Санкт-Петербургский национальный исследовательский университет  
 информационных технологий, механики и оптики  
Кафедра информатики и прикладной математики

Основы программной инженерии  
Лабораторная работа №4  
“Мониторинг и Профилирование приложений ”

Выполнил: Шкаруба Н.Е.  
Проверила: Харитонова   
группа: P3218  
год: 2015

Требования

1. Для своей программы из [лабораторной работы #4](http://helios.cs.ifmo.ru/disciplines/iaps#lab4) 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)

Результат выполнения работы

1. Исходный код разработанных MBean-классов и сопутствующих 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();  
 }  
}

// Point counter MBean  
public interface ModelMBean {   
 int getAllPointsCount();  
 int getHitPointsCount();  
}

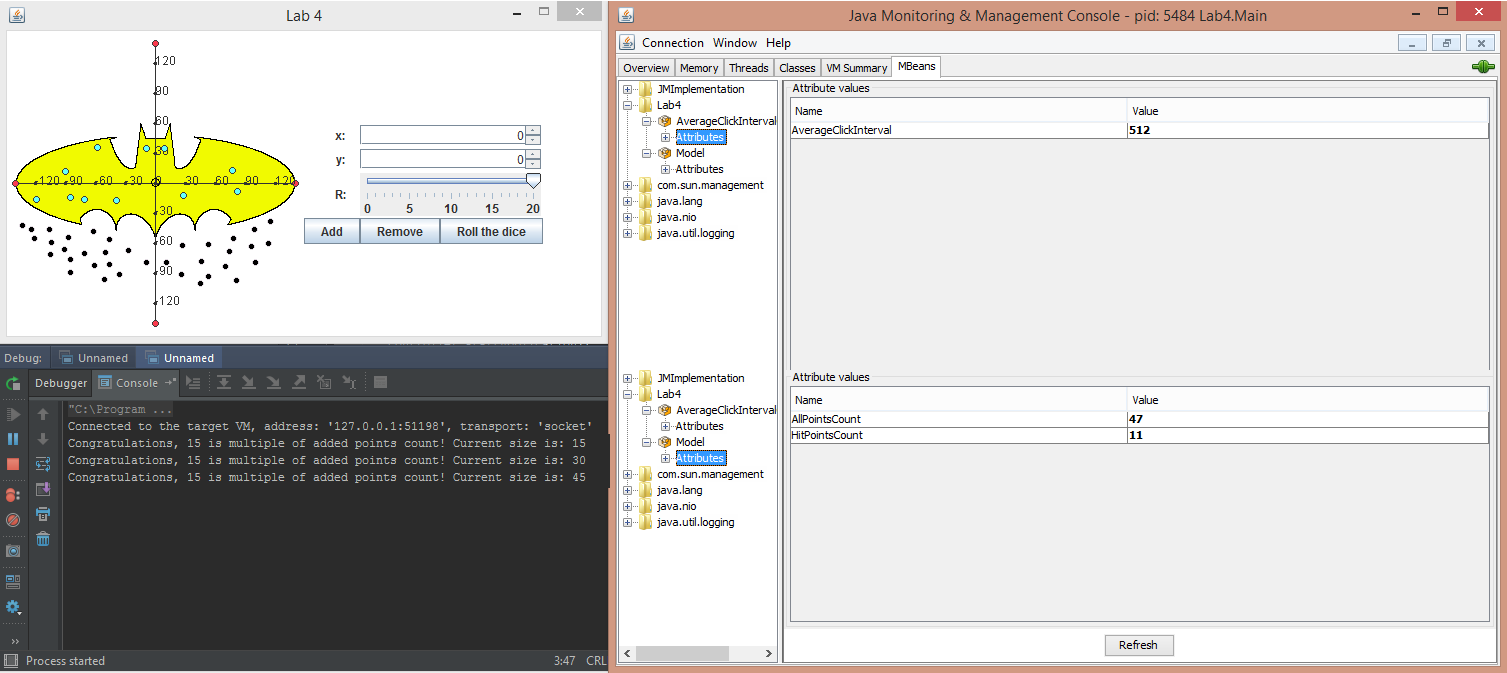
public class Model implements ModelMBean {  
 private Vector<Mark> marks = new Vector<>();

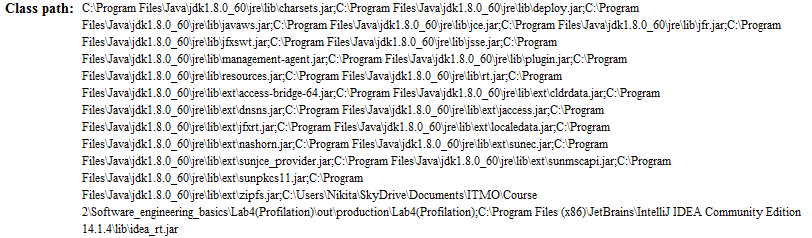
...  
  
