System Evaluator

Project Work

SUBMITTED BY

GIRISH R.
KARTHIKEYAN N.
KIRAN KUMAR B.
RAVINDRANATH M.S.

UNDER THE GUIDANCE OF

Prof. S.Thangasamy B.E. (Hons), Ph.D.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF
THE DEGREE OF
BACHELOR OF ENGINEERING IN
COMPUTER SCIENCE AND ENGINEERING
OF THE BHARATHIAR UNIVERSITY,
COIMBATORE.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Kumaraguru College of Technology

(Affiliated to Bharathiar University)
COIMBATORE 641 006
MARCH 2002

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University, Coimbatore)

CERTIFICATE

THIS IS TO CERTIFY THAT PROJECT WORK ENTITLED

SYSTEM EVALUATOR

is a bona fide record of work done by

GIRISH R.	**9827K0174**
KARTHIKEYAN N.	**9827K0182**
KIRAN KUMAR B.	**9827K0183**
RAVINDRANATH M.S.	**9827K0209**

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING

DURING THE ACADEMIC YEAR 2001-02

13/3/02

Head Of the Department

13/3/02

Staff-in-charge

Submitted for the University Examination held on ________

Internal Examiner

External Examiner

Place: Coimbatore
Date:

We, **Girish R., Karthikeyan N., Kiran Kumar B., Ravindranath M.S.** here by declare that this project entitled "**SYSTEM EVALUATOR**" submitted to **Kumaraguru College of Technology, Coimbatore** (Affiliated to Bharathiar University) is a record of original work done by us under the supervision and guidance of **Prof. S. Thangasamy Ph.D.**, Department of Computer Science & Engineering.

NAME	REGISTRATION NUMBER	SIGNATURE
R. Girish	9827K0174	
N. Karthikeyan	9827K0182	Karthikeyan
B. Kiran Kumar	9827K0183	B.Kirankumar
M.S. Ravindranath	9827K0209	

Countersigned: 13/3/02

Project guide: **Prof. S. Thangasamy B.E. (Hons), Ph.D.,**
Head of the Department,
Department of Computer Science & Engineering,
Kumaraguru College of Technology,
Coimbatore.

Place: Coimbatore.
Date:

We are extremely thankful to **Dr.K.K.Padmanaban B.Sc (Engg.), M.Tech, Ph.D**, Principal, Kumaraguru College of Technology who has given us his valuable guidance and an efficient infrastructure that aided us during the course of our education.

With profound sense of gratitude and regards, we acknowledge with great pleasure the guidance and support extended by **Prof. S.Thangasamy B.E. (Hons), Ph.D.,** the Head of the Department of Computer Science & Engineering and our project guide, for his valuable and continuous guidance and persistent encouragement.

We express our heartfelt gratitude to **Mrs. S.Devaki M.S** Assistant Professor who has motivated and inspired us towards the completion of our project. We would also like to thank **Mr. Karthicraja** and **Ms. Uma Maheshwari** of TBN for extending their valuable support in all our endeavors. We like to express our special thanks to all the staffs and lab technicians in the Department of Computer Science and Engineering who helped us for the successful completion of the project.

Finally, we express our deep sense of gratitude to our parents, friends, roommates and all other persons involved directly or indirectly with this project for their invaluable help and consideration towards us.

Java is a small, simple, safe, object-oriented, interpreted or dynamically optimized, byte-coded, architecture-neutral, garbage-collected, multithreaded programming language with a strongly typed exception-handling mechanism for writing distributed, dynamically extensible programs. Analyzing such applications is a complex task and very few console based performance evaluators are available. But still there are no graphical packages for analyzing the performance of an application created in Java under a PC system.

The evaluator we develop, determines various characteristics of an java executable program (application) such as number of threads under execution and their status, number of classes loaded during the execution along with the number of instances created for each class, heap memory environment under which the application is executed, garbage collection activity and a focus on objects created within the user defined classes in an application.

Various such characteristics are stored in a data store and a graphical representation in the form of graphs and tables are displayed for better understanding. This allows the application developer to have a running commentary of his application. It is useful for a Java developer to evaluate the performance of the application in a particular system or to compare different versions of the same application by evaluating every version of the application.

CONTENTS

Existing System

Currently Java provides support for application evaluation in three ways.

1. Reflection package
2. – prof option in the compiler
3. JVMPI (Java Virtual Machine Profiler Interface)

Our evaluator implements the third method for the purpose of evaluation. In the reflection package the source code of the application receives the information regarding the application and there has to be built in handler within the application for processing the obtained information. This retards the generality of the application because a separate evaluating module has to be written for each and every application by the developer along with the development of the application.

The second type provided huge amount of information at the console command prompt but is not programmable thereby forcing developers to leave that option out. The name profiler is synonymous with the evaluator we develop.

JDK (Java Development Kit) provided by Sun Microsystems, has a sample profiler named HPROF. The disadvantage of HPROF is that it is a console based profiler without Graphical representation. Also profiling does not take place continuously. Evaluators available in the market are similar to HPROF but can profile continuously.

Proposed System

We have built an evaluator capable of continuous evaluation with the output displayed in the form of graphs and tables. Further every data for display is taken from a data store and the data store is the repository for all the information that is represented. A snapshot of the data store could be obtained in a verbose file format for future reference. The evaluator developed is network compatible. Here the default loop back address is used for evaluating applications in a standby computer. The connection established is through TCP/IP sockets which are acknowledgement based thereby preventing data corruption.

Advantages

- User is not overloaded with chunk of information. User can selectively view the information he wishes to.
- Evaluated results could be viewed in a terminal different from where the application to be evaluated is under execution.
- When it is run on a network, the processing overheads are eliminated since the process of evaluation may consume considerable amount of memory and processing time.

PRODUCT DEFINITION

The system evaluator is software that gives out the selected characteristics of a job under execution in a computer. The evaluator refers to the jobs created through source programs written in java and executed on PCs. This evaluator basically serves as a tool for testing a java application dynamically.

Development tools

MS Visual Studio (For generating DLL's from C source program), JDK 1.2

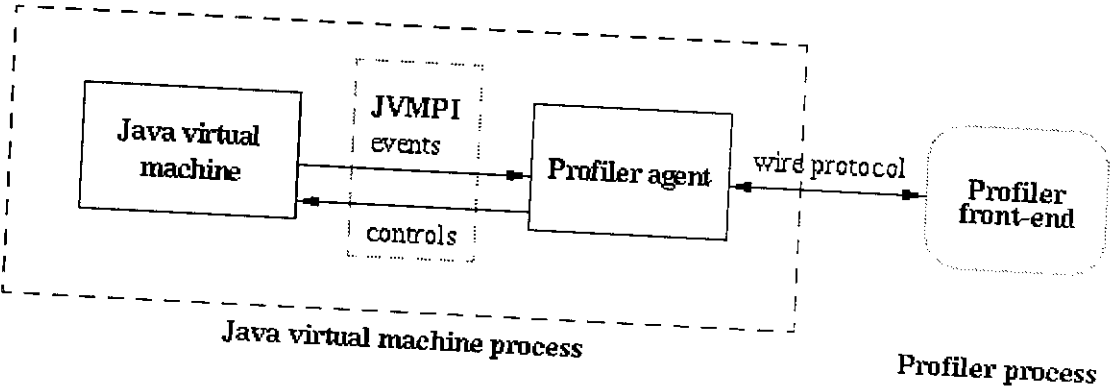
Tools required for execution

- JRE (Java Runtime Environment with support for JVMPI version 1 and swing)
- Operating System - Windows 9x.
- Runtime libraries for Winsock2.

