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**»

Выполнили:

Фирстова Екатерина Витальевна

Абакумова Ирина Андреевна

Факультет: ПИиКТ

Группа: P32131

Преподаватель:

Бострикова Дарья Константинова

Санкт-Петербург, 2023

1.Реализация MBean:

1) MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область:

```
package mbeans;

import model.Point;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.management.*;
import java.lang.management.ManagementFactory;

@ManagedBean(name = "counterBean", eager = true)
@SessionScoped
public class CounterBean extends NotificationBroadcasterSupport implements
CounterBeanMBean {
        private int count = 0;
        private int gotAreaCount = 0;

        public CounterBean() {
        try {
        ObjectName objectName = new
ObjectName("space.arlet.mbeans:name=counterBean");
        MBeanServer server = ManagementFactory.getPlatformMBeanServer();
        server.registerMBean(this, objectName);
        } catch (MalformedObjectNameException |
NotCompliantMBeanException | InstanceAlreadyExistsException |
                MBeanRegistrationException e) {
        throw new RuntimeException(e);
        }
        }

        public void update(Point point) {
        count++;
        if (point.isStatus()) {
        gotAreaCount++;
    }
```

```
        if (gotAreaCount % 5 == 0) {
        sendNotification(
                new Notification("information", this, 1,
                        System.currentTimeMillis(), "5 points got the area")
        );
        }
        }

        public int getCount() {
        return count;
        }

        public int getGotAreaCount() {
        return gotAreaCount;
        }
}
```

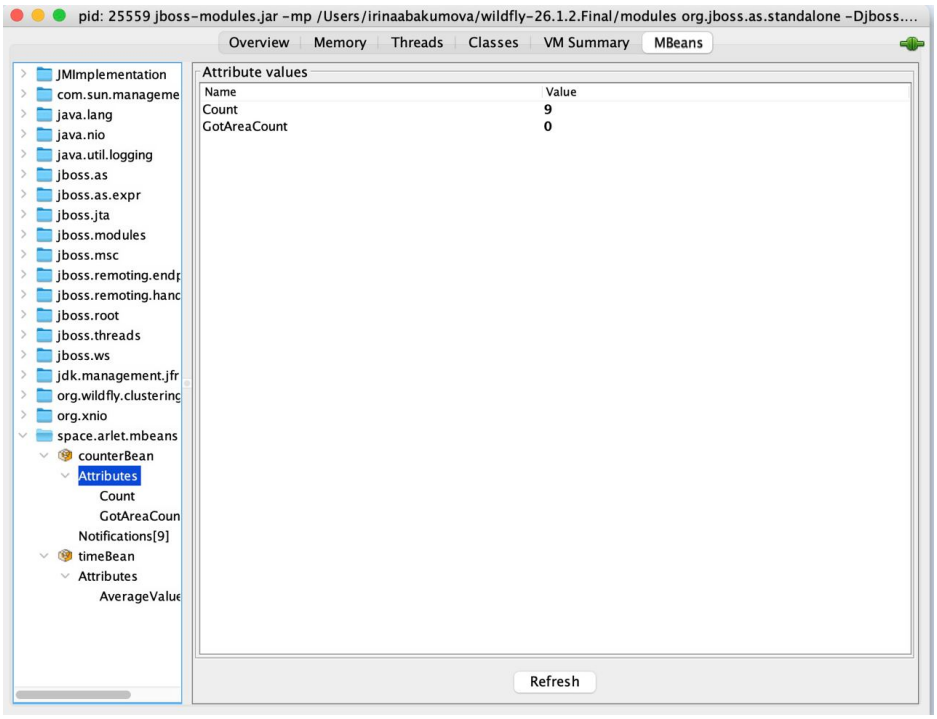
2) MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.

