

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4 по дисциплине
«Методы и средства программной инженерии»

Вариант 1004

Выполнили:

Соболев Иван Александрович,
Тюрин Святослав Вячеславович

Факультет: ПИиКТ

Группа: Р32312, Р32302

Преподаватель:

Исаев Илья Владимирович

Санкт-Петербург, 2023

Задание:

Внимание! У разных вариантов разный текст задания!

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 4 "промаха" подряд, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий площадь получившейся фигуры.

2. С помощью утилиты JConsole провести мониторинг программы:

- Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
- Определить значение переменной classpath для данной JVM.

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 профилировщика IDE NetBeans, Eclipse или Idea локализовать и устранить проблемы с производительностью в [программе](#). По результатам локализации и 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.

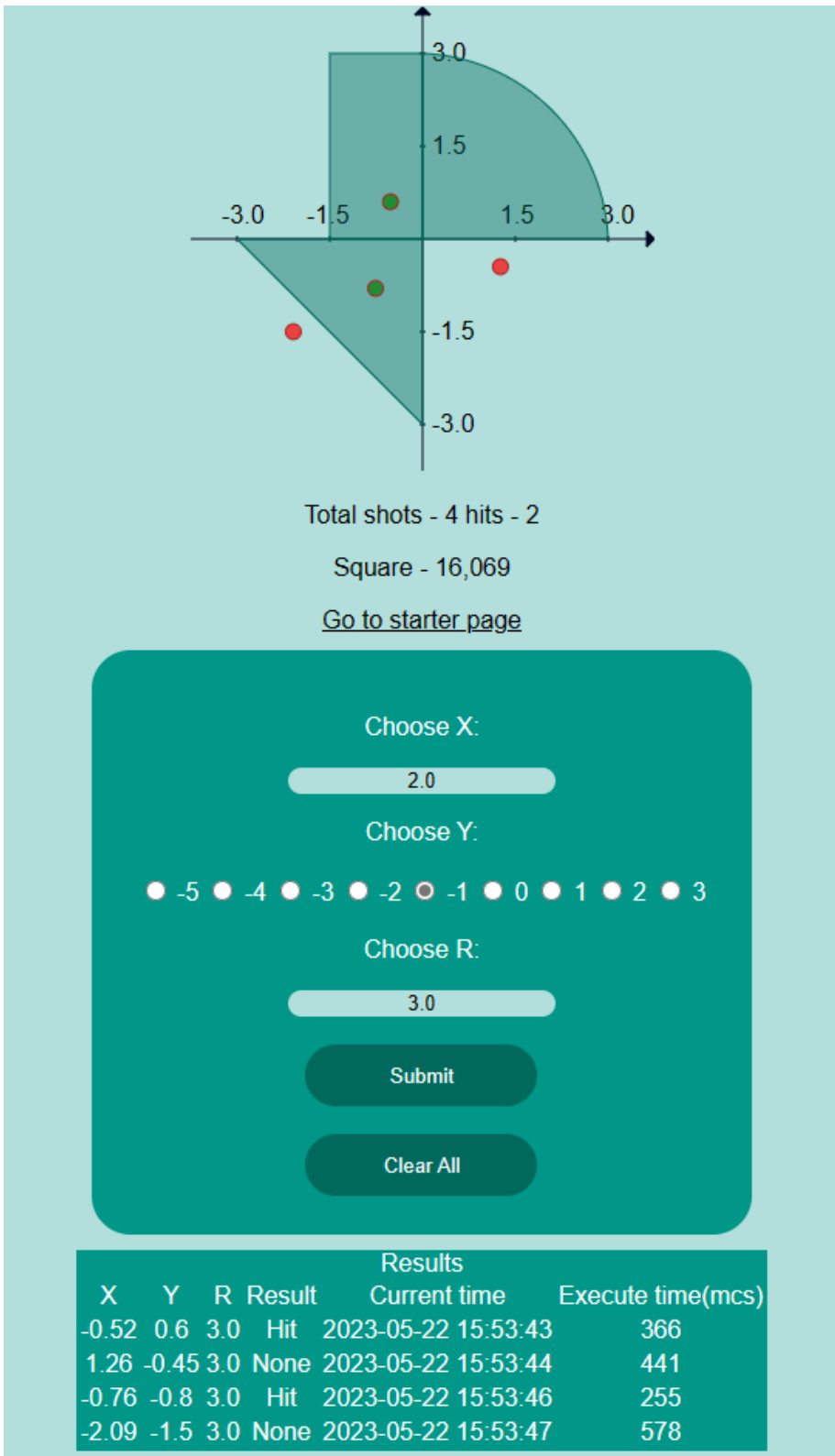
Студент должен обеспечить возможность воспроизведения процесса поиска и локализации проблемы по требованию преподавателя.

