Model Checking in Software and GUI Applications using Java Path Finder

By: Raghavendra Sirigeri - rs3603
Ishaan sayal - is2439

Overview:

1. Model checking algorithms to generic suite of programs in Java
2. Model checking algorithms for GUI based applications to detect for inconsistencies like deadlocks, races and other system specific errors
3. Tools Used - Java Path Finder and FindBugs
4. JPF-AWT extension for GUI Applications

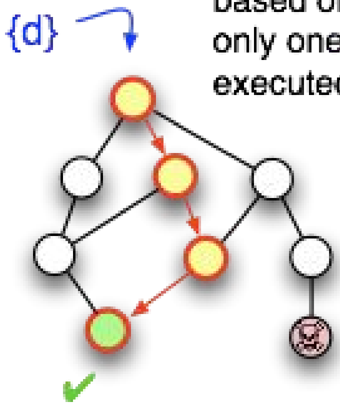
JPF-Core

1. The JPF core is a Virtual Machine (VM) for Java™ bytecode
2. To find defects in these programs, also need to give it the properties to check for as input
3. JPF gets back to you with a report that says if the properties hold
4. Which verification artifacts have been created by JPF for further analysis (like test cases).

Model Checking vs Testing

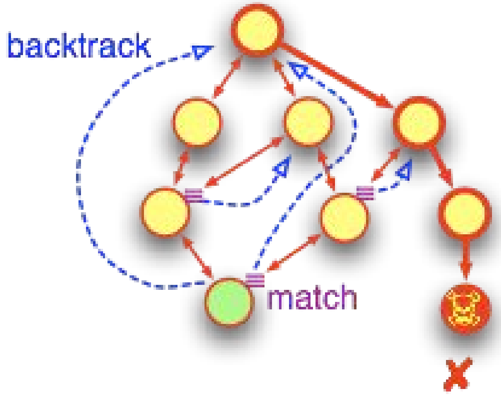
testing:

based on input set {d}
only one path
executed at a time



model checking:

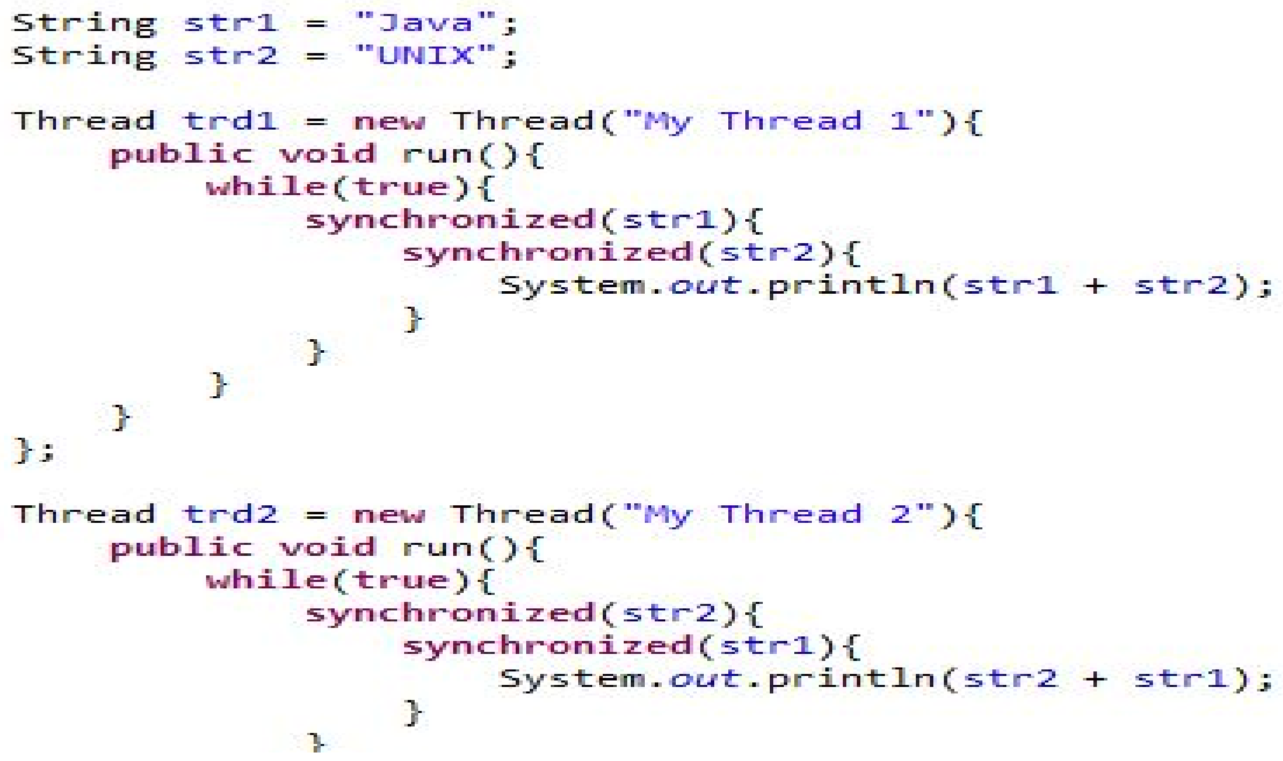
all program state are explored
until none left or defect found



Codes Analysed by JPF

1. Rand.java : A program to divide two random numbers.
2. CheckRace.java : A program that runs two threads and accesses the same variables.
3. CheckNPE.java : A program that does not initialise objects properly.
4. Deadlock.java : A program that concatenates two strings

Deadlock Code



```
String str1 = "Java";
String str2 = "UNIX";

Thread trd1 = new Thread("My Thread 1"){
    public void run(){
        while(true){
            synchronized(str1){
                synchronized(str2){
                    System.out.println(str1 + str2);
                }
            }
        }
    }
};

Thread trd2 = new Thread("My Thread 2"){
    public void run(){
        while(true){
            synchronized(str2){
                synchronized(str1){
                    System.out.println(str2 + str1);
                }
            }
```

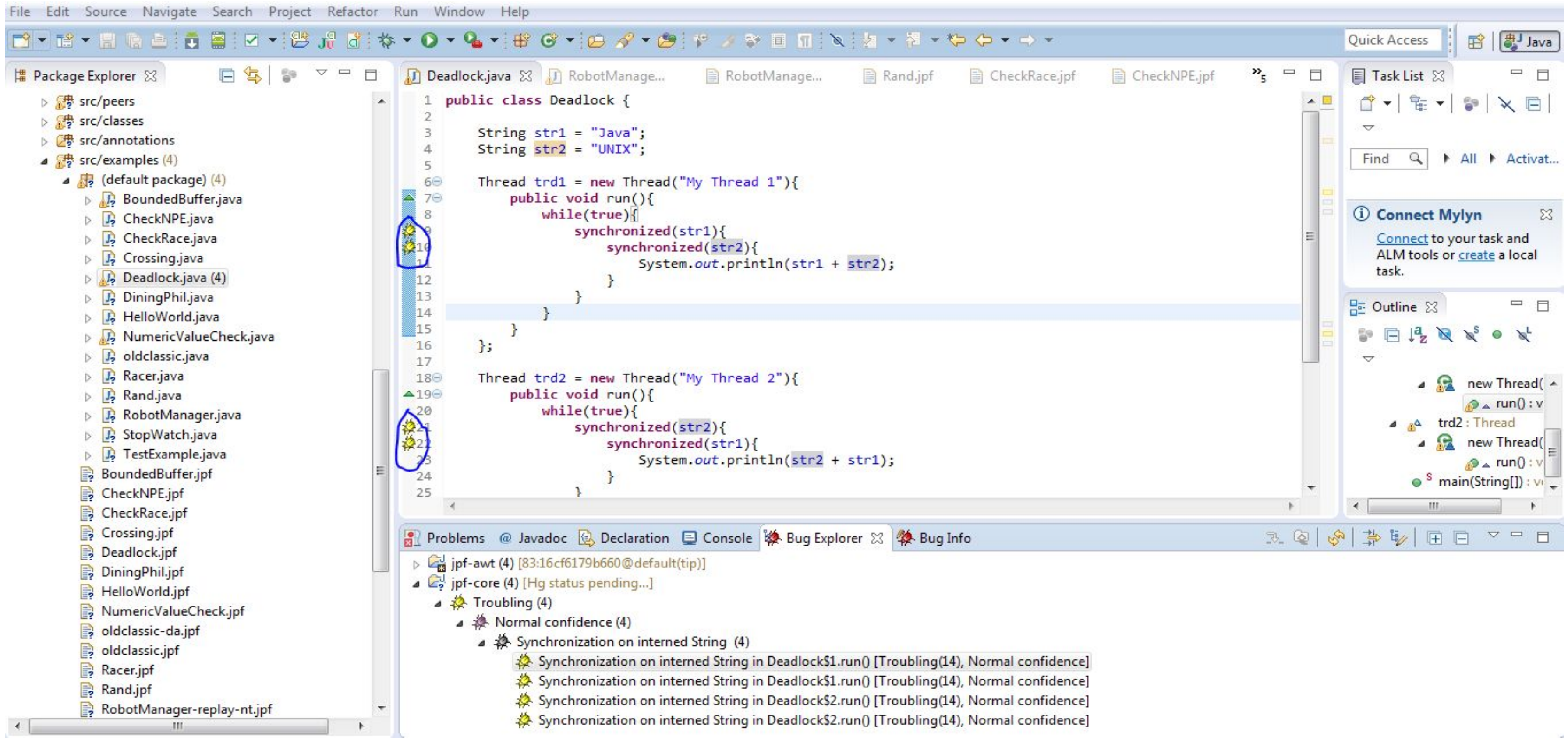
Results of JPF

```
==================================================== results
error #1: gov.nasa.jpf.vm.NotDeadlockedProperty "deadlock encountered:    thread
Deadlock$1:{id:1,n..."

==================================================== statistics
elapsed time:       00:00:01
states:             new=21,visited=3,backtracked=3,end=1
search:             maxDepth=21,constraints=0
choice generators:  thread=21 (signal=0,lock=11,sharedRef=7,threadApi=2,
reschedule=1), data=0
heap:               new=393,released=29,maxLive=366,gcCycles=24
instructions:       3726
max memory:         61MB
loaded code:        classes=64,methods=1474

==================================================== search finished: 22/12/15 12:
02 AM
```

FindBugs Tool

1. FindBugs is a program which uses static analysis to look for bugs in Java code.
2. **Static program analysis** is the analysis of computer software that is performed without actually executing programs
3. Findbugs tool will only help us in detecting a deadlock in the deadlock code. Findbugs does not support exceptions, arithmetic exceptions, etc.

FindBugs



src/peers
src/classes
src/annotations
src/examples (4)
(default package) (4)
BoundedBuffer.java
CheckNPE.java
CheckRace.java
Crossing.java
Deadlock.java (4)
DiningPhil.java
HelloWorld.java
NumericValueCheck.java
oldclassic.java
Racer.java
Rand.java
RobotManager.java
StopWatch.java
TestExample.java
BoundedBuffer.jpf
CheckNPE.jpf
CheckRace.jpf
Crossing.jpf
Deadlock.jpf
DiningPhil.jpf
HelloWorld.jpf
NumericValueCheck.jpf
oldclassic-da.jpf
oldclassic.jpf
Racer.jpf
Rand.jpf
RobotManager-replay-nt.jpf

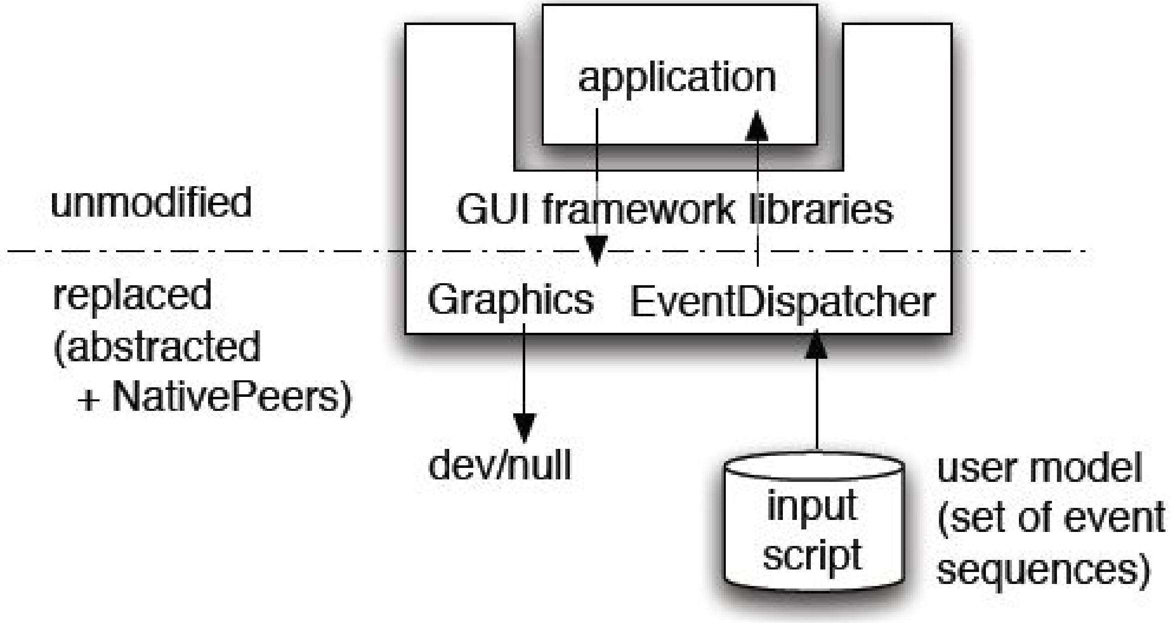
```java
public class Deadlock {

    String str1 = "Java";
    String str2 = "UNIX";

    Thread trd1 = new Thread("My Thread 1"){
        public void run(){
            while(true){
                synchronized(str1){
                    synchronized(str2){
                        System.out.println(str1 + str2);
                    }
                }
            }
        }
    };

    Thread trd2 = new Thread("My Thread 2"){
        public void run(){
            while(true){
                synchronized(str2){
                    synchronized(str1){
                        System.out.println(str2 + str1);
                    }
                }
```

```
jpf-awt (4) [83:16cf6179b660@default(tip)]
jpf-core (4) [Hg status pending...]
  Troubling (4)
    Normal confidence (4)
      Synchronization on interned String  (4)
        Synchronization on interned String in Deadlock$1.run() [Troubling(14), Normal confidence]
        Synchronization on interned String in Deadlock$1.run() [Troubling(14), Normal confidence]
        Synchronization on interned String in Deadlock$2.run() [Troubling(14), Normal confidence]
        Synchronization on interned String in Deadlock$2.run() [Troubling(14), Normal confidence]
```

JPF-AWT

1. JPF-AWT targets GUI applications that use the standard Java AWT and Swing framework libraries
2. Main objective for the JPF-AWT design is to check the application behavior against a potentially large set of different input sequences
3. Achieved by replacing only the low-level, platform-specific parts of the framework libraries that handle rendering and input acquisition, leaving the application itself completely unmodified.

JPF-AWT


application
unmodified
GUI framework libraries
replaced
(abstracted
+ NativePeers)
Graphics
EventDispatcher
dev/null
input
script
user model
(set of event
sequences)

