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"

2. 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()});
      }
    }
```

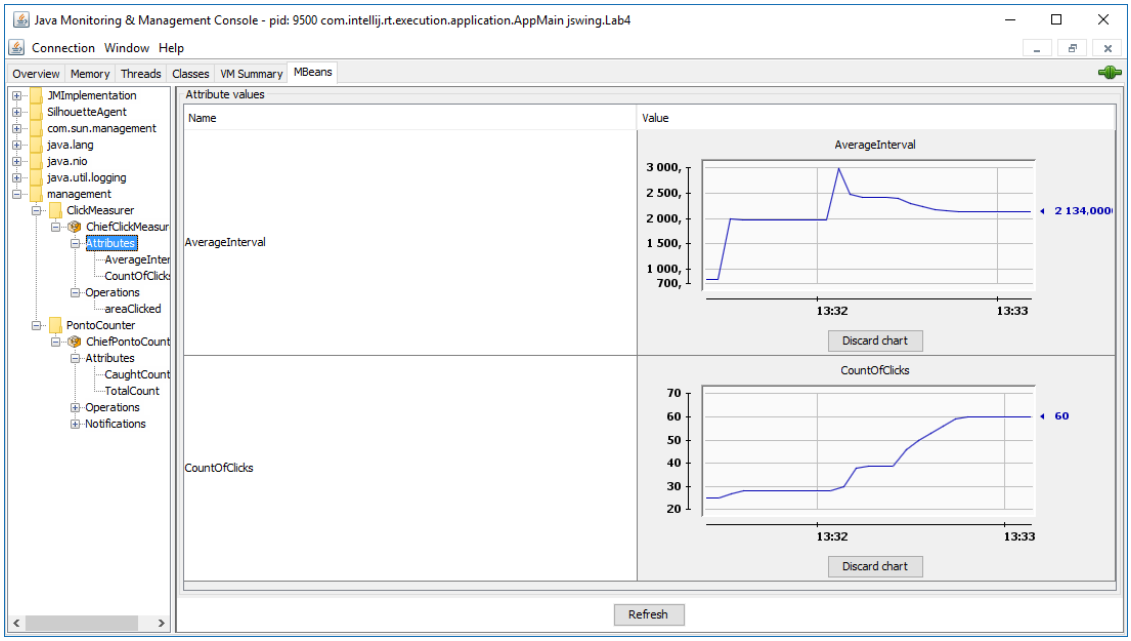
//Файл Lab4.java (приведен не полностью)

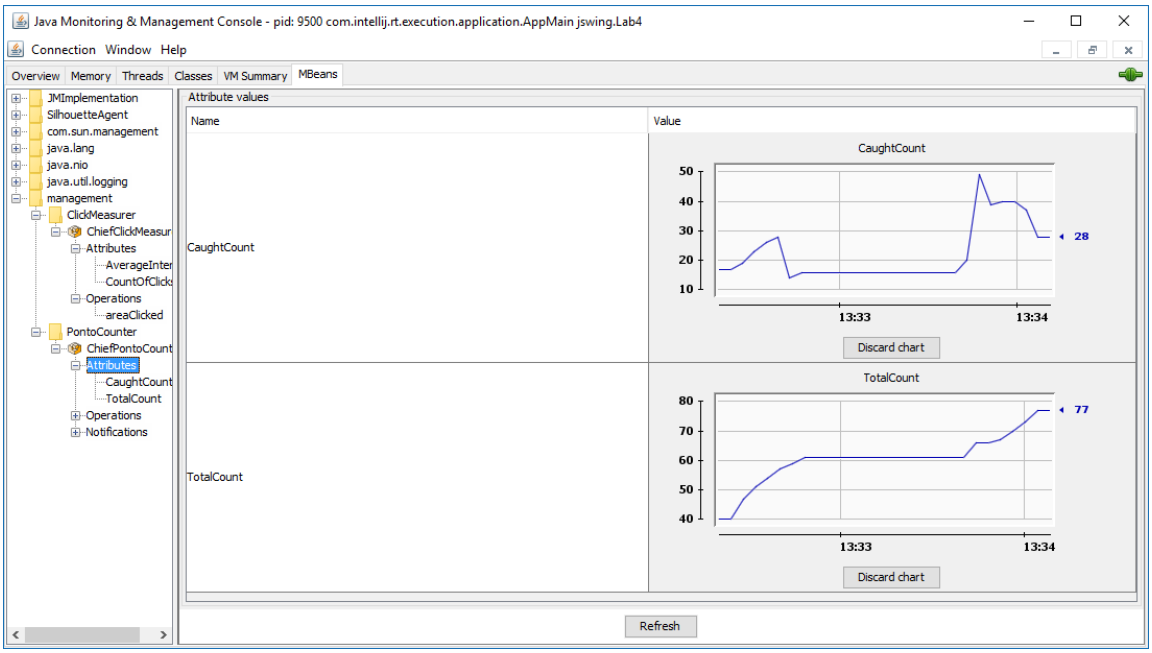
```
package jswing;

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();
```

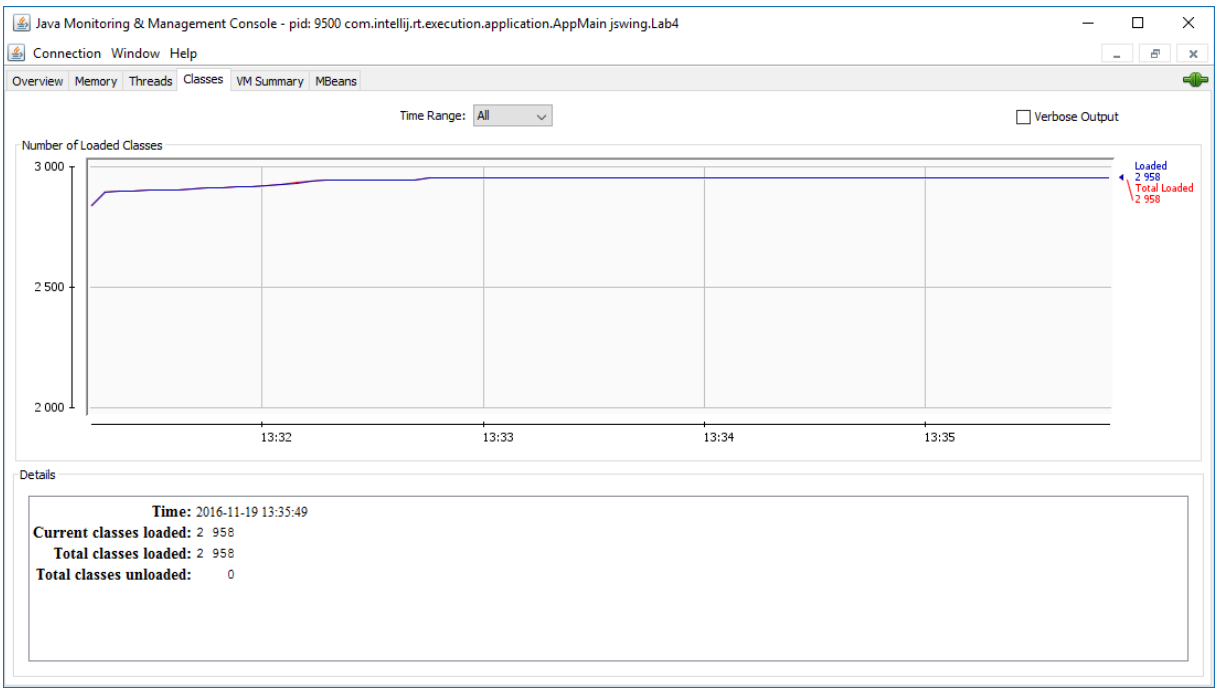
```
    }catch (Exception e){
      e.printStackTrace();
    }
    …
}

…

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();
      }
    }
}
```

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


Java Monitoring & Management Console - pid: 9500 com.intellij.rt.execution.application.AppMain jswing.Lab4
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
JMImplementation
SilhouetteAgent
com.sun.management
java.lang
java.nio
java.util.logging
management
ClickMeasurer
ChiefClickMeasur
Attributes
AverageInter
CountOfClicks
Operations
