OptimizeIt 4.0

User guide

INTUITIVE SYSTEMS, INC. - SOFTWARE LICENSE AGREEMENT

INTUITIVE SYSTEMS, INC. ("LICENSOR") IS WILLING TO LICENSE THE ACCOMPANYING PROGRAM TO YOU ("LICENSEE") ONLY IF YOU ACCEPT ALL OF THE
TERMS IN THIS AGREEMENT. PLEASE READ THE TERMS CAREFULLY. BY INSTALLING THE PACKAGE FROM THE CD-ROM OR BY CLICKING ON THE BUTTON "YES"
BELOW, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT,
LICENSOR WILL NOT LICENSE THIS PROGRAM TO YOU. IN THIS CASE DO NOT INSTALL THE PACKAGE OR CLICK THE BUTTON "NO".

Definitions
1.1 "Agreement" shall mean this Agreement between Licensor and Licensee.
1.2. "Documentation" shall mean the user manual(s) and any other materials supplied by Licensor for use with the Program .
1.3 "Program" shall mean the machine-readable object code of OptimizeIt together with its Documentation.

Grant of License
2. Licensor hereby grants to Licensee, and Licensee hereby accepts, a permanent non-exclusive license to use the Program subject to
the terms ad provisions of this Agreement.

3. The license granted by this Agreement authorizes use of the Program by no more that 1 concurrent user, unless expressly specified
in the materials supplied by Licensor to Licensee together with the Program.

4. Licensee acquires no right to distribute the Program and no right to copy the Program unless as specified in this Agreement

5. Licensee agrees not to decompile, disassemble or reverse engineer the Program.

6. Licensee shall have the right to make one copy of the machine-readable object code of the Program solely for archive purposes. On
such archival copy, Licensee shall mark copyright, trademark, patent, and/or trade secret notices identical to those on the copy of
the Program provided to Licensee. Licensee may not otherwise make copies of the Program.

Acknowledgment of Licensor's ownership rights

7. Licensee acknowledges that it obtains no ownership rights in the Program under the terms of this Agreement. All rights in the
Program including but not limited to trade secrets, trademarks, service marks, patents, and copyrights are, shall be and will remain
the property of Licensor or any third party from whom Licensor has licensed software or technology. All copies of the Program
delivered to Licensee or made by Licensee remain the property of Licensor.

Limited Warranty

8. Licensor warrants that the Program will perform substantially in accordance with accompanying Documentation for a period of ninety
(90) days from the date of Licensee's receipt of the program ("Warranty period"). Any implied warranties on the Program are limited to
ninety (90) days.

9. LICENSOR AND ANY THIRD PARTY FROM WHOM LICENSOR HAS LICENSED SOFTWARE OR TECHNOLOGY DISCLAIM(S) ALL OTHER WARRANTIES, EITHER
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NON-INFRINGEMENT, WITH RESPECT TO THE PROGRAM AND THE ACCOMPANYING WRITTEN MATERIALS.

10. LICENSOR AND ANY THIRD PARTY FROM WHOM LICENSOR HAS LICENSED SOFTWARE OR TECHNOLOGY WILL NOT BE LIABLE FOR LOST PROFITS, LOST
OPPORTUNITIES, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES UNDER ANY CIRCUMSTANCES.

11. EXCLUSIVE REMEDY: LICENSEE'S EXCLUSIVE REMEDY SHALL BE, AT LICENSOR'S CHOICE, EITHER (A) RETURN OF THE PRICE PAID OR (B)
REPLACEMENT OF THE PROGRAM THAT DOES NOT MEET LICENSOR'S LIMITED WARRANTY AND WHICH IS RETURNED TO LICENSOR WITH A COPY OF LICENSEE'S
RECEIPT. Any replacement Program will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is
longer. These remedies are not available outside the United States of America.

12. If any problem, operational failure or error of the Program has resulted from any alteration of the Program, accident, abuse, or
misapplication, then this warranty shall be null and void, at Licensor's option.

13. IN NO EVENT WILL LICENSOR BE LIABLE TO LICENSEE FOR DAMAGES, INCLUDING ANY LOSS OF PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OR INABILITY TO USE THE PROGRAM.

14. This Agreement is governed by the laws of the State of California.

15. U.S. Government Restricted Rights. This Program and documentation are provided with Restricted Rights. Use, duplication, or
disclosure by the Government is subject to restrictions set forth in subparagraph (c)(1) of the Rights in Technical Data and Computer
Software clause at DFARS 252.227-7013 or subparagraphs (c)(1)(ii) and (2) of Commercial Computer Software - restricted Rights at 48
CFR 52.227-19, as applicable.

1 Introducing OptimizeIt

1.1 What is OptimizeIt?

OptimizeIt™ software is a Java™ profiling tool which enables developers to test and improve the performance of their Java applications, applets, servlets, Javabeans™, Enterprise Javabeans™ and Java Server Pages™. OptimizeIt takes you behind the scenes of the Java virtual machine and reveals how a Java program uses computer resources. Using OptimizeIt allows developers to identify any Java code allocating too much memory or using the CPU in an inefficient way.

OptimizeIt is plug and play: there is no need to recompile your program with a custom compiler or to modify class files before execution. Just run your program from OptimizeIt to start testing its performance. Because no code modification is required, any Java code that your program uses is included in the profile.

OptimizeIt has two main components:

- The OptimizeIt user interface is a window that displays profiles and controls for refining the profiles and viewing source code.
- The OptimizeIt audit system is a real-time detective that reports the activity on the Java virtual machine back to the OptimizeIt user interface.

After you invoke your program from OptimizeIt, the OptimizeIt user interface connects to the audit system running in the test application's virtual machine and displays performance related information. For example, when an object is allocated by your program, the OptimizeIt window displays the allocation. You can click a button in the OptimizeIt window to display the source code responsible for allocating the object and click a button to display other information such as CPU usage.

At any time you can open the source code viewer to display and study relevant lines of code. For example, if you identify a performance bottleneck while viewing methods allocating objects, you can click a button to open the source code viewer displaying the problem code.

When you run OptimizeIt again after fixing source code, you can make sure that your new optimization is actually improving the performance of your

application. OptimizeIt removes guesswork from improving Java performance.

1.2 Getting started with OptimizeIt

The basic steps to run OptimizeIt are as follows:

- Create some new settings for the application you want to test
- Launch your test program from OptimizeIt
- Analyze memory use in the test program
- Analyze CPU use in the test program
- Make changes to your source code and repeat these steps until you are satisfied with your program's performance

These basic steps are detailed in the OptimizeIt Quick Tour, also accessible from the user interface. This user manual describes a more broad selection of OptimizeIt tasks. It also includes information about OptimizeIt windows and options.

We strongly recommend that you spend 10 minutes with the OptimizeIt Quick Tour tutorial to understand OptimizeIt quickly.

1.3 Differences when profiling with Java and Java 2

Some OptimizeIt features are different when using Java (JDK 1.1) or Java 2 (JDK 1.2 or JDK 1.3). For example, OptimizeIt provides more features with Java 2. The following logos are used within the documentation when a paragraph only applies to Java or Java 2:

1.4 OptimizeIt main features

Memory Profiler

- Provides real-time display of all classes used by the test program and of the number of allocated instances
- Graphically indicates where instances are allocated
- Filters class lists so you can focus on relevant classes
- Automatically highlights lines of code that allocate object instances
- Gives you control over garbage collection
- Displays incoming and outgoing object references in real time
- Displays string representations of allocated instances
- Provides Java API calls so you can invoke the memory profiler from inside the test program

Java 2

- Computes reduced reference graph for incoming references
- Displays references from reference graph roots

CPU profiler

- Allows you to start or stop profiling at any time
- Displays data for pure CPU use or for elapsed time (pure CPU and inactive phases)
- Graphically represents thread activities for the sampling period
- Displays profiling information for each thread or thread group
- Finds frequently used methods using Hot spot detectors
- Provides Java API calls so you can invoke the CPU profiler from inside the test program

Java 2

- Provides both a sampler based profiler and an instrumentation based profiler
- Can provide millisecond or microsecond precision
- Displays invocation count
- Provides filter to remove fast methods

Other features

- Provides filters for both memory and CPU profiler
- Displays and exports charts showing high level VM information including heap size, heap used, number of threads, number of busy threads, number of loaded classes
- Starts any Java application, applet or servlet directly from OptimizeIt user interface

- Saves snapshots of a profiling session at any time. Snapshots can be reloaded later for analysis or comparison of profilings
- Provides an offline profiling mode to automatically save snapshots at fixed intervals
- Integrates automatically with several IDEs including IBM VisualAge, WebGain VisualCafe, Borland JBuilder, Oracle JDeveloper
- Integrates automatically with several Application Servers including JRun 2.3 and 3.0, WebLogic 4.5, 5.0 and 5.1, iPlanet Web Server (NES 4.0 and 4.1), JServ 1.1, Java Web Server 1.13 and 2.0, ServletExec 2.2 and 3.0, Jakarta Tomcat.
- Provides compatibility with IBM WebSphere, Apple WebObjects and Netscape Application Server (NAS), ATG Dynamo, SilverStream, GemStone
- Pauses and resumes the execution of the test program
- Provides a find tool that can be used in all screens
- Highlights relevant lines of code with the source code viewer
- Includes a Java setup wizard for fast and simple configuration
- Executes your test program remotely while analyzing performance
- Exports any information in HTML or ASCII
- Provides Java API calls so you can invoke the OptimizeIt audit system from any Java code

Java 2

- Supports any virtual machine that is fully JVMPI compliant. This includes any Sun virtual machine derived from JDK 1.1, 1.2, 1.3 and also IBM JDK 1.2 and 1.3
- Displays and exports chart showing garbage collector load factor.
- Starts applications directly from JAR files

2 Configuring OptimizeIt

This chapter describes how to configure the OptimizeIt software for operation with your Java application. It includes the following sections:

These settings are global settings that can be customized for each applet or application you are profiling.

2.1 Selecting a virtual machine

OptimizeIt profiles your program's performance while it runs on a Java virtual machine. By default OptimizeIt profiles using a Java 2 runtime.

OptimizeIt is compatible with most Java version 1.1, 1.2 and 1.3 virtual machines including the following:

- Sun Microsystems® Java Development Kit (JDK) Version 1.1.6 or newer, 1.2 and 1.3
- Borland® JBuilder™ Version 1.0 or higher
- Symantec® Cafe Version 2.5 or 3.0
- IBM® Java Development Kit JDK 1.2.2 and 1.3

OptimizeIt includes a Java Runtime Environment (JRE) 1.2. The first time you run OptimizeIt, a Java setup wizard is started. If you want to use the default JRE 1.2 included with OptimizeIt, just click Cancel. If you want to configure OptimizeIt to use a different virtual machine, click Next. Select the directory where you want the wizard to search for available virtual machines. The wizard scans the selected directory or drive and lists all available virtual machines (after you see the message "Found 1 virtual machine" you can click Stop to end the scan). Select the virtual machine you want to use in the list of found virtual machines, click Next, then click Finish.

After changing the default virtual machine, OptimizeIt prompts you to ask whether the new default virtual machine should be used with the current settings. Click Yes to start using that virtual machine.

At any time, you can change which virtual machine OptimizeIt uses from the Virtual machine tab of the Settings editor, See “Adding a virtual machine” on page 19.

2.2 Setting the source code location

When OptimizeIt has access to the source code for an application, it highlights the relevant lines in the source code. If the source code is not accessible, OptimizeIt can still provide profiling information for the Java classes.

OptimizeIt maintains a source path which is a list of directories containing source code. OptimizeIt searches each directory in the source path in the order specified when it requires a source file. The Preferences panel can be used to change the default source path in OptimizeIt.

Changing the default source path

1. Select Preferences from the Edit menu.
2. Select Default source path in the top selection box.
3. Click the Edit button.
4. The Source Path Chooser opens.
5. In the top box, select the directory you want to add.
6. Choose the directory that contains the top-level package of your Java source code. If you aren't sure, select any Java file in your application and OptimizeIt will add the appropriate directory.
7. Click the down arrow to add your selection to the source path.
8. Repeat these steps for other directories you want to add to the source path.
9. Click the OK button.
10. Click the OK button to close the Preferences dialog box.

Note: *When a Java file is not available, OptimizeIt prompts you to locate the missing Java file. After you select the missing Java file, a dialog box prompts you to add the file's directory to the default source path.*

To set a different source path for an applet, a servlet or an application that you are profiling with OptimizeIt, define the changes to the default source path in the Settings editor.

2.3 Changing the class path

When OptimizeIt launches your applet or application, the Java virtual machine needs the location of classes your application uses. By default, OptimizeIt points to the classes that are defined in your CLASSPATH environment variable.

OptimizeIt maintains a class path which is a list of directories containing class files. OptimizeIt searches each directory in the class path in the order specified when it requires a class. If you have some classes, zip files or jar files that you always want available to all test programs, add them to the default OptimizeIt class path.

Changing the class path

1. Select Preferences from the Edit menu.
2. Select Default class path in the top selection box.
3. Uncheck the Use CLASSPATH environment variable option.
4. Click the Edit button.
5. In the top box, select the directory, zip file or jar file you want to add to the class path. If you choose a directory, make sure you select the directory containing the top-level package of the source tree.
6. Click the down arrow to add your selection in the class path.
7. Click the OK button.
8. Click the OK button to close the Preferences dialog box.

Note: *To set a different class path for an applet or application you are profiling with OptimizeIt, define the changes to the default class path in the Settings Editor.*

2.4 Configuring servlet support

In order to be able to start the profiling of a servlet directly from OptimizeIt, you need to configure OptimizeIt with a servlet runner. You need one of the following servlet runner jar file: jsdk.jar, server.jar or web-server.jar. These jar files can be found in several software installations: JSDK 2.0 or 2.1, JSWDK 1.0, JBuilder 3, JDeveloper 2, VisualCafe 3 or 4, WebSphere. If you don't have any of this software installed, we suggest that you download and install the Java Servlet Development Kit 2.1 (JSDK 2.1) from http://java.sun.com/products/servlet/download.html.

Configuring servlet support

1. Select Preferences from the Edit menu.
2. Select Servlet in the top selection box.
3. Click on Servlet setup. This opens the servlet configuration wizard.
4. Click on next.
5. Select a directory where you want to search for one of the servlet runner jar file then click on next.
6. When OptimizeIt has finished its search, it displays the available jar files found in a table. Select one line in the table then click next.
7. Click finish.

Changing the port used by the servlet runner

When you start the profiling of a servlet from OptimizeIt, OptimizeIt runs your servlet in a servlet runner. The default port used by the servlet runner is 8080. If you have another application that already uses that port, change the port number:

1. Select Preferences from the Edit menu.
2. Select Servlet in the top selection box.
3. Change the value of the port in the corresponding section.

3 Profiling a Java program

3.1 Starting a Java Application

OptimizeIt can profile a Java application that is either packaged in a JAR file, or is given the location of the class file containing the Main method for the application.

Note: *By default OptimizeIt profiles your application with a Java 2 runtime. You can select another virtual machine by using the Virtual Machine tab, see "Virtual machine options" in chapter 3.6.*

To start profiling an application

1. Select New from the File menu.
2. In the Program type section of the dialog box, choose Application.
3. Enter the main class of the application.
- If the application is in a JAR file, click Browse to select the JAR file location.
- If the application is in a ZIP file, enter the fully qualified name of the class containing the Main method. For example:
com.foo.bar.Main
- If the application is not in a JAR or ZIP file, click Browse to select the class file that contains the main method.

4. (Optional) Make sure the working directory is correct. If the application does not require a working directory, ignore this option.
5. (Optional) Add classes to the class path. If the application requires special classes not indicated in the default class path, click Change under the Class path list to select directories, jar file or zip files containing the extra classes.
6. (Optional) Add source code directories to the source path. This allows OptimizeIt to show relevant source code. Specific files can be added later during the profiling session.
7. (Optional) Click the Virtual Machines tab. Select the virtual machine you want to use in the list. For more information on the different options available in this tab, see "Virtual machine options" in chapter 3.6.
8. Click the Start Now button. OptimizeIt starts the application and opens the memory profiler for the application.

Once you have configured OptimizeIt to start your application, you can save this configuration and reopen it later. Save and Open commands are available from the file menu.

3.2 Starting a Java Applet

OptimizeIt can profile a Java applet given an HTML file or a URL.

Note: *Profiling an applet requires a Java Development Kit (JDK). Make sure to select a virtual machine included in a JDK from the Virtual Machine tab, see "Virtual machine options" in chapter 3.6.*

To start profiling an applet

1. Select New from the File menu.
2. In the Program type section of the dialog box, choose Applet.
3. Enter the file name or URL of the applet.
4. If the applet is on your local disk, click the Browse button and select the HTML file. If the applet is a web page, enter the URL of that page.
5. (Optional) Make sure the working directory is correct. If the applet does not require a specific working directory, ignore this option.
6. (Optional) Add classes to the class path. If the applet requires special classes not indicated in the default class path, click Change under the Class path list to select directories, jar files or zip files containing the extra classes.
7. (Optional) Add source code directories to the source path. This allows OptimizeIt to show relevant source code. Specific files can be added later during the profiling session.
8. (Optional) Click the Virtual Machines tab. Select the virtual machine you want to use in the list. For more information on the different options available in this tab, see "Virtual machine options" in chapter 3.6.
9. Click the Start Now button. OptimizeIt starts the applet and opens the memory profiler for the applet.

Once you have configured OptimizeIt to start your applet, you can save this configuration and reopen it later. Save and Open commands are available from the file menu.

Note: *When started from OptimizeIt, an applet is run with AppletViewer. It is also possible to profile an applet that run inside a web browser when using the Java plug-in. This operation is described in the tutorial "Profiling an applet running inside a web browser".*

3.3 Starting a Java Servlet

OptimizeIt can profile a Java servlet given its class.

Note: *By default OptimizeIt profiles your application with a Java 2 runtime. You can select another virtual machine by using the Virtual Machine tab, see "Virtual machine options" in chapter 3.6.*

To start profiling a servlet

1. Select New from the File menu.
2. In the Program type section of the dialog box, choose Servlet.
3. If it is the first time you have profiled a servlet with OptimizeIt, the servlet wizard will start. The wizard will guide you through the servlet configuration, see "Configuring servlet support" in chapter 2.4 for more information.
4. Click Browse to select the class file that contains your servlet.
5. (Optional) Make sure the working directory is correct. If the servlet does not require a specific working directory, ignore this option.
6. (Optional) Add any parameters required by the servlet.
7. (Optional) Add classes to the class path. If the servlet requires special classes not indicated in the default class path, click Change under the Class path list to select directories, jar files and zip files containing the extra classes.
8. (Optional) Add source code directories to the source path. This allows OptimizeIt to show relevant source code. Specific files can be added later during the profiling session.
9. Click the Virtual Machines tab. Select the virtual machine you want to use in the list. For more information on the different options available in this tab, see "Virtual machine options" in chapter 3.6.
10. Click the Start Now button. OptimizeIt starts the servlet and opens the memory profiler for the servlet. The browser is automatically launched to execute the servlet. If any servlet parameters have been specified, they are added to the servlet URL.

3.4 Profiling EJBs or JSPs

Profiling EJBs or JSPs can be done by profiling the application server that runs your Java code. OptimizeIt can be integrated with most application servers. For more information, see “Integration with application servers” in chapter 8.1.

3.5 Start options

OptimizeIt allows you to control profiling through several options. These options affect the Java application selected in the Edit settings panel.

Option	Description
Pause after launch	Pauses the test program just before executing the Main method. Use this option to give yourself some time to configure OptimizeIt or to start some profilers before the tested application starts.
VM cannot exit	Disables the method System.exit() in the virtual machine. Use this option to test a command line program such as a compiler that performs a task and then exits the running virtual machine. Use the Stop button to exit the program when your profiling is complete.
Disable memory profiler	Disables the OptimizeIt memory profiler. The OptimizeIt memory profiler adds overhead that can change the CPU profiler results. Use this option when you are focusing on CPU-related issues only.
Open a console	Opens a console for the program. Use this option if your program expects some input from System.in. When this option is off, anything printed using System.out and System.err is printed in the OptimizeIt console.
Enable audit API	Enables the OptimizeIt audit system API. When the API is enabled, OptimizeIt memory and CPU profilers are disabled by default. This allows the tested program to use OptimizeIt API to precisely enable both profilers when needed.

Option	Description
Auto-start CPU profiler	Starts the CPU profiler just before executing the main method. The CPU profiler is started with the current option selected in the CPU profiler inspector.
Extra Java parameters	Specifies a string passed directly to the virtual machine running the test program. Use this field to add Java virtual machine arguments such as -mx or -verbosegc.
Extra program parameters	Specifies a string passed directly to the tested application when launched.
Class path	Lists the class path defined in the default class path. If the test program requires extra classes, JAR files, or ZIP files, click the Change button to select additions to the class path. These additions apply to this test program only.
Source path	Lists the directories in which OptimizeIt searches for source code. If the test program source code is not included in the default source path, click the Change button to add a location to the source path. These additions apply to this test program only.

3.6 Virtual machine options

The OptimizeIt Virtual Machine Tab allows you to select and configure the virtual machine you want to use to profile your application.

Adding a virtual machine

In order to add a virtual machine click on the Add virtual machines button. This starts the virtual machine wizard:

1. Click on Next.
2. Select a directory from which OptimizeIt will look for available virtual machines.
3. Click on search.
4. Once OptimizeIt has finished its search, it displays the virtual machines found.
5. Click on finish to add those virtual machines to the list of available virtual machines.

Setting the virtual machine properties

When you select a virtual machine from the list, OptimizeIt only enables the available options for that virtual machine.

Java runtime

This property sets the virtual machine runtime used. The following table shows which flag is added to the virtual machine invocation:

Option	Flag added	Description
Default	No flag added	The virtual machine uses its default runtime.
Classic	-classic	OptimizeIt forces the virtual machine to use its classic runtime.
Hotspot	-hotspot	OptimizeIt forces the virtual machine to use its Hotspot runtime.

Audit System

This option sets which library OptimizeIt should use. OptimizeIt automatically selects the appropriate library when the selected virtual machine is recognized. See “OptimizeIt libraries” on page 25 for more information on the different libraries.

Enabling the JIT

You can enable the JIT (Just In Time compiler) when profiling by selecting the corresponding option.

Note: *The JIT can only be enabled when the Universal JVMPI/JNI audit system is selected.*

3.7 Profiling a program started from the command line

In addition to running your test program from OptimizeIt, you can run it from a command prompt.

Invoking your application from outside OptimizeIt allows you to do the following:

- Set custom variables
- Run the application as part of a script
- Run the program on a different machine

To profile your test program invoked from a command prompt:

1. Set up the OptimizeIt audit system
2. Launch the OptimizeIt audit system, specifying your test program
3. Run OptimizeIt
4. Connect the audit system to the OptimizeIt user interface

Setting up the OptimizeIt audit system

To use the OptimizeIt audit system, you need to modify your CLASSPATH and PATH environment variables to include the following directories:

CLASSPATH	<InstallDir>\OptimizeIt\lib\optit.jar
PATH	<InstallDir>\OptimizeIt\lib

Launching your program

The OptimizeIt audit system is a set of Java classes and native code.

To show the audit system options

Run the following command:

```
C:\> java intuitive.audit.Audit
```

Java

To run the audit system with JDK 1.1

Invoke the audit system with the following options:

```
C:\> java -noclassgc -Djava.compiler=NONE intuitive.audit.Audit [-port portNumber] [-pause] [-wait] [-dmp] [-noexit] [-startCPUprofiler[:<CPUProfilerOptions>]] [-offlineprofiling[:<offlineprofilingOptions>]] [-enableAPI] ClassName arg1, arg2, ...
```

Note: *With early Java versions the OptimizeIt audit system requires that the class garbage collection is disabled. Always use the -noclassgc Java virtual machine argument when starting an application from the command line. You also need to disable the JIT. This can be done by adding -Djava.compiler=NONE*

Java 2

To run the audit system with JDK 1.2

Invoke the audit system with the following options:

```
C:\> java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE intuitive.audit.Audit [-port portNumber] [-pause] [-wait] [-dmp] [-noexit] [-startCPUprofiler[:<CPUProfilerOptions>]] [-offlineprofiling[:<offlineprofilingOptions>]] [-enableAPI] ClassName arg1, arg2, ...
```

Note: *With some environments, the -classic option is necessary to force the classic runtime. -Xrunoii is necessary to load OptimizeIt's JVMPI agent. The OptimizeIt audit system requires that the class garbage collection is disabled. Always use the -Xnoclassgc Java virtual machine argument when starting an application from the command line. You also need to disable the JIT. This can be done by using -Djava.compiler=NONE.*

Note: *Running an applet with JDK 1.2 requires that you use oldjava.exe instead of java.exe*

To run the audit system with JDK 1.3

Invoke the audit system with the following options:

```
C:\> java -classic -Xrunoii intuitive.audit.Audit [-port portNumber] [-pause] [-wait] [-dmp] [-noexit] [-startCPUprofiler[:<CPUProfilerOptions>]] [-offlineprofiling[:<offlineprofilingOptions>]] [-enableAPI] ClassName arg1, arg2, ...
```

Audit system options

The following table describes these parameters. Note that many of these options provide the same functionality as the Start options.

Option	Description
-port	Specifies the port you want to use for the communication link between the audit utility and the OptimizeIt application.
-pause	Causes the launched program to be paused immediately after launch.
-dmp	Disables the memory profiler.
-noexit	Disables the method System.exit() in the virtual machine.
-enableAPI	Enables the OptimizeIt audit system API. When the API is enabled, OptimizeIt memory and CPU profiler are disabled. The audit system waits for the test program to enable profilers from Java.
-startCPUprofiler	Starts the CPU profiler just before executing the main method, See “Starting the CPU profiler from the command line” on page 31 for more information on the available options.
-offlineprofiling	Starts the profiling in offline mode. Snapshots are generated automatically at a given time, see “Offline profiling” in chapter 3.9 for more information on the available options.
-dllpath	Specifies the location of OptimizeIt DLLs if different from your PATH environment variable. Note that this option has been deprecated since OptimizeIt 3.1. You should not need to use this option.
ClassName	The main class the for the test program. If the test program is an applet, use sun.applet.AppletViewer and then add the path of the applet HTML file or URL.

OptimizeIt libraries

OptimizeIt comes with several libraries to support the different virtual machines:

Name	Option	Description
Java 2 Universal	-DAUDIT=jni	This library provides universal virtual machine support. It supports any JDK 1.2 or later virtual machine that is fully JVMPI/JNI compliant. It also supports Just In Time compilers (JIT). It should be used with Sun's JDK 1.3 and IBM JDK 1.2.2 and 1.3
JDK 1.2 Compatible	-DAUDIT=12	This library should be used with Sun's JDK 1.2.x
JDK 1.1 Compatible	-DAUDIT=11	This library should be used with JDK 1.1

The audit system automatically detects which virtual machine is used and then selects the matching library. If you want to select a specific library to load, use the -DAUDIT=<libraryType> property, where libraryType is one of the following: 11, 12 or jni.

For example, the following line starts the profiling of the SwingSet program and specifies OptimizeIt to use the Java 2 Universal library:

```
c:\> java -classic -Xrunoii -DAUDIT=jni intuitive.audit.Audit SwingSet
```

Java

Examples with JDK 1.1

Use the following command to launch the applet contained in the file example.html:

```
C:\> java -noclassgc -Djava.compiler=NONE intuitive.audit.Audit
sun.applet.AppletViewer example.html
```

Use the following command to pause the application com.busy.BusyApp immediately after launch:

```
C:\> java -noclassgc -Djava.compiler=NONE intuitive.audit.Audit -pause
com.busy.BusyApp
```

Java 2

Examples with JDK 1.2

Use the following command to launch the applet contained in the file example.html:

```
C:\> oldjava -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE intuitive.audit.Audit sun.applet.AppletViewer example.html
```

Use the following command to pause the application com.busy.BusyApp immediately after launch:

```
C:\> java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE intuitive.audit.Audit -pause com.busy.BusyApp
```

Example with JDK 1.3

Use the following command to start the com.busy.BusyApp application with the CPU profiler automatically started:

```
C:\> java -classic -Xrunoii intuitive.audit.Audit -startCPUprofiler com.busy.BusyApp
```

Connecting the audit system to the OptimizeIt user interface

After the test program is running, you need to direct the results generated by the audit system into the OptimizeIt user interface. This procedure is called “attaching” OptimizeIt to the test program.

To show the audit system results

1. In OptimizeIt, select New from the File menu.
2. In the Edit Settings panel select Remote Application in the Program Type section.
3. Type localhost in the Host Name field.
4. Enter the port number you want to use for the communication link between OptimizeIt and the audit system in the Port number field. Change this value only if you used the -port option when launching the test program.
5. (Optional) Add source code directories to the source path. This allows OptimizeIt to show the relevant source code. Specific files can be added later during the profiling session.
6. Click the Attach Now button.

3.8 Profiling a Java program running on a different machine

Profiling a Java program running on a different machine is similar to testing a Java program started from the command line. Follow the instructions above to start the test program on another machine. Start OptimizeIt, and create some new settings with Remote Application as Program type. In the Host name field, type the name of the machine running the test program.

3.9 Offline profiling

OptimizeIt allows you to start the profiling session from the command line and automatically generate snapshots at fixed intervals. Snapshots can be reloaded in OptimizeIt for later analysis. This feature can be used to profile an application server over a long period or in a production environment (for example). Offline profiling also limits overhead since you do not attach to the application.

Note: *You cannot attach OptimizeIt to an audit system configured for offline profiling.*

Starting offline profiling

In order to start online profiling, you must start your application from the command line with the option:

-offlineprofiling[:<offlineprofilingOptions>]

Syntax

Each offline profiling option is associated with a value. The option and the value must be separated with an equal character '='. The different options are separated by comas ','. No space should be used.

Note: *The value of the directory and comments options may contain spaces. in that case, make sure to include the values between quotes "".*

For the initial delay and delay values, the unit used is specified just after the value, with the following syntax:

Symbol used	Corresponding unit
h	hours
m	minutes
s	seconds
ms	milliseconds

If no unit is specified, seconds are assumed.

Examples:

- delay=1m sets the delay to 1 minute
- initialDelay=2h sets the initial delay to 2 hours
- delay=10 sets the delay to 10 seconds

For values that are boolean the syntax is: true, false.

Options

The following table describes the different options available:

Option	Description
initialDelay	The delay before the generation of the first snapshot. Must be positive or zero. If not specified, the value of the option delay is used
delay	The delay between the generation of 2 snapshots. Must be positive and not zero.
counter	The number of snapshots to be generated. Must be positive. If not specified, the number of snapshots generated is not limited.
directory	The directory where the snapshots are generated. Must be a valid directory. If not specified, snapshots are generated in the current directory.
filename	The name used for the snapshots. If no filename is specified, the filename is constructed from the main class name.
updateFile	Boolean that sets if the same snapshot should be reused. If set to yes, only one file is used for all the snapshots and the preceding snapshot is overwritten each time a new snapshot is generated.

Option	Description
appendTime	Boolean that sets if the date and time of the snapshot generation should be appended at the end of the filename. True by default. If set to false, a counter is used to differentiate the different snapshot files.
includeCPU	Boolean that sets if the CPU profiling information should be stored in the snapshot. If true and the CPU profiler is not automatically started, it starts the CPU profiler with the default options. See “Starting the CPU profiler from the command line” on page 31 for more information.
includeMemory	Boolean that sets if the memory profiler information (heap mode and backtrace mode) should be stored in the snapshot. True by default.
includeReferences	Boolean that sets if the memory profiler reference graph (and reference from roots with JDK 1.2 and later) should be stored in the snapshot. False by default. ***Note:*** *The amount of data to be stored for the reference mode is important. The snapshots generated with this option are larger files and take more time to be generated. Only select this option when you really need the reference mode information.*
comment	String used to add comments to the snapshot. Use quotes “ if the string contains spaces.

Examples

The following command line starts the offline profiling of the application AppServer with the JDK 1.2. A snapshot will be generated every minute. The filenames of the snapshot will be AppServer_snapshot_<dateAndTime>.snp. The snapshots will be generated in the directory c:\TEMP.

```
c:\>java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE
intuive.audit.Audit -offlineprofiling:delay=1m,directory=c:\TEMP
com.busy.AppServer
```

The following command line starts the profiling of the SwingSet application with the JDK 1.2. The first snapshot will be generated after 5 minutes,

and then a snapshot will be generated every hour to reach a total of 10 snapshots. The reference graph will be included. The snapshots will be generated in the current directory, and will be named *profiling<dateAndTime>.snp*.

```
c:\>java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE
intuive.audit.Audit -offlineprofiling:initialDelay=5m,delay=1h,counter=10,file-
name=profiling,includeReferences=true SwingSet
```

The following command line starts the profiling of the BusyApp application with the JDK 1.3. A snapshot will be regenerated every 5 minutes in the current directory, using the filename test.snp. The CPU information will be included. (See “Starting the CPU profiler from the command line” on page 31 for more information on the -startCPUprofiler command).

```
c:\>java -classic -Xrunoii intuive.audit.Audit -startCPUprofiler:type=sam-
pler,samplingPeriod=10 -generatesnapshot:delay=5m,filename=test,inclu-
deCPU=true,updateFile=true BusyApp
```

The following command line starts the profiling of the AppServer application with the JDK 1.1. Just one snapshot will be generated, and will be updated each hour (each new snapshot will overwrite the previous snapshot).

```
c:\>java -noclassgc -Djava.compiler=NONE intuitive.audit.Audit -generate-
snapshot:delay=1h,updateFile=true,comment=”Snapshot of the application
server in a stressfull environment.” AppServer
```

Starting the CPU profiler from the command line

By default, the OptimizeIt CPU profiler is only started when triggered from the OptimizeIt user interface. Use the following option to start the CPU profiler automatically before the tested program is started.

```
-startCPUprofiler[:<CPUoptions>]
```

Syntax

Each option is associated with a value, the different options are separated by comas ‘,’.

Options

The following table describes the different options available:

Option	Values	Description
type	[sampler, instrumentation]	Type of CPU profiler used. Default is sampler. ***Note:*** *Instrumentation is only available with JDK 1.2 or newer*
displayPrecision	[method,line]	Sets the precision of the sampler. This option is only active with type=sampler. Default is method.
samplingPeriod	[1..1000]	Sets the sampling period (in milliseconds) of the sampler. This option is only active with type=sampler. Default is 50 ms.
onlyCPU	[true,false]	If true only the pure CPU usage is taken into account. Default is false.
precision	[micro,milli]	Controls wether the profiler has microsecond or millisecond precision. This option is only active with type=instrumentation. Default is milli.
filterEnabled	[true,false]	If true, methods executed in less than the delay specified by the filterDelay property will be excluded. This option is only active with type=instrumentation. Default is false.
filterDelay	[0..1000]	Sets the delay in milliseconds of the filter used to exclude short methods. This option is only active with type=instrumentation and filterEnabled=true. Default is 100.

Note: *These options are similar to the options of the CPU profiler accessible from OptimizeIt, see "Using the CPU profiler" in chapter 5 for more information on these options.*

Examples

The following command line starts the application BusyApp with JDK 1.2 with the sampler. The sampling rate is 5 ms, the display precision is line and only the real CPU time is recorded.

```
c:\> java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE intuitive.audit.Audit -startCPUprofiler:type=sampler,samplingPeriod=5,onlyCPU=true BusyApp
```

The following command launches the applet contained in the file example.html with JDK 1.2, and starts the instrumentation. The precision is milliseconds, and methods that consumes less than 50 ms are excluded.

```
C:\> java -classic -Xrunoii -Xnoclassgc -Djava.compiler=NONE intuitive.audit.Audit -startCPUprofiler:type=instrumentation,precision=milli,filterEnabled=true,filterDelay=50 sun.applet.AppletViewer example.html
```

Using filters from the command line

OptimizeIt provides filters to filter the profiling information, See "What are filters" on page 35 for more information on filters.

In order to use filters for offline profiling, OptimizeIt provides an option to specify filters from the command line. The syntax of this option is:

```
-filter[:<filterOptions>]
```

Syntax

Each option is associated with a value, the different options are separated by comas ',’.

The value of the option patternList is a list of patterns (one or more) separated by comas ',’. The list is in parenthesis '(' and ')'.

A pattern is a string that describe a package, a class or a method. The patterns can use the well know special characters * and !. For example, java.lang.* means all the packages and classes included under java.lang, and

!java.swing.* means everything that is not included in the java.swing package,

Options

The following table describes the different options available:

Option	Values	Description
CPU	[true,false]	Controls wether or not the filter is applied to CPU profiler. Default is true.
memory	[true,false]	Controls wether or not the filter is applied to memory profiler. Default is true.
patternList	(pattern1,pattern2,...)	The list of patterns describing the filter.
operation	[and,or]	The operation applied to the different patterns. Default is or.

Examples

The following command line starts the application BusyApp with JDK 1.3 and filter for memory profiler only the resources used by the String class and the swing package.

```
c:\> java -classic -Xrunoii intuitive.audit.Audit -filter:CPU=false,memory=true,operation=or,patternList=(java.lang.String,javax.swing.*) BusyApp
```

3.10 Profiling with filters

OptimizeIt provides filters that can be used to automatically remove information about resources used by particular packages, classes or methods when profiling.

What are filters

Filters are sets of patterns that define packages, classes or methods that should be ignored by OptimizeIt when profiling an application, an applet or a remote application. Filters can be applied to the CPU profiler, memory profiler or both.

When a filter is applied to the CPU profiler, OptimizeIt ignores CPU or time usage consumed by Java code matching the filter pattern. When a filter is applied to memory profiler, OptimizeIt ignores object allocations performed by Java code matching the filter pattern.

OptimizeIt provides several filters ready to use, including filters dedicated to many application servers. You can also create your own filters, see “Creating filters” in chapter 7.6.

Filters are very convenient for profiling servlets, EJBs and JSPs. In that context, they can be used to remove resources used by the application server, allowing you to focus on your Java code.

Note: *Filters only remove information about resources used by code matching the filter pattern. Filters don't remove information about methods invoked from filtered methods. For example, if a filtered method calls an unfiltered method that allocates an instance, this instance won't be filtered. Identically, if a filtered method invokes a method that consumes CPU time, this CPU time won't be filtered.*

Using filters

1. Select New from the File menu.
2. Fill the Startup section as described in the previous sections.
3. Switch to the Filters section by clicking the Filters tab.
4. If you want to see the patterns associated with a filter, double click on the filter in the table.
5. To enable a filter with the memory profiler, click the Ignore memory usage column.
6. To enable a filter for CPU profiler, click the Ignore CPU usage column.

7. If you want to disable all filters, click the Disable all filters option.
8. When you are done with your filter selection, click the Start now button (Attach now if you profile a remote application).

Note: *Filters are specified at launch time. You can't change the filters for a profiling already started.*

3.11 Starting OptimizeIt from the test program

In some environments, the Java virtual machine is started automatically. The OptimizeIt audit system can be invoked from the test program using the OptimizeIt application programming interface (API). Invoking OptimizeIt from your application allows you to invoke profiling at specific places in your source code.

To invoke OptimizeIt from inside your test program

1. Set up the OptimizeIt audit system
2. Add OptimizeIt API calls in your program
3. Start your test program
4. Run OptimizeIt
5. Connect the audit system to the OptimizeIt user interface

Setting up the OptimizeIt audit system

To use the OptimizeIt audit system, you need to modify your CLASSPATH and PATH environment variables to include the following directories:

CLASSPATH: <InstallDir>\OptimizeIt\lib\optit.jar

PATH: <InstallDir>\OptimizeIt\lib

Adding OptimizeIt API calls in your program

The following example shows how to start the OptimizeIt audit system from Java code. For more information about OptimizeIt audit API, read the API reference documentation.

```
import intuitive.audit.Audit;
void startAuditSystem() {
/** Start the OptimizeIt audit system with some default options **/
Audit.start(1470, Audit.DEFAULT_OPTIONS);
}

/** Note: after starting the audit system. you need to start the profilers by using
Audit.enableProfiler(). If you want the profilers to be enabled all the time,
```

```
change the option in Audit.start() from Audit.DEFAULT_OPTIONS to
Audit.PROFILERS_ALWAYS_ENABLED **/
```

Starting your test program

Although the test program starts the audit system, some extra parameters need to be added to the code or the shell script that starts the virtual machine.

Java

With JDK 1.1, it is necessary to disable class garbage collection by using the -noclassgc option. It is also necessary to disable the JIT.

Example: `java -noclassgc -Djava.compiler=NONE TestProgram`

Java 2

With JDK 1.2, it is necessary to use the following options:

- disable class garbage collection
- start the oii profiling interface
- disable the JIT
- disable Hotspot

Example: `java -classic -Xnoclassgc -Xrunoii -Djava.compiler=NONE TestProgram`

With JDK 1.3, it is necessary to use the following options:

- start the oii profiling interface
- select the classic runtime

Example: `java -classic -Xrunoii TestProgram`

4 Using the memory profiler

The memory profiler provides information about the objects allocated by your Java program. It displays all allocated instances in real time, and allows you to see precisely which method is responsible for object allocations.

In addition, the memory profiler provides a powerful way to browse incoming and outgoing object references.

Using the OptimizeIt memory profiler, you can improve the performance of your program by:

- Minimizing temporary object allocations. Excessive temporary object allocations can cause the garbage collector to run every few seconds. When running, the garbage collector slows down your Java program.
- Minimizing the number of instances required for a given operation, and therefore decreasing the memory that your program requires.
- Reducing memory waste by making sure every object is garbage collected.

4.1 Memory profiler modes

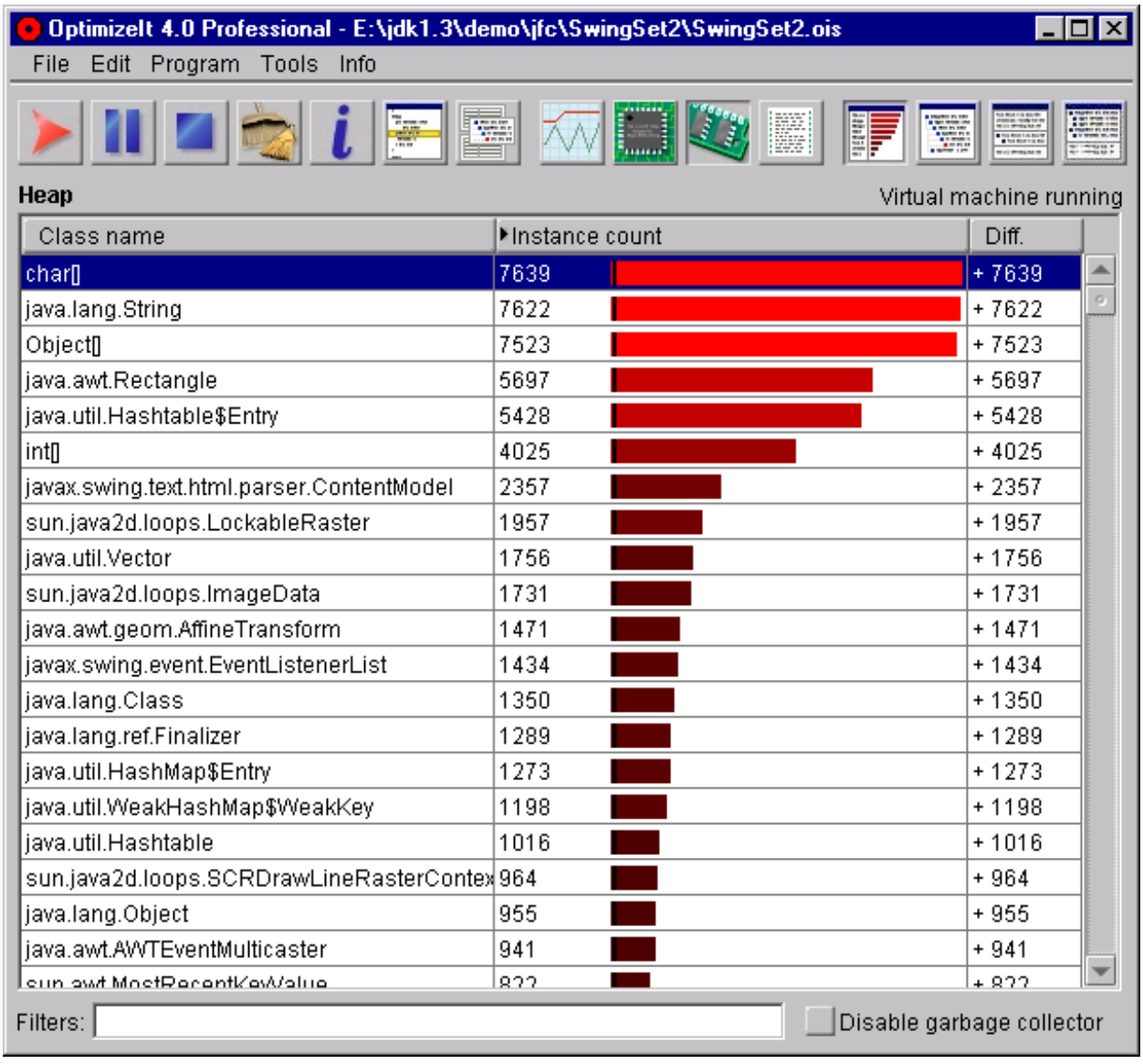
The memory profiler provides many views into the profile information. Switch between these views, called "modes," using the buttons in the tool-bar:

	**Heap** mode	Shows all classes and the number of instances allocated for each.
	**Allocation Backtrace** mode	Shows the methods involved in object allocation for a class selected in Heap mode.
	**Instance Display** mode	Shows the incoming or outgoing references of a given instance.
	**Reference from roots** mode	Shows outgoing references starting from the roots.

Java 2

4.2 Understanding object allocations

By default when your program is started, OptimizeIt appears in the Heap mode, displaying all classes and the number of instances currently allocated. The following picture shows the Heap mode:



To sort the values in a column of the table, click the column header.

The filter on the bottom of the window allows you to select the classes that you want to see. For example, to only see all the AWT classes, type:

```
java.awt.*
```

and press Return.

Both the asterisk wildcard character (*) and the not (!) are supported. It is possible to enumerate more than one pattern using a comma (,) separator.

For example, to see all classes matching "image" except java.awt.Image classes type:

```
*image*, !java.awt.Image*
```

To see all classes, clear the filter completely, or type an asterisk (*).

Click the "Disable garbage collector" option to study object allocations without having the garbage collector removing instances. OptimizeIt "Disable garbage collector" option disables the garbage collector logically for the heap mode.

The garbage collector is not really disabled, however the heap mode shows you what would happen if the garbage collector was not running.

You can also click the Run Garbage Collector button to explicitly run the garbage collector.

Click the Mark button to mark the current instance count. A mark appears on each graph of the instance. This mark represents the number of instances allocated at the time you set the mark. The time difference is then set to zero for all classes. The mark allows you to see the instances allocated by a specific action in your program.

For example, click the Mark button, and then open one of the test program dialog boxes. Click the Diff column header to sort the column so you can see the instances allocated when the dialog box was created. The Allocation Backtrace and Instance Display modes also display information relative to a mark.

Refining the Heap mode display

The Inspector window provides options to refine the Heap mode display. To open the Inspector window, click the Inspector button.

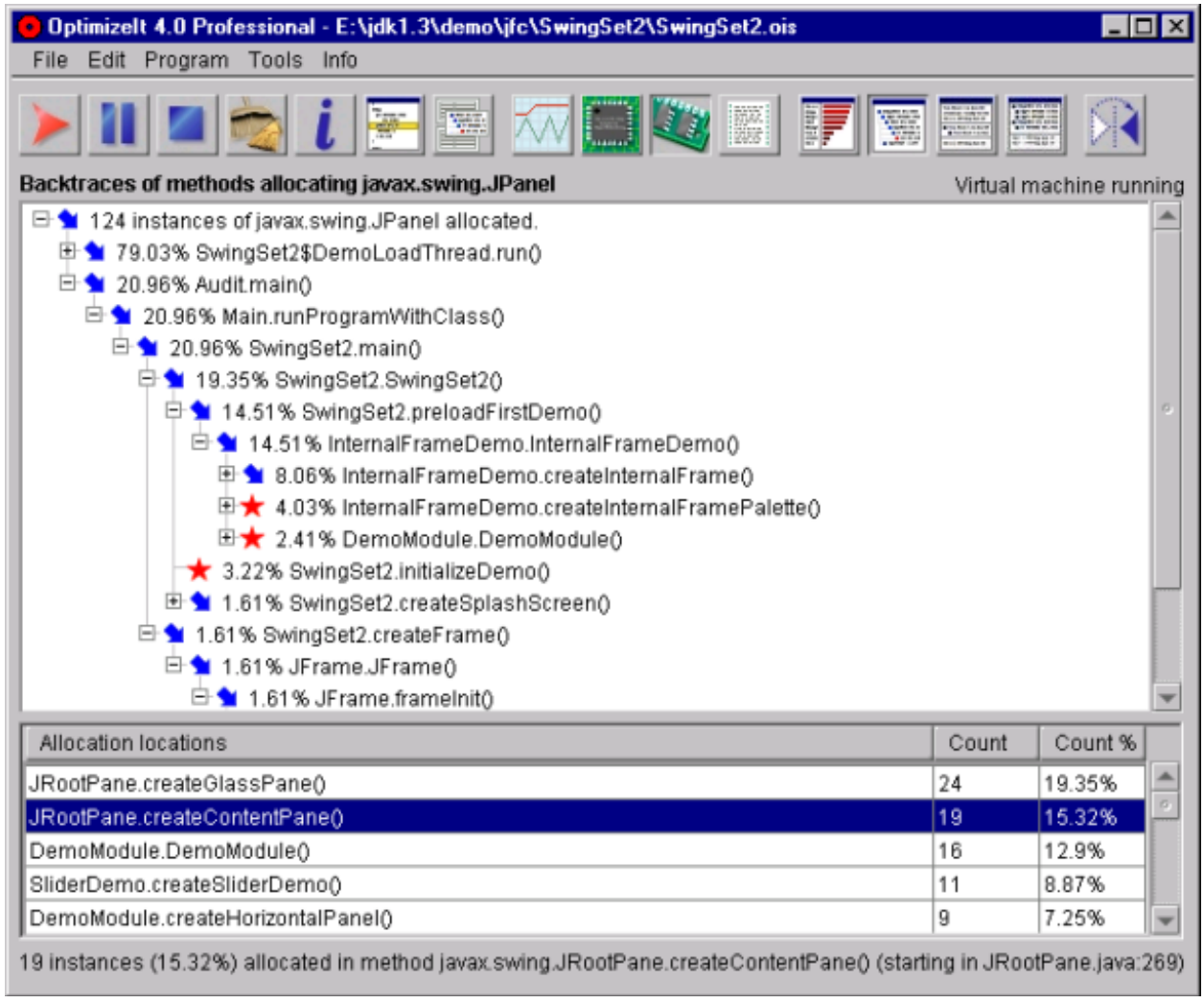
The Inspector window contains the following options for Heap mode:

Option	Description
Always sorted	Sorts the display in real time. Use this option to keep track of specific object allocations while the test program is running.
Classes without instances	Displays all classes in the class list, including classes that currently have no allocated instances.
Show sizes	Adds total memory (Size) and changed memory (Size Difference) columns to the table. The Size column displays the memory required by all instances of each class. The Size Difference column shows the memory size difference since the last time a mark was set. ***Note:*** *The displayed memory size is the "shallow" size of the object. This size only includes size of the memory required for the instance and does not include the size of objects retained by the instance.*
Show freed instances	Adds freed object count and freed object count difference columns in the main display. Use this option to understand performance issues related to the allocation of too many temporary objects.
Relative difference sorting	By default differences are sorted by absolute value. Use the option to sort by relative differences.

4.3 Understanding where objects are allocated

After you have identified a class with an excessive number of instances, the next step is to identify the code or the part of the program that is responsible for these allocations. In Heap mode, select the line displaying the class you want to focus on, and click the Show Allocation Backtraces button. OptimizeIt switches to Allocation Backtrace mode. Allocation Backtrace mode shows which code is responsible for the selected class object allocations.

The following picture shows allocation backtrace mode:



The top section in Allocation Backtrace mode traces calls from the first method of the Java program to where allocations occur. The purpose of this view is to understand which feature of your program is responsible for object allocations. By opening nodes in this view, you can see precisely where allocations originate. Any line with an allocation icon () is a line that is responsible for one or more object allocations.

The bottom section displays the names of methods responsible for object allocations. The purpose of this view is to quickly understand if a single method performs excessive allocations.

By pressing the Reverse Display button in the toolbar, you can reverse the lists to display backtraces from the place where the allocations take place to the Main method of the Java program. This view can be useful when you need to focus on methods or lines of code responsible for object allocations rather than broad features of your program.

To display the code corresponding to a line in the top or bottom sections of the window, select the line, then click the Show Source Code button. You can also double click the line to show the source code. For more information, see “Viewing source code” in chapter 7.5.

Refining the Allocation Backtrace mode display

The Inspector window provides options to refine the Heap mode display. To open the Inspector window, click the Inspector button.

The Inspector window contains the following options for Allocation Backtrace mode:

Option	Description
Show allocations since last mark	Displays only the backtrace for the methods responsible for allocating instances since the last time you pressed the Mark button in Heap mode.
Display precision	By default, graphs show profiles by method; changing the granularity allows you to organize the graph by line of code.

4.4 Tracking temporary object allocations

Excessive temporary object allocations are often a source of performance problems in Java programs. Although allocating an object is a fast operation on most Java virtual machines, excessive temporary objects keep the garbage collector busy. Running the garbage collector can block the Java program for as much as a couple hundred milliseconds. If the garbage collector has to run very often, these interruptions cause the Java program to appear slow to the user.

Temporary objects are hard to track without OptimizeIt because some Java APIs allocate many temporary objects.

Tracking down excessive temporary object allocations

1. Click the Show Heap button.

2. Click the Instance Count column header to sort the display.
3. Exercise your program, noting the classes that show quick changes in their number of instances.
4. Select one of these classes.
5. Click the Run Garbage Collector button to free current temporary objects.

6. Click the Mark button to place a mark on the currently allocated instance.

7. Exercise your program again to recreate the problem.This time the garbage collector is disabled and the number of instances does not decrease.
8. Click the Show Allocation Backtraces button.

9. Click the Show Inspector button.

10. Select the option "Since last mark" to only display newly allocated objects.

The Allocation Backtrace mode displays the code responsible for the allocations.

After you have identified which line of code or API is responsible for all these objects, change your program so it uses different APIs or reuses the same objects.

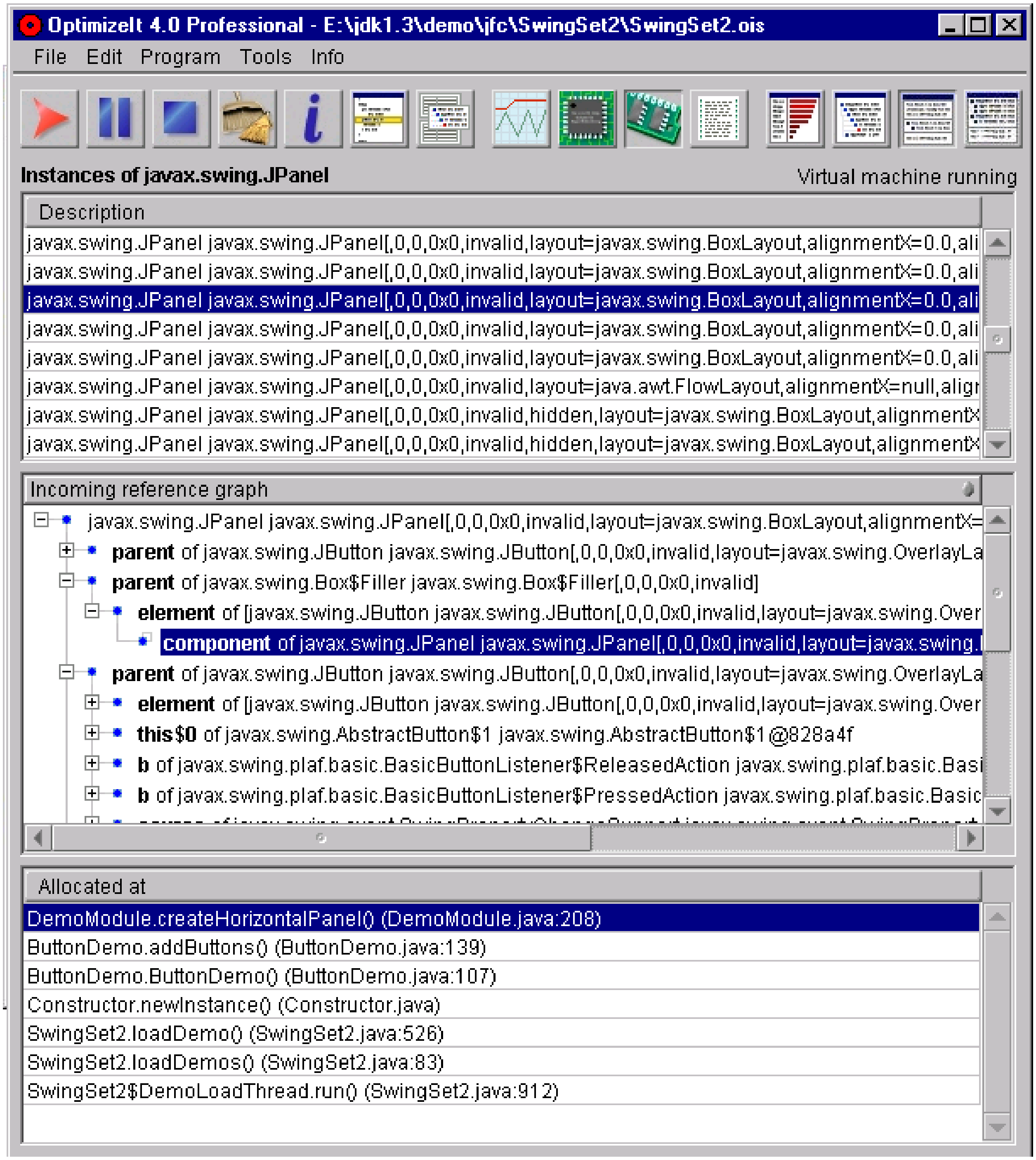
4.5 Identifying objects not freed by the garbage collector

In a development environment that has no garbage collector, a program that does not free the allocated memory loses this memory, creating a "memory leak." With Java's garbage collector, it is no longer necessary for programmers to keep track of allocated objects and free them explicitly when they are no longer required.

However, it is quite common for a Java program to keep some references to some objects that are not really necessary anymore. For example, take a Java program that displays a splash screen at startup. The splash screen image can be quite heavy and is necessary only during startup. If a static variable somewhere references an object that has a hashtable that references the splash screen image, the image will never be garbage collected because it is still accessible from the Java program. Thus, the program requires more resources than necessary to maintain the splash screen image. This situation is similar to a memory leak in non garbage collected environments.

To solve this kind of problems, OptimizeIt Professional provides an Instance Display mode. This mode displays all instances of a given class and their incoming and outgoing references. Incoming references are references from an object to the selected object. Outgoing references are references from the selected object to other objects.

To switch to Instance Display mode, press the Show Instance button. The following picture shows the Instance Display mode displaying some incoming references:



The top view displays the string representations of the selected class instances. Instances are sorted by allocation date, with the most recently allocated instances on top. The string representations are obtained by call-

ing the method toString() on each object. By implementing useful toString() methods in your classes you can use this view to identify the currently allocated instances.

The middle view displays the objects that are referenced by the object selected in the top view. When available, the instance variable that references the object is in bold.

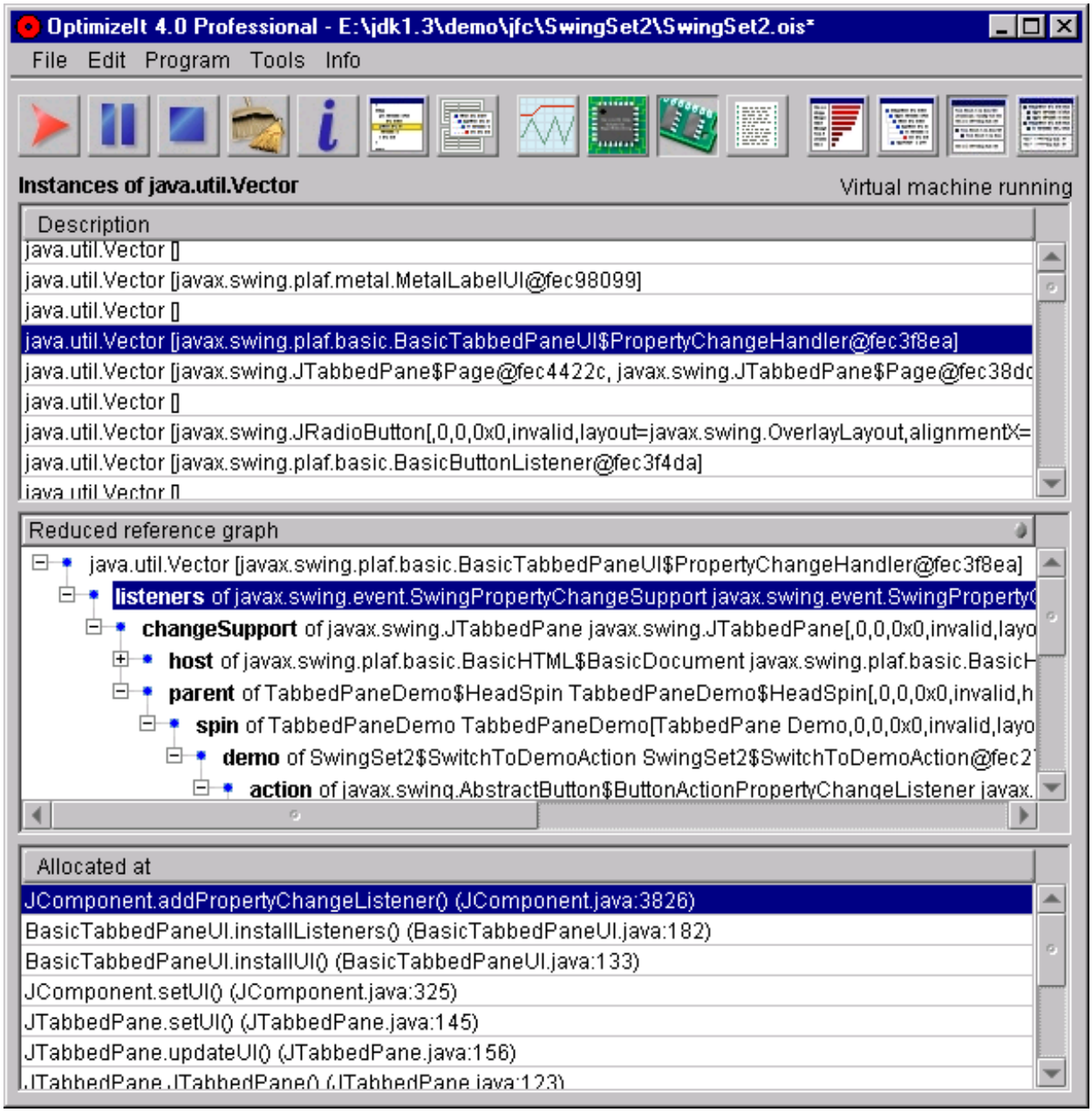
If a cycle is found in the graph, the point where the cycle occurs is displayed with the  icon.

Java 2

Reduced reference graph

With Java 2, OptimizeIt provides a reduced reference graph. A reduced reference graph is the transitive closure of the full reference graph. If an object A is referenced by B and D, and if D also references B, the reference D->A won't be displayed. This mode is extremely interesting to understand which reference should be removed in order to allow the selected object to be garbage collected. All displayed references are references preventing the object from being garbage collected.

The following picture shows the Instance Display mode displaying the reduce reference graph:

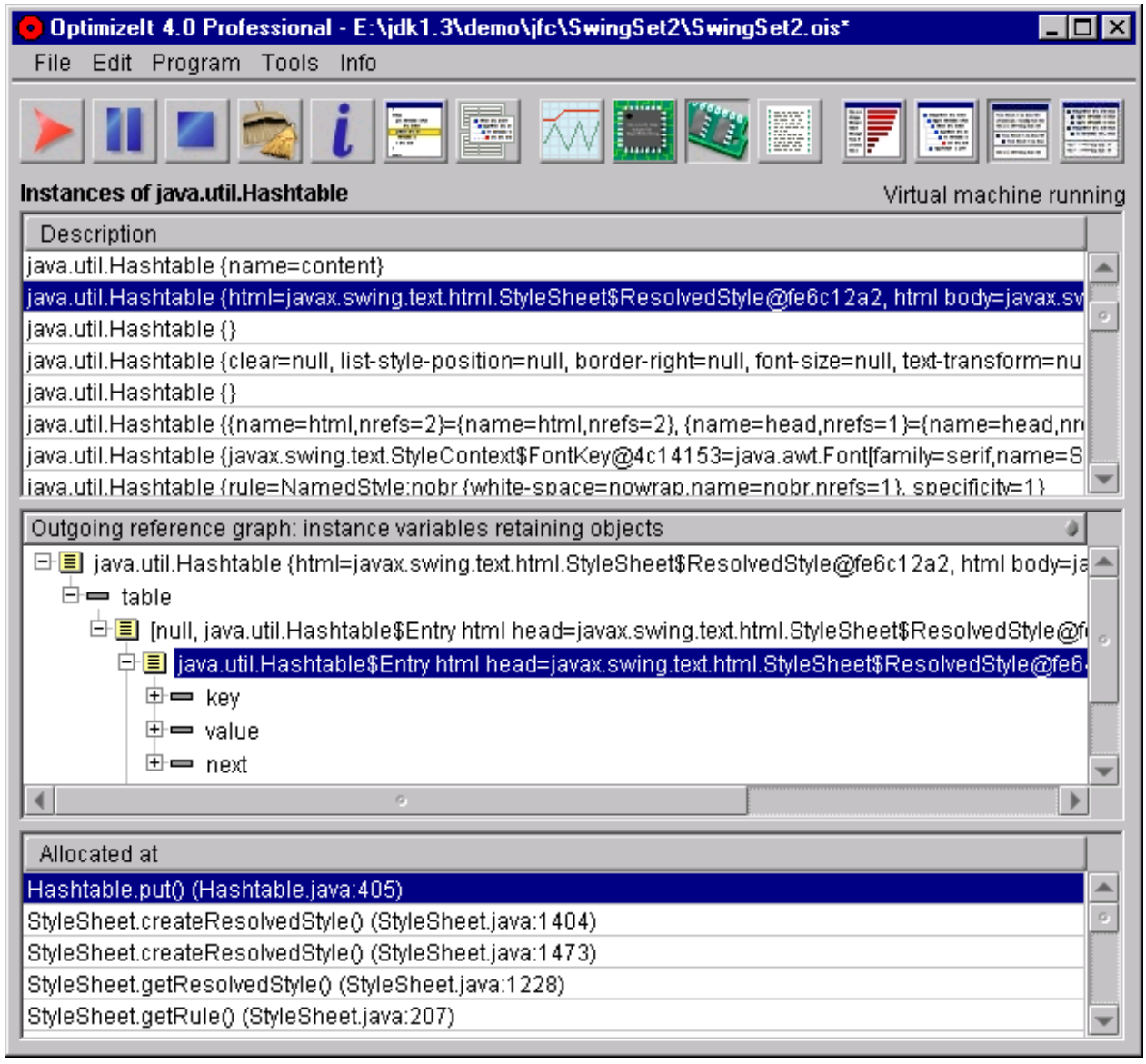


In the example above, the vector selected in the top view cannot be garbage collected because it is referenced by the member variable “listeners” of an event object which itself is referenced by the “changeSupport” member variable of a JTabbedPane. Clearing any of these reference will allow the Vector to be garbage collected.

The standard incoming reference graph is also provided. Click on the reference column header and select Reference graph.

Outgoing references

OptimizeIt can also display out-going references. To display outgoing references, click the Reference Graph column header, and then select Instance variables retaining objects. The following picture shows the Instance Display mode with outgoing references:



The middle view displays objects that are referenced by the object selected in the top view. Icons indicate the meaning of each line:

 Instance variables

 Instances

To display the code corresponding to a line in the middle or bottom sections of the window, select the line, then click the Show Source Code button. You can also double click the line to show the source code. For more information, see “Viewing source code” in chapter 7.5.

Refining Instance Display mode display

The Inspector window provides options to refine the Instance Display mode display. To open the Inspector window, click the Inspector button.

The Inspector window contains the following options for the Instance Display mode:

Option	Description
Show allocations since last mark	Displays only the instances that have been allocated since the last time you pressed the Mark button in Heap mode.
Reference graph type	Allows the type of the graph displayed to be selected. ***Note:*** *Selecting graph from here is similar to selecting by clicking the graph type column header on the Instance Display panel.*

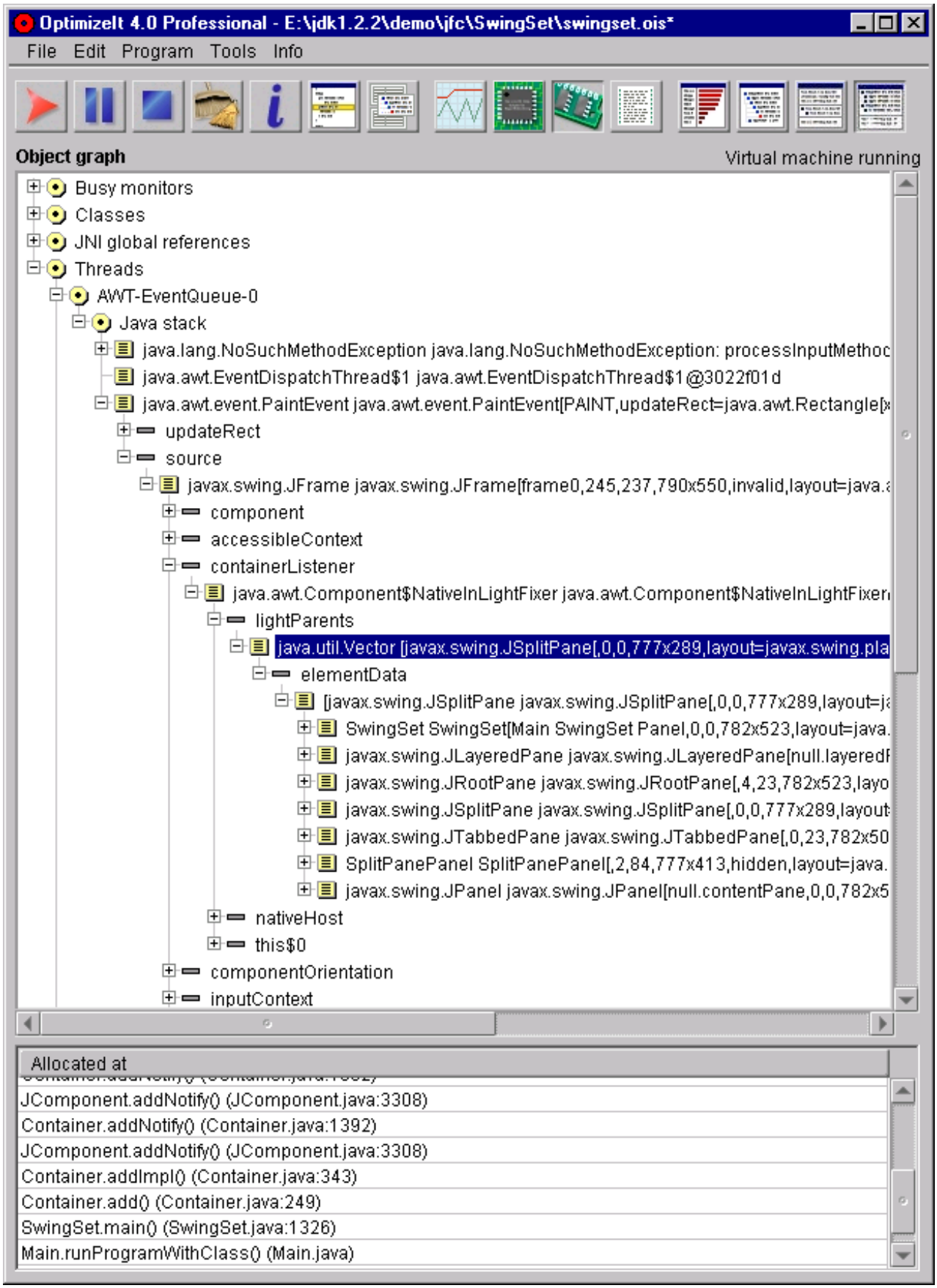
Java 2

Browsing references from roots

With Java 2, OptimizeIt allows you to browse references from roots. Roots are the roots of the reference graph and include:

- Busy monitors
- Class static variables
- Class constants
- JNI global and local references
- Threads Java and native stacks

With this mode, you can see the entire content of the heap and understand exactly the hierarchy between the different objects. The following picture shows the Browsing references from roots mode:


OptimizeIt 4.0 Professional - E:\jdk1.2.2\demo\jfc\SwingSet\swingset.ois*
File Edit Program Tools Info
Object graph
Virtual machine running
Busy monitors
Classes
JNI global references
Threads
AWT-EventQueue-0
Java stack
java.lang.NoSuchMethodException java.lang.NoSuchMethodException: processInputMethoc
java.awt.EventDispatchThread$1 java.awt.EventDispatchThread$1@3022f01d
java.awt.event.PaintEvent java.awt.event.PaintEvent[PAINT,updateRect=java.awt.Rectangle[
updateRect
source
javax.swing.JFrame javax.swing.JFrame[frame0,245,237,790x550,invalid,layout=java.
component
accessibleContext
containerListener
java.awt.Component$NativeInLightFixer java.awt.Component$NativeInLightFixer
lightParents
java.util.Vector [javax.swing.JSplitPane[,0,0,777x289,layout=javax.swing.pla
elementData
[javax.swing.JSplitPane javax.swing.JSplitPane[,0,0,777x289,layout=j
SwingSet SwingSet[Main SwingSet Panel,0,0,782x523,layout=java.
javax.swing.JLayeredPane javax.swing.JLayeredPane[null.layeredP
javax.swing.JRootPane javax.swing.JRootPane[,4,23,782x523,layo
javax.swing.JSplitPane javax.swing.JSplitPane[,0,0,777x289,layout
javax.swing.JTabbedPane javax.swing.JTabbedPane[,0,23,782x50
SplitPanePanel SplitPanePanel[,2,84,777x413,hidden,layout=java.
javax.swing.JPanel javax.swing.JPanel[null.contentPane,0,0,782x5
nativeHost
this$0
componentOrientation
inputContext
Allocated at
JComponent.addNotify() (JComponent.java:3308)
Container.addNotify() (Container.java:1392)
JComponent.addNotify() (JComponent.java:3308)
Container.addImpl() (Container.java:343)
Container.add() (Container.java:249)
SwingSet.main() (SwingSet.java:1326)
Main.runProgramWithClass() (Main.java)

In this example, the stack of the thread AWT-EventQueue-0 references an instance of PaintEvent, which itself references the frame of the application, which references through its containerListener various graphical objects of the application.

The icons used in the graph have the following meaning:

 Root

 Instance variables

 Instances

If the selected row in the top view is an object, the bottom view displays where the object was allocated. As usual, double clicking on the row or clicking on the  button shows the source code corresponding to the selection.

5 Using the CPU profiler

The purpose of the CPU profiler is to understand in which methods your program spends its time in. Using the CPU profiler involves the following steps:

- Launch a Java program (or attach OptimizeIt to it)
- Start the CPU profiler
- Use the Java program to recreate a situation where the program is slow
- Stop the CPU profiler

OptimizeIt then gives you a per-thread description of the time spent in each method or CPU used during the test session.

The OptimizeIt CPU profiler helps you understand what to change in your program to improve its performance.

5.1 Recording a test session

The following procedure describes how to create a test session with the OptimizeIt CPU profiler:

To record a test session

1. Click the Start Java Program button.

2. Click the Show CPU Profiler button.

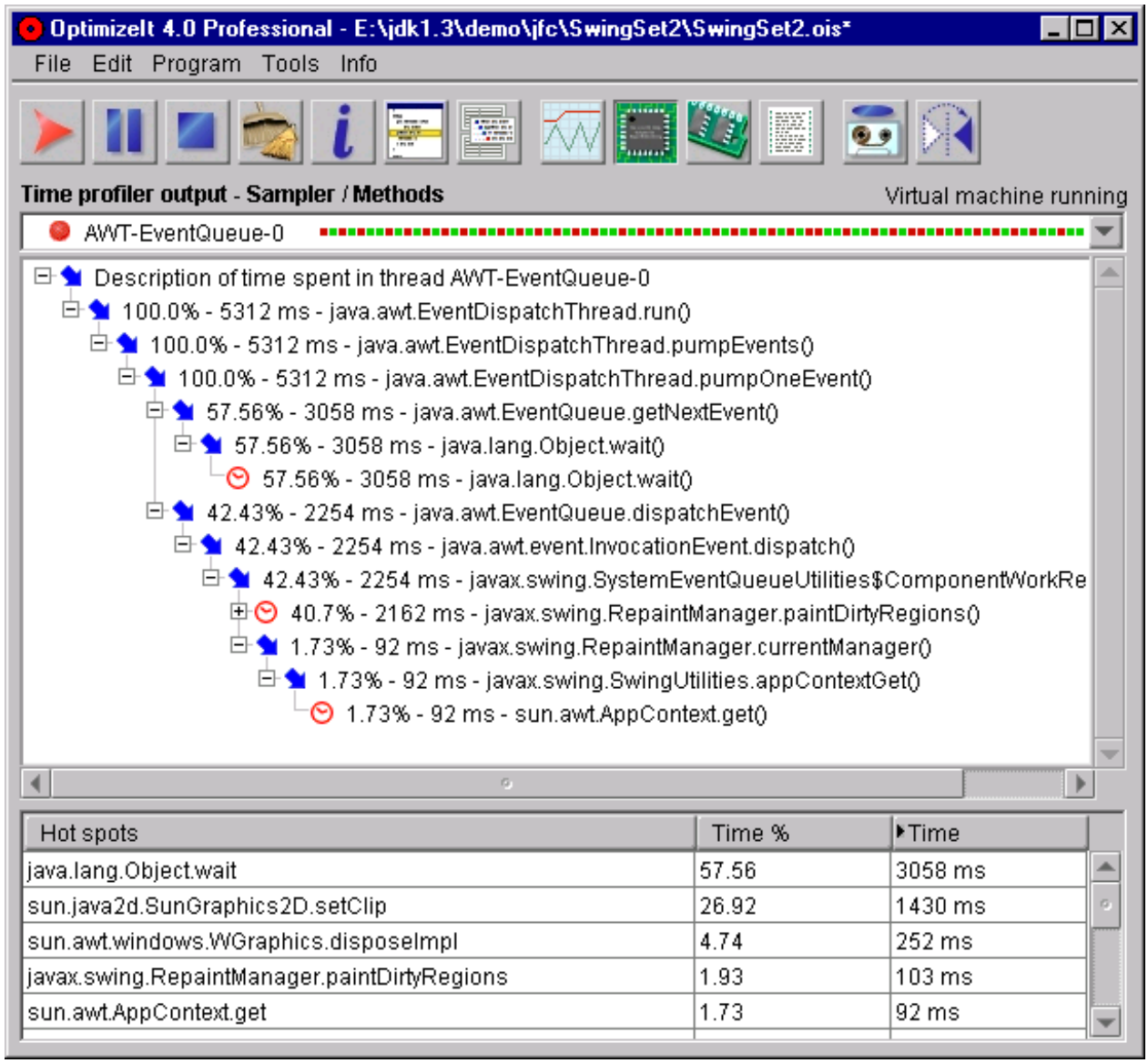
3. Click the Start/Stop CPU Profiler button.

4. Exercise the Java program to recreate a performance problem.
5. When the test program completes the action, click the Stop button.

The OptimizeIt window shows the CPU usage for each thread during the test session.

5.2 Understanding the profiler output

After you have recorded a test session, OptimizeIt displays information describing the time spent in each method, or the CPU usage in the Java program. The following screen shot shows the OptimizeIt CPU profiler output:



The top section displays how the time was spent or how CPU was used during the test session. In the example above, 40.7% of the time was spent painting dirty regions (RepaintManager.paintDirtyRegions()), while1.73% of the time was spent retrieving the “RepaintManager:.

Icons indicate the meaning of each line:

 The method immediately calls another method

 The method actually consumes time or CPU

 By pressing the Reverse Display button, you can reverse the top view to look at the backtrace tree from the leaves to the root. This view can be useful when you need to focus on methods or lines of code rather than broad features in your test program.

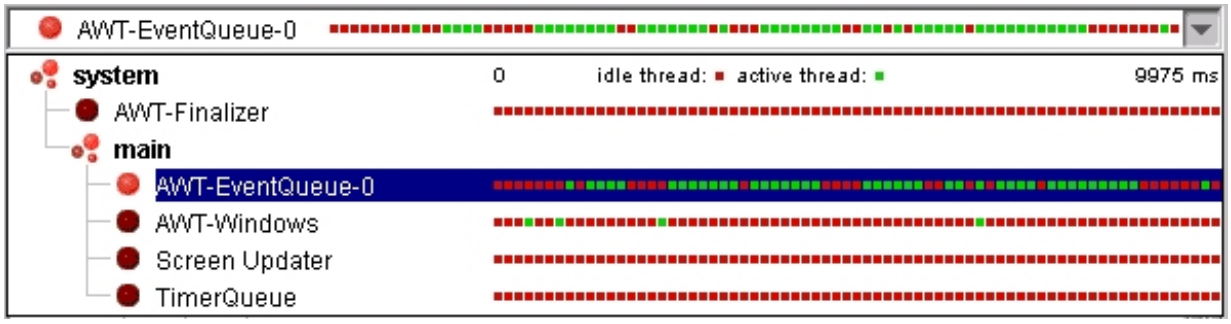
The bottom section displays the methods that were used during the test session, sorted by "hot spots." These are methods where the most time was spent. The time shown is the time the program spent in a method, no matter where the method was called from. The purpose of this view is to understand if a single method acts as a bottleneck and can be optimized to speed up all the tested features.

To display the code corresponding to a line in the display, select the line, then click the Show Source Code button. You can also double click the line to show the source code. For more information, see “Viewing source code” on chapter 7.5.

The graph on the top represents the sampling period of the selected thread. The colors in the graph indicate the state of the thread each time the sample occurred:

- Green dots mean that the thread was using the CPU
- Red dots mean that the thread was waiting on a condition
- Gray dots mean that the thread did not exist at the sampling time.

To show all threads and thread groups, click the graph (or the drop-down box arrow on the right side of the graph). The following screen shot shows the graph:



In this graph you can select a thread or thread group. Selecting a thread group shows how the time was spent for all thread and thread groups belonging to the thread group.

Refining the CPU Profile display

The Inspector window provides options to refine the CPU profile results. To open the Inspector window, click the Inspector button.

The Inspector window contains the following options for the CPU profiler:

Option	Description
Display precision	Controls the granularity of the profiler output. By default the data is organized by method; changing the granularity allows you to organize the data by line of code.
Display CPU usage only	Displays only the pure CPU usage and excludes any methods where the profiled thread was waiting for a condition.
Sampling period	Controls the granularity of the profiler output. Use a small value for a short test session and larger value for a long test session. Usually this value varies between 1 and 100 ms.

5.3 Advanced CPU profiler options

With Java 2 OptimizeIt provides two kinds of CPU profiler:

Sampler A sampler is a profiler that interrupts all running threads every p period. Once all threads are interrupted, it records what each thread is currently doing and whether each thread is currently using CPU. It then resumes all running threads. p is called a sampling period.

Instrumentation An instrumentation is a profiler that intercepts method invocations. Each time a method is called the profiler records the fact that a method was called and gives the control back to the application. The profiler also intercepts when a method returns from executing and records the amount of time/CPU that was spent in the method.

Both profilers have different domain of application. The following table shows pro and cons for both profiler types:

Profiler type	Advantage	Inconvenient
Sampler	• Very low overhead: the tested application runs 10% slower with the profiler running. • Low memory overhead and excellent scalability. • Since the profiler pauses all threads before recording any information, a sampler does not distord performance related data. • Since a sampler is not based on method invocations it can detect performance bottlenecks within methods.	• Lack of precision. A sampler precision is not greater than its sampling period. • Cannot record number of method invocations.
Instrumentation	• Very good precision: each time a method is invoked, it is recorded. • Possibility to measure precision in microseconds. • Possibility to record the number of times a method gets invoked.	• Lack of scalability: an instrumentation needs to record a lot of information. • Information distortion: the instrumentation profiler is actually running in the tested program threads. All method invocations are slower. Even if the profiler compensates, this can lead to distorted results. • Large overhead: the tested application runs several times slower with an instrumentation profiler.

The sampler is very good at profiling a large amount of code for a long time. The instrumentation is very good at precisely profiling small amount of code. The instrumentation is also very useful to understand if a method is slow or if it is called too often.

The following table shows when to use each profiler:

Profiler type	Application
Sampler	• Profiling an application for a very long time. E.g.: a server overnight. • Profiling a feature that requires a lot of different code. E.g.: the startup of a large GUI based application
Instrumentation	• Profiling anything that executes in less than a few hundred milliseconds. E.g.: a menu action • Profiling a system that has many threads executing many small requests. E.g.: a servlet

The Inspector window contains the following new options for the instrumentation profiler:

Option	Description
Precision	Controls whether the profiler has microsecond precision or millisecond precision. The microsecond precision has more overhead.
Filters	To minimize instrumentation overhead, a filter can be used to exclude any methods that need less than n ms to execute.

Note: *In order to minimize overhead, inspector options for the CPU profiler in Java 2 are no longer real-time. When an option is changed, you have to start a new recording session for the change to take effect.*

6 Virtual machine information

OptimizeIt can display in real-time high-level performance related data about the program being tested. Using this feature, you can understand if a performance problem is related to CPU, memory or both.

In this mode, OptimizeIt displays three graphs:

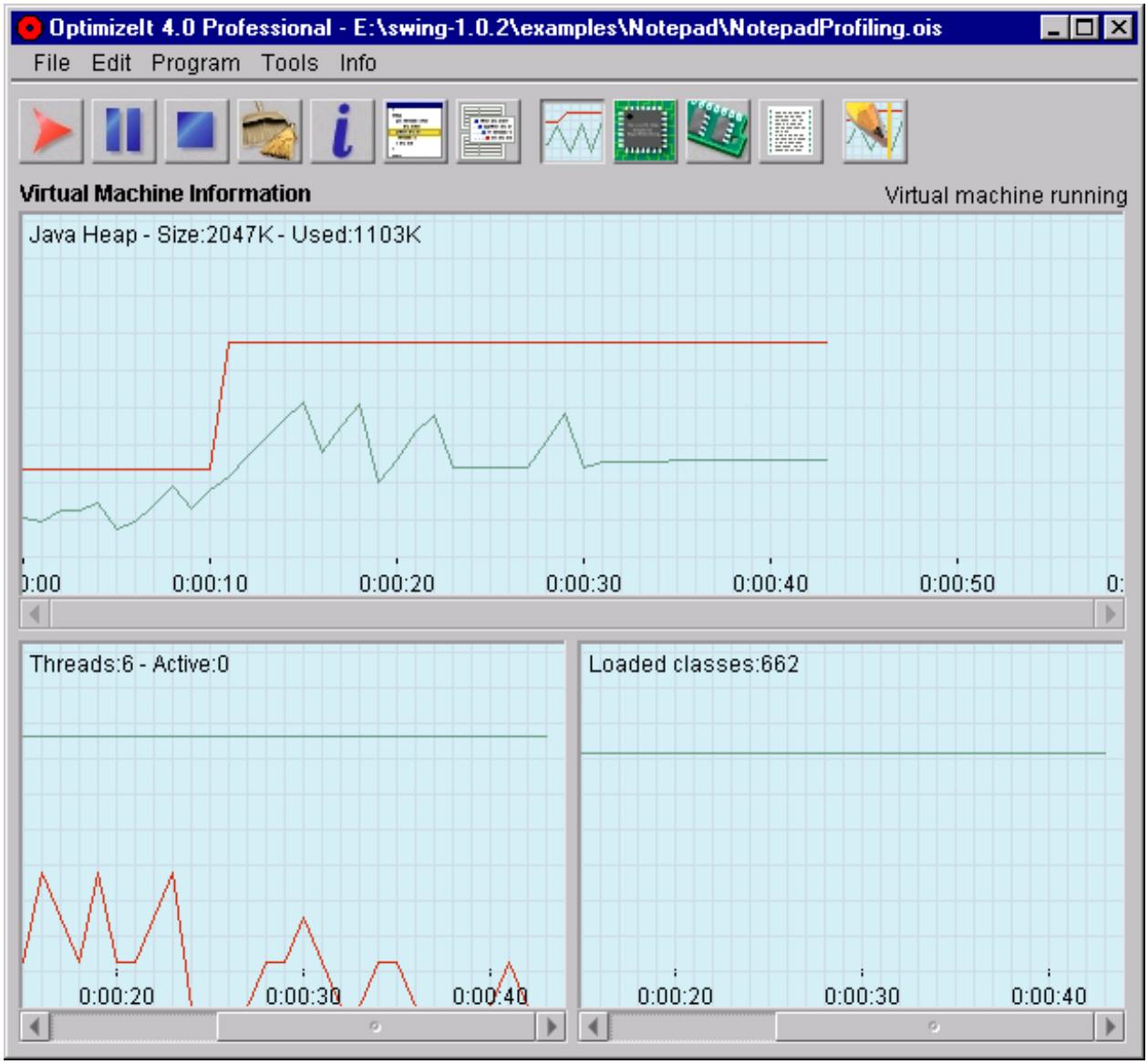
Java 2

Heap graph	This graph shows in red the current heap size required by the tested application and in green the current heap size that is actually used by the application.
Thread graph	This graph shows in red the current number of threads running and in green the number of threads actually using some CPU.
Class graph	This graph shows the number of classes currently loaded in the virtual machine.
GC graph	This graph shows the garbage collector activity, which is the time spent garbage collecting divided by the total time.

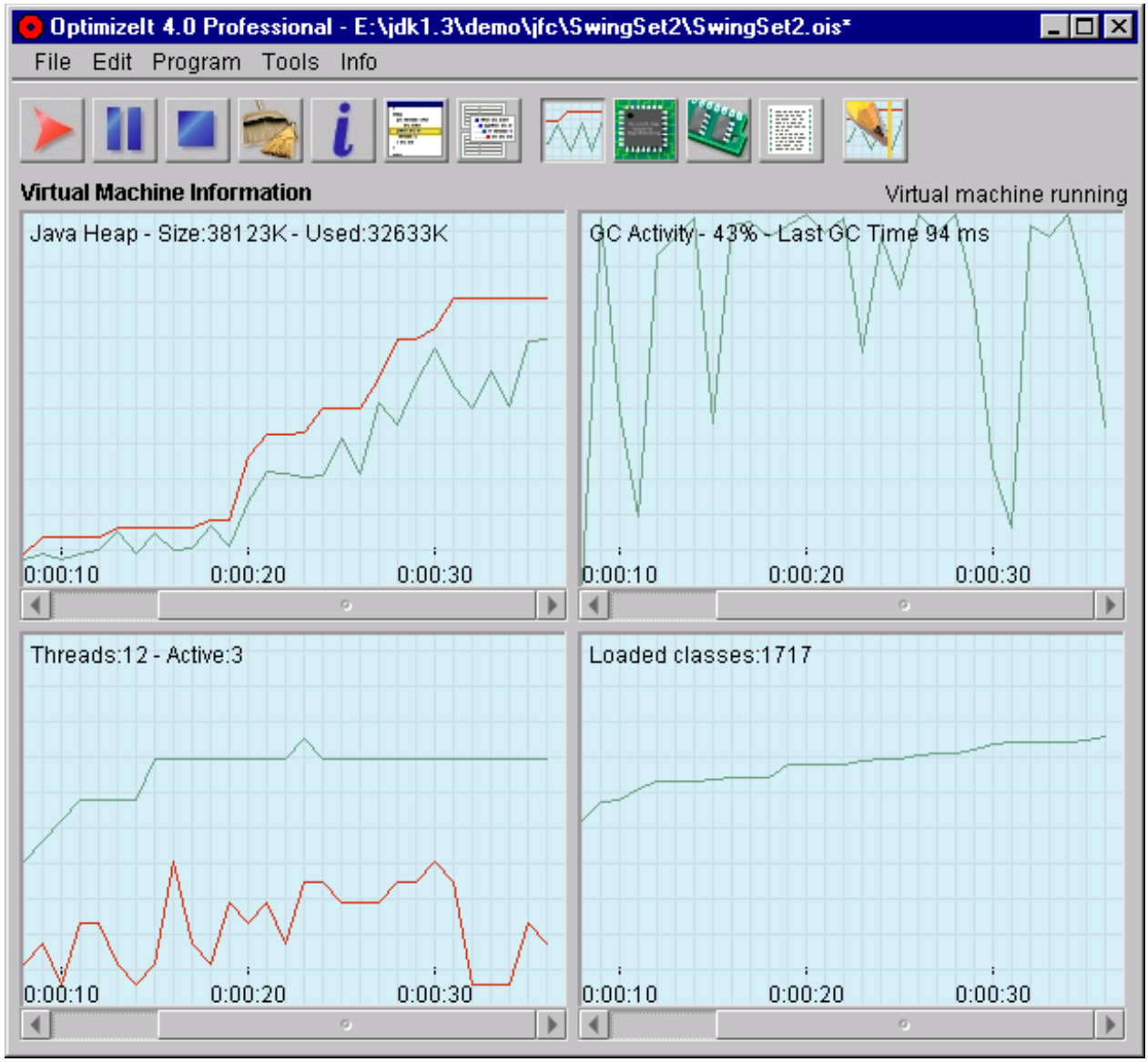
6.1 Using the virtual machine information mode

To switch to the virtual machine mode, click on the  button.The following screen shot shows OptimizeIt virtual machine information mode

Java



Java 2



Click the Mark button to mark the current point in time. A yellow mark appears on each graph.

Click the Export data button to export the graph.

6.2 Virtual machine information mode options

Click on the Inspector button to show the inspector. The virtual machine information mode has the following options:

Option	Description
Sampling period	This option defines how often OptimizeIt updates all charts.
Update when invisible	Defines whether graphs are updated when the virtual machine information mode is not currently selected. When this option is selected OptimizeIt updates the graphs all the time even when you are using the CPU profiler or the memory profiler.

7 Other features

This chapter includes the following sections:

7.1 Controlling the test program

The OptimizeIt toolbar provides the following buttons to control the test program. These buttons can be used from any mode:

 Starts or resumes the test program. This control is red when the test program is running.

 Pauses or resumes the test program. Use this button when it is necessary to freeze the flow of incoming data in the memory profiler to study some specific results more closely.

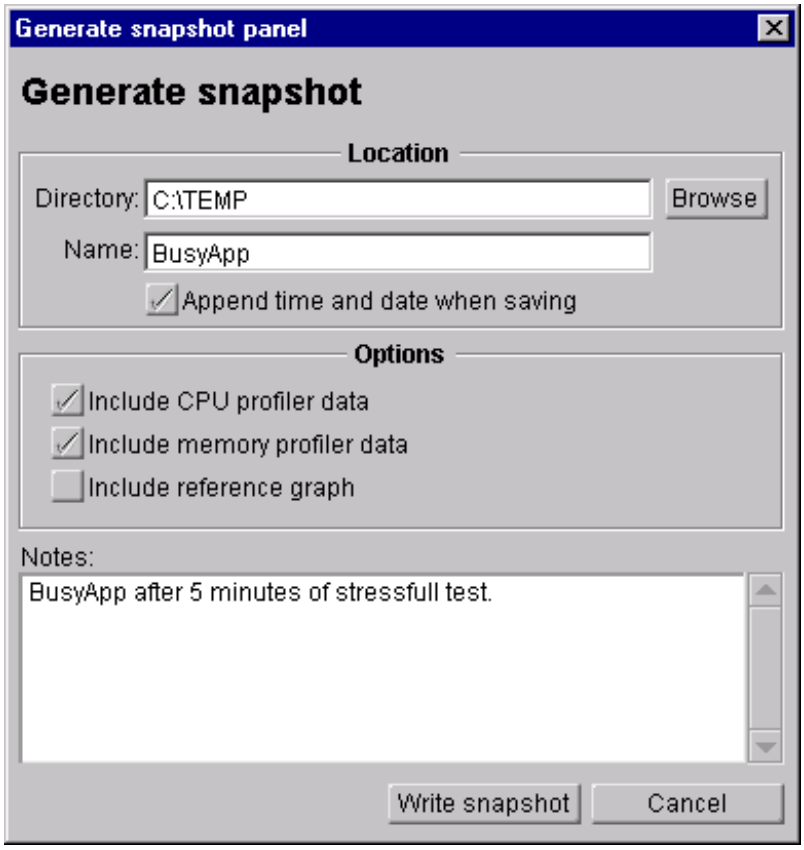
 Stops the test program. The test virtual machine exits.

 Forces the garbage collector to act immediately.

7.2 Generating a snapshot for the current profiling session

At any time during the profiling, OptimizeIt allows you to save the profiling data into a snapshot. You can then reload the snapshot later for further analysis or for performance comparisons.

To generate a snapshot choose Generate Snapshot from the File menu.



The following table describes each option:

Option	Description
Directory	Indicates the directory where the snapshot is created.

Option	Description
Name	The filename of the snapshot.
Include CPU profiler data	Indicates whether or not the CPU profiling information should be included. ***Note:*** *Generating a snapshot with this option stops the CPU profiler.* *If the CPU profiler is not running, this option has no effect*
Include memory profiler data	Indicates whether or not the memory profiler information (heap mode and backtrace mode) should be stored. ***Note:*** *The amount of data to be stored for the reference mode is important. The snapshots generated with this option are larger files and take more time to be generated. Only select this option when you really need the reference mode information.*
Comments	Contains other information, such as addition text to distinguish this snapshot from others.

Note: *The data of the virtual machine information mode is not stored in the snapshot.*

7.3 Opening a snapshot

OptimizeIt works the same way with snapshots as when profiling an application. Nevertheless, there are several restrictions that you should be aware of when you open a snapshot:

- the Virtual Machine Information mode is not available
- the console shows information about the opened snapshot (date of generation, host were it was generated, user that generated it, comments)
- the inspectors are disabled

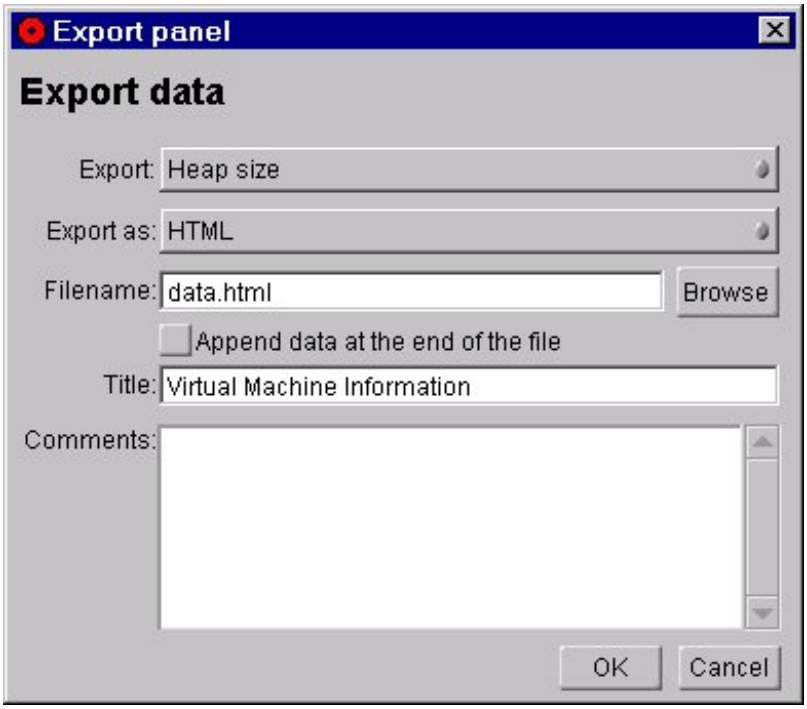
To open a snapshot, choose Open a snapshot from the File menu. Select a snapshot and click open. OptimizeIt opens the snapshot and you can browse the profiling information of the snapshot.

7.4 Exporting data

OptimizeIt can export profile data as ASCII, HTML, or easy-to-parse ASCII. After the data is exported it can be printed, compared and archived.

To export the contents of a screen, choose Export Data from the File menu. You can also click the Export Data button.

The Export Data dialog box provides options to set the content and format of the data exported:



The following table describes each option:

Option	Description
Export	Enumerates all data types that you can export from the current context. Use it to select the data to export.

Option	Description
Export as	Specifies the output of the file format. Select HTML to produce an HTML document that presents data in the same format as the Optimizeit views. Select ASCII for a more compact file. Select Importable ASCII if you expect to use the output as input to another tool.
Filename	Indicates the full path name of the file created.
Title	Contains a description inserted at the top of the exported document.
Comments	Contains other information, such as additional text to distinguish this profile from others.

After exporting data in the specified filename, OptimizeIt opens the file with the corresponding editor or web browser.

7.5 Viewing source code

When available, OptimizeIt can display the source code corresponding to a selection in any of the OptimizeIt windows.

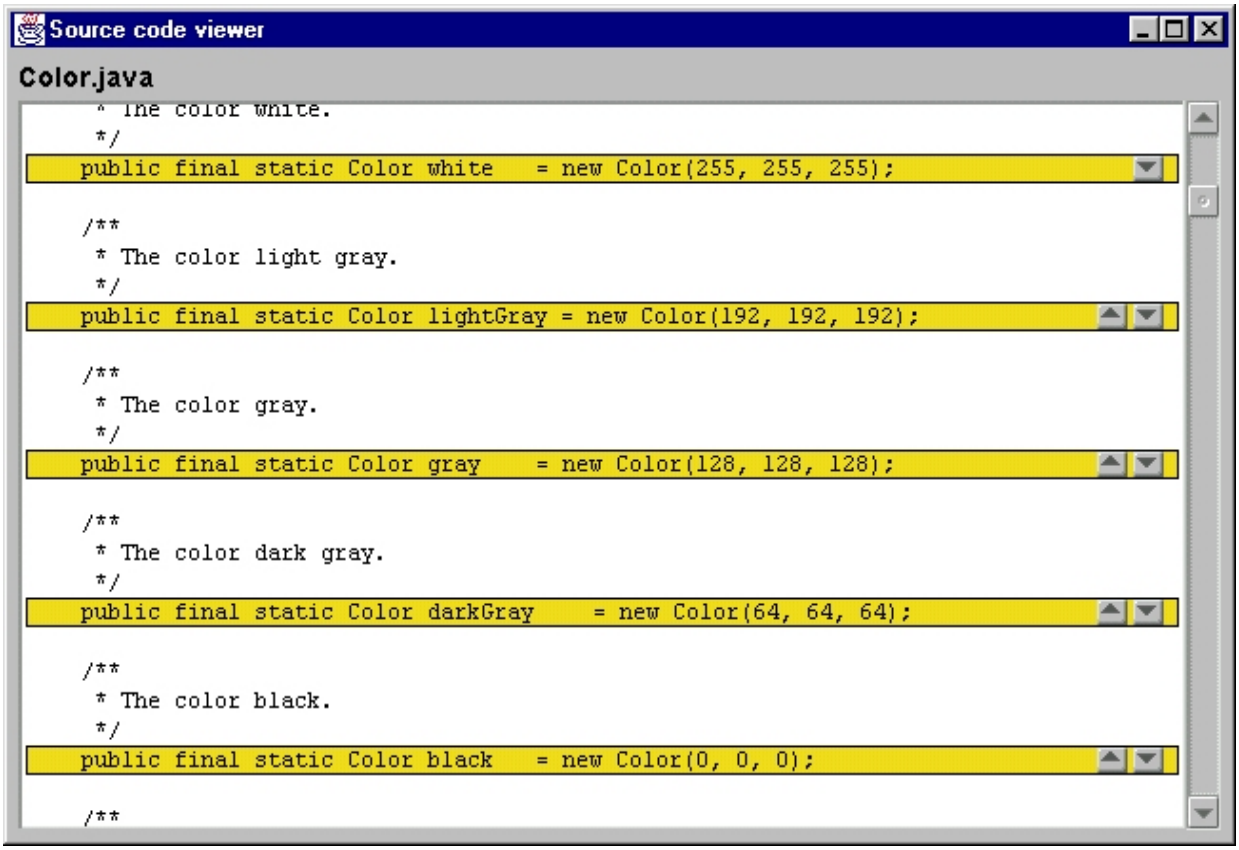
To view source code

Double-click the object or instance in the OptimizeIt window.

You can also click the Show Source Code button  .

OptimizeIt opens a separate window displaying the source code responsible for the allocation or definition of the selected object.

The following screen shot shows the source code viewer:



The relevant lines of code are highlighted in yellow. If more than one line of code is highlighted, use the small arrow buttons to automatically scroll to the next relevant line of code.

If the Java file is not found, the source code viewer provides a button that allows you to browse the file system to retrieve the file. Once the file is loaded, OptimizeIt prompts you to store the source file location in the default source path, so any Java file in the same package is immediately available.

7.6 Creating filters

Although OptimizeIt provides several filters which are ready to use, it is also possible to create your own filters, see “Profiling with filters” on chapter 3.10 for some information about using filters.

To create a new filter

1. Select Settings from the File menu.
2. Switch to the filter section by clicking the Filters tab.
3. Click the New button.
4. Enter the name of your filter in the name textfield.
5. Select "any of the following patterns" if you want to perform a logic OR between the different patterns of your filter, or select "all of the following patterns" if you want to perform a logic AND between the different patterns of your filter.
6. To add a pattern, click the Add button, then enter your pattern in the pattern editor. Note that both the asterisk wildcard character (*) and the not (!) are supported. Your pattern can define packages, classes or methods. When you are done with this pattern click OK.
7. To change a pattern, select the pattern in the list and click the Edit button.
8. To delete a pattern, select the pattern in the list then click the Delete button.
9. When you have finished with your filter, click OK. Notice that your filter appears at the end of the filter list. User filter names are written with bold letters, OptimizeIt filter names are written with normal letters.
10. Once you have created a filter, click on the corresponding column to enable it for the CPU profiler or the memory profiler.

Custom filters are part of OptimizeIt configuration files. They are saved and loaded when using Save and Open commands from the File menu.

7.7 Displaying OptimizeIt console messages

The purpose of the console is to print audit system-specific messages as well as the test program standard output and standard error. Use the console to read messages from the test program or to see errors if the Java program does not start.

Note: *If you select the option "Open a console" in the start options, the standard output and the standard error of the test program won't be redirected to the OptimizeIt console.*

7.8 Find panel

In any view, you can find the information you are looking for by using the find panel. The find panel is accessible from the edit menu.

Note: *Sometimes the list or graph in which the search occurs does not have all the information required for the search. When this happens, the find panel displays "Fetching data..." in its status field. Once the data is fetched, the search occurs.*

8 Integration with other Java environments

8.1 Integration with IDE

OptimizeIt can integrate with the following IDEs:

- IBM VisualAge For Java 2.0, 3.0, 3.5
- WebGain VisualCafe 2.0, 3.0 and 4.0
- Borland JBuilder 2.0, 3.0, 3.5 and 4.0
- Oracle JDeveloper 2.0, 3.0 and 3.1

Running the wizard

The following procedure shows how to integrate with an IDE:

1. Make sure the IDE has been installed correctly
2. Start OptimizeIt.
3. From the Tools menu, select the IDE integration submenu and select the option matching you IDE
4. OptimizeIt starts a wizard that guides you through the integration
5. Once the integration is performed, OptimizeIt gives you the option to quit OptimizeIt and start your IDE.

The OptimizeIt menu

OptimizeIt adds an OptimizeIt menu to your IDE. The location of this menu is indicated in the table below:

IDE	Location of the OptimizeIt menu
VisualCafe	Tools/OptimizeIt
JBuilder 2.0 and 3.0	From the JBuilder main menu
JBuilder 3.5	Tools/OptimizeIt
JDeveloper	From the JDeveloper main menu
VisualeAge	From the Visual Age's workbench, when a project, a package or a class is selected, from the Selected menu, Tools submenu, OptimizeIt option ***Note:*** *You can also access the OptimizeIt menu from Visual Age by selecting a project, a package or a class, then right click, and then select the menu Tools from the popup list-menu, OptimizeIt submenu.*

The following menus are available:

Menu	Description
Start profiling	Starts OptimizeIt if it is not already running and then executes the current project in OptimizeIt.
Stop profiling	Stops profiling the current project.
To front	Moves OptimizeIt's window in front of the IDE window. This can be useful when working with the IDE window maximized.
Close	Stops profiling the current project and causes OptimizeIt to exit.
Options	This menu shows a dialog box giving you access to OptimizeIt start options from the IDE.

Menu	Description
About	Gives information about OptimizeIt.

8.2 Integration with application servers

OptimizeIt can be integrated with most application servers that use Sun JDK 1.1, 1.2, 1.3 or IBM JDK 1.2 or 1.3. Once the integration is performed, the audit system can be started from the application server, and servlets can be profiled by attaching OptimizeIt to the audit system.

OptimizeIt provides wizards for easy application server integration. The following procedure shows how to start the integration:

1. Install the application server on your machine.
2. Start OptimizeIt.
3. From the Tools menu, select the Application server integration submenu and select your application server.
4. OptimizeIt starts a wizard that guides you through the integration.

Tutorials are also available. They describe the different steps carried out by the wizard to perform the integration. You can access the tutorials from the Help/Tutorial menu in OptimizeIt.

9 Index

O

P

R

S

T

V

W