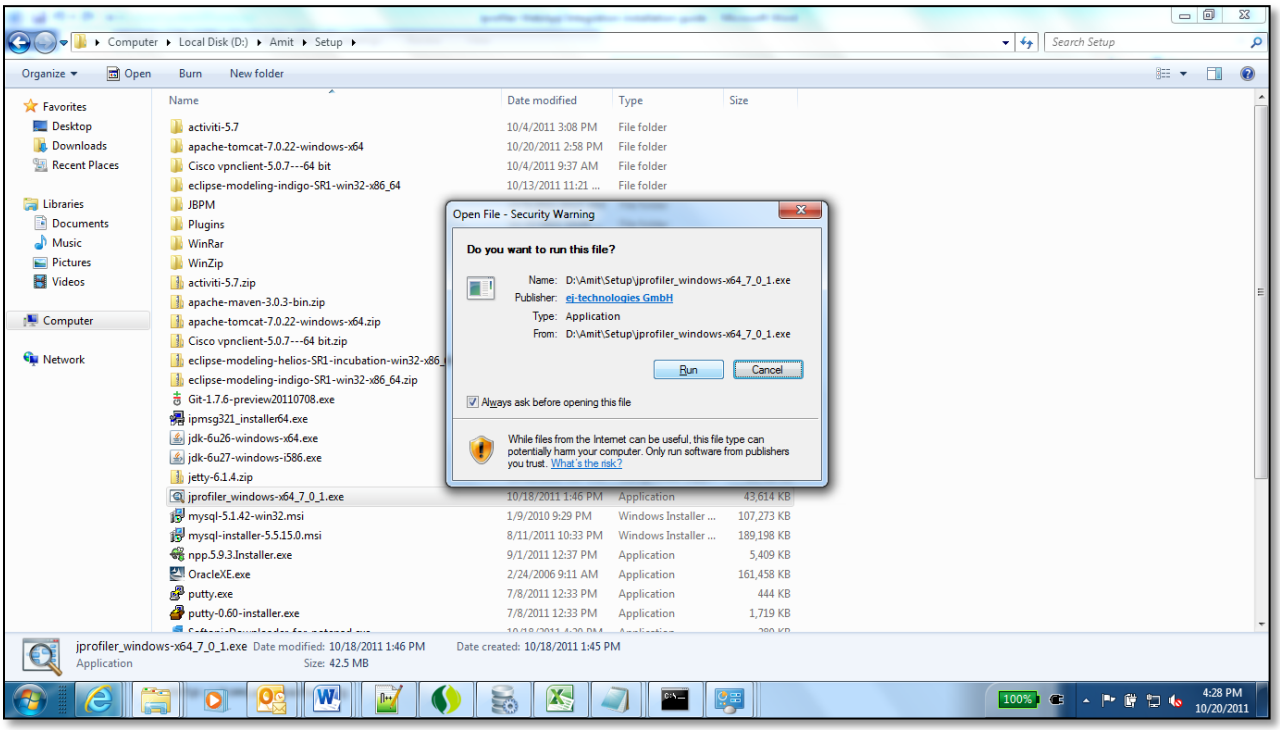
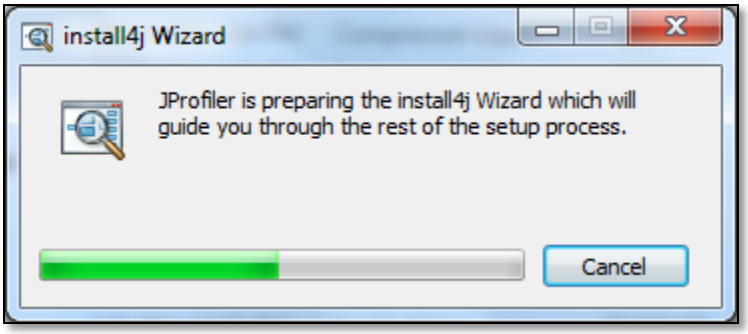
# JPROFILER

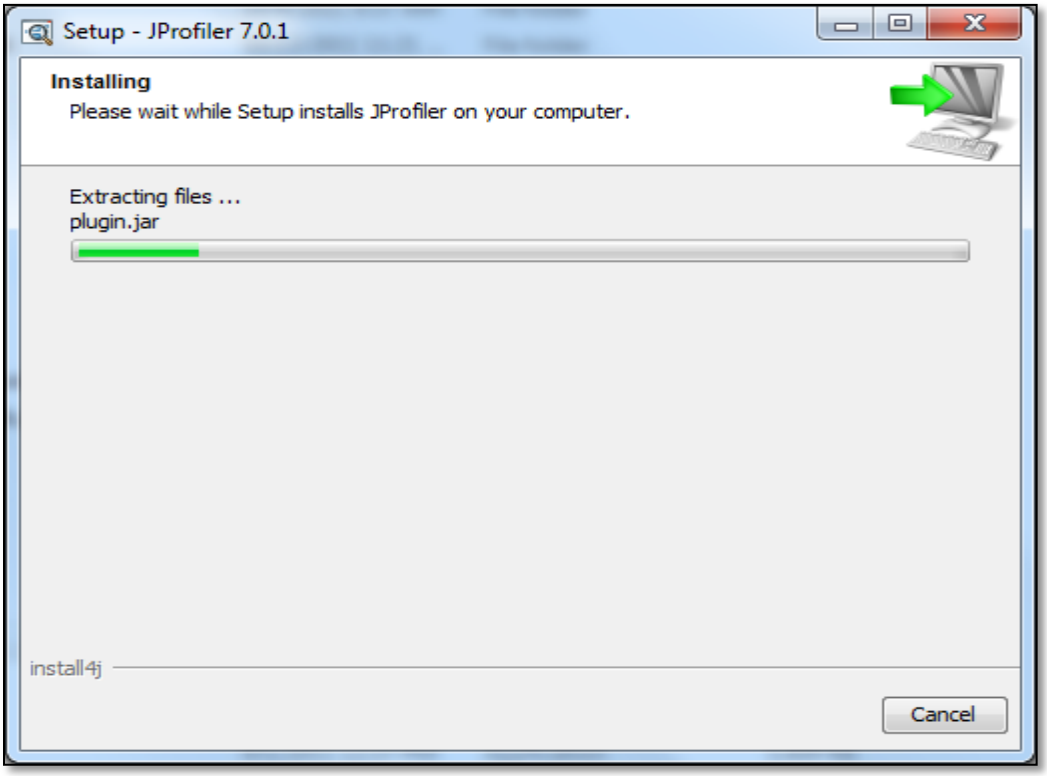
A profiling tool helps a developer to track the performance of his application. JProfiler is considered one of the best profiling tools available. This document gives the details of how to install and configure JProfiler on local as well as remote server. Thereafter, it elaborates on how to profile an application.

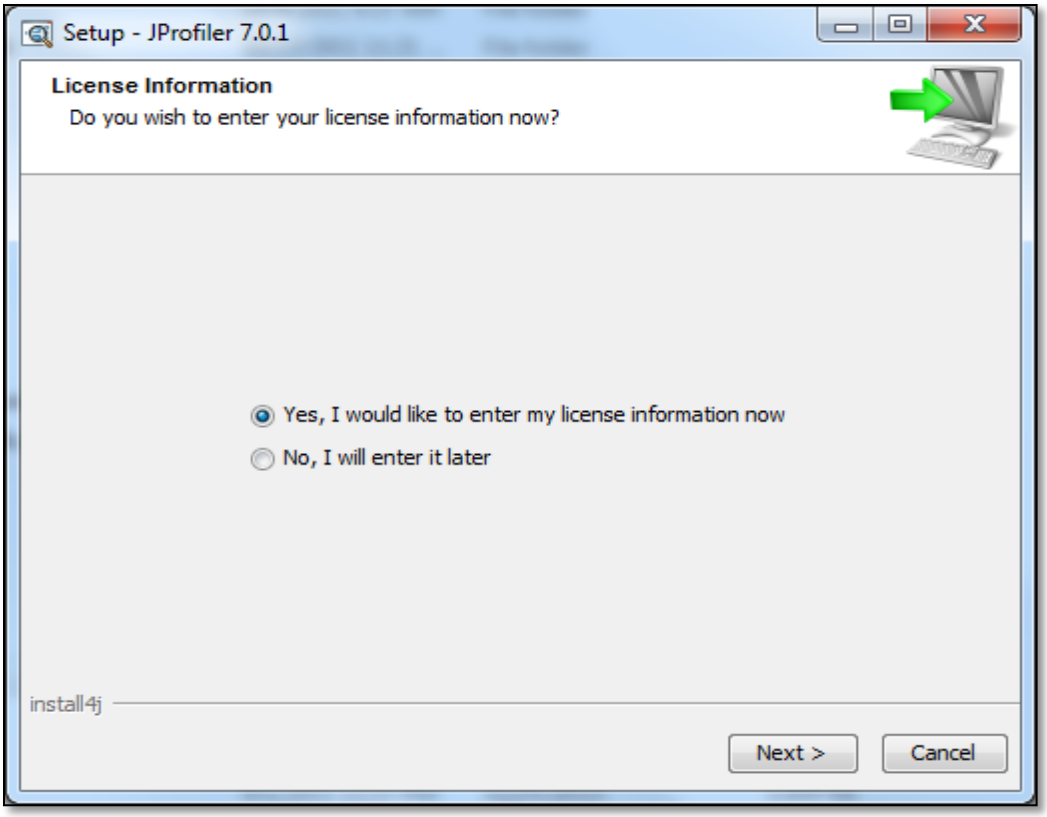
## *Installation*

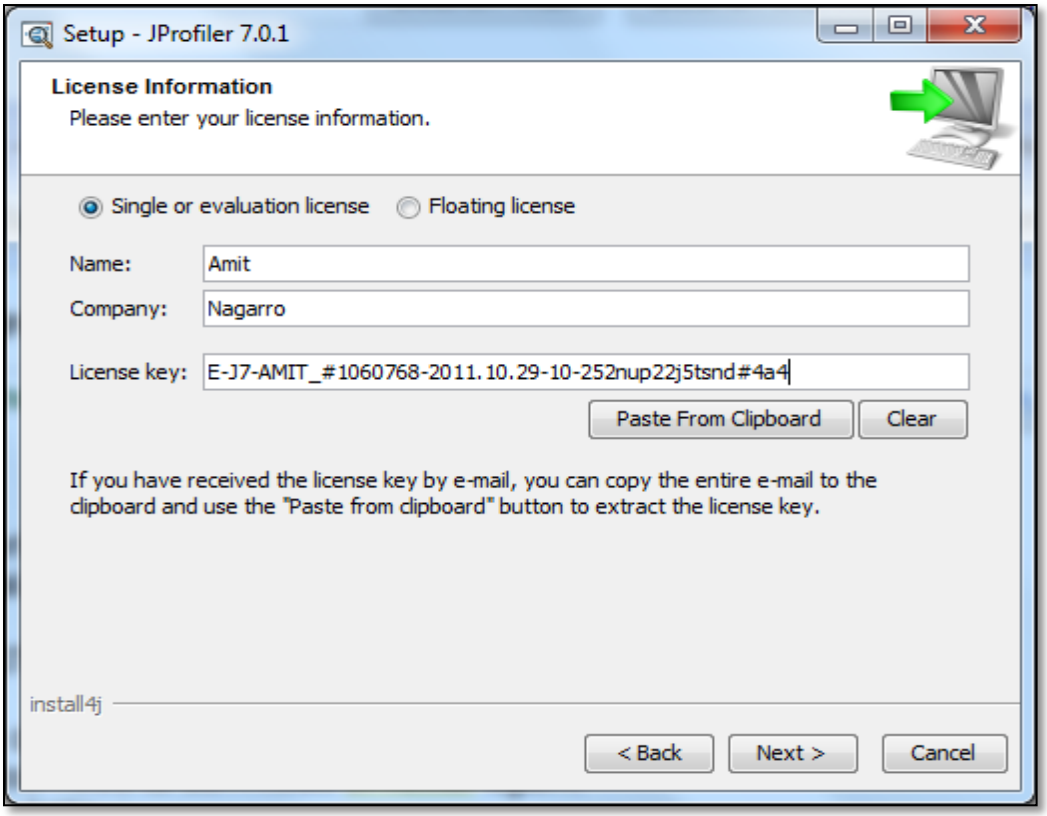



Setup - JProfiler 7.0.1
Welcome to the JProfiler Setup Wizard
This will install JProfiler on your computer. You can either select a quick installation with the default options or a customized installation.
Install into: C:\Program Files\jprofiler7
Customize installation
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel


Setup - JProfiler 7.0.1
Installing
Please wait while Setup installs JProfiler on your computer.
Extracting files ...
plugin.jar
install4j
Cancel

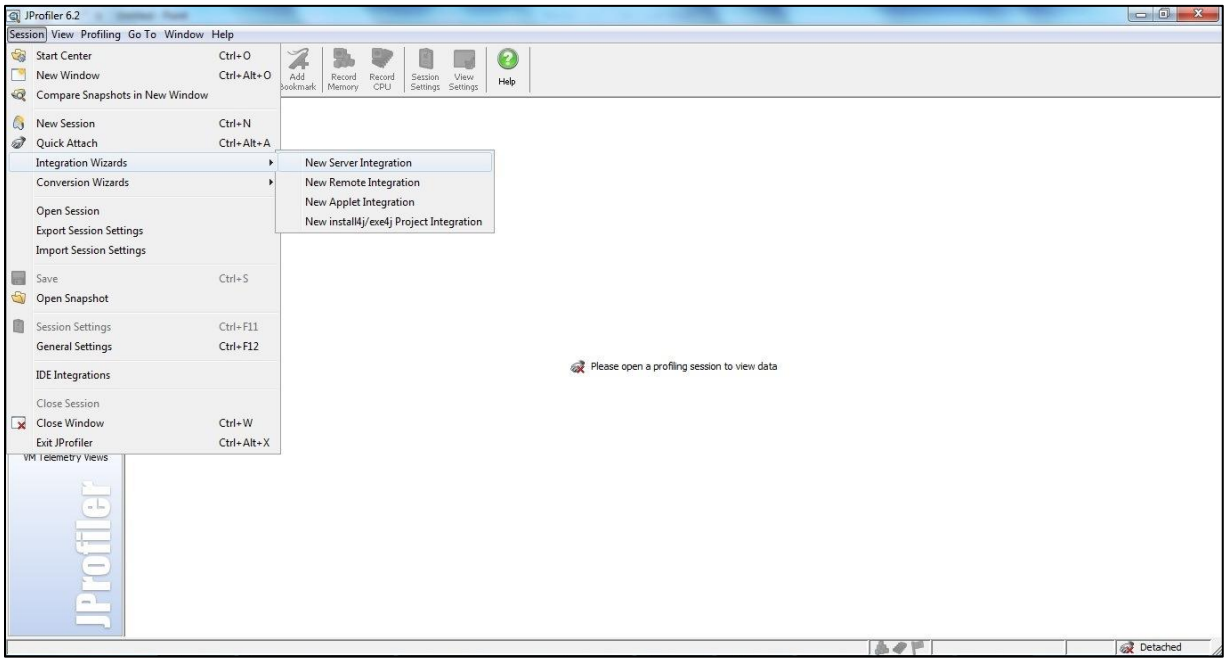

Setup - JProfiler 7.0.1
License Information
Do you wish to enter your license information now?
Yes, I would like to enter my license information now
No, I will enter it later
install4j
Next >
Cancel


Setup - JProfiler 7.0.1
License Information
Please enter your license information.
Single or evaluation license
Floating license
Name:
Amit
Company:
Nagarro
License key:
E-J7-AMIT_#1060768-2011.10.29-10-252nup22j5tsnd#4a4
Paste From Clipboard
Clear
If you have received the license key by e-mail, you can copy the entire e-mail to the clipboard and use the "Paste from clipboard" button to extract the license key.
install4j
< Back
Next >
Cancel

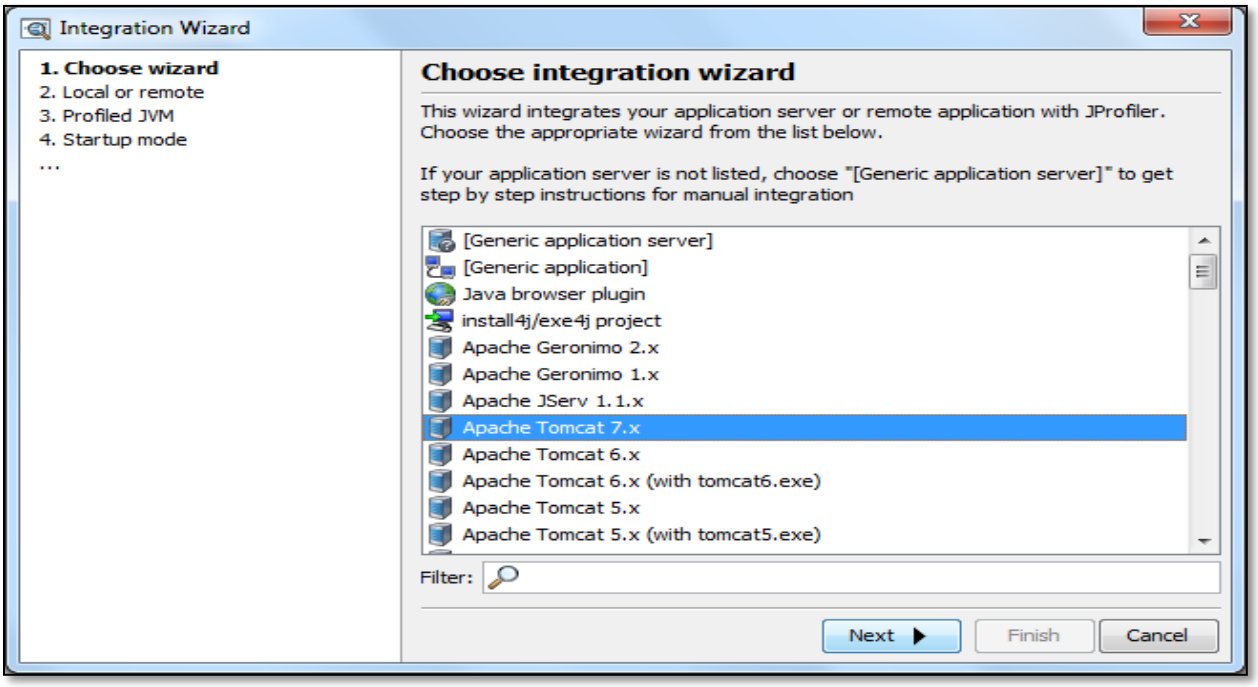
# *Configuration*

## *Web server configuration*

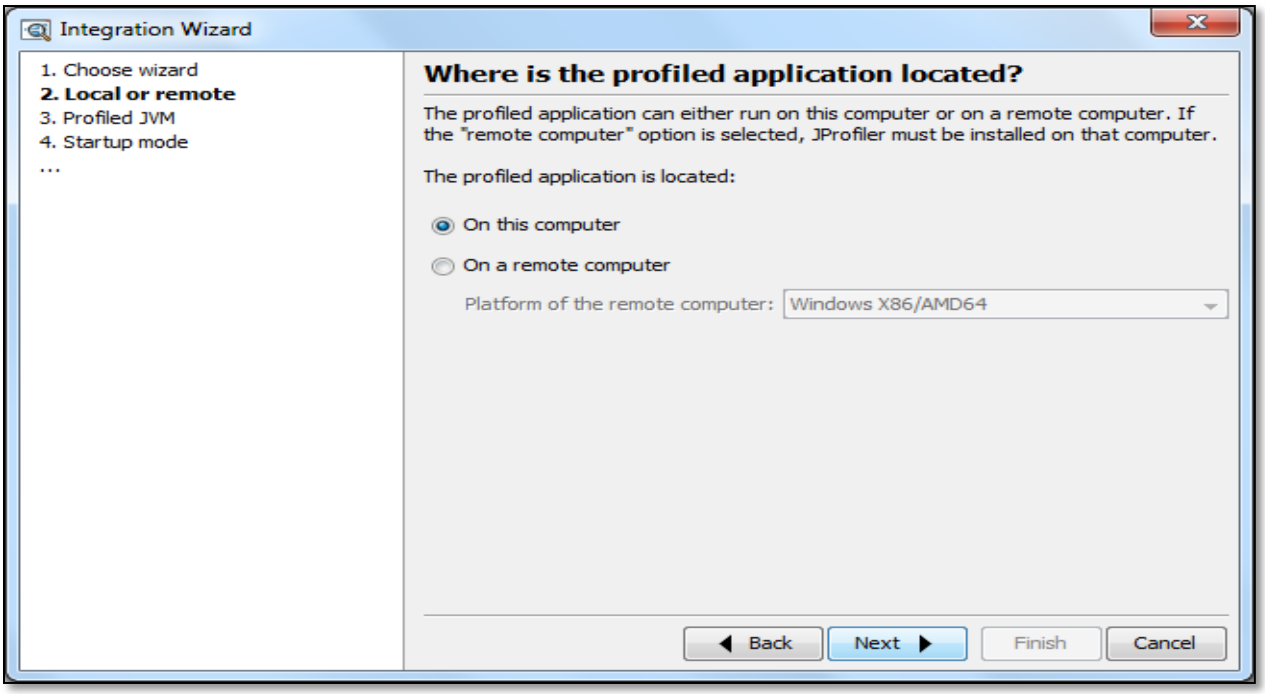
1. Click on **Session** dropdown and select **New Server Integration** from **Integration Wizards**



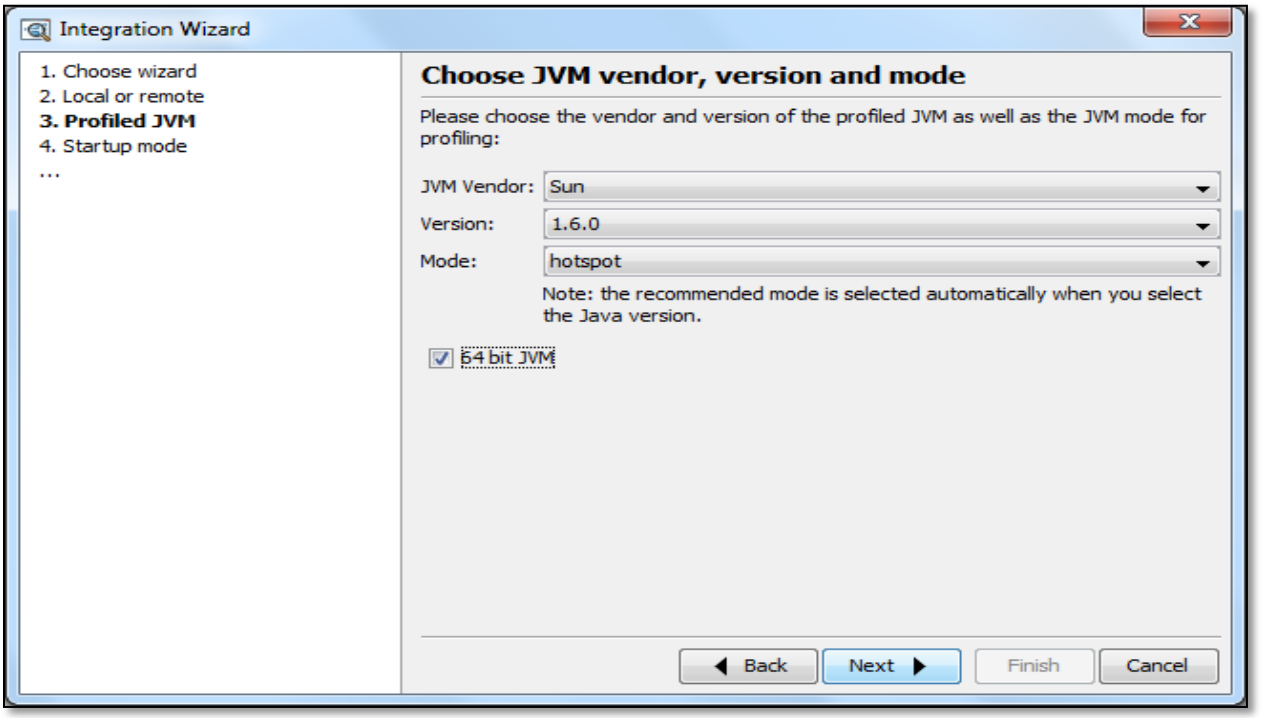

2. Select the server from the listed servers



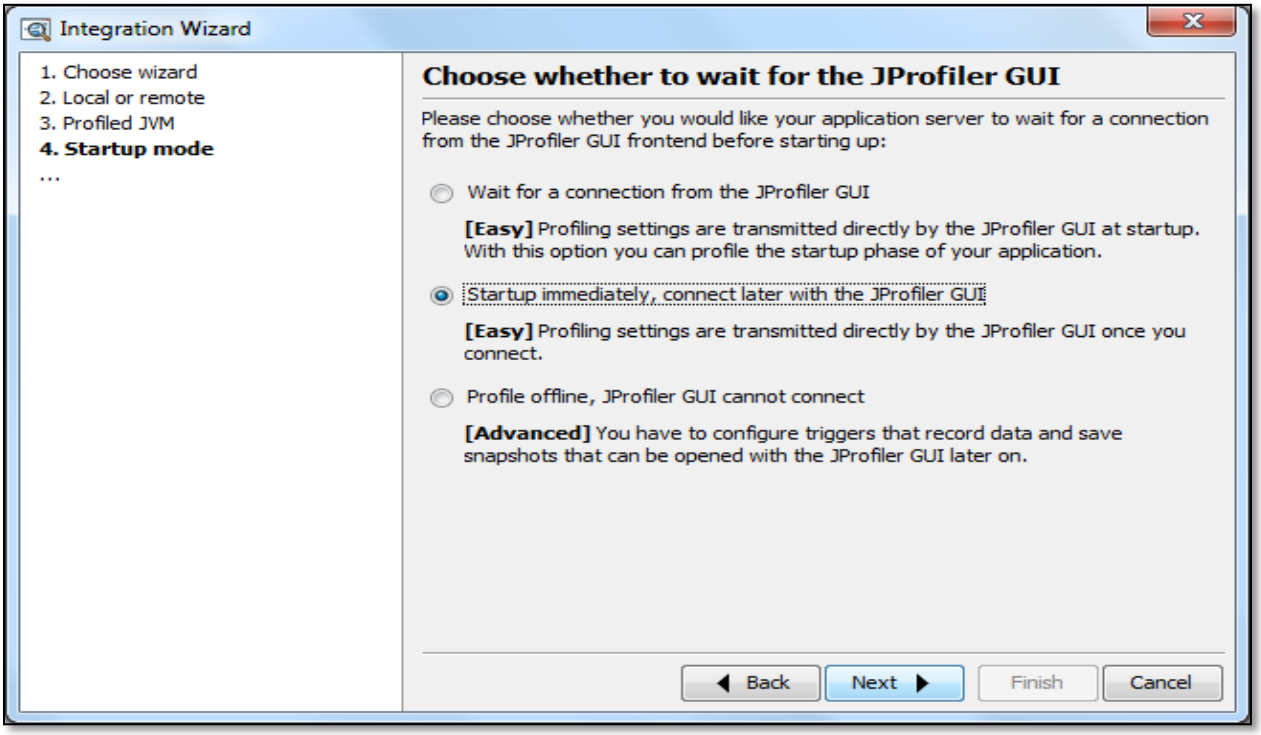
3. Select the system on which the application is running. It can be on the local computer or on a remote computer.



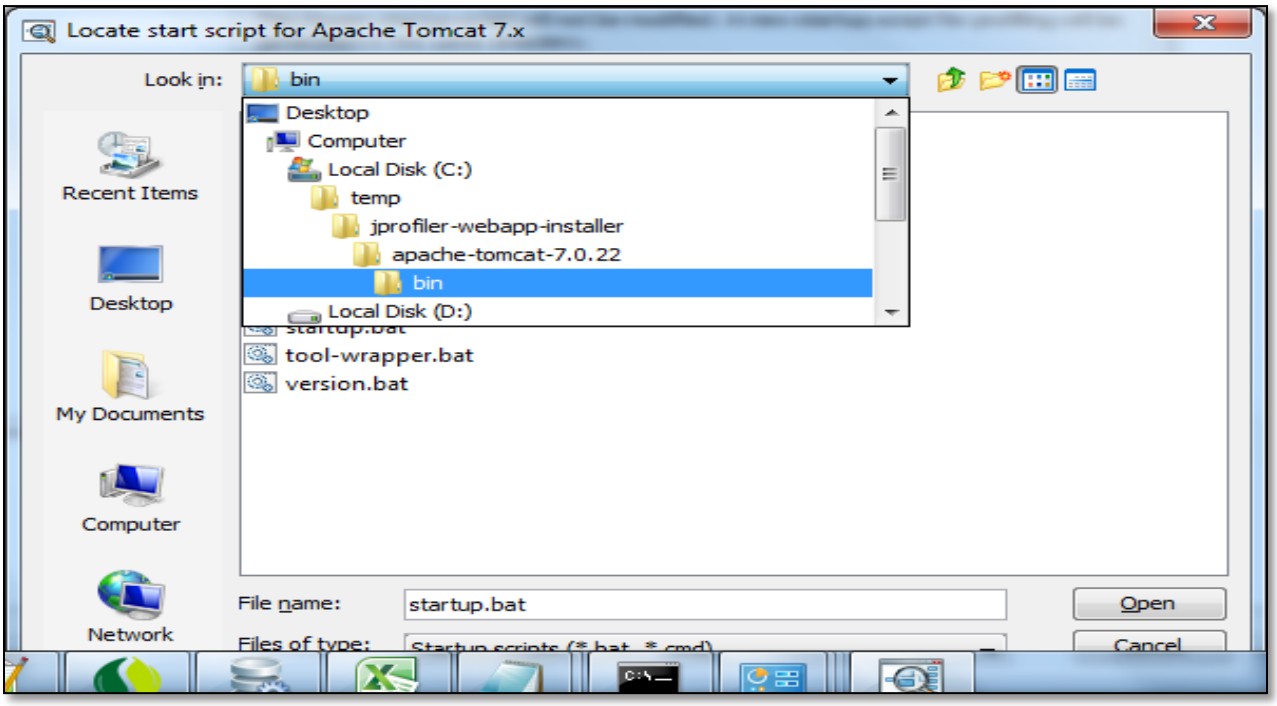

4. Select the **Java Version**

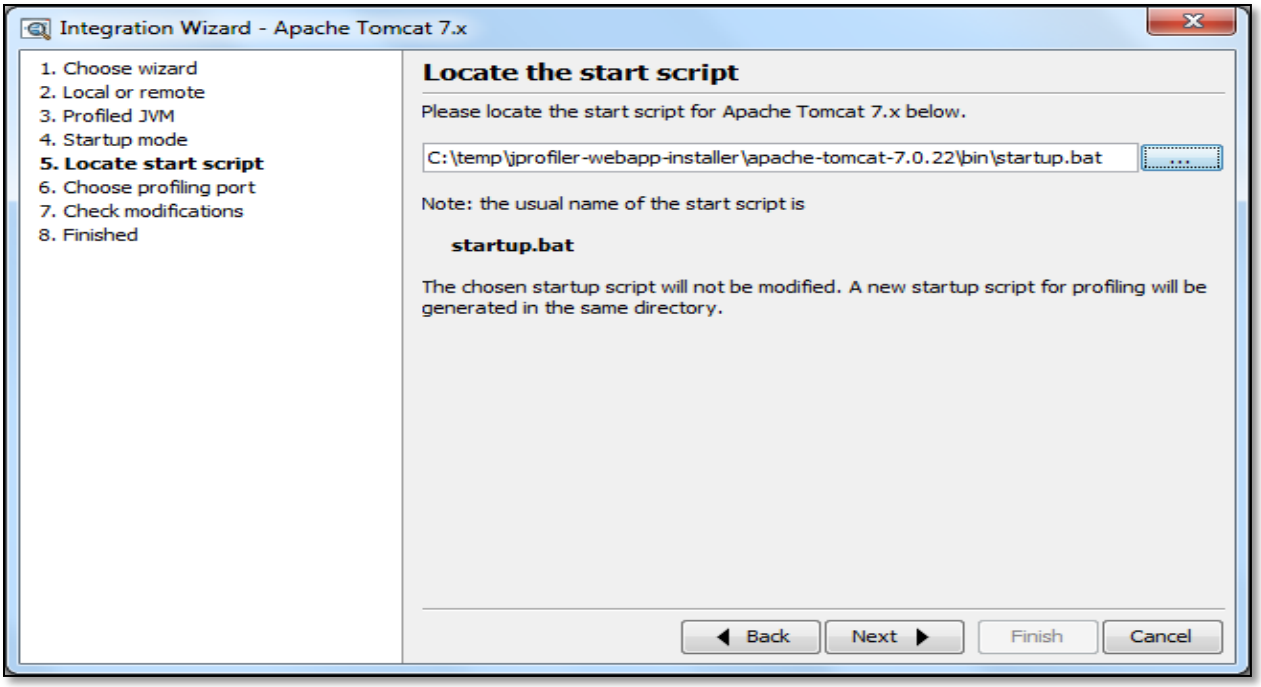


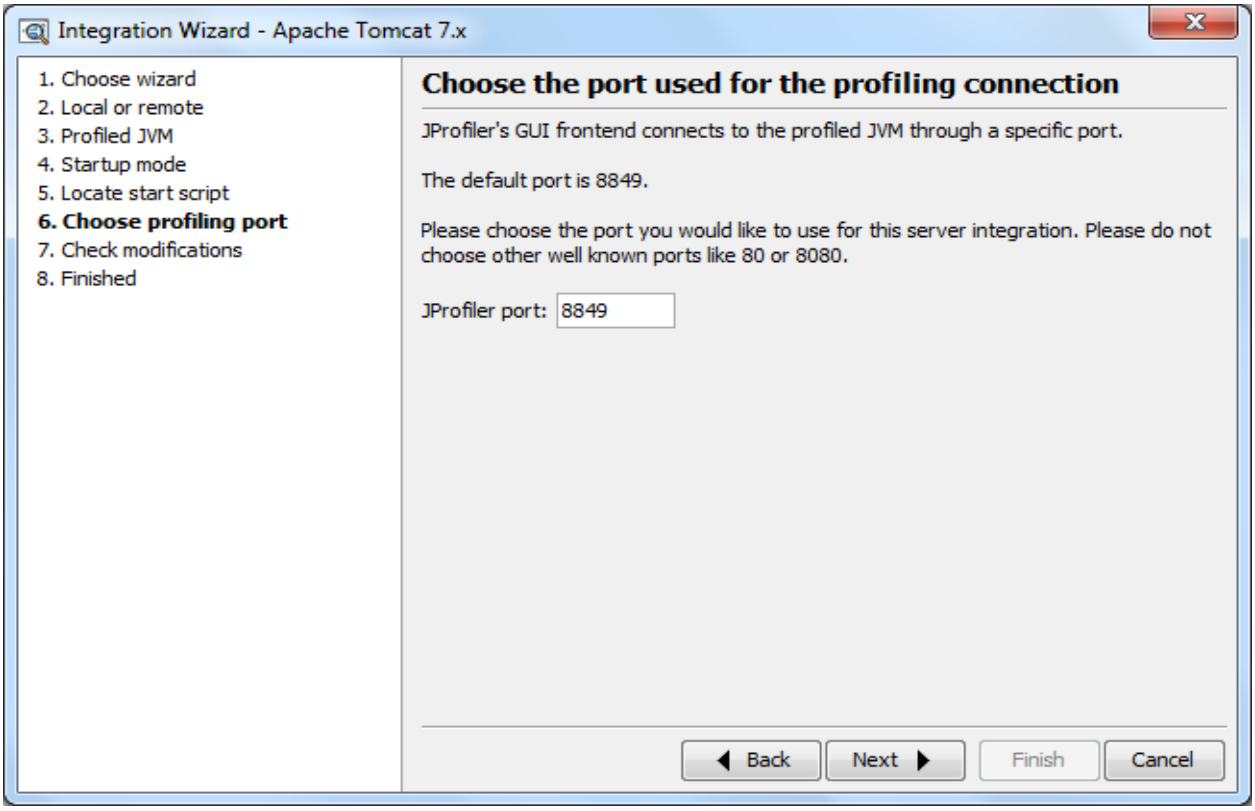
5. Choose the **Startup mode**

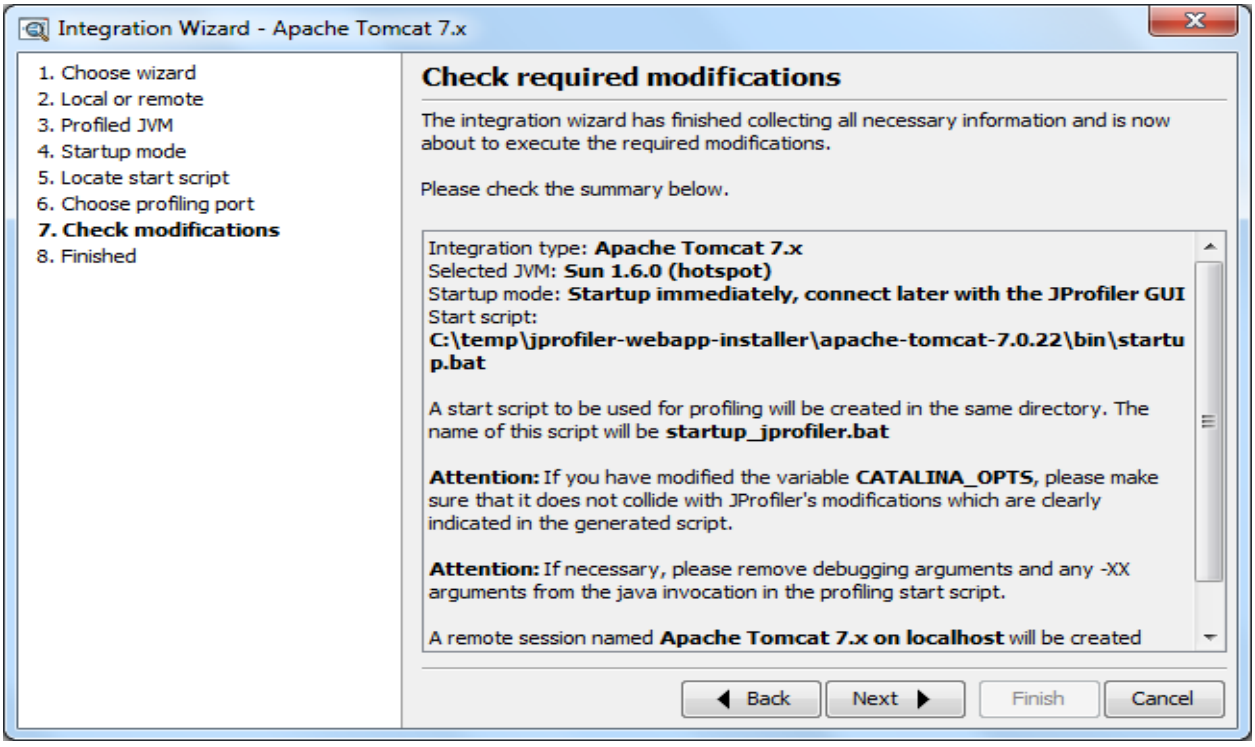


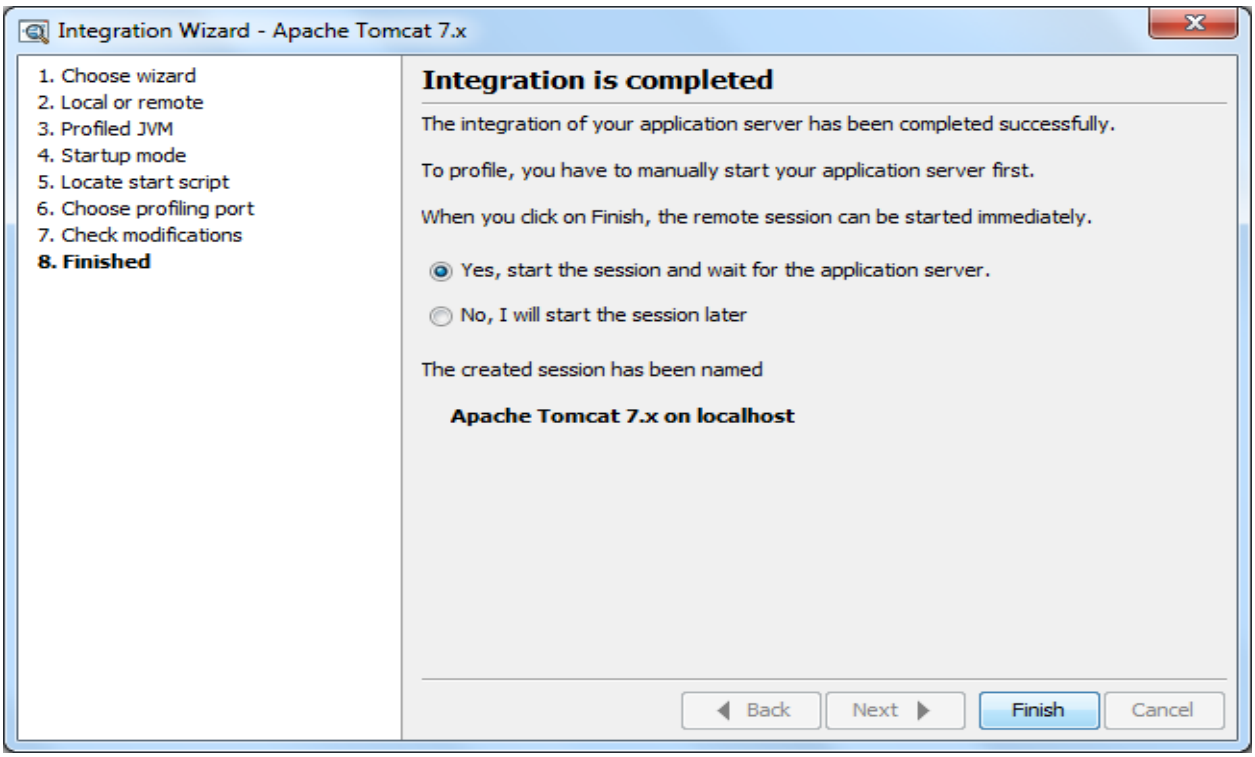

6. We need to configure the startup script. Select the **Startup Script** for the selected server(.bat on windows/.sh on linux)



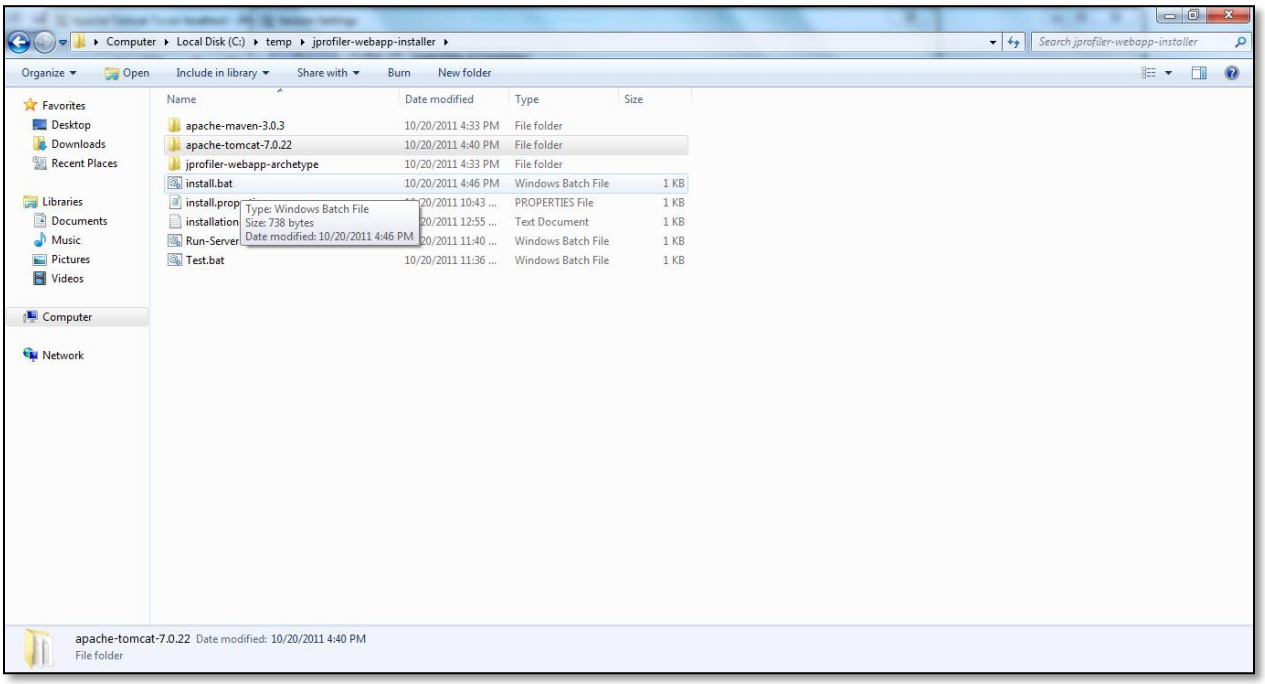

Integration Wizard - Apache Tomcat 7.x
1. Choose wizard
2. Local or remote
3. Profiled JVM
4. Startup mode
5. Locate start script
6. Choose profiling port
7. Check modifications
8. Finished
Locate the start script
Please locate the start script for Apache Tomcat 7.x below.
C:\temp\jprofiler-webapp-installer\apache-tomcat-7.0.22\bin\startup.bat
...
Note: the usual name of the start script is
startup.bat
The chosen startup script will not be modified. A new startup script for profiling will be generated in the same directory.
Back
Next
Finish
Cancel

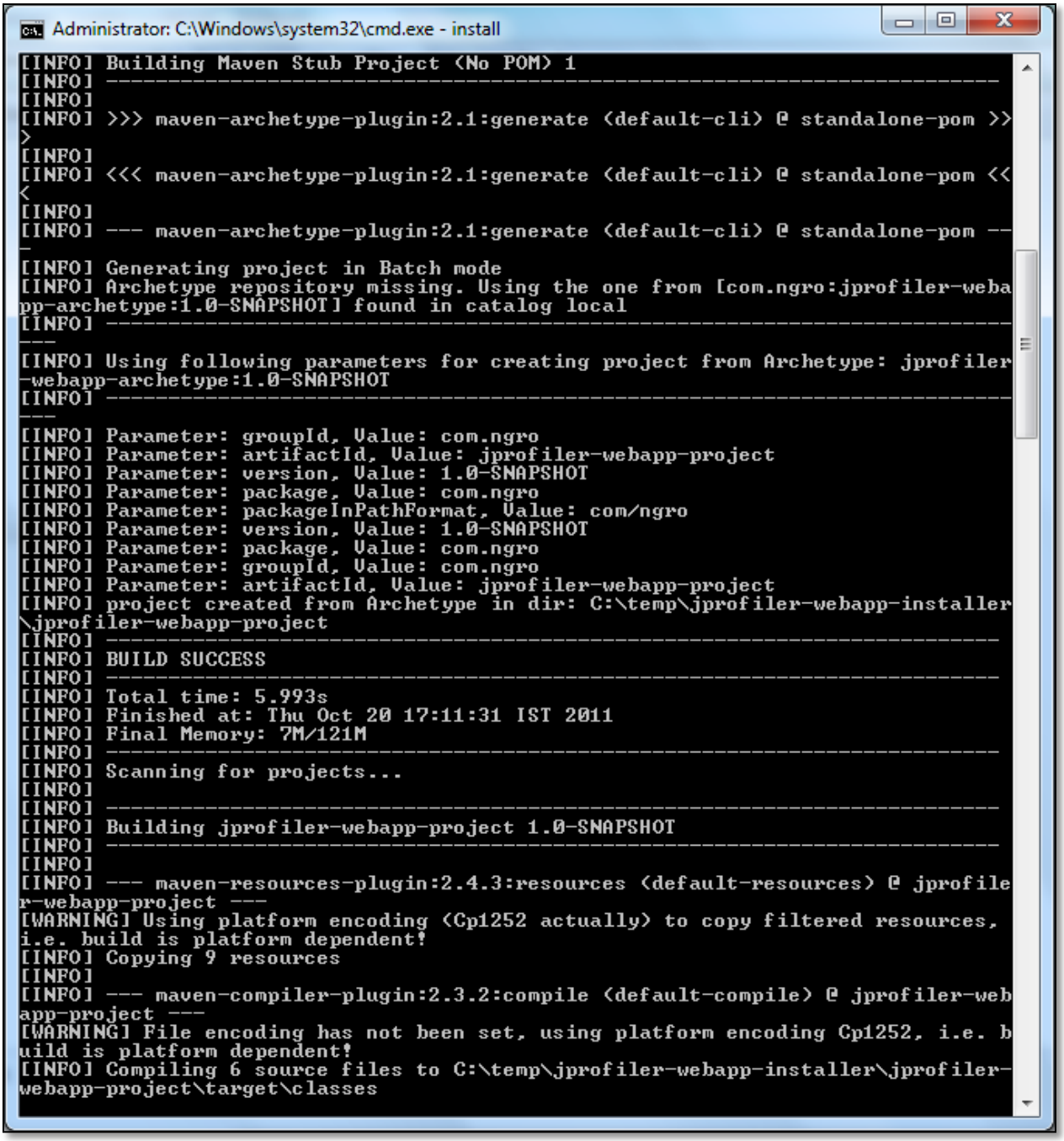

Integration Wizard - Apache Tomcat 7.x
1. Choose wizard
2. Local or remote
3. Profiled JVM
4. Startup mode
5. Locate start script
6. Choose profiling port
7. Check modifications
8. Finished
Choose the port used for the profiling connection
JProfiler's GUI frontend connects to the profiled JVM through a specific port.
The default port is 8849.
Please choose the port you would like to use for this server integration. Please do not choose other well known ports like 80 or 8080.
JProfiler port: 8849
Back
Next
Finish
Cancel

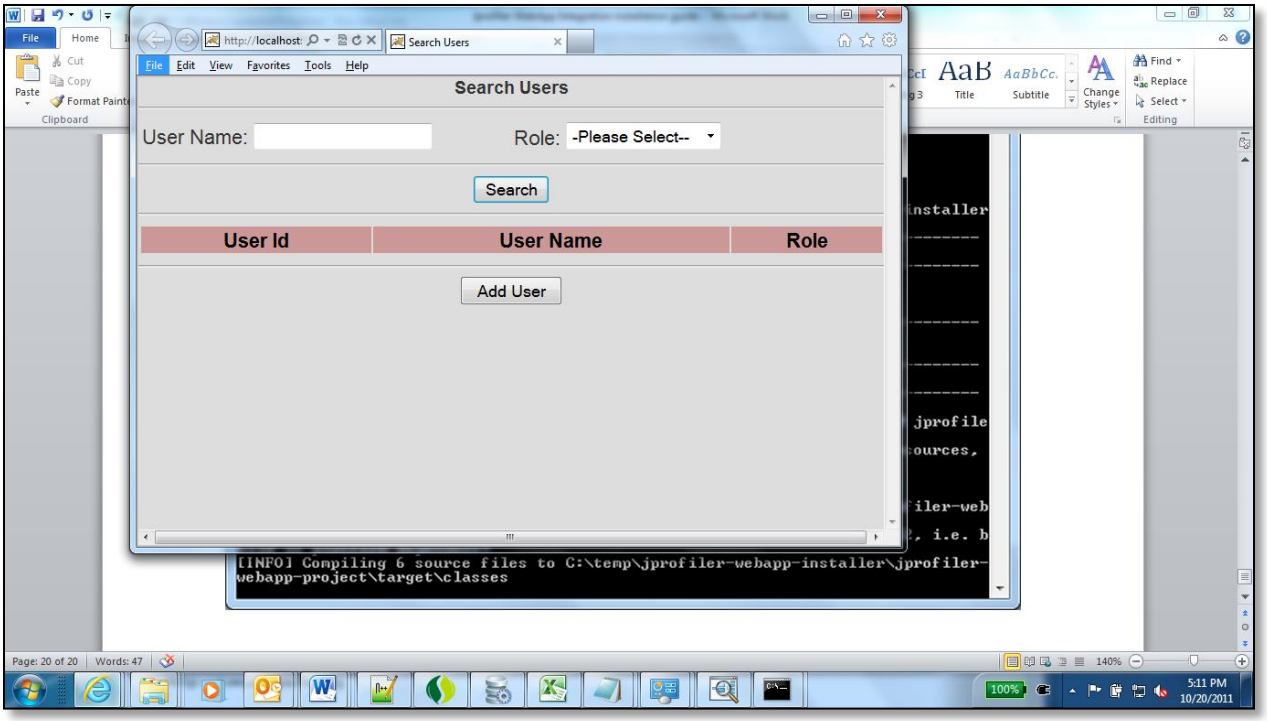

Integration Wizard - Apache Tomcat 7.x
1. Choose wizard
2. Local or remote
3. Profiled JVM
4. Startup mode
5. Locate start script
6. Choose profiling port
7. Check modifications
8. Finished
Check required modifications
The integration wizard has finished collecting all necessary information and is now about to execute the required modifications.
Please check the summary below.
Integration type: Apache Tomcat 7.x
Selected JVM: Sun 1.6.0 (hotspot)
Startup mode: Startup immediately, connect later with the JProfiler GUI
Start script:
C:\temp\jprofiler-webapp-installer\apache-tomcat-7.0.22\bin\startup.bat
A start script to be used for profiling will be created in the same directory. The name of this script will be startup_jprofiler.bat
Attention: If you have modified the variable CATALINA_OPTS, please make sure that it does not collide with JProfiler's modifications which are clearly indicated in the generated script.
Attention: If necessary, please remove debugging arguments and any -XX arguments from the java invocation in the profiling start script.
A remote session named Apache Tomcat 7.x on localhost will be created
Back
Next
Finish
Cancel


Integration Wizard - Apache Tomcat 7.x
1. Choose wizard
2. Local or remote
3. Profiled JVM
4. Startup mode
5. Locate start script
6. Choose profiling port
7. Check modifications
8. Finished
Integration is completed
The integration of your application server has been completed successfully.
To profile, you have to manually start your application server first.
When you click on Finish, the remote session can be started immediately.
Yes, start the session and wait for the application server.
No, I will start the session later
The created session has been named
Apache Tomcat 7.x on localhost
Back
Next
Finish
Cancel

Double click on install.bat inside folder jprofiler-webapp-installer

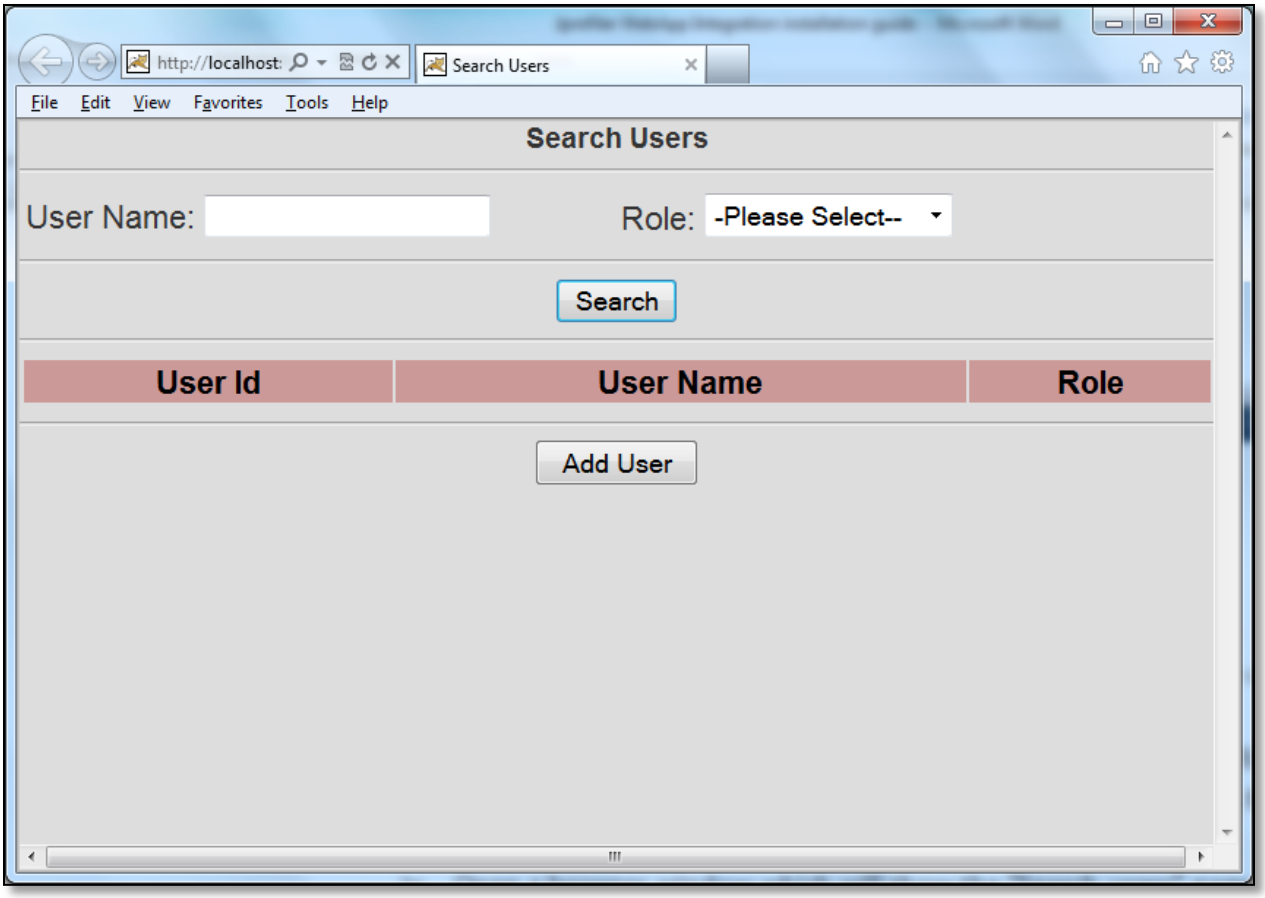



Administrator: C:\Windows\system32\cmd.exe - install
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.1:generate (default-cli) @ standalone-pom >>
>
[INFO]
[INFO] <<< maven-archetype-plugin:2.1:generate (default-cli) @ standalone-pom <<
<
[INFO]
[INFO] --- maven-archetype-plugin:2.1:generate (default-cli) @ standalone-pom --
-
[INFO] Generating project in Batch mode
[INFO] Archetype repository missing. Using the one from [com.ngro:jprofiler-weba
pp-archetype:1.0-SNAPSHOT] found in catalog local
[INFO] ------------------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Archetype: jprofiler
-webapp-archetype:1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
---
[INFO] Parameter: groupId, Value: com.ngro
[INFO] Parameter: artifactId, Value: jprofiler-webapp-project
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.ngro
[INFO] Parameter: packageInPathFormat, Value: com/ngro
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.ngro
[INFO] Parameter: groupId, Value: com.ngro
[INFO] Parameter: artifactId, Value: jprofiler-webapp-project
[INFO] project created from Archetype in dir: C:\temp\jprofiler-webapp-installer
\jprofiler-webapp-project
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.993s
[INFO] Finished at: Thu Oct 20 17:11:31 IST 2011
[INFO] Final Memory: 7M/121M
[INFO] ------------------------------------------------------------------------
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building jprofiler-webapp-project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.4.3:resources (default-resources) @ jprofile
r-webapp-project ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 9 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ jprofiler-web
app-project ---
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. b
uild is platform dependent!
[INFO] Compiling 6 source files to C:\temp\jprofiler-webapp-installer\jprofiler-
webapp-project\target\classes

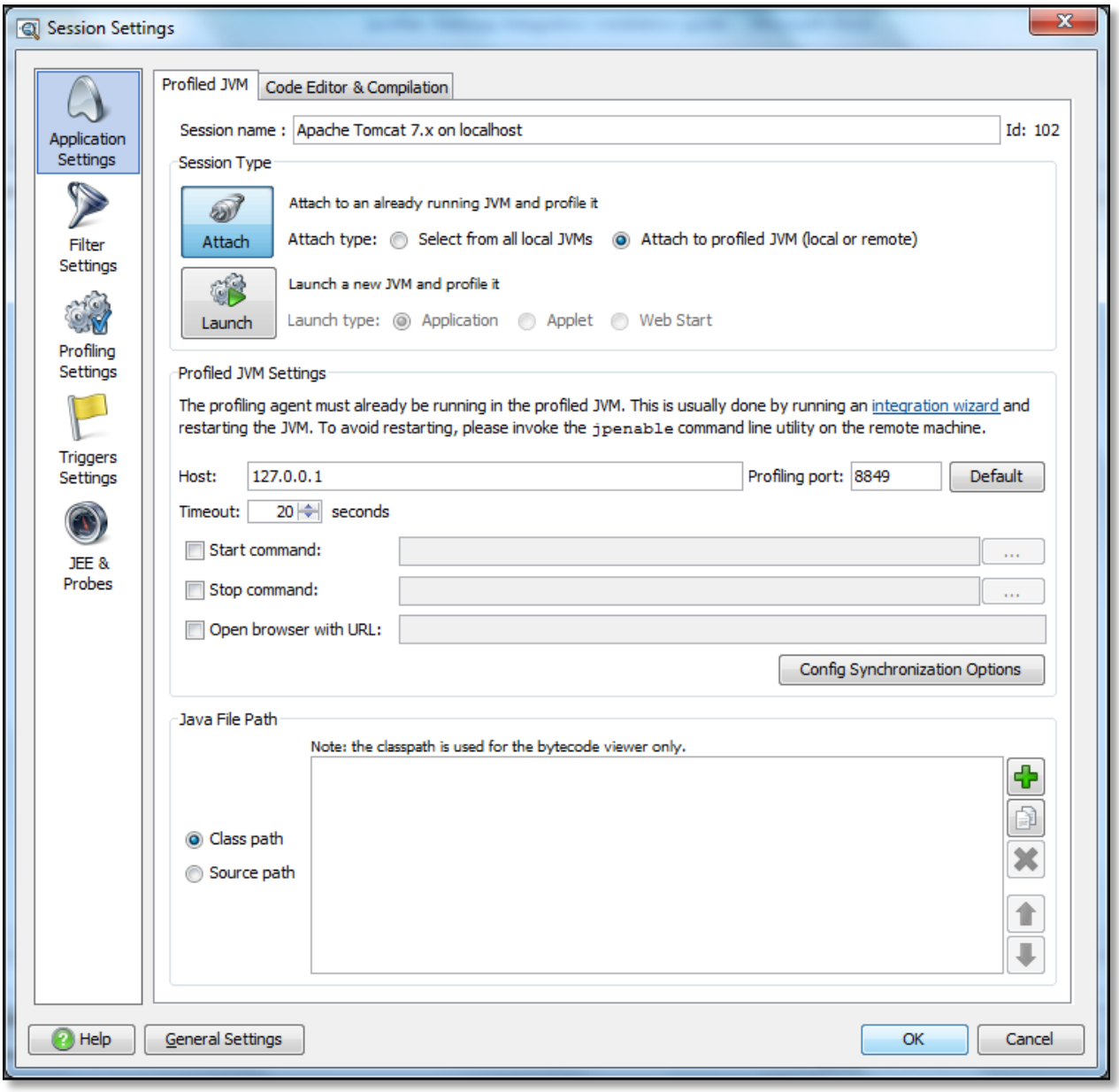


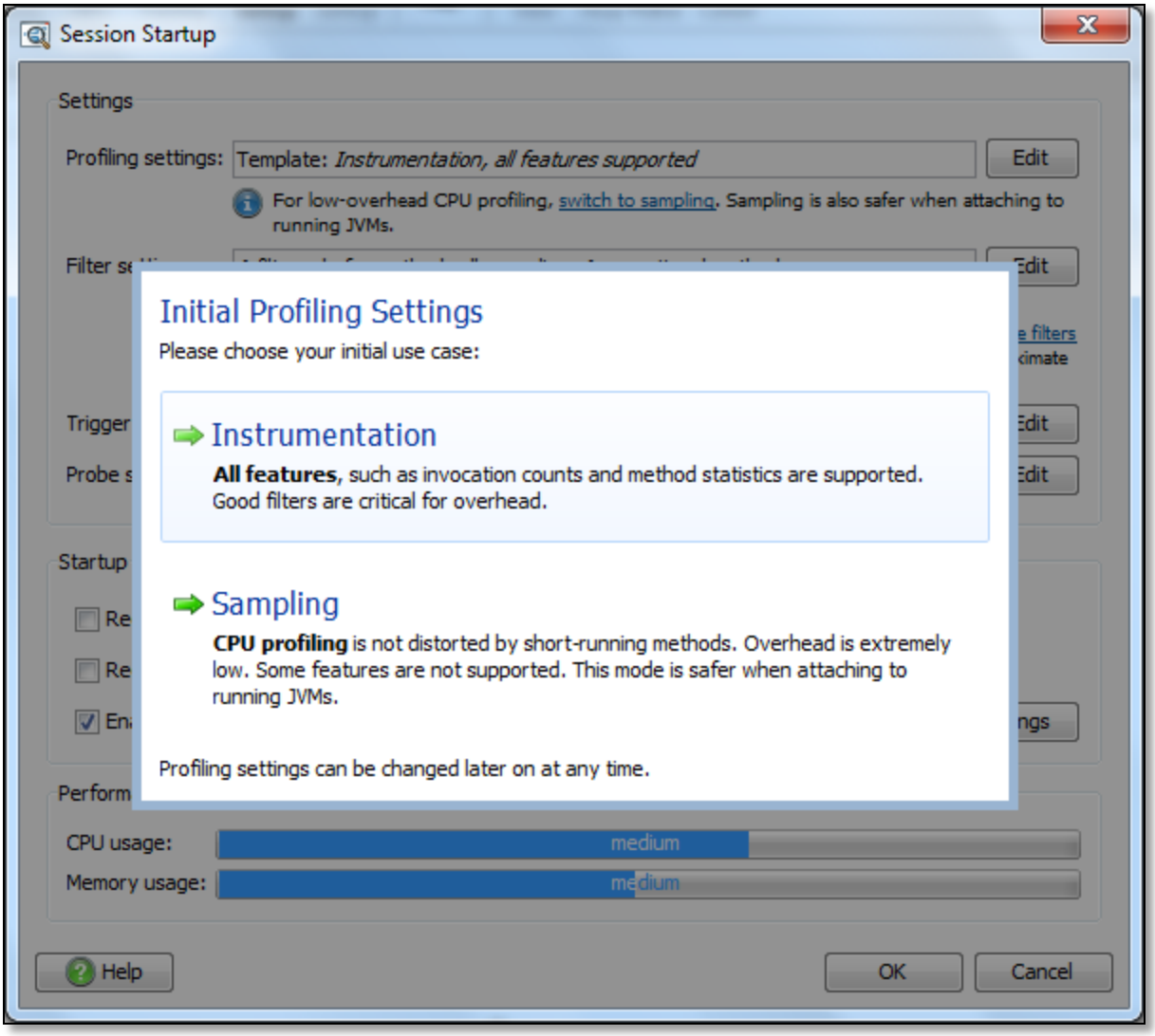

This step will perform the following steps:

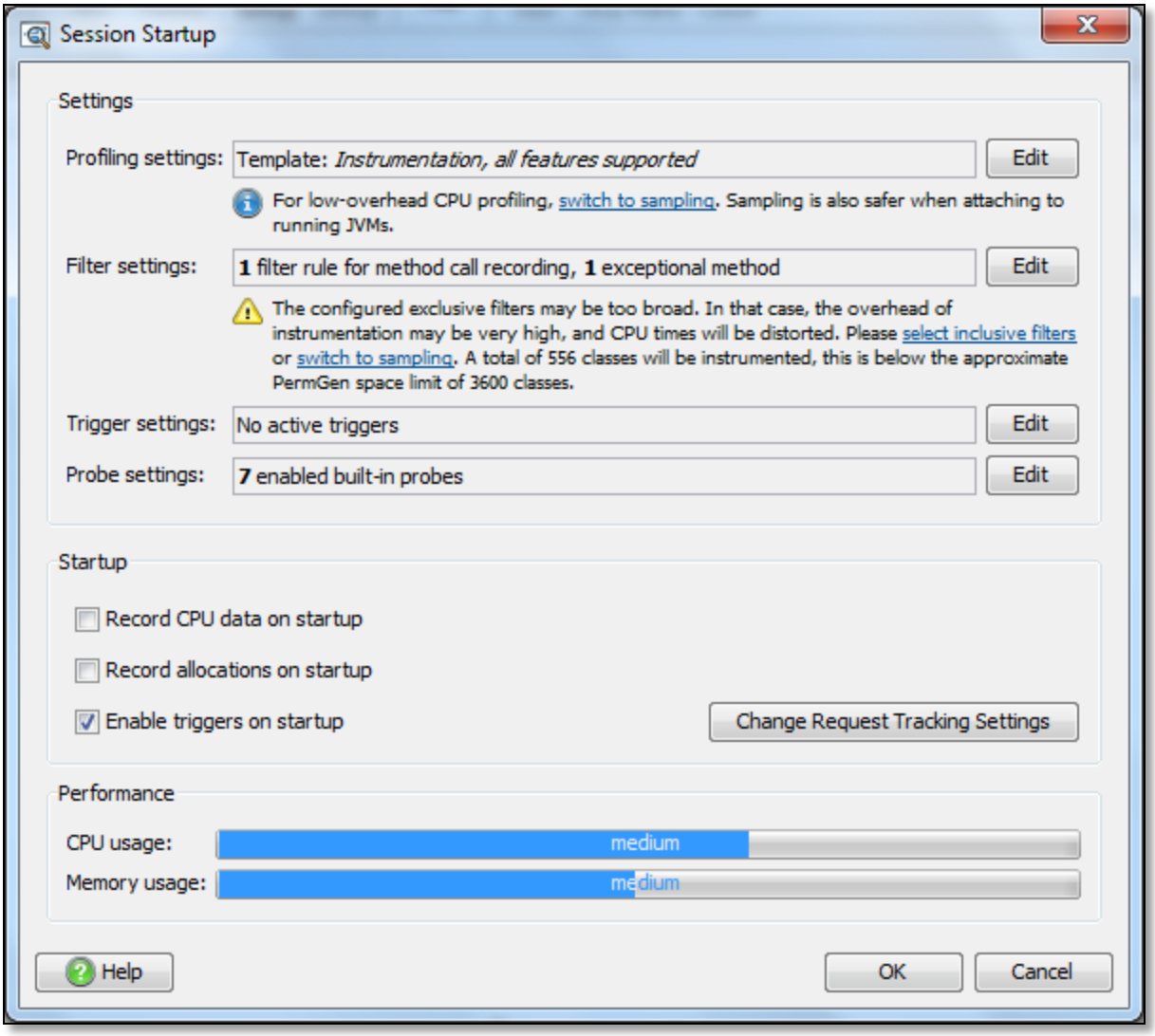
1. Install Maven archetype from folder jprofiler-webapp-archetype
2. Create a web app “jprofiler-webapp-project” from the archetype defined in step i above
3. Package the web project “jprofiler-webapp-project” into a WAR file “jprofiler-webapp-project-1.0-SNAPSHOT.war” and copy it in the webapps folder of te tomcat directory(apache-tomcat-7.0.22)
4. Open a browser window which will show the “Search users” page of the sample web app

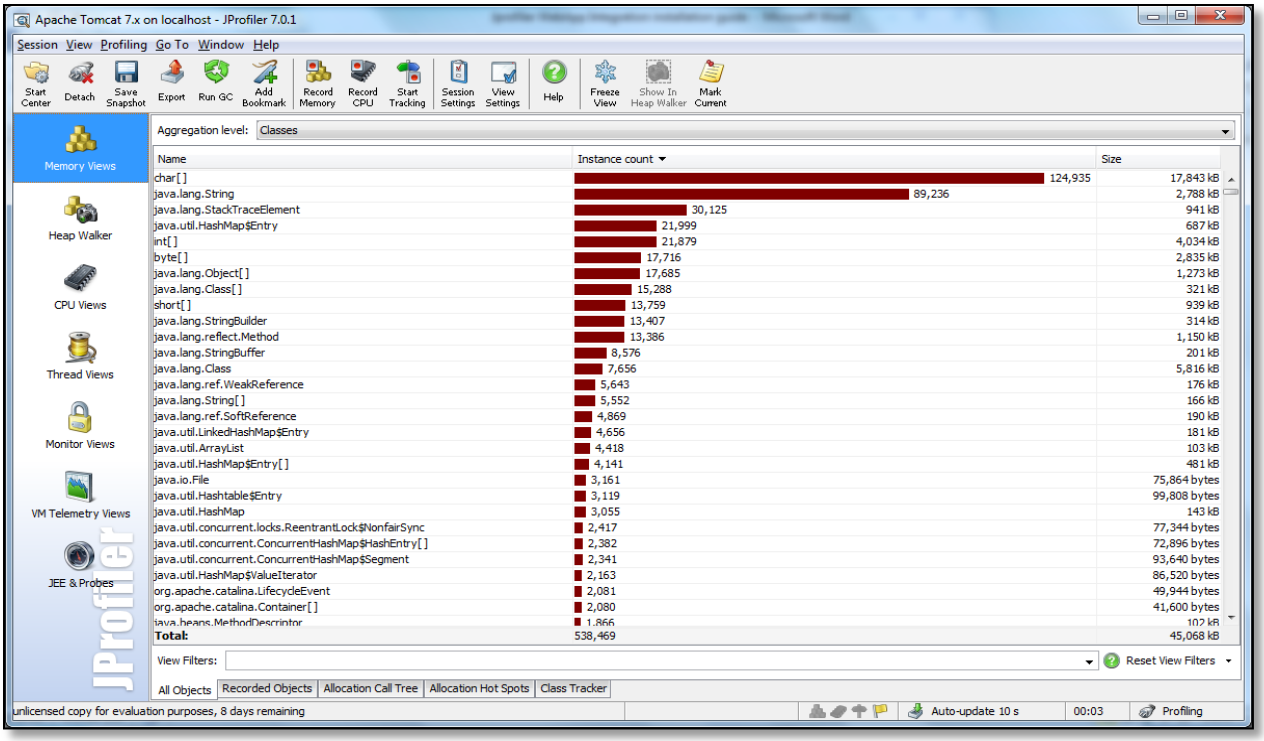

http://localhost:
Search Users
File Edit View Favorites Tools Help
Search Users
User Name:
Role: -Please Select--
Search
User Id
User Name
Role
Add User

Once the Application is up, go back to Jprofiler window and select Ok on the session settings window