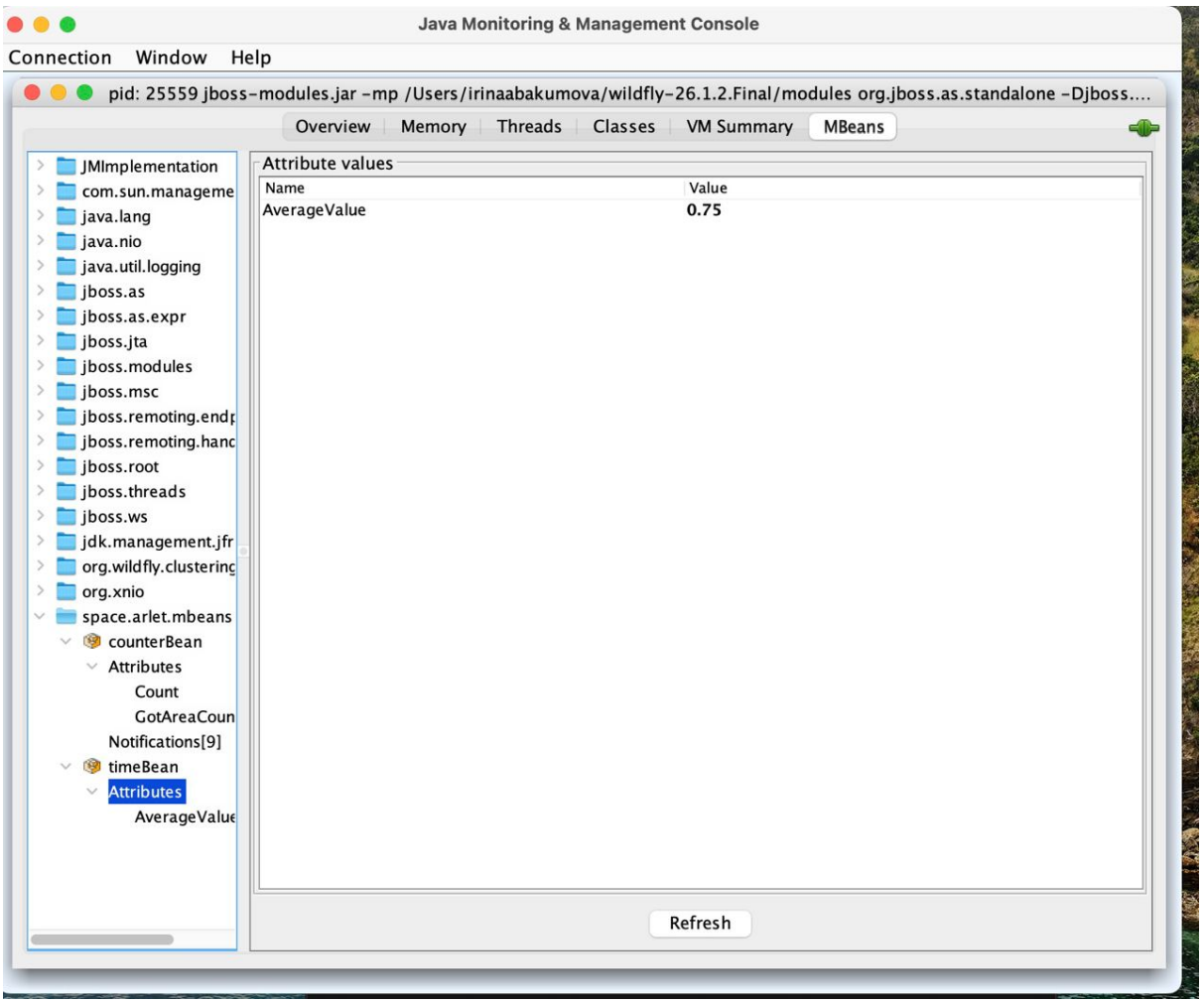
```
package mbeans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.management.*;
import java.lang.management.ManagementFactory;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;


@ManagedBean(name = "timeBean", eager = true)
@SessionScoped
public class AverageClickPerTimeBean implements
AverageClickPerTimeBeanMBean{
        private List<LocalTime> clicksTimeList = new ArrayList<>();

        private Double averageValue = 0.0;

        public AverageClickPerTimeBean() {
```

```
        try {
        ObjectName objectName = new
ObjectName("space.arlet.mbeans:name=timeBean");
        MBeanServer server =
ManagementFactory.getPlatformMBeanServer();
        server.registerMBean(this, objectName);
        } catch (MalformedObjectNameException |
NotCompliantMBeanException | InstanceAlreadyExistsException |
                MBeanRegistrationException e) {
        throw new RuntimeException(e);
        }
        }

        public void update() {

        clicksTimeList.add(LocalTime.now());
        long sum = 0;
        for (int i = 1; i < clicksTimeList.size(); i++) {
        long timeClick = clicksTimeList.get(i).toSecondOfDay();
        long previousTimeClick = clicksTimeList.get(i -
1).toSecondOfDay();
        sum += timeClick - previousTimeClick;
        }
        averageValue = sum / (clicksTimeList.size() * 1.0 - 1);
        }

        public List<LocalTime> getClicksTimeList() {
        return clicksTimeList;
        }

        public Double getAverageValue() {
        return averageValue;
        }
}
```

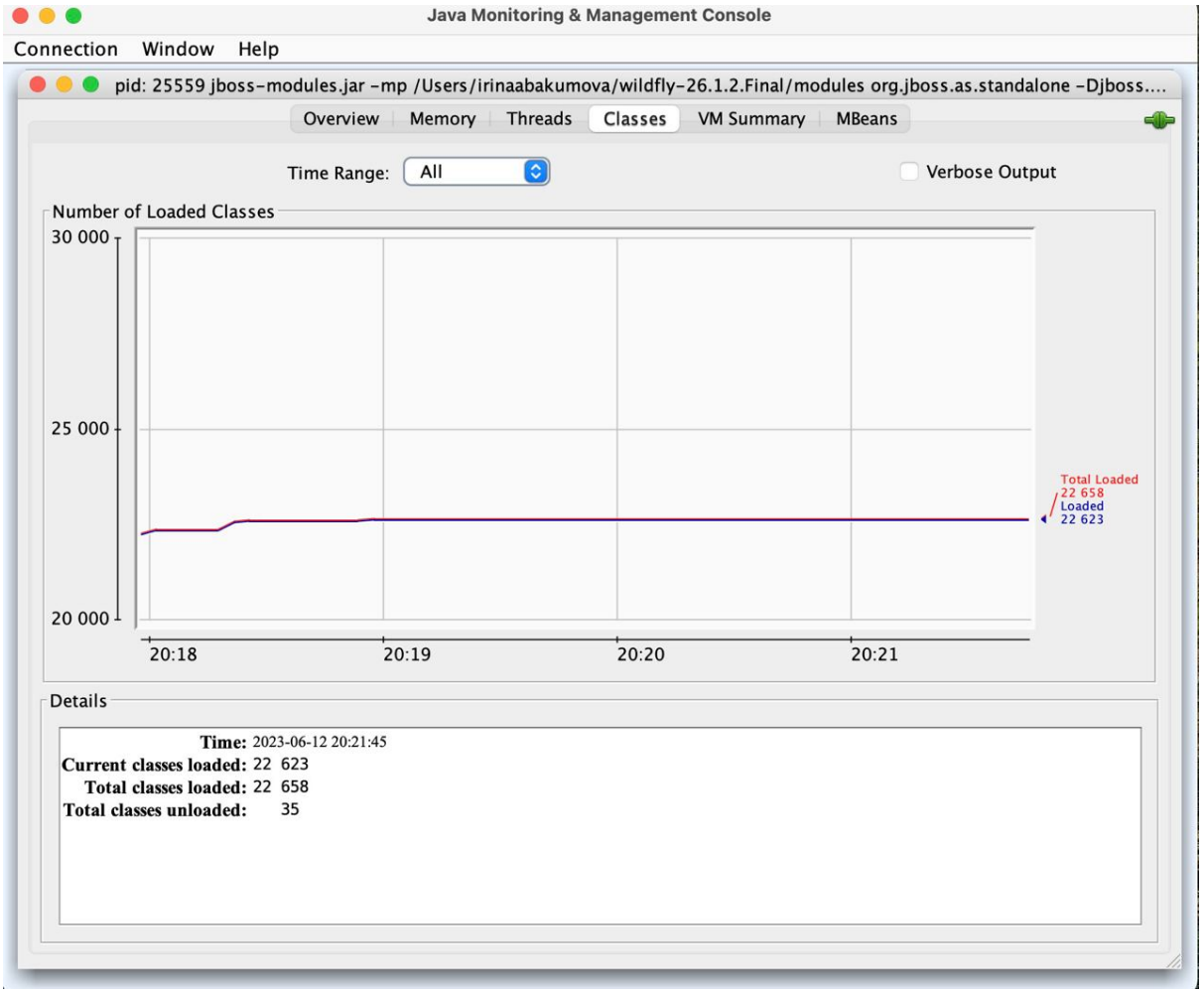
2. Мониторинг программы с помощью утилиты JConsole

показания MBean-классов:

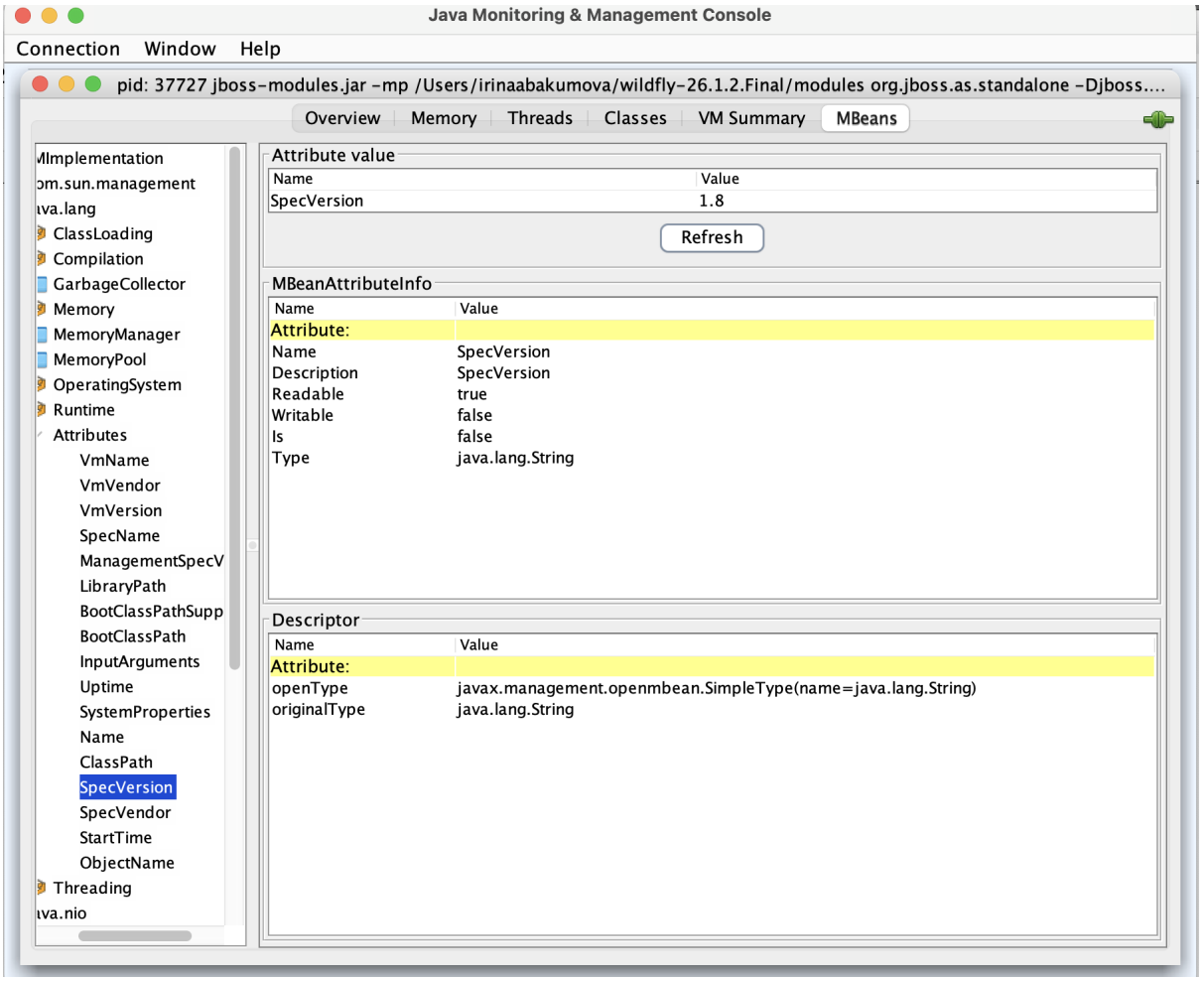




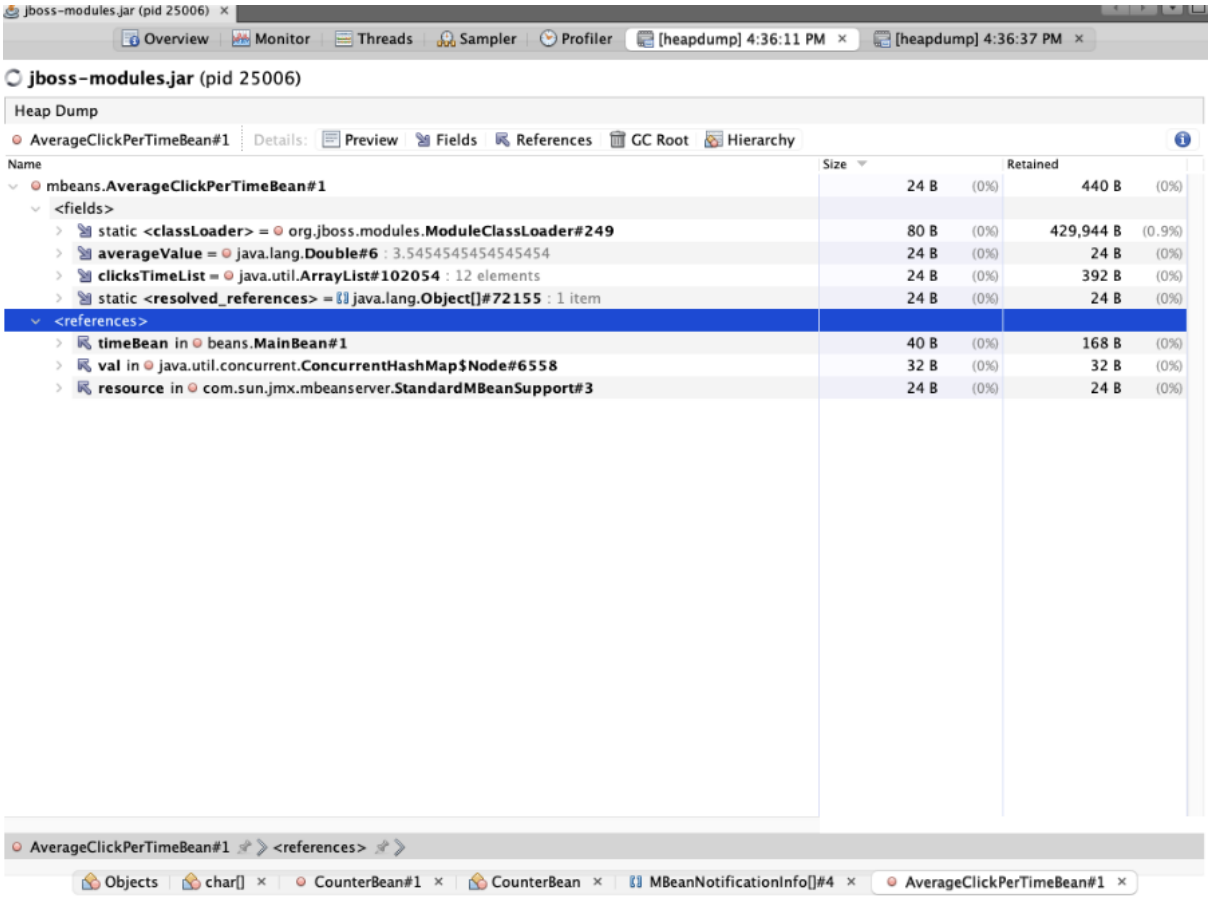
Количество загруженных классов:

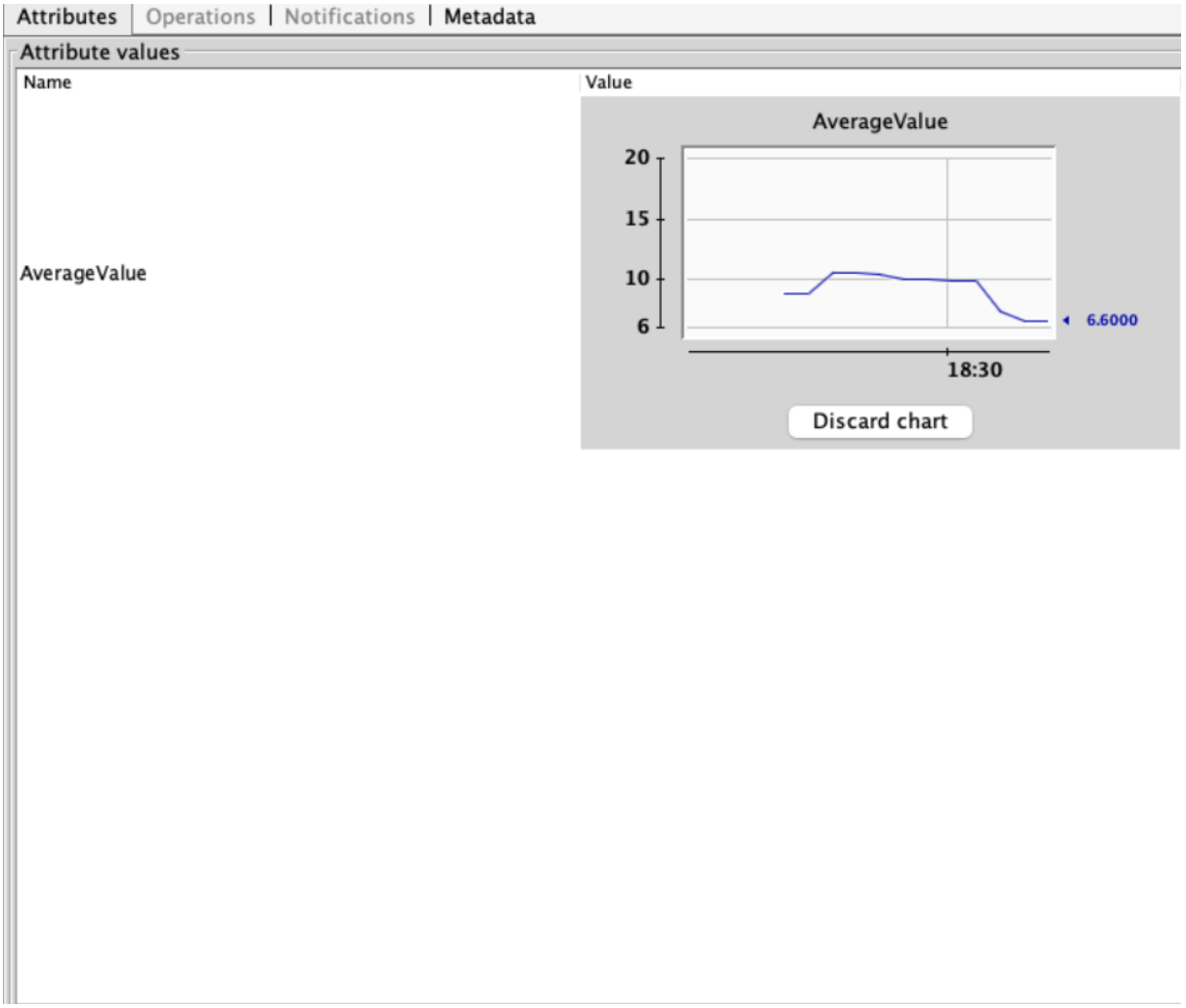

Java Monitoring & Management Console
Connection
Window
Help
pid: 25559 jboss-modules.jar -mp /Users/irinaabakumova/wildfly-26.1.2.Final/modules org.jboss.as.standalone -Djboss....
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range:
All
Verbose Output
Number of Loaded Classes
30 000
25 000
20 000
20:18
20:19
20:20
20:21
Total Loaded
22 658
Loaded
22 623
Details
Time: 2023-06-12 20:21:45
Current classes loaded: 22 623
Total classes loaded: 22 658
Total classes unloaded: 35

