3
Классы и экземпляры

Описание протокола

Категории сообщений

Описание реализации

Объявление переменных

Переменные экземпляра

Общие переменные

Методы

Имена аргументов

Возвращаемые значения

Псевдопеременная self

Временные переменные

Примитивные методы

Словарь терминов.

Объекты представляют различные компоненты системы Smalltalk-80 — числа, структуры данных, процессы, дисковые файлы, планировщики процессов, редакторы текстов, компиляторы и приложения. Сообщения представляют взаимодействия между компонентами системы Smalltalk-80 — арифметические действия, доступ к данным, структуры управления, создание файлов, обработку текстов, компиляцию, использование приложений. Сообщения делают функциональность каждого объекта доступной другим объектам системы, скрывая при этом его реализацию. В предыдущей главе был представлен синтаксис выражений для описания объектов и сообщений. Внимание в основном было сосредоточено на том, как сообщения применяются для доступа к функциональности объектов. В этой главе вводится синтаксис описания методов и классов системы, чтобы показать, как функциональность объектов реализуется.

Каждый объект системы Smalltalk-80 — *экземпляр* некоторого *класса*. Все экземпляры класса имеют одинаковый интерфейс; класс описывает, как выполнять каждую операцию, доступную через этот интерфейс. Каждая операция описывается *методом*. Имя сообщения определяет, какой тип операции получатель сообщения должен выполнить, поэтому класс имеет единственный метод для каждого имени сообщения из своего интерфейса. Когда объекту посылается сообщение, выполняется метод, связанный с этим типом сообщения в классе получателя сообщения. Класс также определяет, какой тип памяти будут иметь его экземпляры.

Каждый класс имеет имя, которое описывает тип компонента, представляемого его экземплярами. Имя класса служит двум главным целям: это простой способ для самоидентификации экземпляров и это возможность ссылаться на класс в любом выражении. Поскольку классы — компоненты системы Smalltalk-80, они тоже представляются объектами. Имя класса автоматически становится именем глобальной общей переменной. Значение этой переменной — объект, представляющий класс. Так как имена классов — имена общих переменных, то они должны начинаться с прописной буквы.

Новые объекты создаются посылкой сообщений классам. Большинство классов создают свой новый экземпляр в ответ на унарное сообщение new (новый). Например,

```
OrderedCollection new
```

возвращает новый набор — экземпляр системного класса OrderedCollection (УпорядоченныйНабор). Новый набор создается пустым. Некоторые классы создают новые экземпляры в ответ на другие сообщения. Например, класс Time (Время), чьи экземпляры представляют время суток, в ответ на сообщение now (сейчас) создает новый экземпляр, представляющий текущее время. Класс Data (Дата), чьи экземпляры представляют день года, в ответ на сообщение today (сегодня) создает новый экземпляр, представляющий текущую дату. Когда создается новый экземпляр, ему автоматически становятся доступны методы из класса, получившего сообщение о создании экземпляра.

В этой главе определяются два способа представления класса, один для описания функциональности экземпляров, другой для описания реализации этой функциональности.

1. *Описание протокола* перечисляет сообщения из интерфейса сообщений экземпляра. Каждое сообщение сопровождается комментарием, который описывает, что будет делать объект, когда получит данный вид сообщения.
2. *Описание реализации* показывает, как реализована функциональность, представленная в описании протокола. Описание реализации задает структуру частной памяти объекта и набор методов, определяющих, как экземпляры будут выполнять требуемые операции.

Есть и третий способ представления классов системы — через интерактивное окно, называемое окном просмотра или браузером системы. Браузер — часть интерфейса программирования и он доступен в работающей системе Smalltalk-80. Описание протокола и описание реализации необходимы для неинтерактивного документирования, как в этой книге. Браузер кратко описывается в главе 17.

Описание протокола

Описание протокола перечисляет сообщения, понимаемые экземплярами рассматриваемого класса. Каждое сообщение приводится с комментарием о его функциональности. Комментарий описывает, какие действия будут выполнены при получении данного сообщения и какое значение при этом будет возвращено. То есть, комментарий описывает, *что* будет сделано, а не то, *как* будет выполнена операция. Если в комментарии ничего не говорится о том, какое значение будет возвращено, предполагается, что таким значением будет получатель сообщения.

Например, часть описания протокола для сообщения с именем spend:for: из класса FinancialHistory (ФинансовыйОтчет) имеет вид:

spend: amount for: reason	запоминает количество денег, amount, потраченных по причине reason.

Сообщения в описании протокола записываются в форме образца сообщения. Образец сообщения содержит имя сообщения и множество имен аргументов (по одному для каждого аргумента), которые должны быть у сообщения с этим именем. Например, образцу сообщения

spend: amount for: reason

соответствует каждое их следующих трех выражений:

```
HouseholdFinances spend: 32.50 for: 'utilities'
HouseholdFinances spend: cost + tax for: 'food'
HouseholdFinances spend: 100 for: usualReason
```

Имена аргументов упоминаются в комментарии для ссылок на аргументы. Комментарий в вышеприведенном примере говорит о том, что первый аргумент представляет количество потраченных денег, а второй аргумент указывает, на что они были потрачены.

Категории сообщений

Сообщения, которые определяют однотипные действия, группируются в категории. Категории имеют имена для общей характеристики сообщений данной группы. Например, сообщения в классе FinancialHistory группируются в три категории с именами transaction recording (ведение записей), inquiries (запросы), initialization (инициализация). Разбиение по категориям выполняется только для того, чтобы сделать протокол более понятным и более удобным для чтения. На операции, выполняемые классом, это разбиение не влияет.

Полное описание протокола для класса FinancialHistory имеет следующий вид.

FinancialHistory protocol

transaction recording	
receive: amount from: source	Запоминает, что количество денег amount было получено из источника source.
spend: amount for: reason	Запоминает количество денег amount, потраченных по причине reason.
inquiries	
cashOnHand	Возвращает текущее количество наличных денег.
totalReceivedFrom: source	Возвращает общее количество денег, полученных из источника source.
totalSpendFor: reason	Возвращает общее количество денег, потраченных по причине reason.
initialization	
initialBalance: amount	Начинает финансовый отчет с наличным количеством денег amount.

Описание протокола предоставляет программисту достаточно информации о том, как использовать экземпляры данного класса. Из приведенного описания протокола мы узнаем, что любой экземпляр класса FinancialHistory должен отвечать на сообщения с именами receive:from:, spend:for:, cashOnHand, totalReceivedFrom:, totalSpendFor: и initialBalance:. Мы можем догадаться, что когда впервые создаем экземпляр этого класса, экземпляру нужно послать сообщение initialBalance:, чтобы определить начальные значения всех его переменных.

Описание реализации

Описание реализации состоит из трех частей:

1. имени класса;
2. объявления переменных, доступных экземплярам;
3. методов, используемых экземплярами для ответа на сообщения.

Образец полного описания реализации для класса FinancialHistory приведен ниже. Методы в описании реализации разбиты на те же категории, что и в описании протокола. В интерактивном браузере системы категории используются для обеспечения иерархического запроса пути доступа к разным частям описания класса. Нет никаких специальных символов, разделяющих различные части описания реализации. Здесь это делается при помощи изменения в шрифтах. Интерактивный браузер системы записывает и хранит части протокола класса независимо, а для доступа к ним он снабжен специальным структурным редактором.

class name FinancialHistory

instance variable names
cashOnHand
incomes
expenditures

instance methods

transaction recording

```
receive: amount from: source
    incomes   at: source
              put: (self totalReceivedFrom: source) + amount.
    cashOnHand ← cashOnHand + amount.

spend: amount for: reason
    expenditures  at: reason
                  put: (self totalSpentFor: reason) + amount.
    cashOnHand ← cashOnHand – amount.
```

inquiries

```
cashOnHand
    ↑cashOnHand

totalReceivedFrom: source
    (incomes includesKey: source)
        ifTrue: [↑incomes at: source]
        ifFalse: [↑0]
```

```
    totalSpentFor: reason
        (expenditures includesKey: reason)
            ifTrue: [↑expenditures at: reason]
            ifFalse: [↑0]

initialization
    initialBalance: amount
        cashOnHand ← amount.
        incomes ← Dictionary new.
        expenditures ← Dictionary new.
```

