1. Каким образом можно создать поток?

**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.

1. Поясните для чего используются run и start методы.

Методы start() и run() используются при запуске и выполнении потока. Метод start() используется для указания того, что поток выполнения нужно запустить. В своем коде метод start() обращается к методу run(). В методе run() указывается код, который должен выполняться в потоке. Метод run() является точкой входа в поток. Метод run() объявлен в интерфейсе Runnable.

t.run(); //код r выполняется в текущем потоке

t.start(); //код r выполняется в новом потоке

1. Чем отличаются методы interrupt, interrupted, isInterrupted?

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

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.

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. **Метод** **sleep**() вызывает засыпание текущего потока на заданное время, состояние изменяется с **работающий** (running) на ожидающий (waiting).

1. Как работает метод Thread.join()?

В Java предусмотрен механизм, позволяющий одному потоку ждать завершения выполнения другого. Для этого используется метод join(). Например, чтобы главный поток подождал завершения побочного потока myThready, необходимо выполнить инструкцию myThready.join() в главном потоке.

10. Что такое deadlock?

Особый тип ошибок, которого следует избегать, имеющий отношение к многозадачности — это взаимная блокировка (deadlock). Она происходит, когда потоки имеют циклическую зависимость от пары синхронизированных объектов.

11. Что значит приоритет потока?

Приоритет потока — это число от 1 до 10, в зависимости от которого, планировщик потоков выбирает какой поток запускать. По умолчанию приоритет потока 5.

Существуют следующие константы для определения приоритета потока:

* *Thread.MIN\_PRIORITY* (1)
* *Thread.NORM\_PRIORITY* (5)
* *Thread.MAX\_PRIORITY* (10)

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 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 в спящее состояние, после сна, поток переходит в пул потоков и 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!

14. Что такое ThreadGroup и зачем он нужен?

ThreadGroup представляет собой группу потоков. Преимущество группы потоков заключается в том, что мы можем выполнять такие операции, как приостановка, возобновление или прерывание для всех потоков, используя один вызов функции.

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

**Процесс**

Процесс (process) - это объект, который создается операционной системой, когда пользователь запускает приложение. Процессу выделяется отдельное адресное пространство, причем это пространство физически недоступно для других процессов. Когда вы запускаете текстовый процессор или программу калькулятора, вы создаете новый процесс.

**Поток**

Для каждого процесса операционная система создает один главный поток (thread), который является потоком выполняющихся по очереди команд центрального процессора. При необходимости главный поток может создавать другие потоки, пользуясь для этого программным интерфейсом операционной системы.

Все потоки, созданные процессом, выполняются в адресном пространстве этого процесса и имеют доступ к ресурсам процесса. Однако поток одного процесса не имеет никакого доступа к ресурсам потока другого процесса, так как они работают в разных адресных пространствах. При необходимости организации взаимодействия между процессами или потоками, принадлежащими разным процессам, следует пользоваться системными средствами, специально предназначенными для этого.

Поток может находиться в одном из следующих состояний:

* ***New***- объект класса *Thread* создан, но еще не запущен. Он еще не является потоком выполнения и естественно не выполняется.
* ***Runnable***- поток готов к выполнению, но планировщик еще не выбрал его.
* ***Running***– поток выполняется.
* ***Waiting/blocked/sleeping*** - поток блокирован или поток ждет окончания работы другого потока.
* ***Dead***- поток завершен. Будет выброшено исключение при попытке вызвать метод *start()* для dead потока.

Существуют перечисление ***Thread.State***, содержащее значения возможных состояний потока: ***NEW, RUNNABLE, BLOCKED, WAITING, TIMED\_WAITING, TERMINATED.***

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

В Java существуют (существовали) средства для принудительного завершения потока. В частности, метод Thread.stop() завершает поток незамедлительно после своего выполнения. Однако этот метод, а также Thread.suspend(), приостанавливающий поток, и Thread.resume(), продолжающий выполнение потока, были объявлены устаревшими и их использование отныне крайне нежелательно. Дело в том, что поток может быть «убит» во время выполнения операции, обрыв которой на полуслове оставит некоторый объект в неправильном состоянии, что приведет к появлению трудноотлавливаемой и случайным образом возникающей ошибке. Изображение выглядит как текст

Автоматически созданное описание

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?

Пакет java.util.concurrent.atomic содержит 9 классов для выполнения атомарных операций. Операция называется атомарной, если её можно безопасно выполнять при параллельных вычислениях в нескольких потоках, не используя при этом ни блокировок, ни синхронизацию [synchronized](https://java-online.ru/java-thread.xhtml#synchronized).

19. В чем разница между блокирующими и неблокирующими очередями?

| **Неблокирующая очередь** |
| --- |

ConcurrentLinkedQueue: это очередь, подходящая для сценариев с высоким уровнем параллелизма. Благодаря методу без блокировки достигается высокая производительность при высоком параллелизме. Как правило, ConcurrentLinkedQueue работает лучше, чем BlockingQueue. Это неограниченная потокобезопасная очередь на основе связанных узлов. Элементы очереди следуют принципу «первым пришел - первым ушел». Заголовок добавляется первым, а хвост добавляется последним. Очередь не допускает пустых элементов.

| **Очередь блокировки** |
| --- |

Блокирующие очереди обычно имеют две дополнительные операции:

1. Когда очередь пуста, поток, который получает элемент, будет ждать, пока очередь не станет непустой.
2. Когда очередь заполнена, поток, хранящий элемент, будет ждать, пока очередь не станет доступной.

20. Различия между CyclicBarrier и CountDownLatch?

В буквальном смысле это закольцованный забор, с помощью которого можно заставить группу потоков ожидать определенного состояния, а затем все выполнять одновременно. Барьер можно использовать повторно.

Один поток (или несколько), ожидая, пока другие N потоков завершат что-либо перед выполнением

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 — простой интерфейс, содержащий метод execute() для запуска задачи, заданной запускаемым объектом Runnable . ExecutorService представляет собой субъинтерфейс Executor , который добавляет функциональность для управления жизненным циклом потоков.

23. Что такое ExecutorService

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