

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 программного обеспечения”**

**Вариант: 203**

Выполнила:

Лушникова Анастасия

Группа: Р32302

Преподаватель:

Исаев Илья Владимирович

# Задание

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 координаты установленной пользователем точки вышли за пределы отображаемой области координатной плоскости, 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 версию Java Language Specification, реализуемую данной средой исполнения.

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:

## Первый Mbean

```
package com.example.demo.mbean;

import com.example.demo.data.Result;
import com.example.demo.data.ResultHit;
import lombok.Getter;

import javax.management.Notification;
import javax.management.NotificationBroadcasterSupport;

import java.lang.Math;

public class PointManager extends NotificationBroadcasterSupport implements
PointManagerMBean {
    @Getter
    private int establishedPoints;
    @Getter
    private int hitPoints;
    private final double max=4;

    public void onClick(Result result){
        System.out.println("execute point manager");
        if(result.getResultHit()== ResultHit.HIT) hitPoints++;
        establishedPoints++;
        if(Math.abs(result.getX())>max || Math.abs(result.getY())>max){
            Notification notification = new Notification("coordsOutOfGraph",
this, establishedPoints,
                "Coords of shot are out of rendered graph");
            sendNotification(notification);
        }
    }
}
```

```
package com.example.demo.mbean;

public interface PointManagerMBean {
    int getHitPoints();
    int getEstablishedPoints();
}
```

## Второй Mbean

```
package com.example.demo.mbean;

import lombok.Getter;
public class Square implements SquareMBean{
```

```

        @Getter
        double square;
        @Override
        public void updateSquare(int r) {
            square=r*r+Math.PI*r*r/16;
        }
    }

package com.example.demo.mbean;

public interface SquareMBean {
    void updateSquare(int r);
    double getSquare();
}

package com.example.demo.mbean;

import javax.annotation.PostConstruct;
import javax.faces.view.ViewScoped;
import javax.inject.Named;
import javax.management.MBeanServer;
import javax.management.ObjectName;
import java.io.Serializable;
import java.lang.management.ManagementFactory;

@Named("squareUtil")
@ViewScoped
public class SquareUtil implements Serializable {
    private static final long serialVersionUID = 222L;
    private Square square;

    @PostConstruct
    private void init() {
        try {
            MBeanServer mBeanServer =
ManagementFactory.getPlatformMBeanServer();
            ObjectName interval = new
ObjectName("com/example/demo/mbean:type=Square");
            square = new Square();
            mBeanServer.registerMBean(square, interval);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void updateRadius(double r) {
        int radius = (int) r;
        square.updateSquare(radius);
    }
}

```

**Результаты работы бинов JConsole:**

- com.sun.management
- com/example/demo/mb
- PointManager
- Attributes
- Notifications[1]
- Square
- Attributes
- Operations

Name	Value
Square	4.785398163397448

- com.sun.management
- com/example/demo/mb
- PointManager
- Attributes
- Notifications[1]
- Square
- Attributes

Name	Value
EstablishedPoints	5
HitPoints	1

- JMImplementation
- com.sun.management
- com/example/demo/mb
- PointManager
- Attributes
- Notifications[1]
- Square

Notification buffer						
TimeStamp	Type	UserData	SeqNum	Message	Event	Source
18:02:41:641	coordsOutOfGraph		5	Coords of shot ar...	javax.manageme...	com/example/demo/...

## Java Language Specification

- JMImplementation
- com.sun.management
- com/example/demo/mb
- java.lang
- ClassLoading
- Compilation
- GarbageCollector
- Memory
- MemoryManager
- MemoryPool
- OperatingSystem
- Runtime
- Attributes
  - Name
  - ClassPath
  - SpecVendor
  - SpecVersion
  - StartTime

Attribute value

Name	Value
SpecVersion	17

Refresh

MBeanAttributeInfo

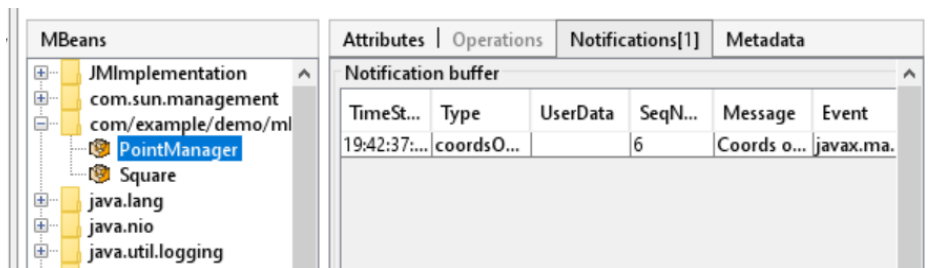
Name	Value
Attribute:	
Name	SpecVersion
Description	SpecVersion
Readable	true
Writable	false
Is	false
Type	java.lang.String