Код: [itmo-mispi/Mispi4 at main · Ivanio1/itmo-mispi \(github.com\)](#)

Выполнение:

Jconsole:

UI:



Показания JConsole:

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

[-]

 JImplementation

[-]

 com.sun.management

[-]

 java.lang

[-]

 java.nio

[-]

 java.util.logging

[-]

 jboss.as

[-]

 jboss.as.expr

[-]

 jboss.jta

[-]

 jboss.modules

[-]

 jboss.msc

[-]

 jboss.remoting.endpoint

[-]

 jboss.remoting.handler

[-]

 jboss.root

[-]

 jboss.threads

[-]

 jboss.ws

[-]

 mbeans

[-]

 CountHits

[-]

 Attributes

[-]

 Operations

[-]

 Notifications[0]

[-]

 Square

[-]

 org.wildfly.clustering.infra

[-]

 org.xnio

Attribute values

Name	Value
CountHits	2
CountShots	4

Refresh

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

[-]

 JImplementation

[-]

 com.sun.management

[-]

 java.lang

[-]

 java.nio

[-]

 java.util.logging

[-]

 jboss.as

[-]

 jboss.as.expr

[-]

 jboss.jta

[-]

 jboss.modules

[-]

 jboss.msc

[-]

 jboss.remoting.endpoint

[-]

 jboss.remoting.handler

[-]

 jboss.root

[-]

 jboss.threads

[-]

 jboss.ws

[-]

 mbeans

[-]

 CountHits

[-]

 Attributes

[-]

 Operations

[-]

 Notifications[0]

[-]

 Square

[-]

 Attributes

[-]

 Operations

[-]

 org.wildfly.clustering.infra

[-]

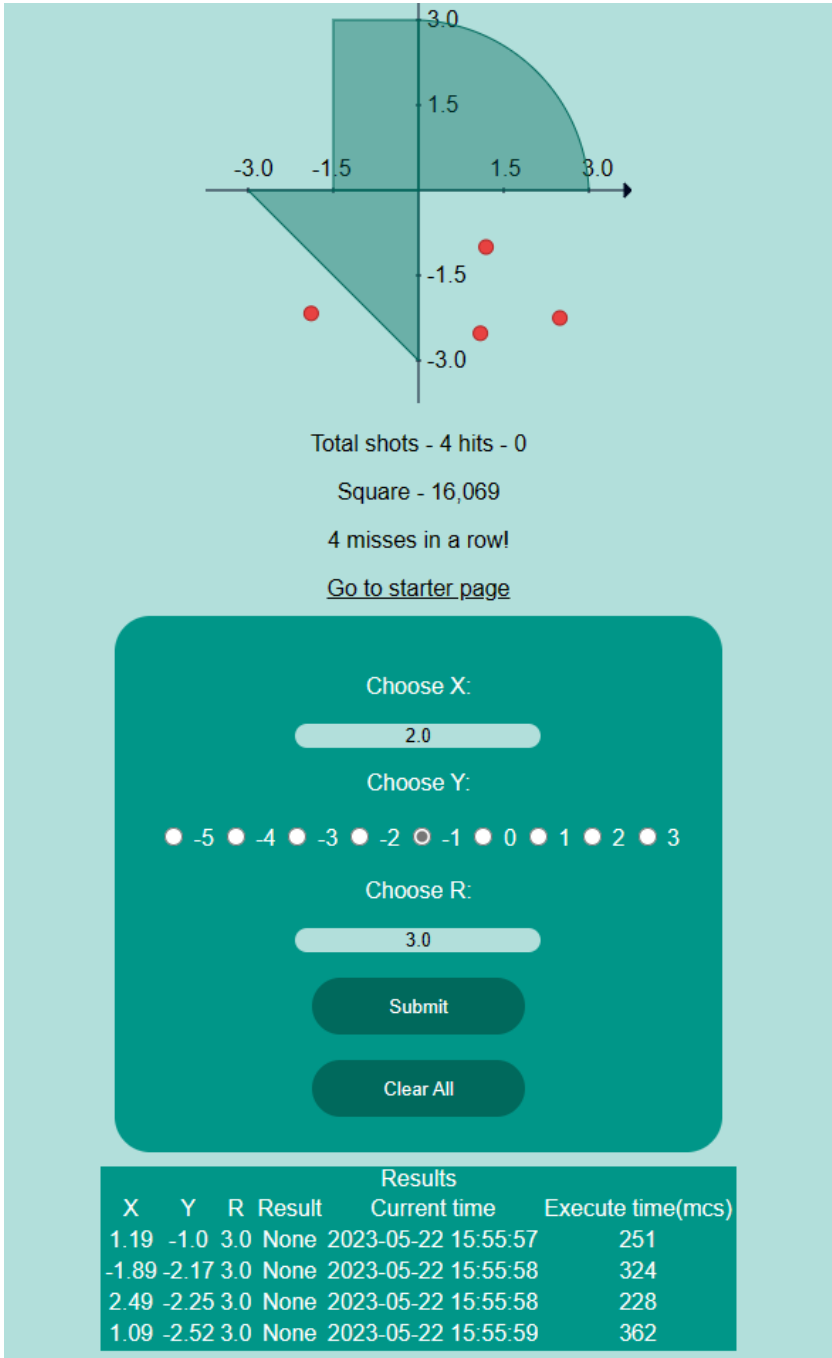
 org.xnio

Attribute values

Name	Value
Area	16.068583470577035
R	3.0

Refresh

UI:



Показания JConsole:

Attribute values	
Name	Value
CountHits	0
CountShots	4

Attribute values	
Name	Value
Area	16.068583470577035
R	3.0

Notification buffer						
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
15:55:59:855	countOfMissedPointsEqualsFour		4	Count of missed points equals 4	javax.management.Notificati...	mbeans:type=Count#its

2)

Java Monitoring & Management Console - pid: 2844 jboss-modules.jar -mp C:\Users\Asus\Downloads\wildfly-26.1.2.Final\wildfly-26.1...

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Current heap size: 206 965 kbytes
Maximum heap size: 524 288 kbytes
Garbage collector: Name = 'G1 Young Generation', Collections = 99, Total time spent = 0,534 seconds
Garbage collector: Name = 'G1 Old Generation', Collections = 0, Total time spent = 0,000 seconds

Committed memory: 307 200 kbytes
Pending finalization: 0 objects

Operating System: Windows 10 10.0
Architecture: amd64
Number of processors: 8
Committed virtual memory: 713 252 kbytes

Total physical memory: 8 246 056 kbytes
Free physical memory: 889 140 kbytes
Total swap space: 15 323 944 kbytes
Free swap space: 3 022 416 kbytes

VM arguments: -Dprogram.name=standalone.bat -Xms64M -Xmx512M -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.desktop/sun.awt=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.idap=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.url.idap=ALL-UNNAMED --add-exports=java.naming/com.sun.jndi.url.idaps=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=java.base/java.security=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.management/javax.management=ALL-UNNAMED --add-opens=java.naming/javax.naming=ALL-UNNAMED -Djava.security.manager=allow -Dorg.jboss.boot.log.file=C:\Users\Asus\Downloads\wildfly-26.1.2.Final\wildfly-26.1.2.Final\standalone\log\server.log -Dlogging.configuration=file:C:\Users\Asus\Downloads\wildfly-26.1.2.Final\wildfly-26.1.2.Final\standalone\configuration\logging.properties

Class path: C:\Users\Asus\Downloads\wildfly-26.1.2.Final\wildfly-26.1.2.Final\jboss-modules.jar

Library path: C:\Program Files\Java\jdk-17\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\CLS Client\;C:\Program Files\Intel\CLS Client\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\WINDOWS\System32\OpenSSH;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files\dotnet\;C:\Program Files\MiKTeX\miktex\bin\x64\;C:\Program Files (x86)\dotnet\;C:\TDM-GCC-32\bin;C:\Program Files\PuTTY\;C:\Program Files\Git\cmd;C:\apache-ant-1.9.16\bin;C:\Program Files\Java\jdk-17\bin;C:\Users\Asus\AppData\Local\Programs\Python\Python310\Scripts\;C:\Users\Asus\AppData\Local\Programs\Python\Python310\;C:\Users\Asus\AppData\Local\Programs\Python\Python38\Scripts\;C:\Users\Asus\AppData\Local\Programs\Python\Python38\;C:\Users\Asus\AppData\Local\Programs\Python\Launcher\;C:\Users\Asus\AppData\Local\Microsoft\WindowsApps\;C:\Users\Asus\AppData\Local\Programs\MiKTeX\miktex\bin\x64\;.

