

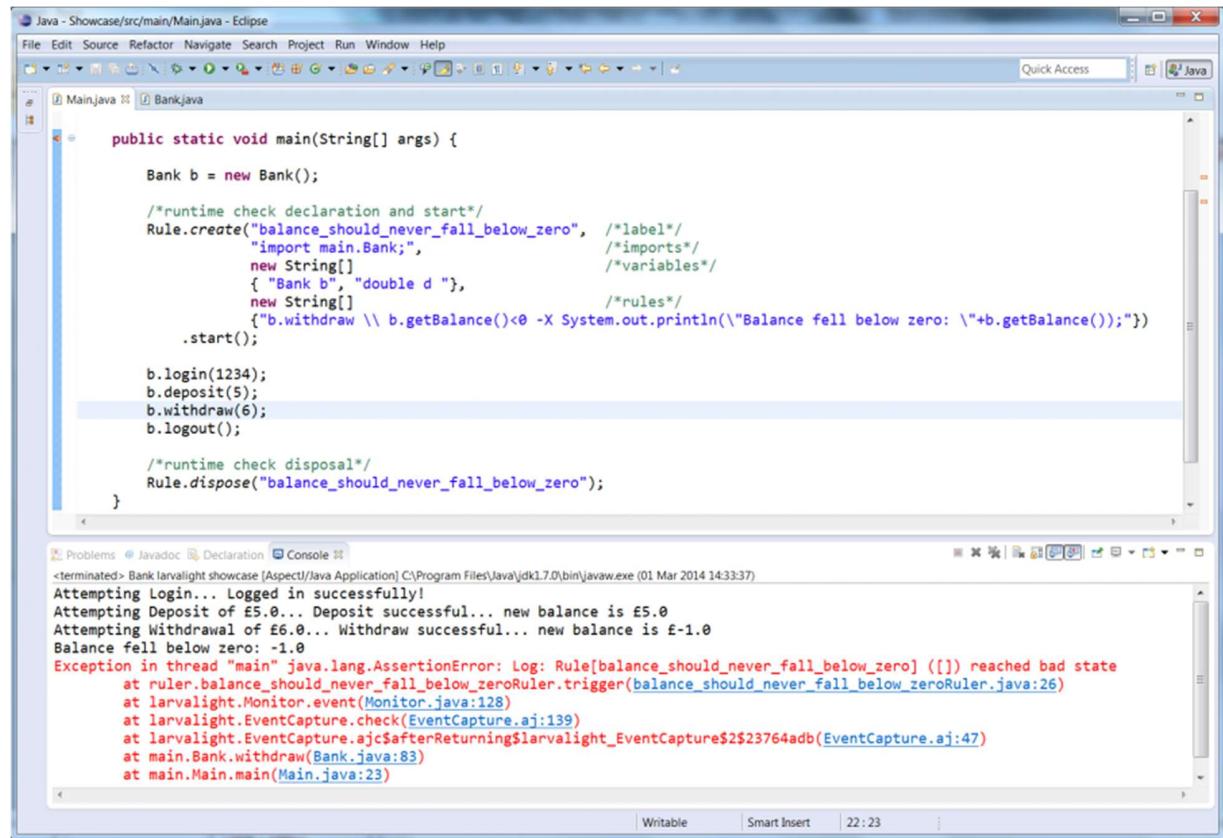
LarvaLight Screenshots

[The code shown in the screenshots below can be downloaded from [here](#)]

Screenshot 1 - An example of a single basic rule

The rule below checks that after a withdrawal, the balance never goes below zero.

The console shows the output of running a faulty implementation which allows the balance to go below zero.



The screenshot shows the Eclipse IDE interface. The top window displays a Java code editor with the following content:

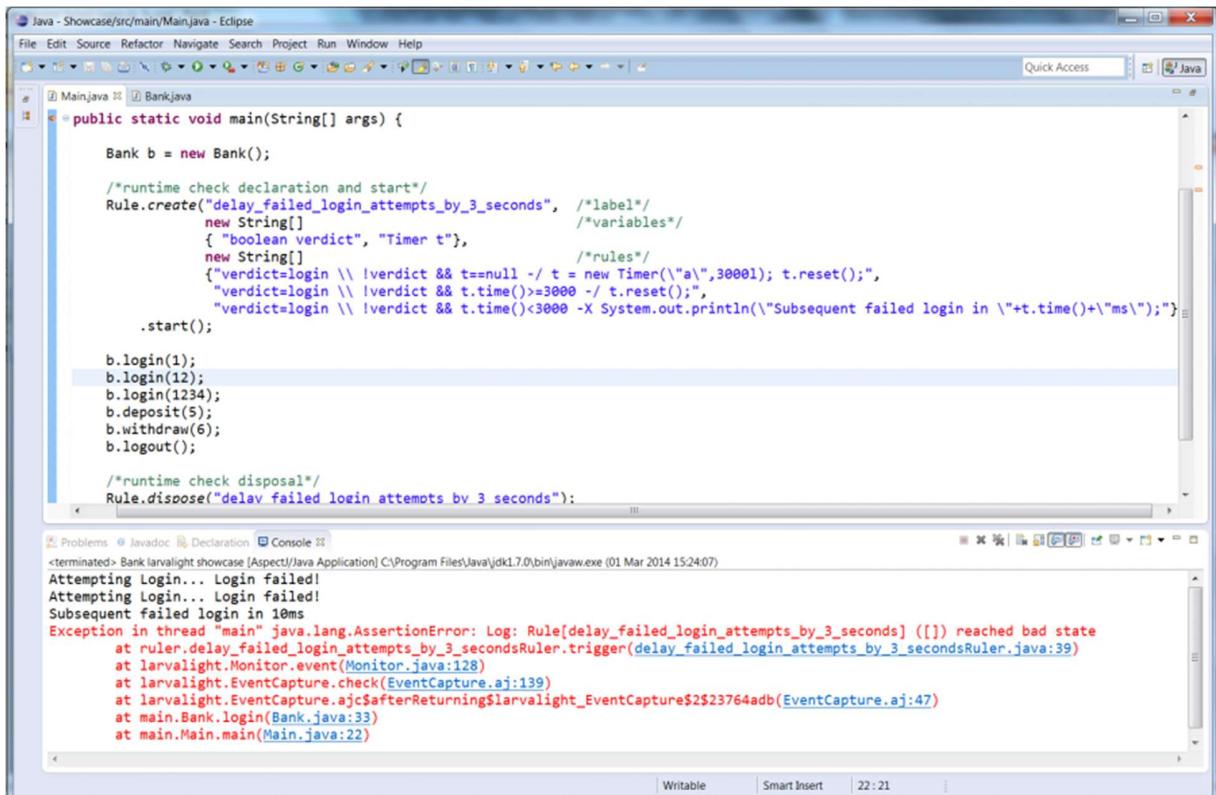
```
Java - Showcase/src/main/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
MainJava BankJava
public static void main(String[] args) {
    Bank b = new Bank();
    /*runtime check declaration and start*/
    Rule.create("balance_should_never_fall_below_zero", /*label*/
               "import main.Bank;", /*imports*/
               new String[] { "Bank b", "double d "}, /*variables*/
               new String[] {"b.withdraw \\" b.getBalance()<0 -X System.out.println(\"Balance fell below zero: \"+b.getBalance());"} /*rules*/
               .start();
    b.login(1234);
    b.deposit(5);
    b.withdraw(6);
    b.logout();
    /*runtime check disposal*/
    Rule.dispose("balance_should_never_fall_below_zero");
}
```

The bottom window is a terminal window showing the execution of the code. The output is:

```
<terminated> Bank larvalight showcase [Aspect/Java Application] C:\Program Files\Java\jdk1.7.0\bin\javaw.exe (01 Mar 2014 14:33:37)
Attempting Login... Logged in successfully!
Attempting Deposit of £5.0... Deposit successful... new balance is £5.0
Attempting Withdrawal of £6.0... Withdraw successful... new balance is £-1.0
Balance fell below zero: -1.0
Exception in thread "main" java.lang.AssertionError: Log: Rule[balance_should_never_fall_below_zero] ([]) reached bad state
    at ruler.balance_should_never_fall_below_zeroRuler.trigger(balance_should_never_fall_below_zeroRuler.java:26)
    at larvalight.Monitor.event(Monitor.java:128)
    at larvalight.EventCapture.check(EventCapture.aj:139)
    at larvalight.EventCapture.ajc$afterReturning$larvalight_EventCapture$2$23764adb(EventCapture.aj:47)
    at main.Bank.withdraw(Bank.java:83)
    at main.Main.main(Main.java:23)
```

Screenshot 2 - An example of a rule taking time into consideration

The check below asserts that at least there is a three second duration between subsequent failed login attempts.



