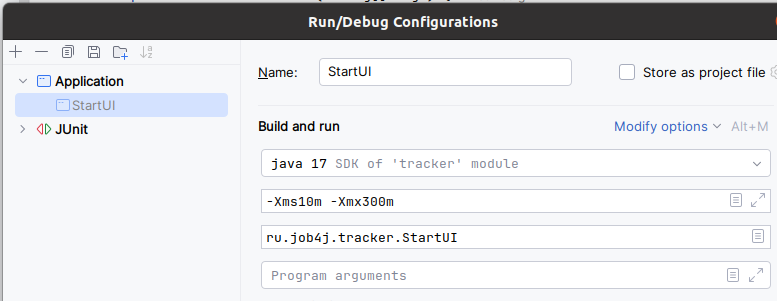
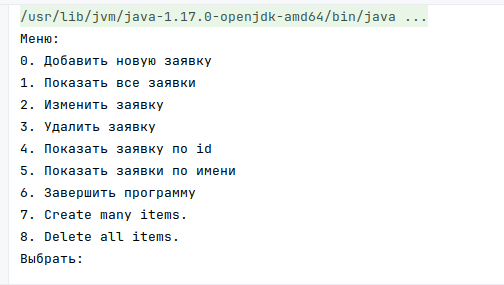
1. **Запустил StartUI с опциями JVM: -Xms10m -Xmx300m, то есть ограничил размер Heap 300МБ**





Вывод команды до создания объектов Item:

**jmap -histo 10900**

num #instances #bytes class name (module)

-------------------------------------------------------

1: 22140 2035312 [B (java.base@17.0.15)

2: 485 640800 [C (java.base@17.0.15)

3: 19932 478368 java.lang.String (java.base@17.0.15)

4: 2777 336480 java.lang.Class (java.base@17.0.15)

5: 1367 333432 [I (java.base@17.0.15)

6: 3983 262688 [Ljava.lang.Object; (java.base@17.0.15)

7: 6231 199392 java.util.HashMap$Node (java.base@17.0.15)

8: 1686 148368 java.lang.reflect.Method (java.base@17.0.15)

9: 4078 130496 java.util.concurrent.ConcurrentHashMap$Node (java.base@17.0.15)

10: 591 90336 [Ljava.util.HashMap$Node; (java.base@17.0.15)

11: 1194 47760 java.util.LinkedHashMap$Entry (java.base@17.0.15)

12: 1382 47504 [Ljava.lang.String; (java.base@17.0.15)

13: 1648 42800 [Ljava.lang.Class; (java.base@17.0.15)

14: 123 41968 [Ljava.util.concurrent.ConcurrentHashMap$Node; (java.base@17.0.15)

15: 461 33192 java.lang.reflect.Field (java.base@17.0.15)

16: 8 32896 [Ljava.nio.ByteBuffer; (java.base@17.0.15)

17: 676 32448 java.util.HashMap (java.base@17.0.15)

18: 377 27144 java.lang.reflect.Constructor (java.base@17.0.15)

19: 418 26752 java.util.concurrent.ConcurrentHashMap (java.base@17.0.15)

20: 550 26400 java.lang.invoke.MemberName (java.base@17.0.15)

В выводе команды jmap отсутствуют объекты Item, созданы стандартные объекты java - String, Class, Object, HashMap.

Вывод команды до создания заявок:

**jstat -gc 10900 1s 150000**

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 0,0 0,0 0,0 5120,0 3072,0 5120,0 0,0 0,0 0,0 0,0 0,0 0 0,000 0 0,000 0 0,000 0,000

0,0 1024,0 0,0 1024,0 2048,0 0,0 9216,0 2040,5 2496,0 2331,1 320,0 229,9 1 0,002 0 0,000 0 0,000 0,002

0,0 1024,0 0,0 801,1 3072,0 0,0 8192,0 3185,5 5568,0 5366,2 704,0 602,8 4 0,012 0 0,000 2 0,001 0,013

0,0 1024,0 0,0 705,1 5120,0 1024,0 10240,0 3898,5 7040,0 6812,0 896,0 769,1 6 0,023 0 0,000 4 0,008 0,031

0,0 1024,0 0,0 705,1 5120,0 1024,0 10240,0 3898,5 7040,0 6812,0 896,0 769,1 6 0,023 0 0,000 4 0,008 0,031

0,0 1024,0 0,0 705,1 5120,0 1024,0 10240,0 3898,5 7040,0 6812,0 896,0 769,1 6 0,023 0 0,000 4 0,008 0,031

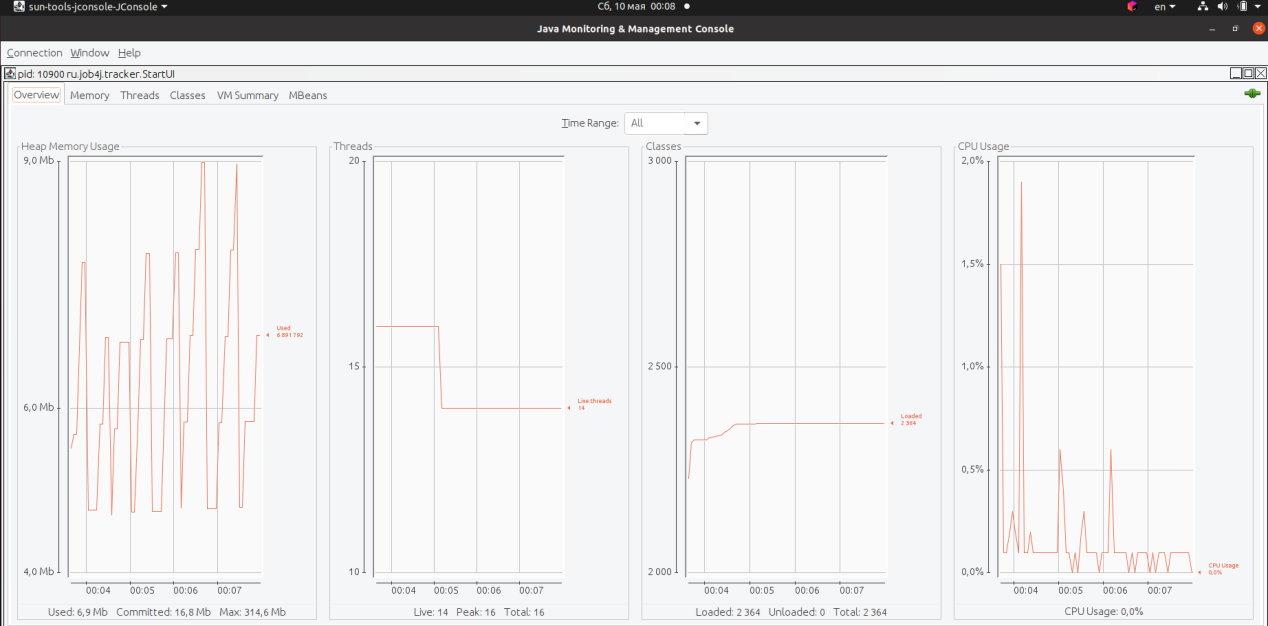
0,0 1024,0 0,0 705,1 5120,0 1024,0 10240,0 3898,5 7040,0 6812,0 896,0 769,1 6 0,023 0 0,000 4 0,008 0,031

0,0 1024,0 0,0 705,1 5120,0 2048,0 10240,0 3898,5 7040,0 6812,0 896,0 769,1 6 0,023 0 0,000 4 0,008 0,031

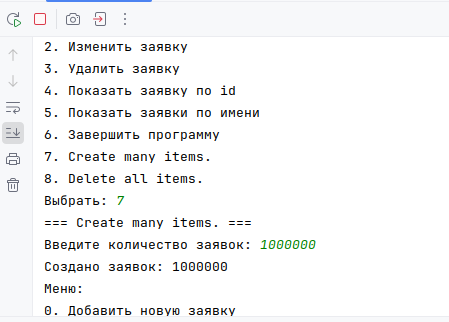
При запуске программы Eden Usage = 3072 Кб, после чего упало до 0, зетем снова выросло до значения 5120КБ. В момент падения EU до 0 видно, что вырос счетчик Young GC, значит была выполнена сборка мусора в Young, которая очистила Eden. Всего было выполнено 6 Young GC

Показатель OU вырос с 0 до 3898,5 Кб, что означает что после 6 сборок мусора в Young GC часть объектов переместилась в Old Generation.

Утилита **jconsole** показывает, что в Heap используется не более 10МБ



1. **Выбираем действие на массовой создание заявок, создаём 1 000 000 заявок**



После создания миллиона заявок:

Вывод команды:

**jmap -histo 10900**

num #instances #bytes class name (module)

-------------------------------------------------------

1: 1026844 51110752 [B (java.base@17.0.15)

2: 1022413 24537912 java.lang.String (java.base@17.0.15)

3: 1000154 24003696 java.time.LocalDateTime (java.base@17.0.15)

4: 1000092 24002208 java.time.LocalDate (java.base@17.0.15)

5: 1000027 24000648 java.time.LocalTime (java.base@17.0.15)

6: 1000000 24000000 ru.job4j.tracker.Item

7: 8478 10726696 [Ljava.lang.Object; (java.base@17.0.15)

8: 2542 6792336 [I (java.base@17.0.15)

9: 1160 2228280 [C (java.base@17.0.15)

10: 2767 335168 java.lang.Class (java.base@17.0.15)

11: 6208 198656 java.util.HashMap$Node (java.base@17.0.15)

12: 4226 169040 java.util.TreeMap$Entry (java.base@17.0.15)

13: 4060 129920 java.util.concurrent.ConcurrentHashMap$Node [(java.base@17.0.15)](mailto:(java.base@17.0.15))

Видим, что появились 1 миллион объектов Item и примерно столько же объектов String и LocalDateTime, которые являются полями класса Item.

Вывод команды

jstat -gc 10900

jstat -gc 10900

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

0,0 1024,0 0,0 331,2 5120,0 1024,0 10240,0 4321,5 7296,0 7059,2 896,0 783,5 7 0,021 0 0,000 4 0,002 0,022

0,0 1024,0 0,0 331,2 5120,0 1024,0 10240,0 4321,5 7296,0 7059,2 896,0 783,5 7 0,021 0 0,000 4 0,002 0,022

0,0 1024,0 0,0 331,2 5120,0 2048,0 10240,0 4321,5 7296,0 7059,2 896,0 783,5 7 0,021 0 0,000 4 0,002 0,022

0,0 3072,0 0,0 3072,0 14336,0 8192,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

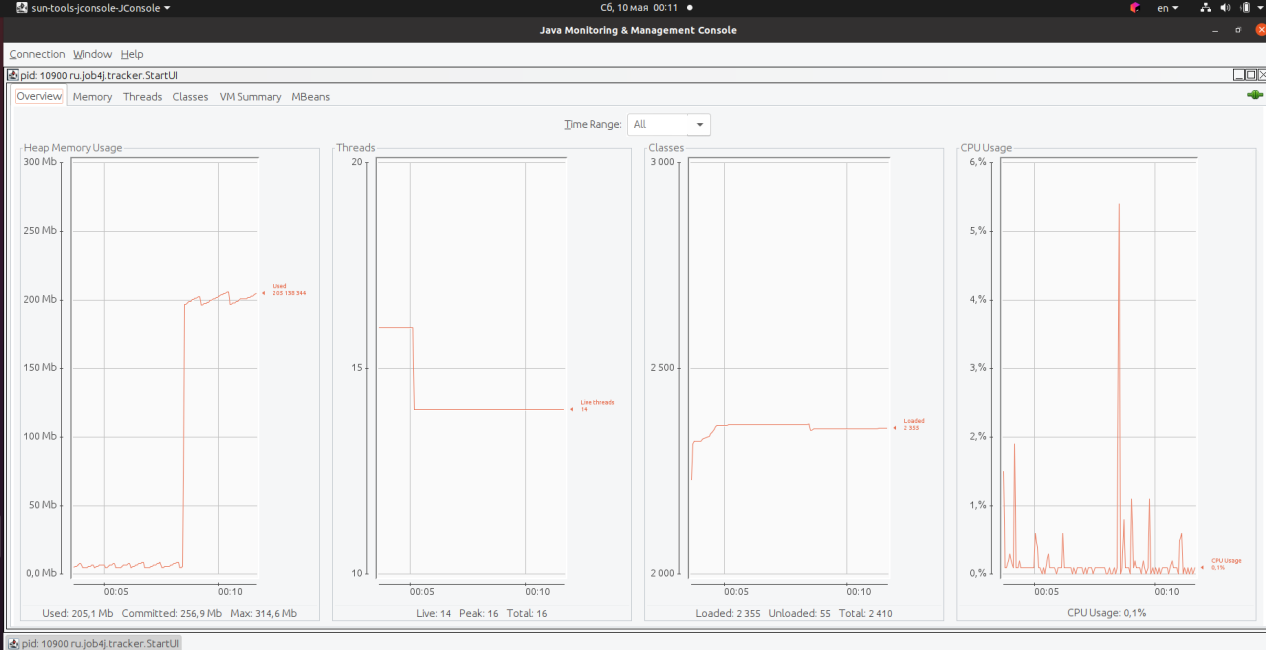
0,0 3072,0 0,0 3072,0 14336,0 9216,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

