Java Pro 6.1

TEL-RAN
by Starta Institute

Модели многопоточности. Исключения

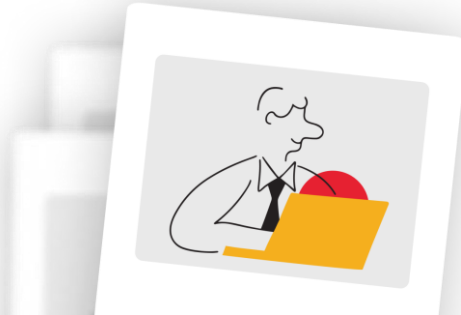
ПРЕПОДАВАТЕЛЬ

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. Чем отличаются “процесс” и “поток”.
2. Дайте определение понятию “синхронизация потоков”.
3. В каких случаях целесообразно создавать несколько потоков?
4. Что вы знаете о главном потоке программы?
5. Какие есть способы создания потоков?
6. Какой метод запускает поток на выполнение?
7. Какой метод описывает действие потока во время выполнения?
8. Когда поток завершает свое выполнение?
9. Как принудительно остановить поток?
10. Дайте определение понятию “поток-демон”.

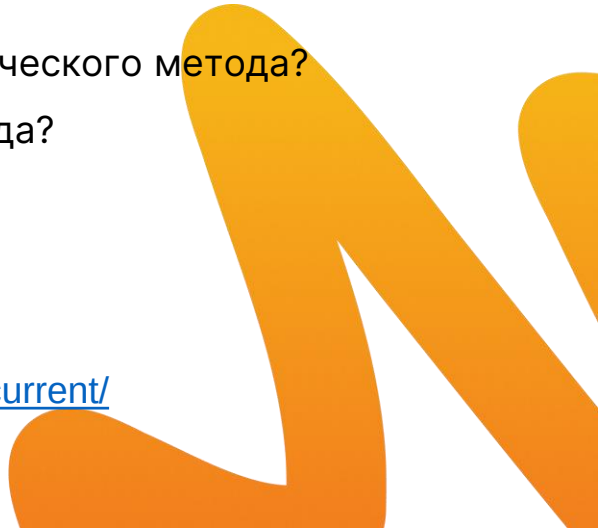
Ответы https://javastudy.ru/interview/concurrent/

Повторение

TEL-RAN
by Starta Institute

11. Как создать поток-демон?
12. Как получить текущий поток?
13. Дайте определение понятию “монитор”.
14. Как приостановить выполнение потока?
15. В каких состояниях может пребывать поток?
16. Что является монитором при вызове нестатического и статического метода?
17. Что является монитором при выполнении участка кода метода?
18. Какой метод переводит поток в режим ожидания?
19. Дайте определение понятию “взаимная блокировка”.
20. Модификаторы volatile и метод yield().

Ответы https://javastudy.ru/interview/concurrent/



Повторение

TEL-RAN
by Starta Institute

Как можно улучшить следующий код?

```
Thread myThread = new Thread(new Runnable() {
    public void run() {
        System.out.println("Hello, Multithreading!");
    }
});
```

Повторение

TEL-RAN
by Starta Institute

Как можно улучшить следующий код?

```
Thread myThread = new Thread(new Runnable() {
    public void run() {
        System.out.println("Hello, Multithreading!");
    }
});
```

Переписать анонимный класс в виде лямбда-выражения, т.к. это более короткий код.

```
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде.

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.run();
```

Повторение

TEL-RAN
by Starta Institute

Исправьте ошибку в коде.

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.run();
```

Запуск потоков осуществляется методом start(). Метод run() хранит, какой код поток должен выполнить после запуска.

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.start();
```

Повторение

TEL-RAN
by Starta Institute

Улучшите следующий код

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.start();
// Останавливаем поток
myThread.stop();
```

Повторение

TEL-RAN
by Starta Institute

Улучшите следующий код

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.start();
// Останавливаем поток
myThread.stop();
```

Остановка потоков выполняется методом interrupt(). Метод stop() помечен как deprecated.

```
// Создаём поток
Thread myThread = new Thread(() -> System.out.println("Hello, Multithreading!"));
// Запускаем поток
myThread.start();
// Останавливаем поток
myThread.interrupt();
```

Повторение

Исправьте ошибки в коде.

```
import java.util.concurrent.ThreadLocalRandom;

public class Main1 {
    public static void main(String[] args) {
        IdGenerator generator = new IdGenerator();
        for (int i = 0; i < 100; i++) {
            Thread myThread = new Thread(generator::getNextId);
            myThread.start();
            myThread.sleep(1000);
        }
    }

    public static class IdGenerator {
        private int lastIdUsed;
        public void getNextId() {
            System.out.println(++lastIdUsed);
        }
    }
}
```

Повторение

Исправьте ошибки в коде.

