Challenge July-2022

Evaluate Team Performance

https://dmcommunity.org/challenge/challenge-july-2022/

Solutions using IBM ODM by Andrew Macdonald – 29 July 2022

Team	Player	Game Date	Efficiency
Mustungs	Brown	4/1/2022	good
	Brown	4/2/2022	better
	Brown	4/3/2022	best
	Robinson	4/1/2022	worst
	Robinson	4/2/2022	better
	Robinson	4/3/2022	best
	Smith	4/1/2022	bad
	Smith	4/2/2022	good
	Smith	4/3/2022	bad
Eagles	Black	4/1/2022	good
	Black	4/2/2022	better
	Black	4/3/2022	best
	White	4/1/2022	worst
	White	4/2/2022	better
	White	4/3/2022	best
	Green	4/1/2022	bad
	Green	4/2/2022	good
	Green	4/3/2022	worst

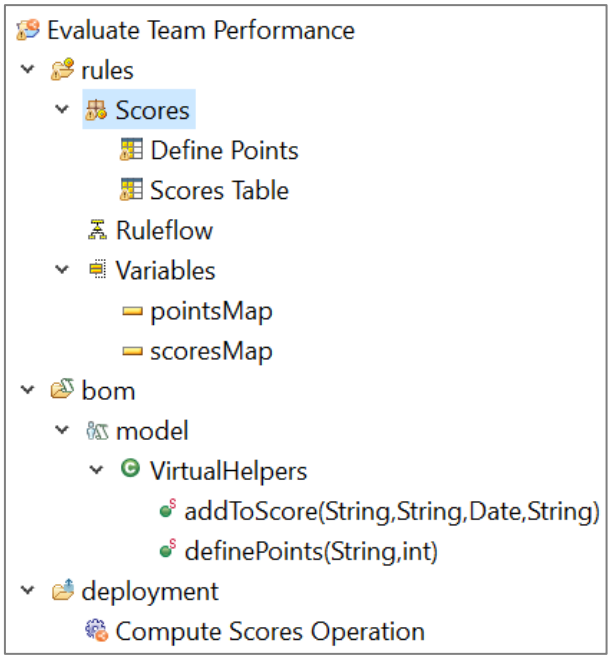
Your decision model should evaluate performance of different teams based on efficiency of their players. On the left you can see how different players performed during different games. Each player receives 5 points for "best" efficiency, 3 points for "better" efficiency, 2 for "good" efficiency. You need to subtract 2 points for "bad" efficiency and 5 points for "worst" efficiency. Which team got the most points?

Solution 1 - *Basic*

This is a simple requirement that is more of a lookup exercise rather than a complex decision service. It can therefore be implemented in a number of ways. As a very quick first test I implemented this as a self-contained set of rules that have no input/output parameters. It was built to solve the problem rather than provide a flexible and extensible decision service (see solution 2 for this).

It follows my usual design principles to keep things simple for the business rule authors and to match the given requirements and data structures where possible. It is low code (just two one line methods) and these where written to make the rules easy to understand and maintain.

The ODM project structure is shown here. The rule logic is contained in a couple of decision table rules.



In this simple example I avoided creating any bespoke data structures by using a couple of ruleset variables of type Map to hold the efficiency points and the scores for each team. Using maps means that it is easy to expand the number of points categories and teams simply by adding them in the decision table rules.

I used a ruleflow as a simple way to initialise the maps and add some trace printouts in the final action after the rules had fired to display the results.

The ‘Define Points’ decision table looks like this. I use a condition that is always true so that every row in the table is evaluated. It also gives you the ability to easily disable individual rows. Using a decision table makes it very easy for non-technical users to change the points and to add/remove efficiency type names.

set <efficiency> points to <a number>

	Active?	Define score as	
		efficiency	a number
1	true	best	5
2	true	better	3
3	true	good	2
4	true	bad	-2
5	true	worst	-5

The action column of the table calls a method to add the points for each efficiency into the pointsMap using one line of code:

```
pointsMap.put(efficiency, points);
```

The 'Scores Table' rule was constructed to be in the same format as the problem description so that if the values were in a CSV/Excel file then the values can be copied directly into the rule:

AddToScore <team> <player> <date> <efficiency>

	Active?	Add Performance to Score			
		team	player	date	efficiency
1	true	Mustungs	Brown	04-Jan-2022	good
2	true	Mustungs	Brown	04-Jan-2022	better
3	true	Mustungs	Brown	04-Jan-2022	best
4	true	Mustungs	Robinson	04-Jan-2022	worst
5	true	Mustungs	Robinson	04-Jan-2022	better
6	true	Mustungs	Robinson	04-Jan-2022	best
7	true	Mustungs	Smith	04-Jan-2022	bad
8	true	Mustungs	Smith	04-Jan-2022	good
9	true	Mustungs	Smith	04-Jan-2022	bad
10	true	Eagles	Black	04-Jan-2022	good
11	true	Eagles	Black	04-Jan-2022	better
12	true	Eagles	Black	04-Jan-2022	best
13	true	Eagles	White	04-Jan-2022	worst
14	true	Eagles	White	04-Jan-2022	better
15	true	Eagles	White	04-Jan-2022	best
16	true	Eagles	Green	04-Jan-2022	bad
17	true	Eagles	Green	04-Jan-2022	good
18	true	Eagles	Green	04-Jan-2022	worst

The action column in this case is essentially calling one line of code to lookup the points value from the PointsMap and then add that to the value to the appropriate team in the ScoresMap:

```
System.out.println(" Adding score "+team+" "+player+" "+date+" "+efficiency+
                   " = "+(Integer)pointsMap.get(efficiency) );
int newScore = (Integer)scoresMap.get(team) + (Integer)pointsMap.get(efficiency);
scoresMap.replace(team, newScore);
```

Running the rules gives the following results:

```
### Starting ruleflow
 Adding score Mustungs Brown Tue Jan 04 13:51:34 GMT 2022 good = 2
 Adding score Mustungs Brown Tue Jan 04 13:51:34 GMT 2022 better = 3
 Adding score Mustungs Brown Tue Jan 04 13:51:34 GMT 2022 best = 5
 Adding score Mustungs Robinson Tue Jan 04 13:51:34 GMT 2022 worst = -5
 Adding score Mustungs Robinson Tue Jan 04 13:51:34 GMT 2022 better = 3
 Adding score Mustungs Robinson Tue Jan 04 13:51:34 GMT 2022 best = 5
 Adding score Mustungs Smith Tue Jan 04 13:51:34 GMT 2022 bad = -2
 Adding score Mustungs Smith Tue Jan 04 13:51:34 GMT 2022 good = 2
 Adding score Mustungs Smith Tue Jan 04 13:51:34 GMT 2022 bad = -2
 Adding score Eagles Black Tue Jan 04 13:51:34 GMT 2022 good = 2
 Adding score Eagles Black Tue Jan 04 13:51:34 GMT 2022 better = 3
 Adding score Eagles Black Tue Jan 04 13:51:34 GMT 2022 best = 5
 Adding score Eagles White Tue Jan 04 13:51:34 GMT 2022 worst = -5
 Adding score Eagles White Tue Jan 04 13:51:34 GMT 2022 better = 3
 Adding score Eagles White Tue Jan 04 13:51:34 GMT 2022 best = 5
 Adding score Eagles Green Tue Jan 04 13:51:34 GMT 2022 bad = -2
 Adding score Eagles Green Tue Jan 04 13:51:34 GMT 2022 good = 2
 Adding score Eagles Green Tue Jan 04 13:51:34 GMT 2022 worst = -5
### Ended ruleflow
  -- Mustungs score = 11
  -- Eagles score = 8
```

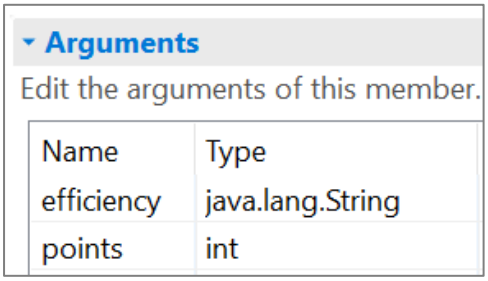
So a non-technical user could add more rows, say with 100 teams, and it would calculate the totals correctly:

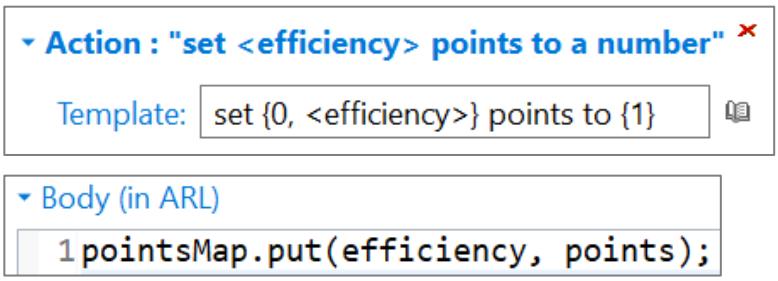
16	true	Eagles	Green	04-Jan-2022	bad
17	true	Eagles	Green	04-Jan-2022	good
18	true	Eagles	Green	04-Jan-2022	worst
19	true	IBM	AndyMac	29-Jul-2022	best

Which gives:

```
 Adding score Eagles Green Tue Jan 04 13:51:34 GMT 2022 worst = -5
 Adding score IBM AndyMac Fri Jul 29 17:01:06 BST 2022 best = 5
### Ended ruleflow
  -- Mustungs score = 11
  -- Eagles score = 8
  -- IBM score = 5
```

ODM provides an easy mechanism for adding custom code to the rule language. You can create an empty Business Object Model (BOM) and add any classes and methods you like. The 'definePoints' method was created as follows with the following arguments and verbalisation:





The Action verbalisation specifies how the method will appear in the rule language e.g.:

```
then
  set "best" points to 5 ;
```

This could be improved by using a dynamic domain of pre-defined efficient types rather than using a text string.

Solution 2 - *Regular Decision Service*

To implement this as a regular callable decision service we need to pass data to/from the service. This is easiest with custom data objects. For IBM ODM it is best to use Java objects but it could equally be XSD schema based or native data types.

For the inputs an array of simple Performance objects were used:

```
package odm;
import java.util.Date;

public class Performance {

	public String	team;
	public String	player;
	public Date	date;
	public String	efficiency;

	public Performance() {}

}
```

For the response object the following class was used which uses a Map to hold the scores for any number of teams and a Vector of Strings to hold a list of user defined messages. There is also one utility method to add the score for a player performance to the appropriate team and also add a reasoning message:

```
package odm;
import java.util.HashMap;
import java.util.Vector;

public class Result {

	public Vector<String> messages = new Vector<String>();
	public HashMap<String, Integer> teamScores = new HashMap<String,Integer>();

	public Result () {}

	public void addScore(Performance perf, int score) {
		System.out.println("Result.addScore - "+perf.team+" gets "
					+score+" points");
		if (teamScores.containsKey(perf.team)) {
			int newScore = teamScores.get(perf.team)+score;
			teamScores.replace(perf.team, newScore);
		}
		else {
			teamScores.put(perf.team, score);
		}
		messages.add(perf.team+" gets "+score+" points for "+perf.player+
				"'s "+perf.efficiency+ " performance on "+ perf.date);

	}
}
```

An array of Performances was created as the rule service input and a single Result as the output:

Variable Set: Variables

Name	Type	Verbalization
performances_in	odm.Performance[]	the performances
result_out	odm.Result	the result

Decision Operation Signature - Compute Scores Operation

Eligible variables

Select the ruleset variables that you want to use as parameters for the decision operation. Ruleset variables are defined in variable sets.

Input Parameters

Define the parameters required to call the execution.

Parameter name	Verbalization	Type
performances_in	the performances	odm.Performance[]

Output Parameters

Define the parameters that are initialized and returned by the execution.

Parameter name	Verbalization	Type
result_out	the result	odm.Result

This time only a single simple rule is required. It has a pre-condition which will cause the decision table rows to evaluate every entry of the performance array input:

Decision Table: Score each Performance

Preconditions

```
definitions
    set Player to a performance in 'the performances' ;
```

	Efficiency	add score
1	best	5
2	better	3
3	good	2
4	bad	-2
5	worst	-5

The condition column has this condition :

```
the efficiency of Player is <a string>
```

and the action column has this action:

```
add score of <score> for Player  to 'the result'
```

So when running the rules in the Decision Server with this JSON input:


```
{
  "performances_in": [
        {"team":"Mustungs", "player":"Brown", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"good"},
        {"team":"Mustungs", "player":"Brown", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"better"},
        {"team":"Mustungs", "player":"Brown", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"best"},
        {"team":"Mustungs", "player":"Robinson", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"worst"},
        {"team":"Mustungs", "player":"Robinson", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"better"},
        {"team":"Mustungs", "player":"Robinson", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"best"},
        {"team":"Mustungs", "player":"Smith", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"bad"},
        {"team":"Mustungs", "player":"Smith", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"good"},
        {"team":"Mustungs", "player":"Smith", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"bad"},
        {"team":"Eagles", "player":"Black", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"good"},
        {"team":"Eagles", "player":"Black", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"better"},
        {"team":"Eagles", "player":"Black", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"best"},
```

```
        {"team":"Eagles", "player":"White", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"worst"},
        {"team":"Eagles", "player":"White", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"better"},
        {"team":"Eagles", "player":"White", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"best"},
        {"team":"Eagles", "player":"Green", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"bad"},
        {"team":"Eagles", "player":"Green", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"good"},
        {"team":"Eagles", "player":"Green", "date": "2022-04-03T00:00:00.000+0100", "efficiency":"worst"}
  ]
}
```

The following results are returned:

```
{
  "__DecisionID__": "ab57dfcb-e2e4-466b-85bb-a6e7ddb728af0",
  "result_out": {
    "messages": [
      "Mustungs gets 2 points for Brown's good performance on Fri Apr 01 00:00:00 BST 2022",
      "Mustungs gets 3 points for Brown's better performance on Sat Apr 02 00:00:00 BST 2022",
      "Mustungs gets 5 points for Brown's best performance on Sun Apr 03 00:00:00 BST 2022",
      "Mustungs gets -5 points for Robinson's worst performance on Fri Apr 01 00:00:00 BST 2022",
      "Mustungs gets 3 points for Robinson's better performance on Sat Apr 02 00:00:00 BST 2022",
      "Mustungs gets 5 points for Robinson's best performance on Sun Apr 03 00:00:00 BST 2022",
      "Mustungs gets -2 points for Smith's bad performance on Fri Apr 01 00:00:00 BST 2022",
      "Mustungs gets 2 points for Smith's good performance on Sat Apr 02 00:00:00 BST 2022",
      "Mustungs gets -2 points for Smith's bad performance on Sun Apr 03 00:00:00 BST 2022",
      "Eagles gets 2 points for Black's good performance on Fri Apr 01 00:00:00 BST 2022",
      "Eagles gets 3 points for Black's better performance on Sat Apr 02 00:00:00 BST 2022",
      "Eagles gets 5 points for Black's best performance on Sun Apr 03 00:00:00 BST 2022",
      "Eagles gets -5 points for White's worst performance on Fri Apr 01 00:00:00 BST 2022",
      "Eagles gets 3 points for White's better performance on Sat Apr 02 00:00:00 BST 2022",
      "Eagles gets 5 points for White's best performance on Sun Apr 03 00:00:00 BST 2022",
      "Eagles gets -2 points for Green's bad performance on Fri Apr 01 00:00:00 BST 2022",
      "Eagles gets 2 points for Green's good performance on Sat Apr 02 00:00:00 BST 2022",
      "Eagles gets -5 points for Green's worst performance on Sun Apr 03 00:00:00 BST 2022"
    ],
     "teamScores": {
       "Mustungs": 11,
       "Eagles": 8
    }
  }
}
```

And if you change the name of any of the teams or add new ones :

```
{
  "performances_in": [
        {"team":"Cheetahs", "player":"Brown", "date": "2022-04-01T00:00:00.000+0100", "efficiency":"good"},
        {"team":"Mustungs", "player":"Brown", "date": "2022-04-02T00:00:00.000+0100", "efficiency":"better"},
```

then they will automatically appear in the results:

```
    "teamScores": {
      "Mustungs": 9,
       "Cheetahs": 2,
      "Eagles": 8
    }
```

The average execution time for this service running in the Rule Server is 1.6 milliseconds.