0,0 3072,0 0,0 3072,0 14336,0 9216,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

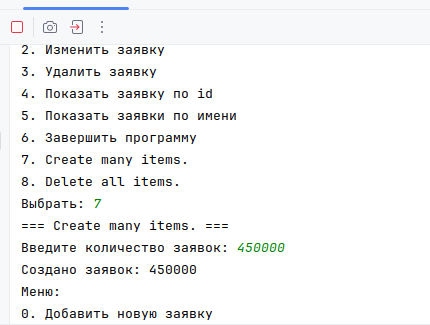
0,0 3072,0 0,0 3072,0 14336,0 10240,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

Показатель EU вырос с 1024,0 КБ до 10240,0 КБ значит в системе активно создаются новые объекты Item. При этом показатель Old Generation Usage вырос с 4321,5 Кб до 182196,0 Кб. Резкий скачок размера Old Generation означает, что происходит активный переход объектов между Young и Old. Видим, что YGC вырос с 7 до 36 - это означает что область Eden заполнялась и запускался механизм сборки мусора в Young. Также было 10 Concurent GC, но они не уменьшили размер Old Generation, так как на объекты Item есть ссылки и они ещё используются.

Утилита **jconsole** сразу после создания 1 миллиона заявок Item показывает рост использования Heap до 200МБ



1. Создаю дополнительные 450 тыс заявок Item:



После создания дополнительных 450 000 заявок Item:

Вывод команды jmap показывает рост объектов Item до 1 450 000 объектов, а также LocalDatetIme, String, которые являются полями класса Item.

**jmap -histo 10900**

num #instances #bytes class name (module)

-------------------------------------------------------

1: 1477213 72819960 [B (java.base@17.0.15)

2: 1472724 35345376 java.lang.String (java.base@17.0.15)

3: 1450154 34803696 java.time.LocalDateTime (java.base@17.0.15)

4: 1450092 34802208 java.time.LocalDate (java.base@17.0.15)

5: 1450027 34800648 java.time.LocalTime (java.base@17.0.15)

6: 1450000 34800000 ru.job4j.tracker.Item

7: 8399 12617088 [Ljava.lang.Object; (java.base@17.0.15)

8: 3390 3937784 [I (java.base@17.0.15)

9: 1176 2441864 [C (java.base@17.0.15)

10: 2765 334920 java.lang.Class [(java.base@17.0.15)](mailto:(java.base@17.0.15))

Вывод команды jstat -gc 10900 после дополнительного создания 450 000 объектов Item:

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

0,0 3072,0 0,0 3072,0 14336,0 8192,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

0,0 3072,0 0,0 3072,0 14336,0 9216,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

0,0 3072,0 0,0 3072,0 14336,0 9216,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

0,0 3072,0 0,0 3072,0 14336,0 10240,0 233472,0 182196,0 7936,0 7510,5 960,0 780,7 36 0,146 0 0,000 10 0,042 0,188

0,0 2048,0 0,0 2048,0 14336,0 10240,0 290816,0 257460,0 8000,0 7551,7 960,0 781,3 45 0,189 0 0,000 14 0,095 0,283

0,0 2048,0 0,0 2048,0 14336,0 10240,0 290816,0 257460,0 8000,0 7551,7 960,0 781,3 45 0,189 0 0,000 14 0,095 0,283

0,0 2048,0 0,0 2048,0 14336,0 10240,0 290816,0 257460,0 8000,0 7551,7 960,0 781,3 45 0,189 0 0,000 14 0,095 0,283

