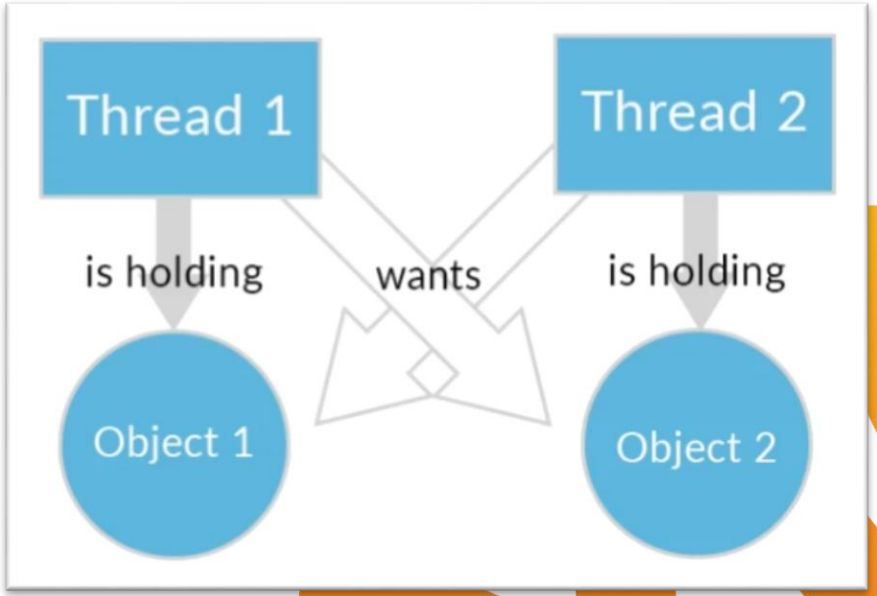
Много потоков – много проблем

Взаимная блокировка

TEL-RAN by Starta Institute

Deadlock - ситуация, при которой несколько потоков находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать выполнение.

Представь, что поток-1 работает с каким-то Объектом-1, а поток-2 работает с Объектом-2.
При этом программа написана так:
Поток-1 перестанет работать с Объектом-1 и переключится на Объект-2, как только Поток-2 перестанет работать с Объектом 2 и переключится на Объект-1.
Поток-2 перестанет работать с Объектом-2 и переключится на Объект-1, как только Поток-1 перестанет работать с Объектом 1 и переключится на Объект-2



Много потоков – много проблем

Взаимная блокировка

TEL-RAN
by Starta Institute

Не все Race condition потенциально производят Deadlock, однако, Deadlock происходят только в Race condition.

Задание

Создайте взаимную блокировку потоков.

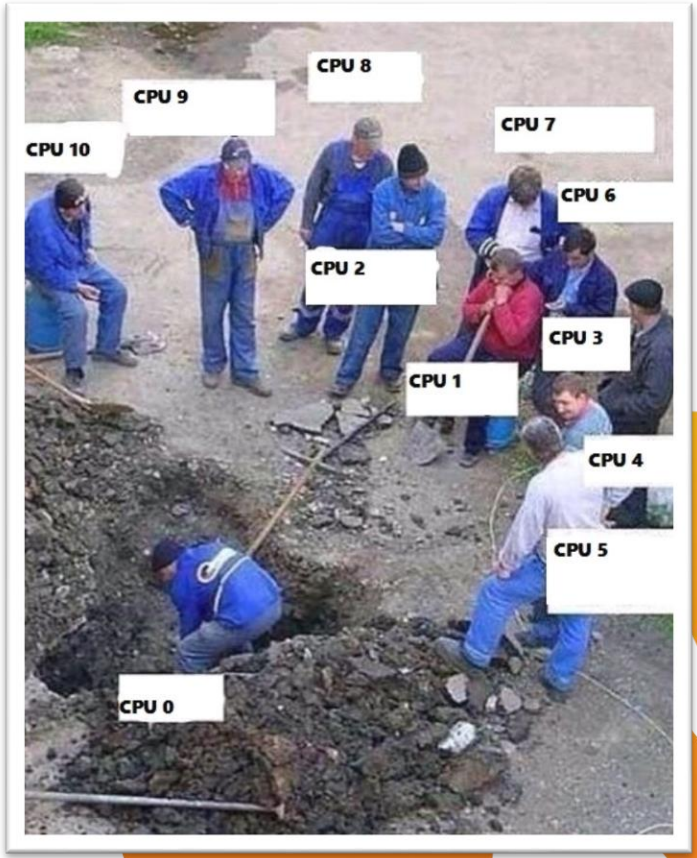
TEL-RAN
by Starta Institute

3 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1 Создайте класс Траншея. У траншеи есть целевая длина и текущая длина. Создайте класс Землекоп, объекты которого копают траншею (увеличивают текущую длину), пока не будет достигнута целевая длина. Каждый землекоп может прокопать 1 м траншеи, а затем он отдыхает 10 секунд. В программе создайте траншею и двух землекопов. Измерьте, за какое время траншею прокопает один землекоп и за какое время с такой же траншеей управятся двое.



Домашнее задание

TEL-RAN
by Starta Institute

2 Напишите программу, которая вычисляет какую-либо сложную функцию для каждого целого числа от 1 до N, N – входной параметр (большое число, например, 10 000 000)

N – ввод с консоли. Результат выводится на экран. Поскольку N – большое, необходимо разбить вычисления на несколько частей и каждую часть вычислить в отдельном потоке параллельно. Для каждой части нужно создать объект Task, внутри которого запомнить данные для начала вычислений, а так же сохранить результат после завершения вычислений. Каждый поток работает со своим объектом Task.

Примеры функций (сходящиеся ряды, удобны тем, что можно проверить результат вычислений программы):

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 2$$

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = \sum_{n=1}^{\infty} \left(\frac{1}{n} - \frac{1}{n+1} \right) = 1$$

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

$$\sum_{n=0}^{\infty} q^n = \frac{1}{1-q}, \text{ где } |q| < 1$$

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute