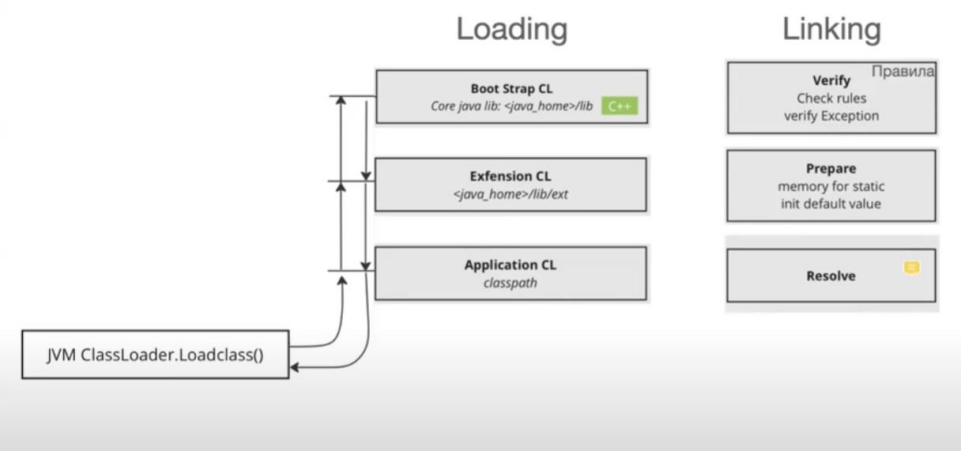
Chapter 1 – Basic Java Main Topc



При запуске можно запустить профайлер? чтобы понять где и сколько оно жрет.

Первое что происходит загрузка класса - есть штука типа класс лоадер - JVM ClassLoader.LoadClass()

Лоадер спрашивает класс у апплекейшн класс лоадера он в экстеншен а тот в бутстрапе - самые базовые классы. если ищут сами.
Самые системные классы написаны на C++ чтобы были максимально быстрые.

После всего идет этап инициализации где всем переменным присваиваются значения.

Операционка оперирует процессами а в процессе потоками. Java не может напрямую влиять какие потоки работают. Она может только сказать - возьми этот поток сейчас пожалуйста. Только рекомендовать может операционка сама решает какой взять.

Итак давайте вернемся к модели памяти Java в Джаве тоже можно создать программу с несколькими процессами.

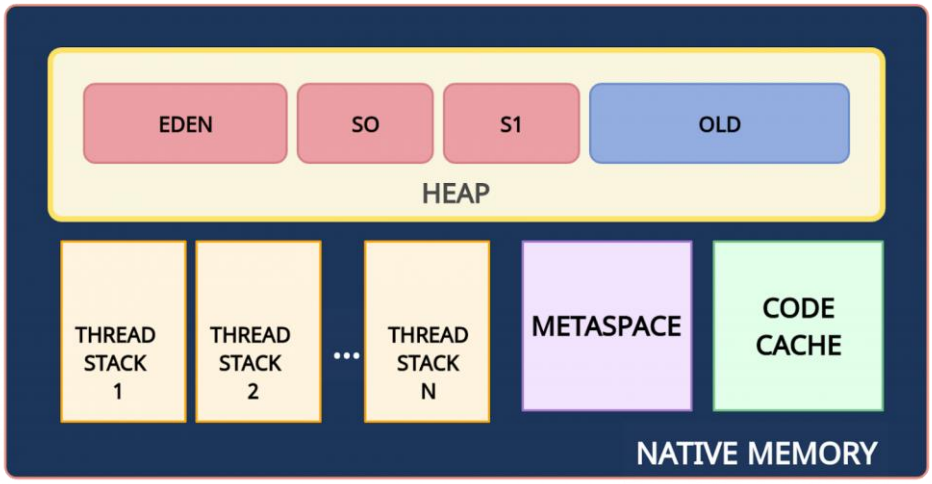
```java
ProcessBuilder processBuilder = new ProcessBuilder();

// -- Linux --

// Run a shell command
processBuilder.command("bash", "-c", "ls /home/mkyong/");

// Run a shell script
processBuilder.command("path/to/hello.sh");

// -- Windows --

// Run a command
processBuilder.command("cmd.exe", "/c", "dir C:\\Users\\mkyong");

// Run a bat file
processBuilder.command("C:\\Users\\mkyong\\hello.bat");

Process process = processBuilder.start();
```

Но каждый новый процесс в джаве это новый(отдельный) инстанс JVM!
Каждый запущенный процесс это отдельная модель памяти и стек вызовов метода.
Один процесс не может просто так получить доступ к памяти другого.

Модель памяти Java

Тут круто описано:

ПРОЧТИ!!!!
https://habr.com/ru/companies/otus/articles/553996/



Native Memory — вся доступная системная память.

