**Проблема многопоточности — локальный кэш. Volatile**

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

[Уровень 7](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=7), Лекция 5

— Привет, Амиго! Помнишь, Элли тебе рассказывала про проблемы при одновременном доступе нескольких нитей к общему (разделяемому) ресурсу?

— Да.

— Так вот – это еще не все. Есть еще небольшая проблема.

Как ты знаешь, в компьютере есть память, где хранятся данные и команды (код), а также процессор, который исполняет эти команды и работает с данными. Процессор считывает данные из памяти, изменяет и записывает их обратно в память. Чтобы ускорить работу процессора в него встроили свою «быструю» память – кэш.

Чтобы ускорить свою работу, процессор копирует самые часто используемые переменные из области памяти в свой кэш и все изменения с ними производит в этой быстрой памяти. А после – копирует обратно в «медленную» память. Медленная память все это время содержит старые(!) (неизмененные) значения переменных.

И тогда может возникнуть проблема. Одна нить меняет переменную, такую как isCancel или isInterrupted из примера ниже, а вторая нить «не видит» этого изменения, т.к. оно было совершено в быстрой памяти. Это следствие того, что нити не имеют доступа к кэшу друг друга. (Процессор часто содержит несколько независимых ядер и нити физически могут исполняться на разных ядрах.)

Вспомним вчерашний пример:

|  |  |
| --- | --- |
| **Код** | **Описание** |
|  |  |
| class Clock implements Runnable  {  private boolean isCancel = false;  public void cancel()  {  this.isCancel = true;  }  public void run()  {  while (!this.isCancel)  {  Thread.sleep(1000);  System.out.println("Tik");  }  }  } | | | Нить «не знает» о существовании других нитей.  В методе run переменная isCancel при первом использовании будет помещена в кэш дочерней нити. Эта операция эквивалентна коду: |

|  |
| --- |
| public void run()  {  boolean isCancelCached = this.isCancel;  while (!isCancelCached)  {  Thread.sleep(1000);  System.out.println("Tik");  }  }  Вызов метода cancel из другой нити поменяет значение переменной isCancel в обычной (медленной) памяти, но не в кэше остальных нитей. |
|  |

|  |  |
| --- | --- |
| public static void main(String[] args)  {  Clock clock = new Clock();  Thread clockThread = new Thread(clock);  clockThread.start();  Thread.sleep(10000);  clock.cancel();  } |  |

— Ничего себе! А для этой проблемы тоже придумали красивое решение, как в случае с **synchronized**?

— Ты не поверишь!

Сначала думали отключить работу с кэшем, но потом оказалось, что из-за этого программы работают в разы медленнее. Тогда придумали другое решение.

Было придумано специальное ключевое слово **volatile**. Помещение его перед определением переменной запрещало помещать ее значение в кэш. Вернее не запрещало помещать в кэш, а просто принудительно всегда читало и писало ее только в обычную (медленную) память.

Вот как нужно исправить наше решение, чтобы все стало отлично работать:

|  |  |
| --- | --- |
| **Код** | **Описание** |
|  |  |
| class Clock implements Runnable  {  private volatile boolean isCancel = false;  public void cancel()  {  this.isCancel = true;  }  public void run()  {  while (!this.isCancel)  {  Thread.sleep(1000);  System.out.println("Tik");  }  }  } | | | Из-за модификатора volatile чтение и запись значения переменной всегда будут происходить в обычной, общей для всех нитей, памяти. |
|  | | |  |

|  |  |
| --- | --- |
| public static void main(String[] args)  {  Clock clock = new Clock();  Thread clockThread = new Thread(clock);  clockThread.start();  Thread.sleep(10000);  clock.cancel();  } |  |

— И все?

— Да. Просто и красиво.