**ИТМО Кафедра Вычислительной техники**

Отчет по лабораторной работе №4

Основы программной инженерии

Вариант 522

**Выполнил: студент группы P3217**

**Плюхин Д.А.**

**2016 год**

1. **Задание к лабораторной работе**

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 координаты установленной пользователем точки вышли за пределы отображаемой области координатной плоскости, 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:

* Снять показания MBean-классов, разработанных в ходе выполнения задания 1.
* Определить количество классов, загруженных в JVM в процессе выполнения программы.

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"

1. **Исходный код разработанных MBean-классов и сопутствующих классов.**

// Файл PontoCounterMBean.java

**package management;**

**import jswing.GeneralSilhouette;**

**import jswing.Ponto;**

**import java.util.Set;**

**public interface PontoCounterMBean {**

**void pontoAdded(Ponto ponto, GeneralSilhouette gsh);**

**void checkAgain(Set<Ponto> pontos, GeneralSilhouette gsh);**

**int getTotalCount();**

**int getCaughtCount();**

**}**

// Файл PontoCounter.java

**package management;**

**import jswing.GeneralSilhouette;**

**import jswing.GraphPanel;**

**import jswing.Ponto;**

**import javax.management.Notification;**

**import javax.management.NotificationBroadcasterSupport;**

**import java.util.Set;**

**public class PontoCounter extends NotificationBroadcasterSupport implements PontoCounterMBean {**

**private int totalCount = 0;**

**private int caughtCount = 0;**

**@Override**

**public void pontoAdded(Ponto ponto, GeneralSilhouette gsh) {**

**increaseTotalCount();**

**if (gsh.checkPonto(ponto)){**

**increaseCaughtCount();**

**}**

**notificateIfNecessary(ponto,gsh.getR());**

**}**

**@Override**

**public void checkAgain(Set<Ponto> pontos, GeneralSilhouette gsh) {**

**resetCounts();**

**for(Ponto ponto : pontos){**

**pontoAdded(ponto,gsh);**

**}**

**}**

**@Override**

**public int getTotalCount() {**

**return totalCount;**

**}**

**@Override**

**public int getCaughtCount() {**

**return caughtCount;**

**}**

**private void notificateIfNecessary(Ponto ponto, double R){**

**double sizeOfArea = R\*GraphPanel.SIZE\_OF\_GRAPH/(2\*GraphPanel.GRAPHICAL\_R);**

**if (((ponto.getX() > sizeOfArea) || (ponto.getX() < -sizeOfArea)) ||**

**((ponto.getY() > sizeOfArea) || (ponto.getY() < -sizeOfArea))){**

**sendNotification(new Notification("management.pontoOutsideOfArea",this,-1,System.currentTimeMillis(),**

**"Ponto outside of the visible area"));**

**}**

**}**

**private void increaseTotalCount(){**

**totalCount+=1;**

**}**

**private void increaseCaughtCount(){**

**caughtCount+=1;**

**}**

**private void resetCounts(){**

**totalCount = 0;**

**caughtCount = 0;**

**}**

**}**

//Файл ClickMeasurerMBean.java

**package management;**

**public interface ClickMeasurerMBean {**

**void areaClicked();**

**double getAverageInterval();**

**int getCountOfClicks();**

**}**

//Файл ClickMeasurer.java

**package management;**

**public class ClickMeasurer implements ClickMeasurerMBean {**

**private int countOfClicks = 0;**

**private int averageInterval = 0;**

**private int allIntervals = 0;**

**private long lastTimeOfClick = 0;**

**@Override**

**public void areaClicked() {**

**if (lastTimeOfClick == 0){**

**lastTimeOfClick = System.currentTimeMillis();**

**return;**

**}**

**long currentTimeOfClick = System.currentTimeMillis();**

**allIntervals += (currentTimeOfClick - lastTimeOfClick);**

**countOfClicks += 1;**

**averageInterval = allIntervals /countOfClicks;**

**lastTimeOfClick = currentTimeOfClick;**

**}**

**@Override**

**public double getAverageInterval() {**

**return averageInterval;**

**}**

**@Override**

**public int getCountOfClicks() {**

**return countOfClicks;**

**}**

**}**

//Файл SilhouetteAgent.java

**package management;**

**import com.sun.jdmk.comm.HtmlAdaptorServer;**

**import jswing.GeneralSilhouette;**

**import jswing.Ponto;**

**import jswing.Silhouette;**

**import javax.management.\*;**

**import java.lang.management.ManagementFactory;**

**import java.util.Set;**

**public class SilhouetteAgent implements NotificationListener{**

**private MBeanServer mBeanServer = null;**

**public SilhouetteAgent() throws Exception{**

**mBeanServer = ManagementFactory.getPlatformMBeanServer();**

**HtmlAdaptorServer htmlAdaptorServer = new HtmlAdaptorServer();**

**ObjectName adapterName = new ObjectName("SilhouetteAgent:name=htmlAdapter,port=9092");**

**htmlAdaptorServer.setPort(9092);**

**mBeanServer.registerMBean(htmlAdaptorServer,adapterName);**

**htmlAdaptorServer.start();**

**ObjectName chiefPontoCounterName = new ObjectName("management:type=PontoCounter,name=ChiefPontoCounter");**

**PontoCounter chiefPontoCounter = new PontoCounter();**

**mBeanServer.registerMBean(chiefPontoCounter,chiefPontoCounterName);**

**ObjectName chiefClickMeasurerName = new ObjectName("management:type=ClickMeasurer,name=ChiefClickMeasurer");**

**ClickMeasurer chiefClickMeasurer = new ClickMeasurer();**

**mBeanServer.registerMBean(chiefClickMeasurer,chiefClickMeasurerName);**

**chiefPontoCounter.addNotificationListener(this,null,null);**

**}**

**@Override**

**public void handleNotification(Notification notification, Object handback) {**

**System.out.println("Receiving information...");**

**System.out.println(notification.getType());**

**System.out.println(notification.getMessage());**

**}**

**public void GraphPanelClicked() throws Exception{**

**ObjectName objectName = new ObjectName("management:type=ClickMeasurer,name=ChiefClickMeasurer");**

**mBeanServer.invoke(objectName, "areaClicked", new Object[]{}, new String[]{});**

**}**

**public void PontoAdded(Ponto ponto, GeneralSilhouette gsh) throws Exception{**

**ObjectName objectName = new ObjectName("management:type=PontoCounter,name=ChiefPontoCounter");**

**mBeanServer.invoke(objectName, "pontoAdded", new Object[]{ponto, gsh}, new String[]{Ponto.class.getName(), GeneralSilhouette.class.getName()});**

**}**

**public void RadiusChanged(Set<Ponto> pontos, GeneralSilhouette gsh) throws Exception{**

**ObjectName objectName = new ObjectName("management:type=PontoCounter,name=ChiefPontoCounter");**

**mBeanServer.invoke(objectName, "checkAgain", new Object[]{pontos, gsh}, new String[]{Set.class.getName(), GeneralSilhouette.class.getName()});**

**}**

**}**

//Файл Lab4.java (приведен не полностью)

**package jswing;**

**import …**

**public class Lab4 extends JFrame{**

**…**

**private SilhouetteAgent silhouetteAgent;**

**public static void main(String[] args) {**

**new Lab4();**

**}**

**public Lab4()**

**{**

**// Initialization**

**super();**

**try {**

**silhouetteAgent = new SilhouetteAgent();**

**}catch (Exception e){**

**e.printStackTrace();**

**}**

**…**

**}**

**…**

**private class GraphPanelMouseListener extends MouseAdapter {**

**@Override**

**public void mouseClicked(MouseEvent e) {**

**try {**

**silhouetteAgent.GraphPanelClicked();**

**} catch (Exception e1) {**

**e1.printStackTrace();**

**}**

**Ponto newPonto = new Ponto(getRealX(e.getX(),R),getRealY(e.getY(),R));**

**if (!findPonto(pontos,newPonto)) {**

**notifyAgentAboutNewPonto(newPonto);**

**pontos.add(newPonto);**

**((GraphPanel) e.getSource()).showPontoAnimated(newPonto, pontos, gsh);**

**pTextField.setText(newPonto.toString());**

**}**

**}**

**}**

**private class SpinnerChangeListener implements ChangeListener{**

**@Override**

**public void stateChanged(ChangeEvent e) {**

**theGraphPanel.paint(theGraphPanel.getGraphics());**

**R = ((Integer)((JSpinner)e.getSource()).getModel().getValue()).doubleValue();**

**notifyAgentAboutChangingRadius();**

**gsh = new GeneralSilhouette(R);**

**for (Ponto ponto : pontos){**

**theGraphPanel.showPonto(ponto,gsh);**

**}**

**}**

**}**

**private class CheckBoxListener implements ActionListener{**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**double x = Double.parseDouble(xComboBox.getModel().getSelectedItem().toString());**

