**Остановить бушующий поток: официальная версия**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 6](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=6), Лекция 9

— Привет, Амиго! Согласись, Элли хорошо придумала с этим Cancel?

— Ага.

— На самом деле нечто подобное существует в классе Thread. Только переменная называется не isCancel, а isInterrupt, и метод остановки, соответственно, не cancel(), а interrupt().

— Да?

— Ага. Вот смотри:

|  |  |
| --- | --- |
| **Код** | **Описание** |
| class Clock implements Runnable {    public void run() {      Thread current = Thread.currentThread();      while (!current.isInterrupted()) {        try {          Thread.sleep(1000);        } catch (InterruptedException e) {          e.printStackTrace();          current.interrupt();        }        System.out.println("Tik");    }    }  } | Т.к. много нитей могут вызвать метод run одного объекта, то объект Clock в своем методе run получает объект вызвавшей его нити («текущей нити»).  Класс Clock (часы) будет писать в консоль раз в секунду слово «Tik», пока переменная **isInterrupt** текущей нити равна false.  Когда переменная **isInterrupt** станет равной **true**, метод **run** завершится. |
| public static void main(String[] args) throws Exception {    Clock clock = new Clock();    Thread clockThread = new Thread(clock);    clockThread.start();    Thread.sleep(10000);    clockThread.interrupt();  } | Главная нить, запускает дочернюю нить – часы, которая должна работать вечно.  Ждет 10 секунд и отменяет задание, вызовом метода interrupt.  Главная нить завершает свою работу.  Нить часов завершает свою работу. |

Более того, в методе **sleep**, который так любят использовать для организации вечного цикла в методе run, есть автоматическая проверка переменной **isInterrupt**. Если нить вызовет метод **sleep**, то этот метод сначала проверит, а не установлена ли для текущей (вызвавшей его нити) переменная **isInterrupt** в true. И если установлена, то метод не будет спать, а выкинет исключение **InterruptedException**.

— А зачем выкидывать исключение? Не лучше ли тоже просто в цикле вместо isCancel подставить isInterrupted()?

— **Во-первых**, не всегда в методе run есть цикл. Метод может состоять просто из двух десятков вызовов других методов. Тогда перед вызовом каждого придется добавлять проверку isInterrupted.

**Во-вторых**, вдруг какой-то метод очень долго исполняется, т.к. делает много разных действий.

**В-третьих**, выкидывание исключения – это не замена проверке isInterrupted, а скорее удобное дополнение. Выкинутое исключение позволяет быстро раскрутить стек вызовов до самого run.

**В-четвертых**, метод sleep часто используют, и, получается, к такому полезному методу неявно добавили не менее полезную проверку. **Вроде бы никто специально проверку не добавлял, а она есть.**Это очень ценно, когда ты используешь много чужого кода и не можешь сам добавить в него проверку.

**В-пятых**, дополнительная проверка не приводит к снижению производительности. Вызов метода sleep значит, что нить должна ничего не делать (спать), поэтому дополнительная работа никому не мешает.

— Серьёзные аргументы.

— И, наконец, последнее: ты можешь в своем методе run вызывать чужой код, к которому у тебя нет доступа (исходников и/или прав их менять). Он может не иметь проверок на isInterrupted, а также перехватывать с помощью try…catch(Exception e) все возникшие исключения.

**Никто не гарантирует, что нить можно остановить. Она может остановиться только сама.**