Nomenclature

Depending on your Object-oriented Programming background (or lack of it), you may find Logtalk nomenclature either familiar or at odds with the terms used in other languages. In addition, being a superset of Prolog, terms such as *predicate* and *method* are often used interchangeably. Logtalk inherits most of its nomenclature from Smalltalk, arguably (and somehow sadly!) not the most popular OOP language nowadays. In this section, we try to map nomenclatures from popular OOP languages such as C++ and Java to the Logtalk nomenclature.

C++ nomenclature

There are several C++ glossaries available on the Internet.

abstract class
: Logtalk supports *interfaces*/*protocols*, which are a much better way to provide the functionality of C++ abstract classes.

base class
: Logtalk uses the term *superclass* with the same meaning.

data member
: Logtalk uses *predicates* for representing both behavior and data.

constructor function
: There are no special methods for creating new objects in Logtalk. Instead, Logtalk provides a built-in predicate, `create_object/4`, which is often used to define more sophisticated object creation predicates.

derived class
: Logtalk uses the term *subclass* with the same meaning.

destructor function
: There are no special methods for deleting new objects in Logtalk. Instead, Logtalk provides a built-in predicate, `abolish_object/1`, which is often used to define more sophisticated object deletion predicates.

friend function
: Not supported in Logtalk. Nevertheless, see the manual section on *meta-predicates*.

member
: Logtalk uses the term predicate.

member function
: Logtalk uses predicates for representing both behavior and data.

nested class
: Logtalk does not support nested classes.

template
: Logtalk supports *parametric objects*, which allows you to get the similar functionality of templates at runtime.

this
: Logtalk uses the built-in context method `self/1` for retriving the current instance. Logtalk also provides a `this/1` but for returning the class containing the method being executed. Why the name clashes? Well, the notion of *self* cames from Smalltalk, which was there first.

virtual member function
: Logtalk allows any predicate defined in a class to be overriden on a descendant class. There is no `virtual` keyword in Logtalk.

Java nomenclature

There are several Java glossaries available on the Internet.

abstract class
: Moreover, there is no `abstract` keyword in Logtalk.

abstract method
: In Logtalk, you may simply declare a method (predicate) in a class without defining it, leaving its definition to some descendant sub-class.

extends
: There is no `extends` keyword in Logtalk. Class inheritance is indicated using *specialization* relations. Morevover, the *extends* relation is used in Logtalk to indicate protocol or prototype extension.

interface
: Logtalk uses the term *protocol* with the same meaning.

callback method
: Logtalk supports *event-driven programming*, the most common use context of callback methods.

class method
: Class methods are implemented in Logtalk by simply defining a predicate inside a class.

class variable
: Class variables are implemented in Logtalk by simply defining a predicate inside a class. Moreover, there is no `static` keyword in Logtalk.

constructor
: There are no special methods for creating new objects in Logtalk. Instead, Logtalk provides a built-in predicate, `create_object/4`, which is often used to define more sophisticated object creation predicates.

final
: There is no `final` keyword in Logtalk; methods may always be redefined in subclasses.

instance
: In Logtalk, an instance can be either created dynamically at runtime or defined statically in a source file in the same way as classes.

method
: Logtalk uses the term *predicate* interchangeably with the term *method*.

method call
: Logtalk usually uses the expression *message sending* for method calls, true to its Smalltalk heritage.

method signature
: Logtalk selects the method/predicate to execute in order to answer a method call based only on the method name and number of arguments. Logtalk (and Prolog) are not typed languages in the same sense as Java.

super
: Instead of a `super` keyword, Logtalk provides a *super* operator, `^^/1`, for calling overridden methods.

synchronized
: Logtalk supports *multi-threading programming* in selected Prolog compilers, including executing an predicate atomically.

this
: Logtalk uses the built-in context method `self/1` for retriving the current instance. Logtalk also provides a `this/1` but for returning the class containing the method being executed. Why the name clashes? Well, the notion of *self* cames from Smalltalk, which was there first.

XHTML + CSS Last updated on: July 23, 2006