VOnDA Cheatsheet

A VOnDA File ...

... has the extension `.rudi`.

... consists of a list of variable and function definitions and possibly nested rule statements.

... makes variables defined at the top-level rule file persistent (final) throughout the whole program.

... may contain Java like commenting.

... may contain Java verbatim code between `/*@...@*/`

Rules consist of ...

... an optional label in snake case followed by a colon.

... optionally labeled `if` blocks with optional `else` blocks. They work like in Java stopping the evaluation of (a sub-tree of) the rules in case the condition doesn't hold.

... possibly two special statements that have a unique behaviour: `propose` and `timeout`. Both create a closure;
`propose` blocks are all collected and the "best" is chosen;
labeled `timeouts` wait for the specified time and the execute their body.

Stopping Rule Evaluation

```
break label_name;
cancel; // cancels all following (same level) rules.
cancel_all; // cancels all following rules.
```

`propose` and `timeout` blocks can be exited early by `return;`.

Overloaded (Boolean) Operators

`a += b` is syntactic sugar for `a.add(b)` (for lists and sets).
`a -= b` is syntactic sugar for `a.remove(b)` (for lists and sets).
The boolean operators `<=`, `>=` can be used to check if an object is of a specific class, for subclass tests between two classes, and for subsumption of dialogue acts. If the type of a right hand side of an expression is inferrable this shorthand exists:

```
if  (! c.user.personality.nonchalance){...}
// translates to ...
if (!((((c != null) && (c.user != null)) && (c.user.personality != null))
&& (c.user.personality.nonchalance != null))) {...}
if (sa <= #Question){...}
// translates to ...
if (sa.isSubsumedBy(new DialogueAct("Question")){...}
```

Functionality from Run-Time System

Short-hand conversion methods from Java

```
int toInt(String s);
float toFloat(String s);
double toDouble(String s);
boolean toBool(String s);
String toStr(T i);
// T in(int, short, byte, float, double, boolean)
```

Other Agent methods

```
// Telling the Agent that something changed
void newData();
String getLanguage();
// Random methods
int random(int limit); // returns [0,limit)
float random(); // returns [0,1)
T random(Collection<T> coll); // select random element
long now(); // return current time since epoch in millisec.
Logger logger; // Global logger instance (slf4j)
// discarding actions and shutdown
void clearBehavioursAndProposals();
void shutdown();
```

Timeouts

```
void newTimeout(String name, int millis);
boolean isTimedOut(String name);
void removeTimeout(String name);
boolean hasActiveTimeout(String name);
// cancel and remove an active timeout
// will not be executed
void cancelTimeout(String name);
```

Functions allowing lambda expressions

```
boolean some(Collection<T> coll, Function<Boolean, T> pred);
boolean all(Collection<T> coll, Function<Boolean, T> pred);
List<T> filter(Collection <T> coll, Function<Boolean, T>pred);
List<T> sort(Collection<T> coll, Function<Integer, T, T> c);
Collection<T> map(Collection<S> coll, Function<T, S> f);
int count(Collection<T> coll, Function<Boolean, T> pred);
T first(Collection<T> coll, Function<Boolean, T> pred);
```

Pre-added Java methods

```
#Object boolean equals(Object e);
#String boolean startsWith(String s);
#String boolean endsWith(String s);
#String String substring(int i);
#String String substring(int begin ,int end);
#String boolean isEmpty();
#String int length();
#List<T> T get(int a);
#Collection<T> void add(Object a);
#Collection<T> boolean contains(Object a);
#Collection<T> int size();
#Collection<T> boolean isEmpty();
#Map<S, T> boolean containsKey(S a);
#Map<S, T> T get(S a);
#Array<T> int length;
```

Methods on RDF and RDFClass Objects

```
Rdf toRdf(String uri);
#Rdf String getURI();
#Rdf boolean has(String predicate);
#Rdf long getLastChange(boolean asSubject, boolean asObject);
RdfClass getRdfClass(String s);
boolean exists(Object o);
// return only the name part of an URI (no namespace or angle
// brackets)
String getUriName(String uri);
```

Methods dealing with dialogue acts

```
// Methods applied to the object DialogueAct
#DialogueAct String getDialogueActType();
#DialogueAct void setDialogueActType(String dat);
#DialogueAct String getProposition();
#DialogueAct void setProposition(String prop);
#DialogueAct boolean hasSlot(String key);
#DialogueAct String getValue(String key);
#DialogueAct void setValue(String key, String val);
#DialogueAct long getTimeStamp();
#DialogueAct void setProposition(String prop);
// ...
```

Dialogue act objects are marked with #.

```
// ...
// sending of dialogue acts
DialogueAct createEmitDA(DialogueAct da);
DialogueAct emitDA(int delay, DialogueAct da);
DialogueAct emitDA(DialogueAct da);
// Access to dialogue acts of the current session
// my last outgoing resp. the last incoming dialogue
// act
DialogueAct myLastDA();
DialogueAct lastDA();
// Did I say something like ta in this session
// (subsumption)? If so, how many utterances
// back was it? (otherwise, -1 is returned)
int saidInSession(DialogueAct da);
// like saidInSession, only for incoming dialogue acts
int receivedInSession(DialogueAct da);
// Check if we asked a question that is still pending
boolean waitingForResponse();
// Mark last incoming DA as treated and not pending
// anymore(stop rules firing)
void lastDAprocessed();
DialogueAct addLastDA(DialogueAct newDA);
```