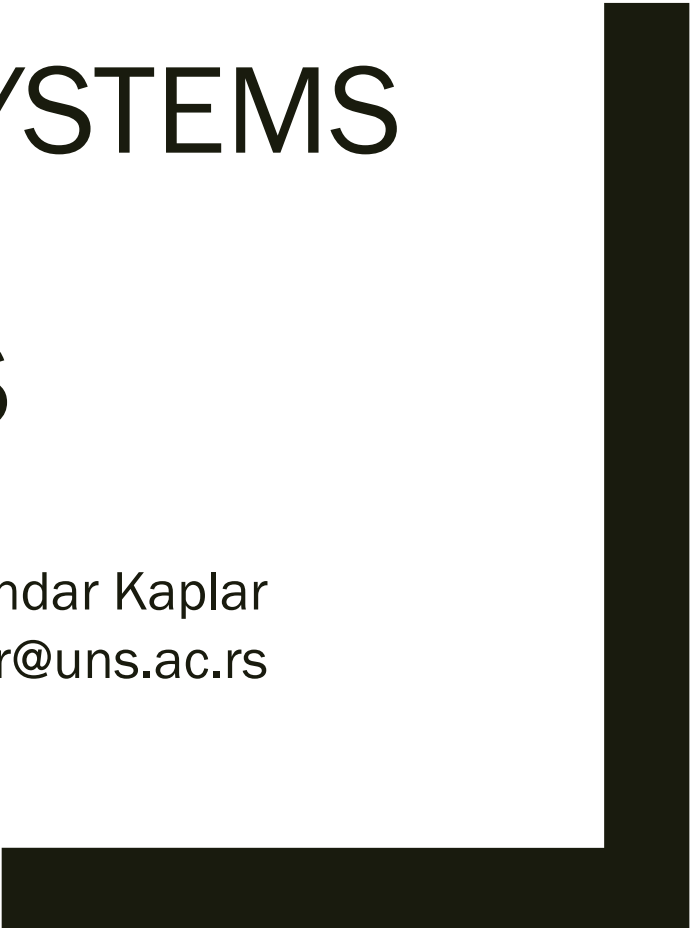




# KNOWLEDGE BASED SYSTEMS

## DROOLS - BRMS

Aleksandar Kaplar  
[aleksandar.kaplar@uns.ac.rs](mailto:aleksandar.kaplar@uns.ac.rs)



# Sadržaj

- Ekspertski sistemi
- Ekspertski sistemi bazirani na pravilima
- Uvod u Drools

# EXPERT SYSTEMS



# Ekspertski sistemi

- grana veštačke inteligencije
- objašnjavaju i definišu rešenja na način blizak korisniku
- osnovna ideja je napraviti inteligentni program, tako što bi program inkorporirao esencijalno domen specifično znanje iz oblasti rešavanja problema
- simuliraju ljudsko rezonovanje u određenom domenu

# Tipovi

- Neural Networks
- Blackboard Systems
- Belief (Bayesian) Networks
- ...
- Case-Based Reasoning
- Rule-Based Systems

# Ekspertski sistemi bazirani na pravilima (Rule Based Expert Systems)

- softver koji sadrži domen specifično znanje ljudi/eksperata
- softver koji pokušava da reši problem tako što će ukloni neodređenosti na način kao što bi to radili stručnjaci koristeći svoje znanje/iskustvo iz određenog domena.

# RULE BASED EXPERT SYSTEMS

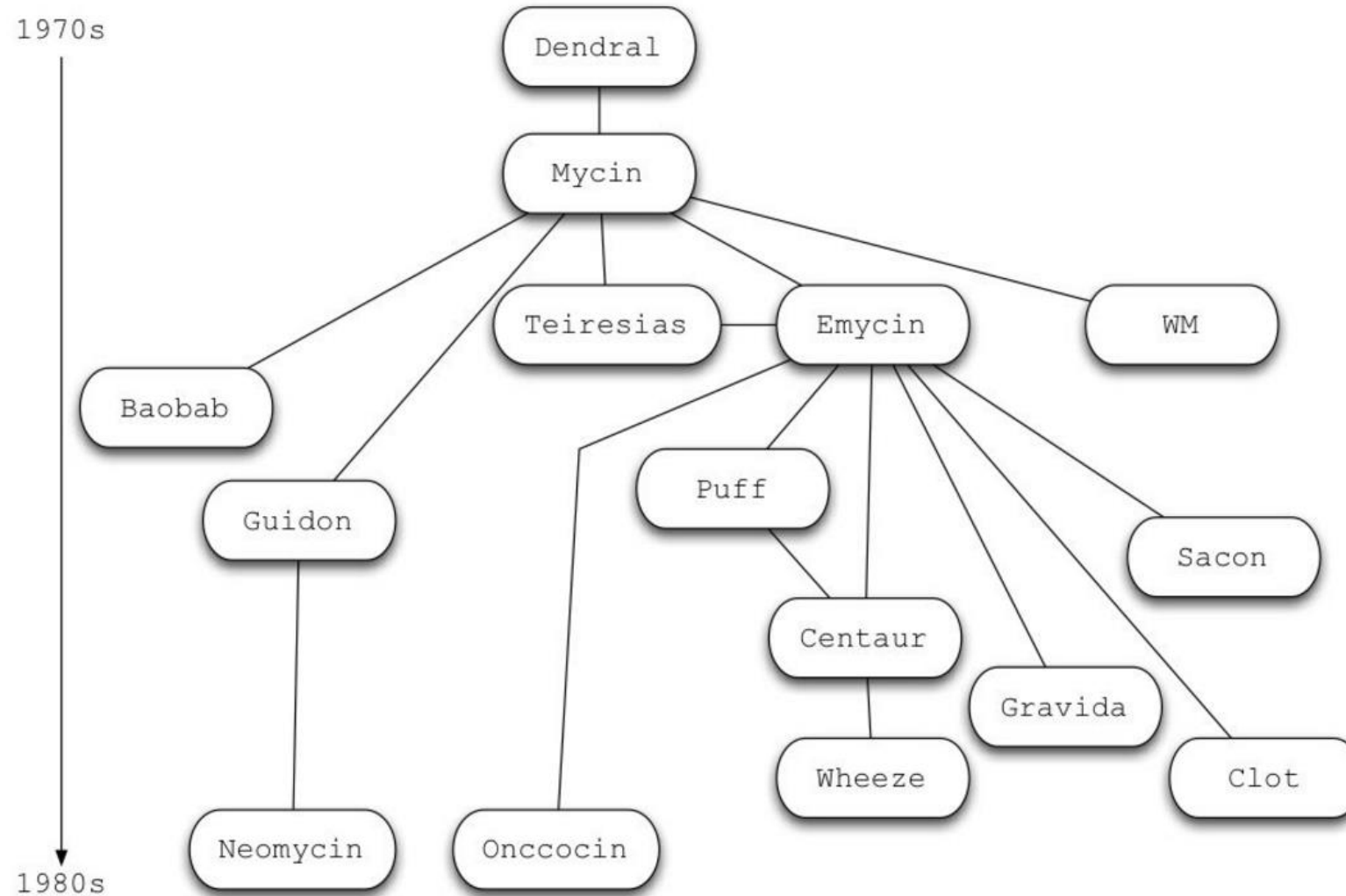


# Rule Based Expert Systems

- nastali tokom istraživanja veštačke inteligencije krajem 70 i početkom 80
- ekspertski sistem koji koristi pravila i činjenice za donošenje odluka
- podaci o problemu skladište se kao činjenice (**facts**)
- rezonovanje primenom IF...THEN...ELSE pravila (**rules**)
- namenjeni ekspertima



# Rule Based Expert Systems



# Kada ih koristiti

- Domen problema – uzak, jasan i dobro definisan
- Predstva znanja – činjenice i pravila nad njima
- Izlaz sistema – preporuke
- Obrazloženje – evidencija/trag okidanja pravila
- Mogućnost učenja – ne postoji

# Rezonovanje unapred i unazad

- podržavaju rezonovanje dedukcijom (forward-chaining) ili indukcijom (backward-chaining).
- forward chaining je vođeno podacima, automatsko (prepoznaš činjenicu pa doneseš zaključak, ne zna se koji se problem unapred rešava)
- backward chaining is vođeno ciljem, rešavanje tačno određenog problema

# Forward-chaining

- osnovna ideja - izvršiti sva pravila čije pretpostavke su zadovoljene, rezultat smestiti u bazu zanja
  - $F1:A$
  - $F2:B$
  - $R1:P \rightarrow Q$
  - $R2:L \wedge M \rightarrow P$
  - $R3:B \wedge L \rightarrow M$
  - $R4:A \wedge B \rightarrow L$

# Backward-chaining

- osnovna ideja - idi unazad od  $Q$  i dokaži da su ispunjene sve pretpostavke za pravilo koje rezultuje sa  $Q$ .
- Ukoliko pretpostavka nije ispunjena definiši je kao novi pod-cilji koji treba dokazati nekim pravilom.
- Izbegavaj petlje i ne ponavljaj dokazivanje pretpostavki koje su već proverene.

# Rule Based vs Konvencionalni sistemi

- Konvencionalni tradicionalni sistemi izvršavaju definisan algoritam
- Ekspertski sistemi oslanjaju se na bazu znanja (**knowledge base**) i na rezoner (**rule engine**) koji uvek izvršava određene procedure rezonovanja koristeći bazu znanja i radnu memoriju (**facts**) kao ulaz

# Zašto ih koristiti

- Stručnjaci nisu uvek dostupni za rešavanje problema
- Stručnjaci nisu 100% pouzdani i konzistentni pri rešavanju problema
- Stručnjaci nekada ne znaju da objasne svoje odluke
- Cena angažana stručnjaka je previsoka

# Ograničenja

- Domen upotrebe je ograničen na usko specifičnu oblast
- Kontinualno ih treba prilagođavati tako da odgovaraju novonastaloj situaciji, ne mogu sami da uče
- Eksperti trebaju da održavaju sistem
- Nedostaje jasna logika izvršavanja programa kao kod tradicionalnih sistema



# Pravilo iz MYCIN projekta

## ■ if

1. *the infection is primary bacteremia, and*
2. *the site of the culture is one of the sterile sites, and*
3. *the suspected portal of entry of the organism is the gastrointestinal tract*

## ■ then

- *there is suggestive evidence (0.7) that the identity of the organism is bacteroides.*

# Konvencionalno VS Deklarativno programiranje

- Konvencionalno programiranje
  - *namenjeno programerima*
  - *definisano kroz programsku implementaciju algoritma*
  - *tok izvršenja programa je sekvencijalni i deterministički*
  - *program upravlja podacima*

# Konvencionalno VS Deklarativno programiranje

## ■ Deklarativno programiranje

- *namenjeno ekspertima (nisu programeri)*
- *definisano kroz formalno navođenje tvrdnji i činjenica*
- *rule-based kod blisko odgovara formatu reprezentacije domenskog problema (pravilima i činjenicama)*
- *ne može se upravljati tokom izvršavanja*
- *program upravlja znanjem*

# Eksperiment

- Zamislite da je potrebno da napišete OOP program za rešavanje papirne slagalice od preko 500+ delova
- Polimorfizam bi imao problem
- Ipak ljudi mogu da reše problem

# Eksperiment

- Naš mozak instiktivno počinje da primenjuje pravila koja vode ka rešavanju slagalice

```
(corner_found  
(piece_is_corner)  
=>  
(assert corner-found)  
(save_piece))
```

postavi parče na ugao

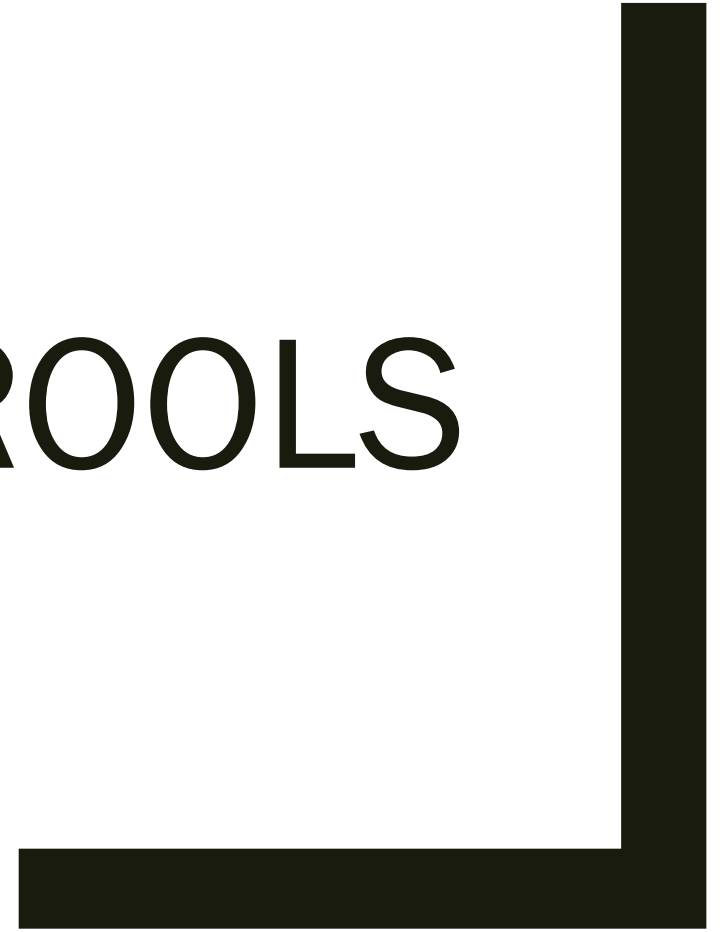
```
(edge_found  
(piece_is_edge)  
=>  
(assert edge-found)  
(save_piece))
```

postvi parče na ivicu ili  
ga sačuvaj za kasnije

# Eksperiment

- Naš mozak traži paterne (**pattern-matches**), postvalja prioritete (sets **agenda**) i primenjuje pravila (**rule engine**) nad podacima

DROOLS

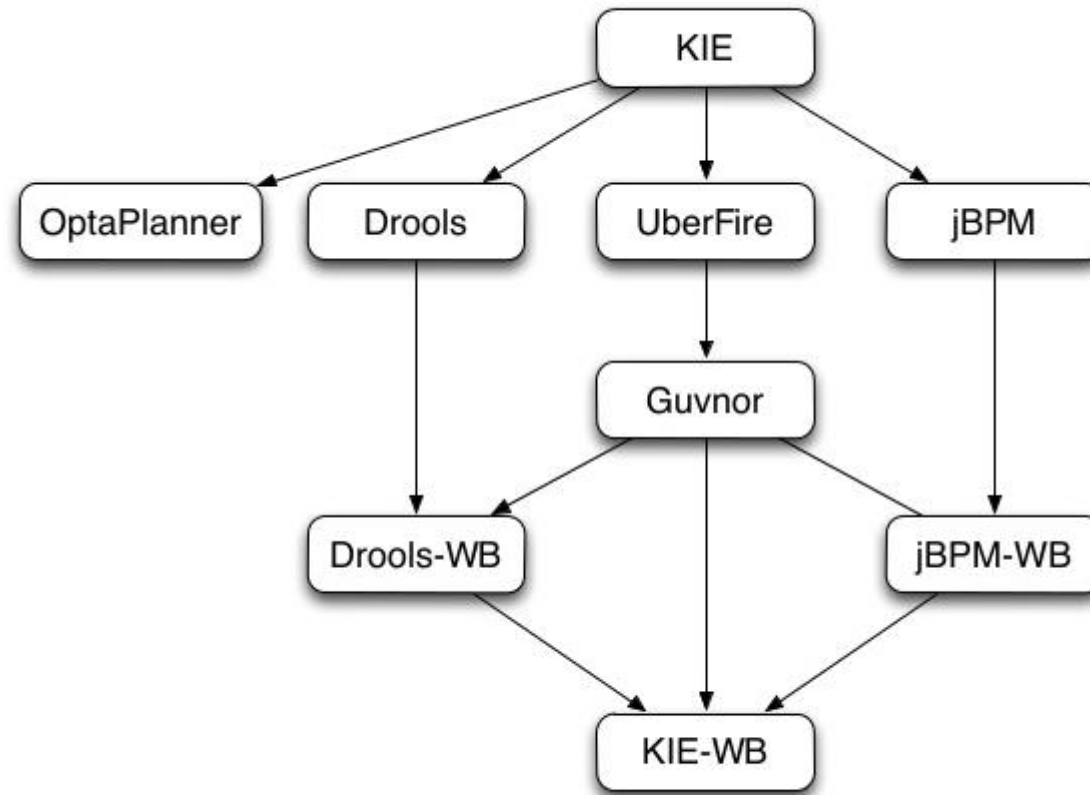


# DROOLS

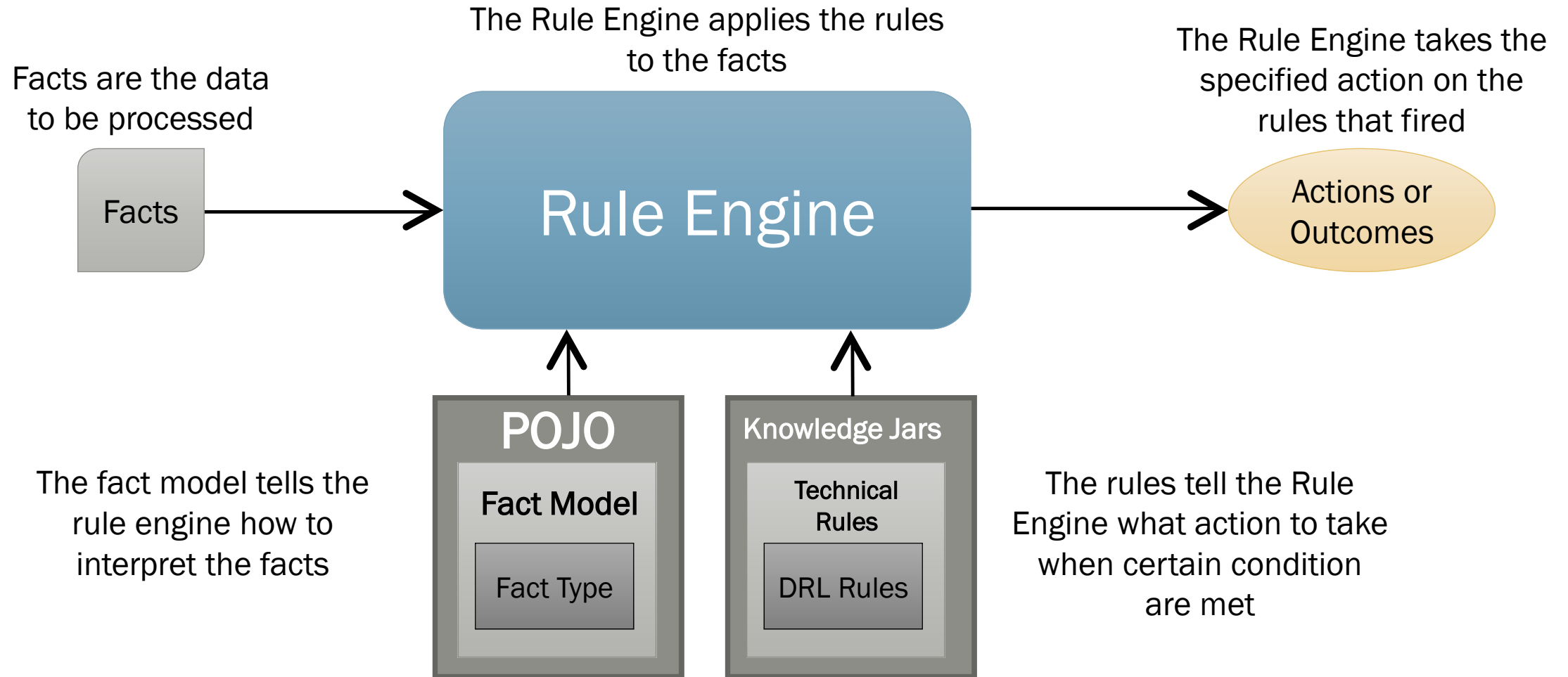
- Business Rule Management System (BRMS)
- Open Source
- Red Hat / JBoss
- Napisan u Javi
- Implementira proširjeni Rete algoritam



# KIE – Knowledge Is Everything



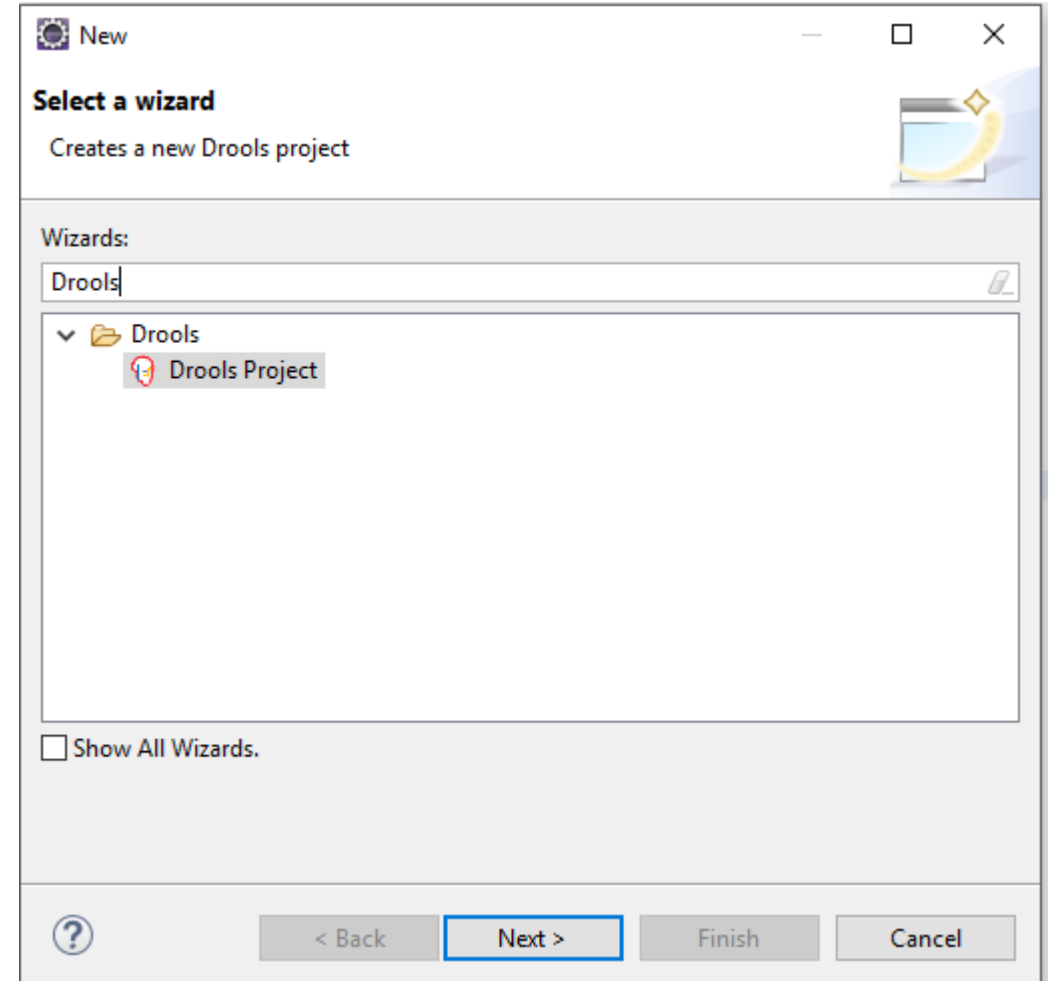
# Procesiranje pravila



# Drools - Projekat

Pravljenje Drools projekta:

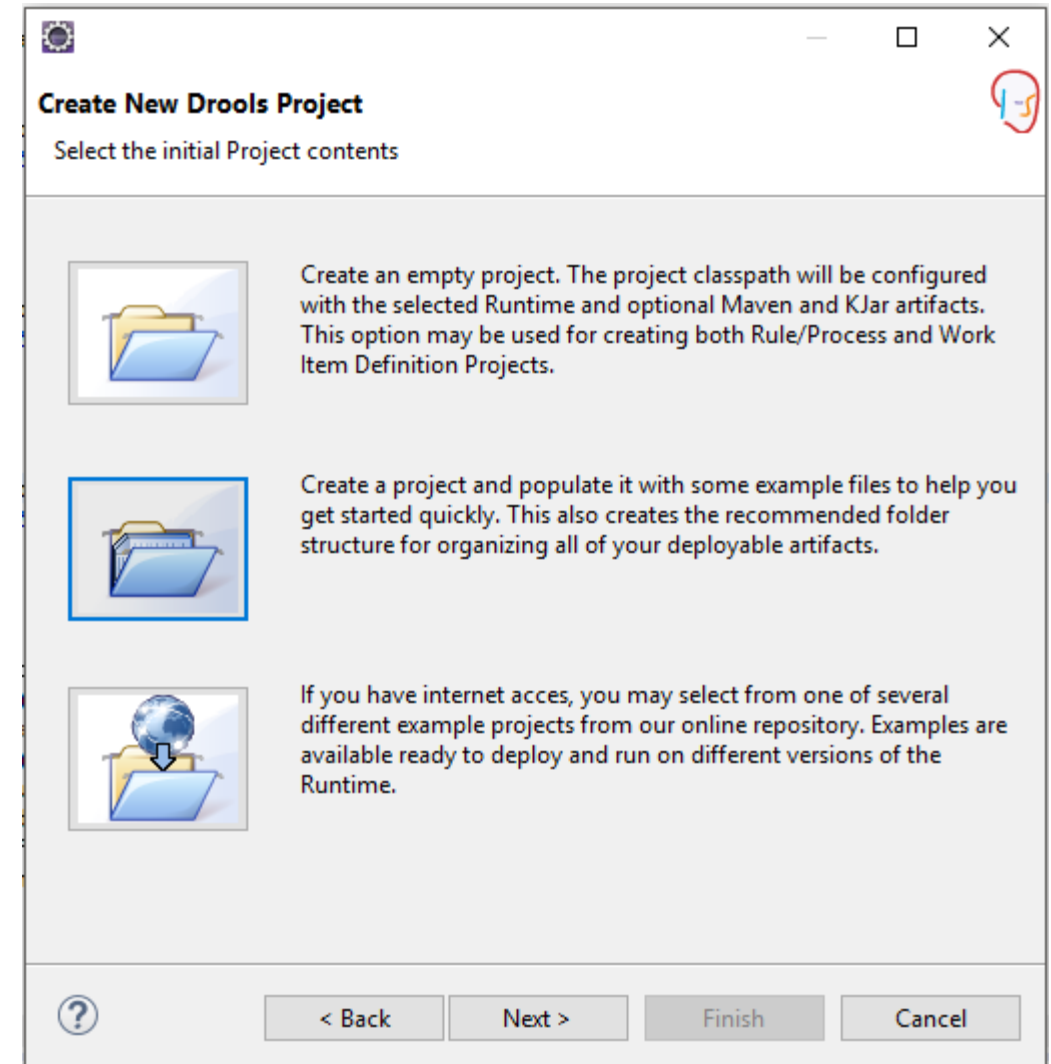
- Izvršiti instalaciju Drools alata prema uputstvu koje se nalazi u pdf fajlu
- U eclipse-u:
  - *File/New/Other*
  - *U filter ukucati Drools*
  - *Izabrati Drools Project*



# Drools - Projekat

Pravljenje Drools projekta:

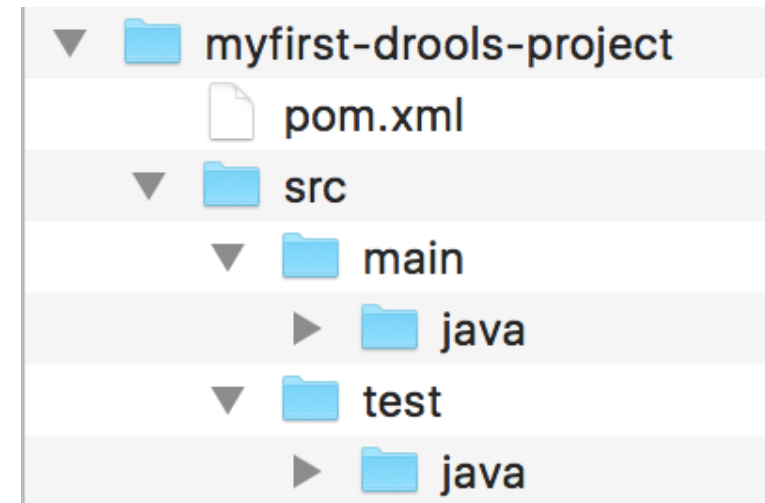
- Izabrati srednju opciju (Create a project and populate it ...)
- Uneti naziv projekta
- Čekirati samo opciju: Add a sample HelloWorld rule file to this project.



# Drools - Projekat

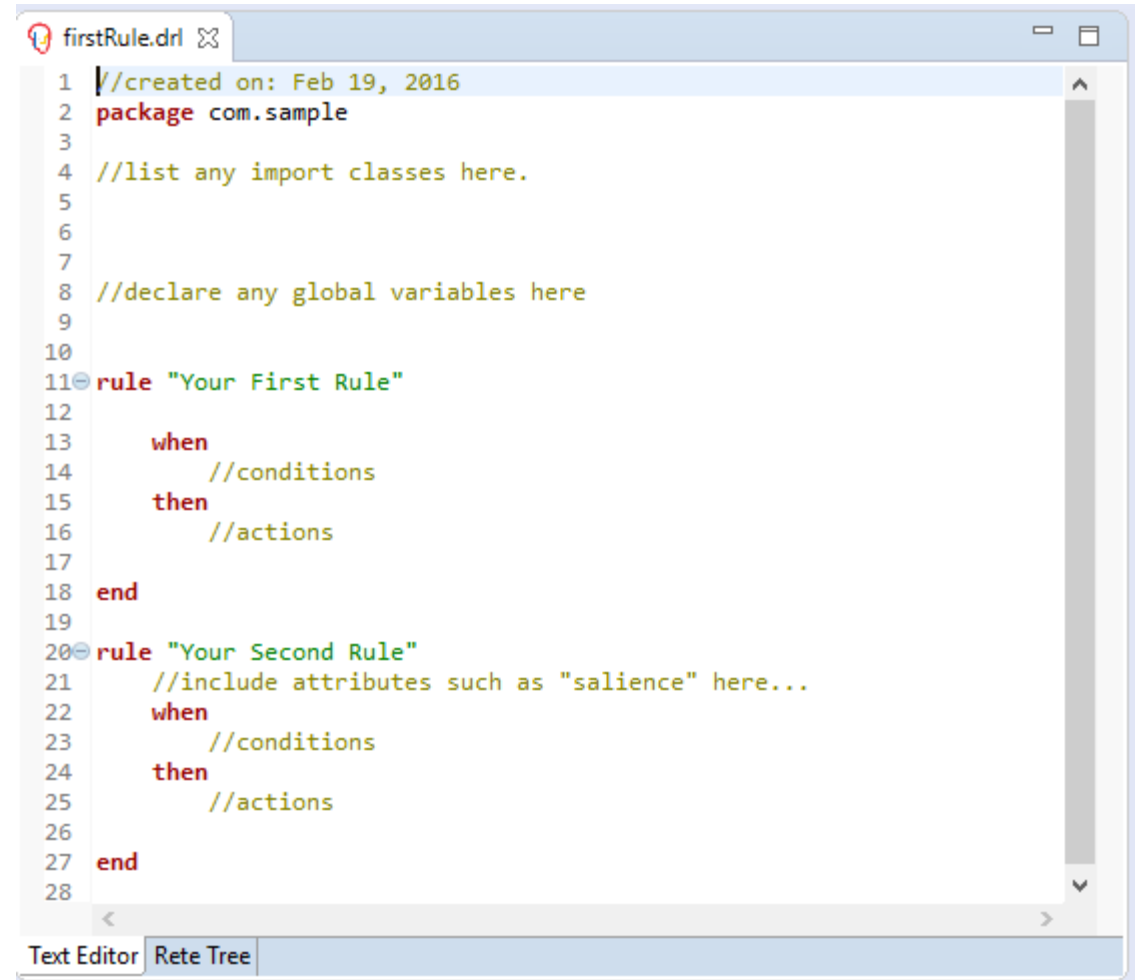
## Struktura projekta:

- pom.xml – (samo za maven projekat) sadrži definiciju projekta i first-level dependencies
- src/main/java – Java klase koje treba kompajlirati
- src/test/java – Test klase koje treba kompajlirati i izvršiti u fazi testiranja
- src/main/resources – Statički resursi koje ne treba kompajlirati, ali koji su potrebni kompajlirani klasama
- src/test/resources – Statički test resursi potrebni test klasama



# Drools – Struktura .drl fajlova

- Pravila se definišu u okviru .drl fajlova
- .drl fajlovi su statički resursi i smeštaju se u okviru src/main/resources direktorijuma
- Svako .drl fajl ima opštu strukturu prikazanu na slici.



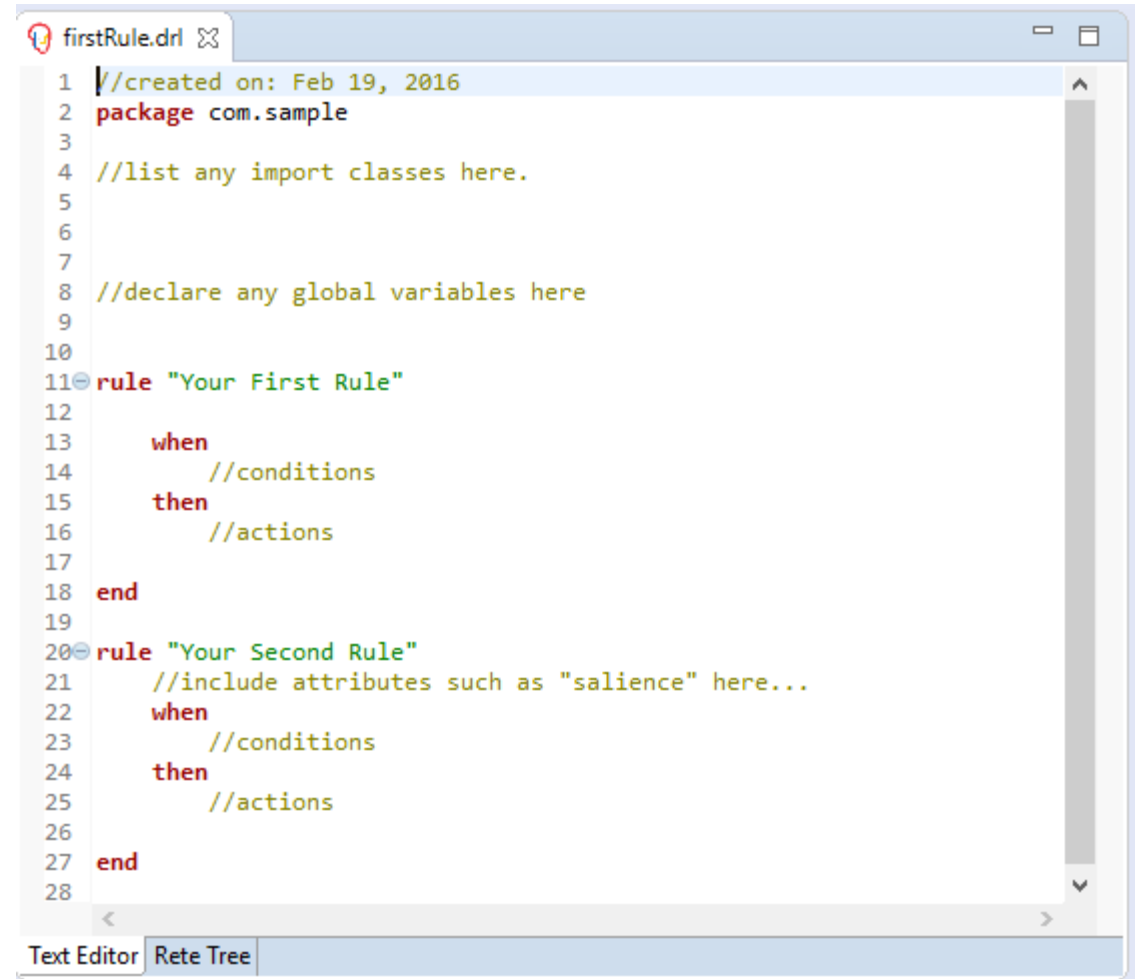
The screenshot shows a text editor window titled 'firstRule.drl'. The code is as follows:

```
1 //created on: Feb 19, 2016
2 package com.sample
3
4 //list any import classes here.
5
6
7
8 //declare any global variables here
9
10
11 rule "Your First Rule"
12
13     when
14         //conditions
15     then
16         //actions
17
18     end
19
20 rule "Your Second Rule"
21     //include attributes such as "salience" here...
22     when
23         //conditions
24     then
25         //actions
26
27     end
28
```

At the bottom of the window, there are two tabs: 'Text Editor' and 'Rete Tree'.

# Drools – Struktura .drl fajlova

- Opšta sturuktura:
  - **Definicija paketa:** Kao u Javi definiše se paket za pravila
  - **Import sekcija:** Potrebno je importovati sve klase koje će biti iskorišćene za realizaciju pravila
  - (Opciono) sekcija za deklaraciju tipova i događaja
  - **Pravila**



```
firstRule.drl
1 //created on: Feb 19, 2016
2 package com.sample
3
4 //list any import classes here.
5
6
7
8 //declare any global variables here
9
10
11 rule "Your First Rule"
12
13     when
14         //conditions
15     then
16         //actions
17
18 end
19
20 rule "Your Second Rule"
21     //include attributes such as "salience" here...
22     when
23         //conditions
24     then
25         //actions
26
27 end
28
```

Text Editor | Rete Tree

# Drools – Struktura pravila

- Conditions se pišu uz pomoć DRL jezika
- LHS pravila se sastoji od *conditional elements*, koji služe kao filteri da definišu uslove koje Facts trebaju da zadovolje kako bi se pravilo izvršilo
- RHS pravila sadrži posledice aktiviranja pravila

```
rule "name"  
when  
    (Conditions) – also called Left  
    Hand Side of the Rule (LHS)  
then  
    (Actions/Consequence) – also  
    called Right Hand Side of the  
    Rule (RHS)  
end
```

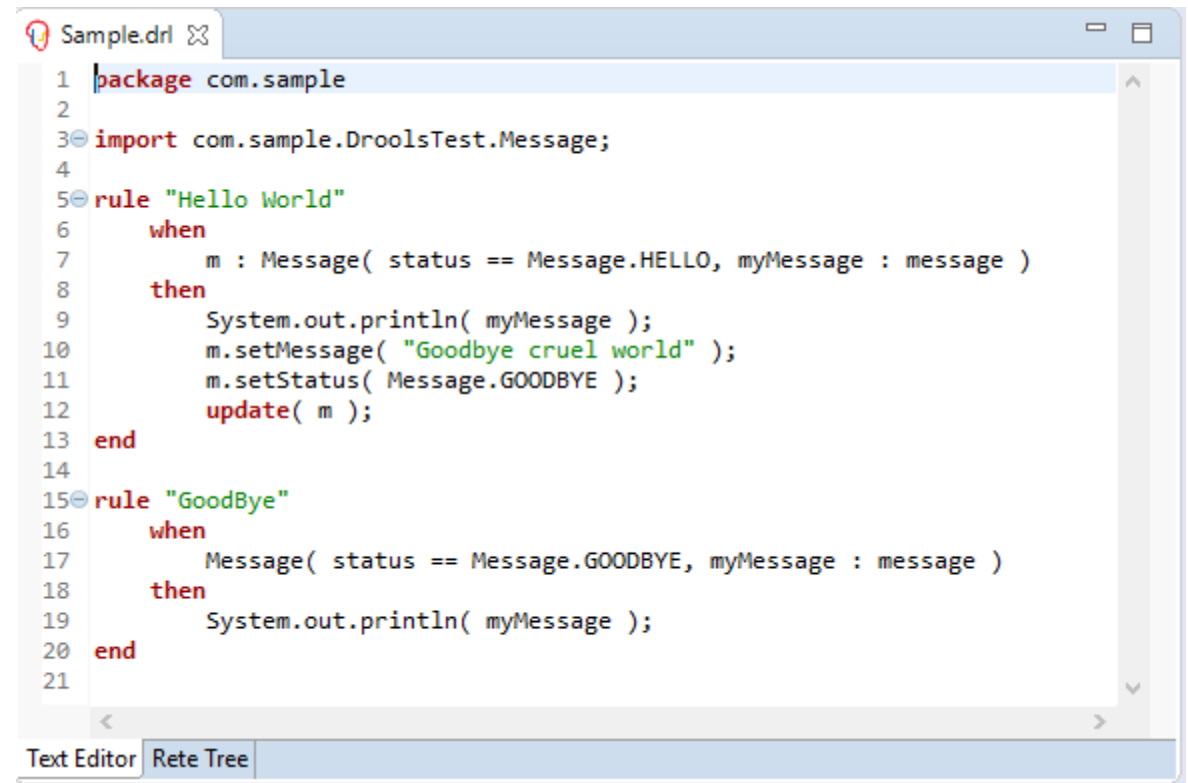


# Saveti za pisanje pravila

- Pravila bi trebalo da budu što jednostavnija
- Kompleksna pravila po mogućnosti razdvojiti
- Pravila bi trebalo da budu nezavisna (bez eksplicitnih poziva iz drugih pravila)
- Povezivanja pravila se vrši dodavanjem informacija o domenu pravila
- Primer:
  1. *When we get a signal from a fire alarm, we infer that there is a fire*
  2. *When there is a fire, we call the fire department*
  3. *When the fire department is present, we let them in to do their work*

# Drools – HelloWorld primer

- Sastoji se od dva pravila:
  - *Hello World i*
  - *GoodBye*
- U import sekciji uključujemo Message klasu koja će predstavljati Facts za pravila koje treba filtrirati
- LHS „Hello World“ pravila sadrži filter Message( ... ) koji filtrira sve objekte tipa Message koji su ubačeni u sesiju

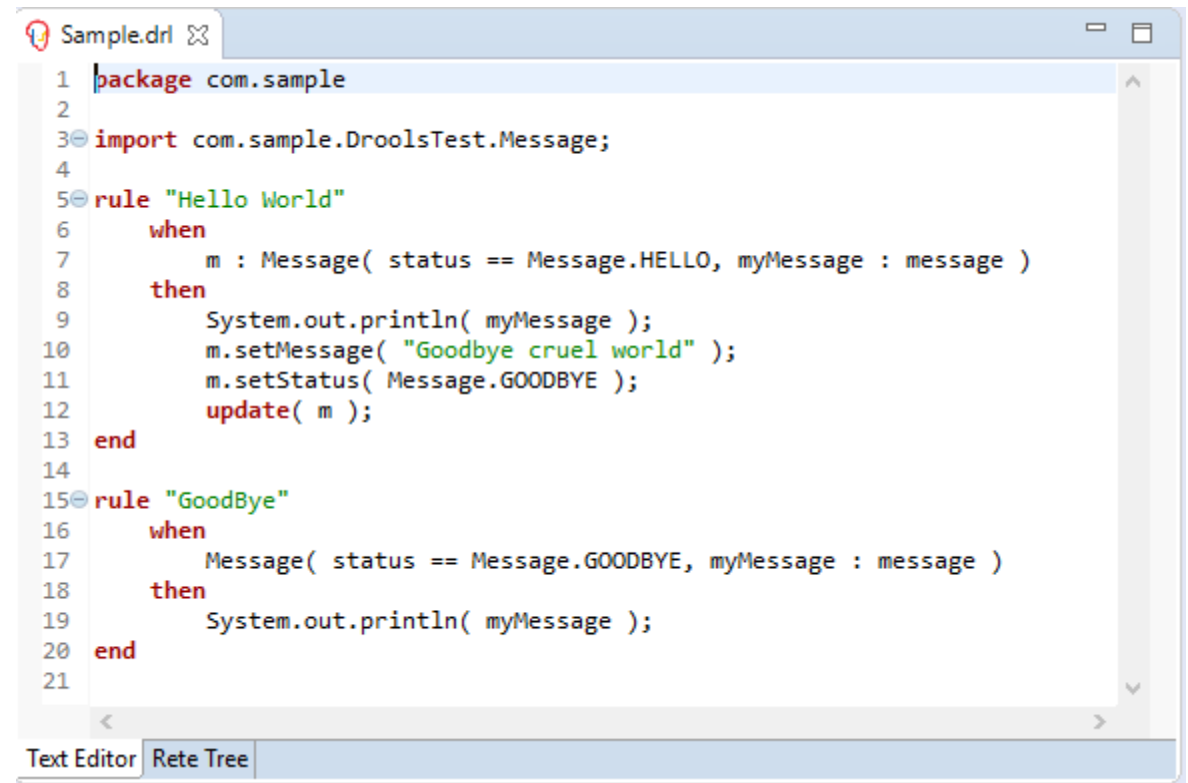


```
1 package com.sample
2
3 import com.sample.DroolsTest.Message;
4
5 rule "Hello World"
6     when
7         m : Message( status == Message.HELLO, myMessage : message )
8     then
9         System.out.println( myMessage );
10        m.setMessage( "Goodbye cruel world" );
11        m.setStatus( Message.GOODBYE );
12        update( m );
13    end
14
15 rule "GoodBye"
16     when
17         Message( status == Message.GOODBYE, myMessage : message )
18     then
19         System.out.println( myMessage );
20    end
21
```

The screenshot shows a text editor window titled 'Sample.drl' containing the above code. The code defines a package 'com.sample', imports 'com.sample.DroolsTest.Message', and defines two rules. The first rule, 'Hello World', has a left-hand side (LHS) that matches a Message object with status 'HELLO' and a right-hand side (RHS) that prints the message, updates it to 'Goodbye cruel world', and changes its status to 'GOODBYE'. The second rule, 'GoodBye', has an LHS that matches a Message object with status 'GOODBYE' and an RHS that prints the message. The editor has tabs for 'Text Editor' and 'Rete Tree' at the bottom.

# Drools – HelloWorld primer

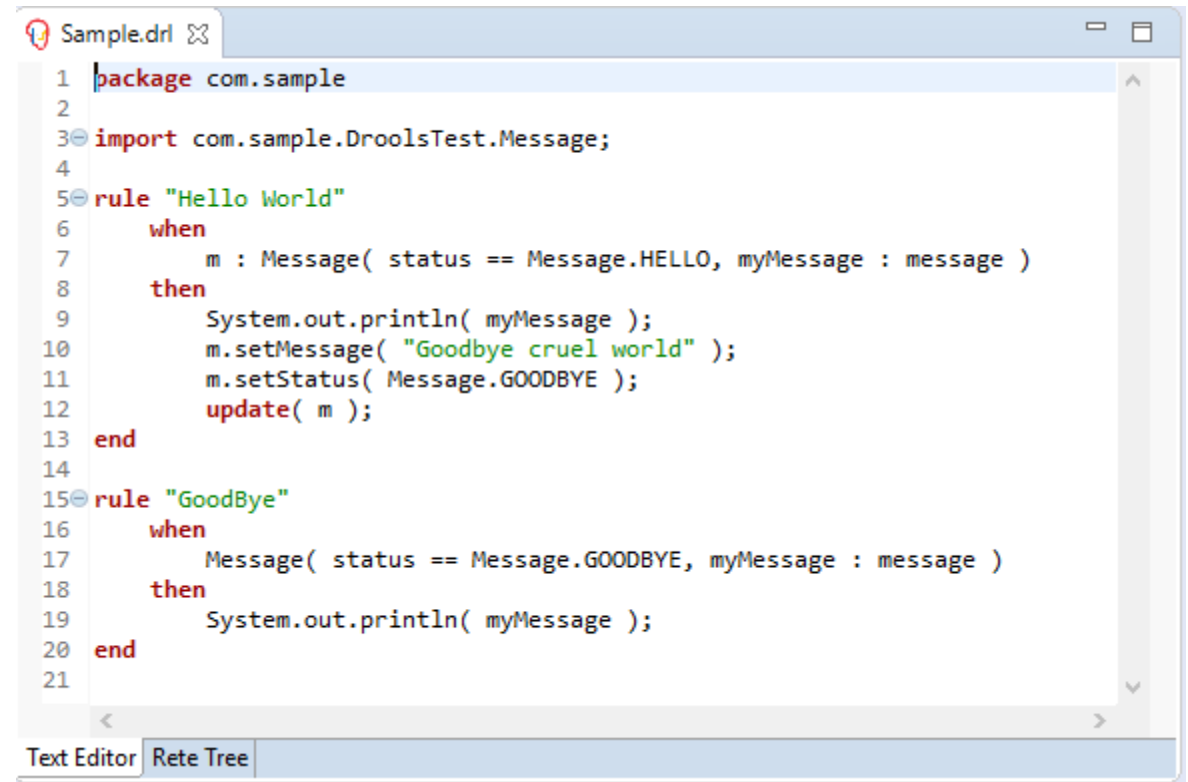
- Pravilo je zadovoljeno ukoliko postoji u sesiji POJO koji kao vrednost polja **status** ima vrednost **Message.HELLO**
- Ukoliko takav Fact (POJO) postoji, u promenljivu **myMessage** se smešta vrednost **message** polja
- **m** predstavlja *variable binding*, odnosno čuva referencu objekta koji je zadovoljio pravilo (Preporuka je da imena tih promenljivih počinju sa znakom **\$** -> umesto **m** trebalo bi da bude **\$m**)



```
1 package com.sample
2
3 import com.sample.DroolsTest.Message;
4
5 rule "Hello World"
6     when
7         m : Message( status == Message.HELLO, myMessage : message )
8     then
9         System.out.println( myMessage );
10        m.setMessage( "Goodbye cruel world" );
11        m.setStatus( Message.GOODBYE );
12        update( m );
13    end
14
15 rule "GoodBye"
16     when
17         Message( status == Message.GOODBYE, myMessage : message )
18     then
19         System.out.println( myMessage );
20    end
21
```

# Drools – HelloWorld primer

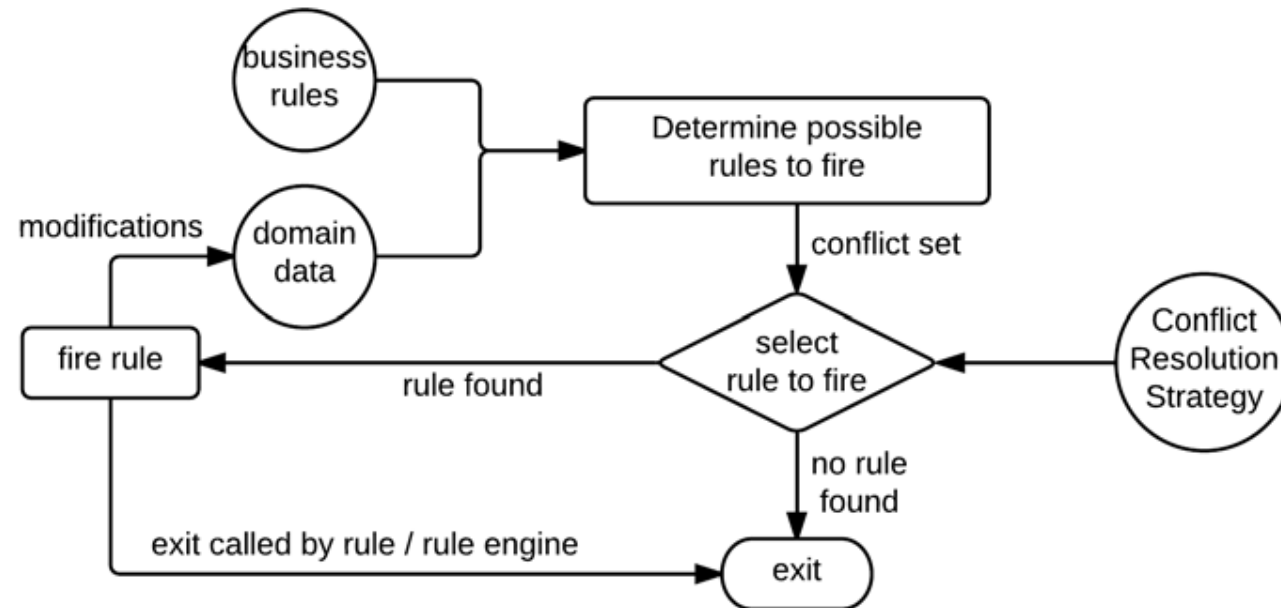
- U okviru RHS se nalazi konsekvencija pravila
- Ispisuje se poruka objekta
- Menja se poruka i status
- Na kraju poziva se **update** metoda koja notifikuje Rule Engine da je činjenica promenjena i da se trebaju evaluirati promene u odnosu na ostala pravila



```
1 package com.sample
2
3 import com.sample.DroolsTest.Message;
4
5 rule "Hello World"
6     when
7         m : Message( status == Message.HELLO, myMessage : message )
8     then
9         System.out.println( myMessage );
10        m.setMessage( "Goodbye cruel world" );
11        m.setStatus( Message.GOODBYE );
12        update( m );
13    end
14
15 rule "GoodBye"
16     when
17         Message( status == Message.GOODBYE, myMessage : message )
18     then
19         System.out.println( myMessage );
20    end
21
```

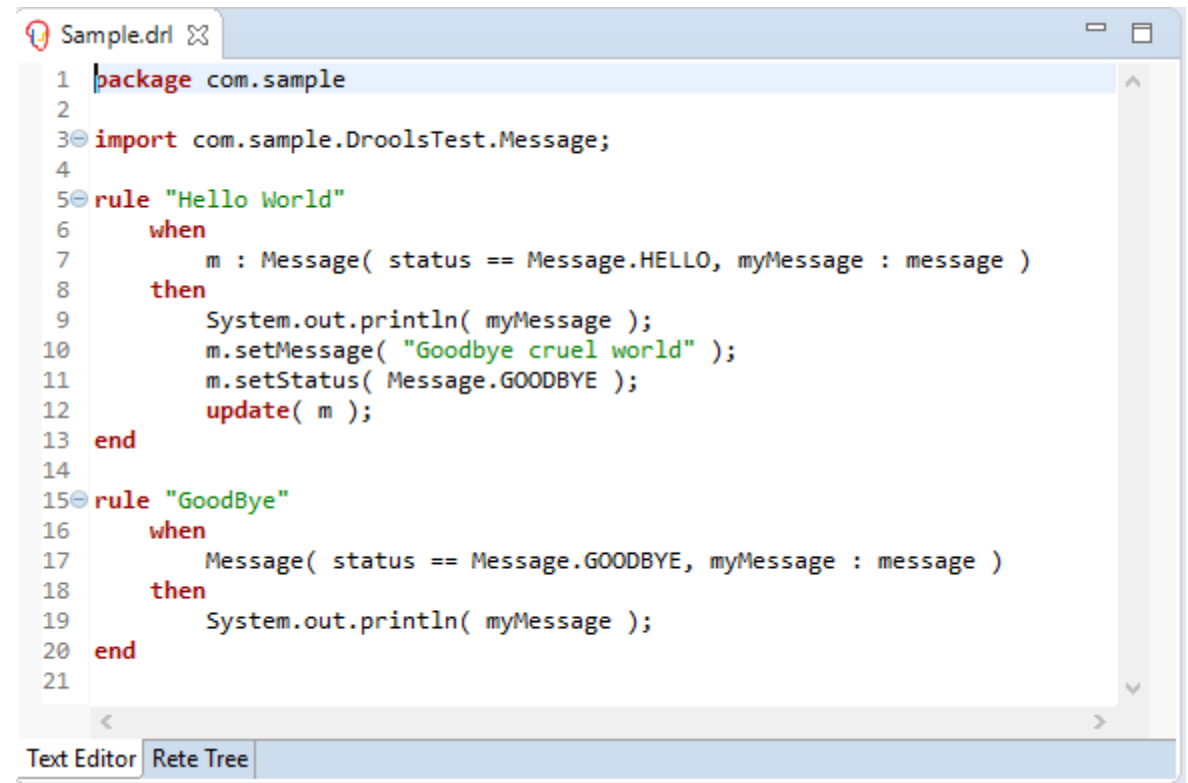
# Drools – HelloWorld primer

## ■ Proces izvršavanja pravila:



# Drools – HelloWorld primer

- Nakon **update** metode postoji činjenica (fact) koja zadovoljava GoodBye pravilo
- GoodBye pravilo samo ispisuje poruku „Goodbye cruel world“



```
Sample.drl
1 package com.sample
2
3 import com.sample.DroolsTest.Message;
4
5 rule "Hello World"
6     when
7         m : Message( status == Message.HELLO, myMessage : message )
8     then
9         System.out.println( myMessage );
10        m.setMessage( "Goodbye cruel world" );
11        m.setStatus( Message.GOODBYE );
12        update( m );
13    end
14
15 rule "GoodBye"
16     when
17         Message( status == Message.GOODBYE, myMessage : message )
18     then
19         System.out.println( myMessage );
20    end
21
```

Text Editor | Rete Tree

# Drools – HelloWorld primer

- Jednostavni oblik pravila koji bi se izvršio svaki put kada Rule Engine pronađe Message:

```
rule "Hello"  
  when  
    Message( )  
  then  
    System.out.println("Message exists!")  
end
```

# Drools – HelloWorld primer

- Kako bi se pravila izvršila potrebno je proslediti Rule Engine-u potrebne podatke
- Sa slike mogu se identifikovati tri celine u main metodi:
  - *Instanciranje Rule Engine-a*
  - *Prosleđivanje podataka Rule Engine-u*
  - *Aktiviranje pravila*



```
1 package com.sample;
2
3 import org.kie.api.KieServices;
4
5
6
7 /**
8  * This is a sample class to launch a rule.
9  */
10 public class DroolsTest {
11
12     public static final void main(String[] args) {
13         try {
14             // load up the knowledge base
15             KieServices ks = KieServices.Factory.get();
16             KieContainer kContainer = ks.getKieClasspathContainer();
17             KieSession kSession = kContainer.newKieSession("ksession-rules");
18
19             // go !
20             Message message = new Message();
21             message.setMessage("Hello World");
22             message.setStatus(Message.HELLO);
23             kSession.insert(message);
24             kSession.fireAllRules();
25         } catch (Throwable t) {
26             t.printStackTrace();
27         }
28     }
29 }
```

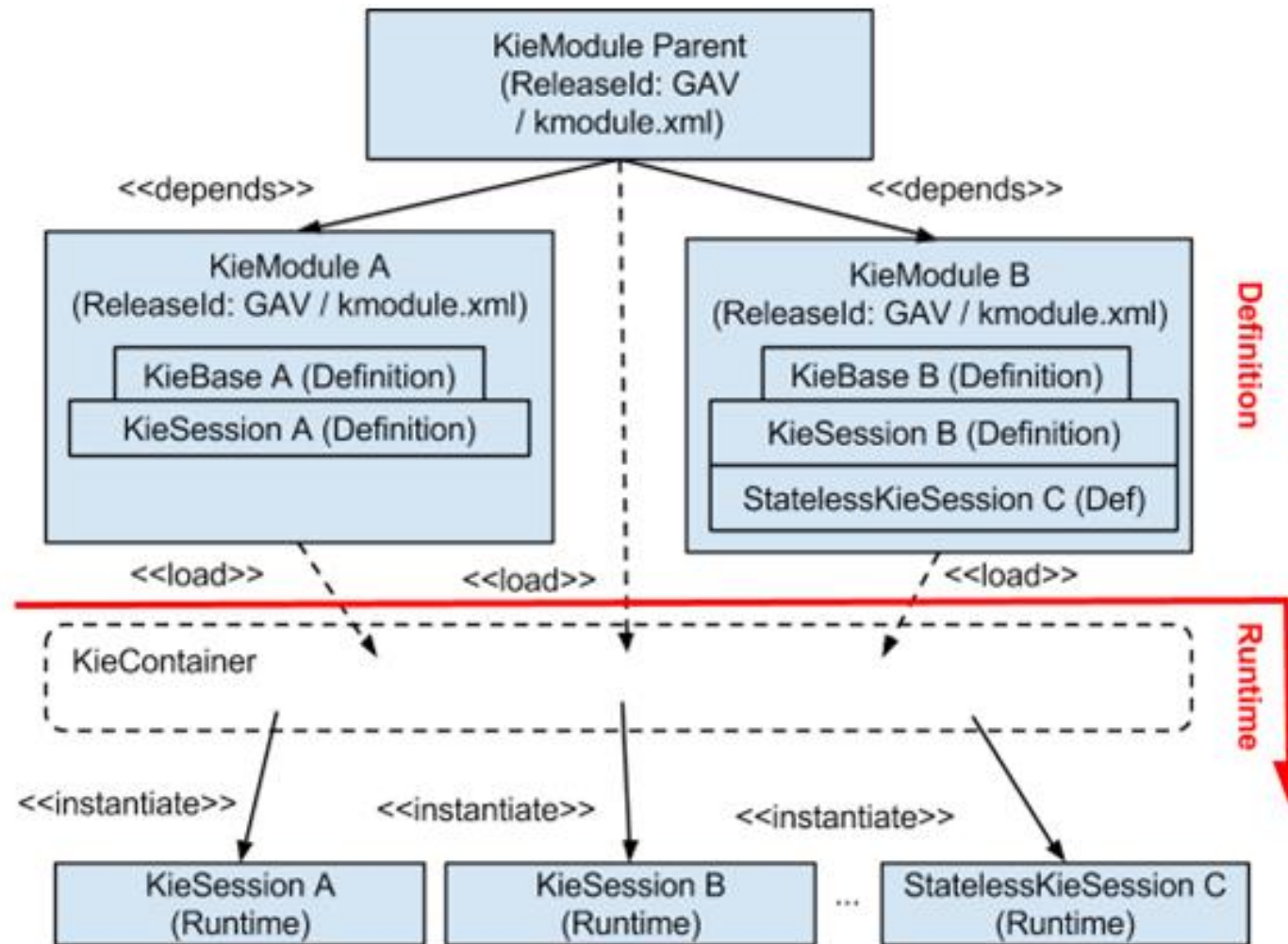


# Drools – Instanciranje Rule Engine-a

Postoje pet bitnih koncepata koje služe za konfigurisanje instanci rule engine-a:

- **KieServices** – KieServices klasa omogućava pristup svim ostalim konceptima. Uz pomoć KieServices klase se pravi nova instanca KieContainer-a.
- **KieContainer** – definiše opseg pravila koje služe za pravljenje novih instanci rule engine-a. KieContainer hostuje KieModule i njihove dependencije.
- **KieModule** – Svaki KieModule sadrži *business assets* (business rules, business processes, decision tables...). KieModule je standardni Java-Maven projekat koji sadrži pravila. Specijalni fajl kmodule.xml definiše konfiguraciju asseta u modulu.
- **KieBase** – predstavlja kompajliranu verziju asseta.
- **KieSession** – predstavlja instancu rule engine-a koja sadrži pravila u KieBase

# Drools – Instanciranje Rule Engine-a



# Drools – Kie Session

- Postoje dva tipa sesija:
  - *Stateful Session (KieSession)*
  - *Stateless Session (StatelessKieSession)*
- U opštem slučaju koriste se stateful sesije.
- Stateless sesije koriste za jednostavne slučajeve korišćenja, obično se posmatraju kao funkcije. Mogućnost primene:
  - *Validacije*
  - *Kalkulacije*
  - *etc.*

# Drools – Prosleđivanje podataka Rule Engine-u

- Rule enignu se prosleđuje POJO koji predstavlja činjenicu (fact)
- Prosleđivanje se vrši pomoću **insert** metode nad sesijom.
- Sam unos podataka neće aktivirati izvršavanje pravila. Da bi to uradili potrebno je pozvati **fireAllRules** metodu na sesijom.



```
1 package com.sample;
2
3 import org.kie.api.KieServices;
4
5
6 /**
7  * This is a sample class to launch a rule.
8  */
9
10 public class DroolsTest {
11
12     public static final void main(String[] args) {
13         try {
14             // load up the knowledge base
15             KieServices ks = KieServices.Factory.get();
16             KieContainer kContainer = ks.getKieClasspathContainer();
17             KieSession kSession = kContainer.newKieSession("ksession-rules");
18
19             // go !
20             Message message = new Message();
21             message.setMessage("Hello World");
22             message.setStatus(Message.HELLO);
23             kSession.insert(message);
24             kSession.fireAllRules();
25         } catch (Throwable t) {
26             t.printStackTrace();
27         }
28     }
29 }
```

# Drools – FactHandle

- Insert metoda vraća FactHandle nad prosleđenom činjenicom
- FactHandle predstavlja referencu ka činjenici u sesiji i može se koristiti za update i delete činjenici direktno iz programa.
- Update metoda kao parametre prima FactHandle i izmenjeni objekat na osnovu koga menja vrednost činjenice.
- Delete metoda briše činjenicu iz sesije, kao parametar prima FactHandle

```
FactHandle messageHandle = kSession.insert(message);  
message.setMessage("New updated message.");  
kSession.update(messageHandle, message);  
  
kSession.delete(messageHandle);  
  
kSession.fireAllRules();
```

# Drools - Facts

- Unos, brisanje i modifikovanje činjenica iz samih pravila moguće je upotrebom DRL keywords:
  - *insert*
  - *delete*
  - *update/modify*
- Primer 1.

# Primer 1 - Model

