t Classes

A class defines a type of object and a set of operations associated with that object. The object is defined in terms of its attributes, while operations on that object are defined in methods of that class. An object of a class is referred to as an instance of that class.

The Prograph class system provides a means for constructing a new class from an existing class, by the mechanism of inheritance. The new class (the subclass) inherits all the attributes and methods of its parent (its immediate ancestor); that parent may in turn have inherited attributes and methods from its parents. The subclass can have its own additional attributes and methods. A class can inherit from at most one parent: this is called single inheritance.

Editor Actions

The Prograph editor depicts classes as hexagonal icons in the Classes window.

o
A new class is created by Command-clicking in unoccupied space (Cmd-clicking space) in the Classes window.

o
The left side of a class icon responds to double-clicking by opening a window onto the attributes of that class.

o
The right side of a class icon responds to double-clicking by opening a window onto the methods of that class.

o A new subclass is created by selecting a class, holding down the Option key, and Cmd-clicking space (somewhere below the parent class is most appropriate). A new class icon appears, with a connecting inheritance link to its parent.

Class Names

Names of classes must be unique.

The following names are reserved for use by the compiler, and cannot be used as names of classes: object, class, method, attribute, natural, number, persistent, array, text, Ptr, Handle. In addition, names of Prograph data types and of Macintosh structures that are supported by Prograph cannot be used as class names.

Attributes

An attribute is a named slot for holding a value. An attribute's name must be unique within its class.

The word default cannot be used as the name of an attribute.

There are two types of attributes

o An Instance attribute can have a different value for each instance of the class.

o A Class attribute has one value for the class as a whole; its value is shared by all the instances of the class.

In the Attributes window a horizontal fuzzy line separates instance attributes (below the line) from class attributes (above the line). An attribute icon has its name below the icon and its default value above. An instance attribute icon has a triangular shape, while a class attribute icon has a hexagonal shape. Inherited attributes are depicted with a downward-pointing arrow within their icons.

An attribute is created by Cmd-clicking space in the attributes window. Its name is typed just below the attribute icon that appears, while double-clicking the icon opens a window for entering its default value.

Methods

There are four types of class methods: Plain, Get, Set, and Initialization. The name of a method must be unique within its class for a method of that type.

In the Methods window, cmd-clicking in space creates a method icon. Its name is typed below the icon. When first created, a method icon is Plain. Selecting the appropriate menu item from the Opers menu transforms it into a Get, Set, or Initialization method, each with a distinctive icon.

Plain Method

This is the default method type.

Get Method

The name of a Get Method must be the same as that of an attribute. The Get Method is used to access the value of that attribute when a Get Attribute operation that has a name in /attribute-name format is executed. A Get Method has one input (the instance) and two outputs (left to right, the instance and the value accessed). See the “Operations” section of this chapter for further details.

Set Method

The name of a Set Method must be the same as that of an attribute. The Set Method is used to set the value of that attribute when a Set Attribute operation that has a name in /attribute-name format is executed. A Set Method has two inputs (left to right, the instance and the value to which the attribute will be set) and one output (the instance). See the “Operations” section of this chapter for further details.

Initialization Method

There can be only one Initialization Method per class. An Initialization Method is always given the name <<>>by the Prograph editor.

An Initialization Method, if it exists, is automatically called immediately upon creation of a new instance of its class. It may be used to perform various initializations on that instance, such as setting attribute values. It has one input (the instance) and one output (the instance).

t Methods

Universal Methods

A universal method is not associated with any class. Its name must be unique among universal methods.

In the Universal methods window, a method is created by cmd-clicking space. A universal method icon looks just like a Plain class method icon.

Cases of Methods

A method consists of a sequence of one or more cases. A case itself is a dataflow diagram of operations. When the method executes, the cases are executed in sequence until one succeeds completely or until activation of a control stops execution of the method.

A case consists of an input bar, an output bar, and, in between, operations with connecting datalinks. Data flows into a case through the input bar, and out of the case through the output bar. Each case of a method has the same number of inputs and outputs (that is, the same arity).

hen a method icon is double-clicked, its first case window is opened. The case window banner displays:

o
the name of the class (if the method is class-based)

o
a slash, “/” (if the method is class-based)

o
the name of the method

o
an expression of the form “n:m”, where n is the number of the case being depicted and m is the total number of cases in the method

A new case is created by cmd-clicking space in the case palette. Cases, and their corresponding numbering, can be reordered by dragging the numbered case icons in the case palette.

Compact Code

The Prograph Classic Editor/Interpreter supports a compact form of Prograph method, in which the visual information for displaying its case structure is not kept in RAM memory. It thus requires less memory to display and execute such methods. In addition, compact methods can be made execute-only, locking out access to their source code.