1. Каким образом можно создать поток?

Запустить новый поток можно двумя способами:

**Способ 1**

Создать объект класса Thread, передав ему в конструкторе нечто, реализующее интерфейс Runnable. Этот интерфейс содержит метод run(), который будет выполняться в новом потоке. Поток закончит выполнение, когда завершится его метод run().

**Способ 2**

Создать потомка класса Thread и переопределить его метод run():

1. В каких состояниях может пребывать поток?

Поток может быть в следующем состоянии: созданный, запущенный, блокированный, остановленный, в режиме ожидания, в режиме ожидания по времени (NEW, RUNNABLE, BLOCKED, TERMINATED, WAITING, TIMED\_WAITING).

1. Можно ли поток запустить дважды?

Поток не может быть запущен дважды. Но вы можете использовать один и тот же Runnable и поместить его в качестве цели в два разных потока, например: Создайте свой Runnable. Runnable runnable = () -> System.out.println("Мой Runnable запущен!"); >

1. Поясните для чего используются run и start методы.

**Метод** **start**() **используется** для запуска новой нити. Несмотря на то, что **start**() вызывает **метод** **run**() внутри себя, это не то же самое, что просто вызвать **run**().Если вы вызываете run() как обычный метод, он вызывается в той же нити и никакая новая нить не запуститься, что происходит, когда вы вызываете метод start()

1. Чем отличаются методы interrupt, interrupted, isInterrupted?

* Метод interrupt() — устанавливает флаг прерывания потока.
* Метод bool isInterrupted() объекта потока возвращает состояние флага прерывания и оставляет этот флаг нетронутым.
* Статический метод bool Thread.interrupted() — возвращает состояние флага и сбрасывает его.

1. Что такое монитор объекта? Как работают методы wait и notify/notifyAll?

Каждый объект в Java имеет ассоциированный с ним монитор. Монитор представляет своего рода инструмент для управления доступа к объекту. Когда выполнение кода доходит до оператора synchronized, монитор объекта res блокируется, и на время его блокировки монопольный доступ к блоку кода имеет только один поток, который и произвел блокировку.

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?

без параметров освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify()/notifyAll()

с параметрами заставит поток ожидать заданное количество времени или вызова notify()/notifyAll()

1. Как работает метод Thread.yield()? Чем отличаются методы Thread.sleep() и Thread.yield()?

**Метод** **yield**() служит причиной того, что поток переходит из состояния **работающий** (running) в состояние работоспособный (runnable), давая возможность другим потокам активизироваться. Но следующий выбранный для запуска поток может и не быть другим. **Метод** **sleep**() вызывает засыпание текущего потока на заданное время, состояние изменяется с **работающий** (running) на ожидающий (waiting).

1. Как работает метод Thread.join()?

**Метод** **join** () заставляет вызвавший поток (в данном случае Main **thread**) ожидать завершения вызываемого потока, для которого и применяется **метод** **join** (в данном случае JThread).

Это **метод** синхронизации, блокирующий вызывающий поток (то есть поток, который вызывает **метод**) до тех пор, пока не **Join** завершится поток, **метод** которого был вызван, или не истечет интервал времени ожидания

10.Что такое dead lock?

**Deadlock или дедлок или взаимная блокировка** — это ошибка, которая происходит когда нити имеют циклическую зависимость от пары синхронизированных объектов.

Взаимная блокировка (**deadlock**) — явление, при котором все потоки находятся в режиме ожидания.

Это когда один поток А получил блокировку на объект А1, а поток В получил блокировку на объект В1. В то время как поток А пытается получит блокировку на объект В1, а поток В на А1.

11.Что значит приоритет потока?

Приоритет потока - это величина, складывающаяся из двух составных частей: приоритета породившего поток процесса и собственного приоритета потока.

Когда поток создается, ему назначается приоритет, соответствующий приоритету породившего его процесса. В свою очередь, процессы могут иметь следующие классы приоритетов:

* **Real time;**
* **High;**
* **Above normal;**
* **Normal;**
* **Below normal;**
* **Idle.**

Приоритет потока - это число от 1 до 10, в зависимости от которого, планировщик потоков выбирает какой поток &nbsp;запускать. Однако полагаться на приоритеты для предсказуемого выполнения многопоточной программы нельзя!

12.Что такое потоки - демоны в Java?

**Потоки**-**демоны** — это низкоприоритетные **потоки**, работающие в фоновом режиме для выполнения таких задач, как сбор «мусора»: они освобождают память неиспользованных объектов и очищают кэш.

Это потоки, которое работают в фоновом &nbsp;режиме и не гарантируют что они завершатся. Тоеть если все потоки завершились, то поток демон просто обрывается вместе с закрытием приложения.

13.Что значит усыпить поток?

При вызове этого метода, поток переходит &nbsp;в спящее состояние, после сна, поток переходит в пул потоков и &nbsp;находится в состоянии "работоспособный", т.е. не гарантируется что после пробуждения он будет сразу выполняться. Также поток не может усыпить другой поток, так как метод sleep - это статический метод! Вы просто усыпите текущий поток и не более того! Также метод sleep() может возбуждать InterruptedException().

14.Что такое ThreadGroup и зачем он нужен? ‘

**ThreadGroup** **–** **это** **класс, который** **управляет** **группами** **нитей.** **Такой** **подход** **позволяет** **защитить** **нити** **от** **нежелательного** **изменения.** Иногда приходится выполнять код, которому нельзя 100% доверять. Поэтому удобно поместить все его нити в отдельную группу и запретить им вмешиваться в работу основной группы нитей. Группа нитей может содержать другие группы. Это позволяет организовать все нити и группы в иерархическое дерево.