**double y = Double.parseDouble(((JCheckBox)e.getSource()).getText());**

**Ponto newPonto = new Ponto(x,y);**

**if (((JCheckBox)e.getSource()).isSelected() && !findPonto(pontos,newPonto)){**

**notifyAgentAboutNewPonto(newPonto);**

**pontos.add(newPonto);**

**pTextField.setText(newPonto.toString());**

**theGraphPanel.showPontoAnimated(newPonto,pontos,gsh);**

**}**

**}**

**}**

**private class ComboBoxListener implements ActionListener{**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**boolean added = false;**

**double x = Double.parseDouble(((JComboBox)e.getSource()).getModel().getSelectedItem().toString());**

**double y = 0;**

**Ponto newPonto = new Ponto(x,y);**

**for(JCheckBox ycheckBox : yCheckBoxes) {**

**y = Double.parseDouble(ycheckBox.getText());**

**newPonto = new Ponto(x,y);**

**if (ycheckBox.isSelected() && !findPonto(pontos,newPonto)) {**

**notifyAgentAboutNewPonto(newPonto);**

**pontos.add(newPonto);**

**added = true;**

**pTextField.setText(newPonto.toString());**

**}**

**}**

**if (added){**

**theGraphPanel.showPontoAnimated(newPonto,pontos,gsh);**

**}**

**}**

**}**

**…**

**private void notifyAgentAboutNewPonto(Ponto newPonto){**

**try {**

**silhouetteAgent.PontoAdded(newPonto,new GeneralSilhouette(R));**

**} catch (Exception e1) {**

**e1.printStackTrace();**

**}**

**}**

**private void notifyAgentAboutChangingRadius(){**

**try {**

**silhouetteAgent.RadiusChanged(pontos,new GeneralSilhouette(R));**

**} catch (Exception e1) {**

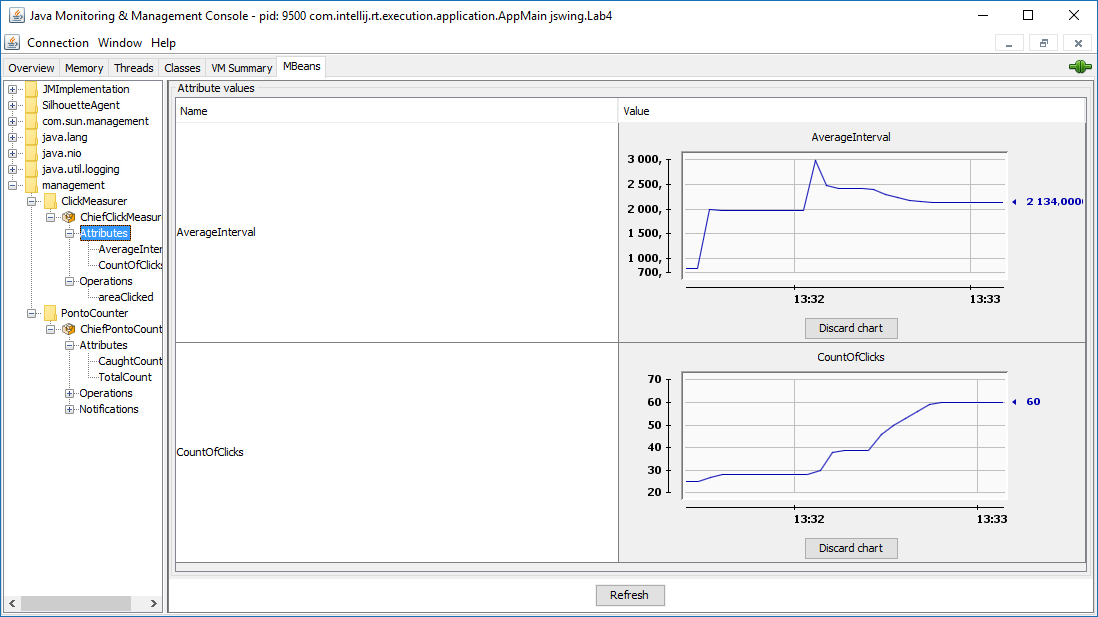