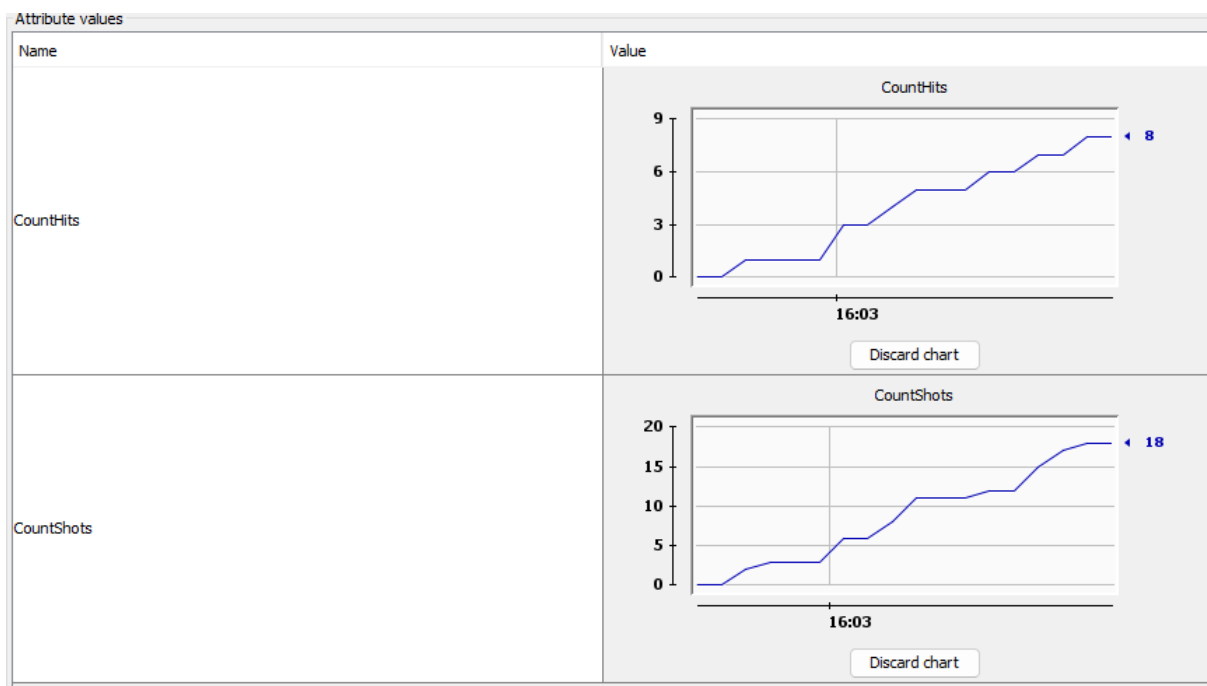
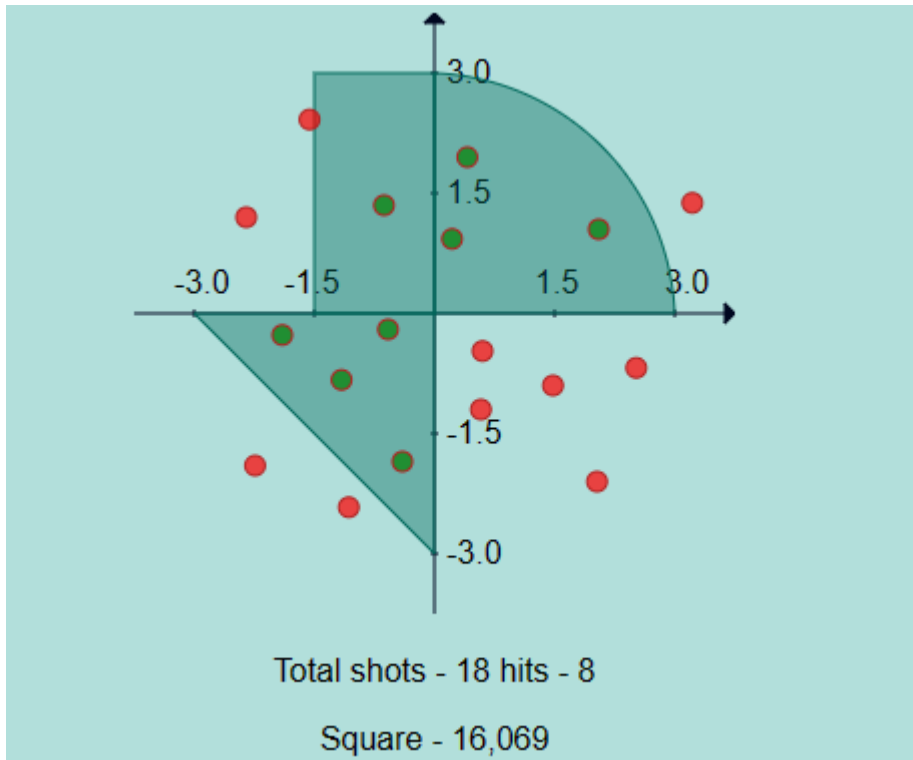
Boot class path: Unavailable

