
ZNS Documentation

Release 1.0.0

Richard Werner

Oct 03, 2018

Contents:

1	Set up project in JetBrains IDEA	1
1.1	Clone project	1
1.2	Set up Idea	1
2	Database description files structure and syntax	7
2.1	Folk knowledge base	7
2.2	Rules knowledge base	8
2.3	Semantic network knowledge base	9
2.4	Uncertainty	9
3	Function to be implemented	11
3.1	DataBase interface	11
3.2	InferenceEngine	11
3.3	KnowledgeBase	11
3.4	UserInterface	12
3.5	UncertaintyModule	12
3.6	Main function	12
4	Indices and tables	13

CHAPTER 1

Set up project in JetBrains IDEA

There is short manual how to setup up project in JetBrains IDEA. Other ideas are not recommended, but you can use them. This manual is written only for JetBrains IDEA.

1.1 Clone project

First step is clone project from GitLab to your folder. Open command line and go to location, where you want to have Java project. To clone the project run these command:

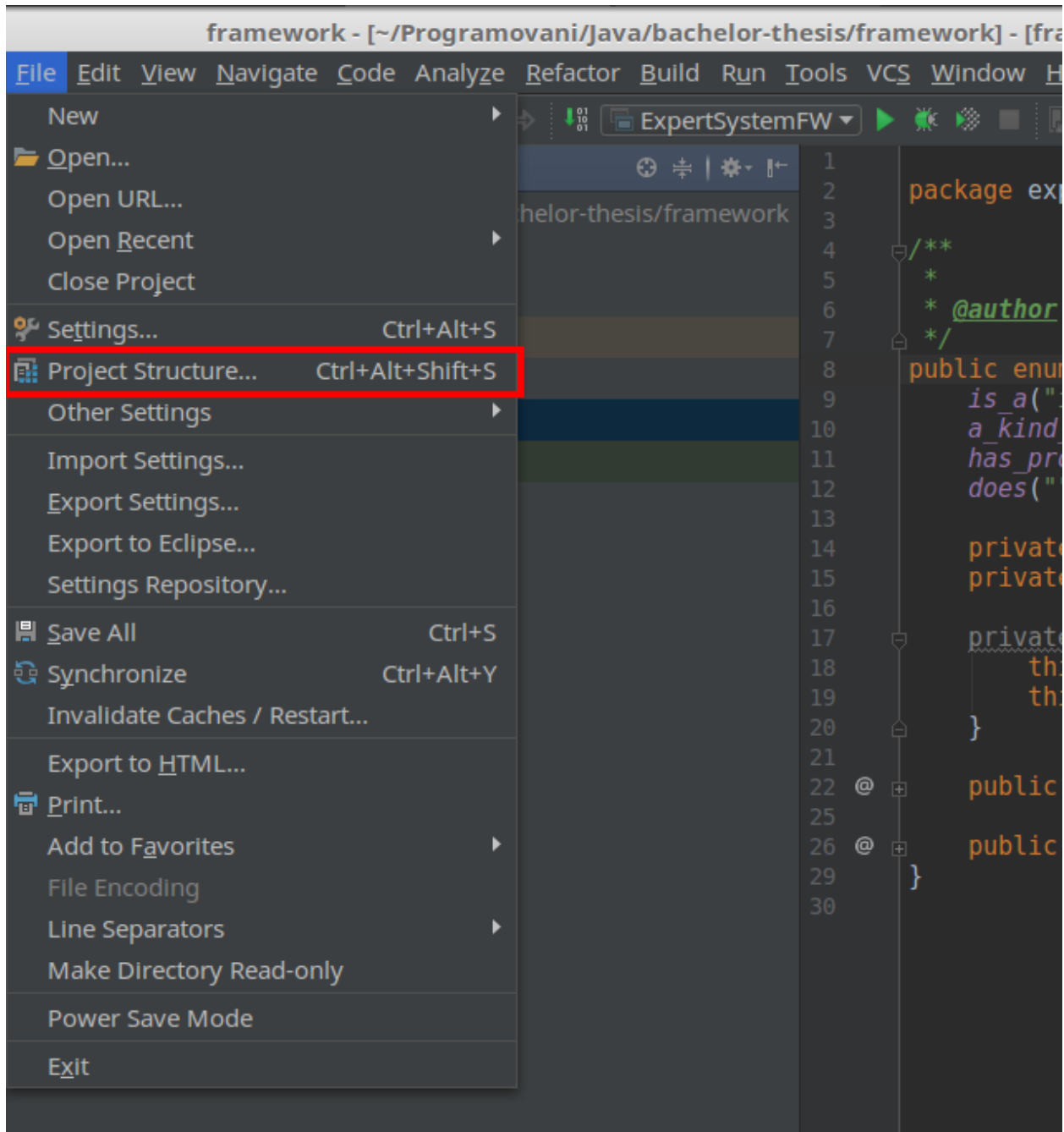
```
$ git clone git@gitlab.fit.cvut.cz:BI-ZNS/framework.git
```

This will create folder for project. If git clone raise exception, that you don't have right for clone this project, go to FIT GitLab and set up your private key.

1.2 Set up Idea

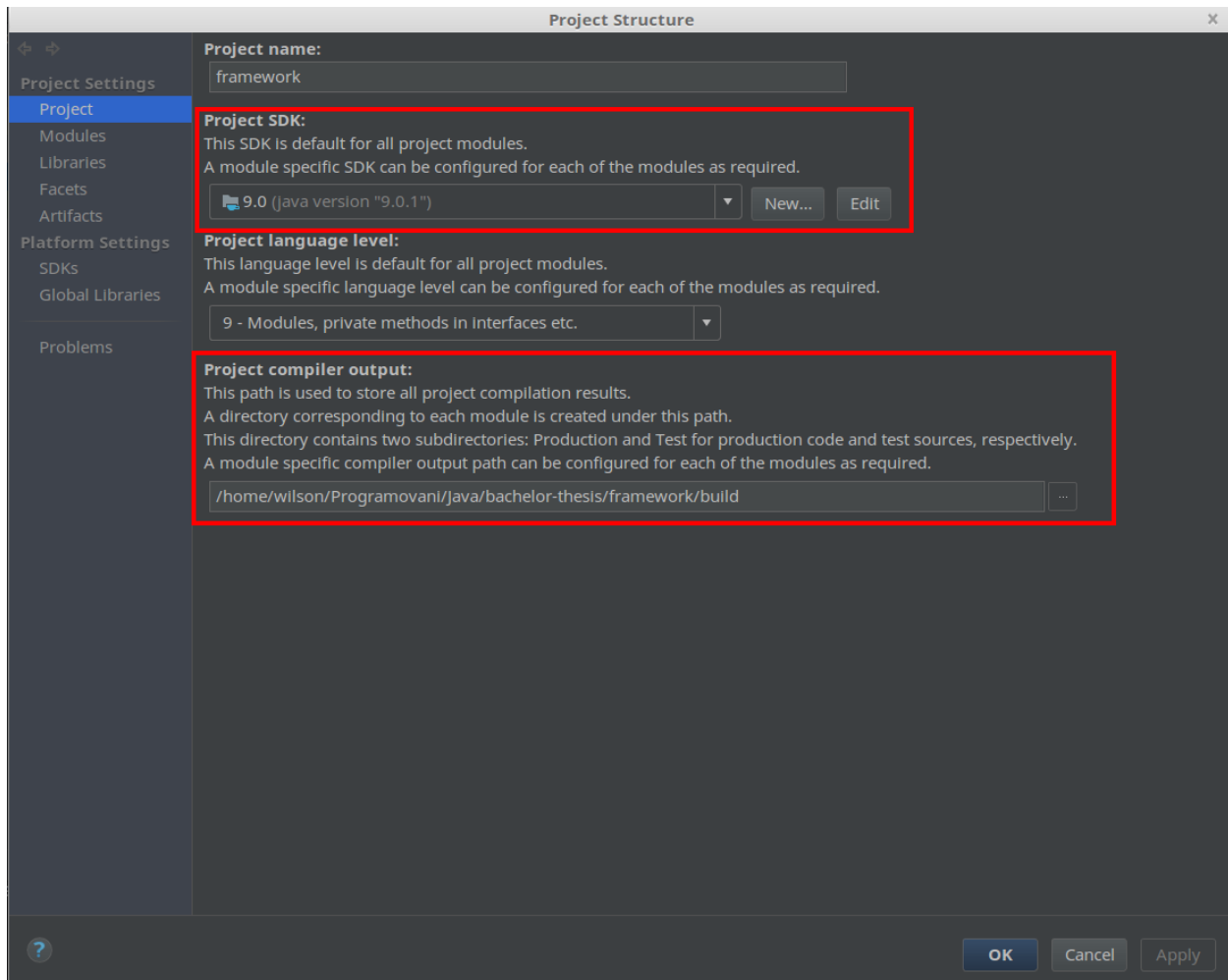
Next start JetBrains IDEA and open folder with cloned project. First step is set up project settings.

1.2.1 1) Go to the File -> Project Structure



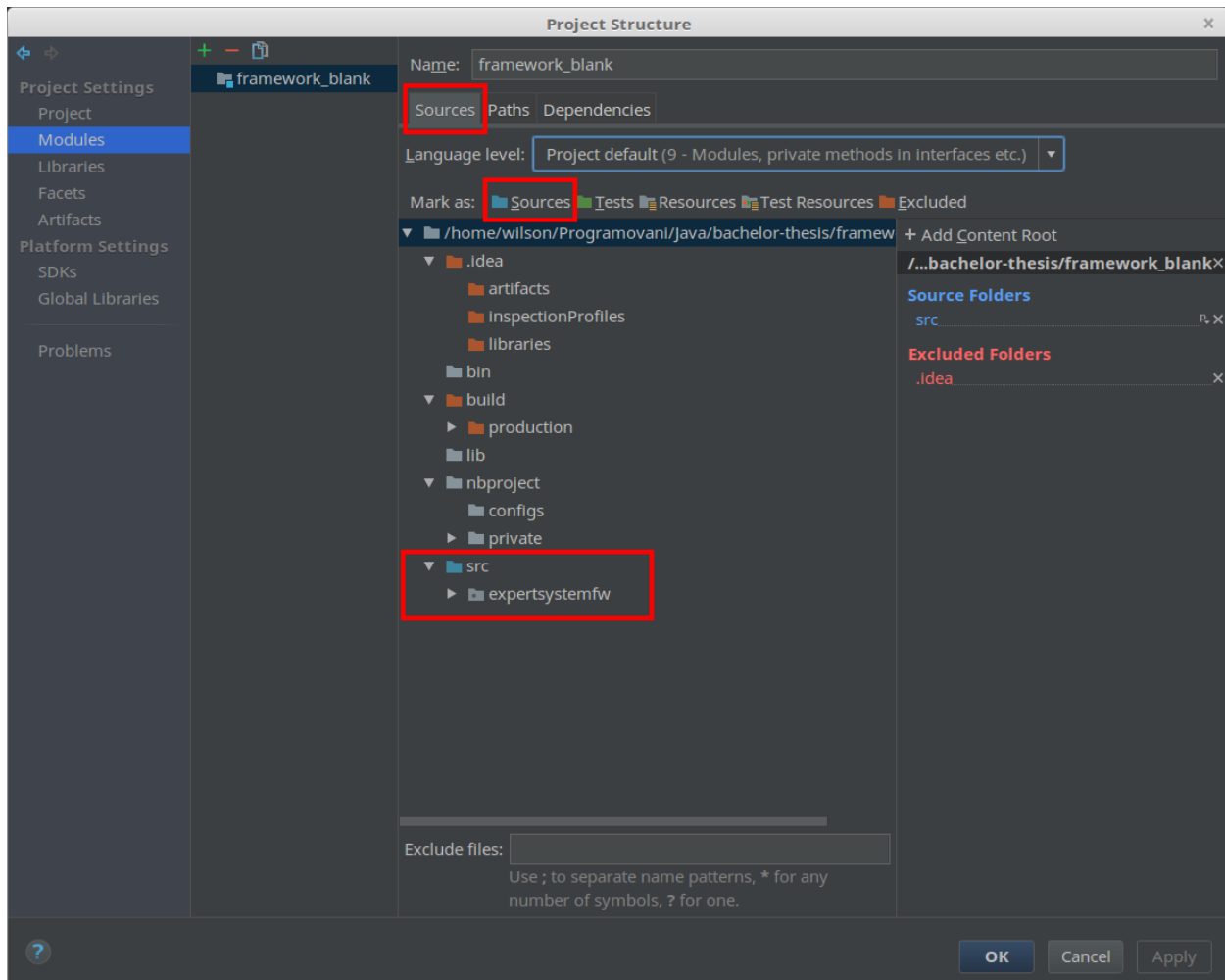
1.2.2 2) Set up project SDK and target folder

In the sub menu **Project** Select SDK. If you have installed Java with SDK, just select your version of Java from drop down list. If your java installation is not listed, add path to your Java installation with new button. If you don't have Java installed, download Java from official [webpage](#). In the same sub menu select the output directory. In this directory will be located build files.



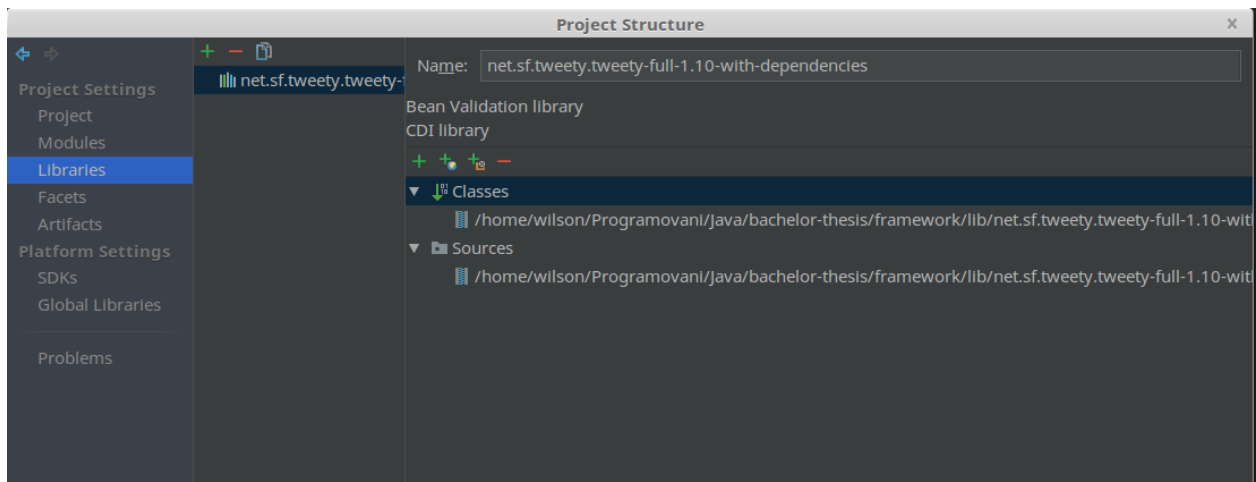
1.2.3 3) Setup source folder in modules

In the sub menu **Modules** go to the tab Sources. There select src folder in the tree view and mark this folder as source folder with button on the top of the tree view (blue icon folder).



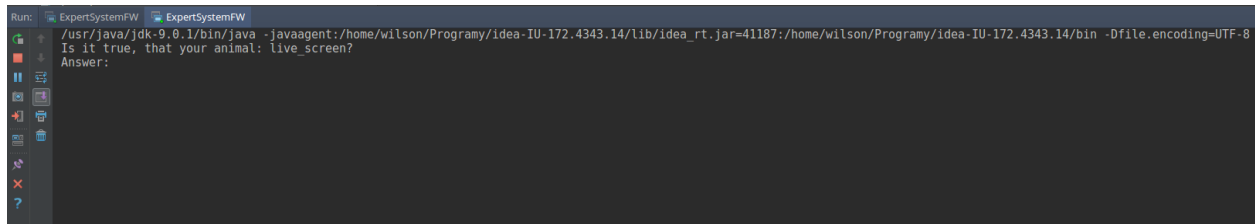
1.2.4 4) Set up external libraries

Go to the sub menu **Libraries** and click on green plus button. From drop down menu select **Java**. In popup windows select path to folder lib in your project and select file **net.sf.tweety.tweety-full-1.10-with-dependencies.jar**. Accept selection with OK button.



1.2.5 5) Run compile and project

Now in the project files locate the main file `src/expertsystemfw/ExpertSystemFW`, right click on that file a select **Run ExpertSystemFW.main()**. This should build java project and run that project in the console.



```
Run: ExpertSystemFW
/usr/java/jdk-9.0.1/bin/java -javaagent:/home/wilson/Programy/idea-IU-172.4343.14/lib/idea_rt.jar=41187:/home/wilson/Programy/idea-IU-172.4343.14/bin -Dfile.encoding=UTF-8
Is it true, that your animal: live_screen?
Answer:
```


Database description files structure and syntax

All input files for this project are located at **framework/src/expertsystemfw/KnowledgeBase/Files/**. In this folder are some examples of that files. You could use them as template for developing.

2.1 Folk knowledge base

This files define knowledge base based on first-order logic. Definition of knowledge base are based on Backus-Naur form.

Backus-Naur form:

```
KB ::= SORTSDEC DECLAR FORMULAS
DECLAR ::= (PREDDDEC) *
SORTSDEC ::=
( SORTNAME "=" "{" (CONSTANTNAME ("," CONSTANTNAME) *)? "}" "\n" ) *
PREDDDEC ::=
"type" "(" PREDICATENAME "(" (SORTNAME ("," SORTNAME) * ")" )? ")" "\n"
FORMULAS ::= ( "\n" FORMULA ) *
FORMULA ::= ATOM | "forall" VARIABLENAME ":" (" FORMULA )" |
"exists" VARIABLENAME ":" (" FORMULA )" |
(" FORMULA )" | FORMULA "&&" FORMULA |
FORMULA "||" FORMULA | "!" FORMULA | "+" | "-"
ATOM ::= PREDICATENAME "(" (TERM ("," TERM) * ")" )?
TERM ::= VARIABLENAME | CONSTANTNAME
```

- **Alphanumeric values starting with symbol „KB”**

- SORTNAME - name of group of constants
- PREDICATE - name of predicate
- CONSTNAME - name of constant
- VARIABLENAME - name of variable

2.1.1 Example:

```
problem={gpuPowerCableDefect, monitorPowerSourceProblem, badDualMonitorSettings,
↪videoCableNotProperlyPlugged, gpuNotSeatedWellOrDefective,
↪ramNotSeatedWellOrDefective, damagedMonitorCable, defectiveMonitor, defectiveFan,
↪gpuDefective}

type (system_powers_up(Problem))
type (live_screen(Problem))
type (monitor_LED_on(Problem))
type (NO_POWER_displayed(Problem))
type (NO_SIGNAL_displayed(Problem))
type (dual_monitors_HDTV(Problem))
type (hearing_a_string_of_beeps(Problem))
type (video_cable_secure(Problem))
type (GPU_seated_tested(Problem))
type (cable_damaged_or_bent_pins(Problem))
type (monitor_tested(Problem))
type (noisy_fan(Problem))
type (works_only_with_new_GPU(Problem))

forall Problem: (system_powers_up(Problem))

system_powers_up(monitorPowerSourceProblem)

live_screen(gpuPowerCableDefect)
NO_POWER_displayed(gpuPowerCableDefect)

monitor_LED_on(badDualMonitorSettings)
dual_monitors_HDTV(badDualMonitorSettings)

live_screen(videoCableNotProperlyPlugged)
monitor_LED_on(videoCableNotProperlyPlugged)
NO_SIGNAL_displayed(videoCableNotProperlyPlugged)

monitor_LED_on(gpuNotSeatedWellOrDefective)
hearing_a_string_of_beeps(gpuNotSeatedWellOrDefective)
```

2.2 Rules knowledge base

Knowledge base defined by rules IF <conclusion> THEN <formula> . In the rules could be used logical operators && and ||. With those easy rules could be defined whole behaviour of Knowledge base.

2.2.1 Example:

```
IF gpuPowerCableDefect THEN system_powers_up && NO_POWER_displayed && live_screen
IF monitorPowerSourceProblem THEN system_powers_up
```

2.3 Semantic network knowledge base

Semantic network knowledge base are file, that create semantic network based on rules. As the results of the expert system (conclusions) can be taken, the nodes that have the input level of the vertex 0 (not entered in the back edge), ie the node is the source. Each relation is define by one line in file. Syntax of the lines is:

node_from, relation, node_to

Delimiter could be comma or semicolon. White space will be ignored. Lines starting with # will be ignored. Relation must be name based on enumeration at: **expertsystemfw.KnowledgeBase.SemanticNetwork.Relation**. Between two nodes could be only one edge.

2.3.1 Example:

```
gpuPowerCableDefect, has_property, system_powers_up
gpuPowerCableDefect, has_property, live_screen
gpuPowerCableDefect, has_property, NO_POWER_displayed
```

2.4 Uncertainty

Uncertainty format must be specified by user. This format should be very simple, you just need specify certainty for conclusions and predicates. For example one part of file should look like example.

2.4.1 Example:

```
monitor_tested, gpuDefective, 0.8
live_screen, defectiveFan, 0.76
live_screen, damagedMonitorCable, 0.91
```

Function to be implemented

In the project, there are many not implemented functions. This function need to be implemented by the user. The main skeleton of the project are done and user need to implement decision part of the project.

3.1 DataBase interface

First part, which must be implemented is DataBase interface. This part is used for storing answers from the console. User could implement this part as he want, for example you can use HashMap. You could add some support function but don't change interface. You need to implement functions:

- **addAnswer** - add new answer to database (based on object Answer)
- **getAnswer** - get answer from database based on predicate
- **getAllAnswers** - return list of all answers
- **contains** - check, if answer was already stored

3.2 InferenceEngine

In the template are not implemented inference engines. In the project is prepared interface for inference engine and also templates for backward and forward chaining. You need to implement function **startInference** for each engine. You could add any support function but don't change interface of engine. You could only add some function to interface if you need.

3.3 KnowledgeBase

Knowledge base is fully implemented in the project. You should use this in Inference engines. Only files, that you need to edit are located in **src/expertsystemfw/KnowledgeBase/Files**. These files are used to load semantic network,

uncertainty model and knowledge base. In these files will be located all settings and informations need to decide conclusion.

- **FOLKnowledgeBase** - In this file knowledge base are defined based on first-order logic
- **RulesKnowledgeBase** - In this files you can specify Knowledge base with IF ... THEN ... rules
- **SemanticNetworkKnowledgeBase** - In this file you can specify knowledge base as semantic network
- **Uncertainty** - In this files specify uncertainty of conclusions

3.4 UserInterface

In this package you need to specify and implement user interface. These function are used for communicate with the user. All of them should be called from inference engine. For calling other function from askUser there is observer pattern. There is prepared notifyWhy, notifyHow, notifyStart and notifyStop. If you notify one of those, you need to override update method in inference engine, which should call target function in inference engine (Backward or Forward). There are located 4 main function for user interface

- **conclude** - Show user conclusion
- **askUser** - Ask user to some question and do something based on answer
- **explainWhy** - Based on the conclusion, explains what is the inference engine up for at the moment.
- **explainHow** - Based on the list of answers, explains how did the inference engine get to the point where it is now.

3.5 UncertaintyModule

User need to implement uncertainty module. Only basic interface is prepared. User need to implement basic Bayes uncertainty. This module should be used by Inference engine, to get some certainty information about predicates and conclusions.

3.6 Main function

In the file **ExpertSystemFW** is located main. You don't need to change here anything. If you want to change inference engine from backward to forward, change class that is created here.

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`