A Pure Java Solution for DMCommunity Challenge Benchmark “Medical Services”

This challenge is described at https://dmcommunity.org/challenge/challenge-feb-2021/. It provides a set of rules that specify the coverage for medical services. There are 16,369 rules in an CSV-file and a several test cases with input medical services in a JSON-file.

I decided to write a pure Java rule engine to solve this problem and test its performance without any real off-the-shelf rule engines. I called it “RuleEngine” – see the source code in the Appendix. My engine uses the inner class “Rule” with interfaces for Condition and Action:

```
class Rule {
    Condition[] conditions;
    Action[] actions;
```

It uses only two simple Condition classes: ConditionString and ConditionDate with simple compare methods, and the class Action that assigns a value to a string.

The class **Rule** has two methods “**isSatisfied**” and “**apply**”:

```
public boolean isSatisfied(MedicalService service) {
    for (Condition c : conditions) {
        if ("Place Of Service".equals(c.getName()))
            if (!c.compare(service.getPlaceOfService()))
                return false;
        if ("Service Type".equals(c.getName()))
            if (!c.compare(service.getType()))
                return false;
        if ("Plan".equals(c.getName()))
            if (!c.compare(service.getPlan()))
                return false;
        if ("Group Size".equals(c.getName()))
            if (!c.compare(service.getGroupSize()))
                return false;
        if ("In Network".equals(c.getName()))
            if (!c.compare(service.getInNetwork()))
                return false;
        if ("Is Covered".equals(c.getName()))
            if (!c.compare(service.getIsCovered()))
                return false;
        if ("Service Type".equals(c.getName()))
            if (!c.compare(service.getType()))
                return false;
        if ("Effective Period Start".equals(c.getName()))
            if (!c.compare(service.getDateOfService()))
                return false;
        if ("Effective Period End".equals(c.getName()))
            if (!c.compare(service.getDateOfService()))
                return false;
    }
    return true;
}
```

and

```
public void apply(MedicalService service) {
    for (Action a : actions) {
        if ("Covered in Full".equals(a.getName())) {
            service.setCoveredInFull(a.getValue());
            continue;
        }
        if ("Copay".equals(a.getName())) {
            service.setCopay(a.getValue());
            continue;
        }
        if ("Coinsurance".equals(a.getName())) {
            service.setCoInsurance(a.getValue());
            continue;
        }
    }
}
```

The class **EngineJava** has the following methods:

- setRules(filename) – reads all rules from an csv file
- execute(MedicalService) – execute the engine against one medical service:

```
public boolean execute(MedicalService service) {
    int n = 0;
    for (Rule rule : rules) {
        n++;
        if (rule.isSatisfied(service)) {
            System.out.println("Rule " + n + " is satisfied");
            rule.apply(service);
            return true;
        }
    }
    System.out.println("No rules satisfied for "+service);
    return false;
}
```

Here is a simple Java bean “MedicalService”:

```
public class MedicalService {

    String placeOfService;
    String type;
    String plan;
    String groupSize;
    String inNetwork;
    String isCovered;
    Date dateOfService;
    String coveredInFull;
    String copay;
    String coInsurance;

    // setters and getters

}
```

Here is a test to read rules and data and to measure the performance:

```
public class Test {

    public static void main(String[] args) throws Exception {

        ObjectMapper mapper = new ObjectMapper();
        DateFormat dt = new SimpleDateFormat("yyyy-MM-dd");
        mapper.setDateFormat(dt);

        long startTime = System.currentTimeMillis();
        MedicalService[] services = mapper.readValue(new File("data/Request.json"), MedicalService[].class);
        List<MedicalService> response = new ArrayList<MedicalService>();

        RuleEngine engine = new RuleEngine();
        engine.setRules("data\\MedicalServicesDecisionTable.csv");

        long endTime = System.currentTimeMillis();
        System.out.println("Engine created in " + (endTime - startTime +1) + " ms");

        int n = 0;
        double average = 0;
        // System.out.println("get update " + elapsed / 1000000 + "ms");
        for (MedicalService service : services) {
            n++;
            System.out.println("\nService #" + n);
            double serviceSart = System.nanoTime();
            engine.execute(service);
            double serviceEnd = System.nanoTime();
            double elapsed = (serviceEnd - serviceSart +1)/1000000;
            System.out.println("Service " + n + " execution time: " + elapsed + " ms");
                average += elapsed;
            response.add(service);
        }
        System.out.println("\nAverage service execution time: " + average/(n-1) + " ms");

        mapper.writerWithDefaultPrettyPrinter().writeValue(new File("data/Response.json"), response);

        endTime = System.currentTimeMillis();
        System.out.println("Total elapsed time: " + (endTime - startTime +1) + " ms");
    }
}
```

Here are the execution results with the performance metrics:

```
Engine created in 236 ms

Service #1
Rule 13 is satisfied
Service 1 execution time: 0.143101 ms

Service #2
Rule 30 is satisfied
Service 2 execution time: 0.104301 ms

Service #3
Rule 16370 is satisfied
Service 3 execution time: 5.858801 ms

Service #4
Rule 7875 is satisfied
Service 4 execution time: 2.053501 ms

Service #5
Rule 10369 is satisfied
Service 5 execution time: 2.695101 ms

Service #6
Rule 16369 is satisfied
Service 6 execution time: 4.365301 ms

Service #7
Rule 16356 is satisfied
Service 7 execution time: 3.594101 ms

Service #8
Rule 14995 is satisfied
Service 8 execution time: 3.373701 ms

Service #9
Rule 9995 is satisfied
Service 9 execution time: 1.754601 ms

Service #10
Rule 13071 is satisfied
Service 10 execution time: 2.231701 ms

Average service execution time: 2.9082455555555553 ms
Total elapsed time: 320 ms
```

Thus, a pure Java solution created specifically for this problem and being executed on a regular laptop shows the average performance of **2.9 milliseconds.**

Appendix.

```
package org.medical.services;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;

public class RuleEngine {

   ArrayList<Rule> rules;
   int numberOfConditions = 8;
   int numberOfActions = 3;
   DateFormat df = new SimpleDateFormat("yyyy-MM-dd");

   public void setRules(String fileName) throws Exception {
      rules = new ArrayList<Rule>();
      String[] columns = null;
      String line = "";
      int l = 0;
      try {
         BufferedReader reader = new BufferedReader(new FileReader(fileName));
         boolean header = true;
         while ((line = reader.readLine()) != null) {
            l++;
            String[] elements = line.split(",");
            if (elements.length != (numberOfConditions + numberOfActions)) {
               System.out.println("Invalid CSV file " + fileName + " line=" + l);
               System.exit(-1);
            }
            if (header) {
               header = false;
               columns = elements;
               continue;
            }
            Rule rule = new Rule(numberOfConditions, numberOfActions);
            rules.add(rule);
            for (int i = 0; i < numberOfConditions; i++) {
               if (i < 6) { // strings
                  rule.setCondition(i, new ConditionString(columns[i], elements[i]));
                  continue;
               }
               Date date = null;
               if (!elements[i].isEmpty())
                  date = df.parse(elements[i]);
               if (i == 6) {
                  rule.setCondition(i, new ConditionDate(columns[i], "<=", date));
               }
```

```
            if (i == 7) {
                rule.setCondition(i, new ConditionDate(columns[i], ">=", date));
            }
        }
        for (int i = 0; i < numberOfActions; i++) {
            int index = numberOfConditions + i;
            rule.setAction(i, new Action(columns[index], elements[index]));
        }
    }
    reader.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

public boolean execute(MedicalService service) {
  int n = 0;
  for (Rule rule : rules) {
    n++;
    if (rule.isSatisfied(service)) {
      System.out.println("Rule " + n + " is satisfied");
      rule.apply(service);
      return true;
    }
  }
  System.out.println("No rules satisfied for "+service);
  return false;
}

public void showRules() {
  int n = 0;
  for (Rule rule : rules) {
    n++;
    System.out.println(n + ": " + rule);
  }
}

public static void main(String[] args) {
  RuleEngine engine = new RuleEngine();

  try {
    engine.setRules("data\\MedicalServicesDecisionTable.csv");
  } catch (Exception e) {
    e.printStackTrace();
  }
  engine.showRules();
}

class Rule {
  Condition[] conditions;
  Action[] actions;

  public Rule(int numberOfConditions, int numberOfActions) {
```

```
    conditions = new Condition[numberOfConditions];
    actions = new Action[numberOfActions];
}

public void setCondition(int index, Condition cond) {
    conditions[index] = cond;
}

public void setAction(int index, Action action) {
    actions[index] = action;
}

public boolean isSatisfied(MedicalService service) {
    for (Condition c : conditions) {
        if ("Place Of Service".equals(c.getName()))
            if (!c.compare(service.getPlaceOfService()))
                return false;
        if ("Service Type".equals(c.getName()))
            if (!c.compare(service.getType()))
                return false;
        if ("Plan".equals(c.getName()))
            if (!c.compare(service.getPlan()))
                return false;
        if ("Group Size".equals(c.getName()))
            if (!c.compare(service.getGroupSize()))
                return false;
        if ("In Network".equals(c.getName()))
            if (!c.compare(service.getInNetwork()))
                return false;
        if ("Is Covered".equals(c.getName()))
            if (!c.compare(service.getIsCovered()))
                return false;
        if ("Service Type".equals(c.getName()))
            if (!c.compare(service.getType()))
                return false;
        if ("Effective Period Start".equals(c.getName()))
            if (!c.compare(service.getDateOfService()))
                return false;
        if ("Effective Period End".equals(c.getName()))
            if (!c.compare(service.getDateOfService()))
                return false;
    }
    return true;
}

public void apply(MedicalService service) {
    for (Action a : actions) {
        if ("Covered in Full".equals(a.getName())) {
            service.setCoveredInFull(a.getValue());
            continue;
        }
        if ("Copay".equals(a.getName())) {
            service.setCopay(a.getValue());
```

```
            continue;
         }
         if ("Coinsurance".equals(a.getName())) {
            service.setCoInsurance(a.getValue());
            continue;
         }
      }
   }

   @Override
   public String toString() {
      return "RuleJava [conditions=" + Arrays.toString(conditions) + ", actions=" + Arrays.toString(actions)
            + "]";
   }

}

abstract public class Condition {
   String name;

   abstract public boolean compare(Object par);

   public String getName() {
      return name;
   }
}

class ConditionString extends Condition {

   String value;

   public ConditionString(String name, String value) {
      this.name = name;
      this.value = value;
   }

   public boolean compare(Object par) {
      if (value.isEmpty())
         return true;
      return value.equals(par);
   }

   @Override
   public String toString() {
      return name + "=" + value;
   }

}

class ConditionDate extends Condition {
   Date date;
   String oper;
```

```
    public ConditionDate(String name, String oper, Date date) {
      this.name = name;
      this.oper = oper;
      this.date = date;
    }

    public boolean compare(Object par) {
      if (date == null)
        return true;
      int comparison = date.compareTo((java.util.Date) par);
      if (oper.contentEquals("<=") && comparison <= 0)
        return true;
      if (oper.contentEquals(">=") && comparison >= 0)
        return true;
      return false;
    }

    @Override
    public String toString() {
      return name + oper + date;
    }
  }


  class Action {
    String name;
    String value;

    public Action(String name, String value) {
      this.name = name;
      this.value = value;
    }

    public void assign(String par) {
      value = par;
    }

    public String getName() {
      return name;
    }

    public String getValue() {
      return value;
    }

    @Override
    public String toString() {
      return name + "=" + value;
    }
  }
}
```