The screenshot shows the Eclipse IDE interface. The top window is titled "Java - Showcase/src/main/Main.java - Eclipse". The code editor displays the following Java code:

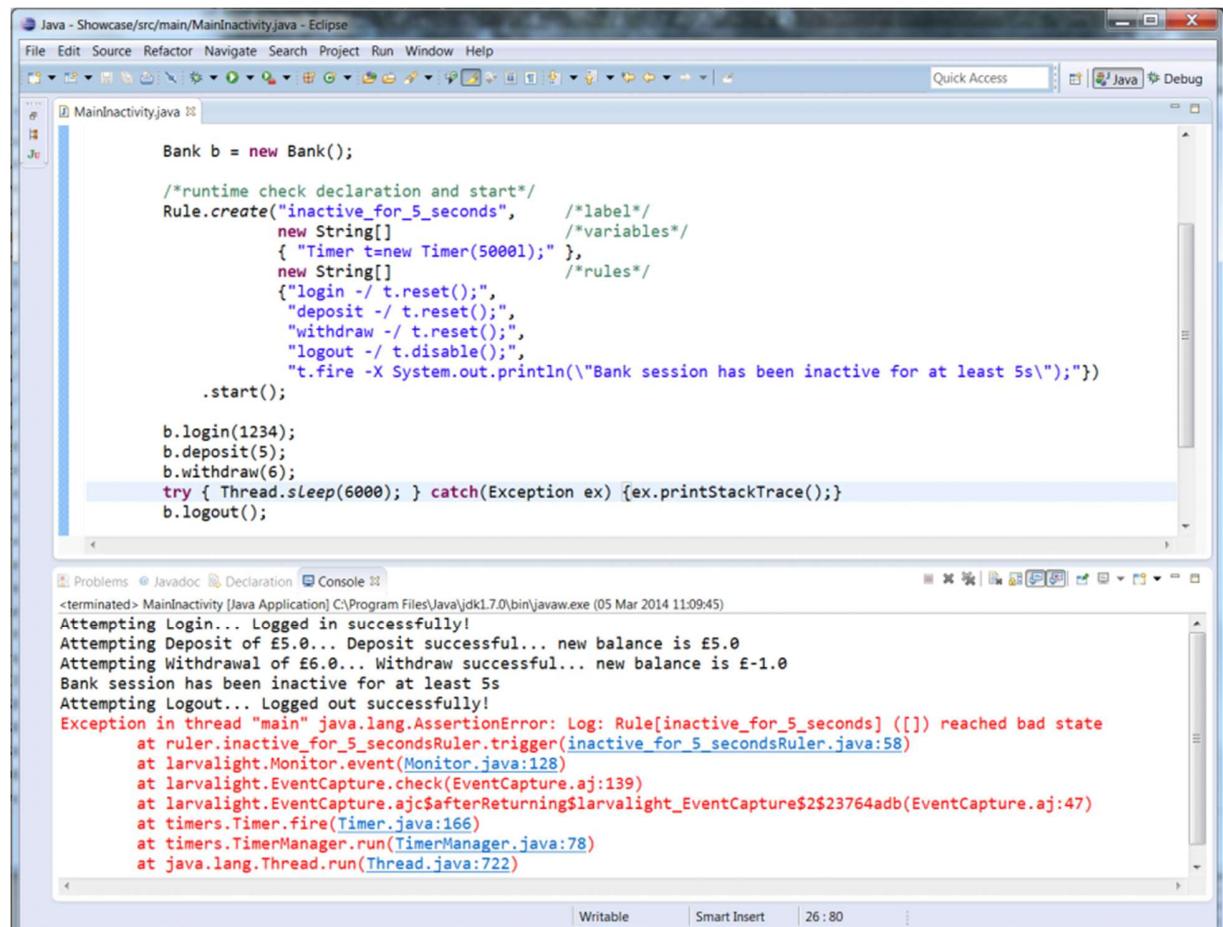
```
Java - Showcase/src/main/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Main.java Bank.java
public static void main(String[] args) {
    Bank b = new Bank();
    /*runtime check declaration and start*/
    Rule.create("delay_failed_login_attempts_by_3_seconds", /*label*/
               new String[] { "boolean verdict", "Timer t" }, /*variables*/
               new String[] { /*rules*/
                   {"verdict=login \&\& t==null -/ t = new Timer(\"a\",3000); t.reset();",
                    "verdict=login \&\& t.time()>=3000 -/ t.reset();",
                    "verdict=login \&\& t.time()<3000 -X System.out.println(\"Subsequent failed login in \"+t.time() +\"ms\");"
                }
            .start();
    b.login(1);
    b.login(12);
    b.login(1234);
    b.deposit(5);
    b.withdraw(6);
    b.logout();
    /*runtime check disposal*/
    Rule.dispose("delay_failed_login_attempts_by_3_seconds");
}
```