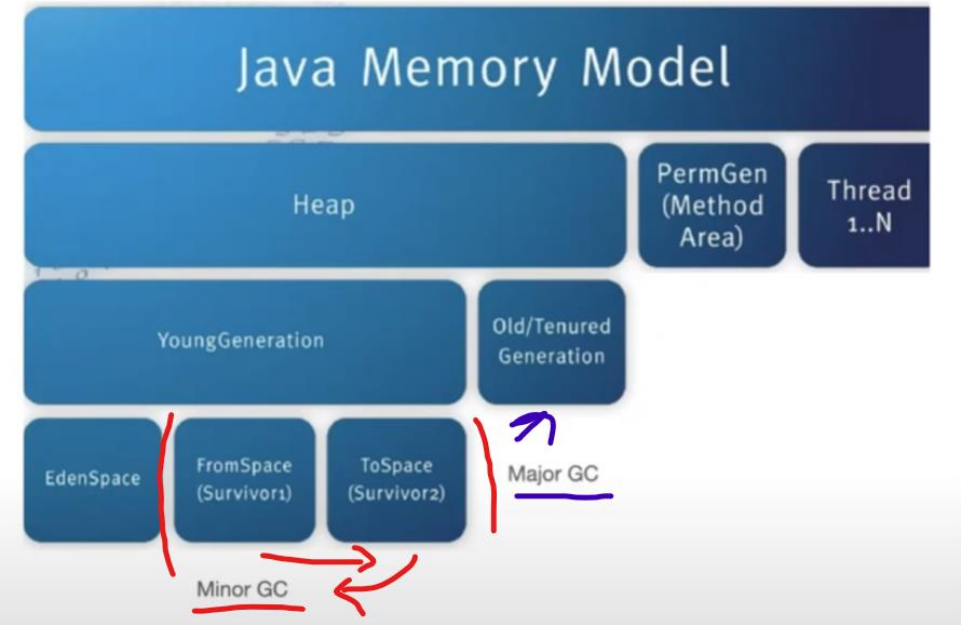
Heap (куча) — часть native memory, выделенная для кучи. Здесь JVM хранит объекты. Это общее пространство для всех потоков приложения. Размер этой области памяти настраивается с помощью параметра -Xms (минимальный размер) и -Xmx (максимальный размер).

Stack (стек) — используется для хранения локальных переменных и стека вызовов метода. Для каждого потока выделяется свой стек.

Metaspace (метаданные) — в этой памяти хранятся метаданные классов и статические переменные. Это пространство также является общими для всех. Так как metaspace является частью native memory, то его размер зависит от платформы. Верхний предел объема памяти, используемой для metaspace, можно настроить с помощью флага MaxMetaspaceSize.

PermGen (Permanent Generation, постоянное поколение) присутствовало до Java 7. Начиная с Java 8 ему на смену пришла область Metaspace.

CodeCache (кэш кода) — JIT-компилятор компилирует часто исполняемый код, преобразует его в нативный машинный код и кеширует для более быстрого выполнения. Это тоже часть native memory.



Сборка мусора

Если на объект не ссылается ни одна из ссылок это признак чтобы его уничтожить.

Когда мы создаем наш первый объект в Java например Student student = new Student()

Что происходит?

Он помещается в самую первую корзину - в Eden Space там мы храним наши на воссозданные объекты в процессе работы программы.

Иногда запускается сборщик мусора что периодически проходит и собирает объекты что ен нужны. Он берет эти объекты и перекладывает их в ячейуц FromSpace потом во второй итерации если тоже самое то в ToSpace и перкладывает опять в FromSpace - когда он видит что этот цикл был пройден множество раз понятно что он не нужен и его кладут в Old Ganeration.

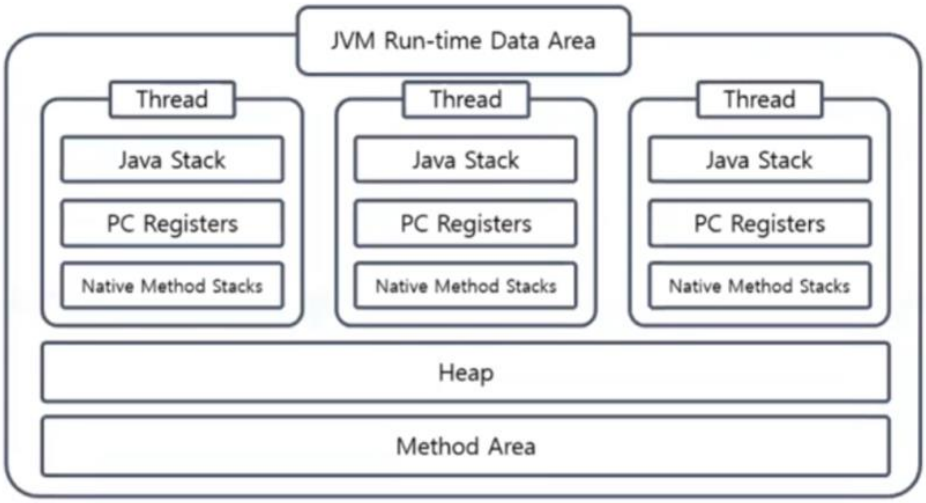
Процесс перекладывания из from в to наоборот это Minor когда уже в Old – Major.

