Hands on Complex Event Processing: Lab Guide

Hands on Complex Event Processing Information	
Technology/Product	Red Hat JBoss Business Rules Management Platform
Difficulty	3
Time	90 minutes
Prerequisites	Knowledge of Java, Eclipse, Junit, Maven

In this lab you will...

- Learn about Complex Event Processing
- Modify and create Complex Event Processing business rules

This is any basic information someone should know before starting your lab. I could be some background information on the subject or something that is covered by your presentation, but is not necessarily provided in the steps of the lab below. It could also just be a further, more detailed, explanation of why you're doing this lab, or how this might help someone in their real lives.

BEFORE YOU BEGIN...

Raise your hand if you have any questions! We are here to help.

Built on Eclipse JBoss® Developer Studio provides superior support for your entire development lifecycle. It includes a broad set of tooling capabilities and support for multiple programming models and frameworks, including Java™ Enterprise Edition 6, RichFaces, JavaServer Faces (JSF), Enterprise JavaBeans (EJB), Java Persistence API (JPA), and Hibernate®, JAX-RS with RESTEasy, Contexts Dependency Injection (CDI), HTML5, and many other popular technologies. It provides developer choice in supporting multiple JVMs, productivity with Maven, and in testing with Arquillian. It is fully tested and certified to ensure that all its plug-ins, runtime components, and their dependencies are compatible with each other.

The lab will use JBoss Developer Studio as the IDE and will run all code inside of Junit Tests. JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. Familiarity with JUnit is helpful but not required.

Have fun!

KEY TERMS

CEP is short for Complex Event Processing. The terms will be used interchangeably throughout the labs.

A **Fact** is a Java POJO. The rules engine analyzes Facts.

An **Event** is both a significant change in state at a particular point in time a Fact that has temporal metadata associated with it. In CEP the rules engine analyzed Events.

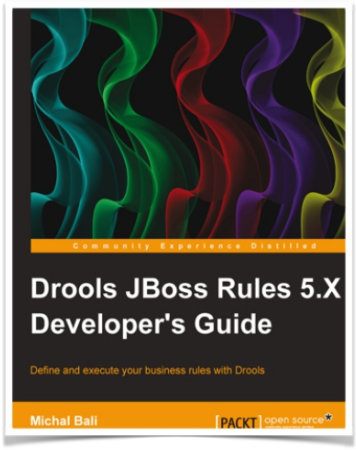
Kie stands for “Knowledge Is Everything.” It is an upstream, community umbrella for projects in the Drools family. Drools Expert, Fusion, OptaPlanner, UberFire and jBPM are included.

Drools Fusion is the upstream project that contains the CEP functionality.

A **.drl** is a file that holds Drools rules.

LAB INSTRUCTIONS

The examples in this lab are from a financial services domain and borrow heavily from **Drools JBoss Rules 5.x** by Michl Bali.



The labs consist of the following Maven projects :

- summit2015-model
- summit2015-tests
- lab1
- lab2
- lab3
- lab4
- lab5

You will import each project into Eclipse and work through the corresponding text. If you encounter any problems along the way ask for help from one of the proctors! Complex Event Processing is a “complex” topic.

PROJECT OVERVIEW

summit2015-model

This project contains the domain model for the labs

summit2015-tests

Contains a base JUnit test handles the boilerplate code for running rules applications inside of Junit

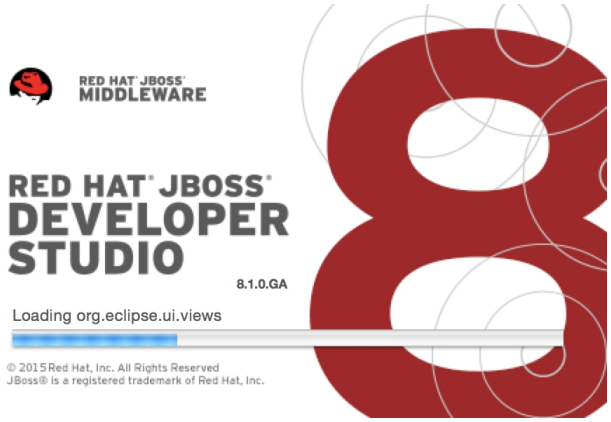
lab1

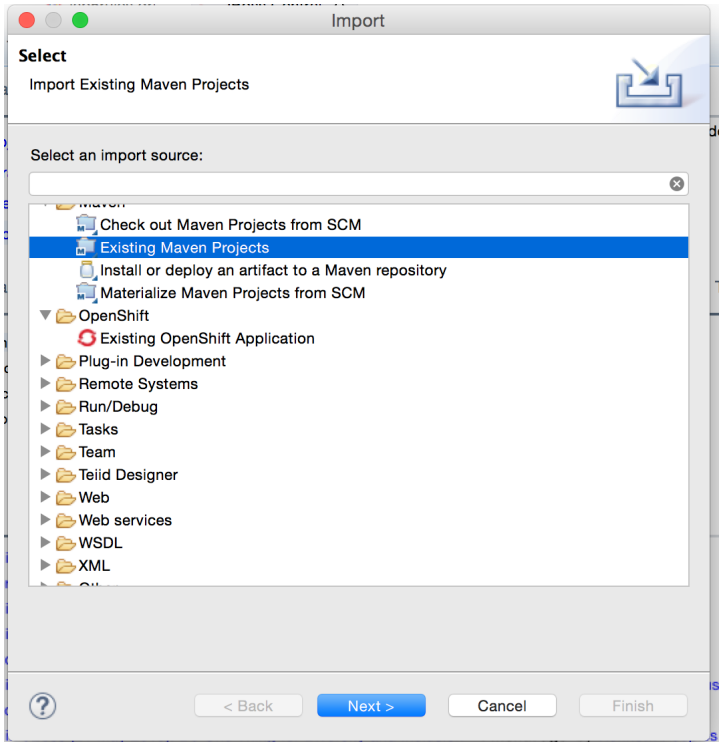
Introduces the basics of authoring and running rules

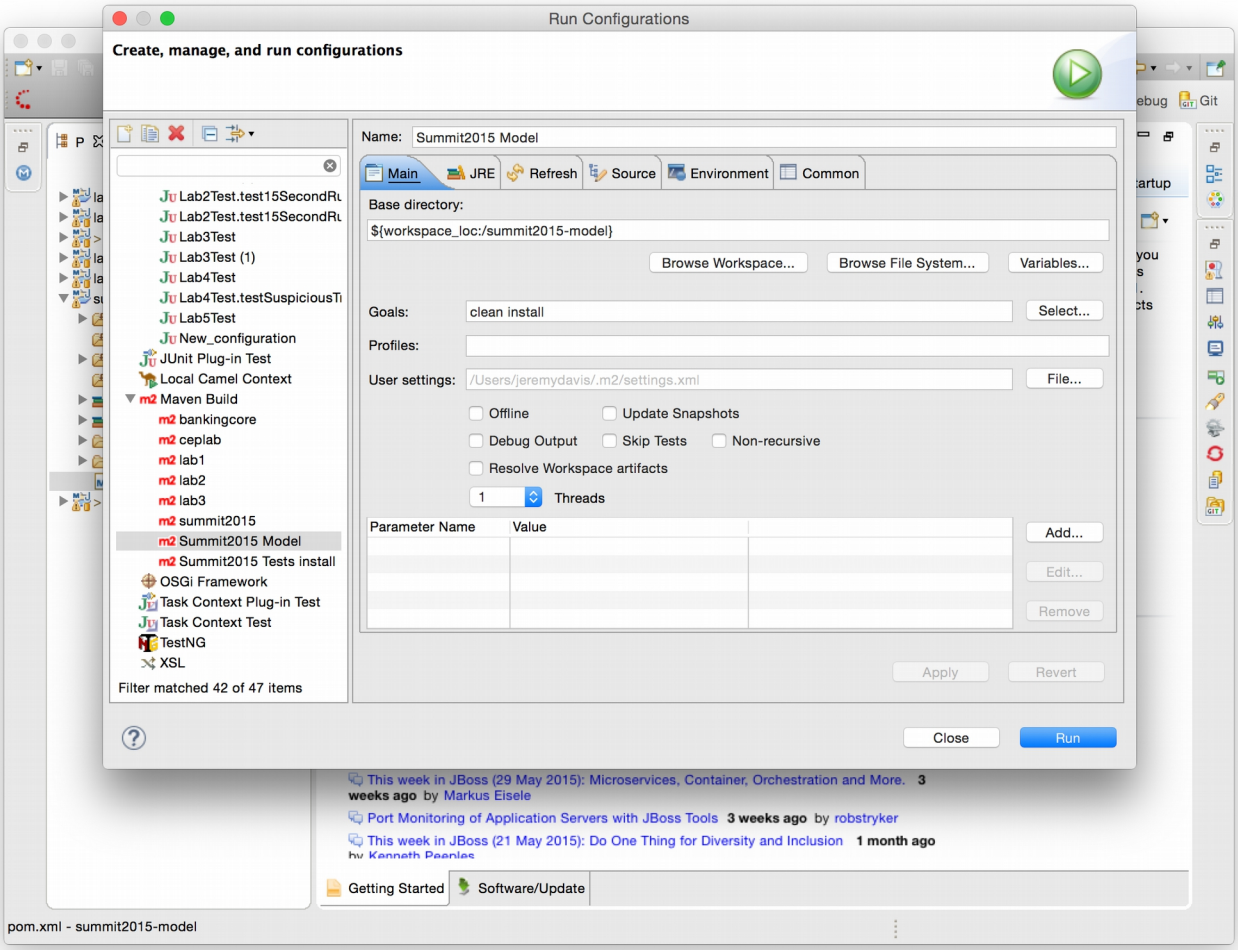
LAB 1

Summary

1. Open JBoss Developer Studio
2. Import summit2015-model and summit2015-tests from /home/student as “Existing Maven Projects”
3. Build and install both projects
4. Browse the domain classes and BaseCEPTestCase






Run Configurations
Create, manage, and run configurations
Name: Summit2015 Model
Main
JRE
Refresh
Source
Environment
Common
Base directory:
${workspace_loc:/summit2015-model}
Browse Workspace...
Browse File System...
Variables...
Goals:
clean install
Select...
Profiles:
User settings:
/Users/jeremydavis/.m2/settings.xml
File...
Offline
Update Snapshots
Debug Output
Skip Tests
Non-recursive
Resolve Workspace artifacts
1
Threads
Parameter Name
Value
Add...
Edit...
Remove
Apply
Revert
Close
Run
Lab2Test.test15SecondRu
Lab2Test.test15SecondRu
Lab3Test
Lab3Test (1)
Lab4Test
Lab4Test.testSuspiciousTi
Lab5Test
New_configuration
JUnit Plug-in Test
Local Camel Context
Maven Build
bankingcore
ceplab
lab1
lab2
lab3
summit2015
Summit2015 Model
Summit2015 Tests install
OSGi Framework
Task Context Plug-in Test
Task Context Test
TestNG
XSL
Filter matched 42 of 47 items
This week in JBoss (29 May 2015): Microservices, Container, Orchestration and More. 3 weeks ago by Markus Eisele
Port Monitoring of Application Servers with JBoss Tools 3 weeks ago by robstryker
This week in JBoss (21 May 2015): Do One Thing for Diversity and Inclusion 1 month ago
Getting Started
Software/Update
pom.xml - summit2015-model

DIRECTIONS

1. Open JBoss Developer Studio
2. Import summit2015-model and summit2015-tests from /home/student as “Existing Maven Projects”
    1. Choose File → Import → Existing Maven Projects
    2. Navigate to the “student2015” directory
    3. Select the folder, “summit2015-model”
    4. Click “Finish”
    5. Repeat for the “summit2015-tests” project
3. Build and install both projects
    1. You can build by choosing “Run Configurations” from the top menu or by right clicking on the pom.xml for the summit2015-model project
    2. Be sure to assign the the targets, “Clean, Install,” in the dialog
4. Browse the domain classes and the BaseCEPTestCase class

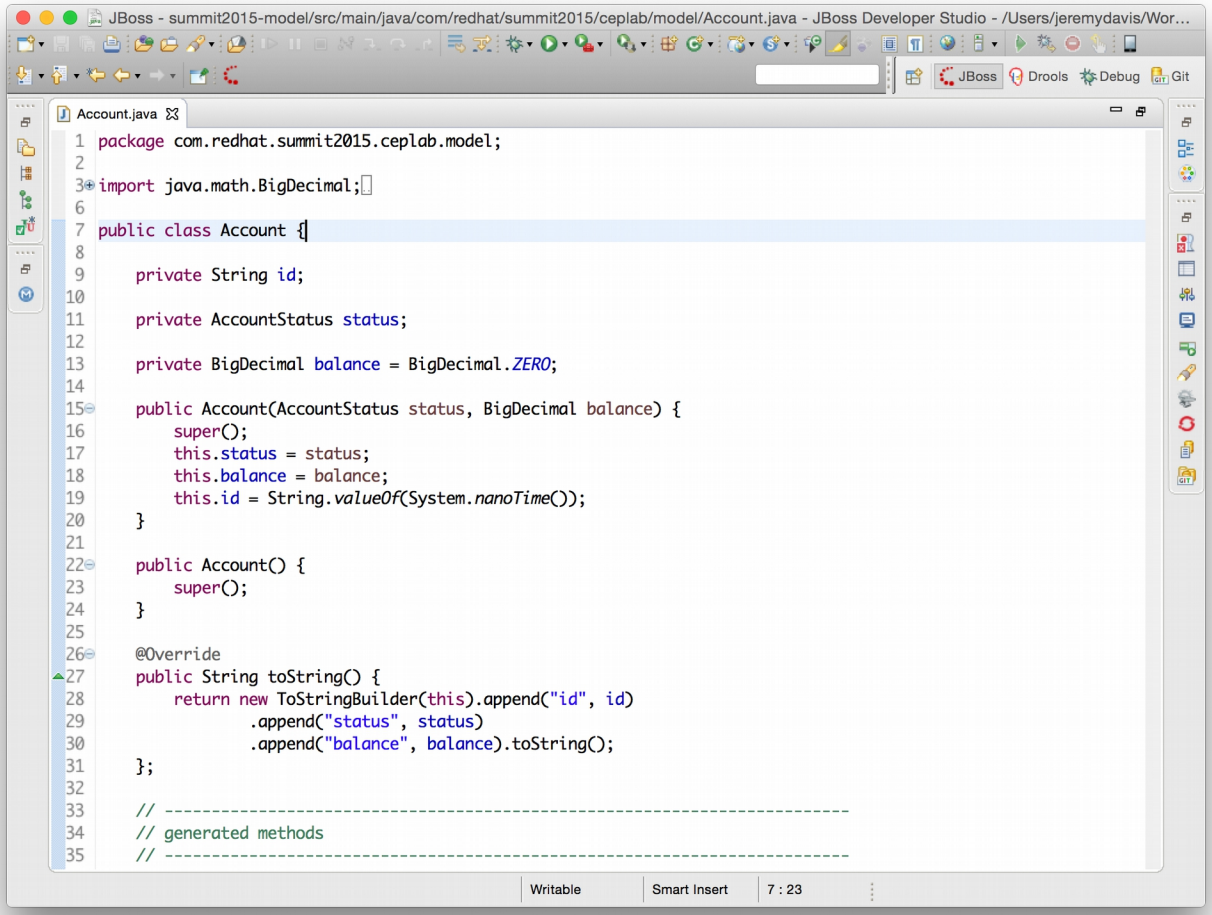
Domain Classes

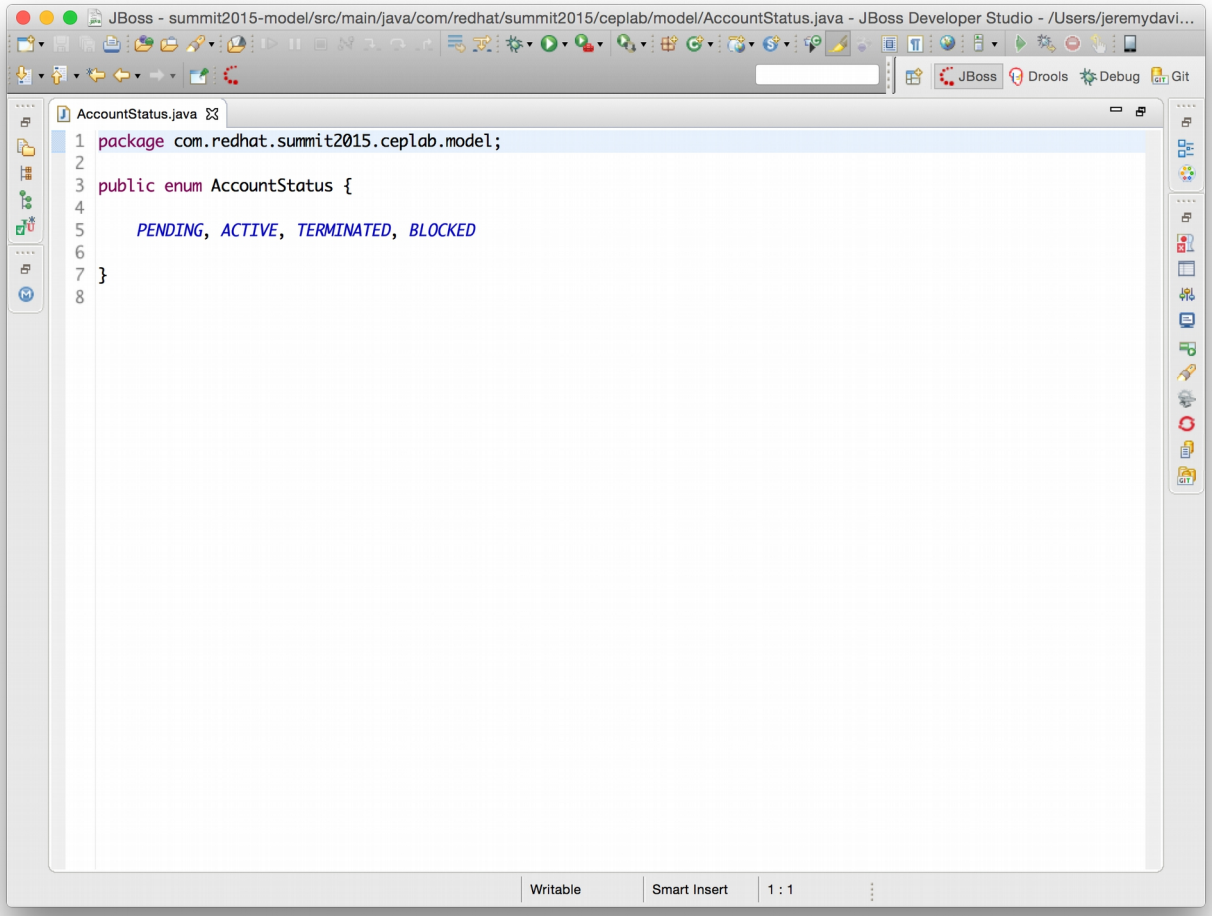
NOTE : SCREENSHOTS OF THE DOMAIN MODEL ARE IN THE FOLLOWING PAGES.

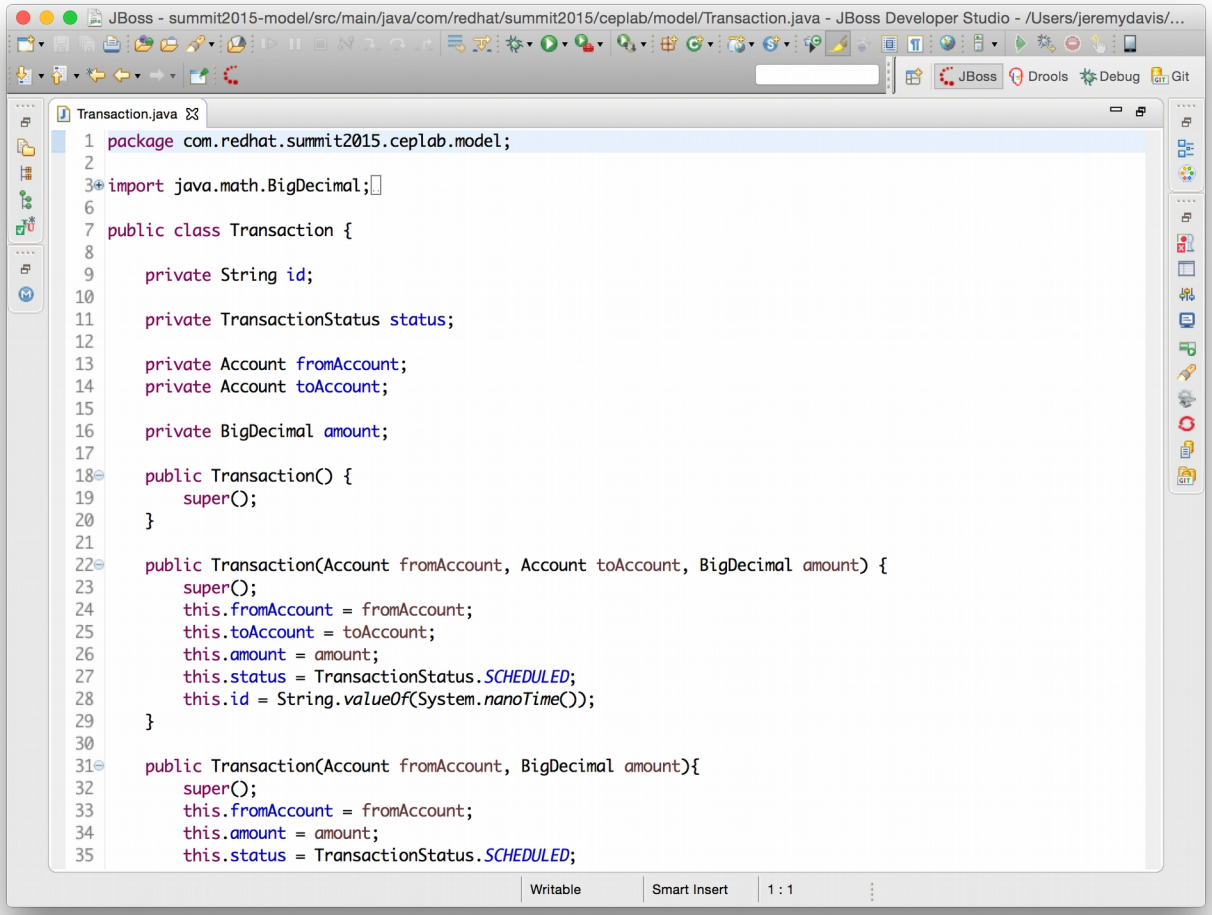
The domain consists of 4 classes :

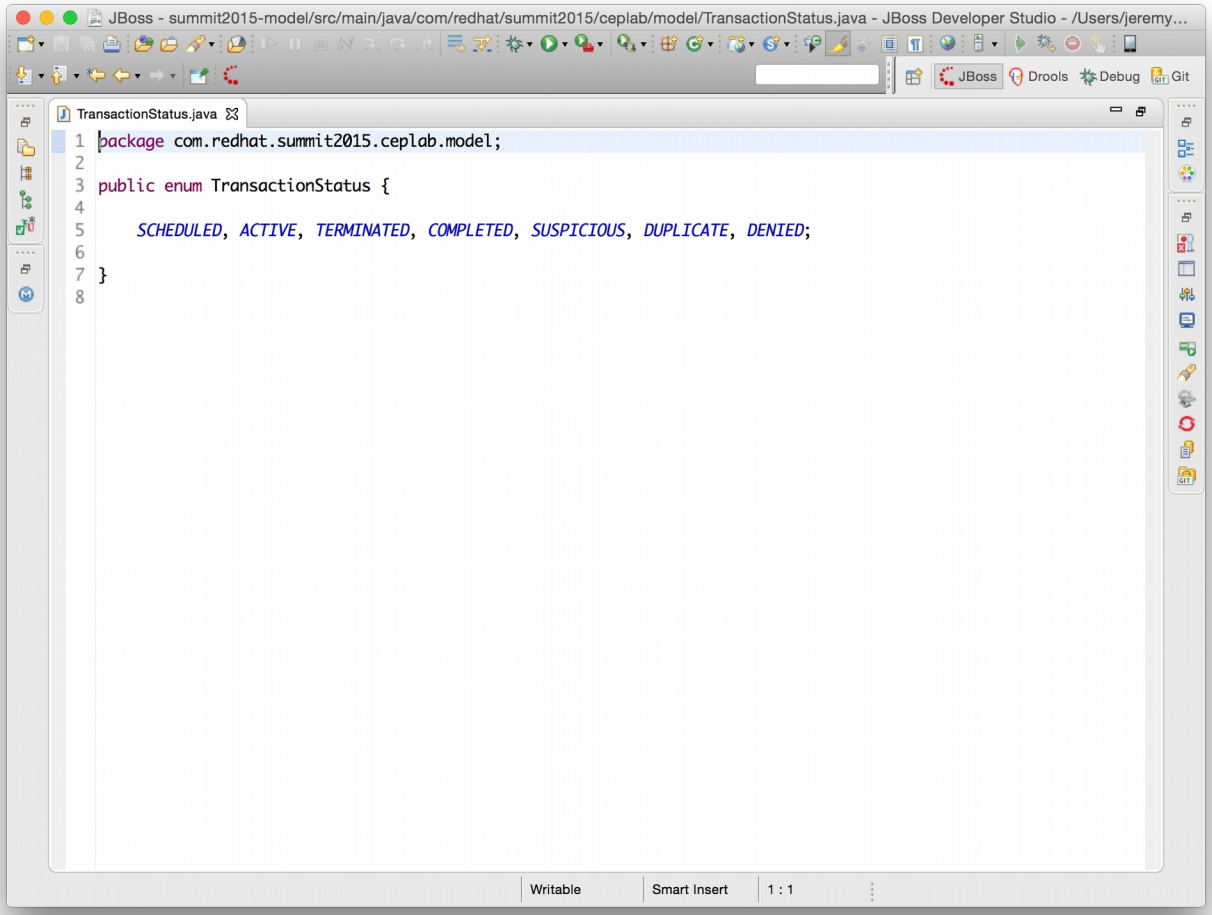
- Account
- AccountStatus
- Transaction
- TransactionStatus

This is obviously an extremely simplistic model; however, it will serve the examples. AccountStatus and TransactionStatus are Enums that we will manipulate as the rules change.


JBoss - summit2015-model/src/main/java/com/redhat/summit2015/ceplab/model/Account.java - JBoss Developer Studio - /Users/jeremydavis/Wor...
JBoss
Drools
Debug
Git
Account.java
package com.redhat.summit2015.ceplab.model;
import java.math.BigDecimal;
public class Account {
private String id;
private AccountStatus status;
private BigDecimal balance = BigDecimal.ZERO;
public Account(AccountStatus status, BigDecimal balance) {
super();
this.status = status;
this.balance = balance;
this.id = String.valueOf(System.nanoTime());
}
public Account() {
super();
}
@Override
public String toString() {
return new ToStringBuilder(this).append("id", id)
.append("status", status)
.append("balance", balance).toString();
};
// --
// generated methods
// --
Writable
Smart Insert
7 : 23


JBoss - summit2015-model/src/main/java/com/redhat/summit2015/ceplab/model/AccountStatus.java - JBoss Developer Studio - /Users/jeremydavi...
JBoss
Drools
Debug
Git
AccountStatus.java
1 package com.redhat.summit2015.ceplab.model;
2
3 public enum AccountStatus {
4
5 PENDING, ACTIVE, TERMINATED, BLOCKED
6
7 }
8
Writable
Smart Insert
1 : 1


JBoss - summit2015-model/src/main/java/com/redhat/summit2015/ceplab/model/Transaction.java - JBoss Developer Studio - /Users/jeremydavis/...
JBoss
Drools
Debug
Git
Transaction.java
package com.redhat.summit2015.ceplab.model;
import java.math.BigDecimal;
public class Transaction {
private String id;
private TransactionStatus status;
private Account fromAccount;
private Account toAccount;
private BigDecimal amount;
public Transaction() {
super();
}
public Transaction(Account fromAccount, Account toAccount, BigDecimal amount) {
super();
this.fromAccount = fromAccount;
this.toAccount = toAccount;
this.amount = amount;
this.status = TransactionStatus.SCHEDULED;
this.id = String.valueOf(System.nanoTime());
}
public Transaction(Account fromAccount, BigDecimal amount){
super();
this.fromAccount = fromAccount;
this.amount = amount;
this.status = TransactionStatus.SCHEDULED;
Writable
Smart Insert
1 : 1


JBoss - summit2015-model/src/main/java/com/redhat/summit2015/ceplab/model/TransactionStatus.java - JBoss Developer Studio - /Users/jeremy...
JBoss
Drools
Debug
Git
TransactionStatus.java
1 package com.redhat.summit2015.ceplab.model;
2
3 public enum TransactionStatus {
4
5 SCHEDULED, ACTIVE, TERMINATED, COMPLETED, SUSPICIOUS, DUPLICATE, DENIED;
6
7 }
8
Writable
Smart Insert
1 : 1

BaseCEPTestCase

NOTE : A SCREENSHOT OF BASECEPTESTCASE IS IN THE FOLLOWING PAGE

BaseCEPTest case is a base class that handles the boilerplate required for interacting with the rules engine.

The important objects to note are :

KieSession

SessionPseudoClock

EntryPoint

Also of note is the **setUp()** method of the class.

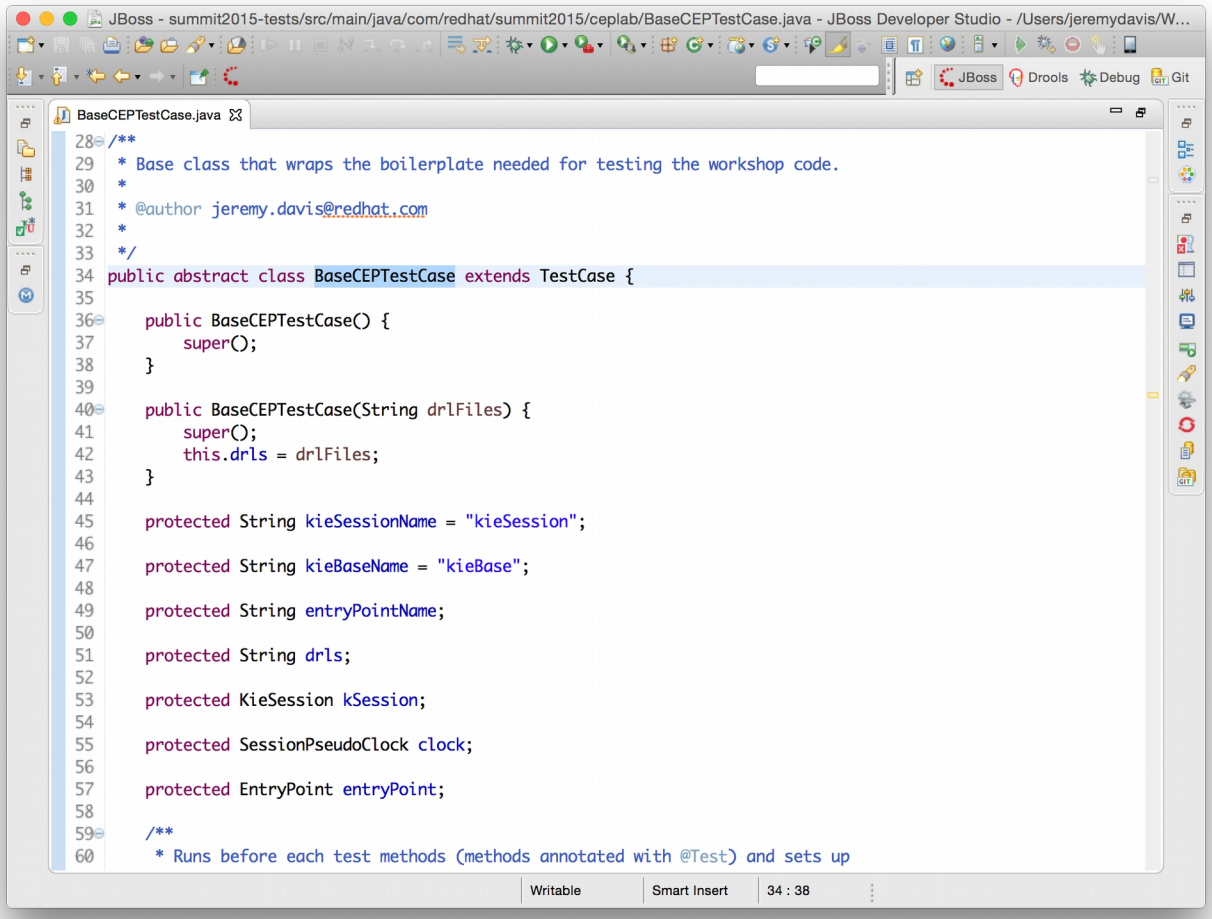
The **KieSession** is the most common way to programmatically interact with the rules engine. A KieSession allows for iterative conversations with the rules engine and keeps state across invocations. Events are inserted into the KieSession. The KieSession then executes the rules. KieSessions can be stateful or stateless. Complex Event Processing uses stateful KieSessions.

The **SessionPseudoClock** is a clock that is controlled programmatically. The SessionPseudoClock contains methods that allow our programs to manipulate the date and time of the KieSession's clock. It is used during testing and development so that we do not have to wait in real time for our rules to fire. It can also be used in production. The reasons are outside the scope of this lab, but feel free to ask the proctors of you would like to discuss CEP deployments in detail!

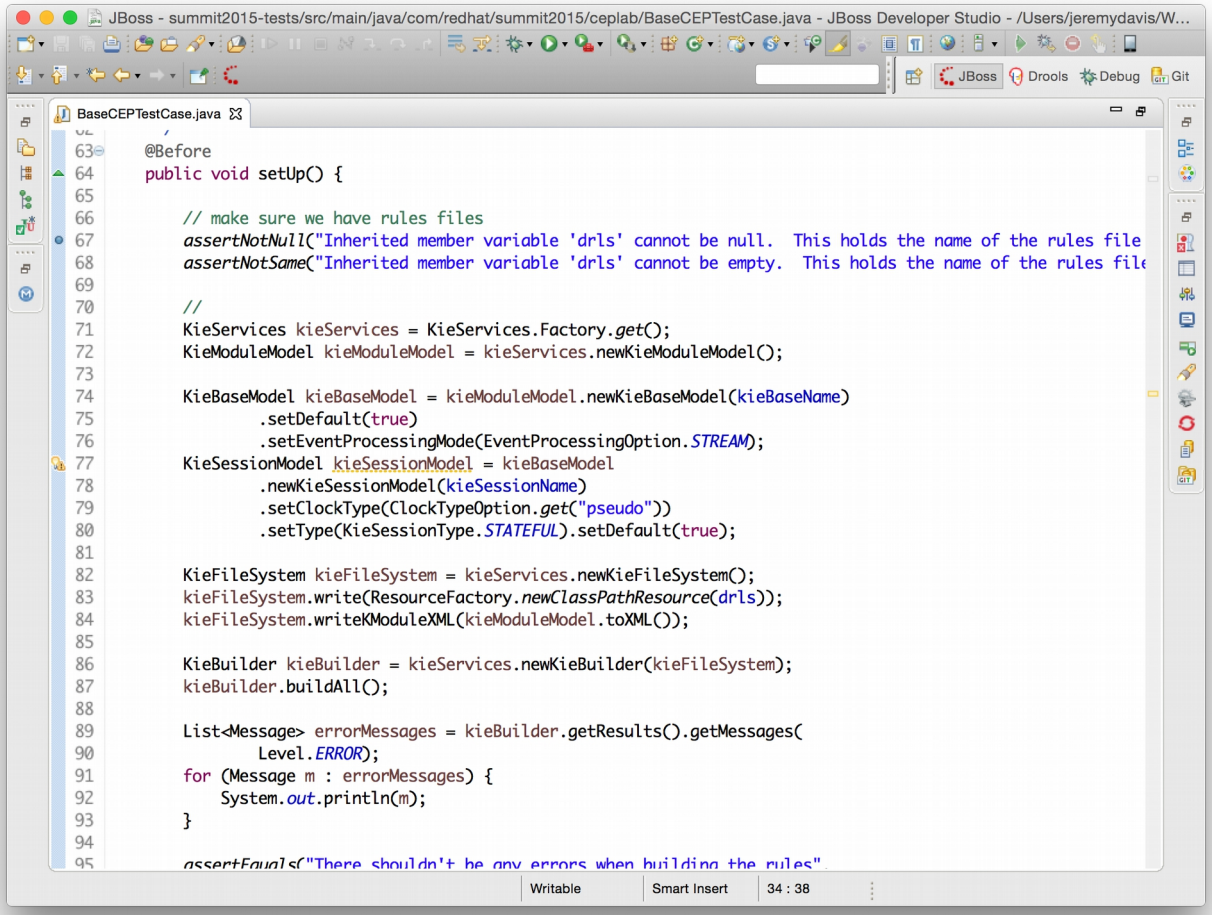
An **EntryPoint** represents a stream through which events enter the engine. Most CEP scenarios use streams. A running engine can contain multiple streams and corresponding EntryPoints.

The **setUp()** method performs the work of initializing the KieSession. The KieSession can discover relevant files from the classpath and is typically configured using an XML file, kmodule.xml, that is included in the META-INF folder of the jar. In this lab we programmatically create the KieSession, lines 81-84, and add the rules files individually.

BaseCEPTestCase



BaseCEPTestCase.setUp()



LAB 1

Summary

1. Import the project lab1
2. Read through the TestCase and the rules file
3. Create a new test and a new rule

NOTE : SCREENSHOTS OF THESE CLASSES ARE IN THE FOLLOWING PAGES.

Import lab1

1. Choose File → Import → Existing Maven Projects
2. Navigate to the “lab1” directory
3. Select the folder, “lab1”
4. Click “Finish”

Lab1Test and lab1rules.drl

- src/main/resources/rules/lab1rules.drl
- src/main/resources/rules/lab1solution.drl.activity
- src/test/java/com/redhat/summit2015/ceplab/Lab1Test.java
- src/test/java/com/redhat/summit2015/ceplab/Lab1TestSolution.java

We won't be doing any Complex Event Processing in this lab. This lab is designed to show the basics of the rules engine. The rules file and unit test ending in “solution” contain solutions to the activity. Ignore those files for now.

Lab1Test

Important things to note :

2 Account objects are inserted into the KieSession, line 41

The KieSession.fireAllRules() method is called to perform analysis on the objects

This test class uses a KieClasspathContainer which loads all rules on the classpath into the KieSession. This approach is different from subsequent tests which will specify the rules files by name.

lab1rules.drl

The local variable $account is assigned to each Fact in memory, line 13.

The logic that matches Facts to this rule are in the parens, line 13.

Activity

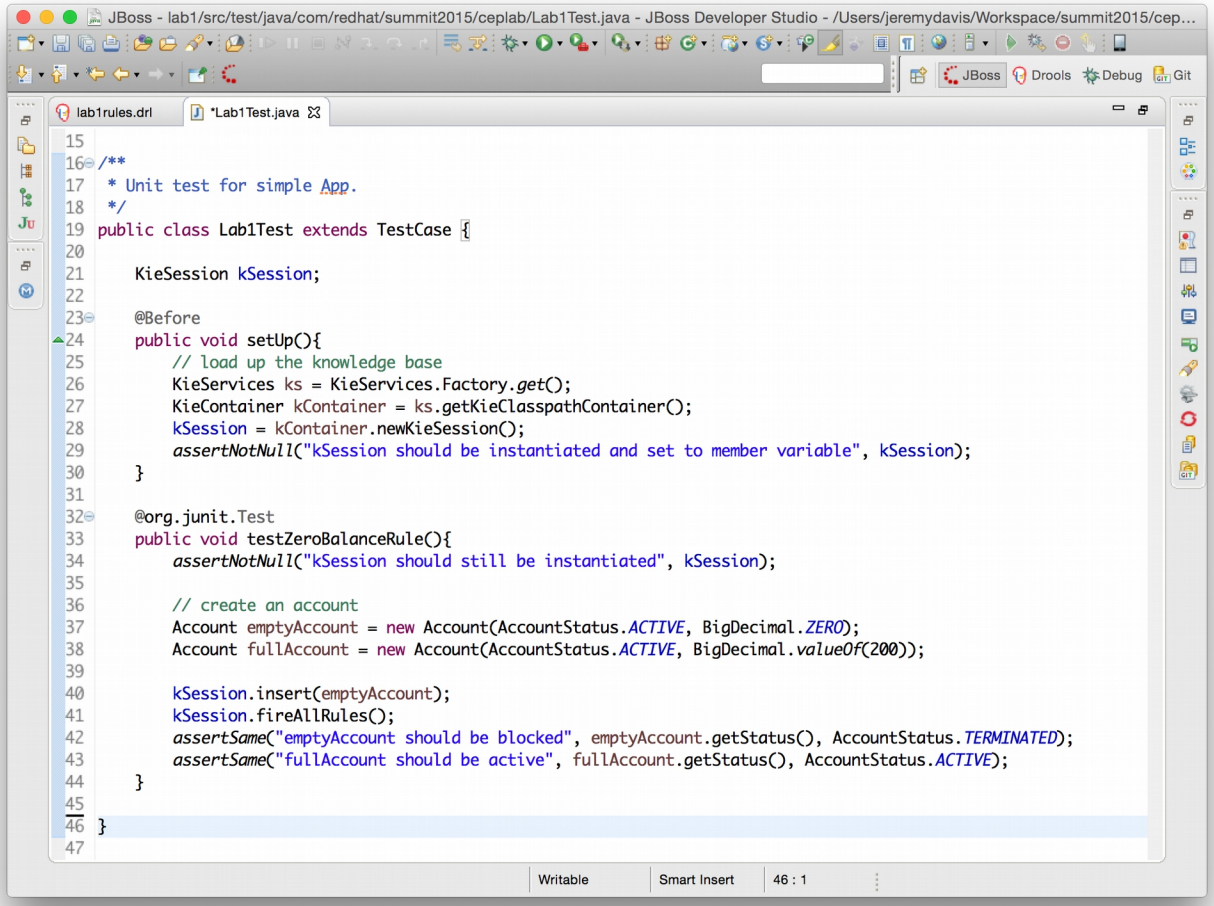
Create a test method that blocks accounts with a balance under $100. Add a rule that passes this test.

HINT : Be careful not to conflict with the existing test that terminates accounts with a zero balance.

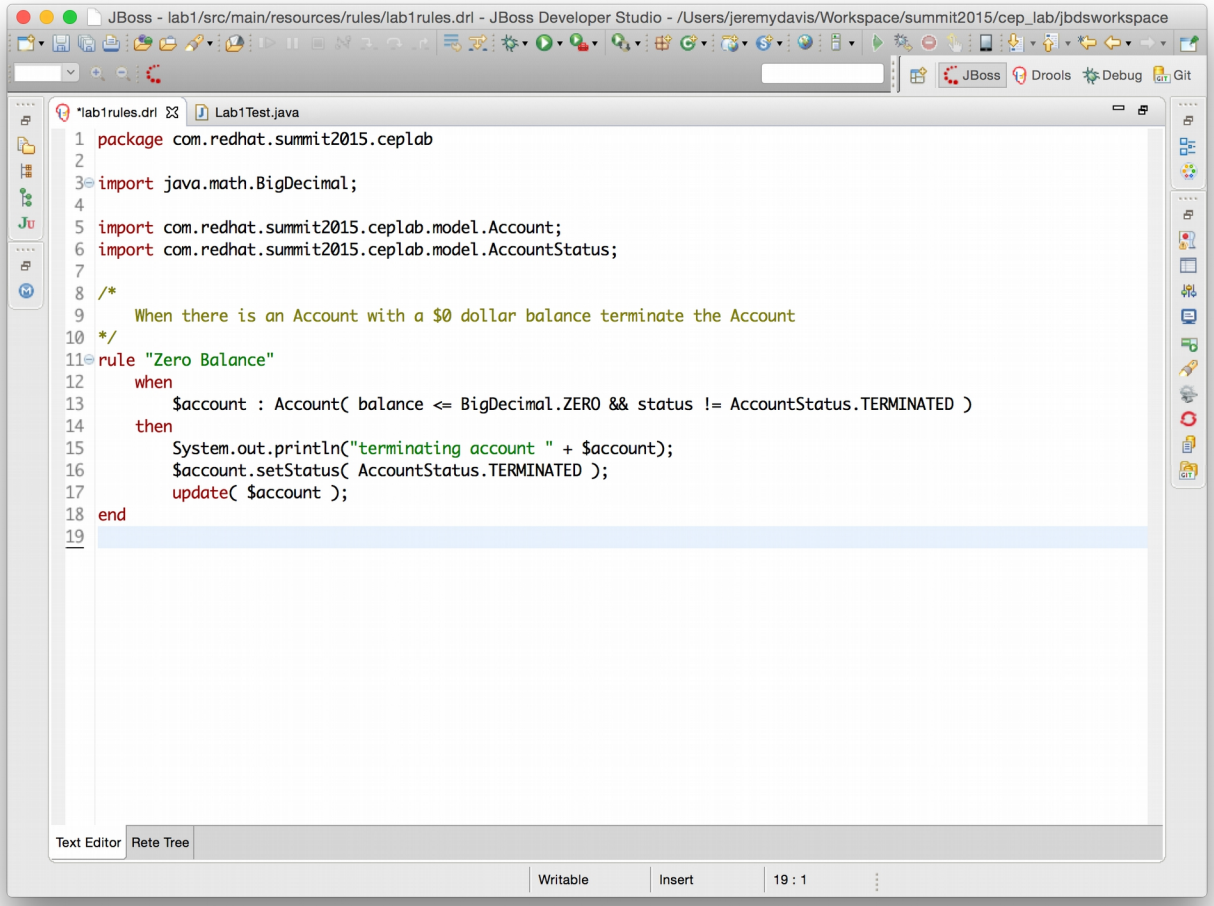
BEST PRACTICE : START YOUR RULE BY WRITING THE RULE IN NATURAL LANGUAGE IN A COMMENT

The TestCase, Lab1TestSolution, contains a possible solution. This test should fail initially. The drl file, “lab1solution.drl.activity” contains a solution for this activity. To use this rule rename it, removing “activity.” In this example the rules engine is picking up all rules files on the classpath so the extension “.activity” prevents the solution from being included.

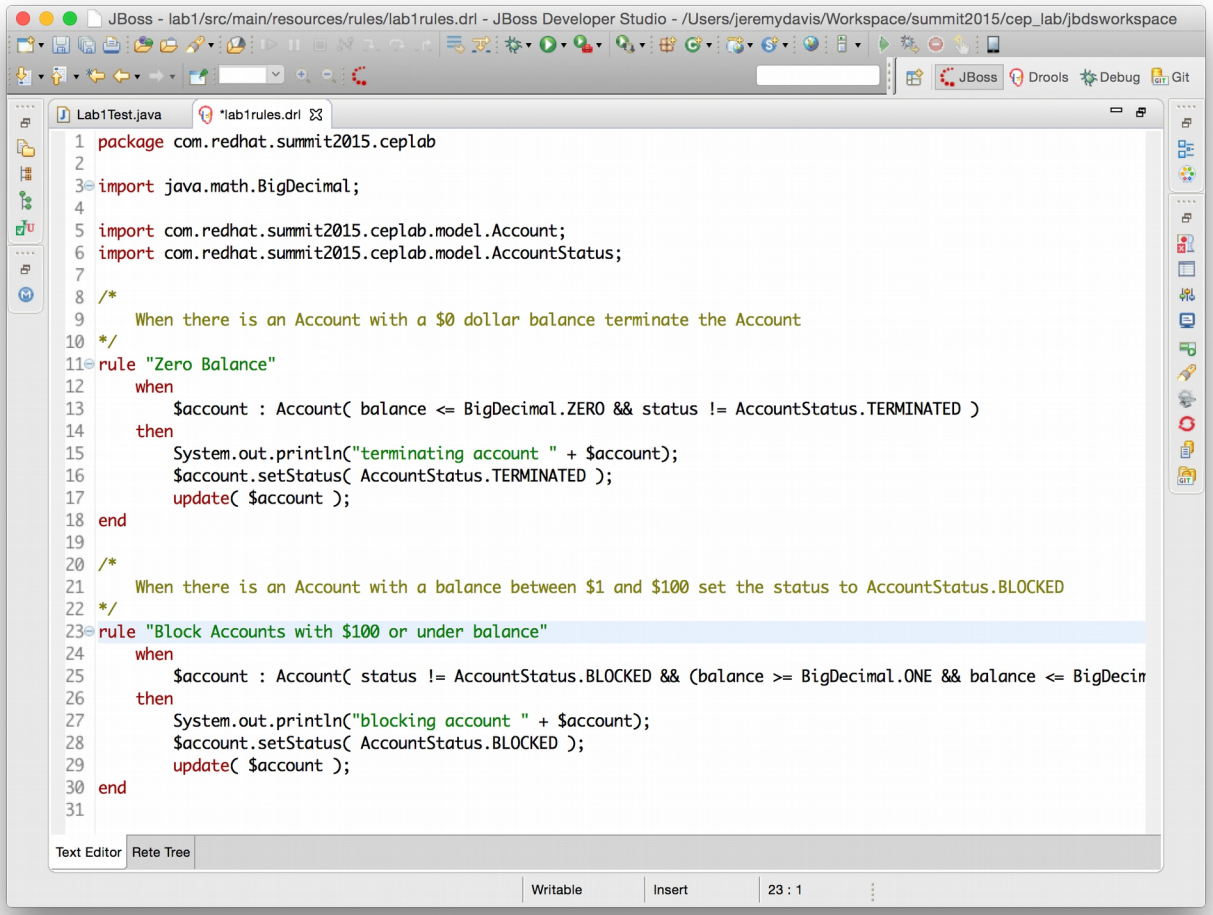
Lab1Test.java



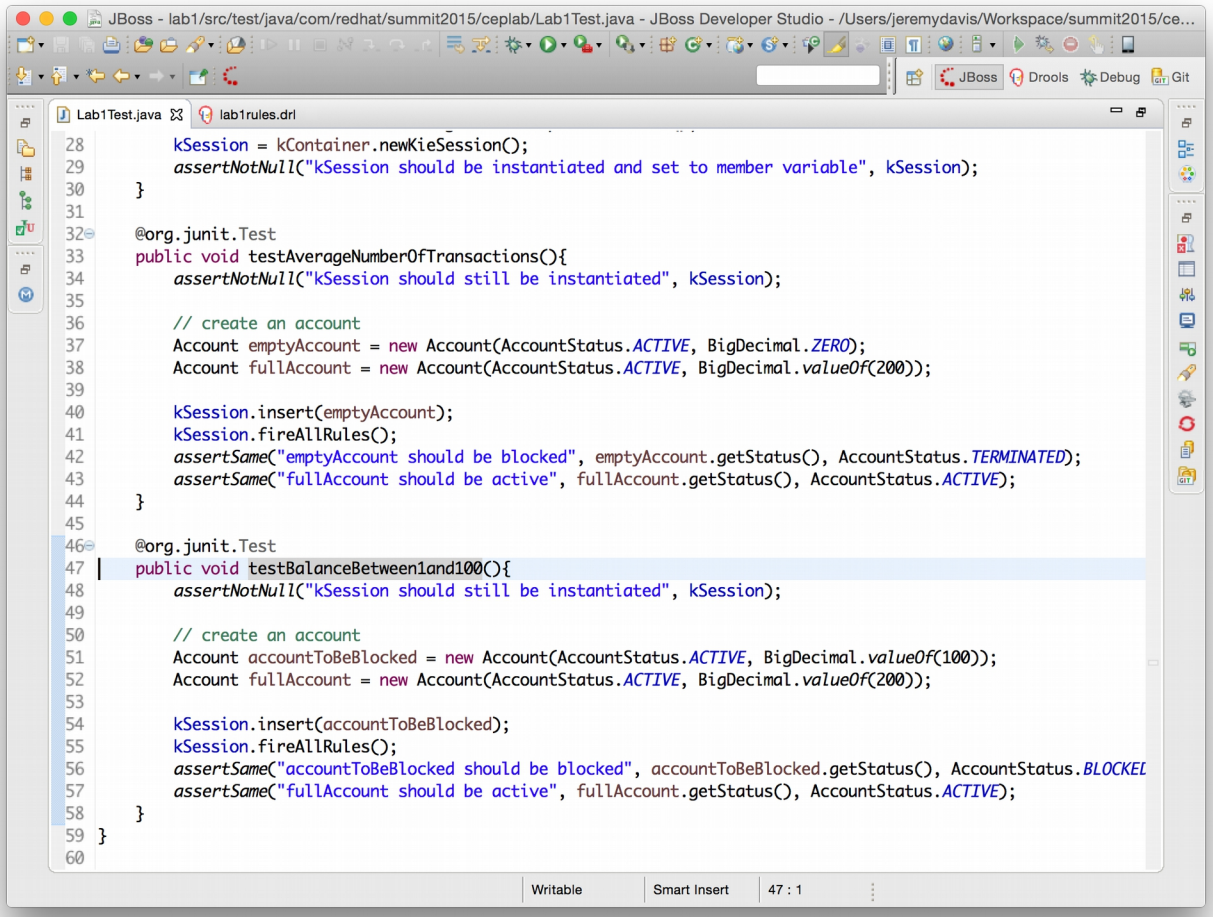
lab1rules.drl



Activity Solution : lab1rules.drl



Lab 1 Activity Solution : Lab1Test



LAB 2 : INTRODUCING TEMPORAL CORRELATION

Summary

1. Import the project lab2
2. Read through the TestCase and the rules file
3. Create a new test and a new rule

NOTE : SCREENSHOTS OF THESE CLASSES ARE IN THE FOLLOWING PAGES.

Import lab2

1. Choose File → Import → Existing Maven Projects
2. Navigate to the “lab2” directory
3. Select the folder, “lab2”
4. Click “Finish”

Lab2Test and lab2rules.drl

- src/main/resources/rules/lab2rules.drl
- src/main/resources/rules/lab2solution.drl
- src/test/java/com/redhat/summit2015/ceplab/Lab2Test.java
- src/test/java/com/redhat/summit2015/ceplab/Lab2TestSolution.java

In this lab we will begin Complex Event Processing.

Lab2Test

Important things to note :

We extend BaseCEPTestCase

lab2rules.drl

The Transaction Fact is declared as an Event, line 8

```
7
8  declare Transaction
9      @role( event )
10 end
11
```

The Transaction Events are inserted from the “Transfers” stream. This prevents unrelated Transaction Events from being evaluated by this rule.

```
19          $t1 : Transaction( $fromAccount : fromAccount ) from entry-point Transfers
20          $t2 : Transaction( this != $t1, fromAccount == $fromAccount, this after [0s, 15s] $t1 )
21              from entry-point Transfers
```

The second Transaction Event must occur within 15 seconds of the first.

```
19          $t1 : Transaction( $fromAccount : fromAccount )  from entry-point Transfers
20          $t2 : Transaction( this != $t1, fromAccount == $fromAccount, this after [0s, 15s] $t1 )
21              from entry-point Transfers
```

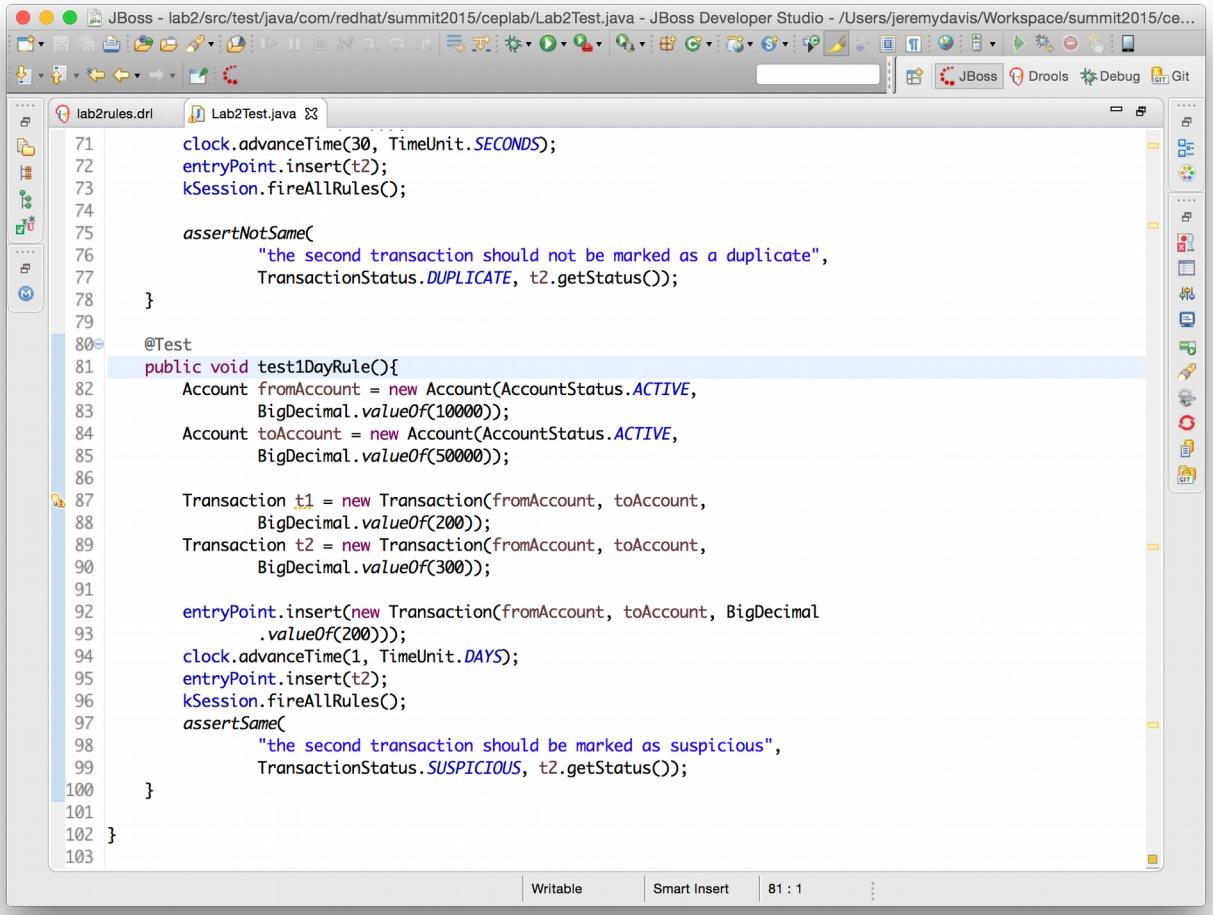
Activity

Create a test method and a rule that flags 2 transfers from the same account within 1 day as suspicious.

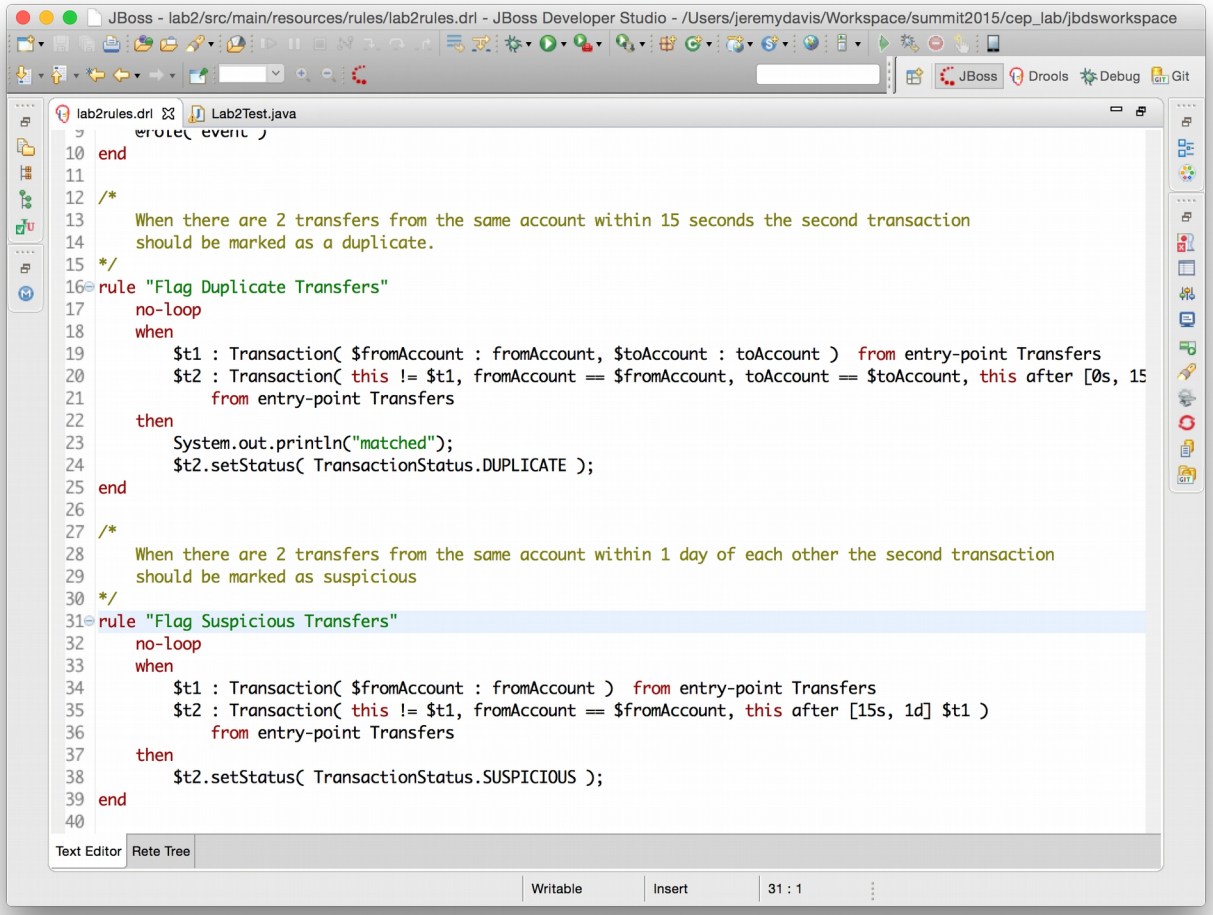
HINT : Be careful not to conflict with the existing test.

BEST PRACTICE : START YOUR RULE BY WRITING THE RULE IN NATURAL LANGUAGE IN A COMMENT

Lab 2 Activity Solution : Lab2Test.java



Lab 2 Activity Solution : lab2rules.drl



LAB 3 : DECLARING FACTS AND @PROPERTYREACTIVE

Summary

1. Import the project lab3
2. Read through the TestCase and the rules file
3. Create a new test and a new rule

NOTE : SCREENSHOTS OF THESE CLASSES ARE IN THE FOLLOWING PAGES.

Import lab3

1. Choose File → Import → Existing Maven Projects
2. Navigate to the “lab3” directory
3. Select the folder, “lab3”
4. Click “Finish”

Lab3Test and lab3rules.drl

- src/main/resources/rules/lab3rules.drl
- src/main/resources/rules/lab3solution.drl
- src/test/java/com/redhat/summit2015/ceplab/Lab3Test.java
- src/test/java/com/redhat/summit2015/ceplab/Lab3TestSolution.java

In this lab we will see some new rule techniques.

Lab3

Important things to note :

Declaring Facts and Events within a drl file

We have a Fact (Java POJO) , AccountInfo, declared inside the rule file. This is useful when there are objects that are not part of the domain but are necessary for executing the rules. The AccountInfo stores average monthly information about an Account. This information is necessary for executing the rules, but is not an important part of the domain. By declaring the AccountInfo Fact in the rules file it can exist in the KieSession longer than individual Transaction Events.

```
12
13 declare AccountInfo
14     @propertyReactive
15     id : String
16     averageBalance : BigDecimal
17     averageAmount : BigDecimal
18     averageNumberOfTransactions : BigDecimal
19     numberOfTransactions1Day : Long
20 end
21
```

FactTypes

A FactType allows us to create an instance of the AccountInfo. In Lab3Test.java we will create a local instance of AccountInfo in order to set values that would otherwise have to be calculated over time.

```
40        FactType accountInfoFactType = kSession.getKieBase().getFactType("com.redhat.summit2015.ceplab", "Accour
41        Object accountInfo = accountInfoFactType.newInstance();
42        accountInfoFactType.set(accountInfo, "averageBalance",
43                BigDecimal.valueOf(1000));
44        accountInfoFactType.set(accountInfo, "id", account.getId());
45        FactHandle accountInfoHandle = kSession.insert(accountInfoFactType);
46        kSession.update(accountInfoHandle, accountInfo);
```

Calling newInstance() on the FactType creates a local instance that we can insert into the KieSession's working memory.

```
40        FactType accountInfoFactType = kSession.getKieBase().getFactType("com.redhat.summit2015.ceplab", "Accour
41        Object accountInfo = accountInfoFactType.newInstance();
42        accountInfoFactType.set(accountInfo, "averageBalance",
43                BigDecimal.valueOf(1000));
44        accountInfoFactType.set(accountInfo, "id", account.getId());
45        FactHandle accountInfoHandle = kSession.insert(accountInfoFactType);
46        kSession.update(accountInfoHandle, accountInfo);
```

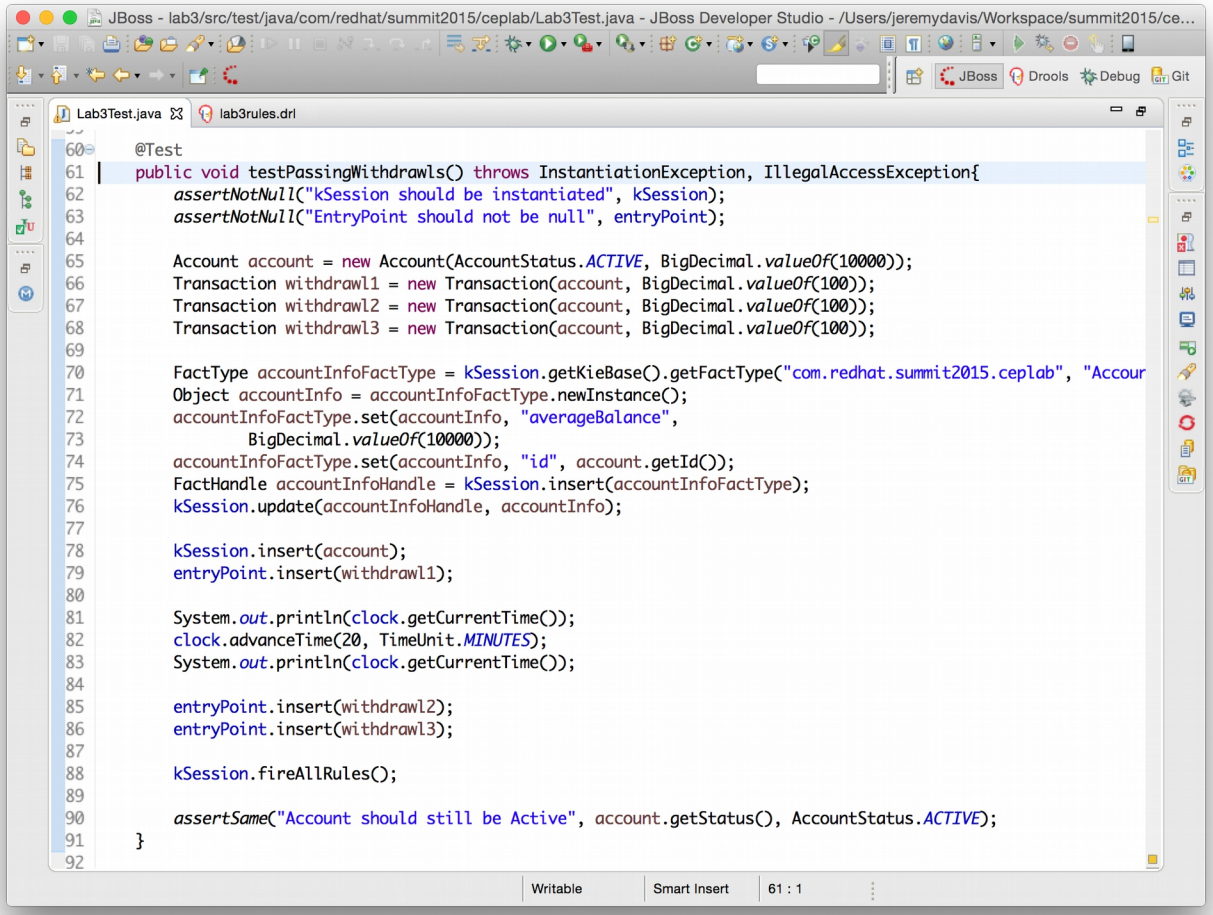
A FactHandle will allow us to keep track of the FactType. It is the return type when inserting a FactType into the KieSession.

```
40        FactType accountInfoFactType = kSession.getKieBase().getFactType("com.redhat.summit2015.ceplab", "Accour
41        Object accountInfo = accountInfoFactType.newInstance();
42        accountInfoFactType.set(accountInfo, "averageBalance",
43                BigDecimal.valueOf(1000));
44        accountInfoFactType.set(accountInfo, "id", account.getId());
45        FactHandle accountInfoHandle = kSession.insert(accountInfoFactType);
46        kSession.update(accountInfoHandle, accountInfo);
```

Activity

Using a FactType and a FactHandle create a test that verifies 3 valid transactions will pass the exising rule. One possible solution is on the next page.

Lab 3 Activity Solution



LAB 4 : SLIDING WINDOWS

Summary

1. Import the project lab4
2. Read through the TestCase and the rules file
3. Create a new test and a new rule

NOTE : SCREENSHOTS OF THESE CLASSES ARE IN THE FOLLOWING PAGES.

Import lab4

1. Choose File → Import → Existing Maven Projects
2. Navigate to the “lab4” directory
3. Select the folder, “lab4”
4. Click “Finish”

Lab4Test and lab4rules.drl

- src/main/resources/rules/lab4rules.drl
- src/main/resources/rules/lab4solution.drl
- src/test/java/com/redhat/summit2015/ceplab/Lab4Test.java
- src/test/java/com/redhat/summit2015/ceplab/Lab4TestSolution.java

This lab introduces sliding windows. Sliding windows allow scoping Events of interest to a constantly moving group. There are two types of sliding windows: length and time. Lab 4 introduces length and Lab 5 contains a temporal example.

Lab4

lab4rules.drl note:

The four most recent transactions are being evaluated regardless of when they occurred.

```
21      when
22          $average: Number() from accumulate (
23              Transaction( $amount: amount    ) over window:length ( 4 ) from entry-point CreditCard,
24              average($amount))
25          $transaction: Transaction( amount > ($average * 2)  ) from entry-point CreditCard
26      then
27          $transaction.setStatus(TransactionStatus.DENIED);
```

The Drools method “from accumulate” is being called to collect all matching Events from the KieSession.

