Monitoring - Lab

IBM

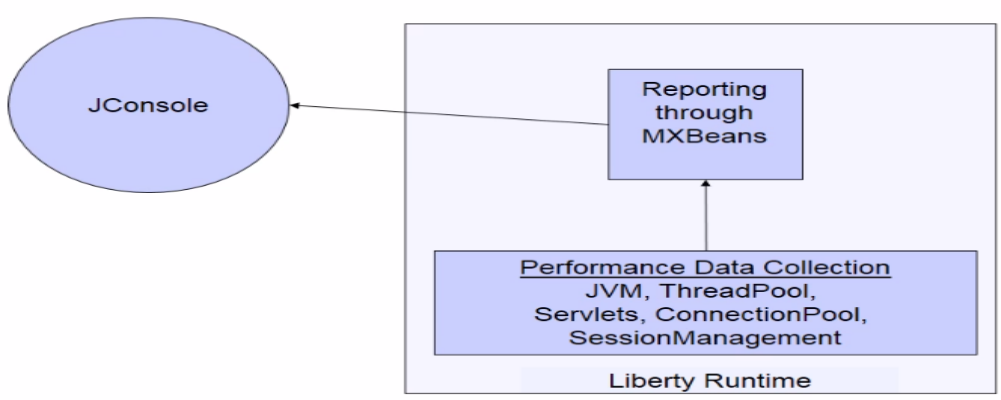
Contents

Monitoring

The **monitor-1.0** feature provides monitoring support for user runtime components like:

1. JVM
2. Web applications
3. Thread pool
4. Session management
5. Connection Pool

This monitoring feature is different from the Performance Monitoring Infrastructure (PMI) available in WebSphere Application Server. The monitoring feature collects performance data at runtime and the data is available as attributes on JMXBeans java objects. You can visualize the same data using Standard Java Development Kit tools such as **jConsole.** Also you can run a standard java client to fetch the performance values.



In this lab exercise, you will learn:

(1) Enabling **monitoring** feature.
(2) Use **JConsole** to monitor the Liberty Server
(3) Use **IBM Monitoring and Diagnostics Tool**s for **Java Health Center** to monitor the Liberty Server
(4) Introduce load to the server and debug a response time issue.

As prerequisites, you should:

- ✓ Complete the Setup lab to set up the Liberty runtime environment.

Note :

If you were running the other administration labs, stop the **adminCenterController** to avoid port conflict.

To run this lab, your workstation must meet the following requirements:

- Approximately 8GB of storage available for the Windows 7 virtual image
- Approximately 3 GB of memory free to run the developer workbench and the server
- Connectivity to the internet is NOT required
- Please refer to the following table for file and resource location references on different operating systems.

Location Ref.	OS	Absolute Path
{LAB_HOME}	Windows	`C:\WLP_<version>`
	Linux	`~/WLP_<version>`
	Mac OSX	

1.1 Enable Monitoring

In this section we will create a new server with monitoring feature and install the **MoneyBank** application.

__1. Change directory to **{LAB_HOME}\wlp\bin**

__2. Run the following command to create a new server

```
server create monitorServer
```

__3. Copy **{LAB_HOME}\labs\management\5_Monitoring<timestamp>\server.xml** to **{LAB_HOME}\wlp\usr\servers\monitorServer** directory.

__4. Review **server.xml**.

__a. **monitor-1.0** feature enables monitoring

__b. localConnector-1.0 feature enables local connector.

__c. **restConnector-1.0** and **ssl-1.0** features are required for remote monitoring.

__5. Copy **{LAB_HOME}\labs\management\5_Monitoring\MoneyBank.war** to **{LAB_HOME}\wlp\usr\servers\monitorServer\dropins** directory

__6. Start the server running the following command

```
server start monitorServer
```

__7. Point your browser to http://localhost:9080/MoneyBank

__8. Try some of the links on the web page to make sure they work. If the links do not work, make sure you are using the proper SDK as highlighted in the “Getting Started” lab.

1.2 Monitoring using JConsole

In this section we will start **JConsole** both locally and remotely for monitoring the application server.

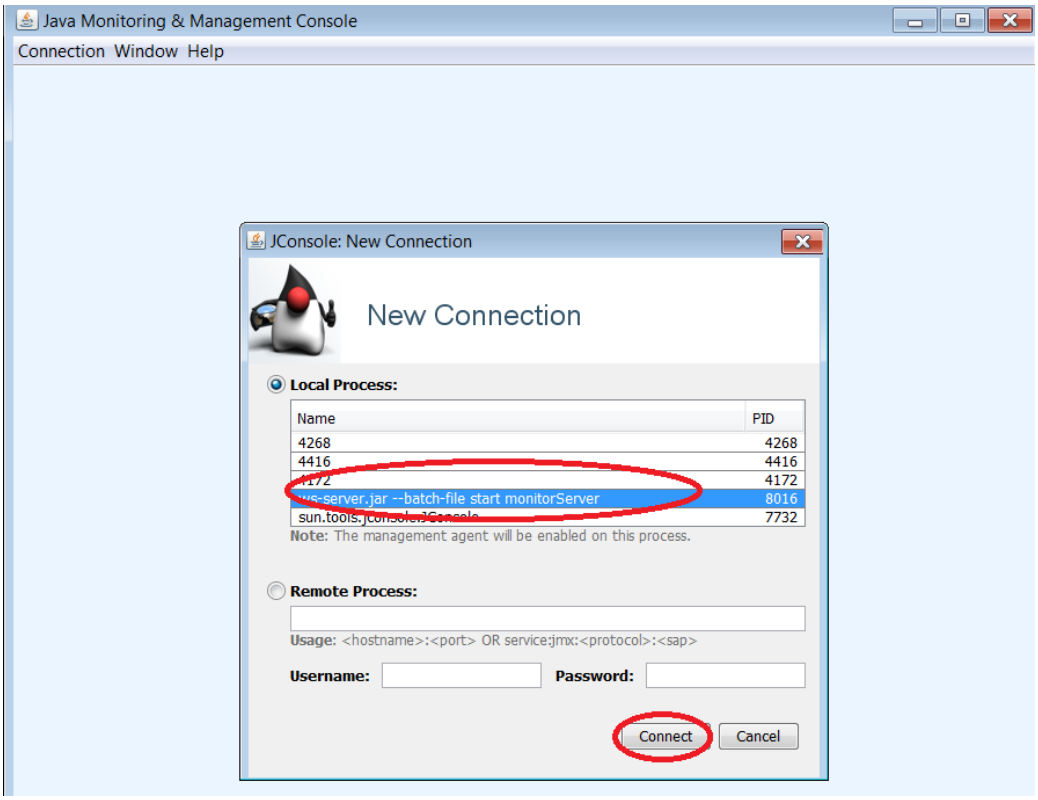
1.2.1 Local monitoring

With the **localConnector-1.0** feature enabled, you may start **JConsole** in local mode. This allows you to use **JConsole** to manage the server without having to provide a user ID and password, as long as the user running **JConsole** is the same as the user who started the **monitorServer** process.

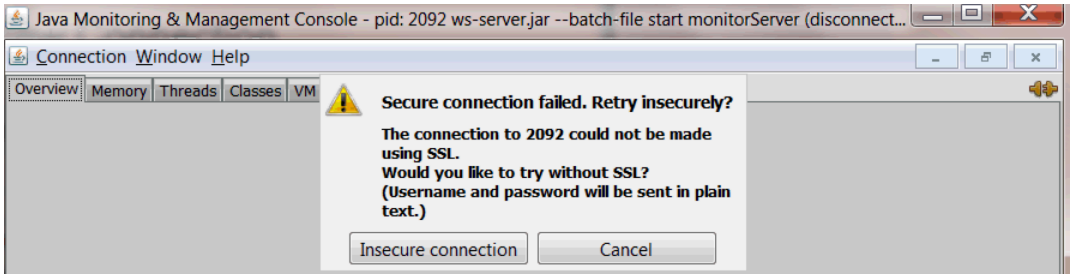
__1. Start **JConsole** executing the following command

```
Windows: {LAB_HOME}\wlp\java\bin\jconsole.exe
Linux: {LAB_HOME}/wlp/java/bin/jconsole
Mac:  jconsole should be on the path.
```

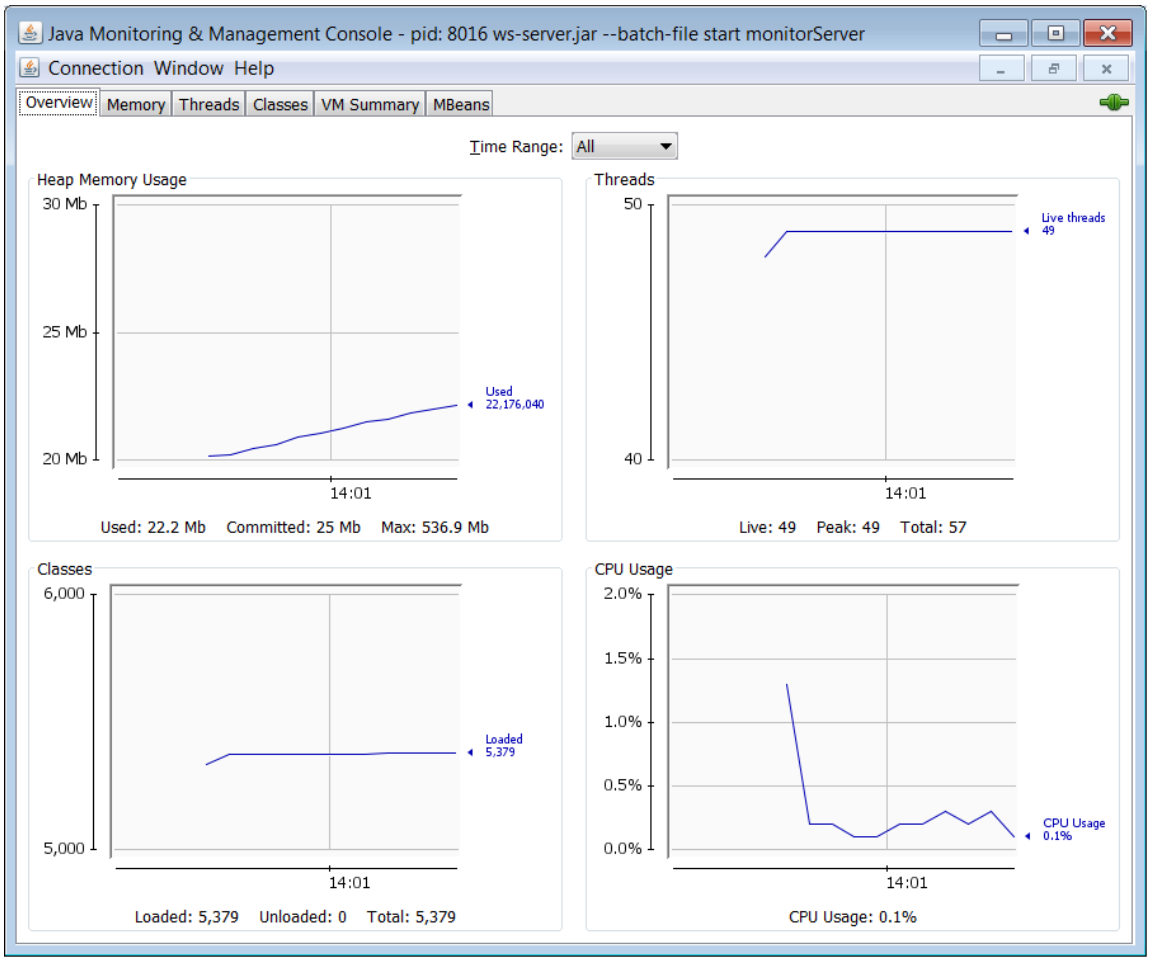
__2. Select **monitorServer** as the server to connect, then click **Connect**:



__3. A dialog box may appear after trying to connect securely. Click **Insecure connection.**



__4. After connecting you should see a graph of different JVM statistics as shown below:



1.2.2 Remote monitoring

__1. To set up remote monitoring using the REST connector restart **JConsole** with the following options:

```
jconsole -J-Djava.class.path={LAB_HOME}\wlp\java\lib\jconsole.jar;{LAB_HOME}\wlp\java
\lib\tools.jar;{LAB_HOME}\wlp\clients\restConnector.jar -J-
Djavax.net.ssl.trustStore={LAB_HOME}\wlp\usr\servers\monitorServer\resources\security\
key.jks -J-Djavax.net.ssl.trustStorePassword=mysecret -J-
Djavax.net.ssl.trustStoreType=jks
```

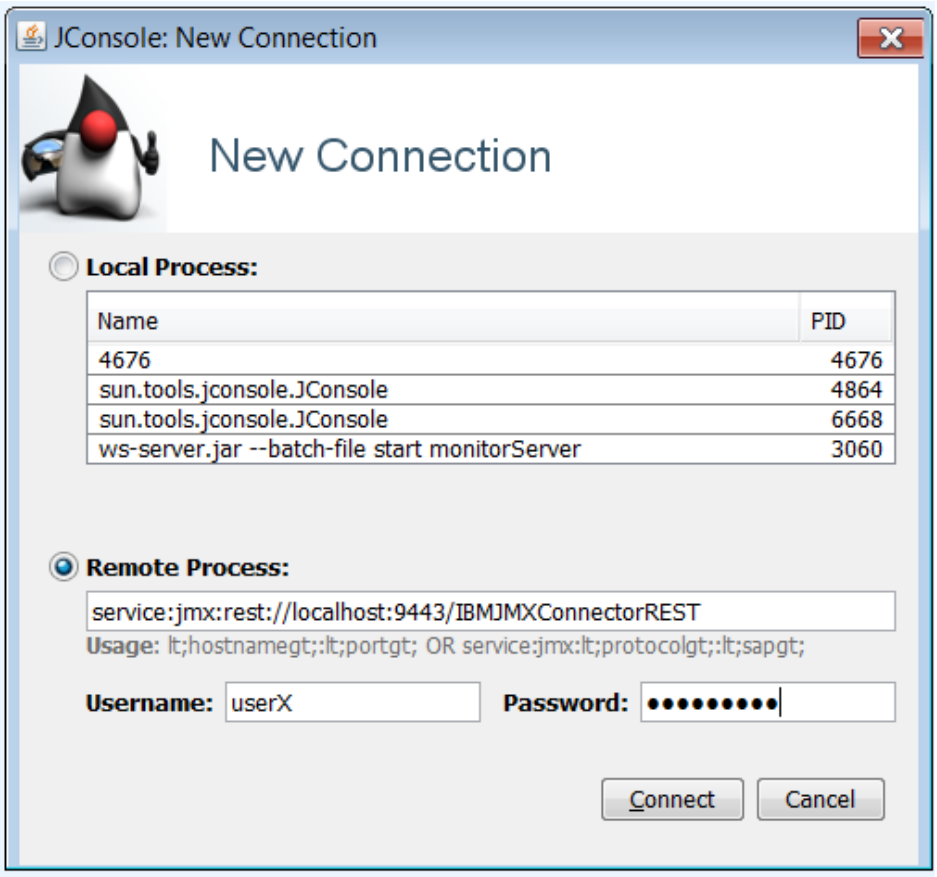
Note: copying & pasting from this document may not work. Make sure there are dashes '-' before and after the four 'J' letters and spaces are correct. ***For Linux and Mac the jar separators on the classpath should be ":" colons not semicolons. Windows are semicolons. Mac users: wlp\java jars on classpath already.***

__2. When prompted for **New connection**, enter the following:

__a. **URL**: service:jmx:rest://localhost:9443/IBMJMXConnectorREST

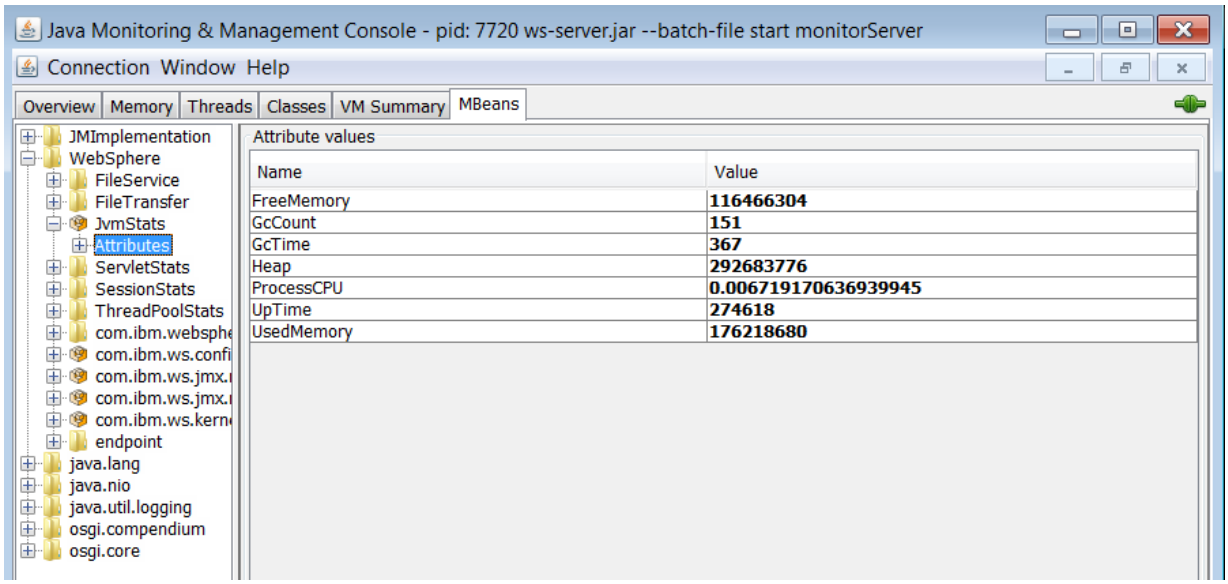
__b. **User**: userX

__c. **Password**: passwordX



1.2.3 Navigation within JConsole

_1. Click on the **MBeans** tab then expand **Websphere → JVMStat → Attributes** to look at the JVM attributes:

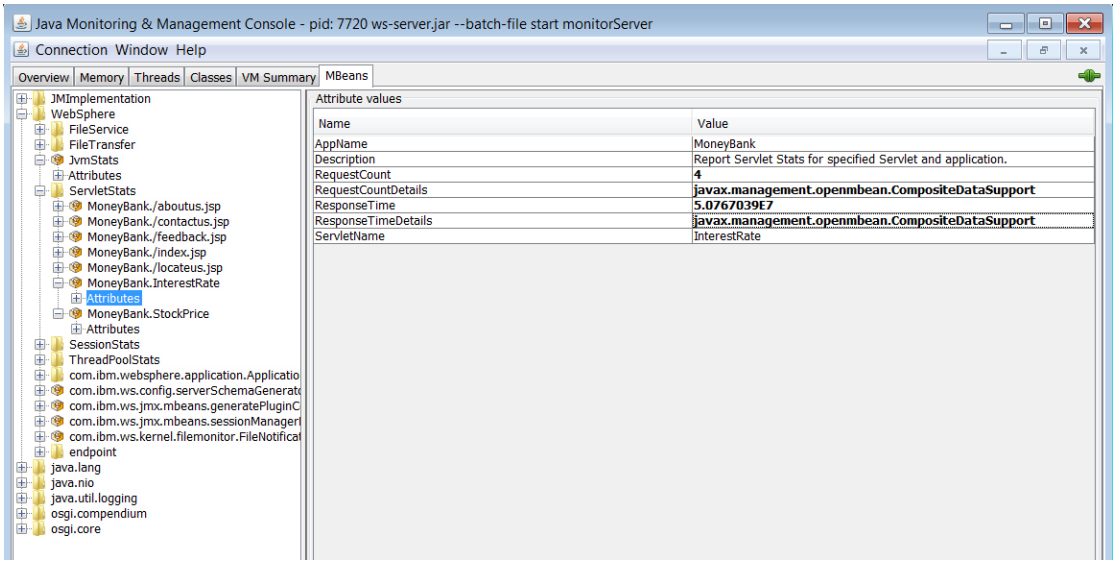


__2. Expand **WebSphere → ServletStats → MoneyBank.InterestRate**.

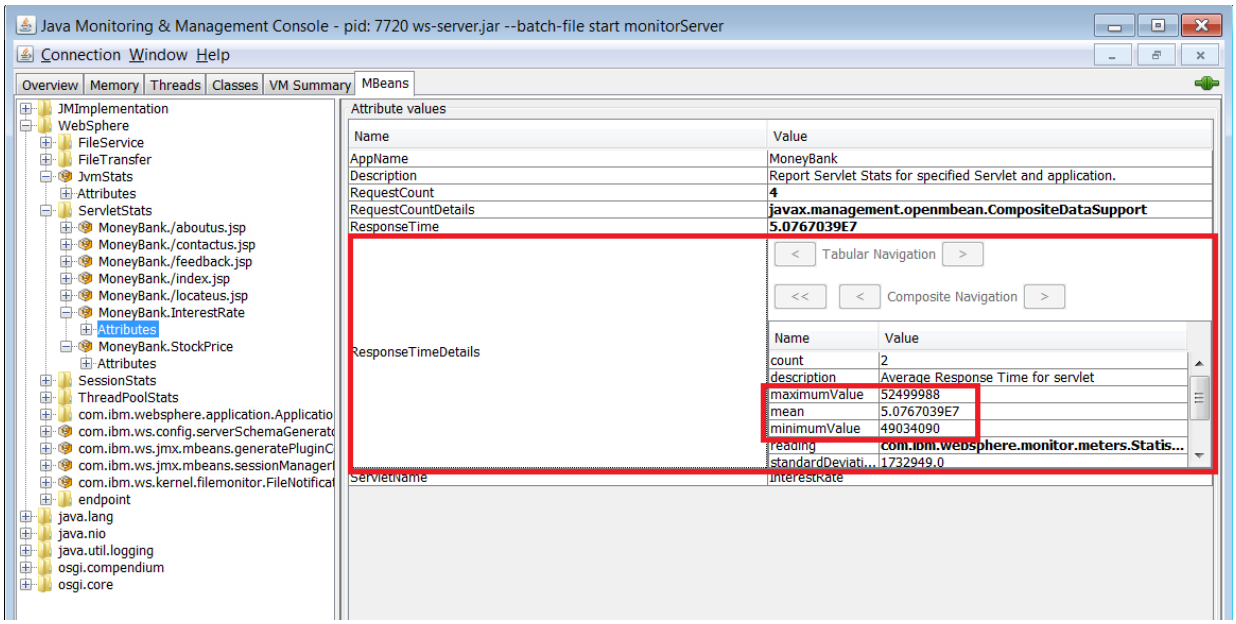
Note :

If you don't see **MoneyBank.InterestRate** under **ServletStats**, first point your browser to:
http://localhost:9080/MoneyBank/InterestRate

__3. Note the **response time** of **5.07 seconds**. (This may vary for your machine) This is a problem that we'll be using the **Health Center** later to identify the root cause.



__4. Double click one of the complex types, such **as javax.management openmbean.CompositeDataSupport** that corresponds to response time detail to get more information. Note that it shows **minimum, maximum**, and **mean** values:



__5. Reload the page to: http://localhost:9080/MoneyBank/InterestRate , then refresh the **JConsole** screen by clicking the **Refresh** button at the bottom and check if the **RequestCount** went up by one.

__6. Explore the attributes of the other **MBeans**, such as **ThreadPool**.

__7. Close **JConsole**

1.3 IBM Monitoring and Diagnostics Tools for Java Health Center

The **IBM Java Health Center** contains a rich set of additional JVM monitoring function to help you monitor and diagnose your application server. Some of these additional benefits include:

__a. Discovery of native memory and java heap leaks

__b. Method profiling to discover which methods are taking the most time. Using sampling, it requires very little CPU overhead.

__c. Identifying I/O bottlenecks

__d. Threads: Lock contentions, Deadlock detection

__e. Class histogram

For more information, visit: http://www.ibm.com/developerworks/java/jdk/tools/healthcenter

Normally **Health Center** is run from within the **IBM Support Assistant**, an Eclipse based environment offering. Many other support related tools are also available as part of **IBM Support Assistant,** such as :

__a. **Garbage Collection and Memory Visualizer** for analyzing garbage collection behavior

__b. **Memory Analyzer** for analyzing heap dumps

__c. **Thread and Monitor Dump Analyzer** for monitoring thread dumps and

__d. **Log Analyzer** for analyzing WebSphere logs.

Another option is to install it directly from the eclipse marketplace.

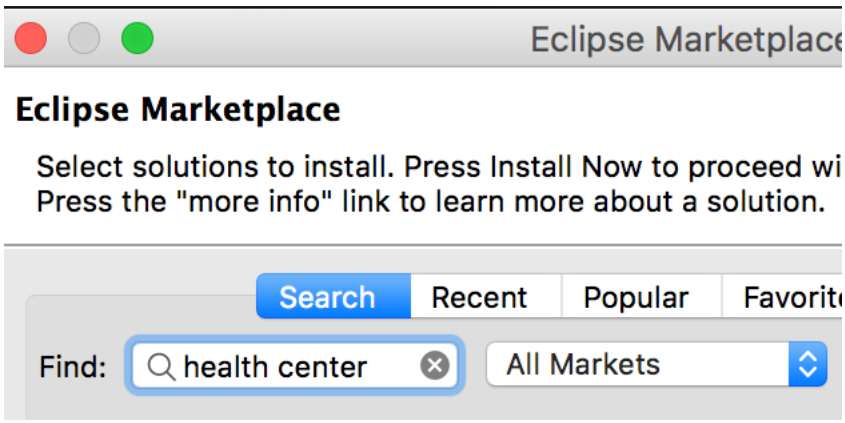
For this lab, we have included one version in the **{LAB_HOME}** directory.

1.3.1 Installing IBM Runtime Monitoring and Diagnostic Tools

__1. IBM Runtime Monitoring and Diagnostic Tools has already been installed in WDT. Note: There is no monitoring agent for the MAC JVM. There is one for the IBM JVM for Windows or Linux. **LINUX BUG WARNING:** Eclipse/Ubuntu/Linux has a bug that is exploited by Health Center. There is ongoing work to fix Health Center, but the rest of the lab will not work.

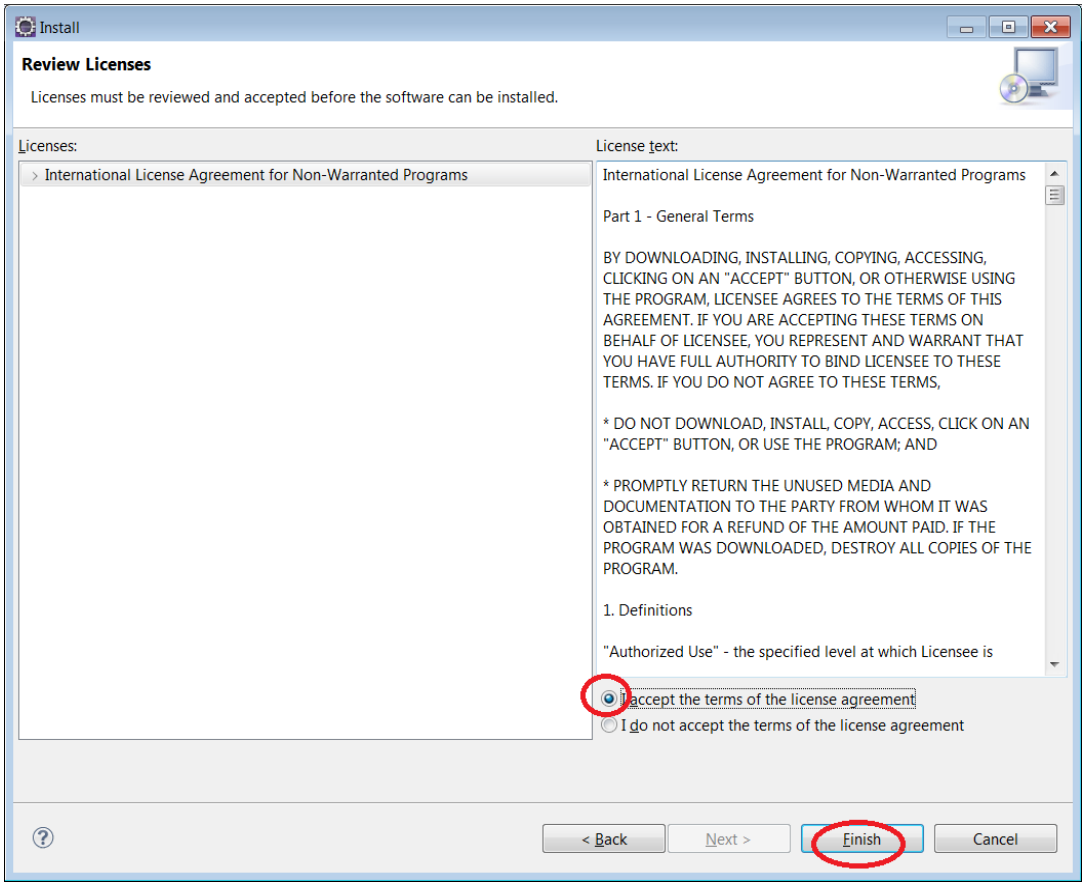
__a. If you want to install on your own Eclipse. **Go to Help > Eclipse Marketplace …**

__b. Search for **“health center”**



__c. Click **Install**

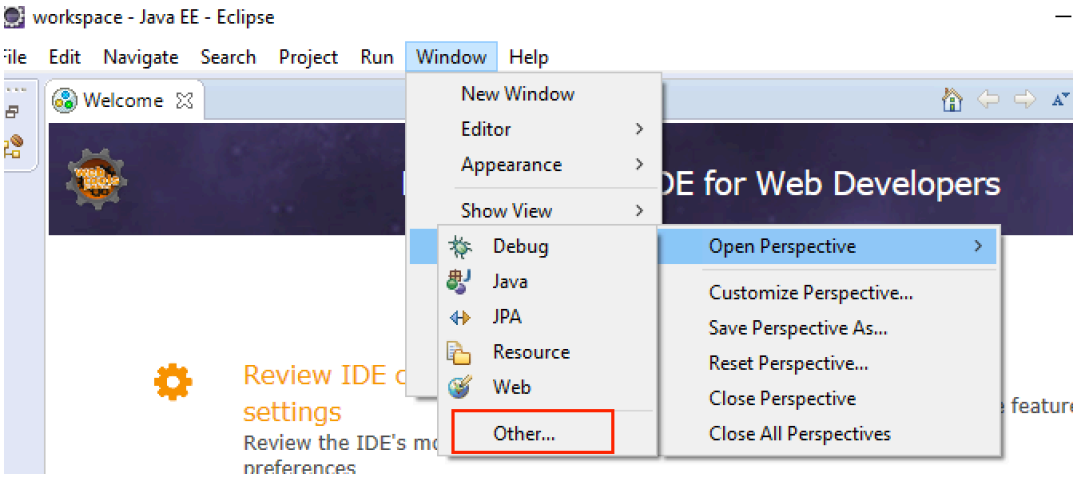
__d. Select **Accept the terms of the license agreement** and Click **Finish**



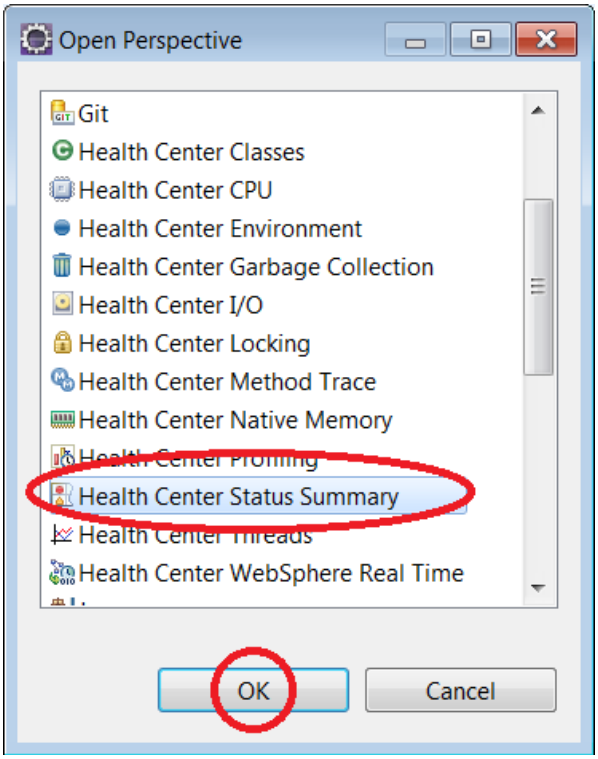
__2. If prompted to restart eclipse, click **Yes**

1.3.2 Configuring the Health Center Agent for Liberty Server

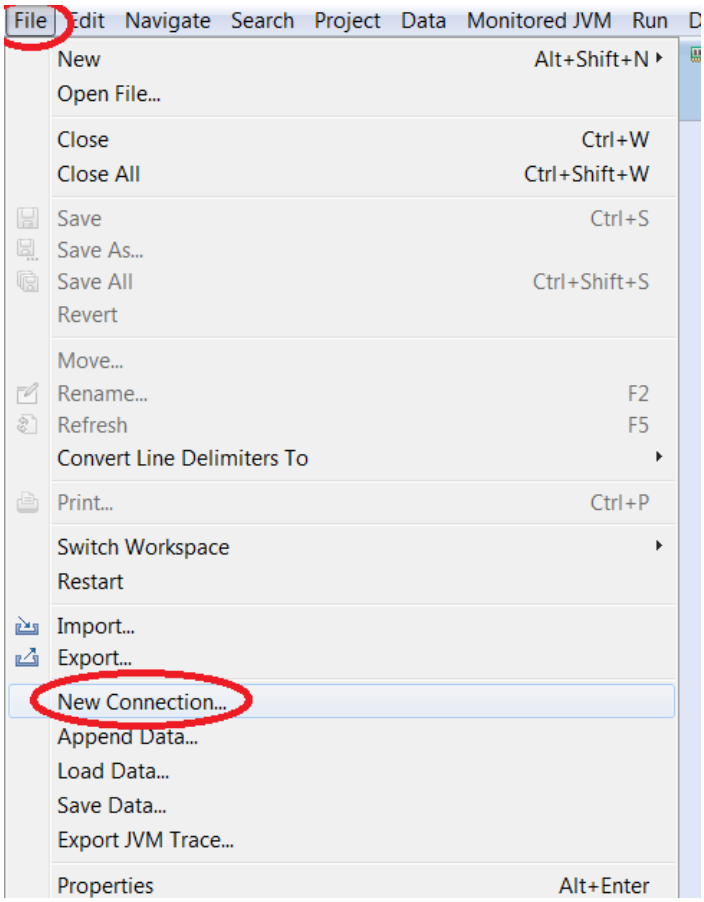
__1. From eclipse, navigate to **Window → Open Perspective → Other**



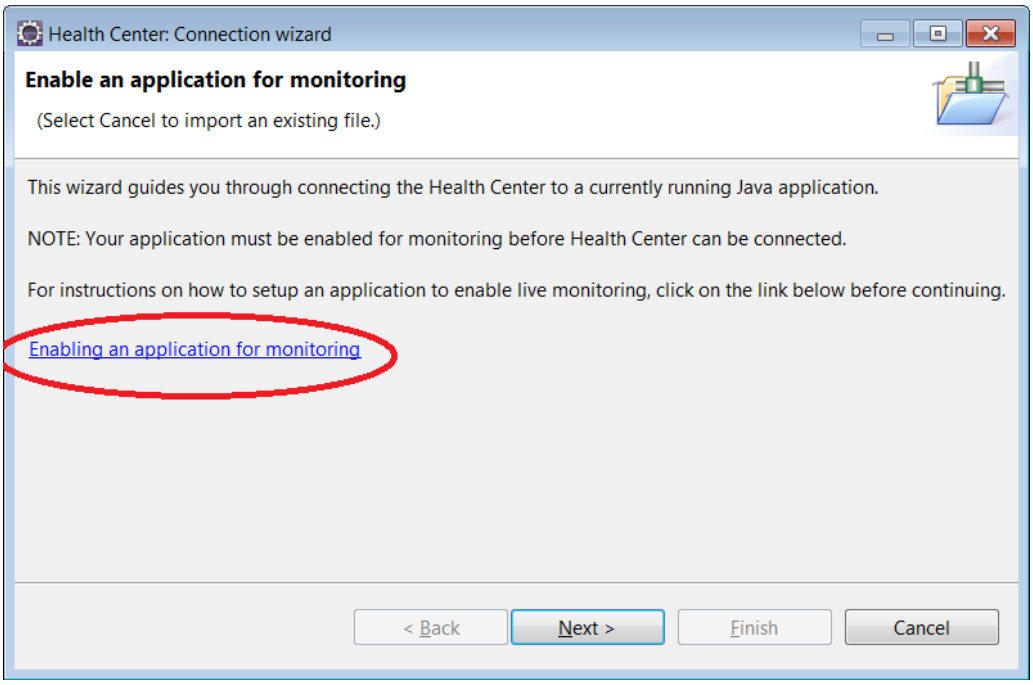
__2. Select **Health Center Status Summary**



__3. Navigate to **File → New Connection**



__4. Verify that you already have the **Health Center Agents** installed by inspecting your **JRE** and ensuring that **jre\lib\ext\healthcenter.jar** exists. The **Java SDKs** included with this lab already have the agents installed. If you don't see the agent, click **Enabling an application for monitoring** and follow the instructions to install the agent for your platform.



__5. Copy **{LAB_HOME}\labs\management\5_Monitoring\jvm.options** included with this lab to **{LAB_HOME}\wlp\usr\servers\monitorServer directory**.

__6. Inspect the file **jvm.options** and note the JVM options used to configure the agent. The configurations include :

__a. Enablement of verbose garbage collection

__b. Enablement of health center

__c. Location of file to store health center data gather by the agent in the JVM

__7. Restart **monitorServer**

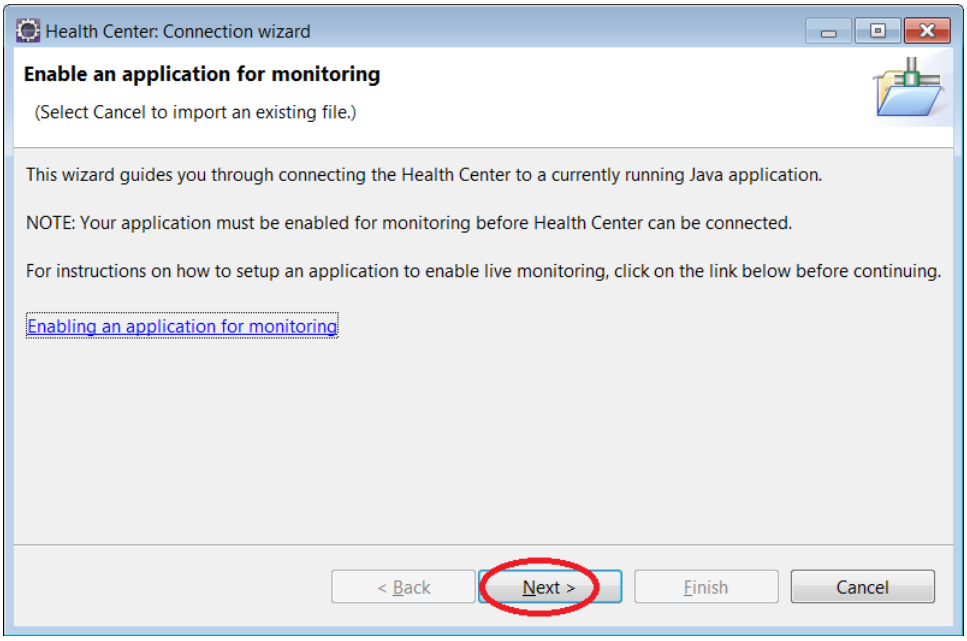
```
server stop monitorServer

server start monitorServer
```

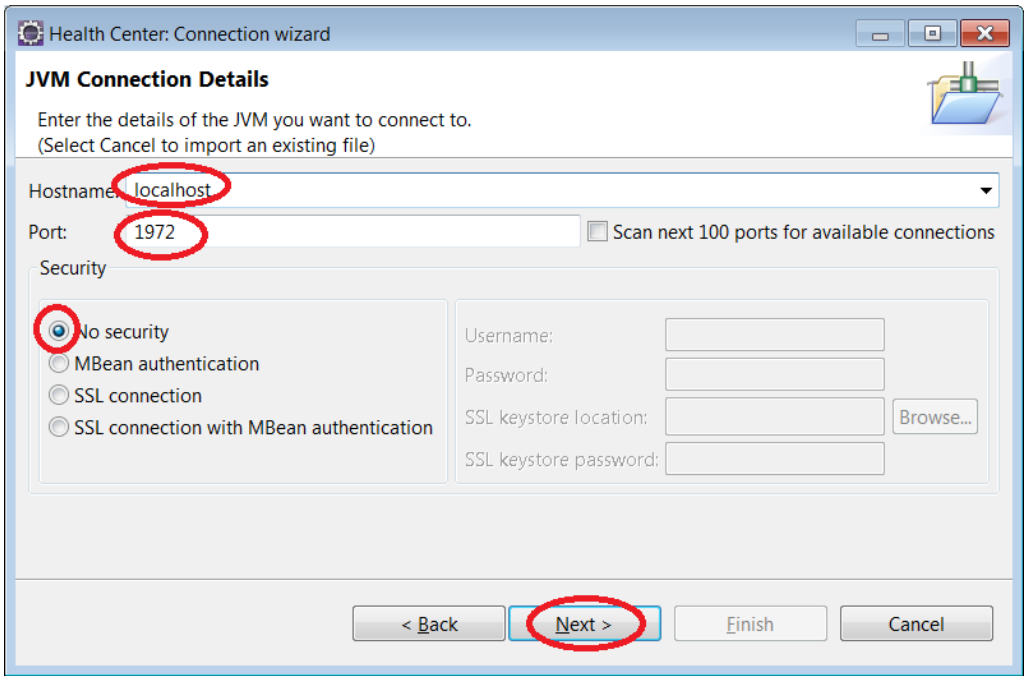
__8. Check **console.log** and ensure that the health center messages and port number are printed as shown below :

```
INFO: Health Center agent started on port 1972
```

__9. Back in eclipse, click **Next**



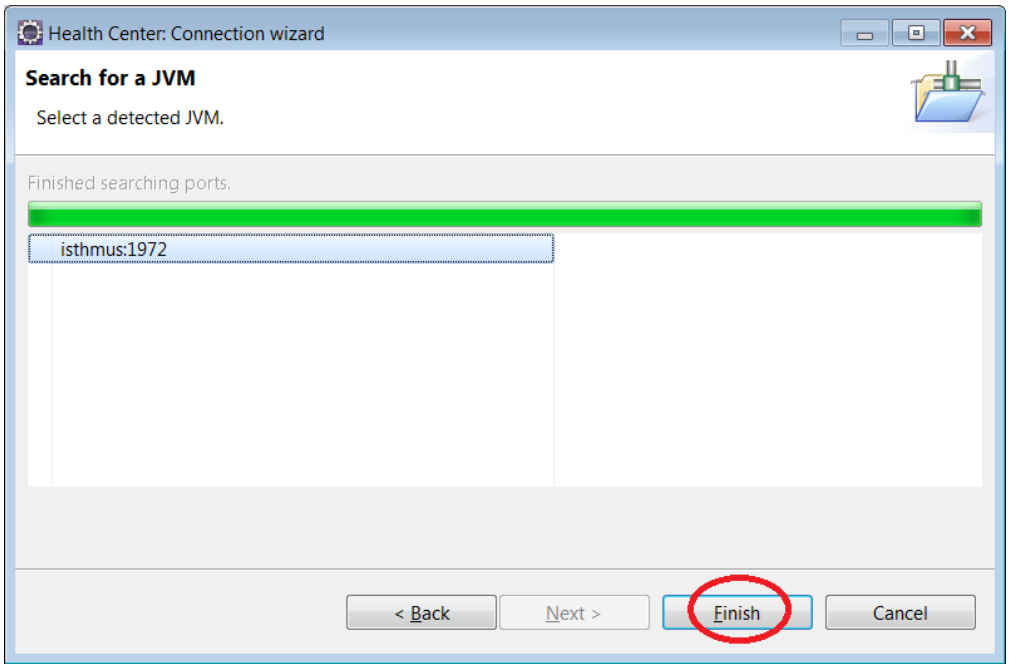
__10. Enter **localhost** for ***Hostname***, **1972** for ***Port***



__11. Click on **No Security** radio button as security is not enabled on the Liberty Server.

__12. Note that you will want to enable security if accessing the server remotely, but it is out of scope of this lab.

__13. Click **Finish**



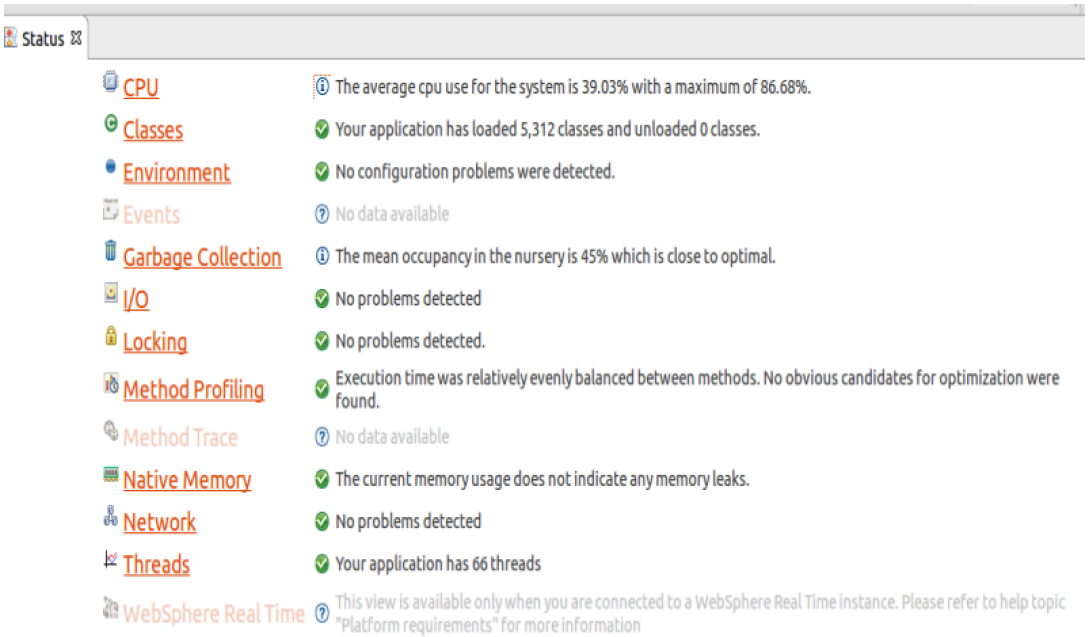
1.3.3 Using IBM Monitoring and Diagnostics Tools for Java Health Center

__1. Look at the **Status** pane, and note that what is unavailable:

__a. **CPU utilization**: a current restriction

__b. **Method Trace**: requires additional configuration, and outside the scope of this lab

__c. **WebSphere Real Time**: requires real time option for the JRE



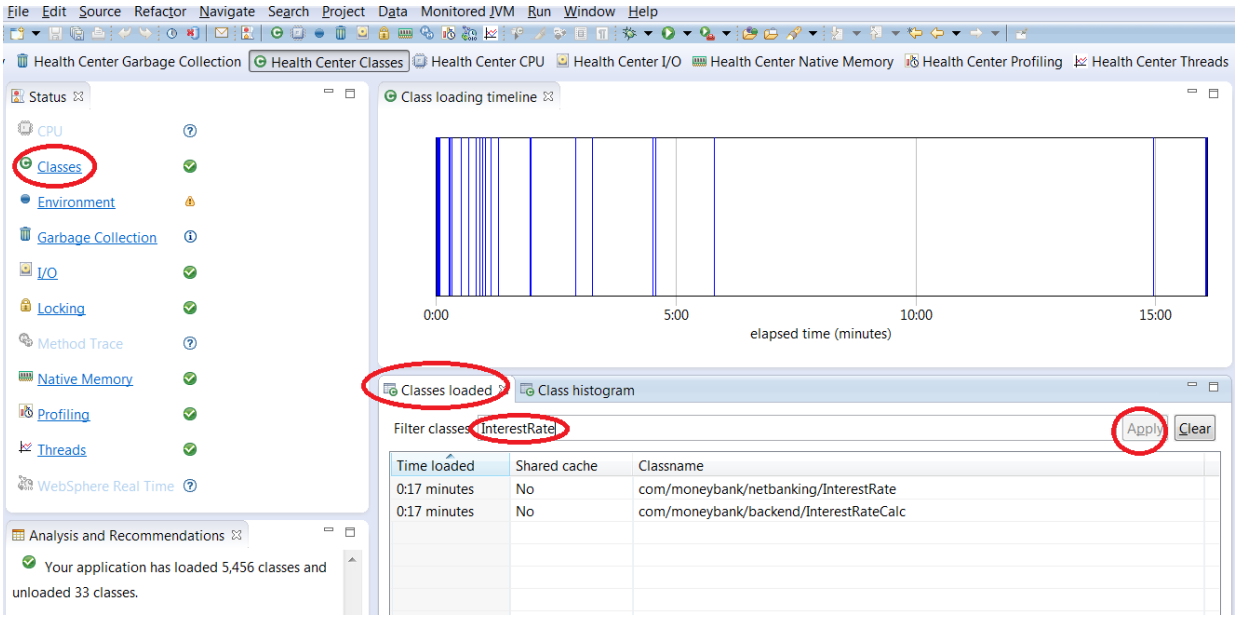
__2. Click **Classes** to see the list of classes loaded in the JVM.

__3. Enter **InterestRate** as filter and click **Apply**.

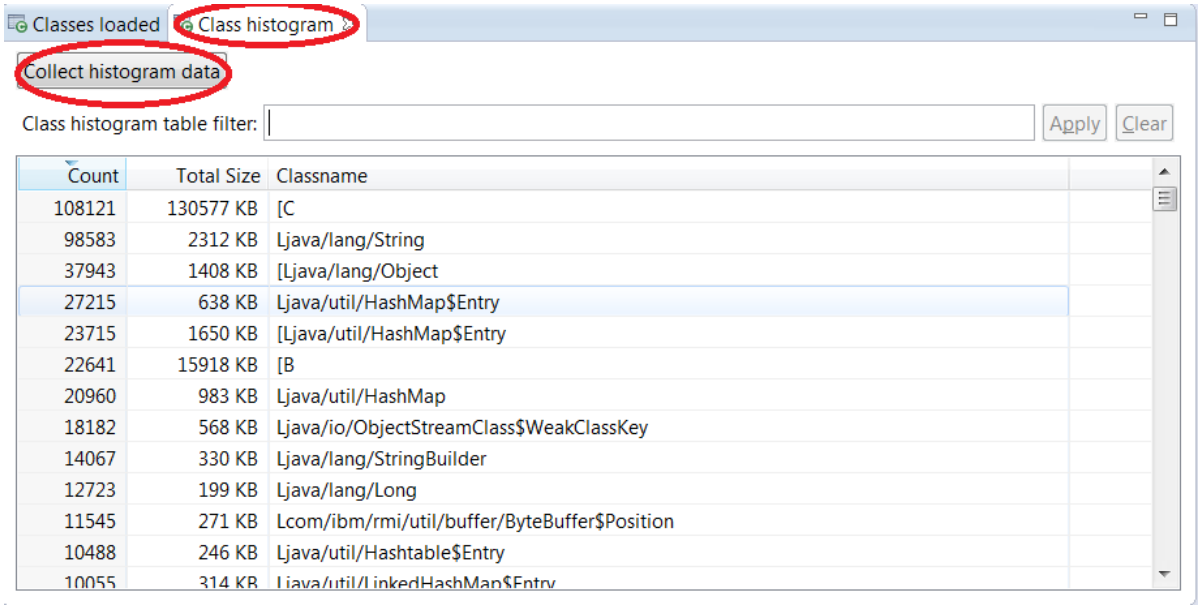
Note :

If you don't see the classes, first access the below web page.

http://localhost:9080/MoneyBank/InterestRate



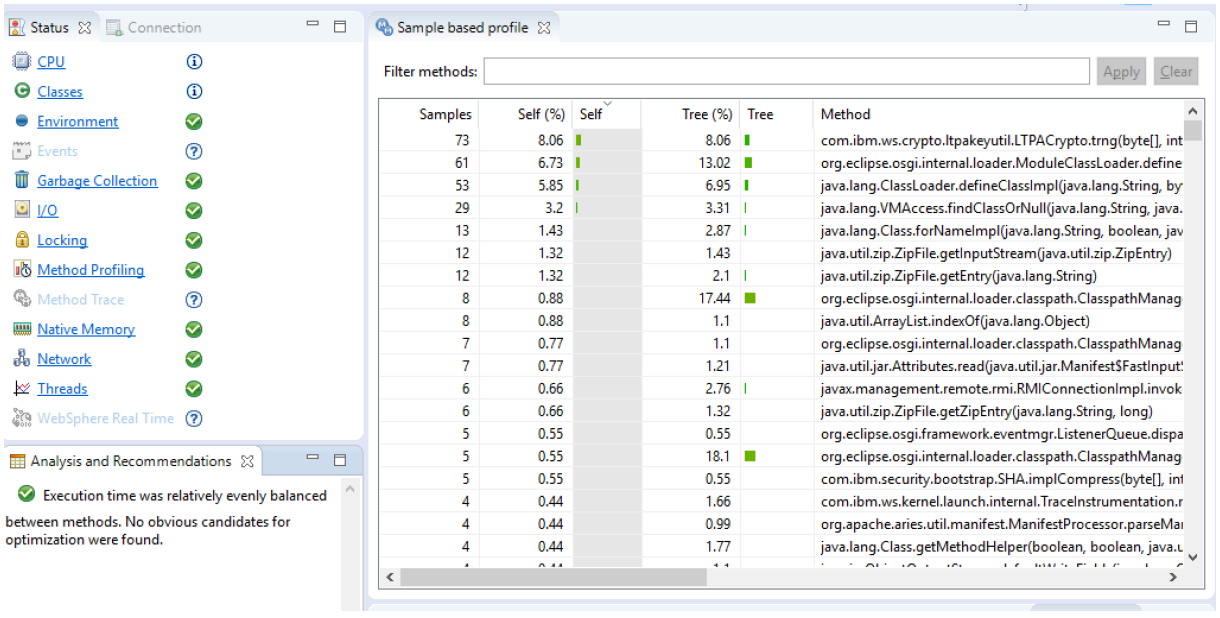
__4. Click **Class Histogram**, followed by **Collect histogram data**. Wait for the data to appear. This tells you the number of instances for each class, sorted by number of instances.



__5. Click **MethodProfiling** then click **Self** to sort in descending order.

Note: For some Linux people, this yields an Operating System stack overflow box which you cannot recover.

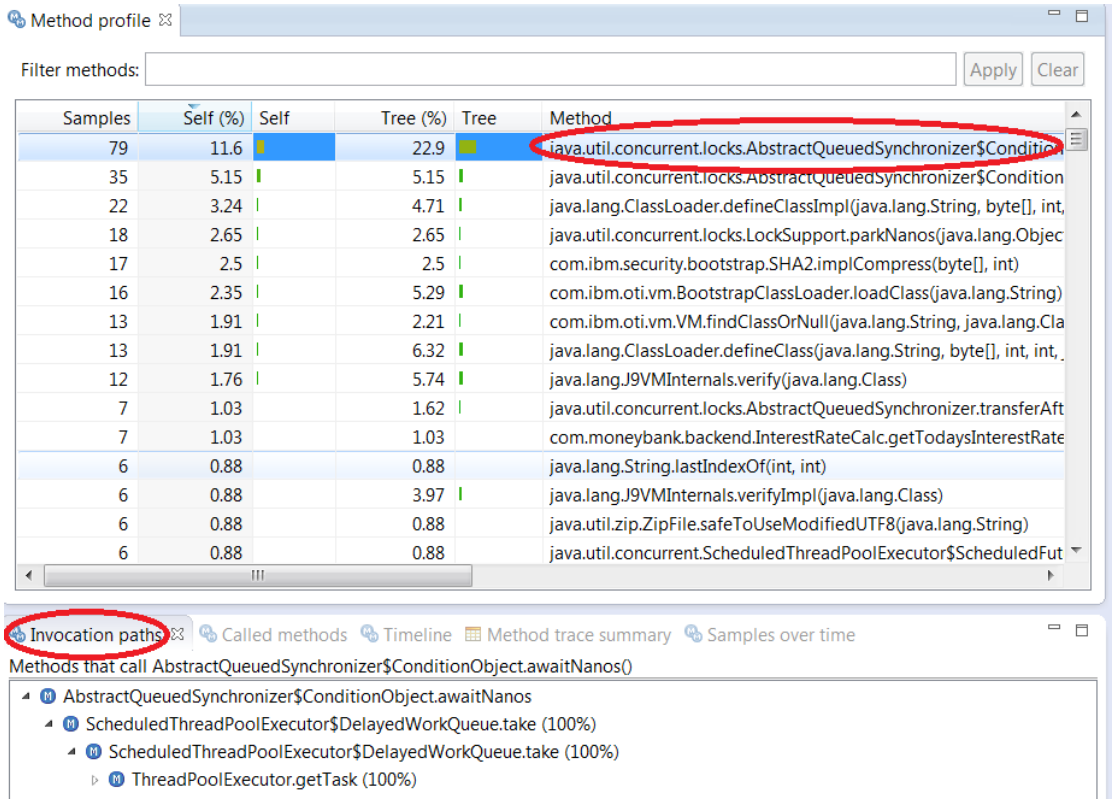
Profiling helps you gauge which methods are taking the most time. It employs statistical sampling to record which methods are running. Methods that are called often, or take a long time to complete, are shown. The **Self** column lists proportion of time each method has been sampled. The one with the largest percentage is sampled the most often, and therefore either called the most often, or spends the most time.



Note :

Profiling is not useful if the application is spending significant time doing I/O, or in garbage collection, or waiting for locks. Therefore, you should check those areas in the health center as well.

__6. Click on one of the methods. The **invocation paths** shows who is calling the method, and the percentage for each caller.

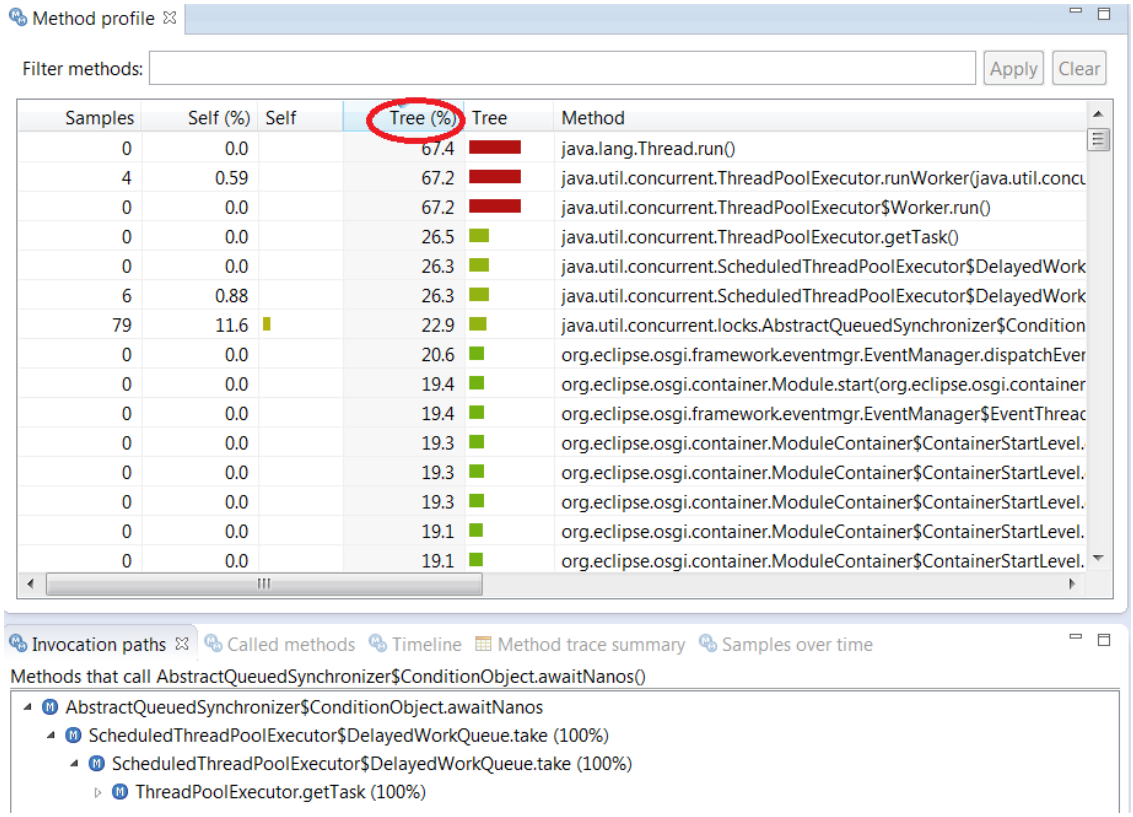


__7. Click **Called methods** to show the methods being called by the method you selected, and percentage of samples

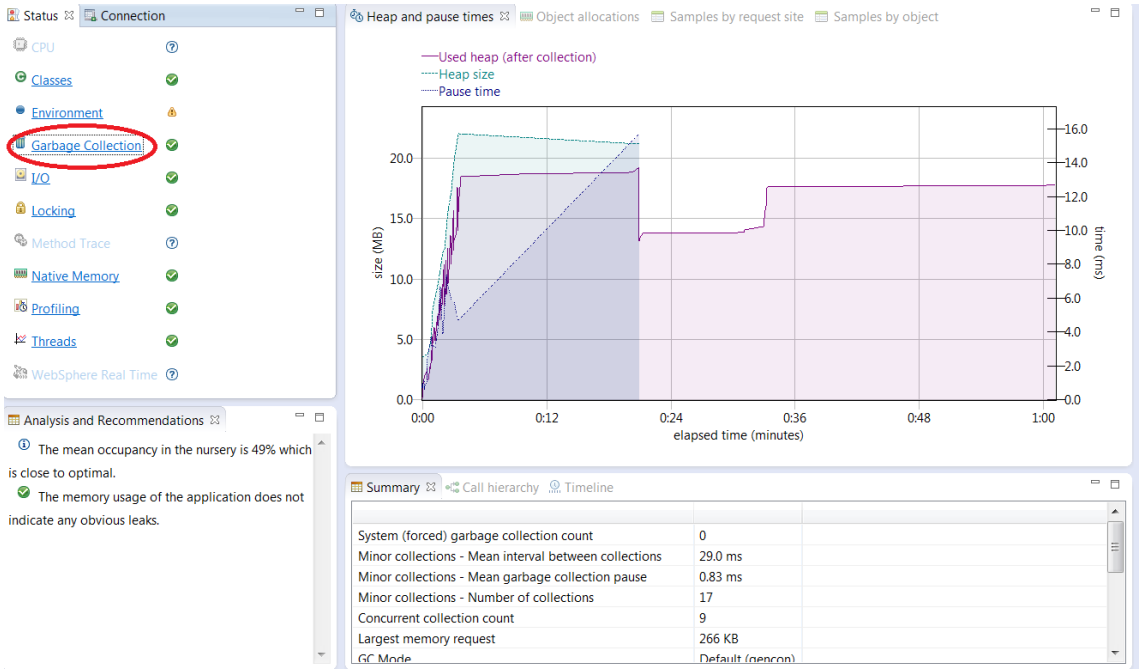
__8. Click **Timeline** to see when the method is called.

__9. Click **Samples over time** to show the number of methods in the sample aggregated every two seconds.

__10. Back in the **Method profile** pane, click **Tree (%)** to sort in descending order. This gives the percentage of samples where the method is found on the call stack. This can also be a good indication of where the code is spending most of its time



__11. Click **Garbage Collection**, and note the information being reported in various panes. Since the JVM is in equilibrium without load, there is not much of interest to see. Take note of the heap size, GC pause time and view the GC summary.



__12. Click on **"Native Memory**" to look at native memory usage.

__13. Click on **I/O** → **Files Open** to view the number of open files over time.

__14. Click **File I/O** to view when files are opened or closed.

__15. Click **Locking** and view the **Monitor** pane. Hover the Mouse on each column name to get a better description.

For Example:

The column **%miss** is the percentage of **"Gets"** that result in a miss. A miss occurs when a thread tries to get a lock already owned by some other thread. A high number of misses is an indication of lock contention.

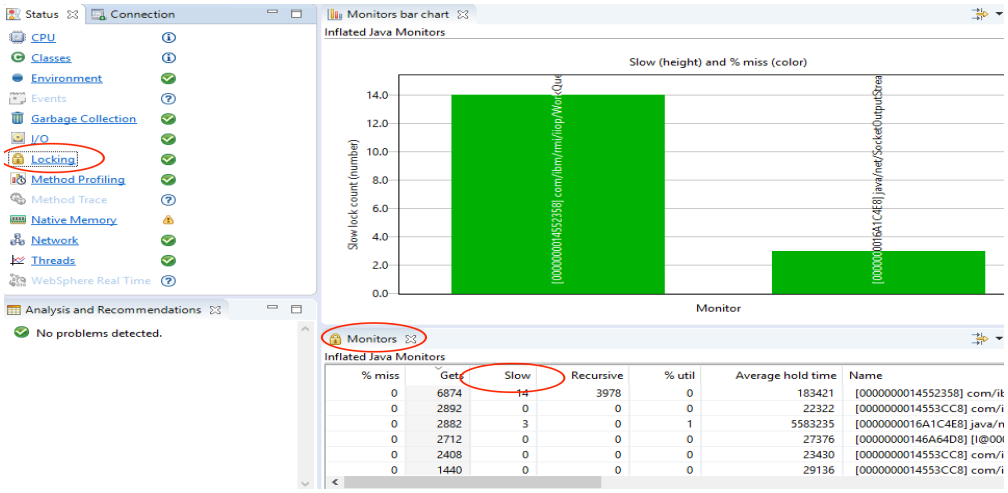
The **Gets** column shows the number of times a thread tries to get a monitor.

The **Slow** column shows the number of "gets" that result in a miss.

The **Recursive** column shows the number of times a thread gets a monitor that it already owns.

The **% util** is the percent utilization, meaning the amount of time the lock was held divided the total time spent monitoring the JVM.

Average hold time is the average time a thread spends while holding the monitor, measured in processor ticks.



__16. Click **Threads** to see a list of threads, and their states. Click on each thread to see what monitor it owns, and what it's waiting on.

1.4 Using IBM Monitoring and Diagnostics Tools for Java Health Center to Debug a Response Time Issue

In this section, we will place the application server under load using **JMeter**, while monitoring from the health center to resolve the high response time issue identified earlier through **JConsole**.

__1. Install **JMeter** by unzipping the contents of **{LAB_HOME}\apache-jmeter-<version>.zip** to **{LAB_HOME}\apache-jmeter-<version>**

__2. Set **JAVA_HOME** and **PATH** variables

__a. Windows

SET JAVA_HOME={LAB_HOME}/wlp/java/jre
SET PATH=%JAVA_HOME%/bin;%PATH%

__b. Linux

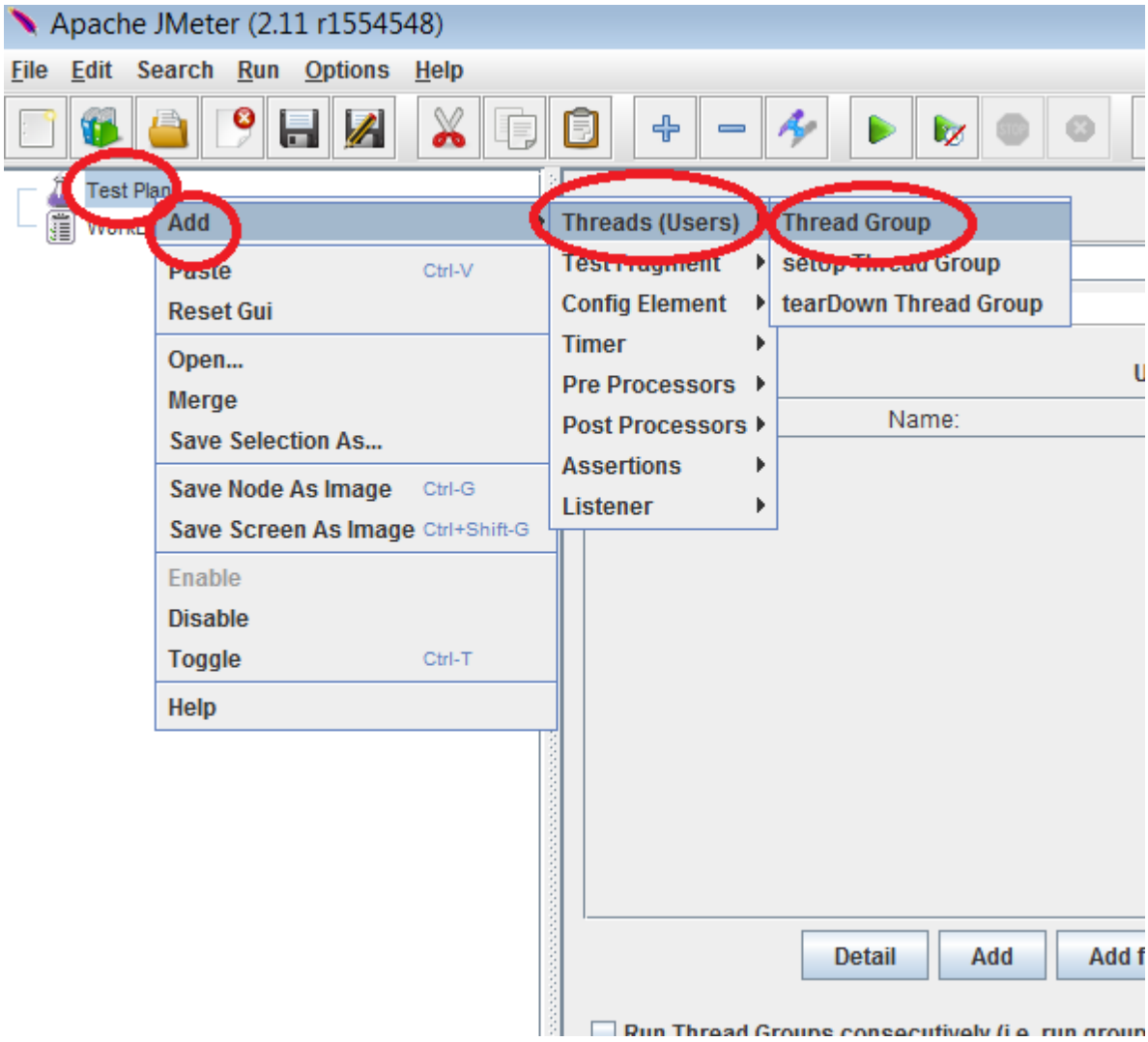
export JAVA_HOME={LAB_HOME}/wlp/java/jre
export PATH=$JAVA_HOME/bin;$PATH

__3.

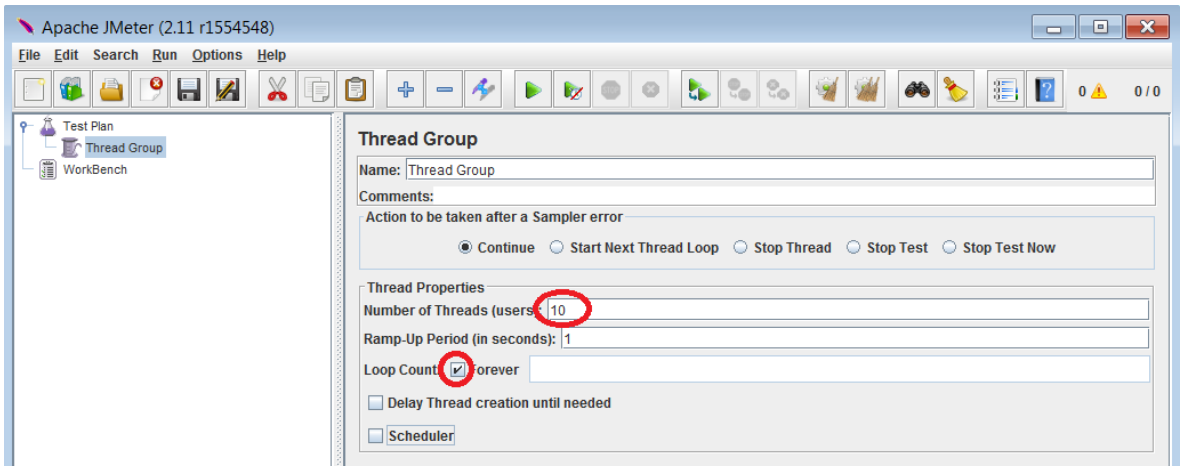
__4. Run the command to start JMeter :

```
{LAB_HOME}\apache-jmeter-<version>\bin\jmeter
```

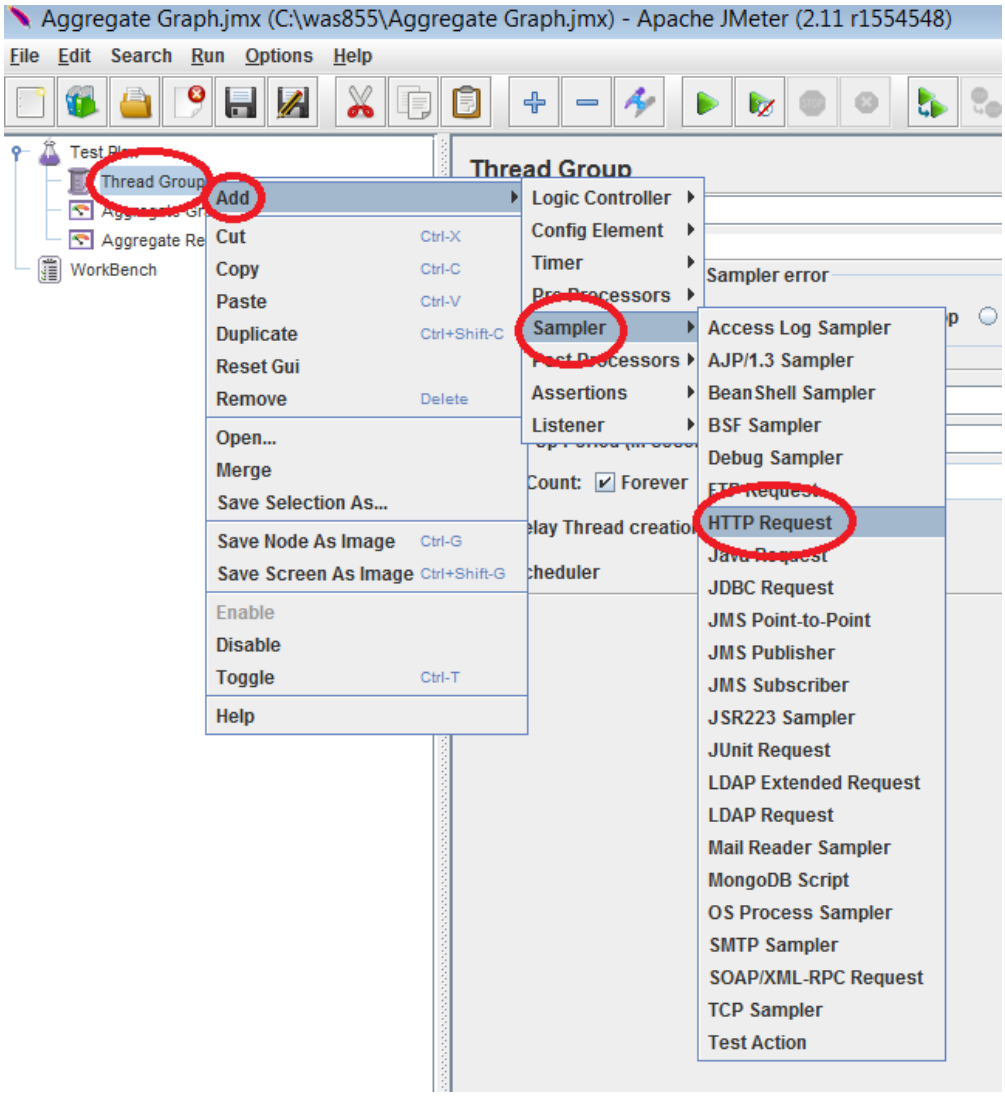
__5. Right click **Test Plan** and navigate **to Add → Threads (User) → Thread**



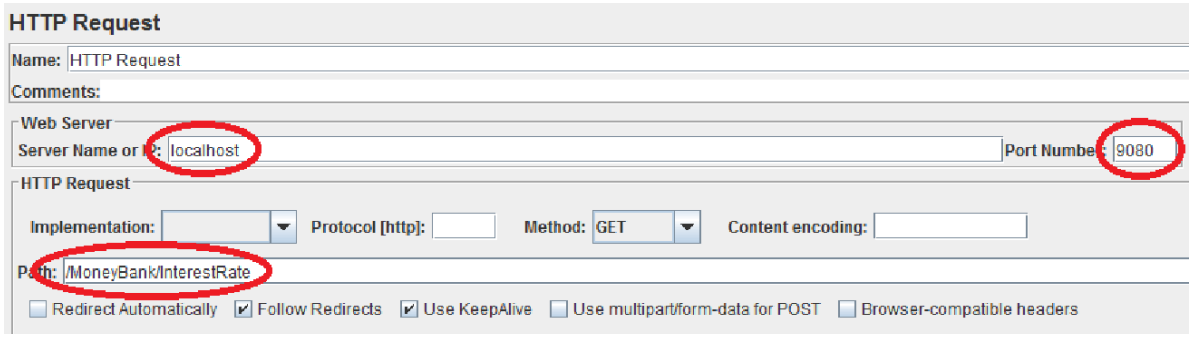
__6. Change number of **threads (users)** to **10** to simulate **10** users, and to **loop forever**



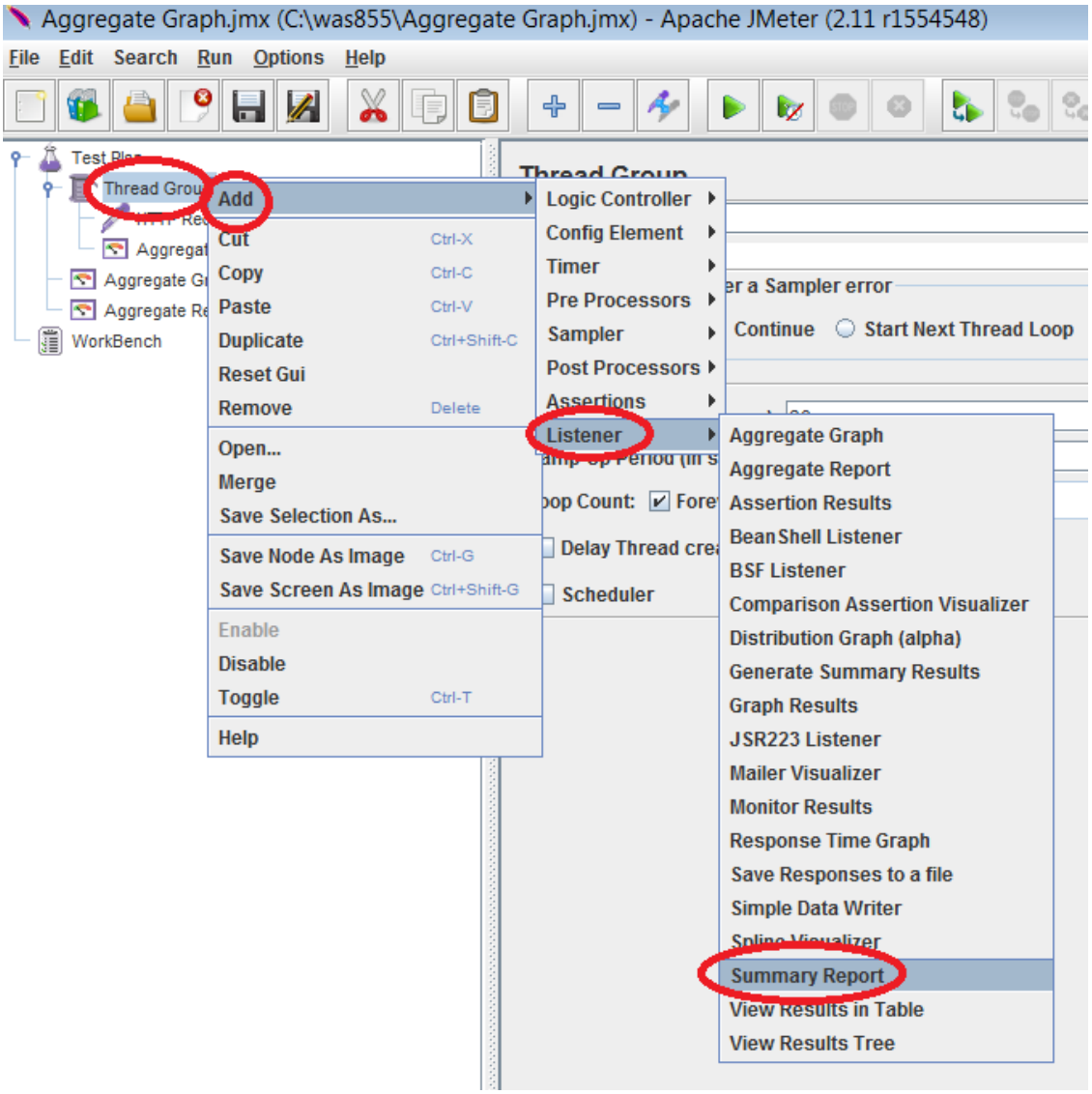
__7. Right click **Thread Group**, navigate to **Add → Sampler → HTTP Request**.



