Pro Java №25

Потоки-демоны (daemon threads) – это потоки, которые работают в фоновом режиме и завершают выполнение, когда все обычные (недемоны) потоки завершили свою работу. Метод `setDaemon(true)` используется для установки потока в режим демона. `setDaemon(true)` нужно вызывать до начала выполнения потока, то есть до вызова метода `start()`. В противном случае, будет выброшено исключение `IllegalThreadStateException`.

Метод `stop()` из класса `Thread` является устаревшим (deprecated) и не рекомендуется к использованию, так как его применение может привести к некорректному завершению потока и потенциальным проблемам с состоянием программы. Этот метод может вызвать непредвиденные ошибки, поскольку не позволяет корректно освободить ресурсы или завершить выполнение блоков `finally`.

Метод `interrupt()` используется для прерывания потока. Когда поток вызывается, метод `interrupt()`, он изменяет внутреннее состояние потока, чтобы показать, что поток был прерван. Если поток находится в состоянии ожидания, сна или других заблокированных операций, таких как `sleep()`, `wait()`, или `join()`, он выбрасывает исключение `InterruptedException`.

В Java класс `Thread` предоставляет метод `getState()`, который возвращает текущее состояние потока. Состояние потока – это один из перечисленных типов в перечислении `Thread.State`, таких как `NEW`, `RUNNABLE`, `BLOCKED`, `WAITING`, `TIMED_WAITING`, `TERMINATED`.

Состояние гонки – это проблема в многопоточных системах, которая возникает, когда два или более потока одновременно обращаются и модифицируют общий ресурс, например, переменную или объект, без должной синхронизации. Это может привести к некорректным результатам и нежелательному поведению программы.

На быстрых компьютерах с современными процессорами и оптимизированной многозадачностью потоки могут выполняться очень быстро. Если код потока простой и выполняется быстро, первый поток может успеть завершиться до того, как начнется выполнение второго потока. В таких случаях вероятность возникновения состояния гонки

уменьшается, так как нет пересечения времени выполнения между потоками.

Синхронизация – это механизм, используемый для управления доступом к общим ресурсам в многопоточных программах. Она гарантирует, что только один поток в определенный момент времени может работать с общим ресурсом, предотвращая состояния гонки и обеспечивая корректность данных.

AtomicInteger – это потокобезопасный класс-оболочка для работы с целыми числами (Integer) в многопоточной среде, который позволяет работать с собой только одному потоку, пока другие потоки ожидают своей очереди.

Потокобезопасные классы используют различные механизмы синхронизации, чтобы обеспечить безопасный доступ к общим ресурсам в многопоточной среде. Эти механизмы могут приводить к некоторым накладным расходам и замедлению выполнения кода, особенно если потоки часто обращаются к общим ресурсам.