ЧАСТЬ ВТОРАЯ

Первая часть книги содержала обзор языка Smalltalk-80, как с точки зрения семантики объектов системы и посылаемых сообщений, так и с точки зрения синтаксиса выражений. Программист в первую очередь должен понимать семантику языка, а именно то, что вся информация в системе представляется в форме объектов и все преобразования осуществляются посредством посылки сообщений объектам. Каждый объект описывается классом. Каждый класс, за исключением класса Object, является подклассом другого класса. Программирование в Smalltalk-80 включает в себя описание новых классов, создание экземпляров классов и посылку сообщений экземплярам. Синтаксис языка Smalltalk-80 определяет три формы, которые могут принимать сообщения: унарную, бинарную и ключевую. Успешное использование языка требует от программиста знания всех основных видов объектов системы и посылаемых им сообщений.

Семантика и синтаксис языка относительно просты. Тем не менее, система является большой и мощной благодаря количеству и разнообразию содержащихся в ней объектов. В системе Smalltalk-80 восемь важнейших категорий классов: ядро системы и его поддержка, линейные величины, числа, наборы, потоки, классы, независимые процессы и графика. Протоколы для этих видов объектов рассматриваются в двенадцати главах второй части книги. Каждая глава начинается с диаграммы иерархии классов, впервые представленной в главе 1, в которой специально выделяются классы, описываемые в этой главе. Еще в трех главах второй части приводятся примеры выражений языка и описаний классов Smalltalk-80.

Классы в системе Smalltalk-80 организованы в линейную иерархию. Главы второй части дают энциклопедическую информацию о протоколе класса: определяются категории сообщений, каждое сообщение комментируется, приводятся примеры. В представляемом протоколе класса описываются, однако, только те сообщения, которые вводятся этим классом. Полный протокол сообщений определяется просмотром протокола самого класса и всех его суперклассов. Поэтому полезно начать изучение с класса Object и продолжать его, спускаясь по иерархии классов вниз, с тем, чтобы наследуемый протокол изучался совместно с собственным протоколом класса.

6
Протокол для всех объектов

Проверка функциональности объекта

Сравнение объектов

Копирование объектов

Доступ к частям объекта

Печать и сохранение объектов

Обработка ошибок

Любая сущность в системе является объектом. Общий для всех объектов протокол представлен в описании класса Object. Это значит, что любой объект, созданный в системе, отвечает на сообщения, определенные в классе Object. Обычно это сообщения, обеспечивающие принятое по умолчанию поведение объектов и, тем самым, создающие тот фундамент, на котором можно разрабатывать новые виды объектов, добавляя новые сообщения или модифицируя ответы на уже существующие сообщения. При рассмотрении протокола класса Object примерами нам служат числа, такие как 3 или 16.23, наборы типа 'this is a string' или #(this is an array), объекты nil и true, объекты, описывающие классы, такие как Collection, SmallInteger и, конечно, сам Object.

Описание протокола для класса Object представлено в этой главе не полностью. Мы опустили сообщения, относящиеся к управлению сообщениями, к специальным отношениям зависимости и к системным примитивам. Они описываются в главе 14.

Проверка функциональности объекта

Каждый объект — экземпляр некоторого класса. Функциональность объекта определяется его классом. Функциональность проверяется двумя способами: явным именованием класса, чтобы определить, является ли он классом или суперклассом объекта, и указанием имени сообщения, чтобы определить, может ли объект отвечать на него. Это отражает два способа мышления об отношениях между экземплярами различных классов: в терминах иерархии "класс-подкласс" и в терминах общих протоколов сообщений.

Object instance protocol

class membership

class	Возвращает объект — класс получателя.
isKindOf: aClass	Определяет, является ли аргумент aClass суперклассом или классом получателя.
isMemberOf: aClass	Определяет, является ли получатель непосредственным экземпляром класса aClass. Это то же самое, что и проверка, является ли ответ получателя на сообщение class тем же (==) объектом, что и aClass.
respondsTo: aSymbol	Определяет, содержит ли словарь методов класса получателя или одного из его суперклассов аргумент aSymbol как имя сообщения.

Вот примеры посылки сообщений объектам и соответствующие ответы:

выражение	*результат*
3 class	SmallInteger
#(this is an array) isKindOf: Collection	true
#(this is an array) isMemberOf: Collection	false
#(this is an array) class	Array
3 respondsTo: #isKindOf:	true
#(1 2 3) isMemberOf: Array	true
Object class	Object class

Сравнение объектов

Так как вся информация в системе представляется в виде объектов, основной протокол поддерживает проверку идентичности объектов и копирование объектов. Важнейшие сообщения сравнения, определенные в классе Object — проверка на идентичность и проверка на равенство. Идентичность или тождественность (==) — проверка того, являются ли два объекта одним и тем же объектом. Равенство (=) — проверка того, представляют ли два объекта один и тот же компонент системы. Решение о том, что значит "представлять один и тот же компонент", принимает получатель сообщения; каждый новый класс, который добавляет новые переменные экземпляра, как правило, обязан переопределить сообщение =, чтобы установить, какие из переменных экземпляра должны войти в проверку на равенство. Например, равенство двух массивов определяется проверкой размеров массивов и, в случае их равенства, последовательной проверкой на равенство соответствующих элементов; равенство двух чисел определяется проверкой того, представляют ли эти числа одно и то же значение; проверка равенства двух банковских счетов может быть ограничена проверкой совпадения их идентификационных номеров.

Сообщение hash (хешировать[1]) — особая часть протокола сравнения. Ответ на hash — целое число. Любые два равных объекта должны возвращать в ответ на hash одинаковые значения. Неравные объекты могут возвращать равные или неравные значения в ответ на сообщение hash. Обычно это целое число используется как индекс, чтобы определить местонахождение объекта в индексированном наборе (как показано в главе 3). Всякий раз, когда переопределяется сообщение =, должно быть переопределено и сообщение hash, чтобы сохранить свойство равных объектов возвращать в ответ на hash равные значения.

Object instance protocol

comparing	
== anObject	Проверяет, являются ли получатель и аргумент одним и тем же объектом.
= anObject	Проверяет, представляют ли получатель и аргумент один и тот же компонент системы.
~= anObject	Проверяет, представляют ли получатель и аргумент разные компоненты системы.
~~ anObject	Проверяет, являются ли получатель и аргумент разными объектами системы.
hash	Возвращает целое число, вычисленное в соответствии с представлением получателя.

По умолчанию, реализация сообщения = такая же, как и сообщения ==.

Специализированный протокол обеспечивает простой способ сравнения объектов с объектом nil.

Object instance protocol

testing	
isNil	Проверяет, является ли получатель объектом nil.
notNil	Проверяет, отличен ли получатель от nil.
isInteger	Возвращает true, если получатель сообщения — целое число; иначе возвращает false.

Первые два сообщения тождественны соответственно == nil и ~~ nil. Выбор сообщения — дело вкуса.

Несколько очевидных примеров:

[1] Иногда этот термин переводится как "расстановка" или "перемешивание". — Примеч. ред. перев.

выражение	*результат*
nil isNil	true
true notNil	true
3 isNil	false
#(a b c) = #(a b c)	true
3 = (6 / 2)	true
#(1 2 3) class == Array	true

Копирование объектов

Существуют два способа копирования объектов. Различие состоит в том, копируются или нет значения переменных экземпляра. Если значения не копируются (сообщение shallowCopy — поверхностнаяКопия), то они будут общими (разделяемыми) для объекта и его копии; если значения переменных копируются (сообщение deepCopy — полнаяКопия), то они не будут общими.

Object instance protocol

copying

copy	Возвращает новый экземпляр объекта, такой же как получатель.
shallowCopy	Возвращает копию получателя, которая разделяет значения переменных экземпляра получателя.
deepCopy	Возвращает копию получателя с собственной копией каждой его переменной экземпляра.

