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 call, 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 1.

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
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 $N^2$ 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]

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.

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
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>
<thead>
<tr>
<th>Program Name</th>
<th>LOC</th>
<th>classes</th>
<th>sync methods</th>
<th>sync blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Webserver</td>
<td>125</td>
<td>6</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>Chat</td>
<td>482</td>
<td>4</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Miasma</td>
<td>360</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>LinkedList</td>
<td>421</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>1,388</td>
<td>16</td>
<td>22</td>
<td>7</td>
</tr>
</tbody>
</table>

Table IV  
EXEMPLARY PROGRAMS USED IN OUR EXPERIMENTS

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


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

<table>
<thead>
<tr>
<th>Mutation Operators</th>
<th>Webserver</th>
<th>Chat</th>
<th>Miasma</th>
<th>LinkedList</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXT</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MSP</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>ESP</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MSF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MXC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MBR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RTXC</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>RCXC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RNA</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RJS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ELPA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EAN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ASTK</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>RSTK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RSK</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>RSB</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>RVK</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>RFU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RXO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EEOLO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHCR</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>SKCR</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>EXCR</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>SPCR</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DelStat</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>ReplArg</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>DelSyncCall</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>ReplMeth</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>InsNegArg</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ReplTargObj</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Mutation Operators</td>
<td>Webserver</td>
<td>Chat</td>
<td>Miasma</td>
<td>LinkedList</td>
<td>Total</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>------</td>
<td>--------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>RKSN</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>RCSN</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>RSSN</td>
<td>21</td>
<td>35</td>
<td>0</td>
<td>3</td>
<td>59</td>
</tr>
<tr>
<td>AKST</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>MASN</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>RSNB</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>RSSB</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>MOSB</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>EOSB</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MVSB</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>RMWN</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>RMSV</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>RKSN+RSSN</td>
<td>21</td>
<td>35</td>
<td>0</td>
<td>3</td>
<td>59</td>
</tr>
<tr>
<td>AKST+MASN</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>RKSN+MASN</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>RSNB+RSSB</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>MOSB+RSSB</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>MOSB+MVSB</td>
<td>2</td>
<td>4</td>
<td>11</td>
<td>2</td>
<td>19</td>
</tr>
</tbody>
</table>
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
            return null;
    }
    public synchronized void add(Object o) {
        first = new Pair(o, first);
    }
}

class WriterThread extends Thread {
    public void run() {

int myMaxNumber = 100;
while (myMaxNumber < 10000) {
    for (int i = 0; i < 100; ++i) {
        Webserver.dataTable.put(
            new Integer(myMaxNumber),
            String.valueOf(myMaxNumber));
        myMaxNumber++;
    }
    synchronized(Webserver.maxNumberLock) {
        Webserver.maxNumber = myMaxNumber;
    }
}  
System.out.println("Writer complete");
}

class ReaderThread extends Thread {
    public void run() {
        int myMaxNumber;
        Random rand = new Random();
        for (int i = 0; i < 1000; ++i) {
            synchronized(Webserver.maxNumberLock) {
                myMaxNumber = Webserver.maxNumber;
            }
            for (int j = 0; j < 100; ++j) {
                int index = Math.abs(
                    rand.nextInt()) % myMaxNumber;
                Webserver.dataTable.get(
                    new Integer(index));
            }
        }
        System.out.println("Reader complete");
    }
}

public class Webserver {
    public static void main(String args[]) {
        /* set up data table */
        maxNumber = 100;
        dataTable = new Table();
        maxNumberLock = new Object();
        for (maxNumber = 0; maxNumber < 100;
            ++maxNumber) {
            dataTable.put(new Integer(maxNumber),
                String.valueOf(maxNumber));
        }
        for (int threadNum = 0; threadNum < 8;
            ++threadNum) {
            new ReaderThread().start();
        }
        new WriterThread().start();
    }
    public static Table dataTable;
    public static int maxNumber;
    public static Object maxNumberLock;
}
Appendix D. Source Code of Chat Program

```java
import javax.swing.JFrame;

/**<n* 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 action
        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 tl
        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
    /**
```
} // end ChatPanel
* Adds a string to the transcript area, followed by a carriage return.
* @param s the string to be added to the transcript.
*/

synchronized private void postMessage(String s) {
    transcript.append(s + "\n");
    // The following line is a nasty kludge that was the only way I could find to force
    // the transcript to scroll so that the text that was just added is visible in
    // the window. Without this, text can be added below the bottom of the visible area
    // of the transcript.
    transcript.setCaretPosition(transcript.getDocument().getLength());
}

/**
 * This method is called when an action event occurs in a component, assuming that
 * this ChatPanel has been "registered" as an ActionListener with that component.
 */
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource(); // This is the object that generated the event.
    if (source == sendButton || source == inputBox) {
        String str = inputBox.getText();
        postMessage("SEND: " + str);
        inputBox.selectAll();
        inputBox.requestFocus();
    }
}

/**
 * This interface should be implemented by an object that wants to use
 * a SimpleNet object for network connectivity. A SimpleNet object must
 * have an "observer" that implements this interface. The SimpleNet
 * object informs its observer about network events by calling methods
 * defined in this interface.
 */
public interface SimpleNetObserver {

    /**
     * This method is called when the connection has been successfully opened and is ready to
     * be used for sending messages back and forth.
     * @param connection the SimpleNet object that is managing the connection. You can ignore
     * this parameter unless you are using several SimpleNet objects and need to tell them apar
     */
    public void connectionOpened(SimpleNet connection);

    /**
     * This method is called whenever a message is received from the other side of the
     * network connection.
     * @param connection the SimpleNet object that is managing the connection. You can ignore
     * this parameter unless you are using several SimpleNet objects and need to tell them apar
     * @param data the message that was received
     */
    public void connectionDataReceived(SimpleNet connection, String data);

    /**
     * This method is called when the connection closes because you called the close() method
     * in the SimpleNet object.
     * @param connection the SimpleNet object that is managing the connection. You can ignore
     * this parameter unless you are using several SimpleNet objects and need to tell them apar
     */
    public void connectionClosed(SimpleNet connection);

    /**
     * This method is called when the connection closes because of action taken on the other
public void connectionClosedByPeer(SimpleNet connection);

/**
 * This method is called when the connection closes because of some sort of network error.
 * Note that this method can be called when the connection is in the process of being open.
 * @param connection the SimpleNet object that is managing the connection. You can ignore
 * @param this parameter unless you are using several SimpleNet objects and need to tell them apar
 */
public void connectionClosedWithError(SimpleNet connection, String errorMessage);

import java.io.*;
import java.net.*;

/**
 * This class supports basic, text-based communication between two computers
 * on the network. For opening a connection, a SimpleNet object can run
 * in either server mode -- where it waits for an incoming connection -- or in
 * client mode -- where it tries to make a connection to a waiting server. Use
 * the listen() method to run as a server; use connect() to run as a client.
 * Once the connection has been opened, it makes no difference whether server
 * or client mode was used. Lines of text can be transmitted in either
 * direction. Use the send() method to transmit a line of text. Receiving
 * messages is more complicated, since they can arrive asynchronously. Also,
 * a connection can be closed asynchronously from the other side. And when
 * operating in server mode, the connection is opened asynchronously. To support
 * asynchronous operation, a SimpleNet object must have an "observer" that
 * implements the SimpleNetObserver interface. This interface defines several
 * methods that are called by the SimpleNet object when network events occur.
 * See that interface for more information.
 */
public class SimpleNet {

    /**
     * Possible state that a SimpleNet object can be in. The states are mostly used
     * internally in this class, but you can find out the state by calling getState().
     */
    public final static int IDLE = 1;
    public final static int WAITING_FOR_CONNECTION = 2;
    public final static int CONNECTING = 3;
    public final static int CONNECTED = 4;
    public final static int CLOSING = 5;

    private SimpleNetObserver owner;
    private int state;
    private volatile PrintWriter out;
    private ConnectionHandler connectionHandler; // ConnectionHandler is a nested class, defined

    /**
     * Create a SimpleNet object that can be used for basic two-way text-based network connecti
     * @param observer A non-null object that implements the SimpleNetObserver interface. When
     * @param certain network events occur, the SimpleNet object will notify the observer by calling
     * one of the methods defined in the interface.
     * @throws IllegalArgumentException if observer is null
     */
    public SimpleNet(SimpleNetObserver observer) {
        if (observer == null)
            throw new IllegalArgumentException("A SimpleNet object requires a non-null SimpleNe
        owner = observer;
        state = IDLE;
    }

    /**
     * Open a connection in server mode. The SimpleNet object will wait for an incoming connec
     */
synchronized public void listen(int portNumber) {
    if (state != IDLE)
        throw new IllegalStateException("Attempt to open a connection while not in idle state");
    state = WAITING_FOR_CONNECTION;
    connectionHandler = new ConnectionHandler(portNumber);
    connectionHandler.start();
}

/**
 * Open a connection in client mode. The SimpleNet object will attempt to connect to a server
 * that is listening on a specified computer and at a specified port number. This method runs
 * immediately, without waiting for the connection to open. The observer will be notified
 * when the connection opens by calling its connectionOpened() method. (Or, if an error occurs,
 * its connectionClosedWithError() method will be called instead.)
 * @param hostNameOrIP the host name (such as "math.hws.edu") or IP address (such as "172."
 * @param portNumber the port number where the server is listening
 * @throws IllegalStateException if this SimpleNet object is already connected or opening a
 */
synchronized public void connect(String hostNameOrIP, int portNumber) {
    if (state != IDLE)
        throw new IllegalStateException("Attempt to open a connection while not in idle state");
    state = CONNECTING;
    connectionHandler = new ConnectionHandler(hostNameOrIP, portNumber);
    connectionHandler.start();
}

/**
 * Closes the current connection, if any. This method returns immediately.
 * The observer will be notified when the connection closes by calling its
 * connectionClosed() method. Note that this method can be called while
 * the connection is still in the process of being opened. In that case,
 * the connection attempt will be aborted; the connectionClosed()
 * method in the observer will still be called in this case.
 */
synchronized public void close() {
    if (state == IDLE)
        return;  // ignore close command if there is no connection
    state = CLOSING;
    connectionHandler.abort();
}

/**
 * Transmit a message to the other side of the connection. Attempts to
 * transmit data when no connection is opened are simply ignored.
 * @param message the text to be transmitted
 */
synchronized public void send(String message) {
    if (out == null || out.checkError() || state != CONNECTED)
        return;
    out.println(message);
    out.flush();
    if (out.checkError())
        close();
}

/**
 * Returns the current state of this SimpleNet object. For the most part,
 * the state information is not needed outside this class, but you can use
 * this method to inquire the current state if you need it.
 * @return the current state, which is one of the constants
 * IDLE, CONNECTING, WAITING_FOR_CONNECTION, CONNECTED, or CLOSING,
 */
/*
 * synchronized public int getState() {
 *     return state;
 * }
 *
 // The remainder of this file is the private implementation part of the class.
 //----------------------------------------------------------------------------
 synchronized private void dataReceived(ConnectionHandler handler, String data) {
     if (state == IDLE || state == CLOSING || handler != connectionHandler)
         return; // ignore possible input from old connection
     owner.connectionDataReceived(this, data);
 }

 synchronized void opened(ConnectionHandler handler) {
     if (handler != connectionHandler || (state != WAITING_FOR_CONNECTION && state != CONNECTION))
         return;
     out = connectionHandler.getOutputStream();
     state = CONNECTED;
     owner.connectionOpened(this);
 }

 synchronized void closed(ConnectionHandler handler) {
     if (state == IDLE || handler != connectionHandler)
         return;
     if (state == CLOSING)
         owner.connectionClosed(this);
     else
         owner.connectionClosedByPeer(this);
     connectionHandler = null;
     out = null;
     state = IDLE;
 }

 synchronized private void error(ConnectionHandler handler, String message, Exception e) {
     if (state == IDLE || connectionHandler != handler)
         return; // Ignore any left-over error from old connections.
     out = null;
     connectionHandler = null;
     if (state == CLOSING) { // don't send error since owner wants to close anyway
         state = IDLE;
         owner.connectionClosed(this);
     } else {
         state = IDLE;
         owner.connectionClosedWithError(this, message + "": " + e.toString());
     } // e.printStackTrace();
 }

 private class ConnectionHandler extends Thread {

     private int port;
     private String host;
     private boolean runAsServer;

     private volatile Socket connection;
     private volatile ServerSocket listener;
     private volatile boolean aborted;
     private volatile Thread connectionOpener;
     private volatile Exception exceptionWhileConnecting;
     private volatile PrintWriter out;
     private BufferedReader in;

     ConnectionHandler(int portNumber) {
         port = portNumber;
         runAsServer = true;
     }

 }
```java
ConnectionHandler(String hostName, int portNumber) {
    host = hostName;
    port = portNumber;
    runAsServer = false;
}

void abort() {
    try {
        if (listener != null)
            listener.close();
        else if (connectionOpener != null)
            this.interrupt();
        else if (connection != null) {
            connection.shutdownInput();
            connection.shutdownOutput();
            connection.close();
        }
    } catch (Exception e) {
    }
    aborted = true;
}

PrintWriter getOutputStream() {
    return out;
}

public void run() {
    BufferedReader in;
    if (!runAsServer) {
        try {
            connectionOpener = new Thread() {
                public void run() {
                    try {
                        try {
                            connection = new Socket(host, port);
                        }
                        catch (Exception e) {
                            exceptionWhileConnecting = e;
                        }
                        finally {
                            synchronized(this) {
                                notify();
                            }
                        }
                    }
                    catch (Exception e) {
                        connection = null;
                    }
                }
            }
            connectionOpener.start();
            synchronized(connectionOpener) {
                try {
                    connectionOpener.wait();
                }
                catch (InterruptedException e) {
                    closed(this);
                    return;
                }
            }
            connectionOpener = null;
            if (exceptionWhileConnecting != null)
                throw exceptionWhileConnecting;
        }
        catch (Exception e) {
            error(this, "Error while attempting to connect to " + host, e);
        }
        return;
    }
    } catch (Exception e) {
```
else {
    try {
        listener = new ServerSocket(port);
        connection = listener.accept();
        listener.close();
        listener = null;
    } catch (Exception e) {
        error(this,"Error while waiting for connection request", e);
        return;
    }
}

if (aborted)
    return;

try {
    in = new BufferedReader(new InputStreamReader( connection.getInputStream() ));
    out = new PrintWriter(connection.getOutputStream());
} catch (Exception e) {
    error(this,"Error while creating network input/ouput streams", e);
    return;
}

if (aborted)
    return;

opened(this);

try {
    while (true) {
        String input = in.readLine();
        if (input == null || aborted)
            break;
        dataReceived(this,input);
    }
} catch (Exception e) {
    error(this,"An error occured while connected", e);
}

finally {
    closed(this);
    if (connection != null) {
        try {
            connection.close(); // Make sure connection is properly closed.
        } catch (Exception e) {
        }
    }
} // end run()

} // end nested class ConnectionHandler

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

    public Miasma()
    {
        pending = new boolean[ 1 ];
        strFPS = "";
    }
public void init()
{
    scale = getInt( "scale", 2 );
    showFPS = getBoolean( "showFPS", true );
    pri = getInt( "pri", Thread.MIN_PRIORITY );
    draw = getBoolean( "draw", true );
    deliver = getInt( "deliver", -1 );
    filter = getBoolean( "filter", false );
    userRGB = getBoolean( "rgb", false );
    sync = getBoolean( "sync", false );
    int avg = getInt( "avg", 10 );
    framesPast = new int[ avg ];
    elapsedPast = new int[ avg ];
    width = size().width;
    height = size().height;
    w = width/scale;
    h = height/scale;
    pixels = new byte[ w * h ];
    pixelsRGB = new int[ pixels.length ];
    size = ((w+h)/2) * 4;
    waveTable = new int[size];
    calculateWaveTable();
    calculatePaletteTable();
}

private boolean
getBoolean( String name, boolean defaultValue )
{
    String val = getParameter( name );
    if ( val == null )
        return ( defaultValue );
    else
        return ( "true".equals( val ) );
}

private int
getInt( String name, int defaultValue )
{
    String val = getParameter( name );
    if ( val == null )
        return ( defaultValue );
    else
        return ( Integer.parseInt( val ) );
}

private void
calculateWaveTable()
{
    double perStep = (2 * Math.PI) / size;
    for ( int i = 0; i < size; ++i )
    {  waveTable[ i ] = (int) (32 * (1 + Math.sin( i * perStep )));  }
}

