Introduzione a Drools

Stefano Bragaglia

Fondamenti di Intelligenza Artificiale M

5 Giugno 2013

Ringraziamenti

- Si ringrazia l'Ing. Sottara per aver fornito la versione iniziale di questa esercitazione.

Sommario

1. Guida Introduttiva
2. Esercizi
   - *Pattern Semplici*
   - *Pattern Compositi*
   - *Quantificatori Esistenziali*
   - *Concatenazione di Regole*
   - *Features Avanzate: FROM, COLLECT, ACCUMULATE*
   - *Query*
   - *Truth Maintenance*
3. Informazioni

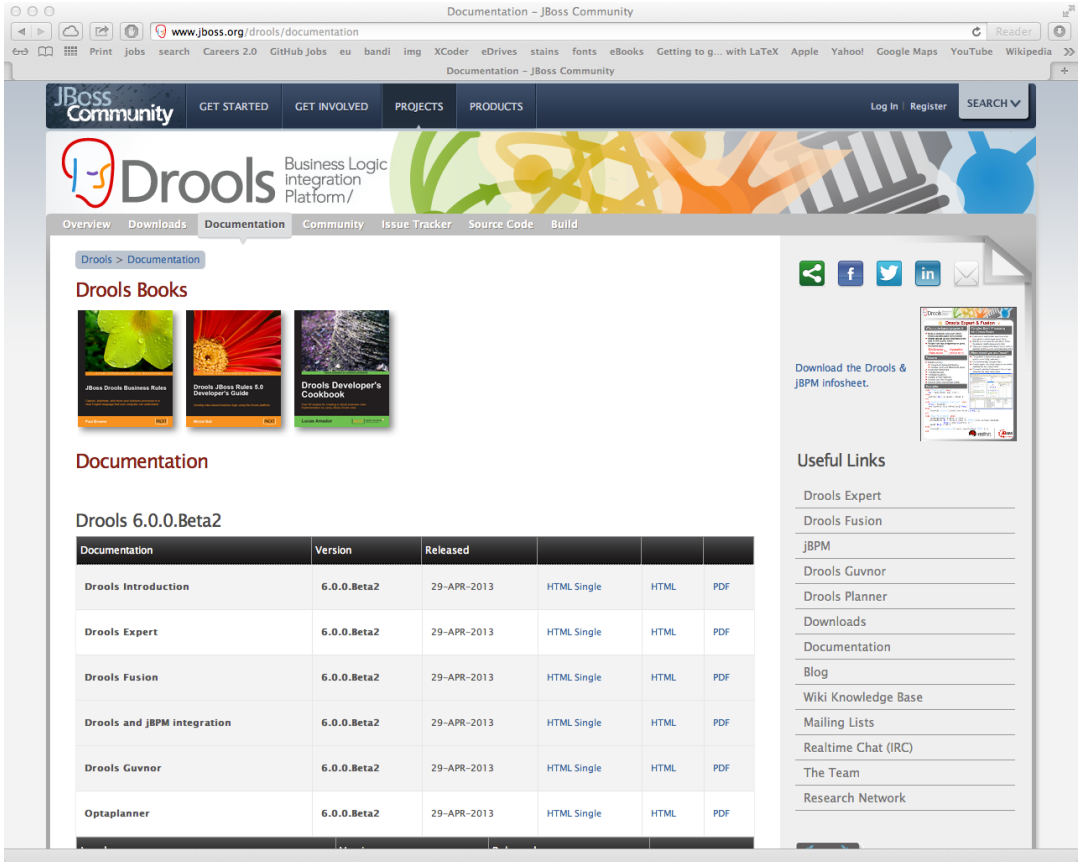
Fondamenti di Intelligenza Artificiale M

GUIDA INTRODUTTIVA

0. Documentazione

Disponibile gratuitamente sul sito
http://www.jboss.org/drools/documentation

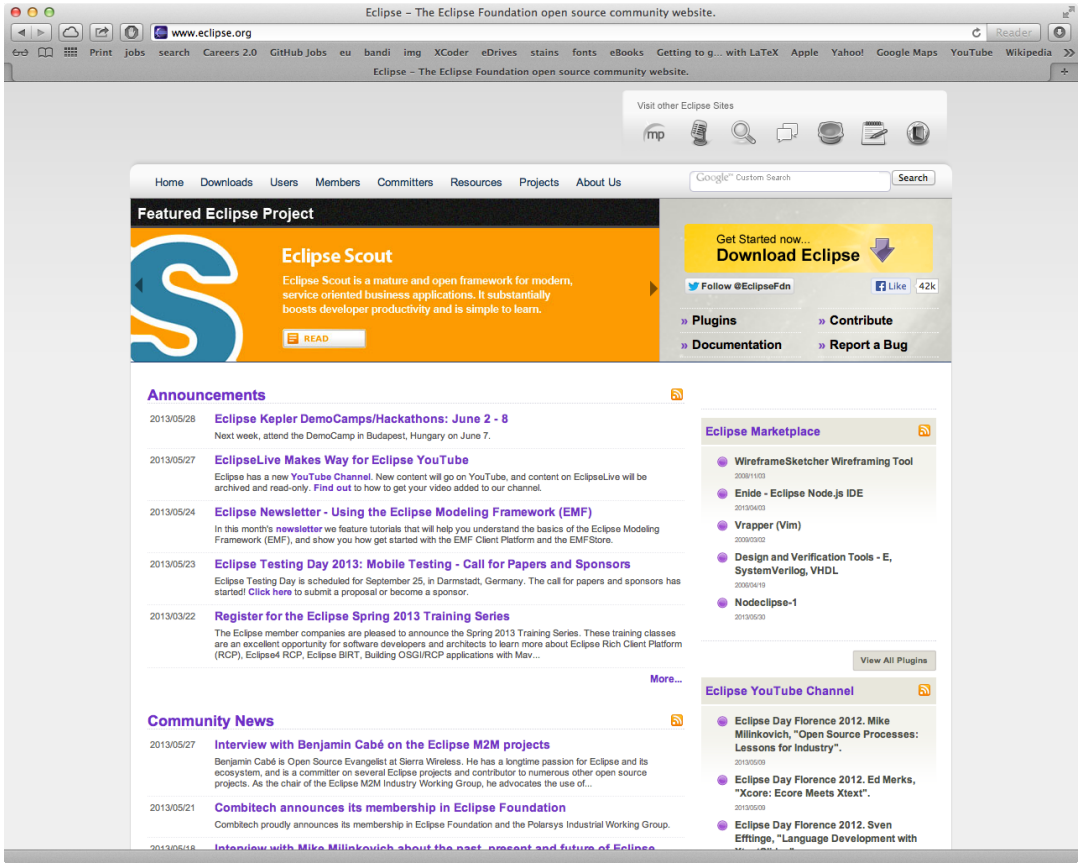
- *Installazione:*
  Drools Introduction (Cap. 2)
- *Esempi di base:*
  Drools Expert (Cap. 7)
- *Ragionamento temporale:*
  Drools Fusion (Cap. 2)



