**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра вычислительной техники**

Основы программной инженерии

Лабораторная работа №4

Вариант 541

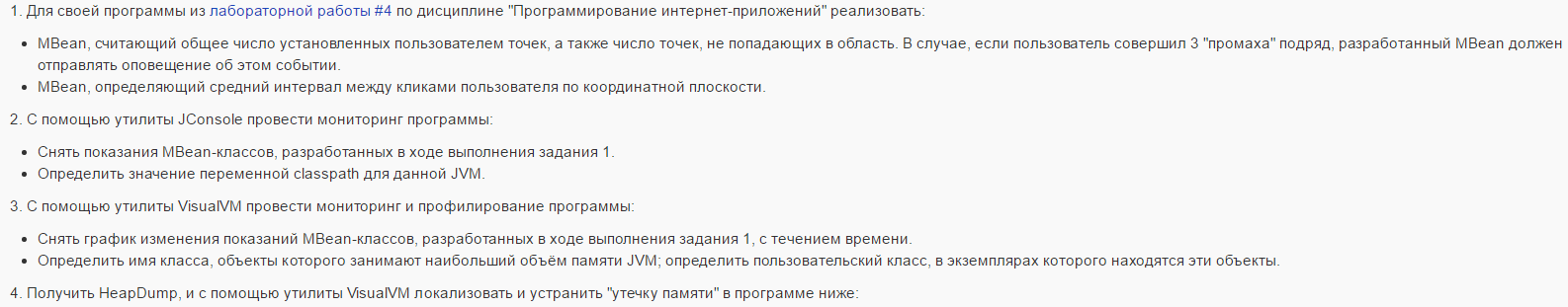
Выполнил: Гхази Даниэль

Группа P3218

Преподаватель: Харитонова А.Е.

2017 г.

1. **Текст задания**



**2. MBean**

**public class** Opi **extends** NotificationBroadcasterSupport **implements** OpiMBean {

**private int** numberOfPoints = 1;

**private int** numberOfOutOfZonePoints = 0;

**private** Integer numberOfConsecutiveOutOfZonePointsPlaced = 0;

**private long** averageTimeBetweenOnChartClicks = 0;

**private long** startTime;

**private long** stopTime = 0;

**private long** timePassedFromProgramLaunch = 0;

**private** List<Lab4.Point> points;

**private final int** MAX\_POINTS\_MISSED = 3;

**public** Opi(**int** numberOfPoints, List<Lab4.Point> points, Integer numberOfConsecutiveOutOfZonePointsPlaced){

**this**.numberOfPoints = numberOfPoints;

startTime = System.currentTimeMillis();

**this**.points = points;

**this**.numberOfConsecutiveOutOfZonePointsPlaced = numberOfConsecutiveOutOfZonePointsPlaced;

}

@Override

**public void** countNumberOfPoints() {

**this**.numberOfPoints = points.size();

}

@Override

**public int** getNumberOfPoints() {

**return** numberOfPoints;

}

@Override

**public void** countPointsOutOfZone() {

Lab4.InZoneChecker inZoneChecker = **new** Lab4.InZoneChecker();

numberOfOutOfZonePoints = 0;

**for**(Lab4.Point point : points) {

**if** (!inZoneChecker.containsPoint(point)) numberOfOutOfZonePoints++;

}

}

@Override

**public int** getPointsOutOfZone() { **return** numberOfOutOfZonePoints; }

@Override

**public void** countConsecutiveOutOfZonePointsPlaced() {

numberOfConsecutiveOutOfZonePointsPlaced = Lab4.getNumberOfConsecutiveOutOfZonePointsPlaced();

**if** (numberOfConsecutiveOutOfZonePointsPlaced == MAX\_POINTS\_MISSED){

Notification n = **new** AttributeChangeNotification(**this**,

1, System.currentTimeMillis(), **"3 consecutive misses."**,

**"NumberOfConsecutiveOutOfZonePointsPlaced"**, **"int"**,

2, numberOfConsecutiveOutOfZonePointsPlaced);

sendNotification(n);

}

}

@Override

**public void** computeAverageTimeBetweenOnChartClicks() {

stopTime = System.currentTimeMillis();

timePassedFromProgramLaunch = (stopTime - startTime) / 1000;

averageTimeBetweenOnChartClicks = timePassedFromProgramLaunch / numberOfPoints;

}

@Override

**public long** getAverageTimeBetweenOnChartClicks() {

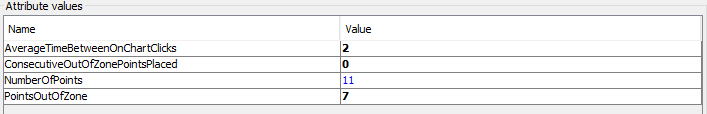
**return this**.averageTimeBetweenOnChartClicks;

}

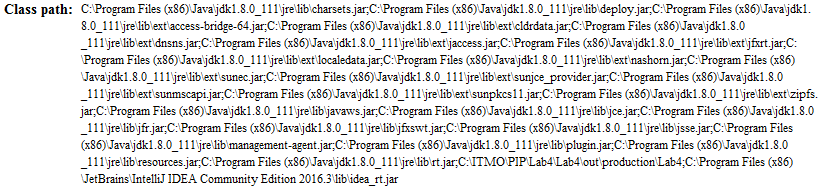
}

**3. JConsole**

**3.1** Изменения показаний.



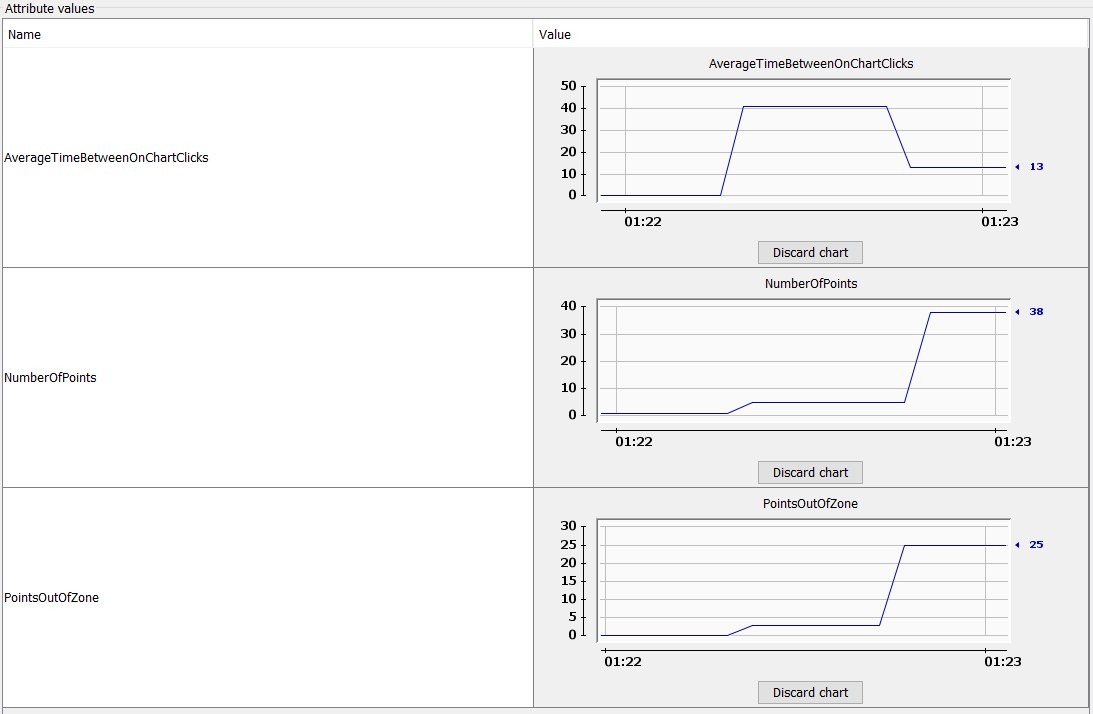
**3.2** Значение classpath.



**Выводы по результатам мониторинга:** мониторинг - это процесс замера показателей программы без выяснения причин. Я проверил корректность работы своих классов MBean, а также научился взаимодействовать с интерфейсом JConsole. Узнал, что даже в небольшой программе используется огромное количество классов.

**4. VisualVM**

**4.1** Графики изменений показаний MBean-классов.

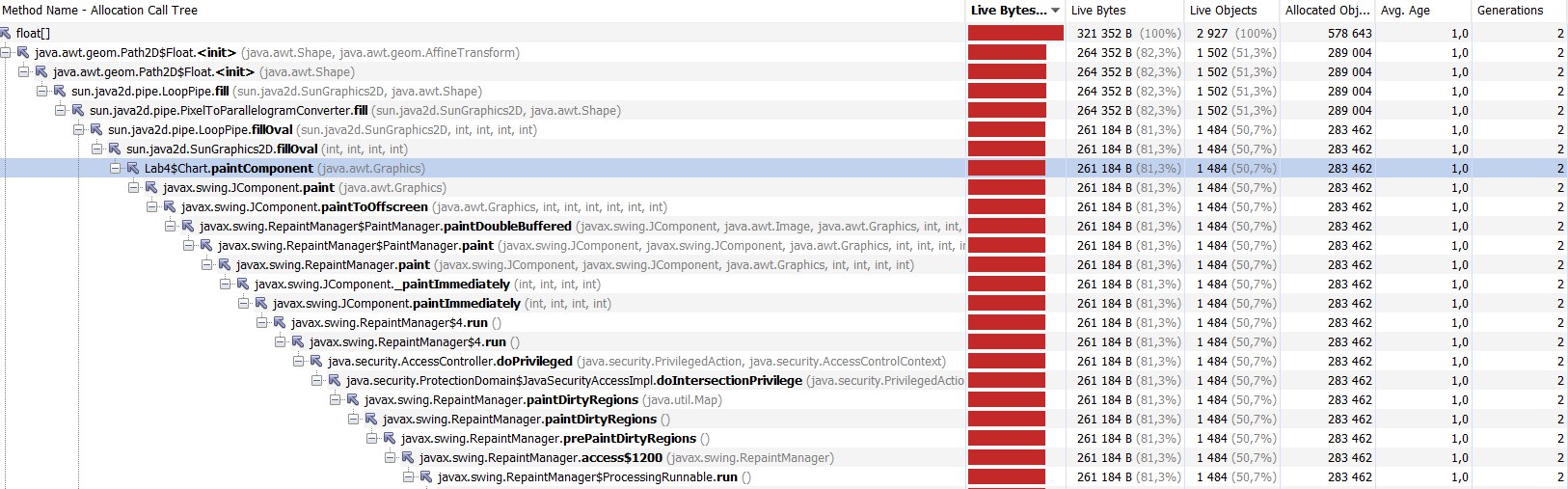


**4.2** Класс, занимающий наибольший объем и его нахождение в пользовательских классах.

Проведя анализ памяти профилировщиком, выяснилось, что больше всего памяти занимают массивы типа float.



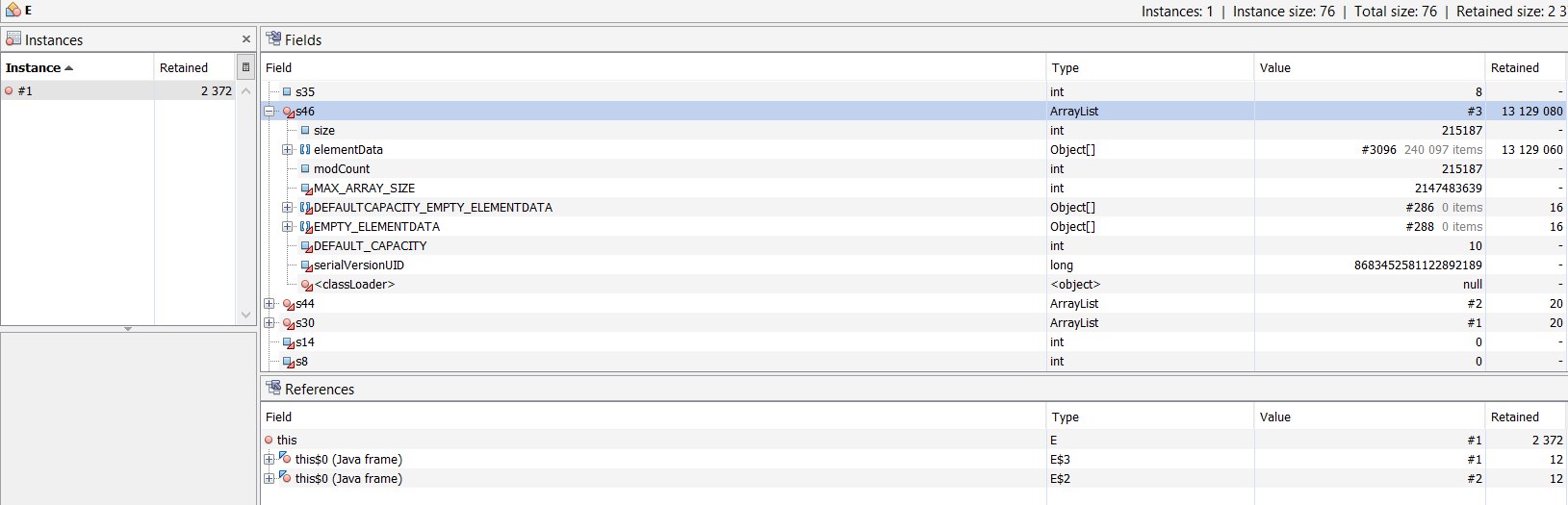
Был найден пользовательский класс, имевший к этому отношение. Это был класс Chart, который хранил List точек.



**Выводы по результатам профилирования:** профилирование - это сбор характеристик работы программы с целью ее дальнейшей оптимизации. С его помощью мне удалось выяснить наиболее нагруженное место моей программы. Полученные данные можно использовать для улучшения производительности.

**5. Поиск и устранение утечки памяти**

Я сделал 2 Snapshot’а с разницей в 12 минут и сравнил их. Выяснилось, что на больше всего со временем было выделено под массивы типа char. В Allocation Stack Traces я обнаружил метод ‘s17’ класса ‘B’, выделяющий память под новые элементы. Я создал Heap Dump в VisualVM и нашел static ArrayList s46, в который добавлялись новые элементы. Посмотрев на метод s17 в коде, я обнаружил, что он вызывается в бесконечном цикле while (true), что и приводит к постоянную добавлению новых элементов в s46.



**Вывод:**

В результате выполнения лабораторной работы мною были получены знания о мониторинге, профилировании и устранении утечек памяти. Я научился работать с такими технологиями, как MBeans, JConsole и VisualVM.   
Полученные знания можно использовать для проверки корректности работы и оптимизации программ.