

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет ПИ и КТ

Лабораторная работа №4

по дисциплине: «Основы программной инженерии»

Вариант 793: “Мониторинг и профилирование”

Выполнили:

Орчиков Даниил Валерьевич,

Болорболд Аригуун,

группа P3211

Преподаватель:

**Клименков Сергей Викторович**

Практик:

**Птицын Максим Евгеньевич**

Санкт-Петербург

2024

1. **Задание:**
2. Для своей программы из [лабораторной работы #3](https://github.com/DaniilOrchikov/WEB3/tree/master/src) по дисциплине "Веб-программирование" реализовать:

* MBean, считающий общее число установленных пользователем точек, а также число точек, не попадающих в область. В случае, если количество установленных пользователем точек стало кратно 10, разработанный MBean должен отправлять оповещение об этом событии.
* MBean, определяющий процентное отношение "попаданий" к общему числу кликов пользователя по координатной плоскости.

1. С помощью утилиты JConsole провести мониторинг программы:

* Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
* Определить имя и версию ОС, под управлением которой работает JVM.

1. 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 класса, объекты которого занимают наибольший объём памяти JVM; определить пользовательский класс, в экземплярах которого находятся эти объекты.

1. 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.

**2. Выполнение:**

Пункт 1: создание MBean-ов

* MBean, считающий общее число установленных пользователем точек, а также число точек, не попадающих в область. В случае, если количество установленных пользователем точек стало кратно 10, разработанный MBean должен отправлять оповещение об этом событии.

ClickRateMBean.java:

*package* com.example.web3.MBeans;

*public interface* ClickRateMBean {

*default void* click(*boolean* hit) {};

*int* getResult();

*int* getHit();

}

ClickRate.java:

*package* com.example.web3.MBeans;

*import* javax.management.Notification;

*import* javax.management.NotificationBroadcasterSupport;

*import* java.io.Serializable;

*public class* ClickRate *extends* NotificationBroadcasterSupport *implements* Serializable, ClickRateMBean {

*int* totalClicks = 0;

*int* hitClicks = 0;

*int* notificationCounter = 0;

*@Override*

*public void* click(*boolean* hit) {

totalClicks++;

*if*(hit) {

hitClicks++;

}

*if*(totalClicks % 10 == 0) {

System.out.println("10!");

Notification notification = *new* Notification("10-кратные выстрелы", *this*, notificationCounter++, System.currentTimeMillis(), "Число выстрелов теперь кратно 10.");

sendNotification(notification);

}

}

*@Override*

*public int* getResult() {

*return* totalClicks;

}

*@Override*

*public int* getHit() {

*return* hitClicks;

}

}

* MBean, определяющий процентное отношение "попаданий" к общему числу кликов пользователя по координатной плоскости.

HitRateMBean.java:

*package* com.example.web3.MBeans;

*public interface* HitRateMBean {

*default void* click(*boolean* hit) {}

*double* getPercentage();

*double* getResult();

*double* getHit();

}

HitRate.java:

*package* com.example.web3.MBeans;

*import* java.io.Serializable;

*public class* HitRate *implements* Serializable, HitRateMBean {

*private double* totalClicks = 0.0;

*private double* hitClicks = 0;

*@Override*

*public void* click(*boolean* hit) {

totalClicks++;

*if*(hit) {

hitClicks++;

}

}

*@Override*

*public double* getPercentage() {

*if*(totalClicks == 0) {

*return* 0;

}

*return* hitClicks / totalClicks \* 100;

}

*@Override*

*public double* getResult() {

*return* totalClicks;

}

*@Override*

*public double* getHit() {

*return* hitClicks;

}

}

Их имплементация в контроллере:

*@Named*

*@ApplicationScoped*

*public class* Controller *implements* Serializable {

*private double* x;

*private double* y;

*private double* r = 1;

*@Inject*

*private* CollectionBean collectionBean;

*private final* HitRateMBean hitRate;

*private final* ClickRateMBean clickRate;

*public* Controller() {

clickRate = *new* ClickRate();

hitRate = *new* HitRate();

*try* {

MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();

ObjectName clickRateName = *new* ObjectName("KanekiKen:name=clickRate");

ObjectName hitRateName = *new* ObjectName("Shizlo:name=hitRate");

mBeanServer.registerMBean(clickRate, clickRateName);

mBeanServer.registerMBean(hitRate, hitRateName);

NotificationListener notificationListener = (notification, event) -> System.out.println("Уведомление от Грега: " + notification.getMessage());

mBeanServer.removeNotificationListener(clickRateName, notificationListener, *null*, *null*);

} *catch* (Exception e) {

e.printStackTrace();

System.out.println("Ты что ахуел?");

}

}

*public void* submit() {

Point point = *new* Point();

point.setX(x);

point.setY(y);

point.setR(r);

point.setHit(hit(x, y, r));

collectionBean.addPoint(point);

clickRate.click(hit(x, y, r));

hitRate.click(hit(x, y, r));