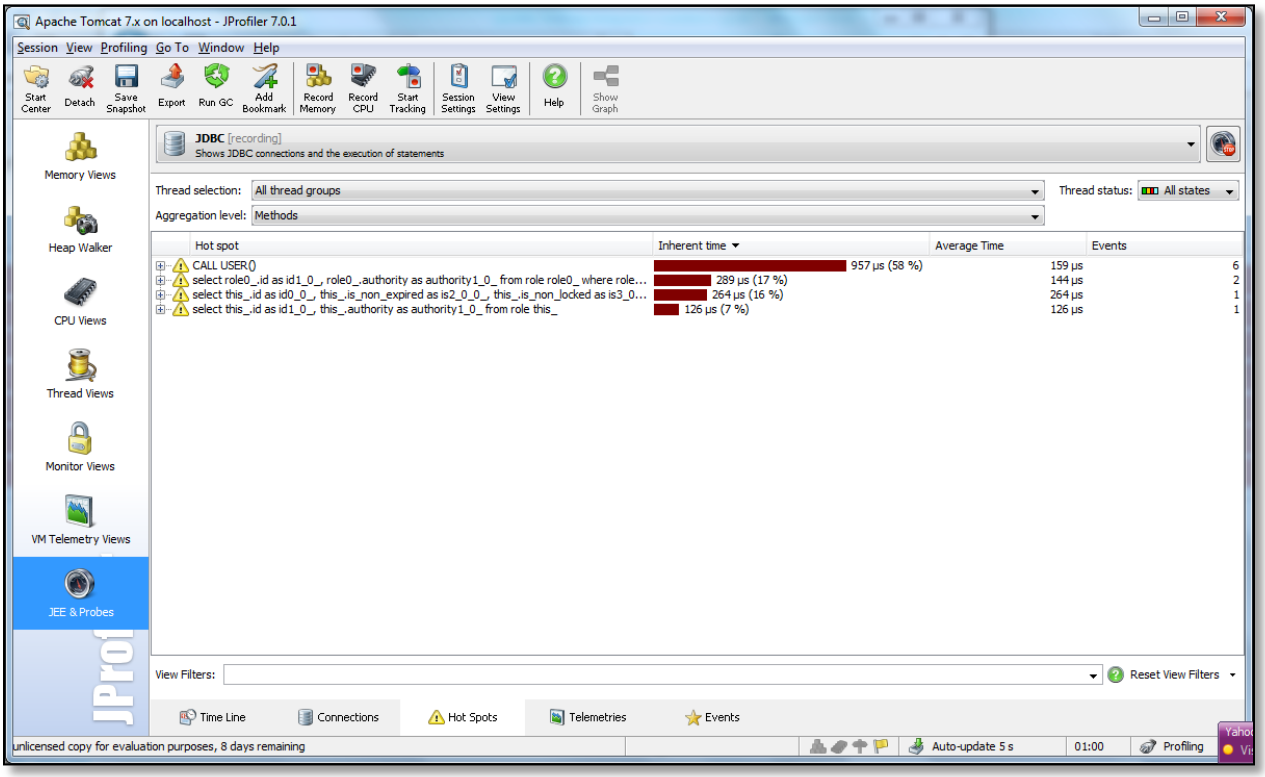


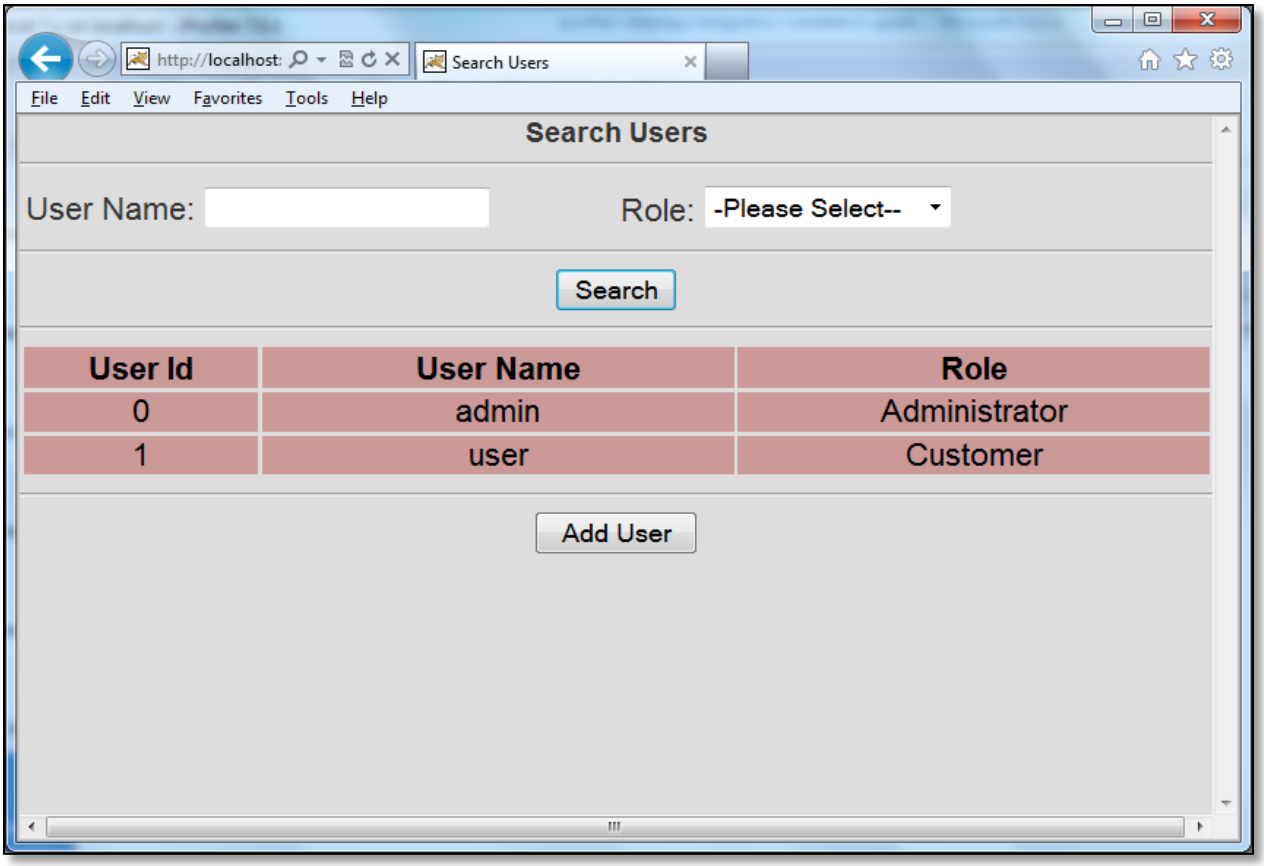

Session Startup
Settings
Profiling settings:
Template: Instrumentation, all features supported
Edit
For low-overhead CPU profiling, switch to sampling. Sampling is also safer when attaching to running JVMs.
Initial Profiling Settings
Please choose your initial use case:
Instrumentation
All features, such as invocation counts and method statistics are supported. Good filters are critical for overhead.
Sampling
CPU profiling is not distorted by short-running methods. Overhead is extremely low. Some features are not supported. This mode is safer when attaching to running JVMs.
Profiling settings can be changed later on at any time.
CPU usage:
medium
Memory usage:
medium
Help
OK
Cancel


Session Startup
Settings
Profiling settings:
Template: Instrumentation, all features supported
Edit
For low-overhead CPU profiling, switch to sampling. Sampling is also safer when attaching to running JVMs.
Filter settings:
1 filter rule for method call recording, 1 exceptional method
Edit
The configured exclusive filters may be too broad. In that case, the overhead of instrumentation may be very high, and CPU times will be distorted. Please select inclusive filters or switch to sampling. A total of 556 classes will be instrumented, this is below the approximate PermGen space limit of 3600 classes.
Trigger settings:
No active triggers
Edit
Probe settings:
7 enabled built-in probes
Edit
Startup
Record CPU data on startup
Record allocations on startup
Enable triggers on startup
Change Request Tracking Settings
Performance
CPU usage:
medium
Memory usage:
medium
Help
OK
Cancel

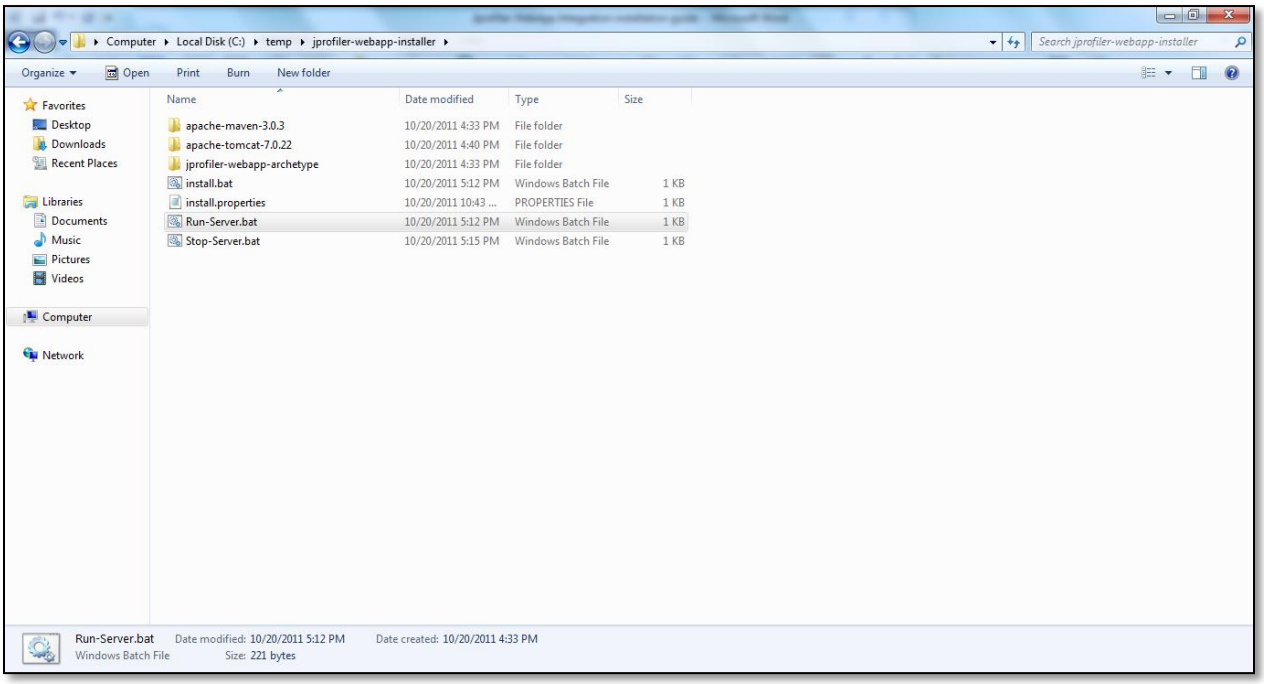



Click on Search Users and see the SQL queries fired in the JEE & Probes → JDBC → HotSpots view as shown below:



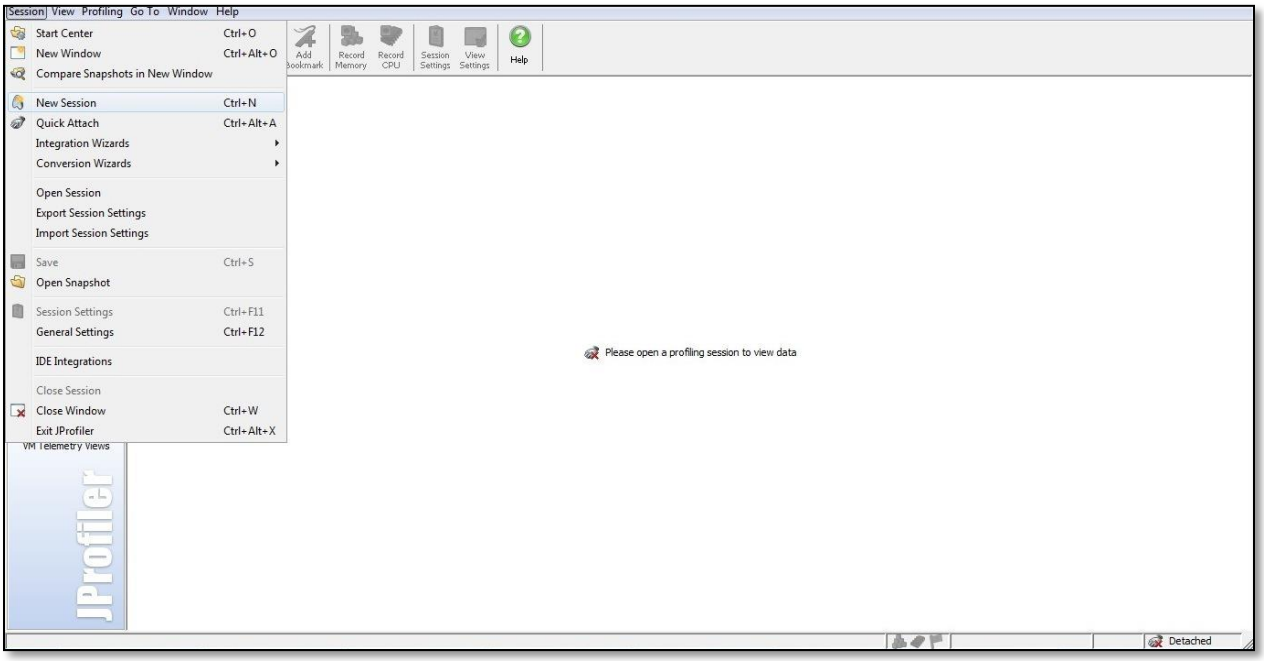



You can run and stop the server using the batch files Run-Server.bat and Stop-Server.bat inside the folder jprofiler-webapp-installer

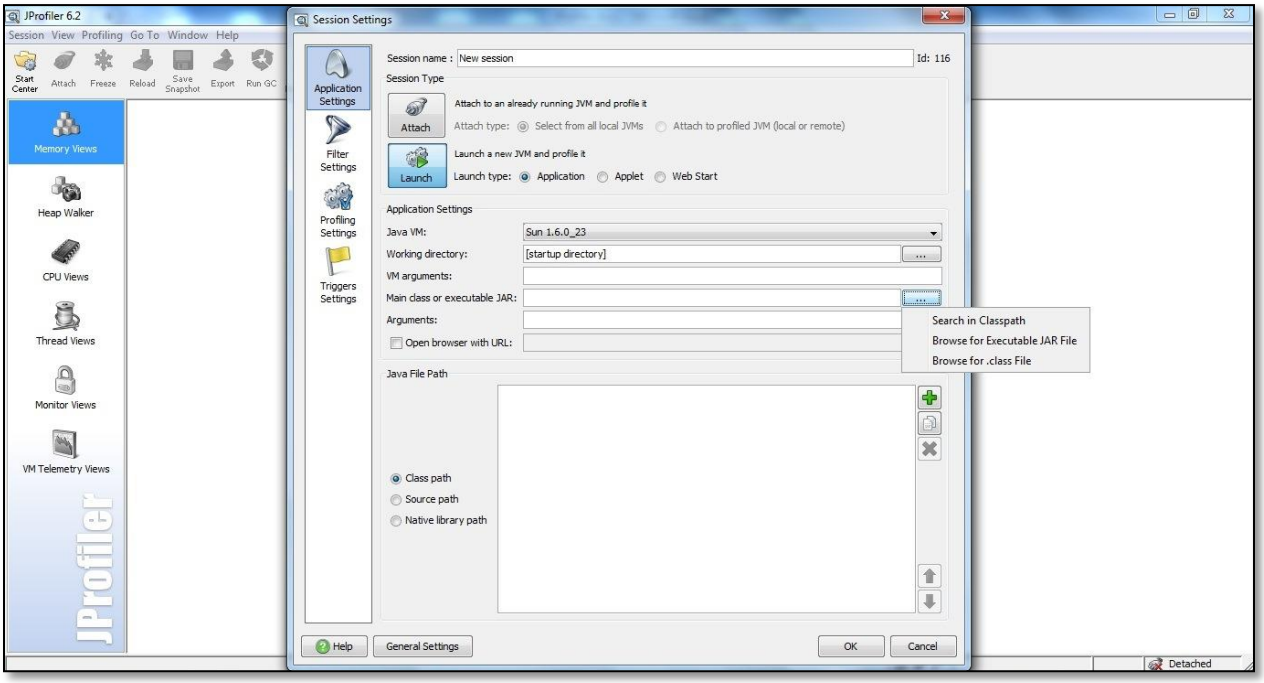


## *Running a local class file:*

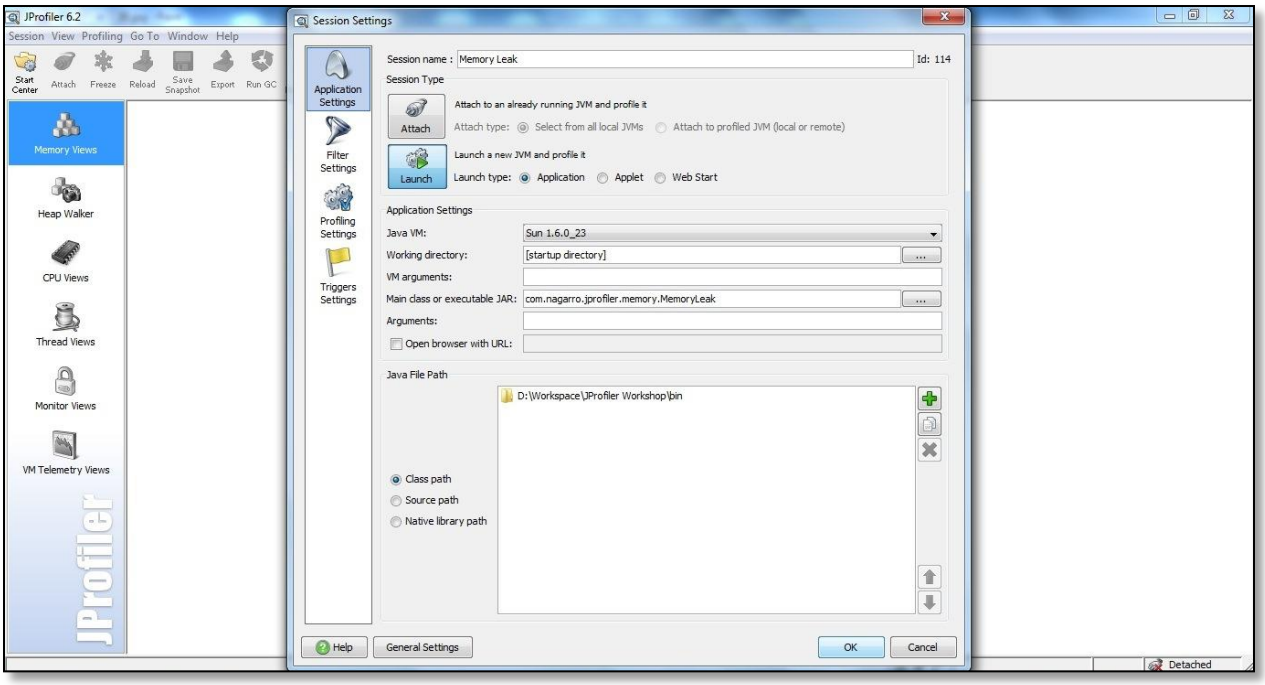
1. Click on **Session** dropdown and select **New Session**.




2. Click on **Launch**. Click on browse button and select the **Browse for .class File** option. Browse the main class file.



3. After selecting the class file the screen will look like as shown below. Click on **OK** to start profiling.




This procedure has to be done for the given code snippets. Each of the code snippets is an example to be used for different kind of profiling which are explained further. The list is as given below:

CPUProfiling - CPU profiling and JDBC Probe

MemoryLeak - Memory profiling for memory leaks

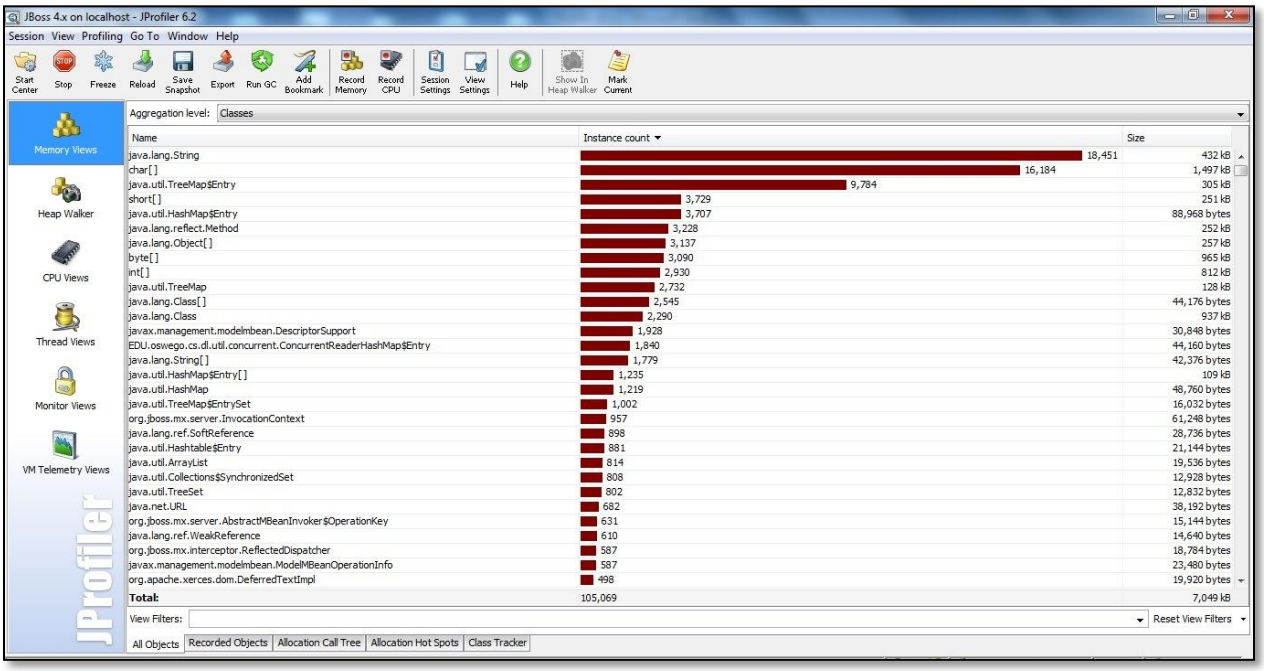
MemoryUsage- Memory profiling for high memory usage

DeadLockDemo- Thread profiling

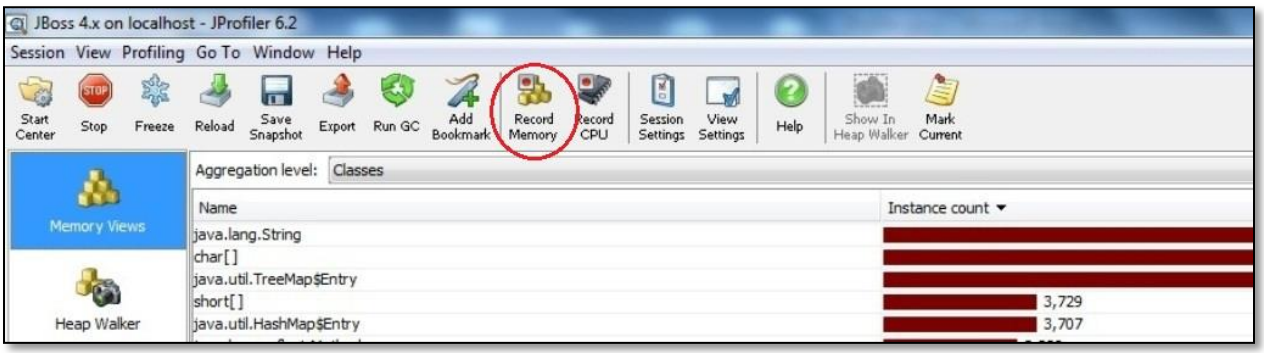
The right hand name indicates the class file name and on the right hand is the profiling which is to be done using this code. It is suggested to first do the profiling using the given code and then improve the results by modifying the code.

## *Memory Profiling*

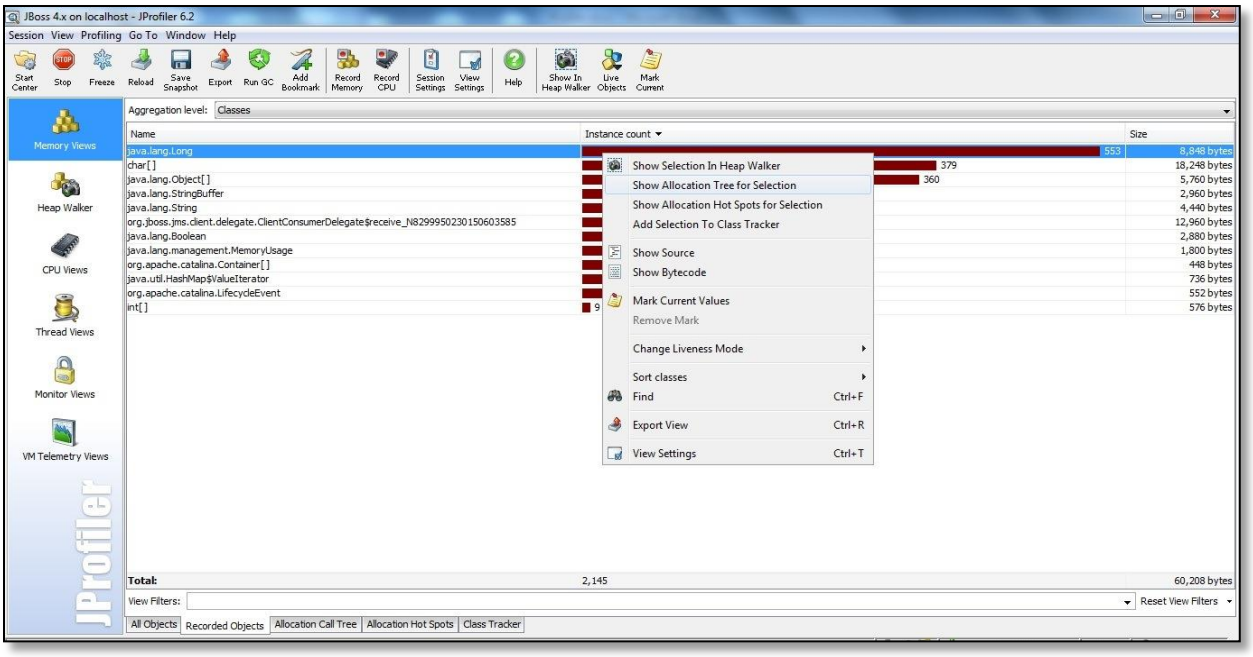
When we start the profiling, we can see number of instances and total size for any object which is being created.



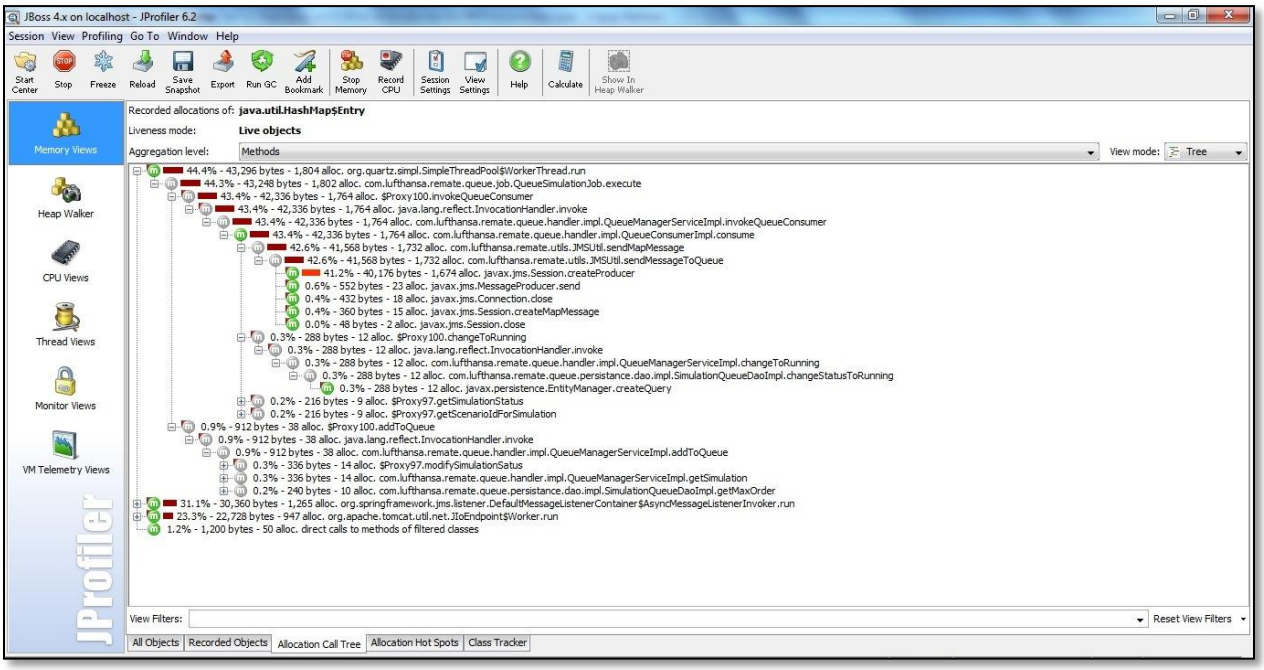

The memory usage can be recorded from the button shown below and the recorded objects can be viewed in the **Recorded Objects** tab:



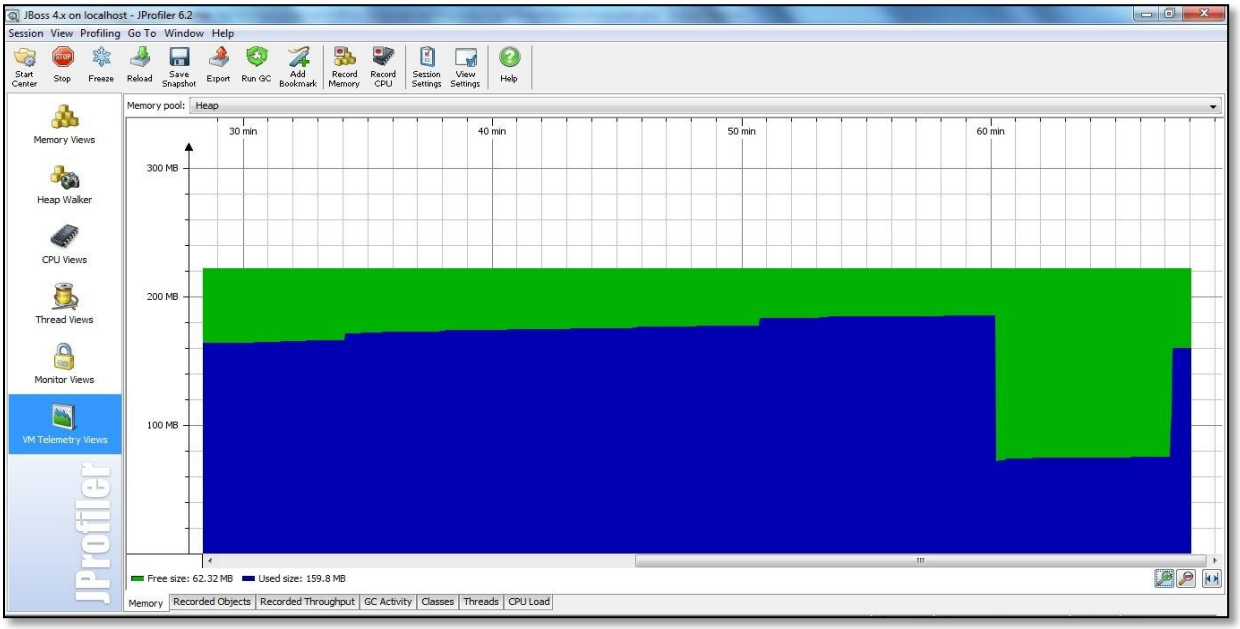
After recording these objects can be analyzed. The allocation of these objects can be seen by right clicking and selecting **Show Allocation Tree for Selection** option.



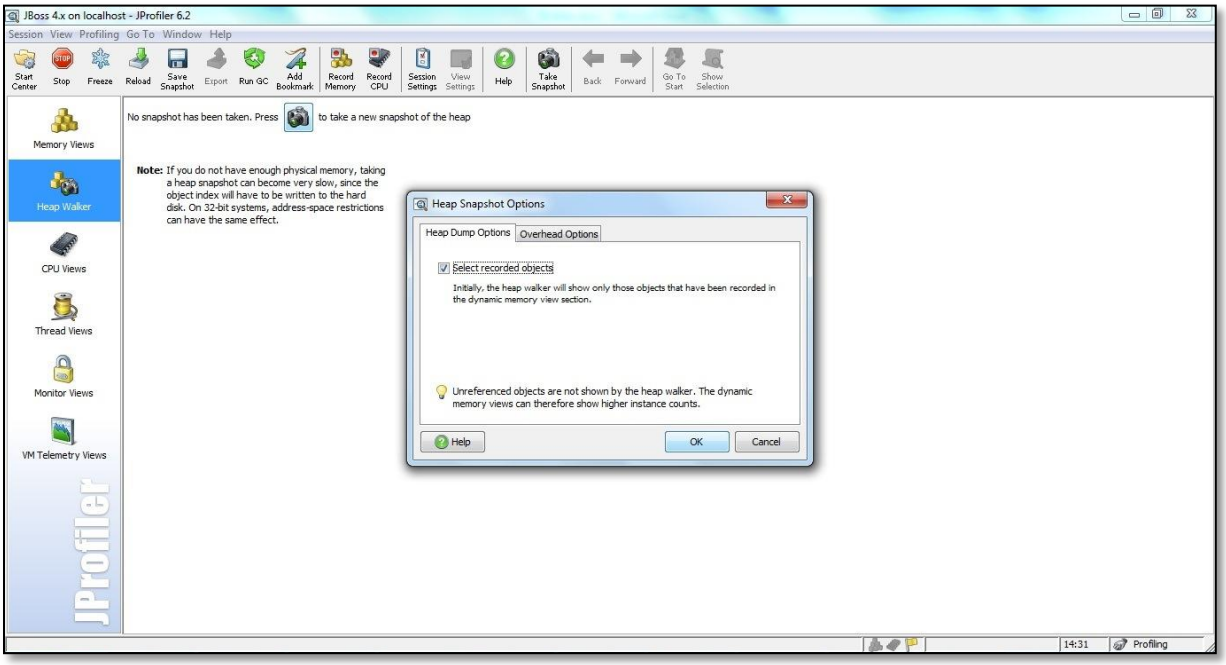

When the above option is selected, the entire tree is shown for the invocation of that particular object. This invocation tree can be seen various **Aggregation levels** like package, classes or method level. We can then narrow down to an API and analyze the problem.



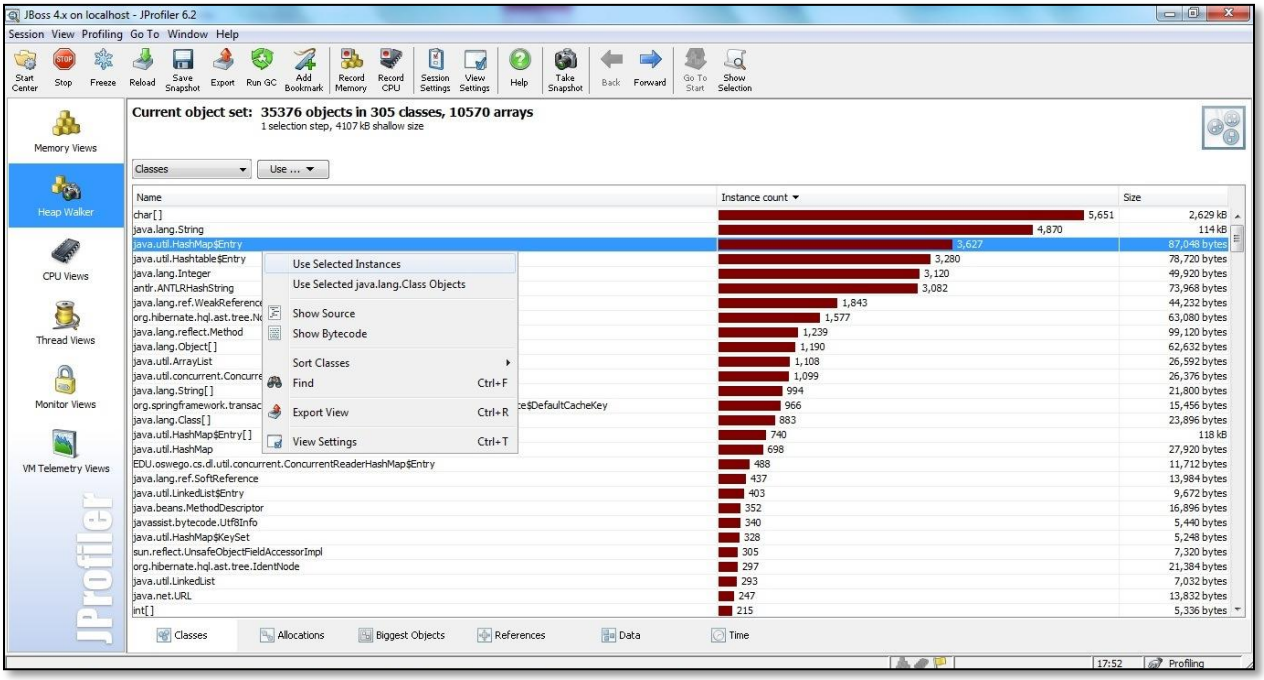
We can also view the **VM Elementary Views** while an application is running. It shows the memory usage at every point. We can then point out the areas where this usage is very high or has increased drastically.

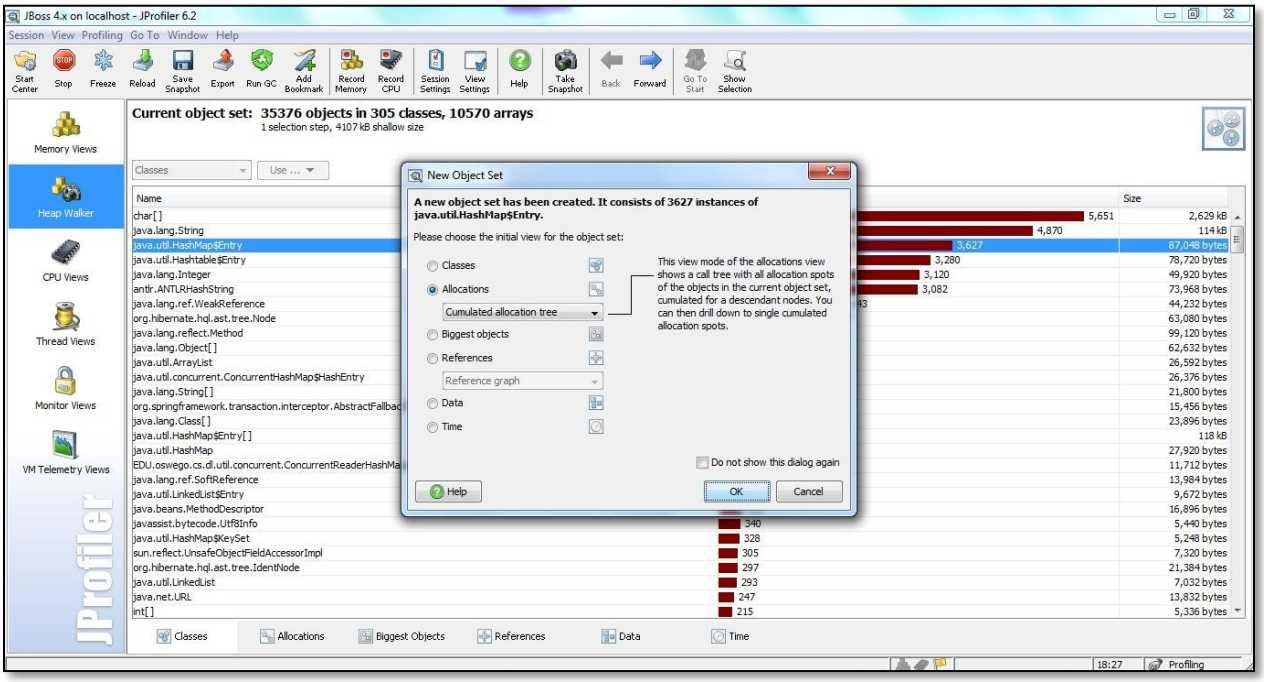



In order to determine the memory leakage, a heap snapshot can be taken of the recorded objects. The snapshot contains the objects which are still being referenced and are not available for garbage collection.

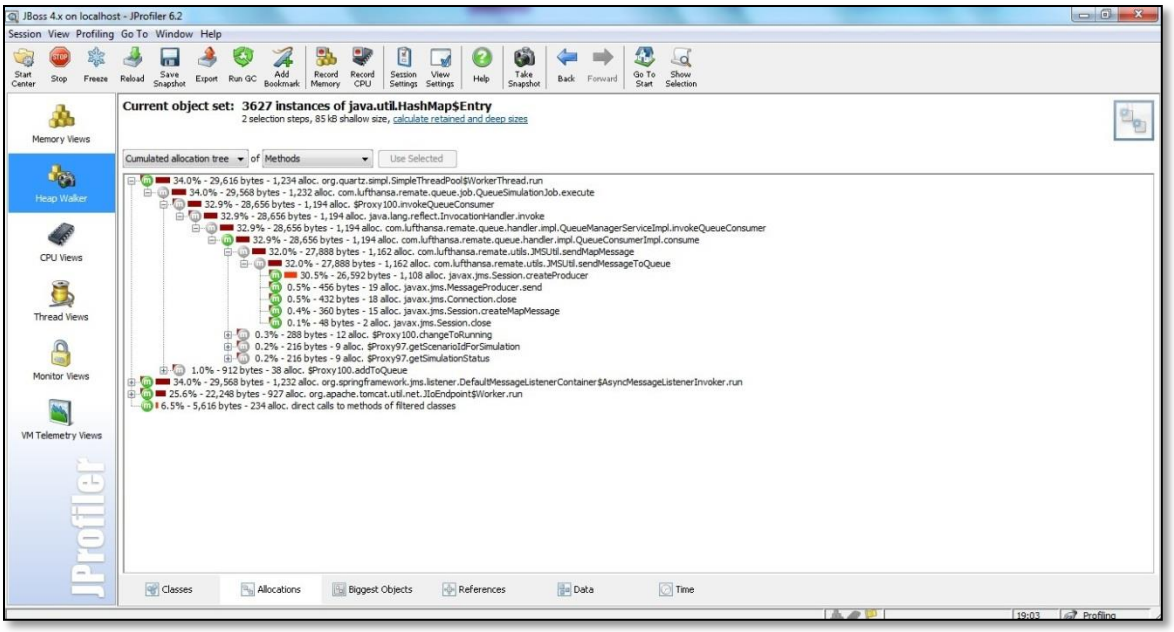


After taking the snapshot we can see the allocation of those objects by following the below steps:

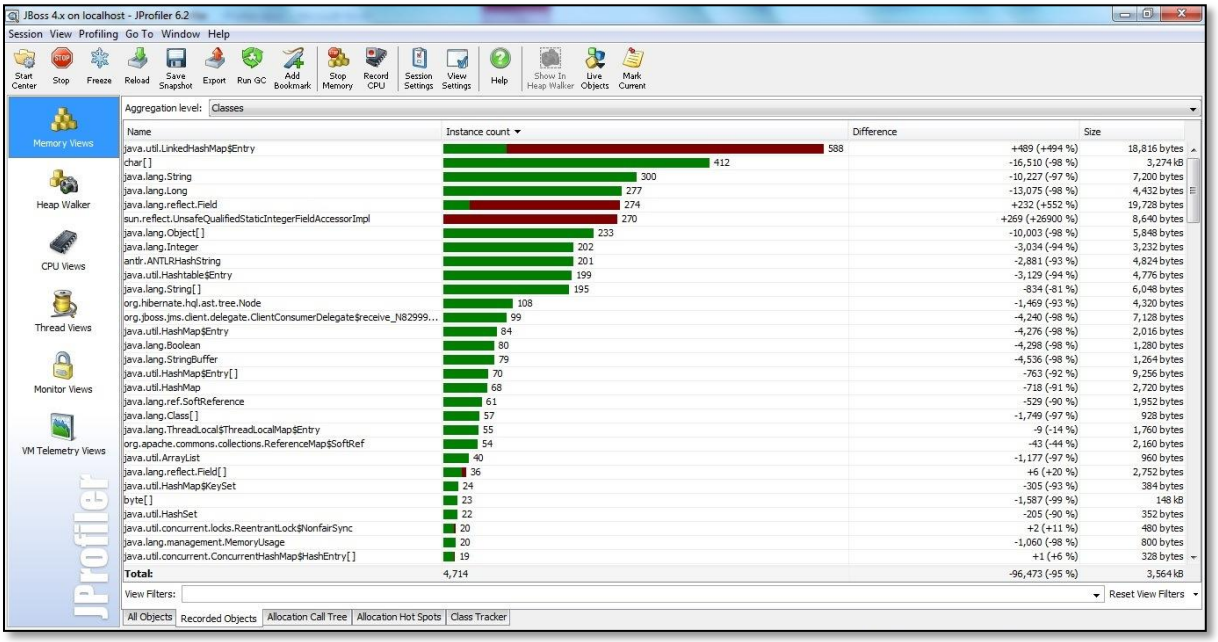


When we click on OK we get the below screen. It shows the division of object instances that have been created by various methods.

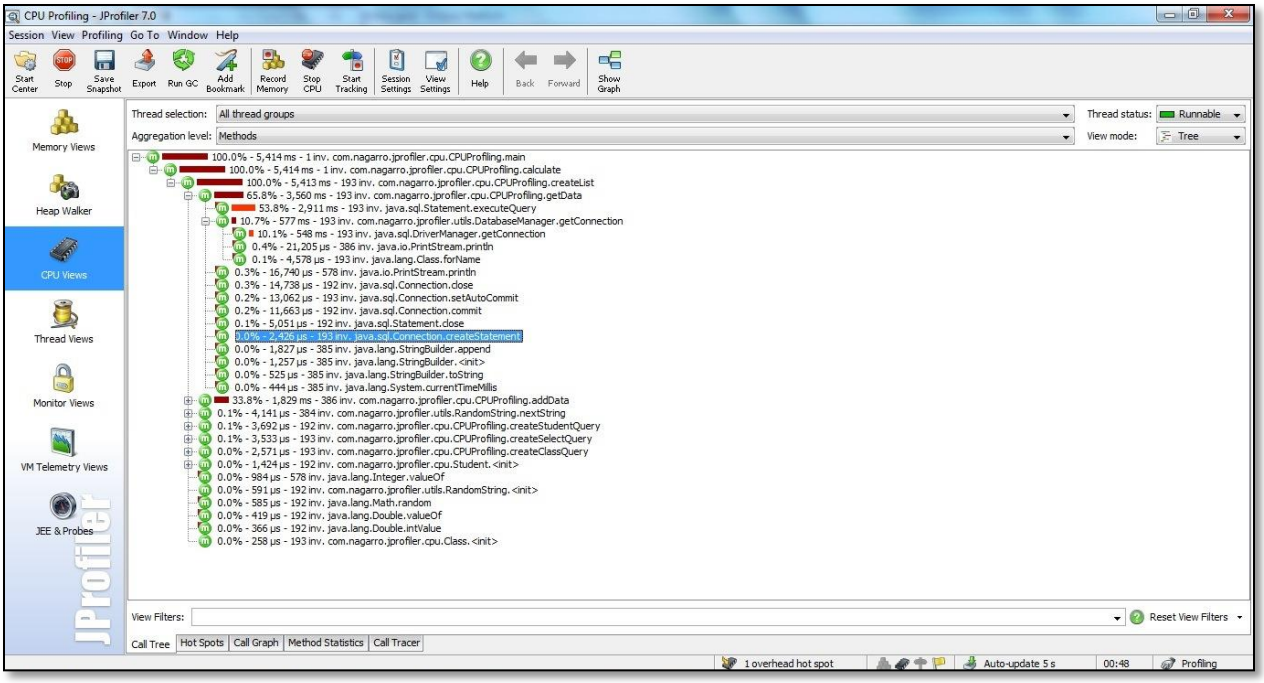



There is another feature in JProfiler which helps us to track the increase in number of instances from a particular point. The current memory usage can be marked by clicking on **Mark Current**. The initial value is shown in green and the increase is shown in red. Also a new column appears which shows the difference between the current value and the initial value.

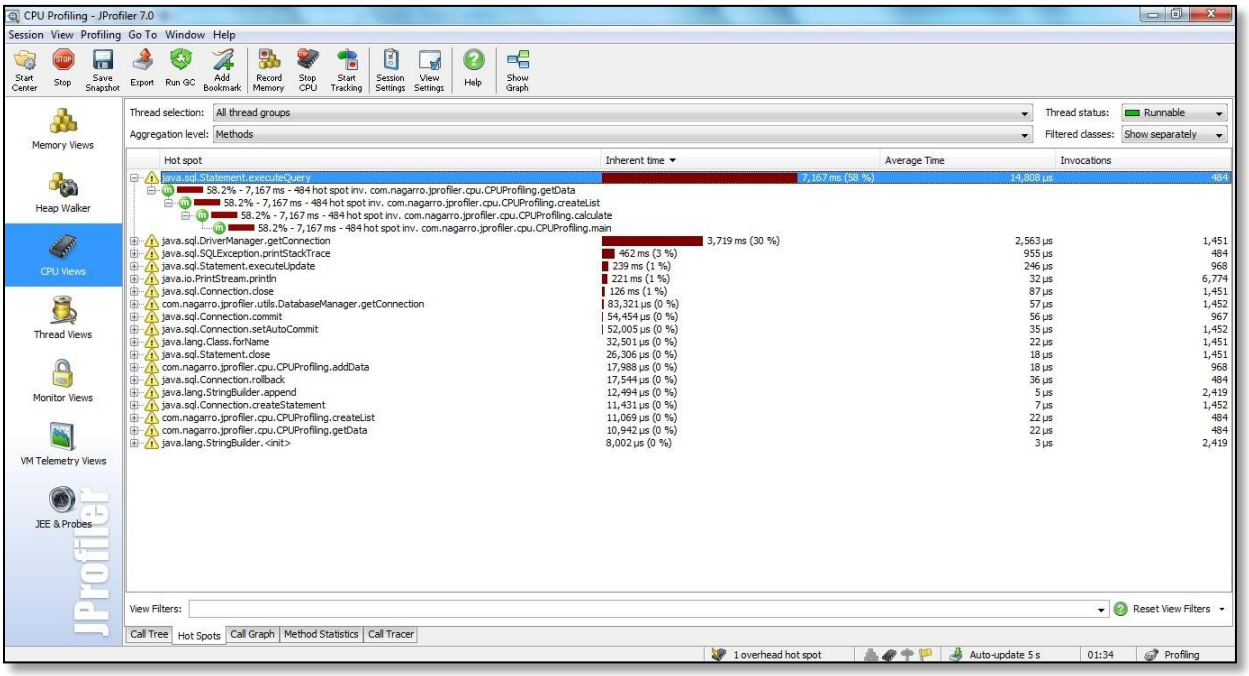


## *CPU Profiling*

It can be done by clicking on start CPU recording button.  The call tree for all the methods can be seen in the Call Tree tab.

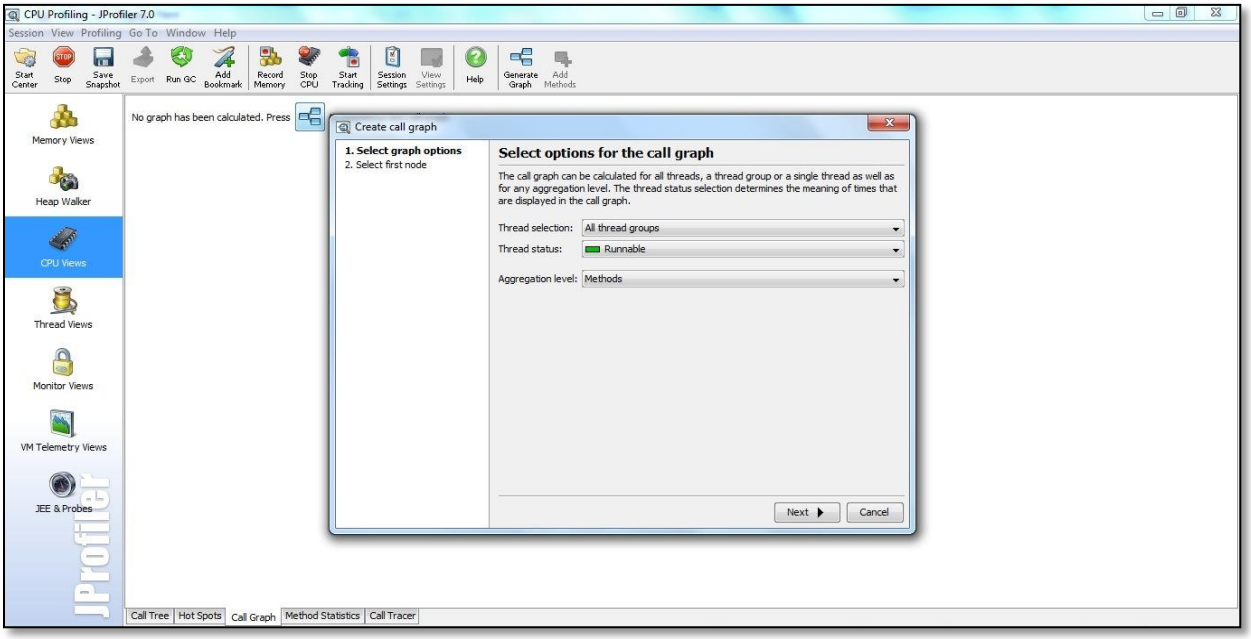


Further, there is a Hot Spots tab which shows the methods which are being invoked. There is an Inherent time column which shows the time spend inside that particular method. It does not include the time spend inside the method which are being invoked from this method. The Invocations column shows the number of times that method was invoked. The average time column shows the inherent time divided by number of invocations.

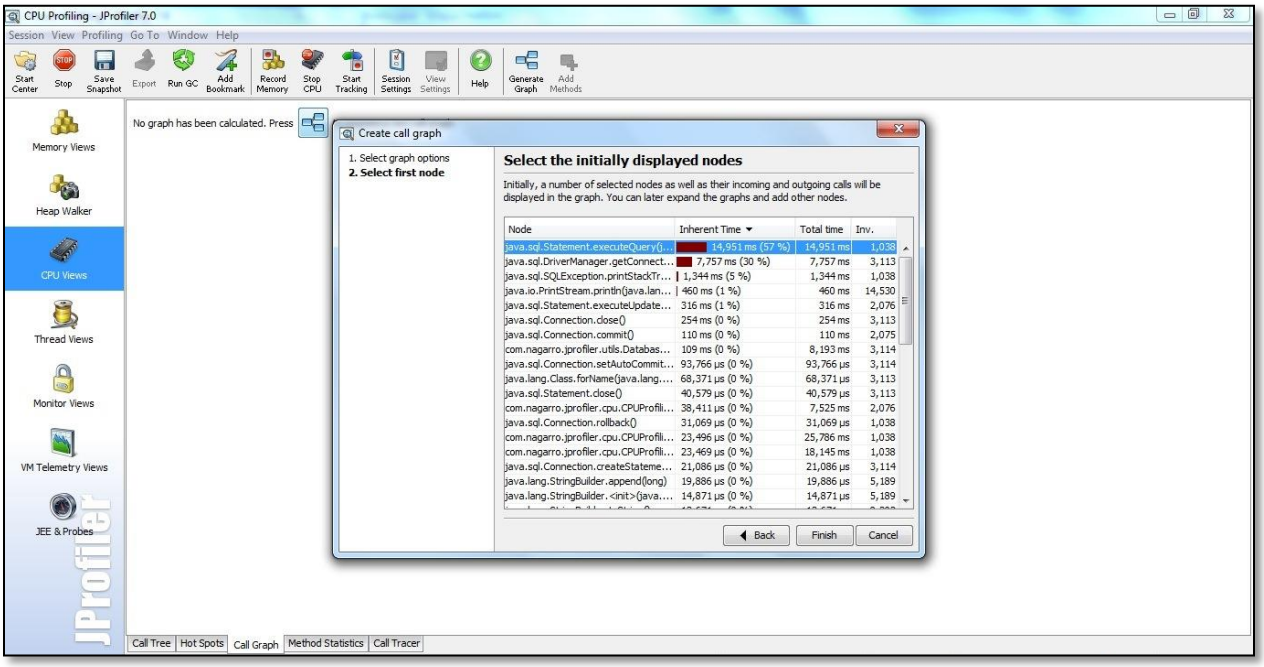


The call graph for any particular method can be seen by clicking on Generate Call Graph icon. It shows all the invocations of that method. This can be done as shown in the steps given below:

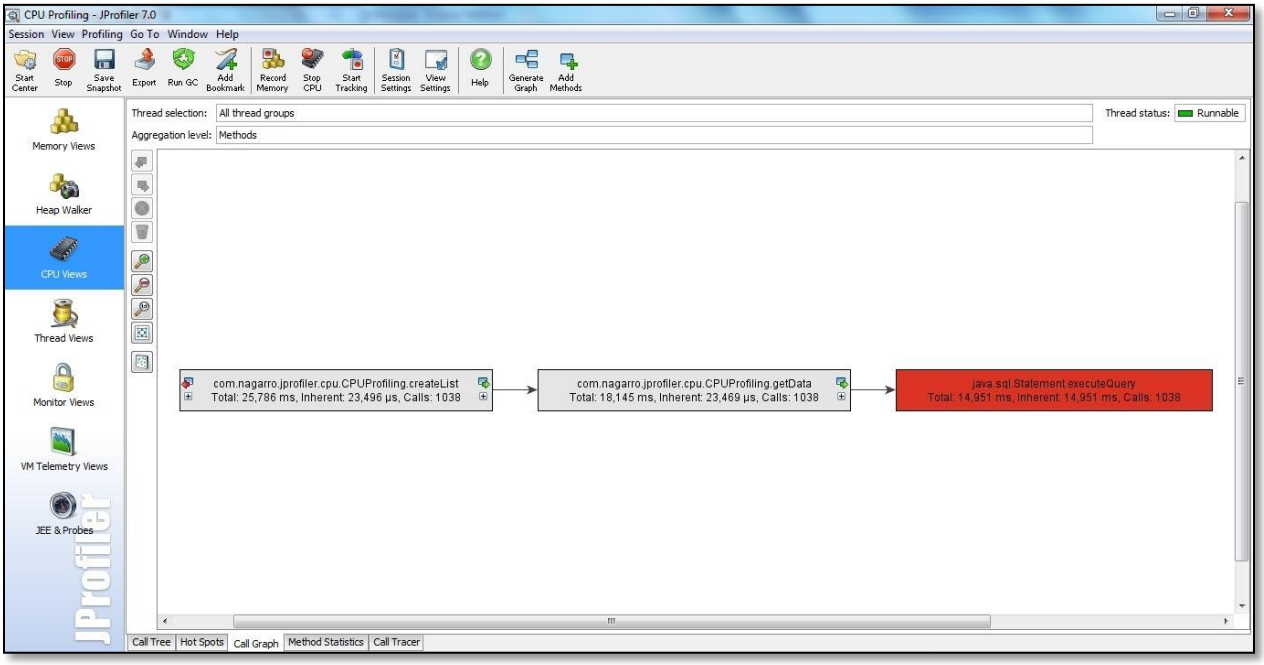
1. Click on Generate Call Graph button.




