Visualization of Thread Interleavings Produced by Java PathFinder

UNDERGRADUATE HONOURS THESIS
FACULTY OF SCIENCE (COMPUTING SCIENCE)
UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

Gowritharan Maheswara

SUPERVISORS:
Dr. Jeremy S. Bradbury
Dr. Christopher Collins

April 23, 2010

Abstract

As we move into the era of multi-core processors, a lot more software developers are writing concurrent programs to best utilize these processors. In a concurrent program many streams of operations are executed in parallel versus a sequential program where only one stream of operations, called threads, is executed. The advantage of concurrent programs is that they are a lot more efficient then sequential programs. The disadvantage of concurrent programs is that they are really difficult to test. One solution to testing this type of concurrent software is to use tools that explore the thread interleaving space like Java PathFinder. Java PathFinder is tool developed by NASA that provides a solution to concurrency testing for Java based programs. This tool uses model checking to search all possible schedules for a given concurrent program and can produce a thread interleaving if a bug is found. One of the problems of Java PathFinder is that it does not include a tool to graphically display schedules and show where defects appear in the source code. To help enhance concurrency testing of Java PathFinder, this thesis has researched and developed a Graphical User Interface tool that is able to solve this problem.

Contents

List of Figures

1. Introduction:

The motivation behind this thesis is to learn and understand the different concepts of concurrent programming since it is one of the more significant topics in computer science today. Another motivation for this thesis is to get a chance to work with the different tools that attempt to solve the test coverage problem that writing a concurrent program creates. Not only did I get a chance to work with these tools but I also developed a Graphical Visualization tool that enhanced programmers concurrency testing and analysis. I have taken many steps to develop my tool and my thesis which I have outlined below.

The first step taken to complete this thesis was to do in-depth research on Microsoft CHESS. This research has given me an understanding of how CHESS handles the problem of concurrency testing and how it uses the Concurrency Explorer Tool to help solve this problem. I have also analyzed the Concurrency Explorer tool to get ideas on how to setup the Graphical User Interface for my own tool and/or how to enhance their Graphical User Interface to use for my own tool. Then I created a set of functions that needs to be implemented for my tool. This was also done by researching about Microsoft CHESS, extracting key ideas, and using my own ideas along with their key ideas. After this was done, I did an in-depth research on Java PathFinder which will gave me an idea of how this tool works and how it schedules the different interleaving of the threads during execution. When this was determined, I started to implement ways to extract specific schedules that caused errors in the program. I have used this information as input into my tool and the basis for developing my Graphical User Interface tool.

This paper is comprised of several sections which outline the ways in which the current research and development of this thesis was completed. Section 2 allows for an understanding of model checking, Microsoft CHESS, and Java Pathfinder through the use of background information. In Section 3 information on the various stages of the tool development will be provided. In Section 4 will outline the overall findings of this study. Lastly, Section 5 will outline the materials utilized to directly support the information gathered as well as the findings of the current study.

2. Background and Related Works:

In this section, I will briefly explain how model checking systems function. Specifically, I will provide information on how this system works, the different problems this system can encounter, and how Java PathFinder utilizes this system. After I have explained how model checking systems function, I will then provide information on two tools that can test concurrency programs called Microsoft CHESS and Java PathFinder. These are similar tools that make use model checkers. The major difference between Microsoft CHESS and Java PathFinder is that CHESS is used against programs written in Microsoft's .NET environment and Java PathFinder is used against programs written in Java.

2.1. Model Checking Systems

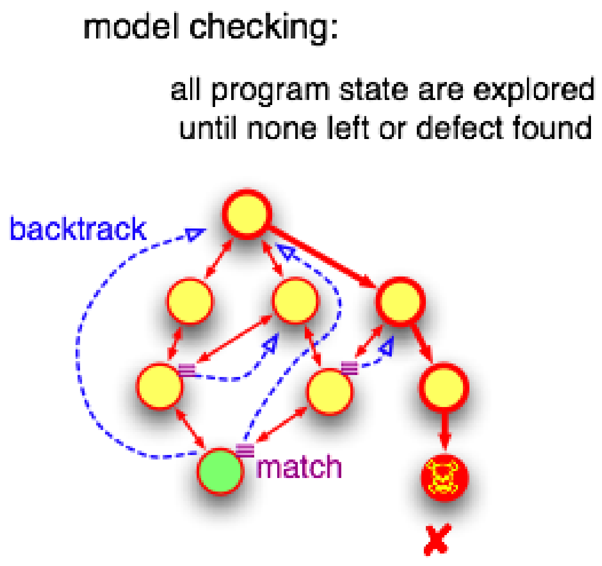

model checking:
all program state are explored
until none left or defect found
backtrack
match

Figure 1 - Model Checking Diagram [2]

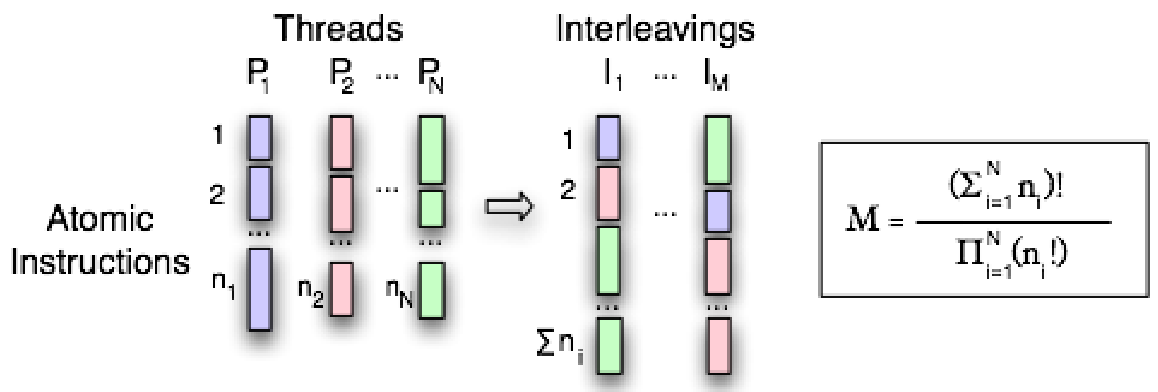
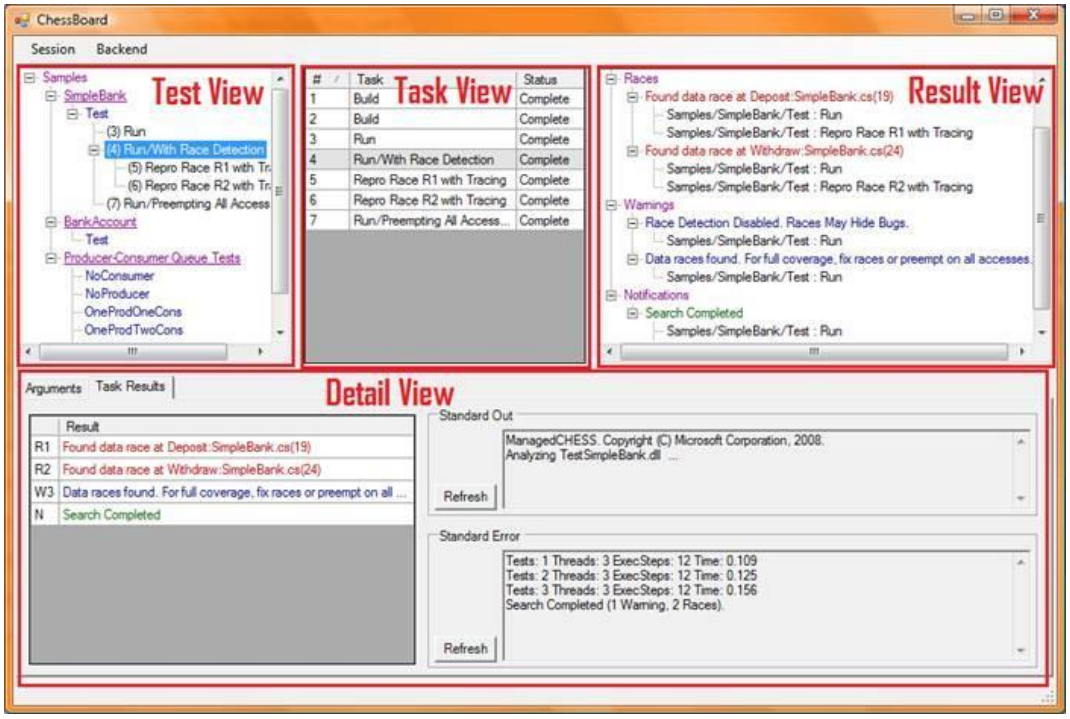


Figure 2 - Thread Scheduling in Concurrent Java Programs [2]

I will not go into great detail about how a model checking system works but I will explain its basic function. Given a model of a system, in this case our program, a model checker will test if this model meets a specific requirement. The specific requirement in this case would be a concurrency related bug or the absence of these bugs. The model checker uses a Finite State Machine to enumerate all possible states of the system and the different paths it can take. This graphical representation can be seen in Figure 1. In this figure, the states are represented by light orange circles (nodes) and the transitions are represented by the orange arrows (edges). The very far right nodes and transitions that are highlighted in dark orange represent a scheduled path with a defect in it. Software model checking has a scalability problem, as the states increase in number, the amount of interleavings that must scheduled grows extremely large. This is called the State Explosion problem. A mathematical diagram of this scheduling problem is shown in Figure 2. On the left hand side, there are a number of different threads in the programs which represent the states. Around the middle of the diagram, there are the interleavings or schedules that can be enumerated using these threads. On the right hand side, there is the mathematical model of the state space growth which is the maximum number of interleavings that need to be scheduled given the number of threads and the number of program instructions. There are many techniques that can be used to solve the State Explosion problem. One of the techniques used to solve this State Explosion problem is to use a Depth-First State Generation model checker. Simply, the model checker determines if the requirement can be reached within a set amount of steps or transitions.

2.2. Microsoft CHESS



- Samples
  - SimpleBank
    - Test
      - (3) Run
      - (4) Run/With Race Detection
        - (5) Repro Race R1 with Tr
        - (6) Repro Race R2 with Tr
      - (7) Run/Preempting All Access
  - BankAccount
    - Test
  - Producer-Consumer Queue Tests
    - NoConsumer
    - NoProducer
    - OneProdOneCons
    - OneProdTwoCons

#	Task	Status
1	Build	Complete
2	Build	Complete
3	Run	Complete
4	Run/With Race Detection	Complete
5	Repro Race R1 with Tracing	Complete
6	Repro Race R2 with Tracing	Complete
7	Run/Preempting All Access...	Complete

- Races
  - Found data race at Deposit:SimpleBank.cs(19)
    - Samples/SimpleBank/Test : Run
    - Samples/SimpleBank/Test : Repro Race R1 with Tracing
  - Found data race at Withdraw:SimpleBank.cs(24)
    - Samples/SimpleBank/Test : Run
    - Samples/SimpleBank/Test : Repro Race R2 with Tracing
- Warnings
  - Race Detection Disabled. Races May Hide Bugs.
    - Samples/SimpleBank/Test : Run
  - Data races found. For full coverage, fix races or preempt on all accesses.
    - Samples/SimpleBank/Test : Run
- Notifications
  - Search Completed
    - Samples/SimpleBank/Test : Run

	Result
R1	Found data race at Deposit:SimpleBank.cs(19)
R2	Found data race at Withdraw:SimpleBank.cs(24)
W3	Data races found. For full coverage, fix races or preempt on all ...
N	Search Completed

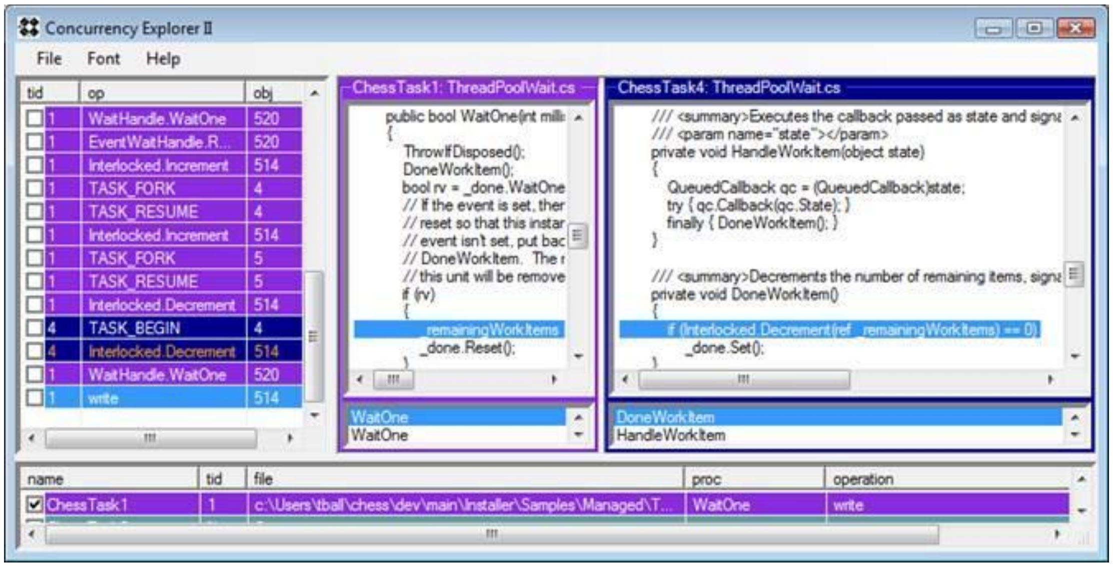
Standard Out

```
ManagedCHESS. Copyright (C) Microsoft Corporation, 2008.
Analyzing TestSimpleBank.dll ...
```

Standard Error

```
Tests: 1 Threads: 3 ExecSteps: 12 Time: 0.109
Tests: 2 Threads: 3 ExecSteps: 12 Time: 0.125
Tests: 3 Threads: 3 ExecSteps: 12 Time: 0.156
Search Completed (1 Warning, 2 Races).
```

Figure 3 - Microsoft's ChessBoard [3]



tid	op	obj
1	WaitHandle.WaitOne	520
1	EventWaitHandle.R...	520
1	Interlocked.Increment	514
1	TASK_FORK	4
1	TASK_RESUME	4
1	Interlocked.Increment	514
1	TASK_FORK	5
1	TASK_RESUME	5
1	Interlocked.Decrement	514
4	TASK_BEGIN	4
4	Interlocked.Decrement	514
1	WaitHandle.WaitOne	520
1	write	514

ChessTask1: ThreadPoolWait.cs

```
public bool WaitOne(int milli
{
    ThrowIfDisposed();
    DoneWorkItem();
    bool rv = _done.WaitOne
    // If the event is set, ther
    // reset so that this instar
    // event isn't set, put bac
    // DoneWorkItem.  The r
    // this unit will be remove
    if (rv)
    {
        remainingWorkItems
        _done.Reset();
    }
```

WaitOne
WaitOne

ChessTask4: ThreadPoolWait.cs

```
/// <summary>Executes the callback passed as state and signa
/// <param name="state"></param>
private void HandleWorkItem(object state)
{
    QueuedCallback qc = (QueuedCallback)state;
    try { qc.Callback(qc.State); }
    finally { DoneWorkItem(); }
}

/// <summary>Decrements the number of remaining items, signa
private void DoneWorkItem()
{
    if (Interlocked.Decrement(ref _remainingWorkItems) == 0)
        _done.Set();
}
```

DoneWorkItem
HandleWorkItem

name	tid	file	proc	operation
✓ ChessTask1	1	c:\Users\tball\chess\dev\main\Installer\Samples\Managed\T...	WaitOne	write

Figure 4 - Microsoft's Concurrency Explorer II [3]

CHESS is a concurrency testing tool developed by Microsoft for their .NET programs. Specifically, it is a tool that can find and reproduce Heisenbugs in concurrent programs. A Heisenbug is a software bug that disappears or alters its characteristics each time the software is run [1]. This specific type of bug makes debugging extremely difficult as it cannot be isolated or reproduced with ease. Most Heisenbug's are caused by data race detection errors - programming faults which cause the program to produce unpredictable results due to unsynchronized concurrent executions. A data race condition usually occurs in a concurrent program when multiple threads access the same shared memory without any synchronization [4]. Also, CHESS can find other types of bugs such as assertion errors, deadlocks, and live-locks. In order solve these types of errors, CHESS starts by using a model checker to determine all schedules that the program can take, saves them, and runs these schedules repeatedly. A schedule is basically a specific interleaving different threads can take to execute the software code. A interleaving can be defined as a transition or sets of transitions where control switches from one thread to a different thread so different pieces of the code can be executed. If a schedule returns an error, Microsoft CHESS can reproduce this error using the saved schedules.

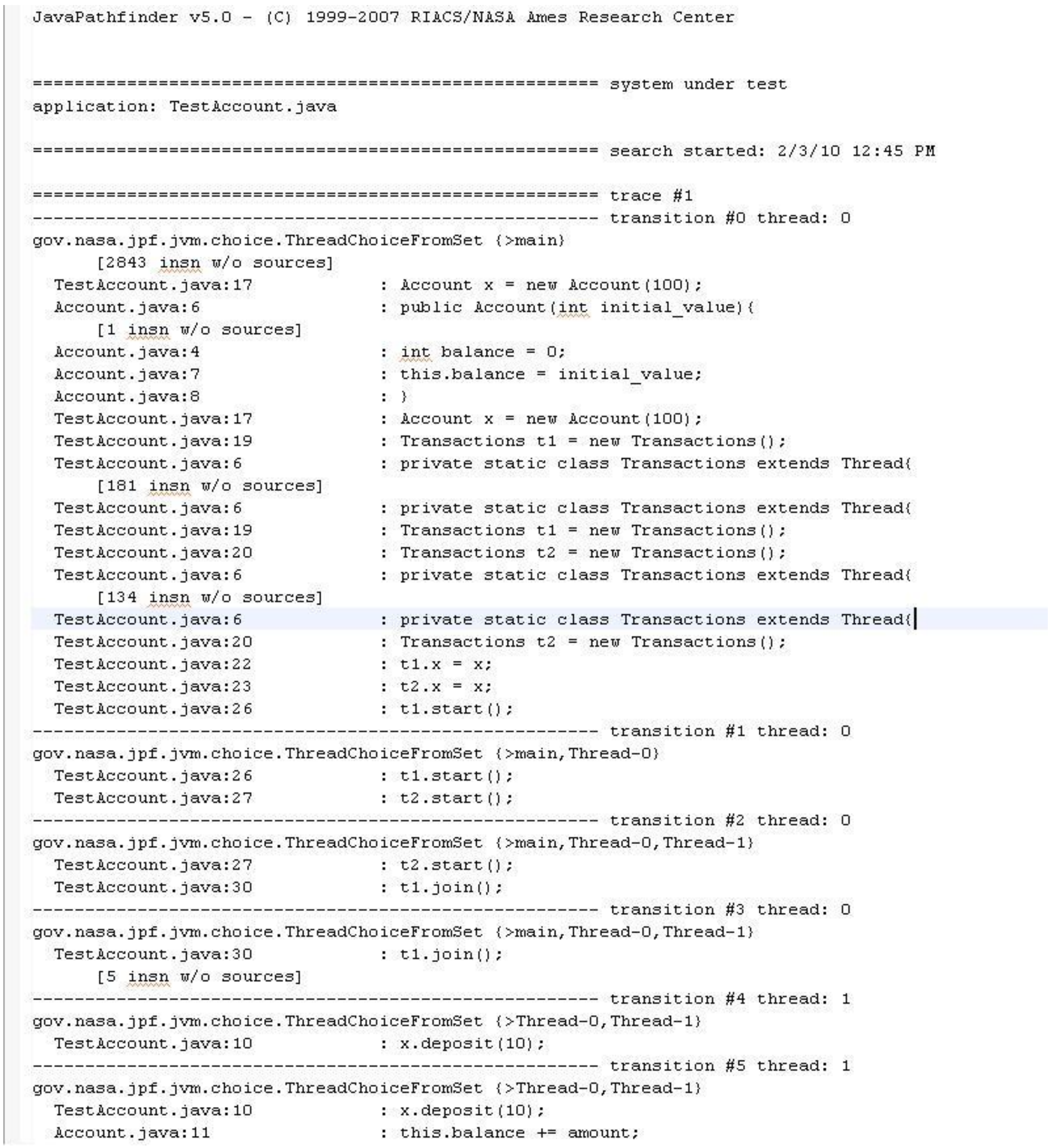
CHESS uses another tool called Chessboard to analyze these errors. As illustrated in Figure 3, Chessboard includes many components to display information of the different runs. These components include:

- Test View – This component provides information on the tests performed on the concurrency program and options to reproduce the trace of the program run
- Task View – This component provides information on the different tasks that were completed before and after running the tool against the current concurrency program

- Results View – This component provides the results of the current program run with extra information about the types of concurrency defects found, warnings, and notifications
- Detailed View – This component provides information on the task results, standard out, and standard error of the run

Also, Chessboard includes the schedule ID, number of execution steps, and the time it takes to execute the program for a specific run. Chessboard uses a tool called Concurrency Explorer to reproduce a specific schedule. It provides information such as the Thread ID, Operation, and Object ID. One important function that this tool includes is that all the source code from the executed program is placed in different windows on the right side of the Graphical User Interface. On the left side of the Graphical User Interface it shows the execution of a specific schedule step by step. Whenever a step is highlighted, the source code that had its code executed for that specific step is highlighted, as illustrated in Figure 4. This tool was closely studied to help develop our GUI for Java PathFinder.

2.3. Java PathFinder



```
JavaPathfinder v5.0 - (C) 1999-2007 RIACS/NASA Ames Research Center


====================================================== system under test
application: TestAccount.java

====================================================== search started: 2/3/10 12:45 PM

====================================================== trace #1
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  TestAccount.java:17              : Account x = new Account(100);
  Account.java:6                   : public Account(int initial_value){
      [1 insn w/o sources]
  Account.java:4                   : int balance = 0;
  Account.java:7                   : this.balance = initial_value;
  Account.java:8                   : }
  TestAccount.java:17              : Account x = new Account(100);
  TestAccount.java:19              : Transactions t1 = new Transactions();
  TestAccount.java:6               : private static class Transactions extends Thread{
      [181 insn w/o sources]
  TestAccount.java:6               : private static class Transactions extends Thread{
  TestAccount.java:19              : Transactions t1 = new Transactions();
  TestAccount.java:20              : Transactions t2 = new Transactions();
  TestAccount.java:6               : private static class Transactions extends Thread{
      [134 insn w/o sources]
  TestAccount.java:6               : private static class Transactions extends Thread{
  TestAccount.java:20              : Transactions t2 = new Transactions();
  TestAccount.java:22              : t1.x = x;
  TestAccount.java:23              : t2.x = x;
  TestAccount.java:26              : t1.start();
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}
  TestAccount.java:26              : t1.start();
  TestAccount.java:27              : t2.start();
------------------------------------------------------ transition #2 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:27              : t2.start();
  TestAccount.java:30              : t1.join();
------------------------------------------------------ transition #3 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:30              : t1.join();
      [5 insn w/o sources]
------------------------------------------------------ transition #4 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10              : x.deposit(10);
------------------------------------------------------ transition #5 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10              : x.deposit(10);
  Account.java:11                  : this.balance += amount;
```

Figure 5 - Java PathFinder Output Example

Java PathFinder is explicit state model checker developed by NASA for Java byte code programs [5]. Java PathFinder has several execution modes and extensions but we will only focus on those modes that are relevant for this thesis. The main focus of this tool is to find concurrency related defects such as deadlocks, live-locks, thread starvation, and data race conditions. We use Java PathFinder with a depth-first model checking scheme to find all schedules a concurrent program can take. The program will typically stop searching once the first defect is detected. This is how Java PathFinder determines if a program has a concurrency related bug, by enumerating all possible interleavings of a software program.

One of the problems of Java PathFinder is that it has a no Graphical User Interface to help with debugging the defect. As illustrated in Figure 5 and Appendix A, the output from Java PathFinder is all textual. This output can get fairly large as the number of threads increase in a program. Personally, I find that as the size of the textual data increases, the harder it is to debug the problem. Not only is it harder to debug a defect using textual outputs that have thousands and thousands of lines, it is also time consuming and stressful. There needs to be a way to take all this textual data and analyze it efficiently. The solution to this problem and the focus of my thesis is to create a Graphical User Interface, similar to Microsoft's Concurrency Explorer, to help debug Java PathFinder output.

3. Interface Design for Visualizing Java PathFinder Thread Schedules

3.1. Information Gathering Process

To determine the set of functions and components that the Graphical Visualization tool needed to provide, we gathered information on Microsoft CHESS and the functions that are provided by this tool to analyze its output. Once this research was completed, we then decided to implement features that Microsoft CHESS had provided with their tool that could work with Java PathFinder. One of the features we decided to include in the Graphical Visualization tool was the ability to display a specific error schedule with execution steps and thread specific coloring. We wanted to enhance this feature for our tool by allowing the user to view multiple schedules with their transitions and thread specific coloring. We also wanted to include the feature where it allowed the user to see the source code that Microsoft CHESS was executed against. Another feature was the ability to step through the given error schedule with code highlighted from the source code. We also simplified this process by adding a "Forward Step" and "Backward Step" buttons to make it easier for the user to step through the execution of a specific error schedule. Microsoft CHESS had done this by allowing the user to click a specific execution step in the error schedule which then highlighted the exact line of code in the source program. One of the disadvantages of Microsoft CHESS was that you can only view one error schedule at a time and so we had decided that our Graphical Visualization tool should support the display of multiple error schedules.

The second step to this process was to determine if Java PathFinder output could actually support the functions we wanted to implement from Microsoft CHESS. We started by examining the output from Java PathFinder. From this output, we could see that most of our functions could be implemented because the output provided detailed information about the run. In section 3.2, the Java PathFinder output gathering process will be explained.

3.2.Creating Input from the Output of Java PathFinder

In this thesis, we did not directly hook into Java PathFinder. We ran Java PathFinder on a sample program which had known concurrency problems using their interfaces and run configurations. I had used an Eclipse Plug-in to do this which made it simpler to get the output I needed from Java PathFinder. This was done by creating a text file with the extension ".jpf" with the name of the concurrent program to be tested, the types of concurrency related bugs that needed to be searched for, and other run configurations. This file is then run in Eclipse and a text file is created as output from the execution of Java PathFinder on the program. The output file from this run is then used as Input for the Graphical Visualization tool.

3.3. Stage One: Visualizing a Single Error Schedule Produced by Java PathFinder

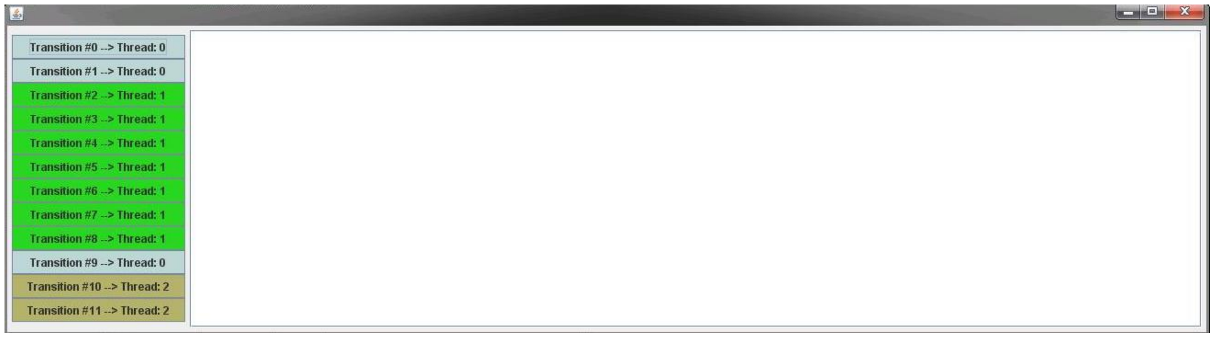

Transition #0 --> Thread: 0
Transition #1 --> Thread: 0
Transition #2 --> Thread: 1
Transition #3 --> Thread: 1
Transition #4 --> Thread: 1
Transition #5 --> Thread: 1
Transition #6 --> Thread: 1
Transition #7 --> Thread: 1
Transition #8 --> Thread: 1
Transition #9 --> Thread: 0
Transition #10 --> Thread: 2
Transition #11 --> Thread: 2

Figure 6 - 1st Graphical User Interface Prototype

In the beginning stages of implementation, I had created a sample program with a data race condition error and used Java PathFinder to find the first occurrence of this type of bug. This meant that Java PathFinder would only output the first interleaving that produced a data race condition bug. We had figured that the best point to start the initial tool is to visualize one schedule before looking at the challenge of visualizing many schedules produced by Java PathFinder. In stage one, this was accomplished by parsing the Java PathFinder output and using Java structures to store the information on the number of threads, the number of transitions, and the code executed during that transition. From this information, I had created a Graphical User Interface. In Figure 6, on the left hand side of the diagram, there is a column of buttons and colors that represent that specific interleaving. The different colors correspond to the different threads that were executed for the sample program. It can be seen that there were a total of three threads executed. There is also a text box in Figure 6 which is used to show the raw output, containing only the transition information, of Java PathFinder. This is can be done when the user clicks each button when the transition name on it on the left hand side of the Graphical User Interface.

3.4. Stage Two: Developing a High Level Visualization for Many Thread Interleavings

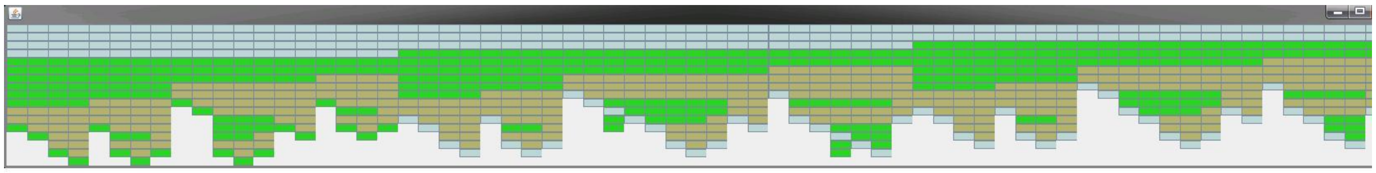
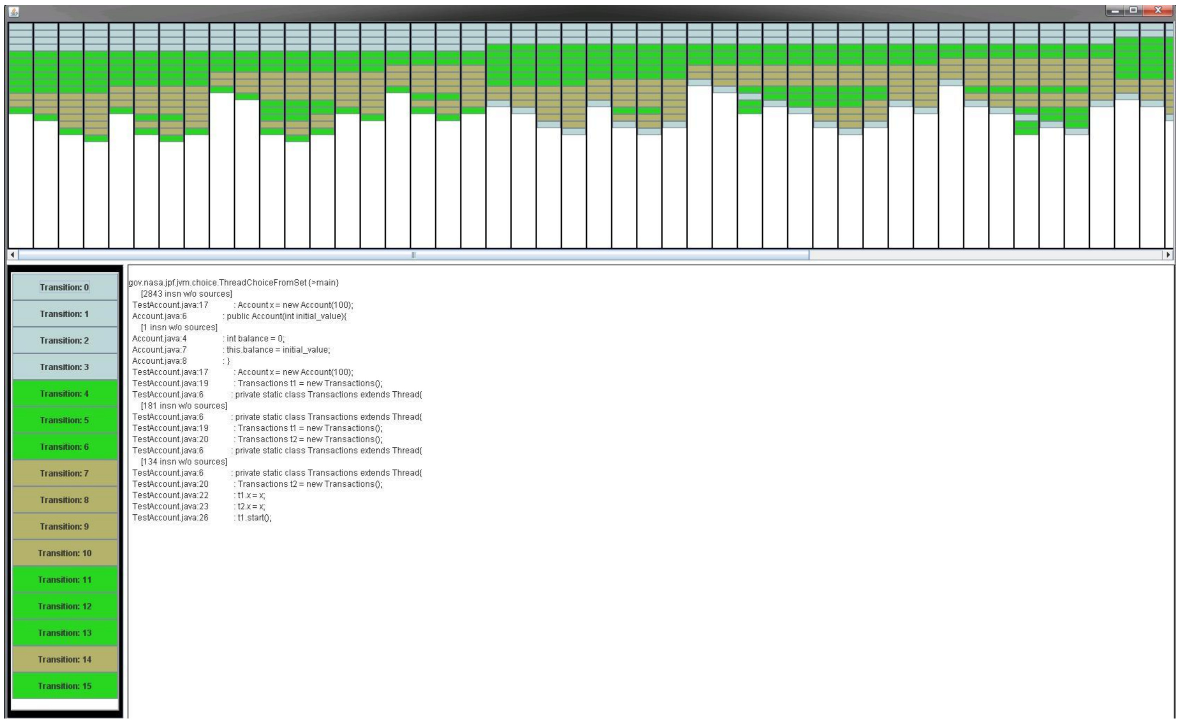


Figure 7 - 2nd Graphical User Interface Prototype

After my tool successfully created a single schedule Graphical User Interface, we had continued to develop the tool so it could take several schedules as input instead of just one. This part was a little more complicated and required the use of more complicated structures. In order to accomplish this, I had used Array Lists of Array Lists (almost like a 2D array of pointers). This structure was used to store information about each schedule and to store information about each transition in each schedule. Figure 7 represents the product of this idea. It is just a Graphical User Interface that contains the merged schedules and transitions of each schedule. Each column in Figure 7 represents a Schedule. Each button in each column represents a transition for that specific schedule. The colors of the buttons still represented the amount of threads in the program. This however was not as informative as the single schedule Graphical User Interface in Figure 6.

3.5. Stage 3: Combining the Visualizations for Many Interleavings and a Single Interleaving into one View



```
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
    [2843 insn w/o sources]
 TestAccount.java:17          : Account x = new Account(100);
 Account.java:6             : public Account(int initial_value){
    [1 insn w/o sources]
 Account.java:4             : int balance = 0;
 Account.java:7             : this.balance = initial_value;
 Account.java:8             : }
 TestAccount.java:17          : Account x = new Account(100);
 TestAccount.java:19          : Transactions t1 = new Transactions();
 TestAccount.java:6          : private static class Transactions extends Thread{
    [181 insn w/o sources]
 TestAccount.java:6          : private static class Transactions extends Thread{
 TestAccount.java:19          : Transactions t1 = new Transactions();
 TestAccount.java:20          : Transactions t2 = new Transactions();
 TestAccount.java:6          : private static class Transactions extends Thread{
    [134 insn w/o sources]
 TestAccount.java:6          : private static class Transactions extends Thread{
 TestAccount.java:20          : Transactions t2 = new Transactions();
 TestAccount.java:22          : t1.x = x;
 TestAccount.java:23          : t2.x = x;
 TestAccount.java:26          : t1.start();
```

Figure 8 – 4th Graphical User Interface Prototype

One of the problems of the Graphical User Interface in Figure 7 is that it was not informative enough. We needed a way to display the executed code of each transition for each schedule. The solution to this problem was to merge the first and second prototypes. The result of this merge can be seen in Figure 8. The top panel in the Graphical User Interface shows all enumerated schedules by Java PathFinder. The left hand Panel shows specific Schedules that dynamically change when you click on one the schedule buttons on the top panel. Also, when a user clicked a transition button on the left hand panel, the code executed code is displayed in the center of the Graphical User Interface in a text box.

One of the problems with Figure 7 prototype was that the user had problems determining what a schedule actually represented. What we had done to correct this confusion was to add bold black lines onto the top panel which separated the schedules. Also, an interesting point to note is that when you look at the top panel, you can visibly see how many transitions it took the sample program, for a specific schedule, to arrive at the error. Some schedules are a lot shorter in length then others. This type of visual information can help the user isolate bug paths that are harder to reproduce just by looking at the top panel.

3.6. Stage 4: A Refined Visualization that includes the Java source files

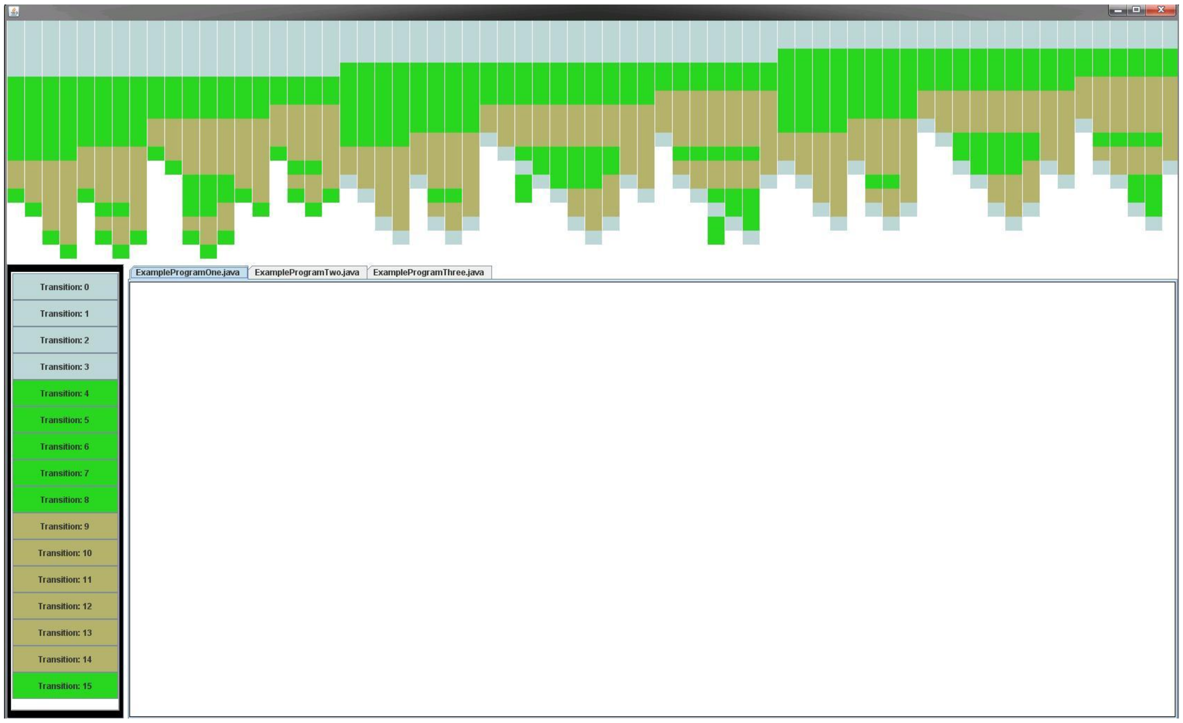
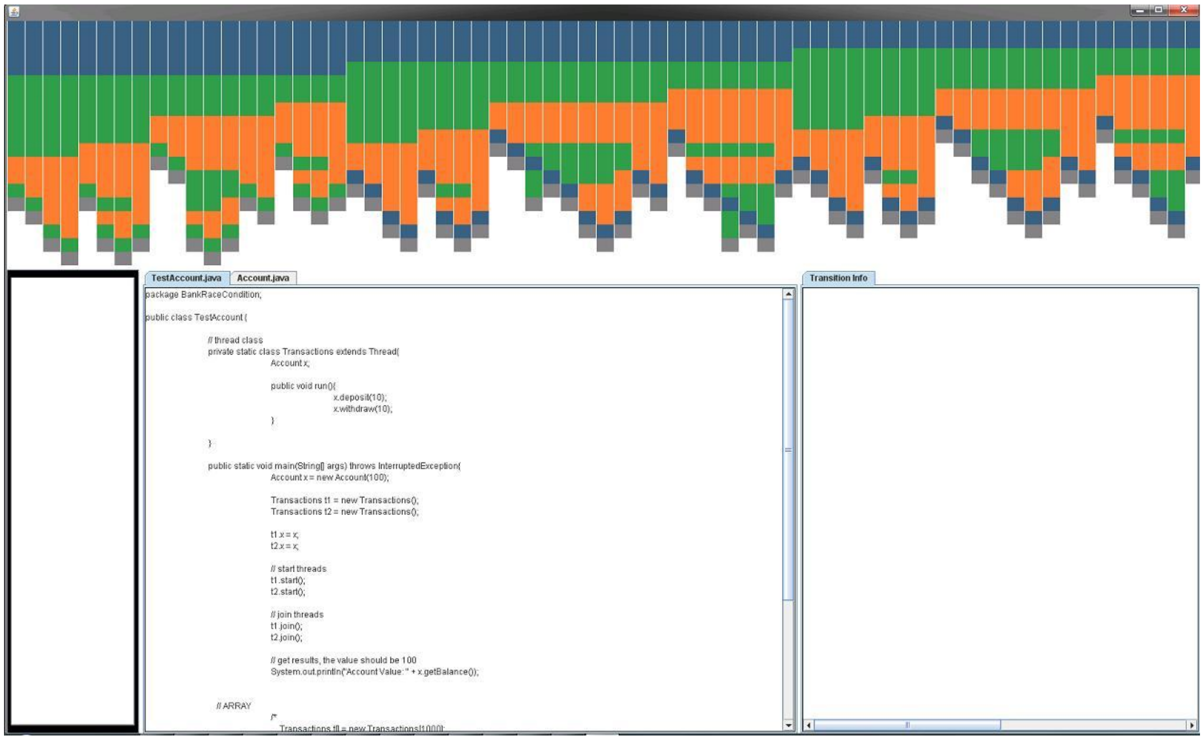


Figure 9 - 7th Graphical User Interface

We had decided that it was best to remove the buttons from the top panel and actually draw the squares using Java Graphics. This is illustrated in Figure 9. This had sharpened the look of the Graphical User Interface. The buttons were replaced by interaction code to determine the coordinates of the mouse clicks. When the user clicks a specific column on the top panel, that specific schedule would be displayed on the lower left hand side of the Graphical User Interface as before. Another interesting thing that was added to this prototype was the tabs in the center of the tool. The functionality of these tabs was not implemented yet but they were supposed to be used to hold the source code for the program that Java PathFinder was run against.