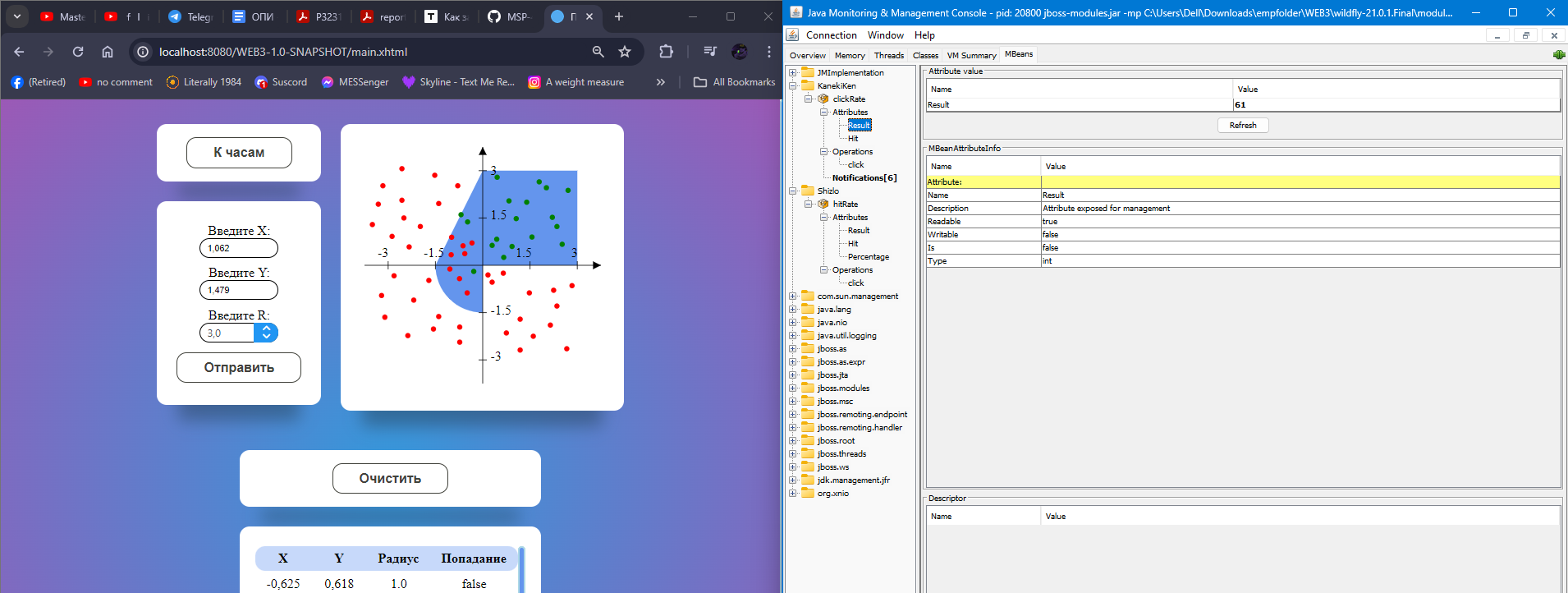
System.out.println("У меня есть " + clickRate.getResult() + " демонов в голове, из них " + clickRate.getHit() + " ещё шизанутые");

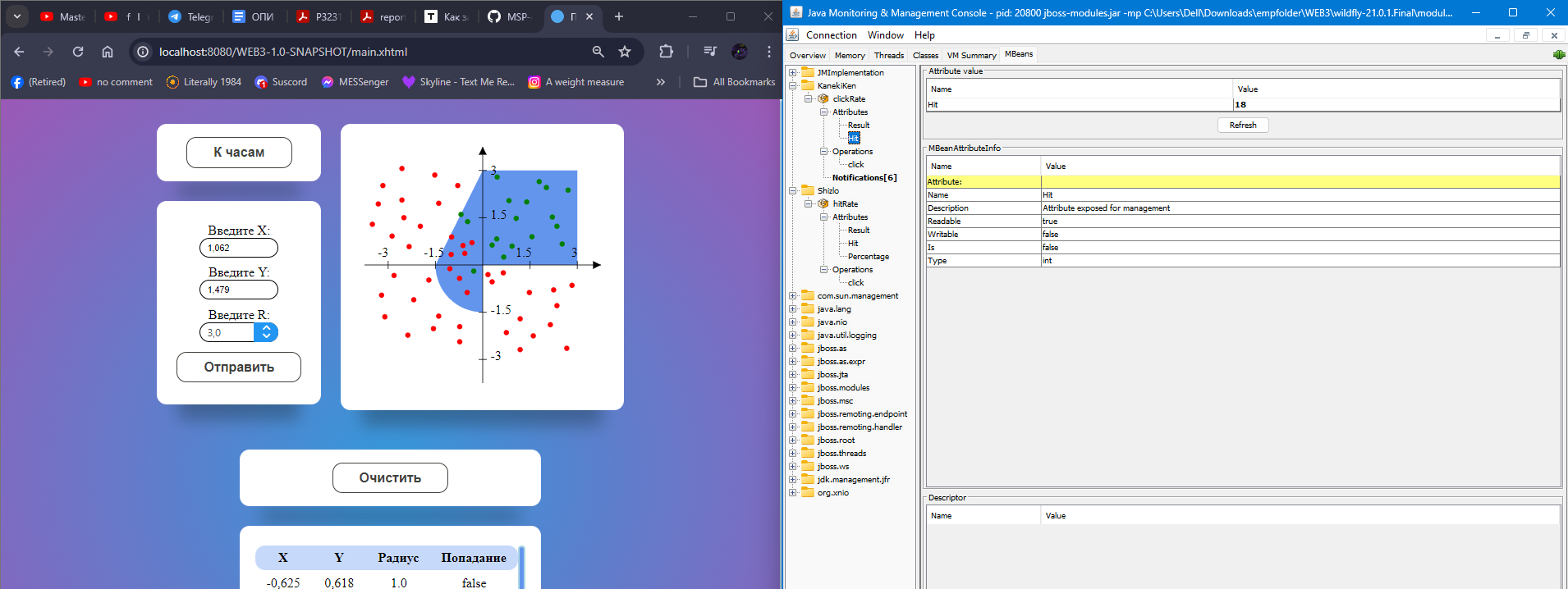
System.out.println(hitRate.getPercentage());

