# Лабораторная работа № 3

## Тема работы:

Синхронизация потоков при помощи методов класса.

## Теоретическая часть:

Методы класса **java.lang.Thread**:

- **public static Thread currentThread()**

Возвращает ссылку на объект текущего выполняющегося потока.

- **public static void yield()**

Подсказка планировщику, что текущий поток готов уступить свое текущее использование процессора. Планировщик может игнорировать эту подсказку.

Yield - это эвристическая попытка улучшить относительное продвижение между потоками, которые в противном случае будут чрезмерно использовать процессор. Его использование должно сочетаться с детальным профилированием и бенчмаркингом, чтобы убедиться, что он действительно дает желаемый эффект.

Использование этого метода редко бывает целесообразным. Он может быть полезен для целей отладки или тестирования, где может помочь воспроизвести ошибки, связанные с условиями гонки. Он также может быть полезен при разработке конструкций управления параллелизмом, таких как конструкции из пакета **java.util.concurrent.locks**.

- **public static void sleep(long millis) throws InterruptedException**
- **public static void sleep(long millis, int nanos) throws InterruptedException**

Заставляет текущий выполняющийся поток заснуть (временно прекратить выполнение) на указанное количество миллисекунд (плюс указанное количество наносекунд), в зависимости от точности и аккуратности системных таймеров и планировщиков. При этом поток не теряет права собственности на какие-либо мониторы.

- **public void interrupt()**

Прерывает данный поток.

Если текущий поток не прерывает сам себя, что всегда разрешено, то вызывается метод checkAccess этого потока, что может привести к выбросу SecurityException.

Если этот поток блокируется при вызове методов wait(), wait(long) или wait(long, int) класса Object или методов join(), join(long), join(long, int), sleep(long) или sleep(long, int)

этого класса, то его статус прерывания будет очищен и он получит исключение InterruptedException.

Если этот поток блокируется при выполнении операции ввода-вывода по прерываемому каналу (InterruptibleChannel), то канал будет закрыт, статус прерывания потока будет установлен, а сам поток получит исключение ClosedByInterruptException.

Если этот поток заблокирован в селекторе, то статус прерывания потока будет установлен, и он немедленно вернется из операции выбора, возможно, с ненулевым значением, как если бы был вызван метод wakeup селектора.

Если ни одно из предыдущих условий не выполняется, то статус прерывания этого потока будет установлен.

Прерывание неживого потока не должно иметь никакого эффекта.

- **public static boolean interrupted()**

Проверяет, был ли прерван текущий поток. Статус прерванного потока очищается этим методом. Другими словами, если вызвать этот метод дважды подряд, то второй вызов вернет false (если только текущий поток не будет прерван снова, после того как первый вызов очистит его статус прерывания и до того, как второй вызов его проверит).

Прерывание потока, проигнорированное из-за того, что поток не был жив в момент прерывания, будет отражено возвратом false.

- **public boolean isInterrupted()**

Проверяет, был ли данный поток прерван. Данный метод не влияет на статус прерванного потока.

Прерывание потока, проигнорированное из-за того, что поток не был жив в момент прерывания, будет отражено этим методом с возвратом false.

- **public final boolean isAlive()**

Проверяет, жив ли данный поток. Поток жив, если он был запущен и еще не умер.

- **public static int activeCount()**

Возвращает оценку количества активных потоков в группе потоков текущего потока и его подгруппах. Рекурсивно перебирает все подгруппы в группе потоков текущего потока.

Возвращаемое значение является лишь приблизительным, поскольку количество потоков может динамически изменяться при обращении к внутренним структурам данных, а также может зависеть от наличия определенных системных потоков. Данный метод предназначен в основном для отладки и мониторинга.

- **public static int enumerate(Thread[] tarray)**

Копирует в указанный массив все активные потоки в группе потоков текущего потока и его подгруппах. Этот метод просто вызывает метод ThreadGroup.enumerate(Thread[]) группы потоков текущего потока.

Приложение может использовать метод activeCount для получения оценки размера массива, однако если массив слишком мал, чтобы вместить все потоки, то лишние потоки будут молча проигнорированы. Если очень важно получить все активные потоки в группе потоков текущего потока и его подгруппах, инвокер должен убедиться, что возвращаемое значение int строго меньше длины массива tarray.

Из-за присущего этому методу состояния гонки рекомендуется использовать его только для отладки и мониторинга.

- **public final void join() throws InterruptedException**
- **public final void join(long millis) throws InterruptedException**
- **public final void join(long millis, int nanos) throws InterruptedException**

Ожидает завершения работы данного потока.

В данной реализации используется цикл вызовов this.wait, обусловленный this.isAlive. При завершении потока вызывается метод this.notifyAll. Приложениям рекомендуется не использовать wait, notify или notifyAll для экземпляров Thread.

- **public final void setDaemon(boolean on)**

Помечает данный поток как поток демона или как пользовательский поток. Виртуальная машина Java завершает работу, когда единственными запущенными потоками являются потоки-демоны.

Этот метод должен быть вызван до запуска потока.

- **public final boolean isDaemon()**

Проверяет, является ли данный поток демоном.

Переведено с помощью www.DeepL.com/Translator (бесплатная версия)

Источник: https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

## Основное задание:

Реализуйте систему многопоточности с синхронизацией, в которой как минимум три потока и два из них имеют какую-либо взаимосвязь

По завершению работы, составьте отчет, в котором должно быть – Ваша фамилия, имя, группа, тема работы, Ваш вариант для реализации задания, краткое описание реализации задания, ссылку на исходный код на GitHub. Исходный код push-ите в вашу ветку в соответствующем репозитории - https://github.com/FCIM-PCD/Practice-Work-RU. Сохранить отчет в формате PDF и отправить на ELSE - https://else.fcim.utm.md/mod/assign/view.php?id=5827.