Part III: Definitions of Terms

Many terms in Smalltalk are also used in other languages such as C++, Eiffel, and Simula. While there may be a surface similarity, the terms often mean quite different things. This chapter defines many common terms in the context of Smalltalk.

Definitions of Terms: Objects

Frequently Asked Question

3.1 • What is an object?

In Smalltalk, an *object* is a collection of data and code. The data is hidden within the object and cannot be accessed except through code in the object.

Frequently Asked Question

3.2 • What is a method?

A method is a Smalltalk subroutine, no more and no less. It always takes at least one parameter (`self`) and always returns one value.

Frequently Asked Question

3.3 • What is a class?

A *class* is the definition of an object. The class of an object holds a list of the data that an object will have, and it holds the code for the object. Each different kind of object has its own class.

Smalltalk programs are created by writing classes. Each class defines data and a set of methods, the code, which operates on that data. Figure 1 illustrates a class and two instances.

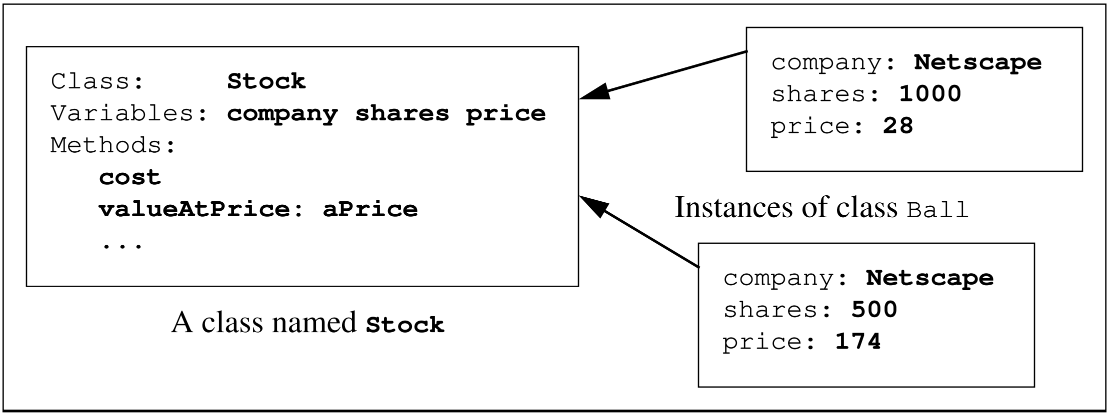

Figure 1: A Class and Instances. A class named `Stock` is illustrated on the left, and two instances of the class are shown on the right. The class contains a list of the variables, `company`, `shares`, and `price`, that each instance of the class will have. The class also contains code that each of the instances will share, in this case including `cost` and `valueAtPrice:`.

Frequently Asked Question

3.4 • What is an instance?

An *instance* of a class is simply an object. It holds data and a pointer to the class which holds the code.

Frequently Asked Question

3.5 • What is a subclass?

A *subclass* is a class that is defined using another class as a basis. The other class is called the *superclass* (or parent class). The subclass adds to the implementation (or shares the implementation) of the superclass.

Subclassing in Smalltalk is implementation sharing not subtyping.

See [4.39] *'What is a type?'* on page 68, [4.40] *'Does Smalltalk have types?'* on page 69, and [4.41] *'Is there any disagreement about types in Smalltalk?'* on page 71.

Frequently Asked Question

3.6 • What are encapsulation and information hiding?

Encapsulation and information hiding refer to the act of hiding data behind a wall of code. Encapsulation is a basic characteristic of all OOP languages and of some non-OOP languages.

Frequently Asked Question

3.7 • What is a hierarchy?

A hierarchy is an outline or tree of class definitions. In Smalltalk, the top (or root) of the hierarchy is a very general object named `Object`. All other objects are subclasses of `Object` or of some other class (which must be a subclass of object or another class).

There are some weird exceptions; see xxxxxxx.

Frequently Asked Question

3.8 • What is polymorphism?

Polymorphism means multiple meanings. In Smalltalk, polymorphism means that there are multiple methods with the same name and that any one of them might be invoked by a single message send.

Example

A group of classes define various kinds of employees. There is one class for retired employees, one for active employees, one for executives, one for part time employees, and so on.

Each class might have a `computePaycheck` method to compute the amount of an employee's paycheck, but the methods would be different because of the differing requirements of each kind of employee.

At the point where a paycheck amount needs to be computed, the `computePaycheck` message is sent to an instance of one of the employee classes:

```
anEmployee computePaycheck
```

Definitions of Terms: Messages

Frequently Asked Question

3.9 • What is a message?

A message is an operation that can be requested of an object. It consists of the name of the message, which is the name of a method, and values for any associated parameters. All messages are considered to have at least one parameter, the value that will become `self`.

If an object has a method named `zotFarb:` then these three parts are a message:

1. The name `zotFarb:`
2. The value that will be `self`
3. A value to pass for its one parameter.

Frequently Asked Question

3.10 • What is a message send?

A message send is the process of invoking a method in an object using a message. It has two steps: finding the method, and invoking the method.

Finding the method

The name of the message is compared with methods in the target object. If not found, then each of the parents are examined successively until one is found. (If none is found, an error condition is raised.) In general, the object to which the message is sent is not known until runtime and this lookup process must logically occur each time a message is sent.

Invoking the method

Once the appropriate method is found, then the method is invoked just as if it were a procedure in a procedural language. Parameters are bound to their local names, the stack is appropriately massaged, and execution begins.

Upon termination, control returns to the method that sent the message to a point just after the message send. A value is always returned although the sender has the option of ignoring it.

Frequently Asked Question

3.11 • How does a message send differ from a procedure call?

A message send is identical to a procedure call once a method is found to invoke.

See See *3.10 'What is a message send?'*

Definitions of Terms: Classes

Frequently Asked Question

3.12 • Are classes objects?

Yes; classes in Smalltalk are objects and are instances of some other object.

See "Metaclasses" in section "Implementation" on Page 148.

Frequently Asked Question

3.13 • What is an abstract class?

An abstract class is just a class; Smalltalk makes no distinction between abstract classes and any other kind. However, programmers find it useful to define classes that are not complete, and that will never have instances. Such classes define some general concept and some protocol (a set of methods) that all its subclasses will inherit.

The base class library contains a number of abstract classes, or abstract superclasses, as they are sometime called. These include:

Class	**Abstracts**
`Object`	The basic idea of an object: existence, size, ...
`Magnitude`	Values that compare uniquely with < and >
`Number`	Values that can be added, subtracted, ...
`Stream`	Sequences of values
`Collection`	Groups of values