Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образование учреждение высшего образования “Национальный исследовательский университет ИТМО”

ОТЧЁТ

По лабораторной работе №4
По дисциплине “Основы Программной Инженерии”
Вариант 345949

Авторы:
Билошицкий Михаил Владимирович
Трошкин Александр Евгеньевич
Факультет:
ПИиКТ
Преподаватель:
Письмак Александр Евгеньевич

Санкт-Петербург, 2024

Содержание

Задание

Вариант 345949

1. Для своей программы из лабораторной работы №3 по дисциплине "Веб-программирование" реализовать:

- MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если пользователь совершил 3 "промаха" подряд, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.

2. С помощью утилиты JConsole провести мониторинг программы:

- Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
- Определить наименование и версию JVM, поставщика виртуальной машины Java и номер её сборки.

3. С помощью утилиты VisualVM провести мониторинг и профилирование программы:

- Снять график изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени.
- Определить имя потока, потребляющего наибольший процент времени CPU.

4. С помощью утилиты VisualVM и профилирование IDE NetBeans, Eclipse или Idea локализовать и устранить проблемы с производительностью в программе. По результатам локализации устранения проблемы необходимо составить отчёт, в котором должна содержаться следующая информация:

- Описание выявленной проблемы.
- Описание путей устранения выявленной проблемы.
- Подробное (со скриншотами) описание алгоритма действий, который позволил выявить и локализовать проблему.

Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

Отчёт по работе должен содержать:

1. Текст задания.
2. Исходный код разработанных MBean-классов и сопутствующих классов.
3. Скриншоты программы JConsole со снятыми показаниями, выводы по результатам профилирования.
4. Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования.
5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.
6. Выводы по работе.

Вопросы к защите лабораторной работы:

1. Мониторинг и профилирование. Основные понятия. Отличия мониторинга от профилирования.
2. Инфраструктура для организации мониторинга и профилирование в составе JDK. JMX.
3. MBeans. Основные понятия. Архитектура фреймворка.
4. Утилита JConsole. Возможности, область применения.
5. Утилита VisualVM. Возможности, области применения.
6. Удаленный мониторинг и профилирование приложений на платформе Java

Выполнение

Исходный код MBean-классов

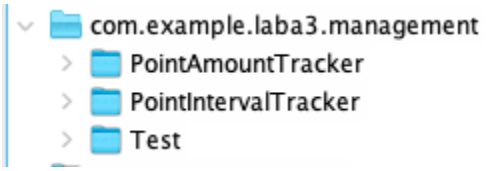
```
public interface PointIntervalTrackerMBean {
   String getInterval();
   void click();
}
```

```
public class PointIntervalTracker implements PointIntervalTrackerMBean {

   private static long DAY = 86400000L;
   private static long HOUR = 3600000L;
   private static long MIN = 60000L;
   private static long SEC = 1000L;

   private long firstClick = 0L;
   private int amount;
   private long interval = 0L;

   @Override
   public String getInterval() {
      long days = interval / DAY;
      long hours = (interval - days * DAY) / HOUR;
      long minutes = (interval - days * DAY - hours * HOUR) / MIN;
      long seconds = (interval - days * DAY - hours * HOUR - minutes * MIN) / SEC;
      String interval = Arrays.asList(
            days != 0L ? days + " days" : null,
            hours != 0L ? hours + " hours" : null,
            minutes != 0L ? minutes + " minutes" : null,
            seconds != 0L ? seconds + " seconds" : null).stream()
               .filter(item -> item != null)
               .collect(Collectors.joining(" "));
      return interval;
   }

   @Override
   public void click() {
      amount++;
      if (firstClick == 0L) {
         firstClick = System.currentTimeMillis();
      } else {
         interval = (System.currentTimeMillis() - firstClick)
               / (amount > 1 ? amount - 1 : 1);
      }
   }
}
```

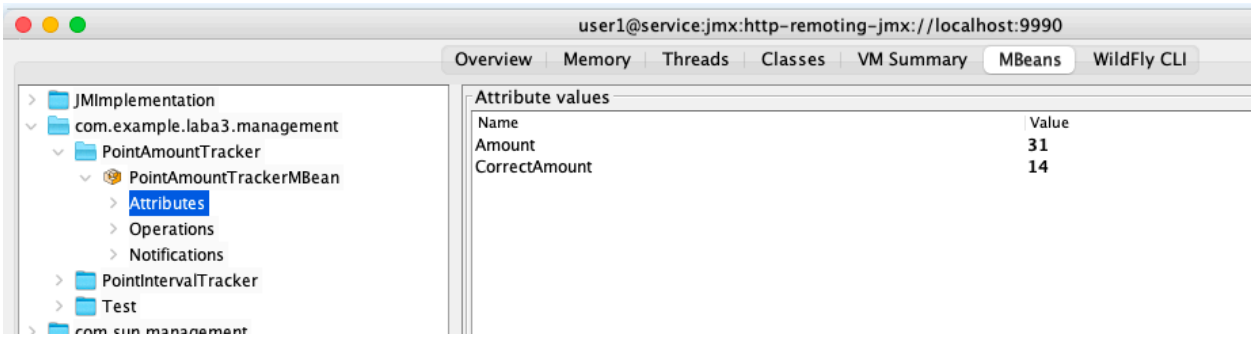
```
public interface PointAmountTrackerMBean {
   int getAmount();
   int getCorrectAmount();
   void click(boolean result);
}
```

```
public class PointAmountTracker extends NotificationBroadcasterSupport implements
PointAmountTrackerMBean {

   private long sequenceNumber = 1L;
   private int amount;
   private int correctAmount;
   private int incorrectCounter;

   public PointAmountTracker() {
       addNotificationListener(new NotificationListener() {
           @Override
           public void handleNotification(Notification notification, Object handback)
{
               System.out.println("*** Handling new notification ***");
               System.out.println("Message: " + notification.getMessage());
               System.out.println("Seq: " + notification.getSequenceNumber());
               System.out.println("*********************************");
           }
       }, null, null);
   }

   @Override
   public int getAmount() {
      return amount;
   }

   @Override
   public int getCorrectAmount() {
      return correctAmount;
   }

   @Override
   public void click(boolean result) {
      amount++;
      if (result) {
         correctAmount++;
         incorrectCounter = 0;
      } else if (++incorrectCounter == 3) {
           Notification n = new AttributeChangeNotification(this, sequenceNumber++,
               System.currentTimeMillis(), "Three incorrect clicks",
               "incorrectCounter", "int",
               2, 3);
           sendNotification(n);
           incorrectCounter = 0;
      }
   }
}
```

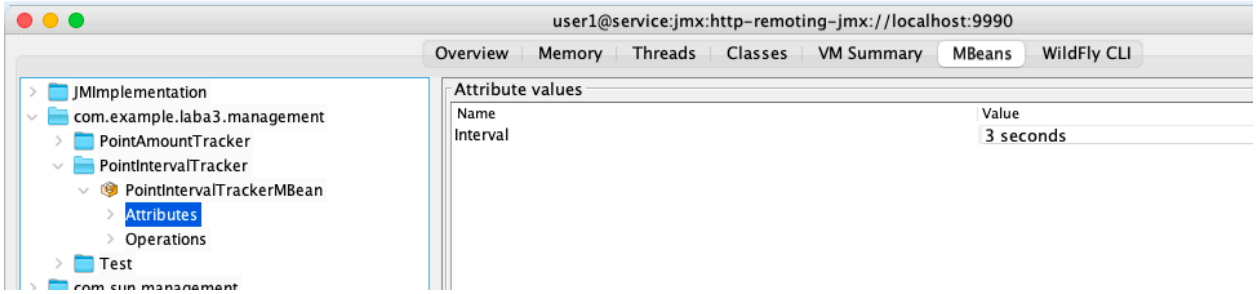
Скриншоты JConsole



Данная картинка говорит о том, что наши MBeans отображаются.



MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если пользователь совершил 3 "промаха" подряд, разработанный MBean должен отправлять оповещение об этом событии.



MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.