1. Eclipse

Scaricare la versione di Eclipse più approrpiata al proprio sistema dal sito

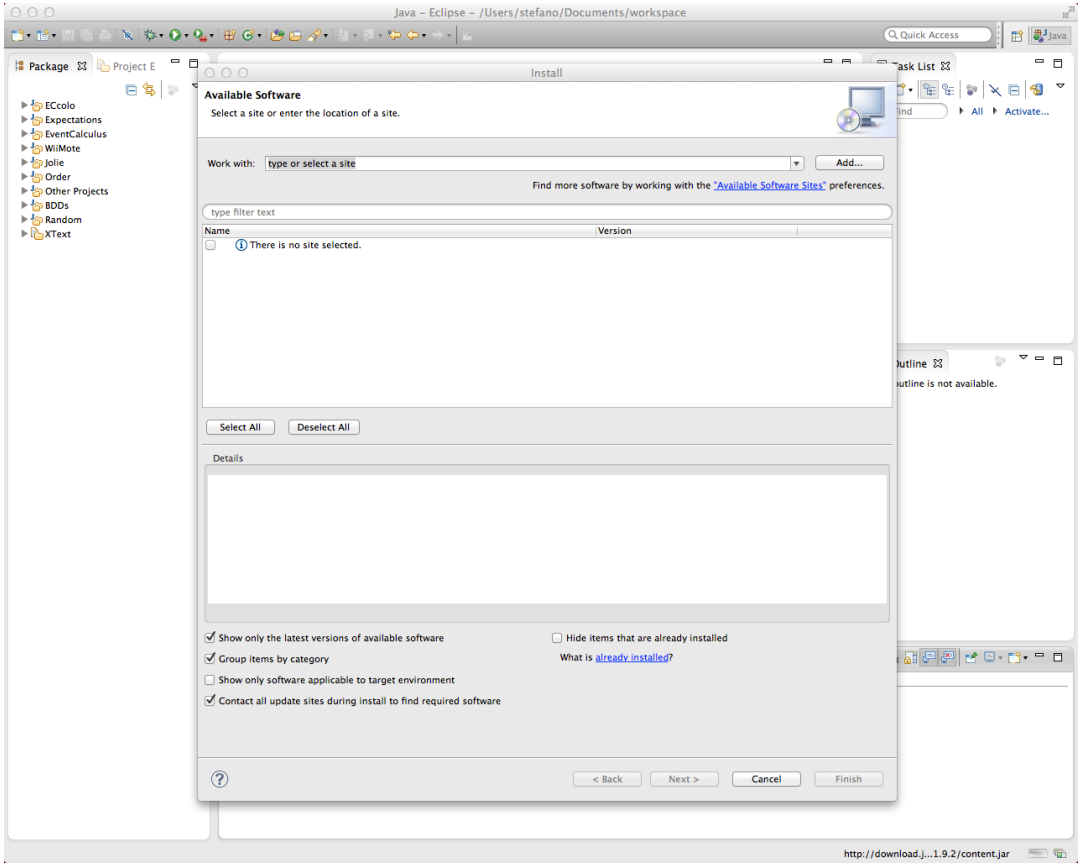
http://www.eclipse.org/downloads/



2. Drools Plugin

Installare il plugin di Drools per Eclipse

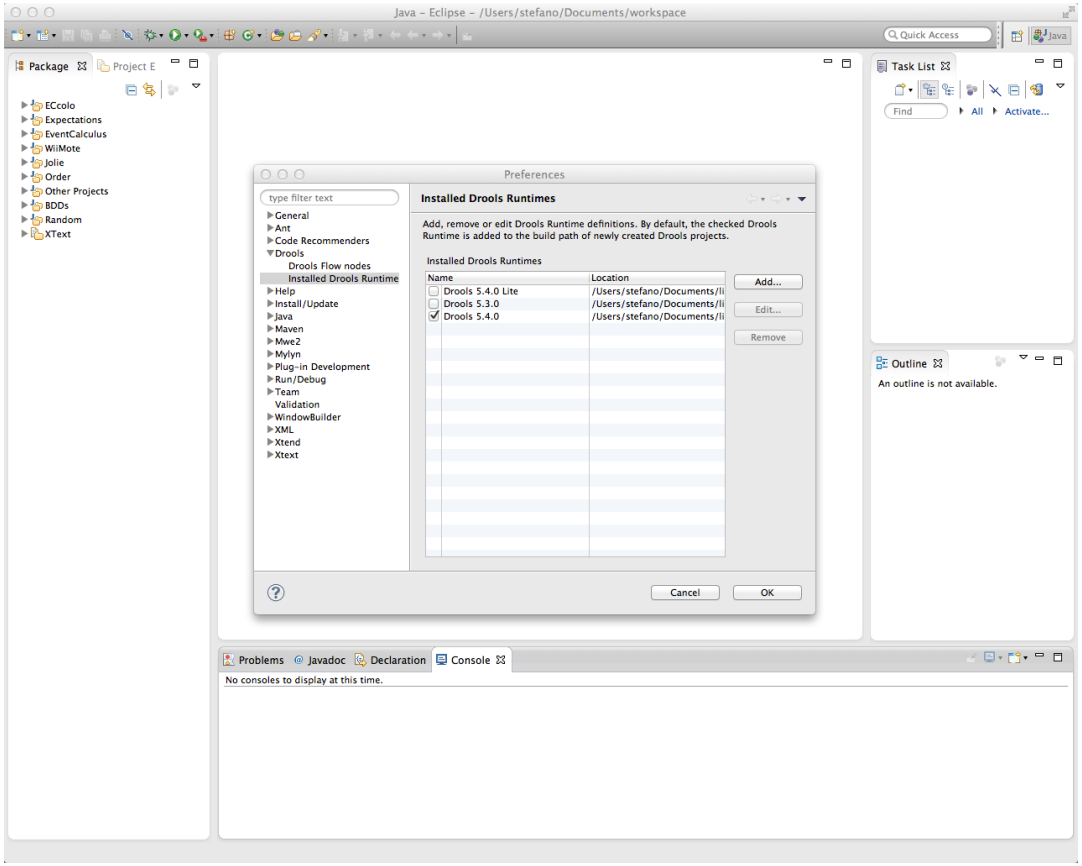
- *Help > Install New Software…*
- *Add…*
- Name:
  Drools Update Site – 5.5.0 Final
- Location:
  http://download.jboss.org/drools/release/5.5.0.Final/org.drools.updatesite/
- *Ok*
- *Drools and jBPM > Jboss Drools Core*
- *Finish*
- …
- Riavviare Eclipse



3. Creare un Runtime

I file per eseguire Drools sono stati già scaricati assieme al plugin, occorre solo indicare al sistema dove trovarli

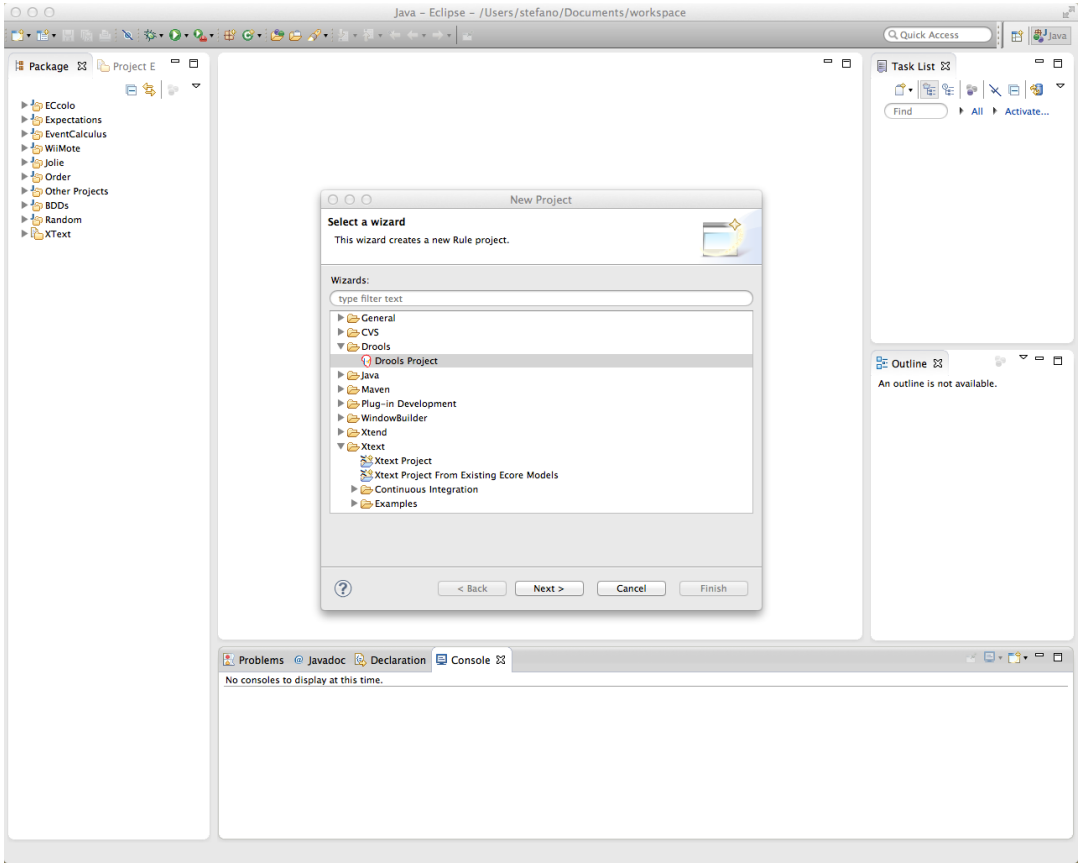
- *Eclipse > Preferences…*
- *Drools > Installed Drools Runtime*
- *Add…*
- *Create a new Drools 5 Runtime…*
- Selezionare la cartella in cui salvare il runtime (in laboratorio /usr/share/eclipse/library/drools-5.5.0)
- *Ok*
- …
- Selezionare il runtime appena creato
- *Ok*



4. Creare un progetto

Ora che la piattaforma è propriamente installata e configurata, creare un progetto Drools "Hello World!"

- *File > New > Project…*
- *Drools > Drools Project*
- *Next*
- *Project name:* **HelloWorld**
- *Next*
- Verificare che siano selezionati
  - *Add a sample HelloWorld rule fine to this project.*
  - *Add a sample Java class for loading and executing the HelloWorld rules.*
- *Next*
- Verificare che sia selezionato il runtime precedentemente creato
- *Finish*



DroolsTest.java: main Java

```
1.  package com.sample
2.
3.  public class DroolsTest {
4.
5.    public static final void main(String[] args) {
6.      try {
7.        // load up the knowledge base
8.        KnowledgeBase kbase = readKnowledgeBase();
9.        StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
10.       KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerFactory.newFileLogger(ksession, "test");
11.       // go !
12.       Message message = new Message();
13.       message.setMessage("Hello World");
14.       message.setStatus(Message.HELLO);
15.       ksession.insert(message);
16.       ksession.fireAllRules();
17.       logger.close();
18.     } catch (Throwable t) {
19.       t.printStackTrace();
20.     }
21.   }
22.
23.   private static KnowledgeBase readKnowledgeBase() throws Exception {..}
24.   ..
25. }
```

DroolsTest.java: caricamento della base di conoscenza

```
1.    package com.sample
2.
3.    public class DroolsTest {
4.
5.      public static final void main(String[] args) {..}
6.
7.      private static KnowledgeBase readKnowledgeBase() throws Exception {
8.        KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
9.        kbuilder.add(ResourceFactory.newClassPathResource("Sample.drl"), ResourceType.DRL);
10.       KnowledgeBuilderErrors errors = kbuilder.getErrors();
11.       if (errors.size() > 0) {
12.         for (KnowledgeBuilderError error: errors) {
13.           System.err.println(error);
14.         }
15.         throw new IllegalArgumentException("Could not parse knowledge.");
16.       }
17.       KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
18.       kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
19.       return kbase;
20.     }
21.     ..
22.   }
```

Sample.drl: file di regole

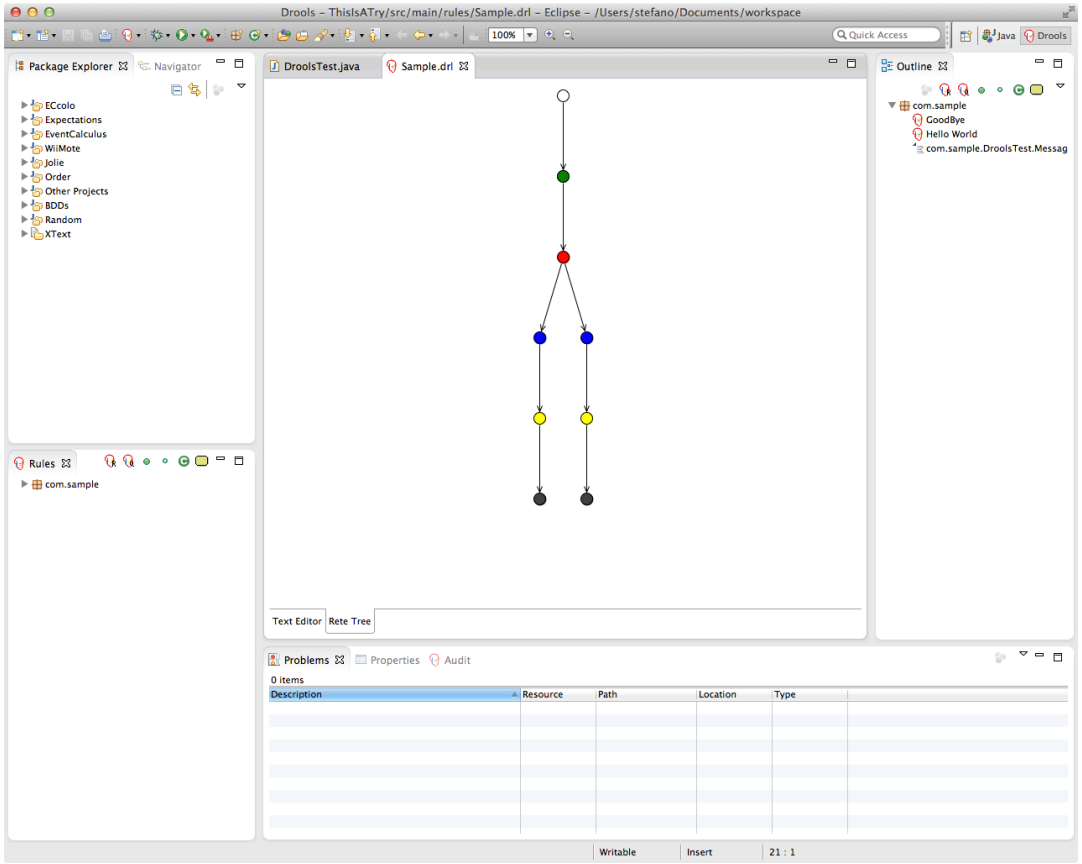
```
1.  package com.sample
2.
3.  import com.sample.DroolsTest.Message;
4.
5.  rule "Hello World"
6.  when
7.      m : Message( status == Message.HELLO, myMessage : message )
8.  then
9.      System.out.println(myMessage);
10.     m.setMessage("Goodbye cruel world");
11.     m.setStatus(Message.GOODBYE);
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
```

5. Debug del progetto

Maggiori dettagli nel Cap. 7.6 della documentazione di Drools Expert

http://www.jboss.org/drools/documentation

- Impostare un breakpoint su `ksession.fireAllRules();`
- Passare alla Drools Perspective
  - *Window > Open Perspective > Other…*
  - Selezionare *Drools*
  - *Ok*
  - Cliccare su *Drools*
- Lanciare una sessione di debug
- Selezionare la sessione tra le variabili di debug
- Aggiungere view:
  - Window > Show View > Other…
  - Agenda, Audit, WM, ecc.
  - Ok



Fondamenti di Intelligenza Artificiale M

ESERCIZI

Esercizi

PATTERN SEMPLICI

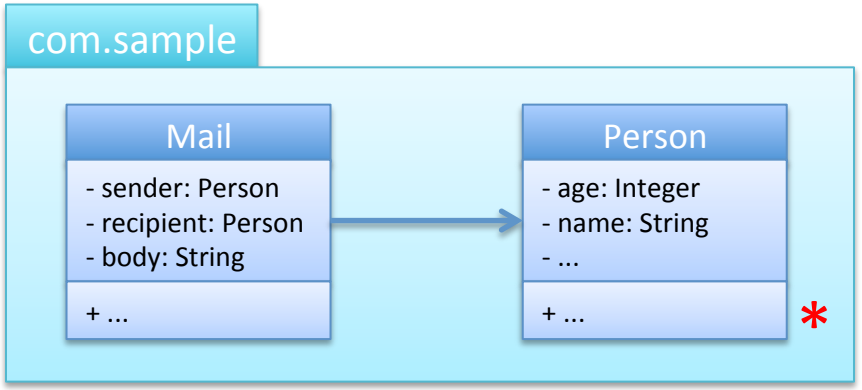
Pattern Semplici

- Un pattern semplice è un insieme di vincoli da rispettare su fatti di uno stesso tipo in WM
- Per poterli applicare, occorre che Drools sappia come sono fatti questi oggetti
- I fatti sono definiti come POJO o BEAN e poi importati nel file `.drl` di Drools col comando

```
import com.sample.DroolsTest.Message;
```

Pattern Semplici

- Creare un nuovo progetto come `HelloWorld`
- Rimuovere ogni riferimento a `Message` da `DroolsTest.java`
- Creare I POJO/BEAN rappresentati nella figura di destra



* Aggiungere i campi costruttore, *getter*, *setter*, `equals(…)`, `hashcode()` e `toString()`.

N.B.: Eclipse consente di generare tutti questi metodi automaticamente.

Pattern Semplici

- Creare un nuovo file di regole (es.: `exercise.drl`)
- Indicare questa nuova risorsa (ancora vuota) come risorsa da caricare in `DroolsTest.java`
- Asserire (almeno) le istanze a destra prima di far scattare la valutazione delle regole nella sessione attuale

```
Person p1 = new Person("Andrew", 35);
Person p2 = new Person("Barbara", 22);
Person p3 = new Person("Cal", 8);
Person p4 = new Person("Cal", 15);
Person p5 = new Person("Daisy", 25);
Person p6 = new Person("Cal", 40);
ksession.insert(p1);
ksession.insert(p2);
ksession.insert(p3);
ksession.insert(p4);
ksession.insert(p5);
ksession.insert(p6);
ksession.insert(new Mail(p1, p4,
  "Do your homeworks!"));
ksession.insert(new Mail(p5, p5,
  "Remember to do the homeworks!"));
...
ksession.fireAllRules();
```

Pattern Semplici

- Scrivere una regola che stampi a video il contenuto di ogni `Mail`
- Scrivere una regola che stampi a video il contenuto di ogni `Person`

```
package com.sample

import com.sample.Mail;
import com.sample.Person;

rule "Regola: Simple 1"
when
  // pattern della premessa
then
  // action della conseguenza
end

rule "Regola: Simple 2"
when
 // pattern della premessa
then
  // action della conseguenza
end

// NB: Le regole scattano per ogni fatto
// appartenente al tipo indicato
```

Pattern Semplici

- Scrivere una regola che stampi a video il contenuto di ogni `Mail`
- Scrivere una regola che stampi a video il contenuto di ogni `Person`

```
package com.sample

import com.sample.Mail;
import com.sample.Person;

rule "Regola: Simple 1"
when
  $m: Mail()
then
  System.out.println("S1-Mail: " + $m);
end

rule "Regola: Simple 2"
when
  $p: Person()
then
  System.out.println("S2-Person: " + $p);
end

// NB: Le regole scattano per ogni fatto
// appartenente al tipo indicato
```

Pattern Semplici

- Trovare tutte le `Person` che si chiamano `"Cal"` e hanno meno di `10` anni o età compresa tra i `18` e i `35` anni

```
package com.sample

import com.sample.Mail;
import com.sample.Person;

rule "Regola: Simple 3"
when
  ...
then
  ...
end

// NB: Uso di operatori di precedenza
// [()], and [&&] e or [||]...
```

Pattern Semplici

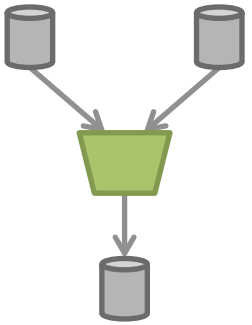
- Trovare tutte le `Person` che si chiamano `"Cal"` e hanno meno di `10` anni o età compresa tra i `18` e i `35` anni

```
package com.sample

import com.sample.Mail;
import com.sample.Person;

rule "Regola: Simple 3"
when
  $p: Person( name == "Cal",
        age < 10 ||
        (age >= 18 && age <= 35)
  )
then
  System.out.println("S3-Person: " + $p);
end

// NB: Uso di operatori di precedenza
// [()], and [&&] e or [||]...
```

Esercizi

PATTERN COMPOSITI

Pattern Compositi: Join



- Più pattern in una stessa regola selezionano oggetti di più tipi
- La regola abbina automaticamente questi oggetti, costruendo tutte le loro possibili combinazioni
- I nodi di Join (Beta nodes) fanno il “prodotto cartesiano” degli oggetti che ricevono in ingresso

Pattern Compositi: Join

- Scrivere una regola per trovare tutte le coppie di persone

```
rule "Regola: Join 1"
when
  ...
then
  ...
end
```

Pattern Compositi: Join

- Scrivere una regola per trovare tutte le coppie di persone

```
rule "Regola: Join 1"
when
  $p1: Person()
  $p2: Person()
then
  System.out.println("J1: " + $p1 +
" vs. " + $p2);
end
```

Pattern Compositi: Join

- Scrivere una regola per trovare tutte le coppie di persone
- Evitare di accoppiare una persona con se stessa poichè non ha senso…

```
rule "Regola: Join 1"
when
  $p1: Person()
  $p2: Person()
then
  System.out.println("J1: " + $p1 +
" vs. " + $p2);
end
```

Pattern Compositi: Join

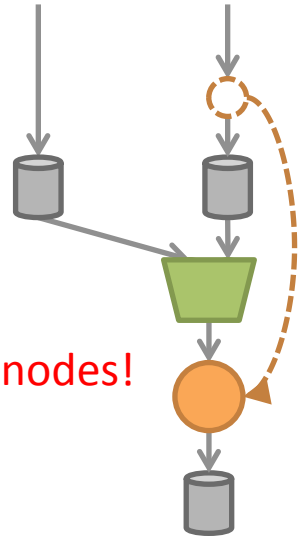
- Scrivere una regola per trovare tutte le coppie di persone
- Evitare di accoppiare una persona con se stessa poichè non ha senso…

```
rule "Regola: Join 1"
when
  $p1: Person()
  $p2: Person()
then
  System.out.println("J1: " + $p1 +
" vs. " + $p2);
end

rule "Regola: Join 2"
when
  $p1: Person()
  $p2: Person( this != $p1 )
then
  System.out.println("J2: " + $p1 +
" vs. " + $p2);
end
```

Pattern Compositi: Join

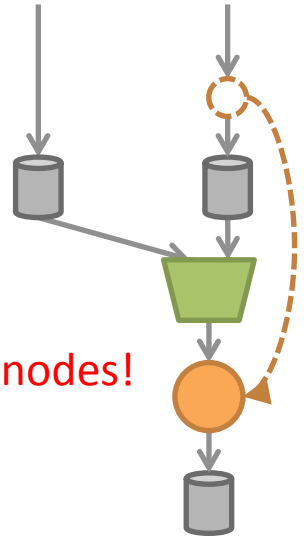
- Trovare le coppie (non degeneri) di omonimi

```
rule "Regola: Join 3"
when
  ...
then
  ...
end
```



Pattern Compositi: Join

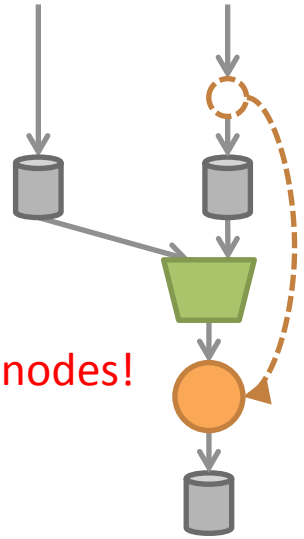
- Trovare le coppie (non degeneri) di omonimi




```
rule "Regola: Join 3"
when
  $p1: Person( $n: name )
  $p2: Person( this != $p1,
               name == $n )
then
  System.out.println("J3: " + $p1 +
" vs. " + $p2);
end
```

Pattern Compositi: Join

- Trovare le coppie (non degeneri) di omonimi
- Evitare le coppie inverse: (a,b) e non (b,a)




```
rule "Regola: Join 3"
when
  $p1: Person( $n: name )
  $p2: Person( this != $p1,
               name == $n )
then
  System.out.println("J3: " + $p1 +
" vs. " + $p2);
end

rule "Regola: Join 4"
when
  $p1: Person( $n: name )
  $p2: Person( this != $p1, $p1 < $p2,
               name == $n )
then
  System.out.println("J4: " + $p1 +
" vs. " + $p2);
end
```

Pattern Compositi: Join

- Trovare le coppie (non degeneri) composte da un uomo ed una donna* in cui lei è più giovane

```
rule "Regola: Join 5"
when
  ...
then
  ...
end
```

* Estendere opportunamente il BEAN

Pattern Compositi: Join

- Trovare le coppie (non degeneri) composte da un uomo ed una donna* in cui lei è più giovane

```
rule "Regola: Join 5"
when
  $p1: Person( gender == "male",
               $e : age )
  $p2: Person( gender == "female",
               age < $e )
then
  System.out.println("J5: " + $p1 +
" vs. " + $p2);
end

// Sufficiente? Inversioni?
// Unica soluzione possibile?
```

* Estendere opportunamente il BEAN

Pattern Compositi: Join

- Stampare il testo del messaggio di una `Mail`, a patto che il mittente sia una persona di almeno 20 anni di nome Daisy che ha mandato la mail a se stessa

```
rule "Regola: Join Multipli"
when
  ...
then
  ...
end
```

Pattern Compositi: Join

- Stampare il testo del messaggio di una `Mail`, a patto che il mittente sia una persona di almeno 20 anni di nome Daisy che ha mandato la mail a se stessa

```
rule "Regola: Join Multipli"
when
  $p:  Person( age >= 20,
         name == "Daisy" )
  $pp: Person( this == $p )
  // $pp serve davvero?
  $m:  Mail ( sender == $p,
         recipient == $pp,
         $b: body )
then
  System.out.println("J5.body: " + $b);
end
```

Pattern Compositi: Join

- Stampare il testo del messaggio di una `Mail`, a patto che il mittente sia una persona di almeno 20 anni di nome Daisy che ha mandato la mail a se stessa

```
rule "Regola: Join Multipli"
when
  Mail ( $p: sender,
         sender.name == "Daisy",
         sender.age >= 20,
         recipient == $p,
         $t: body.toString() )
         // Possibile, ma sconsigliato
then
  System.out.println("J5.body: " + $t);
end
```

NB: La Dot Notation rende i pattern più compatti

Esercizi

QUANTIFICATORI ESISTENZIALI

Quantificatori Esistenziali

- Drools supporta nativamente i quantificatori esistanziali ∃ e ∀ mediante le parole chiave **`exists`** e **`forall`** il cui scope è un pattern
  - **`exists`** `P(…)`
    la WM contiene almeno un fatto fa match
  - **`not`** `P(…)` (sottointeso **`exists`**)
    la WM non contiene alcun fatto che fa match
  - **`forall`** `P(…)`
    tutti gli oggetti di tipo `P` in WM fanno match

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail

```
rule "Regola QE1"
when
  ...
then
  ...
end
```

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail

```
rule "Regola QE1"
when
  $p: Person( $n: name )
  exists Mail( receiver == $p )
then
  System.out.println("QE1: " + $n
    + " ha ricevuto mail");
end
```

NB: le regole scattano 0 o 1 volte per persona

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail
  - Che non hanno ricevuto alcuna mail

```
rule "Regola QE1"
when
  $p: Person( $n: name )
  exists Mail( receiver == $p )
then
  System.out.println("QE1: " + $n
    + " ha ricevuto mail");
end

rule "Regola QE2"
when
  ...
then
  ...
end
```

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail
  - Che non hanno ricevuto alcuna mail

```
rule "Regola QE1"
when
  $p: Person( $n: name )
  exists Mail( receiver == $p )
then
  System.out.println("QE1: " + $n
    + " ha ricevuto mail");
end

rule "Regola QE2"
when
  $p: Person( $n: name )
  not Mail( receiver == $p )
then
 System.out.println("QE2: " + $n
    + " non ha ricevuto mail");
end
```

NB: le regole scattano 0 o 1 volte per persona

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail
  - Che non hanno ricevuto alcuna mail
  - Per cui tutte le mail ricevute provengono da un mittente di nome Cal

```
rule "Regola QE3"
when
  ...
then
  ...
end
```

Quantificatori Esistenziali

- Stampare il nome delle persone
  - Che anno ricevuto almeno una mail
  - Che non hanno ricevuto alcuna mail
  - Per cui tutte le mail ricevute provengono da un mittente di nome Cal

```
rule "Regola QE3"
when
  $p: Person( $n: name )
  forall (
    Mail( $s: sender, receiver == $p )
    Person( this == $s, name == "Cal" )
  )
then
  System.out.println("QE3: " + $n
    + " ha ricevuto solo mail da Cal");
end
```

NB: le regole scattano 0 o 1 volte per persona

Esercizi

CONCATENAZIONE DI REGOLE

Concatenazione di Regole

- Drools supporta la generazione di classi all'interno dei file .drl
- Con opportune dichiarazioni (tipicamente incluse prima delle regole) si possono introdurre BEAN al volo da usare nelle regole:

```
declare Pair
  first: Person
  second: Person
end
```

Concatenazione di Regole

- Oltre ad eseguire codice Java generico, il conseguente di una regola può eseguire "operazioni logiche":
  - Inserimento di nuovi fatti nella WM
  - Rimozione di fatti dalla WM
  - Aggiornamento di fatti nella WM
- Queste operazioni, a loro volta, possono attivare le regole in cascata

Concatenazione di Regole

- Scrivere una regola per stampare un messaggio a video per ogni `Pair` presente in WM

```
rule "Regola CdR 1"
when
  ...
then
  ...
end
```

Concatenazione di Regole

- Scrivere una regola per stampare un messaggio a video per ogni `Pair` presente in WM

```
rule "Regola CdR 1"
when
  $p: Pair()
then
  System.out.println("CdR1 – Pair: " + $p);
end
```

Concatenazione di Regole

- Scrivere una regola per stampare un messaggio a video per ogni `Pair` presente in WM
- Scrivere una regola per generare e inserire in WM un oggetto `Pair` per ogni coppia di persone distinte senza ripetizioni

```
rule "Regola CdR 1"
when
  $p: Pair()
then
  System.out.println("CdR1 – Pair: " + $p);
end

rule "Regola CdR 2"
when
  ...
then
  ...
end
```

Concatenazione di Regole

- Scrivere una regola per stampare un messaggio a video per ogni `Pair` presente in WM
- Scrivere una regola per generare e inserire in WM un oggetto `Pair` per ogni coppia di persone distinte senza ripetizioni

NB: L'ordine delle regole è ininfluente

```
rule "Regola CdR 1"
when
  $p: Pair()
then
  System.out.println("CdR1 – Pair: " + $p);
end

rule "Regola CdR 2"
when
  $p1: Person()
  $p2: Person()
then
  Pair p = new Pair();
  p.setFirst($p1);
  p.setSecond($p2);
  insert(p);
  // oppure
  insert(new Pair($p1, $p2));
end
```

Concatenazione di Regole

- Generare tutte le coppie di `Person` ordinate e distinte, inserendole in WM come `Pair`

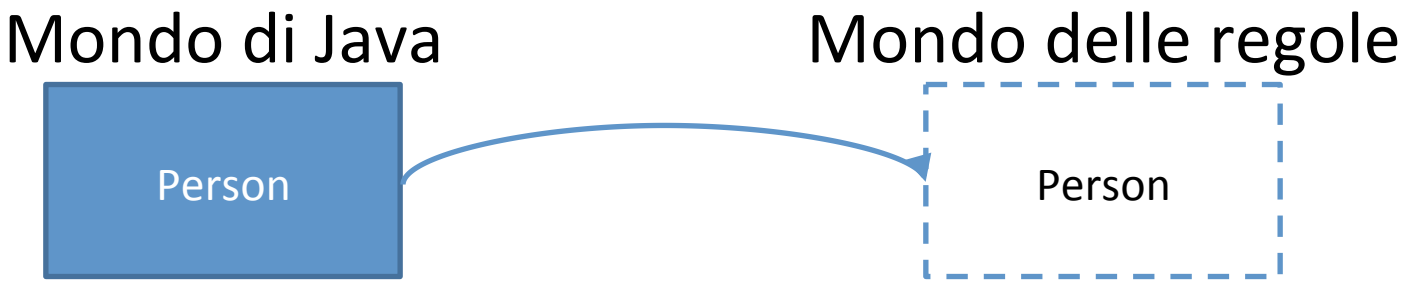
```
rule "Regola CdR 3"
when
  ...
then
  Pair p = new Pair();
  p.setFirst($p1);
  p.setSecond($p2);
  insert(p);
  // oppure
  insert(new Pair($p1, $p2));
end
```

Concatenazione di Regole

- Generare tutte le coppie di `Person` ordinate e distinte, inserendole in WM come `Pair`

```
rule "Regola CdR 3"
when
  $p1: Person()
  $p2: Person( this != $p1 )
  // Verifico che non ci sia la coppia simm.
  not Pair( first == $p2, second == $p1 )
then
  Pair p = new Pair();
  p.setFirst($p1);
  p.setSecond($p2);
  insert(p);
  // oppure
  insert(new Pair($p1, $p2));
end
```

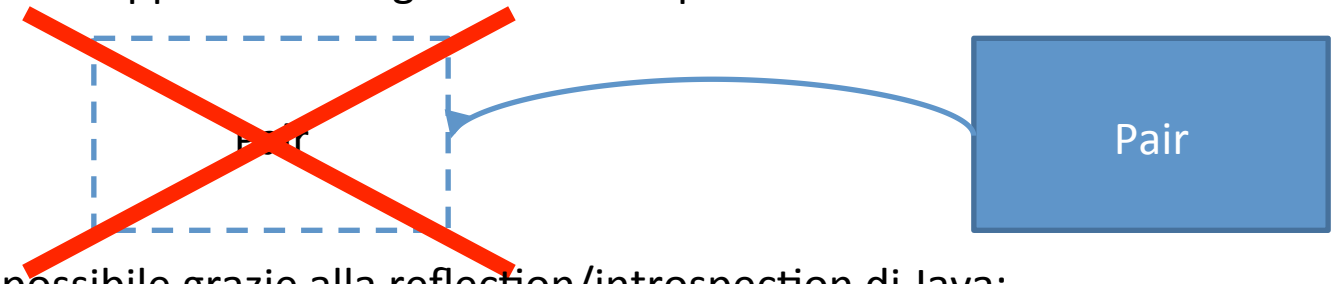
Concatenazione di Regole



La classe `Person`, definita in Java, diventa disponibile nel mondo delle regole mediante:

```
import com.sample.Person;
```

L'operazione opposta non è generalmente possibile.



Diventa possibile grazie alla reflection/introspection di Java:

```
FactType pairClass = ksession.getKnowledgeBase().getFactType("com.sample", "Pair");
Object pairObject = pairClass.newInstance();
pairClass.set(pairObject, "first", p1); pairClass.set(pairObject, "second", p2);
ksession.insert(pairObject);
```

Esercizi

FEATURES AVANZATE: FORM, COLLECT, ACCUMULATE

Features Avanzate: FROM

- Il costrutto **`from`** consente di accedere agli oggetti di una `Collection`, anche se questi non sono esplicitamente presenti nella WM

```
rule "Esempio di FROM"
when
  $l: List()   // inserita come List<? extends Person>
  $p: Person() from $l          // come un join classico
then
  System.out.println($l + "/" + $p);
end
```

NB: Scatta (fino a) `$l.size()` volte!