**e1.printStackTrace();**

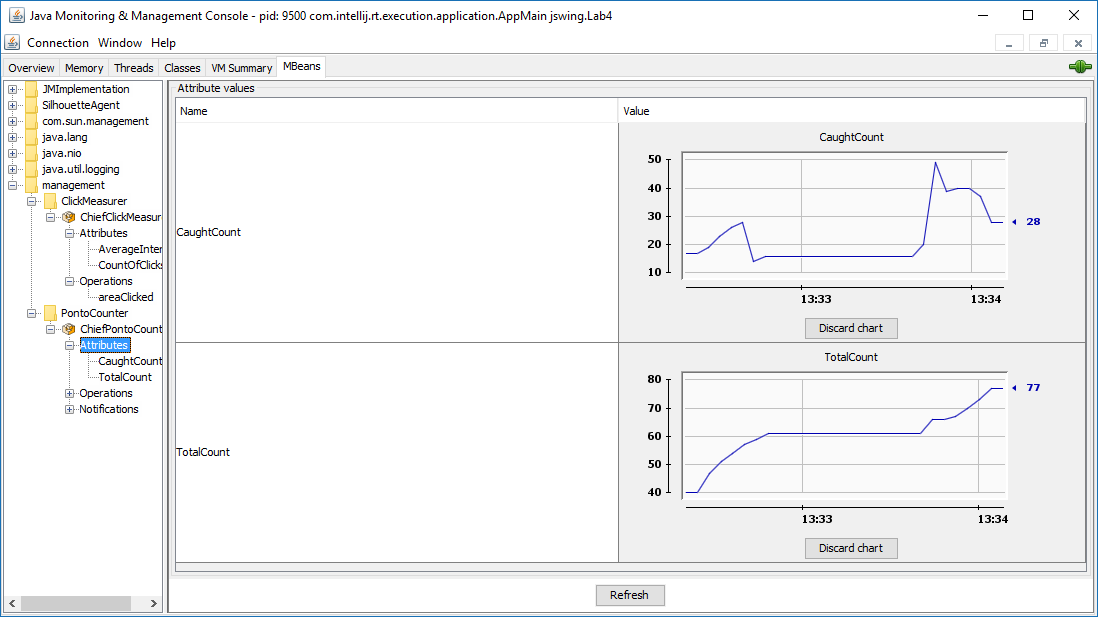
**}**

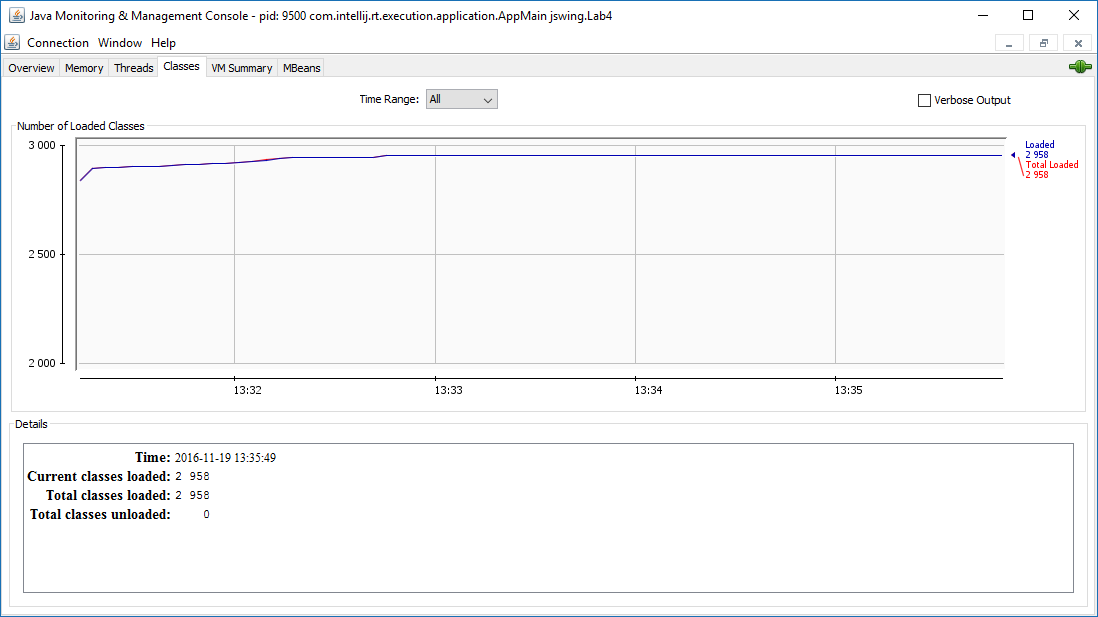
**}**

**}**

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



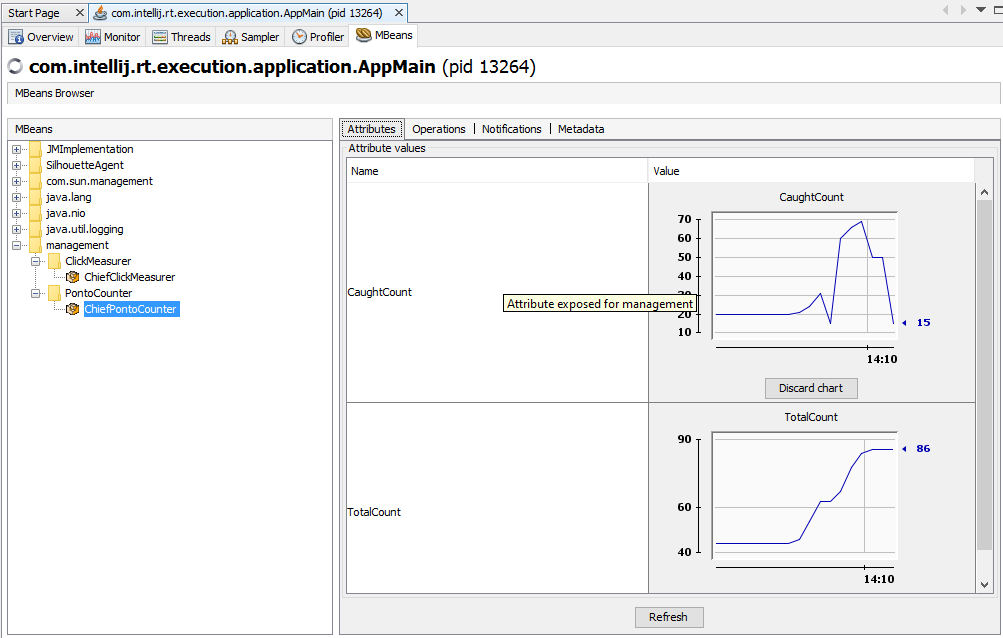


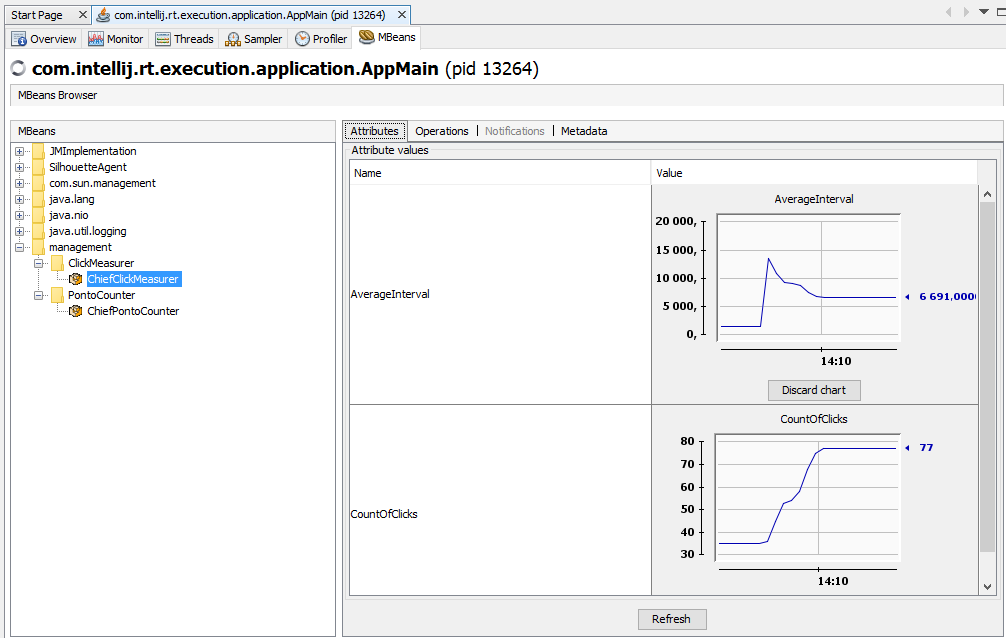


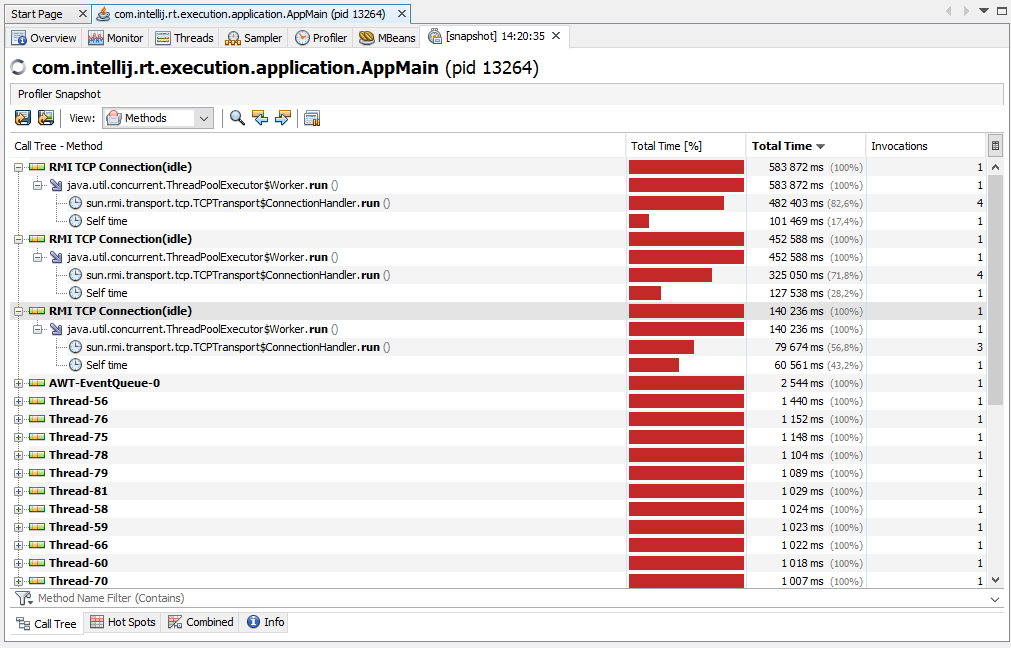
Количество классов, загруженных в результате работы программы : 2958

В результате мониторинга было обнаружено, что наибольшая нагрузка на ЦП приходится на моменты добавления серии точек в область, когда появляется необходимость показа анимации (в такие моменты используется до 9% ресурсов процессора). Также увеличение нагрузки происходит при добавлении одной точки в область с необходимостью запуска анимации (до 2%). Используемая программой память во время всего выполнения не превышает 10 Мб (в куче) и 6 Мб (за пределами кучи). Максимальное количество потоков исполнения составляет 24, среднее значение колеблется в районе 19. Так, можно сделать вывод, что программа не имеет значительных недостатков в плане производительности, хотя стоит обратить внимание на фрагменты, отвечающие за показ анимации и попробовать их оптимизировать.

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







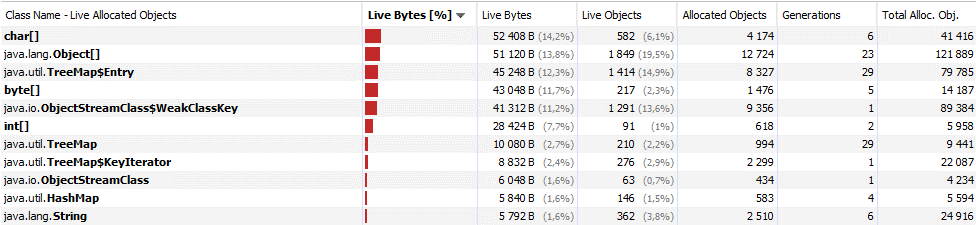
Имя потока, потребляющего наибольший процент времени CPU – RMI TCP Connection

В результате профилирования было обнаружено, что наибольшую нагрузку на ЦП оказывают методы:

* sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run() – 72,4 %
* java.util.concurrent.ThreadPoolExecutor$Worker.run() – 24,6 %
* jswing.AnimationThread.run() – 1,1 %

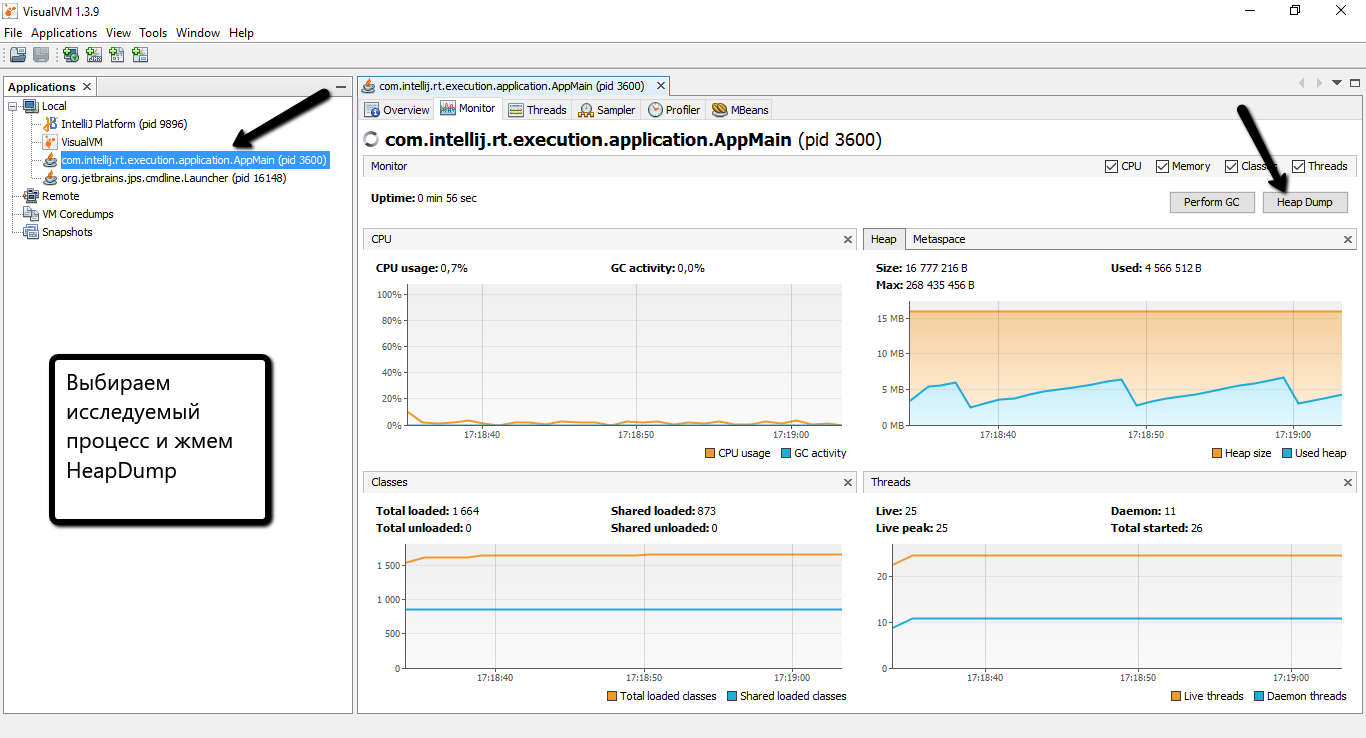
Из которых программа 4-й лабораторной работы по Программированию интернет-приложений запускает только последний метод, соответственно, в наших силах попробовать оптимизировать только его.

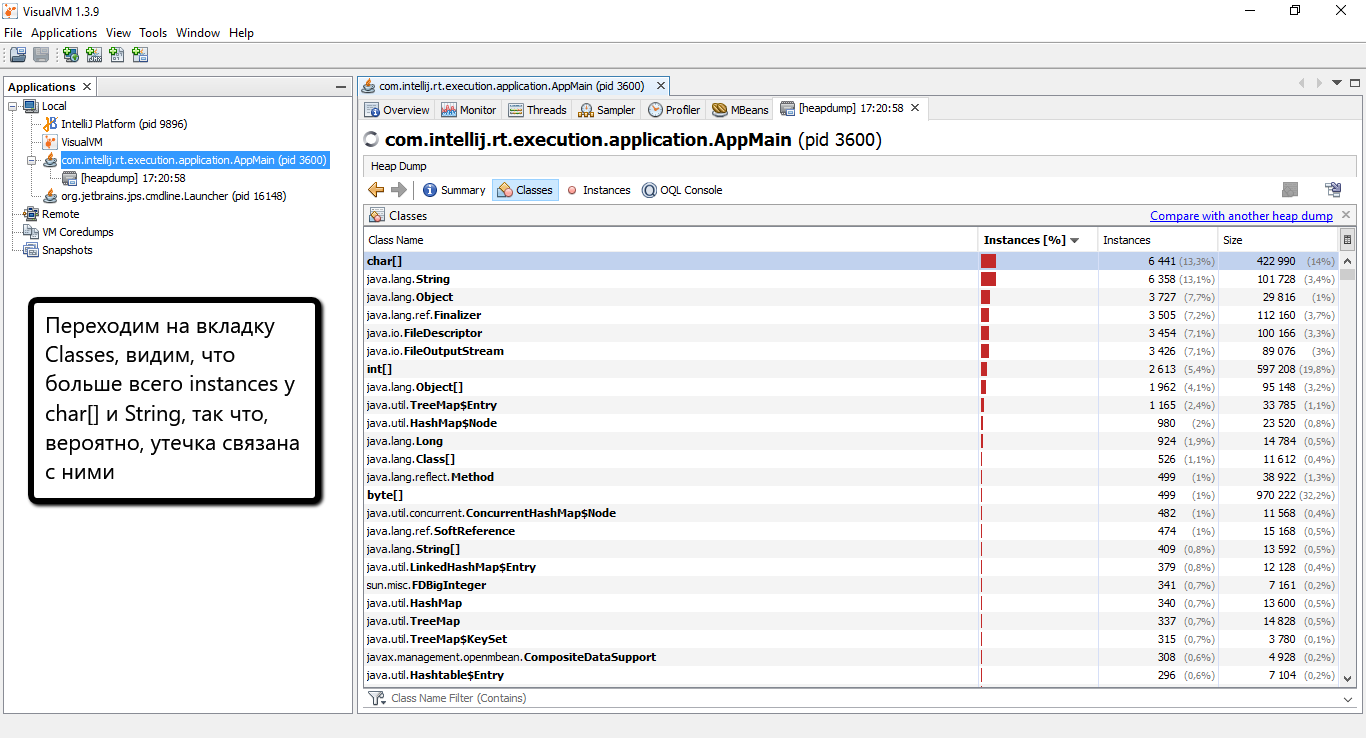
Что касается памяти, то большая её часть уходит на размещение в памяти данных следующих типов:

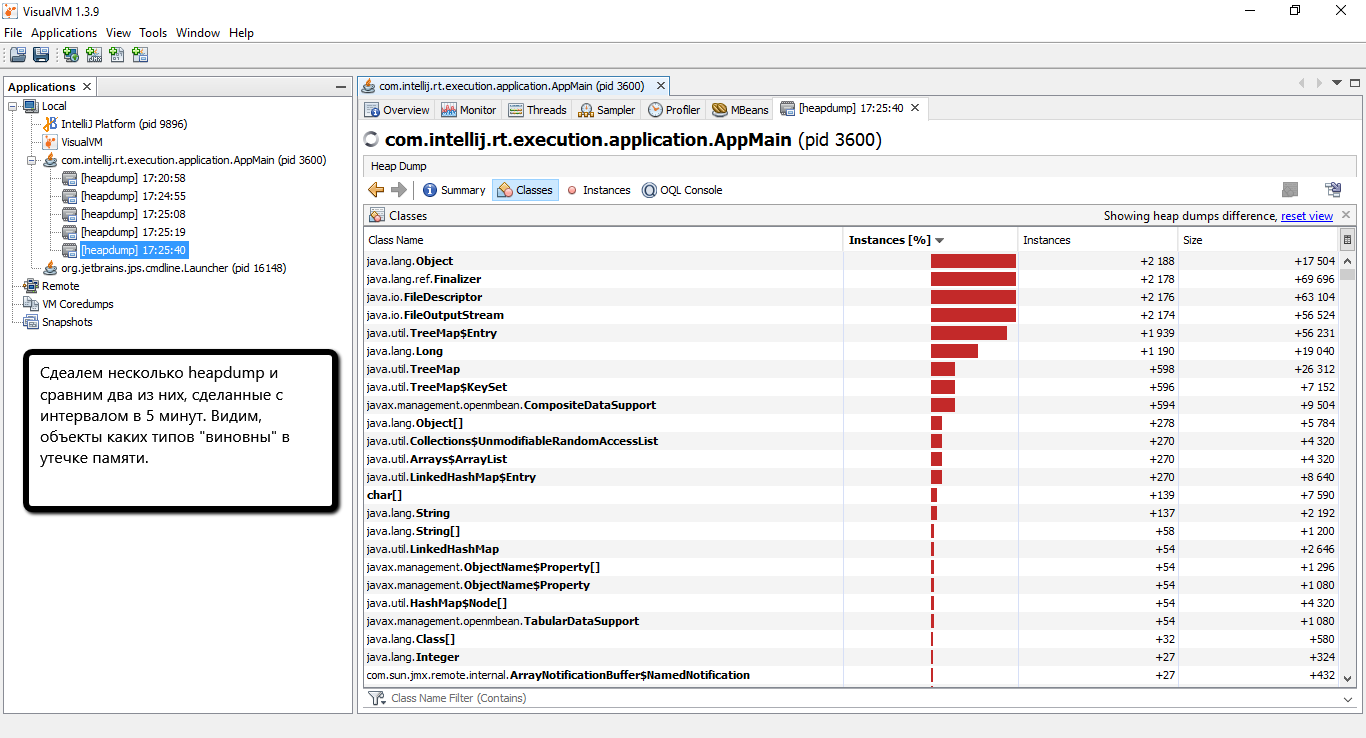


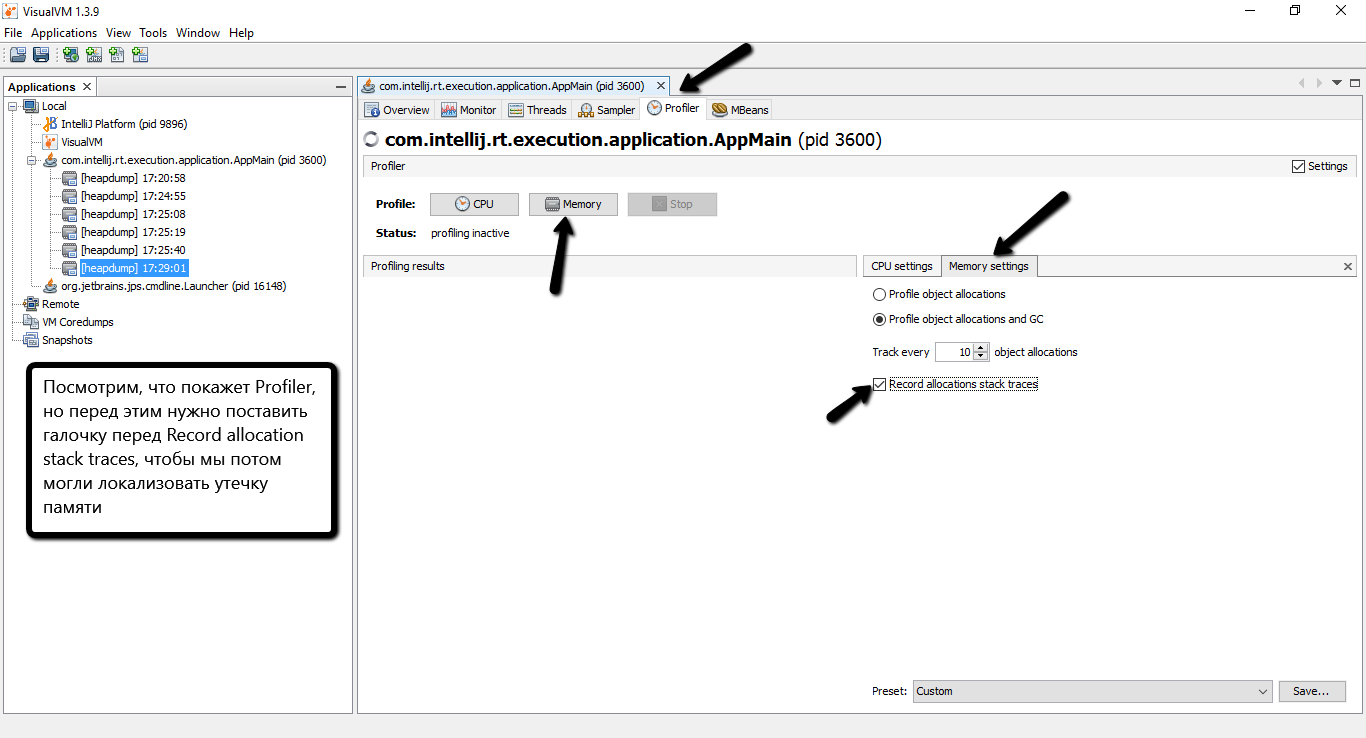
Так, хотя программа и не имеет значительных дефектов в производительности, профилирование показывает, что мы должны еще раз обратить внимание на фрагменты кода, работающие непосредственно с анимацией и попытаться их оптимизировать.

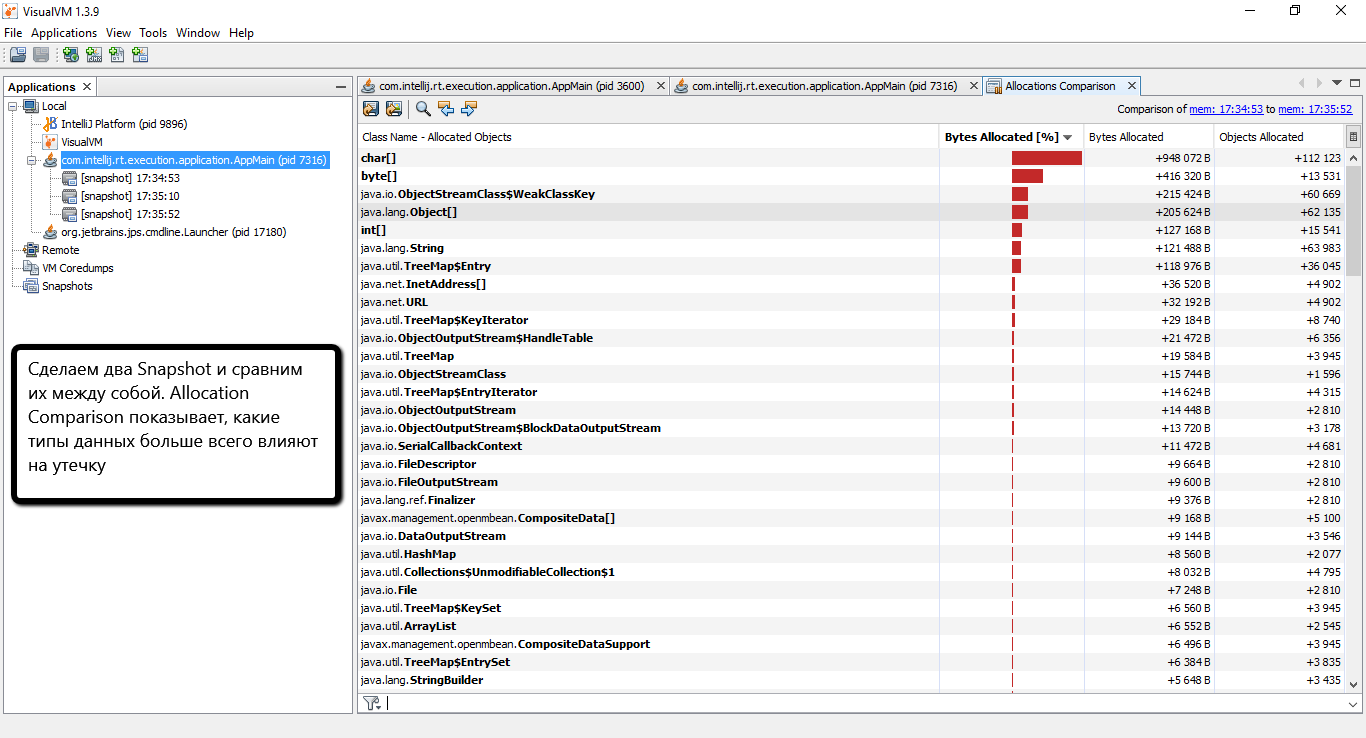
1. **Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.**

