Object Tcl

Contents

Classes

In Object Tcl, a class describes the behavior and state information common to a particular collection of objects. Objects are of a particular class in the same way as variables are of a particular type.

Objects are created by instantiating a class and specifying its initial creation parameters.

There are two parts to the description of a class. The first part is the class interface which is a description of the class with respect to the outside world. The second part is the class implementation which is the description of how the behavior of the class is implemented.

Class Interface

The command:

`otclInterface` *`name ?-isA classlist? body`*

provides a new class interface description.

The *name* parameter specifies the name of the new class.

The *-isA classList* parameter is optional and may be used to specify class inheritance. The *classList* must be a list of names of classes that have already been fully described to the Object Tcl interpreter, or built-in C++ classes. No name may be repeated in the list of classes.

The *body* parameter is a Tcl script that describes the interface of the new class. Within the body script, the following commands may be used:

`constructor` *`argList`*
: This command describes the interface of the constructor method for the class in question. The *argList* parameter specifies a list of arguments for the constructor. Default values may be specified.

`method` *`name argList`*
: This command describes the interface of an instance method for the class in question. The *argList* parameter specifies a list of arguments for the instance method. Default values may be specified.

`classMethod` *`name argList`*
: This command describes the interface of a class method for the class in question. The *argList* parameter specifies a list of argments for the class method. Default values may be specified.

All methods described in the class interface section, class or instance, are public.

Class Implementation

The command:

`otclImplementation` *`name body`*

provides the class implementation description.

The *name* parameter specifies the name of the class that this command is providing the implementation for. The class named must be a class that has previously had its class interface specified using the otclInterface command and has not had its class implementation specified.

The *body* parameter is a Tcl script that describes the implementation of the class. Within the *body* script, the following commands may be used:

`constructor` *`argList parentList body`*

This command describes the implementation of the constructor method. The *argList* parameter specifies the arguments to the constructor, the *parentList* parameter describes the construction of the parent classes and the *body* parameter is a Tcl script that describes the body of the constructor.

`destructor` *`body`*

This command describes the implementation of the destructor method. The *body* parameter is a Tcl script that describes the body of the destructor.

`method` *`name argList body`*

This command describes the implementation of an instance method. The *name* parameter specifies the name of the method, the *argList* specifies the arguments accepted by the method and the *body* is a Tcl script that describes the body of the method.

`classMethod` *`name argList body`*

This command describes the implementation of a class method. The *name* parameter specifies the name of the method, the *argList* specifies the arguments accepted by the method and the *body* is a Tcl script that describes the body of the method.

`attribute` *`name ?value?`*

This command describes an instance attribute. The *name* parameter specifies the name of the attribute and the optional *value* parameter specifies the value to initialize the attribute with when an instance is created.

`classAttribute` *`name value`*

This command describes a class attribute. The *name* parameter specifies the name of the attribute and the *value* parameter specifies the value to initialize the attribute with when the otclImplementation command has completed.

Inheritance

An Object Tcl class may use the *-isA classList* option in the otclInterface command to specify class inheritance. Object Tcl supports single and multiple inheritance. Object Tcl classes may inherit from other Object Tcl classes or built-in C++ classes.

Object Tcl classes may only inherit from other Object Tcl classes that have had both their interface and implementation successfully described.

Multiple inheritance of a single root class results in tree inheritance rather than lattice. Object Tcl does not provide an equivalent to the C++ virtual base classes.

Attributes

Attributes are used to record state information within classes and objects. There are two types of attribute: instance and class. Instance attributes have separate instantiations for each object of that class. Each object may alter its instance attributes independently of any other object of that class. Class attributes have only one instance that is shared by all objects of that class.

Both instance and class attributes are private in that they are not accessible from outside the class, only from within the class by class or instance method bodies.

Both instance attributes and class attributes may have an initial value specified. For class attributes, the initial value is mandatory. Instance attributes are initialized with their initial value or "" before the constructor for that class is executed. Class attributes are initialized with their initial value when the class implementation description is complete.

This

An implicit instance attribute exists called **this**. This attribute is only in scope within an instance method and its value is a reference to the current object. The **this** attribute may be used to invoke other instance methods upon the current object or be passed to other commands as an argument.

It is forbidden to modify the value of **this**, but this is not detected by the Object Tcl system.

Naming

Class attributes and instance attributes share the same name space and must be unique within a class.

The name "this" is reserved.

Access

From within an instance method all instance attributes and class attributes are available as local Tcl variables. The Tcl `$` substitution syntax may be used.

From within class methods all class attributes are available as local Tcl variables. The Tcl `$` substitution syntax may be used.

Arrays

Object Tcl supports instance and class array attributes. The description of an array attribute takes the following form:

`attribute` *`name`*`()` *`?value?`*

The brackets after the attribute *name*, with no white space, indicates that the attribute is an array. If an initial value is supplied it must be a list of `{`*`key value`*`}` pairs. For example:

```
attribute sales() {{jan 12} {feb 13} {mar 44}}
```

Array attributes may be manipulated in the same way as Tcl arrays using the `$` substitution syntax with a key as a bracketed suffix. For example:

```
puts $sales(mar)
```

Methods

Methods are used to give behavior to a class of object. There are two types of methods: class and instance. Class methods operate at the class scope and may manipulate class attributes. Instance methods operate on an object scope and must be invoked upon an object of that class. Instance methods may access class attributes and instance attributes. Instance methods may invoke class methods. Class methods may only invoke instance methods if an object reference is available.

Naming

Method names must be unique within a class. Instance and class methods of the same class share the same name space.

If a class uses inheritance then it may describe a method with the same name as a method in one of its parent classes. This is allowed as the dynamic binding resolves the name at invocation time.

A class may also describe a class method with the same name as a class or instance method in one of its parent classes. In this case the syntax of class method invocation identifies the class to find the method in.

The names "constructor" and "destructor" are reserved.

Naming an instance method unknown implies specific behavior.

Unknown

Object Tcl classes, and even C++ classes exported to Object Tcl, may have an instance method called **unknown**. The **unknown** method is invoked automatically by the Object Tcl system when an attempt is made to invoke a method upon the object that is not in the class's implementation.

The **unknown** method is called with the arguments specified in the original method invocation with the name of the requested method prepended.

The **unknown** method may be inherited from a superclass and the rules of dynamic binding operate.

Dynamic Binding

Classes that use inheritance may provide methods that are also provided by their parent classes. Dynamic binding is the mechanism whereby the method of the most specific class is executed. This allows classes to specialize the classes they inherit from to make them more specific.

Given an object of a particular class, the invocation of a method involves a search through the inheritance hierarchy for that method. The search starts at the most specific class, the instantiated class, and proceeds in a depth-first search. If multiple inheritance is used, the list of superclasses is searched in the order they were given in the *-isA classList* option of the otclInterface command. There is no checking for two descriptions of the named method at the same level in the hierarchy. The first method in the search order will be executed.

Public & Private Methods

Class or instance methods may be public or private. If a method is public then it may be invoked by any other piece of Tcl code. If a method is private then it may only be invoked from another method of that class.

Methods are public if their interface is described in the otclInterface command, otherwise they are private.

For public methods, default parameter values must be specified in the class interface. Private methods must have their default values specified in the class implementation.

Constructors

The constructor method is a special instance method that is invoked automatically upon the construction of a new instance of the class. Constructors may be used to initialize attribute values and perform initialization behavior.

The implementation description for a constructor method contains a parameter called *parentList*. This list contains a description of the parameters to pass to the constructor of any inherited classes. The *parentList* takes the form of a list of scripts, where each script starts with the name of the parent class and is followed by a series of Tcl expressions that are to be passed to the parent class constructor. If an inherited class constructor takes no arguments then it need not appear as an item in the *parentList*. The expressions in the *parentList* may make use of any class or instance attributes of the calling class or any of the formal arguments passed to the calling class constructor.

The *parentList* parameter is mandatory even for classes that do not use inheritance. In this case they must specify an empty *parentList*.

Constructors are optional.

When using inheritance, the most general class constructor body is evaluated first and the most specific last. If multiple inheritance is used then the constructor for the inherited classes are executed in the order they are described in the *-isA classList* option of the otclInterface command for that class.

A constructor is considered an instance method in that the this attribute is available and may be used to invoke other methods upon the object. The dynamic binding of methods will operate correctly but it must be remembered that the object is undergoing construction. Using dynamic binding it would be possible to invoke a method from a part of the class hierarchy for that object that is still waiting for its constructor to be evaluated.

Destructors

The destructor is a special method that cannot be public (there is no command appropriate for making it public anyway). Destructors are the opposite to constructors in that they are invoked automatically upon the destruction of an instance of the class.

Destructors are optional.

Destructors are executed in the reverse order to constructors. The most specific destructor is executed first, and if multiple inheritance is used the destructors for the multiple superclasses are invoked in the reverse order to that specified in the *-isA classList* option of the otclInterface command for that class.

As with constructors, destructors are treated the same as instance methods. They may call other methods upon the object, so it is possible to call methods upon parts of the object that have already had their destructor invoked.

Invocation

Class methods and instance methods are invoked in different ways.

A class method may be invoked using the following syntax:

```
class method ?args ...?
```

where *class* is the name of the class to invoke the method upon, *method* is the name of the class method to invoke and *arg* is an argument to pass to the class method.

An instance method may be invoked using the following syntax:

```
obref method ?arg ...?
```

where *obref* is an object reference, *method* is the name of the method to invoke and *arg* is an argument to pass to the instance method.

Another instance method may be invoked from within the body of another instance method using the following syntax:

```
$this ?-parent? method ?arg ...?
```

where *method* is the name of the method to invoke and *arg* is an argument to pass to the instance method. The *?-parent?* option may be used to force the invocation of the method from the immediate parent class. This facility may be used to override the dynamic binding feature, thus allowing the most specific version of a method to utilize a more general version of the method. The parent class must be one of the classes named in the *-isA classList* section of the otclInterface command.

Constructors and destructors cannot be invoked explicitly.

Results

Class methods and instance methods may return results using the Tcl return command as in normal Tcl procedures.

Objects

Objects in Object Tcl are instances of classes. All instances of the same class share common behavior, through methods, and maintain similar state information, through attributes.

Objects are constructed by instantiating an Object Tcl class or a built-in C++ class. Methods may then be invoked upon that object which alter its state. Finally an object may be destroyed.

Constructing

An object is constructed by instantiating a class using the following command:

`otclNew` *`class ?arg...?`*

where *class* specifies the name of the class to instantiate and *arg* specifies an argument to pass to the constructor of the class. The result of this command is a reference to the new object. This reference may be used to invoke methods upon that object.

Destructing

When an object is finished with it may be deleted using the following command:

`otclDelete` *`objref`*

where *objref* is a reference to an Object Tcl object as created by the otclNew command.

After an object has been deleted, all references to that object are invalid and must not be used. The result of dereferencing an invalid reference will vary depending on the object mapping implementation. If the object mapper is capable of detecting the dereference then a Tcl error will be reported, otherwise the dereference will result in memory misuse and probable core dump.

---

otcl@x.co.uk