private void
calculatePaletteTable()
{
    mapRGB = new int[ 256 ];
    // All G components are 0 in palette, so do nothing to fill the 'gg' array.
    int r, b;
    byte[] rr = new byte[ 256 ];
    byte[] gg = new byte[ 256 ];
    byte[] bb = new byte[ 256 ];
To ensure that the RGB image and the indexed image look different,
the RGB one shows red/green gradients vs. the indexed one's red/blue gradients.
Do this simply by using 'b' as a G component, not a B, in the 24-bit mapRGB values.

```java
for ( int i = 0;  i < 128;  i++ )
{
    rr[ i ] = rr[ 255 - i ] = (byte) (r = i + i + 1);
    bb[ i ] = bb[ 255 - i ] = (byte) (b = 0xFF & -r);
    mapRGB[ i ] = mapRGB[ 255 - i ] = (r << 16) | (b << 8);
}

icm = new IndexColorModel( 8, 256, rr, gg, bb );
```

```java
public void start()
{
    // System.out.println( "codebase = " + getCodeBase() );
    // System.out.println( " docbase = " + getDocumentBase() );
    // Defer creation of Images and MemoryImageSources until the last possible moment.
    // Use source's state as representative of all image-related variables.
    if ( source == null )
    {
        if ( useRGB )
        {
            // could use this.getColorModel() or ColorModel.getRGBdefault() or Toolkit.getColorModel()
            ColorModel modelRGB = getColorModel();
            // System.out.println( "ColorModel: " + modelRGB );
            source = new MemoryImageSource( w, h, modelRGB, pixelsRGB, 0, w );
        }
        else
        {
            // source is indexed image, with bytes for pixels.
            source = new MemoryImageSource( w, h, icm, pixels, 0, w );
        }
        source.setAnimated( true );
        source.setFullBufferUpdates( true );
    }
    ImageProducer producer = source;
    if ( filter )
    {
        // The filter rescales to original size, so drawImage() won't.
        // You could use a different class of rescaling filter, to measure its effect on
        producer = new FilteredImageSource( producer, new ReplicateScaleFilter( width, )
    }
    // img = Toolkit.getDefaultToolkit().createImage( producer );
    img = createImage( producer );
}

public void stop()
{
    if  ( runThread != null )
    {
        runThread.interrupt();
        runThread = null;
    }
}
```
public void update(Graphics g) {
    // Deliver source's pixels to Image, if needed.
    // An int started at -1 will reach 0 after 49.7 days at 1000 fps,
    // or 497 days at 100 fps, etc.
    if (deliver != 0) {
        source.newPixels();
        --deliver;
    }

    // Signal acceptance after new pixels delivered, but before drawing occurs.
    // This maximizes concurrency between producer and consumer threads,
    // while ensuring that every calculated frame is accepted and handled.
    acceptDelivery();

    if (draw) {
        g.drawImage(img, 0, 0, width, height, null);
    }

    ++frames;
    if (showFPS) {
        calculateFPS(System.currentTimeMillis(), g);
    }
}

private void calculateFPS(long now, Graphics g) {
    if (now > first + 1000L) {
        fps = frames;
        frames = 0;

        // Elapsed millis should never overflow the capacity of an int (~2Msecs).
        int elapsed = (int) (now - first);
        first = now;

        int n = framesIndex;
        framesSum = framesSum - framesPast[n] + fps;
        elapsedSum = elapsedSum - elapsedPast[n] + elapsed;
        framesPast[n] = fps;
        elapsedPast[n] = elapsed;
        framesIndex = (n + 1) % framesPast.length;

        // Calculate scaled up by 10, to get an extra decimal digit of precision to display
        // This is reasonable considering the precision of past readings, and the averaging
        n = (framesSum * 10000) / elapsedSum;

        if (past < framesPast.length) {
            ++past;

            strFPS = fps + " fps, avg " + (n/10) + "." + (n%10) + " over " + past + " sec";
            showStatus(strFPS);
        }
    }

    // ## This display now taken over by showStatus()
    // g.clearRect(0, height-15, 200, height);
    // g.drawString(strFPS, 2, height - 2);
}