areaClicked
PontoCounter
ChiefPontoCount
Attributes
CaughtCount
TotalCount
Operations
Notifications
Attribute values
Name
Value
AverageInterval
AverageInterval
3 000,
2 500,
2 000,
1 500,
1 000,
700,
2 134,000
13:32
13:33
Discard chart
CountOfClicks
CountOfClicks
70
60
50
40
30
20
60
13:32
13:33
Discard chart
Refresh

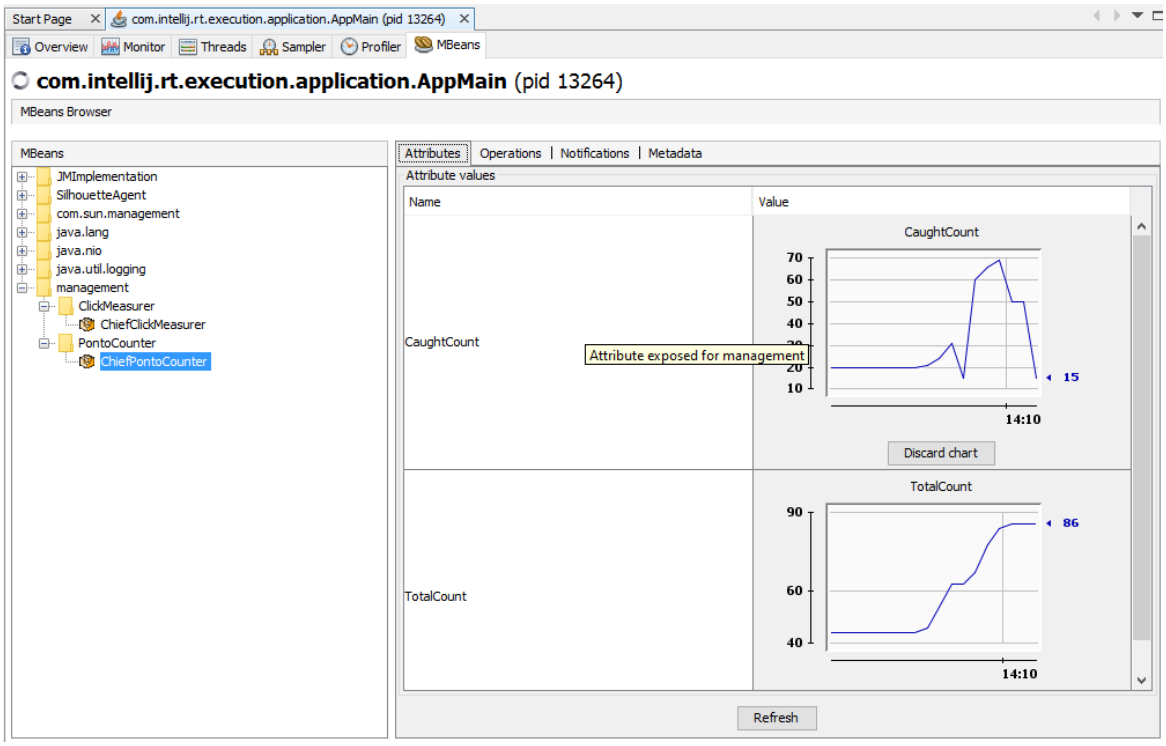

Java Monitoring & Management Console - pid: 9500 com.intellij.rt.execution.application.AppMain jswing.Lab4
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
JMImplementation
SilhouetteAgent
com.sun.management
java.lang
java.nio
java.util.logging
management
ClickMeasurer
ChiefClickMeasur
Attributes
AverageInter
CountOfClicks
Operations
areaClicked
PontoCounter
ChiefPontoCount
Attributes
CaughtCount
TotalCount
Operations
Notifications
Attribute values
Name
Value
CaughtCount
CaughtCount
50
40
30
20
10
28
13:33
13:34
Discard chart
TotalCount
TotalCount
80
70
60
50
40
77
13:33
13:34
Discard chart
Refresh

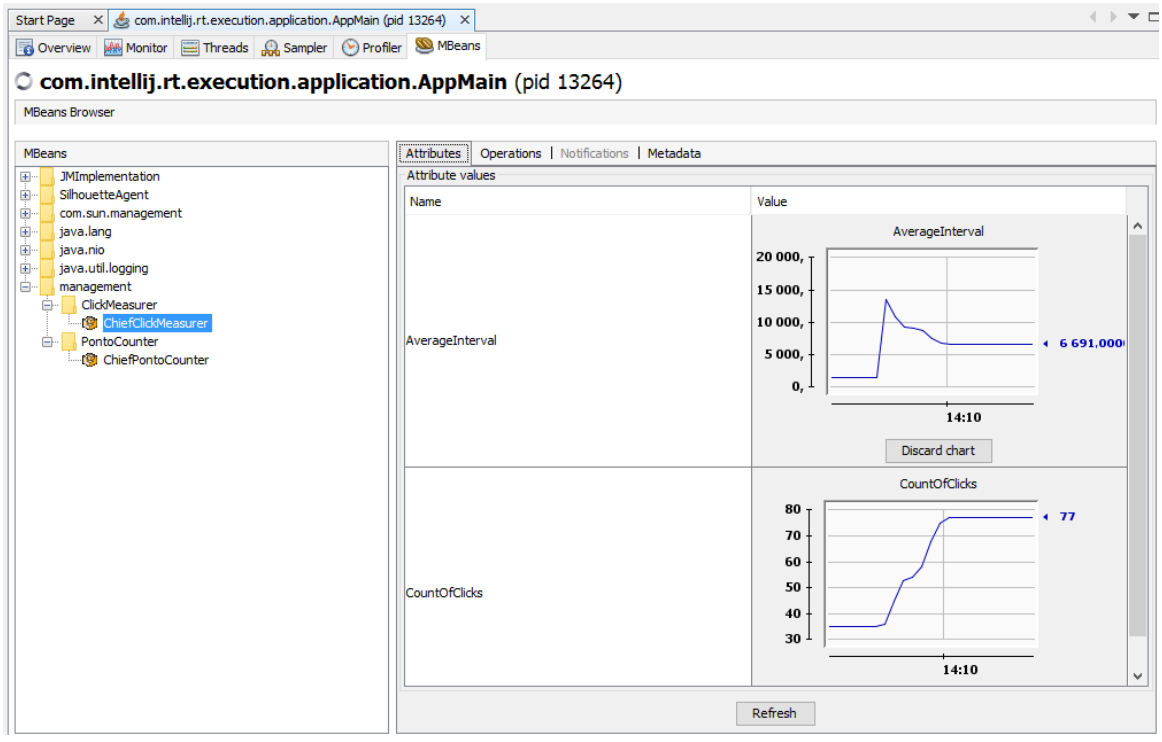

Java Monitoring & Management Console - pid: 9500 com.intellij.rt.execution.application.AppMain jswing.Lab4
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
Time Range: All
Verbose Output
Number of Loaded Classes
3 000
2 500
2 000
Loaded
2 958
Total Loaded
2 958
13:32
13:33
13:34
13:35
Details
Time: 2016-11-19 13:35:49
Current classes loaded: 2 958
Total classes loaded: 2 958
Total classes unloaded: 0

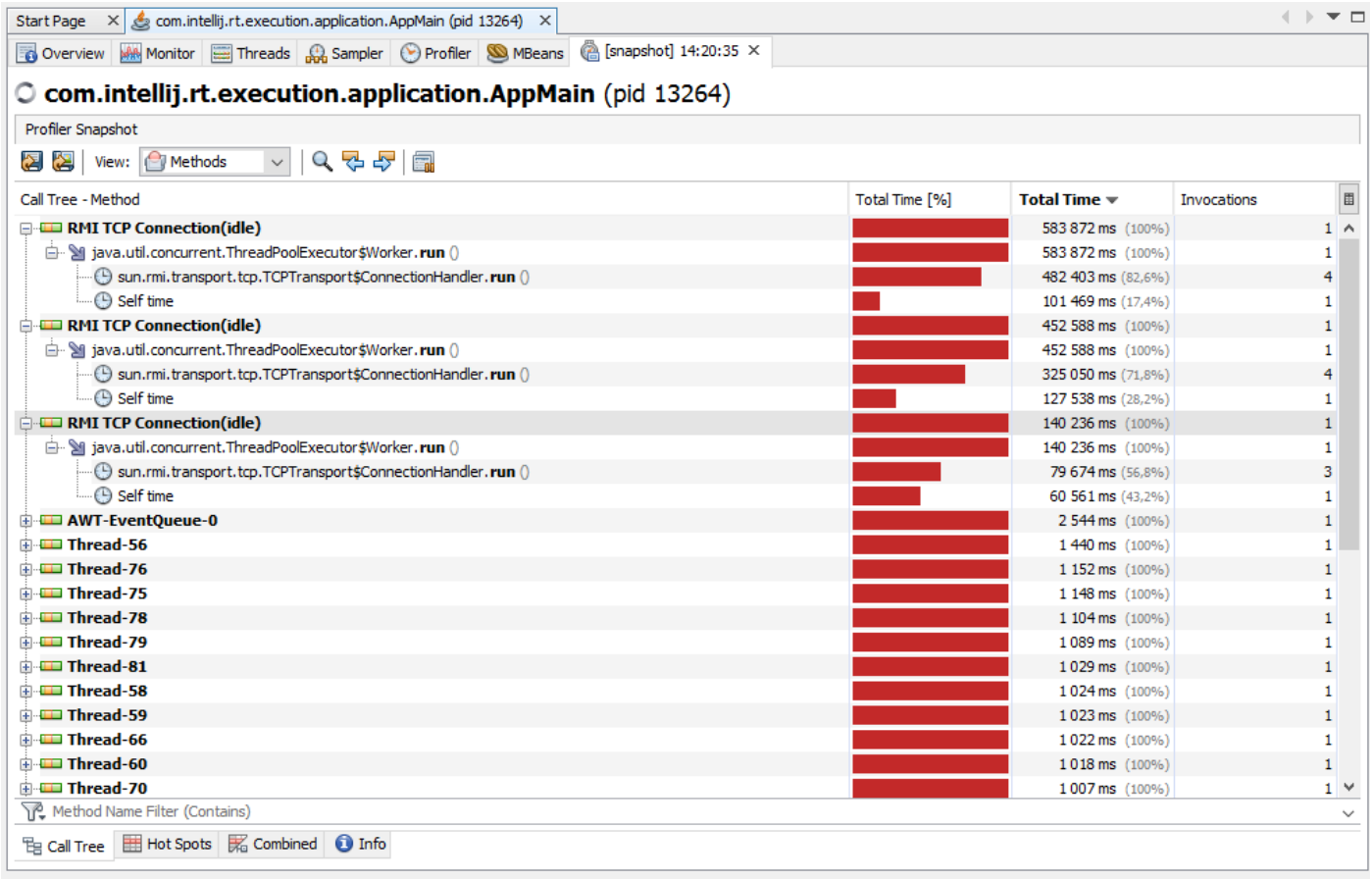
Количество классов, загруженных в результате работы программы : 2958

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.

4. 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 %

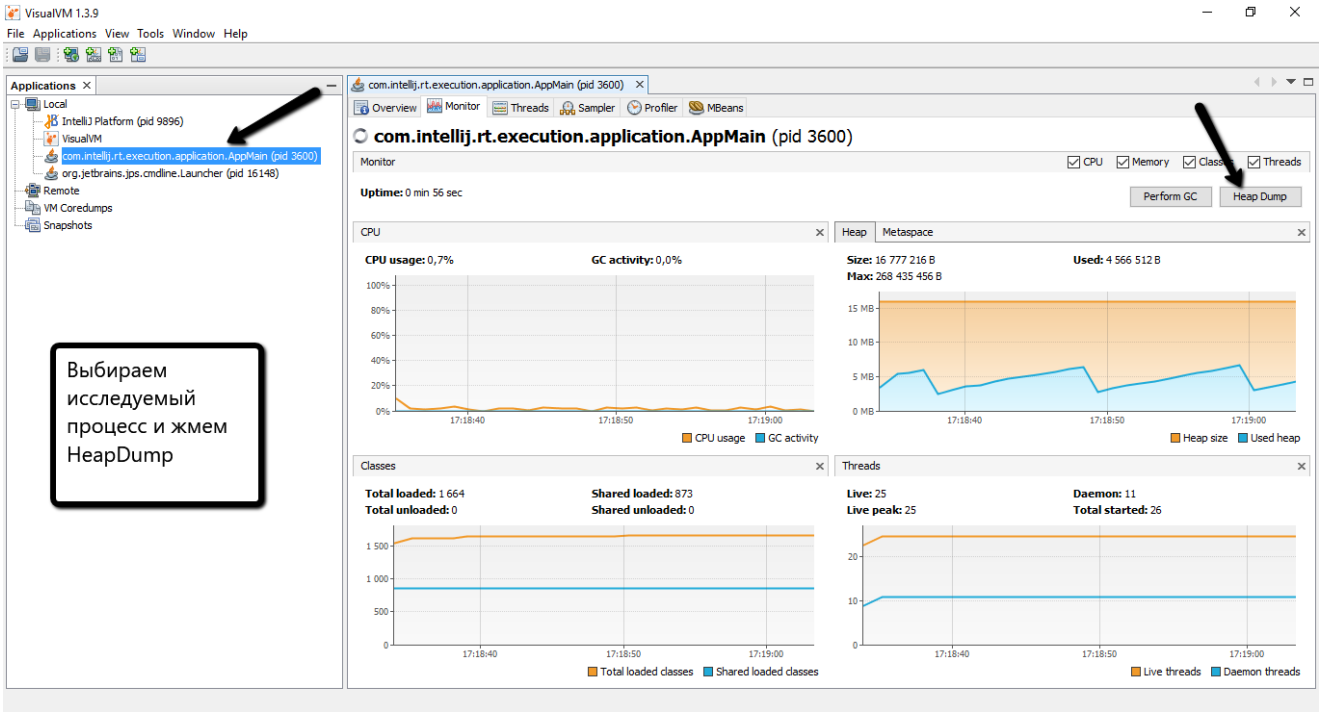
Из которых программа 4-й лабораторной работы по Программированию интернет-приложений запускает только последний метод, соответственно, в наших силах попробовать оптимизировать только его.

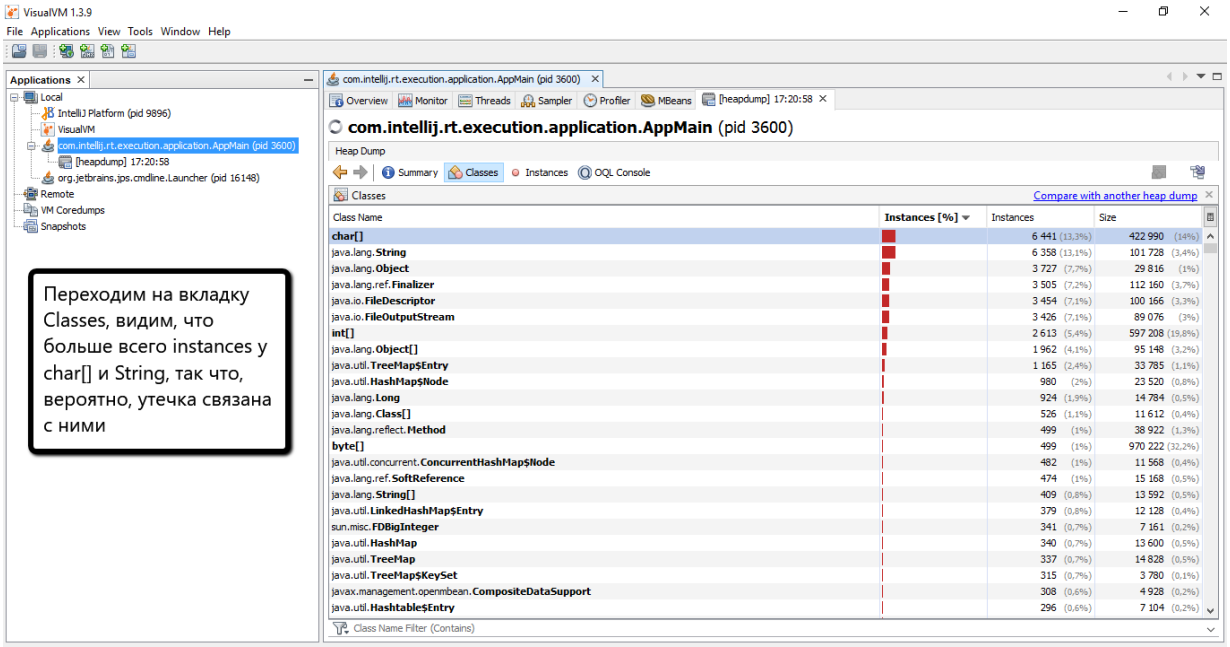
Что касается памяти, то большая её часть уходит на размещение в памяти данных следующих типов:

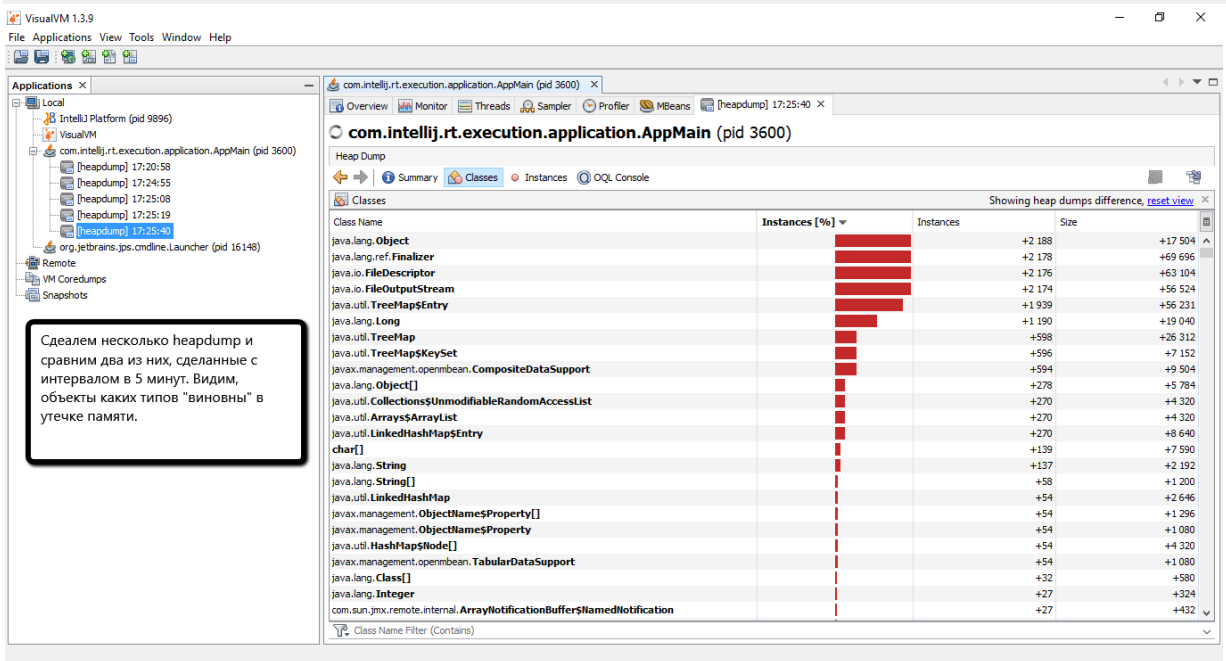
Class Name - Live Allocated Objects	Live Bytes [%]	Live Bytes	Live Objects	Allocated Objects	Generations	Total Alloc. Obj.
char[]		52 408 B (14,2%)	582 (6,1%)	4 174	6	41 416
java.lang.**Object[]**		51 120 B (13,8%)	1 849 (19,5%)	12 724	23	121 889
java.util.**TreeMap$Entry**		45 248 B (12,3%)	1 414 (14,9%)	8 327	29	79 785
byte[]		43 048 B (11,7%)	217 (2,3%)	1 476	5	14 187
java.io.**ObjectStreamClass$WeakClassKey**		41 312 B (11,2%)	1 291 (13,6%)	9 356	1	89 384
int[]		28 424 B (7,7%)	91 (1%)	618	2	5 958
java.util.**TreeMap**		10 080 B (2,7%)	210 (2,2%)	994	29	9 441
java.util.**TreeMap$KeyIterator**		8 832 B (2,4%)	276 (2,9%)	2 299	1	22 087
java.io.**ObjectStreamClass**		6 048 B (1,6%)	63 (0,7%)	434	1	4 234
java.util.**HashMap**		5 840 B (1,6%)	146 (1,5%)	583	4	5 594
java.lang.**String**		5 792 B (1,6%)	362 (3,8%)	2 510	6	24 916

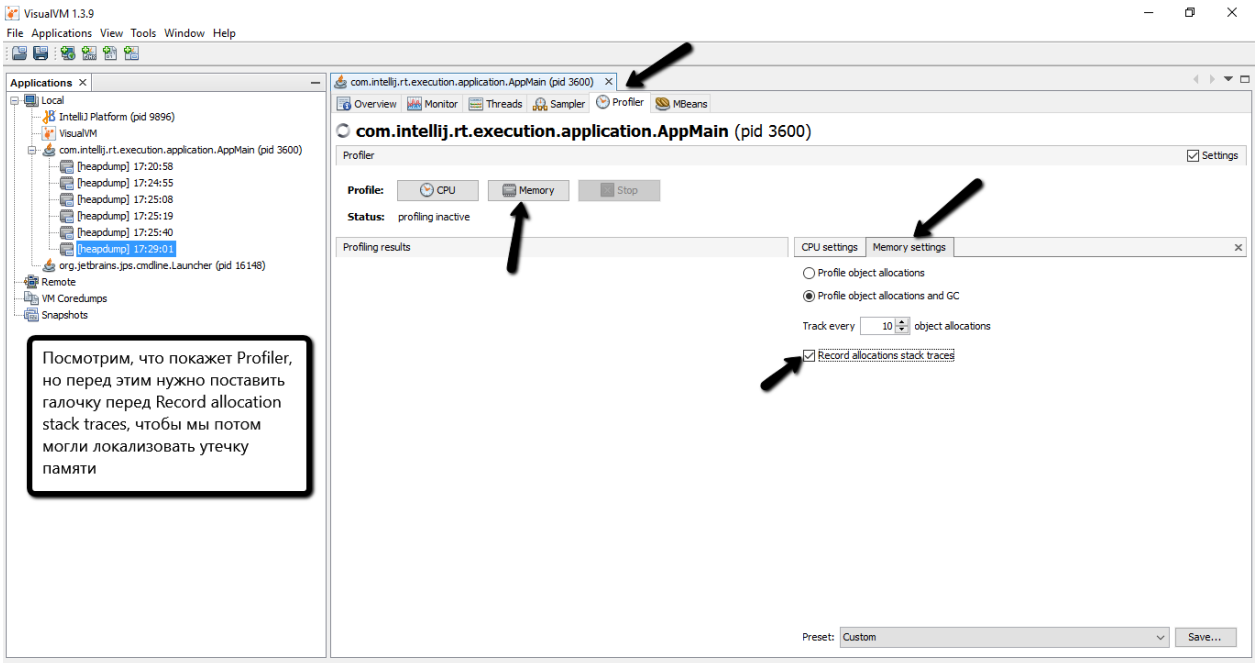
Так, хотя программа и не имеет значительных дефектов в производительности, профилирование показывает, что мы должны еще раз обратить внимание на фрагменты кода, работающие непосредственно с анимацией и попытаться их оптимизировать.

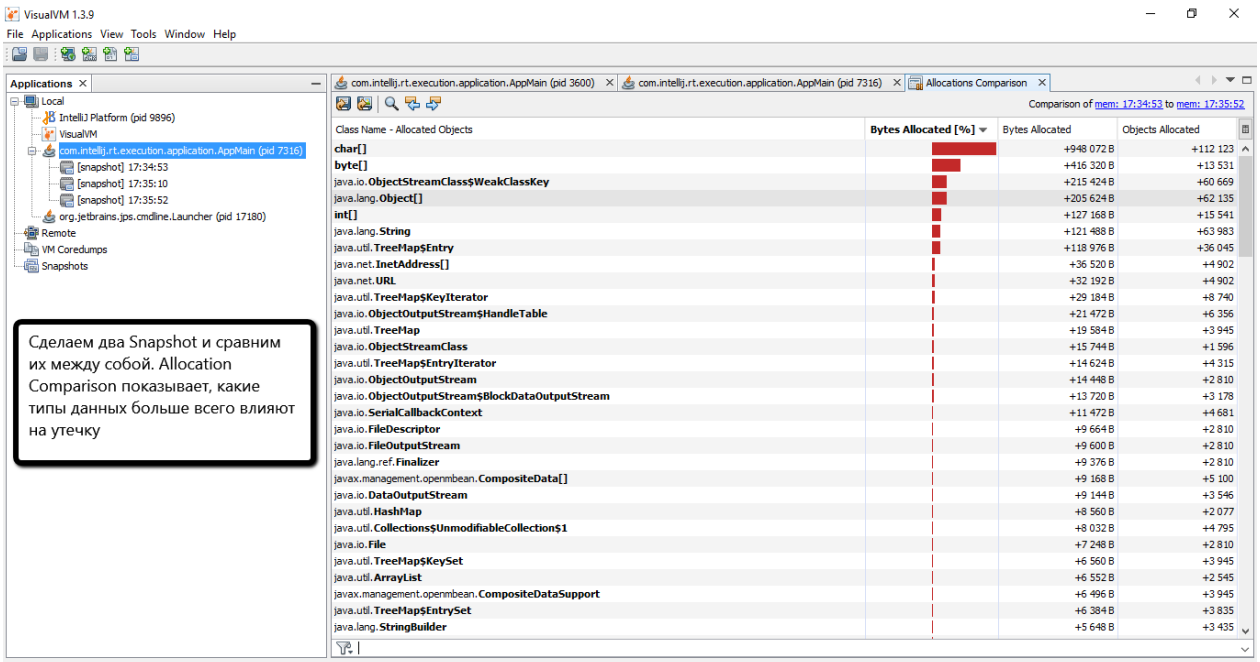
5. Скриншоты программы VisualVM с комментариями по ходу поиска утечки памяти.

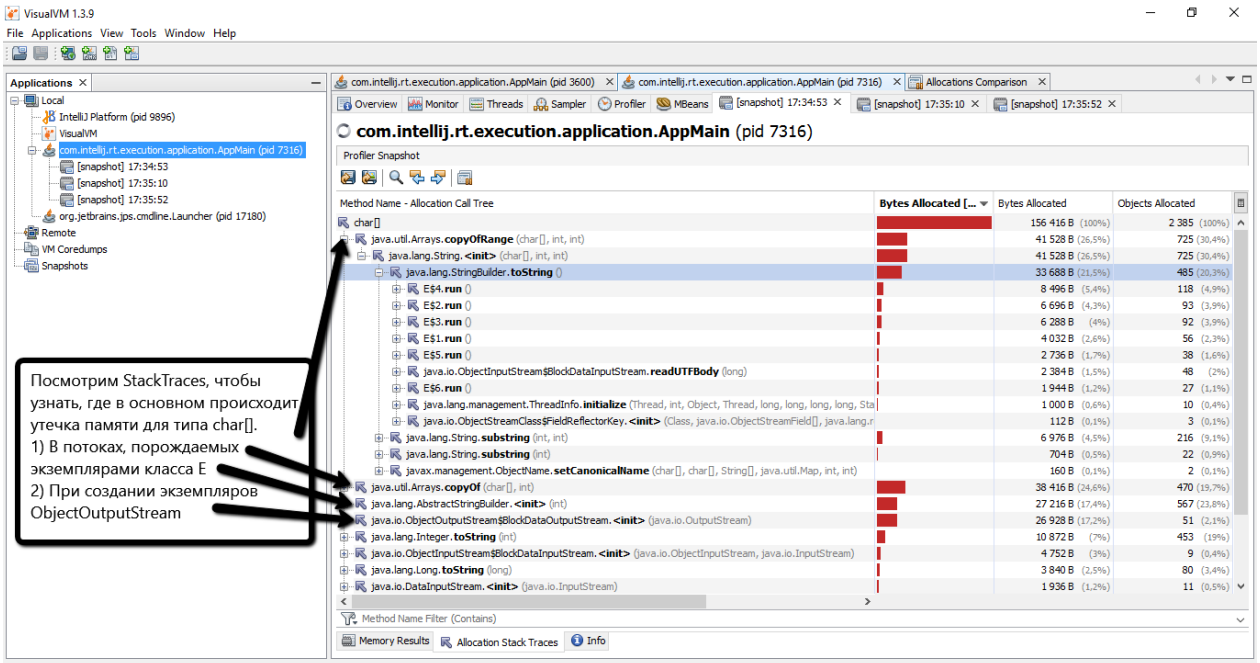

VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 3600)
org.jetbrains.jps.cmdline.Launcher (pid 16148)
Remote
VM Coredumps
Snapshots
com.intellij.rt.execution.application.AppMain (pid 3600)
Overview Monitor Threads Sampler Profiler MBeans
com.intellij.rt.execution.application.AppMain (pid 3600)
Monitor
CPU Memory Classes Threads
Uptime: 0 min 56 sec
Perform GC
Heap Dump
CPU
CPU usage: 0,7%
GC activity: 0,0%
Heap
Metaspace
Size: 16 777 216 B
Used: 4 566 512 B
Max: 268 435 456 B
CPU usage
GC activity
Heap size
Used heap
Classes
Total loaded: 1 664
Shared loaded: 873
Total unloaded: 0
Shared unloaded: 0
Threads
Live: 25
Daemon: 11
Live peak: 25
Total started: 26
Total loaded classes
Shared loaded classes
Live threads
Daemon threads
Выбираем исследуемый процесс и жмем HeapDump

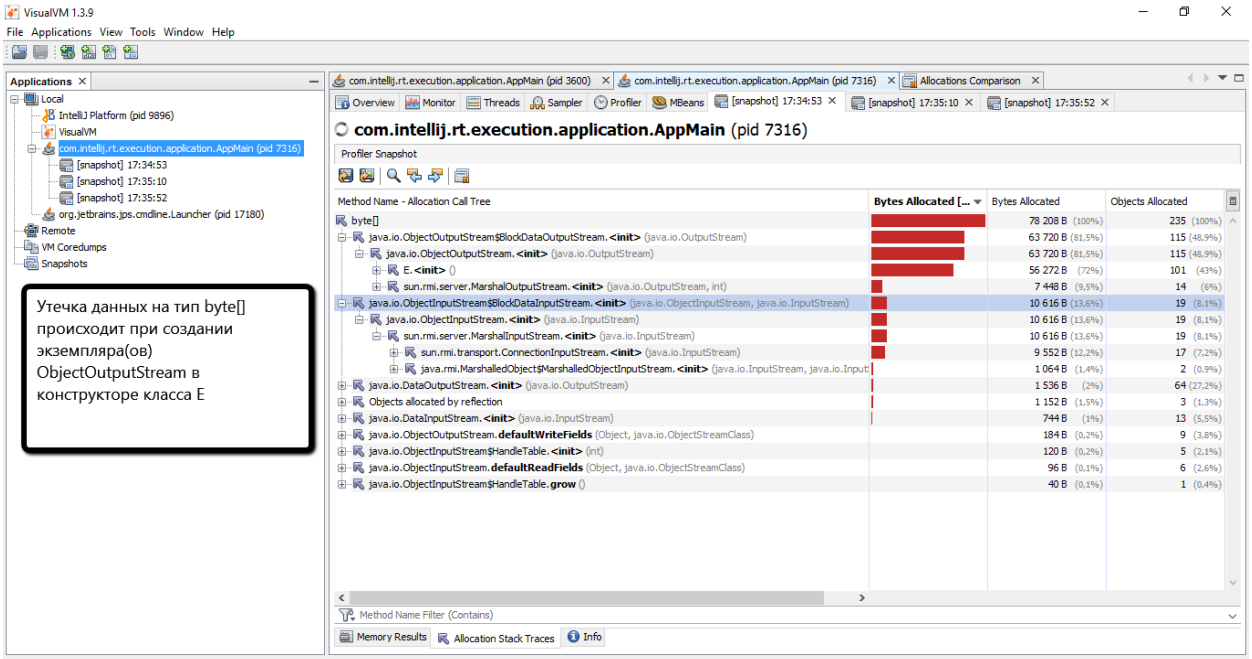

VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 3600)
[heapdump] 17:20:58
org.jetbrains.jps.cmdline.Launcher (pid 16148)
Remote
VM Coredumps
Snapshots
com.intellij.rt.execution.application.AppMain (pid 3600)
Overview Monitor Threads Sampler Profiler MBeans [heapdump] 17:20:58
com.intellij.rt.execution.application.AppMain (pid 3600)
Heap Dump
Summary Classes Instances OQL Console
Classes
Compare with another heap dump
Class Name | Instances [%] | Instances | Size
char[] 6 441 (13,3%) 422 990 (14%)
java.lang.String 6 358 (13,1%) 101 728 (3,4%)
java.lang.Object 3 727 (7,7%) 29 816 (1%)
java.lang.ref.Finalizer 3 505 (7,2%) 112 160 (3,7%)
java.io.FileDescriptor 3 454 (7,1%) 100 166 (3,3%)
java.io.FileOutputStream 3 426 (7,1%) 89 076 (3%)
int[] 2 613 (5,4%) 597 208 (19,8%)
java.lang.Object[] 1 962 (4,1%) 95 148 (3,2%)
java.util.TreeMap$Entry 1 165 (2,4%) 33 785 (1,1%)
java.util.HashMap$Node 980 (2%) 23 520 (0,8%)
java.lang.Long 924 (1,9%) 14 784 (0,5%)
java.lang.Class[] 526 (1,1%) 11 612 (0,4%)
java.lang.reflect.Method 499 (1%) 38 922 (1,3%)
byte[] 499 (1%) 970 222 (32,2%)
java.util.concurrent.ConcurrentHashMap$Node 482 (1%) 11 568 (0,4%)
java.lang.ref.SoftReference 474 (1%) 15 168 (0,5%)
java.lang.String[] 409 (0,8%) 13 592 (0,5%)
java.util.LinkedHashMap$Entry 379 (0,8%) 12 128 (0,4%)
sun.misc.FDBigInteger 341 (0,7%) 7 161 (0,2%)
java.util.HashMap 340 (0,7%) 13 600 (0,5%)
java.util.TreeMap 337 (0,7%) 14 828 (0,5%)
java.util.TreeMap$KeySet 315 (0,7%) 3 780 (0,1%)
javax.management.openmbean.CompositeDataSupport 308 (0,6%) 4 928 (0,2%)
java.util.Hashtable$Entry 296 (0,6%) 7 104 (0,2%)
Class Name Filter (Contains)
Переходим на вкладку Classes, видим, что больше всего instances у char[] и String, так что, вероятно, утечка связана с ними

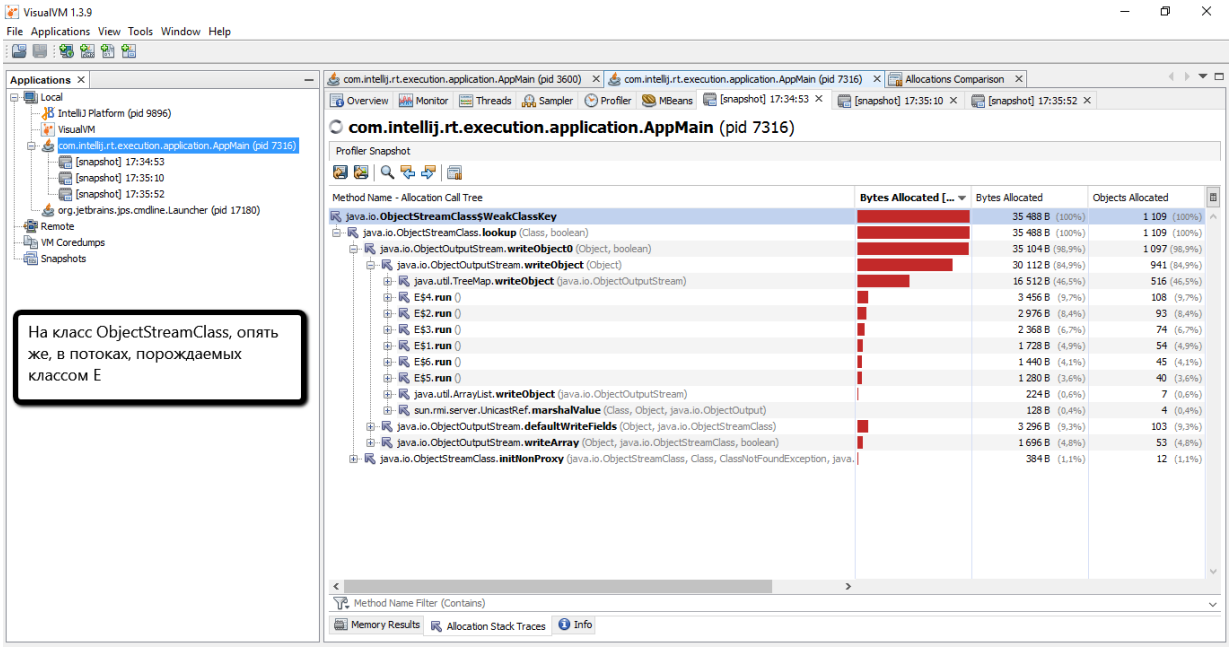

VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 3600)
[heapdump] 17:20:58
[heapdump] 17:24:55
[heapdump] 17:25:08
[heapdump] 17:25:19
[heapdump] 17:25:40
org.jetbrains.jps.cmdline.Launcher (pid 16148)
Remote
VM Coredumps
Snapshots
com.intellij.rt.execution.application.AppMain (pid 3600)
Overview Monitor Threads Sampler Profiler MBeans [heapdump] 17:25:40
com.intellij.rt.execution.application.AppMain (pid 3600)
Heap Dump
Summary Classes Instances OQL Console
Classes
Showing heap dumps difference, reset view
Class Name | Instances [%] | Instances | Size
java.lang.Object +2 188 +17 504
java.lang.ref.Finalizer +2 178 +69 696
java.io.FileDescriptor +2 176 +63 104
java.io.FileOutputStream +2 174 +56 524
java.util.TreeMap$Entry +1 939 +56 231
java.lang.Long +1 190 +19 040
java.util.TreeMap +598 +26 312
java.util.TreeMap$KeySet +596 +7 152
javax.management.openmbean.CompositeDataSupport +594 +9 504
java.lang.Object[] +278 +5 784
java.util.Collections$UnmodifiableRandomAccessList +270 +4 320
java.util.Arrays$ArrayList +270 +4 320
java.util.LinkedHashMap$Entry +270 +8 640
char[] +139 +7 590
java.lang.String +137 +2 192
java.lang.String[] +58 +1 200
java.util.LinkedHashMap +54 +2 646
javax.management.ObjectName$Property[] +54 +1 296
javax.management.ObjectName$Property +54 +1 080
java.util.HashMap$Node[] +54 +4 320
javax.management.openmbean.TabularDataSupport +54 +1 080
java.lang.Class[] +32 +580
java.lang.Integer +27 +324
com.sun.jmx.remote.internal.ArrayNotificationBuffer$NamedNotification +27 +432
Class Name Filter (Contains)
Сделаем несколько heapdump и сравним два из них, сделанные с интервалом в 5 минут. Видим, объекты каких типов "виновны" в утечке памяти.

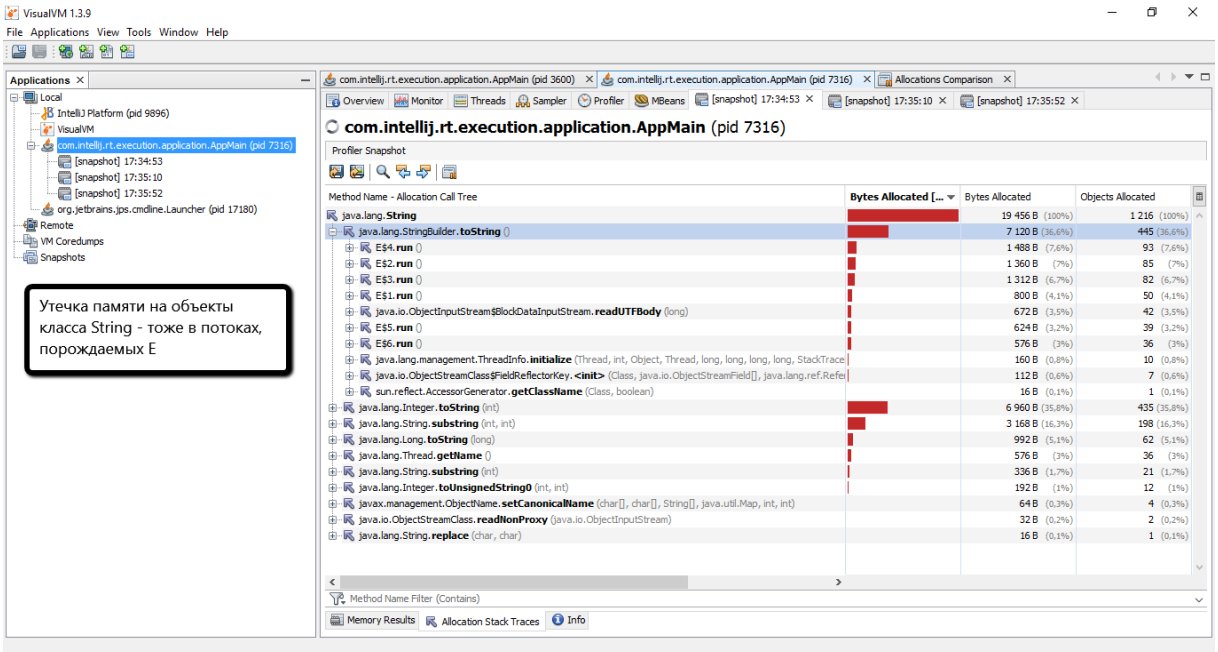

VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 3600)
[heapdump] 17:20:58
[heapdump] 17:24:55
[heapdump] 17:25:08
[heapdump] 17:25:19
[heapdump] 17:25:40
[heapdump] 17:29:01
org.jetbrains.jps.cmdline.Launcher (pid 16148)
Remote
VM Coredumps
Snapshots
Посмотрим, что покажет Profiler, но перед этим нужно поставить галочку перед Record allocation stack traces, чтобы мы потом могли локализовать утечку памяти
com.intellij.rt.execution.application.AppMain (pid 3600)
Overview Monitor Threads Sampler Profiler MBeans
com.intellij.rt.execution.application.AppMain (pid 3600)
Profiler
Settings
Profile: CPU Memory Stop
Status: profiling inactive
Profiling results
CPU settings
Memory settings
Profile object allocations
Profile object allocations and GC
Track every 10 object allocations
Record allocations stack traces
Preset: Custom
Save...


VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 7316)
[snapshot] 17:34:53
[snapshot] 17:35:10
[snapshot] 17:35:52
org.jetbrains.jps.cmdline.Launcher (pid 17180)
Remote
VM Coredumps
Snapshots
Сделаем два Snapshot и сравним их между собой. Allocation Comparison показывает, какие типы данных больше всего влияют на утечку
com.intellij.rt.execution.application.AppMain (pid 3600)
com.intellij.rt.execution.application.AppMain (pid 7316)
Allocations Comparison
Comparison of mem: 17:34:53 to mem: 17:35:52
Class Name - Allocated Objects
Bytes Allocated [%]
Bytes Allocated
Objects Allocated
char[] +948 072 B +112 123
byte[] +416 320 B +13 531
java.io.ObjectStreamClass$WeakClassKey +215 424 B +60 669
java.lang.Object[] +205 624 B +62 135
int[] +127 168 B +15 541
java.lang.String +121 488 B +63 983
java.util.TreeMap$Entry +118 976 B +36 045
java.net.InetAddress[] +36 520 B +4 902
java.net.URL +32 192 B +4 902
java.util.TreeMap$KeyIterator +29 184 B +8 740
java.io.ObjectOutputStream$HandleTable +21 472 B +6 356
java.util.TreeMap +19 584 B +3 945
java.io.ObjectStreamClass +15 744 B +1 596
java.util.TreeMap$EntryIterator +14 624 B +4 315
java.io.ObjectOutputStream +14 448 B +2 810
java.io.ObjectOutputStream$BlockDataOutputStream +13 720 B +3 178
java.io.SerialCallbackContext +11 472 B +4 681
java.io.FileDescriptor +9 664 B +2 810
java.io.FileOutputStream +9 600 B +2 810
java.lang.ref.Finalizer +9 376 B +2 810
javax.management.openmbean.CompositeData[] +9 168 B +5 100
java.io.DataOutputStream +9 144 B +3 546
java.util.HashMap +8 560 B +2 077
java.util.Collections$UnmodifiableCollection$1 +8 032 B +4 795
java.io.File +7 248 B +2 810
java.util.TreeMap$KeySet +6 560 B +3 945
java.util.ArrayList +6 552 B +2 545
javax.management.openmbean.CompositeDataSupport +6 496 B +3 945
java.util.TreeMap$EntrySet +6 384 B +3 835
java.lang.StringBuilder +5 648 B +3 435


VisualVM 1.3.9
File Applications View Tools Window Help
Applications
Local
IntelliJ Platform (pid 9896)
VisualVM
com.intellij.rt.execution.application.AppMain (pid 7316)
[snapshot] 17:34:53