Features Avanzate: FROM

- Aggiungere a `Person` un campo `children` di tipo `Collection<? extends Person>`
- Scrivere una regola che stampi i figli di ogni persona

```
rule "Regola FA:FROM 1"
when
  ...
then
  ...
end
```

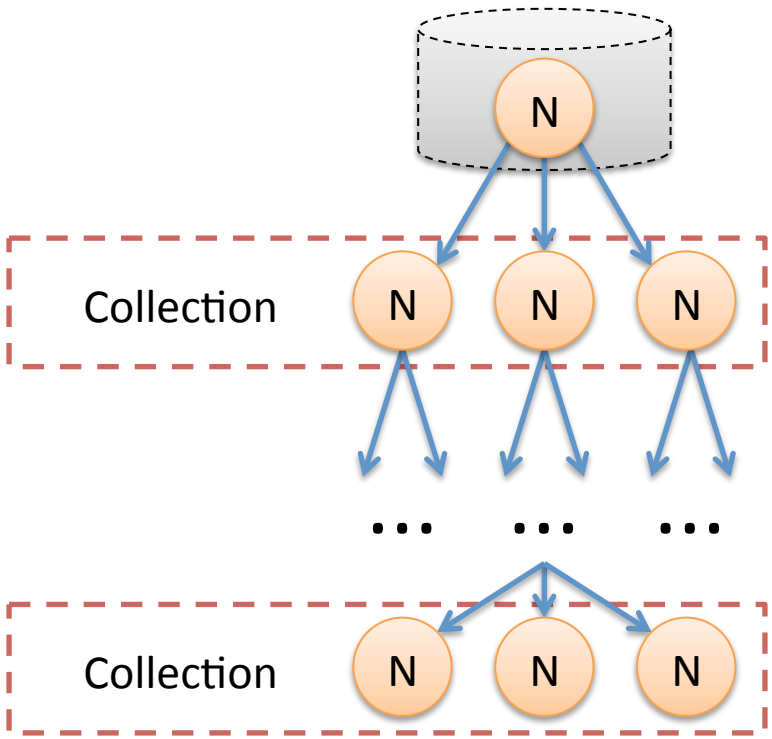
Features Avanzate: FROM

- Aggiungere a `Person` un campo `children` di tipo `Collection<? extends Person>`
- Scrivere una regola che stampi i figli di ogni persona

```
rule "Regola FA:FROM 1"
when
  $parent: Person( $children: children )
  $child:  Person() from $children
then
  System.out.println("FA:FROM1 " + $parent
            + " è genitore di " + $child);
end
```

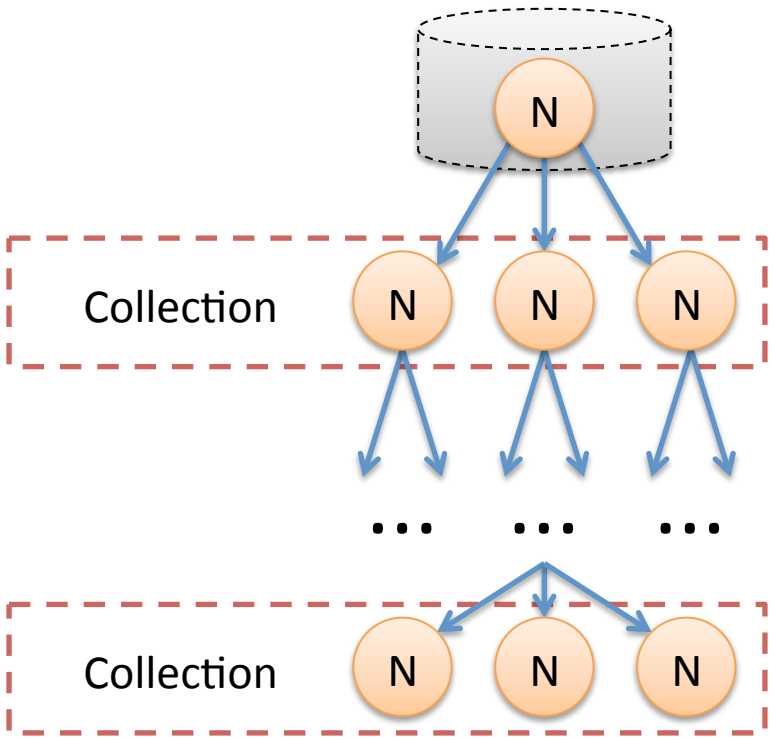
Features Avanzate: FROM

- Definire un POJO/BEAN che catturi l'astrazione di *nodo di un albero*
- Solo il nodo radice è asserito nella WM



Features Avanzate: FROM

- Definire un POJO/BEAN che catturi l'astrazione di *nodo di un albero*
- Solo il nodo radice è asserito nella WM
- Implementare un *Visitor* mediante un set di regole che attraversi l'albero visitando tutti i nodi in ordine qualsiasi



Features Avanzate: FROM

- Definire un POJO/BEAN che catturi l'astrazione di *nodo di un albero*
- Solo il nodo radice è asserito nella WM
- Implementare un *Visitor* mediante un set di regole che attraversi l'albero visitando tutti i nodi in ordine qualsiasi

```
rule "Regola FA:FROM 2"
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

Features Avanzate: FROM

- In quale ordine sono visitati i nodi?
- Del tutto casuale: alcuni visitati in ampiezza, altri in profondità…

```
rule "Regola FA:FROM 2"
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

Features Avanzate: FROM

- In quale ordine sono visitati i nodi?
- Del tutto casuale: alcuni visitati in ampiezza, altri in profondità…
- La **salience** fissa la priorità delle regole e impone il strategia di attraversamento dell'albero

```
rule "Regola FA:FROM 2"
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

Features Avanzate: FROM

- Strategia di ricerca:
  - Breadth-first
  - Depth-first

```
rule "Regola FA:FROM 2"
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

NB: Le attivazioni delle due regole vengono comunque inframmezzate.
È possibile implementare correttamente queste strategia di ricerca complicando un po' i pattern.

Features Avanzate: FROM

- Strategia di ricerca:
  - Breadth-first
  - Depth-first

```
rule "Regola FA:FROM 2”
salience 10
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
salience 5
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

NB: Le attivazioni delle due regole vengono comunque inframmezzate.
È possibile implementare correttamente queste strategia di ricerca complicando un po’ i pattern.

Features Avanzate: FROM

- Strategia di ricerca:
  - Breadth-first
  - Depth-first

```
rule "Regola FA:FROM 2"
salience 5
when
  $n: Node()
then
  System.out.println("FA:FROM2 "
                + $n + " visitato");
end

rule "Regola FA:FROM 3"
salience 10
when
  Node( $children: children )
  $child: Node() from $children
then
  insert($child);
end
```

NB: Le attivazioni delle due regole vengono comunque inframmezzate.
È possibile implementare correttamente queste strategia di ricerca complicando un po' i pattern.

Features Avanzate: COLLECT

- Il costrutto **`collect`** è duale di **`form`** e produce una `Collection` accorpando gli oggetti presenti nella WM che corrispondono ad un certo pattern

```
rule "Esempio di COLLECT"
when
  $c: collect (
        Person()
      ) // costruita come Collection<? extends Person>
  $p: Person() from $c          // come un join classico
then
  System.out.println($c + "/" + $p);
end
```

NB: Scatta esattamente `$c.size()` volte!

Features Avanzate: ACCUMULATE

- Il costrutto **`accumulate`** è concettualmente simile a **`collect`**
- Effettua operazioni generiche su `Collection` di oggetti costruite implicitamente a partire da un pattern
- Si compone di più parti (opzionali)
- Esistono abbreviazioni per le funzioni notevoli: **`min`**, **`max`**, **`count`**, ecc.

Features Avanzate: ACCUMULATE

- Sintassi generale:

```
…
$y: Y(…) from accumulate (
   $p: P(…)
   init(    x = new X(); ) // inizializza…
   action( x = f(x, $p)   ) // accumula/esegui…
   return( new Y(x)       ) // restituisce…
)          // qualcosa filtrabile come Y(…)
…
```

Features Avanzate: ACCUMULATE

- Funzioni notevoli:

```
rule "Esempio di Funzioni notevoli"
when
  accumulate( Person( $a: age ),
    $max: max( $a ),
    $min: min( $a ),
    $avg: average( $a )
  )
then
  System.out.println(
      "M:" + $max + " m:" + $min + " a:" + $avg);
end
```

Features Avanzate: ACCUMULATE

- Scrivere una regola per trovare tutte le persone più giovani della somma dell'età dei loro figli

```
rule "Regola FA:ACCUMULATE 1"
when
  ...
then
  System.out.println("FA:ACCUMULATE1 " + $p);
end
```

Features Avanzate: ACCUMULATE

- Scrivere una regola per trovare tutte le persone più giovani della somma dell'età dei loro figli

```
rule "Regola FA:ACCUMULATE 1"
when
  $p: Person( $age: age,
              $children: children )
  Number( this.intValue() > $age )
    from accumulate (
      Person( $a: age ) from $children,
      init(   int total = 0; ),
      action( total += $a; ),
      result( new Integer(total) )
    )
then
  System.out.println("FA:ACCUMULATE1 " + $p);
end
```

Features Avanzate: ACCUMULATE

- Scrivere una regola per trovare tutte le persone più giovani della somma dell'età dei loro figli

```
// Formulazione alternativa, più semplice
rule "Regola FA:ACCUMULATE 1"
when
$p: Person( $children: children )
  accumulate(
    Person( $a: age ) from $children,
    $sum: sum($a) )
  Person( this == $p, age < $sum )
then
  System.out.println("FA:ACCUMULATE1 " + $p);
end
```

Esercizi

QUERIES

Queries

- Consentono di individuare insiemi di oggetti presenti nella WM che rispondono a certe caratteristiche
- Sfruttano le capacità di filtraggio di RETE (sono come regole senza conseguente)
- Ammettono l'uso di parametri con cui è anche possibile accedere ai risultati

Queries

- Individuare le persone maggiorenni

```
query "Query: Q1"
  ...
end
```

Queries

- Individuare le persone maggiorenni

```
query "Query: Q1"
  $p: Person( age > 18 )
end
```

Queries

- Individuare le persone maggiorenni
- Individuare le persone che hanno una data età

```
query "Query: Q1"
  $p: Person( age > 18 )
end

query "Query: Q2" ( int $a )
  ...
end
```

Queries

- Individuare le persone maggiorenni
- Individuare le persone che hanno una data età

```
query "Query: Q1"
  $p: Person( age > 18 )
end

query "Query: Q2" ( int $a )
  Person( $n: name, age == $a )
end
```

Queries

- Individuare le persone maggiorenni
- Individuare le persone che hanno una data età
- Individuare le persone che hanno una data età o viceversa, o entrambe

```
query "Query: Q1"
  $p: Person( age > 18 )
end

query "Query: Q2" ( int $a )
  Person( $n: name, age == $a )
end

query "Query: Q3" ( String $n, int $a )
  ...
end
```

Queries

- Individuare le persone maggiorenni
- Individuare le persone che hanno una data età
- Individuare le persone che hanno una data età o viceversa, o entrambe

```
query "Query: Q1"
  $p: Person( age > 18 )
end

query "Query: Q2" ( int $a )
  Person( $n: name, age == $a )
end

query "Query: Q3" ( String $n, int $a )
  Person( $n := name, $a := age )
end
```

Queries

- Mondo di Java

```
ksession.fireAllRules();
QueryResults qr = ksession.
  getQueryResults("Q1");
for (QueryResultsRow qrr: qr)
  System.out.println(qrr);
```

- Mondo delle regole

```
$c: collect ?"Q3"( $n, 18 )
```

Esercizi

TRUTH MAINTENANCE

Truth Maintenance

- Oltre ad asserire e ritrarre fatti * (costrutti **`insert`** e **`retract`**) è possibile farlo *logicamente*
- Ovvero i fatti, che vengono *asseriti* (*ritratti*) *al verificarsi di una condizione* (il matching di un pattern), vengono ***automaticamente*** *ritratti* (*asseriti*) quando *la condizione non sussiste più*
- I nuovi costrutti sono **`insertLogical`** e **`retractLogical`**

* Esiste anche **`modify`** che fa in sequenza **`retract`** e **`insert`**

Truth Maintenance

- Partendo dal solito progetto di partenza (`Person` e `Mail` e relative istanze da Java), scrivere un insieme di regole che realizza un semplice servizio pubblicitario che adatta la pubblicità all'età delle persone

Truth Maintenance

- Dichiarare un fatto `Ad` che contiene un riferimento a `Person` e un `message` (`String`) e un fatto `Birthday` con un riferimento a `Person`
- Definire una query con cui individuare i `message` di `Ad`

...

Truth Maintenance

- Dichiarare un fatto `Ad` che contiene un riferimento a `Person` e un `message` (`String`) e un fatto `Birthday` con un riferimento a `Person`
- Definire una query con cui individuare i `message` di `Ad`

```
declare Ad
  target: Person
  message: String
end

declare Birthday
  person: Person
end

query "advertisement" ( Person $p )
  Ad( target == $p, $m: message )
end
```

Truth Maintenance

- Scrivere una regola che suggerisca l'acquisto di un giocattolo ad ogni persona minorenne
- Scrivere una regola che suggerisca l'acquisto di una automobile ad ogni persona maggiorenne

…

NB: operazioni logiche!

Truth Maintenance

- Scrivere una regola che suggerisca l'acquisto di un giocattolo ad ogni persona minorenne
- Scrivere una regola che suggerisca l'acquisto di una automobile ad ogni persona maggiorenne

```
rule "Toy to Kids"
when
  $p: Person( age < 18 )
then
  insertLogical(new Ad($p, $p.name +
    ", compra un'automobilina!"));
end

rule "Car to Men"
when
  $p: Person( age >= 18 )
then
  insertLogical(new Ad($p, $p.name +
    ", compra una spider!"));
end
```

NB: operazioni logiche!

Truth Maintenance

- Scrivere una regola che, per ogni `Ad`, mandi una mail (crei un oggetto `Mail`) usando i campi dell'`Ad` (`sender == recipient`)
- Scrivere una regola che, ogni volta che viene notificato un `Birthday`, aggiorna l'età della relativa `Person`

…

Truth Maintenance

- Scrivere una regola che, per ogni `Ad`, mandi una mail (crei un oggetto `Mail`) usando i campi dell'`Ad` (`sender == recipient`)
- Scrivere una regola che, ogni volta che viene notificato un `Birthday`, aggiorna l'età della relativa `Person`

```
rule "Send a Mail"
when
  Ad( $t: target, $m: message )
then
  insert(new Mail($t, $t, $m));
end

rule "Grow older!"
when
  $p: Person( $a: age )
  $b: Birthday( person == $p )
then
  System.out.println(
   "Buon compleanno, " + $p.getName());
  modify($p) {
    age = $a + 1;
    // $p.setAge($a + 1);
  }
  retract($b);
end
```

Truth Maintenance

1. Da Java: inserire un `Birthday` per una `Person`
2. Far scattare le regole sulla sessione
3. Verificare che quali `Ad` sono in WM mediante l'apposita **`query`**

- Ripetere i passi precedenti finchè una `Person` diventa maggiorenne

- Cosa succede? Perchè?

Fondamenti di Intelligenza Artificiale M

INFORMAZIONI

Informazioni

- Possibilità di svolgere attività progettuali o tesi
  - *Drools, Event Calculus, Expectations, MS-Kinect, Android SDK, SOA/Cloud, …*

- Per domande, dubbi, richieste:

stefano.bragaglia@unibo.it