Метод sleep() является статическим. Его нужно вызывать внутри кода того потока, который нужно «притормозить». Данный метод бросает checked-исключение, которое нужно обработать.

```
import java.util.concurrent.ThreadLocalRandom;

public class Main1 {
    public static void main(String[] args) {
        IdGenerator generator = new IdGenerator();
        for (int i = 0; i < 100; i++) {
            Thread myThread = new Thread(generator::getNextId);
            myThread.start();
        }
    }

    public static class IdGenerator {
        private int lastIdUsed;
        public void getNextId() {
            try {
                Thread.sleep(ThreadLocalRandom.current().nextInt(1000));
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println(++lastIdUsed);
        }
    }
}
```

Повторение

Исправьте ошибки в коде.

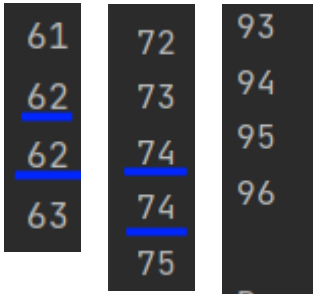
```
import java.util.concurrent.ThreadLocalRandom;

public class Main1 {
    public static void main(String[] args) {
        IdGenerator generator = new IdGenerator();
        for (int i = 0; i < 100; i++) {
            Thread myThread = new Thread(generator::getNextId);
            myThread.start();
        }
    }

    public static class IdGenerator {
        private int lastIdUsed;
        public void getNextId() {
            try {
                Thread.sleep(ThreadLocalRandom.current().nextInt(1000));
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println(++lastIdUsed);
        }
    }
}
```

Повторение

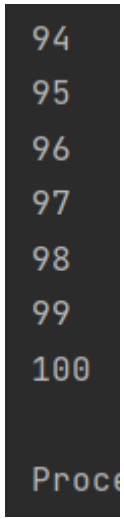
Исправьте ошибки в коде.

Метод getNextId() Должен быть синхронизирован. Вывод в консоль:

До



После




```java
import java.util.concurrent.ThreadLocalRandom;

public class Main1 {
    public static void main(String[] args) {
        IdGenerator generator = new IdGenerator();
        for (int i = 0; i < 100; i++) {
            Thread myThread = new Thread(generator::getNextId);
            myThread.start();
        }
    }

    public static class IdGenerator {
        private int lastIdUsed;
        public synchronized void getNextId() {
            try {
                Thread.sleep(ThreadLocalRandom.current().nextInt(1000));
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println(++lastIdUsed);
        }
    }
}
```

Повторение

В чём прикол мема?

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

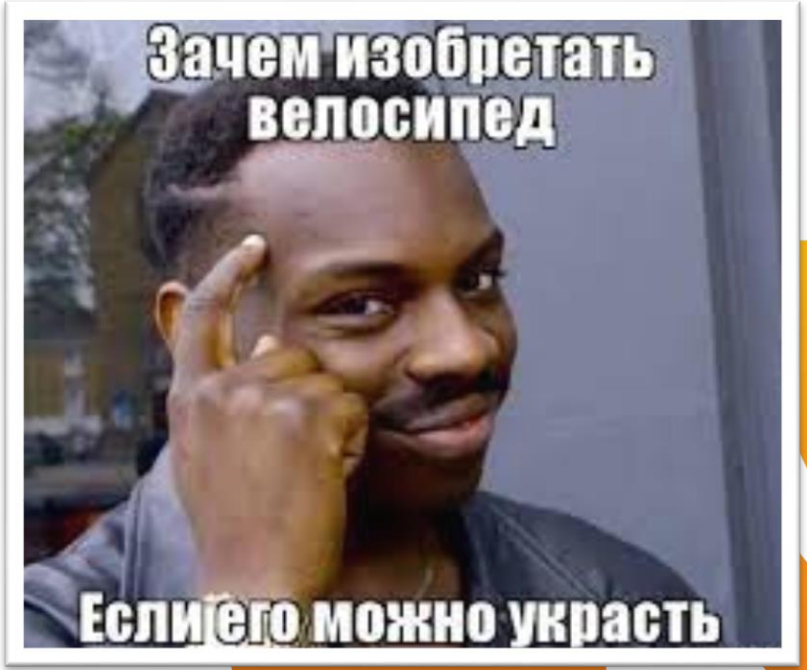
- Клеим модели
- Исключено!

Проблема

TEL-RAN
by Starta Institute

У многопоточных программы есть основные сценарии, связанные с организацией работы с общими ресурсами.

Рассмотрим их, чтобы не изобретать велосипед.