2. Select the method for which the call graph needs to be generated.

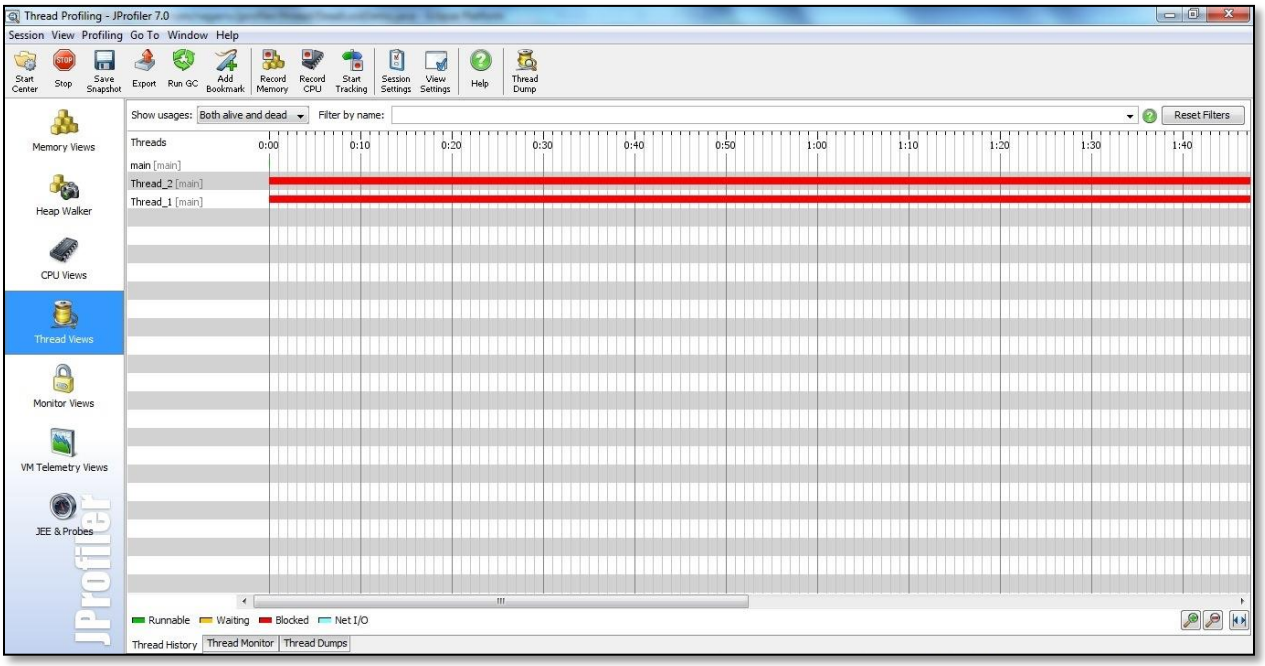


3. The generated call graph looks as shown below.

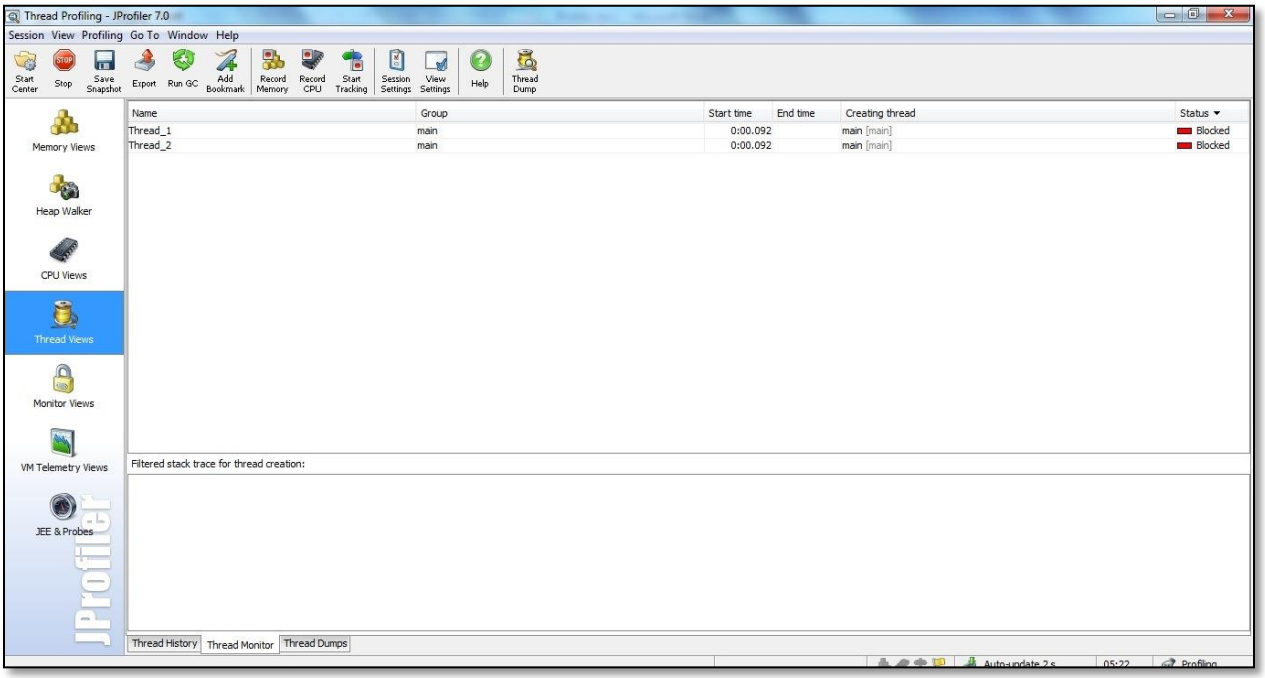


## *Thread Profiling*

It shows the data on thread level. The thread history shows the status of thread over a period of time. The possible states are Runnable, Waiting, Blocked and Net I/O.

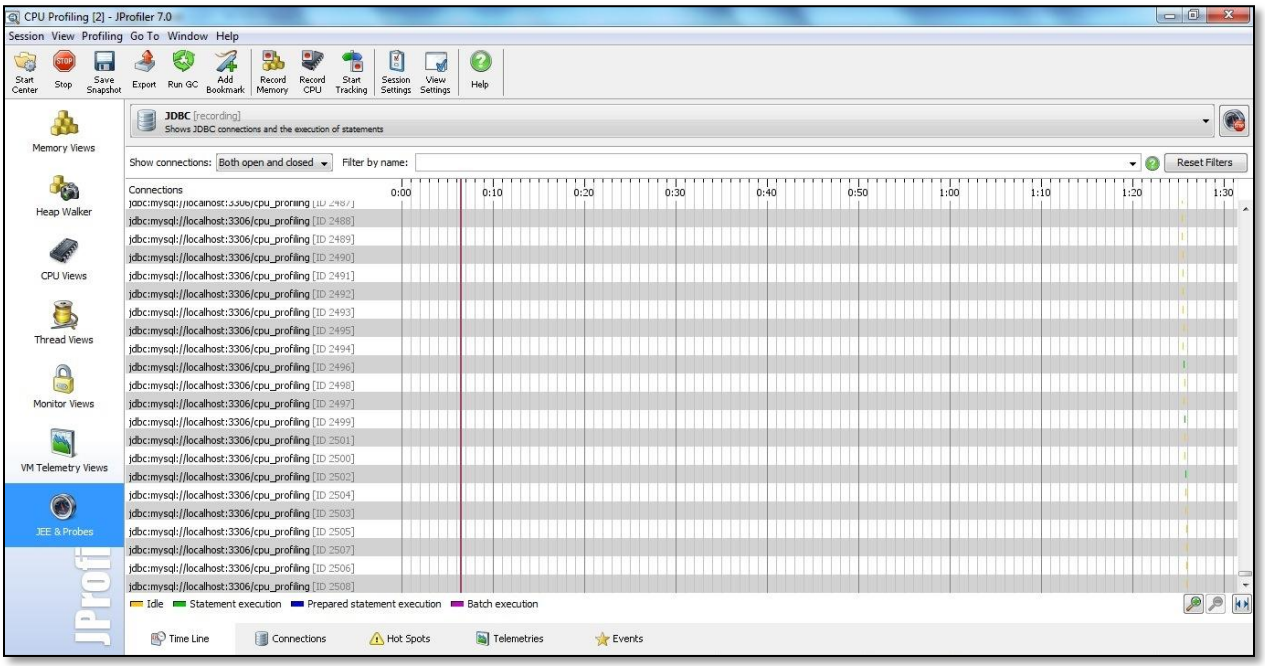



The Thread Monitor shows the state of a thread, its start time, end time and its creator thread.

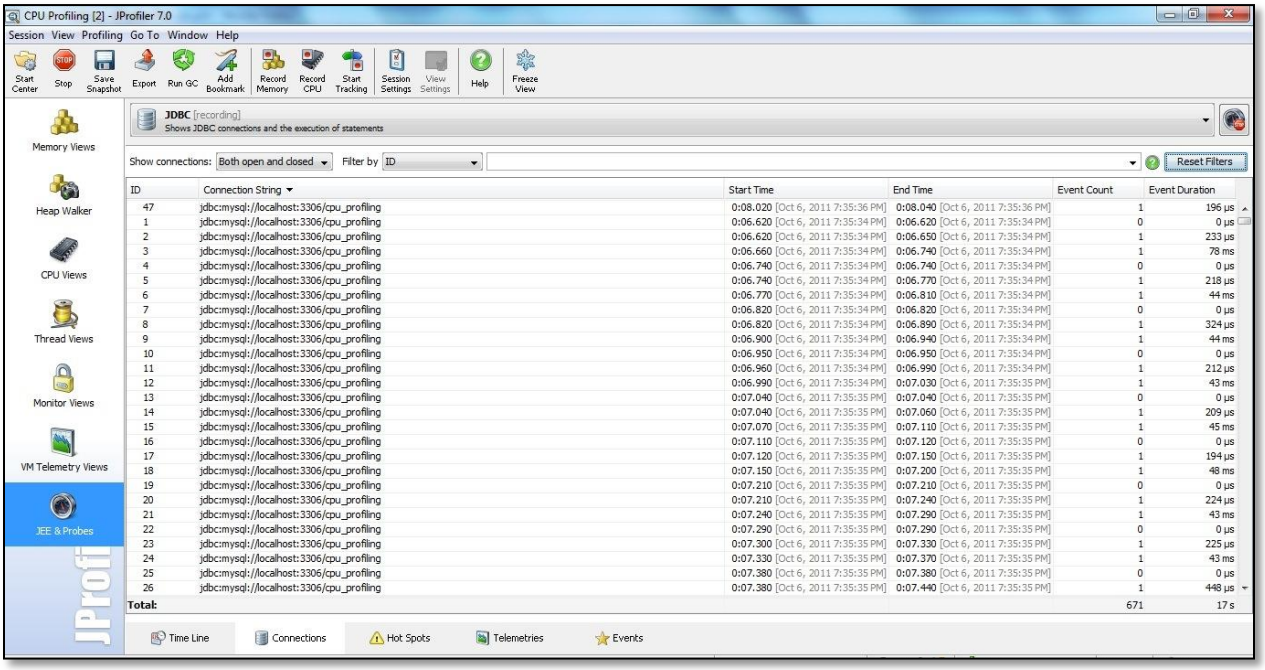


## *JDBC Probe*

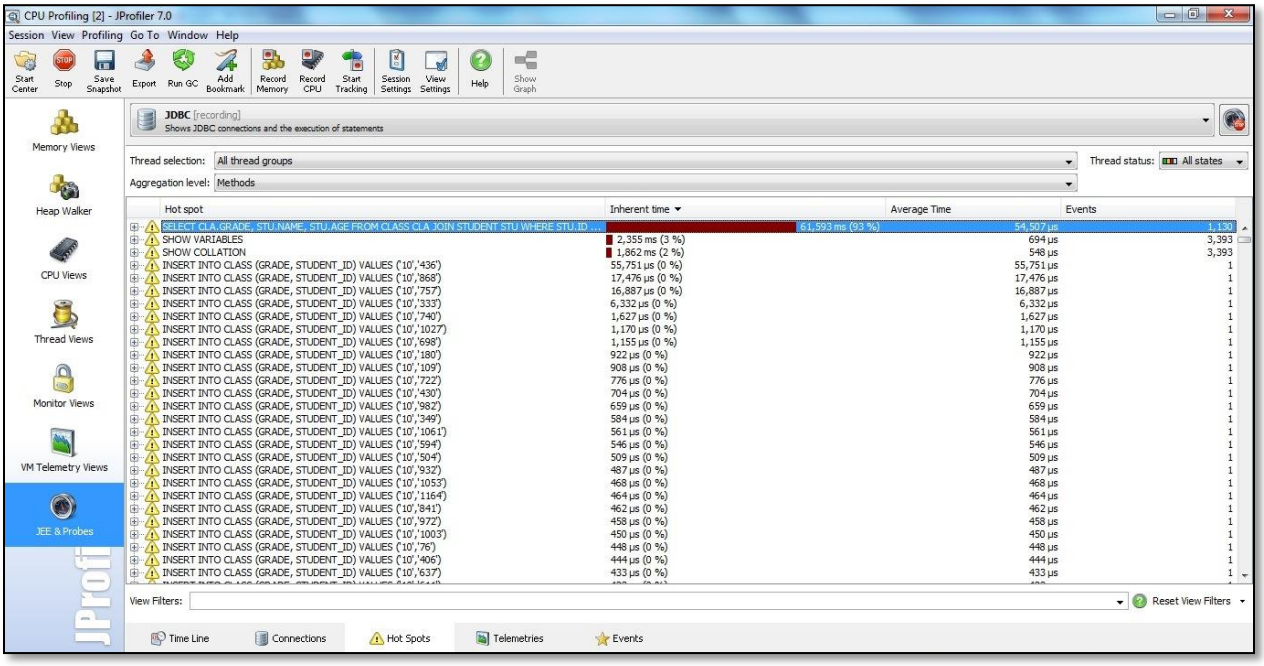
This is a new feature in JProfiler 7. Here we can see the details of various connections which are being created in the JVM. The TimeLine tab shows the state of a connection and the time for which it was open and its state i.e.  what kind of operations is it performing or is it idle.



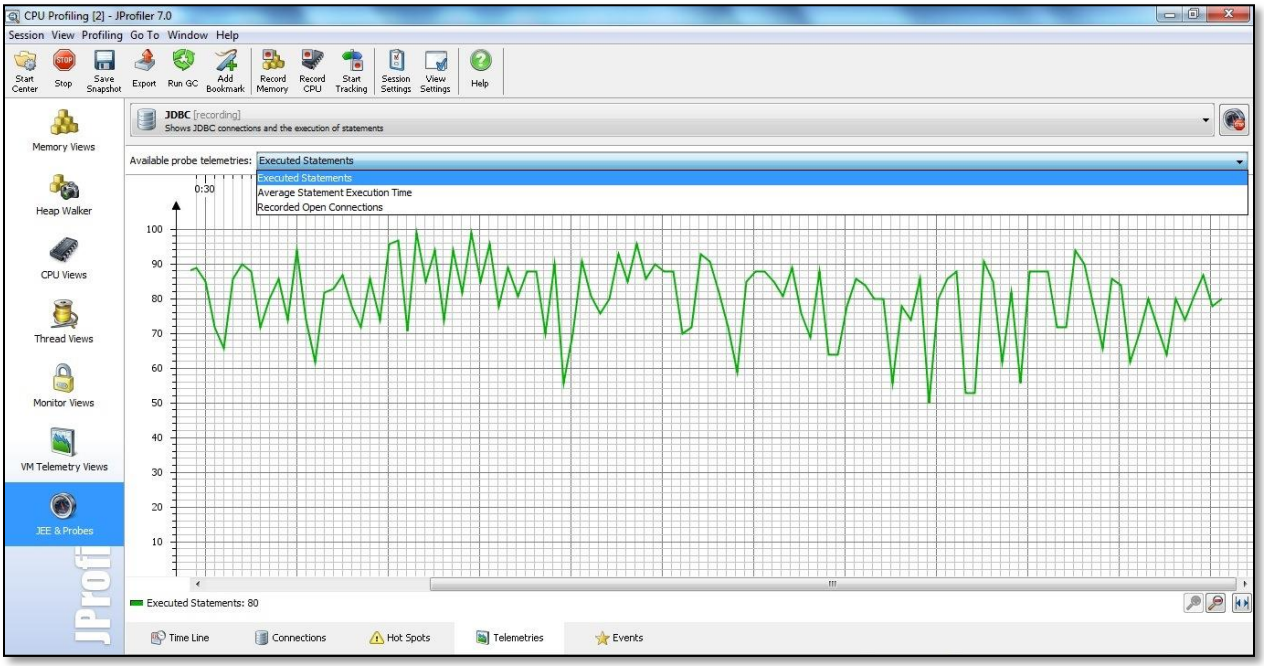

The Connections tab shows the connections by IDs.



The Hot Spots view shows the queries which were fired and the Inherent Time i.e. the total time taken in firing that query. The Events column gives the count of the number of times that query was fired. The Average Time is equal to the Inherent Time / Events.




The Telemetries tab shows the graphical presentation of the count of Executed Statements, Average Statement Execution Time and Recorded Open Connections.



The Events tab shows the events like Connection opened, Connection closed, Statement Execution. In this case if we click on a particular event, we can see the Stack Trace of its invocation. This however is possible only if the CPU recording was enabled parallel to the probe recording.