* void interrupt()-Посылает потоку запрос на прерывание. Признак состояния прерывания потока устанавливается равным логическому значению true. Если поток в данный момент блокирован вызовом метода sleep(), генерируется исключение типа InterruptedException.
* Static Boolean interrupted() – Проверяет, был ли прерван текущий поток . Следует иметь в виду, что это статический метод. Его вызов имеет побочный эффект: признак состояния прерывания текущего потока устанавливается равным логическому значению false.
* Boolean isInterrupted() – Проверяет, был ли прерван поток. В отличие от предыдущего метода не изменяет состояние прерывания потока.
* Static Thread currentThread() – Возвращает объект типа Thread, представляющий текущий выполняющийся поток.
* Когда ты перехватываешь InterruptedException, то ни в коем случае не оставляй блок catch пустым. Оптимально использовать Thread. currentThread().interrupt().(Или вообще выкинуть исключение выше на метод).
* Потоки могут находиться в одном из шести состояний:
  + Новый
  + Исполняемый
  + Блокированный
  + Ожидающий
  + Временно ожидающий
  + Завершенный
* Чтобы определить текущее состояние потока, достаточно вызвать метод getState()
* Void join() – ожидает завершения указанного потока.
* Void join(long millis) – аналогично, только по истечению указанного времени.
* Void setPriority() – устанавливает приоритет потока( от 1 до 10, 5 – стандарт).
* Потоковый демон – это поток, у которого нет других целей кроме служения другим потокам. Если все потоки завершились, то программа завершит свою работу, не обращая внимания на демонов(не, ну а рил, нахера?). Поэтому не рекомендуется использовать демонов для работы с какими-либо базами данных и прочими подобными вещами, т.к. они могут просто криво завершиться в конце выполнения программы.
* Void setDaemon(Boolean true) – помечает выбранный поток как демон. Метод должен вызываться до начала работы потока.
* yield() в потоке отдает квант времени другому потоку, текущий поток перемещается вниз очереди потоков с равным приоритетом. состояние потока остается RUNNABLE
* вы можете подсказать механизму планирования потоков, что процессором теперь может воспользоваться другой поток. Эта подсказка (не более чем рекомендация; нет никакой гарантии, что планировщик потоков «прислушается» к ней) воплощается в форме вызова метода ***yield()***. Вызывая ***yield()***, вы сообщаете системе, что в ней могут выполняться другие потоки того же приоритета.(по сути, это передача управления).
* [Семафор](https://metanit.com/java/tutorial/8.6.php) — объект, ограничивающий количество потоков, которые могут войти в заданный участок кода.
* **Мью́текс** — аналог одноместного семафора, служащий в программировании для синхронизации одновременно выполняющихся потоков. Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом.
* Атомарная операция — это операция, которую не может прервать планировщик потоков — если она начинается, то продолжается до завершения, без возможности переключения контекста (переключения выполнения на другой поток). Не полагайтесь на атомарность, она ненадежна и опасна — используйте ее вместо синхронизации только в том случае, если вы являетесь экспертом в области синхронизации или, по крайней мере, можете получить помощь от такого эксперта.
* Атомарные операции, упоминаемые в таких дискуссиях, включают в себя «простые операции» с примитивными типами, за исключением ***long*** и ***double***.
* Ключевое слово ***volatile*** обеспечивает видимость в рамках приложения. Если поле объявлено как ***volatile***, это означает, что сразу же после записи в поле изменение будет отражено во всех последующих операциях чтения. Утверждение истинно даже при участии локальных кэшей — поля ***volatile*** немедленно записываются в основную память, и дальнейшее чтение происходит из основной памяти.(но лучше все равно использовать synchronized и его блоки).
* Эти (атомарные) классы предназначены для оптимизации с целью использования атомарности на машинном уровне на некоторых современных процессорах, поэтому в общем случае вам они не понадобятся. Иногда они применяются и в повседневном программировании, но только при оптимизации производительности.
* Стоит еще раз подчеркнуть, что классы ***Atomic*** проектировались для построения классов из ***java.util.concurrent***. Используйте их в своих программах только в особых случаях и только тогда, когда вы твердо уверены, что это не создаст новых проблем. В общем случае безопаснее использовать блокировки (с ключевым словом ***synchronized*** или явным созданием объектов ***Lock***).