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

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

Вариант №645

Выполнил

Путинцев Данил Денисович

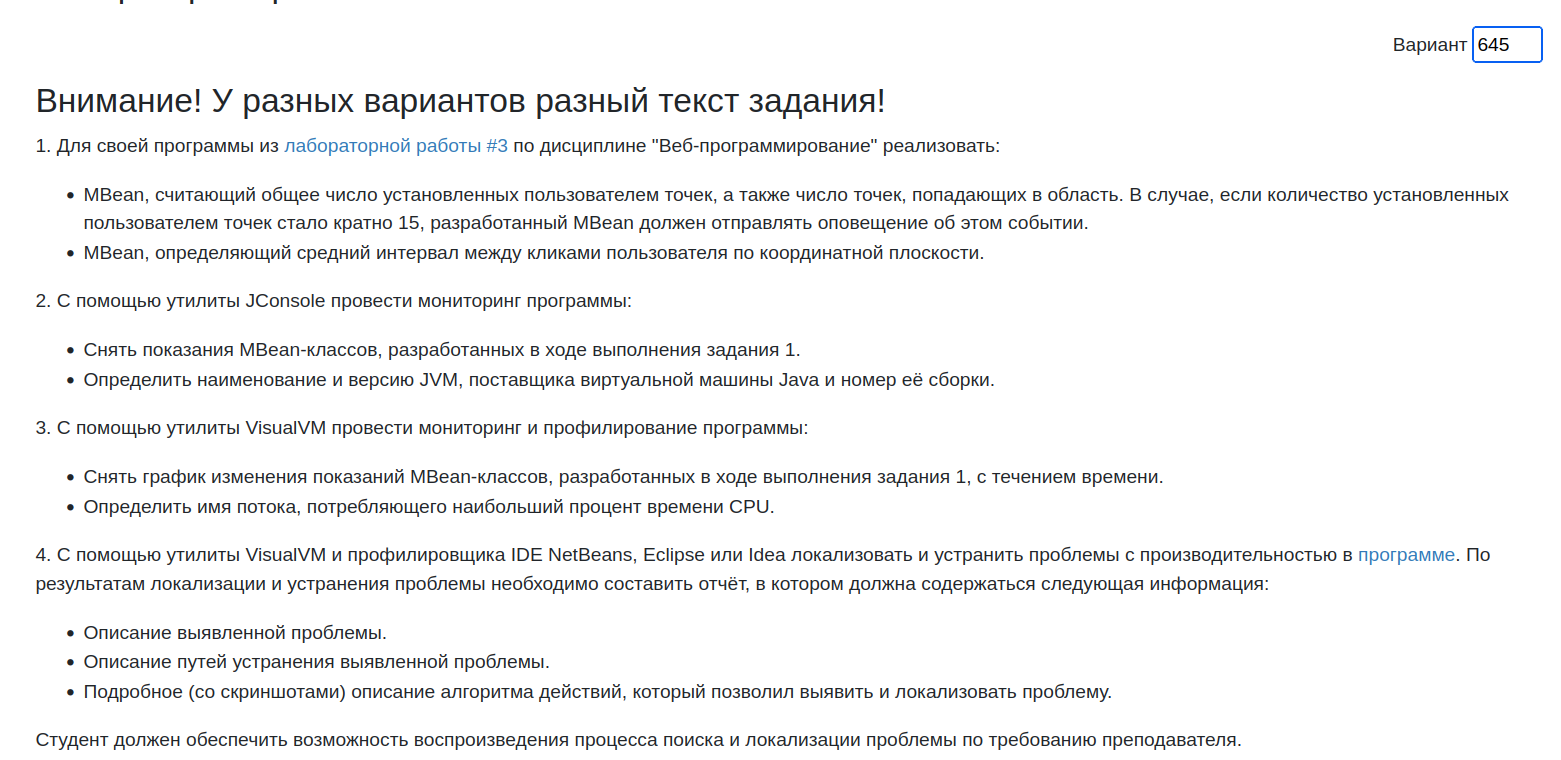
Группа P3207

Проверил(а)

Преподаватель: Миху Вадим Дмитриевич

Санкт-Петербург 2025 год

## Текст задания



## Исходный код разработанных MBean-классов и сопутствующих классов.

package org.example;  
  
import jakarta.enterprise.context.ApplicationScoped;  
import jakarta.inject.Inject;  
import jakarta.inject.Named;  
  
import java.io.Serializable;  
import java.time.Duration;  
import java.time.Instant;  
  
@Named("Interval")  
@ApplicationScoped  
public class Interval implements Serializable, IntervalMBean {  
 private String averageInterval;  
 private Instant lastClickTime;  
 private Long duration = 0L;  
 private Long totalIntervalMillis = 0L;  
  
 public Interval(){  
  
 }  
  
 @Inject  
 public Interval(RegMBeans reg){  
 this.reg = reg;  
 reg.registerBean(this);  
 }  
 RegMBeans reg;  
  
 @Inject  
 Count count;  
  
 @Override  
 public String getAverageInterval() {  
 return String.*format*("%.2f сек", calcAverageInterval() / 1000);  
 }  
  
 public void averageInterval(String averageInterval) {  
 this.averageInterval = averageInterval;  
 }  
  
 public Instant getLastClickTime() {  
 return lastClickTime;  
 }  
 public void setLastClickTime(Instant lastClickTime) {  
 this.lastClickTime = lastClickTime;  
 }  
  
 @Override  
 public void registerClick() {  
 Instant now = Instant.*now*();  
 if (lastClickTime != null) {  
 long interval = Duration.*between*(lastClickTime, now).toMillis();  
 totalIntervalMillis += interval;  
 }  
 lastClickTime = now;  
 }  
  
 @Override  
 public double calcAverageInterval() {  
 if (count.getAllPoints() < 2) return 0;  
 return (double) totalIntervalMillis / (count.getAllPoints() - 1);  
 }  
}

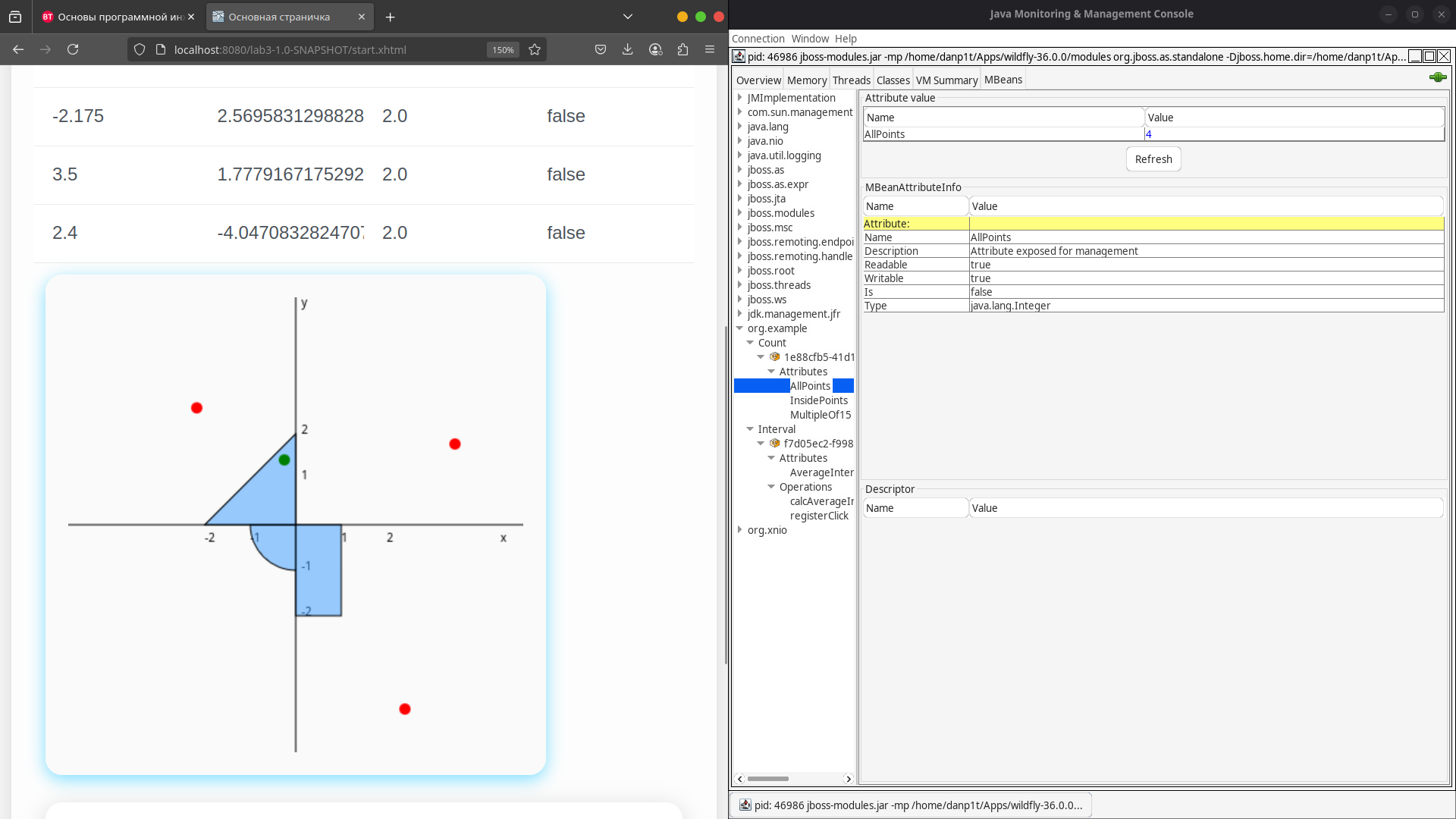
package org.example;  
  
public interface IntervalMBean {  
 void registerClick();  
 double calcAverageInterval();  
 String getAverageInterval();  
}

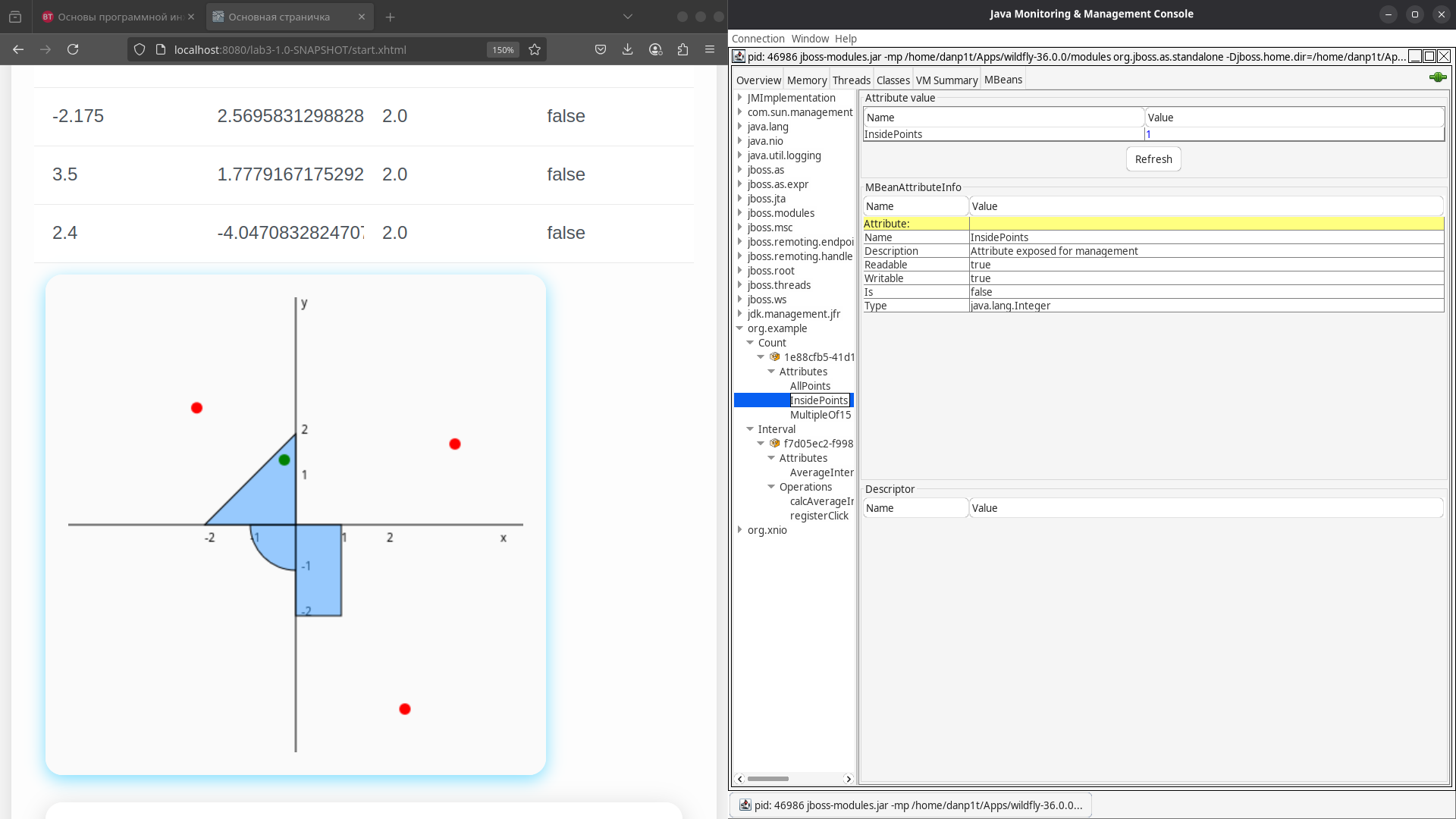
package org.example;  
  
  
import jakarta.enterprise.context.SessionScoped;  
import jakarta.inject.Inject;  
import jakarta.inject.Named;  
  
import java.io.Serializable;  
  
@Named("Count")  
@SessionScoped  
public class Count implements Serializable, CountMBean{  
 private Integer allPoints = 0;  
 private Integer insidePoints = 0;  
  
 public Count(){  
  
 }  
 @Inject  
 public Count(RegMBeans reg){  
 this.reg = reg;  
 reg.registerBean(this);  
 }  
 RegMBeans reg;  
  
 @Override  
 public Integer getAllPoints() {  
 return allPoints;  
 }  
  
 @Override  
 public void setAllPoints(Integer allPoints) {  
 this.allPoints = allPoints;  
 }  
  
 @Override  
 public Integer getInsidePoints() {  
 return insidePoints;  
 }  
  
 @Override  
 public void setInsidePoints(Integer insidePoints) {  
 this.insidePoints = insidePoints;  
 }  
  
 @Override  
 public boolean isMultipleOf15() {  
 return allPoints % 15 == 0 && allPoints != 0;  
 }  
  
  
}

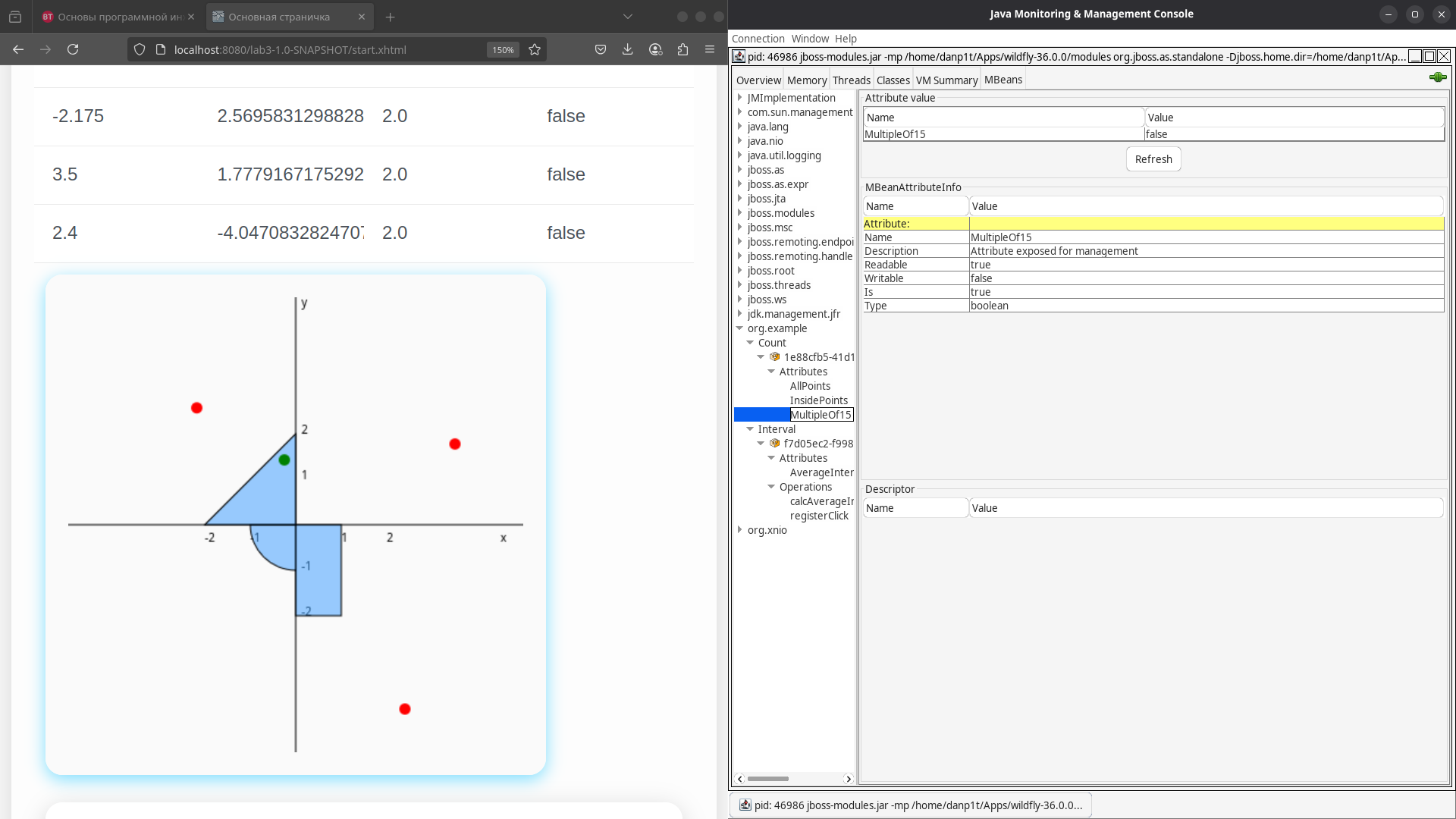
package org.example;  
  
public interface CountMBean {  
 Integer getAllPoints();  
 void setAllPoints(Integer allPoints);  
 Integer getInsidePoints();  
 void setInsidePoints(Integer insidePoints);  
 boolean isMultipleOf15();  
}