## Visual VM

Attribute values	
Name	Value
EstablishedPoints	<p>EstablishedPoints</p> <p>43</p> <p>19:09</p> <p>Discard chart</p>
HitPoints	<p>HitPoints</p> <p>21</p>

Attributes	Operations	Notifications	Metadata
Attribute values			
Name	Value		
Square	<p>Discard chart</p>		
Attribute exposed for management			

The screenshot shows the IDE's MBeans console and the MBean Explorer. The MBeans console displays the 'PointManager' MBean with attributes 'EstablishedPoints' (5) and 'HitPoints' (1). The MBean Explorer shows the package hierarchy including 'com.sun.management', 'com.example.demo.ml', 'PointManager', and 'Square'.



## Visual VM part 2

Status: CPU sampling in progress

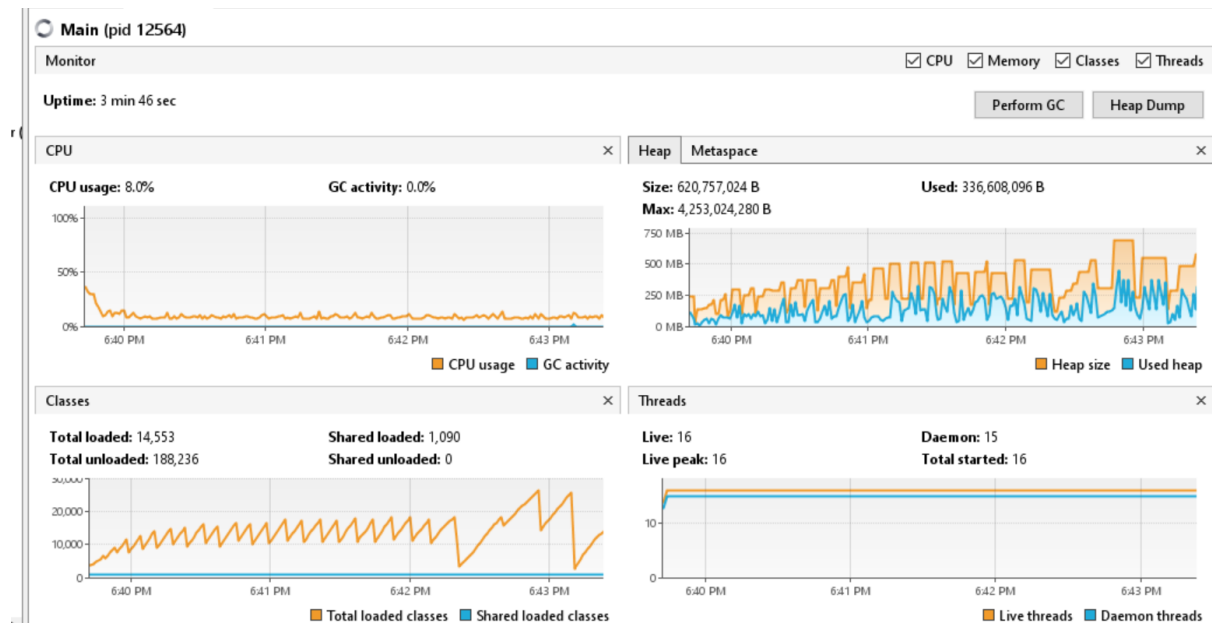
CPU samples: Thread CPU time

Results: Statistics: Threads Count: 97 Total Time (CPU): 49,125 ms Thread Dump

Name	Thread Time (CPU)	Thread Time (CPU) / sec
RMI TCP Connection(8)-192.168.1.1	12,437 ms (25.3%)	0.0 ms (0%)
MSC service thread 1-7	7,390 ms (15%)	0.0 ms (0%)
default task-1	6,500 ms (13.2%)	0.0 ms (0%)
RMI TCP Connection(12)-192.168.1.1	5,015 ms (10.2%)	234 ms (23.5%)
ServerService Thread Pool -- 100	3,609 ms (7.3%)	0.0 ms (0%)
MSC service thread 1-4	2,203 ms (4.5%)	0.0 ms (0%)
default task-2	1,937 ms (3.9%)	0.0 ms (0%)
MSC service thread 1-1	1,875 ms (3.8%)	0.0 ms (0%)
MSC service thread 1-3	1,406 ms (2.9%)	0.0 ms (0%)
DestroyJavaVM	1,406 ms (2.9%)	0.0 ms (0%)
MSC service thread 1-2	921 ms (1.9%)	0.0 ms (0%)

## Monitoring

Как видно из графиков количество используемой памяти растет, более того постоянно дополнительно выделяется память, на исполнение наших процессов, так же видно, что количество загруженных классов тоже растет.