0,0 2048,0 0,0 2048,0 14336,0 11264,0 290816,0 257460,0 8000,0 7551,7 960,0 781,3 45 0,189 0 0,000 14 0,095 0,283

0,0 2048,0 0,0 2048,0 14336,0 11264,0 290816,0 257460,0 8000,0 7551,7 960,0 781,3 45 0,189 0 0,000 14 0,095 0,283

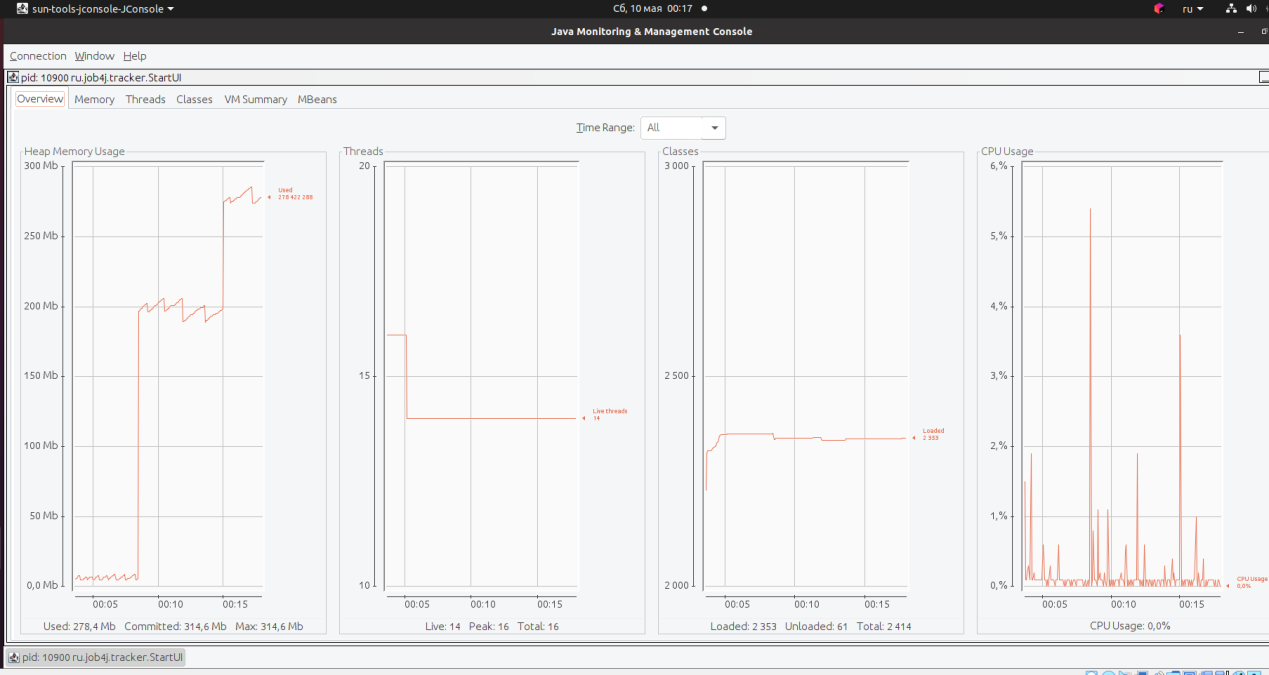
Показатель Eden Used EU вырос до 11264,0 Кб, количестов Young GC увеличилось до 45, значит было выполнено 9 сборок мусора в Young Generation. Область Eden заполнялась новыми объектами и вызывался сборщик мусора в Young Generation.

Показатель Old Used OU увеличилось с 182196,0 Кб до 257460,0 Кб. Это означает что в результате Young GC объекты были перенесены из Young в Old Generation.

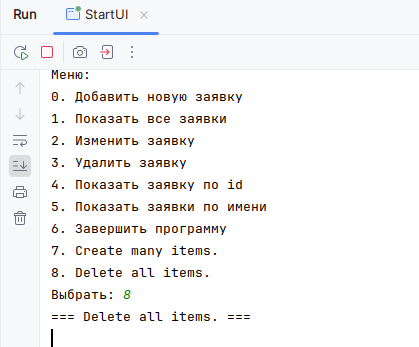
Увеличился показатель счетчика CGC - до 4, но размер Old Used не изменился и это означает, что в Old geneartion всё ещё харянтся живые объекты, на которые есть ссылки.

Full GC всё ещё не выполнялся, так как в Heap всё ещё есть память.

Утилита **jconsole** сразу после создания дополнительных 450 000 заявок Item показывает рост использования Heap до 260МБ



1. **Запускаю действие по удалению всех заявок:**



Вывод команды jmap показывает, что объекты Item удаляются:

jmap -histo 10900

num #instances #bytes class name (module)

-------------------------------------------------------

1: 1110324 54996856 [B (java.base@17.0.15)

2: 1106148 26547552 java.lang.String (java.base@17.0.15)

3: 1084162 26019888 java.time.LocalDateTime (java.base@17.0.15)

4: 1084100 26018400 java.time.LocalDate (java.base@17.0.15)

5: 1084035 26016840 java.time.LocalTime (java.base@17.0.15)

6: 1084008 26016192 ru.job4j.tracker.Item

7: 1450227 23203632 java.lang.Integer (java.base@17.0.15)

8: 8056 13533304 [Ljava.lang.Object; (java.base@17.0.15)

9: 1068 2008712 [C (java.base@17.0.15)

10: 2214 889904 [I (java.base@17.0.15)

11: 2769 335376 java.lang.Class (java.base@17.0.15)

12: 6233 249320 java.util.TreeMap$Entry (java.base@17.0.15)

13: 6173 197536 java.util.HashMap$Node (java.base@17.0.15)

14: 4043 129376 java.util.concurrent.ConcurrentHashMap$Node (java.base@17.0.15)

15: 4232 101568 java.lang.Long (java.base@17.0.15)

16: 892 98512 [Ljava.util.HashMap$Node; (java.base@17.0.15)

17: 1928 92544 java.util.TreeMap [(java.base@17.0.15)](mailto:(java.base@17.0.15))

Число объектов Item в куче уменьшилось с 1 450 000 до 1 084 162объектов.

Вывод команды jstat после запука удаления:

jstat -gc 10900

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

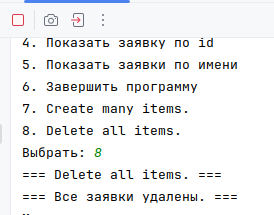
0,0 2048,0 0,0 2048,0 14336,0 3072,0 290816,0 282848,4 8000,0 7592,3 960,0 782,9 55 0,230 1 0,101 16 0,097 0,428

0,0 1024,0 0,0 368,0 15360,0 7168,0 290816,0 232812,4 8064,0 7621,0 960,0 782,9 56 0,233 1 0,101 18 0,098 0,432

0,0 1024,0 0,0 368,0 15360,0 9216,0 290816,0 232812,4 8064,0 7621,0 960,0 782,9 56 0,233 1 0,101 18 0,098 0,432

OU резко снизился с 282848,4 до 232812,4 KB после выполнения 2-х Concurent GC, так как счетчик CGC увеличилс с 16 до 18.

Метод по удалению всех заявок из системы успешно завершился.



Через 5 минут после завершения метода количество объектов в Item упало до 0:

**jmap -histo 10900**

num #instances #bytes class name (module)

-------------------------------------------------------

1: 15330 8089104 [Ljava.lang.Object; (java.base@17.0.15)

2: 38083 5327448 [B (java.base@17.0.15)

3: 2297 5105392 [C (java.base@17.0.15)

4: 4198 914144 [I (java.base@17.0.15)

5: 28568 685632 java.lang.String (java.base@17.0.15)

6: 2775 336048 java.lang.Class (java.base@17.0.15)

7: 7050 282000 java.util.TreeMap$Entry (java.base@17.0.15)

Вывод команды jstat -gc 10900

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

0,0 1024,0 0,0 99,8 15360,0 11264,0 290816,0 232812,4 8064,0 7646,1 960,0 783,4 57 0,234 1 0,101 18 0,098 0,432

0,0 1024,0 0,0 78,0 22528,0 10240,0 283648,0 230815,9 8064,0 7657,5 960,0 785,5 58 0,235 1 0,101 18 0,098 0,433

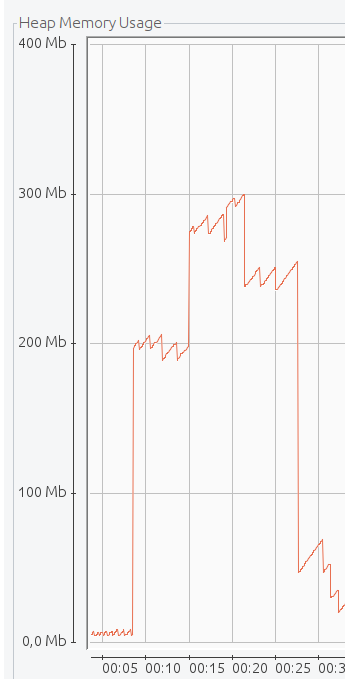
0,0 1024,0 0,0 78,0 22528,0 11264,0 283648,0 230815,9 8064,0 7657,5 960,0 785,5 58 0,235 1 0,101 18 0,098 0,433

0,0 1024,0 0,0 78,1 24576,0 1024,0 142336,0 45703,0 8064,0 7633,5 960,0 781,8 59 0,236 1 0,101 20 0,100 0,437

0,0 1024,0 0,0 78,1 24576,0 2048,0 142336,0 45703,0 8064,0 7633,5 960,0 781,8 59 0,236 1 0,101 20 0,100 0,437

Видим, что память Old Generation резко снизилась с 232812,4 Кб до 45703,0 Кб. В это время увеличилось число Concurent GC с 18 до 20. Concurent GC очистило Old generation. Общее время работы Garbage Collector за всё время рабоыт программы остставляет 0,437 секнуд. Сборка мусора в Heap проходит эффективно.

Утиита **jconsole** показывает, что размер используемой памяти в куче уменьшился:



1. После удаления всех заявок в ситсеме запускаю действие на создание 10 миллионов заявок Item.



При создании 10 миллионов заявок программа завершилась ошибкой:

Exception in thread "main" **java.lang.OutOfMemoryError: Java heap space**

Вывод утилиты **jstat** показал:

S0C S1C S0U S1U EC EU OC OU MC MU CCSC CCSU YGC YGCT FGC FGCT CGC CGCT GCT

0,0 1024,0 0,0 164,5 78848,0 15360,0 88064,0 19456,7 8064,0 7679,9 960,0 786,6 62 0,237 1 0,101 20 0,100 0,438

0,0 1024,0 0,0 164,5 78848,0 15360,0 88064,0 19456,7 8064,0 7679,9 960,0 786,6 62 0,237 1 0,101 20 0,100 0,438

0,0 0,0 0,0 0,0 1024,0 0,0 306176,0 305213,7 8064,0 7664,8 960,0 782,8 113 0,434 6 0,535 22 0,102 1,071

0,0 0,0 0,0 0,0 1024,0 0,0 306176,0 305174,1 8064,0 7605,2 960,0 773,7 117 0,450 13 1,615 22 0,102 2,167

0,0 0,0 0,0 0,0 1024,0 0,0 306176,0 305172,7 8064,0 7605,2 960,0 773,7 119 0,454 18 2,565 22 0,102 3,120

Размер Old Generation OU резко увеличился с 19456,7 Кб до 305172,7 Кб, при том что в опциях JVM указан максимальный размер Heap 300МБ. То есть при создании 10 миллионов заявок Item память Old Generation была полностью заяполнена. При заполнении Old Generation начинает работу Full GC. Мы видим, что резко увеличился счетчик запуска Full GC с 1 до 18, но при этом размер Old Generation так и оставался 306176,0 Кб, то есть очистка Full GC не помогала. Общее время работы GC редко увеличилось с 0,438 до 3,120 секунд, что уже заметно. Full GC не помогла очистить память в Old Generation, так как на создаваемые объекты Item есть ссылки в памяти, а так как мы создаём 10 Миллионов заявок, то заявки заполняют Eden и очнь быстро заполняют всёместо в Old Geneartion. Полная сборка мусора Full GC не помогла очистить Old generation, так как все объекты в OLd generation были живыми. Так как место в Heap закнчилось, а полная очистка Full GC не освобождает память, то программа завершилась выбросом исключения Exception in thread "main" java.lang.OutOfMemoryError: Java heap space/

Утилиты **jconsole** показывает резкий рост утилизации Heap при создании 10 миллионов заявок, а также падение утилизации Heap до 0, так как программа аварийно завершилась выбросом исключения **OutOfMemory**:

