Java Update

Jim Johnston
z/TPF Development

Background – Why Java on z/TPF?

z/TPF application code and development environments are very advanced and can be daunting to new comers. Finding traditional z/TPF programming skills and integrating them quickly can be challenging.

Java on z/TPF can help with skill rebalancing while improving time to market.

Background – Benefits of Java

- ✓ Anything Java™ compatible will run on z/TPF (reduces scope of new projects if technology already implemented in Java)
- ✓ Java hides the plumbing of programming, making it intuitive (less time spent debugging low value issues)
- ✓ Java applications tend to be feature rich
- ✓ Java supports a dependency driven build environment that is ubiquitous
- ✓ The web has a vast repository of coding examples for Java from different sources
- ✓ IBM fully supports Java and is continuously enhancing functionality and performance (source is publicly available)

Background – The Challenge

Need Java and z/TPF to Play Nice!

Background – The Response

- APAR PJ43892 (2017) provided:
  - ✓ Ability for traditional z/TPF applications to invoke local Java services (tpf_srvcInvoke)
  - ✓ Application management for Java (JAM support)
  - ✓ Ability to read Java dumps using Linux utilities
- APAR PJ44844 (2018) provided:
  - ✓ Ability for local Java applications to invoke atomic (stateless) traditional z/TPF application services using REST APIs
- Starter Kits proving ease of adopting opensource Java technologies:
  - ✓ Drools Rules Engine (2017)
  - ✓ Kafka Client (2018)

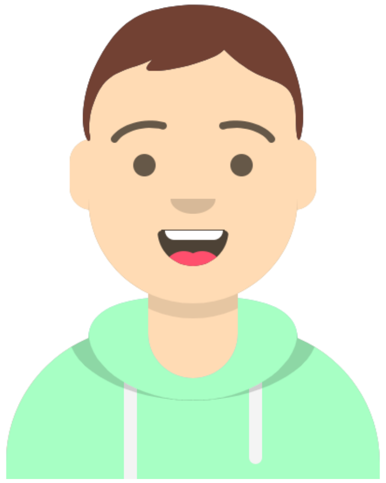
Problem

How can we write Java applications which can exclusively hold native z/TPF resources across the system?

Users

Anna
Application Architect

"I'm excited about Java applications in development which will let us incrementally modernize our applications."



Joe
Java application developer

"I have an idea for a high value Java Application that will service incoming requests by taking advantage of both TPF Databases and Java implemented technologies."

Pain Points

Anna
Application Architect

- Anna wants Java to TPF Applications to **maintain database consistency across multiple calls**

...but the **current Java calling traditional TPF code interface does not have that capability.**

Pain Points

Joe
Java application developer

- Joe likes **how easy it is to use Java to TPF stateless support which uses the Java Object Model**

...but he is **concerned about the change in complexity of his Java application when he goes to convert it to a stateful application.**

Support for Stateful Services for Java on z/TPF

Provides ability to make multiple calls from Java to the same stateful ECB.

- Java applications can now run business logic between service calls to the same ECB while maintaining database consistency and holding other resources.

- Enables new Java applications to incrementally leverage a broader scope of existing TPF applications by combining both the design flexibility inherent with the REST programming model and the new stateful support.

- Stateful services extends existing stateless services taking advantage of easy to use Java Object Model.

PJ45433

Before Stateful Service Support

Stateless Service Balance Transfer #method 1

The services are independent and introduce data consistency challenges.

Each service call is handled by a new ECB

```
JohnAccountInfo = api.readAccount("John");
MaryAccountInfo = api.readAccount("Mary");
//enhance transaction capabilities, i.e., rules, publishing
JohnAccountInfo.setBalance(JohnAccountInfo.getBalance()-1000);
MaryAccountInfo.setBalance(MaryAccountInfo. getBalance()+1000);
api.updateAccount(JohnAccountInfo);
api.updateAccount(MaryAccounInfo);
```

Would need to be optimistic updates.

Stateless Balance Trasfer #method 2

Maintains consistency, but...
Not really rebalancing code distribution, most of heavy lifting still done in traditional applications.

```
api.balanceTransfer("John", "Mary", $1000);
```

After Stateful Service Support

Stateful Service Balance Transfer

```
JohnAccountInfo = api.readAcctwLock ("John");
MaryAccountInfo = api.readAcctwLock ("Mary");
//enhance transaction capabilities, i.e., rules, publishing
JohnAccountInfo.setBalance(JohnAccountInfo.getBalance()-1000);
MaryAccountInfo.setBalance(MaryAccountInfo. getBalance()+1000);
api.updateAccount(JohnAccountInfo);
api.updateAccount(MaryAccounInfo);
api.unLockAll();
```

Each service call is handled by the **same** ECB!

Java can now control the state of the traditional TPF code. (e.g., locking)

No special application code required, looks just like stateless.
Stateful ECB automatically created and destroyed !!

Stateful Service Descriptor

Stateless

```
{
  "version": 1,
  "providerType": “Program",
  “timeout”:”6000”,
  "request" : {…},
  "response" : {…},
  "services" : [ {
        "version" : 1,
        “provider” : “BRED”,
        "operationId" : "readAccount“ },
      { "version" : 1,
        “provider” : “BUPD”,
        "operationId" : “updateAccount“}
  }]
}
```

To define a stateful service we only need to define the ‘providerType’ as StatefulProgram.

Stateful

```
{
  "version": 1,
  "providerType": “StatefulProgram",
  “timeout”:”6000”,
  "request" : {…},
  "response" : {…},
  "services" : [ {
        "version" : 1,
        “provider” : “BRDL”,
        "operationId" : “readAcctwLock“ },
      { "version" : 1,
        “provider” : “BUPD”,
        "operationId" : “updateAccount“}
  }]
}
```

Stateful Service Provider

```
CustomerStateManager  globalInstance;

extern "C" void
ABCD(void *request, int reqsize, tpf_srvc_token http_token)
{
        tpf_srvc_resp response;
        requestFormat *reqFormat = (requestFormat *) request;
        responseFormat respFormat;
        //native stateful service logic goes here
        //Calls existing traditional TPF Applications
        response.status = IHTTPS_STATUS_500;
        response.data = &respFormat;
        response.datalen = sizeof(respFormat);
        tpf_srvcSendResponse(http_token, &response, 0);

        return;    //or tpf_srvcWaitForRequest() could have been called
}
```

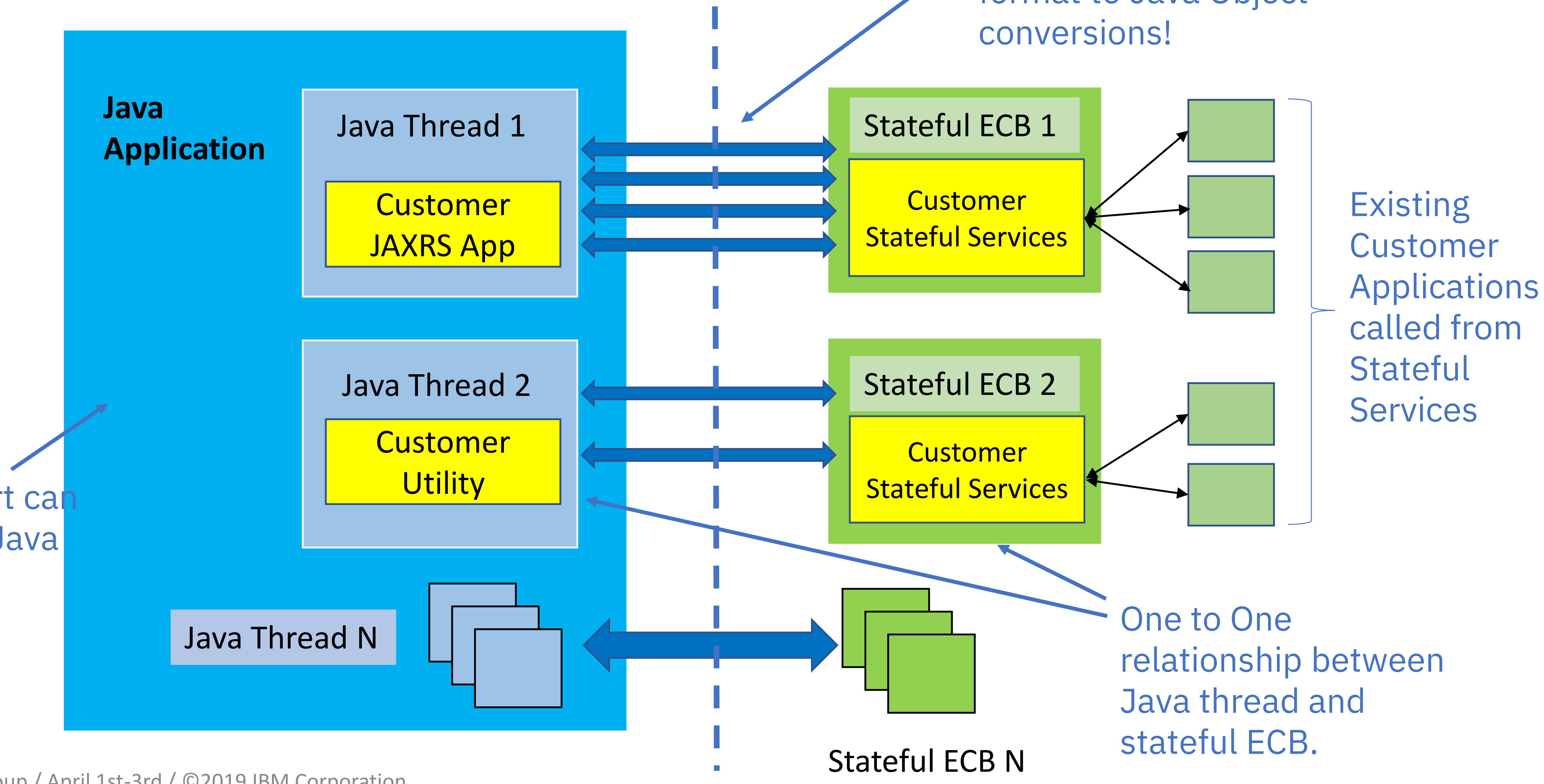
Take advantage of C++ globally scoped Objects. Will automatically run destructors reclaiming resources during exit!

Looks just like a stateless service provider interface.

Next stateful request will have same entry point.

Return does an Enter/Drop not an Exit! Stack is reset. But other resources maintain state, including static storage, malloc storage, locks, etc.

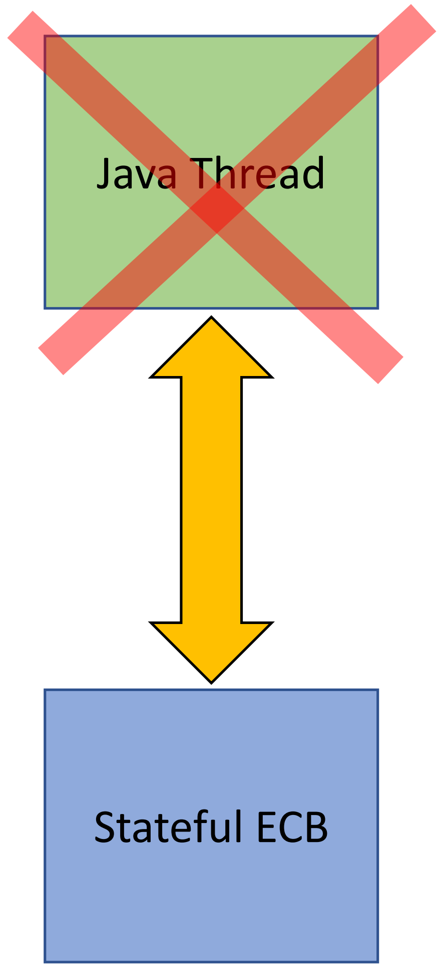
Stateful Architecture


IBM handles plumbing, including Native Structure format to Java Object conversions!
Java Application
Java Thread 1
Customer JAXRS App
Java Thread 2
Customer Utility
Java Thread N
Stateful ECB 1
Customer Stateful Services
Stateful ECB 2
Customer Stateful Services
Stateful ECB N
Existing Customer Applications called from Stateful Services
One to One relationship between Java thread and stateful ECB.
Stateful support can work with any Java application or within a JAM.

Stateful Error Handling

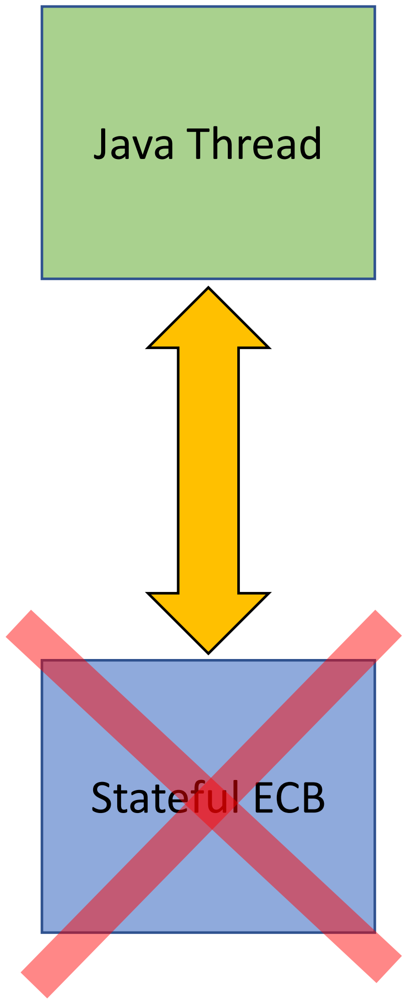
- Java process unexpectedly fails
  - z/TPF system errors and Java dumps are taken
  - Exit processing notifies Stateful ECB

- Stateful ECB will exit quietly
  - z/TPF system errors from Stateful ECB will be suppressed



Stateful Error Handling

- Stateful ECB unexpectedly fails
  - z/TPF system errors are taken as normal
  - Exit processing notifies Java Thread
- Java Thread receives a Status Code Error
  - Stateful invocation must handle error code (application cleanup)
  - The next stateful API issued by the Java thread will create a new stateful ECB



Recap

Apar
PJ45433 (Put 15)

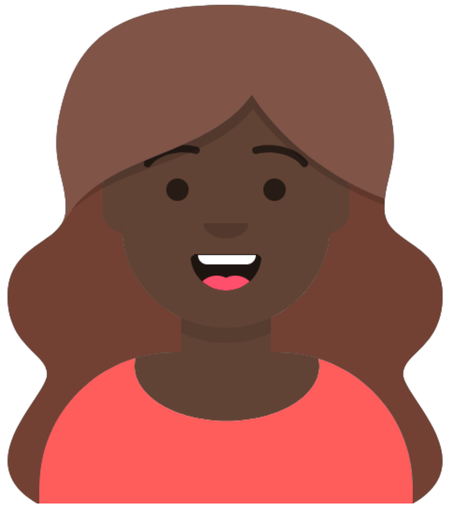
Sample z/TPF Java Stateful Driver
https://www-01.ibm.com/support/docview.wss?uid=ibm10791777

Knowledge Center
https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.15/gtpa2/javatotpf.html
https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.15/gtps6/addprogconsider4stateful.html

Problem

How can we easily get insight into a Java Application running on z/TPF?

Users



“I’m excited about the new Java applications being added to our z/TPF production system and look forward to learning what standard Java tooling exists for Java monitoring.”

Carol
Coverage Programmer

“It was very easy to write this JAX-RS application and I expect to resolve runtime issues before production fairly easily.”

Andrew
New hire Java application programmer

Pain Points

Carol
Coverage Programmer

- Carol wants to **easily view basic information from a Java application without impacting the system in order to pass information to support**

…but **no real-time visual tooling currently exists with that capability for z/TPF.**

Pain Points

Andrew
New hire application programmer

- Andrew can't **determine the performance profile of his Java application ahead of production** because **current tools don't have that capability.**

- He also can't **resolve memory issues in his Java Application** because **current tools are not easy to use and are not capable of pinpointing the caller of the problematic allocations.**

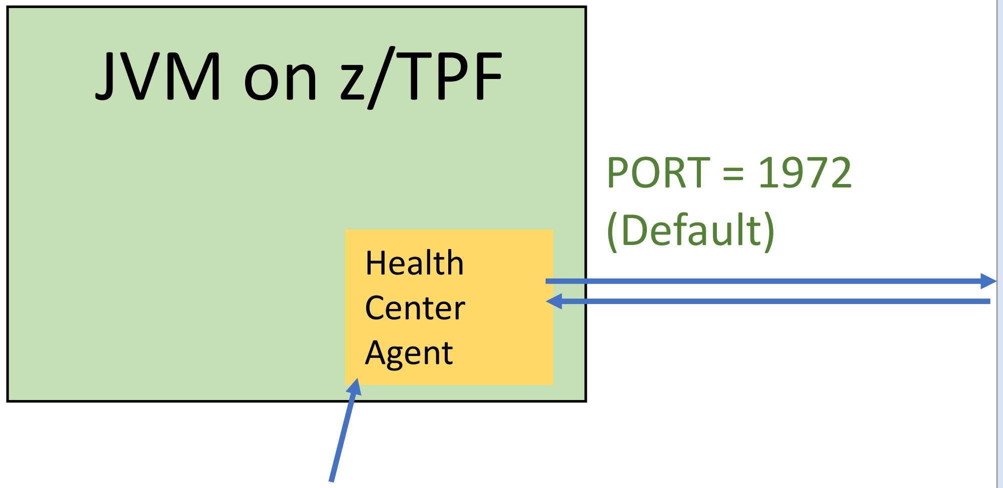
Support for IBM Health Center

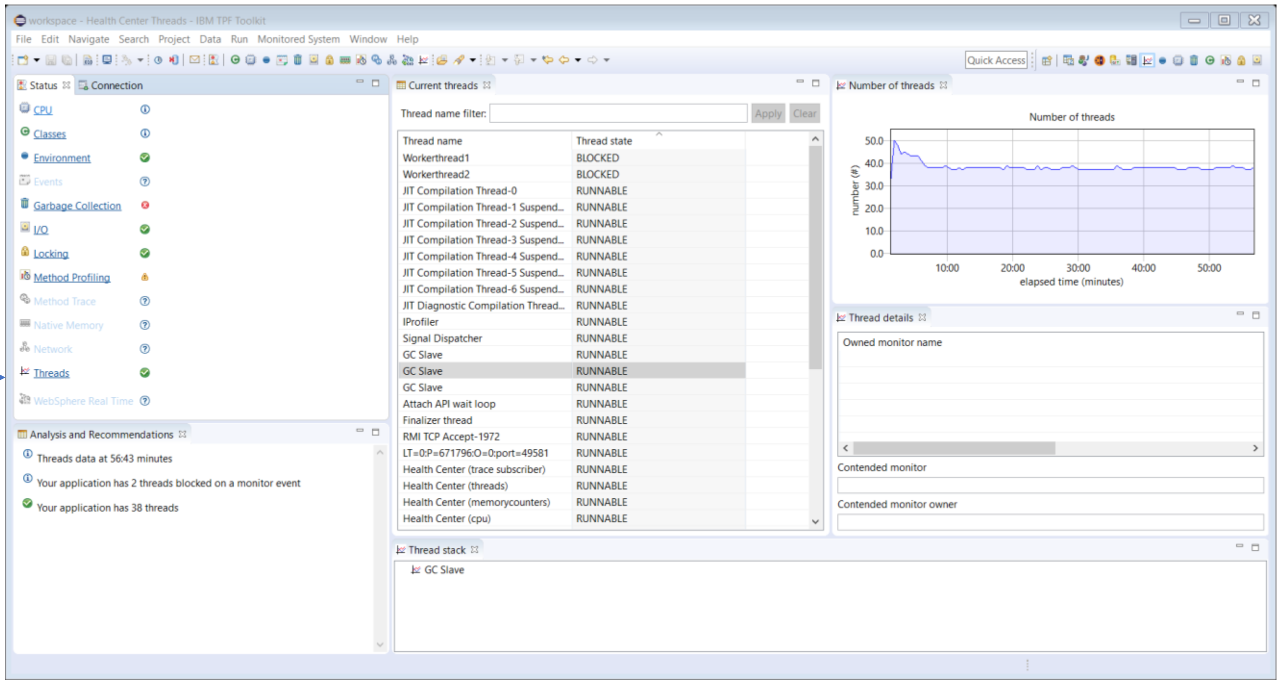
Can identify performance, bottleneck, and garbage collection issues before going into production

- Health Center can attach to an already running Java application.
- Can be used in production with no impact to performance.
- Defines who is allowed to connect via Health Center to that z/TPF production system

PJ45580

Health Center Introduction




workspace - Health Center Threads - IBM TPF Toolkit

File Edit Navigate Search Project Data Run Monitored System Window Help

Quick Access

Status | Connection

CPU	ⓘ
Classes	ⓘ
Environment	✓
Events	?
Garbage Collection	✗
I/O	✓
Locking	✓
Method Profiling	⚠
Method Trace	?
Native Memory	?
Network	?
Threads	✓
WebSphere Real Time	?

Analysis and Recommendations

- ⓘ Threads data at 56:43 minutes
- ⓘ Your application has 2 threads blocked on a monitor event
- ✓ Your application has 38 threads

Current threads

Thread name filter: [Apply] [Clear]

Thread name	Thread state
Workerthread1	BLOCKED
Workerthread2	BLOCKED
JIT Compilation Thread-0	RUNNABLE
JIT Compilation Thread-1 Suspend...	RUNNABLE
JIT Compilation Thread-2 Suspend...	RUNNABLE
JIT Compilation Thread-3 Suspend...	RUNNABLE
JIT Compilation Thread-4 Suspend...	RUNNABLE
JIT Compilation Thread-5 Suspend...	RUNNABLE
JIT Compilation Thread-6 Suspend...	RUNNABLE
JIT Diagnostic Compilation Thread...	RUNNABLE
IProfiler	RUNNABLE
Signal Dispatcher	RUNNABLE
GC Slave	RUNNABLE
GC Slave	RUNNABLE
GC Slave	RUNNABLE
Attach API wait loop	RUNNABLE
Finalizer thread	RUNNABLE
RMI TCP Accept-1972	RUNNABLE
LT=0:P=671796:O=0:port=49581	RUNNABLE
Health Center (trace subscriber)	RUNNABLE
Health Center (threads)	RUNNABLE
Health Center (memorycounters)	RUNNABLE
Health Center (cpu)	RUNNABLE

Number of threads

Thread details

Owned monitor name

Contended monitor

Contended monitor owner

Thread stack

GC Slave

Health Center Client

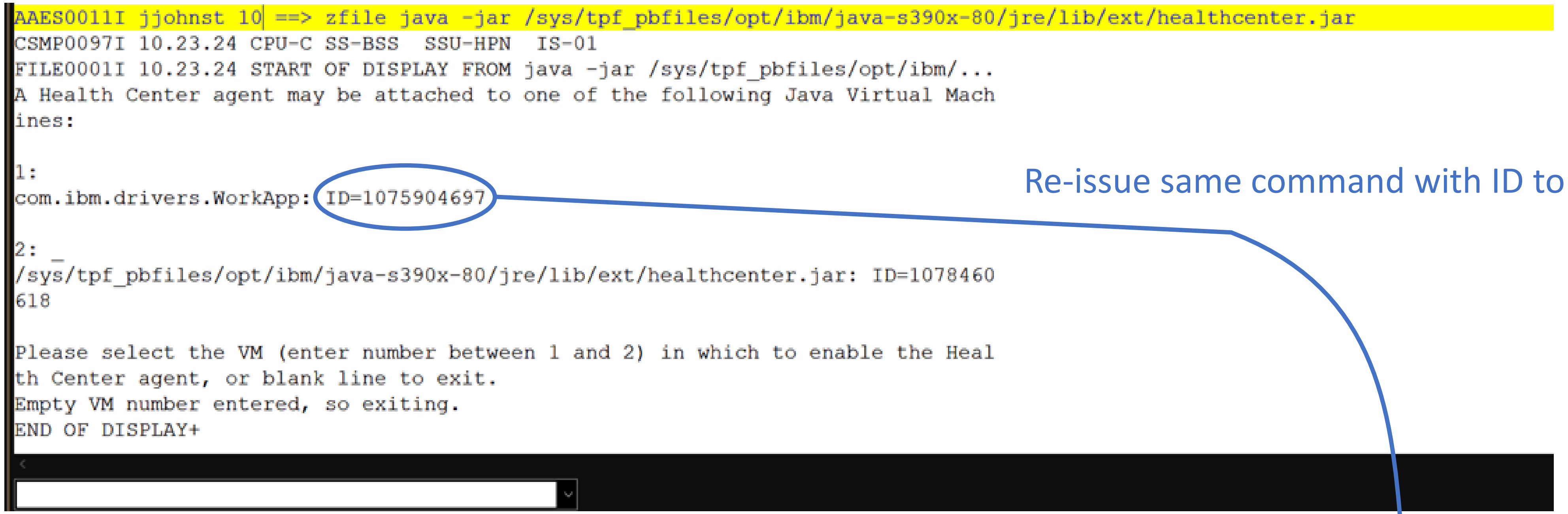
Starting Health Center Agent Options:

1) For 1 JVM:
   add -Xhealthcenter to command line
2) For All JVMs on System:
   add -Xhealthcenter to options.default file or IBM_JAVA_OPTIONS environment variable
3) For JVMs already running use the late attach feature

Health Center Late Attach

Will list all active Java Applications!

zfile java -jar /sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/ext/healthcenter.jar



```
AAES0011I jjohnst 10 ==> zfile java -jar /sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/ext/healthcenter.jar
CSMP0097I 10.23.24 CPU-C SS-BSS  SSU-HPN  IS-01
FILE0001I 10.23.24 START OF DISPLAY FROM java -jar /sys/tpf_pbfiles/opt/ibm/...
A Health Center agent may be attached to one of the following Java Virtual Mach
ines:

1:
com.ibm.drivers.WorkApp: ID=1075904697

2: _
/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/ext/healthcenter.jar: ID=1078460
618

Please select the VM (enter number between 1 and 2) in which to enable the Heal
th Center agent, or blank line to exit.
Empty VM number entered, so exiting.
END OF DISPLAY+
```

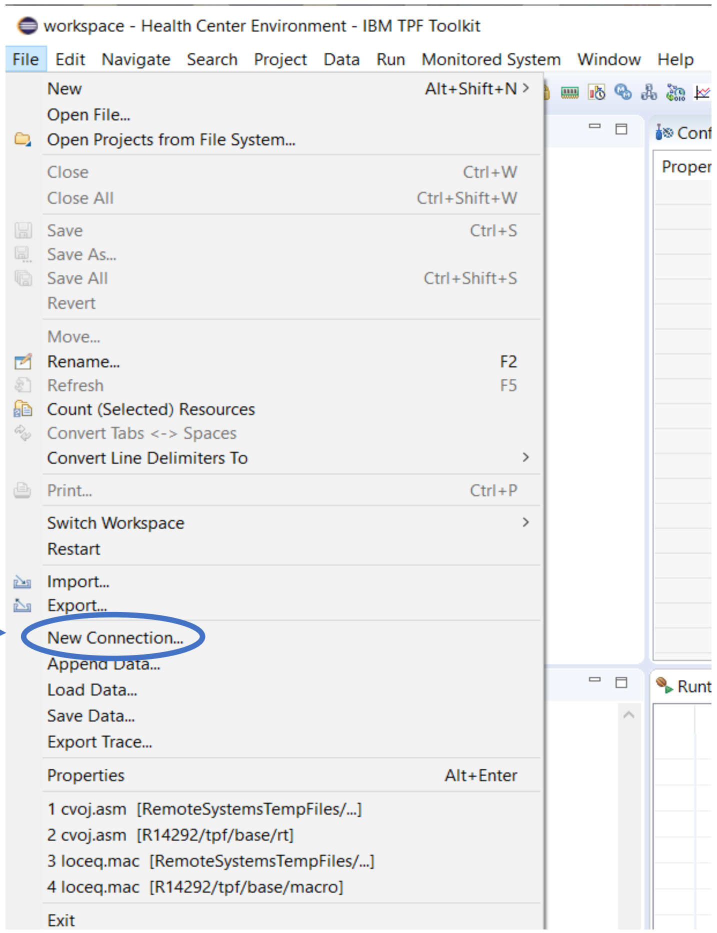
Re-issue same command with ID to attach!

zfile java -jar /sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/ext/healthcenter.jar ID=1075904697

Successful Message:
Successfully enabled Health Center agent in VM: 1075904697…

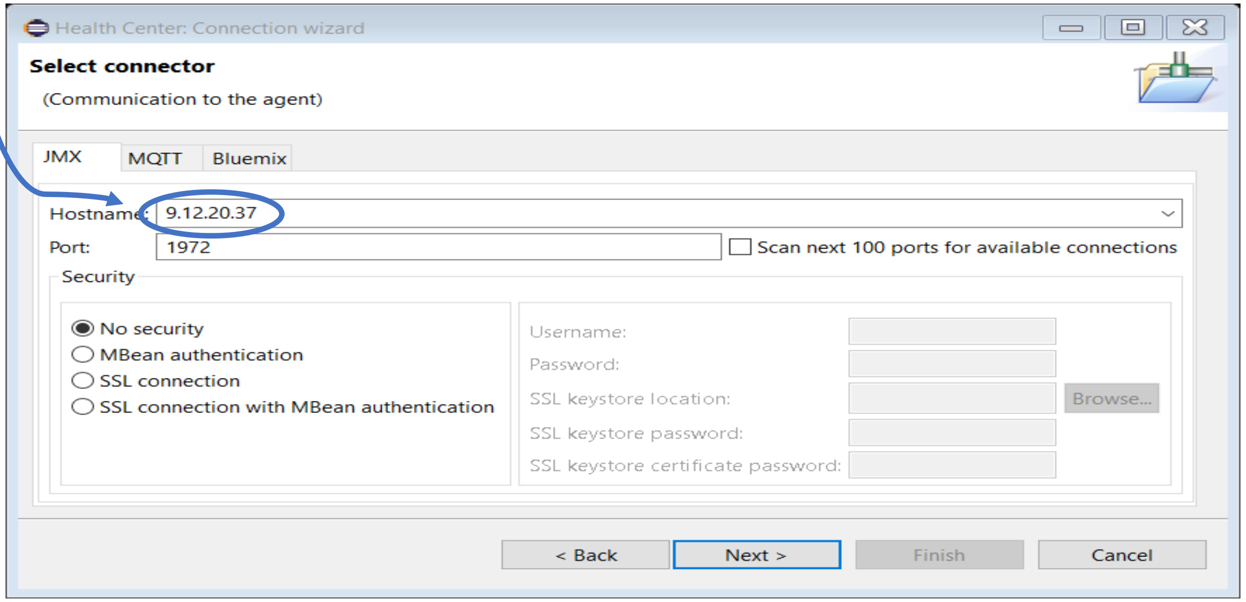
Health Center Client Start

To begin… from Eclipse open any HC view. then File…



workspace - Health Center Environment - IBM TPF Toolkit
File Edit Navigate Search Project Data Run Monitored System Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Count (Selected) Resources
Convert Tabs <-> Spaces
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
New Connection...
Append Data...
Load Data...
Save Data...
Export Trace...
Properties Alt+Enter
1 cvoj.asm [RemoteSystemsTempFiles/...]
2 cvoj.asm [R14292/tpf/base/rt]
3 loceq.mac [RemoteSystemsTempFiles/...]
4 loceq.mac [R14292/tpf/base/macro]
Exit

Specify IP address of z/TPF system:



Health Center: Connection wizard

Select connector

(Communication to the agent)

JMX | MQTT | Bluemix

Hostname: 9.12.20.37

Port: 1972

☐ Scan next 100 ports for available connections

Security

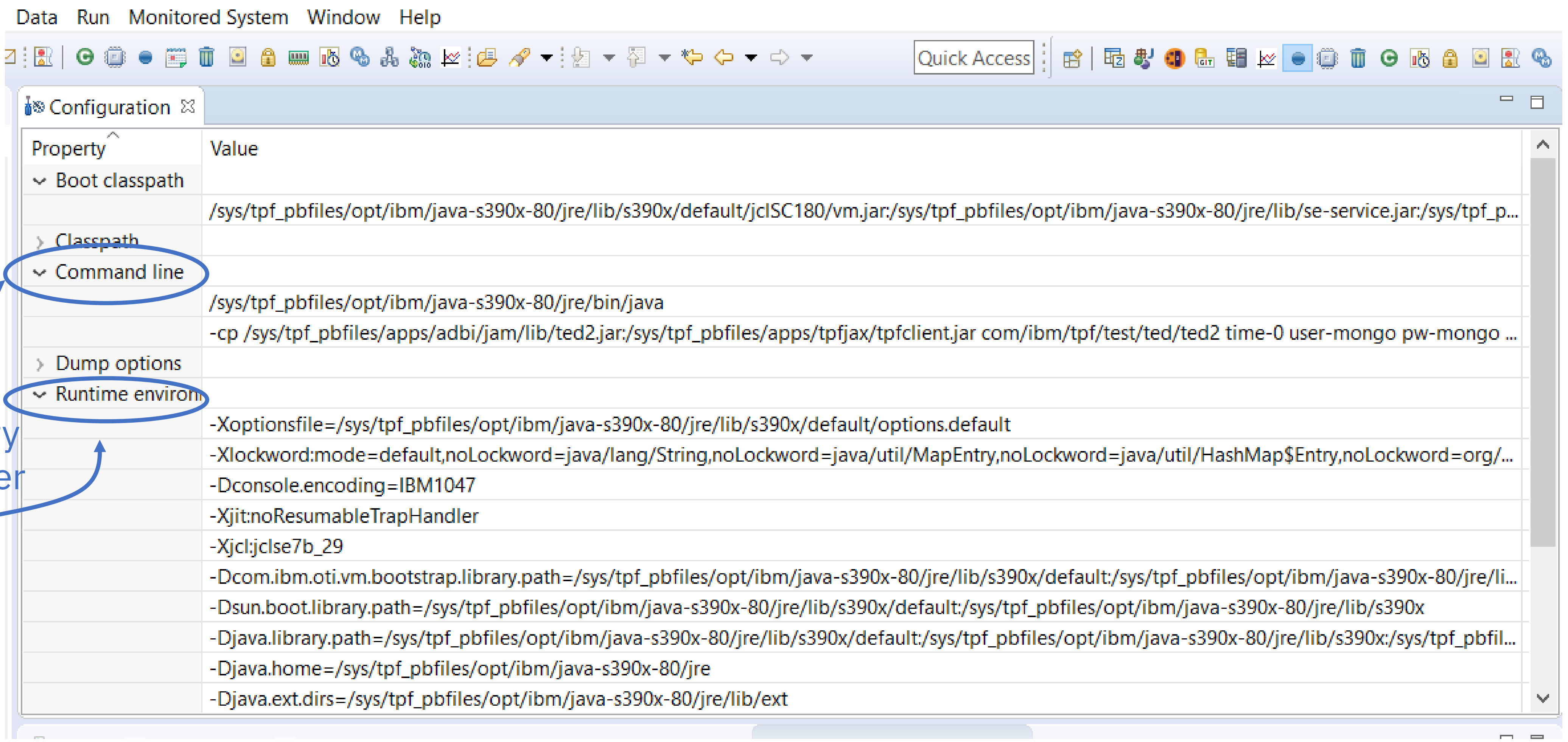
- [✓ No security]
- MBean authentication
- SSL connection
- SSL connection with MBean authentication

Username:
Password:
SSL keystore location:
SSL keystore password:
SSL keystore certificate password:

Health Center Environment View

Discover exact command used to launch Java application

Discover property settings and other default settings.

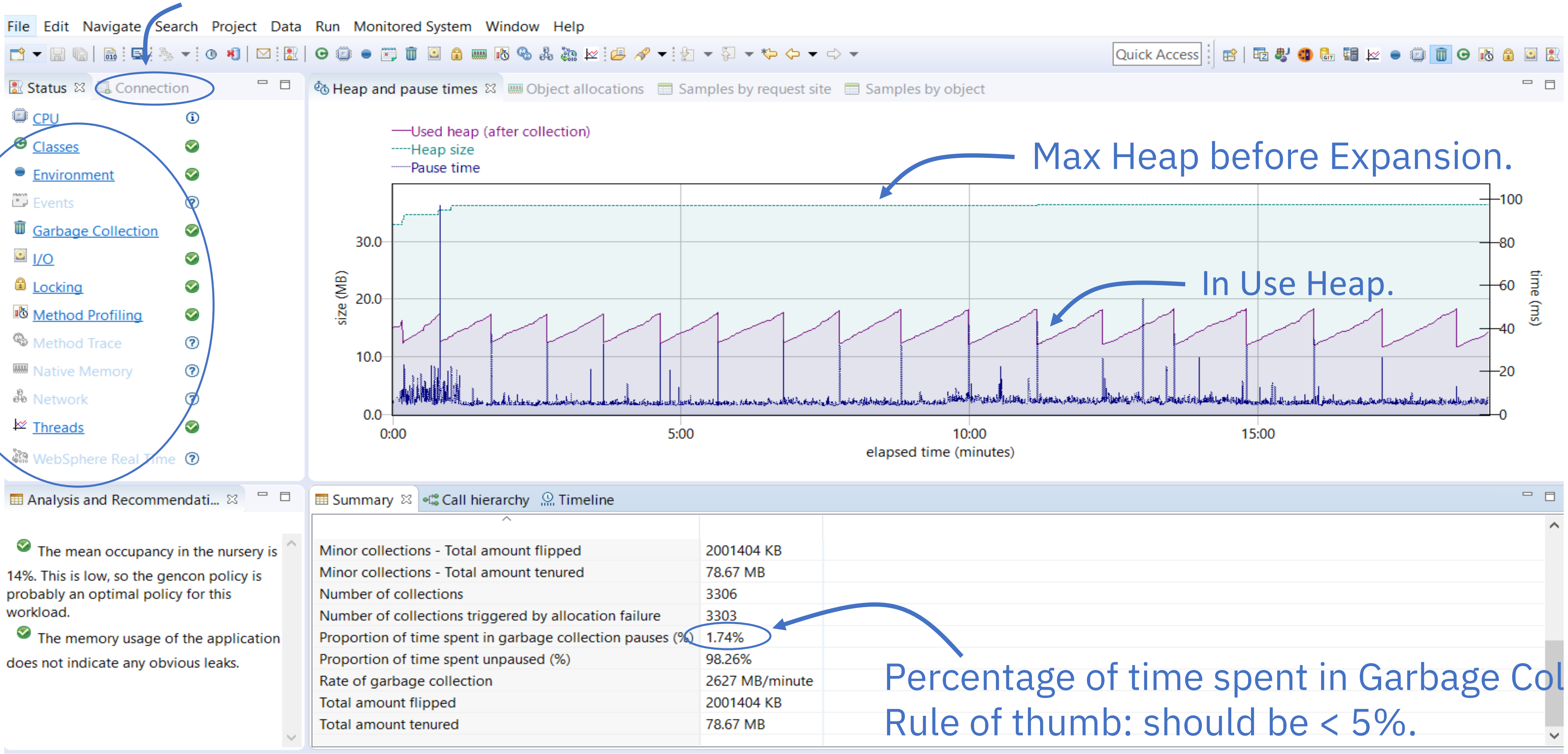


Property	Value
Boot classpath	
	/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x/default/jclSC180/vm.jar:/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/se-service.jar:/sys/tpf_p...
Classpath	
Command line	
	/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/bin/java
	-cp /sys/tpf_pbfiles/apps/adbi/jam/lib/ted2.jar:/sys/tpf_pbfiles/apps/tpfjax/tpfclient.jar com/ibm/tpf/test/ted/ted2 time-0 user-mongo pw-mongo ...
Dump options	
Runtime environ	
	-Xoptionsfile=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x/default/options.default
	-Xlockword:mode=default,noLockword=java/lang/String,noLockword=java/util/MapEntry,noLockword=java/util/HashMap$Entry,noLockword=org/...
	-Dconsole.encoding=IBM1047
	-Xjit:noResumableTrapHandler
	-Xjcl:jclse7b_29
	-Dcom.ibm.oti.vm.bootstrap.library.path=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x/default:/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/li...
	-Dsun.boot.library.path=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x/default:/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x
	-Djava.library.path=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x/default:/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/s390x:/sys/tpf_pbfil...
	-Djava.home=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre
	-Djava.ext.dirs=/sys/tpf_pbfiles/opt/ibm/java-s390x-80/jre/lib/ext

Health Center GC View

Connection Tab easily tells us if Health Center is still connected to the Java Application.

Status Tab facilitates switching between different HC views.



The mean occupancy in the nursery is 14%. This is low, so the gencon policy is probably an optimal policy for this workload.

The memory usage of the application does not indicate any obvious leaks.

Minor collections - Total amount flipped	2001404 KB
Minor collections - Total amount tenured	78.67 MB
Number of collections	3306
Number of collections triggered by allocation failure	3303
Proportion of time spent in garbage collection pauses (%)	1.74%
Proportion of time spent unpaused (%)	98.26%
Rate of garbage collection	2627 MB/minute
Total amount flipped	2001404 KB
Total amount tenured	78.67 MB

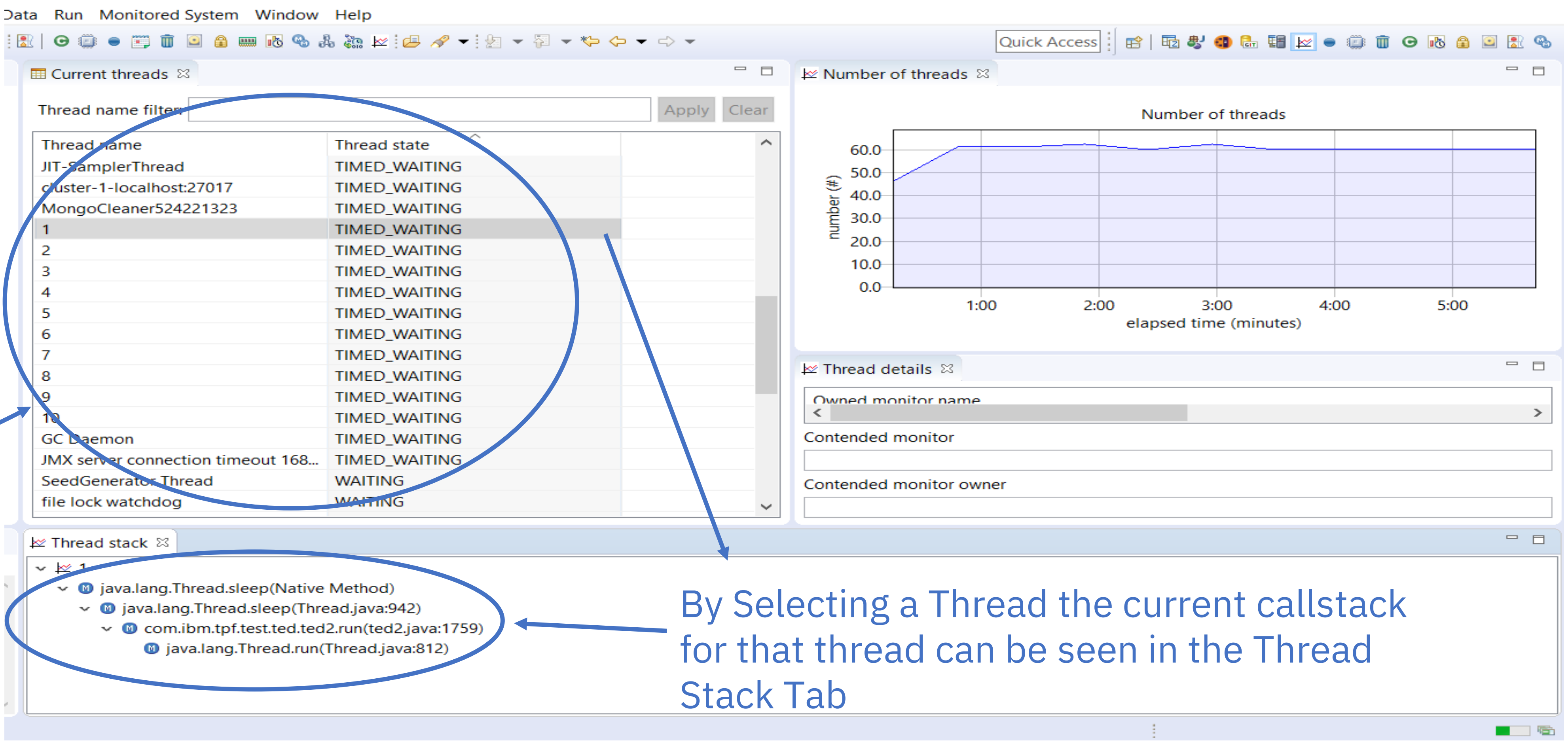
Max Heap before Expansion.

In Use Heap.

Percentage of time spent in Garbage Collection.
Rule of thumb: should be < 5%.

Health Center Thread View

Java application thread list. Includes Thread Name & State.



Thread name	Thread state
JIT-SamplerThread	TIMED_WAITING
cluster-1-localhost:27017	TIMED_WAITING
MongoCleaner524221323	TIMED_WAITING
1	TIMED_WAITING
2	TIMED_WAITING
3	TIMED_WAITING
4	TIMED_WAITING
5	TIMED_WAITING
6	TIMED_WAITING
7	TIMED_WAITING
8	TIMED_WAITING
9	TIMED_WAITING
10	TIMED_WAITING
GC Daemon	TIMED_WAITING
JMX server connection timeout 168...	TIMED_WAITING
SeedGenerator Thread	WAITING
file lock watchdog	WAITING