Class path: C:\Users\Asus\Downloads\wildfly-26.1.2.Final\wildfly-26.1.2.Final\jboss-modules.jar

Показания VisualVM:

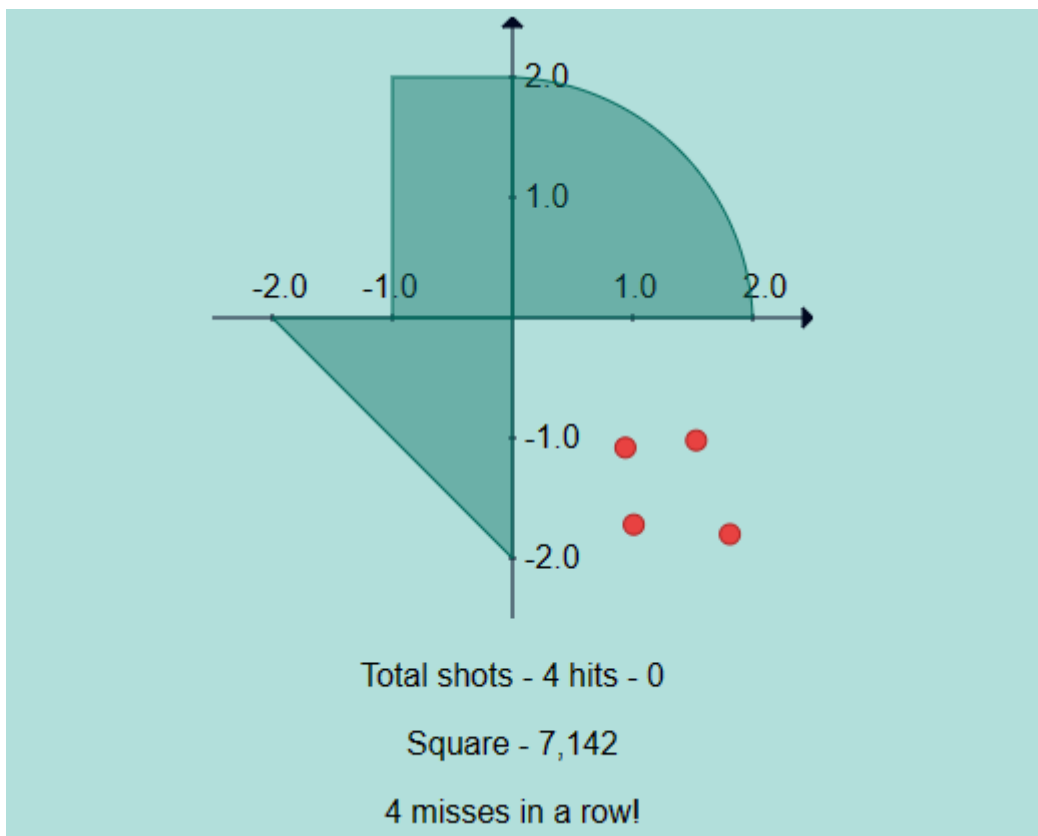
1)

UI:



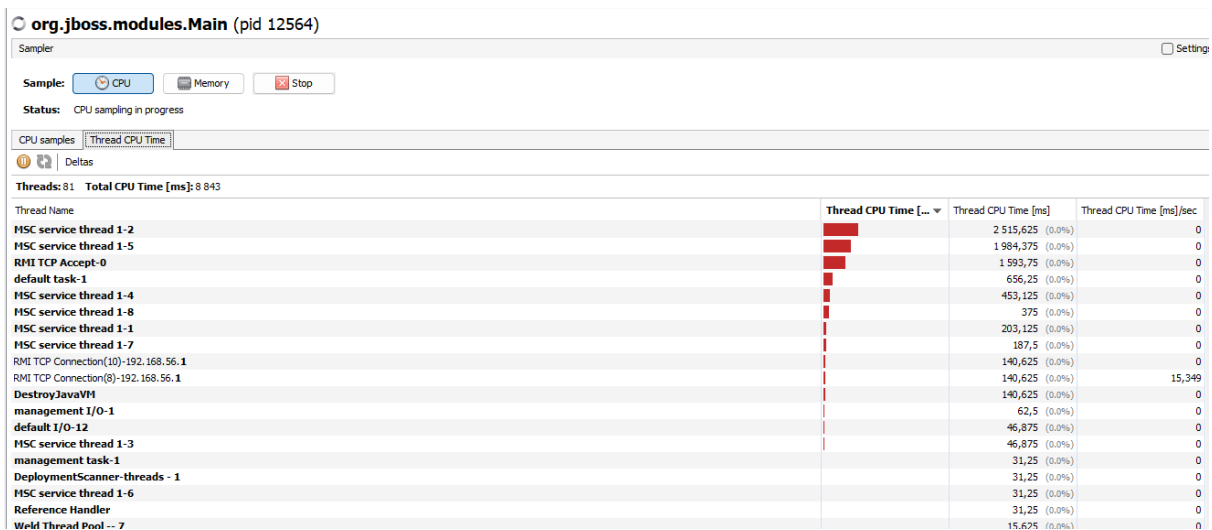


UI:



Notification buffer						
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
16:06:32:292	countOfMissedPointsE...		4	Count of missed point...	javax.management.N...	mbeans:type=CountHits

2) Имя потока, потребляющего наибольший процент времени CPU.



Наибольший процент времени CPU занимает поток **MSC service thread 1-2**. Этот поток является частью механизма управления сервисами, используемого контейнером приложений для запуска, остановки и перезапуска компонентов приложения, таких как EJB-бины или сервлеты.

Поиск утечки памяти:

Шаг №1: Подготовка окружения

Основная часть программы состоит из бесконечного цикла с запросами

```
while (true) {  
    WebResponse response = sc.getResponse(request);  
    System.out.println("Count: " + number++ + response);  
    java.lang.Thread.sleep( millis: 200 );  
}
```

Чтобы ускорить выбрасывание OutOfMemoryError:

- Установим задержку между запросами (Thread.sleep) в 0 мс
- Установим максимальный размер кучи на 30 Мб (Для этого в конфигурации запуска добавим опцию(-Xmx30M) для Vm)

Шаг №2: Анализ используемой памяти

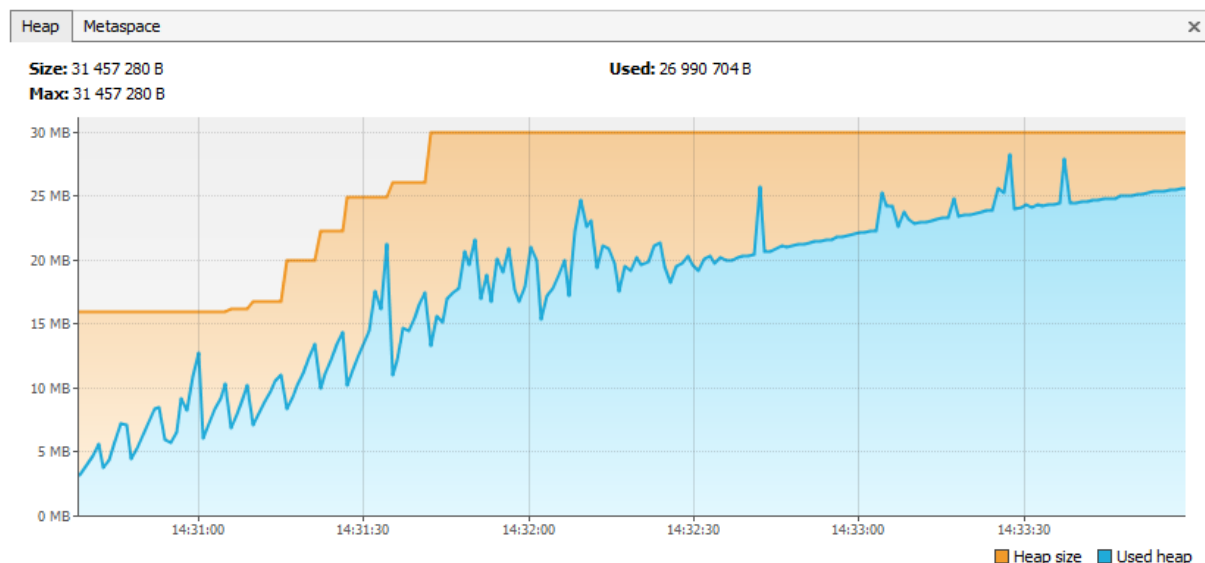
Через несколько минут после запуска программы выбрасывается исключение:

```

Count: 123766[ _response = com.meterware.servletunit.ServletUnitHttpResponse@105b6a8]
Count: 123767[ _response = com.meterware.servletunit.ServletUnitHttpResponse@e45f61]
Count: 123768[ _response = com.meterware.servletunit.ServletUnitHttpResponse@5cdd23]
Count: 123769[ _response = com.meterware.servletunit.ServletUnitHttpResponse@1765654]
Count: 123770[ _response = com.meterware.servletunit.ServletUnitHttpResponse@1ed79e8]
Exception in thread "main" java.lang.OutOfMemoryError: Create breakpoint : Java heap space
    at java.util.Hashtable.rehash(Hashtable.java:402)
    at java.util.Hashtable.addEntry(Hashtable.java:426)
    at java.util.Hashtable.put(Hashtable.java:477)
    at java.util.Properties.load0(Properties.java:392)
    at java.util.Properties.load(Properties.java:341)
    at java.util.PropertyResourceBundle.<init>(PropertyResourceBundle.java:138)
    at java.util.ResourceBundle$Control.newBundle(ResourceBundle.java:2704)
    at java.util.ResourceBundle.loadBundle(ResourceBundle.java:1518)
    at java.util.ResourceBundle.findBundle(ResourceBundle.java:1482)
    at java.util.ResourceBundle.findBundle(ResourceBundle.java:1436)
    at java.util.ResourceBundle.findBundle(ResourceBundle.java:1436)
    at java.util.ResourceBundle.getBundleImpl(ResourceBundle.java:1370)
    at java.util.ResourceBundle.getBundle(ResourceBundle.java:854)
    at org.mozilla.javascript.Context.getMessage(Context.java:1945)
    at org.mozilla.javascript.Context.getMessage(Context.java:1698)

