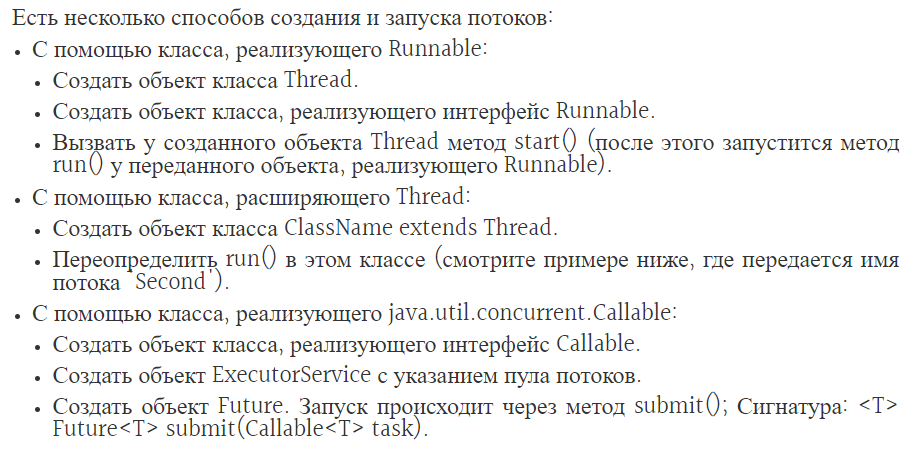
1. **Каким образом можно создать поток?**



**2. В каких состояниях может пребывать поток?**

Каждый поток пребывает в одном из следующих состояний (state):

* Создан (New) – очередь к кадровику готовится, люди организуются.
* Запущен (Runnable) – наша очередь выстроилась к кадровику и обрабатывается.
* Заблокирован (Blocked) – последний в очереди юноша пытается выкрикнуть имя, но услышав, что девушка в соседней группе начала делать это раньше него, замолчал.
* Завершён (Terminated) — вся очередь оформилась у кадровика и в ней нет необходимости.
* Ожидает(Waiting) – одна очередь ждёт сигнала от другой.
  1. **Можно ли поток запустить дважды?**

A java Thread нельзя запустить дважды. После того, как он был запущен и закончил свою работу, он не может быть запущен снова (вызов метода start завершится ошибкой). Поэтому вам придется создать новый экземпляр Thread используя тот же Runnable) и запустить его.

Нет, мы не можем перезапустить поток, так как после запуска и выполнения потока он переходит в состояние Dead.

* 1. **Поясните для чего используются run и start методы.**

Метод run() не создаст новый поток и не выполнит его как отдельный. В этом случае, это простой объект, который вызывает простой метод.

Если мы говорим о методе start(), то там другое дело. Запуская этот метод, runtime запускает новый потом и он уже, в свою очередь, дергает наш метод

* 1. **Чем отличаются методы interrupt, interrupted, isInterrupted?**

Методы interrupt, interrupted, isInterrupted связаны с функцией прерывания потока :

* interrupt() — метод устанавливает флаг прерывания потока;
* bool Thread.interrupted() — статический метод, позволяющий получить состояние флага и сбросить его;
* bool isInterrupted() — метод получения состояния флага прерывания.

В случае, когда поток находится в режиме ожидания, вызванное одним из методов wait, sleep, join, и в это время устанавливается флаг прерывания потока, то будет вызвано исключение InterruptedException.

**6. Что такое монитор объекта? Как работают методы wait и**

**notify/notifyAll?**

Монитор

* контролирует доступ к объекту
* реализует принцип блокировки

У каждого объекта в Java имеется свой монитор (встроен в язык) синхронизировать можно любой объект.

Иногда при взаимодействии потоков встает вопрос о извещении одних потоков о действиях других. Например, действия одного потока зависят от результата действий другого потока, и надо как-то известить один поток, что второй поток произвел некую работу. И для подобных ситуаций у класса Object определено ряд методов:

* wait(): освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify()
* notify(): продолжает работу потока, у которого ранее был вызван метод wait()
* notifyAll(): возобновляет работу всех потоков, у которых ранее был вызван метод wait()

1. **Чем отличается работа метода wait с параметром и без параметра?**

final void wait() - метод используется в многопоточной среде, может вызываться только потоком, владеющим объектом синхронизации. При этом объект синхронизации освобождается, а текущий поток переходит в режим ожидания сигнала освобождения объекта синхронизации другим потоком путем вызова метода notify() либо notifyAll().

final void wait(long time) - аналогично wait() данный метод используется в многопоточной среде, переходит текущий поток в режим ожидания сигнала освобождения объекта синхронизации другим потоком путем вызова метода notify() либо notifyAll(), или ожидание происходит заданное время time, затем выполнение продолжается безусловно.

**8. Как работает метод Thread.yield()? Чем отличаются методы**

**Thread.sleep() и Thread.yield()?**

Метод yield() - пытается сказать планировщику потоков, что нужно выполнить другой поток, что ожидает в очереди на выполнение. Метод не пытается перевести текущий поток в состояние блокировки, сна или ожидания. Он просто пытается его перевести из состояние "работающий" в состояние "работоспособный". Однако выполнение метода может вообще не произвести никакого эффекта. состояние потока остатается RUNNABLE

метод sleep() - приостанавливает поток на указанное. состояние меняется на TIMED\_WAITING, по истечению – RUNNABLE

**9. Как работает метод Thread.join()?**

Java Thread Join — метод, который может быть использован для того, чтобы приостановить выполнение текущего потока до тех пор, пока другой поток не закончит свое выполнение.

**10. Что такое dead lock?**

Взаимная блокировка – это ситуация в которой, два или более процесса занимая некоторые ресурсы, пытаются заполучить некоторые другие ресурсы, занятые другими процессами и ни один из процессов не может занять необходимый им ресурс, и соответственно освободить занимаемый.

Deadlock или дедлок или взаимная блокировка — это ошибка, которая происходит когда нити имеют циклическую зависимость от пары синхронизированных объектов. Представьте, что одна нить входит в монитор объекта x, а другая — объекта y. Если нить в объекте x пытается вызвать любой синхронизированный метод объекта y, а объект y в то же самое время пытается вызвать любой синхронизированный метод объекта x, то нити застрянут в процессе ожидания.

**11. Что значит приоритет потока?**

Для контроля этого процесса был придуман приоритет работы. У каждой нити есть такой приоритет – это число от 1 до 10.

10 – это наивысший приоритет.

1 – самый низкий.

Если приоритет не задан, то нить получает приоритет 5 — средний.

Приоритет нити не сильно влияет на ее работу, а носит скорее рекомендательный характер. Если есть несколько спящих нитей, которые нужно запустить, то Java-машина сначала запустит нить с более высоким приоритетом.

Java-машина управляет нитями так, как посчитает нужным. Нити с низким приоритетом не будут простаивать. Просто они будут получать меньше времени, чем другие, но выполняться все равно будут.

В большинстве случаев нити всегда выполняются с одинаковым приоритетом. Попытка дать одной нити больше времени, чем другим, часто указывает на архитектурную ошибку программы.

**12. Что такое потоки - демоны в Java?**

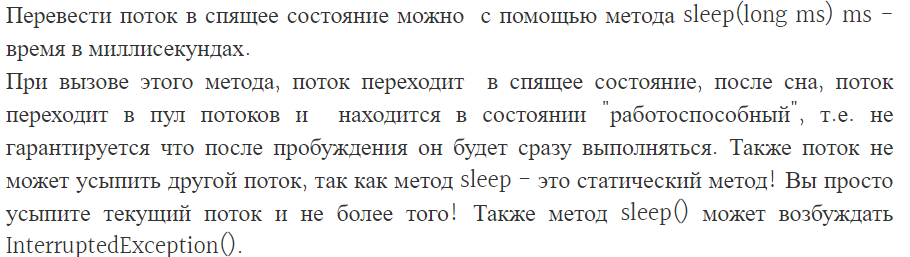
Daemon thread (далее — демон-тред) — это тред, который выполняет задачи в фоне по отношению к другому потоку. То есть, его работа заключается в том, чтоб выполнять задачи вспомогательные, которые нужно делать только в привязке другому (основному) потоку.

Есть много потоков демонов, работающих автоматически, например Garbage Collector, finalizer и т. д.

Почему Java закрывает демон-поток?

Единственная цель потока демона состоит в том, что он предоставляет сервисы потоку пользователя для фоновой задачи поддержки. Поэтому если основной поток завершился, то runtime закрывает автоматически и все его демон-потоки.