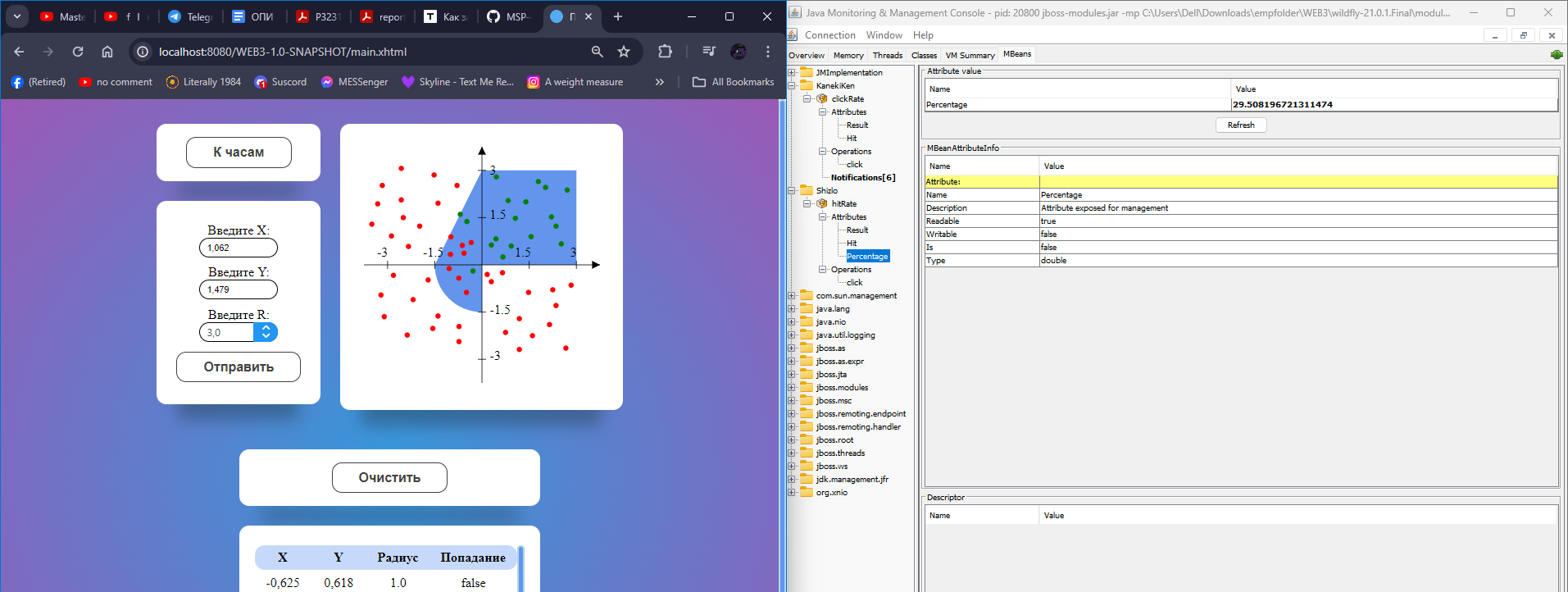
………

Пункт 2: Jconsole

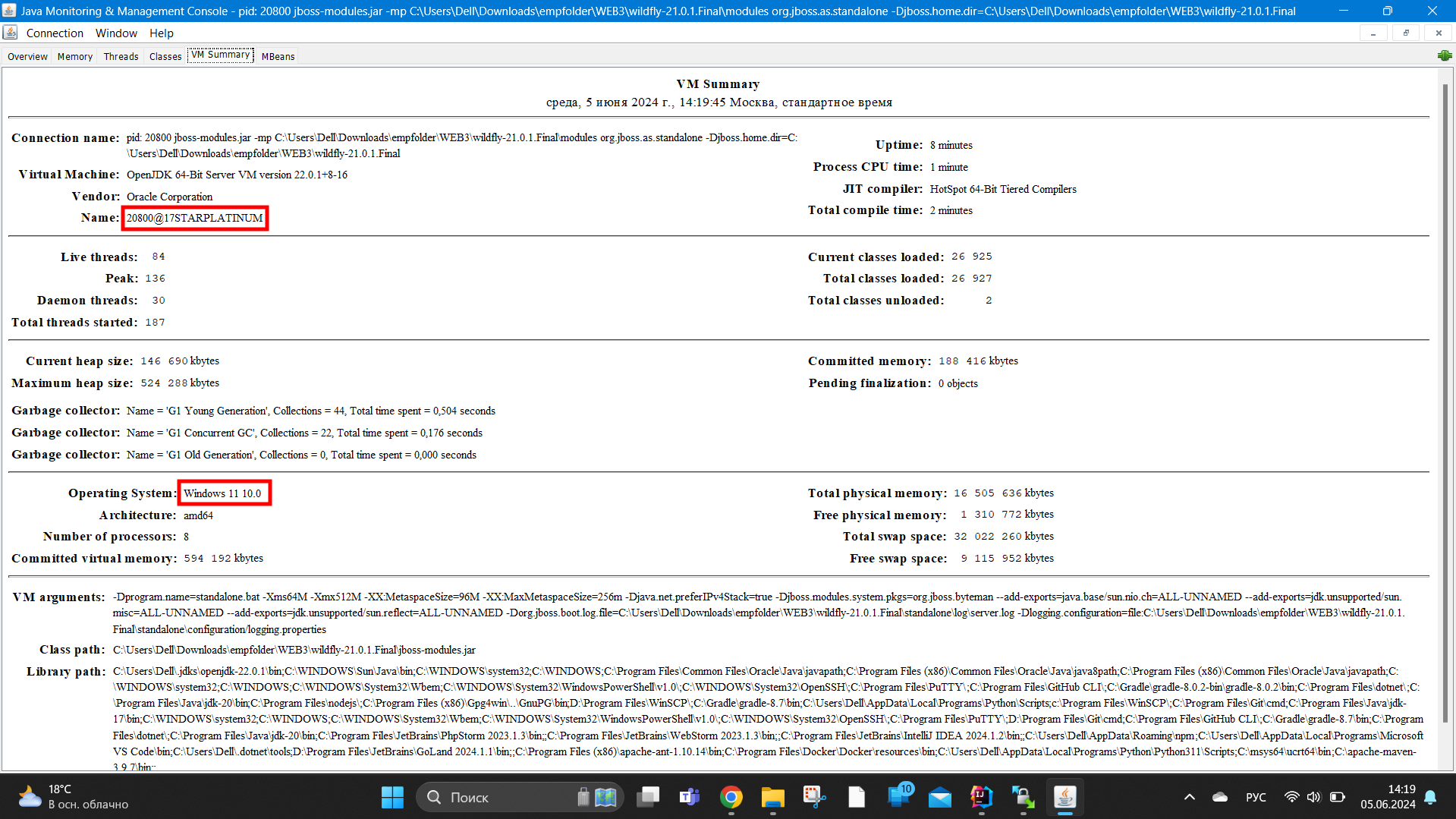
1. Показания разработанных MBean-классов:



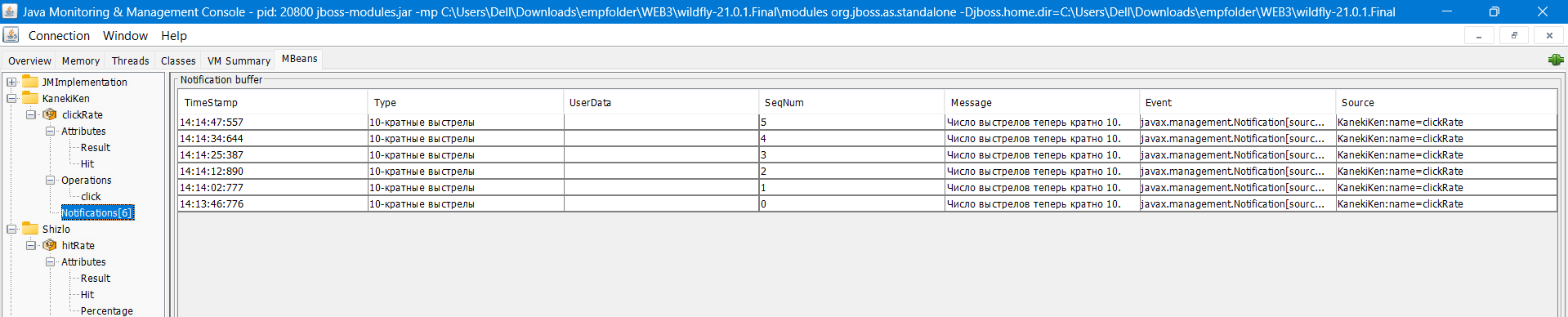




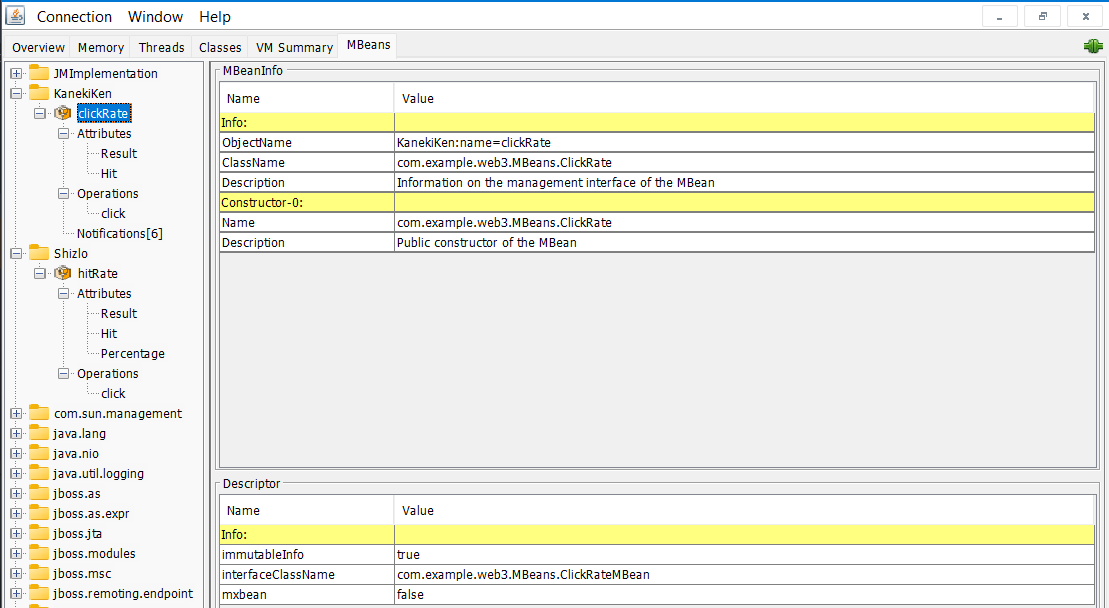
1. Имя и версия ОС:

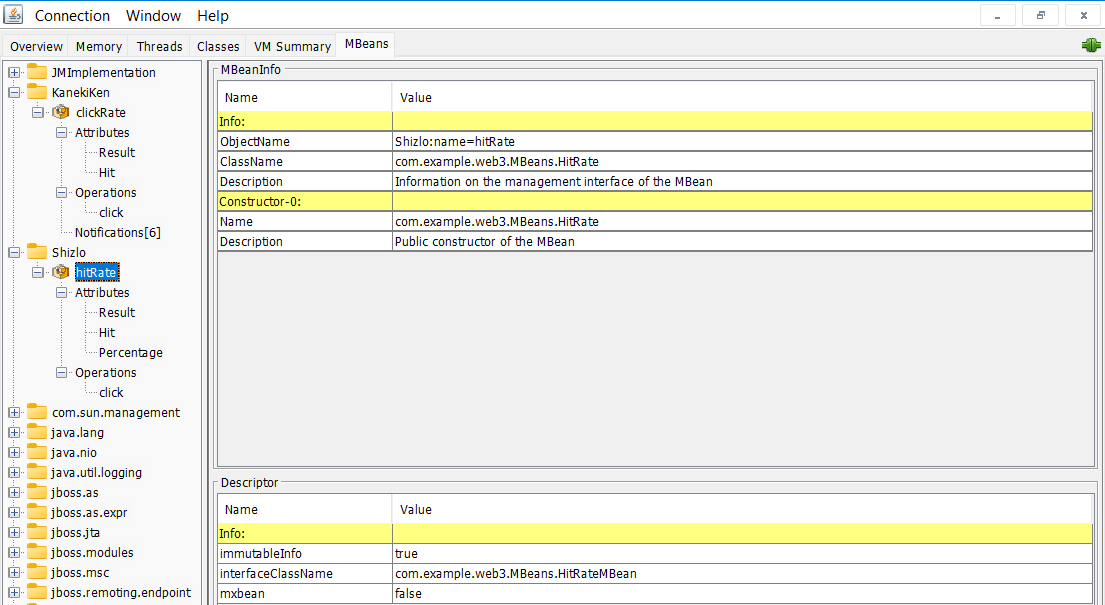


1. Сообщения о выстрелов (Notifications):



1. Метаданные:



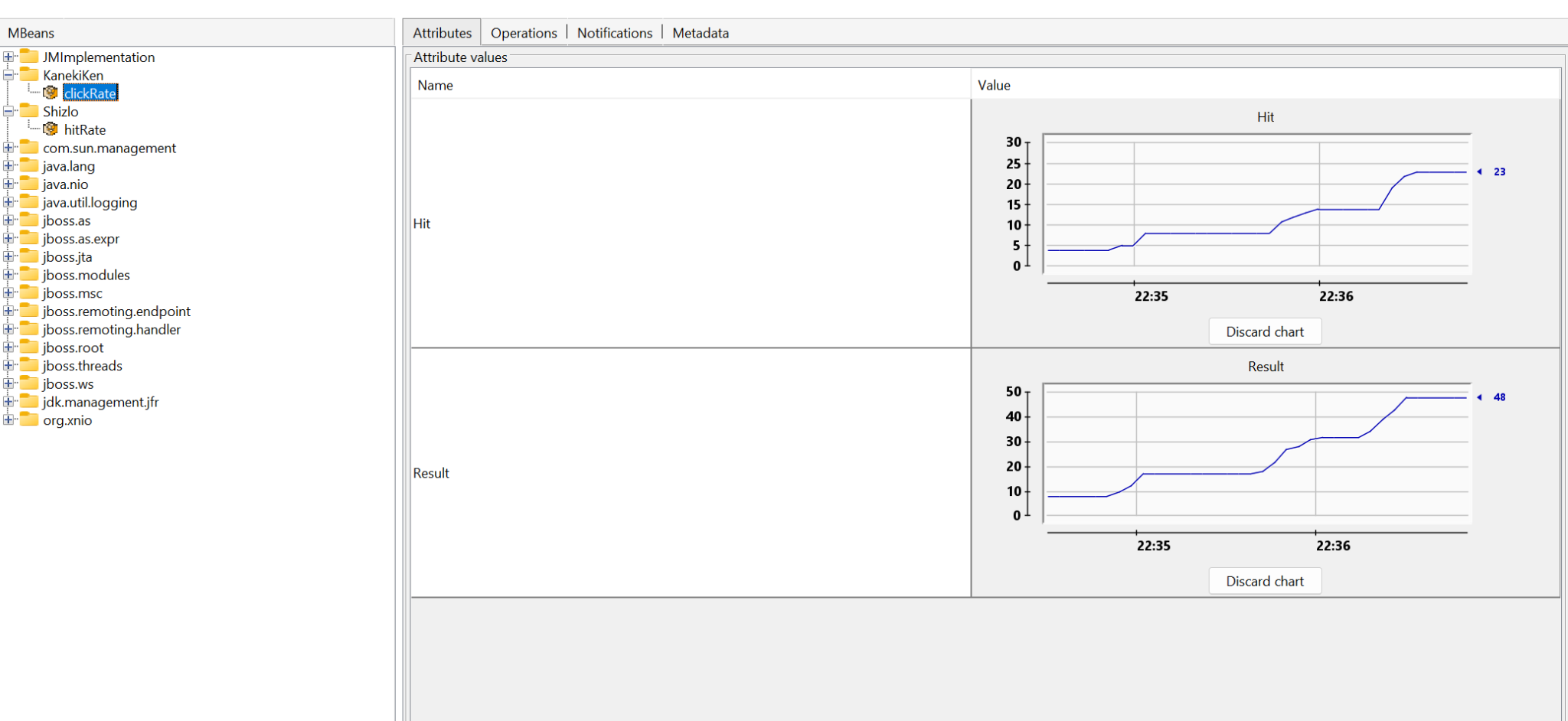
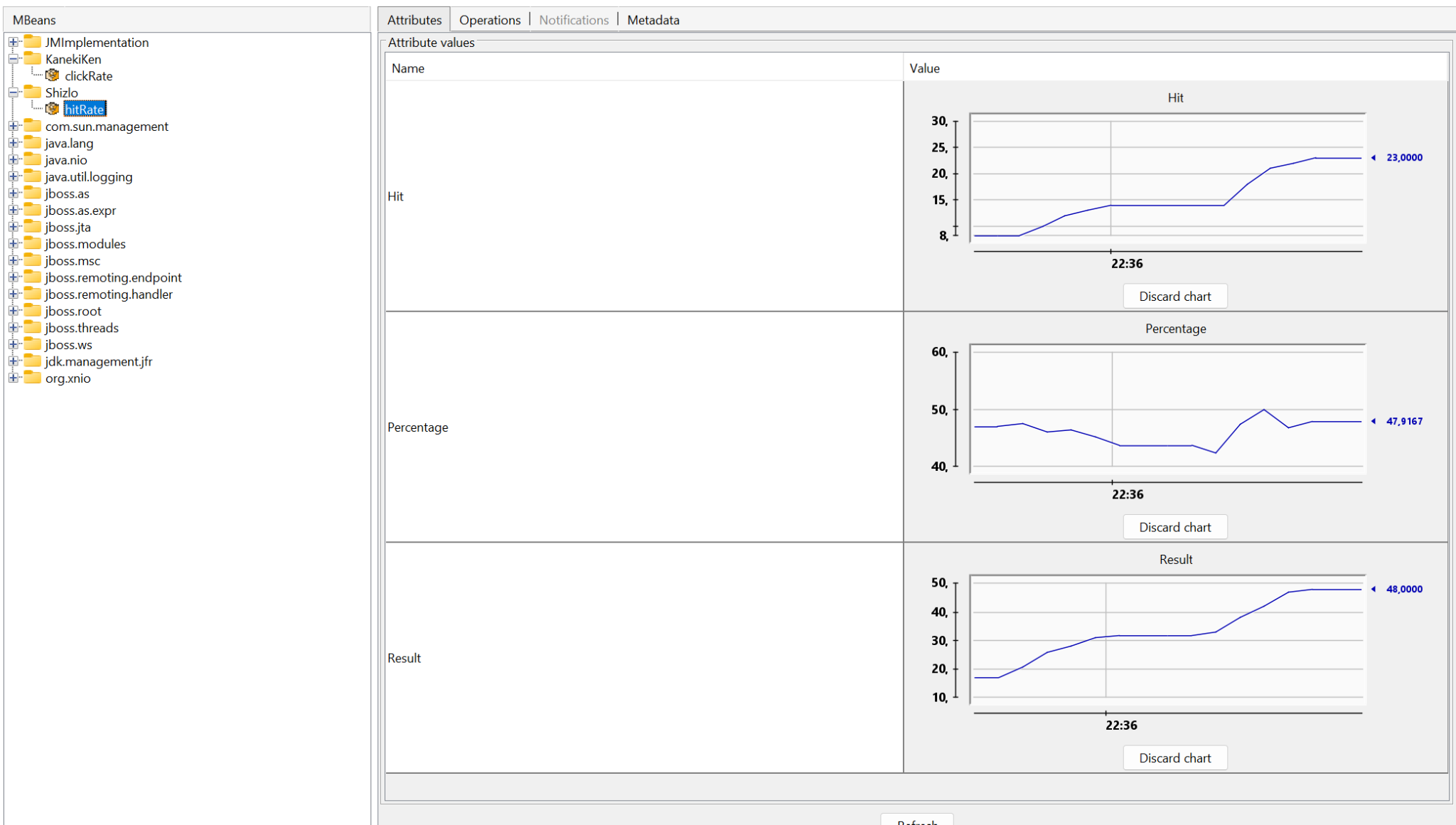
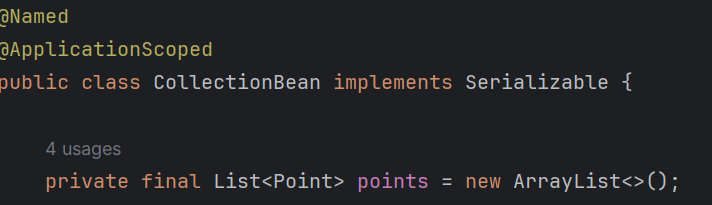