версию Java Language Specification, реализуемую данной средой исполнения

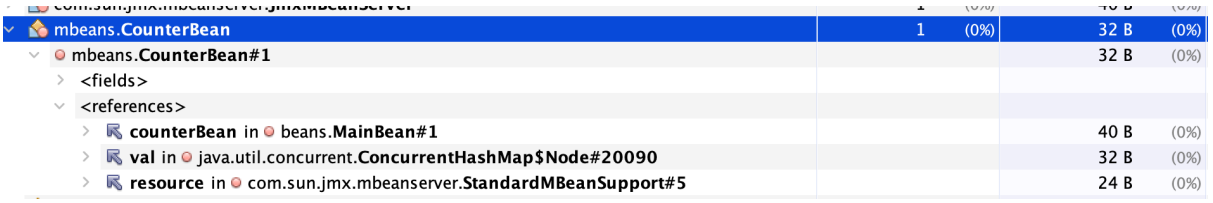


3)

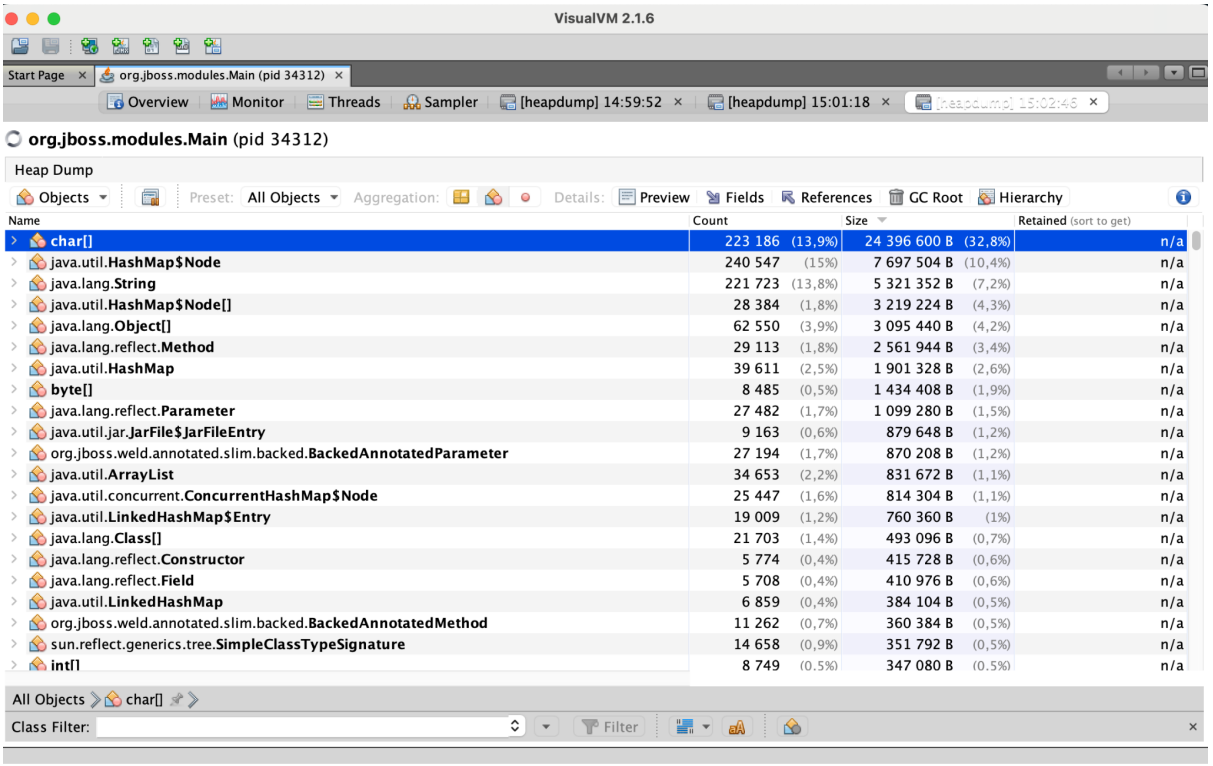

jboss-modules.jar (pid 25006)
Overview
Monitor
Threads
Sampler
Profiler
[heapdump] 4:36:11 PM
[heapdump] 4:36:37 PM
jboss-modules.jar (pid 25006)
Heap Dump
AverageClickPerTimeBean#1
Details:
Preview
Fields
References
GC Root
Hierarchy
Name
Size
Retained
mbeans.AverageClickPerTimeBean#1 24 B (0%) 440 B (0%)
<fields>
static <classLoader> = org.jboss.modules.ModuleClassLoader#249 80 B (0%) 429,944 B (0.9%)
averageValue = java.lang.Double#6 : 3.5454545454545454 24 B (0%) 24 B (0%)
clicksTimeList = java.util.ArrayList#102054 : 12 elements 24 B (0%) 392 B (0%)
static <resolved_references> = java.lang.Object[]#72155 : 1 item 24 B (0%) 24 B (0%)
<references>
timeBean in beans.MainBean#1 40 B (0%) 168 B (0%)
val in java.util.concurrent.ConcurrentHashMap$Node#6558 32 B (0%) 32 B (0%)
resource in com.sun.jmx.mbeanserver.StandardMBeanSupport#3 24 B (0%) 24 B (0%)
AverageClickPerTimeBean#1
<references>
Objects
char[]
CounterBean#1
CounterBean
MBeanNotificationInfo[]#4
AverageClickPerTimeBean#1

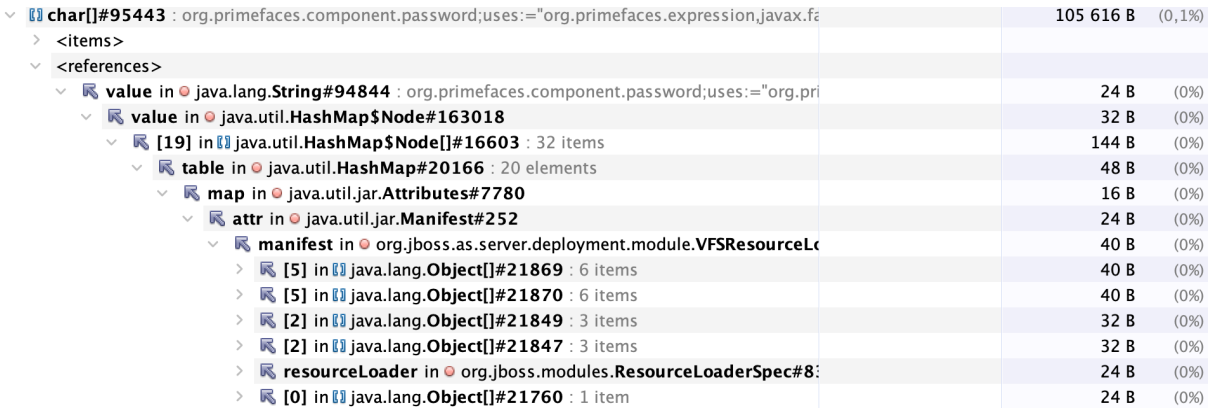


Counter Bean:



Классы, объекты которых занимают наибольший объём памяти





Самый тяжелый instance находится в jboss.as.server.deployment.module.VFSResourceLoader

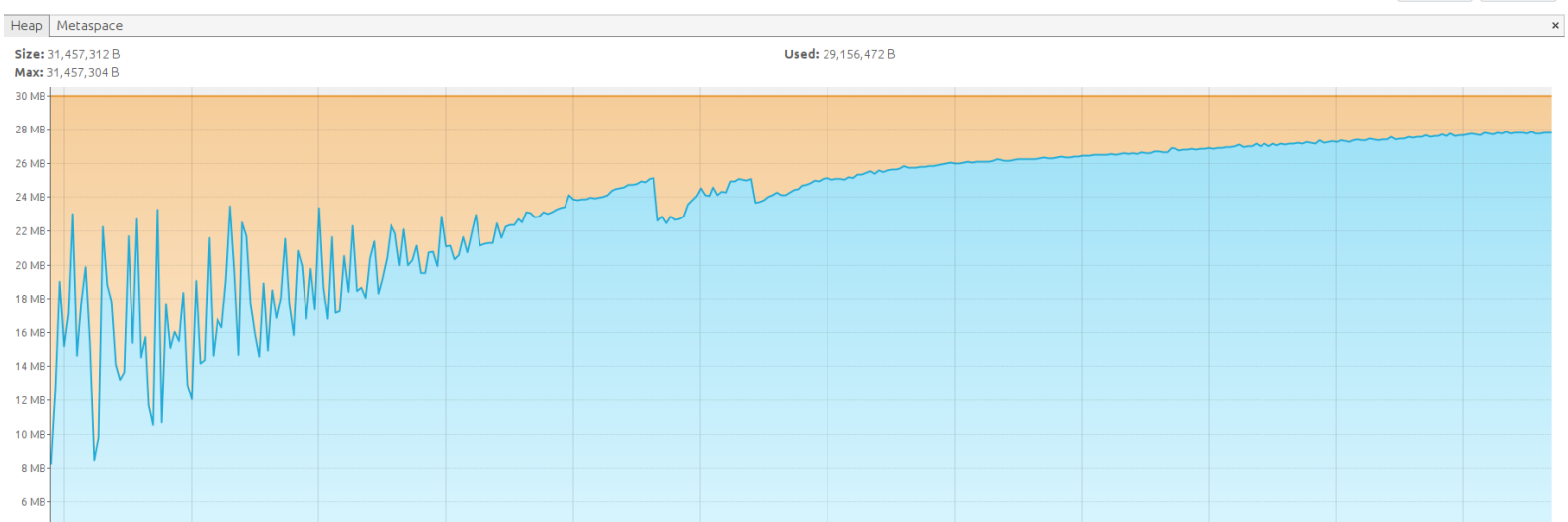
4) Поиск проблем с производительностью

1. Посмотрим на код программы. Можно увидеть, что она входит в бесконечный цикл с запросами. После каждого запроса инициируется засыпание потока на 200 миллисекунд, что приводит к замедлению работы программы.