**13. Что значит усыпить поток?**

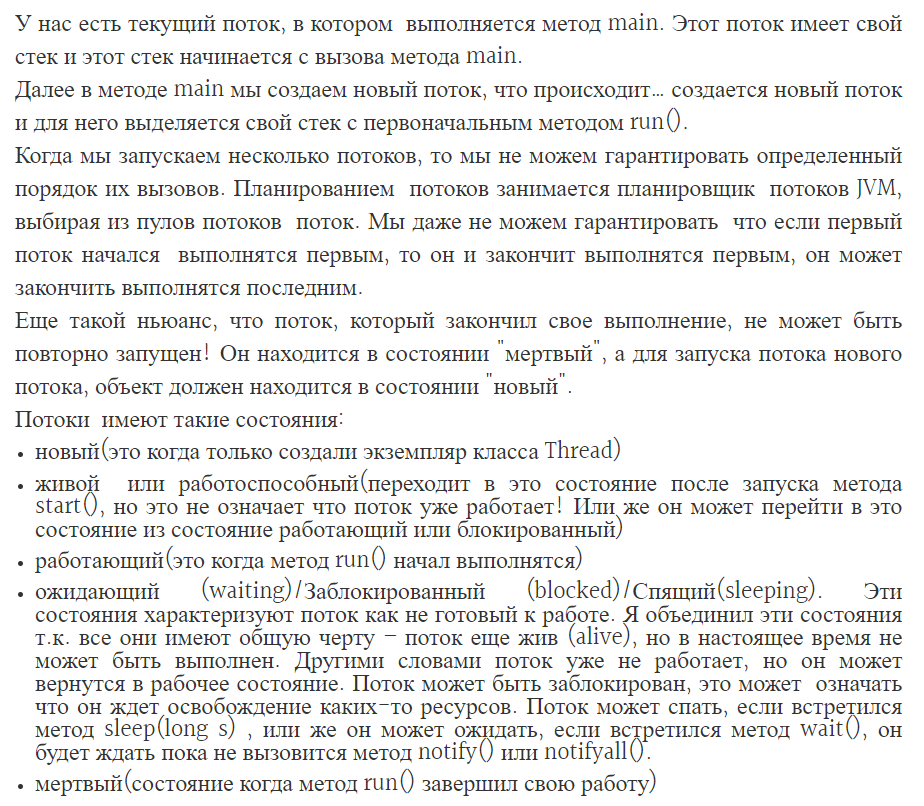


**14. Что такое ThreadGroup и зачем он нужен?**

ThreadGroup представляет собой набор нитей, которые так же могут содержать в себе другие группы потоков. Группа нитей образует дерево, в котором каждая другая группа нитей имеет родителя (кроме исходной). Поток имеет право доступа к данным из своей группы нитей, но не имеет такого доступа к другим группам или к родительской группе потоков.

**15. В каких состояниях может быть поток в Java? Как вообще работает**

**поток?**



**16. Можем ли мы остановить поток? В каких случаях?**

Все потоки нужно прерывать при помощи метода interrupt().

В самом потоке, который возможно будет прерван - нужно устанавливать проверки isInterrupted() во всех ключевых точках (где это необходимо) и обрабатывать соответственно.

**17. Что такое синхронизация?**

Синхронизация (Synchronization) в Java — это возможность контролировать доступ нескольких потоков к любому общему ресурсу. Когда несколько потоков пытаются выполнить одну и ту же задачу, существует вероятность ошибочного результата, поэтому для устранения этой проблемы Java использует синхронизацию, благодаря которой будет только один тред сможет работать в один момент. Синхронизация может быть достигнута тремя способами:

* Синхронизируя метод
* Синхронизируя определенный блок
* Статической синхронизацией

**18. Что такое «атомарные типы» в Java?**

Все атомарные классы переменных имеют базовый элемент Сравнение и назначение (compare-and-set) (аналогичный элементу Сравнение и замена), который реализуется при помощи самого быстрого собственного структурного компонента, который имеется в платформе (Сравнение и замена, Загрузить в связке, Сохранить при условии или в крайнем случае спин-блокировками). В пакет java.util.concurrent.atomic входят 9 видов атомарных переменных (AtomicInteger; AtomicLong; AtomicReference; AtomicBoolean; формы для массивов атомарных целых чисел; длинные (long); ссылки; а также атомарные с пометкой Класс эталона (reference), которые атомарно обновляют две величины).

