jInfer Base Module Description

Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, Matej Vitásek
Advisors: RNDr. Irena Mlýnková, Ph.D., Martin Nečaský, Ph.D.

Praha, 2011

Target audience: developers willing to extend jInfer.

Responsible developer:	Matej Vitásek
Required tokens:	none
Provided tokens:	none
Module dependencies:	none
Public packages:	cz.cuni.mff.ksi.jinfer.base.automaton cz.cuni.mff.ksi.jinfer.base.interfaces cz.cuni.mff.ksi.jinfer.base.interfaces.inference cz.cuni.mff.ksi.jinfer.base.interfaces.nodes cz.cuni.mff.ksi.jinfer.base.objects cz.cuni.mff.ksi.jinfer.base.objects.nodes cz.cuni.mff.ksi.jinfer.base.regexp cz.cuni.mff.ksi.jinfer.base.utils org.apache.log4j.*

1 Introduction

This is the module containing data structures, interfaces and logic shared across the whole jInfer framework. Virtually every other module containing logic should in theory depend on *Base*.

2 Structure

Description of *Base* structure will partially mirror its JavaDoc documentation. For more detailed information, refer to it directly.

Base contains logic for Log4j initialization in `Installer` class. Configuration of the overall logging granularity level (NetBeans options integration) is contained in `cz.cuni.mff.ksi.jinfer.base.options` package. Shared jInfer graphics is contained in the `cz.cuni.mff.ksi.jinfer.base.graphics` and `cz.cuni.mff.ksi.jinfer.base.graphics.icons` packages.

2.1 Data structures

Regular expression representation is contained in `cz.cuni.mff.ksi.jinfer.base.regexp` package. Most important class is of course `Regexp`, assisted by an enum of its type `RegexpType` and representation of its interval `RegexpInterval`.
XML node representation is described by interfaces in `cz.cuni.mff.ksi.jinfer.base.interfaces.nodes` package and more-less concrete implementations in `cz.cuni.mff.ksi.jinfer.base.objects.nodes` package.
Finite state automata representation is contained in the [**?**] package.
Refer to **??** to get an accurate description of all these representations.

Miscellaneous shared classes are contained in `cz.cuni.mff.ksi.jinfer.base.objects`. For example, `Input` is used to provide the *Initial Grammar Generator* with input data. `Pair` is a generic class binding two object together in a *pair*.

2.2 Interfaces

Apart from interfaces contained in already discussed `cz.cuni.mff.ksi.jinfer.base.interfaces.nodes` package, there is an important group of interfaces contained in `cz.cuni.mff.ksi.jinfer.base.interfaces.inference` package: the *inference* interfaces. They come in two flavours: the actual inference interface, and its callback. For a comprehensive description of them and their interaction refer again to **??**.

There is one more package containing interfaces: `cz.cuni.mff.ksi.jinfer.base.interfaces`. There are a few groups of them.

- Module lookup support: `NamedModule` and `UserModuleDescription`.
- Inference support: `Capabilities`.
- Service provider definitions: `Expander`, `Processor` and `RuleDisplayer`.

2.3 Utility logic

Useful logic for the whole framework is focused in the `cz.cuni.mff.ksi.jinfer.base.utils` package. A few highlights from here follow.

- Testing whether a collection is empty: `BaseUtils.isEmpty()` - handles `null` and zero elements.
- Filtering a list based on a predicate: `BaseUtils.filter()`.
- Cloning a list N times in a row: `baseUtils.cloneList()` - e.g. from *abc, 3 times* creates *abcabcabc*.
- Deep cloning a grammar: `CloneHelper.cloneGrammar()`.
- Writing out a list separated by specified separator: `CollectionToString.colToString()` - accepts the list, operation to perform on each element and separator character. Separator is placed smartly, i.e. only between elements.
- XML element comparison while ignoring specified members: `EqualityUtils`.
- Module lookup/selection: `ModuleSelectionHelper`.
- Singleton class reporting on the currently running inference: `RunningProject`.
- Various utilities for JUnit tests: `TestUtils`.
- Topological sorting of grammar: `TopologicalSort`.

3 Data flow

This is not an inference module and there is no real data flow in it. Refer to **??** to understand the inference process described by interfaces from `cz.cuni.mff.ksi.jinfer.base.interfaces.inference` package.

4 Extensibility

From one point of view, extensibility of *`Base`* means creating service providers implementing inference or other interfaces. This is described in the documentation for respective modules that use these interfaces.
On the other hand, if there is a need to share data structures, interfaces or logic across multiple developed modules (for example between custom schema generator and simplifier), it is advised to create a module similar to *`Base`* instead of changing it directly.