Assignment # 1 Report

Deliverables:

Sr. No.	File Name	Description
1.	run_ps.java	This is the main file from where the execution begins. To run the file type: **java run_ps wm rules**
2.	wm.java	This class file contains the working memory.
3.	rules.java	This class file contains the production rules.

Description of Functions included in run_ps (Algorithm and Program Logic):

1. **main()**
   - It takes input the class names where the working memory (facts) and production rules are specified. Both of these classes use ArrayList data structure to store the facts and production rules.
   - It calls *match_rules()* which does the job of applying the rules to the working memory. It keeps calling match_rules() until no new pattern Is returned by *match_rules()*, thus repeating the cycles.
   - After match_rules() returns the new pattern or assertions they are added to the working memory.
   - It prints the working memory [printWorkingMemory()]before each cycle of match_rules() is performed.
   - In the end when no new pattern is returned, it displays the additional facts that were added to the working memory.

2. **match_rules()**
   - It calls match_rule() on each rule and passess all the facts to this function.

3. **match_rule()**
   - It calls *mr_helper()* which takes as input the states (Antecedent list belonging to the single rule and the substitutions if any) and the facts. Initially substitution list is blank which is populated by mr_helper.
   - The antecedent list and the substitutions reside in an ArrayList data structure called state1 which is put inside a queue and passed to mr_helper alongwith facts.
   - If *mr_helper()* returns true it means there were new substitutions available and hence match_rule calls **execute()** which takes the substitution, the right-hand side of the rule (one or more consequents) and a working memory and applies the substitution to each of the consequents.
   - If no substitutions are generated then it returns a blank ArrayList.
4. mr_helper()
   - It gets the list of antecedents from the state object popped from the queue.

- If the list of antecedents is empty then it returns false.
- In case of non-zero antecedent list, it calls match_antecedent() passing the list of antecedents, substitutions and facts.
- If any substitutions are returned from match_antecedent() it returns true else it returns false. This is to give indication to match_rule() to check for any substitutions.

5. **match_antecedent()**
   - This function calls *ma_helper()* with each antecedent.
   - Every time this function is called for a fact, the fact is removed from the working memory and the remaining set of working memory is passed on to the next iteration of *match_antecedent()*.
   - If a particular antecedent is doesn't match that particular antecedent is removed from the head of the queue and placed in the end of the queue. This is because it is possible that the antecedent might match after some later antecedents are matched.
   - doContinue() checks if any iteration between multiple antecedents in a single state is a success. If it is then it moves to a new state. A State comprises of antecedents of a rule and the substitution list. This function lays an important role in avoiding infinite loop.

6. **ma_helper()**
   - This is a recursive function. It calls the unify function for every antecedent in the antecedent list and with every fact in the working memory.
   - The working memory set reduces in each recursive call until no fact is pending in the working memory set.
   - It calls *unify()* to check if an antecedent matches with a fact. If it matches then *success_state_antec* flag is set to true.
   - Whether or not unification succeeds, ma_helper is called until no fact is pending in the working memory set.
7. **Unify()**
   - It takes an antecedent (for example "%x very_high fever"), a fact (for example, Ed very_high fever) and list of substitutions (which initially is blank) and returns the substitution in the substitution list. (For instance, in this example it is (%x, Ed)).
   - It internally calls *isVariable()* to check if a particular string is a variable or not. It also calls *compare()* to compare the list of strings in the antecedent and the fact.
   - This function will apply an already set of available substitutions in the antecedent before proceeding with the matching of antecedent and the fact.

8. **execute() and substitute()**
   - *execute()* takes a substitution, the right hand side of a rule (one or more components), and a working memory as input. It applies the substitution to each of the consequents.

- *substitute()* is called internally by execute(). It takes a substitution (for example : (%x, Ed)) and a pattern (for example: (%x high_fever)) and returns the substituted pattern (i.e. Ed high_fever).

OUTPUT :

```
CYCLE 1

Current WM :
[Ed very_high fever]
[Ed cough]
[Alice no_poison_ivy]
[Max says Alice poison_ivy]
[Grace says Don healthy]
[Grace doctor]
[whooping_cough contagious]
[Ed contacts Alice]

Attempting to match rule 1 : Match Succeeds
Adding assertions to WM: Ed high_fever
Attempting to match rule 2 : Failed
Attempting to match rule 3 : Failed
Attempting to match rule 4 : Failed
Attempting to match rule 5 : Failed
Attempting to match rule 6 : Match Succeeds
Adding assertions to WM: Ed has whooping_cough
Attempting to match rule 7 : Failed
Attempting to match rule 8 : Failed
Attempting to match rule 9 : Failed
Attempting to match rule 10 : Match Succeeds
Adding assertions to WM: Grace true
Attempting to match rule 11 : Failed
Attempting to match rule 12 : Failed

CYCLE 2
Current WM :
[Ed very_high fever]
[Ed cough]
[Alice no_poison_ivy]
[Max says Alice poison_ivy]
[Grace says Don healthy]
[Grace doctor]
[whooping_cough contagious]
[Ed contacts Alice]
[Ed high_fever]
[Ed has whooping_cough]
[Grace true]

Attempting to match rule 1 : Assertion already exists
Ed high_fever
Attempting to match rule 2 : Assertion already exists
Ed cough
Attempting to match rule 3 : Failed
```

```
Attempting to match rule 4 : Failed
Attempting to match rule 5 : Failed
Attempting to match rule 6 : Assertion already exists
Ed has whooping_cough
Attempting to match rule 7 : Failed
Attempting to match rule 8 : Match Succeeds
Adding assertions to WM: Alice has whooping_cough
Attempting to match rule 9 : Failed
Attempting to match rule 10 : Assertion already exists
Grace true
Attempting to match rule 11 : Failed
Attempting to match rule 12 : Failed

CYCLE 3
Current WM :
[Ed very_high fever]
[Ed cough]
[Alice no_poison_ivy]
[Max says Alice poison_ivy]
[Grace says Don healthy]
[Grace doctor]
[whooping_cough contagious]
[Ed contacts Alice]
[Ed high_fever]
[Ed has whooping_cough]
[Grace true]
[Alice has whooping_cough]

Attempting to match rule 1 : Assertion already exists
Ed high_fever
Attempting to match rule 2 : Assertion already exists
Ed cough
Attempting to match rule 3 : Failed
Attempting to match rule 4 : Failed
Attempting to match rule 5 : Failed
Attempting to match rule 6 : Assertion already exists
Ed has whooping_cough
Attempting to match rule 7 : Failed
Attempting to match rule 8 : Assertion already exists
Alice has whooping_cough
Attempting to match rule 9 : Failed
Attempting to match rule 10 : Assertion already exists
Grace true
Attempting to match rule 11 : Failed
Attempting to match rule 12 : Failed

Following Assertions were added after applying the Production Rules :
[Ed high_fever]
[Ed has whooping_cough]
[Grace true]
[Alice has whooping_cough]
```