Клеим модели

Производители-потребители

TEL-RAN
by Starta Institute

Один или несколько потоков-производителей создают задания и помещают их в буфер или очередь.
Один или несколько потоков-потребителей извлекают задания из очереди и выполняют их.

Очередь между производителями и потребителями является связанным ресурсом. Это означает, что производители перед записью должны дожидаться появления свободного места в очереди, а потребители должны дожидаться появления заданий в очереди для обработки.

Клеим модели

Производители-потребители

TEL-RAN
by Starta Institute

Координация производителей и потребителей основана на передаче сигналов. Производитель записывает задание и сигнализирует о том, что очередь не пуста. Потребитель читает задание и сигнализирует о том, что очередь не заполнена. Обе стороны должны быть готовы ожидать оповещения о возможности продолжения работы.

Клеим модели

Производители-потребители

TEL-RAN
by Starta Institute

Пример

ThreadModels.zip

Клеим модели

Производители-потребители

TEL-RAN
by Starta Institute

Методы *produce()* и *consume()* отмечены модификатором *synchronized.*

Помещение вызова метода *notify()* в блок синхронизации связано с тем, что когда поток вызывает метод *notify(),* он оповещает другие потоки, ожидающие на том же объекте блокировки, о том, что они могут продолжить выполнение. Однако, чтобы вызвать *notify(),* поток должен владеть монитором (блокировкой) этого объекта.

Если вызов *notify()* не находится внутри синхронизированного блока, то может возникнуть гонка за ресурсами (*race condition*). Например, если один поток вызовет *notify(),* а другой поток в это время пытается войти в критическую секцию, возможна ситуация, когда *notify()* будет выполнен, но монитор еще не освобожден, и второй поток не сможет войти в критическую секцию.

С использованием блока синхронизации гарантируется, что поток, вызывающий *notify(),* владеет монитором объекта, и другие потоки будут ждать входа в критическую секцию, прежде чем они смогут продолжить выполнение.

Задание

TEL-RAN
by Starta Institute

Напишите программу о бабушках/дедушках и внуках. Каждая бабушка и каждый дедушка периодически кладут несколько конфет в специальную корзинку. Внуки иногда заглядывают в корзинку, чтобы взять конфетку. Если в корзинке нет конфет, то внуки ждут, пока они появятся. В корзинку помещается не больше 10 конфет. Когда корзинка полная, то бабушки и дедушки не могут добавить в неё новые конфеты.

Клеим модели

Читатели-писатели

TEL-RAN
by Starta Institute

Если в системе имеется общий ресурс, который в основном служит источником информации для потоков-«читателей», но время от времени обновляется потоками-«писателями», на первый план выходит проблема оперативности обновления.

Если обновление будет происходить недостаточно часто, это может привести к зависанию и накоплению устаревших данных. С другой стороны, слишком частые обновления влияют на производительность.

Клеим модели

Читатели-писатели

TEL-RAN
by Starta Institute

Координация работы читателей так, чтобы они не пытались читать данные, обновляемые писателями, и наоборот, - весьма непростая задача. Писатели обычно блокируют работу многих читателей в течение долгого периода времени, а это отражается на производительности.

Проектировщик должен найти баланс между потребностями читателей и писателей, чтобы обеспечить правильный режим работы, нормальную производительность системы и избежать зависания. В одной из простых стратегий писатели дожидаются, пока в системе не будет ни одного читателя, и только после этого выполняют обновление.

Клеим модели
Читатели-писатели

TEL-RAN
by Starta Institute

Пример

ThreadModels.zip

Задание

TEL-RAN
by Starta Institute

Вы разрабатываете систему управления информацией в базе данных. Реализуйте модель "читатели-писатели" для безопасного доступа к данным. Создайте классы Database, Reader, и Writer. Читатели считывают данные из базы, а писатели записывают новые данные.

Клеим модели

Обедающие философы

TEL-RAN
by Starta Institute

Представьте нескольких философов, сидящих за круглым столом. Слева у каждого философа лежит вилка, а в центре стола стоит большая тарелка спагетти. Философы проводят время в размышлениях, пока не проголодаются. Проголодавшись, философ берет вилки, лежащие по обе стороны, и приступает к еде.

Для еды необходимы две вилки. Если сосед справа или слева уже использует одну из необходимых вилок, философу приходится ждать, пока сосед закончит есть и положит вилки на стол. Когда философ поест, он кладет свои вилки на стол и снова погружается в размышления.

Клеим модели

Обедающие философы

TEL-RAN
by Starta Institute

Заменив философов программными потоками, а вилки - ресурсами, мы получаем задачу, типичную для многих корпоративных систем, в которых приложения конкурируют за ресурсы из ограниченного набора.