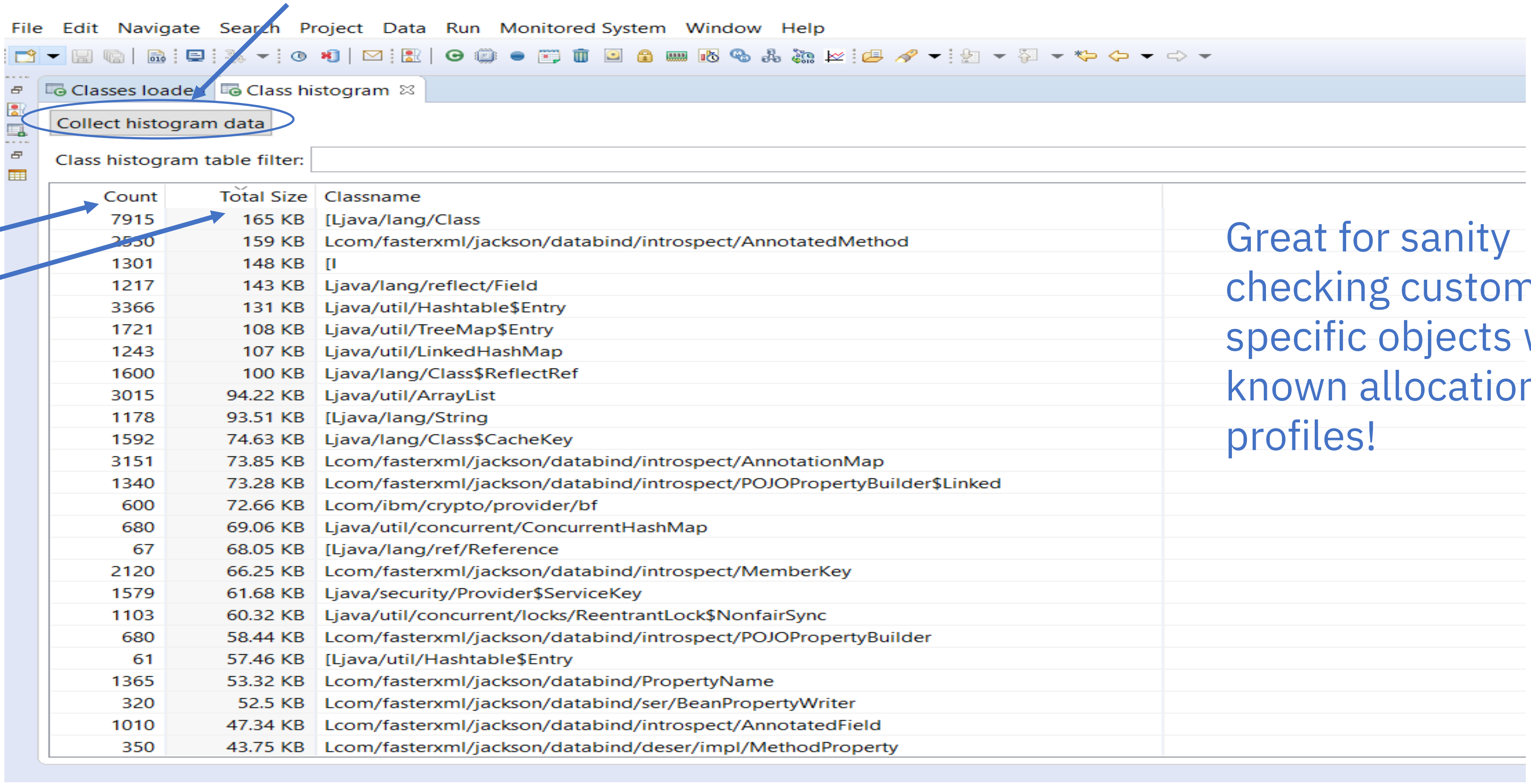
Number of threads: y-axis number (#) 0.0–60.0; x-axis elapsed time (minutes) 1:00–5:00.

```
1
  java.lang.Thread.sleep(Native Method)
    java.lang.Thread.sleep(Thread.java:942)
      com.ibm.tpf.test.ted.ted2.run(ted2.java:1759)
        java.lang.Thread.run(Thread.java:812)
```

By Selecting a Thread the current callstack for that thread can be seen in the Thread Stack Tab

Health Center Class Histogram

Provides a snapshot of all Java Object Instances and total bytes consumed!



File Edit Navigate Search Project Data Run Monitored System Window Help
Classes loaded
Class histogram
Collect histogram data
Class histogram table filter:

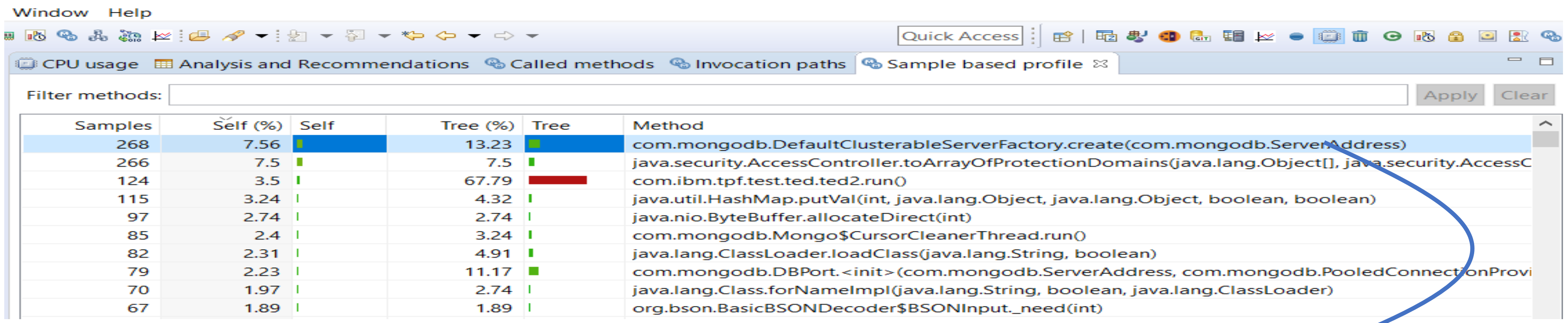
Count	Total Size	Classname
7915	165 KB	[Ljava/lang/Class
2530	159 KB	Lcom/fasterxml/jackson/databind/introspect/AnnotatedMethod
1301	148 KB	[I
1217	143 KB	Ljava/lang/reflect/Field
3366	131 KB	Ljava/util/Hashtable$Entry
1721	108 KB	Ljava/util/TreeMap$Entry
1243	107 KB	Ljava/util/LinkedHashMap
1600	100 KB	Ljava/lang/Class$ReflectRef
3015	94.22 KB	Ljava/util/ArrayList
1178	93.51 KB	[Ljava/lang/String
1592	74.63 KB	Ljava/lang/Class$CacheKey
3151	73.85 KB	Lcom/fasterxml/jackson/databind/introspect/AnnotationMap
1340	73.28 KB	Lcom/fasterxml/jackson/databind/introspect/POJOPropertyBuilder$Linked
600	72.66 KB	Lcom/ibm/crypto/provider/bf
680	69.06 KB	Ljava/util/concurrent/ConcurrentHashMap
67	68.05 KB	[Ljava/lang/ref/Reference
2120	66.25 KB	Lcom/fasterxml/jackson/databind/introspect/MemberKey
1579	61.68 KB	Ljava/security/Provider$ServiceKey
1103	60.32 KB	Ljava/util/concurrent/locks/ReentrantLock$NonfairSync
680	58.44 KB	Lcom/fasterxml/jackson/databind/introspect/POJOPropertyBuilder
61	57.46 KB	[Ljava/util/Hashtable$Entry
1365	53.32 KB	Lcom/fasterxml/jackson/databind/PropertyName
320	52.5 KB	Lcom/fasterxml/jackson/databind/ser/BeanPropertyWriter
1010	47.34 KB	Lcom/fasterxml/jackson/databind/introspect/AnnotatedField
350	43.75 KB	Lcom/fasterxml/jackson/databind/deser/impl/MethodProperty

Can sort by total instance count or by total bytes for given Class!

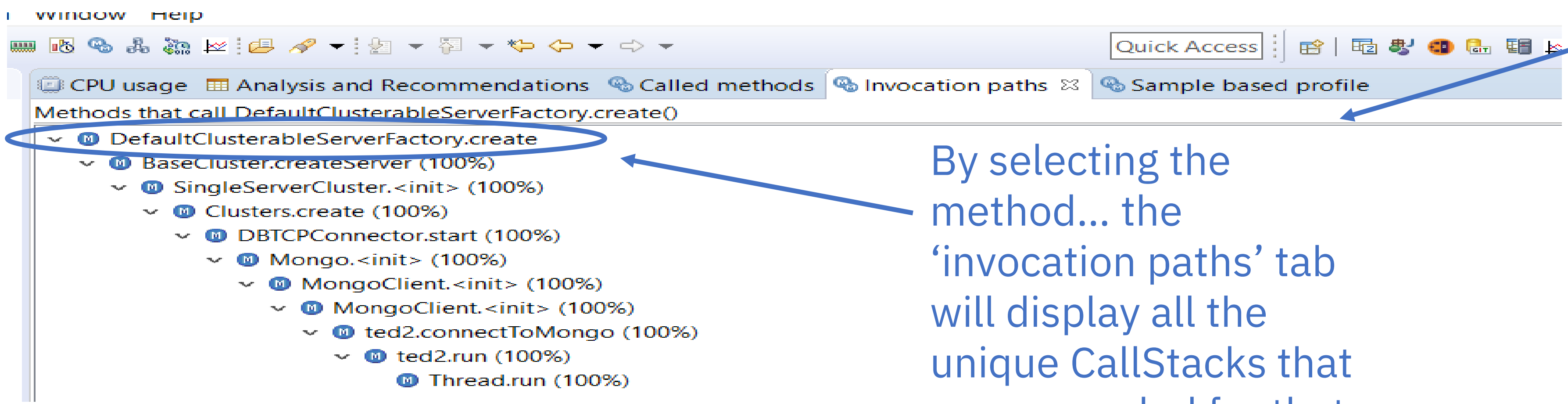
Great for sanity checking customer specific objects with known allocation profiles!

Health Center Profiling View

Great for figuring out what Java methods are consuming the most cycles relative to the Java application itself!



Samples	Self (%)	Self	Tree (%)	Tree	Method
268	7.56	🟩	13.23	🟩	com.mongodb.DefaultClusterableServerFactory.create(com.mongodb.ServerAddress)
266	7.5	🟩	7.5	🟩	java.security.AccessController.toArrayOfProtectionDomains(java.lang.Object[], java.security.AccessC
124	3.5	🟩	67.79	🟥	com.ibm.tpf.test.ted.ted2.run()
115	3.24	🟩	4.32	🟩	java.util.HashMap.putVal(int, java.lang.Object, java.lang.Object, boolean, boolean)
97	2.74	🟩	2.74	🟩	java.nio.ByteBuffer.allocateDirect(int)
85	2.4	🟩	3.24	🟩	com.mongodb.Mongo$CursorCleanerThread.run()
82	2.31	🟩	4.91	🟩	java.lang.ClassLoader.loadClass(java.lang.String, boolean)
79	2.23	🟩	11.17	🟩	com.mongodb.DBPort.<init>(com.mongodb.ServerAddress, com.mongodb.PooledConnectionProvi
70	1.97	🟩	2.74	🟩	java.lang.Class.forNameImpl(java.lang.String, boolean, java.lang.ClassLoader)
67	1.89	🟩	1.89	🟩	org.bson.BasicBSONDecoder$BSONInput._need(int)



Methods that call DefaultClusterableServerFactory.create()

- DefaultClusterableServerFactory.create
  - BaseCluster.createServer (100%)
    - SingleServerCluster.<init> (100%)
      - Clusters.create (100%)
        - DBTCPConnector.start (100%)
          - Mongo.<init> (100%)
            - MongoClient.<init> (100%)
              - MongoClient.<init> (100%)
                - ted2.connectToMongo (100%)
                  - ted2.run (100%)
                    - Thread.run (100%)

By selecting the method… the 'invocation paths' tab will display all the unique CallStacks that were sampled for that method!

Health Center Locking View

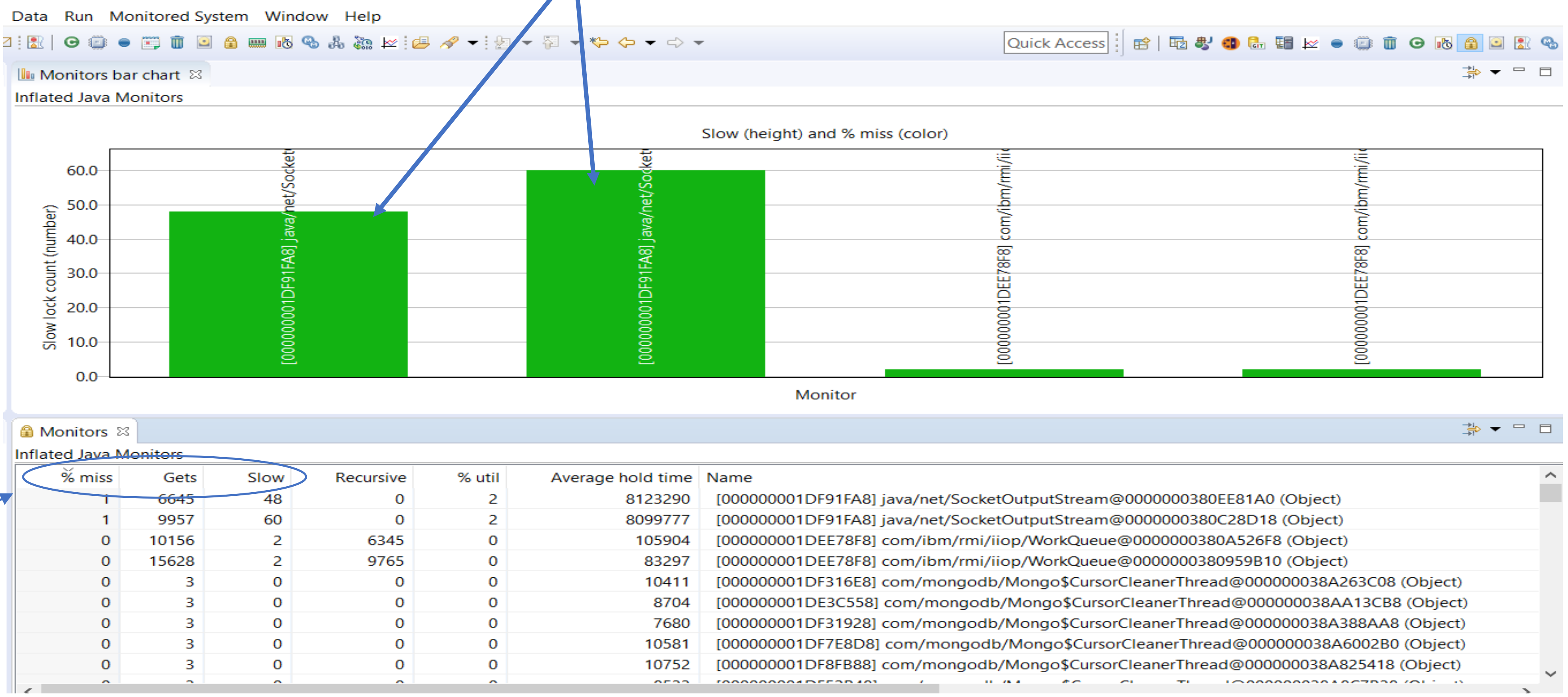
Will change to yellow/red if attention is required.

Great for figuring out Java application scoped locking profiles.

%miss = slow/gets

Slow = # of times had to wait before getting lock

Gets = attempts



Data
Run
Monitored System
Window
Help
Quick Access
Monitors bar chart
Inflated Java Monitors
Slow (height) and % miss (color)
Slow lock count (number)
Monitor
[000000001DF91FA8] java/net/Socket
[000000001DF91FA8] java/net/Socket
[000000001DEE78F8] com/ibm/rmi/iio
[000000001DEE78F8] com/ibm/rmi/iio
Monitors

Inflated Java Monitors

% miss	Gets	Slow	Recursive	% util	Average hold time	Name
1	6645	48	0	2	8123290	[000000001DF91FA8] java/net/SocketOutputStream@0000000380EE81A0 (Object)
1	9957	60	0	2	8099777	[000000001DF91FA8] java/net/SocketOutputStream@0000000380C28D18 (Object)
0	10156	2	6345	0	105904	[000000001DEE78F8] com/ibm/rmi/iiop/WorkQueue@0000000380A526F8 (Object)
0	15628	2	9765	0	83297	[000000001DEE78F8] com/ibm/rmi/iiop/WorkQueue@0000000380959B10 (Object)
0	3	0	0	0	10411	[000000001DF316E8] com/mongodb/Mongo$CursorCleanerThread@000000038A263C08 (Object)
0	3	0	0	0	8704	[000000001DE3C558] com/mongodb/Mongo$CursorCleanerThread@000000038AA13CB8 (Object)
0	3	0	0	0	7680	[000000001DF31928] com/mongodb/Mongo$CursorCleanerThread@000000038A388AA8 (Object)
0	3	0	0	0	10581	[000000001DF7E8D8] com/mongodb/Mongo$CursorCleanerThread@000000038A6002B0 (Object)
0	3	0	0	0	10752	[000000001DF8FB88] com/mongodb/Mongo$CursorCleanerThread@000000038A825418 (Object)

Recap

Apar

PJ45580 (March 2019)

Health Center Client

https://marketplace.eclipse.org/content/ibm-monitoring-and-diagnostic-tools-health-center

Knowledge Center (Health Center User Guide)

https://www.ibm.com/support/knowledgecenter/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/introduction.html

Sponsor Users

- Get involved!

- Email:
jjohnst@us.ibm.com
dgritter@us.ibm.com

Thank You!

Questions or Comments?

Trademarks