Это описание реализации класса отлично от того, что приведено на форзаце книги. Описание на форзаце содержит дополнительную часть class methods, которая будет рассматриваться в главе 5; там также опущен приведенный здесь метод инициализации.

Объявление переменных

Определенным в классе методам доступны пять различных видов переменных. Эти виды переменных различаются по широте их области доступа и по времени существования в системе.

Существуют два вида частных переменных, доступных только одному объекту.

1. *Переменные экземпляра* существуют в течение всего времени жизни объекта.
2. *Временные переменные* создаются для некоторого действия и доступны только во время этого действия.

Переменные экземпляра представляют текущее состояние объекта. Временные переменные отражают промежуточное состояние, необходимое для выполнения некоторого действия. Временные переменные обычно связаны с единичным выполнением метода: они создаются, когда сообщение вызывает метод для выполнения, и уничтожаются, когда метод завершает свою работу возвращением значения.

Три других вида переменных доступны более чем одному объекту и называются общими переменными. Они различаются по тому, насколько широко они доступны объектам системы.

3. *Переменные класса* доступны всем экземплярам данного класса.
4. *Глобальные переменные* доступны всем экземплярам всех классов (то есть всем объектам системы).
5. *Переменные пула* доступны экземплярам некоторого подмножества классов системы.

Большинство общих переменных в системе — либо переменные класса, либо глобальные переменные. Большинство глобальных переменных ссылаются на классы системы. В предыдущих примерах использовался экземпляр класса FinancialHistory с именем HouseholdFinances (ДомашниеФинансы). Мы использовали переменную HouseholdFinances так, как если бы она была определена в качестве имени глобальной переменной. Глобальные переменные используются для ссылок на объекты, которые не являются частями других объектов.

Имена общих переменных (пп. 3–5) пишутся с прописной буквы, в то время как имена частных переменных (пп. 1–2) — со строчной. Значение общей переменной не зависит от того, какой объект системы использует метод, в котором присутствует ее имя. Значение переменных экземпляра и временных переменных зависит от экземпляра, использующего метод, то есть от экземпляра, получившего сообщение.

Переменные экземпляра

Существует два типа переменных экземпляра: именованные и индексированные. Они различаются по описанию и использованию. Класс может иметь только именованные переменные, только индексированные переменные, или и те, и другие.

☐ *Именованные переменные экземпляра* Описание реализации включает множество имен для переменных экземпляра, которые составляют каждый конкретный экземпляр. Каждый экземпляр содержит по одной переменной, соответствующей каждому имени переменной экземпляра. Часть объявления переменных в описании реализации класса начинается словами instance variable names (имена переменных экземпляра). Класс FinancialHistory определяет три имени переменных экземпляра:

```
instance variable names    cashOnHand
                           incomes
                           expenditures
```

Каждый экземпляр класса FinancialHistory имеет два словаря для записи потраченных и полученных денег, и переменную для записи суммы наличных денег.

- expenditures ссылается на словарь, связывающий причины расходов с потраченными суммами.
- incomes ссылается на словарь, связывающий источники поступлений с суммами полученных денег.
- cashOnHand ссылается на число, представляющее сумму наличных денег.

Когда выражения в методах класса используют одно из имен cashOnHand, incomes или expenditures, эти выражения ссылаются на значения соответствующих переменных в экземпляре, который получил сообщение.

Когда путем посылки сообщения классу создается новый экземпляр, он получает полный набор переменных экземпляра, значения которым присваиваются с помощью некоторого сообщения инициализации. По умолчанию все переменные экземпляра инициализируются значением nil.

Например, для того, чтобы предыдущие примеры сообщений к HouseholdFinances работали, необходимо выполнить выражение, подобное следующему:

```
HouseholdFinances ← FinancialHistory new initialBalance: 350
```

Сообщение FinancialHistory new создает новый объект с тремя переменными экземпляра, ссылающимися на nil, а сообщение initialBalance: присваивает этим переменным более осмысленные значения.

☐ *Индексированные переменные экземпляра* Экземпляры некоторых классов могут иметь переменные, не доступные по имени. Такие переменные называются *индексированными переменными экземпляра*. Вместо доступа по имени, доступ к таким переменным происходит с помощью сообщений, использующих в качестве аргумента целое число, называемое индексом. Так как индексирование — форма ассоциативной связи между объектами, в системе существуют два фундаментальных индексирующих сообщения с теми же самыми именами, что и соответствующие сообщения к словарям — at: (вПозиции:) и at:put: (вПозиции:разместить:).

Например, экземпляры класса Array (Массив) имеют индексированные переменные. Пусть names — экземпляр класса Array, тогда выражение

```
names at: 1
```

вернет значение его первой индексированной переменной. А выражение

```
names at: 4 put: 'Adele'
```

сохранит строку 'Adele' в качестве значения четвертой индексированной переменной экземпляра. Допустимые значения индексов — целые числа от 1 до количества индексированных переменных в данном экземпляре.

Если предполагается, что экземпляры некоторого класса должны иметь индексированные переменные, то описание переменных должно включать строку indexed instance variables (индексированные переменные экземпляра). Например, часть описания реализации системного класса Array содержит строки:

```
class name                          Array
indexed instance variables
```

Каждый экземпляр класса, допускающего индексированные переменные, может иметь различное их количество. Все экземпляры класса FinancialHistory имеют по три переменных экземпляра, а экземпляры класса Array могут иметь любое число переменных.

Класс, чьи экземпляры имеют индексированные переменные, может иметь и именованные переменные. Тогда все экземпляры такого класса будут иметь одинаковое число именованных переменных, но могут иметь разное число индексированных переменных. Например, системный класс OrderedCollection, представляющий наборы, элементы которых упорядочены, имеет индексированные переменные экземпляра для хранения содержимого наборов. Экземпляр класса OrderedCollection может иметь больше пространства для хранения своих элементов, чем фактически используется в данное время. Две именованные переменные экземпляра запоминают индексы первого и последнего элементов набора.

```
class name                          OrderedCollection
instance variable names             firstIndex
                                    lastIndex
indexed instance variables
```

Все экземпляры класса OrderedCollection имеют две именованные переменные, но один экземпляр может иметь 5 индексированных переменных, другой — 15, следующий — 18, и так далее.

Именованные переменные экземпляра класса FinancialHistory — частные переменные в том смысле, что доступ к значениям этих переменных контролируется экземпляром. Класс может иметь, а может и не иметь сообщений, разрешающих прямой доступ к переменным экземпляра. Индексированные переменные в этом смысле не являются частными, так как их значения всегда доступны с помощью сообщений at: и at:put:. Поскольку эти сообщения — единственный способ получить доступ к значениям индексированных переменных, то они должны быть определены в системе.

Классы с индексированными переменными экземпляра создают новые экземпляры с помощью сообщения new: вместо обычного new. Аргумент сообщения new: определяет сколько должно быть создано индексированных переменных экземпляра. Например, выражение

```
list ← Array new: 10
```

создает массив из 10 элементов, каждый из которых первоначально ссылается на nil. Количество индексированных переменных экземпляра можно узнать, посылая экземпляру сообщение size. Так, в ответ на сообщение

```
list size
```

получим для данного экземпляра число 10.

Последовательное выполнение следующих выражений:

```
list ← Array new: 3.
list at: 1 put: 'one'.
list at: 2 put: 'two'.
list at: 3 put: 'three'
```

эквивалентно выполнению одного выражения

```
list ← #('one' 'two' 'three')
```

Общие переменные

Переменные, доступные более чем одному объекту, объединяются в группы, называемые *пулами*. Каждый класс имеет два или более пулов, доступных его экземплярам. Один пул с именем Smalltalk, содержащий глобальные переменные, доступен всем объектам системы. Каждый класс имеет и второй пул переменных, который содержит переменные класса и доступен только его экземплярам.

Кроме этих двух обязательных пулов, классу могут быть доступны и некоторые другие, специально созданные пулы переменных, которые доступны нескольким классам. Например, в системе существуют классы, представляющие и обрабатывающие текстовую информацию. Таким классам нужен доступ к ASCII-кодам символов, в том числе и к таким, которые не могут быть представлены визуально, например к возврату каретки, табуляции или пробелу. Все такие коды включены в виде переменных в пул с именем TextConstants (ТекстовыеКонстанты), который доступен всем классам, реализующим отображение и редактирование текста.

Если бы класс FinancialHistory имел переменную класса с именем SalesTaxRate и общий с другими классами пул с именем FinancialConstants, то его объявление начиналось бы так:

class name	FinancialHistory
instance variable names	cashOnHand incomes expenditures
class variable names	SalesTaxRate
shared pools	FinancialConstants

SalesTaxRate — имя переменной класса, поэтому оно может использоваться в любых методах в классе. С другой стороны, FinancialConstants — имя пула и все входящие в этот пул переменные также могут использоваться в выражениях.

Для того чтобы определить глобальную переменную, доступную всем классам и интерактивной системе пользователя, имя переменной должно быть внесено как ключ в словарь Smalltalk. Например, чтобы сделать имя AllHistories глобальным, надо выполнить выражение

```
Smalltalk at: #AllHistory put: nil
```

Затем, используя присваивание, можно задать необходимое значение этой переменной.

Методы

Метод описывает, как объект будет выполнять одну из своих операций. Метод состоит из образца сообщения и последовательности выражений, разделенных точками. Приводимый ниже пример описывает реакцию экземпляра класса FinancialHistory на сообщение, информирующее его о сделанных расходах.

```
spend: amount for: reason
    expenditures  at: reason
                  put: (self totalReceivedFrom: source) + amount.
    cashOnHand ← cashOnHand – amount
```

Образец сообщения, **spend: amount for: reason** (потрачено: количество на: цель), показывает, что этот метод будет использоваться в ответ на все сообщения с именем сообщения spend:for: к экземплярам класса. Первое выражение в теле этого метода добавит вновь потра-

ченные на цели reason деньги в количестве amount к уже потраченным на те же цели. Второе выражение метода — присваивание, уменьшит количество наличных денег на величину amount.

Имена аргументов

Понятие образца сообщения было введено в этой главе раньше. Образец сообщения состоит из имени сообщения и множества имен аргументов, по одному на каждый аргумент, входящий в сообщение с этим именем. Под образец сообщения подходит любое сообщение с тем же самым именем. Класс может иметь только один метод с именем, подходящим под образец сообщения. Когда сообщение посылается объекту, метод с подходящим образцом сообщения ищется в классе объекта-получателя. Выражения из найденного метода выполняются одно за другим. По завершении всех действий отправителю сообщения возвращается некоторое значение.

Имена переменных, содержащиеся в образце сообщения, рассматриваются как имена псевдопеременных, ссылающиеся на аргументы действительно посланного сообщения. Если, например, приведенный выше метод был бы вызван выражением

```
HouseholdFinances spend: 30.45 for: 'food'
```

то в процессе выполнения выражений метода псевдопеременная amount получила бы значение 30.45, а псевдопеременная reason указывала бы на строку 'food'. Если бы тот же самый метод был бы вызван выражением

```
HouseholdFinances spend: cost + tax for: 'food'
```

то сообщение + tax было бы послано объекту cost и на полученное значение в методе ссылались бы как на amount. Если бы переменная cost ссылалась на 100, а tax — на 6.5, то значение amount стало бы 106.5.