 @Override  
 public int getAllPointsCount() {  
 return marks.size();  
 }  
 @Override  
 public int getHitPointsCount() {  
 return (int)marks.stream().filter(mark -> mark.isHighlighted == true).count();  
 }

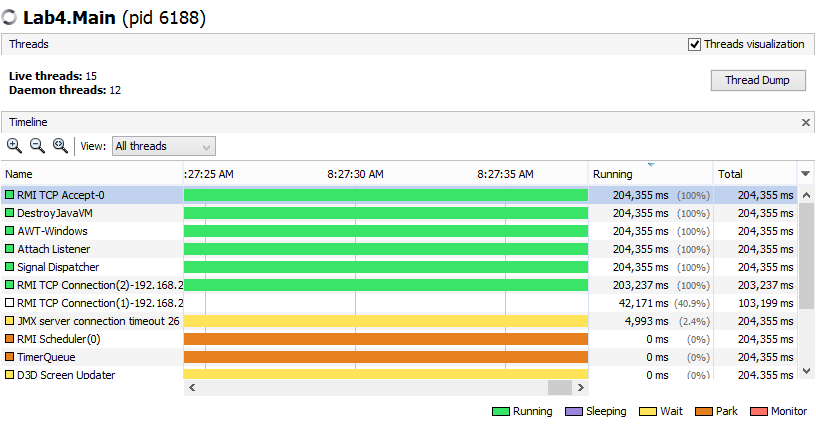
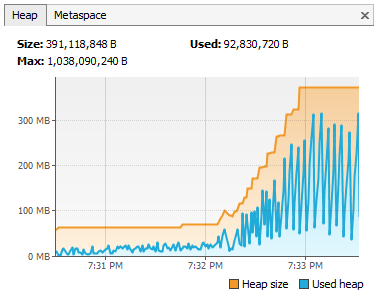
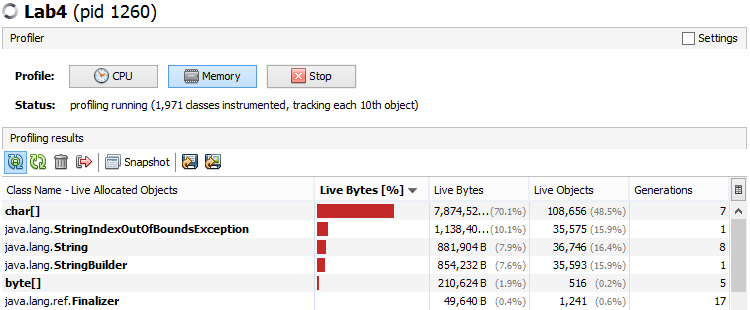
public void addMark(Point point) {  
 boolean doContains = batFigure.Contains(new Point(point.x/R, point.y/R));  
 this.marks.add(new Mark(point, doContains));  
 if (marks.size() % 15 == 0)  
 System.*out*.println("Attention! Current size is: " + marks.size());  
 }  
 public void removeLastMark() {  
 if (marks.size() != 0) {  
 marks.removeElementAt(marks.size()-1);  
 if (marks.size() % 15 == 0)  
 System.*out*.println("Attention! Current size is: " + marks.size());  
 }  
}

// MBean 2  
public interface AverageClickIntervalCalculatorMBean {  
 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;  
 }  
}

1. Мониторинг написанных MBean’ов с помощью **Jconsole**

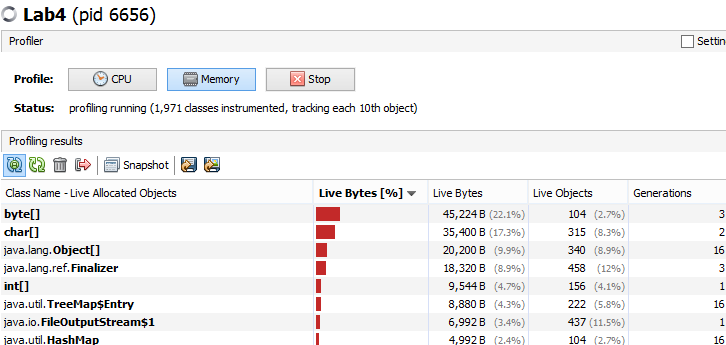
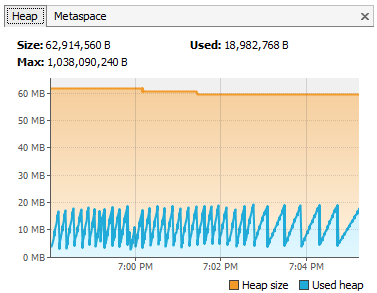


1. Профилирование написанных Bean’ов с помощью VisualVM(Ничего нового не нашёл), поиск потока, потребляющего больше всего процессорного времени.
2. Поиск утечки памяти с помощью VisualVM
   1. Бегло смотрим код, и профилируем в поисках проблемы:   
        
      Что-то не так со строками – почти восемь миллионов байт на массивы символов, а также генерируется достаточно **StringIndexOutOfBoundsException**(вряд ли это задумка автора)
   2. Запускаем в любом java debugger’е данную программу(Я воспользуюсь встроенным в Intellij IDEA), ловим исключение **StringIndexOutOfBoundsException**
   3. Я поймал его в строках ниже

public void m25() {  
 Thread t = 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) {  
 // Do nothing  
 }  
 }  
 }  
 });  
 t.start();  
}

* 1. Лечим код

synchronized(*stack*) {  
 if (stack.length() >= 1)  
 *stack* = *stack*.substring(0, *stack*.length() - 1); // It’s him(!)  
 Thread.*sleep*(6);  
}

* 1. Проверяем результат

1. Вывод по работе: