# 12
# Протокол класса Stream

## Класс Stream

## Позиционируемые потоки

*Класс* ReadStream

*Класс* WriteStream

*Класс* ReadWriteStream

## Потоки генерируемых элементов

## Потоки для наборов без внешних ключей

## Внешние потоки и файловые потоки

Конфиденциально

Классы наборов обеспечивают систему Smalltalk-80 основными структурами данных для совместного хранения объектов, как в линейно упорядоченных, так и в неупорядоченных группах. Протокол этих классов обеспечивает прямой доступ (сохранение и выборку) к индивидуальным элементам наборов. Также, через сообщения перечисления, поддерживается непрерываемый доступ ко всем элементам по порядку. Смешивание двух видов операций доступа — перечисления и сохранения — однако, не поддерживается. Также не поддерживается протоколом классов наборов обращение к индивидуальным элементам, по одному за обращение, если только не поддерживается отдельно внешняя ссылка на позицию элемента набора.

Если нельзя легко вычислить внешнее имя для каждого элемента набора, то непрерывное перечисление индивидуальных элементов не может выполняться эффективно. Можно, например, последовательно читать элементы из экземпляра класса OrderedCollection, используя комбинации сообщений first и after:, до тех пор, пока элементы экземпляра уникальны. Другой возможный подход при перечислении подразумевает, что набор некоторым образом сам помнит, какой элемент был доступен последним. Мы называем это запоминаемое положение “указателем позиции” или “позиционным указателем”. Возможность общего доступа к последовательности элементов, однако, означает что необходимо поддерживать отдельную, внешнюю память для последнего доступного элемента.

Класс Stream предоставляет возможность поддерживать позиционный указатель на элементы из набора объектов. Мы используем фразу *“поток над набором* (или *на наборе)”*, понимая под этим доступность элементов набора таким образом, что можно перечислять или сохранять каждый элемент, один за обращение, возможно, смешивая эти операции. Создавая несколько потоков над одним набором, можно поддерживать несколько позиционных указателей на одном и том же наборе.

Есть несколько способов поддерживать позиционный указатель для потока над набором. Общий подход состоит в употреблении целого числа как индекса. Этот подход может применяться для любого набора, чьи элементы имеют внешний индекс. Класс SequenceableCollection и его подклассы относятся к этой категории и, как будет сказано ниже, такие потоки представляются в системе Smalltalk-80 экземплярами класса PositionableStream (ПозиционируемыйПоток). Второй способ поддерживать позиционный указатель состоит в том, чтобы использовать некоторый *источник* для генератора объектов. Пример этого вида потока в Smalltalk-80 — класс Random, который уже рассматривался в главе 8. И третий способ состоит в том, чтобы поддерживать нечисловой позиционный указатель, типа ссылки на узел в некоторой последовательности; такой подход иллюстрируется в этой главе примером класса, который поддерживает поток над списком связей или структурой дерева.

---

## Класс Stream

Класс Stream (Поток) — подкласс класса Object, является суперклассом, который определяет протокол доступа для потока над набором. В этот протокол включаются и сообщения для чтения (выборки) и записи (сохранения) элементов в наборе, хотя не все подклассы класса Stream могут поддерживать эти два вида операций.

Основное сообщение чтения — next; ответ на него — следующий элемент в наборе, на который ссылается поток, экземпляр класса Stream. Обладая возможностью обращаться к следующему элементу, можно уже создавать более общие сообщения для чтения, например, такие, как next: anInteger, которое возвращает набор из следующих (в количестве anInteger штук) элементов; nextMatchFor: anObject, которое читает следующий элемент и определяет равен ли он аргументу anObject; contents, которое возвращает набор всех элементов.

Конфиденциально

Stream instance protocol

accessing-reading

| | |
|---|---|
| next | Возвращает следующий элемент, доступный из получателя. |
| next: anInteger | Возвращает следующие anInteger доступных элементов из получателя. Обычно это набор того же класса, на котором организует доступ получатель. |
| nextMatchFor: anObject | Читает следующий элемент и определяет, равен ли он аргументу anObject. |
| contents | Возвращает все элементы набора, доступного получателю. Обычно это набор того же класса, на котором организует доступ получатель. |

Основное сообщение записи — nextPut: anObject; оно означает требование сохранить аргумент anObject, как следующий элемент, доступный получателю. Если в потоке возможны сообщения и чтения, и записи, то сообщение next, посланное потоку сразу после сообщения nextPut: anElement прочтет не только что сохраненный элемент, а следующий после него элемент из набора. К сообщениям записи также относятся сообщение nextPutAll: aCollection, которое сохраняет все элементы из аргумента aCollection в наборе, доступном получателю, и сообщение next: anInteger put: anObject, которое сохраняет аргумент anObject, как следующий элемент набора anInteger раз.