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

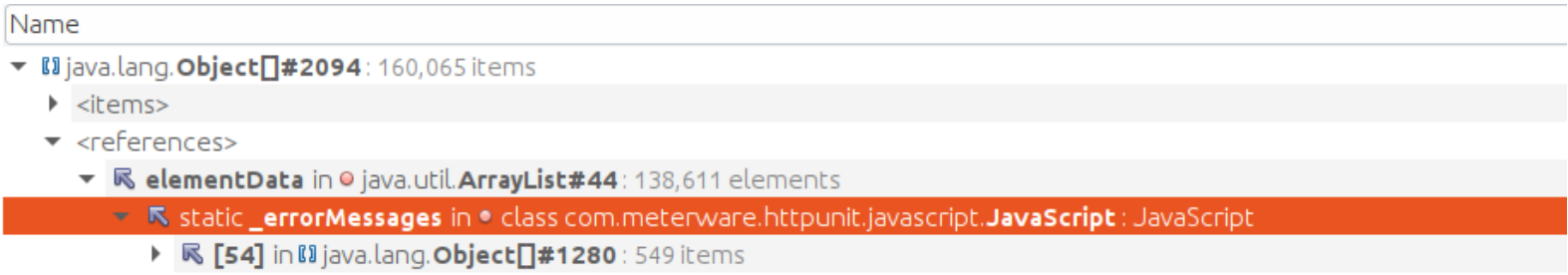
2. Уменьшаем размер кучи до 30 Мб и убираем задержку между запросами, чтобы ускорить выявление ошибки.

   Во вкладке Monitor посмотрим на использование ресурсов. Видно, что размер увеличивается, приближаясь к максимальному. Можно сделать вывод, что утечка памяти существует.



```
Count: 169848[ _response = com.meterware.servletunit.ServletUnitHttpResponse@6b1af67d]
Count: 169849[ _response = com.meterware.servletunit.ServletUnitHttpResponse@101e09ac]
Count: 169850[ _response = com.meterware.servletunit.ServletUnitHttpResponse@35c45cd1]
Count: 169851[ _response = com.meterware.servletunit.ServletUnitHttpResponse@1f227e2b]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

3. Анализируем heap dump и находим объекты, которые занимают больше всего памяти и смотрим, в каком они классе находятся:



Искомый класс - com.meterware.httpunit.javascript.JavaScript. Найдем его в коде и посмотрим, что там находится:

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

```
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 );
    }
}
```

Видно, что элементы накапливаются в списке, но не очищаются, что приводит к утечке памяти.

4. Устраним ошибку.

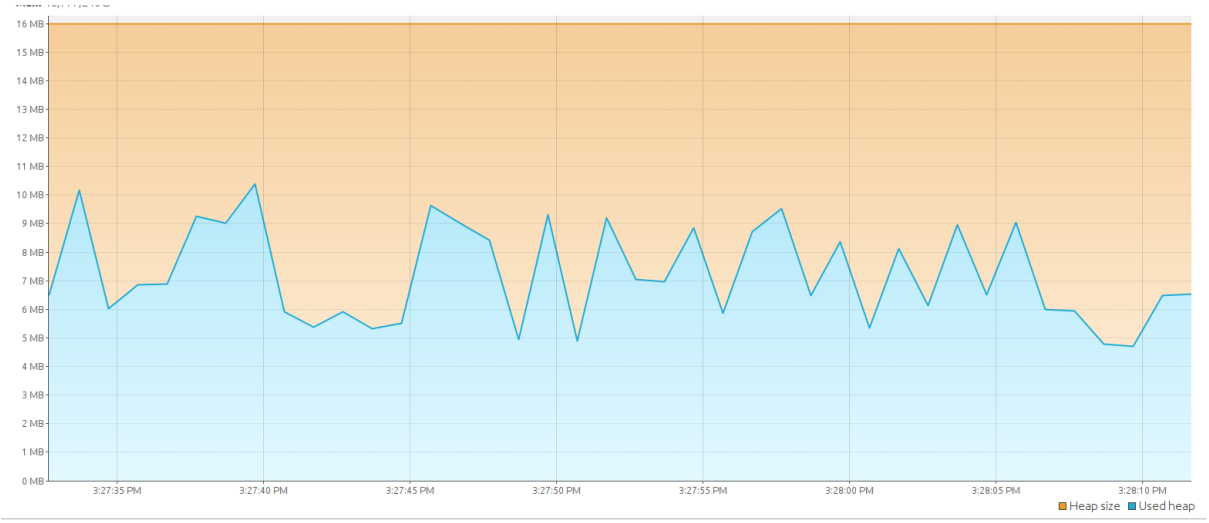
В исходном классе есть метод clearErrorMessages()

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

Добавим очистку списка после каждого запроса в классе Main:

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

5. Проверим нашу программу еще раз:



Видим, что сейчас размер кучи постоянно не растет, сборщик мусор работает в нормально режиме. Следовательно, проблема утечки памяти была устранена.

Вывод:

В результате работы была изучена технология JMX, созданы MBeans для получения информации в JMX Agent (JConsole). В JConsole были получены метрики и информация о JVM. С помощью VisualVM и профилировщика Idea были выявлены и устранены проблемы в выданной по варианту программе.