CSG2341 Intelligent Systems

Workshop 2: A fuzzy decision support system for assessing mortgage applications

Related Objectives from the Unit outline:

- Identify appropriate intelligent system solutions for a range of computational intelligence tasks.
- Demonstrate the ability to apply computational intelligence techniques to a range of tasks normally considered to require computational intelligence.

Learning Outcomes:

After completing this workshop, you should be able to demonstrate an understanding of the resources required to implement a problem solution based on fuzzy reasoning, to describe and analyse the steps in designing a fuzzy system to carry out approximate reasoning (in this case, a decision support system) and use a computer package to develop such a fuzzy decision support system.

Background

Note: To complete this workshop, you will need to use information and data from your textbook. Please ensure that you bring your textbook along to class, or prepare beforehand and bring the needed information and data.

Section 9.3, pp 318-323 of your textbook describes an intelligent system for assessing mortgage applications using a fuzzy expert system. In this workshop you will implement this system. It would be a good idea to read through this section of your textbook first.

As there are one or two of you who are repeating this unit and may have done this exercise before, you can either redo the workshop, or do an alternate task (details at the end of this document). Otherwise, read on…

A loan applicant wants to take out a loan to purchase a property. The system uses information about the property and the applicant to decide how much credit to offer. The system has five input variables

- The market value of the property in $000's;
- A location rating on a scale from 0 to 10;
- The applicant's total assets in $000's; and
- The applicant's annual income in $000's.
- The interest rate to be charged in %.

The first two are used to calculate a rating between 0 and 10 for the property (called “House”), and the last two are used to calculate a rating between 0 and 10 for the applicant. Finally, the output of the system is the amount of credit to be offered in $000’s.

Task:

Write a Java class whose instances implement the mortgage application system. A suitable driver class to test the system has been provided.

When you have completed this workshop, you should study the completed source file carefully – it should be useful as a template for other fuzzy systems applications that you build.

Step 1
To complete steps in this workshop at home, you will need a version of the NetBeans IDE, e.g. go to https://netbeans.org/downloads/

Your implementation will use the “fuzzy” package, available in the Java archive file is.jar in the “Overview” section of CSG2341 on BlackBoard. You could create a CSG2341 folder on your desktop and put it in there, or in some other location of your choice.

Step 2
Write and test the class `MortgageES`.

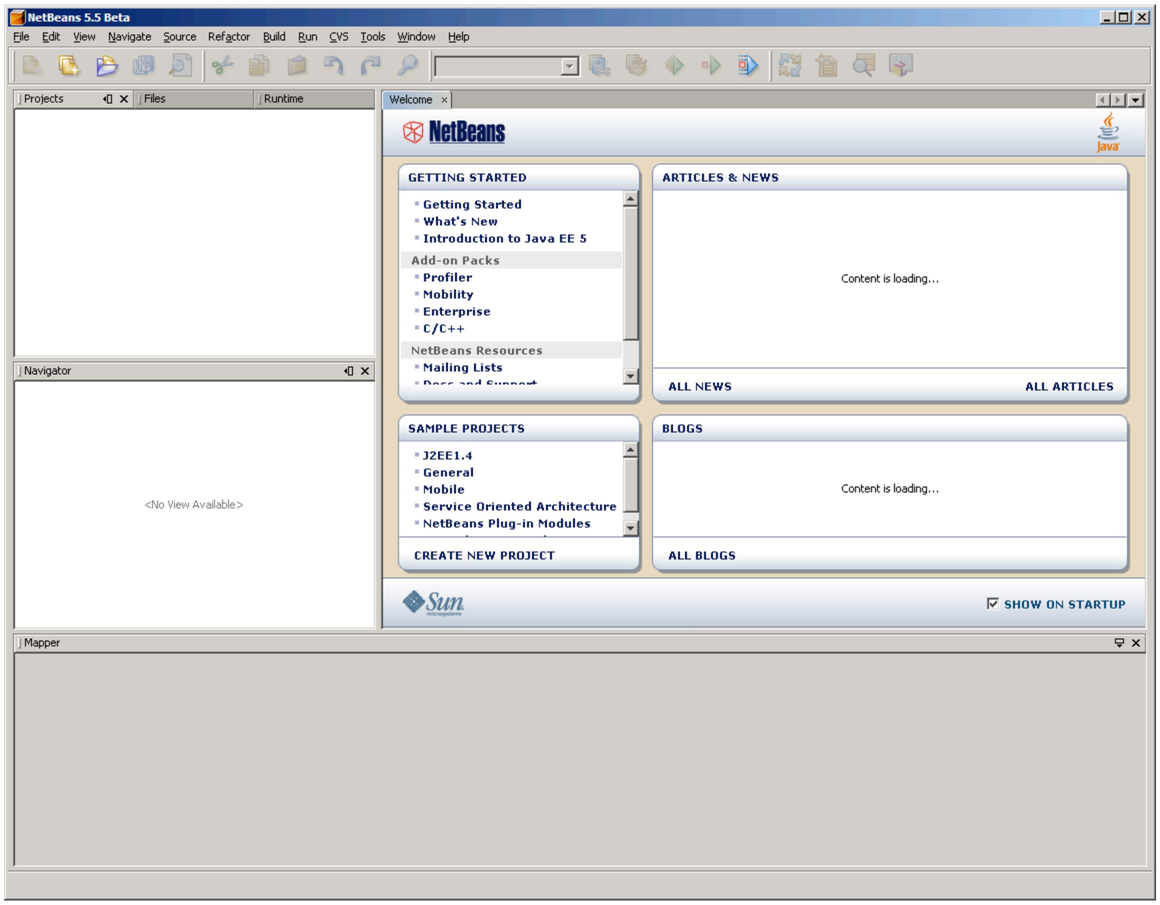
Step 2a

- Create a folder for this workshop somewhere, say on the Desktop.
- Go to BlackBoard, locate and download the zip file MortgageES.zip to the folder.
- Extract the files e.g. by right-clicking on the zip file and choosing “Extract here” (the method for doing this may be different for different operating systems). You should get a folder called MortgageES, with a subfolder called src, containing two .java files.

Step 2b

- Start up NetBeans. NetBeans provides a powerful and friendly development environment for editing, compiling and debugging your Java programs. You can start NetBeans by double-clicking on its icon (at home), or selecting it from the programs menu.

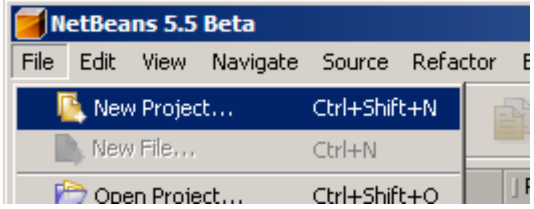
  After a minute or two of loading, you should get a window something like this, depending on your OS and version of NetBeans:



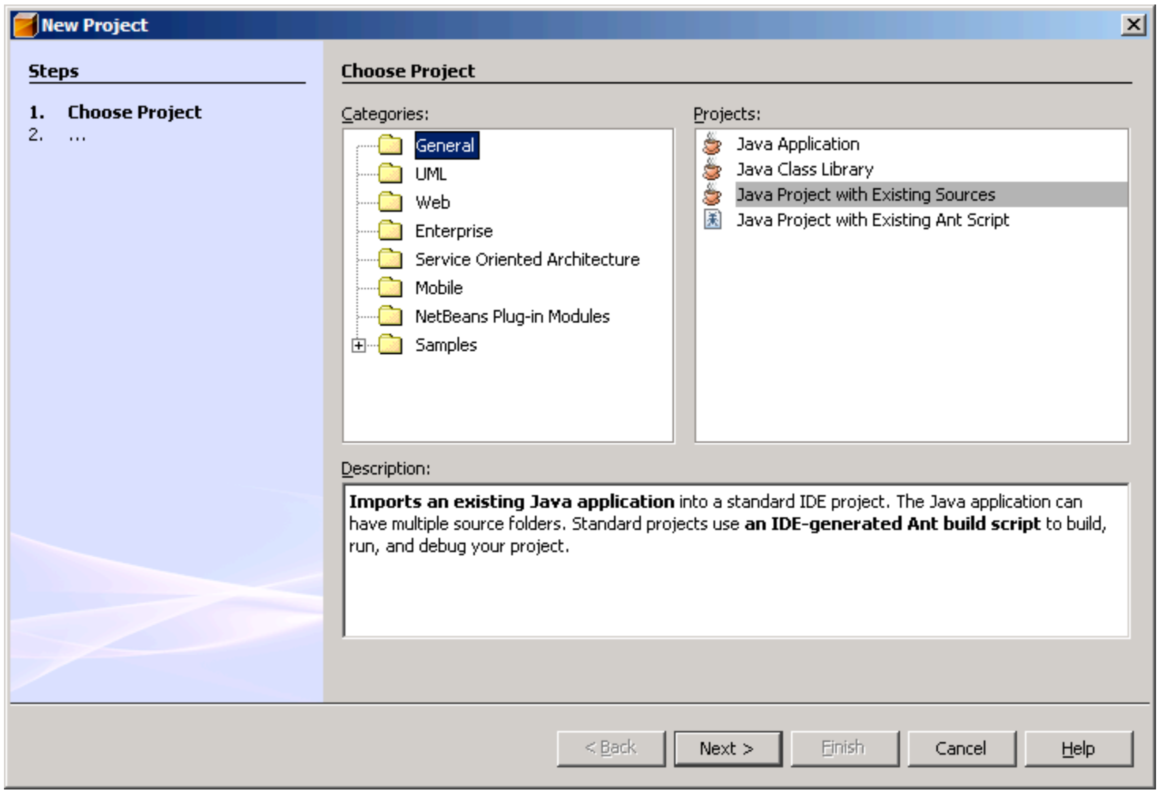
I suggest you uncheck the box “SHOW ON STARTUP”, and close the Welcome window. Now let’s create a project for this workshop.

A Java project is a container for source folders or packages which could include other information such as references and declarations of Java elements. To create a new Java project:

- ➢ On the File menu select the menu item New>Project as shown below.

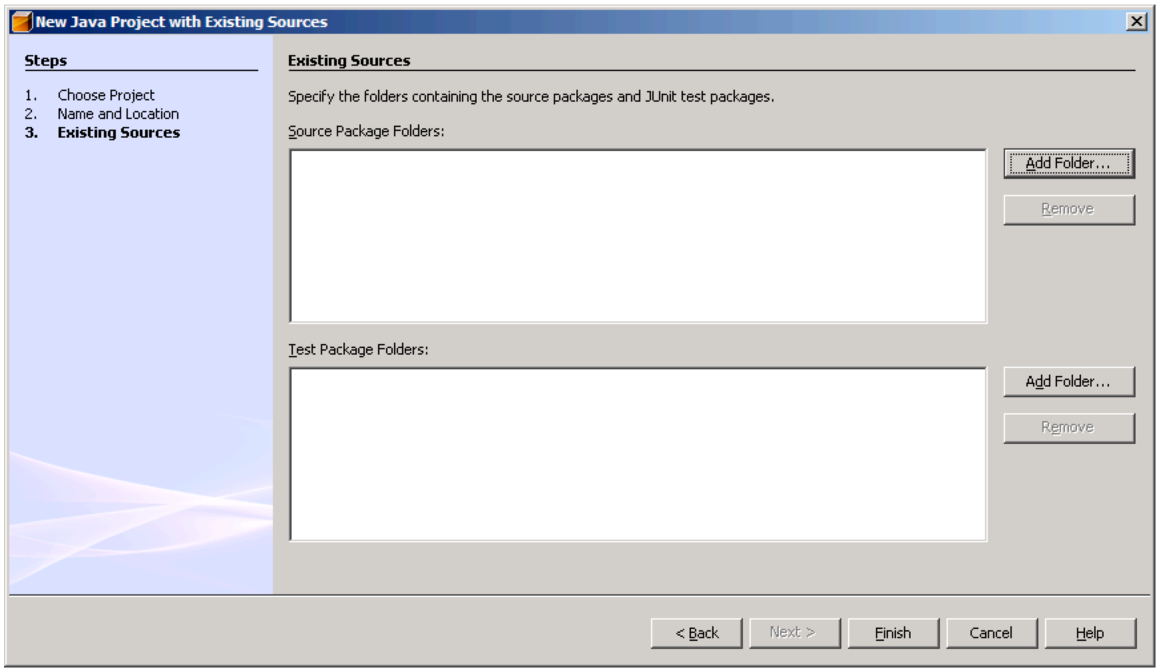

NetBeans 5.5 Beta
File Edit View Navigate Source Refactor
New Project... Ctrl+Shift+N
New File... Ctrl+N
Open Project... Ctrl+Shift+O

You should then see something like this dialog:



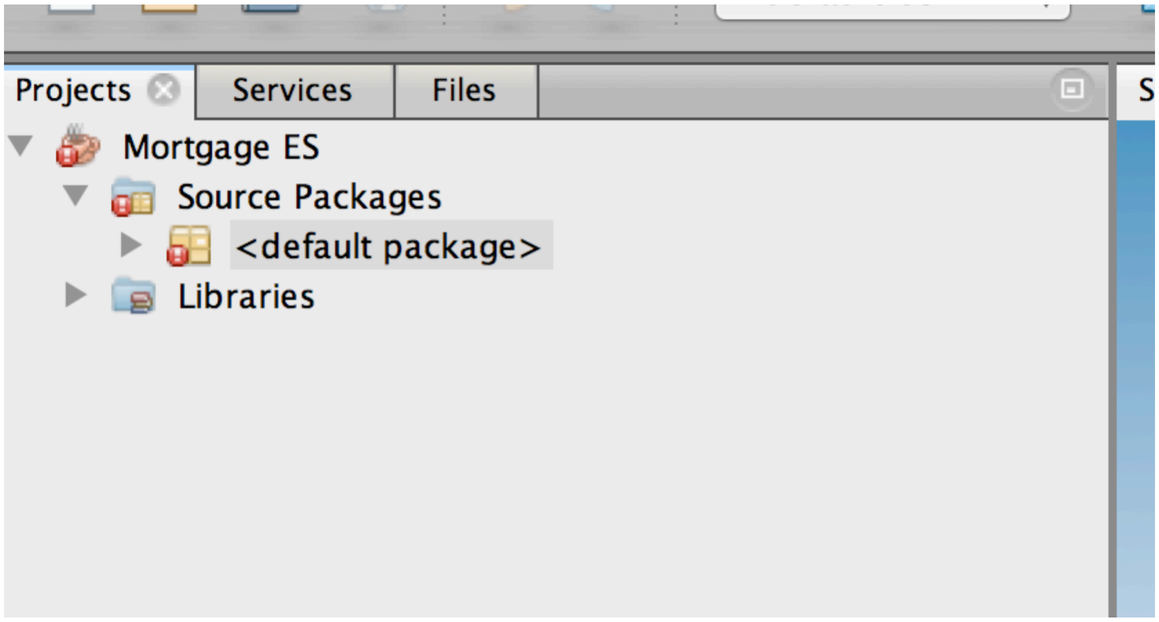
Imports an existing Java application into a standard IDE project. The Java application can have multiple source folders. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

- ➢ In this New Project dialog box, select Java Project with Existing Sources as the project type, and then click on "Next".
- ➢ When the "Java Project with Existing Sources" dialog comes up, choose a name for the project, e.g. "Mortgage ES", and use the "Browse" button to choose the MortgageES folder as the project folder, then click "Next". You should see something like this:



Specify the folders containing the source packages and JUnit test packages.

- Next, using “Add Folder”, select the “src” folder as the source package folder. Then click “Finish”.
- You should see something like this:



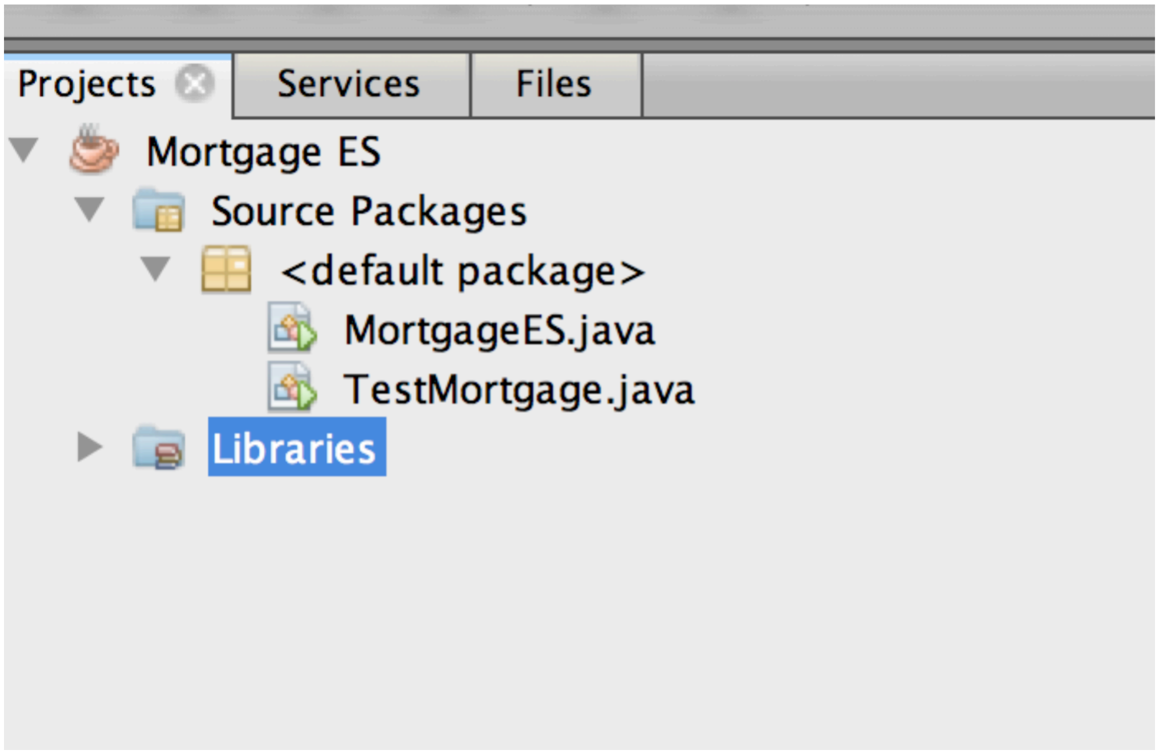
Next, we are going to add a jar file to the *Libraries* for this project. You will need to do this step in future, every time you create a new project.

- In the “Projects” view, right click on “Libraries” and choose “Add JAR/Folder”.

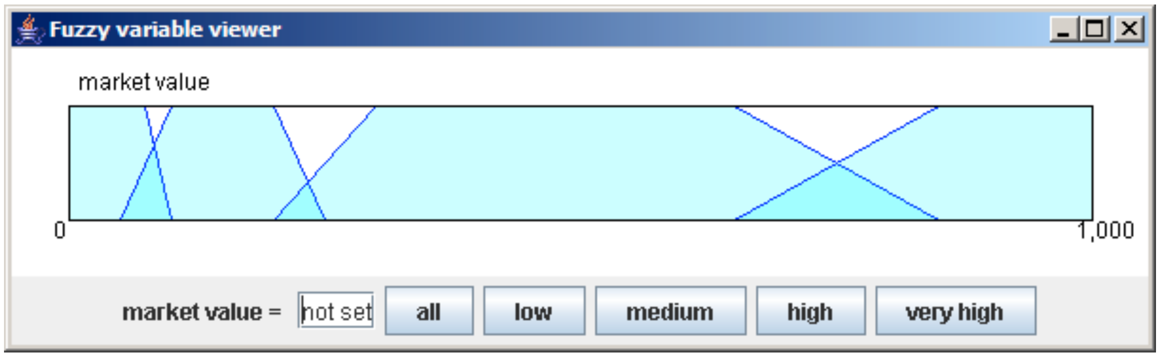
- In the “Add JAR/Folder” dialog that appears, navigate to wherever you put is.jar, and select it. The red ! icons in the Projects view should disappear.
-

REMEMBER: You will need to add this jar file to each new project for CSG2341.

- Click on the “+” sign alongside “Source Packages” and then “<default package>”. You should see:



➢ Right click on “MortgageES.java and select “Debug File”. The following window should open. If not, then go back and recheck.



The display shows you the fuzzy sets for the fuzzy variable `marketValue`. Try clicking on the buttons and typing into the text box to understand how it works.

➢ Now close the “Fuzzy Variable Viewer” window and select “Finish Debugger Session” from the “Run” menu.

Step 2c

To complete this workshop, you will need to create fuzzy sets, variables, and rules to implement the expert system described in your text. The fuzzy variable `marketValue` has been completed for you, as follows:

```
marketValue = new FuzzyVariable("market value","$000's",0.0,1000.0,3);
```

This creates a new fuzzy variable called "market value", having units of $000's, with minimum value 0.0 and maximum value 1000.0, whose values are printed to 3 decimal places.

- ➢ Double-click on MortgageES.java and locate the code above, in the constructor of MortgageES.

Some fuzzy sets have been created for this variable, e.g.:

```
    FuzzySet mvHigh = new FuzzySet("high", 20.0, 30.0, 650.0, 850.0);

    marketValue.add(mvHigh);
```

The first statement creates a new `FuzzySet` named "high", with a trapezoidal membership function that starts at 0 when market value is $20,000, rising to 1 at $30,000, stays at 1 up to $650,000 and drops to 0 again at $850,000. The second statement adds this fuzzy set as a linguistic value for the fuzzy variable `marketValue`.

- ➢ Complete the rest of the fuzzy variables and their fuzzy sets. You should create 8 fuzzy variables in all. These are to be instantiated in the constructor of MortgageES. Each fuzzy variable needs some fuzzy sets. Again, this should be done in the constructor. Create more fuzzy sets as linguistic values for all the fuzzy variables. There will be quite a lot of these. You will find that part of this has been done for you --- look for comments with "Step 2c." in them.

  The correct values to be used for your fuzzy sets can be found from the diagrams in your textbook. Just estimate them by eye.

To check what you have done, you could add lines like:

```
    marketValue.display();
```

for each fuzzy variable, which will bring up more windows like the one shown above. As each fuzzy variable is added, the corresponding "set" method can be uncommented.

Step 2d

Rules can now be added. The ones that determine the house rating have been done for you. Now that you have added all the variables, you can try this out by uncommenting the indicated lines. First, a rule set has been created:

```
            ruleSet = new MamdaniRuleSet();
```

All the rules that you create will be added to this rule set. An example shows how a simple rule can be added:

```
    // if (market value is low) then (house is low)
    ruleSet.addRule(marketValue, mvLow, house, houseLow);
```

There is also a version of `addRule` that lets you create and add a rule with two conditions, like this:

```
ruleSet.addRule(    location, locBad,
                    marketValue, mvLow,
                    house, houseVeryLow);
```

(This rule says that if the location is bad and the market value is low, then the house rating is very low.)

Now instead of you having to add 12 separate rules to cover each combination of location and market value, there is a special shortcut so that you can create and add a whole matrix of rules with one method call, like this:

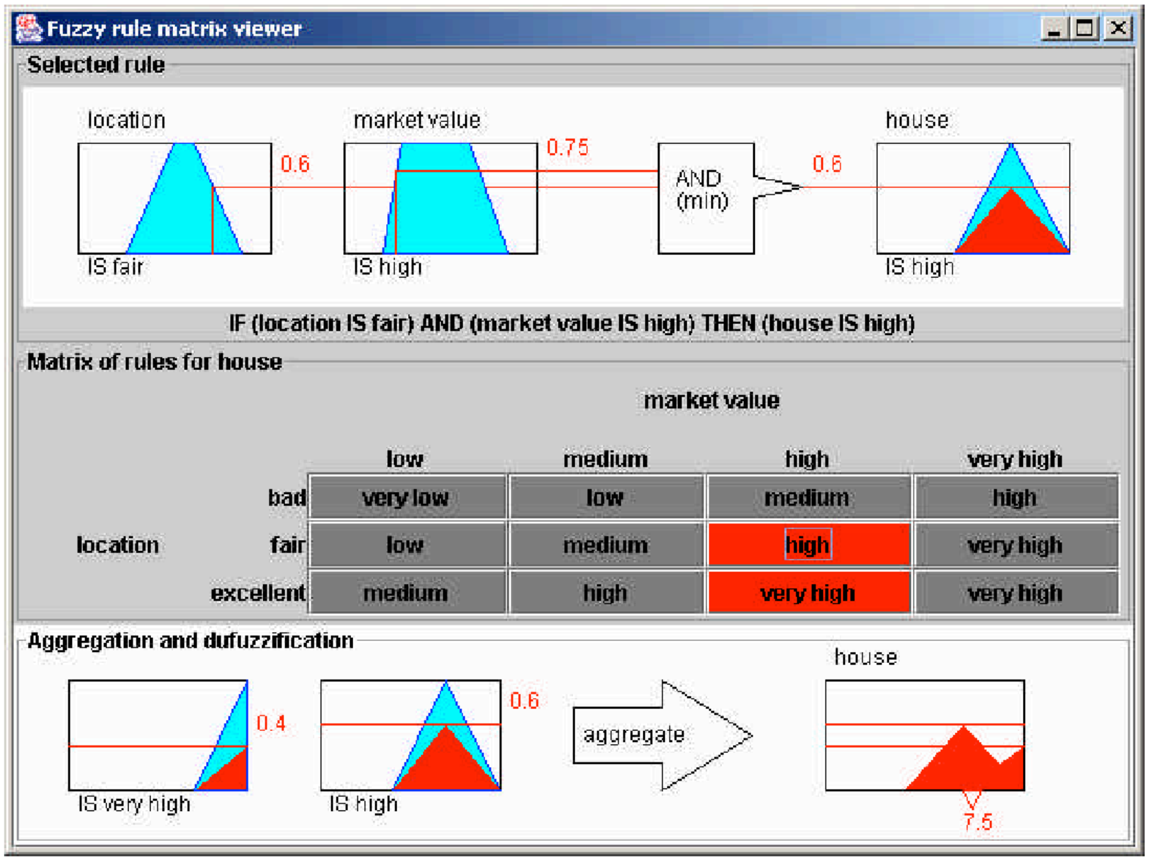
```
FuzzySet[] locationSets = {locBad, locFair, locExcellent};
FuzzySet[] mvSets = {mvLow, mvMedium, mvHigh, mvVeryHigh};
FuzzySet[][] houseMatrix =
{
        {houseVeryLow, houseLow, houseMedium, houseHigh},
        {houseLow, houseMedium, houseHigh, houseVeryHigh},
        {houseMedium, houseHigh, houseVeryHigh, houseVeryHigh}
};
ruleSet.addRuleMatrix(
            location, locationSets,
            marketValue, mvSets,
            house, houseMatrix
            );
```

While the syntax takes some getting used to, this will save you a lot of time in the long run. If you have forgotten Java's syntax for arrays, or have trouble understanding the example code above, please ask your tutor to explain. You will want to master this technique for future workshops and assignments in this unit.

You can check all the rules added with `addRuleMatrix` using code like this:

```
ruleSet.displayRuleMatrix(
   location, locationSets,
   marketValue, mvSets,
   house);
```

which will bring up a window like this:



IF (location IS fair) AND (market value IS high) THEN (house IS high)

Matrix of rules for house

location \ market value	low	medium	high	very high
bad	very low	low	medium	high
fair	low	medium	high	very high
excellent	medium	high	very high	very high

This display is similar to those you will find in the text book. The top panel shows the currently selected rule. The fuzzy sets used in the left hand side of the rule are shown on the left of this top panel. The blue shapes are the membership functions for the fuzzy sets. The red lines show the membership values for the fuzzy variables, which are combined by the “AND” box to give the firing strength of the rule. The right hand side of the rule appears at the far right, where you can see the membership function (for a Mamdani-style rule), and the scaled membership function (scaled by the rule’s firing strength).

The middle panel lets you select which rule is displayed by clicking on the relevant matrix entry. Rules that are firing are coloured red in this matrix. The bottom panel shows the aggregation of the rules that are firing, with defuzzification using the COG method illustrated on the far right of the panel.

All these displays are dynamic.

Step 2e

Complete the rest of the fuzzy rules. First complete and check the rules that determine the applicant rating. Finally, complete and check the rules that determine the credit to be offered.

Step 2f

Uncomment the line

```
        //System.out.println(marketValue.checkGaps());
```

This command does a check to make sure there are no “gaps” in the fuzzy sets for the fuzzy variable `marketValue`. This is useful because if there are any gaps (any values in the range of the variable, where all the fuzzy sets have a membership value of 0), it could cause bugs in your applications that are difficult to find. Add similar lines for other *input* variables.

Step 2g

Complete the other methods. Make sure you have completed and uncommented everything than needs to be. The application is now complete. You can test it by running the `TestMortgage` class.

A bit more about rules – just for your information

Rules are actually objects in the IS library. You can create them and manipulate them like so:

```
      MamdamiRule rule;

      // if (market value is low) then (house is low)

      rule = new MamdaniRule(house, houseLow);

      rule.addCondition(marketValue, mvLow);

      ruleSet.addRule(rule);
```

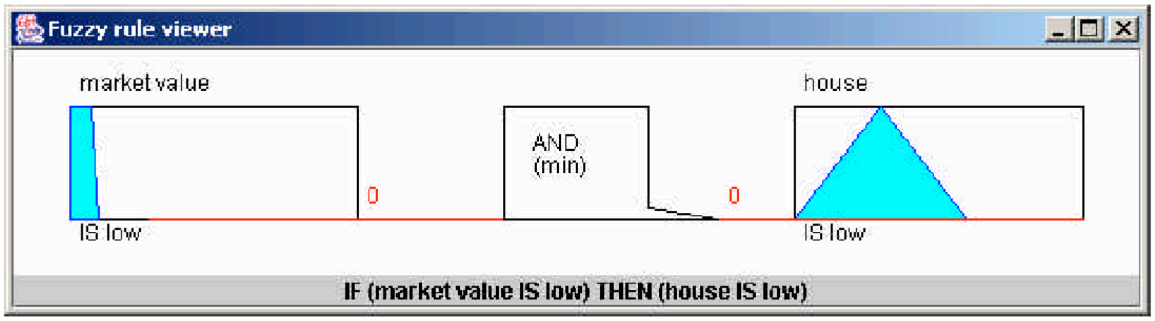
The first statement creates a new rule whose right hand side means that the house has a low rating (this example assumes that you have created a fuzzy variables “house” and given it a linguistic value “houseLow”). The second statement adds the condition that the market value is low to the left hand side of the rule. More conditions can be added to the left hand side of the rule using `addCondition`. If there is more than one, these conditions are assumed to be “AND”-ed together.

The final statement adds this completed rule to the rule set. At this point, checks are made to ensure that the rule set makes sense as a whole. Once the rule has been added to a rule set, extra conditions cannot be added.

You can check the rule that you have added with the code:

```
      rule.display();
```

which will bring up a window like this:



IF (market value IS low) THEN (house IS low)

Sometimes, it may be useful to create rules this way, instead of adding them directly into the rule set as has been done in `MortgagaES.java` (for example, you might want to make a rule with more than two conditions).

When you have completed this workshop, submit your Java source file for MortgageES to your tutor using the submission facility on BlackBoard. The submission is due before midnight on Sunday.

Alternate task:

For those who have done this exercise before, here is an alternative task if you prefer it:

Devise a fuzzy expert system along similar lines to this one, on a topic of your choice (e.g. a system to help car buyers). Your submission should include

- an explanation of the purpose of the system
- a description of the input variables (at least 4)
- a description of the output variable(s)
- a set of rules in English language form (at least two different kinds of rules in the sense that they have different input variables)
- a program similar to the one above that allows users to input values for the input variables and to see the resulting output(s). The program should make use of at least one rule matrix (FAM).