NAME

oo::object -
root class of the class hierarchy

SYNOPSIS

```
package require TclOO

oo::object method ?arg ...?
```

CLASS HIERARCHY

oo::object

DESCRIPTION

The **oo::object** class is the root class of the object hierarchy; every object is an instance of this class. Since classes are themselves objects, they are instances of this class too. Objects are always referred to by their name, and may be **rename**d while maintaining their identity.

Instances of objects may be made with either the **create** or **new** methods of the **oo::object** object itself, or by invoking those methods on any of the subclass objects; see **oo::class** for more details. The configuration of individual objects (i.e., instance-specific methods, mixed-in classes, etc.) may be controlled with the **oo::objdefine** command.

Each object has a unique namespace associated with it, the instance namespace. This namespace holds all the instance variables of the object, and will be the current namespace whenever a method of the object is invoked (including a method of the class of the object). When the object is destroyed, its instance namespace is deleted. The instance namespace contains the object's **my** command, which may be used to invoke non-exported methods of the object or to create a reference to the object for the purpose of invocation which persists across renamings of the object.

CONSTRUCTOR

The **oo::object** class does not define an explicit constructor.

DESTRUCTOR

The **oo::object** class does not define an explicit destructor.

EXPORTED METHODS

The **oo::object** class supports the following exported methods:

obj **destroy**
: This method destroys the object, *obj*, that it is invoked upon, invoking any destructors on the object's class in the process. It is equivalent to using **rename** to delete the object command. The result of this method is always the empty string.

NON-EXPORTED METHODS

The **oo::object** class supports the following non-exported methods:

obj **eval** ?*arg ...*?
: This method concatenates the arguments, *arg*, as if with **concat**, and then evaluates the resulting script in the namespace that is uniquely associated with *obj*, returning the result of the evaluation.

obj **unknown ?***methodName*? ?*arg ...*?
: This method is called when an attempt to invoke the method *methodName* on object *obj* fails. The arguments that the user supplied to the method are given as *arg* arguments. If *methodName* is absent, the object was invoked with no method name at all (or any other arguments). The default implementation (i.e., the one defined by the **oo::object** class) generates a suitable error, detailing what methods the object supports given whether the object was invoked by its public name or through the **my** command.

obj **variable** ?*varName ...*?
: This method arranges for each variable called *varName* to be linked from the object *obj*'s unique namespace into the caller's context. Thus, if it is invoked from inside a procedure then the namespace variable in the object is linked to the local variable in the procedure. Each *varName* argument must not have any namespace separators in it. The result is the empty string.

obj **varname** *varName*
: This method returns the globally qualified name of the variable *varName* in the unique namespace for the object *obj*.

obj **<cloned>** *sourceObjectName*
: This method is used by the **oo::object** command to copy the state of one object to another. It is responsible for copying the procedures and variables of the namespace of the source object (*sourceObjectName*) to the current object. It does not copy any other types of commands or any traces on the variables; that can be added if desired by overriding this method in a subclass.

EXAMPLES

This example demonstrates basic use of an object.

```
set obj [oo::object new]
$obj foo             -> error "unknown method foo"
oo::objdefine $obj method foo {} {
    my variable count
    puts "bar[incr count]"
}
$obj foo             -> prints "bar1"
$obj foo             -> prints "bar2"
$obj variable count  -> error "unknown method variable"
$obj destroy
$obj foo             -> error "unknown command obj"
```

SEE ALSO

my(n), oo::class(n)

KEYWORDS

base class, class, object, root class