/** Trigger delivery of new pixels, waiting for prior pixels to be accepted, if necessary. */
private void triggerDelivery() {
    if (sync) {
    }
long failsafe = 1000;
long abandon = System.currentTimeMillis() + failsafe;
synchronized ( pending )
{
    // Wait for prior delivery to be accepted before triggering another one.
    // Interrupted wait()'s return without triggering a delivery.
    while ( pending[ 0 ] )
    {
        if ( System.currentTimeMillis() >= abandon )
            break;
        try
        {
            pending.wait( failsafe );
        }
        catch ( InterruptedException why )
        {
            Thread.currentThread().interrupt();
            return;
        }  // reassert interrupt, then re-
        pending[ 0 ] = true;
        repaint();
    }
    else
    {
        // Identical to original Plasma code.
        repaint();
        Thread.yield();
    }
}

/** Accept delivery of new pixels, allowing calculation of new pixels to proceed. */
private void acceptDelivery()
{
    if ( sync )
    {
        synchronized ( pending )
        {
            pending[ 0 ] = false;
            pending.notifyAll();
        }
    }
}

public void run()
{
    int index, bottom;
    int result, tempval;
    int tpos1, tpos2, tpos3, tpos4;
    int inc1=6, inc2=3, inc3=3, inc4=9;
    int pos1=0, pos2=0, pos3=0, pos4=0;
    int spd1=2, spd2=5, spd3=1, spd4=4;

    while ( ! Thread.currentThread().isInterrupted() )
    {
        tpos1 = pos1;  tpos2 = pos2;
        for( index = pixels.length - 1;  index >= 0;  )
        {
            tpos3 = pos3 - inc3;  tpos4 = pos4 - inc4;
            tempval = waveTable[ tpos1 %= size ] + waveTable[ tpos2 %= size ];
            for ( bottom = index - w;  index > bottom;  )
            {
                tpos3 = (tpos3 + inc3) % size;  tpos4 = (tpos4 + inc4) % size;
                result = tempval + waveTable[ tpos3 ] + waveTable[ tpos4 ];
                // Fill in pixelsRGB[] and pixels[], though only one has its data delivered
                pixelsRGB[ index ] = mapRGB( 0xFF & result );
                // pixelsRGB[ index ] = (0xFF & result) << 8;
                pixels[ index-- ] = (byte) result;
            }
            tpos1 += inc1;  tpos2 += inc2;
        }
    }
}
triggerDelivery();
pos1+=spd1; pos2+=spd2; pos3+=spd3; pos4+=spd4;
}
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]);
            }
            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]);
            }
            //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
        }
    }  //catch

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

    catch(Exception e)
    {
        e.getMessage();
e.printStackTrace();
    }
import java.io.*;

class MyLinkedList
{
    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 b
    {
        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;
        itr = new MyListNode( x, null, _lTime );
//Retrieves list size
public int size()
{
    MyListNode itr = this._header;
    int i = 0;
    while( itr._next != null )
    {
        i++;
        itr = itr._next;
    }
    return i;
}

//Finds 'x' element in list
public MyLinkedListItr find( Object x )
{
    MyListNode itr = this._header._next;
    while( itr != null && !itr._element.equals( x ) )
        itr = itr._next;
    return new MyLinkedListItr( itr );
}

//Finds 'x' previous element in list
public MyLinkedListItr findPrevious( Object x )
{
    MyListNode itr = this._header;
    while( itr._next != null && !itr._next._element.equals( x ) )
        itr = itr._next;
    return new MyLinkedListItr( itr );
}

//Removes 'x' element from list
public void remove( Object x )
{
    MyLinkedListItr p = findPrevious( x );
    if( p._current._next != null )
        p._current._next = p._current._next._next;  // Bypass deleted node
}

//Prints list
public void printList( MyLinkedList theList ) throws IOException
{
    PrintWriter of = new PrintWriter( new FileWriter("\" + _fileName,true),true );
    if( theList.isEmpty( ) )
        of.println( "Empty list" );
    else
    {
        of.print("list : (->");
        MyLinkedListItr itr = theList.first( );
        for( ; !itr.isPastEnd( ); itr.advance() )
            of.print( (Integer)itr.retrieve( ) + "->" );
        of.print(")",");
    }
    if ( this.size() == 10 )  //theoretical size of list is 10
        of.print("length : " + this.size() + " , No Bug >");
    else
        of.print("length : " + this.size() + " , Non-Atomic Bug >");
}
of.close();
}