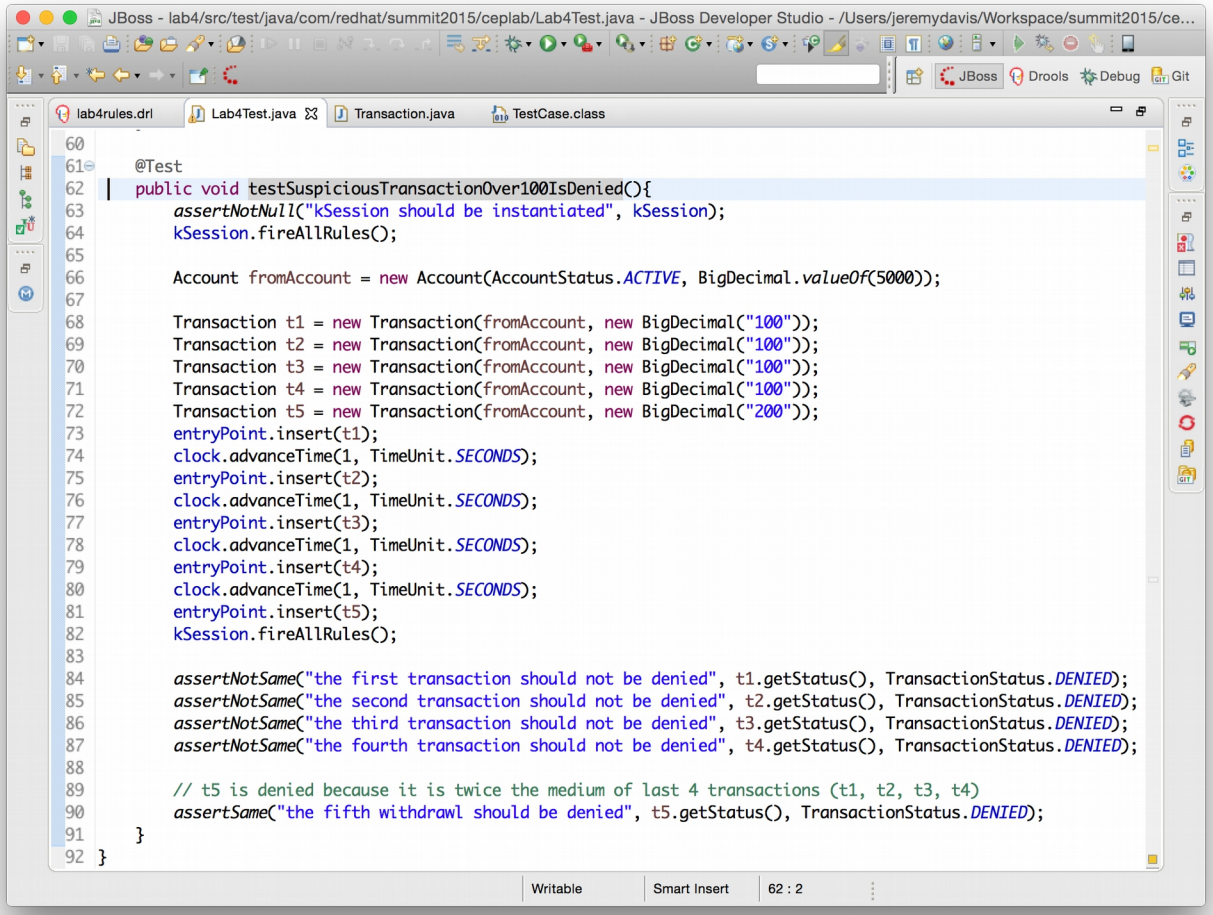
```
21      when
22          $average: Number() from accumulate (
23              Transaction( $amount: amount    ) over window:length ( 4 ) from entry-point CreditCard,
24              average($amount))
25          $transaction: Transaction( amount > ($average * 2)  ) from entry-point CreditCard
26      then
27          $transaction.setStatus(TransactionStatus.DENIED);
```

Activity

Create a test method that checks for a Transaction that is greater than the average of the previous 4 transactions when all Transacitons are over $100.

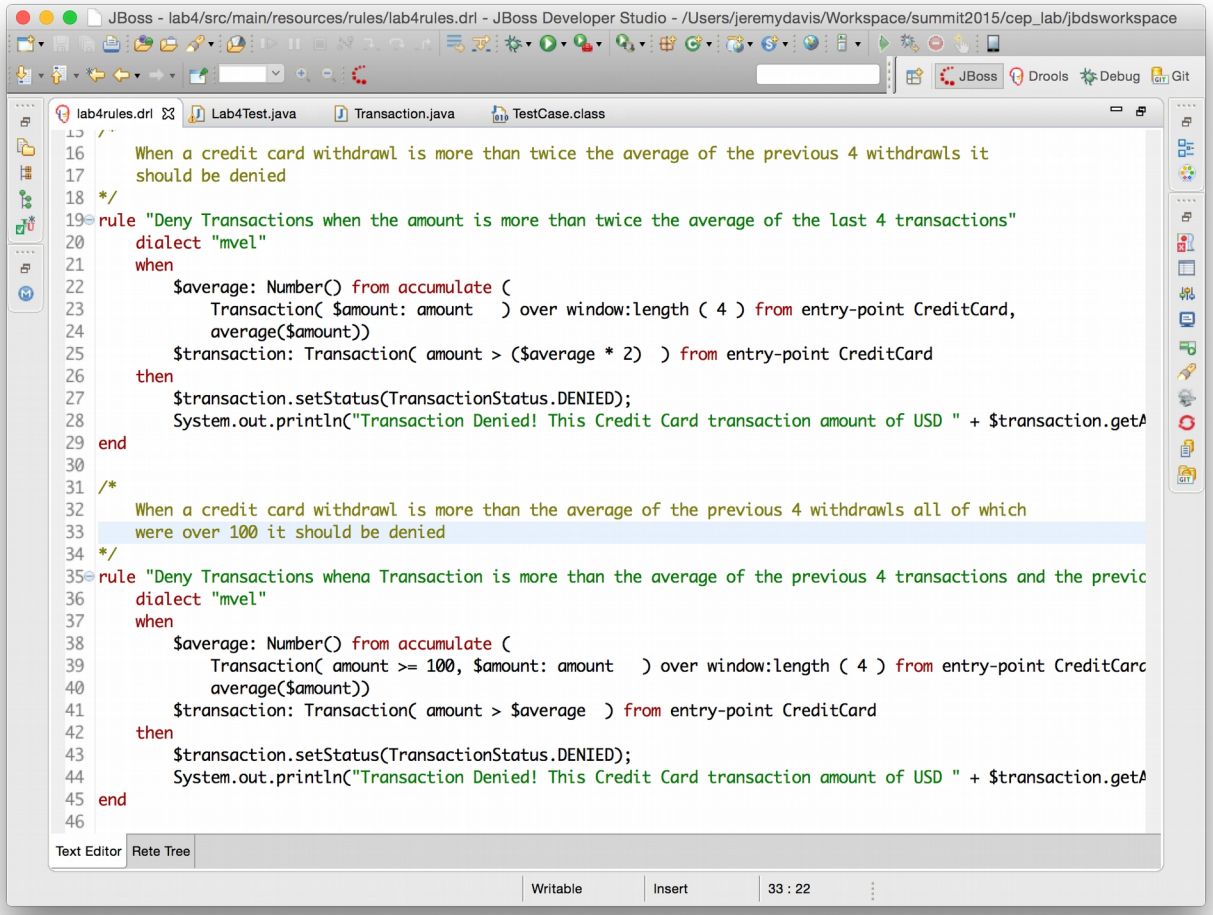
Create a rule that passes the tes.

Lab 4 Activity Solution: Lab4Test




```
60
61      @Test
62      public void testSuspiciousTransactionOver100IsDenied(){
63          assertNotNull("kSession should be instantiated", kSession);
64          kSession.fireAllRules();
65
66          Account fromAccount = new Account(AccountStatus.ACTIVE, BigDecimal.valueOf(5000));
67
68          Transaction t1 = new Transaction(fromAccount, new BigDecimal("100"));
69          Transaction t2 = new Transaction(fromAccount, new BigDecimal("100"));
70          Transaction t3 = new Transaction(fromAccount, new BigDecimal("100"));
71          Transaction t4 = new Transaction(fromAccount, new BigDecimal("100"));
72          Transaction t5 = new Transaction(fromAccount, new BigDecimal("200"));
73          entryPoint.insert(t1);
74          clock.advanceTime(1, TimeUnit.SECONDS);
75          entryPoint.insert(t2);
76          clock.advanceTime(1, TimeUnit.SECONDS);
77          entryPoint.insert(t3);
78          clock.advanceTime(1, TimeUnit.SECONDS);
79          entryPoint.insert(t4);
80          clock.advanceTime(1, TimeUnit.SECONDS);
81          entryPoint.insert(t5);
82          kSession.fireAllRules();
83
84          assertNotSame("the first transaction should not be denied", t1.getStatus(), TransactionStatus.DENIED);
85          assertNotSame("the second transaction should not be denied", t2.getStatus(), TransactionStatus.DENIED);
86          assertNotSame("the third transaction should not be denied", t3.getStatus(), TransactionStatus.DENIED);
87          assertNotSame("the fourth transaction should not be denied", t4.getStatus(), TransactionStatus.DENIED);
88
89          // t5 is denied because it is twice the medium of last 4 transactions (t1, t2, t3, t4)
90          assertSame("the fifth withdrawl should be denied", t5.getStatus(), TransactionStatus.DENIED);
91      }
92  }
```

Lab 4 Activity Solution: lab4rules.drl



LAB 5

Summary

1. Import the project lab
2. Read through the TestCase and the rules file
3. Experiment!

NOTE : SCREENSHOTS OF THESE CLASSES ARE IN THE FOLLOWING PAGES.

Import lab5

1. Choose File → Import → Existing Maven Projects
2. Navigate to the “lab5” directory
3. Select the folder, “lab5”
4. Click “Finish”

Lab5Test and lab5rules.drl

- src/main/resources/rules/lab5rules.drl
- src/test/java/com/redhat/summit2015/ceplab/Lab5Test

This lab introduces temporal sliding windows.

lab5rules.drl

The sliding windows in this lab are temporal

```
20        $average: Number() from accumulate (
21            Transaction( $amount: amount   ) over window:time ( 30d ) from entry-point CreditCard,
22            average($amount))
23        $transaction: Transaction( amount > $average  ) from entry-point CreditCard
```

Activity

Experiment! Create some tests, change rules around, and ask for help if you get stuck.

5. **[student@serverX ~]$ sudo run-some-command**

Show any expected output

...

...

6. Instruction two

   [student@serverX ~]$ echo huzzah

   huzzah

   More detail on output or command that was run to help it make sense to people.

7. MOAR INSTRUCTIONS!!!