1. Чем java отличается от с++?

Конкретно по памяти - наличием GC и отсутствием ручного удаления памяти

1. Что такое менеджер памяти?

Менеджер памяти — часть системы (программы или ОС), обрабатыващая запросы на выделение и освобождения оперативной памяти.

1. Какой механизм используется в Java для управления памятью?

GC

1. Опишите процесс работы сборщика мусора?

Проверяет YoungGen, помечает мертвые объекты/переносит из Eden’a в survive переносят те, которые пережили несколько запусков GC, исходя из философии: “Чем дольше живёт, тем сомнительнее, что умрёт”.

1. Какие алгоритмы сборщика вы знаете?

MSC – mark-sweep-compact. Пометить – убить – уплотнить.

1. Чем отличаются сборщики мусора?

Режимами работы.

1. Расскажите про утилиты для анализа памяти?  
   jconsole – встроенная gui утилита
2. Что такое ссылки?

Указатели на объекты -адрес ячейки памяти, где хранится объект

По сути это марка нашего объекта для gc, как именно его воспринимать и как к нему относиться.

1. Какие типы ссылок вы знаете?

Strong

Weak

Soft

Phantom

1. Чем они отличаются?  
   отсоритрованы по стойкости

Strong – живёт вечно

Soft – при нехватке памяти, GC их убьёт

Weak – при первом же вызове GC умрёт

Phantom – Если остались только фантом ссылки, то gc вызовет finalize, потом добавит объект в очередь ссылок, при этом сам объект еще не будет удален, пока не выйдет из очереди

1. Расскажите про String pool, Int pool.

В IntPool хранятся конкретные значения в диапазоне byte, на которые обёртки объектов 1) byte и boolean; 2) int и long в интервале [-128...127] и 3) char в интервале [0...127] будут ссылаться напрямую без создания объекта.Таким образом

Integer a = 127;

Integer b = 127;

Integer c = 128;

Integer d = 128;

System.out.println(a == b); // true

System.out.println(c == d); // false

При этом не стоит забывать, что new форсит создание объекта В ЛЮБОМ СЛУЧАЕ, что в случае StringPool, что в случае intpool и new Integer(127) будет неравно new Integer(127).

1. Расскажите о String.intern.

Форсируем добавление строки в StringPool

1. Расскажите, что такое профайлер?

Java Profiler - это инструмент, отслеживающий конструкции и операции байт-кода Java на уровне JVM . Эти конструкции и операции кода включают в себя создание объектов, итеративные выполнения (включая рекурсивные вызовы), выполнение методов, выполнение потоков и сборку мусора.

1. Расскажите, как использовать VisualVM.

Расскажите, чем отличается sampling от profiling? Это типы аудита. Режим работы в профайлере.

<https://stackoverflow.com/questions/39655945/whats-the-difference-between-sampler-and-profiler-in-jvisualvm#:~:text=A%20profiler%20is%20running%20all,data%20within%20a%20few%20seconds>.

A profiler is running all the time, so it gives you the complete call stack; at any given point in time.

A sampler only takes "snapshots" at distinct point in times.

Thing is: when you "profile" everything, then that slows down your JVM significantly; and it creates **enormous** amounts of data within a few seconds. Think about: the profiler will write down **each and any** method invocation that takes place!

So typically, you initially use a sampler, when you have "no idea" what is going on within your application. And then you just hope that the samples tell you something; like "hey, within our 10 000 samples, we are in that one method most of the time, why is that?" But as soon as you have a better understanding what you are "hunting" for, you would try to do a full profiler run in order to capture the whole call chain that leads into some method.

And then there is some "middle ground" - where you profile "everything" but exclude things. In other words: most profilers allow you to say "do **not** profile methods in classes in this or that package". But of course - excluding packages/hierarchies only makes sense when you already have a pretty good feeling which direction you intend to investigate.

1. Расскажите о benchmark. Приведите примеры кода.

Аннотации для проверки очень маленьких методов, для которых даже вызов GC существенно затормозит работу. Используется как альтернатива разницы таймстампов для анализа работы. Включён в JMH.

1. Расскажите о методы finalized.

Вызывается когда объект помечен как мертвый, но ещё не удален.

Выполняется в отдельном потоке finalized. Объект не может быть удалён, пока метод не выполнится. Поток финализатор не добавляет в себя объекты у которыхfinalized не переопределён или пустой.

Все exception в finalize он пропускает. В finalize можно создать strong ссылку на объект и тем самым спасти его от GC, но сам объект, КЯП, уже будет отмечен, как finalize-runned и второй раз, когда он будет помечен трупом – метод вызван не будет.

1. Расскажите о методы clone? Deep clone and shallow clone.

Сам интерфейс – маркировочный для JVM, говорящий о том, что объект может быть клонирован.

Shallow clone – базовый метод clone, который просто копирует все поля одного объекта. Если полем был Объект, то скопируется ссылка. И таким образом через копию такого объекта можно будет продолжать влиять на прародителя.

DeepClone – переопределение метода clone() ручками, где мы явно прописываем что делать с объектами. Таким образом, мы хотим получать непересекающиеся на 100% объекты.

В обоих случаях будет создан новый объект в куче.

При этом стоит учесть, что ShallowClone дешевый и быстрый, DeepClone – дорогой и долгий.

1. Расскажите о Stack, Heap.

1) Stack (стек) - область памяти для хранения ЛОКАЛЬНЫХ переменных, создаваемых в методе.

2) Если локально создается объект, то сам объект помещается в Heap, а ссылка на него - в Stack.

3) Stack значительно меньше, чем Heap, скорость доступа - значительно быстрее.

4) Если память стека полностью занята (н-р бесконечная рекурсия), выбрасывается StackOverflowException.

5) Память Stack не требует сборщика мусора - блок полностью очищается при выходе из метода.

6.1) Устройство: по принципу LIFO. Вызван метод - ему выделяется блок стек-памяти и кладется "сверху".

6.2) Внутри вызван подметод - выделяется еще блок памяти и кладется "сверху".

6.3) Всегда доступен только последний, самый "верхний" блок памяти.

6.4) Метод отработал - его верхний блок памяти очищается, становится доступен более нижний блок - обрамляющего метода.

Heap бьётся на:

1. Eden + Survived - youngGen
2. olgGen

**PermGen**

1) PermGen существовал в Java версии 7 и ниже. В Java 8+ заменен на MetaSpace.

2) PermGen - это область в heap, зарезервированная под описания классов, методов, статических переменных.

3) Область PermGen не проверялась сборщиком мусора.

4) До Java 7 в PermGen также находился String pool.

**MetaSpace получается отдельно от Heap!**

MetaSpace (java 8+) permGen (jave 7-)

1) MetaSpace - область памяти, отдельная от Heap. Зарезервирована JVM для размещения классов, методов, статических переменных.

2) MetaSpace отделена от основной памяти JVM. Умеет автоматически расширяться, пока позволяет операционная система (или до заданного макс. значения).

3) MetaSpace появилась в Java 8+ как замена PermGen.

4) MetaSpace проверяется сборщиком мусора на "мертвые объекты".

5) Раньше класс загружался вместе с JVM и умирал с ее выключением. Сейчас есть динамическая загрузка/выгрузка классов в ходе жизни JVM. Отсюда - нужна сборка "мертвых" классов и расширение под добавленные.

*1) PermGen не проверяется сборщиком мусора, MetaSpace - проверяется. Уменьшили вероятность OutOfMemoryError.*

*2) PermGen не могла расширяться, MetaSpace - способна автоматически расширяться, пока есть доступная память операционной системы.*