

---

# Программная инженерия

---

Дмитриев Андрей Владиславович

[andrei-dmitriev@yandex.ru](mailto:andrei-dmitriev@yandex.ru)

<http://in4mix2006.narod.ru/>

2008

---

# Обнаружение узких мест и анализ утечек памяти

## Обнаружение узких мест и анализ утечек памяти

Дмитриев Андрей Владиславович  
[andrei-dmitriev@yandex.ru](mailto:andrei-dmitriev@yandex.ru)  
<http://in4mix2006.narod.ru/>  
2008

---

# Качество программы

Качество можно оценивать по следующим критериям:

- Результативность – решает ли программа поставленную задачу?
- Быстродействие – насколько быстро программа решает задачу?
- Надежность – насколько можно доверять результатам работы программы?
- Ресурсоемкость – как много ресурсов нужно программе для решения задачи?
- ...

---

# Программа

- Определения.
  - Возможности профилировщика.
  - Запуск.
  - Мониторинг приложения.
  - Анализ производительности.
  - Анализ использования памяти.
-

# Определения (1/2)

- *Куча (heap)* – область памяти, где размещаются объекты, создаваемые виртуальной машиной.
- *Сборка мусора* – удаление объектов, которые больше не используются.
- *Утечка памяти* – объект, который более не используется, но, тем не менее, не может быть удален сборщиком мусора.

# Определения (2/2)

- *Собственное время исполнения* – время, необходимое методу для выполнения своих инструкций. Время на выполнение вызываемых из него методов не учитывается.
  - *Hot spot* – метод, имеющий сравнительно большое значение собственного времени исполнения.
  - *Профилировщик* – утилита, показывающая параметры работы приложения.
  - *Профилитрование* – обнаружение проблем производительности измерительным методом.
-

---

# Цель профилирования

- По статистике, относительно небольшое число методов занимают больше всего процессорного времени.
    - Поиск таких методов и устранение задержек позволяет улучшить скорость работы приложения.
  - Неудаленные объекты, находясь все время в памяти, замедляют скорость работы приложения.
    - Обнаружение и устранение утечек памяти является важной составляющей повышения качества приложения.
-

# Состав профилировщика

## ■ Агент

- Загружается при старте виртуальной машины Java (Java Virtual Machine, JVM).
- Взаимодействует с JVM с использованием протокола JVM TI (Tool Interface).

## ■ Клиент

- Предоставляет пользовательский интерфейс.
- Взаимодействие с агентом:
  - переключение режимов профилирования,
  - запись результатов измерений в файл.
- Визуализирует результаты профилирования.



---

# Вопрос

Зачем заниматься исследованием потребления  
**памяти**, если существует автоматическая  
сборка мусора?

---

---

# Ответ

Эффективность потребления памяти Java программой может существенно влиять на производительность и надежность этой программы.

---

# Почему важно избавляться от утечек?

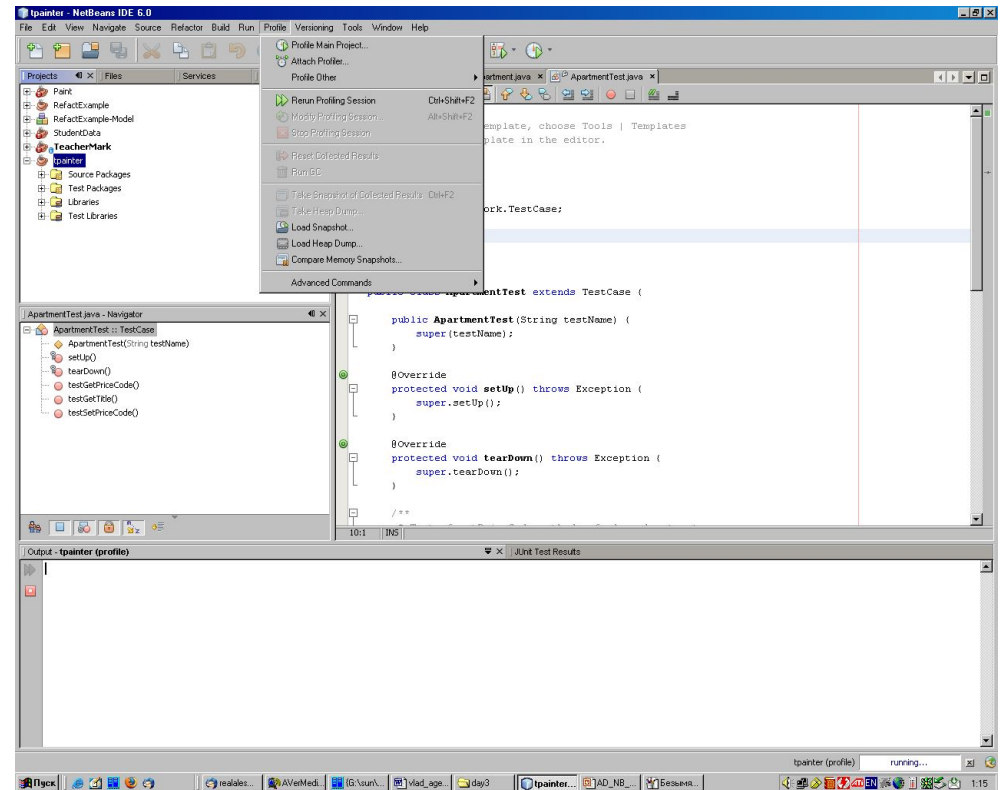
- В случае невозможности выделить память под новый объект возникает `OutOfMemoryError`.
  - Аварийное завершение программы или ее дальнейшая нестабильная работа.
  - Возможна потеря результатов работы и/или повреждение файлов данных.
- Использование больших объемов памяти может привести к увеличению частоты обращения ОС к файлу подкачки.
  - Общее снижение производительности системы.
- При создании большого количества временных объектов VM тратит больше времени на сборку мусора.

# Запуск

- С версии NetBeans 6.0 профилировщик входит в состав среды разработки.
- Для начала профилирования проекта нужно выбрать соответствующий пункт меню.
- Провести калибровку (определение специфичных для данной среды работы условий) окружения.

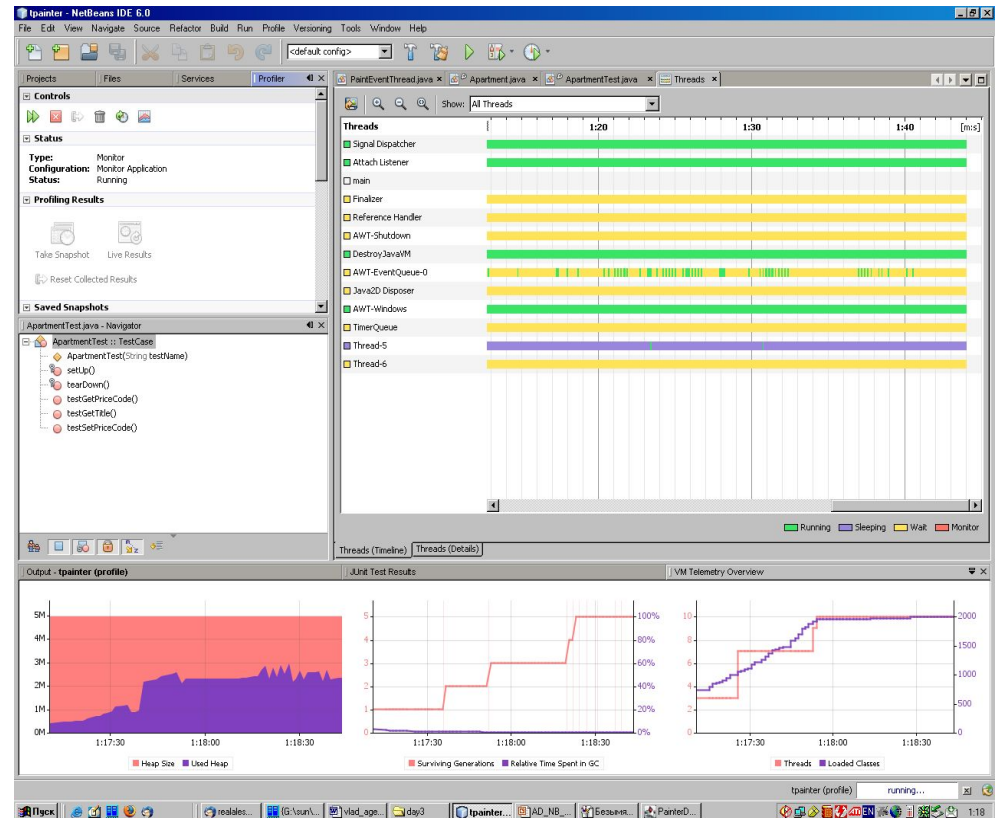
# Возможности профилировщика

- Взаимодействие с SE, EE и ME приложениями.
- Оценка скорости выполнения отдельных частей программы.
- Отслеживание создаваемых Java объектов.
- Поиск утечек памяти.
- Создание отчетов.



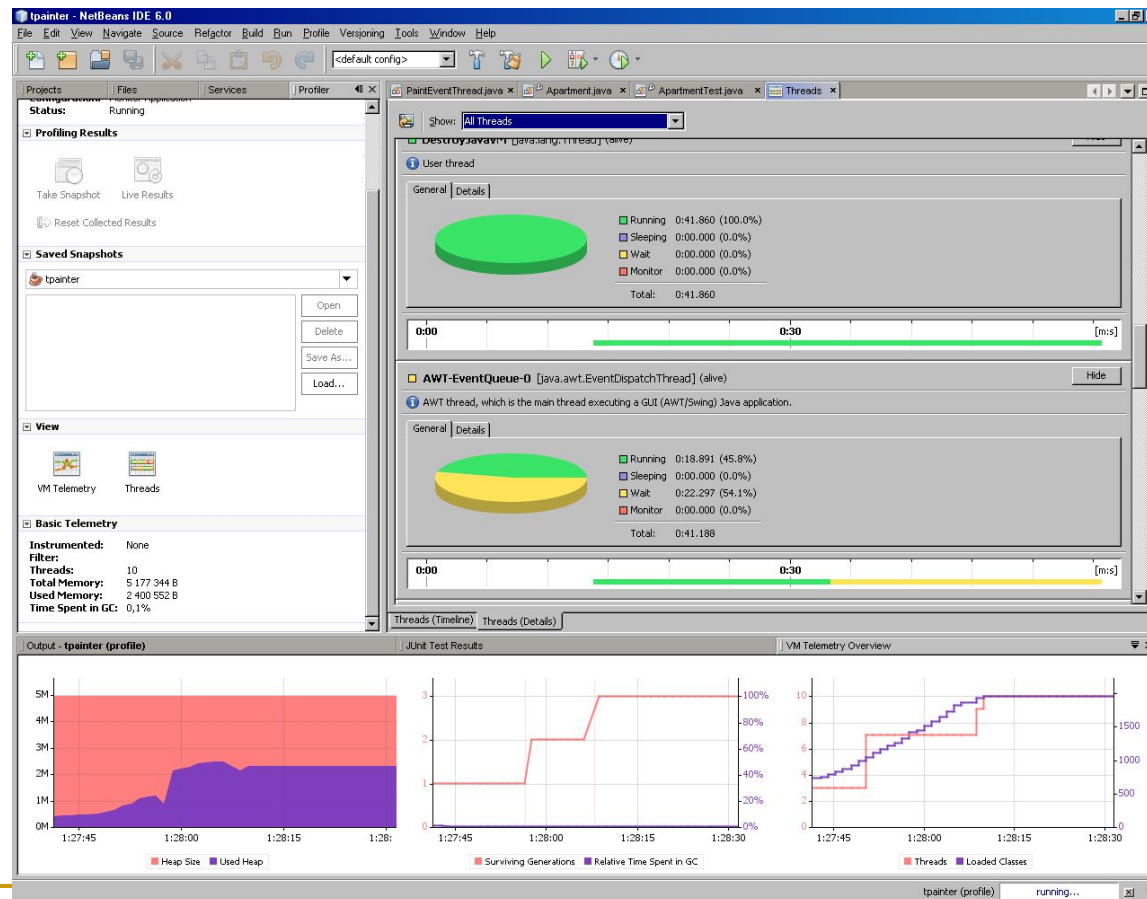
# Мониторинг приложения

- Полезен для получения общей статистики о работе приложения.
- Отображает в реальном времени сведения :
  - Об использованной памяти,
  - О работе сборщика мусора,
  - О потоках Java.



# Мониторинг приложения: статистика

Приводит наглядную статистику параметров работы.



---

# Мониторинг приложения: другие ВОЗМОЖНОСТИ

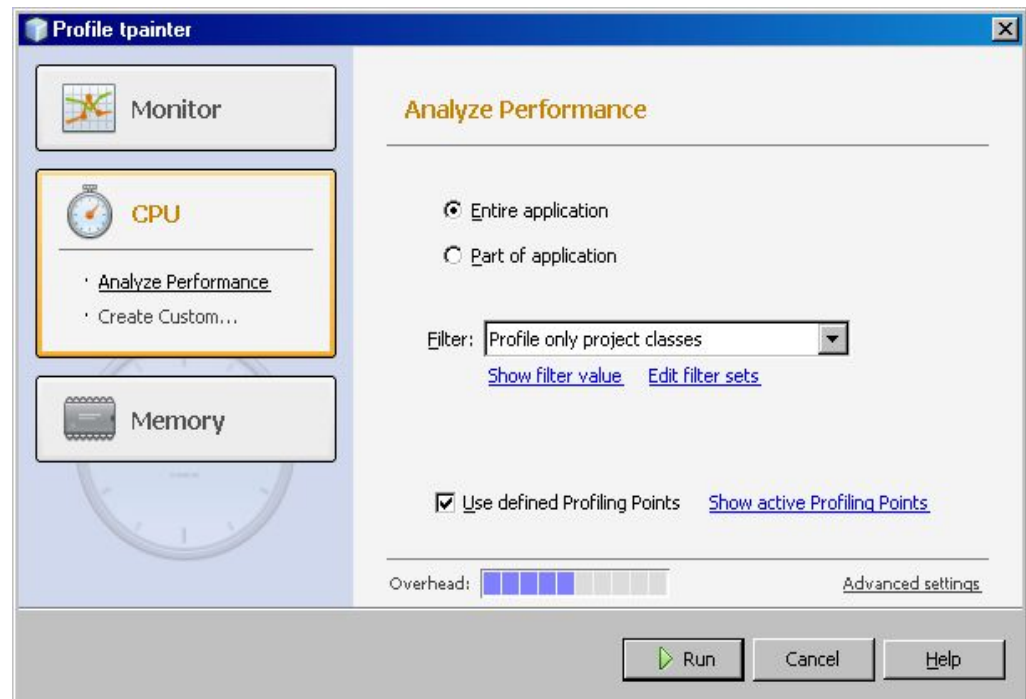
- Сохранение результатов измерений в файл для последующего просмотра.
- Вызов сборщика мусора.





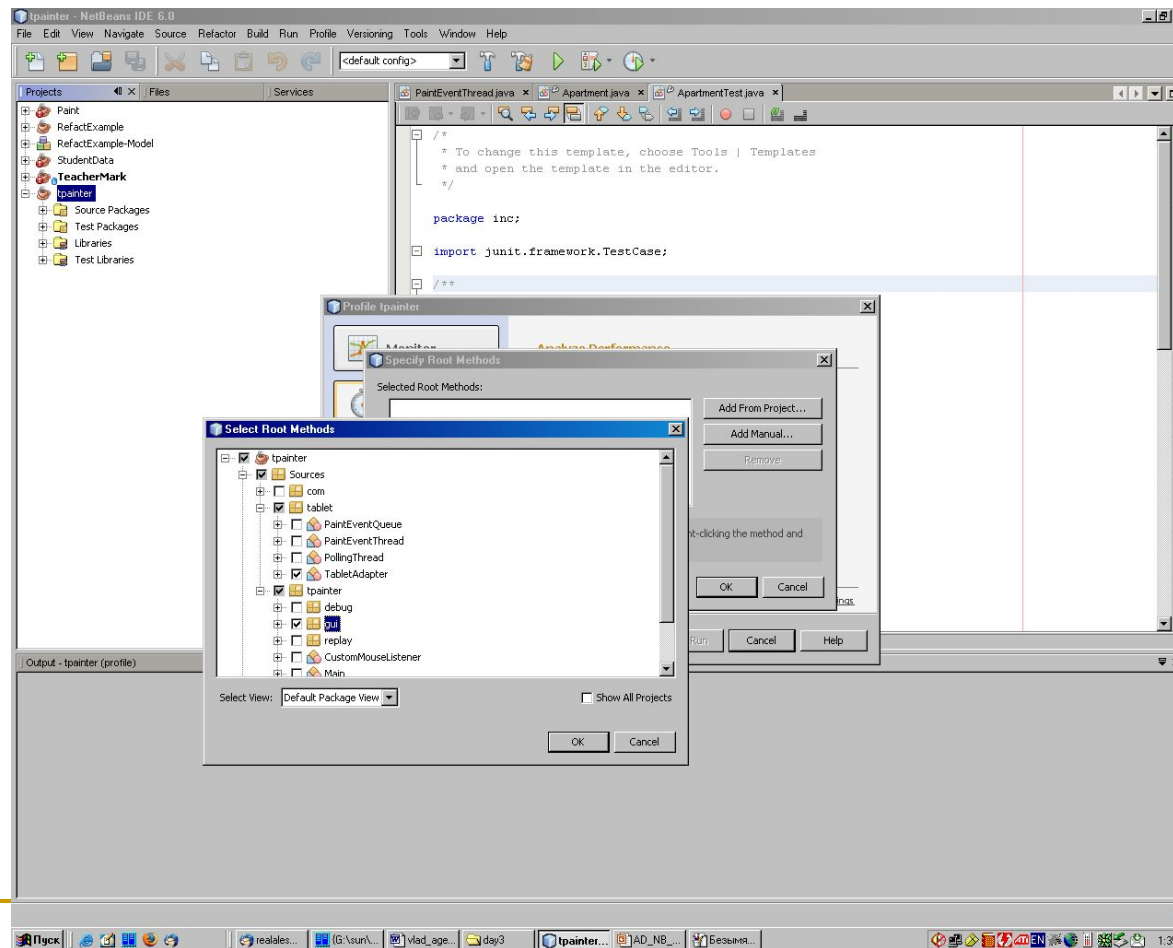
# Анализ производительности

- Необходим для получения детализированной информации о методах приложения:
  - Время исполнения,
  - Количество вызовов.
- Можно анализировать:
  - Все приложение,
  - Приложение и классы JDK,
  - Только некоторые методы.



# Производительность блоков кода

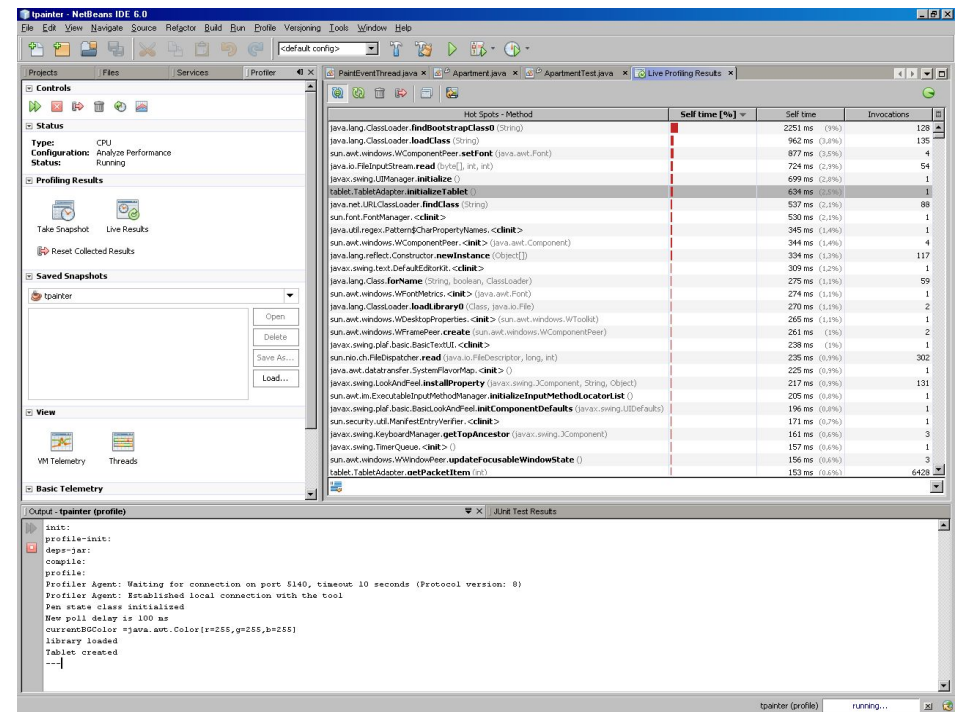
Для проведения анализа блока кода нужно выбрать интересующие классы и/или методы.



# Производительность: результаты в реальном времени

Каждый метод снабжен:

- временем исполнения,
- количеством вызовов,
- процентной шкалой от всего времени работы приложения.



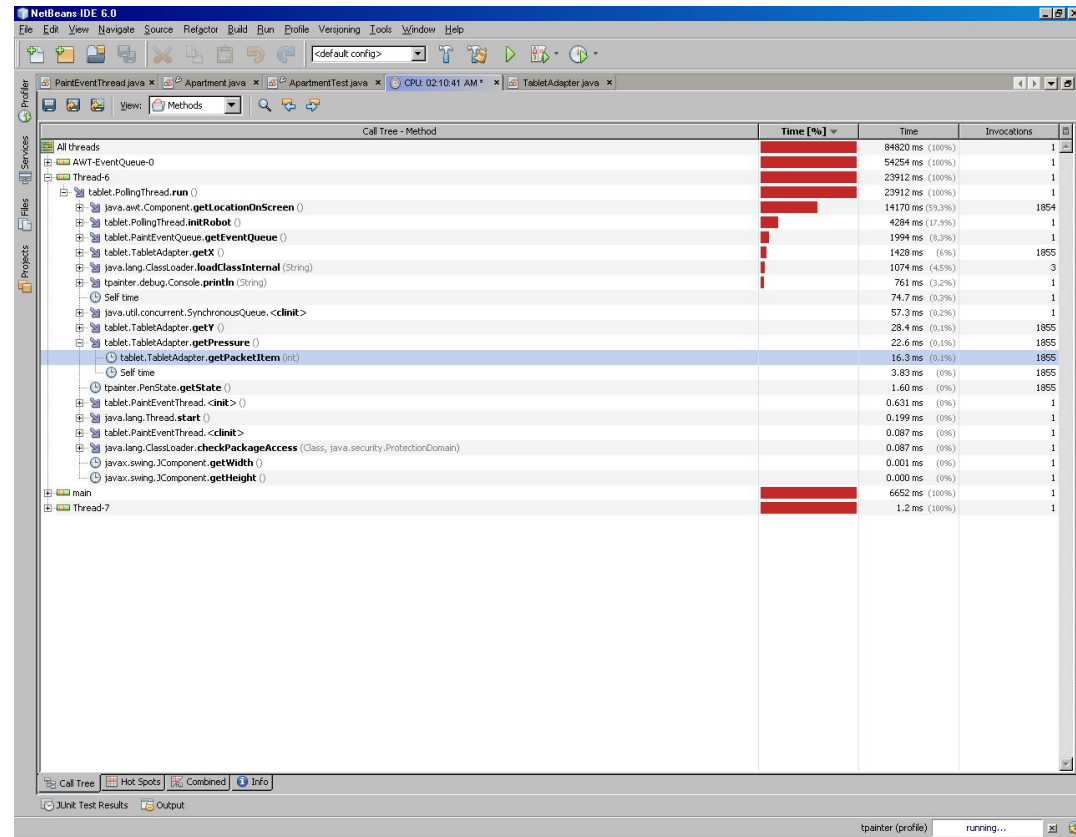
---

# Производительность: сохраненные результаты

- Для более детального анализа можно сделать слепок состояния приложения.
  - По сохраненным данным анализа можно построить:
    - Дерево вызовов.
    - Отсортированный список всех методов.
-

# Дерево вызовов

- Все вызванные методы помещаются в дерево
- Каждому узлу ставится в соответствие время работы и количество вызовов.



# Отсортированный список всех методов

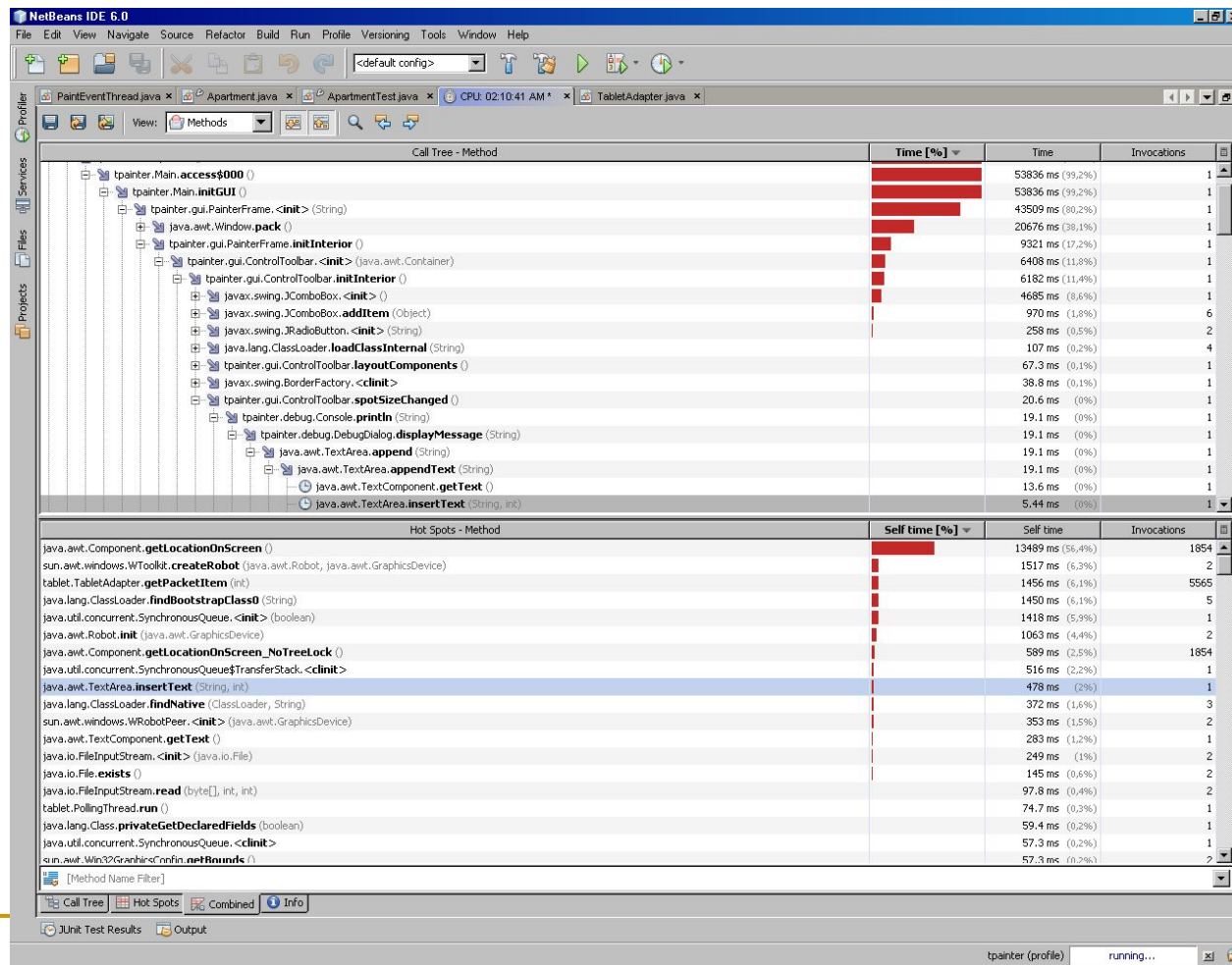
Каждый метод снабжен:

- временем исполнения,
- количеством вызовов,
- процентной шкалой от всего времени работы приложения.

Method	Self time [%]	Self time	Invocations
java.awt.Component.getLocationOnScreen ()	15.9%	13489 ms	1854
javax.swing.plaf.basic.BasicMenuItem.layoutMenuItem (java.awt.FontMetrics, String, java.awt.FontMetrics, String, javax.swing.Icon, javax.swing.Icon, javax.swing.I...	7.2%	6187 ms	20
java.lang.ClassLoader.findBootstrapClass0 (String)	5.5%	4643 ms	157
sun.awt.windows.WFramePeer.reshape (int, int, int, int)	4.4%	3918 ms	3
sun.awt.windows.WComponentPeer.show ()	3.6%	3058 ms	3
javax.swing.plaf.metal.MetalComboBoxUI.layoutComboBox (java.awt.Container, javax.swing.plaf.metal.MetalComboBoxUI.MetalComboBoxLayoutManager)	2.7%	2255 ms	1
java.lang.Class.privateGetDeclaredMethods (boolean)	2%	1726 ms	47
sun.awt.windows.WToolkit.createRobot (java.awt.Robot, java.awt.GraphicsDevice)	1.8%	1517 ms	2
java.awt.TextArea.insertText (String, int)	1.7%	1481 ms	6
tablet.TabletAdapter.getPacketItem (int)	1.7%	1456 ms	5566
java.util.concurrent.SynchronousQueue.<init> () (boolean)	1.7%	1418 ms	1
java.lang.ClassLoader.loadLibrary0 (Class, java.io.File)	1.6%	1346 ms	15
sun.awt.windows.WComponentPeer.updateCursorImmediately ()	1.3%	1139 ms	1
java.awt.Robot.init (java.awt.GraphicsDevice)	1.3%	1063 ms	2
java.io.File.exists ()	1.1%	919 ms	86
java.io.FileInputStream.read (byte[], int, int)	1%	849 ms	60
sun.java2d.windows.Win32SurfaceData.<init> (sun.awt.windows.WComponentPeer, sun.java2d.loops.SurfaceType, int)	1%	837 ms	10
javax.swing.JMenu.isTopLevelMenu ()	1%	828 ms	130
java.awt.TextArea.append (String)	0.8%	719 ms	6
java.lang.Thread.start ()	0.8%	687 ms	3
sun.java2d.loops.GraphicsPrimitiveMgr.<clinit>	0.8%	666 ms	1
sun.awt.Win32GraphicsEnvironment.initDisplayWrapper ()	0.7%	603 ms	1
java.awt.Component.getLocationOnScreen_NoTreeLock ()	0.7%	589 ms	1854
java.io.FileInputStream.<init> (java.io.File)	0.6%	534 ms	46
sun.font.FontDesignMetrics.stringWidth (String)	0.6%	532 ms	29
java.util.concurrent.SynchronousQueue\$TransferStack.<clinit>	0.6%	516 ms	1
sun.awt.windows.WComponentPeer.setFont (java.awt.Font)	0.6%	509 ms	4
sun.swing.SwingUtilities2\$BearingCacheEntry.getLeftSideBearing (char)	0.6%	508 ms	29
java.lang.ClassLoader.findNative (ClassLoader, String)	0.6%	485 ms	113
sun.nio.ch.FileDispatcher.read (java.io.FileDescriptor, long, int)	0.6%	482 ms	302
sun.util.resources.TimeZoneNames.getContents ()	0.5%	436 ms	1
sun.font.FontManager.<clinit>	0.5%	424 ms	1
sun.awt.GlobaCursorManager.<init> ()	0.5%	414 ms	1
sun.java2d.SurfaceData.<clinit>	0.5%	404 ms	1
tpainter.Main.initGUI ()	0.4%	372 ms	1
tablet.TabletAdapter.initializeTablet ()	0.4%	358 ms	1
sun.awt.windows.WRobotPeer.<init> (java.awt.GraphicsDevice)	0.4%	353 ms	2
sun.net.www.protocol.file.FileURLConnection.initializeLoaders ()	0.4%	338 ms	1
sun.awt.windows.WToolkit.<clinit>	0.4%	336 ms	1
sun.font.FontStrike.getGlyphImagePtr (int)	0.4%	319 ms	58

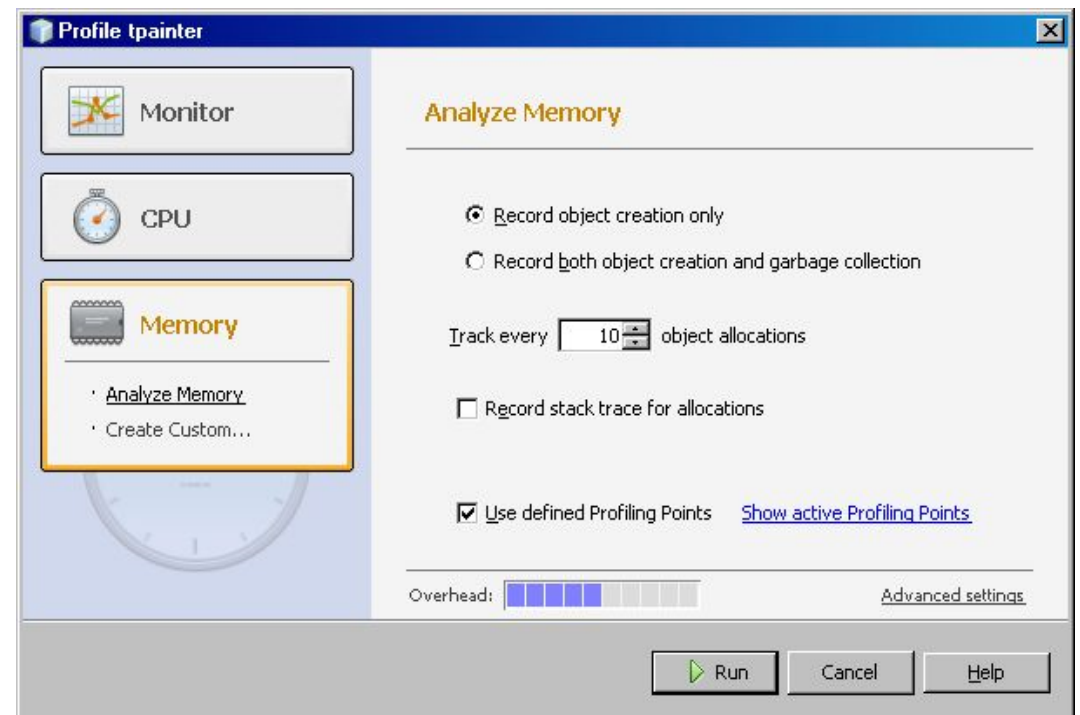
# Комбинированное представление

Отображается соответствие стека вызова имени метода.



# Анализ использования памяти

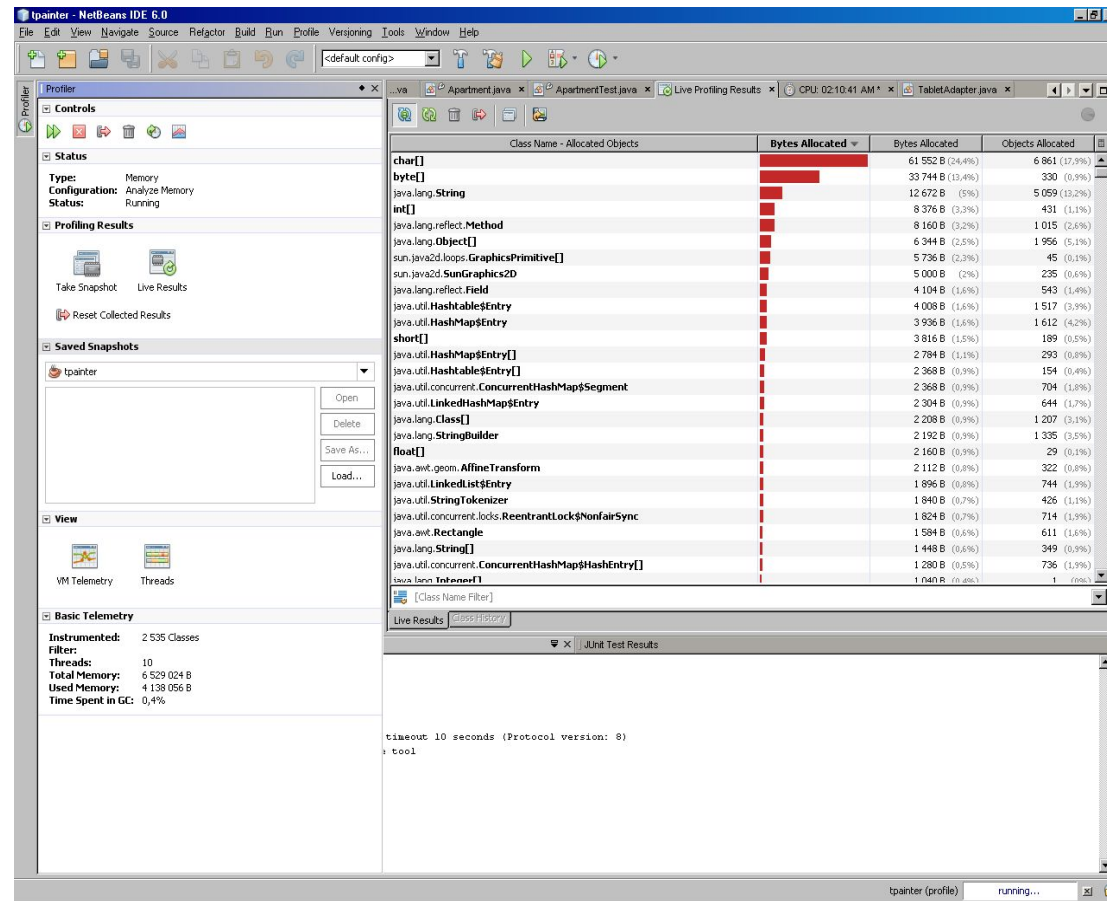
- Используется
  - > для оценки объема используемой памяти
  - > для нахождения неудаляемых объектов.





# Анализ памяти: результаты в реальном времени

- Отслеживание состава кучи (heap).
- Таблица содержит:
  - > Имя класса.
  - > Количество доступных (живых) объектов.
  - > Расход памяти (в байтах).
  - > Усредненное поколение\* объекта.
  - > Число поколений объекта.



\*Поколение - число, отражающее количество пережитых данным объектом сборок мусора)

# Анализ памяти: сохраненные результаты

Позволяет получить стек создания каждого объекта.

The screenshot displays the NetBeans IDE 6.0 interface with the Profiler tool active. The main window shows the 'Live Profiling Results' tab, which contains a table of memory allocation data. The table has four columns: 'Method Name - Allocation Call Tree', 'Bytes Allocated [...]', 'Bytes Allocated', and 'Objects Allocated'. Two rows are visible, both with red highlights in the 'Bytes Allocated' column.

Method Name - Allocation Call Tree	Bytes Allocated [...]	Bytes Allocated	Objects Allocated
javax.swing.JScrollPane		376 B (100%)	1 (100%)
componentcutoff.ComponentCutOff.main (String[])		376 B (100%)	1 (100%)

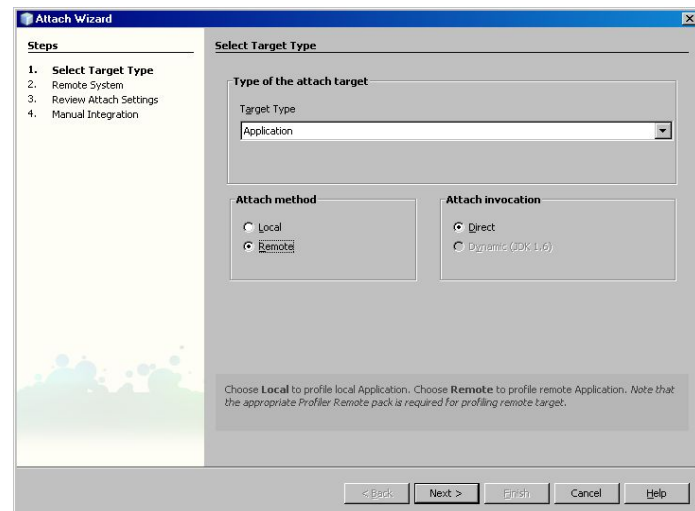
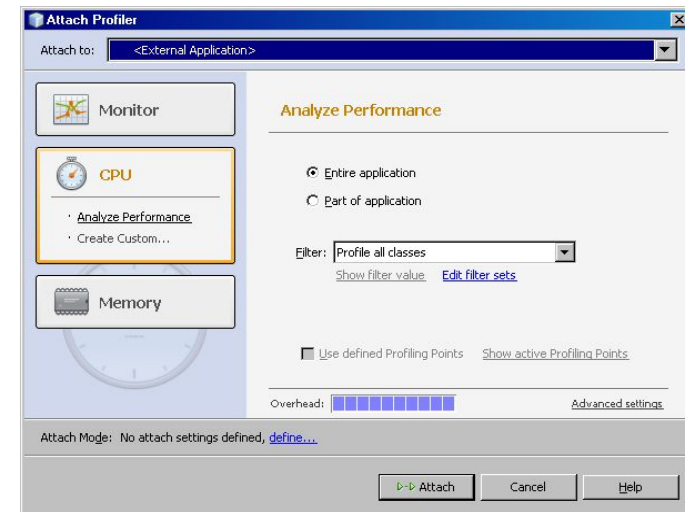
Below the table, the 'Output - ComponentCutOff (profile)' window shows the following log messages:

```
init:  
profile-init:  
deps-jar:  
compile:  
profile:  
Profiler Agent: Waiting for connection on port 5140, timeout 10 seconds (Protocol version: 8)  
Profiler Agent: Established local connection with the tool
```

The status bar at the bottom indicates 'ComponentCutOff (profile) running...'.

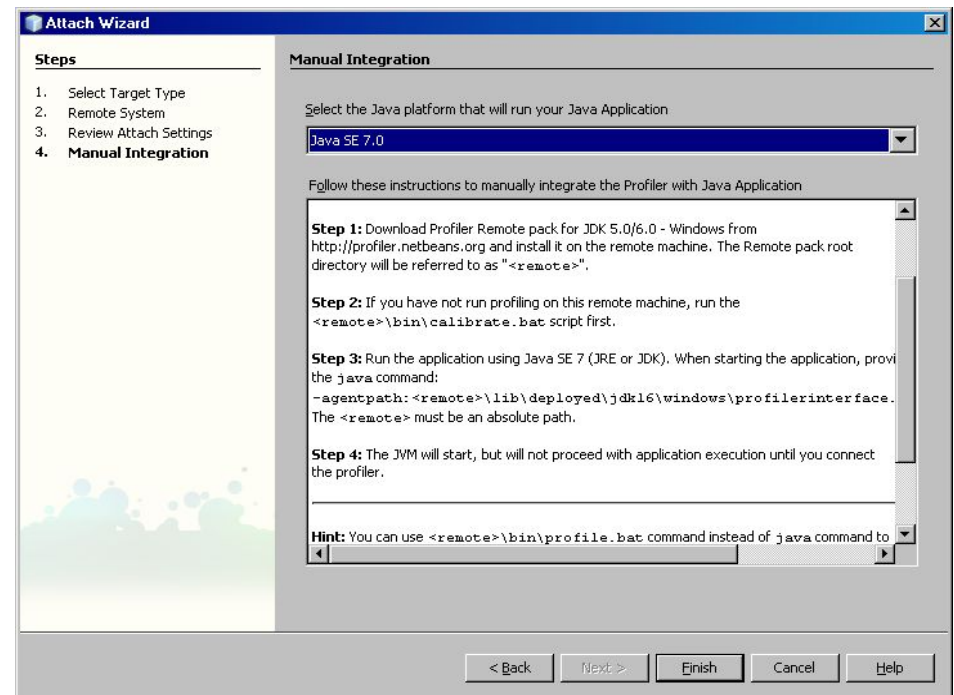
# Профилрование внешней программы

Среда разработки позволяет проводить анализ приложения, запущенного вне среды разработки.



# Профилирование внешней программы: запуск целевого приложения

- Для удаленного анализа потребуется запустить целевое приложение с указанными параметрами.
- Среда предоставляет рекомендованные параметры для использования.



---

# ССЫЛКИ

- Сайт NetBeans:
    - <http://netbeans.org/>
  - Описание профилировщика:
    - <http://www.netbeans.org/kb/60/java/profiler-intro.html>
  - Поиск утечек памяти:
    - [http://www.netbeans.org/kb/articles/nb-profiler-uncoveringleaks\\_pt1.html](http://www.netbeans.org/kb/articles/nb-profiler-uncoveringleaks_pt1.html)
  - Онлайн-курсы:
    - <http://javapassion.com/>
  - NetBeans IDE Field Guide by Patrick Keegan, Ludovic Champenois, etc.
  - Язык программирования Java и среда NetBeans
-

---

Q&A

---

Спасибо!

Дмитриев Андрей Владиславович  
[andrei-dmitriev@yandex.ru](mailto:andrei-dmitriev@yandex.ru)

