# Empirical Study of Concurrency Mutation Operators for Java

Leon Wu and Gail Kaiser
*Department of Computer Science*
*Columbia University*
*New York, NY 10027*
*Email: {leon, kaiser}@cs.columbia.edu*

***Abstract*—Mutation testing is a white-box fault-based software testing technique that applies mutation operators to modify program source code or byte code in small ways and then runs these modified programs (*i.e.*, mutants) against a test suite in order to measure its effectiveness and locate the weaknesses either in the test data or in the program that are seldom or never exposed during normal execution.**

**In this paper, we describe our implementation of a generic mutation testing framework and the results of applying three sets of concurrency mutation operators on four example Java programs through empirical study and analysis.**

***Keywords*-software testing; mutation testing; mutation operators; Java; concurrent programs; synchronization;**

## I. Background

*Mutation testing* is a white-box fault-based software testing technique that uses mutants, slightly modified variants of the program source code or byte code, to characterize the effectiveness of a testing suite and locate weaknesses in the test data or program that are seldom or never exposed during normal execution. [11] It has been studied and used since the 1970s. [16] Mutation testing is based on the *Competent Programmer Hypothesis* [1][11] and the *Coupling Effect Hypothesis* [11][21]. The *Competent Programmer Hypothesis* assumes that programmers are competent and normally write programs that are close to perfect; program faults are syntactically small and can be corrected with a few small code modifications. The *Coupling Effect Hypothesis* states that complex bugs in the software are closely coupled to small, simple bugs. Thus, mutation testing can be effective in simulating complex real-world bugs and may be used for testing programs.

Mutation testing typically involves three stages: (1) Mutant generation, the goal of which is the generation of mutants of the program under test. (2) Mutant execution, the goal of which is the execution of test cases against both the original program and the mutants. (3) Result analysis, the goal of which is to check the mutation score obtained by the test suite. [24] For the first stage, a predefined set of mutation operators are used to generate mutants from program source code or byte code. A *mutation operator* is a rule that is applied to a program to create mutants, [23] and researchers have developed different sets of mutation operators [23][18], targeting a variety of programming languages. Budd *et al.* first applied it to Fortran, the first language used in mutation. [7] Offutt *et al.* formally defined the mutation operators for Fortran 77 and DeMillo *et al.* developed the Mothra mutation toolset for Fortran 77. [22][10]

Carver investigated the nondeterminism issue related to mutation testing of concurrent programs and described a combination approach of deterministic execution mutation testing. He also described the implementation of a TDCAda framework for Ada and concurrent C. [8] Aichernig *et al.* worked on specification mutation and defined a set of mutation operators for Full LOTOS. [2] To measure the testability of concurrent Java programs, Ghosh described mutation based on two mutation operators that remove the keyword `synchronized`. [15] Long *et al.* tested mutation-based exploration for concurrent Java components and applied this method to the readers-writers problem. Although this paper mentioned the mutants were based on common concurrency faults, the details of how these mutants were created were not described and the mutation operators used are unknown. [20]

Delamaro *et al.* proposed a set of 15 concurrency mutation operators for Java within four groups: monitor lock code, methods related to wait set manipulation that are defined in the Java core API, use of `synchronized` methods, and use of other methods related to synchronization and concurrency. [9] Later, Bradbury *et al.* proposed a new set of 24 mutation operators for concurrent Java (J2SE 5.0) within five categories: modify parameters of concurrent method, modify the occurrence of concurrency method calls, modify keywords, switch concurrent objects, and modify critical region. [5] Bradbury *et al.* used a subset of these mutation operators and the ExMAn mutation analysis framework to assess the IBM tool ConTest and performed model checking with Java PathFinder on four selected Java example programs. [6][4] For comparison and further study, we have listed the mutation operators proposed by Delamaro *et al.* and Bradbury *et al.* in Table I.

## II. Synchronization-Centric Mutation Operators

Based on our study on mutation for concurrent Java programs, we proposed a new set of first and second-order synchronization-centric mutation operators for mutant

Table I
PREVIOUS CONCURRENCY MUTATION OPERATORS FOR JAVA

| Bradbury[5] | Delamaro[9] | Description |
|---|---|---|
| MXT | ReplWait, IncrDecrWait | Modify time parameter `t` of `wait(t)`, `sleep(t)`, `join(t)`, `await(t)` |
| MSP | ReplSyncObject | Modify parameter `obj` of block `synchronized(obj){...}` |
| ESP | SwitchArg | Exchange parameter `obj` of block `synchronized(obj){...}` |
| MSF | N/A | Modify Semaphore Fairness |
| MXC | N/A | Modify Permit Count in Semaphore and Modify Thread Count in Latches and Barriers |
| MBR | N/A | Modify Barrier Runnable Parameter |
| RTXC | DelNotify, DelWait | Remove Thread Call `wait()`, `join()`, `sleep()`, `yield()`, `notify()`, `notifyAll()` |
| RCXC | N/A | Remove Concurrency Call (methods in Locks, Semaphores, Latches, Barriers, etc.) |
| RNA | ReplNotify | Replace `notifyAll()` with `notify()` |
| RJS | N/A | Replace `join()` with `sleep()` |
| ELPA | N/A | Exchange Lock/Permit Acquisition |
| EAN | N/A | Exchange Atomic Call with Non-Atomic |
| ASTK | N/A | Add `static` Keyword to `synchronized` Method |
| RSTK | N/A | Remove `static` Keyword from `synchronized` Method |
| RSK | DelSync | Remove `synchronized` Keyword from Method |
| RSB | N/A | Remove `synchronized` block |
| RVK | N/A | Remove `volatile` Keyword |
| RFU | N/A | Remove `finally` Around Unlock |
| RXO | N/A | Replace One Concurrency Mechanism-X with Another (Locks, Semaphores, etc.) |
| EELO | N/A | Exchange Explicit Lock Objects |
| SHCR | MoveBrace | Shift Critical Region |
| SKCR | N/A | Shrink Critical Region |
| EXCR | N/A | Expand Critical Region |
| SPCR | N/A | Split Critical Region |
| N/A | DelStat | Deletes a statement from a `synchronized` block |
| N/A | ReplArg | Replaces argument with constant in a `synchronized` method |
| N/A | DelSyncCall | Deletes a call to a `synchronized` method |
| N/A | ReplMeth | Uses method with same name and other signature |
| N/A | InsNegArg | Inserts unary (negation) operators in an argument |
| N/A | ReplTargObj | Replaces the object in a call to `synchronized` method |

generation.

### A. First-Order Concurrency Mutation Operators for Java

We first selected several concurrency mutation operators based on their relevance to synchronization. These operators are listed in Table II.

### B. Second-Order Concurrency Mutation Operators for Java

Polo *et al.* studied ways to decrease the cost of mutation testing with second-order mutants. [24] They also described the potential problem of creating huge amounts of mutants, thus leading to higher computing cost, because of second-order mutation. Under a brute force second-order mutation strategy, *N* number of mutation operators may lead to *N*N* number of second-order mutants. Jia *et al.* described the general case of higher order mutation and three reduction algorithms: greedy, genetic and hill climbing algorithm along with specially constructed *fitness* function. [17]

Table II
FIRST-ORDER CONCURRENCY MUTATION OPERATORS FOR JAVA

| | | |
|---|---|---|
| sync meth | RKSN | Remove `synchronized` Keyword |
| | RCSN | Remove Call to `synchronized` Method |
| | RSSN | Remove a Statement from `synchronized` Method |
| | AKST | Add `static` Keyword to `synchronized` Method |
| | MASN | Modify `synchronized` Argument with Constant |
| sync bloc | RSNB | Remove `synchronized` Block |
| | RSSB | Remove a statement from `synchronized` Block |
| | MOSB | Modify `synchronized` Object |
| | EOSB | Exchange `synchronized` Object |
| | MVSB | Move Statement(s) Out of `synchronized` Block |
| misc | RMWN | Remove `wait()`, `notify()`, `nofityAll()` |
| | RMSV | Remove `static`, `volatile` Keywords |

We constructed second-order concurrency mutation operators by fixing one of the two operators to be a `synchronized` block or method modification and the other to perform code changes related to the first `synchronized` block or method. Some subtle concurrency bugs can be generated using such second-order mutation operators. These operators are presented in Table III.