1. Выводы по результатам мониторинга:

В ходе мониторинга с использованием утилиты JConsole было определено, что:

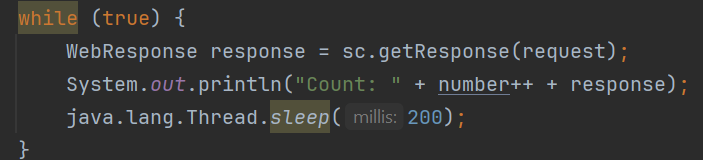
* 1. JVM работает под управлением ОС Windows 11 версии 10.0
  2. MBean ClickRate (KanekiKen) был успешно разработан и зарегистрирован. В случае если количества нажатий стане кратным 10 MBean отправляет уведомления, которые были получены с помощью JConsole. Таким образом, было зарегистрировано и протестировано получение уведомлений от MBean, что позволяет оперативно реагировать на события, что является важной частью мониторинга. Аналогично с MBean HitRate (Shizlo).

Пункт 3: VisualVM

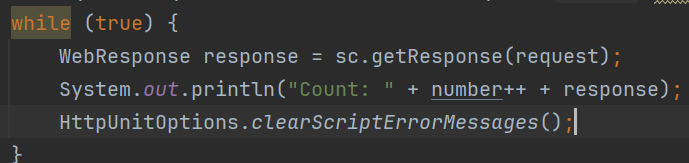
1. Графики показаний MBean классов  
   ClickRate:  
   HitRate:
2. Определение класса, объекты которого занимают больше всего места в памяти - класс Point  
   Пользовательский класс в котором хранятся объекты класса Point

Пункт 4

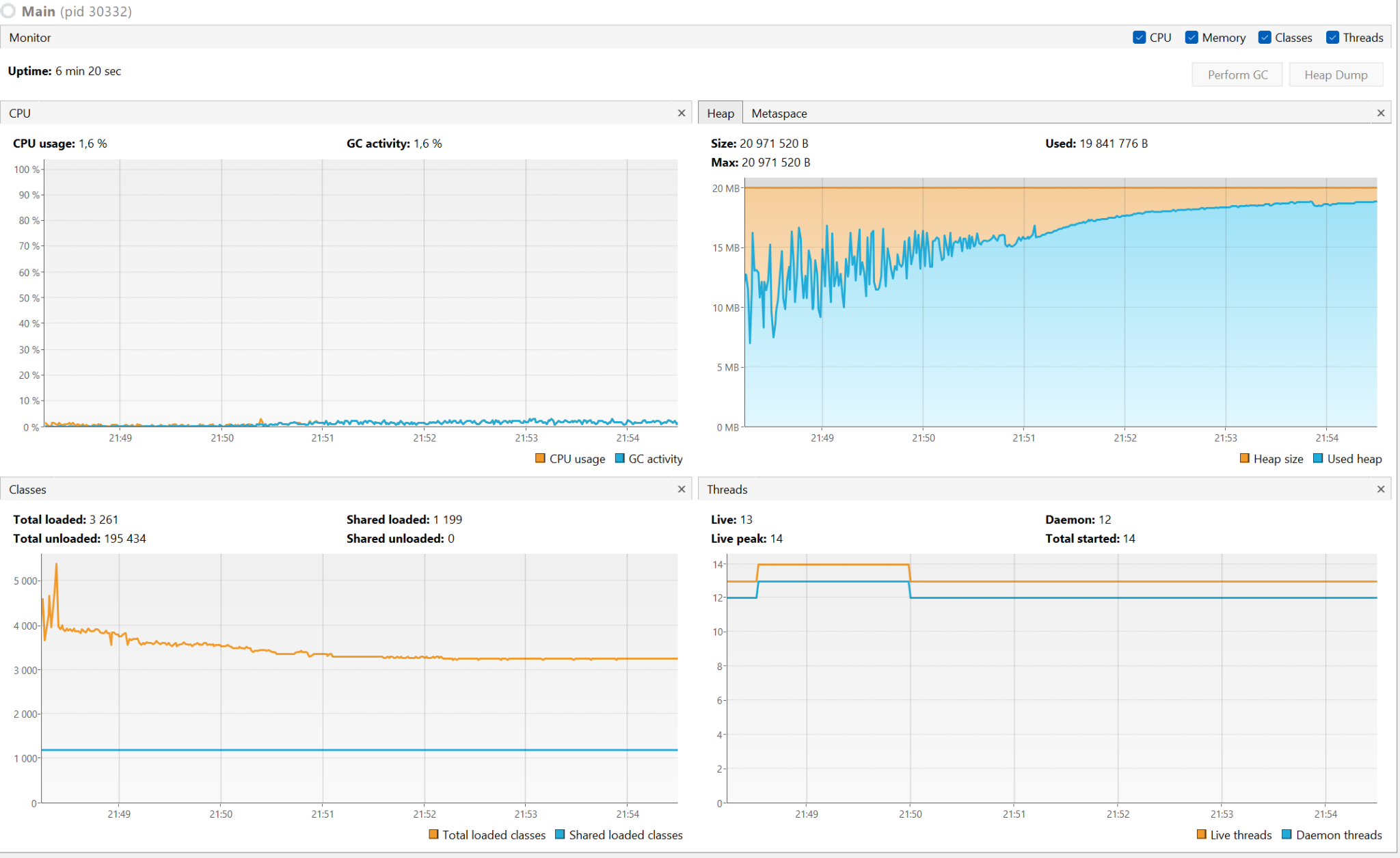
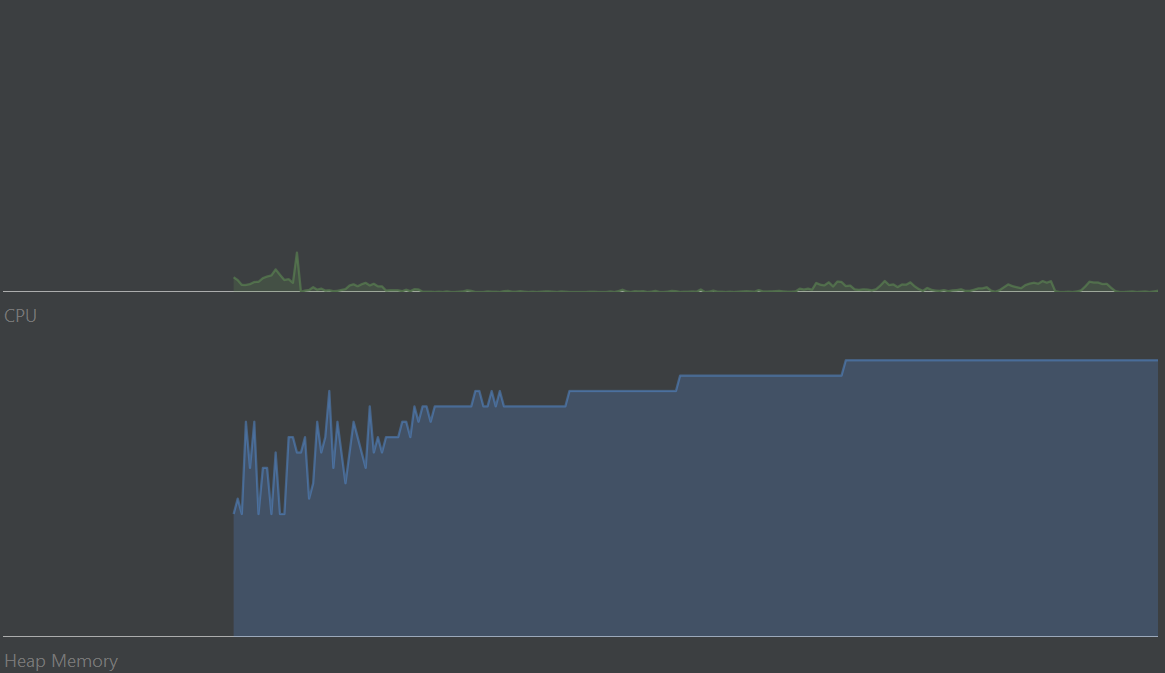
Выявленные проблемы:

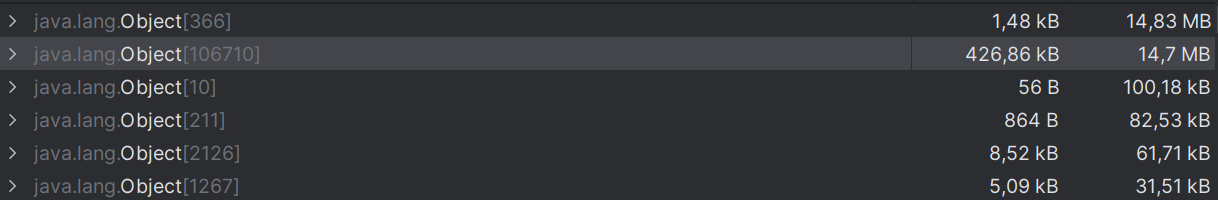
1. sleep(200) из-за которого каждую итерацию цикла программа засыпает на 200 мс.  
   
2. Отсутствие очищения коллекции сообщений об ошибках, что через некоторое время приводит к переполнению кучи.

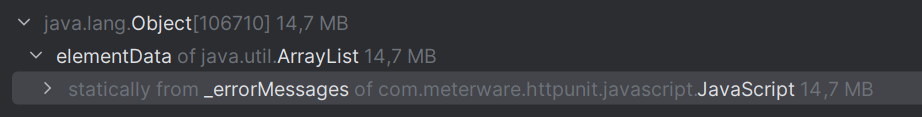
Пути устранения проблем:

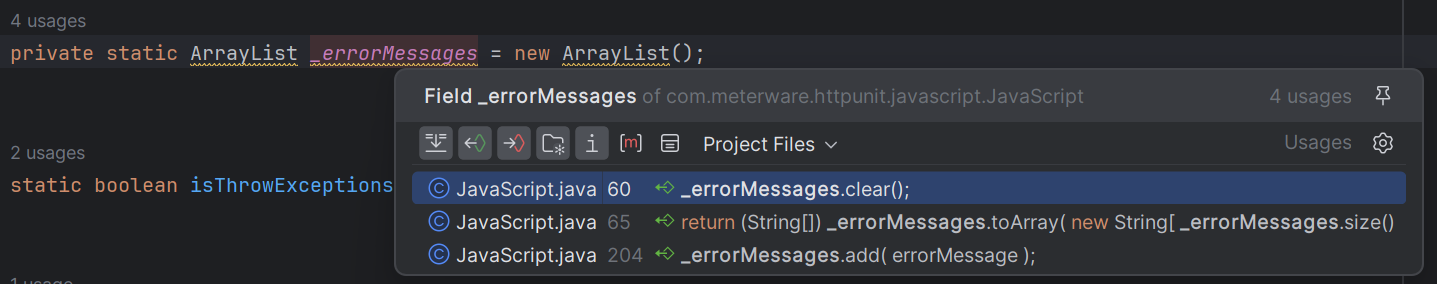
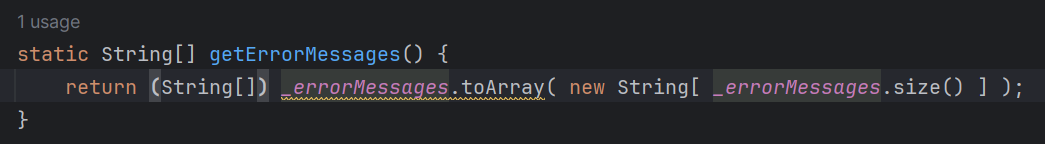
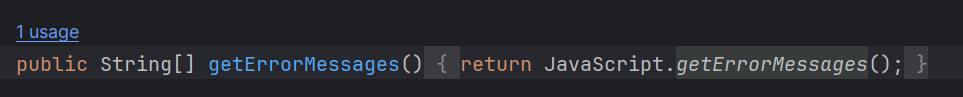
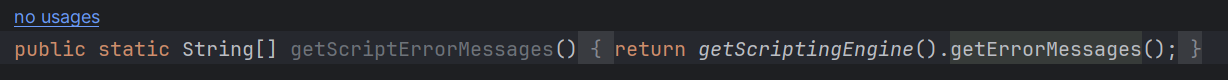
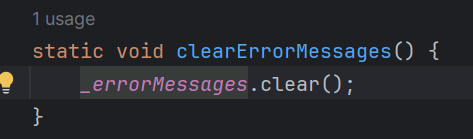
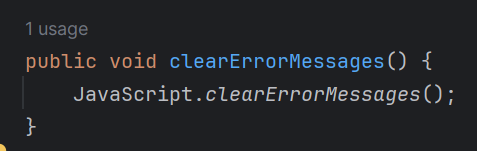
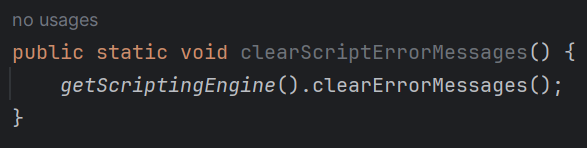
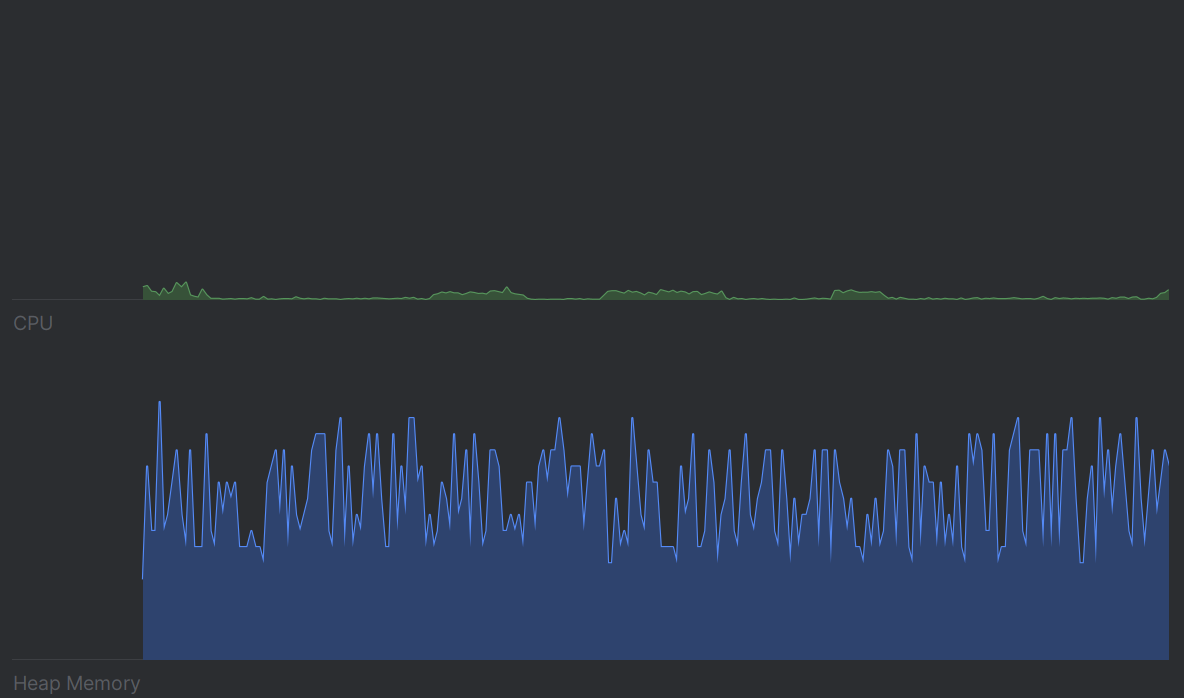
1. Удаляем строку java.lang.Thread.sleep(200); из файла Main.java
2. Вместо удаленной строки пишем HttpUnitOptions.clearScriptErrorMessages();

Алгоритм действий, которые помогли локализовать проблему

1. Снизим максимальный размер кучи до 20 мб (для этого в конфигурации запуска запишем в VM options -Xmx20m)
2. Запустим программу и откроем вкладку Monitor с графиками нагрузки на процессор и заполнения кучи в VisualVM:  
    и в профилировщике Intellij IDEA:
3. Видно, что через некоторое время куча заполняется почти полностью а также возрастает нагрузка на процессор. Под конец измерений программы упала с ошибкой 
4. Делаем дамп памяти

Сразу видно каких объектов у нас очень много:  




1. Перемещаемся в код и видим что у данного поля есть метод очищающий его, геттер и метод, добавляющий в него ошибки.  
   
2. При этом можно проследить вызовы геттера и заметить, что в коде содержимое объекта не используется:  
     
     
   
3. Также можно заметить что объект нигде не очищается  
     
     
   
4. Будем вызывать clearScriptErrorMessages в конце каждого цикла - после каждого запроса для избавления от переполнения памяти
5. Запустим программу и снова проанализируем ее в VisualVM и в профилировщике Intellij IDEA  
   
6. Видим, что проблема с переполнением кучи решена

Вывод:

В общем, защитить лабу в Ягодной это что-то с чем-то. Здесь мы рассматривали мониторинг и профилирование в контексте Java: JMX, M(X)Beans, JConsole, VisualVM. VisualVM является более расширенной по сравнению с JConsole, поэтому его можно ещё использовать для профилирования.

Сам предмет был довольно каламбурным (практик особенно способствовал этому) в изучении. Зато мы уверены, что мы теперь распоряжаемся полным арсеналом для программной инженерии, что будет сильным преимуществом по сравнению с остальными неитмошными разрабами, которые в основном занимались самоизучением.