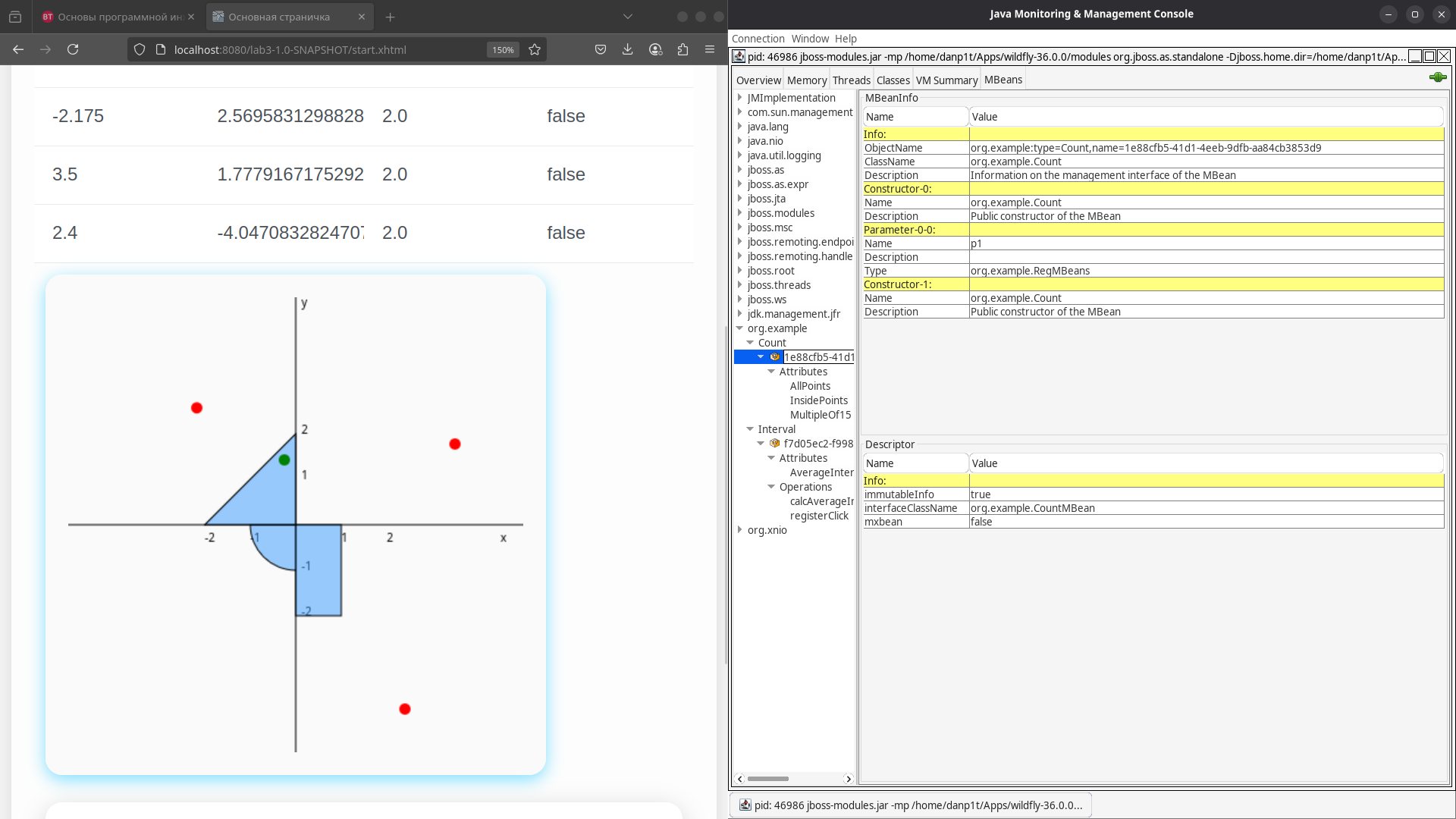
package org.example;  
  
import jakarta.enterprise.context.ApplicationScoped;  
import jakarta.inject.Named;  
import jakarta.servlet.ServletContextListener;  
  
import javax.management.\*;  
import java.lang.management.ManagementFactory;  
import java.util.HashMap;  
import java.util.UUID;  
@Named  
@ApplicationScoped  
public class RegMBeans implements ServletContextListener {  
 private final HashMap<Object, ObjectName> bean\_names = new HashMap<>();  
 public void registerBean(Object bean) {  
 try {  
 String cur\_name = UUID.*randomUUID*().toString();  
 var domain = bean.getClass().getPackageName();  
 var type = bean.getClass().getSimpleName();  
 var objectName = new ObjectName(String.*format*("%s:type=%s,name=%s", domain, type, cur\_name));  
 bean\_names.put(bean, objectName);  
 ManagementFactory.*getPlatformMBeanServer*().registerMBean(bean, objectName);  
 } catch (InstanceAlreadyExistsException | MBeanRegistrationException | NotCompliantMBeanException | MalformedObjectNameException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void unregisterBean(Object bean) {  
 try {  
 ManagementFactory.*getPlatformMBeanServer*().unregisterMBean(bean\_names.get(bean));  
 } catch (InstanceNotFoundException | MBeanRegistrationException e) {  
 e.printStackTrace();  
 }  
 }  
}

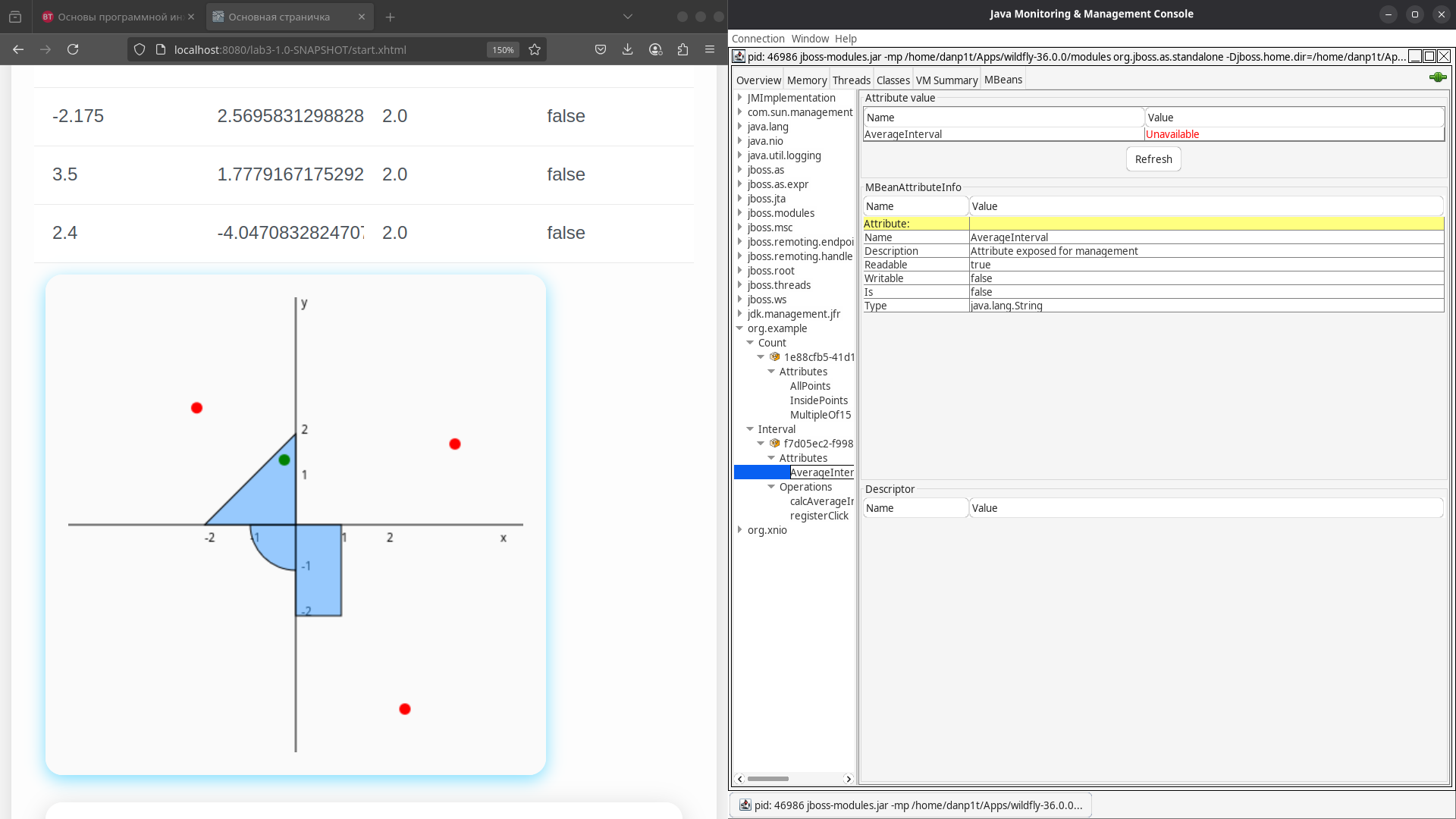
## Скриншоты программы JConcole со снятыми показаниями, выводы по результатам мониторинга.