Table III
SECOND-ORDER CONCURRENCY MUTATION OPERATORS FOR JAVA

| | |
|---|---|
| RKSN+RSSN | Remove `synchronized` Keyword and a Statement from `synchronized` Method |
| AKST+MASN | Add `static` Keyword and Modify Argument with Constant to `synchronized` Method |
| RKSN+MASN | Remove `synchronized` Keyword and Modify Argument with Constant |
| RSNB+RSSB | Remove `synchronized` Block and a Statement from `synchronized` Block |
| MOSB+RSSB | Modify `synchronized` Object and Remove a Statement from `synchronized` Block |
| MOSB+MVSB | Modify `synchronized` Object and Move Statement(s) Out of `synchronized` Block |

## III. IMPLEMENTATION

For empirical study purpose, we developed an Eclipse Plug-in [13] named *BugGen* that is able to automatically generate mutants using selected mutation operators. Eclipse is a popular multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. Building the *BugGen* as an Eclipse Plug-in leverages the functionalities and user-friendly platform that Eclipse provides, thus reduces GUI development time.

## IV. EXAMPLE JAVA PROGRAMS

In order to study the quantity of mutants generated, as well as the cost and effectiveness of each proposed concurrency mutation operator, we used the following four example programs in our experiments:

- Webserver, a Java web server program that supports concurrent client connections and synchronization. It

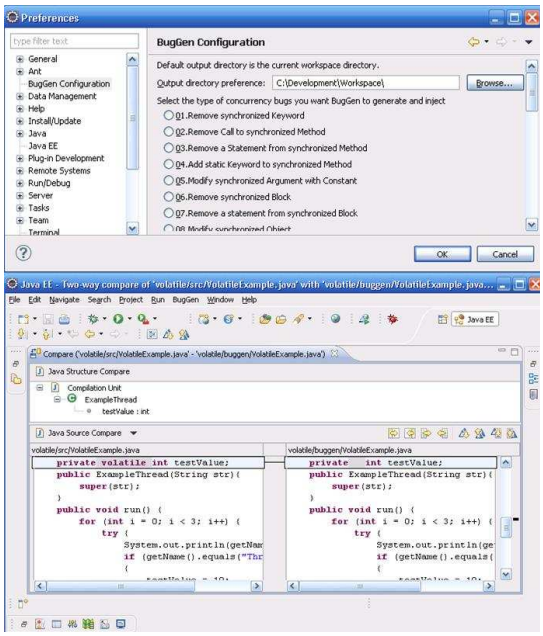


Figure 1. BugGen Eclipse Plug-in

was used by Aldrich *et al.* and its source code can be found in their paper's appendix section. [3]

- Chat, a Java chat program that supports multiple clients exchanging messages. Threads listen to the network and make connections whenever necessary. [12]
- Miasma, a graphical Java applet program from the NIH web-site that generates an animated display by summing four sine waves into an array. It supports synchronization and has `wait(t)` for prior pixels to be accepted before triggering another one. [19]
- LinkedList, a modified Java program from the IBM concurrency benchmark programs repository. [14] The original program was developed to emulate the concurrency bug in using Java linked list, which is a non-synchronized collection. The program was modified by adding one `synchronized` keyword to one method, following the suggestion commented in the benchmark to fix the bug.

The source code of the above example programs can also be found at the appendix section of this paper. The basic statistical information for each program's source code is listed in Table IV.

Table IV
EXAMPLE PROGRAMS USED IN OUR EXPERIMENTS

| Program Name | LOC | classes | sync methods | sync blocks |
|---|---|---|---|---|
| Webserver | 125 | 6 | 11 | 2 |
| Chat | 482 | 4 | 10 | 2 |
| Miasma | 360 | 1 | 0 | 2 |
| LinkedList | 421 | 5 | 1 | 1 |
| Total | 1,388 | 16 | 22 | 7 |

## V. RESULTS AND ANALYSIS

In our experiments, we applied each of the mutation operators listed in Table I, II and III on the example programs and counted how many mutants were generated by each operator for each program. Our quantitative data and summations for each category are recorded in the appendix section.

Our empirical study and analysis of mutant generation for concurrent Java programs shows that there are certain limitations in previously proposed mutation operators. The first problem we found is that almost half of proposed mutation operators are not effective in generating mutants because they generate very few or zero mutants. Our study also shows that synchronization-centric mutation operators generate the most mutants. In addition, some mutation operators generate functionally equivalent mutants. Finally, some subtle concurrency bugs are not generated by these mutation operators at all. Our study also shows that the new set of first and second-order synchronization-centric mutation operators are more effective and applicable in mutant generation for concurrent Java programs.

## VI. CONCLUSION AND FUTURE WORK

This paper describes our implementation of a generic mutation testing framework and the results of applying three sets of concurrency mutation operators on four example Java programs through empirical study and analysis.

For future work, we plan to do further empirical studies, especially involving test suite comparison and evaluation, along with further investigations of the special characteristics of mutation testing for concurrent programs.

## APPENDIX

*A. Number of Mutants Generated (Delamaro & Bradbury)*

*B. Number of Mutants Generated (New)*

*C. Source Code of Webserver Program*

*D. Source Code of Chat Program*

*E. Source Code of Miasma Program*

*F. Source Code of LinkedList Program*

## ACKNOWLEDGMENT

The authors are members of the Programming Systems Laboratory, funded in part by NSF CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 1 U54 CA121852-01A1.

## REFERENCES

[1] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Mutation analysis," Georgia Institute of Technology, Atlanta, Georgia, Tech. Rep. GIT-ICS-79/08, 1979.

[2] B. K. Aichernig and C. C. Delgado, "From faults via test purposes to test cases: On the fault-based testing of concurrent systems," in *9th International Conference on Fundamental Approaches to Software Engineering (FASE '06)*. Vienna, Austria: Springer-Verlag, 2006, pp. 324–338.

[3] J. Aldrich, E. G. Sirer, C. Chambers, and S. J. Eggers, "Comprehensive synchronization elimination for java," *Sci. Comput. Program.*, vol. 47, no. 2-3, pp. 91–120, —2003—, 772608.

[4] J. S. Bradbury, J. R. Cordy, and J. Dingel, "Exman: A generic and customizable framework for experimental mutation analysis," in *Mutation Analysis, 2006. Second Workshop on*, 2006, pp. 4–4.

[5] ——, "Mutation operators for concurrent java (j2se 5.0)," in *Proceedings of the Second Workshop on Mutation Analysis*. IEEE Computer Society, 2006, pp. 11–11.

[6] ——, "Comparative assessment of testing and model checking using program mutation," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007*. IEEE Computer Society, 2007, 1308285210-222.

[7] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings of the AFIPS National Computer Conference*, Anaheim, New Jersey, 5-8 June 1978, pp. 623–627.

[8] R. Carver, "Mutation-based testing of concurrent programs," in *Test Conference, 1993. Proceedings., International*, 1993, pp. 845–853.

[9] M. Delamaro, M. Pezz, A. M. R. Vincenzi, and J. C. Maldonado, "Mutant operators for testing concurrent java programs," in *XV Simpsio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, Brasil, 2001, pp. 272 – 285.

[10] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, and K. N. King, "An extended overview of the mothra software testing environment," in *Software Testing, Verification, and Analysis, 1988., Proceedings of the Second Workshop on*, 1988, pp. 142–151.

[11] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[12] D. J. Eck, "Chat," 2006, available at `http://math.hws.edu/eck/cs124/s06/lab11/index.html`.

[13] Eclipse, "Eclipse.org," 2010, available at `http://www.eclipse.org`.

[14] Y. Eytani and S. Ur, "Compiling a benchmark of documented multi-threaded bugs," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 266.

[15] S. Ghosh, "Towards measurement of testability of concurrent object-oriented programs using fault insertion: a preliminary investigation," in *Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on*, 2002, pp. 17–25.

[16] R. G. Hamlet, "Testing programs with the aid of a compiler," *Software Engineering, IEEE Transactions on*, vol. SE-3, no. 4, pp. 279–290, 1977.

[17] Y. Jia and M. Harman, "Constructing subtle faults using higher order mutation testing," in *Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on*, 2008, pp. 249–258.

[18] ——, "An analysis and survey of the development of mutation testing," *IEEE Transactions of Software Engineering*, 2010.

[19] JRP, "Miasma," 2010, available at `http://rsb.info.nih.gov/miasma/Miasma.java`.

[20] B. Long, R. Duke, D. Goldson, P. Strooper, and L. Wildman, "Mutation-based exploration of a method for verifying concurrent java components," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 265.

[21] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 1, pp. 5–20, 1992, 125473.

[22] A. J. Offutt and K. N. King, "A fortran 77 interpreter for mutation analysis," *ACM SIGPLAN Notices*, vol. 22, no. 7, pp. 177–188, July 1987.

[23] A. J. Offutt and R. H. Untch, "Mutation 2000: uniting the orthogonal," *Mutation testing for the new century*, pp. 34–44, 2001, 571314.

[24] M. Polo, M. Piattini, and I. Garcia-Rodriguez, "Decreasing the cost of mutation testing with second-order mutants," *Softw. Test. Verif. Reliab.*, vol. 19, no. 2, pp. 111–131, 2009, 1552140.

**Appendix A. Number of Mutants Generated (Delamaro & Bradbury)**

| Mutation Operators | Webserver | Chat | Miasma | LinkedList | Total |
|---|---|---|---|---|---|
| MXT | 0 | 0 | 1 | 0 | 1 |
| MSP | 2 | 2 | 2 | 1 | 7 |
| ESP | 1 | 1 | 1 | 0 | 3 |
| MSF | 0 | 0 | 0 | 0 | 0 |
| MXC | 0 | 0 | 0 | 0 | 0 |
| MBR | 0 | 0 | 0 | 0 | 0 |
| RTXC | 0 | 2 | 3 | 2 | 7 |
| RCXC | 0 | 0 | 0 | 0 | 0 |
| RNA | 0 | 0 | 1 | 0 | 1 |
| RJS | 0 | 0 | 0 | 2 | 2 |
| ELPA | 0 | 0 | 0 | 0 | 0 |
| EAN | 0 | 0 | 0 | 0 | 0 |
| ASTK | 11 | 10 | 0 | 1 | 22 |
| RSTK | 0 | 0 | 0 | 0 | 0 |
| RSK | 11 | 10 | 0 | 1 | 22 |
| RSB | 2 | 2 | 2 | 1 | 7 |
| RVK | 0 | 7 | 0 | 0 | 7 |
| RFU | 0 | 0 | 0 | 0 | 0 |
| RXO | 0 | 0 | 0 | 0 | 0 |
| EELO | 0 | 0 | 0 | 0 | 0 |
| SHCR | 0 | 3 | 2 | 2 | 7 |
| SKCR | 2 | 2 | 5 | 2 | 11 |
| EXCR | 4 | 8 | 4 | 1 | 17 |
| SPCR | 0 | 0 | 3 | 1 | 4 |
| DelStat | 2 | 4 | 7 | 2 | 15 |
| ReplArg | 7 | 25 | 0 | 2 | 34 |
| DelSyncCall | 11 | 10 | 0 | 1 | 22 |
| ReplMeth | 11 | 10 | 0 | 1 | 22 |
| InsNegArg | 0 | 2 | 0 | 0 | 2 |
| ReplTargObj | 4 | 0 | 0 | 0 | 4 |

**Appendix B. Number of Mutants Generated (New)**

| Mutation Operators | Webserver | Chat | Miasma | LinkedList | Total |
|---|---|---|---|---|---|
| RKSN | 11 | 10 | 0 | 1 | 22 |
| RCSN | 11 | 10 | 0 | 1 | 22 |
| RSSN | 21 | 35 | 0 | 3 | 59 |
| AKST | 11 | 10 | 0 | 1 | 22 |
| MASN | 7 | 25 | 0 | 2 | 34 |
| RSNB | 2 | 2 | 2 | 1 | 7 |
| RSSB | 2 | 4 | 7 | 2 | 15 |
| MOSB | 2 | 2 | 2 | 1 | 7 |
| EOSB | 1 | 1 | 1 | 0 | 3 |
| MVSB | 2 | 2 | 5 | 2 | 11 |
| RMWN | 0 | 2 | 3 | 2 | 7 |
| RMSV | 0 | 7 | 0 | 0 | 7 |
| RKSN+RSSN | 21 | 35 | 0 | 3 | 59 |
| AKST+MASN | 7 | 25 | 0 | 2 | 34 |
| RKSN+MASN | 7 | 25 | 0 | 2 | 34 |
| RSNB+RSSB | 2 | 4 | 7 | 2 | 15 |
| MOSB+RSSB | 2 | 4 | 7 | 2 | 15 |
| MOSB+MVSB | 2 | 4 | 11 | 2 | 19 |

```
Appendix C. Source Code of Webserver Program


class Pair {
    private Object first;
    private Object second;
    public Pair(Object f, Object s) {
        first = f; second = s; }
    public synchronized Object getFirst() {
        return first; }
    public synchronized Object getSecond() {
        return second; }
    public synchronized void setFirst(Object f)
        { first = f; }
    public synchronized void setSecond(
        Object s) { second = s; }
}

class Table {
    private List entries[];
    private int capacity;
    public Table() {
        capacity = 13587;
        entries = new List[capacity];
        for (int i = 0; i < capacity; ++i)
            entries[i] = new List();
    }
    public synchronized Object get(Object key) {
        return getEntry(key).getSecond();
    }
    public synchronized void put(Object key,
            Object value) {
        Pair entry = getEntry(key);
        entry.setSecond(value);
    }
    private synchronized Pair getEntry(Object
            key) {
        int index = key.hashCode() % capacity;
        List l = entries[index];
        l.reset();
        while (l.hasMore()) {
            Pair p = (Pair) l.getNext();
            if (p.getFirst().equals(key))
            return p;
        }
        Pair p = new Pair(key, null);
        l.add(p);
        return p;
        }
}

class List {
    private Pair first;
    private Pair current;
    public synchronized void reset() {
        current = first; }
    public synchronized boolean hasMore() {
        return current != null; }
    public synchronized Object getNext() {
        if (current != null) {
            Object value = current.getFirst();
            current = (Pair) current.getSecond();
            return value;
        }
        else
    }
    public synchronized void add(Object o) {
        first = new Pair(o, first); }
}
class WriterThread extends Thread {
    public void run() {
```

```
72          int myMaxNumber = 100;
73          while (myMaxNumber < 10000) {
74              for (int i = 0; i < 100; ++i) {
75                  Webserver.dataTable.put(
76                      new Integer(myMaxNumber),
77                      String.valueOf(myMaxNumber));
78                  myMaxNumber++;
79              }
80              synchronized(Webserver.maxNumberLock) {
81                  Webserver.maxNumber = myMaxNumber;
82              }
83          }
84          System.out.println("Writer complete");
85      }
86  }
87
88  class ReaderThread extends Thread {
89      public void run() {
90          int myMaxNumber;
91          Random rand = new Random();
92          for (int i = 0; i < 1000; ++i) {
93              synchronized(Webserver.maxNumberLock) {
94                  myMaxNumber = Webserver.maxNumber;
95              }
96              for (int j = 0; j < 100; ++j) {
97                  int index = Math.abs(
98                      rand.nextInt()) % myMaxNumber;
99                  Webserver.dataTable.get(
100                     new Integer(index));
101             }
102         }
103         System.out.println("Reader complete");
104     }
105 }
106
107 public class Webserver {
108     public static void main(String args[]) {
109         /* set up data table */
110         maxNumber = 100;
111         dataTable = new Table();
112         maxNumberLock = new Object();
113         for (maxNumber = 0; maxNumber < 100;
114                 ++maxNumber) {
115             dataTable.put(new Integer(maxNumber),
116                 String.valueOf(maxNumber));
117         }
118         for (int threadNum = 0; threadNum < 8;
119                 ++threadNum) {
120             new ReaderThread().start();
121         }
122         new WriterThread().start();
123     }
124     public static Table dataTable;
125     public static int maxNumber;
126     public static Object maxNumberLock;
127 }
128
```

```
Appendix D. Source Code of Chat Program


import javax.swing.JFrame;

/**
 * Runs a program that opens a "simple network chat" window that
 * supports two-way connections.
 */
public class Chat {

    /**
     * Main program just creates a JFrame that shows a ChatPanel,
     * and makes that window visible on the screen.
     */
    public static void main(String[] args) {
        JFrame window = new JFrame("Simple Network Chat");
        window.setContentPane( new ChatPanel() );
        window.pack();
        window.setLocation(100,50);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
    }

}


import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class ChatPanel extends JPanel implements ActionListener {

    JTextField inputBox;   // User types messages here.
    JButton sendButton;    // Message is sent when user clicks this button.
    JTextArea transcript;  // All messages are posted here.

    public ChatPanel() {

        sendButton = new JButton("Send");

        sendButton.addActionListener(this);  // A button-click will cause the actionPerformed()

        inputBox = new JTextField(40);       // Create an input box sized to hold 40 characters
        inputBox.setBackground(Color.WHITE);
        inputBox.addActionListener(this);    // Pressing return in the input box will call acti

        transcript = new JTextArea();
        transcript.setLineWrap(true);       // Lines will wrap at the right margin.
        transcript.setWrapStyleWord(true);  // Line wrap will not split words.
        transcript.setEditable(false);      // User can't type in the transcript area.
        transcript.setBackground(Color.WHITE);
        JScrollPane scroller = new JScrollPane(transcript);  // Required for adding a scrollbar

        setPreferredSize( new Dimension(650,450));  // This is the size that will be used for t
        setBackground(Color.GRAY);                  // This color will show between components
        setLayout(new BorderLayout(5,5));           // Use a border layout with 5-pixel gaps be

        JPanel bottom = new JPanel();      // Create a panel to hold the input box and send but
        bottom.setBackground(Color.GRAY);
        bottom.add(inputBox);              // Add the input box to the bottom panel.
        bottom.add(sendButton);            // Add the send button to the bottom panel.

        add(bottom, BorderLayout.SOUTH);     // Puts the bottom panel at the bottom (SOUTH) of
        add(scroller, BorderLayout.CENTER);  // Puts the transcript (in its scroller) in the ma

    } // end constructor


    /**
```

```
72       * Adds a string to the transcript area, followed by a carriage return.
73       * @param s the string to be added to the transcript.
74       */
75      synchronized private void postMessage(String s) {
76          transcript.append(s + "\n");
77             // The following line is a nasty kludge that was the only way I could find to force
78             // the transcript to scroll so that the text that was just added is visible in
79             // the window.  Without this, text can be added below the bottom of the visible area
80             // of the transcript.
81          transcript.setCaretPosition(transcript.getDocument().getLength());
82      }
83
84
85      /**
86       * This method is called when an action event occurs in a component, assuming that
87       * this ChatPanel has been "registered" as an ActionListener with that component.
88       */
89      public void actionPerformed(ActionEvent e) {
90
91          Object source = e.getSource();  // This is the object that generated the event.
92
93          if (source == sendButton || source == inputBox) {
94              String str = inputBox.getText();
95              postMessage("SEND:  " + str);
96              inputBox.selectAll();
97              inputBox.requestFocus();
98          }
99
100     } // end actionPerformed()
101
102
103 }
104
105
106
107 /**
108  * This interface should be implemented by an object that wants to use
109  * a SimpleNet object for network connectivity.  A SimpleNet object must
110  * have an "observer" that implements this interface.  The SimpleNet
111  * object informs its observer about network events by calling methods
112  * defined in this interface.
113  */
114 public interface SimpleNetObserver {
115
116     /**
117      * This method is called when the connection has been successfully opened and is ready to
118      * be used for sending messages back and forth.
119      * @param connection the SimpleNet object that is managing the connection.  You can ignore
120      * this parameter unless you are using several SimpleNet objects and need to tell them apar
121      */
122     public void connectionOpened(SimpleNet connection);
123
124     /**
125      * This method is called whenever a message is received from the other side of the
126      * network connection.
127      * @param connection the SimpleNet object that is managing the connection.  You can ignore
128      * this parameter unless you are using several SimpleNet objects and need to tell them apar
129      * @param data the message that was received
130      */
131     public void connectionDataReceived(SimpleNet connection, String data);
132
133     /**
134      * This method is called when the connection closes because you called the close() method
135      * in the SimpleNet object.
136      * @param connection the SimpleNet object that is managing the connection.  You can ignore
137      * this parameter unless you are using several SimpleNet objects and need to tell them apar
138      */
139     public void connectionClosed(SimpleNet connection);
140
141     /**
142      * This method is called when the connection closes because of action taken on the other
```

```
143      * side of the network connection.
144      * @param connection the SimpleNet object that is managing the connection.  You can ignore
145      * this parameter unless you are using several SimpleNet objects and need to tell them apar
146      */
147     public void connectionClosedByPeer(SimpleNet connection);
148 
149     /**
150      * This method is called when the connection closes because of some sort of network error.
151      * Note that this method can be called when the connection is in the process of being opene
152      * @param connection the SimpleNet object that is managing the connection.  You can ignore
153      * this parameter unless you are using several SimpleNet objects and need to tell them apar
154      */
155     public void connectionClosedWithError(SimpleNet connection, String errorMessage);
156 
157 }
158 
159 
160 import java.io.*;
161 import java.net.*;
162 
163 
164 /**
165  * This class supports basic, text-based communication between two computers
166  * on the network.  For opening a connection, a SimpleNet object can run
167  * in either server mode -- where it waits for an incoming connection -- or in
168  * client mode -- where it tries to make a connection to a waiting server.  Use
169  * the listen() method to run as a server; use connect() to run as a client.
170  * Once the connection has been opened, it makes no difference whether server
171  * or client mode was used.  Lines of text can be transmitted in either
172  * direction.  Use the send() method to transmit a line of text.  Receiving
173  * messages is more complicated, since they can arrive asynchronously.  Also,
174  * a connection can be closed asynchronously from the other side.  And when
175  * operating in server mode, the connection is opened asynchronously.  To support
176  * asynchronous operation, a SimpleNet object must have an "observer" that
177  * implements the SimpleNetObserver interface.  This interface defines several
178  * methods that are called by the SimpleNet object when network events occur.
179  * See that interface for more information.
180  */
181 public class SimpleNet {
182 
183     /**
184      * Possible state that a SimpleNet object can be in.  The states are mostly used
185      * internally in this class, but you can find out the state by calling getState().
186      */
187     public final static int IDLE = 1;
188     public final static int WAITING_FOR_CONNECTION = 2;
189     public final static int CONNECTING = 3;
190     public final static int CONNECTED = 4;
191     public final static int CLOSING = 5;
192 
193     private SimpleNetObserver owner;
194     private int state;
195     private volatile PrintWriter out;
196     private ConnectionHandler connectionHandler; // ConnectionHandler is a nested class, define
197 
198     /**
199      * Create a SimpleNet object that can be used for basic two-way text-based network connecti
200      * @param observer A non-null object that implements the SimpleNetObserver interface.  When
201      * certain network events occur, the SimpleNet object will notify the observer by calling
202      * one of the methods defined in the interface.
203      * @throws IllegalArgumentException if observer is null
204      */
205     public SimpleNet(SimpleNetObserver observer) {
206         if (observer == null)
207             throw new IllegalArgumentException("A SimpleNet object requires a non-null SimpleNe
208         owner = observer;
209         state = IDLE;
210     }
211 
212     /**
213      * Open a connection in server mode.  The SimpleNet object will wait for an incoming connec
```

```
214       * request.  This method returns immediately, without waiting for the connection to open.  T
215       * observer will be notified when the connection opens by calling its connectionOpened() me
216       * (Or, if an error occurs, its connectionClosedWithError() method will be called instead.)
217       * @param portNumber the port number on which the server will listen.  A network server mus
218       * have a port number in the range 1 to 65535.  Numbers less than 1024 are reserved for sys
219       * use, and attempting to use one will produce an error.  It is also an error to try to use
220       * a port number that is already being used by another server.
221       * @throws IllegalStateException if this SimpleNet object is already connected or opening a
222       */
223      synchronized public void listen(int portNumber) {
224          if (state != IDLE)
225              throw new IllegalStateException("Attempt to open a connection while not in idle sta
226          state = WAITING_FOR_CONNECTION;
227          connectionHandler = new ConnectionHandler(portNumber);
228          connectionHandler.start();
229      }
230
231      /**
232       * Open a connection in client mode.  The SimpleNet object will appempt to connect to a ser
233       * that is listening on a specified computer and at a specified port number.  This method r
234       * immediately, without waiting for the connection to open.  The observer will be notified
235       * when the connection opens by calling its connectionOpened() method.  (Or, if an error oc
236       * its connectionClosedWithError() method will be called instead.)
237       * @param hostNameOrIP  the host name (such as "math.hws.edu") or IP address (such as "172.
238       * of the computer when the server is listeneing
239       * @param portNumber the port number where the server is listening
240       * @throws IllegalStateException if this SimpleNet object is already connected or opening a
241       */
242      synchronized public void connect(String hostNameOrIP, int portNumber) {
243          if (state != IDLE)
244              throw new IllegalStateException("Attempt to open a connection while not in idle sta
245          state = CONNECTING;
246          connectionHandler = new ConnectionHandler(hostNameOrIP,portNumber);
247          connectionHandler.start();
248      }
249
250      /**
251       * Closes the current connection, if any.  This method returns immediately.
252       * The observer will be notified when the connection closes by calling its
253       * connectionClosed() method.  Note that this method can be called while
254       * the connection is still in the process of being opened.  In that case,
255       * the connection attempt will be aborted; the connectionClosed()
256       * method in the observer will still be called in this case.
257       */
258      synchronized public void close() {
259          if (state == IDLE)
260              return;  // ignore close command if there is no connection
261          state = CLOSING;
262          connectionHandler.abort();
263      }
264
265      /**
266       * Transmit a message to the other side of the connection.  Attempts to
267       * transmit data when no connection is opened are simply ignored.
268       * @param message the text to be transmited
269       */
270      synchronized public void send(String message) {
271          if (out == null || out.checkError() || state != CONNECTED)
272              return;
273          out.println(message);
274          out.flush();
275          if (out.checkError())
276              close();
277      }
278
279      /**
280       * Returns the current state of this SimpleNet object.  For the most part,
281       * the state information is not needed outside this class, but you can use
282       * this method to inquire the current state if you need it.
283       * @return  the current state, which is one of the contstants
284       * IDLE, CONNECTING, WAITING_FOR_CONNECTION, CONNECTED, or CLOSING,
```

```
285       */
286      synchronized public int getState() {
287          return state;
288      }
289
290
291      //-------------------------------------------------------------------------
292      // The remainder of this file is the private implementation part of the class.
293      //-------------------------------------------------------------------------
294
295      synchronized private void dataReceived(ConnectionHandler handler, String data) {
296          if (state == IDLE || state == CLOSING || handler != connectionHandler)
297              return; // ignore possible input from old connection
298          owner.connectionDataReceived(this,data);
299      }
300
301      synchronized void opened(ConnectionHandler handler) {
302          if (handler != connectionHandler || (state != WAITING_FOR_CONNECTION && state != CONNEC
303              return;
304          out = connectionHandler.getOutputStream();
305          state = CONNECTED;
306          owner.connectionOpened(this);
307      }
308
309      synchronized void closed(ConnectionHandler handler) {
310          if (state == IDLE || handler != connectionHandler)
311              return;
312          if (state == CLOSING)
313              owner.connectionClosed(this);
314          else
315              owner.connectionClosedByPeer(this);
316          connectionHandler = null;
317          out = null;
318          state = IDLE;
319      }
320
321      synchronized private void error(ConnectionHandler handler, String message, Exception e) {
322          if (state == IDLE || connectionHandler != handler)
323              return;  // Ignore any left-over error from old connections.
324          out = null;
325          connectionHandler = null;
326          if (state == CLOSING) { // don't send error since owner wants to close anyway
327              state = IDLE;
328              owner.connectionClosed(this);
329          }
330          else {
331              state = IDLE;
332              owner.connectionClosedWithError(this,message + ":  " + e.toString());
333          }
334          // e.printStackTrace();
335      }
336
337      private class ConnectionHandler extends Thread {
338
339          private int port;
340          private String host;
341          private boolean runAsServer;
342
343          private volatile Socket connection;
344          private volatile ServerSocket listener;
345          private volatile boolean aborted;
346          private volatile Thread connectionOpener;
347          private volatile Exception exceptionWhileConnecting;
348          private volatile PrintWriter out;
349          private BufferedReader in;
350
351          ConnectionHandler(int portNumber) {
352              port = portNumber;
353              runAsServer = true;
354          }
355
```

```
356         ConnectionHandler(String hostName, int portNumber) {
357             host = hostName;
358             port = portNumber;
359             runAsServer = false;
360         }