__8. Set the host to **localhost**, port to **9080**, and the path to **/MoneyBank/InterestRate.**

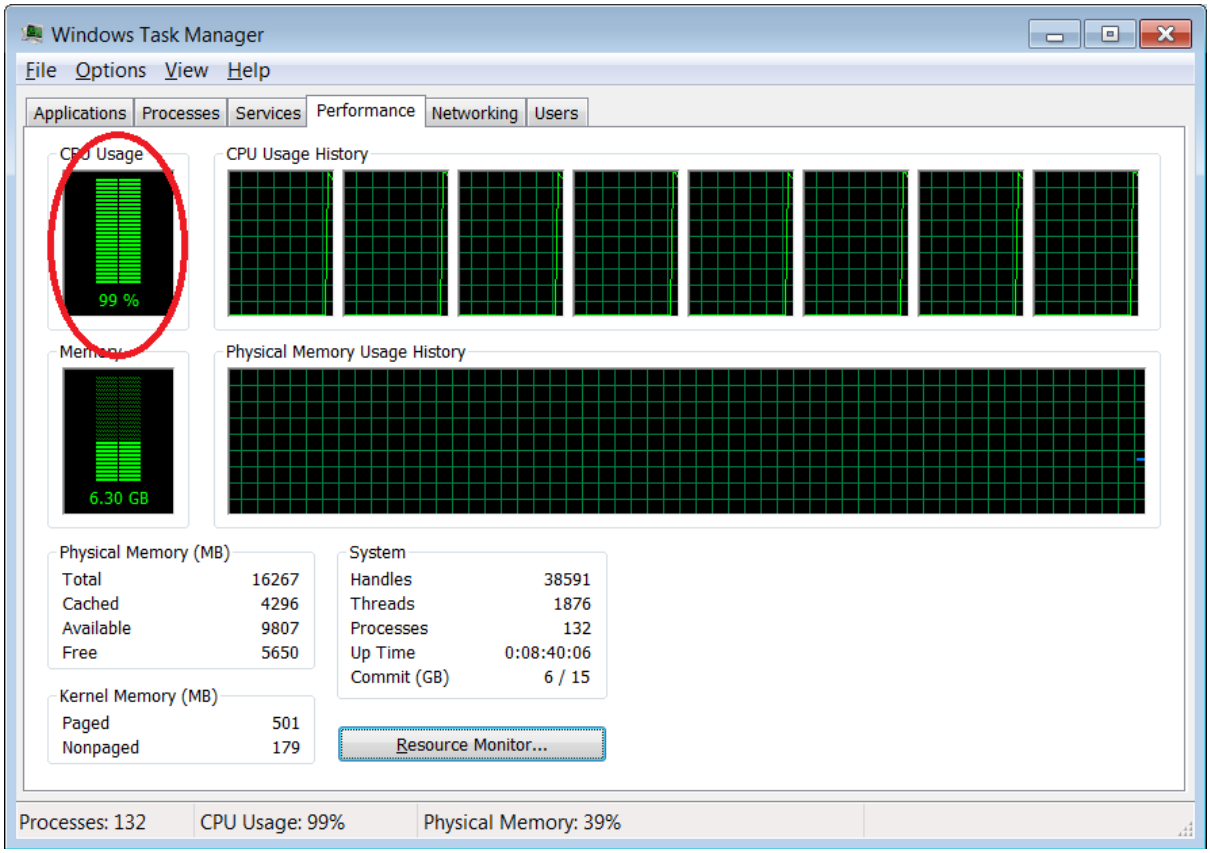


__9. Right click **Thread Group** and navigate to **Add → Listener → Summary Report**



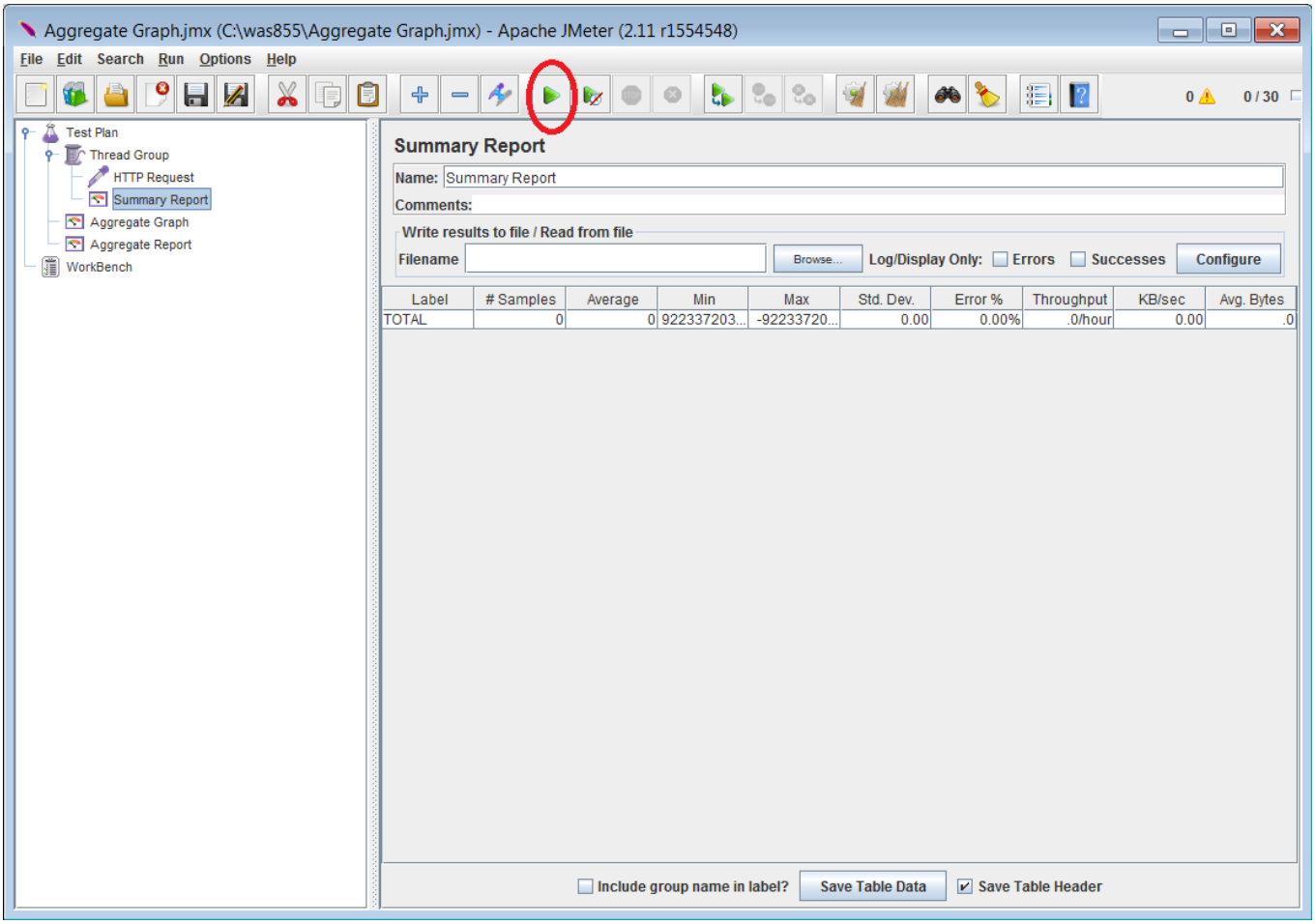
__10. Restart **monitorServer** so we start from a new baseline

__11. Start **Task Manager** on Windows to monitor the CPU to verify whether the application is CPU bound. Use equivalent CPU monitoring tool, like **top**, for your operating system if not using Windows. Below is an example during a sample run :

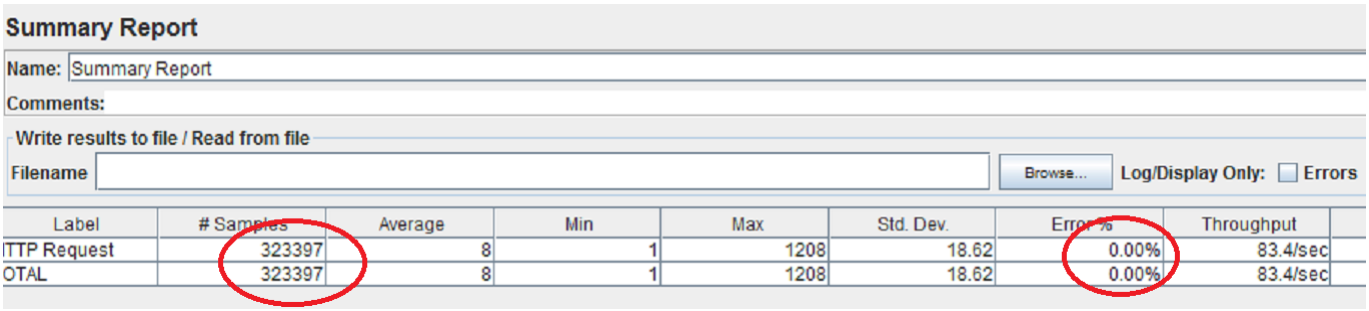


__12. Back to **JMeter**, start the run by clicking on the **Start** button.

__13. Save the changes if prompted.



__14. Verify there are no errors.

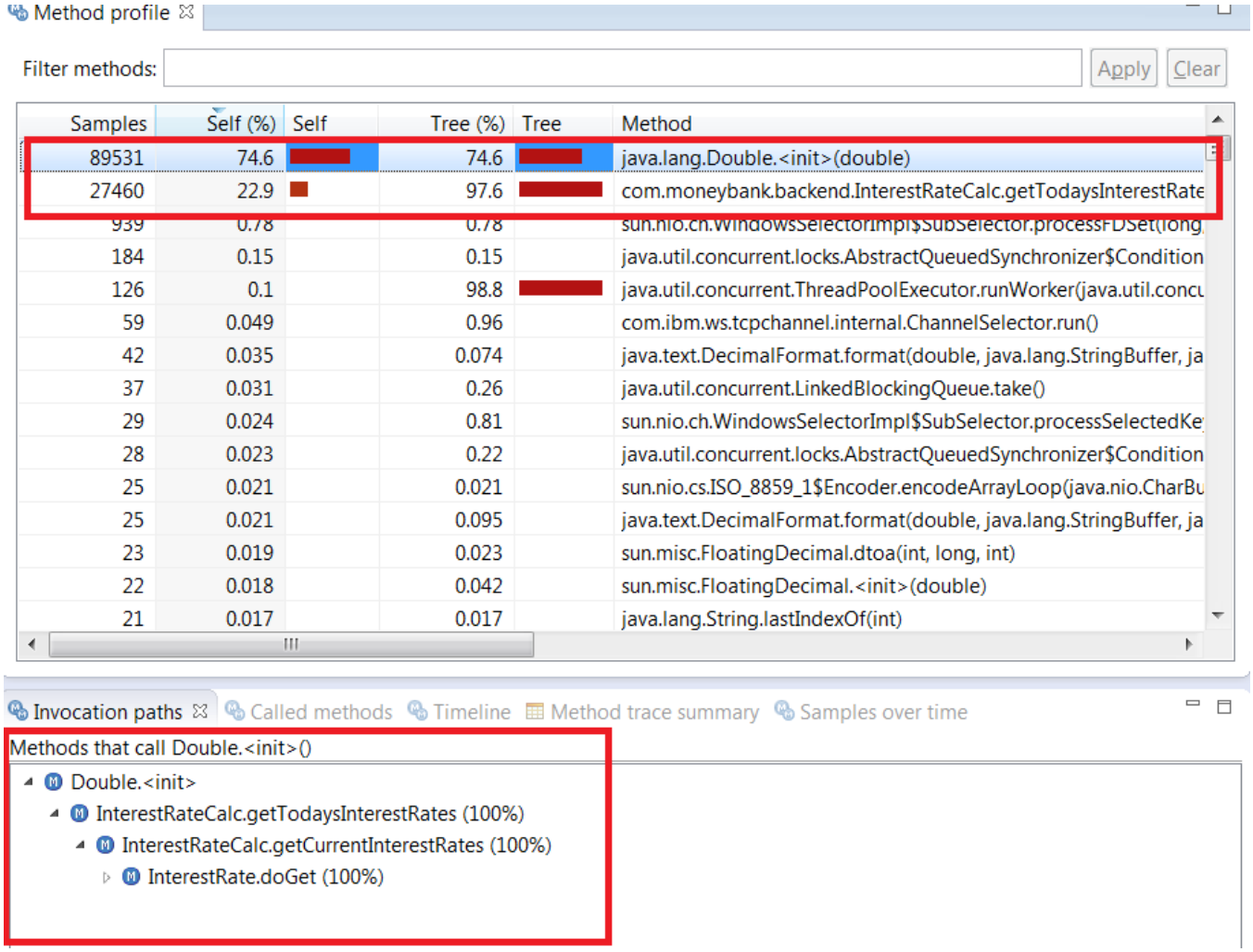


__15. Connect **Health Center** to the **monitorServer**

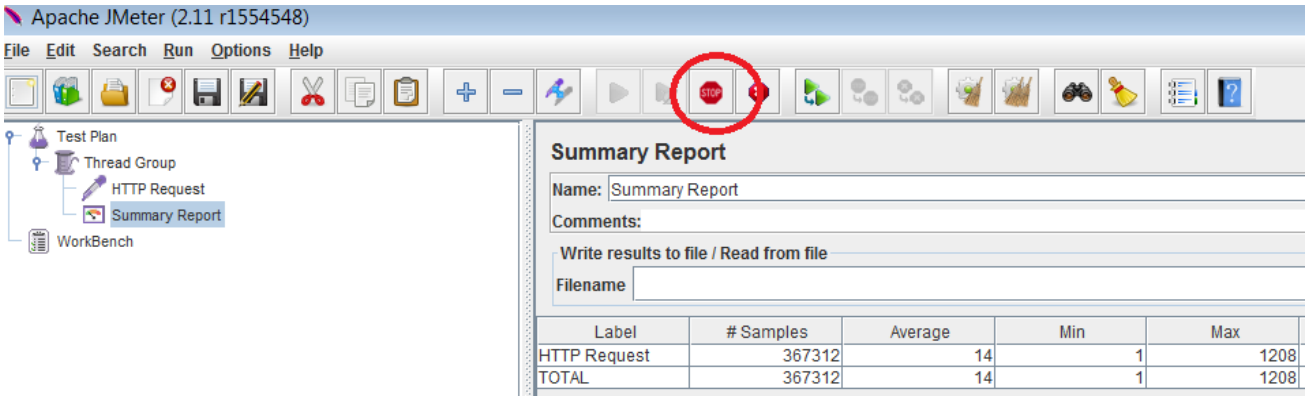
__16. Verify Garbage Collection behavior looks OK

__17. Click on **Profiling**. Note that most of the time is spent in the **Double()** constructor while processing servlet request on **GetInterest** servlet.

__18. **Optional:** Examine the **MoneyBank** application source code to find where the **Double()** constructor is called, and why it's using so much CPU.



__19. Back to JMeter, click **Stop** button.



__20. Stop the Application Server **monitorServer**

1.5 Summary

In this lab you have learned:

- ✓ Enabling **monitoring** feature on .
- ✓ Use **JConsole** to monitor the Liberty Server
- ✓ Use **IBM Monitoring and Diagnostics Tool**s for **Java Health Center** to monitor the Liberty Server
- ✓ Introduce load to the server and debug a response time issue.

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. See Java Guidelines

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Other company, product and service names may be trademarks or service marks of others.

NOTES

NOTES

Please Recycle

WebSphere® software