**1. Каким образом можно создать поток?**Для создания нового потока мы можем создать новый класс, либо наследуя его от класса Thread, либо реализуя в классе интерфейс Runnable.

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

**1. New -** Когда создается новый поток, он находится в состоянии **New**, причем запускаться поток еще не начал.

**2. Runnable** - В этом состоянии поток либо запускается, либо готов в любой момент запуститься.

## 3. Blocked - При попытке выполнить задачу, которая не может быть завершена в данный момент времени, поток из состояния runnable переходит в состояние blocked.

**4. Waiting -** Когда поток находится в состоянии waiting, он ждет другой поток, связанный условием.

## 5. Time waiting - Поток находится в состоянии runnable. Теперь он вызывает метод sleep(t), wait(t) или join(t) с неким промежутком времени в качестве параметра и переходит в состояние time waiting.

## 6. Terminate – поток завершается

**3. Можно ли поток запустить дважды?**

Нет

Переиспользование потоков не принесло бы ощутимых преимуществ. Наоборот, повторный запуск потоков создал бы неудобства для программистов:

1. Если поток можно запустить дважды, то его можно запустить дважды из нескольких потоков. При этом неясно нужно ли уведомлять потоки-клиенты о «двойственном» состоянии потока-субъекта.
2. Отслеживание состояния потока теряет смысл. Поток никогда не будет по-настоящему завершен, т.к. может перезапустить сам себя либо быть перезапущен другим потоком.

**4. Поясните для чего используются run и start методы.   
 run –** определяет точку входа в поток  
 **start –** запускает поток, вызывая метод **run**.

**5. 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.

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

Монитор(мьютекс) — высокоуровневый механизм взаимодействия и синхронизации потоков, обеспечивающий доступ к неразделяемым ресурсам.

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()

**7. 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().

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

Вызов метода Thread. yield() позволяет досрочно завершить квант времени текущей нити или, другими словами, переключает процессор на следующую нить.

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

Метод join() не завершает главный поток, пока дочерний не будет выполнен.

**10.Что такое dead lock?**

Deadlock, он же взаимная блокировка, явление при котором все потоки находятся в режиме ожидания. Чтобы уменьшить шанс появления deadlock’a не рекомендуется использовать методы wait() и notify().

**11.Что значит приоритет потока?**

Приоритет потока в Java определяет, насколько часто поток получает доступ к процессору по сравнению с другими потоками. П

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

Потоками-демонами называются потоки, работающие в фоновом режиме для нашей программы.

**13.Что значит усыпить поток?**

Это значит приостановить его на определенный промежуток времени, вызвав в ходе его выполнения статический метод Thread

**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?**

В Java потоки представляют собой независимые, параллельно выполняющиеся последовательности инструкций. Каждый поток имеет свой собственный стек вызовов, в котором хранятся все локальные переменные и временные значения, связанные с выполнением кода внутри этого потока.

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

Да, поток можно остановить в Java. Остановка потока происходит с помощью метода interrupt(), который посылает потоку сигнал о том, что ему необходимо завершить свое выполнение.

**17.Что такое синхронизация?**

Синхронизация в программировании - это механизм, который позволяет управлять доступом к общим ресурсам из нескольких потоков выполнения программы.

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

Атомарные типы в Java - это специальные типы данных, которые гарантируют выполнение операций чтения и записи атомарно, т.е. операции над ними выполняются как единое целое и не могут быть прерваны другими потоками в процессе выполнения.

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

Блокирующие очереди в Java блокируют поток исполнения, если очередь пуста (при попытке извлечения элемента) или заполнена (при попытке добавления элемента). Это означает, что поток будет ждать, пока не будет возможности добавить или извлечь элемент из очереди. Блокирование может вызвать простой и замедлить работу программы, но обеспечивает корректное и безопасное взаимодействие между потоками.

Неблокирующие очереди, напротив, не блокируют поток исполнения, а возвращают специальные значение, указывающие на результат операции: успешность добавления или извлечения элемента. Это означает, что поток может продолжать работу независимо от результата операции с очередью.

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

(CyclicBarrier) позволяет нескольким потокам синхронизироваться в одной точке, чтобы продолжить выполнение задачи. Он ждет, пока все потоки не достигнут барьера, затем выполняет заданное действие и перезапускает барьер.

CyclicBarrier может быть переиспользован после выполнения всех задач. CountDownLatch не может быть переиспользован, после того как он достигнет нулевого значения

1. В CyclicBarrier можно добавить дополнительное действие, которое будет выполняться после достижения барьера. В CountDownLatch такое дополнительное действие не поддерживается.
2. В CyclicBarrier можно указать действие, которое должно выполняться при возникновении ошибки в одном из потоков. В CountDownLatch такой функциональности нет.

**21.Как работает Semaphor?**

Semaphore содержит счетчик, который указывает количество доступных разрешений. Когда поток хочет получить доступ к ресурсу, он вызывает метод acquire() у Semaphore. Если счетчик больше нуля, Semaphore уменьшает счетчик на единицу и разрешает доступ к ресурсу. Если счетчик равен нулю, Semaphore блокирует поток до тех пор, пока какой-нибудь другой поток не освободит ресурс и увеличит счетчик на единицу.

**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