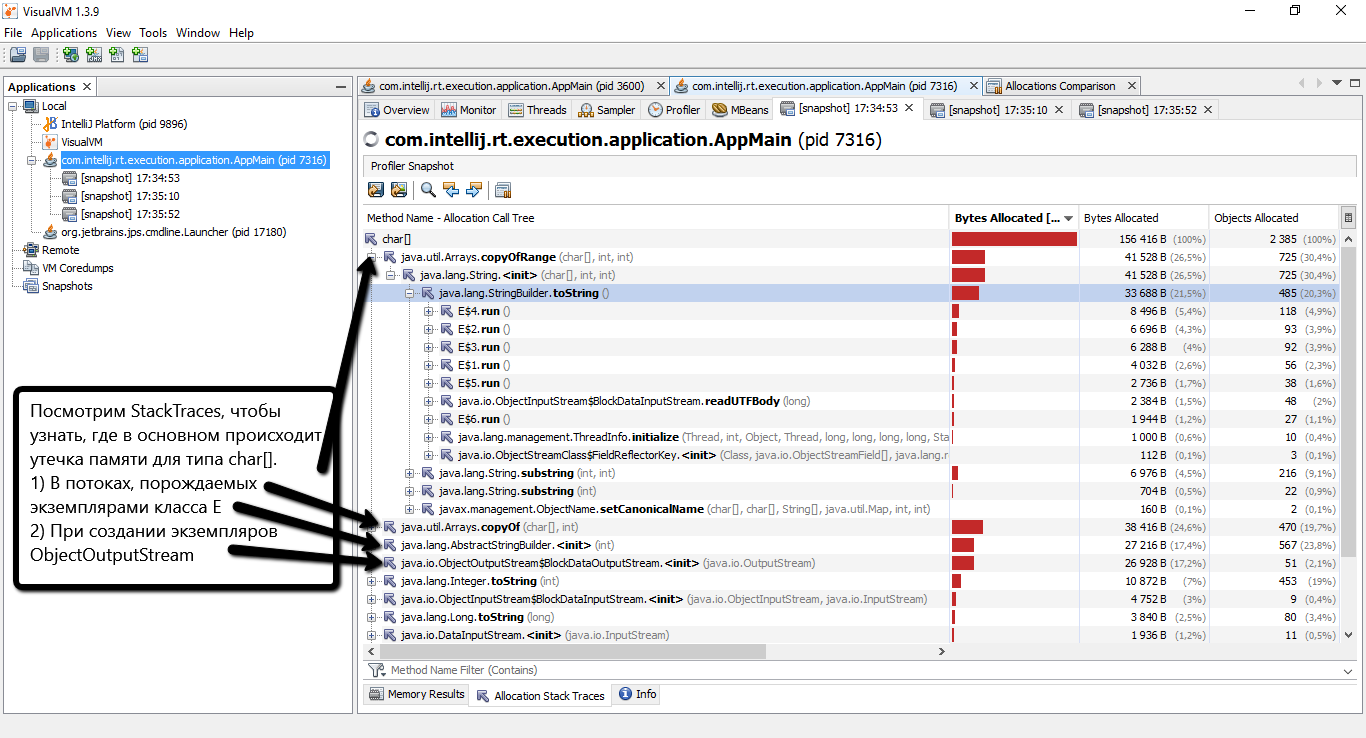


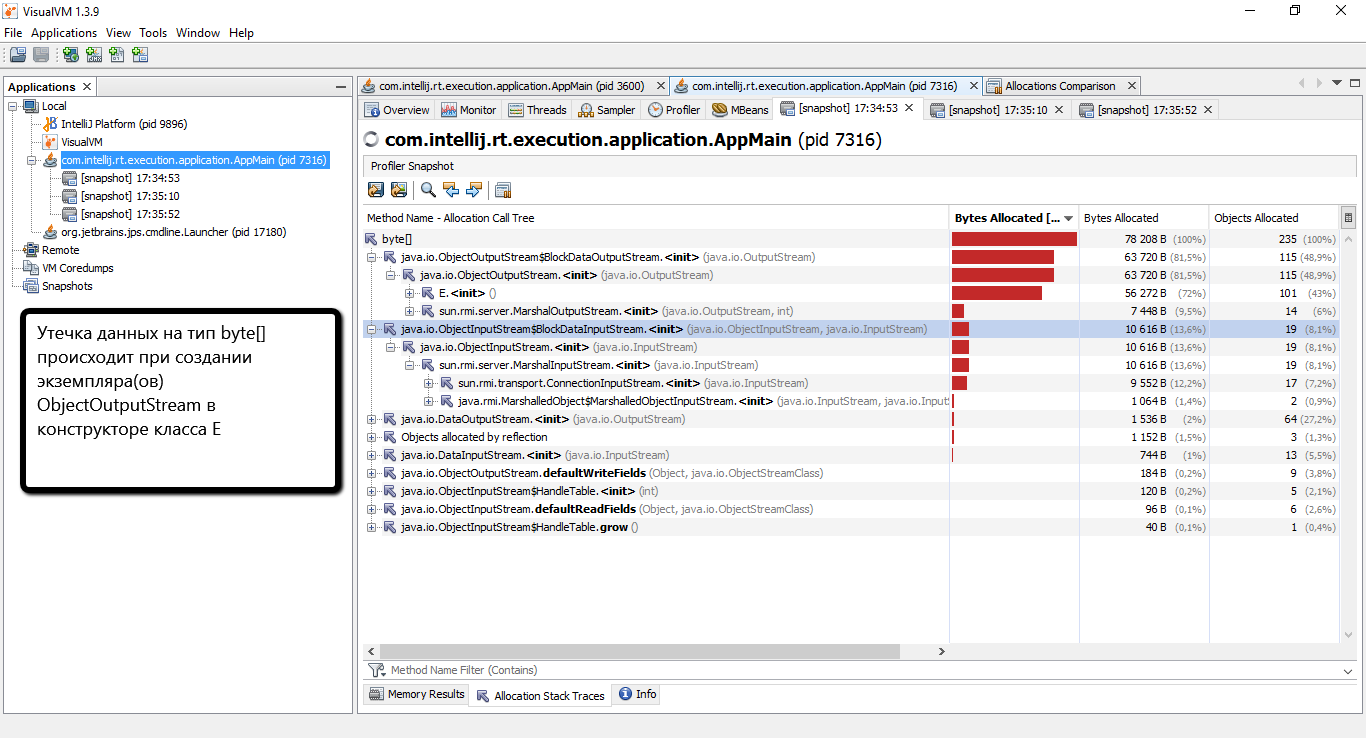


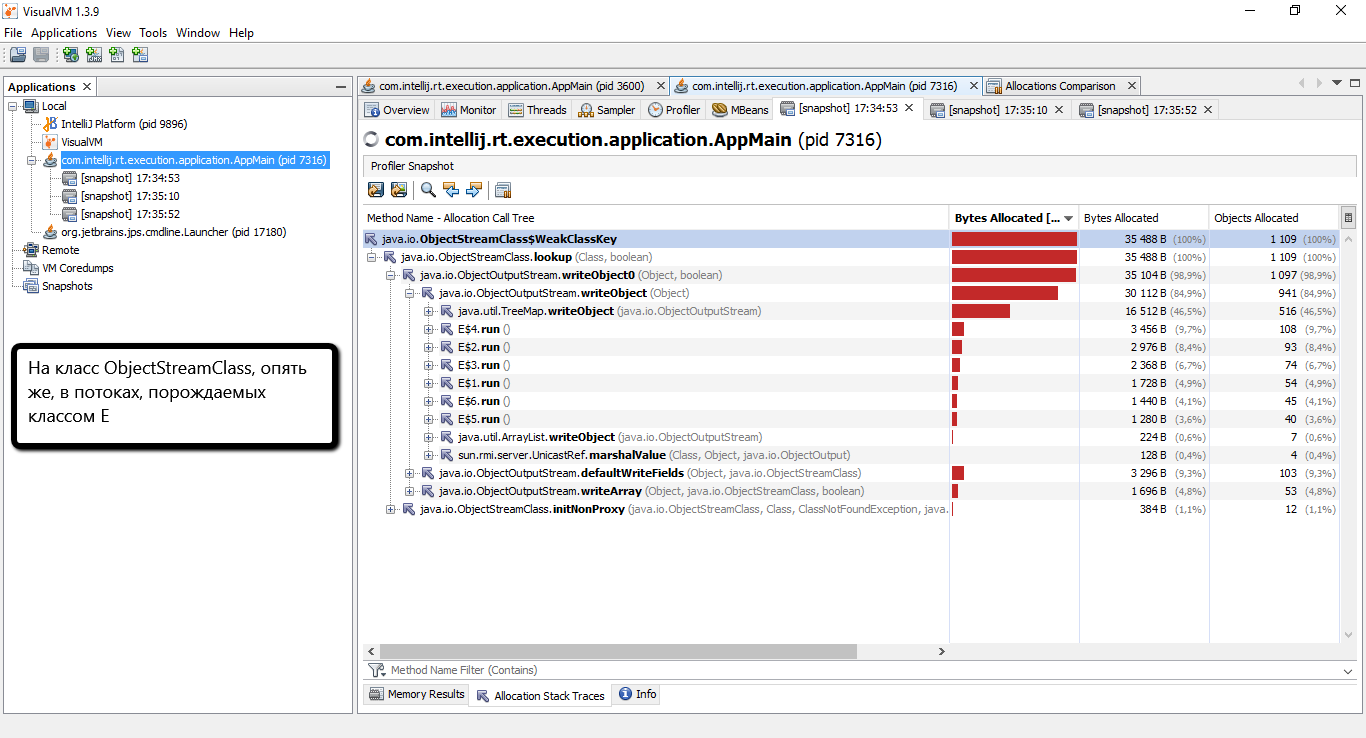


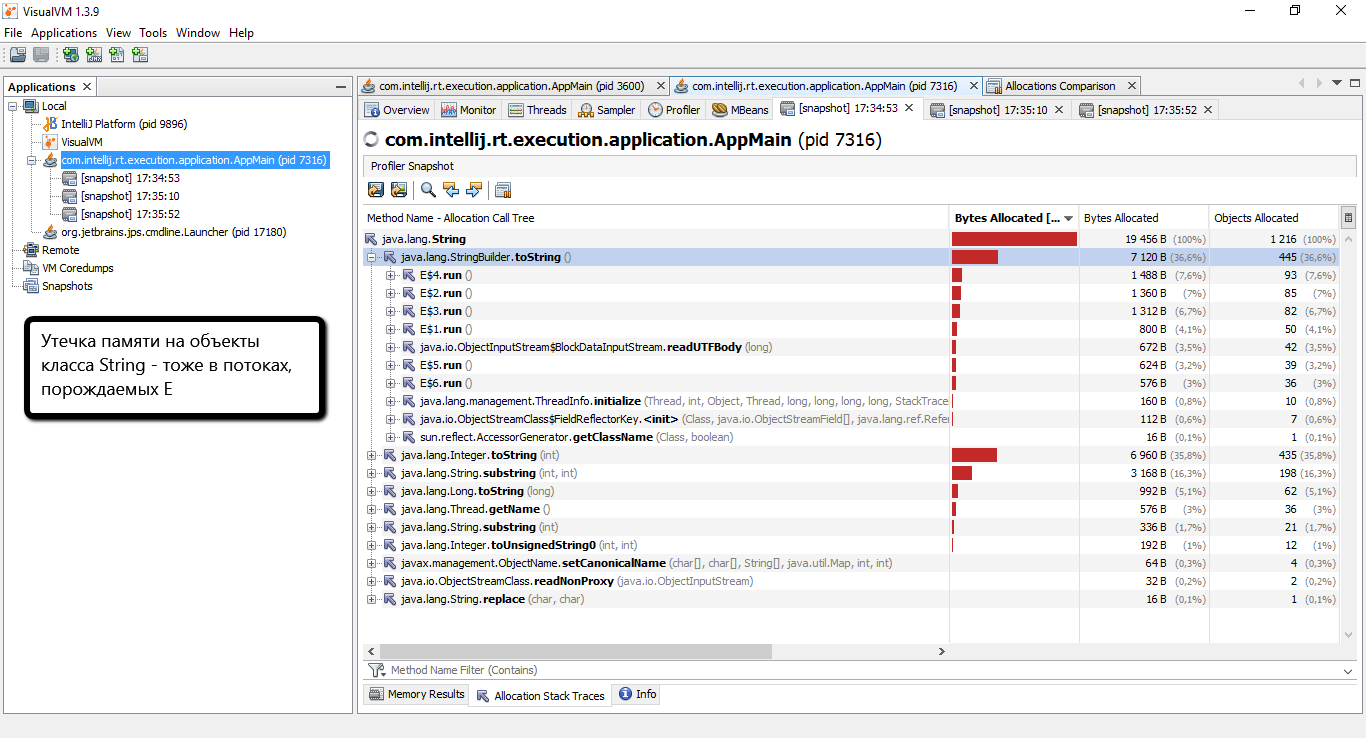








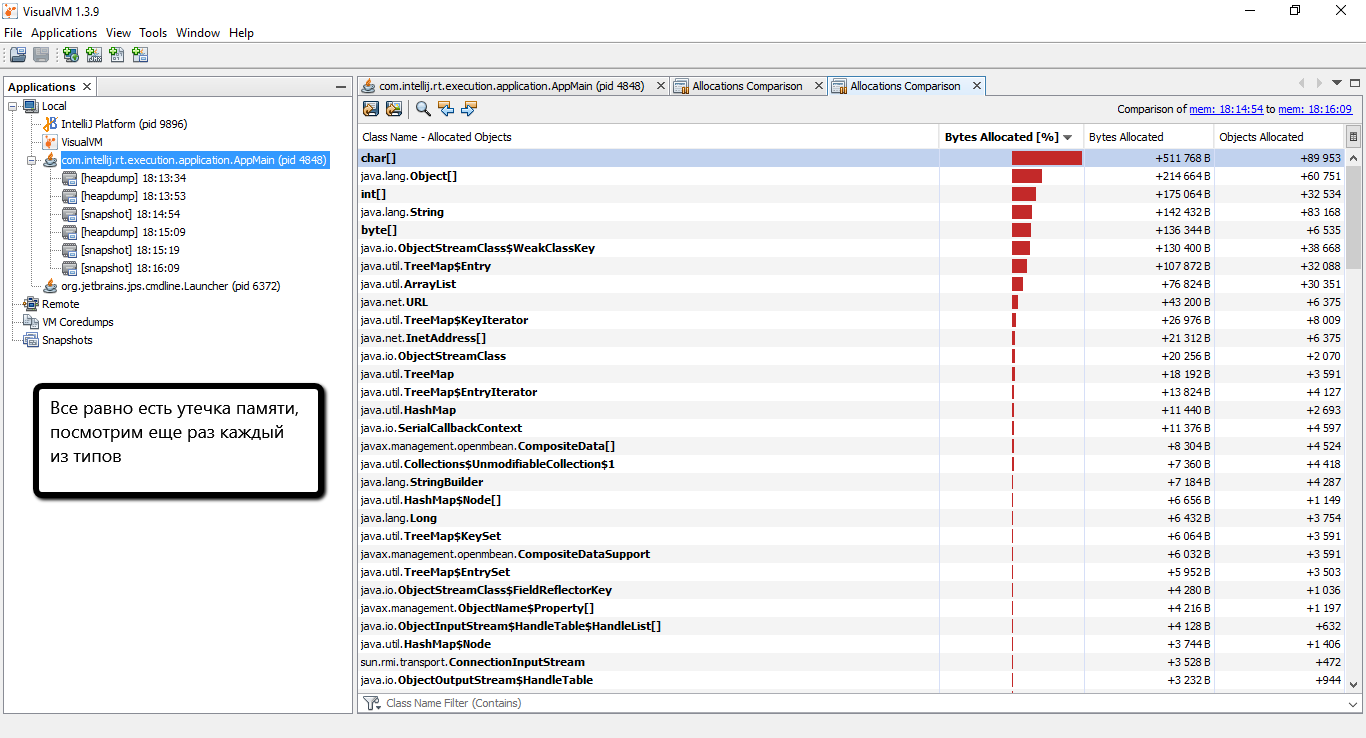


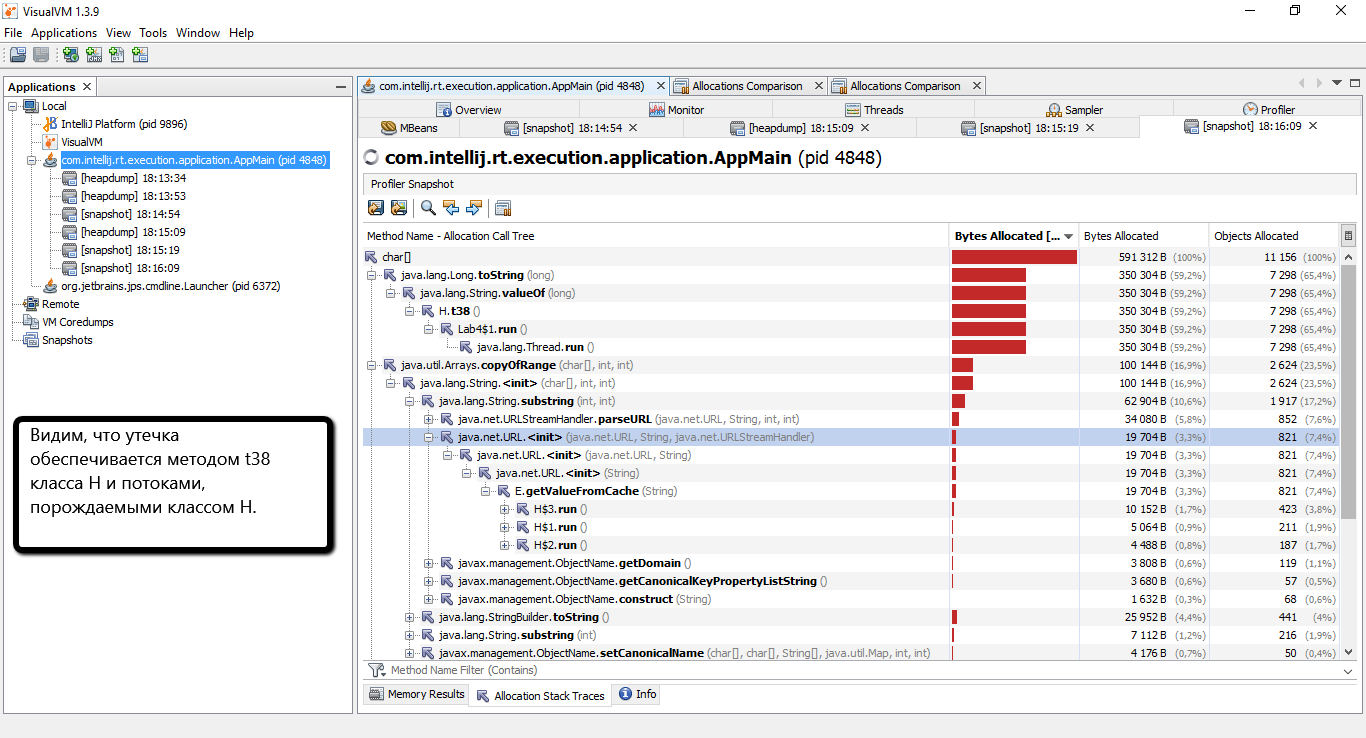


Вывод : нужно обратить внимание на конструктор класса E и на потоки, порождаемые классом E

В конструкторе класса E создается несколько экземпляров ObjectOutputStream с одними и теми же параметрами для каждого объекта. Сделаем соответствующие переменные статическими и будем осуществлять инициализацию только если эти переменные еще не инициализированы.

Что касается потоков, порождаемых объектами E, в них расположены бесконечные циклы вывода однотипных строк в файлы. Сделаем циклы конечными, для этого добавим новую final переменную, с помощью которой можно будет задавать количество итераций. Кроме того, вынесем общие фрагменты кода потоков в одну новую функцию. После этого еще раз проведем профилирование.