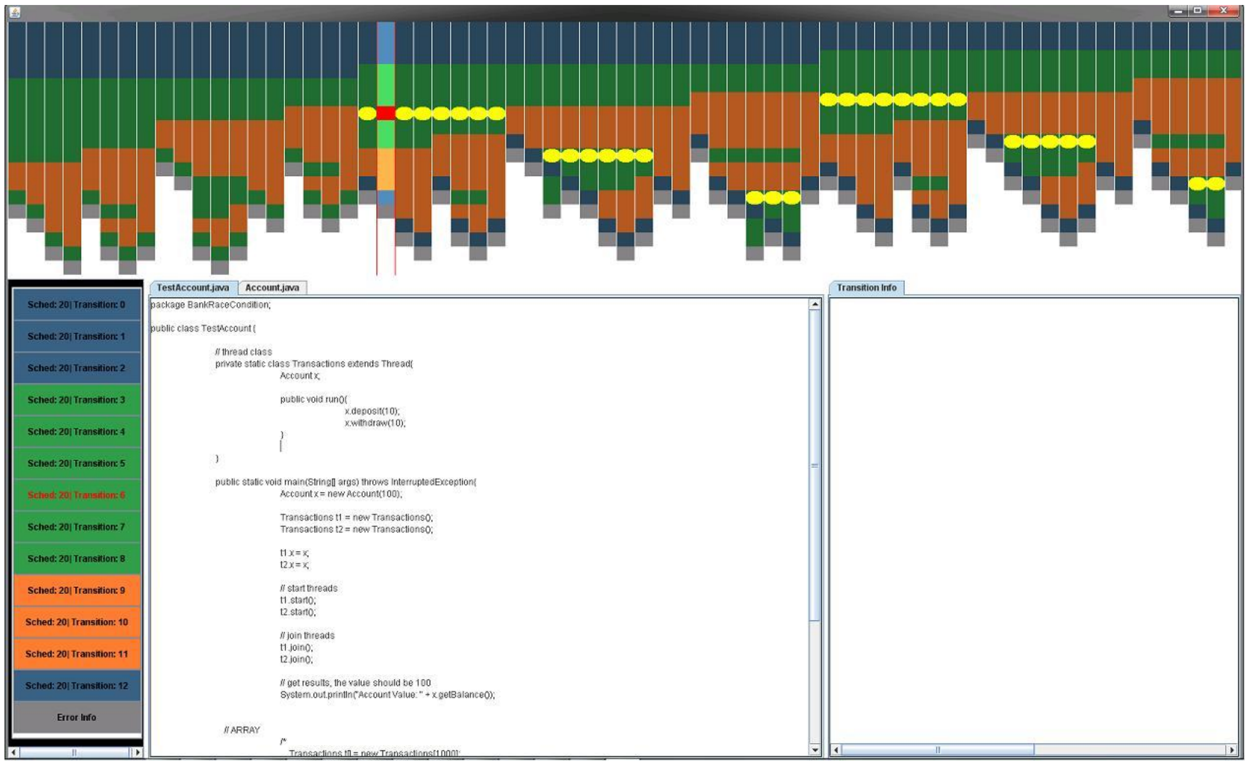
3.7. Stage Five: A Refined Visualization that includes new Color Theme and Separate Transition Information



```
package BankRaceCondition;

public class TestAccount {

        // thread class
        private static class Transactions extends Thread{
                Account x;

                public void run(){
                        x.deposit(10);
                        x.withdraw(10);
                }

        }

        public static void main(String[] args) throws InterruptedException{
                Account x = new Account(100);

                Transactions t1 = new Transactions();
                Transactions t2 = new Transactions();

                t1.x = x;
                t2.x = x;

                // start threads
                t1.start();
                t2.start();

                // join threads
                t1.join();
                t2.join();

                // get results, the value should be 100
                System.out.println("Account Value: " + x.getBalance());


         // ARRAY
                /*
                  Transactions t[] = new Transactions[1000];
```

Figure 10 - 9th Graphical User Interface Prototype

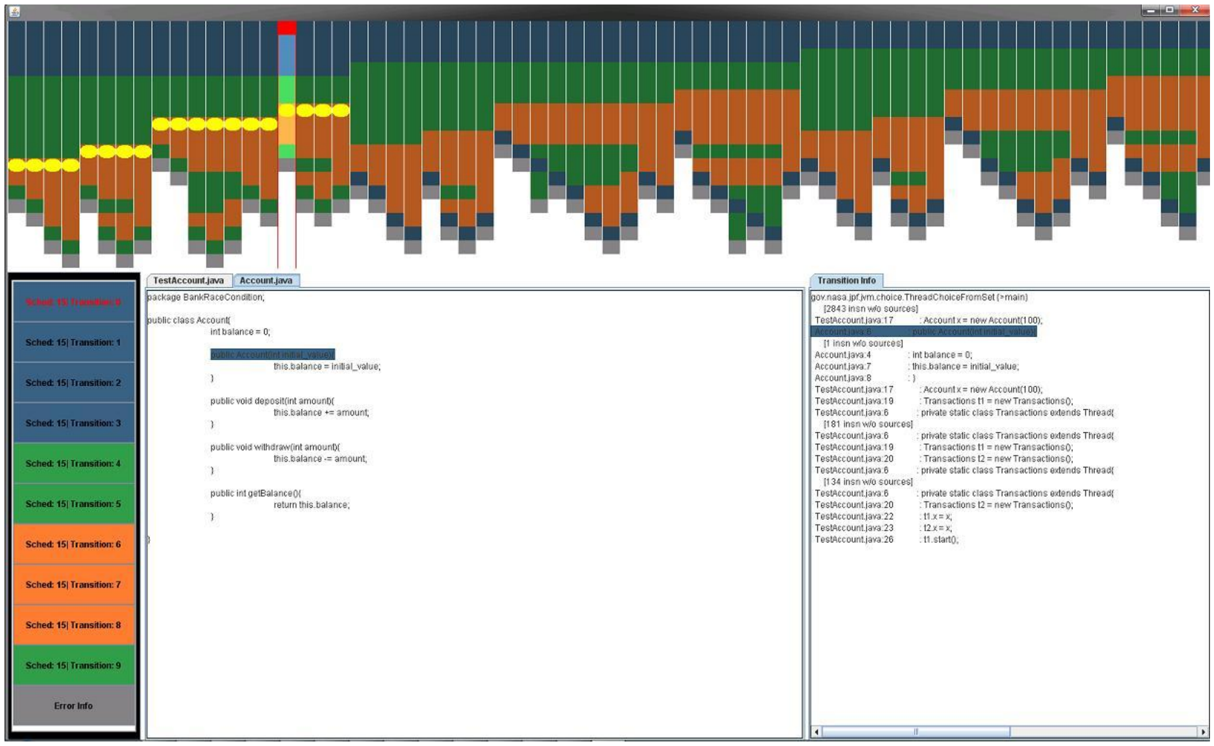
In Figure 9, it can be seen that many things have been added to the previous Graphical User Interface prototype. One of the major changes that were made is the color scheme of the threads. We decided that the original color scheme could cause perception issues for the user. For example, the green and beige color were too similar which could distort the users analyzes of the top view. We had then changed the color scheme from the randomized thread color to preset color values for the first ten threads which clearly could be distinguished. This means that Thread 0 to Thread 9 will always have the same color regardless of the program Java PathFinder runs against. Also, it can be seen in Figure 9 that the source code now loads into the middle text box. For this specific sample program, there are two Java files used to run this program, thus there are two tabs for each of the files. On the lower right hand side of the Graphical User Interface we had decided to add the transition information back just in case users wanted to still see the raw data. Another change that can be seen in Figure 10 is the addition of grey blocks at the end of each schedule. This is called the error block. When you click the grey block, the final error information of the specific schedule is displayed in the "transition info" tab.



```
package BankRaceCondition;

public class TestAccount {

        // thread class
        private static class Transactions extends Thread{
                Account x;

                public void run(){
                        x.deposit(10);
                        x.withdraw(10);
                }
                |
        }

        public static void main(String[] args) throws InterruptedException{
                Account x = new Account(100);

                Transactions t1 = new Transactions();
                Transactions t2 = new Transactions();

                t1.x = x;
                t2.x = x;

                // start threads
                t1.start();
                t2.start();

                // join threads
                t1.join();
                t2.join();

                // get results, the value should be 100
                System.out.println("Account Value: " + x.getBalance());


         // ARRAY
                /*
```

Figure 11 - 9th Graphical User Interface Prototype with Highlights

Figure 11 shows the same prototype once again but with one of the tool's newest functionality. The red box around the transition now represents that the user has clicked that transition in that schedule. On the lower left hand side, the transition's text is also highlighted in red. Also, as well as highlighting the transition selected by the user, the yellow dots now represent similar code that was executed in other transitions in other schedules. This new functionality can help the user do many things. First, the user can now identify which schedules have the same error codes or resulted in the same error if he clicks the grey block of a schedule. Second, the user can also find out if there are repeated schedules that he or she can ignore. And finally, the user can sort schedules based on the types of errors encountered, but the sorting functionality has not been implemented.



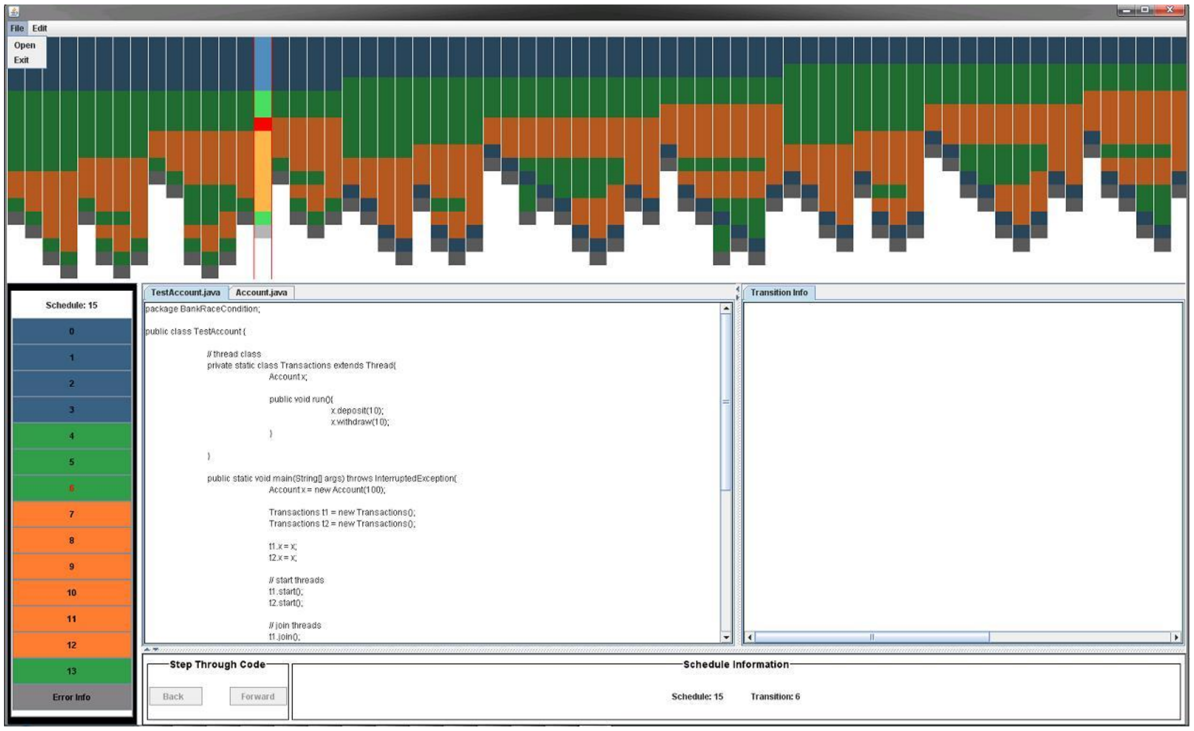
```
package BankRaceCondition;

public class Account{
          int balance = 0;

          public Account(int initial_value){
                    this.balance = initial_value;
          }

          public void deposit(int amount){
                    this.balance += amount;
          }

          public void withdraw(int amount){
                    this.balance -= amount;
          }

          public int getBalance(){
                    return this.balance;
          }

}
```

```
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
   [2843 insn w/o sources]
 TestAccount.java:17        : Account x = new Account(100);
 Account.java:6             : public Account(int initial_value){
   [1 insn w/o sources]
 Account.java:4             : int balance = 0;
 Account.java:7             : this.balance = initial_value;
 Account.java:8             : }
 TestAccount.java:17        : Account x = new Account(100);
 TestAccount.java:19        : Transactions t1 = new Transactions();
 TestAccount.java:6         : private static class Transactions extends Thread{
   [181 insn w/o sources]
 TestAccount.java:6         : private static class Transactions extends Thread{
 TestAccount.java:19        : Transactions t1 = new Transactions();
 TestAccount.java:20        : Transactions t2 = new Transactions();
 TestAccount.java:6         : private static class Transactions extends Thread{
   [134 insn w/o sources]
 TestAccount.java:6         : private static class Transactions extends Thread{
 TestAccount.java:20        : Transactions t2 = new Transactions();
 TestAccount.java:22        : t1.x = x;
 TestAccount.java:23        : t2.x = x;
 TestAccount.java:26        : t1.start();
```

Figure 12 - 9th Graphical User Interface with Code Walk through Highlights

Another functionality that was added in this prototype is the ability to walk through the code that is executed for a specific schedule in order to reach the bug. This is an example of a classic debugger, but this is done visually. When the user chooses a specific schedule and it is displayed on the lower left hand panel, the user now has an option to walk through the execution of the schedule transition by transition. When you click a transition on the left panel, the transition information will appear on the far right panel. Also, the first line of code executed in that transition will also be highlighted on the far right panel as well as the source code it appears in on the middle tabbed panel. This can be seen in Figure 12. If there is more code to be executed in the current transition, clicking the button once again will take you to the next line of code that was executed and highlight where it was executed in the source code. This can be done for every transition in every schedule.

3.8. Stage Six: Final Visualization Tool

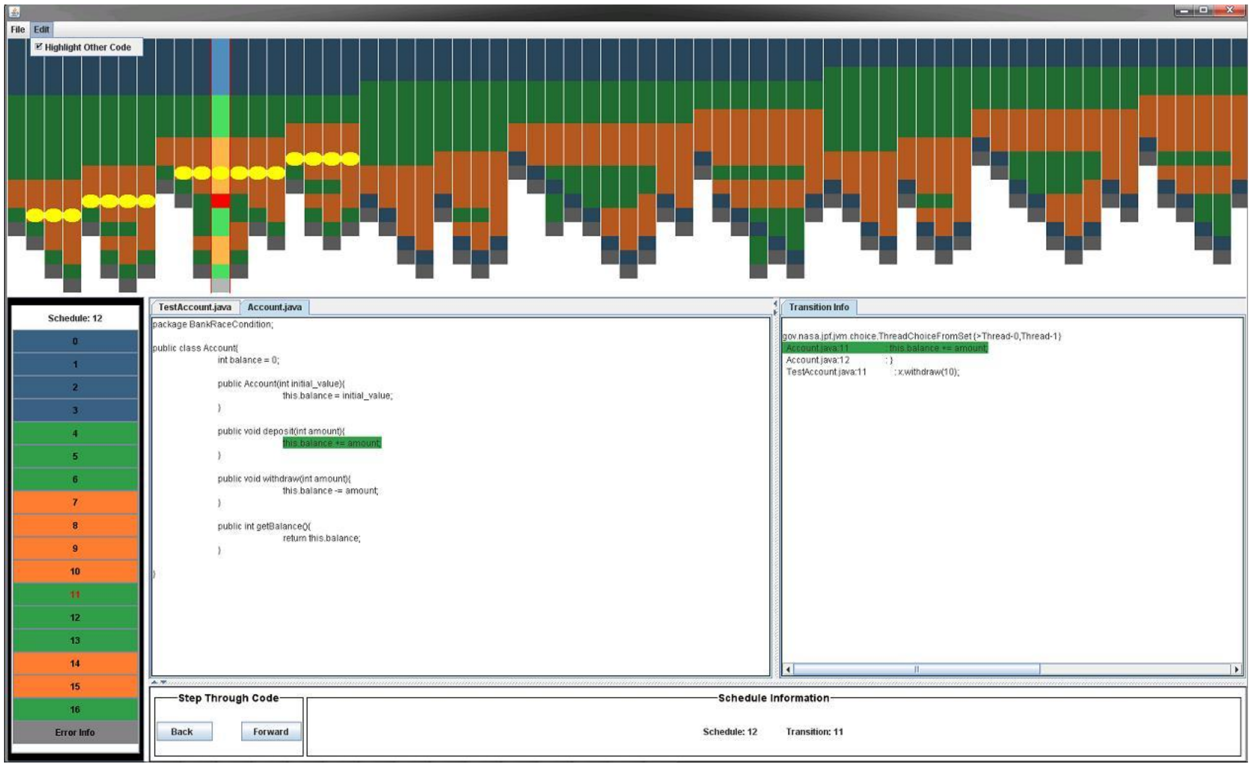


```
package BankRaceCondition;

public class TestAccount {

        // thread class
        private static class Transactions extends Thread{
                Account x;

                public void run(){
                        x.deposit(10);
                        x.withdraw(10);
                }

        }

        public static void main(String[] args) throws InterruptedException{
                Account x = new Account(100);

                Transactions t1 = new Transactions();
                Transactions t2 = new Transactions();

                t1.x = x;
                t2.x = x;

                // start threads
                t1.start();
                t2.start();

                // join threads
                t1.join();
```

Schedule: 15 Transition: 6

Figure 13 - Final Graphical User Interface

One of the new additions to add to the final version of the Graphical User Interface is the option to load Java PathFinder output files manually. This can be seen in Figure 13 on the top left hand corner. There is now “File” menu option along with a “Edit” option. This allows the user to now quickly load in any output file he or she needs to analyze. A change was made to the left hand side panel which was the amount of text that appeared on the buttons. There was too much repetitive text on these buttons and so we had decided to just keep the numbering of the transitions and the schedule identification on top part of the left hand side panel.



```
package BankRaceCondition;

public class Account{
        int balance = 0;

        public Account(int initial_value){
                this.balance = initial_value;
        }

        public void deposit(int amount){
                this.balance += amount;
        }

        public void withdraw(int amount){
                this.balance -= amount;
        }

        public int getBalance(){
                return this.balance;
        }

}
```

```
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
 Account.java:11          : this.balance += amount;
 Account.java:12          : }
 TestAccount.java:11          : x.withdraw(10);
```

Schedule: 12   Transition: 11

Figure 14 - Final Graphical User Interface with Automatic Debug Buttons

The "Edit" menu button in Figure 14 allows you to toggle the option to keep the code highlighting functionality which highlights similar code in other schedules. Also, a major addition to this Graphical User Interface is the addition of the lower panel. The lower panel contains two buttons, Back and Forward. These buttons now automatically step through the code for a specific transition line by line without having to actually press the transition buttons on the left hand side panel. Users can now step forward through the execution or you can step backwards in the execution to determine how the bug had been reached. This is one of the more significant functionalities in this tool.

4. Conclusion

There were several motivations behind the reasons for researching concepts of concurrent programming. One of the major motivations was that I wanted to research concurrent programming because it was one of the more significant topics in computer science. During this research, I have learned a great deal about concurrency releated issues. I have also learned about how model checking systems work and the different problems that can occur when using a model checker. From my research, it can be seen that one of the major problems of concurrent programming is testing these programs. Several notewhorthy companies have created tools to help solve this significant problem such as Microsoft and NASA. Microsoft had created a tool called CHESS which uses a model checker to determine defects found in a concurrent program. Microsoft CHESS uses sever al Graphical User Interfaces to help programmers debug their concurrent programs. One of the major things I have learned from researching into Microsoft is the way they visually represent the output data from their tool and their different functionality for this tool. NASA had created a tool called PathFinder that also uses a model checker to find defects in concurrent programs. One of the problems of Java PathFinder was that it did not provide a Graphical User Interface for debugging the defects found in the concurrent program but instead gave a textual output outlining the errors. From the research of Microsoft CHESS, we have determined that the best possible solution to NASA's textual output problem is to create a Graphical Visualization tool similar to CHESS. In this thesis, I researched and developed a Graphical Visualization tool for NASA PathFinder to help enhance concurrency testing.

5. Benefits and Future Work for Graphical Visualization Tool

There are several benefits provided by the Graphical Visualization tool when comparing it to the textual output that Java PathFinder gives. One of the advantages is that it simplifies debugging. I have implemented functions that allows a user to find the schedule they want to debug and use different buttons to step forward or backward within the transitions to determine how the defect was found. Another benefit of the Graphical Visualization tool is that it allows the user to determine whether or not the schedules that were enumerated were repeated by enabling code highlights of similar code and the top panel Graphical User Interface. Code highlighting also allows the user to determine what schedules ended with the same type of defect. Another importance of the top panel Graphical User Interface is that the user can quickly determine the difficult bugs to reproduce by looking at the depth of each schedule. I think the biggest benefit of this tool is the simple fact that it takes less time to debug a defect using a Graphical User Interface versus a textual representation of the output. Not only does it take less time, but it is more efficient and less stressful for users. This tool can be enhanced further even after this thesis is done. Some of the future works include syntax highlighting for the source code, the ability to sort the different schedules, drag and drop options when moving a specific schedule from the top panel to the left panel, and different graphical representations of the schedules.

6. Bibliography

1. M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. Nainar, and I. Neamtiu, "Finding and reproducing heisenbugs in concurrent programs," Proc. of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08), 2008.

2. NASA**.** Testing vs. Model Checking. *JPF.* [Online] 2010. http://babelfish.arc.nasa.gov/trac/jpf/wiki/intro/testing_vs_model_checking.

3. Microsoft Corporation (2010, April 23)**.** CHESS. *Microsoft Research*. http://research.microsoft.com/en-us/projects/chess/.

4. R.H. Netzer and B.P. Miller, "What are race conditions?: Some issues and formalizations," ACM Lett. Program. Lang. Syst., vol. 1, 1992, pp. 74-88.

5. K. Havelund and T. Pressburger, "Model Checking: Java Programs Using Java PathFinder," International ournal on Software Tools for Technology Transfer (STTT), vol. 2, 2000.

7. Appendix A – Example Input for Java PathFinder “TestAccount.jpf”

```
# JPF properties to run the TestAccount example

target=TestAccount

report.publisher=console,xml
listener=gov.nasa.jpf.listener.PreciseRaceDetector
report.console.property_violation=trace
search.multiple_errors=true
report.console.file=Test_Gowritharan
```

8. Appendix B – Condensed Example Output from Java PathFinder Run against Sample Program

```
JavaPathfinder v5.0 - (C) 1999-2007 RIACS/NASA Ames Research Center


====================================================== system under test
application: TestAccount.java

====================================================== search started: 2/3/10 12:45 PM

====================================================== trace #1
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  TestAccount.java:17            : Account x = new Account(100);
  Account.java:6                 : public Account(int initial_value){
      [1 insn w/o sources]
  Account.java:4                 : int balance = 0;
  Account.java:7                 : this.balance = initial_value;
  Account.java:8                 : }
  TestAccount.java:17            : Account x = new Account(100);
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [181 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [134 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:22            : t1.x = x;
  TestAccount.java:23            : t2.x = x;
  TestAccount.java:26            : t1.start();
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}
  TestAccount.java:26            : t1.start();
  TestAccount.java:27            : t2.start();
------------------------------------------------------ transition #2 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:27            : t2.start();
  TestAccount.java:30            : t1.join();
------------------------------------------------------ transition #3 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:30            : t1.join();
      [5 insn w/o sources]
------------------------------------------------------ transition #4 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #5 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #6 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #7 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
  Account.java:12                : }
  TestAccount.java:11            : x.withdraw(10);
------------------------------------------------------ transition #8 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:11            : x.withdraw(10);
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #9 thread: 1
```

```
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #10 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #11 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #12 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:15                : this.balance -= amount;

====================================================== trace #2
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  TestAccount.java:17            : Account x = new Account(100);
  Account.java:6                 : public Account(int initial_value){
      [1 insn w/o sources]
  Account.java:4                 : int balance = 0;
  Account.java:7                 : this.balance = initial_value;
  Account.java:8                 : }
  TestAccount.java:17            : Account x = new Account(100);
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [181 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [134 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:22            : t1.x = x;
  TestAccount.java:23            : t2.x = x;
  TestAccount.java:26            : t1.start();
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}
  TestAccount.java:26            : t1.start();
  TestAccount.java:27            : t2.start();
------------------------------------------------------ transition #2 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:27            : t2.start();
  TestAccount.java:30            : t1.join();
------------------------------------------------------ transition #3 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:30            : t1.join();
      [5 insn w/o sources]
------------------------------------------------------ transition #4 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #5 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #6 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #7 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
  Account.java:12                : }
  TestAccount.java:11            : x.withdraw(10);
------------------------------------------------------ transition #8 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  TestAccount.java:11            : x.withdraw(10);
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #9 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:15                : this.balance -= amount;
```

```
------------------------------------------------------ transition #10 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #11 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #12 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {Thread-0,>Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #13 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>Thread-0,Thread-1}
  Account.java:15                : this.balance -= amount;
```

…
…   **(trace 3 to 65 information removed)**
…

```
====================================================== trace #66
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  TestAccount.java:17            : Account x = new Account(100);
  Account.java:6                 : public Account(int initial_value){
      [1 insn w/o sources]
  Account.java:4                 : int balance = 0;
  Account.java:7                 : this.balance = initial_value;
  Account.java:8                 : }
  TestAccount.java:17            : Account x = new Account(100);
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [181 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [134 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:22            : t1.x = x;
  TestAccount.java:23            : t2.x = x;
  TestAccount.java:26            : t1.start();
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}
  TestAccount.java:26            : t1.start();
  TestAccount.java:27            : t2.start();
------------------------------------------------------ transition #2 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #3 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #4 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #5 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #6 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #7 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:11                : this.balance += amount;
  Account.java:12                : }
  TestAccount.java:11            : x.withdraw(10);
------------------------------------------------------ transition #8 thread: 1
```

```
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #9 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:11            : x.withdraw(10);
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #10 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #11 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  Account.java:11                : this.balance += amount;
  Account.java:12                : }
  TestAccount.java:11            : x.withdraw(10);
------------------------------------------------------ transition #12 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  TestAccount.java:11            : x.withdraw(10);
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #13 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #14 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:27            : t2.start();

====================================================== trace #67
------------------------------------------------------ transition #0 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main}
      [2843 insn w/o sources]
  TestAccount.java:17            : Account x = new Account(100);
  Account.java:6                 : public Account(int initial_value){
      [1 insn w/o sources]
  Account.java:4                 : int balance = 0;
  Account.java:7                 : this.balance = initial_value;
  Account.java:8                 : }
  TestAccount.java:17            : Account x = new Account(100);
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [181 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:19            : Transactions t1 = new Transactions();
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:6             : private static class Transactions extends Thread{
      [134 insn w/o sources]
  TestAccount.java:6             : private static class Transactions extends Thread{
  TestAccount.java:20            : Transactions t2 = new Transactions();
  TestAccount.java:22            : t1.x = x;
  TestAccount.java:23            : t2.x = x;
  TestAccount.java:26            : t1.start();
------------------------------------------------------ transition #1 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0}
  TestAccount.java:26            : t1.start();
  TestAccount.java:27            : t2.start();
------------------------------------------------------ transition #2 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #3 thread: 1
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,>Thread-0,Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #4 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
------------------------------------------------------ transition #5 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:10            : x.deposit(10);
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #6 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:11                : this.balance += amount;
------------------------------------------------------ transition #7 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:11                : this.balance += amount;
  Account.java:12                : }
  TestAccount.java:11            : x.withdraw(10);
------------------------------------------------------ transition #8 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  TestAccount.java:11            : x.withdraw(10);
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #9 thread: 2
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {main,Thread-0,>Thread-1}
  Account.java:15                : this.balance -= amount;
------------------------------------------------------ transition #10 thread: 0
gov.nasa.jpf.jvm.choice.ThreadChoiceFromSet {>main,Thread-0,Thread-1}
  TestAccount.java:27            : t2.start();

====================================================== results
error #1: gov.nasa.jpf.listener.PreciseRaceDetector "race for field Account@290.balance
Thread-0 at ..."
error #2: gov.nasa.jpf.listener.PreciseRaceDetector "race for field Account@290.balance
Thread-0 at ..."



…
…                      (trace 3 to 65 information removed)
…



error #66: gov.nasa.jpf.listener.PreciseRaceDetector "race for field Account@290.balance
Thread-0 at ..."
error #67: gov.nasa.jpf.listener.PreciseRaceDetector "race for field Account@290.balance
Thread-0 at ..."

====================================================== statistics
elapsed time:       0:00:00
states:             new=447, visited=401, backtracked=847, end=18
search:             maxDepth=18, constraints=0
choice generators:  thread=381, data=0
heap:               gc=936, new=429, free=420
instructions:       7495
max memory:         6MB
loaded code:        classes=75, methods=1015

====================================================== search finished: 2/3/10 12:45 PM
```

9. Appendix C – Source Code from Sample Program “Account.java”

```
package BankRaceCondition;
public class Account{
        int balance = 0;
        public Account(int initial_value){
                this.balance = initial_value;
        }
        public void deposit(int amount){
                this.balance += amount;
        }

        public void withdraw(int amount){
                this.balance -= amount;
        }

        public int getBalance(){
                return this.balance;
        }

}
```

10. Appendix D – Source Code from Main Sample Program “TestAccount.java”

```
package BankRaceCondition;
public class TestAccount {
		// thread class
		private static class Transactions extends Thread{
				Account x;
				public void run(){
						x.deposit(10);
						x.withdraw(10);
				}

		}

		public static void main(String[] args) throws InterruptedException{
				Account x = new Account(100);
				Transactions t1 = new Transactions();
				Transactions t2 = new Transactions();

				t1.x = x;
				t2.x = x;
				// start threads
				t1.start();
				t2.start();
				// join threads
				t1.join();
				t2.join();
				// get results, the value should be 100
				System.out.println("Account Value: " + x.getBalance());
	// ARRAY
				/*
				  Transactions t[] = new Transactions[1000];
						for(int i = 0; i < t.length; i++){
								t[i] = new Transactions();
								t[i].x = x;
						}
						for(int i = 0; i < t.length; i++){
						t[i].start();
				  }
						for(int i = 0; i < t.length; i++){
						t[i].join();
				  }
				*/
		}
}
```

11. Appendix E – Source Code for Completed Visualization Tool

CreateGUI.java

```
package ThreadGUI;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.LayoutManager;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Vector;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.LineBorder;
import javax.swing.border.TitledBorder;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultHighlighter;
import javax.swing.text.Highlighter;


public class CreateGUI extends JFrame {

	// parseFile variables
	private static int trace_Num =0;
	private static int thread_Num=0;
```

```
		private static List<String> programFileNames = new ArrayList<String>(); // This array is to keep track of the different program files
used
		private static List<String> transition_states_error = new ArrayList<String>();

		private static List<Integer> transition_Num = new ArrayList<Integer>();


		private static List<List<Integer>> transition_states = new ArrayList<List<Integer>>();
		private static List<List<String>> transition_states_info = new ArrayList<List<String>>();

		// to generate random numbers
		private Dimension scrnsize;

		private List<List<Integer>> transition_states_color = new ArrayList<List<Integer>>();
		private List<List<Integer>> transition_states_ours = new ArrayList<List<Integer>>();
		private List<String> transition_errors = new ArrayList<String>();
		private List<List<String>> transition_states_info_ours = new ArrayList<List<String>>();

		private final PaintTopGUI topPaint;
		private Color[] threadColor;
		private Color[] genColors;
		private int tranistionInfoPos = 0; // we are going to process transition info line by line, this keep tracks of which line we are on
		private int buttonID = 0;
		private String buttonCMD = "";
		private String currentLine = "";
		private String currentLineProgram = "";
		private int currentLineProgramIndex =0;
		private int startIndex = 0;
		private int endIndex = 0;
		private int curTrace = 0;
		private boolean programLineFound = false;

		private static CreateGUI temp;

		private Highlighter.HighlightPainter[] threadHighlighter;
		private List<String> programFileNames_ours = new ArrayList<String>();

		private final List<List<JButton>> leftPanelBtns;
		private final List<JLabel> leftPanelLbls;
		private int curLeft_rowPos = 0;
		private int curLeft_colPos = 0;

		private JScrollPane[] scrollingArea;
		private JTabbedPane tabbedPaneCenter;
		private JTabbedPane tabbedPaneEast;
		private JTextArea[] programInfo;

		private int[][] transitionInfoPos;
		private int transitionInfoPos_size;

		// File Chooser
		private static JFileChooser fc;

		// Misc.
    private boolean menuHighlight = false;

    // Label for the left buttons
    private JLabel leftPanelLabel;


    // Step Through Panel
    private JPanel stepThroughPane;
    private JPanel stepThroughPaneBtns;
    private JPanel stepThroughPaneInfo;
    private JButton stepForward;
    private JButton stepBack;
private JLabel stepThroughCurrentTransition;
private JLabel stepThroughCurrentSched;

private boolean nextTransition = false;
private int oldTransInfoPos = -1;

        public CreateGUI(){

                  /*
                   *  Step Through Panel Set Up
                   */
                  stepThroughPaneBtns = new JPanel();
                  stepThroughPaneBtns.setLayout(new GridBagLayout());
                  stepThroughPaneBtns.setBackground(Color.WHITE);

                  stepThroughPaneInfo = new JPanel();
                  stepThroughPaneInfo.setLayout(new GridBagLayout());
                  stepThroughPaneInfo.setBackground(Color.WHITE);

                  stepThroughPane = new JPanel();
                  stepThroughPane.setLayout(new BorderLayout());
                  stepThroughPane.setBackground(Color.WHITE);

                  GridBagConstraints d = new GridBagConstraints();

                  // add Step Back Button
                  d.fill = GridBagConstraints.WEST;
                  stepBack = new JButton(" Back   ");
                  stepBack.addActionListener(
                              new ActionListener() {
                                public void actionPerformed(ActionEvent e) {
                                }
                              }
                              );


                  d.gridx = 0;
                  d.gridy = 3;
                  stepThroughPaneBtns.add(stepBack, d);

                  // space filler
                  JLabel fillerOne = new JLabel("            ");
                  d.gridx = 6;
                  d.gridy = 3;
                  stepThroughPaneBtns.add(fillerOne, d);

                  // add Step Back Button
                  d.fill = GridBagConstraints.EAST;
                  stepForward = new JButton("Forward");



                  d.gridx = 12;
                  d.gridy = 3;
                  stepThroughPaneBtns.add(stepForward, d);

                  // disable them
                  stepForward.setEnabled(false);
                  stepBack.setEnabled(false);

                  TitledBorder stepThroughBtnLbl = new TitledBorder(
      new LineBorder(Color.black),
      "Step Through Code",
      TitledBorder.CENTER,
      TitledBorder.CENTER);
```

```
                        stepThroughBtnLbl.setTitleColor(Color.black);

                        Font curFontA = stepThroughBtnLbl.getTitleFont();
                        stepThroughBtnLbl.setTitleFont(new Font(curFontA.getFontName(), curFontA.getStyle(), 14));

                        stepThroughPaneBtns.setBorder(stepThroughBtnLbl);

                        // Creating the Information tab
                        d.gridx = 0;
                        d.gridy = 0;
                        stepThroughCurrentSched = new JLabel("Schedule: N/A");
                        stepThroughPaneInfo.add(stepThroughCurrentSched, d);

                        // add filler
                        JLabel fillerTwo = new JLabel("            ");
                        d.gridx = 6;
                        d.gridy = 0;
                        stepThroughPaneInfo.add(fillerTwo, d);

                        d.gridx = 12;
                        d.gridy = 0;
                        stepThroughCurrentTransition = new JLabel("Transition: N/A");
                        stepThroughPaneInfo.add(stepThroughCurrentTransition, d);

                        TitledBorder stepThroughInfoLbl = new TitledBorder(
        new LineBorder(Color.black),
        "Schedule Information",
        TitledBorder.CENTER,
        TitledBorder.CENTER);
                        stepThroughInfoLbl.setTitleColor(Color.black);

                        Font curFontB = stepThroughInfoLbl.getTitleFont();
                        stepThroughInfoLbl.setTitleFont(new Font(curFontB.getFontName(), curFontB.getStyle(), 14));

                        stepThroughPaneInfo.setBorder(stepThroughInfoLbl);


                        // add it to the main step through panel

                        stepThroughPane.add(stepThroughPaneBtns, BorderLayout.WEST);
                        stepThroughPane.add(stepThroughPaneInfo, BorderLayout.CENTER);


                        // set the title
                        TitledBorder stepThroughMainLbl = new TitledBorder(
        new LineBorder(Color.black),
        "",
        TitledBorder.CENTER,
        TitledBorder.BELOW_TOP);
                        stepThroughMainLbl.setTitleColor(Color.black);

                        Font curFont = stepThroughMainLbl.getTitleFont();
                        stepThroughMainLbl.setTitleFont(new Font(curFont.getFontName(), curFont.getStyle(), 18));

stepThroughPane.setBorder(stepThroughMainLbl);


                        // Left Panel Label
                        leftPanelLabel = new JLabel("", JLabel.CENTER);

                        //
                        JMenuBar menuBar = new JMenuBar();

// Define and add two drop down menu to the menubar
JMenu fileMenu = new JMenu("File");
JMenu editMenu = new JMenu("Edit");
```

```
menuBar.add(fileMenu);
menuBar.add(editMenu);

this.setJMenuBar(menuBar);

// File Menu Options
JMenuItem exitAction = new JMenuItem("Exit");
JMenuItem openAction = new JMenuItem("Open");
JMenuItem highlightOffAction = new JCheckBoxMenuItem("Highlight Other Code");

// open Menu Option
openAction.addActionListener(
                new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                int returnVal = fc.showOpenDialog(temp);

                                if (returnVal == JFileChooser.APPROVE_OPTION) {
                                        temp.dispose();
                                        File inputFile = fc.getSelectedFile();
                                parseFile(inputFile.toString());
                                temp = new CreateGUI();
                                }

                        }
                }
);

// exit Menu Option
exitAction.addActionListener(
                new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                temp.dispose();
                        }
                }
);


// highlight Menu Option
highlightOffAction.addActionListener(
                new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                topPaint.setMenuHighlight();
                        }
                }
);

fileMenu.add(openAction);
fileMenu.add(exitAction);
editMenu.add(highlightOffAction);

// Get the default toolkit
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                int state = 0;
                this.transition_states_ours.clear();
                this.transition_states_ours = transition_states;

                this.transition_errors.clear();
                this.transition_errors = transition_states_error;

                this.programFileNames_ours.clear();
                this.programFileNames_ours = programFileNames;

                this.transition_states_color.clear();
                this.transition_states_color = transition_states;

                this.transition_states_info_ours.clear();
```

```
        this.transition_states_info_ours = transition_states_info;
        transitionInfoPos = new int[trace_Num][2];

        // Get the current screen size
        scrnsize = toolkit.getScreenSize();
        scrnsize.width = scrnsize.width;
        scrnsize.height = scrnsize.height - 20;

        //this.setExtendedState(MAXIMIZED_BOTH);
        this.setPreferredSize(scrnsize);

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout(5,5));
        mainPanel.setBackground(Color.WHITE);

        JPanel topPanel = new JPanel();
        topPanel.setLayout(new GridLayout(1, trace_Num));

        final JPanel leftPanel = new JPanel();
        leftPanel.setLayout(new GridLayout(150, 1));
        leftPanel.setBackground(Color.white);

        JButton threadLegend;
        JLabel legendText;

        // Borders
        Border blackline = BorderFactory.createLineBorder(Color.black);
        Border raisedbevel = BorderFactory.createRaisedBevelBorder();
        Border loweredbevel = BorderFactory.createLoweredBevelBorder();
        Border compoundBorderRaised = BorderFactory.createCompoundBorder(
                            BorderFactory.createMatteBorder(
                10, 5, 10, 5, Color.black), raisedbevel);

        Border compoundBorderLowered = BorderFactory.createCompoundBorder(
                            BorderFactory.createMatteBorder(
                2, 2, 2, 2, Color.black), loweredbevel);

leftPanel.setBorder(compoundBorderRaised);
topPanel.setBorder(blackline);


        // Defining a Color to each thread and creating a seperate highlighter for each color
genColors = new Color[10];
threadHighlighter = new Highlighter.HighlightPainter[10];
this.genThreadColors();
        threadColor = new Color[thread_Num];
    for (int i=0; i < thread_Num; i++){
             // assign colors
             threadColor[i] = genColors[i];
        }

    // Text Area for transitions
    final JTextArea transitions = new JTextArea("");
    transitions.setColumns(100);
    transitions.setRows(20);
    transitions.setBackground(Color.white);
    transitions.setFocusable(false);


    // COME BACK
    // mainPanel.add(scrollingArea, BorderLayout.CENTER);

        Vector<JPanel> schedPaneArray = new Vector();
        Vector<JPanel> schedPaneArrayWrap = new Vector();

        // Left Panel Buttons and Labels
```

```
leftPanelBtns = new ArrayList<List<JButton>>();
leftPanelLbls = new ArrayList<JLabel>();

for(int j = 0; j < trace_Num; j++){
                //schedPaneArray.add(new JPanel());
                schedPaneArrayWrap.add(new JPanel());
                schedPaneArrayWrap.get(j).setBorder(blackline);

                leftPanelBtns.add(new ArrayList<JButton>());
                JLabel tempLbl = new JLabel("Schedule: " + j);
                leftPanelLbls.add(tempLbl);

                //schedPaneArray.get(j).setLayout(new GridBagLayout());
                schedPaneArrayWrap.get(j).setLayout(new BorderLayout(10,10));
                schedPaneArrayWrap.get(j).setBackground(Color.WHITE);

                GridBagConstraints c = new GridBagConstraints();
                c.fill = GridBagConstraints.HORIZONTAL;


 // Adding buttons for each transition (add '+ 1' for the error button)
 JButton[] threadButton = new JButton[transition_Num.get(j)];
// System.out.println(j + " --> Size: " + transition_Num.get(j));
 for(int i = 0; i < transition_Num.get(j); i++){
                threadButton[i] = new JButton("Col: " + j + "x Row: " + i );
                leftPanelBtns.get(j).add(new JButton("Col: " + j + "x Row: " + i ));
 }

 //System.out.println("SIZE: " + leftPanelBtns.get(j).size() );

 for(int i=0; i < transition_states.get(j).size(); i++){
   final String info = transition_states_info.get(j).get(i);

   c.gridx = 0;
   c.gridy = i;

   state = transition_states.get(j).get(i);
   leftPanelBtns.get(j).get(i).setText("" + i );
   leftPanelBtns.get(j).get(i).setName("" + i);
   leftPanelBtns.get(j).get(i).setBackground(threadColor[state]);
   leftPanelBtns.get(j).get(i).setForeground(Color.BLACK);
   curTrace = j;

   // when you press these buttons on the left, transition info will be displayed at the bottom
   leftPanelBtns.get(j).get(i).addActionListener(
                                    new ActionListener(){
                                      public void actionPerformed(ActionEvent e){

                                                    // disable them
                                                    stepForward.setEnabled(true);
                                                    stepBack.setEnabled(true);

                                            //System.out.println("\nSOURCE: " + e.getSource().toString());
                                            Pattern number = Pattern.compile("(.*)JButton\\[([0-9]+)(,)(.*)");
                                            Matcher partOne = number.matcher(e.getSource().toString());
                                            if(partOne.matches()){
                                                          final int rowValue = Integer.parseInt(leftPanelLabel.getName());
                                                          final int colValue = Integer.parseInt(partOne.group(2));

                                            //System.out.println("\nROW: " + rowValue + "\nCOL: " + colValue);

                                             // reset the old button highlighted color
                                                          // check if its last button
                                        int lastBtn = leftPanelBtns.get(curLeft_rowPos).size()-1;
                                        if(lastBtn == curLeft_colPos){
```

```
leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setBackground(Color.GRAY);
                                                                        }else{

leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setBackground(threadColor[transition_states_ours.get(curLeft_rowPos).get(curLeft_colP
os)]);
                                                                        }


leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setForeground(Color.BLACK);

                                                                                        // set colors for both top panel and left panel
                                                                             topPaint.setHighlight(colValue, rowValue, null, 0);

leftPanelBtns.get(rowValue).get(colValue).setForeground(Color.RED);
                                                                             remove(leftPanel, rowValue, colValue,leftPanelBtns, transitions);

                                                                             // assign new current button highlighted
                                                                             curLeft_rowPos = rowValue;
                                                                             curLeft_colPos = colValue;
                                                                             }

                                                                            transitions.setText(info);
                                                                            // Indicates that a same button has been pressed
                                                                            if(buttonID == e.getID() && buttonCMD.equals(e.getActionCommand())){
                                                                                        //tranistionInfoPos++;
                                                                                        buttonID = e.getID();
                                                                                        buttonCMD = e.getActionCommand();

                                                                            }
                                                                            else{ // new button is pressed
                                                                                        tranistionInfoPos = 0;
                                                                                        buttonID = e.getID();
                                                                                        buttonCMD = e.getActionCommand();

                                                                            }

                                                                            // try highlighting text
                                                                            try{

                                                                              // check if the line is in a program
                                                                              while(!programLineFound){
                                                                                 // if program line is not found
                                                                                         //System.out.println(currentLine);
                                                                                 if(!programLineFound){
                                                                                         tranistionInfoPos++;
                                                                                   startIndex = transitions.getLineStartOffset(tranistionInfoPos);
                                                                                               endIndex =
transitions.getLineEndOffset(tranistionInfoPos);
                                                                                               currentLine = transitions.getText().substring(startIndex,
endIndex);

                                                                                 }


                                                                                 for(int i = 0; i < programFileNames_ours.size(); i++){
                                                                                           if(currentLine.contains(programFileNames_ours.get(i)) &&
(!programLineFound) ){
                                                                                                   programLineFound = true;
                                                                                                   currentLineProgram = programFileNames_ours.get(i);

                                                                                                   currentLineProgramIndex = i;
                                                                                                   //System.out.println("Program Name: " +
currentLineProgram + " -- At index: " + currentLineProgramIndex);

                                                                                           }
```

```
                                                                      }



                                                                    }
                                                                    programLineFound = false;
                                                                    transitions.getHighlighter().removeAllHighlights();
                                                                    int stateColor =
transition_states_color.get(curLeft_rowPos).get(curLeft_colPos);
                                                                    transitions.getHighlighter().addHighlight(
transitions.getLineStartOffset(tranistionInfoPos), transitions.getLineEndOffset(tranistionInfoPos), threadHighlighter[stateColor] );

                                                                    /*
                                                                     * This part of the code takes care of highlighting the code in java programs
                                                                     */
                                                                    tabbedPaneCenter.setSelectedIndex(currentLineProgramIndex); //
highlights tab
                                                                    currentLine = currentLine.substring(currentLine.indexOf(": ")+2); // finds the
actual line of code

                                                                    // need to go through the code and determine what the line number is
                                                                    String temp = programInfo[currentLineProgramIndex].getText();
                                                                    startIndex = temp.indexOf(currentLine);


programInfo[currentLineProgramIndex].getHighlighter().removeAllHighlights();
                                                                    programInfo[currentLineProgramIndex].getHighlighter().addHighlight(
startIndex, startIndex+currentLine.length(), threadHighlighter[stateColor] );

                                                                   }
                                                                   catch (BadLocationException bl){
                                                                           System.out.println("Bad Location: " + bl.toString());
                                                                           tranistionInfoPos = 0;
                                                                           nextTransition = true;
                                                                   }
                                                              }
                                                            }
                                                          );

                                  }

                                   // this code takes care of last error button
                                   int lastBtn = leftPanelBtns.get(j).size()-1;
                                   leftPanelBtns.get(j).get(lastBtn).setText("Error Info");
                                   leftPanelBtns.get(j).get(lastBtn).setBackground(Color.GRAY);
                                   leftPanelBtns.get(j).get(lastBtn).setForeground(Color.BLACK);

                                   leftPanelBtns.get(j).get(lastBtn).addActionListener(
                                                   new ActionListener() {
                                                           public void actionPerformed(ActionEvent e) {
                                                                   transitions.setText(transition_errors.get(curTrace));
                                                           }
                                                   }
                                           );

    // defaults
                                   topPanel.add(schedPaneArrayWrap.get(j));

                          }



                                  Dimension topScrollDim = new Dimension();
                                  topScrollDim.height = scrnsize.height / 3;
                                  topScrollDim.width = scrnsize.width;
```

```
                                    topPaint = new PaintTopGUI(trace_Num, thread_Num, transition_Num, transition_states, transition_states_info,
transition_states_error);
                                    topPaint.setPreferredSize(topScrollDim);

                                    topPaint.addMouseListener( new MouseAdapter() {
                                      public void mousePressed( MouseEvent e ) {
                                              // reset the old button highlighted color
                                              leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setForeground(Color.BLACK);

                                              curLeft_rowPos = topPaint.getTraceNumber(e.getX());
                                              curLeft_colPos = topPaint.getTransitionNumber(curLeft_rowPos, e.getY());
                                        //System.out.println("Trace Pos: " + trace_yPos);

                                              //find all the other occurrences of this transition
                                              findAllTransitionInfo(curLeft_rowPos, curLeft_colPos);

                                              // highlight the button pressed / all the occurrences of simalar transitions
                                        topPaint.setHighlight(curLeft_colPos, curLeft_rowPos, transitionInfoPos, transitionInfoPos_size);

                                        if(curLeft_colPos >= leftPanelBtns.get(curLeft_rowPos).size()){
                                              leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setForeground(Color.RED);
                                        }else{
                                              leftPanelBtns.get(curLeft_rowPos).get(curLeft_colPos).setForeground(Color.RED);
                                        }


                                        remove(leftPanel, curLeft_rowPos, curLeft_colPos, leftPanelBtns, transitions);
                                        topPaint.repaint();

                                      }
                                    } );
                                    mainPanel.add(topPaint, BorderLayout.NORTH);

        JScrollPane leftScroll = new JScrollPane(leftPanel);
        Dimension leftScrollDim = new Dimension();
        leftScrollDim.height = scrnsize.height;
        leftScrollDim.width = scrnsize.width / 9;
        leftScroll.setPreferredSize(leftScrollDim);

                                    mainPanel.add(leftScroll, BorderLayout.WEST);
        this.add(mainPanel);

        // text area for each program file (we also upload all information to these areas)
        programInfo = new JTextArea[programFileNames.size()];
        String programName = "";
        String line = "";
        for (int i = 0; i < programInfo.length; i++){
                programInfo[i] = new JTextArea("");
                programName = programFileNames.get(i);
                try{
                  BufferedReader reader = new BufferedReader(new FileReader("src/BankRaceCondition/" + programName));
                  line = reader.readLine();
                  while(line != null){
                          programInfo[i].append(line + "\n");
                          line = reader.readLine();
                  }

                }
                catch(IOException e){
                          System.out.println("IOException: " + e.toString());
                }

                programInfo[i].setFocusable(false);

        }
```

```
// scrolling area for each program file
// we add '+ 1' because we want last tab to hold default transitions info
scrollingArea = new JScrollPane[programFileNames.size() + 1];
for (int i = 0; i < programInfo.length; i++){
        scrollingArea[i] = new JScrollPane(programInfo[i]);
        scrollingArea[i].setBorder(blackline);
}

// transitions info part
scrollingArea[scrollingArea.length - 1] = new JScrollPane(transitions);
        scrollingArea[scrollingArea.length - 1].setBorder(blackline);

// Tabbed Stuff ( adding program file names to tabs)
        tabbedPaneCenter = new JTabbedPane();
        tabbedPaneEast = new JTabbedPane();

for (int i = 0; i < programFileNames.size(); i++){
        tabbedPaneCenter.addTab(programFileNames.get(i), scrollingArea[i]);
}

tabbedPaneEast.addTab("Transition Info", scrollingArea[scrollingArea.length - 1]);
Dimension eastTabDim = new Dimension();
eastTabDim.height = scrnsize.height;
eastTabDim.width = scrnsize.width / 10;
tabbedPaneEast.setPreferredSize(eastTabDim);

JSplitPane centerEastPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, tabbedPaneCenter, tabbedPaneEast);
centerEastPane.setOneTouchExpandable(true);
centerEastPane.setDividerLocation(scrnsize.width / 2);

JSplitPane mainSplit = new JSplitPane(JSplitPane.VERTICAL_SPLIT, centerEastPane, stepThroughPane);
mainSplit.setFocusable(false);
mainSplit.setOneTouchExpandable(true);
mainSplit.setDividerLocation(scrnsize.height / 2);
mainSplit.setEnabled(false);

mainPanel.add(mainSplit, BorderLayout.CENTER);

// Adding Action Listeners to Step Back And Step Forward
// step forward algorithm
                stepForward.addActionListener(
                        new ActionListener() {
                          public void actionPerformed(ActionEvent e) {
                                int sched = Integer.parseInt(stepThroughCurrentSched.getName());
                                int trans =Integer.parseInt(stepThroughCurrentTransition.getName());


                                if(trans < (leftPanelBtns.get(sched).size() - 1)){
                                        leftPanelBtns.get(sched).get(trans).doClick();
                                        if(nextTransition){
                                                trans++;
                                                leftPanelBtns.get(sched).get(trans).doClick();
                                                nextTransition = false;
                                        }
                                }

                          }
                        }
                );


                // step back algorithm
                stepBack.addActionListener(
                        new ActionListener() {
                          public void actionPerformed(ActionEvent e) {
```

```
                                        int sched = Integer.parseInt(stepThroughCurrentSched.getName());
                                        int trans =Integer.parseInt(stepThroughCurrentTransition.getName());


                                        if(trans < (leftPanelBtns.get(sched).size() - 1) && trans >= 0){

                                                tranistionInfoPos -= 2;

                                                leftPanelBtns.get(sched).get(trans).doClick();
                                                System.out.println("transition Info Pos: " + tranistionInfoPos);
                                                System.out.println("OLD Info Pos: " + oldTransInfoPos);
                                                if(oldTransInfoPos == tranistionInfoPos && oldTransInfoPos != -1){

                                                        if(trans != 0){
                                                                trans--;
                                                        }
                                                        tranistionInfoPos = 0;
                                                        leftPanelBtns.get(sched).get(trans).doClick();
                                                }

                                                oldTransInfoPos = tranistionInfoPos;

                                        }
                                    }
                                  }
                );


this.pack();
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
        }

        /*
         * This is the dynamic part of the GUI
         * When we load a new File, we call this method
         */

        /*
         * - clears the left panel
         * - adds current set of buttons to left panel
         */
        public void remove(JPanel leftPanel, int j, int k, List<List<JButton>> leftPanelBtns,  JTextArea scrollText){
                int size = leftPanelBtns.get(j).size();
                topPaint.repaint();

                leftPanel.removeAll();
                scrollText.setText("");


          stepThroughCurrentSched.setText("Schedule: " + (j + 1));
          stepThroughCurrentSched.setName("" + j);

          //System.out.println("Size: " + size + " --> J: " + k);
          if((size - 1) == k){
                stepThroughCurrentTransition.setText("Transition: Error Info");
                stepThroughCurrentTransition.setName("-1");
          }else{
                stepThroughCurrentTransition.setText("Transition: " + k);
                stepThroughCurrentTransition.setName("" + k);
          }

                leftPanel.setLayout(new GridLayout(size + 1, 1));
                leftPanel.setBackground(Color.white);
                //leftPanel.add(leftPanelLbls.get(j));
```

```
		leftPanelLabel.setText("Schedule: " + (j+1));
		leftPanelLabel.setName(""+ j);

		leftPanel.add(leftPanelLabel);


		for (int l = 0; l < size; l++){
			leftPanel.add(leftPanelBtns.get(j).get(l));
		}

		//this.setPreferredSize(new Dimension(this.MAXIMIZED_HORIZ, this.MAXIMIZED_VERT));
		//this.setExtendedState(MAXIMIZED_BOTH);
		this.setPreferredSize(scrnsize);
		leftPanel.repaint();

		this.pack();
	}

	/*
	 * This method generates colors for up to 10 different threads (RGB values)
	 * - We are not using red because it is the high lighter color
	 */
	public void genThreadColors(){

		genColors[0] = new Color(58,98,132); // blue
		threadHighlighter[0] = new DefaultHighlighter.DefaultHighlightPainter( genColors[0] );

		genColors[1] = new Color(51,157,70); // green
		threadHighlighter[1] = new DefaultHighlighter.DefaultHighlightPainter( genColors[1] );

		genColors[2] = new Color(255,127,50); // orange
		threadHighlighter[2] = new DefaultHighlighter.DefaultHighlightPainter( genColors[2] );

		genColors[3] = new Color(145,105,171); // purple
		threadHighlighter[3] = new DefaultHighlighter.DefaultHighlightPainter( genColors[3] );

		genColors[4] = new Color(136,89,79); // brown
		threadHighlighter[4] = new DefaultHighlighter.DefaultHighlightPainter( genColors[4] );

		genColors[5] = new Color(226,123,182); // pink
		threadHighlighter[5] = new DefaultHighlighter.DefaultHighlightPainter( genColors[5] );

		genColors[6] = new Color(88,88,88); // grey
		threadHighlighter[6] = new DefaultHighlighter.DefaultHighlightPainter( genColors[6] );

		genColors[7] = new Color(84,143,174); // teal
		threadHighlighter[7] = new DefaultHighlighter.DefaultHighlightPainter( genColors[7] );

		genColors[8] = new Color(191,186,57); // gold
		threadHighlighter[8] = new DefaultHighlighter.DefaultHighlightPainter( genColors[8] );

		genColors[9] = new Color(1,71,189); // royal blue
		threadHighlighter[9] = new DefaultHighlighter.DefaultHighlightPainter( genColors[9] );
	}

	/*
	 * this Method, given the position the transition info, finds out the positions
	 * of all other similar transition info's
	 */
	public void findAllTransitionInfo(int row, int col){
		transitionInfoPos_size = 0;
		String matchInfo = transition_states_info_ours.get(row).get(col);

		for(int i = 0; i < transition_states_info_ours.size(); i++){
			for(int j = 0; j < transition_states_info_ours.get(i).size(); j++){
```

```
                    //
                    if( matchInfo.equals(transition_states_info_ours.get(i).get(j))){
                        //System.out.println("Match Found!");
                        transitionInfoPos[transitionInfoPos_size][0] = i;
                        transitionInfoPos[transitionInfoPos_size][1] = j;

                        transitionInfoPos_size++;
                    }
                }

        }

}


/*
 * Parses the file given to retrieve certain information
 */
public static void parseFile(String fileName){

        programFileNames.clear();
        transition_states_error.clear();
        transition_states.clear();
        transition_Num.clear();
        transition_states_info.clear();

        // Pattern for matching Java program File names
        Pattern filePattern = Pattern.compile("\\s(.+).java.*");

        // Pattern for matching the final error for the schedule
        Pattern endError = Pattern.compile("error #([0-9]+): (.*)");

        // Pattern for matching thread number
        Pattern threadPattern = Pattern.compile(".* thread: ([0-9]+)");
        int fileThreadNum=0;

        // Pattern for matching transition number
        Pattern transitionPattern = Pattern.compile("(.*)transition #([0-9]+)(.*)");
        int fileTransitionNum=0;

        // Pattern for matching trace number
        Pattern tracePattern = Pattern.compile("(.*)trace #([0-9]+)");
        int fileTraceNum=0;

        // Keep track on when to start ALL trace gathering
        boolean getInfoAll = false;

        // These variables are used to gather specific transition info JPF outputs
        String transition_info = "";
        boolean start_gathering = false;
        Pattern stopPattern = Pattern.compile("([=]+).*"); // to stop taking down transition info

        try {

            FileReader input = new FileReader(fileName);
            BufferedReader buffRead = new BufferedReader(input);

            String line ="";
            String fileNameFound = "";
            line = buffRead.readLine();
            boolean addFile = true;

            while(line != null){

                // matcher for file name
```

```
Matcher fileNameMatch = filePattern.matcher(line);
if(fileNameMatch.matches()){
        addFile = true;
        fileNameFound = fileNameMatch.group(1);
        fileNameFound = fileNameFound.trim();

         if(!fileNameFound.matches("(.*)\\s(.*)")){

         //System.out.println("Found File: " + fileNameFound);

         if(programFileNames.size() == 0){
                programFileNames.add(fileNameFound + ".java");
         }
         else{
                for(int i = 0; i < programFileNames.size(); i++){
                        if(programFileNames.get(i).equals(fileNameFound + ".java")){
                                addFile = false;
                        }
                }

                // if file name is not found in the current list, add it to list
                if(addFile){
                        programFileNames.add(fileNameFound + ".java");
                }
         }
        }
}


// matcher for final error
Matcher errorMatch = endError.matcher(line);
if(errorMatch.matches()){
        transition_states_error.add(errorMatch.group(2));
}

// matcher for trace
Matcher traceMatch = tracePattern.matcher(line);
// parse file to see the trace number
if(traceMatch.matches()){
  fileTraceNum = Integer.parseInt(traceMatch.group(2));
  trace_Num = fileTraceNum - 1;
  transition_states.add(new ArrayList<Integer>());
  transition_states_info.add(new ArrayList<String>());
  transition_Num.add(0);
  //System.out.println("TraceNum: " + trace_Num);
  getInfoAll = true;
}

if(getInfoAll){

        // matcher for thread
        Matcher threadMatch = threadPattern.matcher(line);

        // parse file to see the schedule of the thread
        if(threadMatch.matches()){
          fileThreadNum = Integer.parseInt(threadMatch.group(1));
          transition_states.get(trace_Num).add(fileThreadNum);

          // keeps track of the overall number of threads
          if(fileThreadNum > thread_Num){
                  thread_Num = fileThreadNum;
          }
        }

        // matcher for transition
        Matcher transitionMatch = transitionPattern.matcher(line);
```

```
                                        // parse file to see the transitions of the threads
                                        if(transitionMatch.matches()){
                                          fileTransitionNum = Integer.parseInt(transitionMatch.group(2));
                                          //transition_Num[trace_Num] = fileTransitionNum;

                                          if(fileTransitionNum > transition_Num.get(trace_Num)){
                                             transition_Num.set(trace_Num, fileTransitionNum);
                                          }

                                          // gather information on the transition
                                          if(fileTransitionNum == 0){
                                                        transition_info = "";
                                                        start_gathering = true;
                                          }else{
                                                       transition_states_info.get(trace_Num).add(transition_info);
                                                       transition_info = "";
                                          }
                                        }

                                        // to check when to stop taking down info
                                        Matcher stopMatch = stopPattern.matcher(line);
                                        if(stopMatch.matches()){

                                                      // Saves information to the transition_state_info variable
                                                      if(trace_Num == 0){
                                                                    if((!traceMatch.matches()) && start_gathering){
                                                                                  transition_states_info.get(trace_Num).add(transition_info);
                                                                    }

                                                      }
                                                      else{
                                                                    if((traceMatch.matches())){
                                                                       transition_states_info.get(trace_Num-1).add(transition_info);
                                                                    }

                                                                    // this one is to add info for last transition of last trace
                                                                    else if((!traceMatch.matches()) && start_gathering){
                                                                                  transition_states_info.get(trace_Num).add(transition_info);
                                                                    }
                                                      }

                                                      start_gathering = false;
                                        }

                                        // transition info gathering
                                        if((start_gathering) && (!traceMatch.matches()) && (!threadMatch.matches()) &&
(!transitionMatch.matches())){
                                                      transition_info += "\n" + line;
                                        }

                          }

                          line = buffRead.readLine();
               }


               input.close();

            }
            catch(IOException e){

            }

            for(int i = 0; i < transition_states.size(); i++){
                          transition_states.get(i).add(0);
                          if(transition_states_error.size() > 0){
```

```
                                                transition_states_info.get(i).add(transition_states_error.get(i));
                                            }
                                            else{
                                                        transition_states_error.add("NO ERROR FOUND IN OUTPUT");
                                                        transition_states_info.get(i).add("NO ERROR FOUND IN OUTPUT");
                                            }
                                }

                                trace_Num++;
                                thread_Num++;
                                for(int i = 0; i < transition_Num.size(); i++){
                                            transition_Num.set(i, transition_Num.get(i) + 2);
                                }

                    }

                    /*
                     * Main Function
                     */
                    public static void main(String[] args){
                                //Create a file chooser
        fc = new JFileChooser();

       int returnVal = fc.showOpenDialog(temp);
                     File inputFile = fc.getSelectedFile();
                     parseFile(inputFile.toString());


                     /*
                       for(int i = 0; i < transition_states.size(); i++){
                                 System.out.println("------- Schedule: " + i + " ------- " + transition_states.get(i).size());
                                 for(int j = 0; j < transition_states.get(i).size(); j++){
                                             //System.out.println("Transition: " + transition_states.get(i).get(j));
                                             //System.out.println("Info: " + transition_states_info.get(i).get(j));
                                 }

                                 System.out.println("----------------------------****-------------------------------");
                       }
                       */

                                temp = new CreateGUI();

                    }
}
```

12. Appendix F – Source Code for Completed Visualization Tool

TopPanel.java

```
package ThreadGUI;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.LayoutManager;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.Border;

public class PaintTopGUI extends JPanel{

		// to generate random numbers
		private Dimension scrnsize;
		private int sq_height;
		private int sq_width;
    private int max_Transitions;
		private final int y_offset = 0;
		private List<Integer> transition_Num;
		private List<List<Integer>> transition_states;
		private List<List<String>> transition_states_info;
		private int trace_Num;
		private int thread_Num;
		private Dimension topDim;
		private Graphics gCopy;

		// highlight vlaues
		private boolean highlight = false;
		private int highlight_x = -1;
		private int highlight_y = -1;

		int[][] otherHighlight_ours;

		// colors
		private Color[] genColors;
		private Color[] threadColor ;
```

```
                  // When a certain column is highlighted, we need to darken the colour with this constant
                  private int colorInc = 100;
                  private boolean MenuHighlight = false;



                  public PaintTopGUI(int trace_Num, int thread_Num, List<Integer> transition_Num, List<List<Integer>> transition_states,
List<List<String>> transition_states_info, List<String> transition_states_error){

                              this.transition_Num = transition_Num;
                              this.transition_states = transition_states;
                              this.transition_states_info = transition_states_info;
                              this.trace_Num = trace_Num;
                              this.thread_Num = thread_Num;

                              // get max transitions
                              max_Transitions = 0;
                              for(int i=0; i < transition_Num.size(); i++){
            if(transition_Num.get(i) > max_Transitions){
                  max_Transitions = transition_Num.get(i);
            }
                        }

                              // Get the current screen size
                              Toolkit toolkit = Toolkit.getDefaultToolkit();
                              scrnsize = toolkit.getScreenSize();
                              scrnsize.width = scrnsize.width;
                              scrnsize.height = scrnsize.height - 20;

        JPanel topPanel = new JPanel();
        topPanel.setLayout(null);

        topPanel.setBackground(Color.WHITE);
        topPanel.setVisible(true);

        topDim = new Dimension();
        topDim.height = scrnsize.height / 3;
        topDim.width = scrnsize.width;
        topPanel.setBackground(Color.white);

                              // initializing the square height
                              sq_width = (int)(topDim.width / trace_Num);
                              sq_height = (int)(topDim.height / max_Transitions);

        this.add(topPanel);
        this.setSize(topDim);
        this.setVisible(true);

                  }


    public void paint(Graphics g) {
                   gCopy = g;
                               g.setColor(Color.WHITE);
                               g.fillRect(0, 0, scrnsize.width, scrnsize.height);
                               int y = 0;
                               int x = 0;
                               int threadNum = 0;


                                          // Defining a Color to each thread
                  genColors = new Color[10];
                  genThreadColors();
                                          Color[] threadColor = new Color[thread_Num];
                                 for (int i=0; i < thread_Num; i++){
                                                      // assign colors
                        threadColor[i] = genColors[i];
            }


for(int j = 0; j < trace_Num; j++){
            x = j * sq_width;
            y = y_offset;

            // coloring the panel
            for(int i=0; i < transition_states.get(j).size(); i++){
                        y = y_offset + i*sq_height;

                        threadNum = transition_states.get(j).get(i);
               g.setColor(threadColor[threadNum]);
               g.fillRect(x, y, sq_width, sq_height);

               // error button
               if(i == transition_states.get(j).size() - 1){
                                    g.setColor(Color.GRAY);
                                    g.fillRect(x, y, sq_width, sq_height);
               }

               // specific highlight
               if(highlight){

                        if(j == highlight_x){
                          Color tempLight = threadColor[threadNum].brighter();

                          g.setColor(tempLight);
                            g.fillRect(x, y, sq_width, sq_height);
                                    // error button
                            if(i == transition_states.get(j).size() - 1){
                                                                        g.setColor(Color.GRAY.brighter());
                                                                        g.fillRect(x, y, sq_width, sq_height);
                                            }
                        }
                        else{
                                    Color tempDark = threadColor[threadNum].darker();

                                    g.setColor(tempDark);
                                      g.fillRect(x, y, sq_width, sq_height);
                                      // error button
                                      if(i == transition_states.get(j).size() - 1){
                                                                        g.setColor(Color.GRAY.darker());
                                                                        g.fillRect(x, y, sq_width, sq_height);
                                            }
                        }

                        // highlight similar transitions
                        if(MenuHighlight){
                                    for(int z = 0; z < otherHighlight_ours.length; z++){
                                                if(j == otherHighlight_ours[z][0]  && i == otherHighlight_ours[z][1]){

                                                            g.setColor(Color.yellow);
                                                            // g.setColor(threadColor[threadNum]);
                                                            g.fillOval(x, y, sq_width, sq_height);
                                                            //g.drawRect(x, y, sq_width, sq_height);
                                                }

                                    }
                        }


                        if(j == highlight_x  && i == highlight_y){
                                    g.setColor(Color.RED);
                                      g.fillRect(x, y, sq_width, sq_height);
                                    // g.setColor(threadColor[threadNum]);
                                     //g.fillOval(x, y, sq_width, sq_height);
                                     //g.drawRect(x, y, sq_width, sq_height);
                                  }
                          }

                        }

                        // drawing the error button
                        y = y_offset + transition_states.get(j).size()* sq_height;


                        // drawing the borders
                        if(j == highlight_x || (j-1) == highlight_x && highlight){
                                g.setColor(Color.RED);
                                g.drawLine(x, 0, x, max_Transitions*sq_height);
                        }else{
                          g.setColor(Color.WHITE);
                          g.drawLine(x, 0, x, max_Transitions*sq_height);
                        }

                        //g.setColor(Color.RED);
                        //g.drawLine(x, 0, x, (transition_states.get(j).size() + 1)*sq_height);

                        //g.setColor(Color.RED);
                        //g.drawLine((j+1) * sq_width, 0, (j+1) * sq_width, transition_states.get(j).size()*sq_height);

                        //g.setColor(Color.ORANGE);
                        //g.drawLine(j * sq_width, transition_states.get(j).size()*sq_height, (j+1) * sq_width,
transition_states.get(j).size()*sq_height);

                        //g.setColor(Color.DARK_GRAY);
                        //g.drawLine(x, y, x+sq_width, y+sq_height);
                        //g.drawLine(x, y+sq_height, x+sq_width, y);


                 }
  }

        /*
         * used with mouse listener to get the column (x)
         */
        public int getTraceNumber(int x){

                for (int i = 1; i < trace_Num ; i++){

                        if((x > (i-1)*sq_width) && (x < i*sq_width)){
                                return (i-1);
                        }
                }

                return trace_Num - 1;
        }

        /*
         * used with mouse listener to get the row (y)
         */
        public int getTransitionNumber(int trace, int y){

                for (int i = 1; i < transition_states.get(trace).size(); i++){

                        if((y > (i-1)*sq_height) && (y < i*sq_height)){
                                //System.out.println("ROW: " + (i - 1));
                                return (i-1);
                        }
```

```
			}

			return transition_states.get(trace).size() - 1;
		}

		/*
		 * This method is used to highlight the current transition
		 * selected.
		 */
		public void setHighlight(int row, int col, int[][] otherHighlight, int size){
			highlight = true;
			highlight_x = col;
			highlight_y = row;

			// fill in row and col values for the other cubes that need to be selected
			if(otherHighlight != null){
			   otherHighlight_ours = new int[size][2];
			   for(int i = 0; i < size; i++){
					otherHighlight_ours[i][0] = otherHighlight[i][0];
					otherHighlight_ours[i][1] = otherHighlight[i][1];
			   }
			}

		}



		/*
		 * Setter Method for Menu Highlight Option
		 */
		int checkTimes = 0;
		public void setMenuHighlight(){
			checkTimes++;
			if(checkTimes % 2 == 0){
			   this.MenuHighlight = false;
			}
			else{
				this.MenuHighlight = true;
			}
		}

		/*
		 * This method generates colors for up to 10 different threads (RGB values)
		 * - We are not using red because it is the high lighter color
		 */
		public void genThreadColors(){
			genColors[0] = new Color(58,98,132); // blue
			genColors[1] = new Color(51,157,70); // green
			genColors[2] = new Color(255,127,50); // orange
			genColors[3] = new Color(145,105,171); // purple
			genColors[4] = new Color(136,89,79); // brown
			genColors[5] = new Color(226,123,182); // pink
			genColors[6] = new Color(88,88,88); // grey
			genColors[7] = new Color(84,143,174); // teal
			genColors[8] = new Color(191,186,57); // gold
			genColors[9] = new Color(1,71,189); // royalblue
		}
	}
```