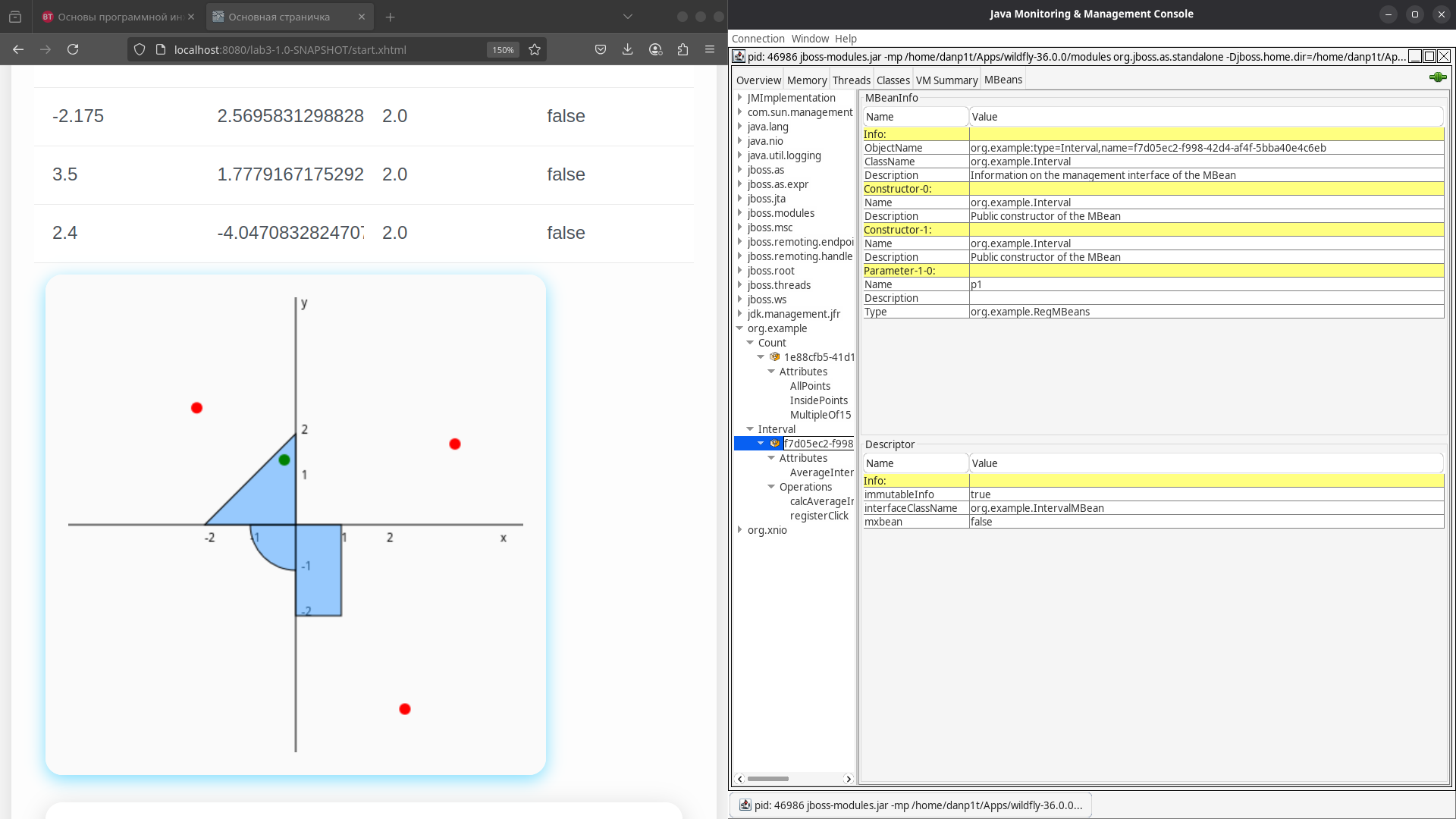


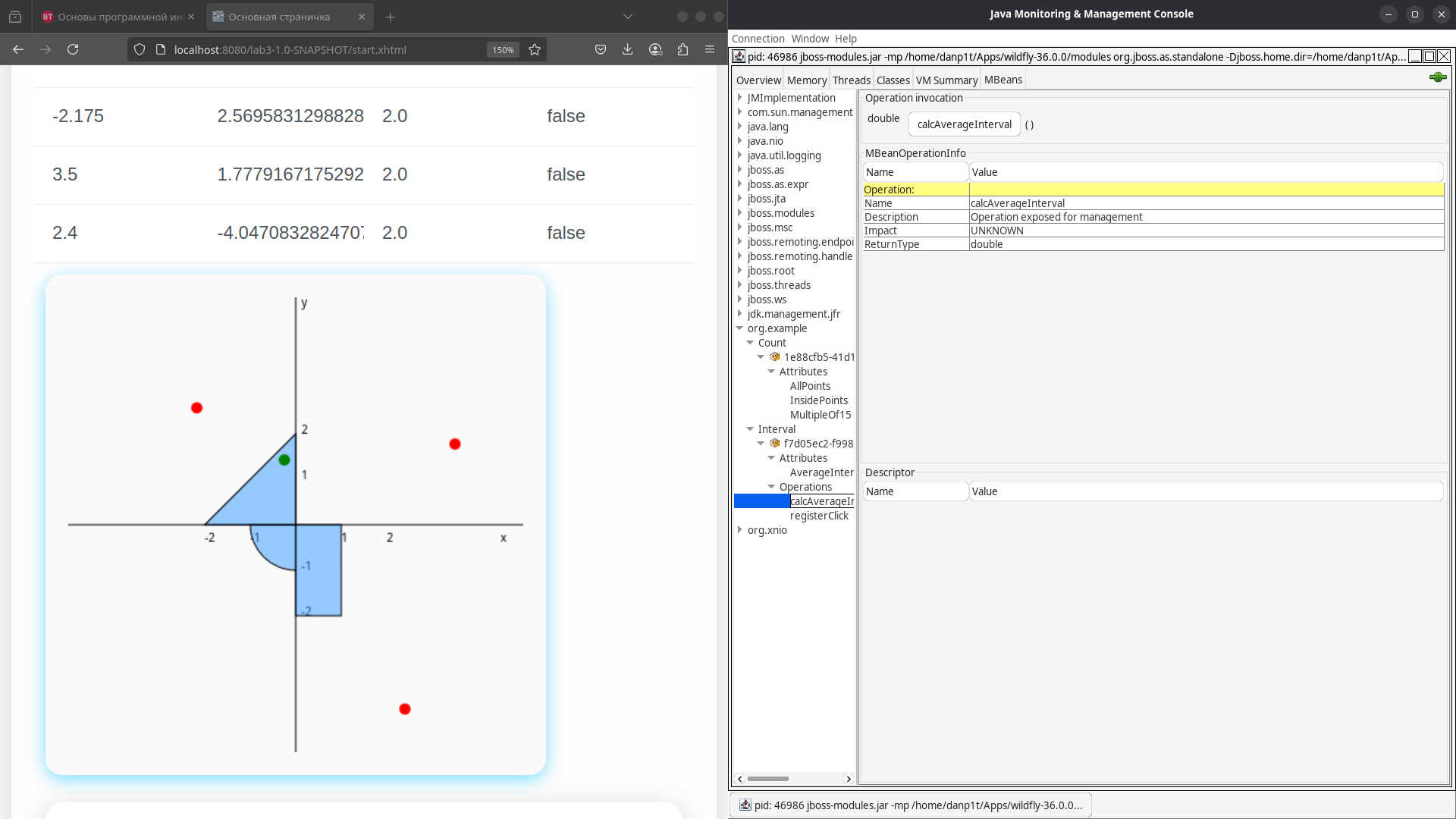


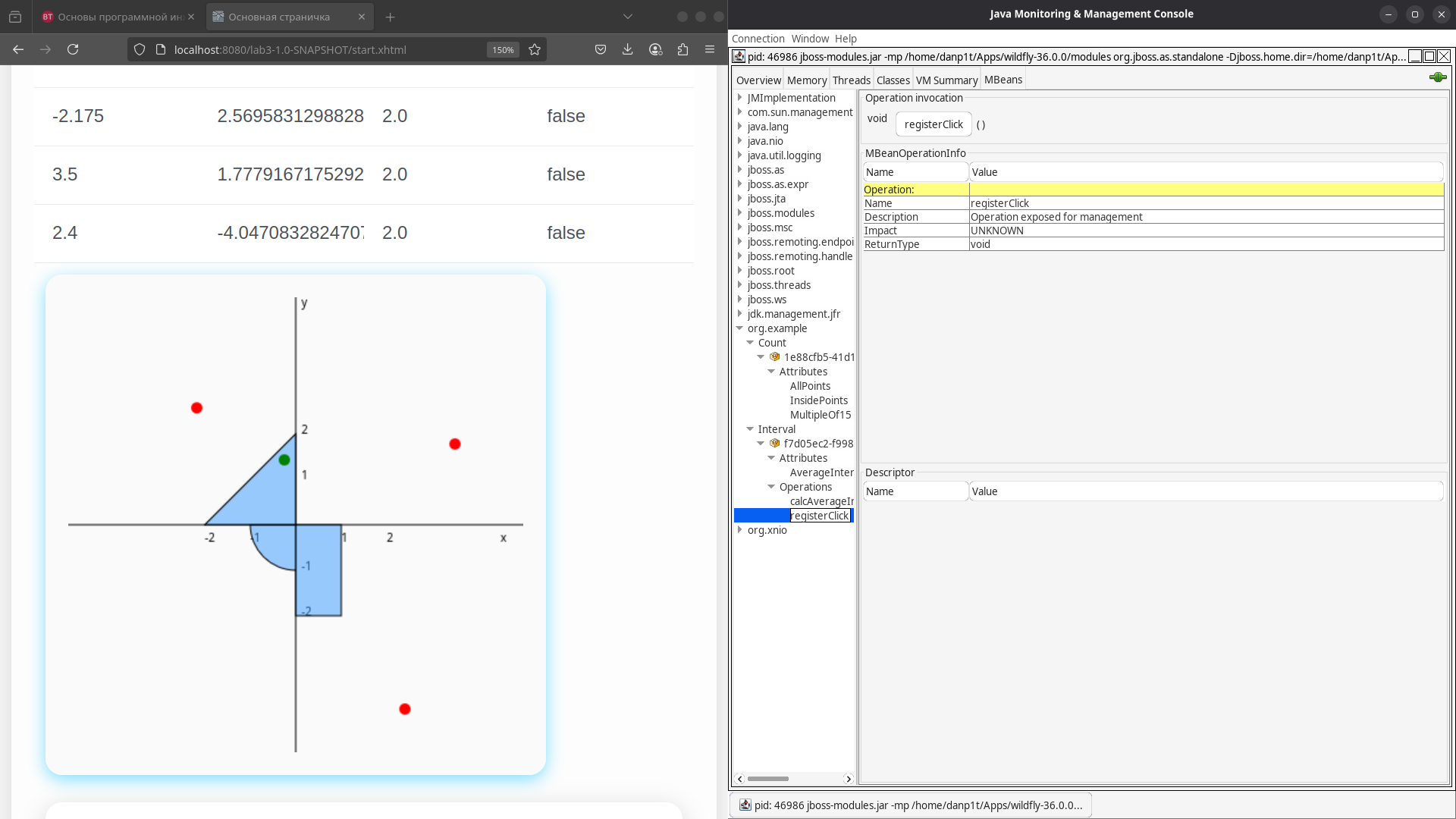


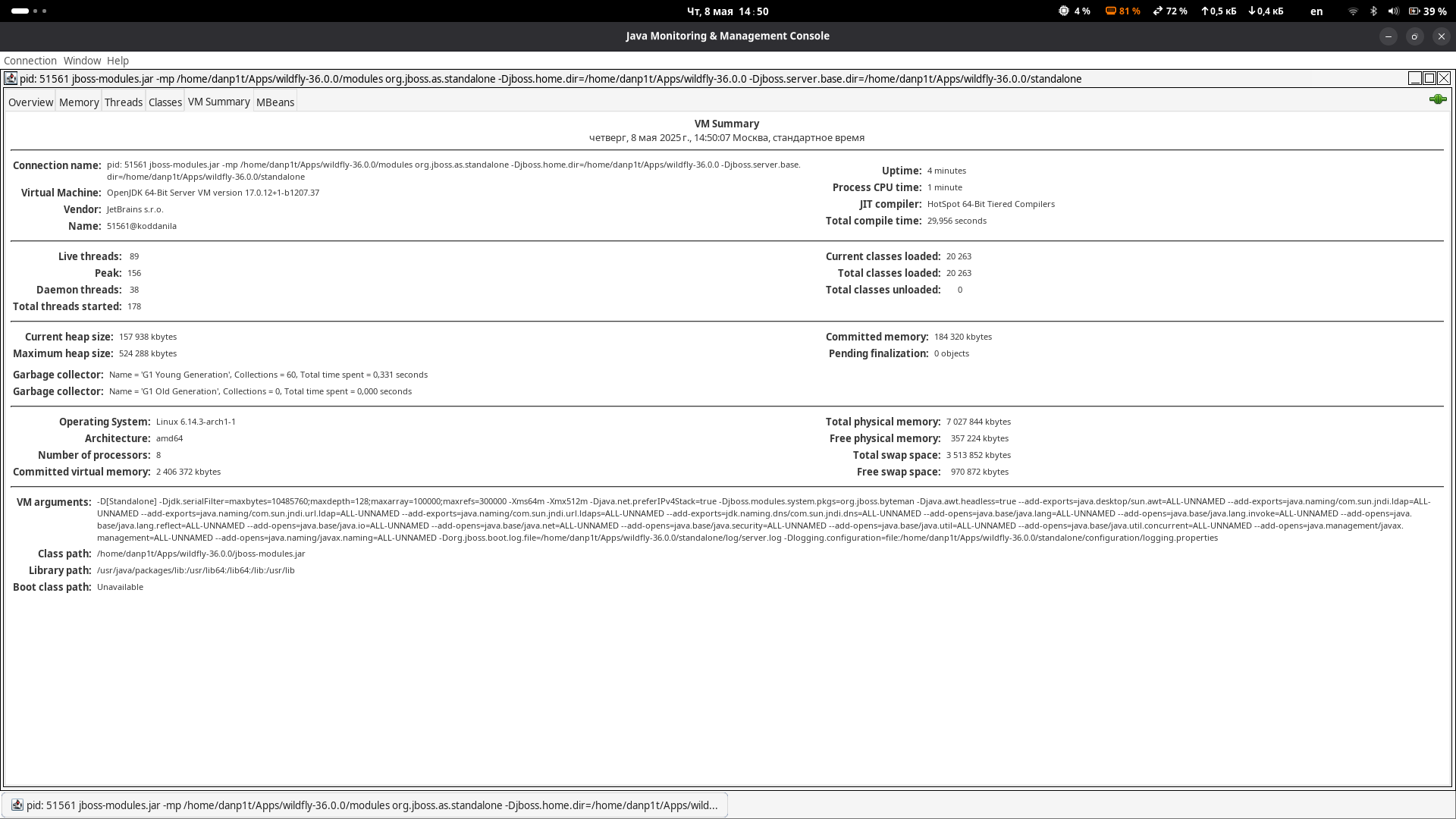










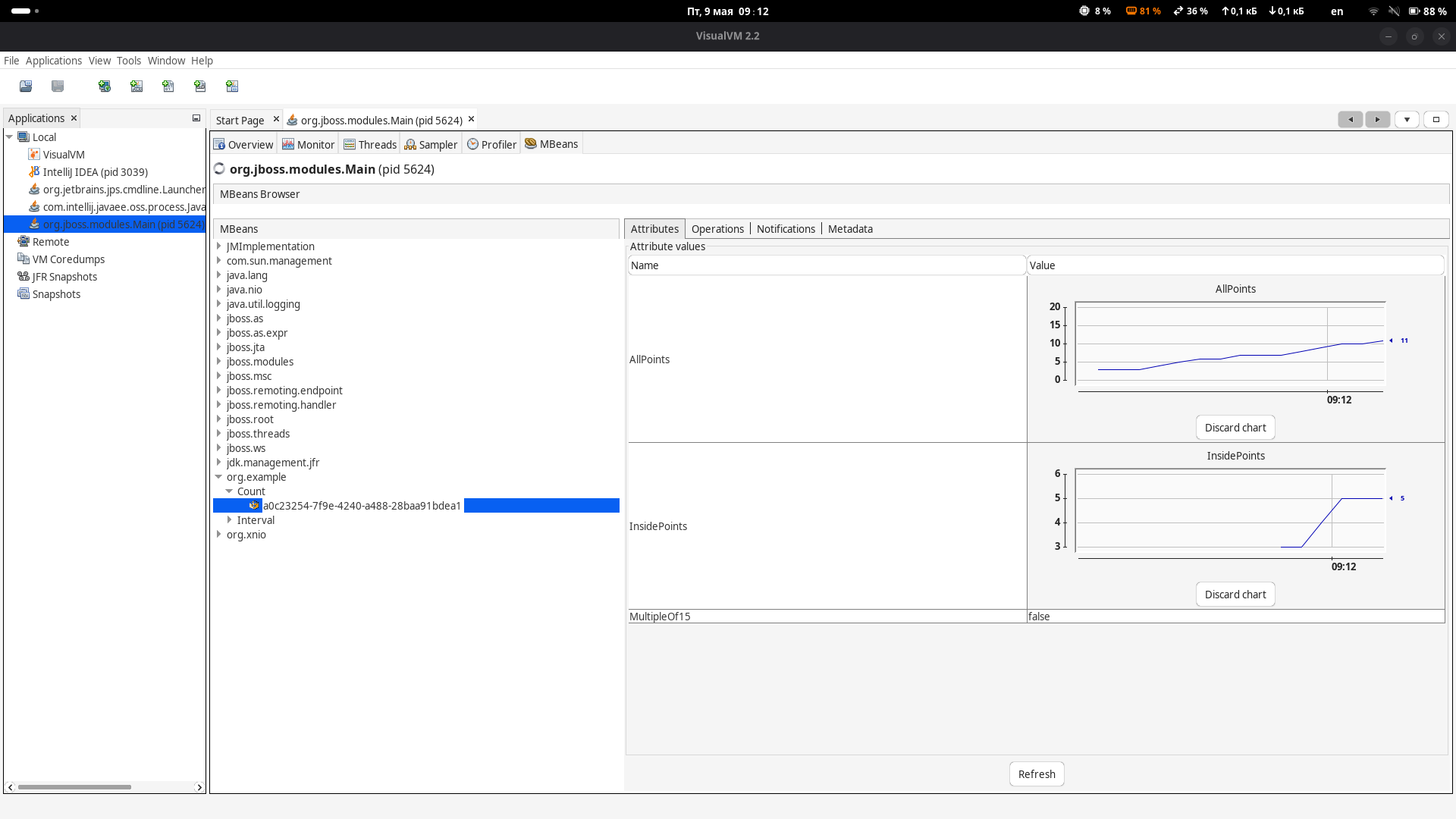
Благодаря утилите Jconsole мы можем увидеть информацию о Mbean, в частности, название класса, конструкторы и параметры конструкторов, также можем посмотреть значение полей Mbean, можем их изменять из утилиты. Также можем увидеть наименование и версию JVM

У меня наименование и версия JVM: OpenJDK-64-Bit Server VM version 17.0.12+1-b1207.37

Поставшик виртутальной машины Java: JetBrains s.r.o

## Скриншоты программы VisualVM со снятыми показаниями, выводы по результатам профилирования.

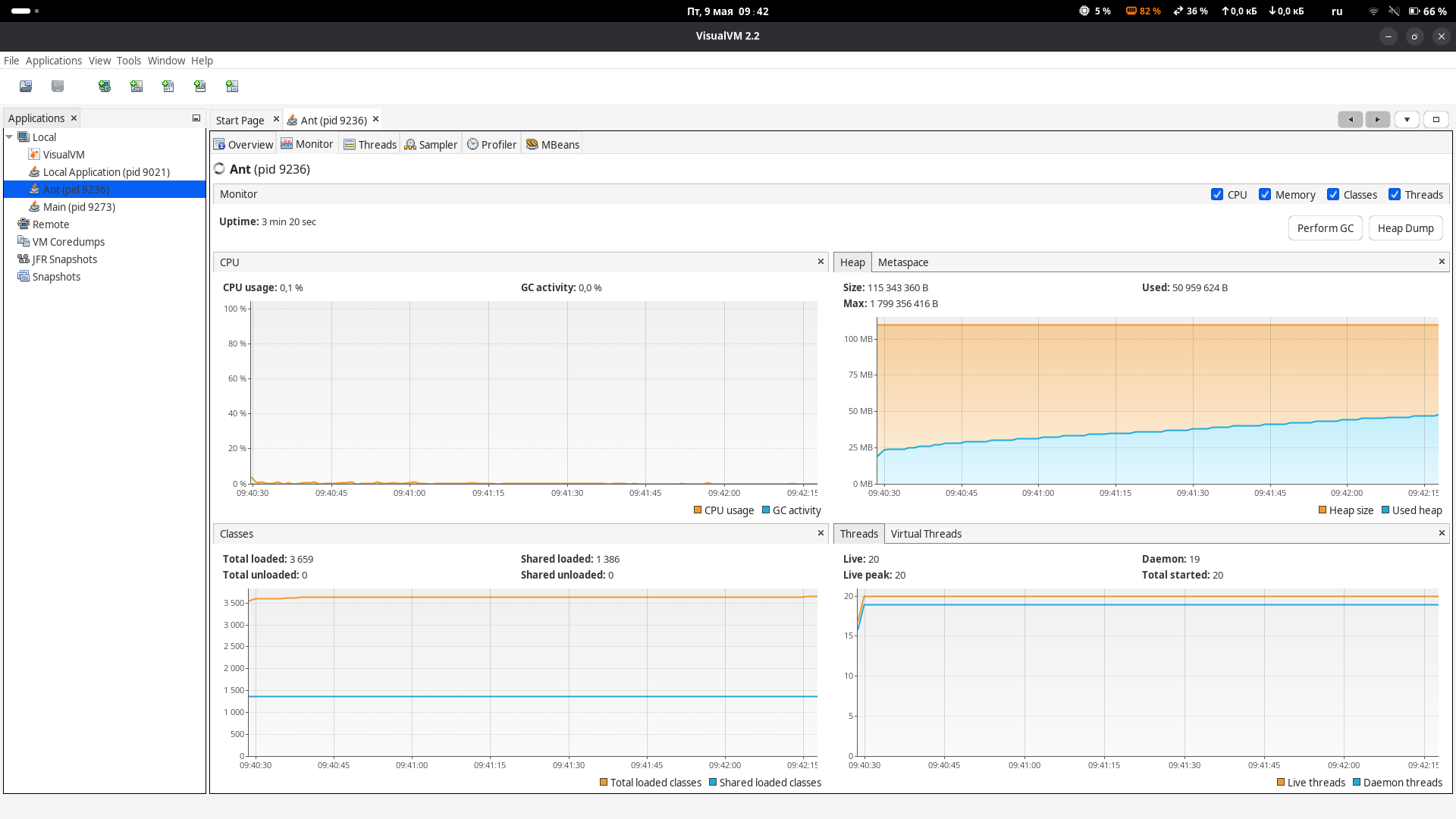
Снять график изменения показаний MBean-классов, разработанных в ходе выполнения задания 1, с течением времени:

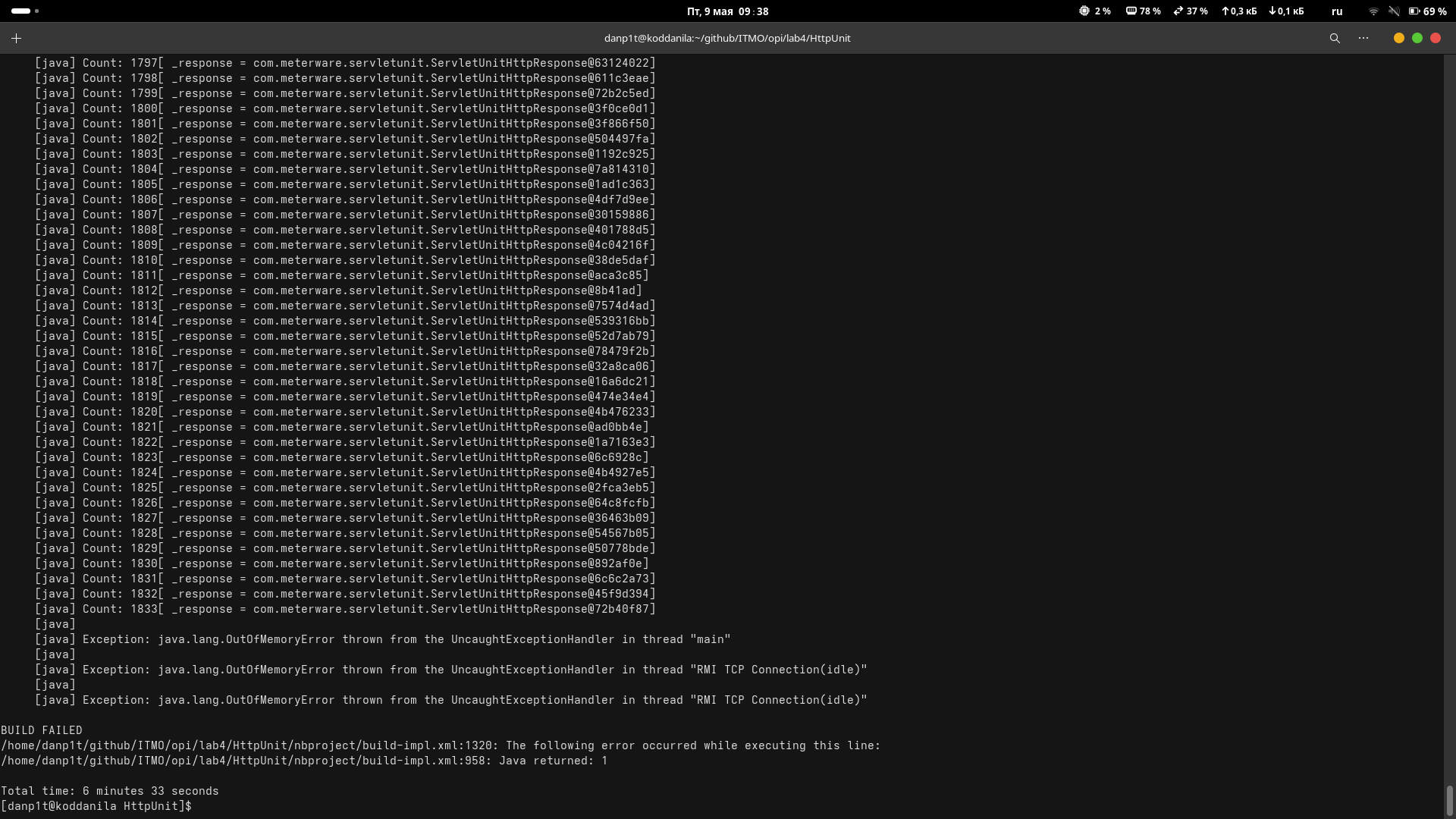


Определить имя потока, потребляющего наибольший процент времени CPU:

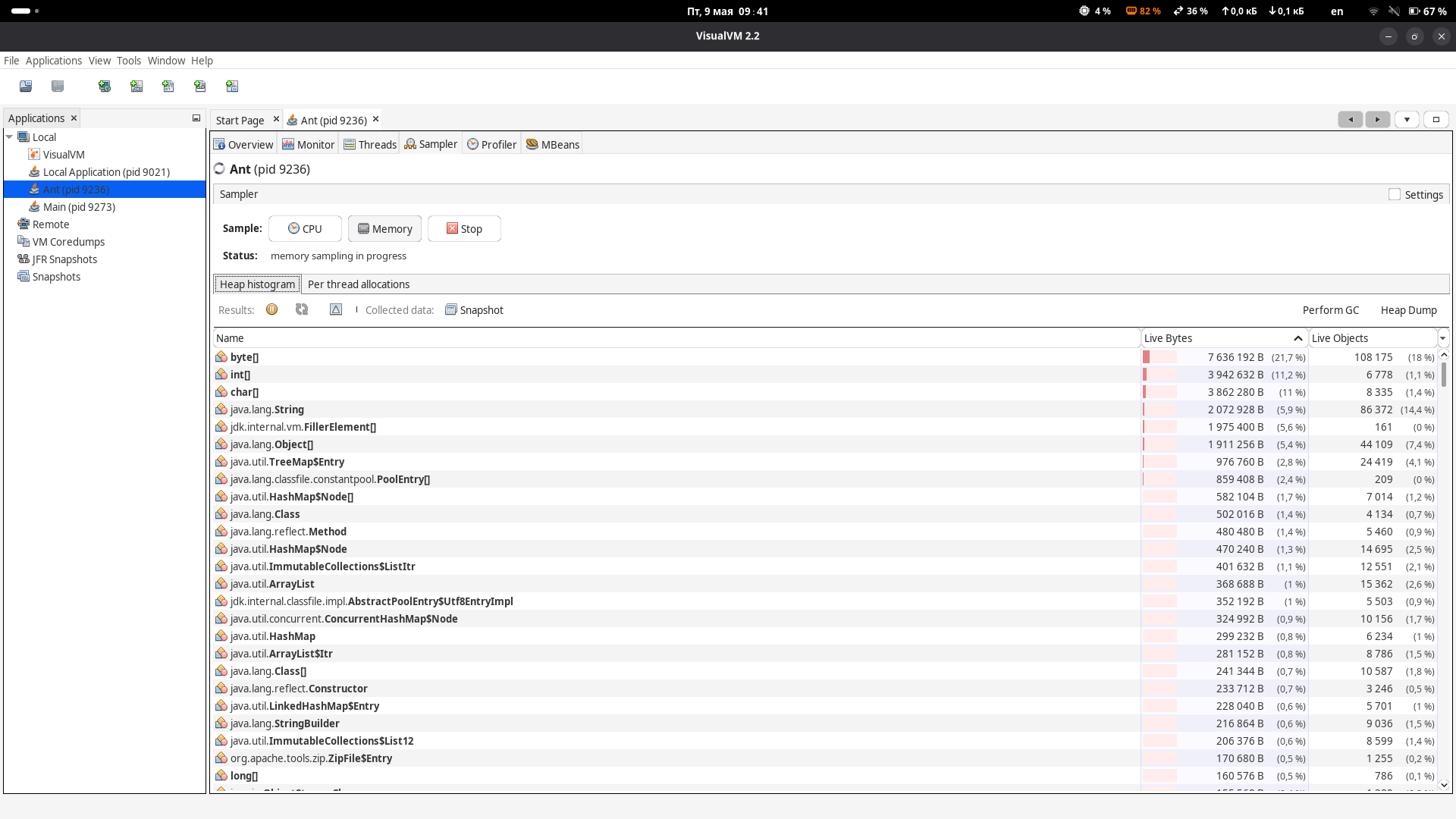
## 

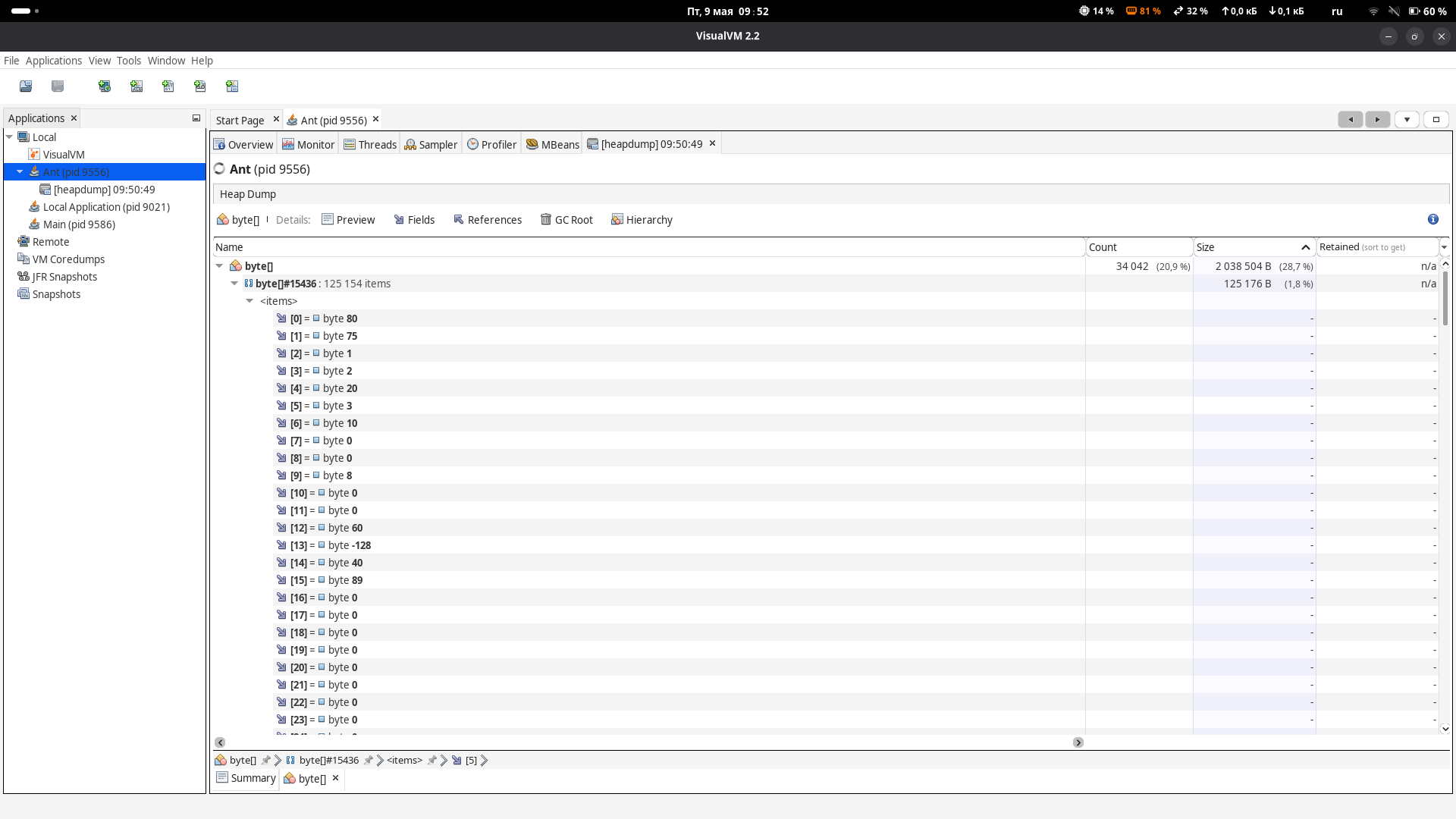
## Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.

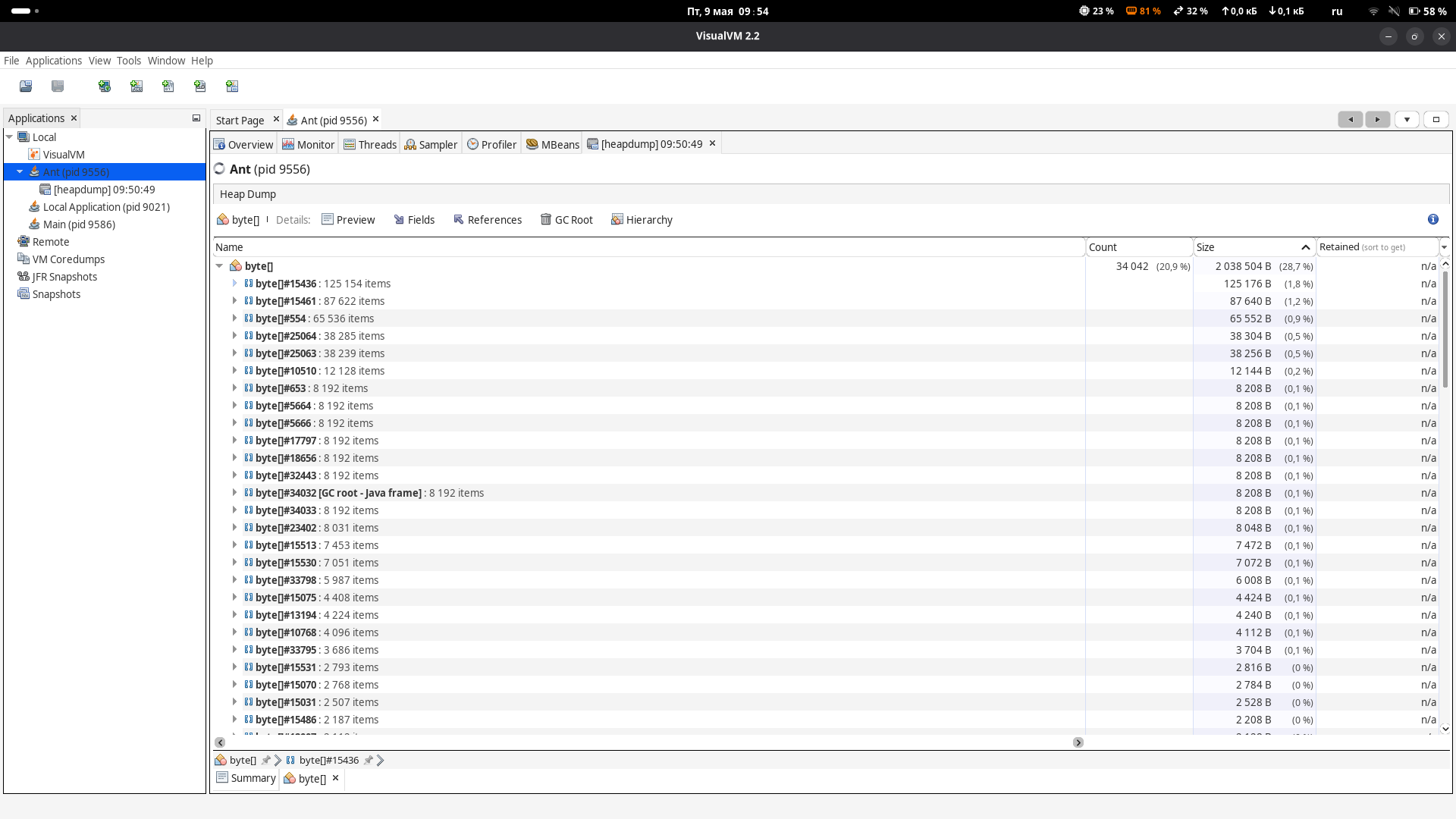


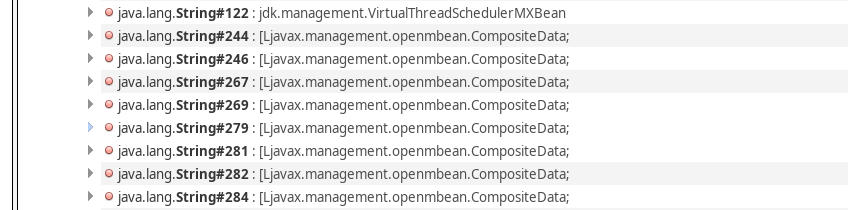


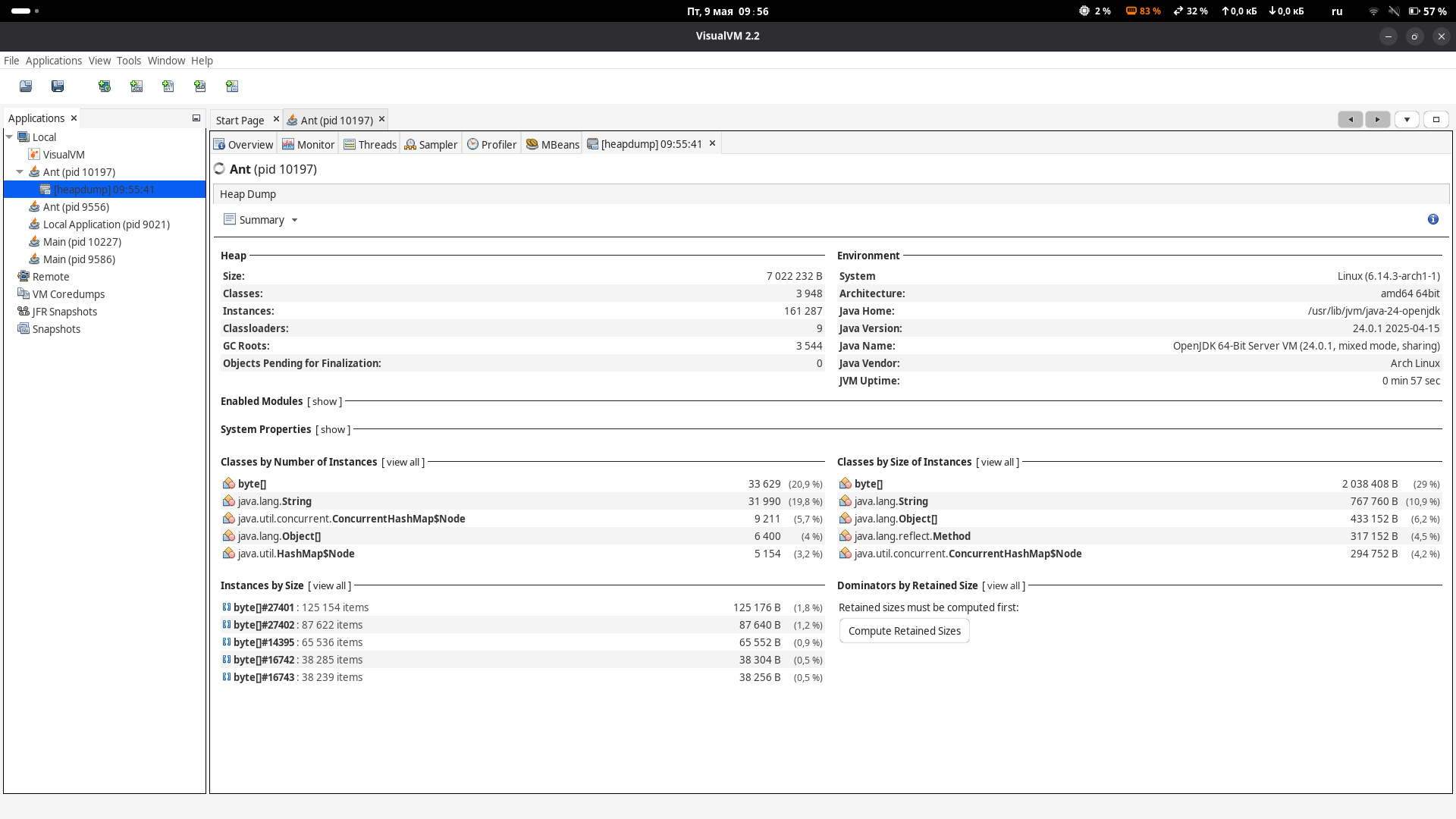
По графику и скриншоту видно, что у нас происходит утечка памяти. Перейдем в вкладку Sampler для просмотра в каком конкретно месте у нас происходит это.