Если небрежно отнестись к проектированию такой системы, то конкуренция между потоками может привести к возникновению взаимных блокировок, обратимых блокировок, падению производительности и эффективности работы.

Клеим модели

Обедающие философы

TEL-RAN
by Starta Institute

Пример

ThreadModels.zip

Задание

TEL-RAN
by Starta Institute

Вы разрабатываете систему управления задачами в проекте. Философы представляют собой участников проекта, а вилки - ресурсы (например, рабочие часы разработчика и тестировщика). Реализуйте модель "обедающих философов", где каждый участник проекта (философ) должен получать доступ к двум ресурсам (вилкам) для выполнения задач.

Проблема

TEL-RAN
by Starta Institute

Многопоточность тесно связана с обработкой исключений.

Каким образом выбрасываются и обрабатываются исключения в многопоточной программе.

Исключено!

Исключения в многопоточности

TEL-RAN
by Starta Institute

Основная особенность исключений в многопоточной среде в том, что исключение одного потока могут быть выброшены в этом потоке и остановить его. Остальные потоки при этом могут даже не узнать о выброшенном исключении и, соответственно, не реагируют адекватно на сложившуюся ситуацию. Обработку исключений нужно делать в том потоке, где исключение выброшено.

Исключено!

InterruptedException

TEL-RAN
by Starta Institute

Наиболее распространённое исключение в многопоточных программах. Его выбрасывают методы wait(), sleep() и join(). Так поток сообщает, что он был в ожидании, но в этот момент его прервали.

Исключено!
InterruptedException

TEL-RAN by Starta Institute

При получении статуса потока с помощью метода *isInterrupted()* следует учитывать, что если мы обрабатываем в цикле исключение *InterruptedException* в блоке *catch*, то при перехвате исключения статус потока автоматически сбрасывается, и после этого *isInterrupted* будет возвращать *false*.

```
class MyThread implements Runnable {
    public void run(){
        System.out.printf("%s started... \n",
Thread.currentThread().getName());
        int counter=1; // счетчик циклов
        while(!Thread.currentThread().isInterrupted()){
            System.out.println("Loop " + counter++);
        }
        System.out.printf("%s finished... \n",
Thread.currentThread().getName());
    }
}
```

```
public class Program {

    public static void main(String[] args) {
        System.out.println("Main thread started...");
        MyThread myThread = new MyThread();
        Thread t = new Thread(myThread,"MyThread");
        t.start();
        try{
            Thread.sleep(150);
            t.interrupt();
            Thread.sleep(150);
        }
        catch(InterruptedException e){
            System.out.println("Thread has been interrupted");
        }
        System.out.println("Main thread finished...");
    }
}
```

Исключено!
IllegalMonitorStateException

TEL-RAN
by Starta Institute

Если у в программе есть монитор, на котором мы хотим выполнить синхронизацию, то *IllegalMonitorStateException* выбрасывается, чтобы указать, что поток пытался дождаться или уведомить другие потоки, ожидающие на этом мониторе, не владея им.

Проще говоря, мы получим это исключение, если вызовем один из методов wait(), notify() или notifyAll() класса Object вне синхронизированного блока.

Задание

TEL-RAN
by Starta Institute

Выполните задание из TODO-комментария.

ThreadExceptionTask.zip

TEL-RAN
by Starta Institute

3 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

Многопоточное производство. Написать реализовать схему производства - потребления ресурсов.

1. Есть producerCount потоков, производящих ресурс. Каждый производитель производит 1 единицу ресурса в producerTime времени.
2. Производители помещают ресурсы на склад, размера storageSize. Больше, чем storageSize ресурсов, на складе хранить нельзя.
3. Есть consumerCount потоков, потребляющих ресурсы. Каждый потребитель забирает со склада 1 единицу ресурса в consumerTime времени.
4. Если склад пустой, потоки-потребители ждут. Как только ресурсы появились, потоки сразу же начинают их потреблять.
5. Если склад переполнен, потоки-производители ждут, пока потребители не освободят место.

Домашнее задание

TEL-RAN
by Starta Institute

6. producerCount, consumerCount, producerTime, consumerTime, storageSize – конфигурируются в .properties-файле
7. Ресурс имеет уникальный идентификатор для отслеживания
8. Производители и потребители так же имеют идентификаторы (порядковые номера)
9. Писать в лог (консоль) сообщение: время, номер и тип потока, id-ресурса, событие (произведен или потреблен).
10. Писать в лог сообщение: количество ресурсов на складе при доставке/потреблении на склад.
11. Писать в лог сообщение: время, номер и тип потока, событие (когда потоки переходят в режим ожидания или возобновляют работу).
12. В задаче нельзя использовать java.util.concurrency

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute