# Part 1:

First the user is asked if they would like through the prompt of input being between -1 to 5. 1 being the ability to add to the knowledge base; 2 to clear the current knowledge base; 3 to display the current knowledge base; 4 to enter a query; 5 to run the query that has been entered on the knowledge base and finally, -1 will exit out of part one and return to the main menu.

If 1 is chosen, the user will be prompted to enter the data into the knowledge base. This data is to be structured in [Literal1,Literal2,!Literal3] with the number of literals being able to be any number greater than one. The ‘¬’ symbol was replaced with the ‘!’ due to issues in Java in processing the symbol. The program gets the user input via starting an infinite while loop, that can be exited by inputting -1. If the input is not -1 then the program will create a new instance of the Clause object with the input that the user entered being the only parameter. The Clause object first takes the data and strips the surrounding square brackets from the input. It then splits up the string using the regex of ‘,’ it then loops through these new strings creating new Literal objects. It checks to see if the string contains the ‘!’ symbol if it does then the symbol is removed. When the Literal is made it takes two parameters, the string of the name of the literal and a boolean. The boolean marks if the literal is false. It then adds these new literals to a list called “listOfLiterals” which can be retrieved from the object later. This will continue until the user enters -1 to go back to the menu to choose another action.

If 2 is entered, then the knowledge base stored in the local space is set to null and the clearKnowledgeBase method in the InputManager class is called. This sets the knowledge base that is also stored in this class to null then creates a new empty ArrayList.

If 3 is inputted, then first it checks if the knowledge base is not null. If it is not, then it will start a for loop of the length of the knowledge base and output each element in the knowledge base. If the knowledge base is null, then it will do nothing.

4 prompts the user to input the query in the negative form and stores it in the Input Manager.

5 first makes sure the knowledge base is not null, the calls the method to run the query in the Query class. It then checks if the boolean returned by this function is true or false, if its true it will output ‘SOLVED’ otherwise it will output ‘NOT SOLVED’. The runQuery in the Query class works out if to return to true or false, first by taking 2 parameters the knowledgebase (ArrayList of clauses) and the query(Literal). First a boolean called *change* is defined then a new ArrayList of Literals called currentQuery the query that was passed to this method is then added to this new list. A do while loop is then started with the condition of *change* being true and will loop until change is false after all the subsequent loops. Within the first do while loop change is set to false, the knowledgebase is sorted by the length of the literals inside. Int i,j and k are declared and set to 0. If current query is empty, then the method will return true otherwise 3 nested do while loops are then created. The inner most loop is as follows:

1. Check if currentQuery(K)s data is equal to knowledgebase(i)s sub list literal (j)s data
2. Check to see if the above have opposite polarity if they are equal.
3. Remove k from the current query
4. The remove j from the knowledgebase.
5. Check if i is then empty if it is then remove it from the knowledge base otherwise add everything left in the Clause to the currentQuery
6. Increments k

This first loop will loop while k < the size of the current query and while change is false. The middle do while loop increments j and sets k back to 0. This loop will loop while j < than the size of the list of literals in element I in knowledge and while change is equal to false. The outer most loop increments I and sets j and k to 0 this will loop through until i < the size of the knowledgebase and change is not true. If it ever gets outside the look that checks if a change has been made this means, there is nothing that can cancel out in the knowledge base and query therefore the method returns false.

If anything, else other than the above defined inputs are inputted then the program will inform the user of an invalid inputs.

## Testing:

# Part 2:

Firstly, another loop is started once again using -1 to exit the loop, with two other options 1 and 2.

1 adds the information to the network first by starting a loop. This will loop until the user enters ‘-1’ otherwise it prompts the user to input the new data in the form “Child IS-A/IS-NOT-A Parent”. If the input does contain the ‘IS-A’ or ‘IS-NOT-A’ then the string is split based on this regex with a space on both sides. In the case that it is a ‘IS-A’ then the polarity is set to true otherwise it is set to false. After this a new Connection object that is made with the input in the form of (‘Parent’, ‘Child’, Polarity). All the nodes are then searched to see if any of them match the name of the ‘child’. If one of them does, then a new connection is added to the list of connections in the node object and a boolean called connectionFound is set to true. After this search is completed. There is a check to see if connectionFound is false then a new node is created with the parameter ‘child’ after this new node is created the new connection that was made previously was added to this node. This new node is added to the list of nodes. Finally, the connectionFound boolean is set back to false.

When the user inputs 2 first it prompts the user to input the start and end points. First it is checked if the string contains IS-A or IS-NOT-A if it is the later then the boolean isNot is set to true. The input is then split with the respective regex. All the nodes are then looped through to see if the start node exists and if it does then the startNodeFound boolean is set to true the startNode is then set to the node that matched the name given in the original input. After this loop the endNodeName is set to the name given in the original input.

START AT IF(STARTNODEFOUND)