Классы атомарных переменных можно рассматривать как обобщение volatile переменных, если расширить понятие изменяемых переменных до переменных с поддержкой атомарных обновлений методом Сравнение и назначение. Чтение и запись атомарных переменных имеет такую же семантику памяти как доступ к чтению и записи изменяемых переменных.

**19. В чем разница между блокирующими и неблокирующими очередями?**

Queues — неблокирующие и блокирующие очереди с поддержкой многопоточности. Неблокирующие очереди заточены на скорость и работу без блокирования потоков. Блокирующие очереди используются, когда нужно «притормозить» потоки «Producer» или «Consumer», если не выполнены какие-либо условия, например, очередь пуста или перепонена, или же нет свободного «Consumer»'a.

**20. Различия между CyclicBarrier и CountDownLatch?**

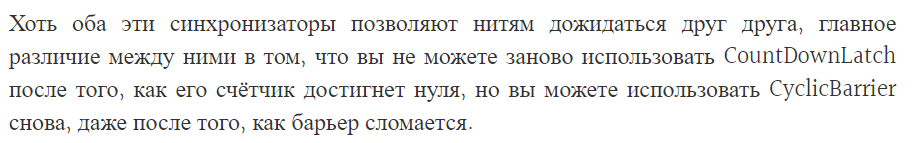
Объект синхронизации потоков CountDownLatch иначе называют «защелкой с обратным отсчетом». Также, как и семафор CountDownLatch работает со счетчиком, обнуление которого снимает самоблокировки выполняемых потоков.

Принцип работы объекта синхронизации CountDownLatch связан с выполнением определенного условия. Так, например, несколько машин подъезжают к паромной переправе. Пока паром не прибыл с другого берега, все находятся в ожидании. После прибытия парома только ограниченное количество машин смогут отправиться на другой берег; остальные останутся ждать возвращения парома. В этом случае паром играет роль объекта синхронизации, а машины выполняют роль потоков. В качестве дополнительного условия может служить время отправки и т.д.

Объект CyclicBarrier представляет собой барьерную синхронизацию, как правило используемую в распределённых вычислениях. Особенно эффективно использование барьеров при циклических расчетах. При барьерной синхронизации алгоритм расчета делят на несколько потоков. С помощью барьера организуют точку сбора частичных результатов вычислений, в которой подводится итог этапа вычислений.

В исходном коде для группы потоков барьер означает, что каждый поток должен остановиться в определенном месте и ожидать прихода остальных потоков группы. Как только все потоки достигнут барьера, их выполнение продолжится.

Класс CyclicBarrier имеет 2 конструктора. В первом конструкторе задается количество потоков, которые должны достигнуть барьера. Во втором конструкторе дополнительно задается реализующий интерфейс Runnable класс, который должен быть запущен после прихода к барьеру всех потоков.



**21. Как работает Semaphor?**

Семафор — это средство для синхронизации доступа к какому-то ресурсу.

Его особенность заключается в том, что при создании механизма синхронизации он использует счетчик.

Счетчик указывает нам, сколько потоков одновременно могут получать доступ к общему ресурсу.

Счетчик указывает нам, сколько потоков одновременно могут получать доступ к общему ресурсу.

*Semaphore(int permits)*

*Semaphore(int permits, boolean fair)*

В конструктор мы передаем:

**int permits** — начальное и максимальное значение счетчика. То есть то, сколько потоков одновременно могут иметь доступ к общему ресурсу;

**boolean fair** — для установления порядка, в котором потоки будут получать доступ. Если fair = true, доступ предоставляется ожидающим потокам в том порядке, в котором они его запрашивали. Если же он равен false, порядок будет определять планировщик потоков.

**22. Что такое Executor?**

Executor — интерфейс, который может выполнять подтвержденные задачи. Интерфейс предоставляет возможность избежать вникания в механику выполнения задачи и деталей использования выполняемого потока. Executor обычно используется для явного создания нитей.

**23. Что такое ExecutorService?**

ExecutorService исполняет асинхронный код в одном или нескольких потоках. Создание инстанса ExecutorService’а делается либо вручную через конкретные имплементации (ScheduledThreadPoolExecutor или ThreadPoolExecutor), но проще будет использовать фабрики класса Executors.