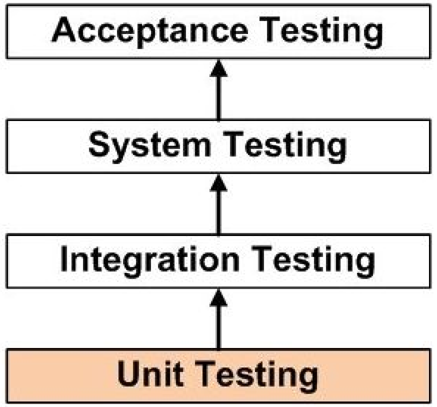
Air Traffic Controller Formal Verification

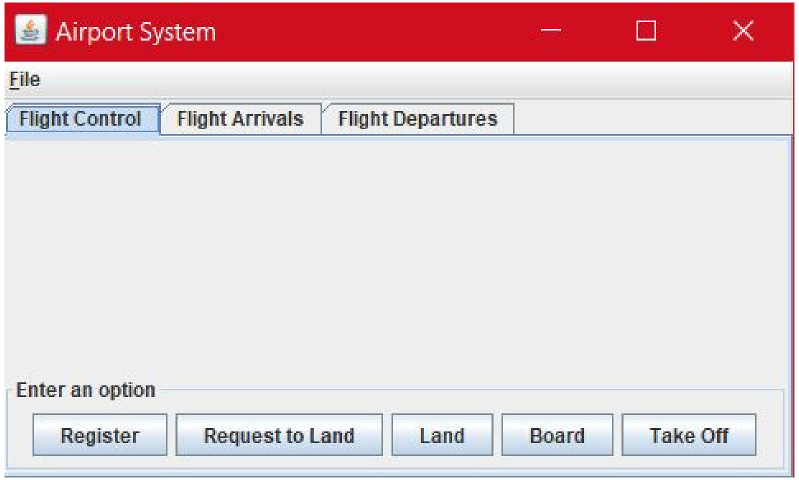
Tahir Casey
Yuval Schaal

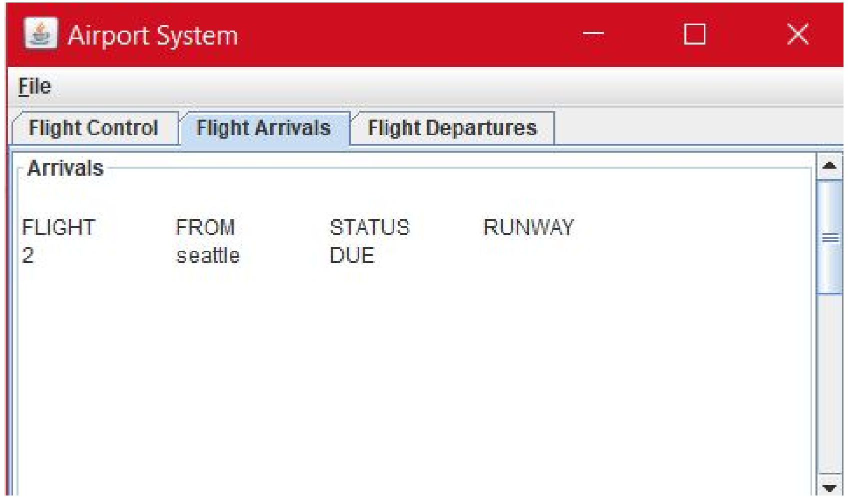
Background

- Air traffic control because is very important in everyday air flights, and is consequently a suitable example of the type of critical software that should be subjected to an exhaustive method like model checking and similar formal approaches.
- The goal was to perform a case study of verification tools using a test program and compare their performance to the traditional static analysis and unit testing approaches which are favored in software engineering, as well as to find the limitations of the verification tools used.

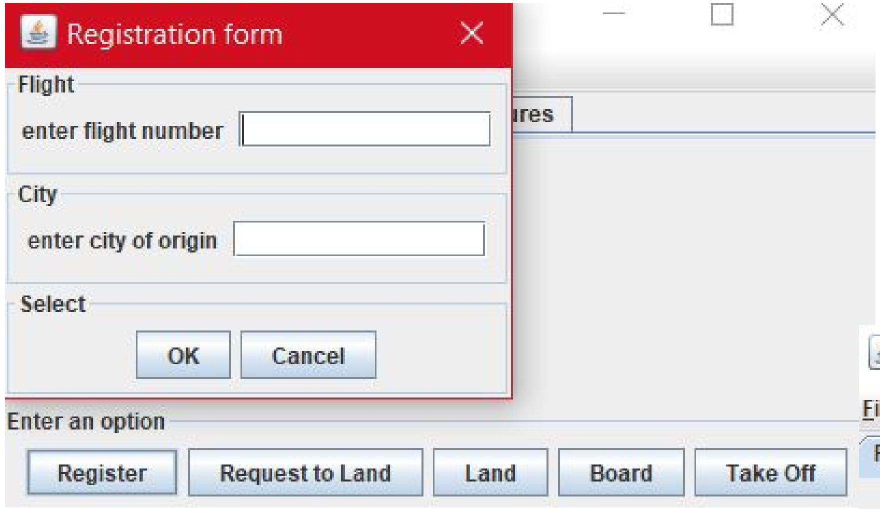
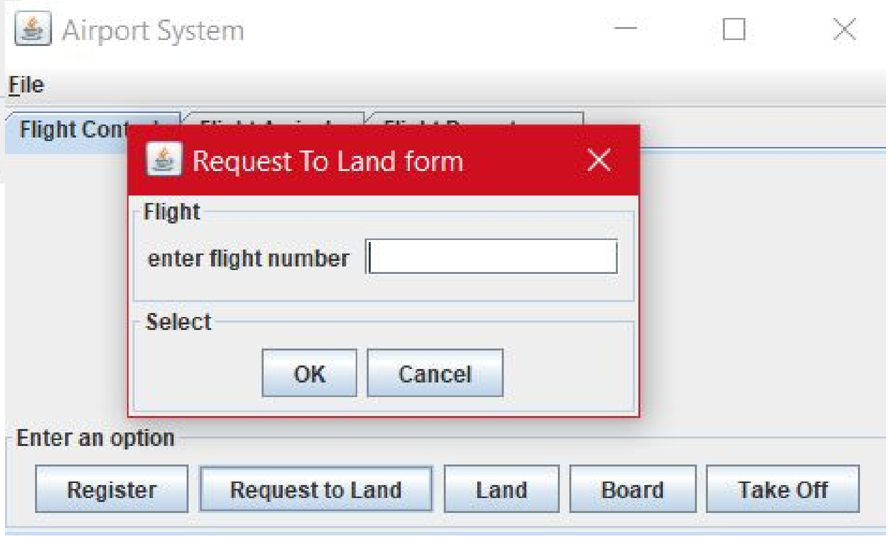


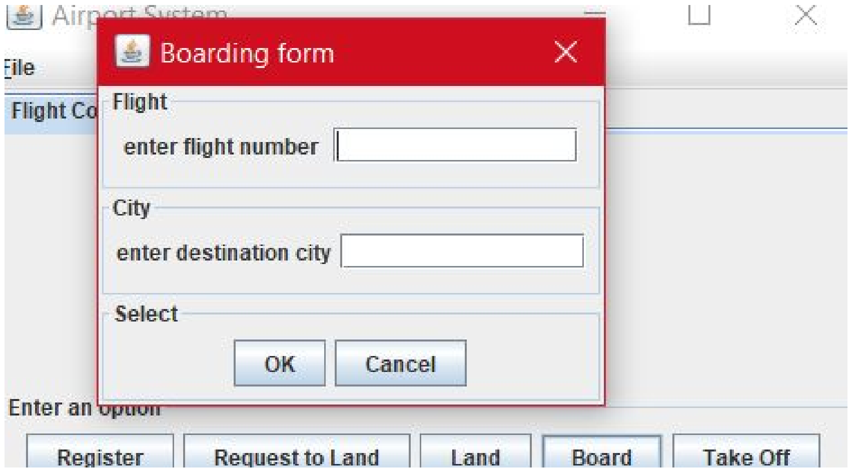
The Java Application - Description





```
FLIGHT        FROM          STATUS        RUNWAY
2             seattle       DUE
```


Registration form
Flight
enter flight number
City
enter city of origin
Select
OK
Cancel
ures
Enter an option
Register
Request to Land
Land
Board
Take Off

Airport System
File
Flight Cont
Request To Land form
Flight
enter flight number
Select
OK
Cancel
Enter an option
Register
Request to Land
Land
Board
Take Off


Airport System

File

Flight Co

Boarding form

Flight

enter flight number

City

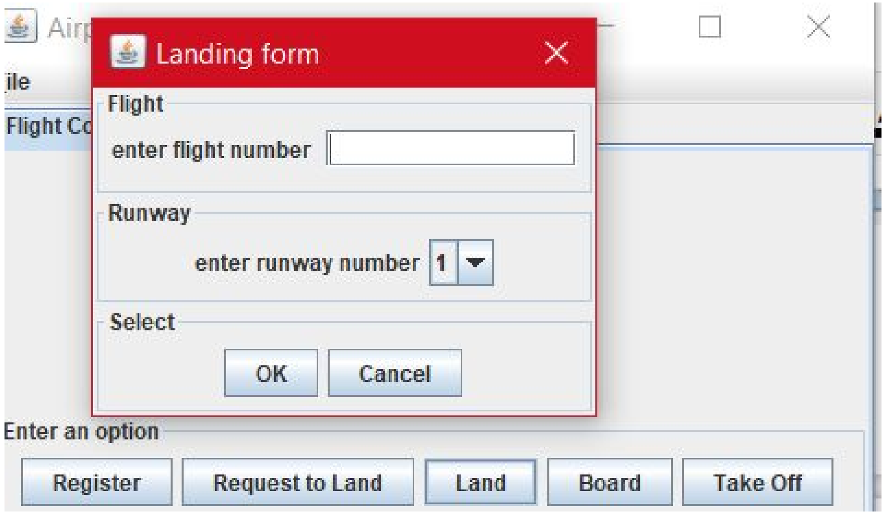
enter destination city

Select

OK Cancel

Enter an option

Register | Request to Land | Land | Board | Take Off


Airp

ile

Flight Co

Landing form

Flight

enter flight number

Runway

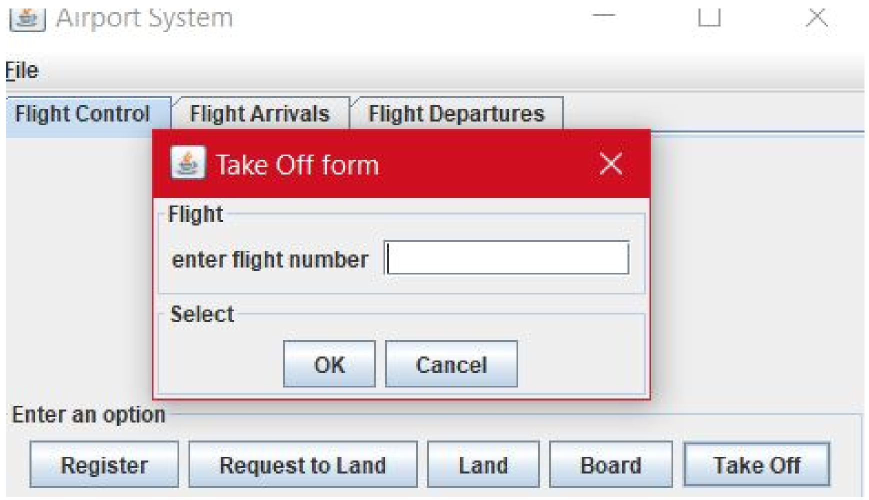
enter runway number 1

Select

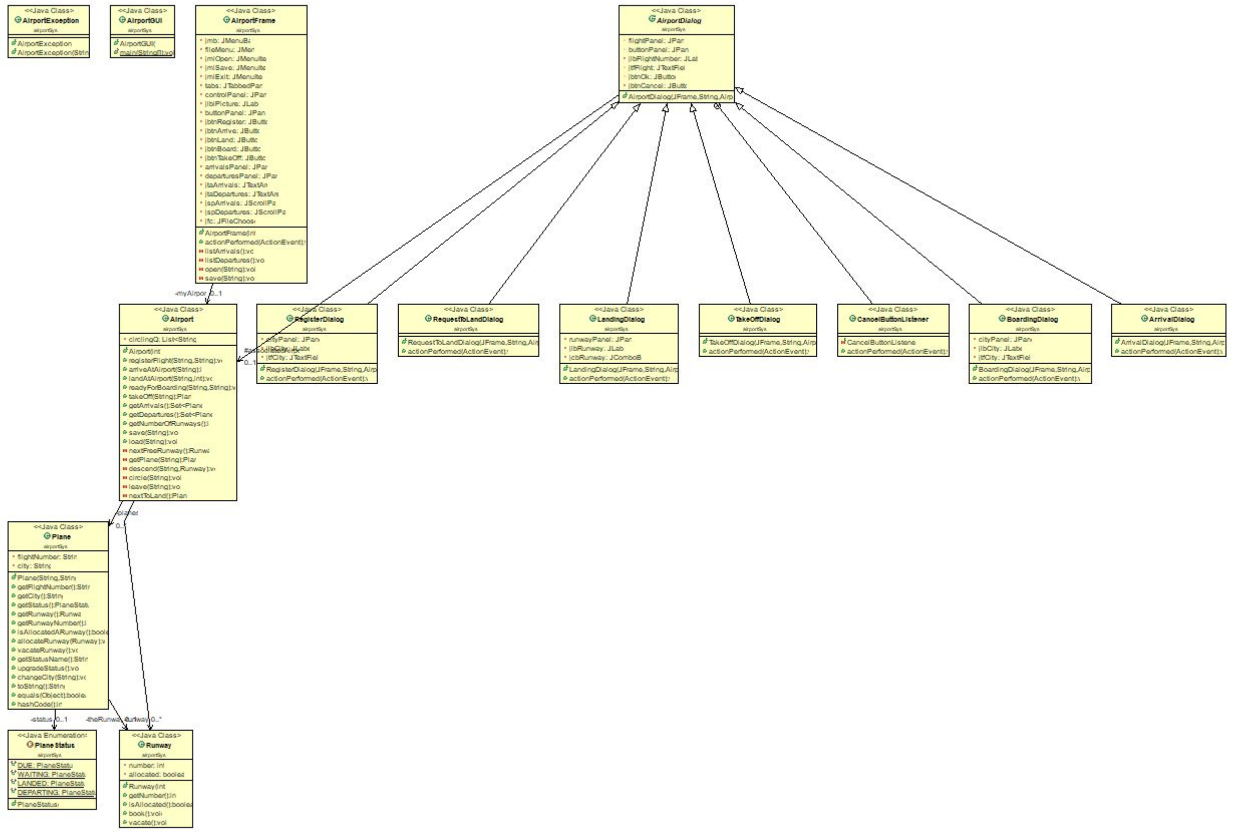
OK Cancel

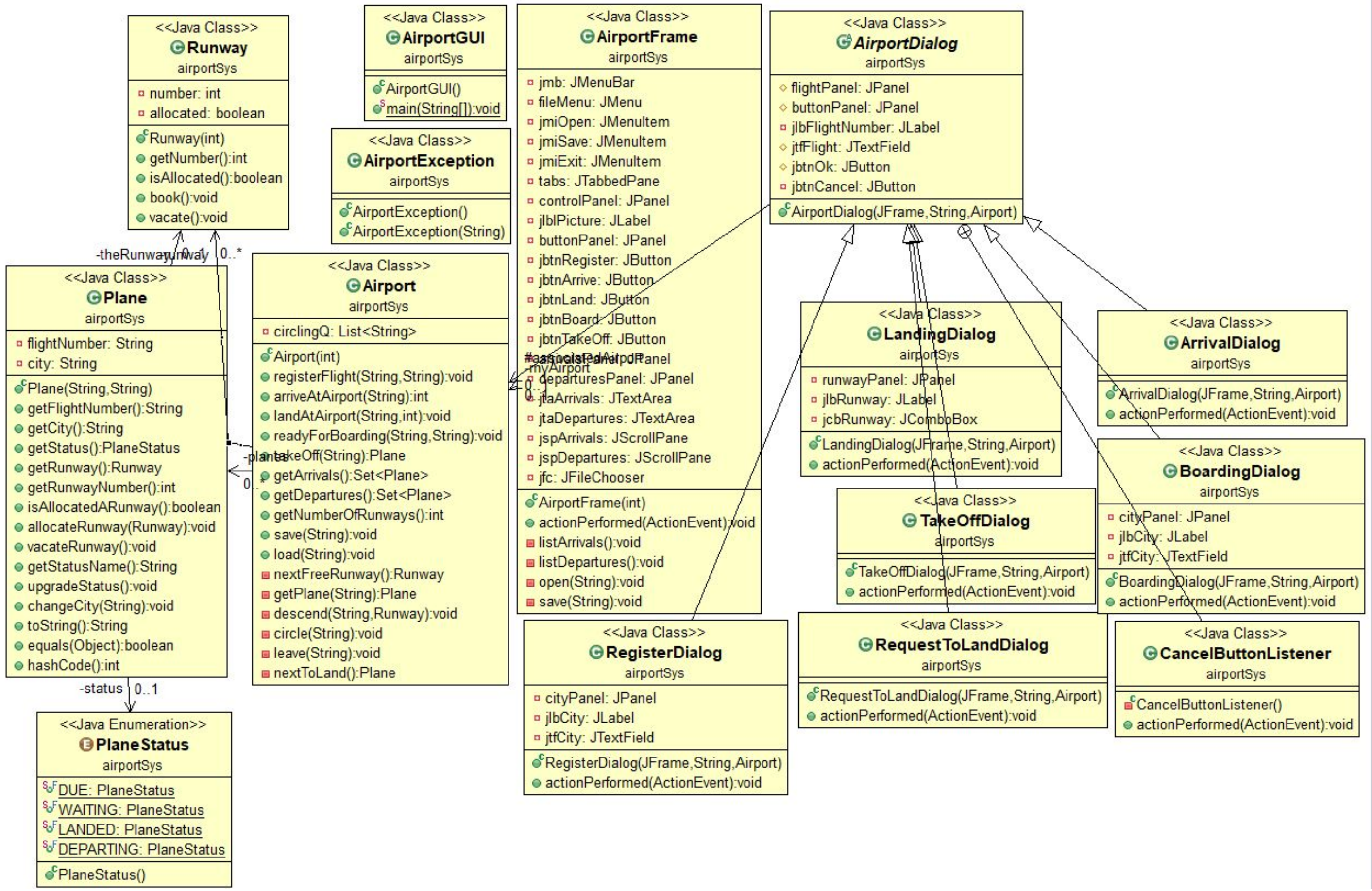
Enter an option

Register | Request to Land | Land | Board | Take Off


Airport System
File
Flight Control
Flight Arrivals
Flight Departures
Take Off form
Flight
enter flight number
Select
OK
Cancel
Enter an option
Register
Request to Land
Land
Board
Take Off

The System Under Test





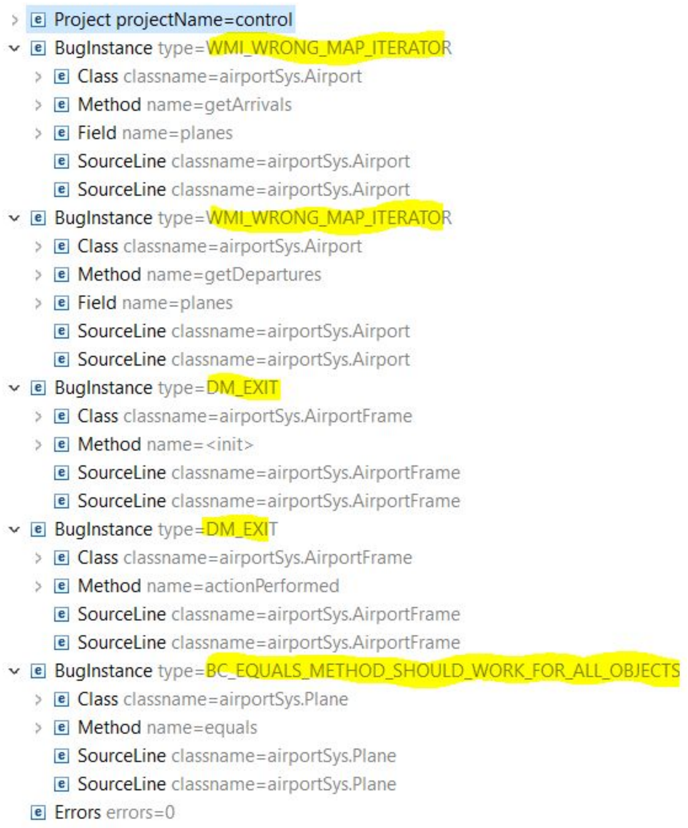
<<Java Class>>
Runway
airportSys
number: int
allocated: boolean
Runway(int)
getNumber():int
isAllocated():boolean
book():void
vacate():void
<<Java Class>>
AirportGUI
airportSys
AirportGUI()
main(String[]):void
<<Java Class>>
AirportException
airportSys
AirportException()
AirportException(String)
<<Java Class>>
AirportFrame
airportSys
jmb: JMenuBar
fileMenu: JMenu
jmiOpen: JMenuItem
jmiSave: JMenuItem
jmiExit: JMenuItem
tabs: JTabbedPane
controlPanel: JPanel
jlblPicture: JLabel
buttonPanel: JPanel
jbtnRegister: JButton
jbtnArrive: JButton
jbtnLand: JButton
jbtnBoard: JButton
jbtnTakeOff: JButton
arrivalsPanel: JPanel
departuresPanel: JPanel
jtaArrivals: JTextArea
jtaDepartures: JTextArea
jspArrivals: JScrollPane
jspDepartures: JScrollPane
jfc: JFileChooser
AirportFrame(int)
actionPerformed(ActionEvent):void
listArrivals():void
listDepartures():void
open(String):void
save(String):void
<<Java Class>>
AirportDialog
airportSys
flightPanel: JPanel
buttonPanel: JPanel
jlbFlightNumber: JLabel
jtfFlight: JTextField
jbtnOk: JButton
jbtnCancel: JButton
AirportDialog(JFrame,String,Airport)
-theRunway
-runway
0..1
0..*
<<Java Class>>
Plane
airportSys
flightNumber: String
city: String
Plane(String,String)
getFlightNumber():String
getCity():String
getStatus():PlaneStatus
getRunway():Runway
getRunwayNumber():int
isAllocatedARunway():boolean
allocateRunway(Runway):void
vacateRunway():void
getStatusName():String
upgradeStatus():void
changeCity(String):void
toString():String
equals(Object):boolean
hashCode():int
-planes
0..*
<<Java Class>>
Airport
airportSys
circlingQ: List<String>
Airport(int)
registerFlight(String,String):void
arriveAtAirport(String):int
landAtAirport(String,int):void
readyForBoarding(String,String):void
takeOff(String):Plane
getArrivals():Set<Plane>
getDepartures():Set<Plane>
getNumberOfRunways():int
save(String):void
load(String):void
nextFreeRunway():Runway
getPlane(String):Plane
descend(String,Runway):void
circle(String):void
leave(String):void
nextToLand():Plane
#myAirport
-myAirport
0..1
<<Java Class>>
LandingDialog
airportSys
runwayPanel: JPanel
jlbRunway: JLabel
jcbRunway: JComboBox
LandingDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
<<Java Class>>
ArrivalDialog
airportSys
ArrivalDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
<<Java Class>>
BoardingDialog
airportSys
cityPanel: JPanel
jlbCity: JLabel
jtfCity: JTextField
BoardingDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
<<Java Class>>
TakeOffDialog
airportSys
TakeOffDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
-status
0..1
<<Java Enumeration>>
PlaneStatus
airportSys
DUE: PlaneStatus
WAITING: PlaneStatus
LANDED: PlaneStatus
DEPARTING: PlaneStatus
PlaneStatus()
<<Java Class>>
RegisterDialog
airportSys
cityPanel: JPanel
jlbCity: JLabel
jtfCity: JTextField
RegisterDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
<<Java Class>>
RequestToLandDialog
airportSys
RequestToLandDialog(JFrame,String,Airport)
actionPerformed(ActionEvent):void
<<Java Class>>
CancelButtonListener
airportSys
CancelButtonListener()
actionPerformed(ActionEvent):void

Tools

- Java Path Finder (JPF)
    - What can it do?
        - Find and explain defects
        - Collect "deep" runtime information like coverage metrics
        - Deduce interesting test vectors and create corresponding test drivers
- Bug Finder
    - FindBugs, SpotBugs
        - Detect possible bugs in java programs
        - Potential errors are classified in different ranks

Tool 1: FindBugs, SpotBugs, and Error Prone

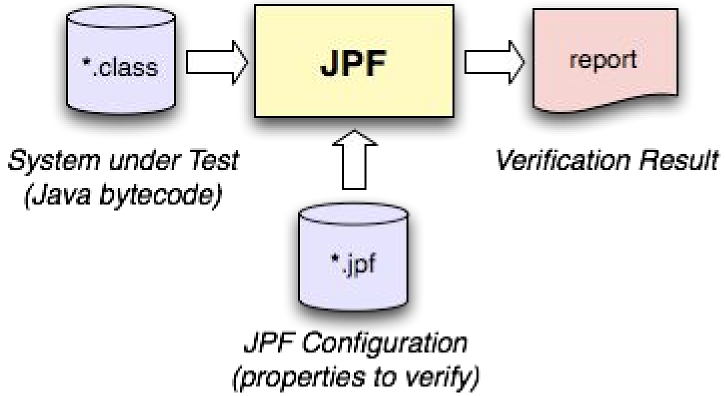
- Uses
  - To find potential bugs in our Java program
  - Outline of output of FindBugs can be seen on the right
- Results
  - There were no significant bugs in our system
  - Bugs could not be replicated using our modules



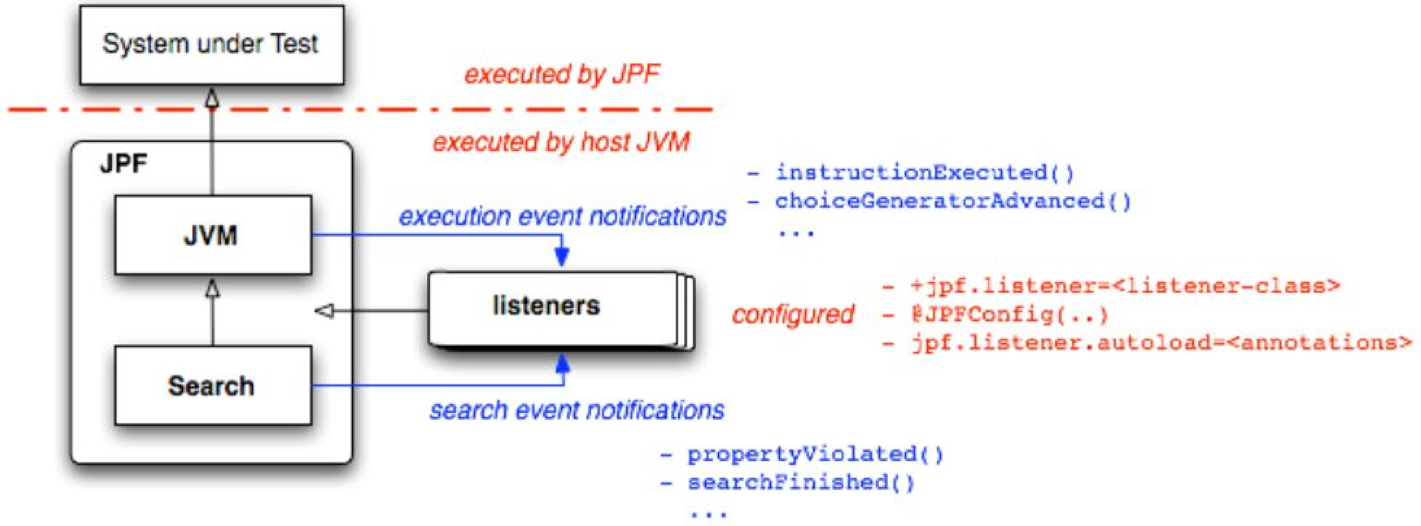
```
> e Project projectName=control
v e BugInstance type=WMI_WRONG_MAP_ITERATOR
    > e Class classname=airportSys.Airport
    > e Method name=getArrivals
    > e Field name=planes
      e SourceLine classname=airportSys.Airport
      e SourceLine classname=airportSys.Airport
v e BugInstance type=WMI_WRONG_MAP_ITERATOR
    > e Class classname=airportSys.Airport
    > e Method name=getDepartures
    > e Field name=planes
      e SourceLine classname=airportSys.Airport
      e SourceLine classname=airportSys.Airport
v e BugInstance type=DM_EXIT
    > e Class classname=airportSys.AirportFrame
    > e Method name=<init>
      e SourceLine classname=airportSys.AirportFrame
      e SourceLine classname=airportSys.AirportFrame
v e BugInstance type=DM_EXIT
    > e Class classname=airportSys.AirportFrame
    > e Method name=actionPerformed
      e SourceLine classname=airportSys.AirportFrame
      e SourceLine classname=airportSys.AirportFrame
v e BugInstance type=BC_EQUALS_METHOD_SHOULD_WORK_FOR_ALL_OBJECTS
    > e Class classname=airportSys.Plane
    > e Method name=equals
      e SourceLine classname=airportSys.Plane
      e SourceLine classname=airportSys.Plane
  e Errors errors=0
```

Tool 2: Java Path Finder

JPF is verification tool that was initially developed by NASA Ames Research Center and open sourced in 2005. The base configuration is centered on the JPF core which is Java Virtual Machine that detects unhandled exceptions, deadlocks, and races by supplanting the default JVM during execution.

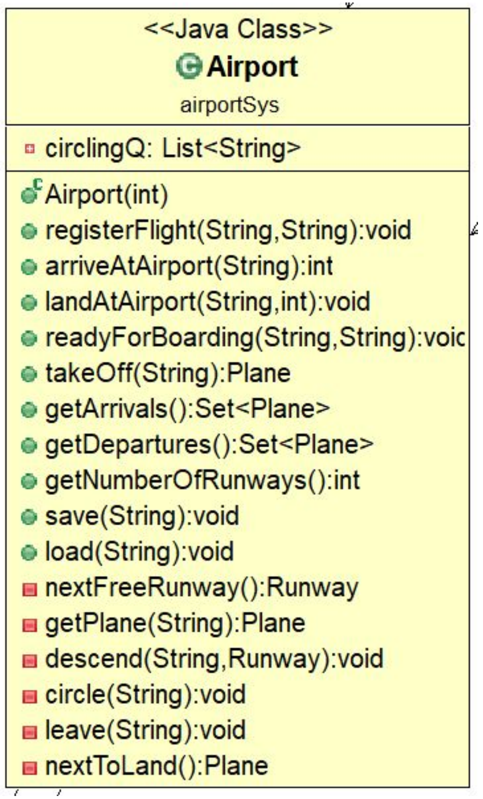


Listeners in JPF

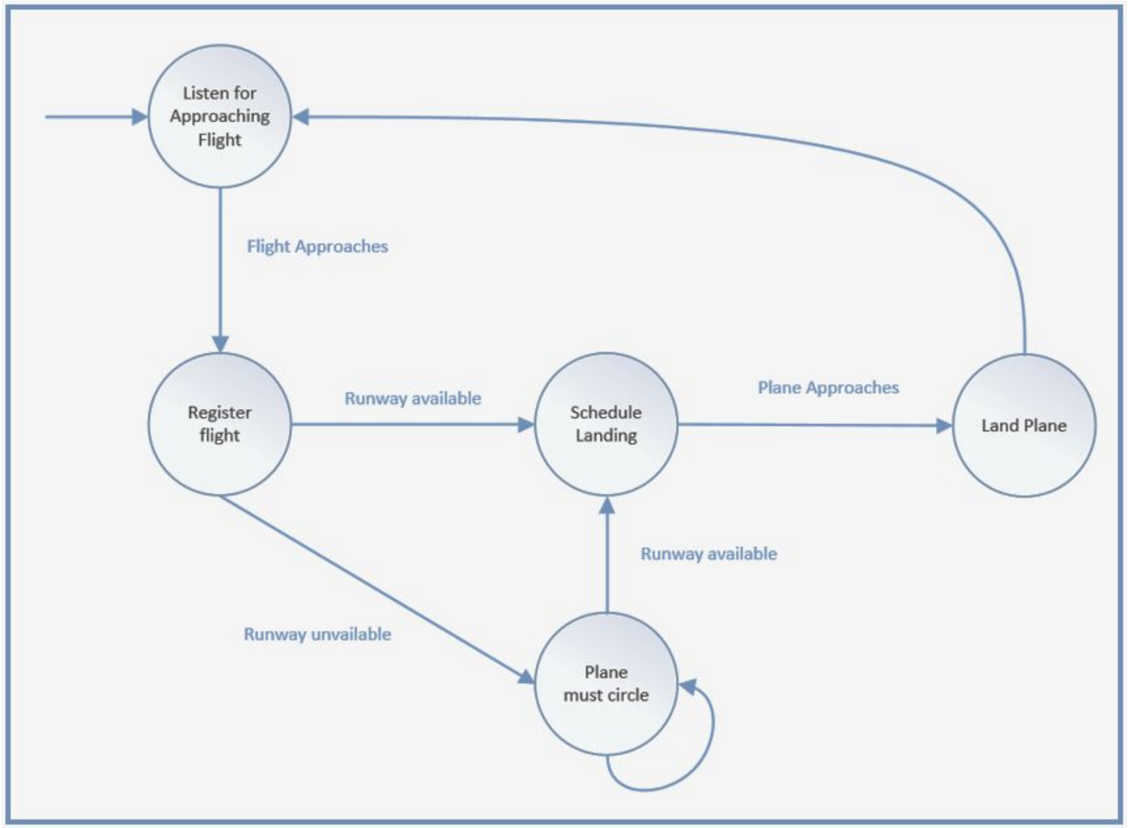


How listeners work

The Airport Class



The Airport Class



Property Testing

- No two flights have the same flight number. ***verified***
- No more than one flight is scheduled to land on a given runway. ***verified***
- A flight that is not registered cannot take off, land or board. ***verified***

Properties were verified using Java Assertions and the AssertionProperty listener.

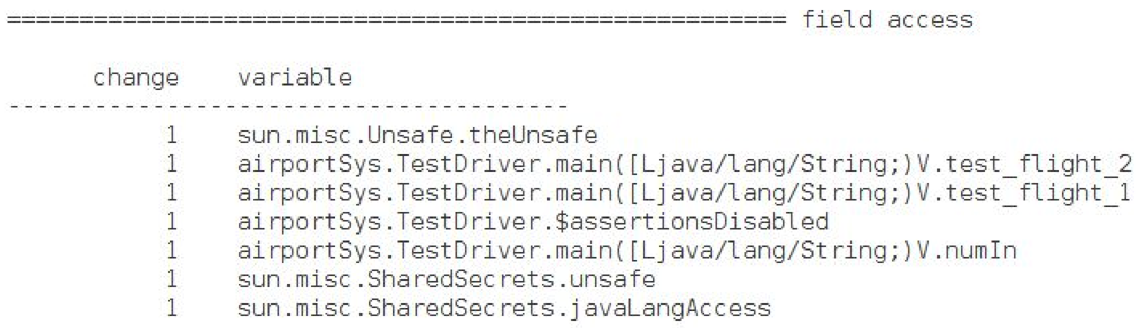
JPF-Core Listeners Used in Testing

AssertionProperty	Adds additional capabilities for native assertion functionality in java.lang.
ErrorTrace Generator	This is a listener to output a lightweight error trace. It prints the instructions at POR boundaries or points where there are multiple choices. An example is shown below.
ExceptionInjector	Injects exceptions for testing purposes.

JPF-Core Listeners Used in Testing Continued...

CoverageAnalyzer	Collects and tabulates method and class information based on executed instructions.
VarTracker	Tracks which variables are changed, how often and

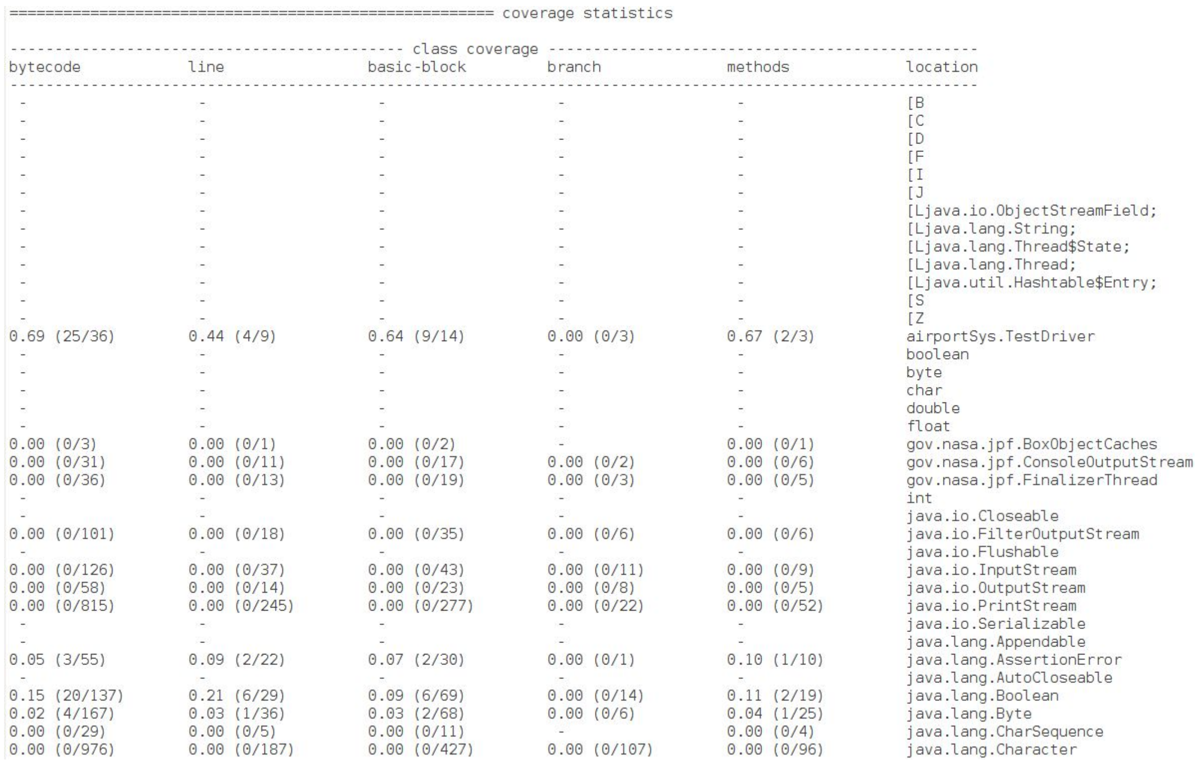
VarTracker Output



```
====================================================== field access

      change    variable
---------------------------------------
           1    sun.misc.Unsafe.theUnsafe
           1    airportSys.TestDriver.main([Ljava/lang/String;)V.test_flight_2
           1    airportSys.TestDriver.main([Ljava/lang/String;)V.test_flight_1
           1    airportSys.TestDriver.$assertionsDisabled
           1    airportSys.TestDriver.main([Ljava/lang/String;)V.numIn
           1    sun.misc.SharedSecrets.unsafe
           1    sun.misc.SharedSecrets.javaLangAccess
```

CoverageAnalyzer Output



```
====================================================== coverage statistics

-------------------------------------------- class coverage ------------------------------------------------
bytecode            line                basic-block         branch              methods             location
------------------------------------------------------------------------------------------------------------
 -                   -                   -                   -                   -                  [B
 -                   -                   -                   -                   -                  [C
 -                   -                   -                   -                   -                  [D
 -                   -                   -                   -                   -                  [F
 -                   -                   -                   -                   -                  [I
 -                   -                   -                   -                   -                  [J
 -                   -                   -                   -                   -                  [Ljava.io.ObjectStreamField;
 -                   -                   -                   -                   -                  [Ljava.lang.String;
 -                   -                   -                   -                   -                  [Ljava.lang.Thread$State;
 -                   -                   -                   -                   -                  [Ljava.lang.Thread;
 -                   -                   -                   -                   -                  [Ljava.util.Hashtable$Entry;
 -                   -                   -                   -                   -                  [S
 -                   -                   -                   -                   -                  [Z
0.69 (25/36)        0.44 (4/9)          0.64 (9/14)         0.00 (0/3)          0.67 (2/3)          airportSys.TestDriver
 -                   -                   -                   -                   -                  boolean
 -                   -                   -                   -                   -                  byte
 -                   -                   -                   -                   -                  char
 -                   -                   -                   -                   -                  double
 -                   -                   -                   -                   -                  float
0.00 (0/3)          0.00 (0/1)          0.00 (0/2)           -                  0.00 (0/1)          gov.nasa.jpf.BoxObjectCaches
0.00 (0/31)         0.00 (0/11)         0.00 (0/17)         0.00 (0/2)          0.00 (0/6)          gov.nasa.jpf.ConsoleOutputStream
0.00 (0/36)         0.00 (0/13)         0.00 (0/19)         0.00 (0/3)          0.00 (0/5)          gov.nasa.jpf.FinalizerThread
 -                   -                   -                   -                   -                  int
 -                   -                   -                   -                   -                  java.io.Closeable
0.00 (0/101)        0.00 (0/18)         0.00 (0/35)         0.00 (0/6)          0.00 (0/6)          java.io.FilterOutputStream
 -                   -                   -                   -                   -                  java.io.Flushable
0.00 (0/126)        0.00 (0/37)         0.00 (0/43)         0.00 (0/11)         0.00 (0/9)          java.io.InputStream
0.00 (0/58)         0.00 (0/14)         0.00 (0/23)         0.00 (0/8)          0.00 (0/5)          java.io.OutputStream
0.00 (0/815)        0.00 (0/245)        0.00 (0/277)        0.00 (0/22)         0.00 (0/52)         java.io.PrintStream
 -                   -                   -                   -                   -                  java.io.Serializable
 -                   -                   -                   -                   -                  java.lang.Appendable
0.05 (3/55)         0.09 (2/22)         0.07 (2/30)         0.00 (0/1)          0.10 (1/10)         java.lang.AssertionError
 -                   -                   -                   -                   -                  java.lang.AutoCloseable
0.15 (20/137)       0.21 (6/29)         0.09 (6/69)         0.00 (0/14)         0.11 (2/19)         java.lang.Boolean
0.02 (4/167)        0.03 (1/36)         0.03 (2/68)         0.00 (0/6)          0.04 (1/25)         java.lang.Byte
0.00 (0/29)         0.00 (0/5)          0.00 (0/11)          -                  0.00 (0/4)          java.lang.CharSequence
0.00 (0/976)        0.00 (0/187)        0.00 (0/427)        0.00 (0/107)        0.00 (0/96)         java.lang.Character
```

Comparison of Tools

	Java PathFinder	**Static Analysis Tools**
Bugs found	1	9
Types of bugs found	UncaughtException	Compiler Warnings, error handling issues, syntax and style.
Test preparation time	Requires configuration and preparation	minimal
Test execution time	Involves compiling code then running it on a JVM. State space exploration is also timely by nature.	< 1 minute
Ease of use	Has a moderate learning curve.	Simple to use.

Limitations and Drawbacks

- JPF requires code to compile and run before it can be tested.

- Test configuration required editing the system and writing drivers, yet we still needed to modify the system classes e.g. exposing private variables and changing the exception handling.

- JPF is not a cure-all - black box tests are still necessary. Verification using JPF cannot easily be adapted for software engineering practices like Continuous Integration and Test Driven Development.

- It requires a detailed understanding of the implementation. Therefore separation of labor may not be the best approach.

Limitations and Drawbacks

- We found that model checking with JPF is more practical for targeted tests at the unit level, or even the block level than it is for holistic system tests.
- This is because configuration system for JPF modules and runtime options can be complex.
- For this reason, verification with JPF outside of property testing should be focused on the potentially problematic sections of code in a system.

Some Advantages of JPF

- JPF has a lot of functionality and much of it is unique to model checking.

- It is possible to get better debugging information than a simulation would provide.

- Has some useful tools like the VarTracker and CoverageAnalyzer listeners.

- State space exploration produces a robust system.

Conclusion

- As expected, we were able to perform tests that exceeded the boundaries of black box testing.
- We conclude that for high fault-tolerance applications, formal verification should be used alongside black box unit testing even though it cannot replace it.

Thank you!