Реализация сообщения copy по умолчанию — shallowCopy. В тех подклассах, в которых копирование должно давать в результате специальную комбинацию разделяемых и неразделяемых переменных, обычно переопределяется метод, связанный с сообщением copy, а не методы, связанные с сообщениями shallowCopy и deepCopy.

Например, (поверхностная) копия экземпляра класса Array ссылается на те же элементы, что и оригинал экземпляра класса Array, но копия — это другой объект. Замена элемента в копии не изменяет оригинала. Таким образом:

выражение	*результат*
a ← #('first' 'second' 'third')	('first' 'second' 'third')
b ← a copy	('first' 'second' 'third')
a = b	true
a == b	false
(a at: 1) == (b at: 1)	true
b at: 1 put: 'newFirst'	'newFirst'
a = b	false
a ← 'hello'	'hello'
b ← a copy	'hello'
a = b	true
a == b	false

Рис. 6.1 показывает отношение между поверхностным и полным копированием. Для дальнейшей иллюстрации различия между shallowCopy и deepCopy возьмем в качестве примера класс PersonnelRecord (ПерсональнаяКарта). Предположим, что он включает в себя переменную insurancePlan (планСтрахования), экземпляр класса Insurance (Страхование). Далее предположим, что каждый экземпляр класса Insurance имеет значение, связанное с предельным значением суммы медицинской страховки. Теперь предположим, что мы создали объект EmployeeRecord (КартаСлужащего), как макетный экземпляр класса PersonnelRecord. Под "макетным" мы понимаем такой экземпляр, который имеет все первоначальные атрибуты нового экземпляра этого класса, так что новые экземпляры могут создаваться простым копированием макетного экземпляра, вместо посылки последовательности инициализационных сообщений. Предположим далее, что этот макетный экземпляр — переменная класса PersonnelRecord и что в ответ на создание нового экземпляра класса PersonnelRecord делается ее поверхностная копия; то есть, метод, связанный с сообщением new — это выражение ↑EmployeeRecord copy.

Рис. 6.1.

В результате выполнения выражения

```
joeSmithRecord ← PersonnelRecord new
```

объект joeSmithRecord будет ссылаться на копию (в данном случае, поверхностную) объекта EmployeeRecord.

Макетный экземпляр EmployeeRecord и реальная запись joeSmithRecord имеют общую ссылку на один план страхования. Страховая политика компании может изменяться. Предположим, класс PersonnelRecord понимает сообщение changeInsuranceLimit: aNumber (изменитьПределСтрахования: число), которое реализуется посредством переустановки имеющимся макетным экземпляром класса PersonnelRecord, объектом EmployeeRecord, предельной планируемой суммы медицинской страховки. Так как план страхования общий, то в результате выполнения выражения

```
PersonnelRecord changeInsuranceLimit: 4000
```

изменится медицинская страховка для всех служащих. В примере изменяется медицинская страховка, на которую ссылаются как объект EmployeeRecord, так и его копия, объект joeSmithRecord. Сообщение changeInsuranceLimit: посылается классу PersonnelRecord, поскольку класс — самый подходящий объект для передачи сообщения об изменении всем своим экземплярам.

Доступ к частям объекта

В системе Smalltalk-80 существует два типа объектов: объекты с именованными переменными и объекты с индексированными переменными. Объекты с индексированными переменными могут также иметь и именованные переменные экземпляра. Это различие объясняется в главе 3. Класс Object содержит шесть сообщений, предназначенных для доступа к индексированным переменным объекта.

Object instance protocol

accessing

at: index	Возвращает значение индексированной переменной получателя, с индексом равным аргументу index. Если получатель не имеет индексированных переменных, или если аргумент больше, чем число индексированных переменных, то выдает сообщение об ошибке.

at: index put: anObject	Сохраняет аргумент anObject как значение индексированной переменной получателя, индекс которой задается аргументом index. Если получатель не имеет индексированных переменных, или если аргумент index больше, чем число индексированных переменных, то выдает сообщение об ошибке. Возвращает аргумент anObject.
basicAt: index	Аналогично сообщению at: index. Однако метод, связанный с этим сообщением, не может быть изменен ни в одном из подклассов.
basicAt: index put: anObject	Аналогично сообщению at: index put: anObject. Однако метод, связанный с этим сообщением, не может быть изменен ни в одном из подклассов.
size	Возвращает число индексированных переменных получателя. Это величина равна наибольшему допустимому индексу.
basicSize	Аналогично сообщению size. Однако метод, связанный с этими сообщением, не может быть переопределен.
yourself	Возвращает получателя сообщения.

Заметим, что сообщения доступа присутствуют в протоколе парами; одно сообщение в каждой паре имеет приставку basic, обозначающую, что это фундаментальное системное сообщение и его реализация не может быть изменена ни в одном подклассе. Назначение данных пар в том, что внешний протокол сообщений at:, at:put: и size может быть переопределен в подклассах с целью обработки специальных случаев, но при этом сохранится возможность воспользоваться примитивными методами. (Глава 4 содержит объяснение “примитивных” методов, реализуемых в виртуальной машине системы.) Таким образом, в любом методе иерархии классов системы сообщения basicAt:, basicAt:put: и basicSize предоставляют доступ к примитивным реализациям. Сообщение basicSize может быть послано любому объекту; если объект не имеет индексированных переменных, то ответом будет 0.

Экземпляры класса Array — объекты различной длины. Пусть letters — массив #(a b d f j m p s). Тогда:

выражение	*результат*
letters size	8
letters at: 3	d
letters at: 3 put: #c	c
letters	(a b c f j m p s)

Печать и сохранение объектов

Существуют различные способы создавать последовательности символов для описания объекта. Описание могло бы давать только ключ к идентификации объекта. Но оно могло бы давать достаточно информации для того, чтобы сконструировать подобный объект. В первом случае (printing — печать) описание может быть или не быть хорошо форматированным и визуально понятным, как это, например, обеспечивается структурной распечаткой программ Лиспа. Во втором случае (storing — хранение), описание могло бы сохранять информацию, которой затем можно воспользоваться для восстановления объекта.

Протокол сообщений классов в системе Smalltalk-80 обеспечивает и печать, и сохранение. Реализация этих сообщений в классе Object предоставляет минимальные возможности; большинство подклассов переопределяют эти сообщения для того, чтобы улучшить создаваемые описания. Аргументами двух сообщений служат экземпляры класса Stream (Поток). Этот класс и его подклассы описываются в главе 12.

printing	
printString	Возвращает строку, содержащую описание получателя.
printOn: aStream	Добавляет в поток aStream строку, содержащую описание получателя.
storing	
storeString	Возвращает строку, представляющую получателя, из которой он может быть восстановлен.
storeOn: aStream	Добавляет в поток aStream строку, представляющую получателя, из которой он может быть восстановлен.
readFromString: aString	Создает объект, основанный на содержимом строки aString.
isLiteral	Возвращает ответ на вопрос, имеет ли получатель литеральную текстовую форму, распознаваемую компилятором.

Каждый из двух видов печати основывается на выдаче некоторой последовательности символов, которую можно либо увидеть на экране дисплея, либо записать в файл, либо переслать по сети. Последовательность, созданную сообщениями storeString или storeOn:, можно рассматривать как одно или более выражений, которые следует выполнить для того, чтобы восстановить исходный объект. Например, экземпляр класса Set (Множество), состоящий из трех элементов $a, $b, $c, может быть напечатан в виде

```
Set($a $b $c)
```

в то же время, сохранен он будет в виде

```
(Set new add: $a; add: $b; add: $c)
```

Литералы могут использовать одно и то же представление и для печати, и для хранения. Строка 'hello', будет напечатана и сохранена как 'hello'. Имя #name печатается как name, а сохраняется как #name.

