1. Каким образом можно создать поток?

Создание класса, который наследуется от класса Thread и переопределением метода run().

Создание класса, который реализует интерфейс Runnable и передачей этого объекта в конструктор Thread.

2. В каких состояниях может пребывать поток?

NEW - когда поток создан, но еще не запущен.

RUNNABLE - когда поток запущен и может выполняться.

BLOCKED - когда поток заблокирован и ожидает освобождения блокировки.

WAITING - когда поток находится в режиме ожидания до получения сигнала от другого потока.

TIMED\_WAITING - когда поток находится в режиме ожидания в течение определенного времени.

TERMINATED - когда поток завершил свое выполнение.

3. Можно ли поток запустить дважды?

Нет, поток нельзя запустить дважды. Если попытаться запустить поток, который уже запущен, будет выброшено исключение IllegalThreadStateException.

4. Поясните для чего используются run и start методы.

Метод run() содержит код, который будет выполняться в потоке при его запуске. Метод start() запускает новый поток и вызывает метод run() в этом потоке.

5. Чем отличаются методы interrupt, interrupted, isInterrupted?

Метод interrupt() посылает сигнал потоку о том, что он должен прервать свое выполнение. Метод interrupted() возвращает true, если поток был прерван, и сбрасывает флаг прерывания. Метод isInterrupted() возвращает true, если флаг прерывания потока установлен.

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

Монитор объекта - это механизм синхронизации, который позволяет потокам синхронизироваться при работе с общим ресурсом. Метод wait() переводит поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify() или notifyAll(). Метод notify() выбирает один поток из очереди ожидания и продолжает его выполнение, а метод notifyAll() выбирает все потоки из очереди ожидания и продолжает их выполнение.

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

Метод wait() с параметром задает время ожидания, в течение которого поток будет находиться в состоянии ожидания. Метод wait() без параметра ожидает до тех пор, пока другой поток не вызовет метод notify() или notifyAll().

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

Метод Thread.yield() приостанавливает выполнение текущего потока и позволяет другим потокам получить доступ к процессору. Метод Thread.sleep() приостанавливает выполнение текущего потока на указанное количество времени, а метод Thread.yield() позволяет другим потокам выполниться на короткое время.

9. Как работает метод Thread.join()?

Метод Thread.join() позволяет потоку, который вызывает метод, ожидать завершения другого потока. Если вызвать метод join() у потока, который еще не завершил свое выполнение, то текущий поток будет находиться в состоянии ожидания до завершения этого потока.

10.Что такое dead lock?

Dead lock - это ситуация, когда два или более потоков блокируют друг друга, ожидая ресурсы, которые уже заблокированы другим

11.Что значит приоритет потока?

Потоки с более высоким приоритетом более важны для программы, и в первую очередь им должно выделяться процессорное время.

Потоки с более высоким приоритетом более важны для программы, и в первую очередь им должно выделяться процессорное время.

12.Что такое потоки - демоны в Java?

В Java существует два типа потоков — пользовательские (те, которые мы создаем) потоки и потоки-демоны. Когда запускается Java-программа, сразу же начинается выполняться один поток — основной поток. Основной поток запускает метод main(). Мы можем создавать новые потоки из основного потока. Основной поток завершает выполнение последним, поскольку он выполняет различные операции завершения работы c потоками ввода и вывода, отключает соединения с базами данных и т. д.

Потоки-демоны в основном функционируют как вспомогательные потоки, они выполняют разные операции в фоновом режиме. Например,Garbage Collection в Java выполняется в фоновом режиме как поток-демон.

В основном потоке мы можем создавать столько потоков, сколько необходимо. Более того, класс Thread предоставляет метод setDaemon(boolean), который позволяет рабочему (=пользовательскому) потоку превратиться в поток-демон.

**Потоки-демоны:**

* поток с низким приоритетом, работающий в фоновом режиме;
* поток-демон автоматически завершается виртуальной машиной Java, когда все остальные рабочие (worker threads) потоки завершают выполнение;
* обычно потоки-демоны используются для операций ввода-вывода и сервисов (в смартфонах для связи Bluetooth или NFC).

Основное различие между пользовательским потоком и потоком-демон заключается в том, что когда все рабочие (=основные или пользовательские) потоки завершают выполнение или умирают, потоки-демоны автоматически завершаются JVM, даже если они все еще выполняются.

13.Что значит усыпить поток?

Это значит приостановить его на определенный промежуток времени, вызвав в ходе его выполнения статический метод Thread.sleep() передав в качестве параметра необходимое количество времени в миллисекундах.

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?

**Состояния потоков**

Потоки могут пребывать в нескольких состояниях:

* **New***–* когда создается экземпляр класса Thread, поток находится в состоянии new. Он пока еще не работает.
* **Running***—* поток запущен и процессор начинает его выполнение. Во время выполнения состояние потока также может измениться на Runnable, Dead или Blocked.
* **Suspended** — запущенный поток приостанавливает свою работу, затем можно возобновить его выполнение. Поток начнет работать с того места, где его остановили.
* **Blocked** — поток ожидает высвобождения ресурсов или завершение операции ввода-вывода. Находясь в этом состоянии поток не потребляет процессорное время.
* **Terminated** — поток немедленно завершает свое выполнение. Его работу нельзя возобновить. Причинами завершения потока могут быть ситуации, когда код потока полностью выполнен или во время выполнения потока произошла ошибка (например, ошибка сегментации или необработанного исключения).
* **Dead** — после того, как поток завершил свое выполнение, его состояние меняется на dead, то есть он завершает свой жизненный цикл.

Приложение, создающее экземпляр класса Thread, должно предоставить код, который будет работать в этом потоке. Существует два способа, чтобы добиться этого:

* **Предоставить реализацию объекта Runnable.** Интерфейс Runnable определяет единственный метод — run, который должен содержать код, выполняющийся в потоке. Объект Runnable передается конструктору Thread. Например:

public class HelloRunnable implements Runnable {

public void run() {

System.out.println("Hello from a thread!");

}

public static void main(String args[]) {

(new Thread(new HelloRunnable())).start();

}

}

* **Использовать подкласс Thread.** Класс Thread сам реализует Runnable, хотя его метод run не делает ничего. Можно объявить класс Thread подклассом, предоставляя собственную реализацию метода run, как в примере:

public class HelloThread extends Thread {

public void run() {

System.out.println("Hello from a thread!");

}

public static void main(String args[]) {

(new HelloThread()).start();

}

}

Обратите внимание, что оба примера вызывают **Thread.start**, чтобы запустить новый поток.

**Какой из способов выбрать?** Первый — с использованием объекта Runnable — более общий, потому что этот объект может превратить отличный от Thread класс в подкласс. Этот способ более гибкий и может использоваться для высокоуровневых API управления потоками.

Второй способ больше подходит для простых приложений, но есть условие: класс задачи должен быть потомком Thread.

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

**Завершение потока**

Распространенный способ завершения потока представляет опрос логической переменной. И если она равна, например, false, то поток завершает бесконечный цикл и заканчивает свое выполнение.

Еще один способ вызова завершения или прерывания потока представляет метод interrupt(). Вызов этого метода устанавливает у потока статус, что он прерван. Сам метод возвращает true, если поток может быть прерван, в ином случае возвращается false.

При этом сам вызов этого метода НЕ завершает поток, он только устанавливает статус: в частности, метод isInterrupted() класса Thread будет возвращать значение true. Мы можем проверить значение возвращаемое данным методом и прозвести некоторые действия.

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?

Синхронизация относится к многопоточности. Синхронизированый блок кода может быть выполнен только одним потоком одновременно. 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. Синхронизация позволяет избежать ошибок согласованности памяти, вызванные из-за непоследовательного доступа к общей памяти. Когда метод объявлен как синхронизированный — нить держит монитор для объекта, метод которого исполняется. Если другой поток выполняет синхронизированный метод, ваш поток заблокируется до тех пор, пока другой поток не отпустит монитор. Синхронизация достигается в Java использованием зарезервированного слова synchronized. **Вы можете использовать его в своих классах определяя синхронизированные методы или блоки. Вы не сможете использовать synchronized в переменных или атрибутах в определении класса.**

Все потоки, принадлежащие одному процессу, разделяют некоторые общие ресурсы (адресное пространство, открытые файлы). Что произойдет, если один поток еще не закончил работать с каким-либо общим ресурсом, а система переключилась на другой поток, использующий тот же ресурс?

Когда два или более потоков имеют доступ к одному разделенному ресурсу, они нуждаются в обеспечении того, что ресурс будет использован только одним потоком одновременно. Процесс, с помощью которого это достигается, называется синхронизацией.

18.Что такое «атомарные типы» в Java?

Пакет **java.util.concurrent.atomic** содержит девять 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" \l "synchronized).

19.В чем разница между блокирующими и неблокирующими очередями?

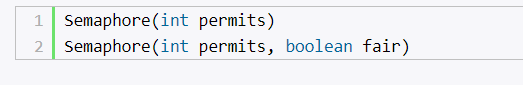
Разница между блокирующей очередью и обычной очередью заключается в том, что, когда очередь пуста, операция получения элементов из очереди будет заблокирована, или когда очередь будет заполнена, операция добавления элементов в очередь будет заблокирована. Потоки, пытающиеся получить элементы из пустой очереди блокировки, будут заблокированы до тех пор, пока другие потоки не вставят новые элементы в пустую очередь. Точно так же потоки, пытающиеся добавить новые элементы в полную очередь блокировки, также будут заблокированы до тех пор, пока другие потоки не освободят очередь снова, например, удалив один или несколько элементов из очереди или полностью опустошив очередь.

20.Различия между CyclicBarrier и CountDownLatch?

CyclicBarrier похож на CountDownLatch , но главное различие между ними в том, что использовать «замок» можно лишь единожды — после того, как его счётчик достигнет нуля, а «барьер» можно использовать неоднократно, даже после того, как он «сломается».

21.Как работает Semaphor?

Для управления доступом к ресурсу семафор использует счетчик, представляющий количество разрешений. Если значение счетчика больше нуля, то поток получает доступ к ресурсу, при этом счетчик уменьшается на единицу. После окончания работы с ресурсом поток освобождает семафор, и счетчик увеличивается на единицу. Если же счетчик равен нулю, то поток блокируется и ждет, пока не получит разрешение от семафора.



3 22.Что такое Executor?

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

23.Что такое ExecutorService

ExecutorService представляет собой суб-интерфейс Executor , который добавляет функциональность для управления жизненным циклом потоков. Он также включает в себя метод submit() , который аналогичен методу execute() , но более универсален