**Модель памяти Java – семантика которая описывает поведение разрешенное многопоточным программам.**

**Правила:**

* **Однопоточные программы выполняются псевдопараллельно, т.е процессор может выполнять несколько операций за один такт, меняя их порядок, однако все зависимости по данным остаются такими если бы поведение было последовательным**
* **Любое чтение переменной выдаст либо значение по-умолчанию либо то что туда было записано(кроме не-volatile long, double)**
* **Остальные события выполняются по порядку в соответствии с happens-before.**

**Happens-before – отношение строгого частичного порядка, введенное между атомарными командами и означающее “физически прежде”**

**Синхронизация:**

**Каждый объект в java ассоциирован с монитором который может быть заблокирован и разблокирован.**

**Только один поток может удерживать монитор.**

**Любой другой поток будет заблокирован до того момента пока первый не отпустит захват а второй не захватит монитор. Поток может захватить монитор несколько раз и каждая разблокирован отменят только одну блокировку.**

**Оператор synchronized получает ссылку на объект и пытается заблокировать монитор связанный с этим объектом, пока монитор не будет получен поток не выполняется.**

**Если метод synchronized статический то выполняется блокировка монитора связанного с объектом Class**

**Java не отслеживает и не предотвращает deadlock.**

**Альтернативный метод синхронизации java.util.Concurrent далее.**

**Каждый объект так же имеет wait set потоков.**

**Любая операция добавления или удаления из сета атомарна.**

**Wait set управляетс исключетельно с помощью операция Object.notify, Object.notifyAll, Object.wait()**

**Wait**

**Пусть поток t удерживающий n блокировок на объекте m выполняет wait, тогда:**

* **n == 0, поток не удерживает блокировок – ошибка**
* **иначе, поток добавляется в wait set и происходит n снятия блокировок с монитора объекта m.**
* **Поток не выполняет никаких инструкций пока не будет удален из wait set, он может быть удален, если:**
  + **Другой поток произвел Notify, либо NotifyAll на объекте m.**
  + **Поток t был прерван.**
  + **Закончилось время ожидания, если был вызван wait с параметрами**
  + **Реализация должна разрешать ситуации когда notify и intteraption произошли одновременно**
* **Поток t выполняет n захватов монитора объекта m, если был удален по notify, Если был удален как interrupt то пробрасывает ошибку.**

**Notification**

**Пусть поток t сделал n блокировок на объекте, тогда:**

* **Если N==0 – ошибка, иначе**
* **Если wait set не пустой, то следующий поток удаляется из wait set**
* **Если NotifyAll то все потоки удаляются из сета.**

**Interrupt**

**Если поток вызывает у себя или у другого потока метода interrupt, то статус потока устанавливается в interruption – true, кроме того если поток находится в каком-то wait set, то он удаляется из него.**

**Sleep and yield**

**При переходе в спящий режим поток не освобождает захват мониторов. При использовании Thread.sleep или Thread.yeild отстутсвует какая-либо семантика синхронизации. В частности, кэшированные в регистрах данные не сбрасываются в память.**

**Java.util.concurrent –**

**Элементы синхронизации:**

* **Semaphore: имеет set разрешений (permits), по сути целочисленная переменная увеличивающаяся и уменьшающаяся.**

**Acquire() – получает permit на доступ к данным защищенным семафором. Блокирует поток, пока permit не будет доступен или поток не будет прерван.**

**Release () – освобождает permit, возвращая его в set of permits, потенциально разблокирую один из потоков, ожидающих acquire().**

**Fairness параметр (если стоит true) гарантирует, что любой ожидающий поток приобретет permit раньше чем новый поток пытающийся получить permit.**

**CountDownLatch – средство синхронизации позволяющее потокам ждать завершения набора операций в другом потоке. Инициализируется некоторым числом.**

**Await() – блокирует поток до того момента пока CountDownLatch не достигнет 0. После чего все ожидающие потоки немедленно продолжают свое выполнение.**

**countDown() – уменьшает счетчик на 1.**

**CyclicBarrier -**