Скриншоты VisualVM с показаниями

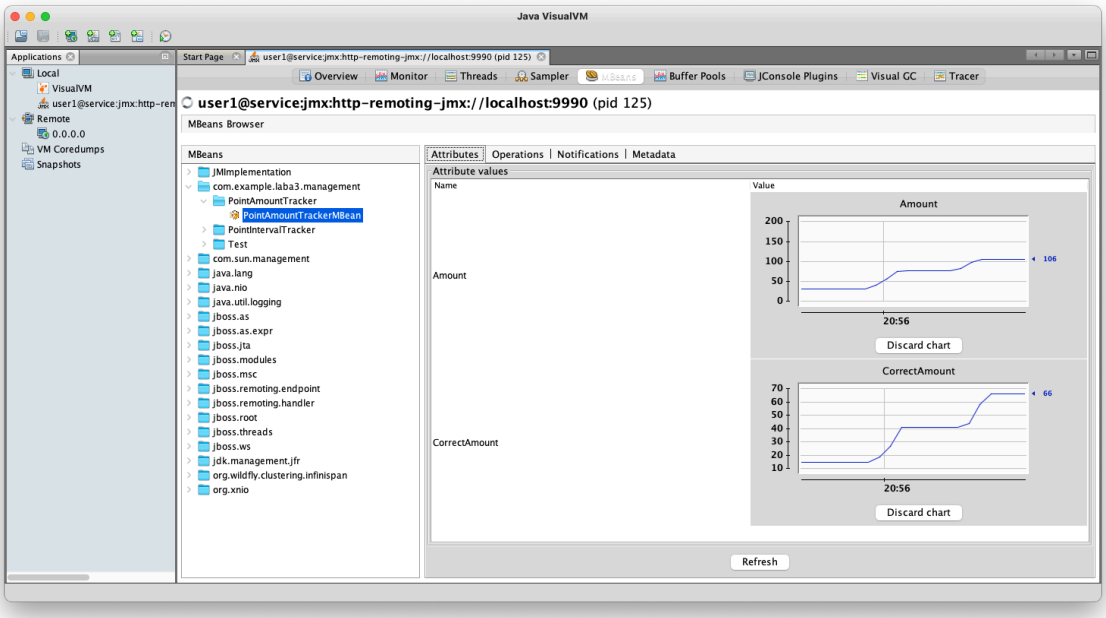
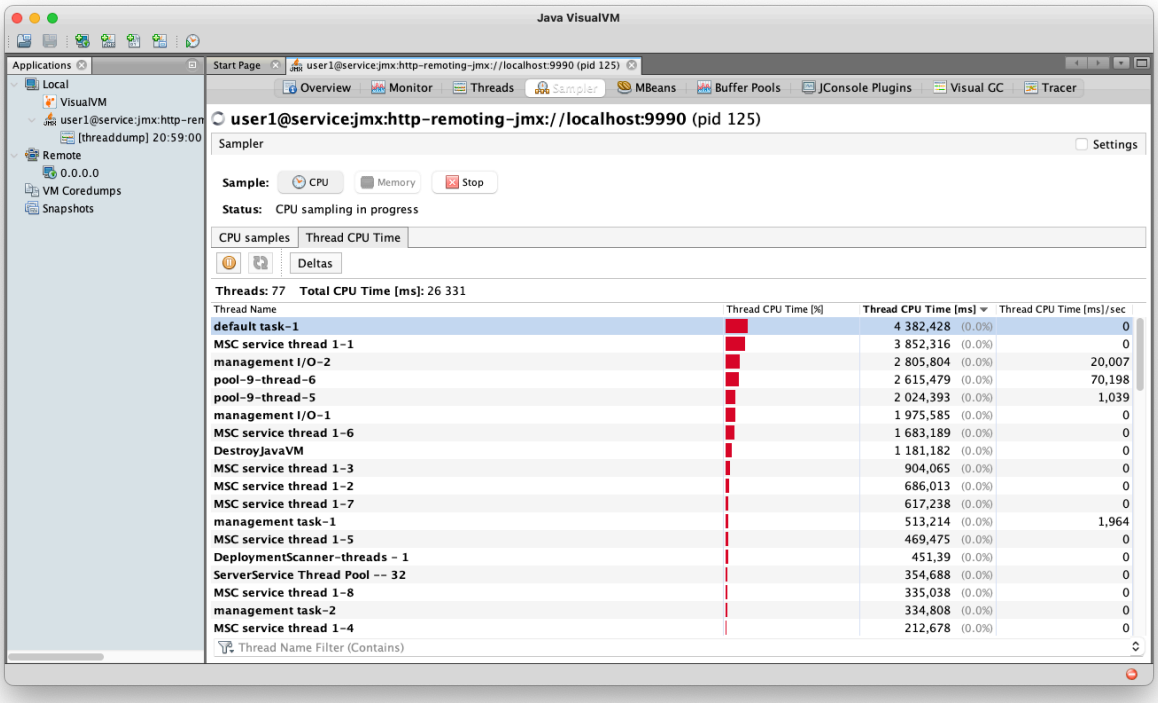


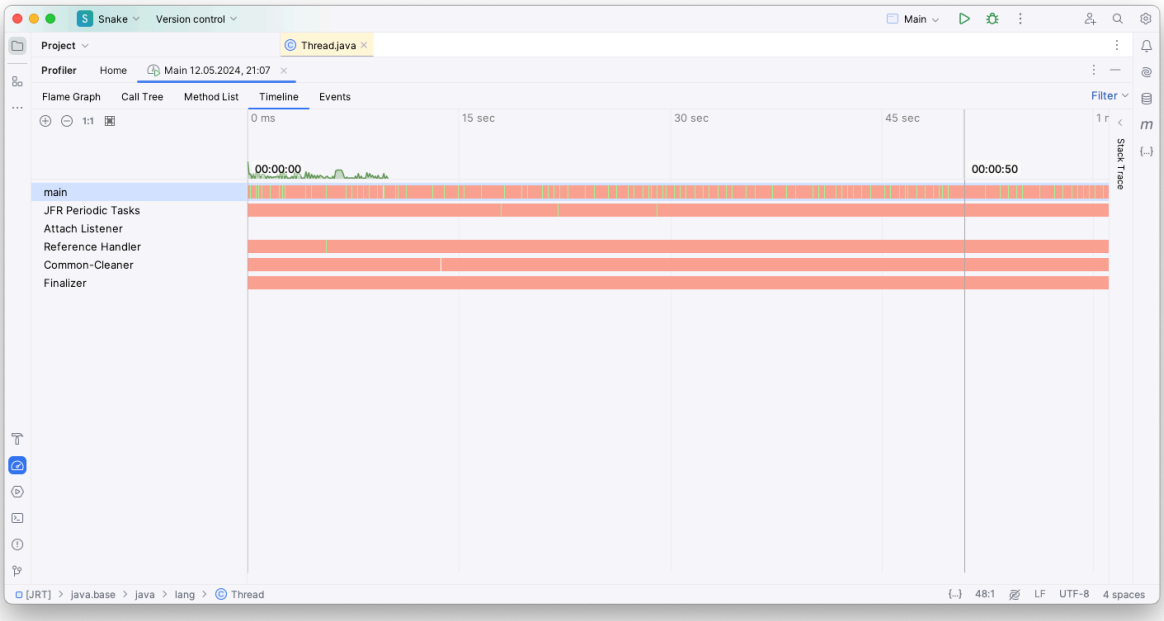
График показаний MBean, считавшего общее число точек, установленных пользователем, с течением времени



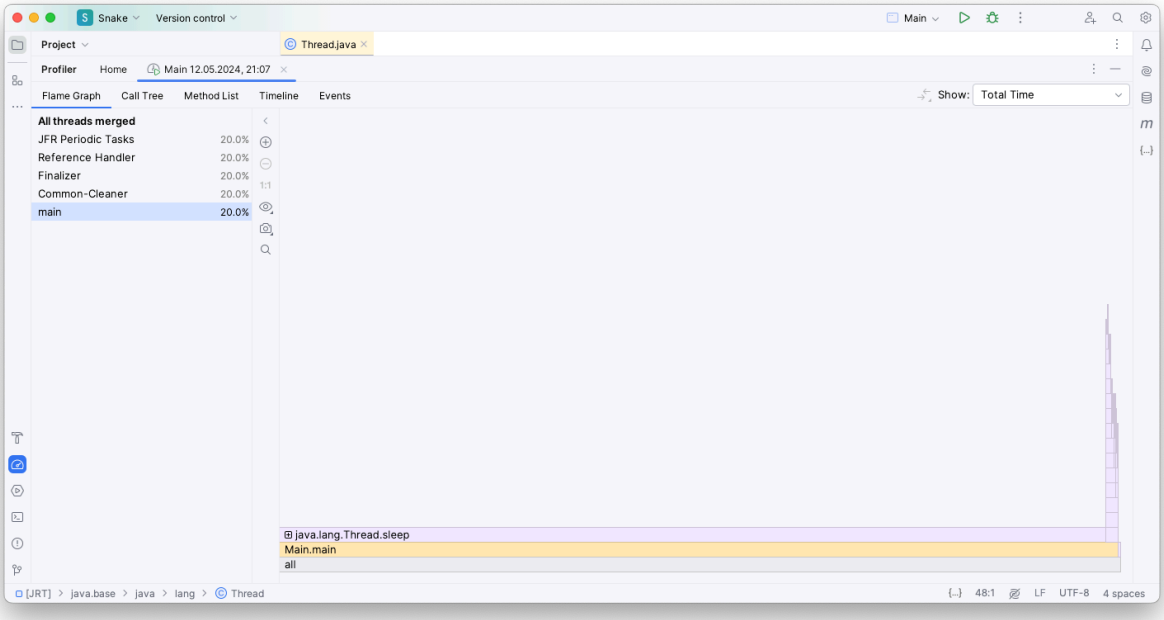
Имя потока, который потребляет наибольший процент CPU: `default task-1`

Поиск утечки памяти с помощью утилиты VisualVM в IDE

Устранение проблемы с производительностью



Запуск программы через профилировщик Intellij (видим, что main процесс большую часть времени находится в состоянии WAITING)