```

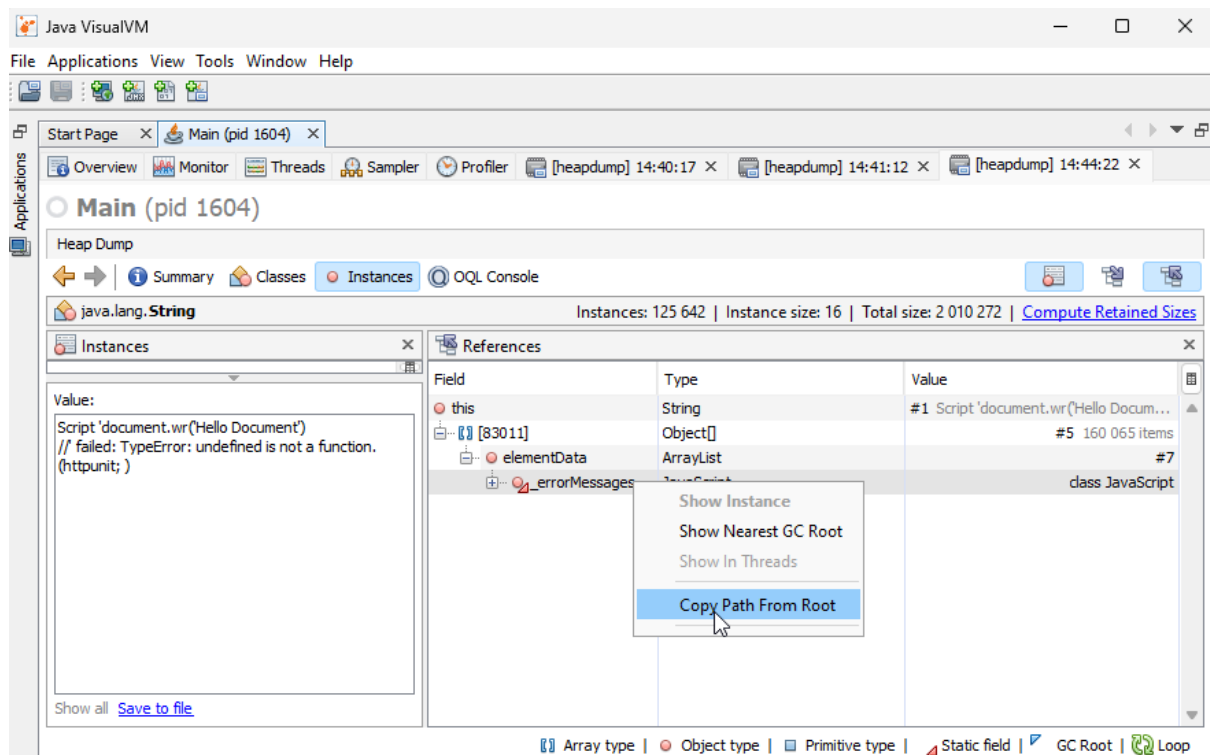
По графику использования памяти видно, что размер кучи постоянно увеличивается:



Следовательно, можно сделать вывод, что утечка памяти действительно существует.