361 
362         void abort() {
363             try {
364                 if (listener != null)
365                     listener.close();
366                 else if (connectionOpener != null)
367                     this.interrupt();
368                 else if (connection != null) {
369                     connection.shutdownInput();
370                     connection.shutdownOutput();
371                     connection.close();
372                 }
373             }
374             catch (Exception e) {
375             }
376             aborted = true;
377         }
378 
379         PrintWriter getOutputStream() {
380             return out;
381         }
382 
383         public void run() {
384             BufferedReader in;
385 
386             if (!runAsServer) {
387                 try {
388                     connectionOpener = new Thread() {
389                         public void run() {
390                             try {
391                                 try {
392                                     connection = new Socket(host,port);
393                                 }
394                                 catch (Exception e) {
395                                     exceptionWhileConnecting = e;
396                                 }
397                                 finally {
398                                     synchronized(this) {
399                                         notify();
400                                     }
401                                 }
402                             }
403                             catch (Exception e) {
404                                 connection = null;
405                             }
406                         }
407                     };
408                     connectionOpener.start();
409                     synchronized(connectionOpener) {
410                         try {
411                             connectionOpener.wait();
412                         }
413                         catch (InterruptedException e) {
414                             closed(this);
415                             return;
416                         }
417                     }
418                     connectionOpener = null;
419                     if (exceptionWhileConnecting != null)
420                         throw exceptionWhileConnecting;
421                 }
422                 catch (Exception e) {
423                     error(this,"Error while attempting to connect to " + host, e);
424                     return;
425                 }
426             }
```

```
427             else {
428                 try {
429                     listener = new ServerSocket(port);
430                     connection = listener.accept();
431                     listener.close();
432                     listener = null;
433                 }
434                 catch (Exception e) {
435                     error(this,"Error while waiting for connection request", e);
436                     return;
437                 }
438             }
439 
440             if (aborted)
441                 return;
442 
443             try {
444                 in = new BufferedReader(new InputStreamReader( connection.getInputStream() ));
445                 out = new PrintWriter(connection.getOutputStream());
446             }
447             catch (Exception e) {
448                 error(this,"Error while creating network input/ouput streams", e);
449                 return;
450             }
451 
452             if (aborted)
453                 return;
454 
455             opened(this);
456 
457             try {
458                 while (true) {
459                     String input = in.readLine();
460                     if (input == null || aborted)
461                         break;
462                     dataReceived(this,input);
463                 }
464             }
465             catch (Exception e) {
466                 error(this,"An error occured while connected", e);
467             }
468             finally {
469                 closed(this);
470                 if (connection != null) {
471                     try {
472                         connection.close();  // Make sure connection is properly closed.
473                     }
474                     catch (Exception e) {
475                     }
476                 }
477             }
478 
479         } // end run()
480 
481     } // end nested class ConnectionHandler
482 
483 }
484 
```

```
Appendix E. Source Code of Miasma Program


/*
Miasma is based on the Plasma applet.
Modifications by J. Random Programmer.
Modifications released into the public domain.

I've fixed a number of flaws and bugs, and added features to break
out the individual elements that contribute to overall speed.
The original Plasma applet is public domain, and so is this.

Do what you want with Miasma, but DON'T SEND ME NUMBERS because I don't care.
If you care, then go ahead and post numbers, but that doesn't mean I have to care.
And, no, I don't have to explain why I don't care.
  -- JRP

From the original Plasma applet:
This applet creates an animated display by summing four
sine waves into an array. Example FPS rates are at
http://rsb.info.nih.gov/plasma.
It is based on "Sam's Java Plasma Applet"
(http://www.dur.ac.uk/~d405ua/Plasma.html) by Sam Marshall
(t-sammar@microsoft.com). It was modified to use 8-bit images
by Menno van Gangelen (M.vanGangelen@element.nl). Improved
frame rate calculation and code for using MemoryImageSource.setAnimated()
contributed by andy@mindgate.net.
*/

import java.awt.*;
import java.awt.image.*;

public class Miasma
  extends java.applet.Applet
 implements Runnable
{
    private Image img;
    private Thread runThread;
    private long first;
    private int frames, fps;
    private int width, height;
    private int w,h,size;
    private int scale;
    private boolean showFPS;
    private IndexColorModel icm;
    private int[] waveTable;
    private byte[] pixels;
    private MemoryImageSource source;

    private boolean draw;
    private int deliver;
    private boolean filter;
    private boolean sync;
    private int pri;

    private boolean useRGB;
    private int[] pixelsRGB;
    private int[] mapRGB;

    private final boolean[] pending;
    private String strFPS;

    int framesIndex, past;
    int framesSum, elapsedSum;
    int[] framesPast, elapsedPast;

    public
    Miasma()
    {
        pending = new boolean[ 1 ];
        strFPS = "";
```

```
72      }
73
74
75      public void init()
76      {
77          scale = getInt( "scale", 2 );
78          showFPS = getBoolean( "showfps", true );
79
80          pri = getInt( "pri", Thread.MIN_PRIORITY );
81          draw = getBoolean( "draw", true );
82          deliver = getInt( "deliver", -1 );
83          filter = getBoolean( "filter", false );
84          useRGB = getBoolean( "rgb", false );
85          sync = getBoolean( "sync", false );
86
87          int avg = getInt( "avg", 10 );
88          framesPast = new int[ avg ];
89          elapsedPast = new int[ avg ];
90
91          width = size().width;
92          height = size().height;
93          w = width/scale;
94  //      h = w;  // from Plasma original.  Huh?
95          h = height/scale;
96          pixels = new byte[ w * h ];
97          pixelsRGB = new int[ pixels.length ];
98          size = ((w+h)/2) * 4;
99          waveTable = new int[size];
100         calculateWaveTable();
101         calculatePaletteTable();
102     }
103
104     private boolean
105     getBoolean( String name, boolean defaultValue )
106     {
107         String val = getParameter( name );
108         if ( val == null )
109             return ( defaultValue );
110         else
111             return ( "true".equals( val ) );
112     }
113
114     private int
115     getInt( String name, int defaultValue )
116     {
117         String val = getParameter( name );
118         if ( val == null )
119             return ( defaultValue );
120         else
121             return ( Integer.parseInt( val ) );
122     }
123
124     private void
125     calculateWaveTable()
126     {
127         double perStep = (2 * Math.PI) / size;
128         for ( int i = 0;  i < size;  ++i )
129         {  waveTable[ i ] = (int) (32 * (1 + Math.sin( i * perStep )));  }
130     }
131
132     private void
133     calculatePaletteTable()
134     {
135         mapRGB = new int[ 256 ];
136
137         // All G components are 0 in palette, so do nothing to fill the 'gg' array.
138         int r, b;
139         byte[] rr = new byte[ 256 ];
140         byte[] gg = new byte[ 256 ];
141         byte[] bb = new byte[ 256 ];
142
```

```
143         // To ensure that the RGB image and the indexed image look different,
144         // the RGB one shows red/green gradients vs. the indexed one's red/blue gradients.
145         // Do this simply by using 'b' as a G component, not a B, in the 24-bit mapRGB values.
146         for ( int i = 0;  i < 128;  i++ )
147         {
148             rr[ i ] = rr[ 255 - i ] = (byte) (r = i + i + 1);
149             bb[ i ] = bb[ 255 - i ] = (byte) (b = 0xFF & -r);
150             mapRGB[ i ] = mapRGB[ 255 - i ] = (r << 16) | (b << 8);
151         }
152         icm = new IndexColorModel( 8, 256, rr, gg, bb );
153     }
154 
155 
156     public void start()
157     {
158 //      System.out.println( "codebase = " + getCodeBase() );
159 //      System.out.println( " docbase = " + getDocumentBase() );
160 
161         // Defer creation of Images and MemoryImageSources until the last possible moment.
162         // Use source's state as representative of all image-related variables.
163         if ( source == null )
164         {
165             if ( useRGB )
166             {
167                 // could use this.getColorModel() or ColorModel.getRGBdefault() or Toolkit.getC
168                 ColorModel modelRGB = getColorModel();
169 //              System.out.println( "ColorModel: " + modelRGB );
170                 source = new MemoryImageSource( w, h, modelRGB, pixelsRGB, 0, w );
171             }
172             else
173             {
174                 // source is indexed image, with bytes for pixels.
175                 source = new MemoryImageSource( w, h, icm, pixels, 0, w );
176             }
177             source.setAnimated( true );
178             source.setFullBufferUpdates( true );
179 
180             ImageProducer producer = source;
181             if ( filter )
182             {
183                 // The filter rescales to original size, so drawImage() won't.
184                 // You could use a different class of rescaling filter, to measure its effect o
185                 producer = new FilteredImageSource( producer, new ReplicateScaleFilter( width, 
186             }
187 
188 //          img = Toolkit.getDefaultToolkit().createImage( producer );
189             img = createImage( producer );
190         }
191 
192         if ( runThread == null )
193         {
194             for ( int i = 0;  i < framesPast.length;  ++i )
195             {  framesPast[ i ] = elapsedPast[ i ] = 0;  }
196             framesIndex = past = framesSum = elapsedSum = frames = 0;
197             first = System.currentTimeMillis();
198 
199             pending[ 0 ] = false;
200 
201             runThread = new Thread( this );
202             runThread.setPriority( pri );
203             runThread.setDaemon( true );
204             runThread.start();
205         };
206     }
207 
208     public void stop()
209     {
210         if  ( runThread != null )
211         {
212             runThread.interrupt();
213             runThread = null;
```

```
214         }
215     }
216
217     public void update(Graphics g)
218     {
219         // Deliver source's pixels to Image, if needed.
220         // An int started at -1 will reach 0 after 49.7 days at 1000 fps,
221         // or 497 days at 100 fps, etc.
222         if ( deliver != 0 )
223         {
224             source.newPixels();
225             --deliver;
226         }
227
228         // Signal acceptance after new pixels delivered, but before drawing occurs.
229         // This maximizes concurrency between producer and consumer threads,
230         // while ensuring that every calculated frame is accepted and handled.
231         acceptDelivery();
232
233         if ( draw )
234             g.drawImage( img, 0, 0, width, height, null );
235
236         ++frames;
237         if ( showFPS )
238             calculateFPS( System.currentTimeMillis(), g );
239     }
240
241
242     private void
243     calculateFPS( long now, Graphics g )
244     {
245         if ( now > first + 1000L )
246         {
247             fps = frames;
248             frames = 0;
249
250             // Elapsed millis should never overflow the capacity of an int (~2Msecs).
251             int elapsed = (int) (now - first);
252             first = now;
253
254             int n = framesIndex;
255             framesSum = framesSum - framesPast[ n ] + fps;
256             elapsedSum = elapsedSum - elapsedPast[ n ] + elapsed;
257             framesPast[ n ] = fps;
258             elapsedPast[ n ] = elapsed;
259             framesIndex = (n + 1) % framesPast.length;
260
261             // Calculate scaled up by 10, to get an extra decimal digit of precision to display
262             // This is reasonable considering the precision of past readings, and the averaging
263             n = (framesSum * 10000) / elapsedSum;
264
265             if ( past < framesPast.length )
266                 ++past;
267
268             strFPS = fps + " fps, avg " + (n/10) + "." + (n%10) + " over " + past + " sec";
269             showStatus( strFPS );
270         }
271
272         // ## This display now taken over by showStatus()
273 //      g.clearRect( 0, height-15, 200, height );
274 //      g.drawString( strFPS, 2, height - 2 );
275     }
276
277
278
279     /** Trigger delivery of new pixels, waiting for prior pixels to be accepted, if necessary.
280     private void
281     triggerDelivery()
282     {
283         if ( sync )
284         {
```

```
285             long failsafe = 1000;
286             long abandon = System.currentTimeMillis() + failsafe;
287             synchronized ( pending )
288             {
289               // Wait for prior delivery to be accepted before triggering another one.
290               // Interrupted wait()'s return without triggering a delivery.
291               while ( pending[ 0 ] )
292               {
293                 if ( System.currentTimeMillis() >= abandon )
294                   break;
295
296                 try
297                 {  pending.wait( failsafe );  }
298                 catch ( InterruptedException why )
299                 {  Thread.currentThread().interrupt();  return;  }  // reassert interrupt, then r
300               }
301               pending[ 0 ] = true;
302               repaint();
303             }
304         }
305         else
306         {
307             // Identical to original Plasma code.
308             repaint();
309             Thread.yield();
310         }
311     }
312
313     /** Accept delivery of new pixels, allowing calculation of new pixels to proceed. */
314     private void
315     acceptDelivery()
316     {
317         if ( sync )
318         {
319             synchronized ( pending )
320             {
321                 pending[ 0 ] = false;
322                 pending.notifyAll();
323             }
324         }
325     }
326
327
328     public void
329     run()
330     {
331         int index, bottom;
332         int result, tempval;
333         int tpos1, tpos2, tpos3, tpos4;
334         int inc1=6, inc2=3, inc3=3, inc4=9;
335         int pos1=0, pos2=0, pos3=0, pos4=0;
336         int spd1=2, spd2=5, spd3=1, spd4=4;
337
338         while ( ! Thread.currentThread().isInterrupted() )
339         {
340             tpos1 = pos1;  tpos2 = pos2;
341             for( index = pixels.length - 1;  index >= 0;  )
342             {
343                 tpos3 = pos3 - inc3;  tpos4 = pos4 - inc4;
344                 tempval = waveTable[ tpos1 %= size ] + waveTable[ tpos2 %= size ];
345                 for ( bottom = index - w;  index > bottom;  )
346                 {
347                     tpos3 = (tpos3 + inc3) % size;  tpos4 = (tpos4 + inc4) % size;
348                     result = tempval + waveTable[ tpos3 ] + waveTable[ tpos4 ];
349                     // Fill in pixelsRGB[] and pixels[], though only one has its data delivered
350                     pixelsRGB[ index ] = mapRGB[ 0xFF & result ];
351 //                  pixelsRGB[ index ] = (0xFF & result) << 8;
352                     pixels[ index-- ] = (byte) result;
353                 }
354                 tpos1 += inc1;  tpos2 += inc2;
355             }
```

```
356             triggerDelivery();
357             pos1+=spd1; pos2+=spd2; pos3+=spd3; pos4+=spd4;
358         }
359
360     }
361
362 }
363
```

```
Appendix F. Source Code of LinkedList Program


//BugTester.java
//implements two threads that are building the same list
//and are conflicting each other next pointer in the latency between
//fetch and write back

import java.util.*;

public class BugTester
{
    public static void main(String[] args)
    {
        try
        {
            MyListBuilder mlist1;
            MyListBuilder mlist2;

            int lT = 0;         //times to sleep
            int nT = 0;
            if ( args.length >= 1 )
            {
                if ( args.length > 1 && args[1].equals("1") )
                {
                    if ( args.length >= 3 )
                    {
                        lT = Integer.parseInt(args[2]);
                        nT = Integer.parseInt(args[3]);
                    }
                    //no else

                    MyLinkedList mlst = new MyLinkedList(lT,nT,args[0]);

                    mlist1 = new MyListBuilder(mlst,0,5,true,args[0]);
                    mlist2 = new MyListBuilder(mlst,5,10,true,args[0]);
                }
                else        //showing the case in the linked list of java's collection
                {
                    LinkedList lst = new LinkedList();

                    mlist1 = new MyListBuilder(lst,0,5,false,args[0]);
                    mlist2 = new MyListBuilder(lst,5,10,false,args[0]);
                }

                Thread t1 = new Thread(mlist1);
                Thread t2 = new Thread(mlist2);

                t1.start();             //starting the two threads
                t2.start();

                t1.join();              //waiting for all threads to finish
                t2.join();

                mlist1.print();         //prints results to output file

                mlist1.empty();         //empties list
            }
            else
                System.out.println("Name of output file is required as argument!!!");
        }

        catch(InterruptedException e)
        {
            e.getMessage();
            e.printStackTrace();
        }
        catch(Exception e)
        {
            e.getMessage();
            e.printStackTrace();
```

```
        }

    }


}

    //MyLinkedList.java
    //This class implements a linked list class .

    import java.io.*;

    class MyLinkedList
    {
        /*Class Member*/
        public String _fileName = "ID_029646965.txt";
        private MyListNode _header; //The list head pointer
        private int _lTime = 0;     //The time to sleep

        //C'tor
        public MyLinkedList(int lT,int nT,String fName)
        {
            this._fileName = fName;
            this._lTime = lT;
            this._header = new MyListNode( null,nT );
        }

        /*Methods*/

        //Checks if list is empty
        public boolean isEmpty( ){ return this._header._next == null; }

        //Empties list
        public void clear( ){ this._header._next = null; }

        //Returns first element in list
        public MyLinkedListItr first( )
        {
            return new MyLinkedListItr( this._header._next );
        }

        //Inserts element anywhere in list just after current
        public void insert( Object x, MyLinkedListItr p )
        {
            if( p != null && p._current != null )
                p._current._next = new MyListNode( x, p._current._next , p._current._nTime );
        }

        //Inserts element to the end of list .
        //If this func is synchronized the bug will not apear
        public synchronized void addLast( Object x )   //modified by LEON to make the program bu
        {
            MyListNode itr = this._header;

            /*
            //just a sleep noise to the system
            try
            {
                Thread.sleep(this._lTime);
            }
            catch(InterruptedException e)
            {
                e.getMessage();
                e.printStackTrace();
            }
            ////////////////////////////////
            */

            while( itr._next != null )
                itr = itr._next;

```

```
143             insert(x,new MyLinkedListItr(itr));
144         }
145 
146         //Retrieves list size
147         public int size()
148         {
149             MyListNode itr = this._header;
150             int i = 0;
151 
152             while( itr._next != null )
153             {
154                 i++;
155                 itr = itr._next;
156             }
157 
158             return i;
159         }
160 
161         //Finds 'x' element in list
162         public MyLinkedListItr find( Object x )
163         {
164             MyListNode itr = this._header._next;
165 
166             while( itr != null && !itr._element.equals( x ) )
167                 itr = itr._next;
168 
169             return new MyLinkedListItr( itr );
170         }
171 
172         //Finds 'x' previous element in list
173         public MyLinkedListItr findPrevious( Object x )
174         {
175               MyListNode itr = this._header;
176 
177               while( itr._next != null && !itr._next._element.equals( x ) )
178                   itr = itr._next;
179 
180               return new MyLinkedListItr( itr );
181         }
182 
183         //Removes 'x' element from list
184         public void remove( Object x )
185         {
186             MyLinkedListItr p = findPrevious( x );
187 
188             if( p._current._next != null )
189                 p._current._next = p._current._next._next;  // Bypass deleted node
190         }
191 
192         //Prints list
193         public void printList( MyLinkedList theList ) throws IOException
194         {
195             PrintWriter of = new PrintWriter(new FileWriter(".\\" + _fileName,true),true);
196 
197             if( theList.isEmpty( ) )
198                 of.println( "Empty list" );
199             else
200             {
201                 of.print("list : (->");
202 
203                 MyLinkedListItr itr = theList.first( );
204                 for( ; !itr.isPastEnd( ); itr.advance( ) )
205                     of.print( (Integer)itr.retrieve( ) + "->" );
206 
207                 of.print(") , ");
208             }
209 
210             if ( this.size() == 10 )                        //theoretical size of list is 10
211                 of.print("length : " + this.size() +  " , No Bug >");
212             else
213                 of.print("length : " + this.size() +  " , Non-Atomic Bug >");
```

```
214
215             of.close();
216
217
218         }
219
220     }
221     //MyLinkedListItr.java
222     //This class implements iterator to a linked list .
223
224     class MyLinkedListItr
225     {
226         /*Class Memeber*/
227         public MyListNode _current;    // Current position
228
229
230         //C'tor
231         MyLinkedListItr( MyListNode theNode ){ this._current = theNode; }
232
233
234         /*Methods*/
235
236         public boolean isPastEnd( ){ return this._current == null; }
237
238         public Object retrieve( )
239         {
240             return isPastEnd( ) ? null : this._current._element;
241         }
242
243         public void advance( )
244         {
245             if( !isPastEnd( ) )
246                 this._current = this._current._next;
247         }
248
249
250     }
251
252
253 //MyListBuilder.java
254 //This class builds a shared list from given threads .
255
256 import java.util.*;
257 import java.io.*;
258
259 class MyListBuilder implements Runnable
260 {
261     /*Class Members*/
262     public boolean _debug = true;
263     public String _fileName = "ID_029646965.txt";
264     public Object _list = null;
265     public int _bound1 = 0;
266     public int _bound2 = 0;
267
268     //C'tor
269     public MyListBuilder(Object lst,int bnd1,int bnd2,boolean dbg,String fName)
270     {
271         this._debug = dbg;
272
273         if ( _debug == true )
274             this._list = (MyLinkedList)lst;
275         else
276             this._list = (LinkedList)lst;
277
278         this._fileName = fName;
279
280         this._bound1 = bnd1;
281         this._bound2 = bnd2;
282     }
283
284     /*Methods*/
```

```
285
286     //The processor
287     public void run()
288     {
289         for ( int i = this._bound1; i < this._bound2 ;i++ )
290         {
291             /*
292             //just a sleep noise to the system
293             try
294             {
295                 Thread.sleep(i);
296             }
297             catch(InterruptedException e)
298             {
299                 e.getMessage();
300                 e.printStackTrace();
301             }
302             ////////////////////////////////
303             */
304
305             if ( _debug == true )
306                 ((MyLinkedList)_list).addLast(new Integer(i));
307             else
308                 ((LinkedList)_list).addLast(new Integer(i));
309         }
310     }
311
312     //Prints list elements
313     public void print()
314     {
315         int size;
316
317         if ( _debug == true )
318             size = ((MyLinkedList)_list).size();
319         else
320             size = ((LinkedList)_list).size();
321
322         try
323         {
324             PrintWriter of = new PrintWriter(new FileWriter(".\\" + _fileName),true);
325
326             of.print("< " + "BugTester Program" + " , ");
327
328             if ( _debug == true )
329             {
330                 of.close();
331
332                 ((MyLinkedList)this._list).printList((MyLinkedList)_list);
333
334             }
335             else
336             {
337                 of.print("list : (->");
338
339                 Iterator iter = ((LinkedList)_list).iterator();
340
341                 while( iter.hasNext() )
342                     of.print((Integer)iter.next() + "->");
343
344                 of.print(") , ");
345
346                 if ( size == 10 )                       //theoretical size of list is 10
347                     of.print("length : "  + size + " , No Bug >");
348                 else
349                     of.print("length : "  + size + " , Non-Atomic Bug >");
350
351                 of.close();
352
353             }
354
355         }
```

```
356         catch(IOException e)
357         {
358             System.out.println("Problems with output file name : " + _fileName);
359             e.getMessage();
360             e.printStackTrace();
361         }
362 
363     }
364 
365 
366     //Empties list
367     public void empty()
368     {
369         if ( _debug == true )
370             ((MyLinkedList)_list).clear();
371         else
372             ((LinkedList)_list).clear();
373     }
374 
375 }
376 
377     //MyListNode.java
378     //This class implements basic node stored in a linked list .
379 
380     class MyListNode
381     {
382         /*Class Members*/
383         public Object _element;         //Node's data
384         public MyListNode _next;        //Pointer to next node
385         public int _nTime = 0;          //The  time to sleep
386 
387         //C'tor - 1
388         MyListNode( Object theElement,int nT ){ this( theElement, null , nT ); }
389 
390         //C'tor - 2
391         MyListNode( Object theElement, MyListNode n , int nT )
392         {
393             this._nTime = nT;
394 
395             synchronized ( this )
396             {
397                 this._element = theElement;
398                 this._next = n;
399             }
400 
401             /*
402             //a sleep before the last element can be added to list .
403             //it conflicts with the while looop in addLast func in MyLinkedList.java file .
404             //the if condition is in order to show the case that no noise is added
405             //to this c'tor in which case it is hard to acheive the bug
406             if ( this._nTime > 0 )
407             {
408                 try
409                 {
410                     Thread.sleep(this._nTime);
411                 }
412                 catch(InterruptedException e)
413                 {
414                     e.getMessage();
415                     e.printStackTrace();
416                 }
417             }
418             //no else
419             ////////////////////////////////////////////
420             */
421         }
422 
423     }
424 
```