Garbage Collector - очень тяжелая штука. Надо пройтись по всем памяти чтобы определить какие надо положить из одной корзины в другую, это очень затратно для JVM поэтому создали специальный который быстро отрабатывает но пр этом он есть очень много доп памяти ест.

Бывают различные сборщики мусора:

Сборщик мусора	Описание	Преимущества	Когда использовать	Флаги для включения
Serial	Использует один поток.	Эффективный, т.к. нет накладных расходов на взаимодействие потоков.	Однопроцессорные машины. Работа с небольшими наборами данных.	-XX:+UseSerialGC
Parallel	Использует несколько потоков.	Многопоточность ускоряет сборку мусора.	В приоритете пиковая производительность. Допустимы паузы при GC в одну секунду и более. Работа со средними и	-XX:+UseParallelGC
Z1	Выполняет всю тяжелую работу параллельно с работой приложения.	Низкая задержка.	В приоритете время отклика.	+UseZGC
G1	Выполняет некоторую тяжелую работу параллельно с работой приложения.	Может использоваться как на небольших системах, так и на больших с большим количеством процессоров и большим	Когда время отклика важнее пропускной способности. Паузы GC должны быть меньше одной секунды.	-XX:+UseG1GC

Та же схема работы памяти:



И тут видно что память общая и что могу быть проблемы обращаясь ко одной переменной из разных потоков.

Как создаются потоки

Два основных способа создать поток (я знала))) Унаследоваться от Thread или имплементировать Runnable.

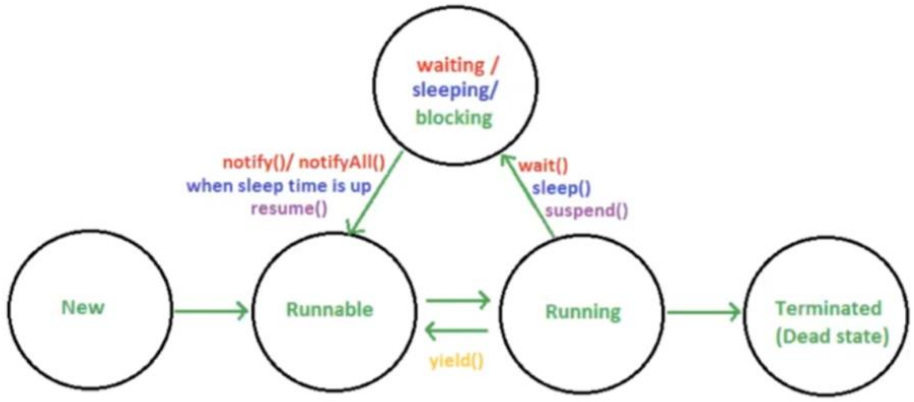


Fig. THREAD STATES

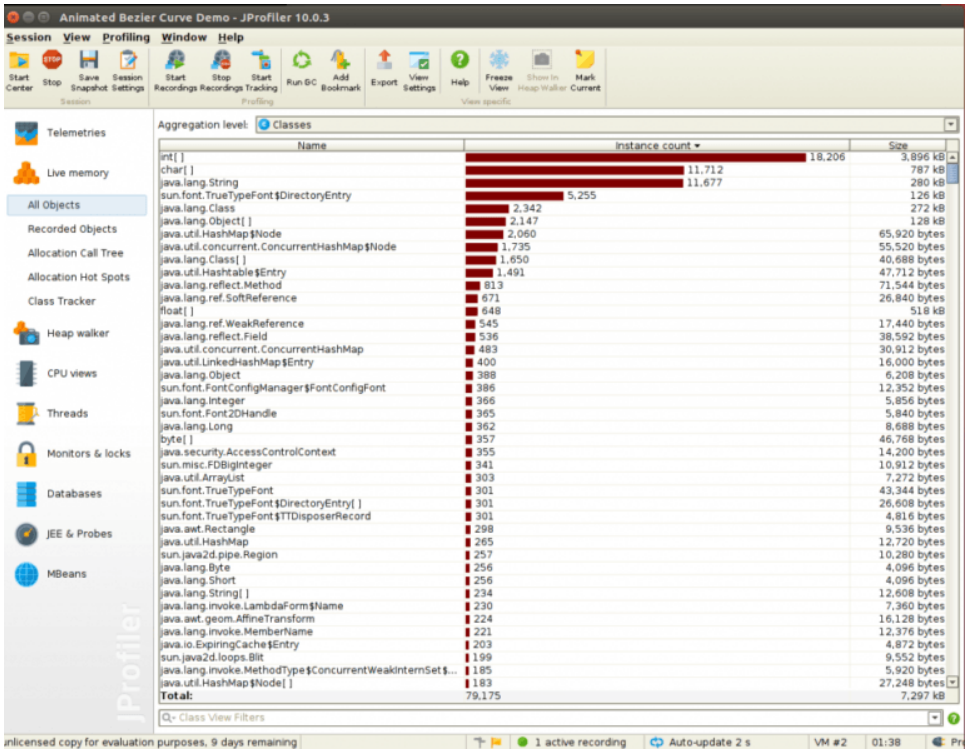
Когда создаем поток у него есть 3 статуса. Он может либо работать либо ждать.

Он как только создан он в статусе новый, когда вызываем метод старт - он переходит в runnable и дальше мы не можем сказать джаве взять поток. Мы можем только попросиить его (он готов возьми его) но операционка сама решит какой брать и когда.
Когда поток перейдет в состояние Running отсюда он работает и его можно остановить. Заставить ждать спать или он может быть заблокирован чем-то другим.

И как только он закончится он в статусе terminated.

Где можно увидеть эти статусы потока?

- В объекте потока по идее можно
- Метод аля isAlive или isInterrupted
- Но удобнее всего - (для больших приложений) - Profiler



При приеме на работу спрашивают что такое синхронизация потоков, изучить самостоятельно.

По простому: что такое синхронизация потоков?

Это как сделать так - чтобы какой-то определенный участок кода работал, только в одну единицу времени с одним потоком.

То есть чтобы два потока не могли пройтись по одному участку кода дновременно. В этом суть этого самого синхронизатора.

```
Object lock = new Object();

    Runnable task = () -> {
        synchronized (lock) {
            ...
        }
    };

    Thread th1 = new Thread(task);
    th1.start();
    synchronized (lock) {
        ...
    }
```

В Heape есть хедеры заголовки и MarkWord в котором есть ид потока который захватыает данный поток.

Mark Word

Bitfields			Tag	State
Hashcode	Age	0	01	Unlocked
Lock record address			00	Light-weight locked
Monitor address			10	Heavy-weight locked
Forwarding address, etc.			11	Marked for GC
Thread ID	Age	1	01	Biased / biasable

Практика

Kafka маст хев) 😀

Что такое JDK, JRE и JVM в Java

Javac – оптимизирующий компилятор языка java, включенный в состав многих

- JDK нужен для разработки (это компилятор, отладчик и т.д.).
- JRE нужен для запуска Java программ (содержит в себе JVM).
- JDK и JRE содержат JVM, которая нужна для запуска программ на Java.
- JVM является сердцем языка программирования Java и обеспечивает независимость от платформы.

Java Development Kit (JDK)

`Java Development Kit` является основным компонентом среды Java и предоставляет все инструменты, исполняемые и бинарные файлы, которые нужны для компиляции, отладки и выполнения программы на Java. JDK является платформо-зависимым программным обеспечением, поэтому есть отдельные инсталляторы для Windows, Mac и Unix-систем. Можно сказать, что `JDK` является надстройкой `JRE`, так как он содержит `JRE` с Java-компилятором, отладчиком и базовыми классами.

Виртуальная машина Java (JVM)

`JVM` является сердцем языка программирования Java. Когда мы запускаем программу, `JVM` несет ответственность за преобразование байт-кода в машинный код. JVM также зависит от платформы и предоставляет основные функции, такие как управления памятью Java, сборкой мусора, и т.д. Мы также можем выделять определенный объем памяти для JVM. JVM является виртуальной машиной, потому что обеспечивает интерфейс, который не зависит от операционной системы и аппаратных средств. Эта независимость от аппаратного обеспечения и операционной системы дает Java-программам возможность выполняться на

любом устройстве без необходимости внесения изменений — *Write once, run anywhere (Напиши раз — запускай где угодно)*.

Java Runtime Environment (JRE)?

`JRE` является реализацией `JVM`, которая предоставляет платформу для выполнения Java-программ. `JRE` состоит из виртуальной машины Java, бинарных файлов и других классов. `JRE` не содержит инструменты для разработки (компилятор Java, отладчик и т.д). Если вы хотите запустить любую Java программу, вы должны установить `JRE`.

Just-in-time Compiler (JIT) в Java

Just-in-time Compiler (JIT) является частью JVM. Он оптимизирует байт-код, уменьшая общее время, необходимое для компиляции байт-кода в машинный код.