Шаг №3: Локализация проблемы

Анализируя Heap dump находим объекты, которые занимают много памяти:



Получаем

this - value: java.lang.String #1

<- [83011] - class: java.lang.Object[], value: java.lang.String #1

<- elementData - class: java.util.ArrayList, value: java.lang.Object[] #5

<- _errorMessages - class: com.meterware.httpunit.javascript.JavaScript, value: java.util.ArrayList #7

Искомый класс - com.meterware.httpunit.javascript.JavaScript, посмотрим, что в нем находится.

```
private static ArrayList _errorMessages = new ArrayList();
```

Добавление элементов в данный список происходит только в одном методе:

```

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 );
    }
}
}

```

Таким образом, элементы накапливаются в списке при этом нигде не очищаются, что приводит к утечке памяти в связи с отсутствием очистки.

Шаг №4: Устранение проблемы

Попробуем найти в этом классе метод для очистки списка:

```

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

```

Данный метод используется в классе
com.meterware.httpunit.javascript.JavaScriptEngineFactory:

```

public void clearErrorMessages() {
    JavaScript.clearErrorMessages();
}

```

Финальный метод для очистки списка находится в классе
com.meterware.httpunit.HttpUnitOptions:

```

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

```

Добавим очистку списка после выполнения каждого запроса в главном цикле программы:

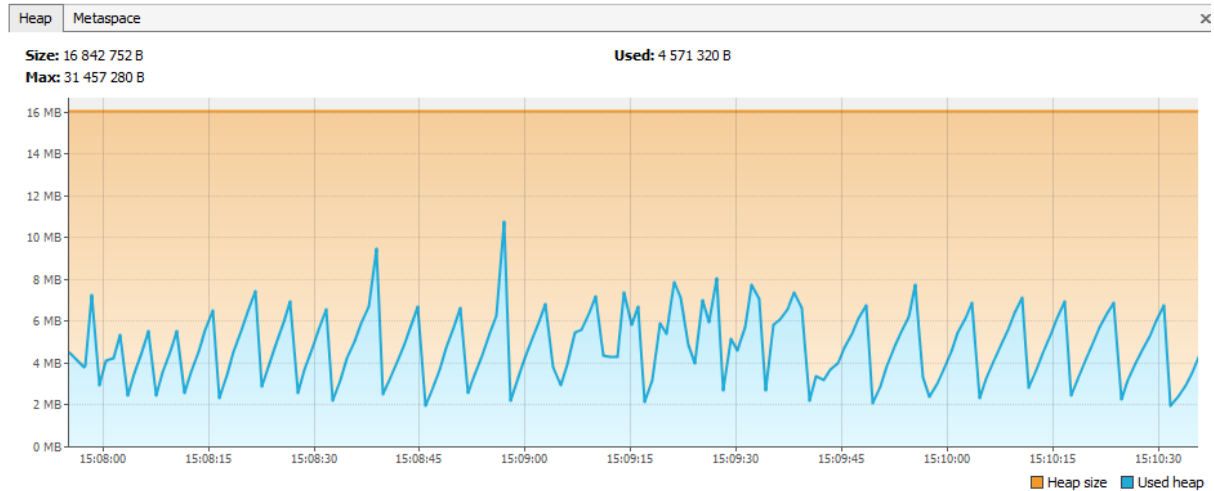
```

while (true) {
    WebResponse response = sc.getResponse(request);
    System.out.println("Count: " + number++ + response);
    java.lang.Thread.sleep( millis: 0);
    HttpUnitOptions.clearScriptErrorMessages();
}

```

Шаг №5: Проверка устранения утечки памяти

После запуска программы наблюдаем, что теперь не расходуется больше 30 Мб кучи, размер кучи постоянно не растет и сборщик мусора работает в нормальном режиме:



Следовательно, утечка памяти была успешно устранена.

Вывод: Во время выполнения лабораторной работы мы изучили утилиты JConsole и VisualVM для мониторинга и профилирования Java-приложений. Это помогло нам разобраться в деталях работы JVM и научиться определять, какие компоненты приложения влияют на его производительность. Благодаря этому, мы смогли успешно локализовать и устранить проблемы, связанные с производительностью, на основе собранных данных и анализа.