The terminal window below the code editor shows the execution output:

```
<terminated> Bank larvalight showcase [Aspect/Java Application] C:\Program Files\Java\jdk1.7.0\bin\javaw.exe (01 Mar 2014 15:24:07)
Attempting Login... Login failed!
Attempting Login... Login failed!
Subsequent failed login in 10ms
Exception in thread "main" java.lang.AssertionError: Log: Rule[delay_failed_login_attempts_by_3_seconds] ([]) reached bad state
    at ruler.delay_failed_login_attempts_by_3_seconds$Ruler.trigger(delay_failed_login_attempts_by_3_secondsRuler.java:39)
    at larvalight.Monitor.event(Monitor.java:128)
    at larvalight.EventCapture.check(EventCapture.aj:139)
    at larvalight.EventCapture.ajc$afterReturning$larvalight_EventCapture$2$23764adb(EventCapture.aj:47)
    at main.Bank.login(Bank.java:33)
    at main.Main.main(Main.java:22)
```

Screenshot 3 - A time-triggered rule

The rule checks that there is no five-second period during which there is no activity. Note how the last event is timer-based, i.e. it fires upon the elapsing of 5 seconds.



The screenshot shows the Eclipse IDE interface with the following details:

- Java - Showcase/src/main/MainInactivity.java - Eclipse** is the active project and file.
- MainInactivity.java** contains Java code that simulates a bank session with a timer rule:

```
Bank b = new Bank();

/*runtime check declaration and start*/
Rule.create("inactive_for_5_seconds",      /*label*/
           new String[]                  /*variables*/
           { "Timer t=new Timer(5000);",   /*rules*/
             new String[]                /*rules*/
             {"login -/ t.reset();",
              "deposit -/ t.reset();",
              "withdraw -/ t.reset();",
              "logout -/ t.disable();",
              "t.fire -X System.out.println(\"Bank session has been inactive for at least 5s\");"
            }
           ).start();

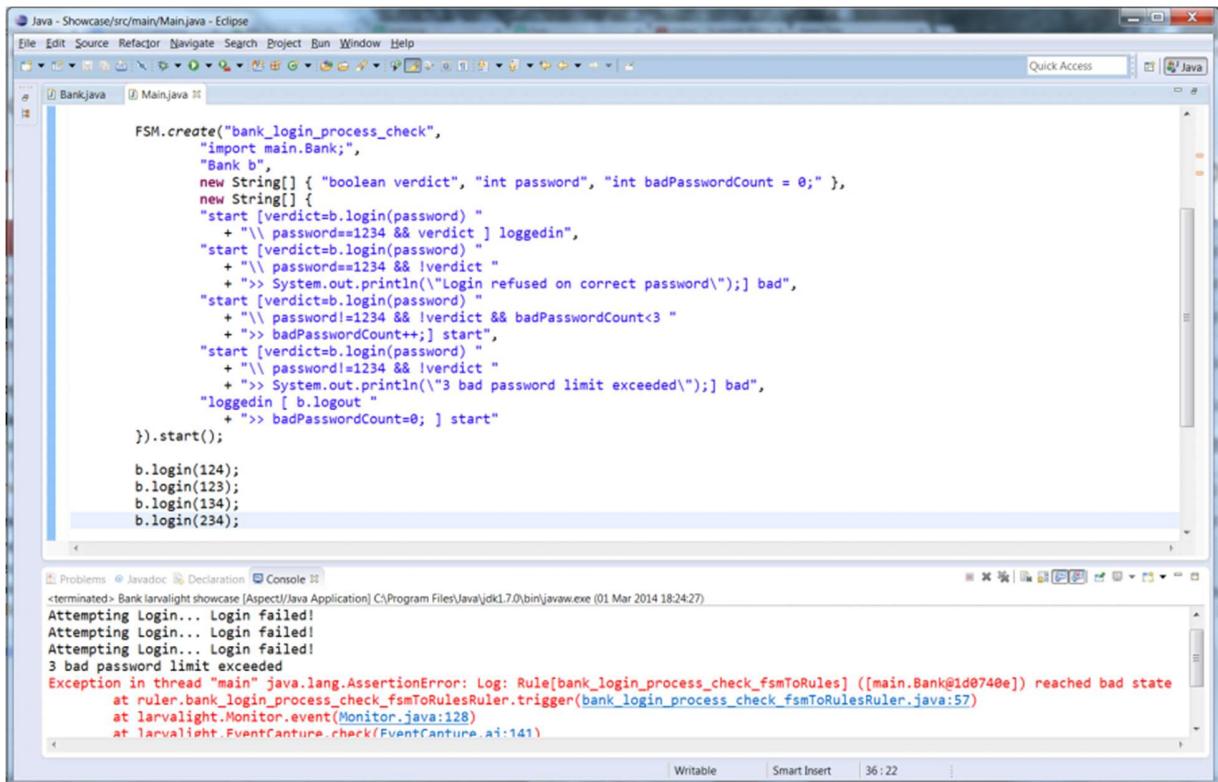
b.login(1234);
b.deposit(5);
b.withdraw(6);
try { Thread.sleep(6000); } catch(Exception ex) {ex.printStackTrace();}
b.logout();
```

- Console** tab shows the application's output:

```
<terminated> MainInactivity [Java Application] C:\Program Files\Java\jdk1.7.0\bin\javaw.exe (05 Mar 2014 11:09:45)
Attempting Login... Logged in successfully!
Attempting Deposit of £5.0... Deposit successful... new balance is £5.0
Attempting Withdrawal of £6.0... Withdrawal successful... new balance is £-1.0
Bank session has been inactive for at least 5s
Attempting Logout... Logged out successfully!
Exception in thread "main" java.lang.AssertionError: Log: Rule[inactive_for_5_seconds] ([]) reached bad state
    at ruler.inactive_for_5_secondsRuler.trigger(inactive_for_5_secondsRuler.java:58)
    at larvalight.Monitor.event(Monitor.java:128)
    at larvalight.EventCapture.check(EventCapture.aj:139)
    at larvalight.EventCapture.aj$afterReturning$larvalight_EventCapture$2$23764adb(EventCapture.aj:47)
    at timers.Timer.fire(Timer.java:166)
    at timers.TimerManager.run(TimerManager.java:78)
    at java.lang.Thread.run(Thread.java:722)
```

Screenshot 4 - A finite state machine example

The finite state machine below checks the log in logic of the system. Note that it is completely up to the user to decide the level of abstract the checking goes into.



The screenshot shows the Eclipse IDE interface. The top bar displays 'Java - Showcase/src/main/Main.java - Eclipse'. The main area shows a code editor with the following Java code:

```
FSM.create("bank_login_process_check",
    "import main.Bank;",
    "Bank b",
    new String[] { "boolean verdict", "int password", "int badPasswordCount = 0;" },
    new String[] {
        "start [verdict=b.login(password) "
        + "\\" password==1234 && verdict ] loggedin",
        "start [verdict=b.login(password) "
        + "\\" password==1234 && !verdict "
        + ">> System.out.println(\"Login refused on correct password\");] bad",
        "start [verdict=b.login(password) "
        + "\\" password!=1234 && verdict && badPasswordCount<3 "
        + ">> badPasswordCount++;] start",
        "start [verdict=b.login(password) "
        + "\\" password!=1234 && !verdict "
        + ">> System.out.println(\"3 bad password limit exceeded\");] bad",
        "loggedin [ b.logout "
        + ">> badPasswordCount=0; ] start"
    }).start();

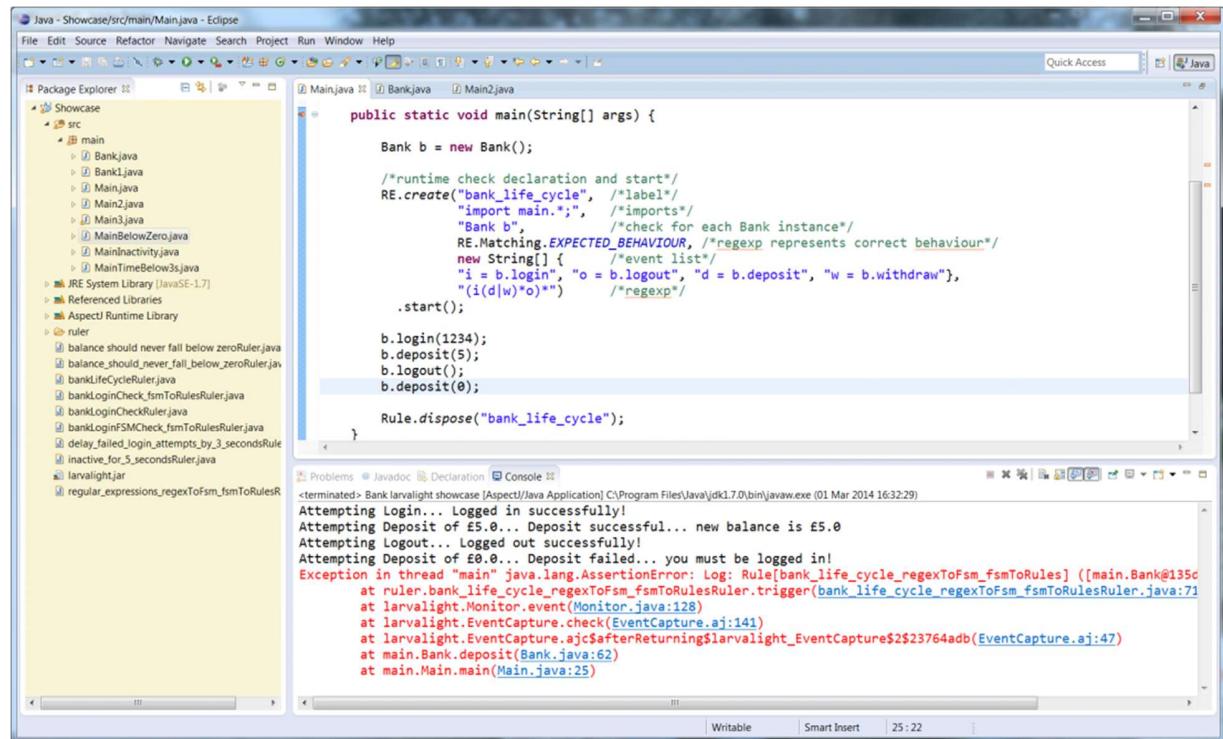
b.login(124);
b.login(123);
b.login(134);
b.login(234);
```

Below the code editor is a terminal window showing the execution of the code. The output is:

```
<terminated> Bank larvalight showcase [AspectJ/Java Application] C:\Program Files\Java\jdk1.7.0\bin\javaw.exe (01 Mar 2014 18:24:27)
Attempting Login... Login failed!
Attempting Login... Login failed!
Attempting Login... Login failed!
3 bad password limit exceeded
Exception in thread "main" java.lang.AssertionError: Log: Rule[bank_login_process_check_fsmToRules] ([main.Bank@1d0740e]) reached bad state
    at ruler.bank_login_process_check_fsmToRulesRuler.trigger(bank_login_process_check_fsmToRulesRuler.java:57)
    at larvalight.Monitor.event(Monitor.java:128)
    at larvalight.EventCapture.check(EventCapture.ai:141)
```

Screenshot 5 - A regular expression

Using the regular expression below, we show how the lifecycle of a bank session can be captured very succinctly.



The screenshot shows the Eclipse IDE interface with a Java project named "Showcase" open. The "Main.java" file is the active editor, displaying the following code:

```
public static void main(String[] args) {
    Bank b = new Bank();

    /*runtime check declaration and start*/
    RE.create("bank_life_cycle", /*label*/
              "import main.*; /*imports*/
              "Bank b", /*check for each Bank instance*/
              RE.Matching.EXPECTED_BEHAVIOUR, /*regexp represents correct behaviour*/
              new String[] { /*event list*/
                  "i = b.login", "o = b.logout", "d = b.deposit", "w = b.withdraw",
                  "(i|d|w|o)*" /*regexp*/
              }.start();

    b.login(1234);
    b.deposit(5);
    b.logout();
    b.deposit(0);

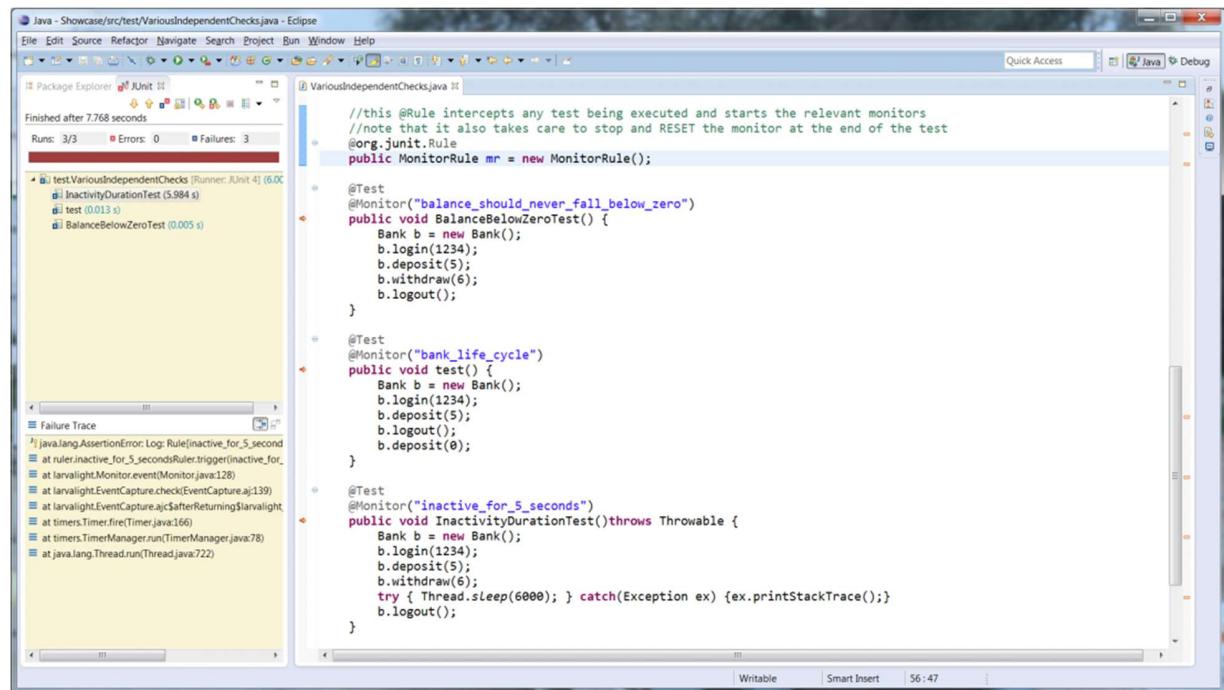
    Rule.dispose("bank_life_cycle");
}
```

The "Console" tab shows the following output:

```
Attempting Login... Logged in successfully!
Attempting Deposit of £5.0... Deposit successful... new balance is £5.0
Attempting Logout... Logged out successfully!
Attempting Deposit of £0.0... Deposit failed... you must be logged in!
Exception in thread "main" java.lang.AssertionError: Log: Rule[bank_life_cycle_regexToFsm_fsmToRules] ([main.Bank@135c
at ruler.bank_life_cycle_regexToFsm_fsmToRulesRuler.trigger(bank_life_cycle_regexToFsm_fsmToRulesRuler.java:71
at larvalight.Monitor.event(Monitor.java:128)
at larvalight.EventCapture.check(EventCapture.aj:141)
at larvalight.EventCapture.ajc$afterReturning$larvalight_EventCapture$2$23764adb(EventCapture.aj:47)
at main.Bank.deposit(Bank.java:62)
at main.Main.main(Main.java:25)
```

Screenshot 6 - JUnit with monitor annotations

Complementing typical assertions, monitors can provide extra power to capture checks easily across your tests.



The screenshot shows the Eclipse IDE interface with the following details:

- Java - Showcase/src/test/VariousIndependentChecks.java - Eclipse** is the active project.
- JUnit** perspective is selected.
- Run** status: 3/3, Errors: 0, Failures: 3.
- Failure Trace** shows the following stack trace for a failure:

```
java.lang.AssertionError: Log: Rule[inactive_for_5_second]
at ruler.inactive_for_5_secondsRuler.trigger(inactive_for_
at larvalight.Monitor.event(Monitor.java:128)
at larvalight.EventCapture.check(EventCapture.java:139)
at larvalight.EventCapture$yc$AfterReturning$larvalight.
at timers.Timer.fire(Timer.java:166)
at timers.TimerManager.run(TimerManager.java:78)
at java.lang.Thread.run(Thread.java:722)
```

- Code Editor** displays the `VariousIndependentChecks.java` file with JUnit test cases using monitor annotations:

```
public class VariousIndependentChecks {
    //this @Rule intercepts any test being executed and starts the relevant monitors
    //note that it also takes care to stop and RESET the monitor at the end of the test
    @org.junit.Rule
    public MonitorRule mr = new MonitorRule();

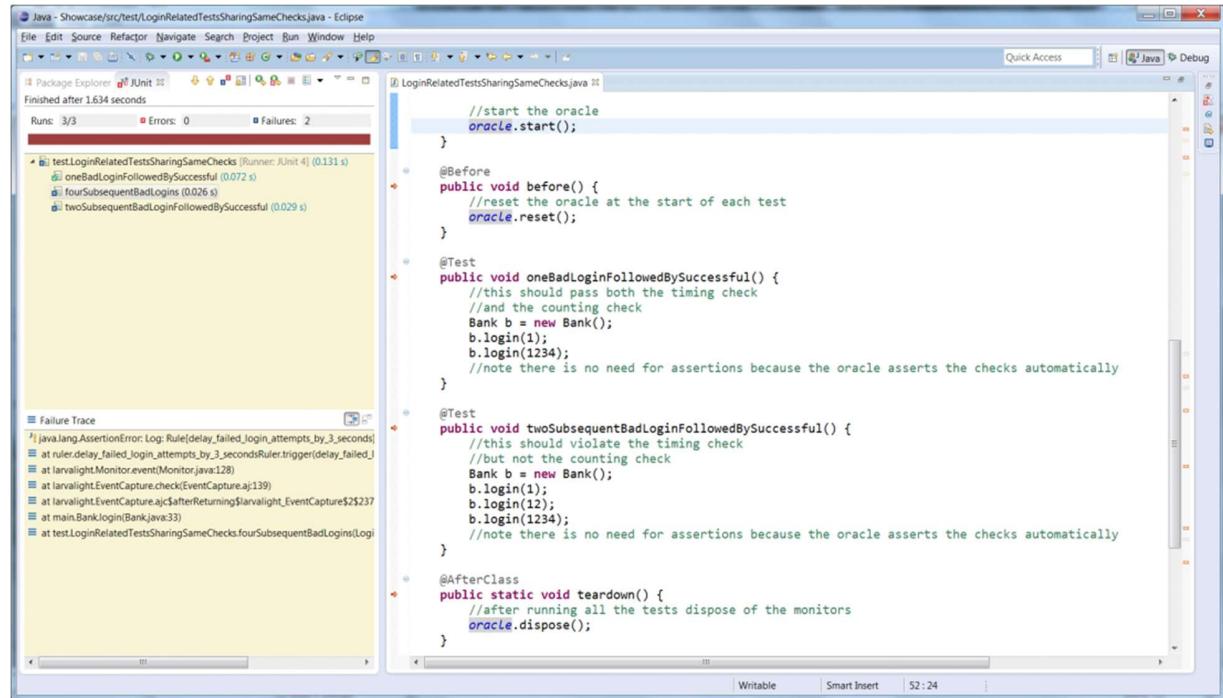
    @Test
    @Monitor("balance_should_never_fall_below_zero")
    public void BalanceBelowZeroTest() {
        Bank b = new Bank();
        b.login(1234);
        b.deposit(5);
        b.withdraw(6);
        b.logout();
    }

    @Test
    @Monitor("bank_life_cycle")
    public void test() {
        Bank b = new Bank();
        b.login(1234);
        b.deposit(5);
        b.logout();
        b.deposit(0);
    }

    @Test
    @Monitor("inactive_for_5_seconds")
    public void InactivityDurationTest() throws Throwable {
        Bank b = new Bank();
        b.login(1234);
        b.deposit(5);
        b.withdraw(6);
        try { Thread.sleep(6000); } catch(Exception ex) {ex.printStackTrace();}
        b.logout();
    }
}
```

Screenshot 7 - Managing multiple monitors

To make it easier to handle a number of monitors, LarvaLight provide the Oracle class - essentially a collection of monitors which can be started, reset, or disposed all at once.



The screenshot shows the Eclipse IDE interface with a Java file named `LoginRelatedTestsSharingSameChecks.java` open in the editor. The code demonstrates the use of the `oracle` class to manage multiple monitors. The code includes annotations for `@Before`, `@Test`, and `@AfterClass`, and methods for `start`, `reset`, and `dispose`. The editor shows the results of a JUnit 4 run with 3/3 tests passed, 0 errors, and 2 failures. A failure trace is also visible.

```
//start the oracle
oracle.start();

@Before
public void before() {
    //reset the oracle at the start of each test
    oracle.reset();
}

@Test
public void oneBadLoginFollowedBySuccessful() {
    //this should pass both the timing check
    //and the counting check
    Bank b = new Bank();
    b.login(1);
    b.login(1234);
    //note there is no need for assertions because the oracle asserts the checks automatically
}

@Test
public void twoSubsequentBadLoginFollowedBySuccessful() {
    //this should violate the timing check
    //but not the counting check
    Bank b = new Bank();
    b.login(1);
    b.login(12);
    b.login(1234);
    //note there is no need for assertions because the oracle asserts the checks automatically
}

@AfterClass
public static void teardown() {
    //after running all the tests dispose of the monitors
    oracle.dispose();
}
```