Так как имена аргументов в образце сообщения — псевдопеременные, то они могут использоваться для доступа к значениям подобно именам обычных переменных, но их значения не могут быть изменены присваиванием. Так, в методе с именем spend:for: выражение вида

```
amount ← amount * taxRate
```

было бы синтаксически неверно, так как значение amount не может быть изменено.

Возвращаемые значения

Метод для spend:for: не указывает, каким должно быть значение сообщения. Поэтому, по умолчанию возвращается получатель сообщения. Когда должно быть возвращено другое значение, в метод необходимо включить одно или несколько выражений возврата. Любое выражение может быть превращено в выражение возврата, если перед ним поставить символ ↑ (стрелка вверх). Можно возвратить значение переменной, например, как в выражении

```
↑cashOnHand
```

Можно вернуть значение другого сообщения:

```
↑expenditures at: reason
```

Литеральный объект может быть возвращен выражением вида

```
↑0
```

Даже выражение присваивания можно превратить в выражение возврата:

```
↑initialIndex ← 0
```

Вначале выполняется присваивание, а затем возвращается новое значение переменной.

Еще один пример использования выражения возврата реализован в totalSpentFor:.

```
totalSpentFor: reason
    (expenditures includesKey: reason)
        ifTrue: [↑expenditures at: reason]
        ifFalse: [↑0].
```

Этот метод состоит из единственного условного выражения. Если причина расходов reason присутствует в словаре expenditures, то возвращается количество потраченных по этой причине денег; в противном случае возвращается 0.

Псевдопеременная self

Вместе с именами псевдопеременных, использующимися для ссылки на аргументы сообщения, всем методам доступна псевдопеременная с именем self (сам), которая ссылается на самого получателя сообщения. Например, в методе для spend:for: сообщение totalSpentFor: посылается получателю сообщения spend:for:.

```
spend: amount for: reason
    expenditures  at: reason
                  put: (self totalSpentFor: reason) + amount.
    cashOnHand ← cashOnHand – amount
```

Когда этот метод начнет выполняться, первым делом сообщение totalSpentFor: будет послано тому же самому объекту (self), который получил первоначальное сообщение spend:for:. Результату, возвращенному этим сообщением, будет послано сообщение + amount, а полученный результат будет использован как второй аргумент в сообщении at:put:.

Псевдопеременную self можно использовать для реализации рекурсивных функций. Рассмотрим, например, сообщение factorial, понимаемое целыми числами, и вычисляющее соответствующую функцию факториала. Метод, связанный с сообщением factorial, следующий:

```
factorial
    self = 0 ifTrue: [↑1].
    self < 0
        ifTrue: [self error: 'factorial invalid']
        ifFalse: [↑self * (self – 1) factorial]
```

Получатель этого сообщения — целое число. Первое выражение проверяет, равен ли получатель сообщения нулю и если равен, возвращает единицу как результат. Второе выражение проверяет знак получателя. Если его знак отрицательный, то метод информирует программиста о том, что произошла ошибка (все объекты отвечают на сообщение error: извещением о возникшей ошибке). Если же получатель положительный, то возвращаемое методом значение равно

```
self * (self – 1) factorial
```

То есть, возвращаемое значение равно произведению получателя сообщения на факториал числа, на единицу меньшего получателя.

Временные переменные

Имена аргументов и псевдопеременная self доступны только во время выполнения метода. В добавление к этим именам псевдопеременных, метод во время выполнения может получить доступ и к некоторым другим переменным. Эти переменные называются временными. Временные переменные объявляются в методе между образцом сообщения и телом метода. Объявление временных переменных состоит из множества имен, заключенных между двумя вертикальными линиями. Метод для spend:for: в классе FinancialHistory может быть переписан с временной переменной для хранения предыдущих расходов:

```
spend: amount for: reason
    | previousExpenditures |
    previousExpenditures ← self totalSpendFor: reason.
    expenditures  at: reason
                  put: previousExpenditures + amount.
    cashOnHand ← cashOnHand – amount
```

Значения временных переменных доступны методу только во время его выполнения и уничтожаются по его завершении. Вначале все временные переменные ссылаются на nil.

В интерактивной среде Smalltalk-80 программист может проверять любые алгоритмы, использующие временные переменные. Проверка выполняется с помощью вертикальных линий для объявления переменных, существующих только во время проверки. Предположим, что проверяемое выражение включает ссылку на переменную list. Если переменная list не объявлена, то попытка выполнить выражение вызовет сообщение о синтаксической ошибке. Необходимо предварительно объявить list как временную переменную, поставив перед выполняемыми выражениями объявление | list |. Выполняемые выражения должны разделяться точками, как и в синтаксисе метода.

```
| list|
list ← Array new: 3.
list at: 1 put: 'one'.
list at: 2 put: 'four'.
list printString
```

Программист выбирает все пять строк (объявление и выражения) и запрашивает их выполнение. Переменная list доступна только во время выполнения выбранных выражений.

Примитивные методы

Когда объект получает сообщение, он обычно сразу начинает посылать другие сообщения. Где же *реально* что-то происходит? Объект может в ответ на полученное сообщение, например, изменить значения своих переменных экземпляра, и это, конечно, квалифицируется как "нечто происшедшее". Однако только таких изменений едва ли достаточно. И их действительно недостаточно. Все изменения в системе *определяются* сообщениями, однако *не* на все сообщения система отвечает вызовами методов Smalltalk-80. Существует около сотни *примитивных методов*, которые умеет выполнять только виртуальная машина Smalltalk-80. Примеры сообщений, которые вызывают примитивы: сообщение +, посылаемое малых целым числам; сообщение at: объектам с индексированными переменными экземпляра; сообщения new и new: классам. Когда число 3 получает сообщение + 4, оно не выполняет метод Smalltalk-80. Число 7 как значение выражения возвращает примитивный метод.

Метод, реализованный как примитивный, начинается с выражения вида

```
<primitive #>
```

где # — целое число, указывающее на номер примитивного метода, который здесь определен. Если примитивный метод не может корректно выполниться, то выполняются следующие за ним выражения языка Smalltalk-80, которые обрабатывают аварийную ситуацию.

Словарь терминов.

класс	объект системы, описывающий реализацию множества однотипных объектов.
экземпляр	один из объектов, описанных классом; он имеет память и отвечает на сообщения.
переменная экземпляра	переменная, доступная одному объекту во время существования этого объекта в системе; может быть именованной или индексированной.
описание протокола	описание класса на языке сообщений, посылаемых его экземплярам.
описание реализации	описание класса в терминах памяти его экземпляров и множества методов, определяющих, как экземпляры выполняют необходимые операции.
образец сообщения	имя сообщения и множество имен переменных, по одному для каждого аргумента, которые сообщение с данным именем должно иметь.
временная переменная	переменная, создаваемая для специального действия и доступная только во время выполнения этого действия.
переменная класса	переменная, общая для всех экземпляров одного класса.
глобальная переменная	переменная, общая для всех объектов системы.
переменная пула	переменная, общая для всех экземпляров некоторого множества классов.
Smalltalk	пул переменных, содержащий глобальные переменные, общие для всех классов.
метод	процедура, описывающая, как выполняется одна из операций объекта; состоит из образца сообщения, описания временных переменных и последовательности выражений. Метод выполняется, когда сообщение, соответствующее его образцу, посылается экземпляру класса, в котором метод находится.
имя аргумента	имя псевдопеременной, доступное методу только во время его выполнения; значение имени аргумента — аргумент сообщения, которое вызвало данный метод.
↑	когда используется в методе, показывает, что значение следующего за ним выражения должно быть возвращено как значение метода.
self	псевдопеременная, указывающая на получателя сообщения.
категория сообщений	группа методов в описании класса.
примитивный метод	операция, которая не является последовательностью выражений языка Smalltalk-80, а выполняется непосредственно виртуальной машиной.