Как видно из скриншота, что-то не так с массивом byte[]. Так как его размер постоянно увеличивается.



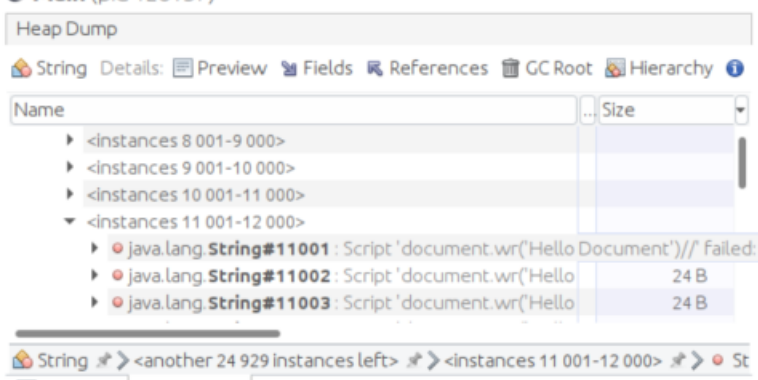
A

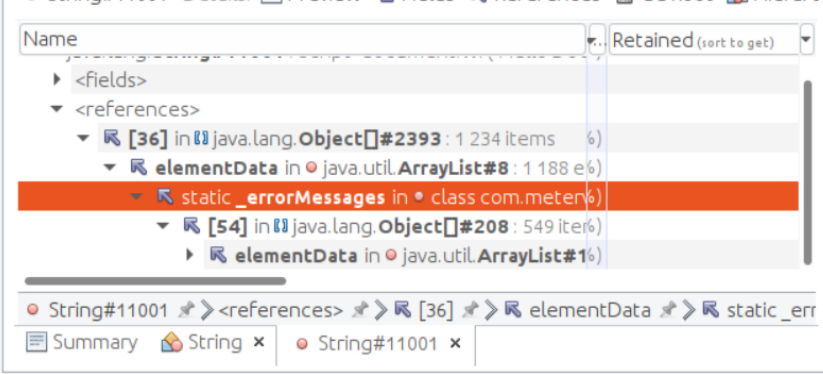


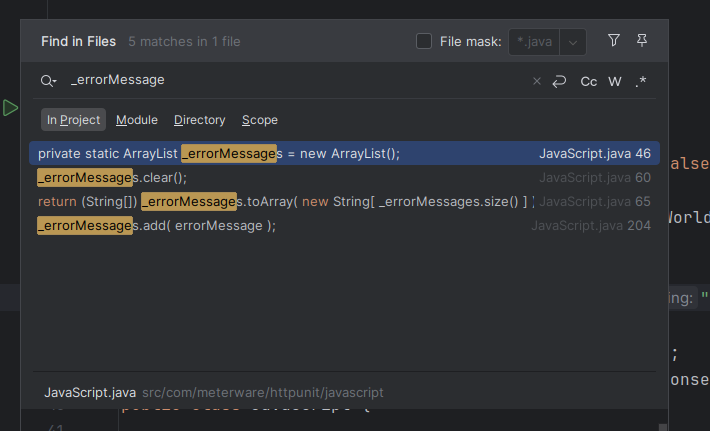


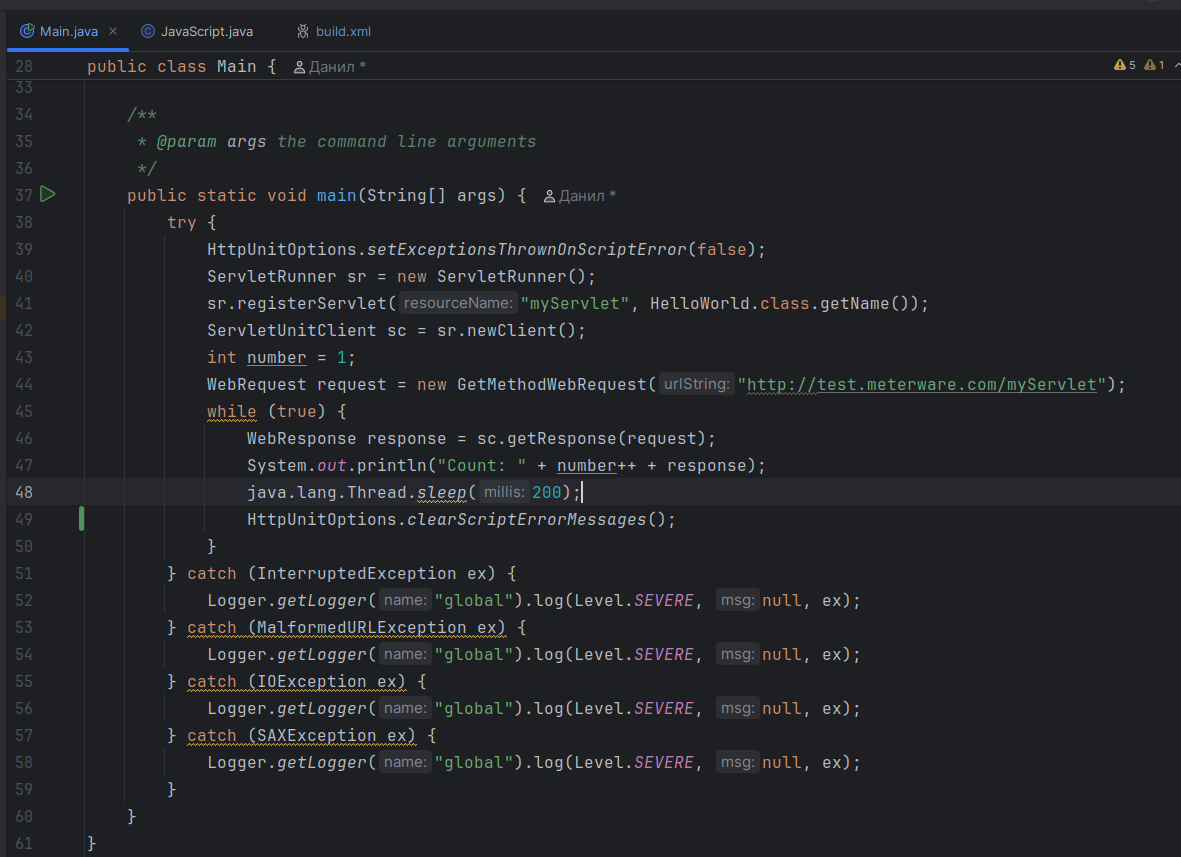
Можем заметить, что создается очень много String и массивов byte и потом их Garbage Collector не убивает, тем самым они остаются в памяти и еще они создаются каждую итерацию. Что приводит к ошибке OutOfMemoryError

Найдем повторяющиеся строчки:



Найдем где они создаются.





## Выводы по работе.

В ходе выполнения этой лабораторной работы я очень устал… Сначала было необходимо вспомнить, что такое JSF, потом написать Bean, потом узнать, что Jconsole поддерживает только ManagedBean. Сделал костыль, который позволяет регистрировать CDI бины как ManagedBean. С Jconsole разобрался, настало время VisualVM, первое что удивило, оно не захотело запускаться на Java 24, потому что некоторые методы стали deprecated и компьютер отказывался запускать эту утилиту. С установкой старой версии Java, утилита заработала. Но тут возникла проблема в том, что не было вкладки для просмотра содержимого ManagedBean, тут моя неокрепшая нервная система запаниковала и начала консультироваться с высшим разумом для прояснения этого вопроса. Выяснилось, что необходимо установить плагин для их отображения. Всё. Бины видны, но тут возникла проблема в том, что компоненты моих бинов отображаются синим цветом, а не черным, чтобы построить для них график. В итоге я 3 часа решал этот вопрос, не решил и лег спать, на следующее утро я проснулся и понял, что у меня в интерфейсе для ManagedBean указаны сеттеры, а чтобы делать график компоненты должны быть read-only. Починил, все заработало. Настало время HttpUnit… Это монстр. Он съел всю мою память на компьютере… И моей задачей было найти место, где протикает у нас память. В ходе поисков я научился делать HeapDump, да что уж там, я нашел OQL скрипты на StackOverFlow для обнаружения строк, которые часто появляются. Одно дело их найти с помощью запроса, а другое дело в HeapDump, чтобы найти какая функция их генерирует. Около 50000 строчек String было просмотрено мной, чтобы найти их! То ли я их пропускал своим глазом, либо что, но поиск с первого раза не сработал, пришлось пересматривать… И случилось чудо, я их нашел! Далее было дело техники, найти эту функцию, понять что массив у нас статический, а это значит, он сам не умрет, надо ~~убивать~~ отищать. ­

На самом деле лаба очень устарела, как морально так и физически. Большую часть своего времени я либо делал костыли, чтобы у меня заработали утилиты, либо настраивал рабочее окружение, чтобы эти утилиты корректно работали.

Что же тогда предложить вместо этой лабы. Можно использовать Grafana, Prometheus для сбора метрик и построения дашбордов. Так студенты получат реливантый опыт, а не опыт страдания...