[snapshot] 17:35:10
[snapshot] 17:35:52
org.jetbrains.jps.cmdline.Launcher (pid 17180)
Remote
VM Coredumps
Snapshots
Посмотрим StackTraces, чтобы узнать, где в основном происходит утечка памяти для типа char[].
1) В потоках, порождаемых экземплярами класса E
2) При создании экземпляров ObjectOutputStream
com.intellij.rt.execution.application.AppMain (pid 3600)
com.intellij.rt.execution.application.AppMain (pid 7316)
Allocations Comparison
Overview Monitor Threads Sampler Profiler MBeans [snapshot] 17:34:53 [snapshot] 17:35:10 [snapshot] 17:35:52
com.intellij.rt.execution.application.AppMain (pid 7316)
Profiler Snapshot
Method Name - Allocation Call Tree
Bytes Allocated [...
Bytes Allocated
Objects Allocated
char[] 156 416 B (100%) 2 385 (100%)
java.util.Arrays.copyOfRange (char[], int, int) 41 528 B (26,5%) 725 (30,4%)
java.lang.String.<init> (char[], int, int) 41 528 B (26,5%) 725 (30,4%)
java.lang.StringBuilder.toString () 33 688 B (21,5%) 485 (20,3%)
E$4.run () 8 496 B (5,4%) 118 (4,9%)
E$2.run () 6 696 B (4,3%) 93 (3,9%)
E$3.run () 6 288 B (4%) 92 (3,9%)
E$1.run () 4 032 B (2,6%) 56 (2,3%)
E$5.run () 2 736 B (1,7%) 38 (1,6%)
java.io.ObjectInputStream$BlockDataInputStream.readUTFBody (long) 2 384 B (1,5%) 48 (2%)
E$6.run () 1 944 B (1,2%) 27 (1,1%)
java.lang.management.ThreadInfo.initialize (Thread, int, Object, Thread, long, long, long, Sta 1 000 B (0,6%) 10 (0,4%)
java.io.ObjectStreamClass$FieldReflectorKey.<init> (Class, java.io.ObjectStreamField[], java.lang.r 112 B (0,1%) 3 (0,1%)
java.lang.String.substring (int, int) 6 976 B (4,5%) 216 (9,1%)
java.lang.String.substring (int) 704 B (0,5%) 22 (0,9%)
javax.management.ObjectName.setCanonicalName (char[], char[], String[], java.util.Map, int, int) 160 B (0,1%) 2 (0,1%)
java.util.Arrays.copyOf (char[], int) 38 416 B (24,6%) 470 (19,7%)
java.lang.AbstractStringBuilder.<init> (int) 27 216 B (17,4%) 567 (23,8%)
java.io.ObjectOutputStream$BlockDataOutputStream.<init> (java.io.OutputStream) 26 928 B (17,2%) 51 (2,1%)
java.lang.Integer.toString (int) 10 872 B (7%) 453 (19%)
java.io.ObjectInputStream$BlockDataInputStream.<init> (java.io.ObjectInputStream, java.io.InputStream) 4 752 B (3%) 9 (0,4%)
java.lang.Long.toString (long) 3 840 B (2,5%) 80 (3,4%)
java.io.DataInputStream.<init> (java.io.InputStream) 1 936 B (1,2%) 11 (0,5%)
Method Name Filter (Contains)
Memory Results
Allocation Stack Traces
Info



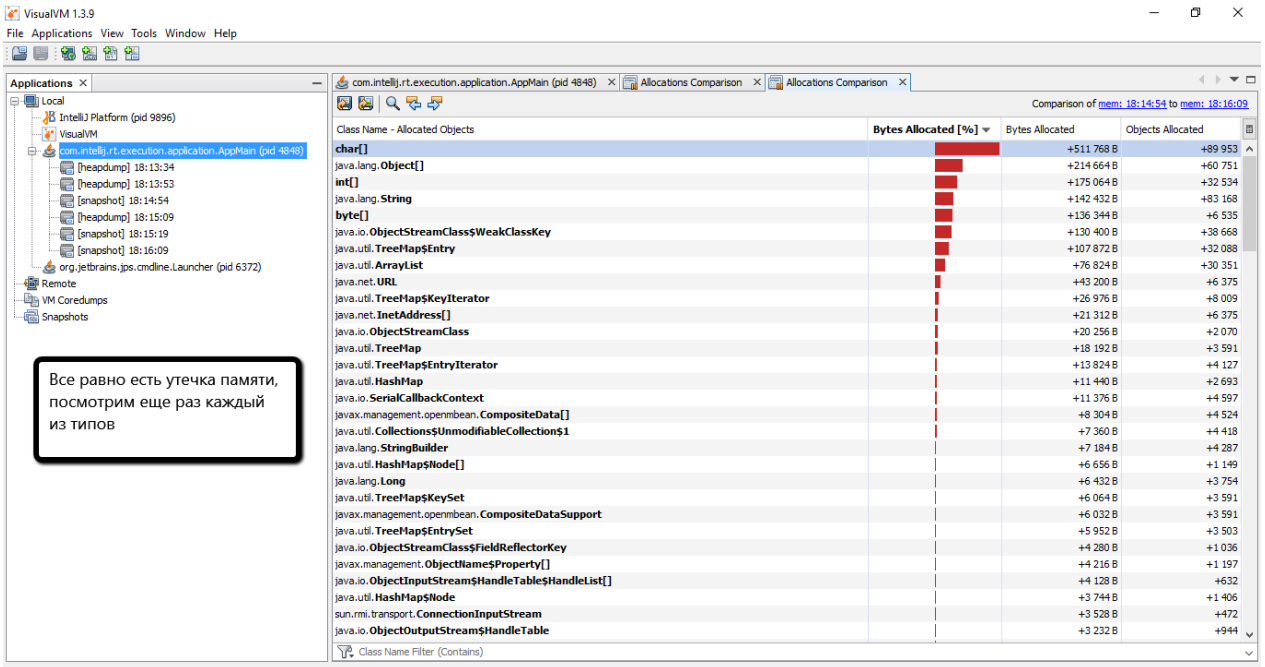


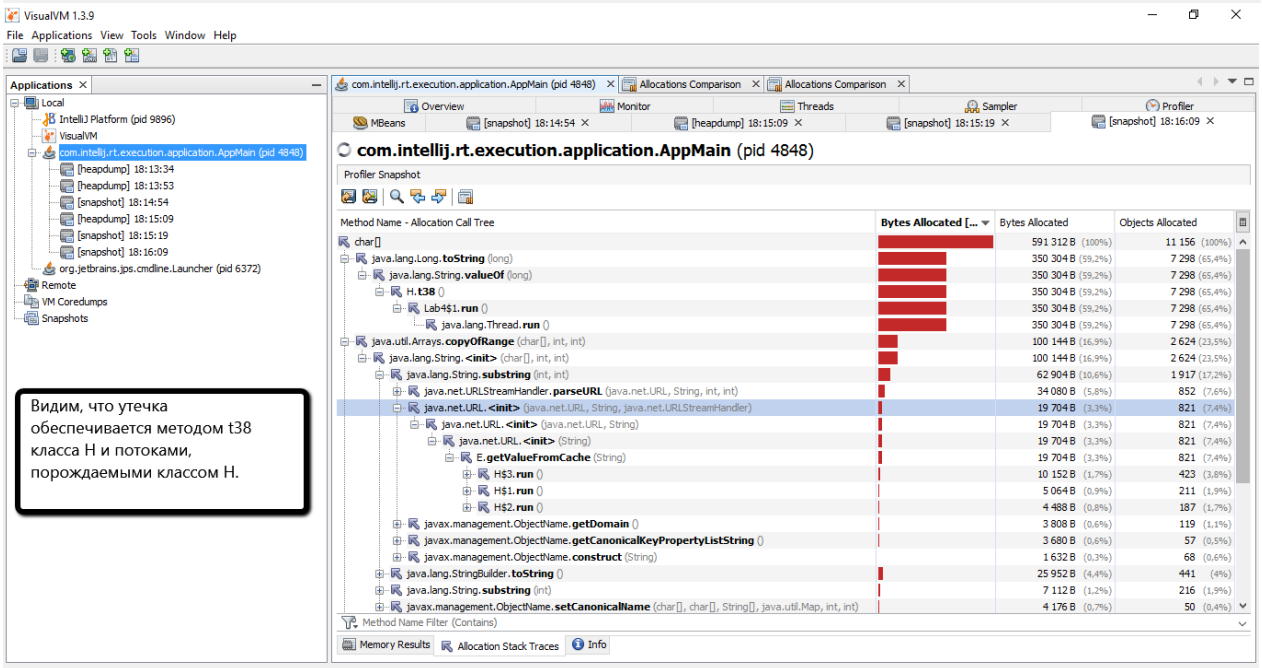


Вывод : нужно обратить внимание на конструктор класса E и на потоки, порождаемые классом E

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.



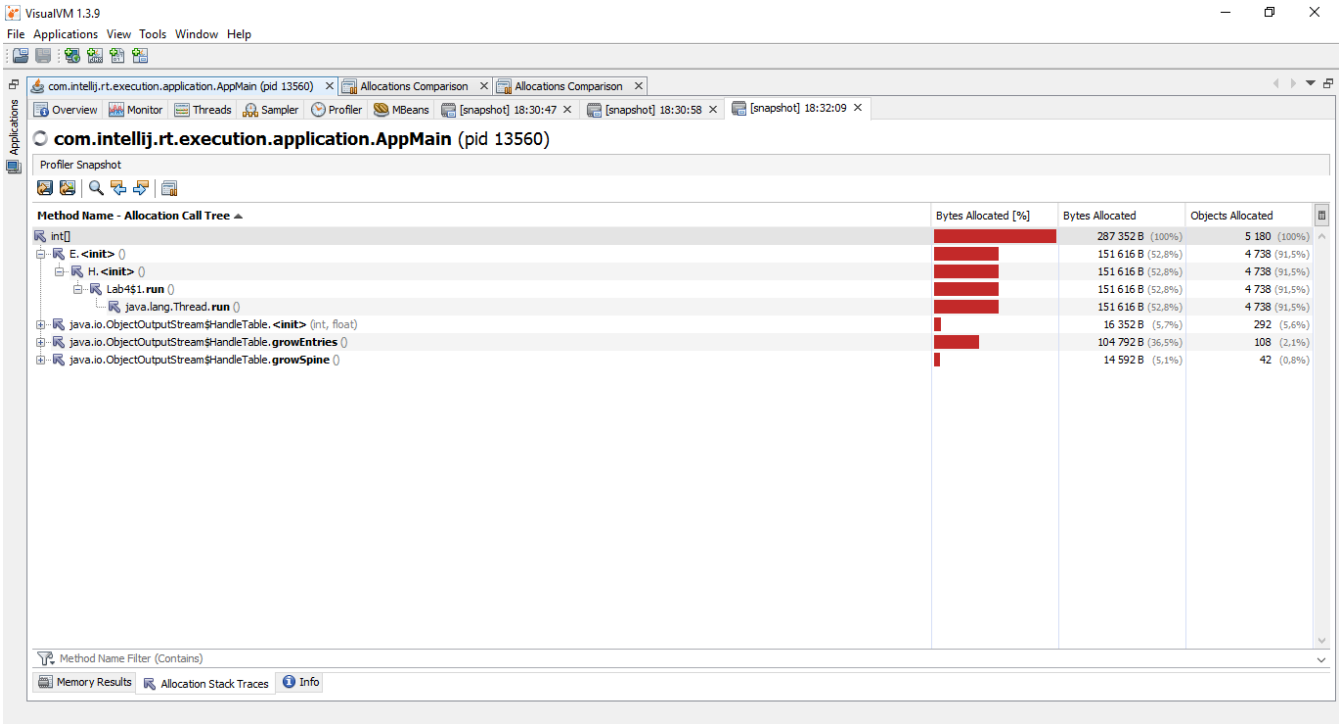


Для устранения утечки в методе t38 сделаем поле t3 статическим и ограничим новой final переменной количество записей в нем для предотвращения бесконечного разрастания коллекции.

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.



5. 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.