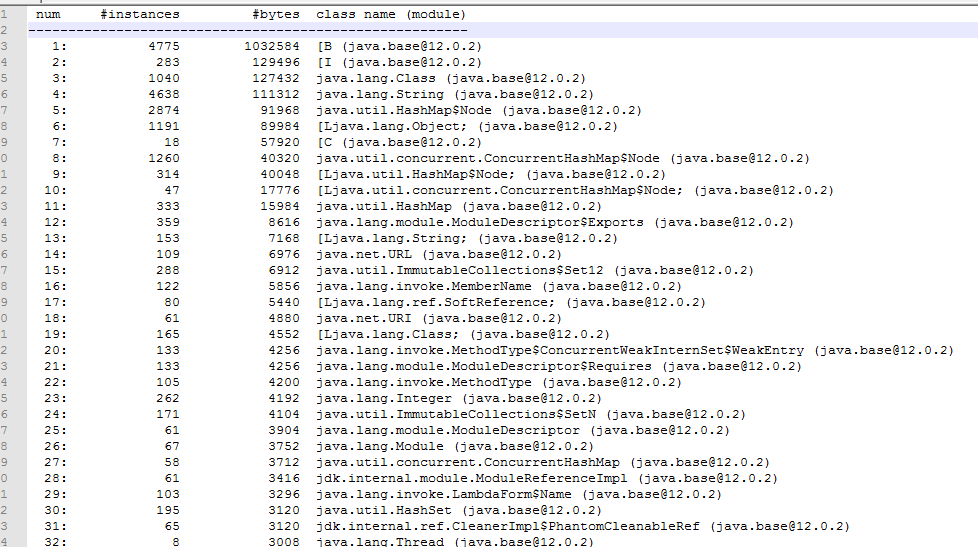
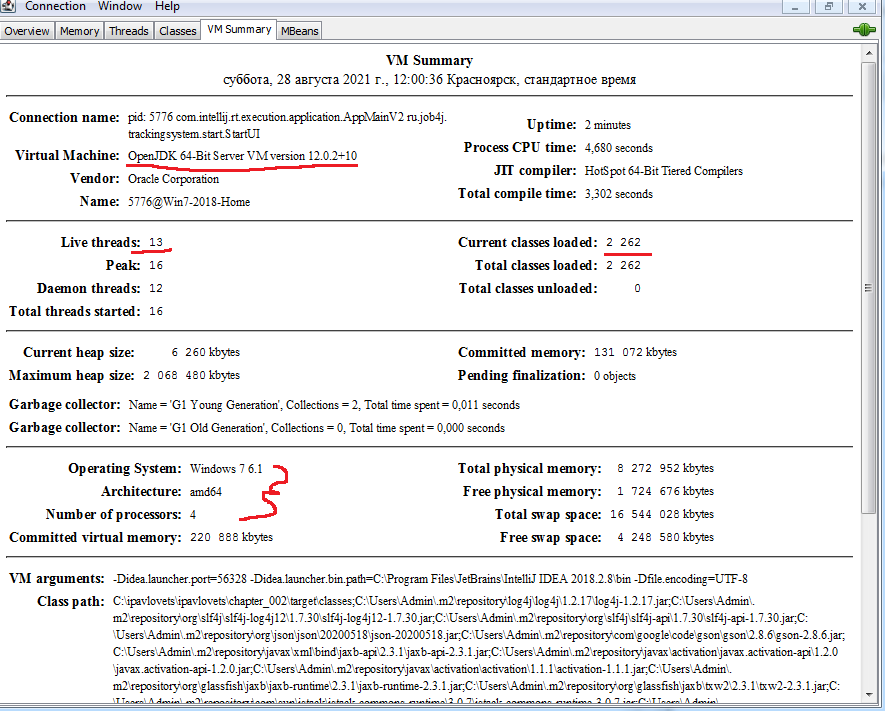
1) Команда **jmap –histo** распечатает гистограмму кучи. Когда команда выполняется в запущенном процессе, инструмент печатает количество объектов, размер памяти в байтах и полное имя класса для каждого класса. Внутренние классы виртуальной машины Java HotSpot заключены в угловые скобки. Гистограмма полезна для понимания того, как используется куча. “jmap -histo:live pid>a.log” – вывод в файл



2) с помощью инструмента **jconsole** можно узнать некоторые параметры окружения системы в которой запущено приложение:

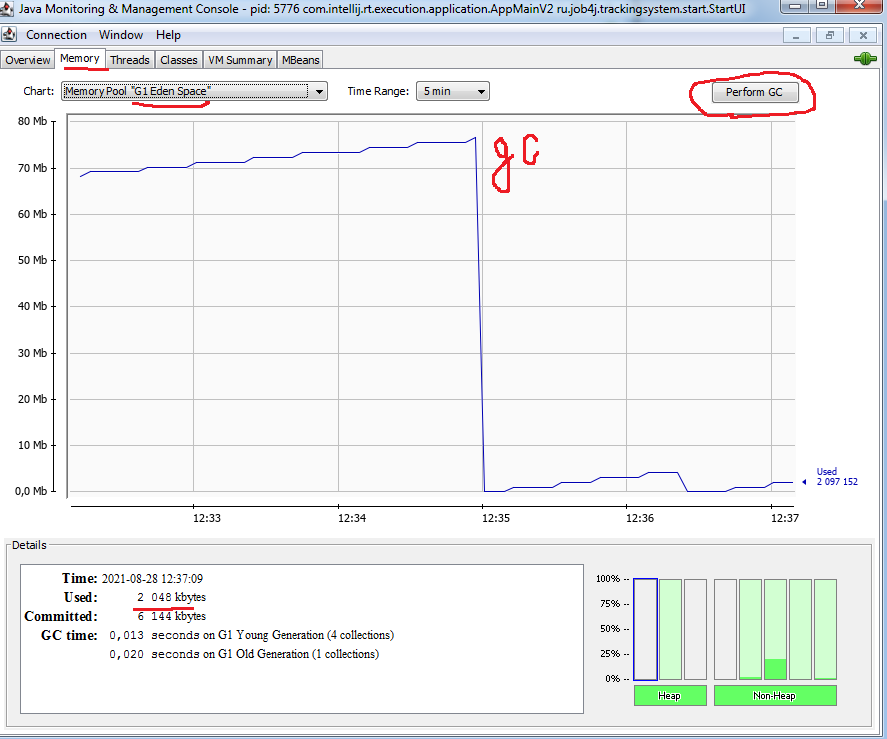
* версия JDK;
* версия операционной системы, архитектура и количество процессоров машины;
* количество запущенных потоков;
* размеры heap;
* количество используемых классов в системе.

Также мы можем выбрать версию GC который выбрала JVM для этого приложения, в нашем случае G1.

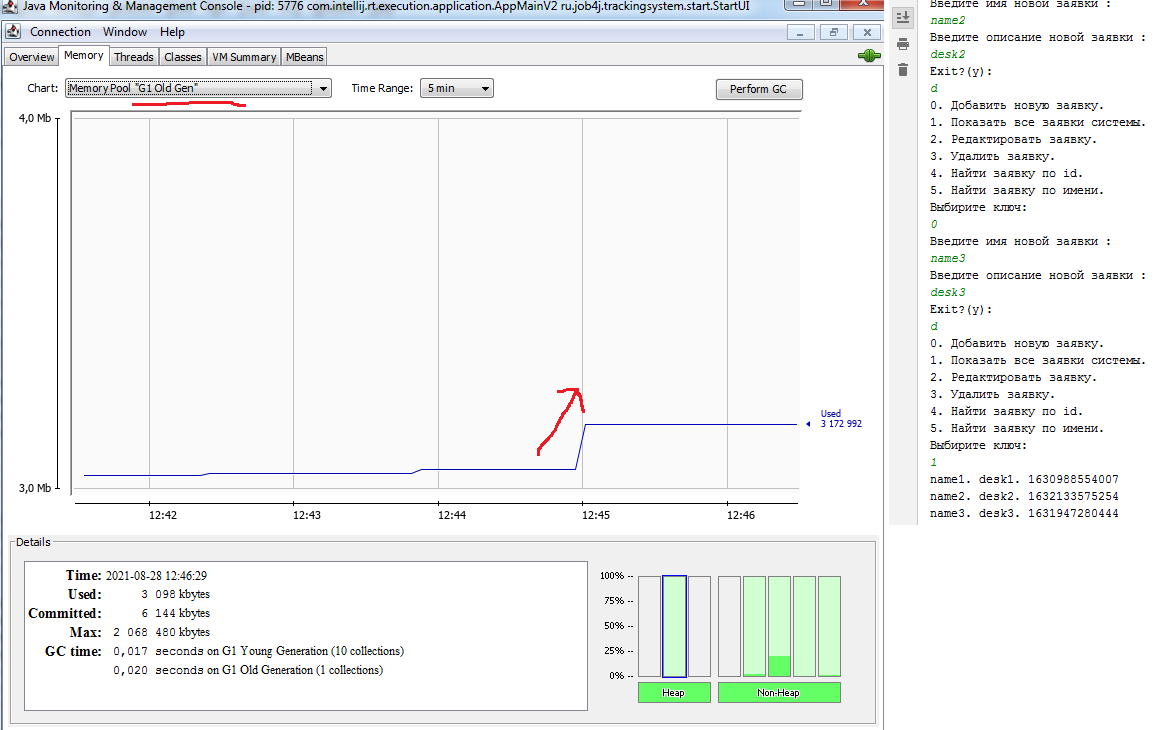


Соответственно в динамике некоторые из этих показателей могут меняться по ходу работы программы, например при создании новой заявки в приложении tracer, с реализацией MemTracker()количество используемых классов возрастёт.

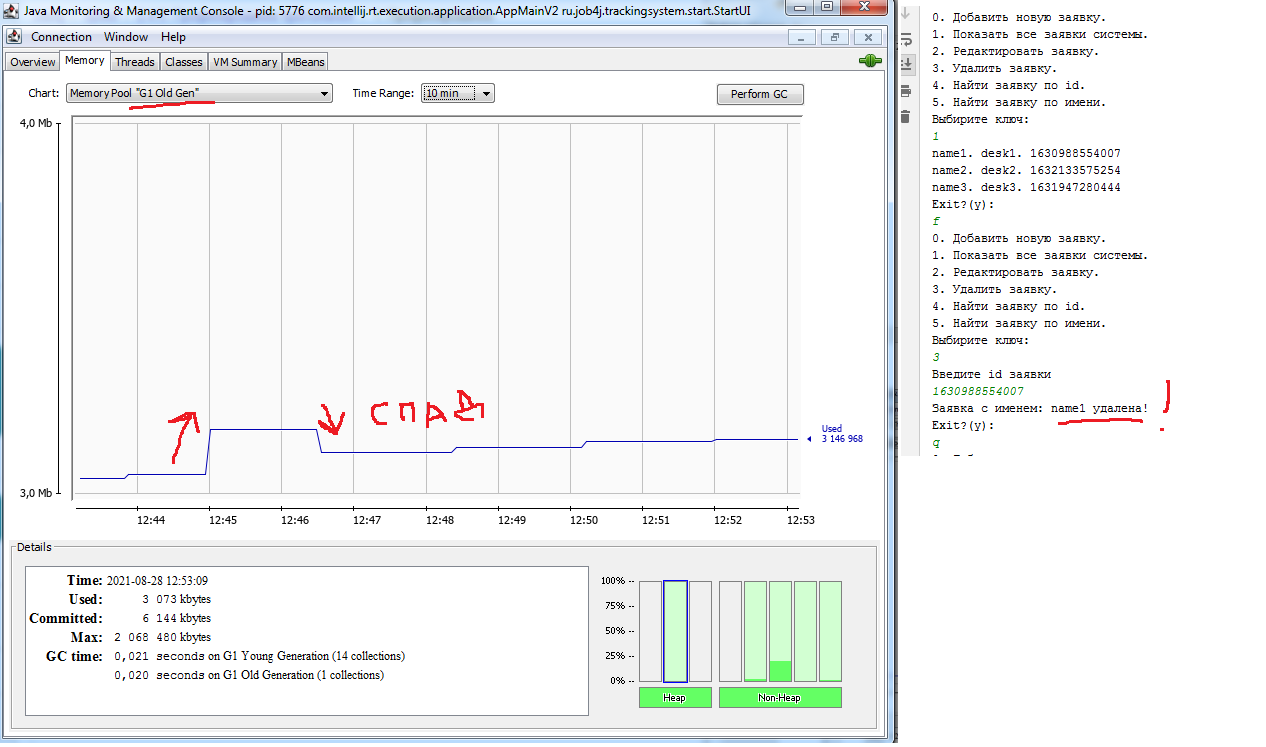
Вкладка **Memory** позволяет рассмотреть различные области памяти и выполнить очистку мусора (perform gc) самостоятельно. Ниже можно видеть как происходит очищение этого региона (Eden) после gc, кривая показывает сокращение количества обьектов в Eden.



При создании нескольких заявок и выводи информации о них в консоль можно увидеть рост количества занимаемых объектов в Old Gen.

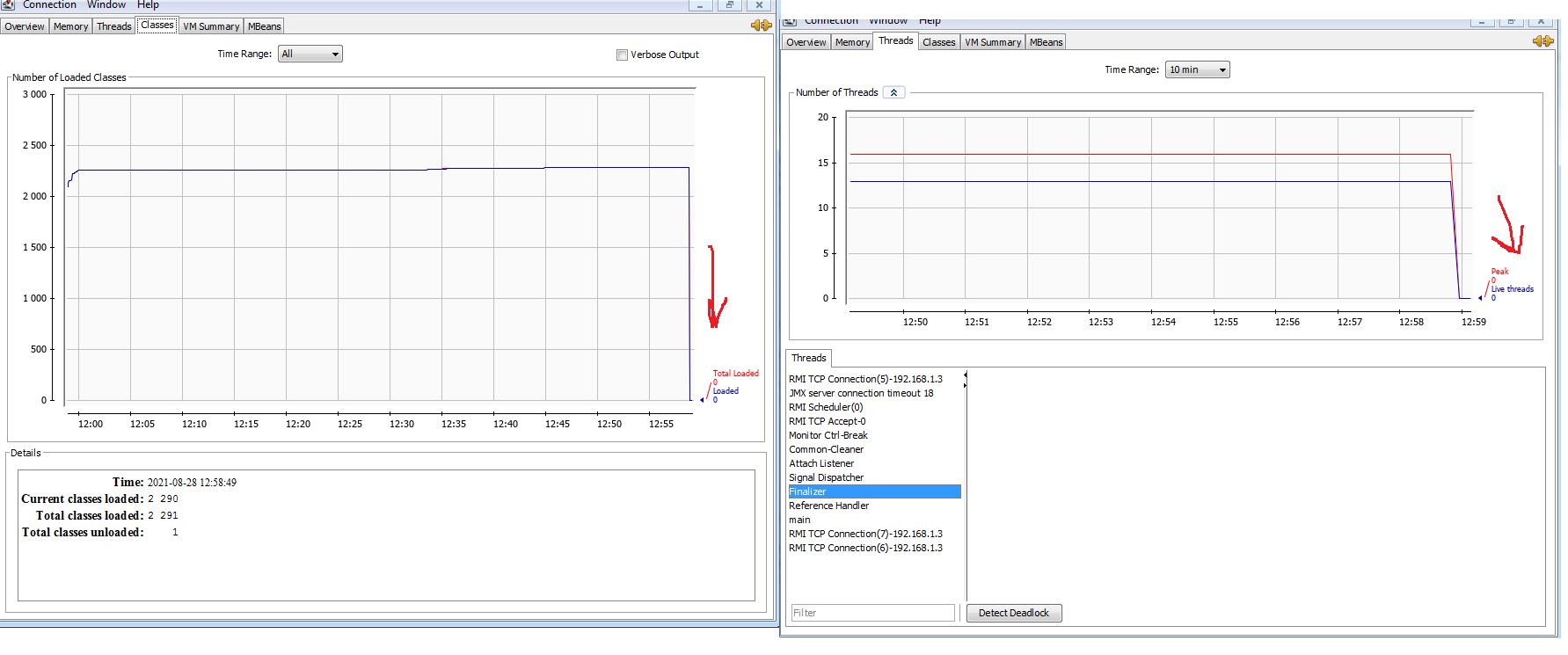


После этого я удалил одну заявку и сразу кривая распределения памяти показывает спад.



После завершения приложения по ключу “y” можно видеть как все кривые распределения памяти, классов и потоков упали до нуля.





Подытоживая можно сказать что с помощью инструмента jcosole можно видеть наглядно, с помощью кривых, распределение памяти в динамике, при выборе различных действий приложения пользователем. Также можно посмотреть некоторую информацию о системном окружении.