PROJECT PLAN

In the analysis phase of System Evaluator, classes, user interface prototype, other aspects were identified and the ambiguities in them were eliminated. From the results obtained from the analysis phase, the system design was established. In the implementation phase, coding was written for each related module and finally they were integrated. Testing was finally done to verify and validate all the modules individually and as a whole.

The evaluator that we develop is also referred to as profiler agent. JVMPI is a two-way function call interface between the Java virtual machine and an in-process profiler agent. On one hand, the virtual machine notifies the profiler agent of various events, for example, heap allocation, thread start, etc. On the other hand, the profiler agent issues controls and requests for more information through the JVMPI. For example, the profiler agent can turn on/off a specific event notification, based on the needs of the profiler front-end.



The evaluator traps all the critical and most used messages while intercepting the performance of the application.

The front end part does not run in the same process as the profiler agent. There is no wired protocol for communication between the profiler agent and the server that stores the messages into the data store.

Purpose

The purpose of this SRS is to elaborate the requirements of the System Evaluator. Evaluator is the software that would bring out selected characteristics of a job under execution in a computer. This Evaluator refers to the jobs created through source programs written in Java and executed on PCs.

Scope

This SRS focuses mainly on the requirements to be met by the System Evaluator. The well-structured format and descriptive nature of the SRS is aimed at aiding the developers in developing an efficient evaluator.

Overview

Exploring further the Software Requirement Specification gives information about the specific requirements like functionality, usability and supportability of the evaluator.

Overall Description

- In the layman's perspective, the evaluator is a descriptive tool evaluating spatial and temporal parameters such as memory usage and thread time usage of a Java application.

➢For a developer it gives a comprehensive enumeration of various aspects of his code, enabling him to evaluate and optimize his code.

➢Requirements also involve logging Memory usage in terms of KB (Kilo Bytes), logging Processor usage as a function of time, monitoring the methods where these usages take place. Call charts, usage graphs are used to represent the above.

➢The objective of the evaluator ends with listing out the key parameters and it is up to the user to analyze and interpret the output generated.

Specific Requirements

Functionality

When a java application is under the process of execution:

a) The classes loaded are unknown to the user. The number of classes and their names are to be informed to the user.

b) The application contains classes specific to it. Among all the classes loaded those classes specific to that application needs to be informed to the user.

c) Objects are created from classes. The number of instances created from the classes loaded needs to be known to the user.

d) The application-specific object created within a method of the application is identified and all details pertaining to it needs to be reported to the user.

e) Threads are created during the process of execution. The number of threads under execution needs to be reported to the user.

f) The status of the thread may vary from time to time. The current status of the thread is to be reported to the user.

g) Heap memory is allocated for the java jobs. The total heap memory allocated is to be displayed to the user.

h) Out of the total heap memory utilized varies. So the memory that is free from utilization is to be displayed to the user.

i) Garbage collection takes place to free the memory unnecessarily occupied by the objects created. The memory occupied and hence the space recovered is to be reported to the user.

j) The time at which the garbage collection takes place should be displayed.

Supportability

The runtime files generated is platform dependant and a version for every platform must be released. Currently we have developed the evaluator for the Windows 98 Environment.

Design Constraints

- Languages having packages, to analyze the running instances, are imminent.
- Evaluating is limited to one application at a time as the function interface provided by Java is limited to one instance at a time.
- Runtime Environment and support for sockets are mandatory for the evaluator to operate.

Interfaces

User Interfaces

User interface is graphical based for better interpretation. Graphs and charts are created to specify the information in a better fashion.

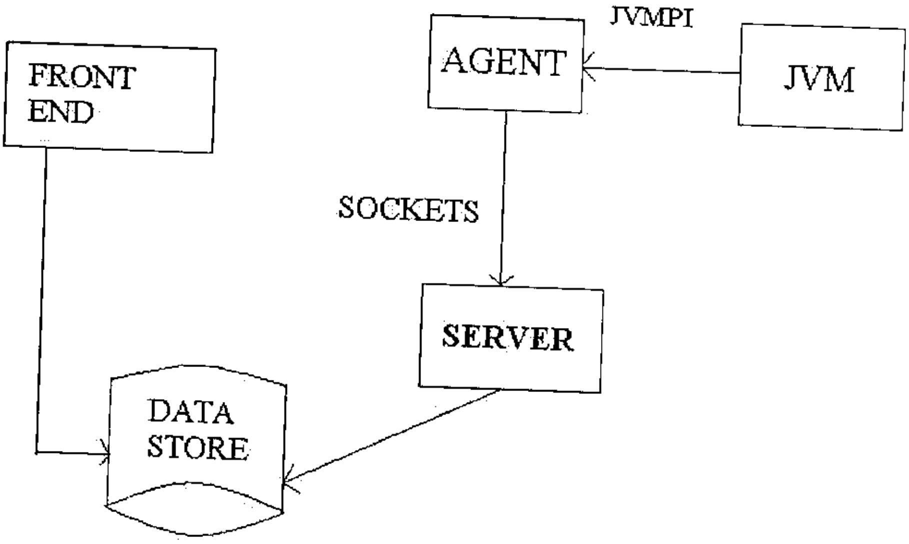
Software Interfaces

The JVMPI (Java Virtual Machine Profiler Interface) is used to obtain critical information from the runtime environment (JVM). JVMPI has certain standard methods and event notifications for interfacing to other applications.

Communication Interfaces

The methods that obtain the information from the JVMPI send all the information to a TCP/IP sockets (logical ports). The front end obtains the data from these sockets and displays them.

This document provides a comprehensive architectural overview of the system. A number of different architectural views are used to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions, which have been made on the system. This document guides the forthcoming implementation and testing activities.



The above diagram shows the design structure with which the entire process takes place. The application to be evaluated executes the instructions through the JVM (Java Virtual Machine). The profiler agent obtains the details from the JVM through JVMPI (Java Virtual Machine Profiler Interface) which is a programmable interface. The agent then sends that information in the form of assembled messages (non standard message format) to the server. The server parses the message to obtain information available in each message and stores the requisite ones in the data store. The front end reads information from the data store and displays them in the form of graphs.

In the current version of JVMPI, only one agent per virtual machine can be supported. Hence not more than one application can be evaluated at a time.

JVMPI issues various events generated during the execution of the application. The events to be notified are to be enabled before they are generated. The events of our concern are:

- JVMPI_EVENT_CLASS_LOAD
- JVMPI_EVENT_CLASS_UNLOAD
- JVMPI_EVENT_THREAD_START
- JVMPI_EVENT_THREAD_END
- JVMPI_EVENT_OBJECT_ALLOC
- JVMPI_EVENT_OBJECT_FREE
- JVMPI_EVENT_GC_FINISH
- JVMPI_EVENT_GC_START
- JVMPI_EVENT_JVM_SHUT_DOWN

When the events are notified they contain event specific information with which the required details could be obtained and then sent to the server.

Further to obtain the status of various threads, a separate thread has been created which, after a definite interval of time, will send the status of all threads to the server.

Information obtained is packed in a non-standard message format and sent through TCP/IP sockets. Different components of the message are

- Header that identifies the type of the message
- Information specific to the message. This is of variable size.
- Other details which should be entered into the data store are attached.

Each individual portion is separated by a '~'. On the server side the entire message is tokenized using the '~' character and individual components of the message are segregated. If any part of the information contains the '~' character then one another '~' will be padded to the next character. Token handling is written to handle these cases and is an extension of the standard String Tokenizer available in Java.

The data store structure used to store the information has the following schema.

CLASS LIST

CLASS ID	The unique identifier of the class while dynamically loaded.
CLASS NAME	The name of the class along with the package preceding its name.
INSTANCE COUNT	The count of instances that is currently available for the specific class.

METHOD TAB

This table contains the methods that are available in the user defined classes only. The constructers are named as <init>.

METHOD ID	The unique identifier of the method while dynamically loaded.
METHOD NAME	The name of the method, the method signature is not specified here.
CLASS ID	The class id under which the method is present. This is refers to the class id of the class list.

THREAD LIST

THREAD ENV	The unique identifier of the thread while dynamically loaded.
THREAD NAME	This field contains the name of the thread.
STATUS	This field represents the current status of the thread. The status of the thread has a specific number associated with it.

THREADTAB.STATUS Value	STATUS OF THE THREAD
1	Thread is runnable.
2	Thread is waiting on a monitor
3	Thread is waiting on a condition variable.

Other than these three values the status may have the SUSPENDED (32768) or INTERRUPTED (16384) bit set.

USEROBJECTLIST

OBJECT ID	Unique identifier of the object.
METHODID	The identifier of the method from where the object was created.
CLASSTYPEID	The class id of the object created.

During the process of development of the evaluator we followed the following testing techniques.

I. Unit Testing

Different units of the project such as profiler agent, the server, the token parser, and the display module were identified and developed separately. Each of them was tested individually with sample inputs and the output was verified with manual calculations.

During the process of testing, we encountered some strange abnormalities in the Java runtime. For example, all the standard objects such as char, int, boolean etc., were registered as a class with a unique class id. But when an instance was created, all the standard class objects had an ID of 0. Further every element was defined as an array even if the element be a separate non-array instance.

One another interesting piece of information we noted while evaluating a frame (GUI) based application was that the 'main' thread which is the entry & exit point of any application exited after the frame was loaded and visible. It was not the terminating block of the frame.

Further in the unit testing, the schema of the data store, the range of values (i.e., the boundary limits) and the control flow were examined and rectified for optimum performance. While developing the profiler agent we came across several memory overruns and hence experienced a lot of memory errors and exceptions. These were successfully corrected by properly allocating the memory for the various components involved.

II. Integration Testing

While integrating separate modules together, we experienced minimum trouble as the interfaces for integration were well defined. Hence the test of combining modules together went flawless. Integration basically involved connecting the profiling agent with the server through sockets. It also involved integrating the drawing module to the server. The server maintains and communicates with the data store for storing and handling information.

The inbuilt delay supplied for the refresh rate of the graph was sufficient enough to portray the requirements and no lag was found in communication between the graph and the data store.

- The server for obtaining data currently supports only one client at a time. This could be extended to multiple clients as an additional feature.

- The information that is stored in the data store could be stored in a XML document thereby enabling it to be viewed from the web.

- The current display of information could be linked with the source file where the event takes place. Currently the profiler does not include link to the source document of the application to be evaluated.

The evaluator was able to profile an application successfully. The graphs and the tables drawn were easily discernable and the information provided meets all the requirements that were supposed to be met. This evaluator brings out a new trend of evaluating applications to judge their performance and capabilities. A little has been done, a lot more to go as the project has good scope for further development and maintenance.

1. JVMPI documentation

   http://java.sun.com/j2se/1.3/docs/guide/jvmpi/jvmpi.html

2. UML Use Case Diagrams -Engineering Notebook -C++ Report -Nov-Dec 98
3. http://www.rational.com/uml/resources
4. Patrick Naughton & Herbert Schildt, 'Java Complete Reference', Tata McGraw Hill, 2000.
5. Michael Morrison, 'Java 1.1 Unleashed', Sams net, 1996.
6. MSDN library – October 1999.

SOURCE CODE

Code for the Profiler agent

```
#include<winsock2.h>
#include<process.h>
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include <jvmpi.h>
#define SERV_ADDR "127.0.0.1"
#define SERV_PORT 2000
static JVMPI_Interface *jvmpi_interface;
static SOCKET s;
static struct sockaddr_in addr;
static WSADATA WSAData;
struct list_thread {
        JNIEnv *data;
        struct list_thread *next;
};
typedef struct list_thread node;
node *start;
int listflag=0;
int globalisarray;
char  *sourcename;
jobjectID objclass;
char globalmsg[100];
unsigned __stdcall MyThread(void * param);
char * padchar(char * got);
static DWORD tid1;
```

```
  void listcreate(node *inp,JNIEnv *data1)
  {
          inp->data=data1;
          inp->next=NULL;
  }
  void listinsert(node *inp,JNIEnv *data1)
  {
          node *newrecord,*back;
          newrecord=(node *)malloc(sizeof(node));
          newrecord->data=data1;
          newrecord->next=NULL;
          back=inp;
          while(inp->next!=NULL)
                  inp=inp->next;
          inp->next=newrecord;
          inp=back;
}
void listremove(node *inp,JNIEnv *data1)
{
        node *back,*pr;
        back=inp;
        if(inp->data==data1)
                inp=inp->next;
        else
        {
                while(inp->next->data!=data1)
                        inp=inp->next;
                pr=inp->next->next;
                free(inp->next);
                inp->next=pr;
                inp=back;
          }
  }
  void notifyEvent(JVMPI_Event *event) {
          int nummethods,i,numframes;char *msg,msgtemp[40],*tosend;
          JVMPI_Method *back;
          JVMPI_CallTrace *trace;
          jobjectID clsid;
   switch(event->event_type) {
   case JVMPI_EVENT_CLASS_LOAD:
          sprintf(msg,"classload~");
          itoa(event->u.class_load.class_id,msgtemp,10);
          strcat(msg,msgtemp);
          strcat(msg,"~");
          tosend = padchar(event->u.class_load.class_name);
          strcat(msg,tosend);
          if(event->u.class_load.source_name!=0 && strcmp(event-
>u.class_load.source_name,sourcename)==0)
         {
         nummethods=event->u.class_load.num_methods;
         back=event->u.class_load.methods;
         strcat(msg,"~");
         for(i=0;i<nummethods;i++)
         {
                 itoa(event->u.class_load.methods->method_id,msgtemp,10);
                 strcat(msg,msgtemp);
                 strcat(msg,"~");
                 tosend=padchar(event->u.class_load.methods->method_name);
                 strcat(msg,tosend);
                 strcat(msg,"~");
                 event->u.class_load.methods++;
      }
```

```
event->u.class_load.methods=back;
}
strcat(msg,"\n");
send(s,msg,strlen(msg),MSG_DONTROUTE);
break;
case JVMPI_EVENT_OBJECT_ALLOC:
globalisarray=event->u.obj_alloc.is_array;
objclass=event->u.obj_alloc.class_id;
switch(globalisarray)
{
case JVMPI_NORMAL_OBJECT:
case JVMPI_CLASS:
sprintf(msg,"oa~");
itoa(objclass,msgtemp,10);
strcat(msg,msgtemp);
strcat(msg,"\n");
send(s,msg,strlen(msg),MSG_DONTROUTE);
        break;
case JVMPI_BOOLEAN:
        send(s,"stdoa~boolean\n",14,MSG_DONTROUTE);
        break;
case JVMPI_BYTE:
        send(s,"stdoa~byte\n",11,MSG_DONTROUTE);
        break;
case JVMPI_CHAR:
        send(s,"stdoa~char\n",11,MSG_DONTROUTE);
        break;
case JVMPI_SHORT:
        send(s,"stdoa~short\n",12,MSG_DONTROUTE);
        break;
case JVMPI_INT:
```

```
                send(s,"stdoa~int\n",10,MSG_DONTROUTE);
                break;
         case JVMPI_LONG:
                send(s,"stdoa~long\n",11,MSG_DONTROUTE);
                break;
         case JVMPI_FLOAT:
                send(s,"stdoa~float\n",12,MSG_DONTROUTE);
                break;
         case JVMPI_DOUBLE:
                send(s,"stdoa~double\n",13,MSG_DONTROUTE);
                break;
         }
        trace=malloc(1*sizeof(JVMPI_CallTrace));
        trace->env_id=event->env_id;
        jvmpi_interface->GetCallTrace(trace, 1);
        numframes=trace->num_frames;
        if(numframes!=0)
        {
        clsid=jvmpi_interface->GetMethodClass((jmethodID)trace->frames-
>method_id);
        sprintf(globalmsg,"userobjalloc~");
        itoa(event->u.obj_alloc.obj_id,msgtemp,10);
        strcat(globalmsg,msgtemp);
        strcat(globalmsg,"~");
        itoa(trace->frames->method_id,msgtemp,10);
        strcat(globalmsg,msgtemp);
        strcat(globalmsg,"~");
        jvmpi_interface->RequestEvent(JVMPI_EVENT_CLASS_LOAD,clsid);
        }
        break;
        case JVMPI_EVENT_OBJECT_FREE:
            sprintf(globalmsg,"objfree~");
            itoa(event->u.obj_free.obj_id,msgtemp,10);
            strcat(globalmsg,msgtemp);
            strcat(globalmsg,"~");
            jvmpi_interface-
>RequestEvent(JVMPI_EVENT_OBJECT_ALLOC,event->u.obj_free.obj_id);
            break;
        case JVMPI_EVENT_OBJECT_ALLOC | JVMPI_REQUESTED_EVENT:
            if(event->u.obj_alloc.class_id==0)
            {
            switch(event->u.obj_alloc.is_array)
            {
                case JVMPI_BOOLEAN:
                    strcat(globalmsg,"boolean~o\n");
                    break;
                case JVMPI_BYTE:
                    strcat(globalmsg,"byte~o\n");
                    break;
                case JVMPI_CHAR:
                    strcat(globalmsg,"char~o\n");
                    break;
                case JVMPI_SHORT:
                    strcat(globalmsg,"short~o\n");
                    break;
                case JVMPI_INT:
                    strcat(globalmsg,"int~o\n");
                    break;
                case JVMPI_LONG:
                    strcat(globalmsg,"long~o\n");
                    break;
                case JVMPI_FLOAT:
                        strcat(globalmsg,"float~o\n");
                        break;
                  case JVMPI_DOUBLE:
                        strcat(globalmsg,"double~o\n");
                        break;
            }
            }
            else
            {
                  itoa(event->u.obj_alloc.class_id,msgtemp,10);
                  strcat(globalmsg,msgtemp);
                  strcat(globalmsg,"\n");
            }
            send(s,globalmsg,strlen(globalmsg),MSG_DONTROUTE);
            break;
      case JVMPI_EVENT_CLASS_UNLOAD:
  sprintf(msg,"classunload~");
      itoa(event->u.class_unload.class_id,msgtemp,10);
      strcat(msg,msgtemp);
  strcat(msg,"\n");
      send(s,msg,strlen(msg),MSG_DONTROUTE);
      break;
     case JVMPI_EVENT_CLASS_LOAD | JVMPI_REQUESTED_EVENT:
           if(event->u.class_load.source_name!=0 && strcmp(event-
>u.class_load.source_name,sourcename)==0)
           {
            if(objclass!=0)
            {
                  if(event->u.class_load.class_id!=0)
                  {
                        itoa(objclass,msgtemp,10);
```

```
                strcat(globalmsg,msgtemp);
                strcat(globalmsg,"\n");
                send(s,globalmsg,strlen(globalmsg),MSG_DONTROUTE);
            }
        }
        else
        {
            switch(globalisarray)
            {
                case JVMPI_BOOLEAN:
                strcat(globalmsg,"boolean~");
                break;
                case JVMPI_BYTE:
                strcat(globalmsg,"byte~");
                        break;
                case JVMPI_CHAR:
                strcat(globalmsg,"char~");
                        break;
                case JVMPI_SHORT:
                strcat(globalmsg,"short~");
                        break;
                case JVMPI_INT:
                strcat(globalmsg,"int~");
                        break;
                case JVMPI_LONG:
                strcat(globalmsg,"long~");
                        break;
                case JVMPI_FLOAT:
                strcat(globalmsg,"float~");
                        break;
                case JVMPI_DOUBLE:
                      strcat(globalmsg,"double~");
                            break;


                 }
                 strcat(globalmsg,"arr\n");
    send(s,globalmsg,strlen(globalmsg),MSG_DONTROUTE);
          }
         }
  break;
  case JVMPI_EVENT_THREAD_START:
    sprintf(msg,"thstart~");
        itoa(event->u.thread_start.thread_env_id,msgtemp,10);
    strcat(msg,msgtemp);
        strcat(msg,"~");
        tosend=padchar(event->u.thread_start.thread_name);
        strcat(msg,tosend);
strcat(msg,"\n");
    send(s,msg,strlen(msg),MSG_DONTROUTE);
        if(listflag==0)
        {
              start=(node *)malloc(sizeof(node));
              listcreate(start,event->u.thread_start.thread_env_id);
              listflag=1;
              return;
        }
  listinsert(start,event->u.thread_start.thread_env_id);
  break;
 case JVMPI_EVENT_THREAD_END:
   sprintf(msg,"thend~");
       itoa(event->env_id,msgtemp,10);
   strcat(msg,msgtemp);
```

```
      strcat(msg,"\n");
          send(s,msg,strlen(msg),MSG_DONTROUTE);
         listremove(start,event->env_id);
         break;
        case JVMPI_EVENT_GC_FINISH:
             sprintf(msg,"gc~");
             itoa(event->u.gc_info.total_object_space,msgtemp,10);
             strcat(msg,msgtemp);
             strcat(msg,"~");
             itoa(event->u.gc_info.used_object_space,msgtemp,10);
             strcat(msg,msgtemp);
             strcat(msg,"\n");
             send(s,msg,strlen(msg),MSG_DONTROUTE);
       break;
   case JVMPI_EVENT_GC_START:
       break;
  case JVMPI_EVENT_JVM_SHUT_DOWN:
       shutdown(s,SD_SEND);
       closesocket(s);
       WSACleanup();
  break;
  }
}
 JNIEXPORT jint JNICALL JVM_OnLoad(JavaVM *jvm, char *options, void
*reserved)
 {
  int res;
  res= (*jvm)->GetEnv(jvm,(void **)&jvmpi_interface, JVMPI_VERSION_1);
     if(res<0)
          return JNI_ERR;
 jvmpi_interface->NotifyEvent = notifyEvent;
```

```
   jvmpi_interface->EnableEvent(JVMPI_EVENT_CLASS_LOAD, NULL);
   jvmpi_interface->EnableEvent(JVMPI_EVENT_CLASS_UNLOAD, NULL);
   jvmpi_interface->EnableEvent(JVMPI_EVENT_THREAD_START, NULL);
   jvmpi_interface->EnableEvent(JVMPI_EVENT_THREAD_END, NULL);
        jvmpi_interface->EnableEvent(JVMPI_EVENT_OBJECT_ALLOC, NULL);
        jvmpi_interface->EnableEvent(JVMPI_EVENT_OBJECT_FREE, NULL);
        jvmpi_interface->EnableEvent(JVMPI_EVENT_GC_FINISH,NULL);
        jvmpi_interface->EnableEvent(JVMPI_EVENT_GC_START,NULL);
        jvmpi_interface->EnableEvent(JVMPI_EVENT_JVM_SHUT_DOWN, NULL);
        sourcename=malloc(strlen(options)*sizeof(char)+10);
        strcpy(sourcename,options);
        strcat(sourcename,".java");
        WSAStartup(MAKEWORD(1,1), &WSAData);
        s=socket(AF_INET,SOCK_STREAM,0);
        if(s==INVALID_SOCKET)
                printf("Invalid Socket");
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = inet_addr(SERV_ADDR);
        addr.sin_port  = htons(SERV_PORT);
        if(connect(s,(struct sockaddr *) &addr,sizeof(addr))==SOCKET_ERROR)
                printf("connection error");
        CreateThread(NULL,0,MyThread,(LPVOID)jvmpi_interface,0,(unsigned
*)&tid1);
  }
DWORD WINAPI MyThread(void * param)
{
                node *temp=NULL;char *msg,msgtemp[20];
                //jvmpi_interface-
>RequestEvent(JVMPI_EVENT_MONITOR_DUMP,NULL);
        while(1)
        {
                sprintf(msg,"thstatus~");
                if(start!=NULL)
                {
                        temp=start;
                while(temp->next!=NULL)
                {


                        //printf("%d\n",temp->data);
                        itoa(temp->data,msgtemp,10);
                        strcat(msg,msgtemp);
                        strcat(msg,"~");
                        itoa(jvmpi_interface->GetThreadStatus(temp->data),msgtemp,10);
                        strcat(msg,msgtemp);
                        strcat(msg,"~");
                        temp=temp->next;
                }
                //printf("%d\nEND",temp->data);
                itoa(temp->data,msgtemp,10);
                strcat(msg,msgtemp);
                strcat(msg,"~");
                itoa(jvmpi_interface->GetThreadStatus(temp->data),msgtemp,10);
                strcat(msg,msgtemp);
                strcat(msg,"\n");
                send(s,msg,strlen(msg),MSG_DONTROUTE);
                Sleep(3000);
                }
        }
        return 0;
}
char * padchar(char * got)
{
        char *result,*retstring,*prev,*back=got;
        result=malloc(sizeof(char)*strlen(got));
        retstring=malloc(sizeof(char)*2*strlen(got));
        sprintf(retstring,"");
        result=strstr(got,"~");
        if(result==NULL)
                retstring=got;
        else
        {
        while(result!=NULL)
        {
                prev=malloc(sizeof(char)*strlen(got));
                strncpy(prev,got,result-got);
                prev[result-got]='\0';
                strcat(retstring,prev);
                strcat(retstring,"~~");
                got=result+1;
                result=strstr(got,"~");
        }
        strcat(retstring,got);
        got=back;
        }
        return retstring;
}
```

Code of the Front end /Server

```
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;
class wl extends WindowAdapter
{
  fra frame;
        wl(fra frame)
        {
         this.frame=frame;
        }
 public void windowClosing(WindowEvent w)
 {
   //System.exit(0);
        frame.setVisible(false);
 }
 public void windowActivated(WindowEvent w)
 {
        frame.setVisible(true);
        frame.stat=true;
 }
 public void windowDeactivated(WindowEvent w)
 {
        frame.stat=false;
 }
}
```

```
   class fra extends JFrame
   {
           boolean stat;
           fra()
           {
            stat=false;
           }
  }
  class ClassList
  {
   String name;
   int instancecount;
   ClassList(String name,int instancecount)
   {
    this.name=name;this.instancecount=instancecount;
   }
  }
 class ThreadList
 {
  String name;
  int status;
  ThreadList(String name,int status)
  {
   this.name=name;this.status=status;
  }
 }
class GCList
{
 int totalobjspace;
 int usedobjspace;
GCList(int totalobjspace, int usedobjspace)
```

```
  {
  this.totalobjspace=totalobjspace;this.usedobjspace=usedobjspace;
  }
  }
  class UserObjList
  {
  int methodid;
  int classtypeid;
  UserObjList(int methodid,int classtypeid)
  {
  this.methodid=methodid;this.classtypeid=classtypeid;
  }
 }
 class MethodList
 {
 String name;
 int classid;
 MethodList(String name,int classid)
 {
 this.name=name;this.classid=classid;
 }
}
public class JavaServ
{
static Hashtable classlist;
static Hashtable threadlist;
static Hashtable methodlist;
static Hashtable gclist;
static Hashtable userobjlist;
static
{
```

```
            classlist=new Hashtable();
            threadlist=new Hashtable();
            methodlist=new Hashtable();
            gclist=new Hashtable();
            userobjlist=new Hashtable();
    }
    public static void main(String a[])throws Exception
    {
    int port = 2000;
    String inputLine;
    Socket s;
    BufferedReader inbound;
    ServerSocket ss =new ServerSocket(port);
    newthread threadobj1=new newthread(1);
    threadobj1.start();
    newthread threadobj2=new newthread(2);
    threadobj2.start();
    newthread threadobj3=new newthread(3);
    threadobj3.start();
    newthread threadobj4=new newthread(4);
    threadobj4.start();
    newthread threadobj5=new newthread(5);
    threadobj5.start();
    newthread threadobj6=new newthread(6);
    threadobj6.start();
    while(true)
    {
     s=ss.accept();
     inbound=new BufferedReader(new InputStreamReader(s.getInputStream()));
     while((inputLine=inbound.readLine())!=null)
     {
```

```
String op[]=toksep.parseString(inputLine);
    if(op[0].equals("classload"))
    {
    classlist.put(op[1],new ClassList(op[2],0));
     for(int i=3;i<op.length;i=i+2)
     {
      methodlist.put(op[i],new
    MethodList(op[i+1],Integer.parseInt(op[1])));
     MethodList temp=(MethodList)methodlist.get(op[i]);
     }
    }
    else if(op[0].equals("classunload"))
    {
    classlist.remove(op[1]);
    }
   else if(op[0].equals("oa"))
   {
    if(!(op[1].equals("0")))
    {
ClassList temp=(ClassList)classlist.get(op[1]);
    temp.instancecount++;
classlist.put(op[1],temp);
    }
   }
   else if(op[0].equals("stdoa"))
   {
   Set keylist=classlist.keySet();
   Iterator itr=keylist.iterator();
   while(itr.hasNext())
   loop1:
   {
       String str=(String)itr.next();
  ClassList temp=(ClassList)classlist.get(str);
       if(temp.name.equals(op[1]))
       {
                temp.instancecount++;
     classlist.put(str,temp);
                break loop1;
      }
     }
    }
   else if(op[0].equals("thstart"))
   {
    threadlist.put(op[1],new ThreadList(op[2],0));
   }
   else if(op[0].equals("thend"))
   {
   threadlist.remove(op[1]);
  }
  else if(op[0].equals("thstatus"))
  {
  for(int i=1;i<op.length;i=i+2)
  {
   ThreadList temp=(ThreadList)threadlist.get(op[i]);
   temp.status=Integer.parseInt(op[i+1]);
  threadlist.put(op[i],temp);
  }
}
else if(op[0].equals("gc"))
{
java.util.Date d=new java.util.Date();
```

```
             gclist.put(d.toString(),new
   GCList(Integer.parseInt(op[1]),Integer.parseInt(op[2])));
            }
            else if(op[0].equals("userobjalloc"))
            {
             if(op.length==5)
             {
                      Set keylist=classlist.keySet();
                      Iterator itr=keylist.iterator();
                      while(itr.hasNext())
                      loop1:
                      {
                      String str=(String)itr.next();
                  ClassList temp=(ClassList)classlist.get(str);
                      if(temp.name.equals(op[3]))
                      {
                              userobjlist.put(op[1],new
 UserObjList(Integer.parseInt(op[2]),Integer.parseInt(str)));
                              break loop1;
                      }
                     }
            }
       else
                  userobjlist.put(op[1],new
UserObjList(Integer.parseInt(op[2]),Integer.parseInt(op[3])));
          }
          else if(op[0].equals("objfree"))
          {
          if(op.length!=4)
          {
          if(!(op[2].equals("0")))
```

```
         {
      ClassList temp=(ClassList)classlist.get(op[2]);
         if(temp.instancecount>0)
                  temp.instancecount--;
         if(userobjlist.containsKey(op[1]))
                 userobjlist.remove(op[1]);
         }
         }
         else
         {
                 Set keylist=classlist.keySet();
                 Iterator itr=keylist.iterator();
                 while(itr.hasNext())
                 loop1:
                 {
                 String str=(String)itr.next();
             ClassList temp=(ClassList)classlist.get(str);
                 if(temp.name.equals(op[2]))
                 {
                       //userobjlist.put(op[1],new
UserObjList(Integer.parseInt(op[2]),Integer.parseInt(str)));
                       if(temp.instancecount>0)
                         temp.instancecount--;
                       if(userobjlist.containsKey(op[1]))
                               userobjlist.remove(op[1]);
                       break loop1;
                }
               }
       }
      }
}//while inner
```

```
 }//while outer
} //  end of main
} //  end of class

class toksep
{
 public static Vector parse(String inp)
 {
  Vector tokens=new Vector();
  int index;
  int traver;
  if((traver=inp.indexOf("~~"))==-1)
  {
   StringTokenizer tokenizer=new StringTokenizer(inp,"~");
   int num_of_tokens=tokenizer.countTokens();
   for(index=0;index<num_of_tokens;index++)
   {
    tokens.addElement(tokenizer.nextToken());
    //System.out.println(tokens.elementAt(index));
   }
  }
  else
  {
   int traverse=inp.indexOf("~~");
   int inpindex=inp.indexOf("~");
   String temp=inp.substring(0,traverse+1);
   inp=inp.substring(traverse+2);
   Vector tempvector=parse(temp);
   int vecsize=tempvector.size();
   for(index=0;index<vecsize-1;index++)
   {
```

```
   tokens.addElement(tempvector.elementAt(index));
  }
  temp=tempvector.lastElement().toString()+"~";
  traverse=inp.indexOf("~~");
  inpindex=inp.indexOf("~");
  while(inpindex==traverse && traverse!=-1)
  {
   temp=temp+inp.substring(0,traverse+1);
   inp=inp.substring(traverse+2);
   traverse=inp.indexOf("~~");
   inpindex=inp.indexOf("~");
  }
  if(inpindex!=-1)
  {
   temp=temp+inp.substring(0,inpindex);
   inp=inp.substring(inpindex+1);
  }
  else
  {
   temp=temp+inp;
   inp="";
  }
  tokens.addElement(temp);
  tempvector=parse(inp);
  vecsize=tempvector.size();
  for(index=0;index<vecsize;index++)
  {
   tokens.addElement(tempvector.elementAt(index));
  }
 }
return tokens;
```

```
 }
 public static String[] parseString(String inp)
 {
  Vector v=new Vector();
  v=parse(inp);
  int vecsize=v.size();
  String ret[]=new String[vecsize];
  for(int index=0;index<vecsize;index++)
  {
   ret[index]=v.elementAt(index).toString();
  }
  return ret;
 }
}
class DrawClassCount
{
  public static int vecmax(Vector coords,int vecsize)
  {
   int maxvalue=Integer.parseInt(coords.elementAt(0).toString()),temp;
   for(int index=1;index<vecsize;index++)
   {
    if((temp=Integer.parseInt(coords.elementAt(index).toString()))>maxvalue)
     maxvalue=temp;
   }
   return maxvalue;
  }
  public DrawClassCount() throws Exception
  {
    fra f1=new fra();
    int num = 0,a1=0,a2=10,a3=15,a4=20;
    String s1,s2,s3,s4,s5,p1,p2,p3,p4;
String z1,z2,z3,z4,z5,z6,z7;
int q = 1;
Container c = f1.getContentPane();
f1.setSize(800,570);
f1.setVisible(true);
f1.addWindowListener(new wl(f1));
Graphics g = c.getGraphics();
 c.setBackground(new Color(255,255,255));
Vector coords=new Vector();
int vecsize,newval,maxvalue=0,newmaxvalue;
float ydraw1,ydraw2;
while(true)
{
//newval=(int)Math.ceil(Math.random()*300);
newval=JavaServ.classlist.size();
vecsize=coords.size();
if(vecsize<=20)
 coords.addElement(new Integer(newval));
else
{
 coords.removeElementAt(0);
 coords.addElement(new Integer(newval));
}
newmaxvalue=vecmax(coords,vecsize);
if(newmaxvalue!=maxvalue)
{
 maxvalue=newmaxvalue;
//redraw the graph with modified scale
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
// grey lines
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


int jj=10;
        for(int ii=1;ii<=30;ii++)
            {
             g.setColor(Color.red);
             g.setColor(new Color(200,200,200));
             g.drawLine(170,300-jj,600,300-jj);
             jj=jj+10;
             }
jj=20;
  for(int ii=1;ii<=20;ii++)
             {
             g.setColor(new Color(200,200,200));
            g.drawLine(170+jj,0,170+jj,300);
            jj=jj+20;
            }


// <<<------------------------------------------------------>>> //




         int y1=0;
      for(int i=0;i<(vecsize-1);i++)
       {
       y1=Integer.parseInt(coords.elementAt(i).toString());
       int y2=Integer.parseInt(coords.elementAt(i+1).toString());


          if(maxvalue!=0)
           {
            ydraw1=300-((float)200/maxvalue*y1);
          ydraw2=300-((float)200/maxvalue*y2);
         }
        else
         {ydraw1=ydraw2=300;}


////////////////////////////////////
// Drawing the graph
////////////////////////////////////


      g.setColor(new Color(181,31,4));
     g.drawLine((vecsize-i)*20+150,(int)ydraw1,(vecsize-i-1)*20+150,(int)ydraw2);
     g.drawLine((vecsize-i)*20+150,(int)ydraw1+1,(vecsize-i-
1)*20+151,(int)ydraw2+1);
     g.setColor(Color.red);
}
//////////////////////////////////////////////////////////
//  Drawing X-axis and Y-axis
 //   COLOR ADDITIONS
//////////////////////////////////////////////////////////


     if(num>1)
      {
      s1=""+(a1);
      s2=""+(a1-5);
      s3=""+(a1-10);
      s4=""+(a1-15);
      s5=""+(a1-20);
      g.drawString(s1,170,315);
      if(num>5)   g.drawString(s2,265,315);
      if(num>10)  g.drawString(s3,362,315);
```

```
  if(num>15)  g.drawString(s4,463,315);
  if(num>20)  g.drawString(s5,564,315);


      // ------------------------------------------- //
      //  Right hand corner display                  //
      // ------------------------------------------- //


        g.setColor(new Color(0,0,0));
          g.drawRect(610,100,175,100);
                  g.drawRect(609,99,177,102);
                  g.drawRect(608,98,179,104);
                  g.drawRect(607,97,181,106);
                  g.drawLine(610,147,785,147);
                  g.drawLine(610,149,785,149);
        g.setColor(new Color(209,36,5));


                g.drawString("TOTAL TIME ELAPSED",620,125);
                g.drawString(""+s1,684,140);
                  g.setColor(Color.red);
}


float q1= (float)maxvalue/200;
if(q1==0) q1=1;


z1=""+(int)(0*q1);
z2=""+(int)(50*q1);
z3=""+(int)(100*q1);
z4=""+(int)(150*q1);
z5=""+(int)(200*q1);
z6=""+(int)(250*q1);
z7=""+(int)(300*q1);
```

```
        g.drawString(z1,140,300);
        g.drawString(z2,140,250);
        g.drawString(z3,140,200);
        g.drawString(z4,140,150);
        g.drawString(z5,140,100);
        g.drawString(z6,140,50);
        g.drawString(z7,140,0);



        g.setColor(new Color(0,0,255));
        g.drawLine(170,0,170,300);
        g.drawLine(171,0,171,300);
        g.drawLine(170,300,600,300);
        g.drawLine(170,301,600,301);

//  X-axis Caption
Font f = new Font("TimesRoman", Font.BOLD,14);
g.setFont(f);
g.setColor(new Color(160,0,200));
g.drawString("TIME IN SECONDS",340,340);
//  Y-axis Caption
Font fff = new Font("TimesRoman", Font.BOLD,14);
g.setFont(fff);
g.drawString("Number",70,140);
g.drawString("of",70,155);
g.drawString("Classes  ",70,170);
g.drawString("Loaded   ",70,185);
Font ff = new Font("TimesRoman", Font.BOLD,14);
g.setFont(ff);
g.drawString("No of Classes loaded :  "+y1,620,170);
//////////////////////////////////////////////////
```

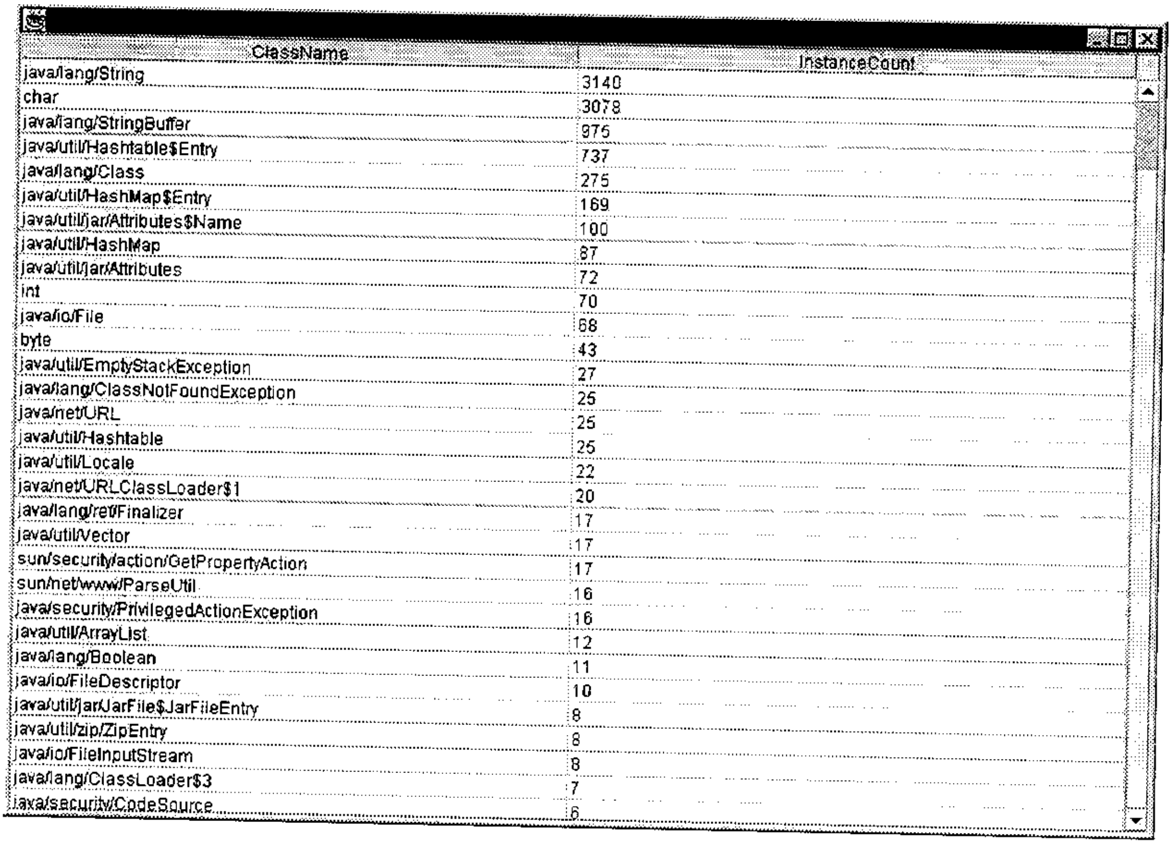
```
//End of drawing X-axis and Y-axis
///////////////////////////////////////////////

        num=num+1;
        a1=a1+1;
         Thread.sleep(1000);
        c.repaint();
      }
    }
}


class InstanceCountGraph{
 public InstanceCountGraph()throws Exception
   {

               fra f1=new fra();
           JCheckBox jcb=new JCheckBox("Refresh",true);
           JScrollPane temp=new JScrollPane();
           boolean flag=false;
        Container c = f1.getContentPane();
        Graphics g = c.getGraphics();
        c.setLayout(new BorderLayout());
             f1.setVisible(true);
        f1.setSize(800,570);
        f1.addWindowListener(new wl(f1));
        while(true)
          {
                      if(jcb.isSelected())
                      {
                      if(flag)
```

```
                c.remove(temp);
                flag=true;
          r22 obj=new r22();
          c.add(obj.jsp,BorderLayout.CENTER);
             c.add(jcb,BorderLayout.NORTH);
                if(f1.stat==true)
           f1.setVisible(true);
                Thread.sleep(3000);
                temp=obj.jsp;
                }
        }
   }
}


//----------------------------------------------------------------------


 class r22 {
final String[] colHeads = {"ClassName","InstanceCount"};
Object[][] data;
 JScrollPane jsp;
 JTable table;
 int v,h;

        r22()
          {
try{
              data = new Object[JavaServ.classlist.size()+1][2];
```

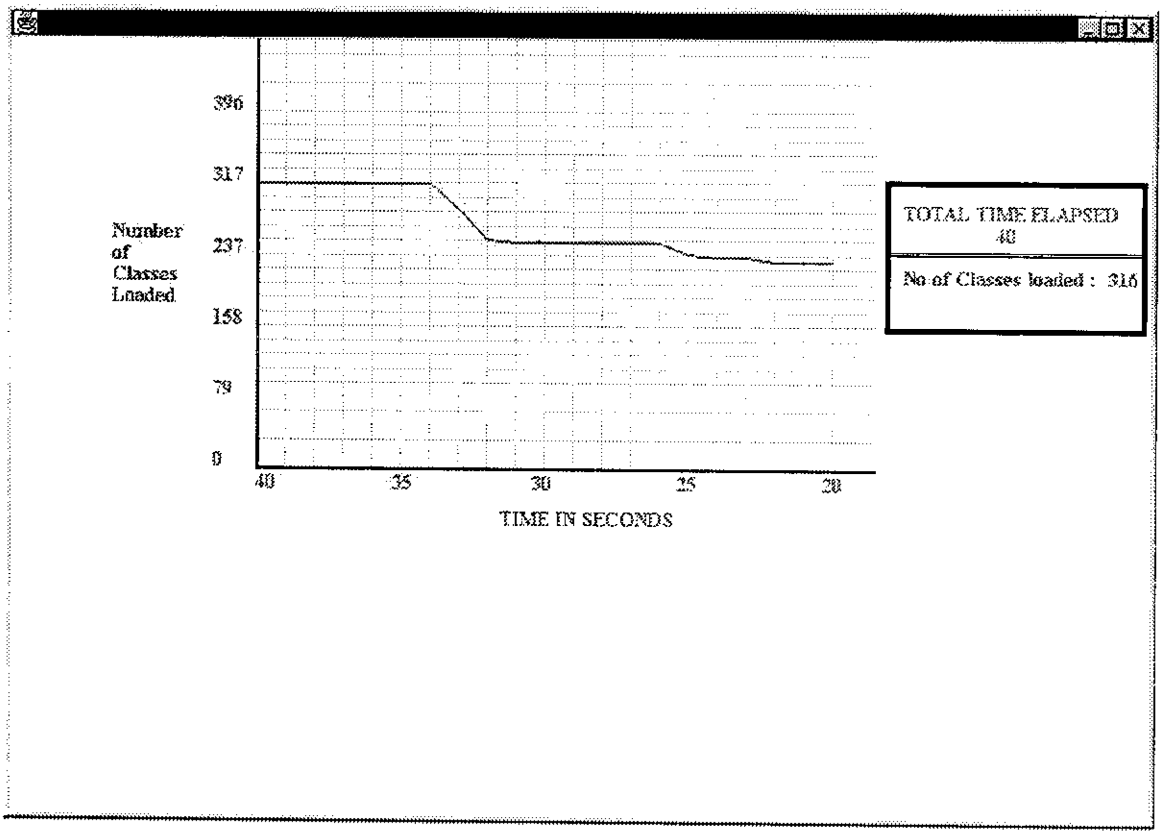
```
/////////////////////////////////////////////////////
// table drawing
/////////////////////////////////////////////////////
int r1=0;
          Set keylist=JavaServ.classlist.keySet();
          Iterator itr=keylist.iterator();
          while(itr.hasNext())
             {
                          String str=(String)itr.next();
                          ClassList temp=(ClassList)JavaServ.classlist.get(str);
                          String cname1=temp.name;
                          String cname2=(new Integer(temp.instancecount)).toString();
                          data[r1][0]     = cname1;
              data[r1++][1]   = cname2;
             }

 } catch(Exception e){System.out.println(e);}



         table = new JTable(data,colHeads);
         v=ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
      h=ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
      jsp = new JScrollPane(table,v,h);
 }  // end of r constructor
}
class newthread extends Thread
{
         int type;
newthread(int type)
{        this.type=type;}
public void run()
```

```
 {        try
          {
          if(type==1)
          {
          DrawClassCount obj=new DrawClassCount();
          }
          else if(type==2)
          {
          DrawThreadCount obj=new DrawThreadCount();
          }
          else if(type==3)
          {
          HeapGraph obj=new HeapGraph();
          }
          else if(type==4)
          {
          ThreadTable obj=new ThreadTable();
          }
          else if(type==5)
          {
          InstanceCountGraph obj=new InstanceCountGraph();
          }
          else if(type==6)
          {       GCGraph obj=new GCGraph();       }
          }catch(Exception e){}
}
}
```

OUTPUT SCREEN

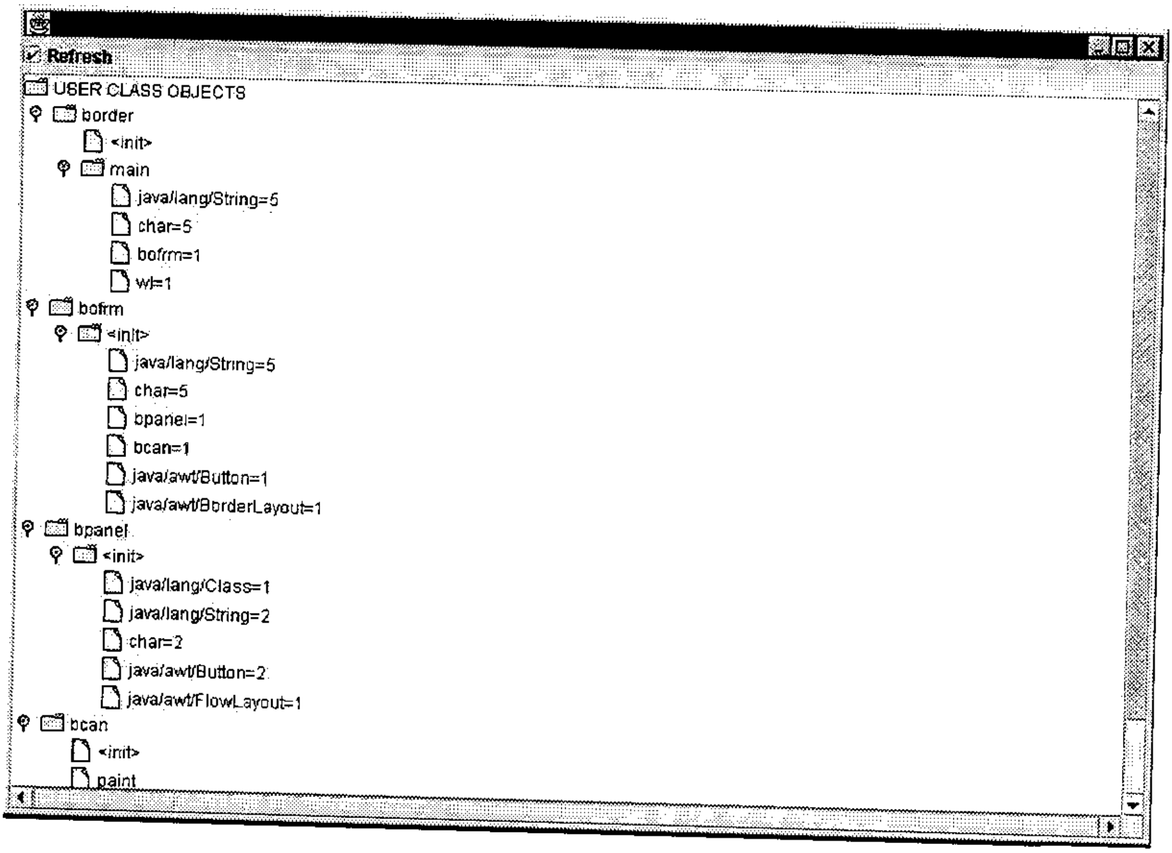


ClassName	InstanceCount
java/lang/String	3140
char	3078
java/lang/StringBuffer	975
java/util/Hashtable$Entry	737
java/lang/Class	275
java/util/HashMap$Entry	169
java/util/jar/Attributes$Name	100
java/util/HashMap	87
java/util/jar/Attributes	72
int	70
java/io/File	68
byte	43
java/util/EmptyStackException	27
java/lang/ClassNotFoundException	25
java/net/URL	25
java/util/Hashtable	25
java/util/Locale	22
java/net/URLClassLoader$1	20
java/lang/ref/Finalizer	17
java/util/Vector	17
sun/security/action/GetPropertyAction	17
sun/net/www/ParseUtil	16
java/security/PrivilegedActionException	16
java/util/ArrayList	12
java/lang/Boolean	11
java/io/FileDescriptor	10
java/util/jar/JarFile$JarFileEntry	8
java/util/zip/ZipEntry	8
java/io/FileInputStream	8
java/lang/ClassLoader$3	7
java/security/CodeSource	6



TOTAL TIME ELAPSED
40

No of Classes loaded : 316



☑ Refresh

```
USER CLASS OBJECTS
  border
      <init>
    main
        java/lang/String=5
        char=5
        bofrm=1
        wl=1
  bofrm
    <init>
        java/lang/String=5
        char=5
        bpanel=1
        bcan=1
        java/awt/Button=1
        java/awt/BorderLayout=1
  bpanel
    <init>
        java/lang/Class=1
        java/lang/String=2
        char=2
        java/awt/Button=2
        java/awt/FlowLayout=1
  bcan
      <init>
      paint
```