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

У нас есть текущий поток, в котором выполняется метод main. Этот поток имеет свой стек и этот стек начинается с вызова метода main.

Далее в методе main мы создаем новый поток, что происходит… создается новый поток и для него выделяется свой стек с первоначальным методом run().

Когда мы запускаем несколько потоков, то мы не можем гарантировать определенный порядок их вызовов. Планированием потоков занимается планировщик потоков JVM, выбирая из пулов потоков поток. Мы даже не можем гарантировать что если первый поток начался выполнятся первым, то он и закончит выполнятся первым, он может закончить выполнятся последним.

Еще такой ньюанс, что поток, который закончил свое выполнение, не может быть повторно запущен! Он находится в состоянии "мертвый", а для запуска потока нового потока, объект должен находится в состоянии "новый".

**Потоки имеют такие состояния:**

1.**новый**(это когда только создали экземпляр класса Thread)

2.**живой** или работоспособный(переходит в это состояние после запуска метода start(), но это не означает что поток уже работает! Или же он может перейти в это состояние из состояние работающий или блокированный)

3**.работающий**(это когда метод run() начал выполнятся)

4.**ожидающий (waiting)/Заблокированный (blocked)/Спящий(sleeping).** Эти состояния характеризуют поток как не готовый к работе. Я объединил эти состояния т.к. все они имеют общую черту – поток еще жив (alive), но в настоящее время не может быть выполнен.

Другими словами поток уже не работает, но он может вернутся в рабочее состояние

Поток может быть заблокирован, это может означать что он ждет освобождение каких-то ресурсов. Поток может спать, если встретился метод sleep(long s) , или же он может ожидать, если встретился метод wait(), он будет ждать пока не вызовится метод notify() или notifyall().

**мертвый(**состояние когда метод run() завершил свою работу)

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

**Остановить** **поток** можно с помощью метода interrupt(). В **случае**, если **поток** не является демоном, то возникает исключение InterruptedException.

В Java поток представлен классом Thread. В нём есть метод stop(), но пользоваться им нельзя, метод помечен как deprecated. Такая жесткая остановка моментально возвращает все захваченные потоком [мониторы](https://itsobes.ru/JavaSobes/chto-takoe-synchronized), и защищенные ими данные могут оказаться в неконсистентном состоянии.  
  
Разработчики [рекомендуют](https://docs.oracle.com/javase/1.5.0/docs/guide/misc/threadPrimitiveDeprecation.html) вместо этого использовать флаг, который будет показывать о намерении остановить поток. Флаг выставляется извне потока, а внутри проверяется в подходящий момент. Если нужно остановиться, поток просто выходит из метода run(). В качестве такого флага подойдет переменная типа AtomicBoolean.  
Когда в потоке используются блокирующие операции, обычно для определенного типа операции существует свой способ её прервать. Например, можно закрыть сокет, на котором поток ожидает. Для большинства блокирующих операций сработает метод Thread.interrupt(). С его помощью можно прервать [Object.wait()](https://itsobes.ru/JavaSobes/wait-notify-notifyall) и операции из [NIO](https://itsobes.ru/JavaSobes/iz-chego-sostoit-paket-java-nio).  
  
Останется только правильно обработать такое прерывание. Прерванный wait() выбросит InterruptedException, Selector.select() вернет результат. Чтобы отличить осознанное прерывание с целью завершить тред от какого-либо другого, его обработку всё ещё необходимо снабдить проверкой флага.

17.Что такое синхронизация?  **Java** поддерживает несколько потоков для выполнения. Это может привести к тому, что два или более потока получат доступ к одному и тому же полю или объекту. **Синхронизация** — это процесс, который позволяет выполнять все параллельные потоки в программе синхронно.

18.Что такое «атомарные типы» в Java?

Атомарный объект – это такой объект операции над которым можно считать неделимыми, т.е. такими, которые не могут быть прерваны или результат которых не может быть получен, до окончания операции. Таким образом исключается сама возможность получения некорректных данных в результате наблюдения половины записанных данных, к примеру.

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. Synchronizers — вспомогательные утилиты для синхронизации потоков.

20.Различия между CyclicBarrier и CountDownLatch?

Буквально**CountDown означает вычитание количества, Latch означает дверную защелку. Когда счетчик равен 0, дверную защелку можно открыть.**。**Циклический барьер представляет собой препятствие циркуляции**. Оба класса имеют это значение: следующий шаг выполняется после того, как соответствующие потоки завершили работу, то есть следующий шаг выполняется после того, как все будут готовы.

Хоть оба эти синхронизаторы позволяют нитям дожидаться друг друга, главное различие между ними в том, что вы не можете заново использовать CountDownLatch после того, как его счётчик достигнет нуля, но вы можете использовать CyclicBarrier снова, даже после того, как барьер сломается.

21.Как работает Semaphor?

Семафоры представляют еще одно средство синхронизации для доступа к ресурсу. В Java семафоры представлены классом **Semaphore**, который располагается в пакете **java.util.concurrent**.

В основном **семафор** используется для управления доступом к общему ресурсу с помощью переменной счетчика. Используя **семафор** в **Java**, мы можем ограничить количество потоков, которые могут получить доступ к общему ресурсу.

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.

23.Что такое ExecutorService

**ExecutorService** в **Java** - это интерфейс, который является частью пакета **java**.util.concurrent. Эта утилита параллелизма **Java** помогает выполнять асинхронные задачи одновремеsнно.