Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

Основы программной инженерии

Лабораторная работа №4

“Мониторинг и Профилирование”

Выполнил: **Шкаруба Н.Е.**

Проверила: **Харитонова**

группа: **P3218**

год: **2015**

Требования

1. Для своей программы из лабораторной работы #4 по дисциплине "Программирование интернет-приложений" реализовать:

- MBean, считающий общее число установленных пользователем точек, а также число точек, попадающих в область. В случае, если количество установленных пользователем точек стало кратно 15, разработанный MBean должен отправлять оповещение об этом событии.
- MBean, определяющий средний интервал между кликами пользователя по координатной плоскости.

2. С помощью утилиты **JConsole** провести мониторинг программы:

1. Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
2. Определить значение переменной classpath для данной JVM.

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. Получить HeapDump, и с помощью утилиты **VisualVM** локализовать и устранить "утечку памяти" в указанной программе (Её смотреть в соответственном пункте выполнения ниже)

1. Исходный код разработанных MBean-классов

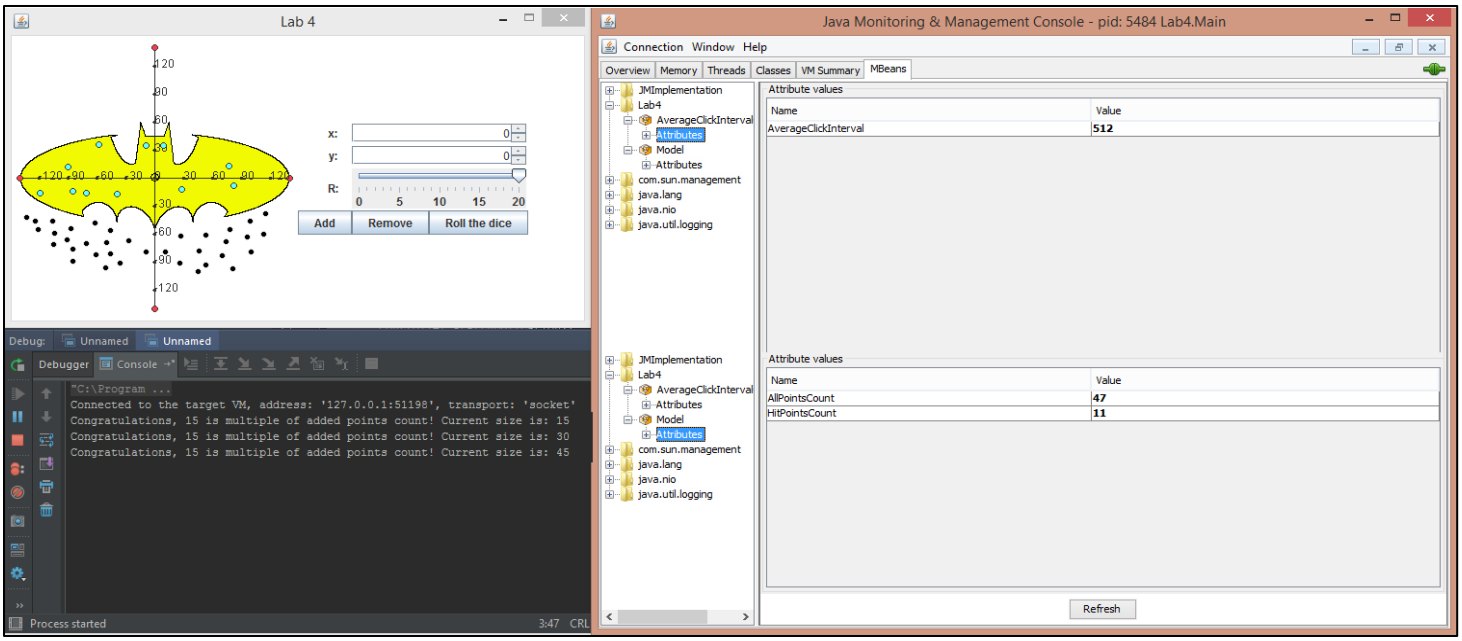
```
// MBean registration
public class Main {
    public static void main(String[] args) throws Exception {
        Controller controller = new Controller();

        MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

        mbs.registerMBean(controller.getModel(), new ObjectName("Lab4:type=Model"));
        mbs.registerMBean(controller.getAverageClickIntervalCalculator(), new
ObjectName("Lab4:type=AverageClickIntervalCalculator"));

        controller.startApplication();
    }
}
```

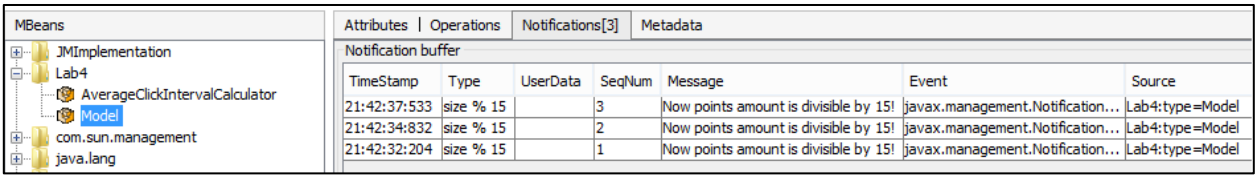
* Прошу прощения за несоответствие стиля кода стандартам. Опускание отступов и т.д присутствует ради компактности

```
// MBean 1
public interface ModelMBean extends NotificationEmitter {
    int getAllPointsCount();
    int getHitPointsCount();
}

public class Model extends NotificationBroadcasterSupport implements ModelMBean {
    private long sequenceNumber = 1;
    private Vector<Mark> marks = new Vector<>();
...
    @Override
    public int getAllPointsCount() {
        return marks.size();
    }
    @Override
    public int getHitPointsCount() {
        return (int)marks.stream().filter(mark -> mark.isHighlighted == true).count();
    }
    public void addMark(Point point) {
        boolean doContains = batFigure.Contains(new Point(point.x/R, point.y/R));
        this.marks.add(new Mark(point, doContains));
        if (marks.size() % 15 == 0)
            sendNotification(new Notification("size % 15", this, sequenceNumber++,
System.currentTimeMillis(), "Now points amount is divisible by 15!"));
    }
    public void removeLastMark() {
        if (marks.size() != 0) {
            marks.removeElementAt(marks.size()-1);
        if (marks.size() % 15 == 0)
             sendNotification(new Notification("size % 15", this, sequenceNumber++,
System.currentTimeMillis(), "Now points amount is divisible by 15!"));

    }
}
```

```
// MBean 2
public interface AverageClickIntervalCalculatorMBean extends NotificationEmitter {
    long getAverageClickInterval();
}
class AverageClickIntervalCalculator implements MouseListener,
AverageClickIntervalCalculatorMBean {
    Vector<Long> measurements = new Vector<>();
    long startTime;

    public AverageClickIntervalCalculator() {
        startTime = System.currentTimeMillis();
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        measurements.add(System.currentTimeMillis() - startTime);
    }
    @Override
    public long getAverageClickInterval() {
        long sum = 0;
        for(int i = 0; i < measurements.size()-1; i++)
            sum += measurements.get(i+1) - measurements.get(i);

        return sum / measurements.size()-1;
    }
}
```

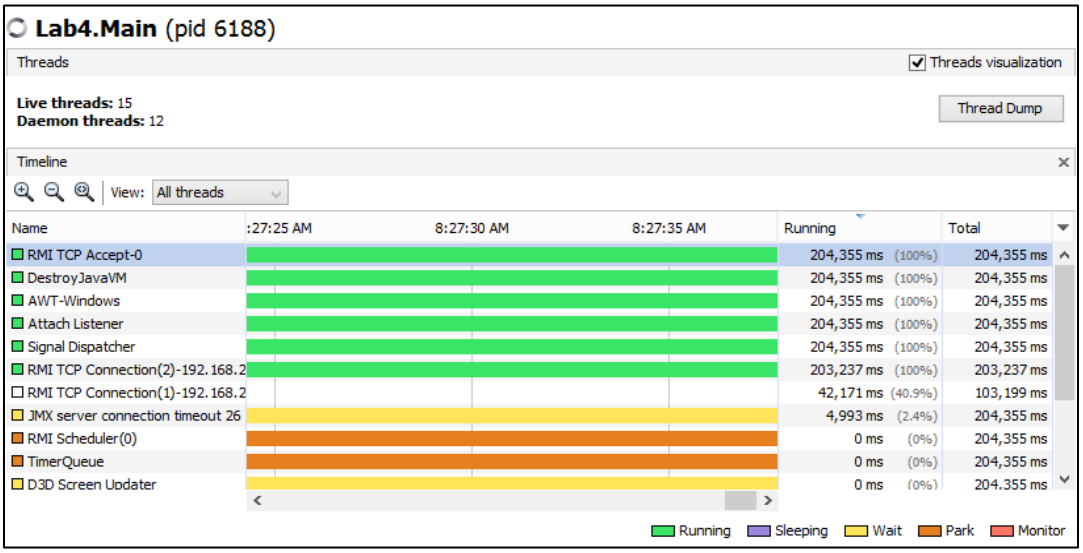
2. Мониторинг написанных MBean’ов с помощью Jconsole





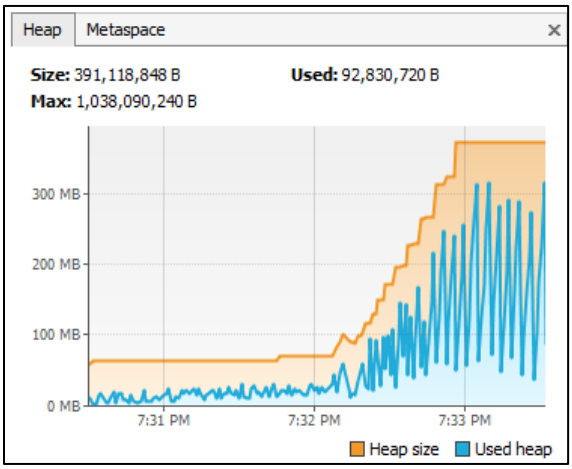
Class path: C:\Program Files\Java\jdk1.8.0_60\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\deploy.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\javaws.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\jce.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\jfr.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\jfxswt.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\jsse.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\management-agent.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\plugin.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\resources.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\rt.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\cldrdata.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\dnsns.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\jaccess.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\jfxrt.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\localedata.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\nashorn.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\sunec.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\sunjce_provider.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\sunmscapi.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\sunpkcs11.jar;C:\Program Files\Java\jdk1.8.0_60\jre\lib\ext\zipfs.jar;C:\Users\Nikita\SkyDrive\Documents\ITMO\Course 2\Software_engineering_basics\Lab4(Profilation)\out\production\Lab4(Profilation);C:\Program Files (x86)\JetBrains\IntelliJ IDEA Community Edition 14.1.4\lib\idea_rt.jar

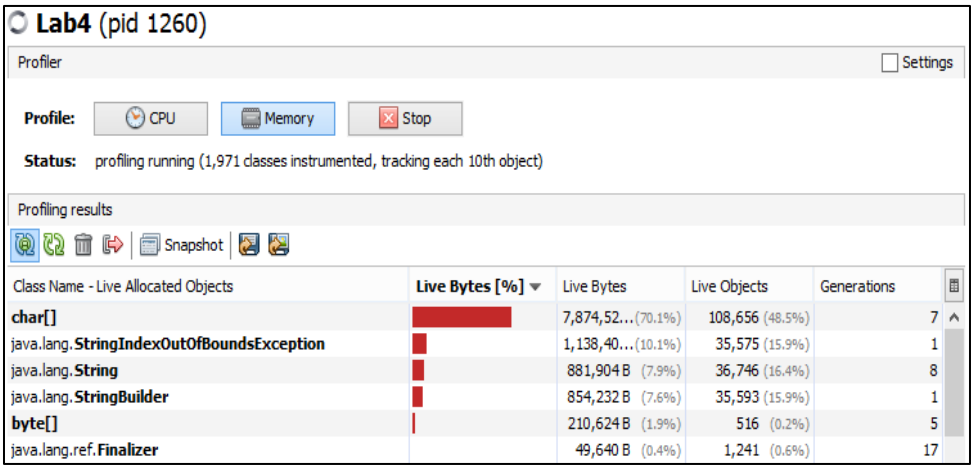
3. Профилирование написанных Bean’ов с VisualVM



4. Поиск утечки памяти с помощью VisualVM

a) Бегло смотрим код, запускаем профайлер в помощь





/* Куча постепенно расширяется, пока места попросту не остаётся, что указывает на утечку памяти. Профайлер показывает, что почти восемь миллионов байт выделено на массивы символов, а также генерируется достаточно **StringIndexOutOfBoundsException** */

b) Запускаем в любом java debugger'е данную программу(Я воспользуюсь встроенным в Intellij IDEA), ловим исключение **StringIndexOutOfBoundsException**

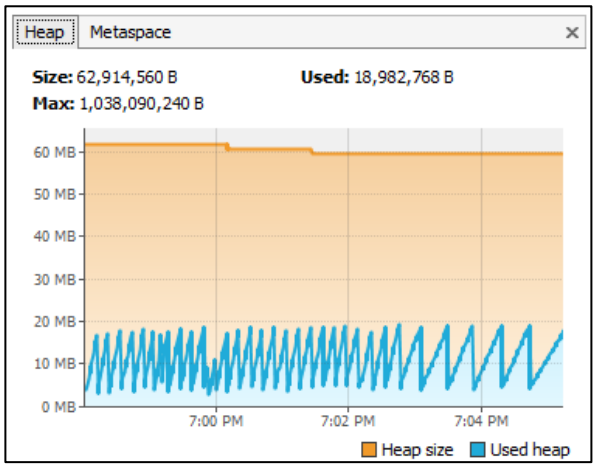
c) Обнаружил его в строках ниже

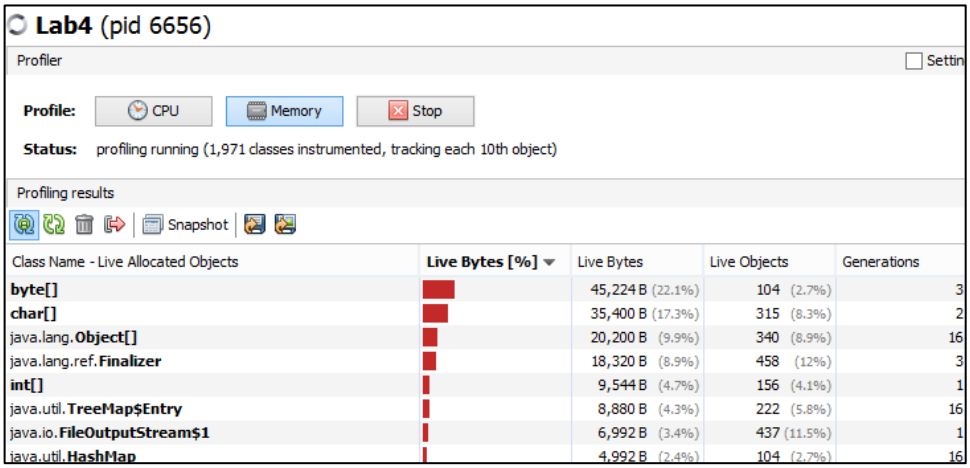
```
public void m25() {
    new Thread(new Runnable() {
        public void run() {
            int i = 0;
            while(true) {
                i++;
                try {
                    synchronized(stack) {
                        stack = stack.substring(0, stack.length() - 1); // It's him(!)
                        Thread.sleep(6);
                    }
                } catch(Exception e) { }
            }
        }
    }).start();
}
```

d) Исправляем код

```
synchronized(stack) {
    if (stack.length() >= 1)  // fix
        stack = stack.substring(0, stack.length() - 1);
    Thread.sleep(6);
}
```

e) Проверяем результат





// Утечки памяти не наблюдается, что не может не радовать!

5. Вывод

Данная лабораторная работа, с моей точки зрения, расширит кругозор студентов, не знакомых с профилированием приложений. Объяснит, что же такое профилирование, его суть и покажет его прикладное использование.