//MyLinkedListItr.java
//This class implements iterator to a linked list.

class MyLinkedListItr {
    /*Class Member*/
    public MyListNode _current;    // Current position

    //C'tor
    MyLinkedListItr( MyListNode theNode ) { this._current = theNode; }

    /*Methods*/
    public boolean isPastEnd() { return this._current == null; }
    public Object retrieve() {
        return isPastEnd() ? null : this._current._element;
    }
    public void advance() {
        if( !isPastEnd() )
            this._current = this._current._next;
    }

//MyListBuilder.java
//This class builds a shared list from given threads.
import java.util.*;
import java.io.*;

class MyListBuilder implements Runnable {
    /*Class Members*/
    public boolean _debug = true;
    public String _fileName = "ID_029646965.txt";
    public Object _list = null;
    public int _bound1 = 0;
    public int _bound2 = 0;

    //C'tor
    public MyListBuilder(Object lst,int bnd1,int bnd2,boolean dbg,String fName) {
        this._debug = dbg;
        if ( _debug == true )
            this._list = (MyLinkedList)lst;
        else
            this._list = (LinkedList)lst;
        this._fileName = fName;
        this._bound1 = bnd1;
        this._bound2 = bnd2;
    }
    /*Methods*/
//The processor
public void run()
{
    for ( int i = this._bound1; i < this._bound2 ;i++ )
    {
        /*
        //just a sleep noise to the system
        try
        {
            Thread.sleep(i);
        }
        catch(InterruptedException e)
        {
            e.getMessage();
            e.printStackTrace();
        }
        ///////////////////////////////////
        */
        if ( _debug == true )
        {
            ((MyLinkedList)_list).addLast(new Integer(i));
        }
        else
        {
            ((LinkedList)_list).addLast(new Integer(i));
        }
    }
}

//Prints list elements
public void print()
{
    int size;
    if ( _debug == true )
        size = ((MyLinkedList)_list).size();
    else
        size = ((LinkedList)_list).size();
    try
    {
        PrintWriter of = new PrintWriter(new FileWriter("\" + _fileName),true);
        of.print("< " + "BugTester Program" + " , ");
        if ( _debug == true )
        {
            of.close();
            ((MyLinkedList)this._list).printList((MyLinkedList)_list);
        }
        else
        {
            of.print("list : (->");
            Iterator iter = ((LinkedList)_list).iterator();
            while( iter.hasNext() )
                of.print((Integer)iter.next() + "->");
            of.print("\", ");
            if ( size == 10 ) //theoretical size of list is 10
                of.print("length : " + size + " , No Bug >");
            else
                of.print("length : " + size + " , Non-Atomic Bug >");
            of.close();
        }
    }
}
catch (IOException e) {
    System.out.println("Problems with output file name : "+_fileName);
    e.getMessage();
    e.printStackTrace();
}

//Empties list
public void empty() {
    if (_debug == true) {
        ((MyLinkedList)_list).clear();
    } else {
        ((LinkedList)_list).clear();
    }
}

//MyListNode.java
//This class implements basic node stored in a linked list.

class MyListNode {
    /*Class Members*/
    public Object _element; //Node's data
    public MyListNode _next; //Pointer to next node
    public int _nTime = 0; //The time to sleep

    //C'tor - 1
    MyListNode( Object theElement, int nT ) { this( theElement, null, nT ); }
    //C'tor - 2
    MyListNode( Object theElement, MyListNode n, int nT ) {
        this._nTime = nT;
        synchronized ( this ) {
            this._element = theElement;
            this._next = n;
        }
    }

    /*
    //a sleep before the last element can be added to list.
    //it conflicts with the while loop in addLast func in MyLinkedList.java file.
    //the if condition is in order to show the case that no noise is added
    //to this c'tor in which case it is hard to acheive the bug
    if ( this._nTime > 0 )
    {
        try {
            Thread.sleep(this._nTime);
        } catch (InterruptedException e) {
            e.getMessage();
            e.printStackTrace();
        }
    } //no else
    //**************************
    */
}