При отсутствии дополнительной информации, выполнение по умолчанию сообщения printString возвращает имя класса получателя, а выполнение по умолчанию сообщения storeString возвращает имя класса, сопровождаемое сообщением basicNew о создании экземпляра, за которым следует последовательность сообщений о сохранении каждой переменной экземпляра. Например, если подкласс класса Object, класс Example, демонстрирует поведение по умолчанию, то для объекта eg класса Example без переменных экземпляра мы имели бы:

выражение	*результат*
eg printString	'an Example'
eg storeString	'(Example basicNew)'

Класс Object отзывается на сообщение readFrom: aStream тем, что создает объект на основе содержимого потока aStream. Это сообщение переопределяется большинством классов для учета специфики создания объектов.

Обработка ошибок

Тот факт, что все действия в системе происходят как реакция на посылку сообщения объекту, означает, что система должна обрабатывать одно основное условие возникновения ошибки: сообщение посылается объекту, но оно не определено ни в одном из классов в цепочке суперклассов объекта. Такая ошибка определяется интерпретатором системы, который в этом случае посылает первоначальному объекту сообщение doesNotUnderstand:

aMessage (неПонимает: сообщение). Аргумент aMessage, представляет собой имя невыполненного сообщения и его аргументы, если они есть. Метод, связанный с именем doesNotUnderstand:, сообщает программисту, что произошла ошибка. О том, в каком виде сообщение об ошибке представляется пользователю, заботится существующий в системе (графический) интерфейс пользователя, который здесь не рассматривается; минимальные требования к системе взаимодействия с пользователем состоят в том, что сначала сообщение об ошибке отображается на устройстве вывода, а затем пользователю должна предоставляться возможность исправить ошибку. Глава 17 иллюстрирует механизмы извещения об ошибках и отладки в системе Smalltalk-80.

В дополнение к упомянутой основной ошибке, методы могут при необходимости явно запустить системный механизм обработки ошибок в тех случаях, когда система может определить, что программа пользователя собирается делать что-то недопустимое. В таких случаях метод может дать пояснение к происшедшему, которое следовало бы представить программисту. Типичное действие — послать активному объекту сообщение error: aString, где аргумент представляет собой предлагаемое пояснение. По умолчанию реализуется существующий в системе механизм извещения. Программист может предложить альтернативную реализацию сообщения error:, которая будет использовать механизм извещения об ошибках, ориентированный на приложение.

Протоколом класса Object поддерживаются общие сообщения об ошибках. Сообщение об ошибке может указывать на ошибку примитивного метода, или на то, что подкласс отменил наследуемое сообщение, которое он не может поддерживать и, следовательно, пользователь не имеет права его вызывать; или на то, что суперкласс определяет сообщение, которое должно быть реализовано в подклассе.

Object instance protocol

error handling

doesNotUnderstand: aMessage	Сообщает пользователю, что получатель не понимает аргумент aMessage как сообщение.
error: aString	Сообщает пользователю, что произошла ошибка в контексте ответа на сообщение получателю; пользователю выдается строка aString как часть пояснения к ошибке.
primitiveFailed	Сообщает пользователю, что не смог правильно выполнится метод, реализованный как системный примитив.
shouldNotImplement	Сообщает пользователю, что хотя суперкласс получателя и определяет, что сообщение будет реализовано подклассом, класс получателя не может обеспечить соответствующей реализации.
subclassResponsibility	Сообщает пользователю, что метод, определенный в суперклассе получателя, должен быть реализован в классе получателя.

Подкласс может переопределять сообщения обработки ошибок для того, чтобы обеспечить специальную поддержку для исправления ошибочной ситуации. Глава 13, в которой рассматривается реализация классов-наборов, демонстрирует примеры использования последних двух сообщений.

Некоторые сообщения обработки ошибок характерны для интерфейса пользователя. Это confirm:, halt:, halt и notify:. В качестве отладчиков они создают окна извещения об ошибке, помогая отладке определения класса (см. главу 17).

Остальные сообщения из протокола класса Object приведены в главе 14.