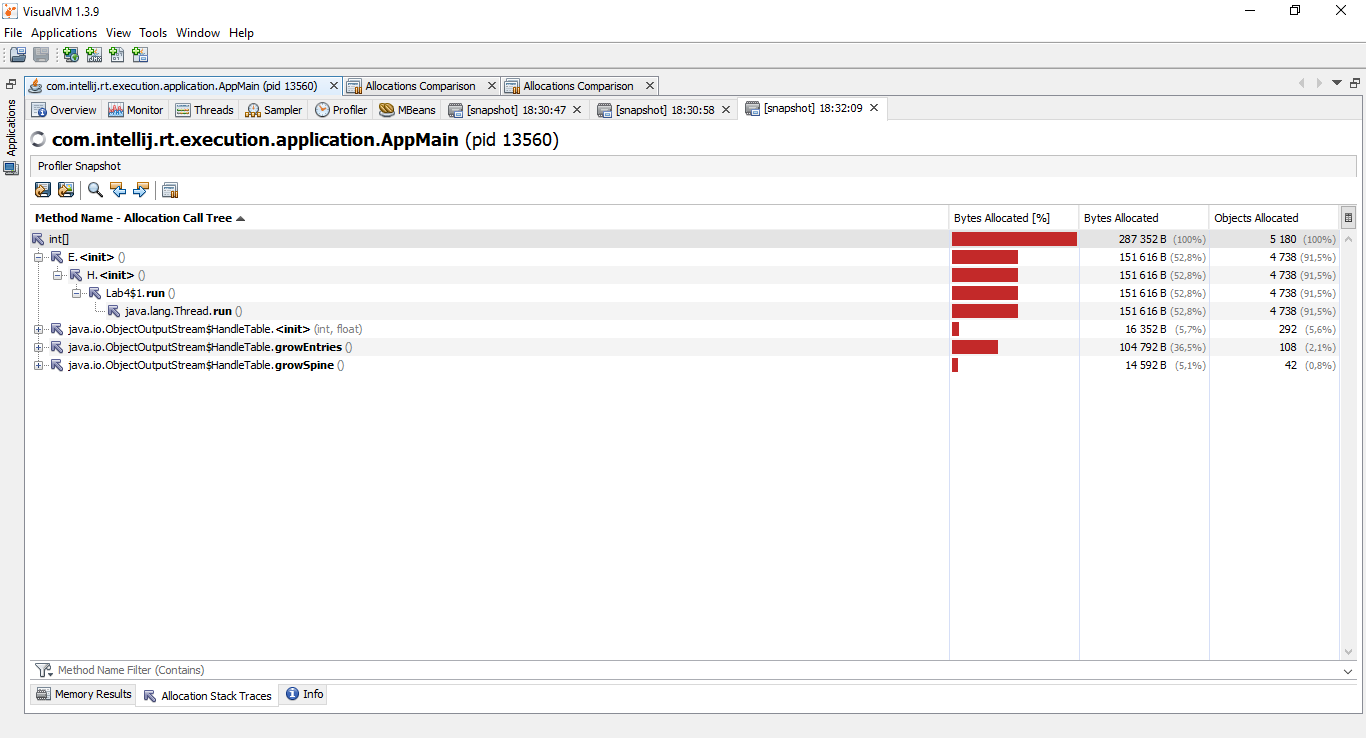


Для устранения утечки в методе t38 сделаем поле t3 статическим и ограничим новой final переменной количество записей в нем для предотвращения бесконечного разрастания коллекции.

В каждом из трех потоков, порождаемых классом H производится извлечение из хеш-таблицы значения по одному и тому же ключу. Однако это происходит в бесконечном цикле и полученное значение никак не используется и нигде не сохраняется. Поэтому уберем цикл вовсе и оставим только один вызов метода, кроме того, одинаковые фрагменты вынесем в функцию. После этого снова посмотрим, что получилось.

Теперь, как видно на скриншоте снизу, утечка происходит в конструкторе E на тип int. В конструкторе E происходит инициализация нескольких целочисленных переменных сделаем их статическими и будем производить инициализацию только если они еще не инициализированы.

При последующих запусках VisualVM также обнаруживаются утечки памяти. Последнее, что можно сделать – вынести создание экземпляра H из бесконечного цикла.



* 1. **Вывод**

Таким образом, профилирование – это измерение характеристик процесса выполнения программы с целью дальнейшей оценки, анализа и, возможно, предприятия попыток повышения производительности, мониторинг – слежение за ресурсами, используемыми программой во время работы, дабы убедиться, что «все идет по плану».

Мониторинг и профилирование представляют собой необходимые этапы разработки и оценки качества программного продукта.

Сегодня на рынке ПО существует огромное множество средств мониторинга и профилирования, в частности, Java имеет в составе jdk устаревший набор утилит, на смену которым пришла jconsole и visualVM, причем большим плюсом второй, в отличие от первой, является возможность профилирования и более гибкой настройки с минимальными затратами ресурсов.