Нашли участок в коде, где вызывается `Thread.sleep`

fix: speeding up the program

Michael-Bill d18fc2f +0 -1

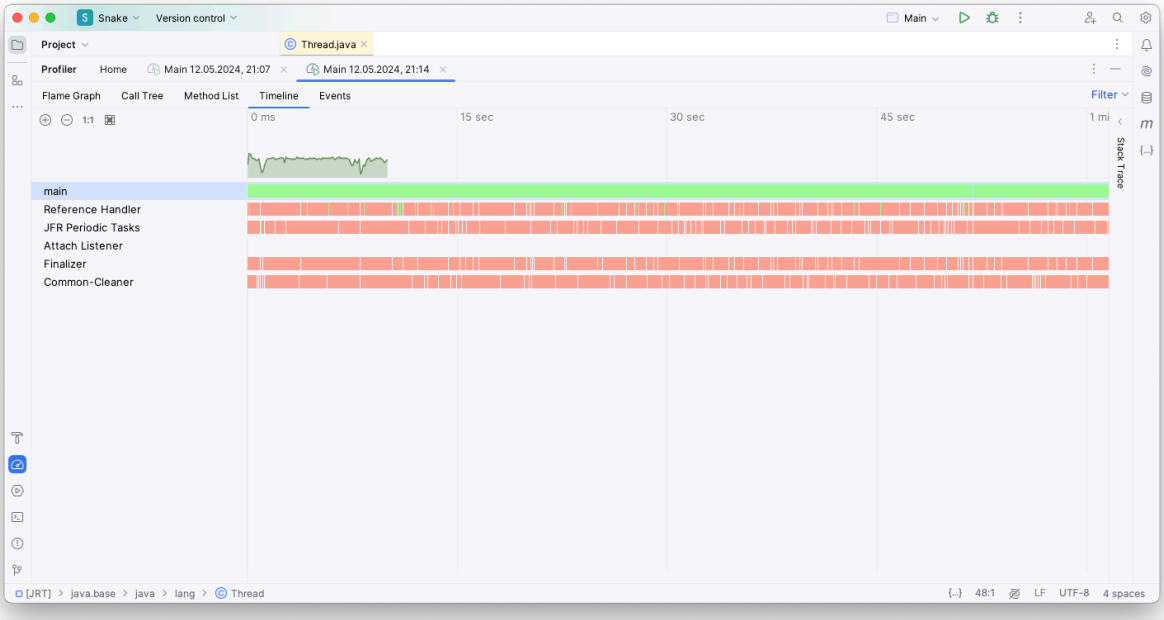
1 changed file

HttpUnit/src/Main.java

HttpUnit/src/Main.java

```
@@ -45,7 +45,6 @@ public class Main {
45  45              while (true) {
46  46                  WebResponse response = sc.getResponse(request);
47  47                  System.out.println("Count: " + number++ + response);
48      -               java.lang.Thread.sleep(200);
49  48              }
50  49          } catch (InterruptedException ex) {
51  50              Logger.getLogger("global").log(Level.SEVERE, null, ex);
```

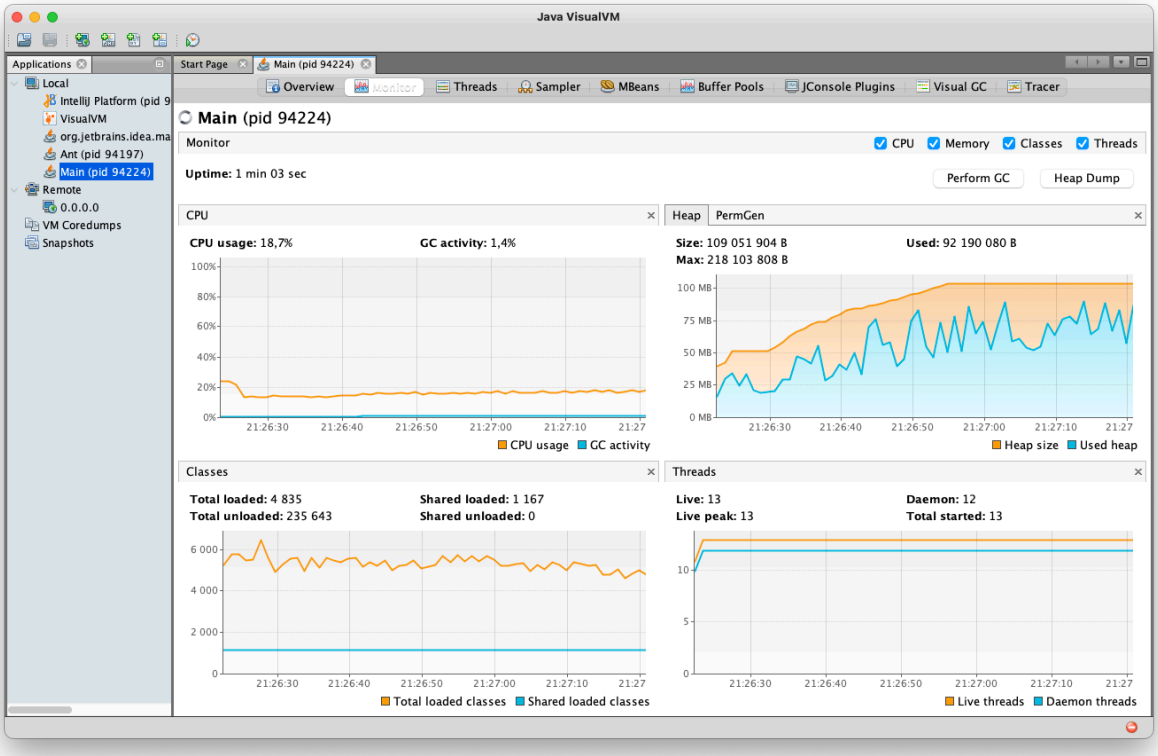
Удаляем замедляющий работу программы `Thread.sleep`



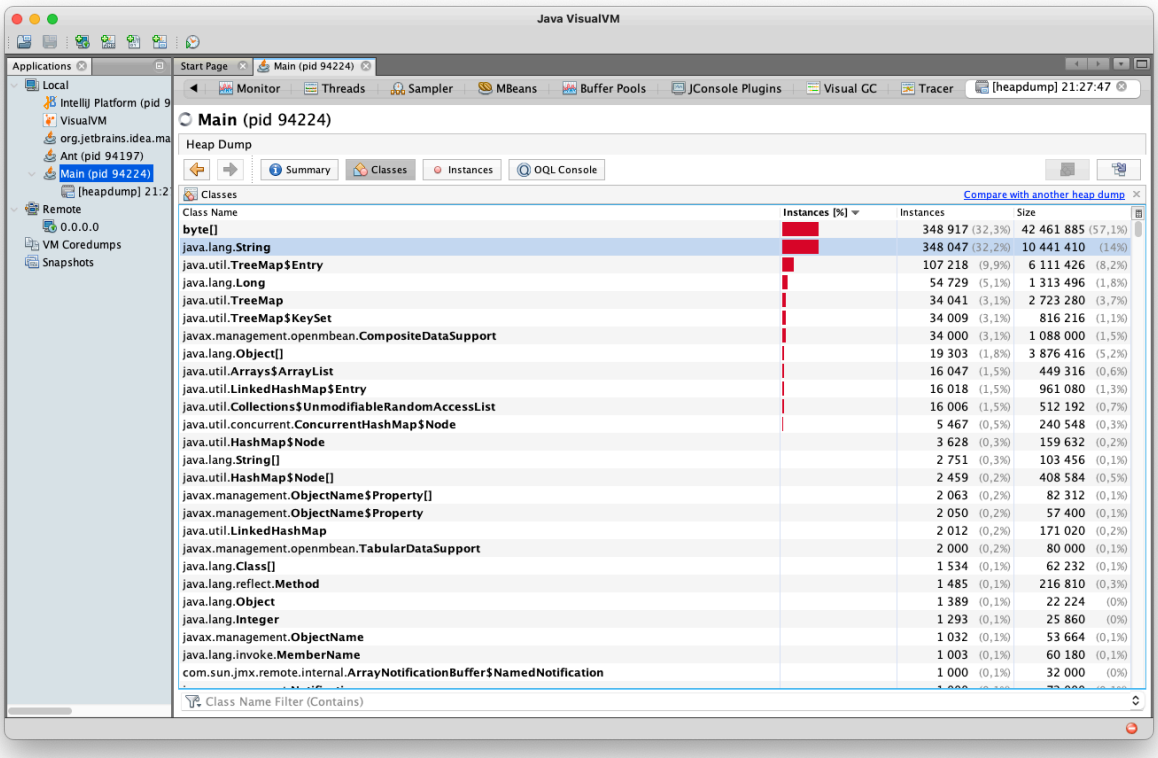
Теперь видим, как всё работает супер (main процесс занят большую часть времени)

Устранение утечки памяти

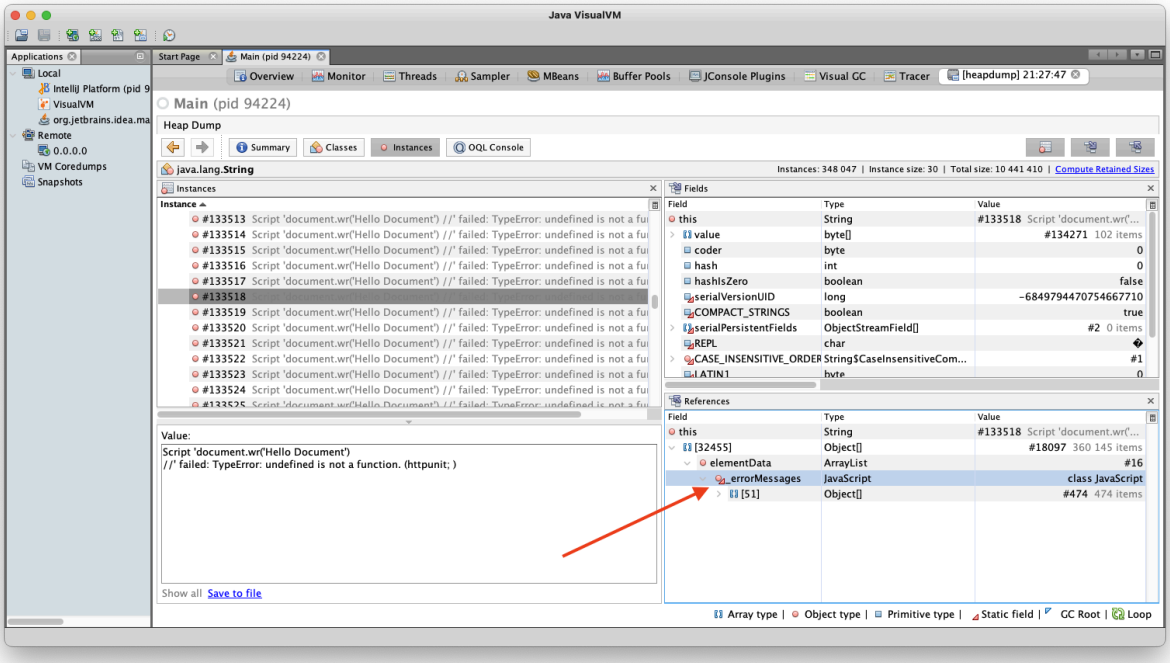
Для начала установим ограничение максимального размера кучи на 100 Мб с помощью флага -Xmx100m



видим, что память растёт вверх (график Heap Size)



Сделали Heap Dump и видим, что стрингов уж очень много



Во вкладке Instances нашли массив объектов, который содержит огромное количество элементов и класс, в котором он создан

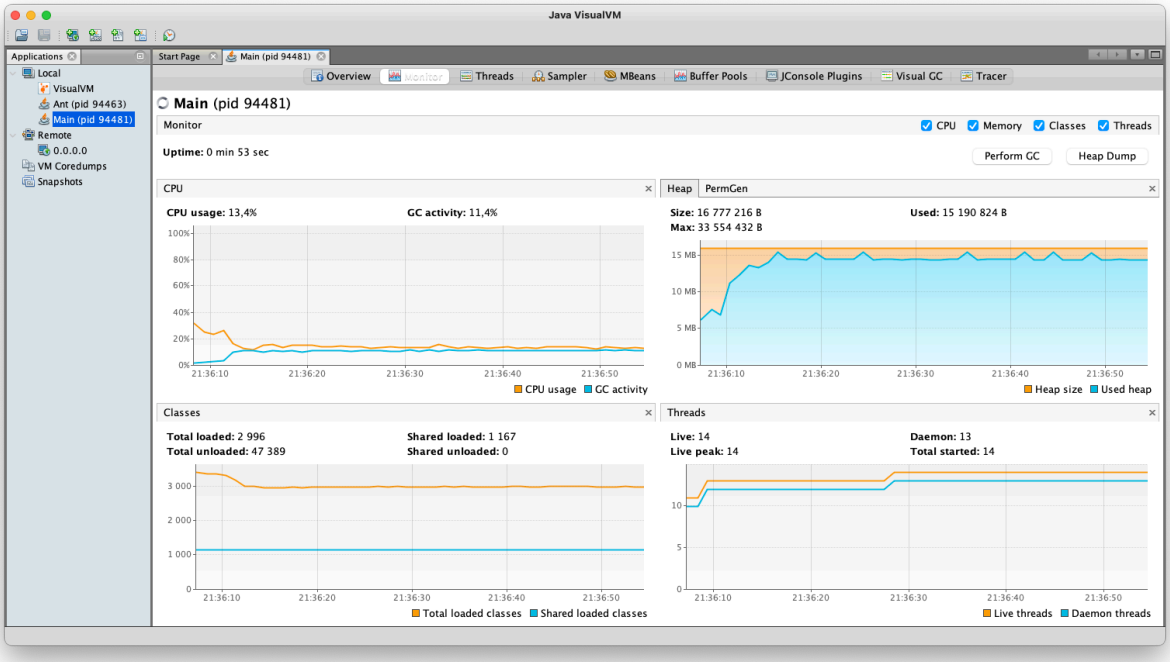
```
private void handleScriptException( Exception e, String badScript ) {
    final String errorMessage = badScript + " failed: " + e;
    if (!(e instanceof EcmaError) && !(e instanceof EvaluatorException)) {
        e.printStackTrace();
        throw new RuntimeException( errorMessage );
    } else if (isThrowExceptionsOnError()) {
        e.printStackTrace();
        throw new ScriptException( errorMessage );
    } else {
        _errorMessages.add( errorMessage );
    }
}
```

Открыли этот класс и в нужном методе добавили ещё одно сообщение об ошибке

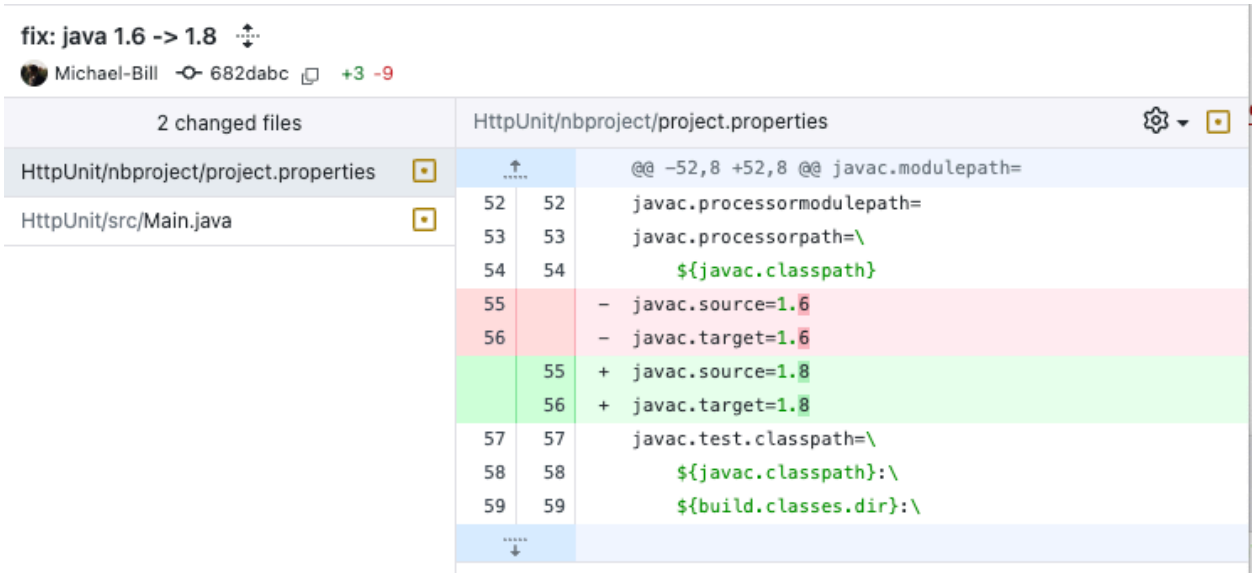
```
static void clearErrorMessages() {
    _errorMessages.clear();
}


/**
 * Clears the accumulated script error messages.
 */
public static void clearScriptErrorMessages() {
    getScriptingEngine().clearErrorMessages();
}
```

Метод `clearScriptErrorMessages` теперь будем вызывать в `main`

```
while (true) {
    WebResponse response = sc.getResponse(request);
    System.out.println("Count: " + number++ + response);
    HttpUnitOptions.clearScriptErrorMessages();
}
```



Теперь через VisualVM видим, что heap больше не растёт



В качестве бонуса обновили версию java

Вывод

В лабораторной работе мы ознакомились с MBean, как они работают, почему это круто. Также узнали о доп. возможностях языка Java, такими как JConsole. И уже было решили, что это совершенство. Но, как оказалось есть ещё и VisualVM, которое не может оставить равнодушными ни одного человека на этой планете. Но и это ещё не конец. Конечной точкой нашего путешествия по лабораторной работе оказалась возможность профилировать программу прямо через среду разработки (да да, вы не ослышались). В общем, спасибо авторам за лабораторную работу. Это был действительно интересный и редкий опыт!