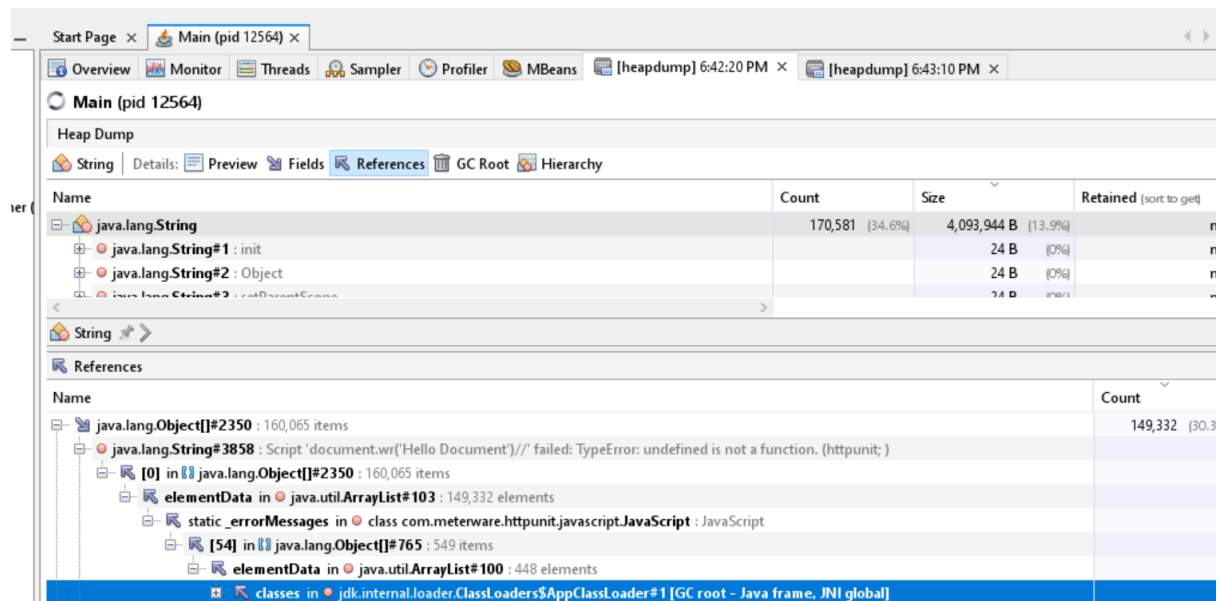


Сделаем heapdump с разными временными промежутками

Classes by Number of Instances [ view all ]			Classes by Size of Instances [ view all ]		
	byte[]	171,318 (34.8%)		byte[]	19,293,400 B (65.6%)
	java.lang.String	170,581 (34.6%)		java.lang.String	4,093,944 B (13.9%)
	java.util.TreeMap\$Entry	28,444 (5.8%)		java.util.TreeMap\$Entry	1,137,760 B (3.9%)
	java.lang.Long	14,588 (3%)		java.lang.Object[]	977,288 B (3.3%)
	java.util.TreeMap	9,012 (1.8%)		java.util.TreeMap	432,576 B (1.5%)
Instances by Size [ view all ]			Dominators by Retained Size [ view all ]		
	byte[]	210,408 (35.8%)		byte[]	23,969,008 B (66.8%)
	java.lang.String	209,670 (35.6%)		java.lang.String	5,032,080 B (14%)
	java.util.TreeMap\$Entry	33,923 (5.8%)		java.util.TreeMap\$Entry	1,356,920 B (3.8%)
	java.lang.Long	17,338 (2.9%)		java.lang.Object[]	1,316,912 B (3.7%)
	java.util.TreeMap	10,750 (1.8%)		java.util.TreeMap	516,000 B (1.4%)

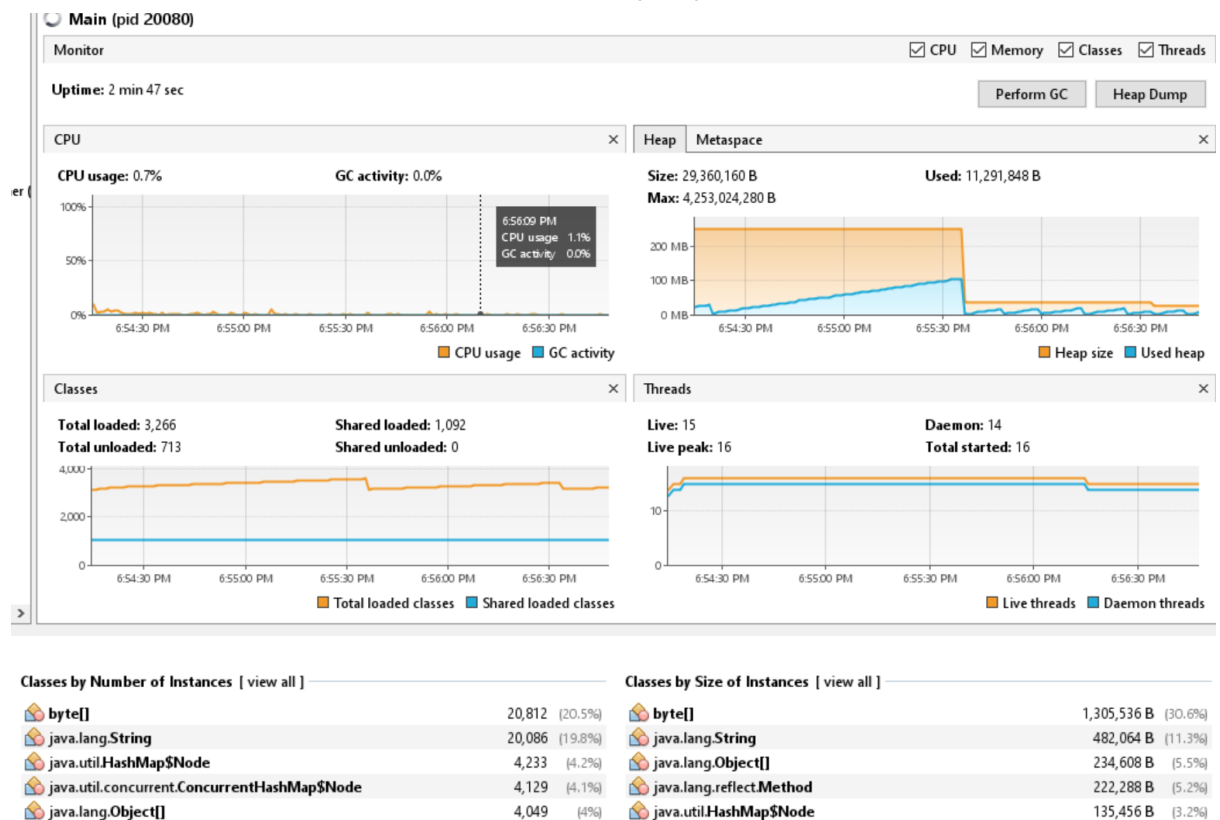
видно, что количество `bute[]` и `java.lang.String` растёт довольно быстро, и занимает около 30% всей памяти. Изучив данные объекты, мы можем заметить, что довольно большая часть стрингов, это элементы `_errorMessages`, изучив код, понимаем, что эти элементы добавляются в лист, но не удаляются















Попробуем в функцию Main в бесконечном цикле добавить очистку массива.

Показатели значительно улучшились



Classes by Number of Instances [ view all ]			Classes by Size of Instances [ view all ]		
 byte[]	21,060	(20.3%)	 byte[]	1,324,696 B	(30.3%)
 java.lang.String	20,329	(19.6%)	 java.lang.String	487,896 B	(11.1%)
 java.util.HashMap\$Node	4,262	(4.1%)	 java.lang.Object[]	238,008 B	(5.4%)
 java.lang.Object[]	4,148	(4%)	 java.lang.reflect.Method	222,288 B	(5.1%)
 java.util.concurrent.ConcurrentHashMap\$Node	4,130	(4%)	 java.util.HashMap\$Node	136,384 B	(3.1%)

Вывод:

Профилирование и мониторинг программ позволяют выявлять потенциальные проблемы в работе программы, выявлять их причины и устранять их до сильных последствий.