RobotManager

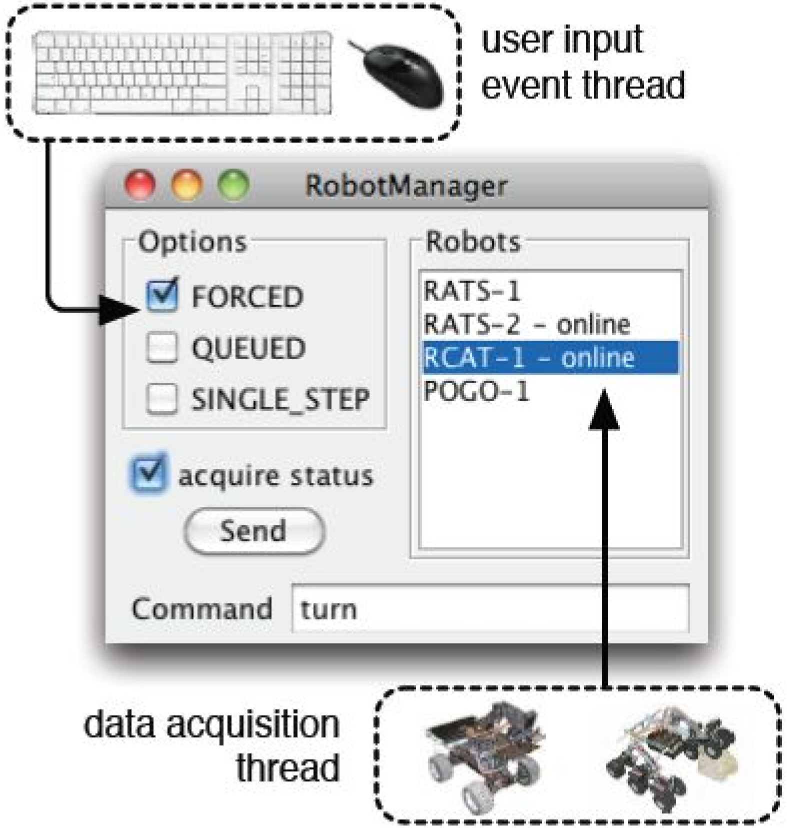
The nominal control procedure consists of the following steps:

1) entering a command

2) selecting command options (if any)

3) selecting a robot from the list

4) sending the command to the robot by means of clicking the Send button.



Options
- [✓] FORCED
- [ ] QUEUED
- [ ] SINGLE_STEP

[✓] acquire status

Robots
- RATS-1
- RATS-2 - online
- RCAT-1 - online
- POGO-1

Command: turn

Exception

The exception is caused by a race condition between the data acquisition thread and the EventDispatchThread, the first one setting the selected robot offline right in the middle of the send button click processing.

Exception

```
================================================= error #1
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
java.lang.NullPointerException: Calling
’processSequence(Ljava/lang/String;)Ljava/lang/String;’
on null object
at RobotManager.sendSequence(RobotManager.java:265)
...
================================================= statistics
elapsed time: 0:00:03
states: new=1320, visited=207, backtracked=1490, end=0
search: maxDepth=68, constraints hit=0
choice gens: thread=41 (signal=0, lock=10, shared ref=28),
data=1444
heap: new=21524, released=11537, max live=3333,
gc-cycles=1480
instructions: 1677327
max memory: 81MB
loaded code: classes=404, methods=4959
```

Application Specific Error

Now suppose we are given a specification that robot POGO cannot force queued sequences. This is an example of an application specific property.

Application Specific Error

```
====================================== error
1
gov.nasa.jpf.vm.NoUncaughtExceptionsProperty
java.lang.AssertionError: POGOs cannot force queued
sequences
   at POGO.processSequence(RobotManager.java:93)
   at RobotManager.sendSequence(RobotManager.java:269)
   at RobotManagerView.sendSequence(RobotManager.java:569)
   at RobotManagerView$3.actionPerformed(RobotManager.java:
341)
```

Robot Manager Modification

1. Starting another thread to keep robot ‘RATS-2’ always online ,SetOnlineRobo
2. This creates a potential race condition with DataAcquisition Thread
3. Potential race condition
4. JPF max memory reached, not able to detect

Modified Application

```
======================================================= system under test
RobotManager.main()

======================================================= search started:
21/12/15 10:10 PM
[SEVERE] JPF out of memory

======================================================= search constraint
JPF out of memory
```

Thank You