Stream instance protocol

accessing-writing

| | |
|---|---|
| nextPut: anObject | Сохраняет аргумент anObject, как следующий элемент, доступный получателю. Возвращает anObject. |
| nextPutAll: aCollection | Сохраняет элементы аргумента aCollection, как следующие элементы, доступные получателю. Возвращает aCollection. |
| next: anInteger put: anObject | Сохраняет аргумент anObject, как следующий элемент в получателе anInteger раз. Возвращает anObject. |

Сообщения чтения и записи определяют, может ли следующий элемент быть прочитан или записан и, если нет, извещают об ошибке. В связи с этим программист может пожелать определить, возможен ли еще доступ к потоку; это выполняется посылкой потоку сообщение atEnd.

Stream instance protocol

testing

| | |
|---|---|
| atEnd | Проверяет, нет ли в получателе доступных объектов. |

Непрерывное чтение элементов, которые применяются как аргументы блока, может быть выполнено с помощью сообщения do: aBlock, подобно тому, как это происходило в классах наборов.

Stream instance protocol

enumerating

| | |
|---|---|
| do: aBlock | Выполняет аргумент aBlock для каждого оставшегося доступного элемента получателя. |

Реализация этого сообщения основывается на посылке получателю сообщений atEnd и next. Как пример их использования мы приводим исходный текст метода:

```
do: aBlock
    [self atEnd] whileFalse: [aBlock value: self next]
```

Каждый вид потоков должен определять свои сообщения создания экземпляра. Отметим, что поток не может создаваться просто посылкой классу сообщения new, потому что поток

Конфиденциально

должен быть проинформирован о том, какой набор ему доступен и каков начальный позиционный указатель.

В качестве простого примера давайте предположим, что доступный потоку набор — массив, а имя потока — accessor. Пусть элементами массива будут имена:

Bob Dave Earl Frank Harold Jim Kim Mike Peter Rick Sam Tom.

Пусть, наконец, позиционный указатель таков, что Bob — следующий доступный элемент массива. Тогда

| *выражение* | *результат* |
|---|---|
| accessor next | Bob |
| accessor next | Dave |
| accessor nextMatchFor: #Bob | false |
| accessor nextMatchFor: #Frank | true |
| accessor next | Harold |
| accessor nextPut: #James | James |
| accessor contents | (Bob Dave Earl Frank Harold James Kim Mike Peter Rick Sam Tom) |
| accessor nextPutAll: #(Karl Larry Paul) | (Karl Larry Paul) |
| accessor contents | (Bob Dave Earl Frank Harold James Karl Larry Paul Rick Sam Tom) |
| accessor next: 2 put: #John | John |
| accessor contents | (Bob Dave Earl Frank Harold James Karl Larry Paul John John Tom) |
| accessor next | Tom |
| accessor atEnd | true |

## Позиционируемые потоки

Пожалуйста, обратите внимание, что в тех версиях системы Smalltalk-80, где реализованы внешние потоки, может существовать новая иерархия, в которой есть отличные друг от друга классы ExternalStream (ВнешнийПоток) и InternalStream (ВнутреннийПоток). Во введении к этой главе мы указали на три возможных способа, которыми потоки могли бы поддерживать позиционный указатель. Первый путь — использование индекса, увеличиваемого каждый раз, когда поток обращается к элементам. Такому потоку доступны только те наборы, чьи элементы имеют целые числа (индексы) в качестве внешних ключей; к таким наборам относятся все подклассы класса SequenceableCollection.

Класс PositionableStream (ПозиционируемыйПоток) — подкласс класса Stream, который обеспечивает дополнительный протокол для потоков, которые могут переустанавливать позиционный указатель, но это — абстрактный класс, потому что он не обеспечивает реализацию наследуемых сообщений next и nextPut: anObject. Реализация этих сообщений оставлена для подклассов класса PositionableStream: ReadStream, WriteStream и ReadWriteStream.

Экземпляр класса PositionableStream создается посредством посылки классу либо сообщения on: aCollection, либо сообщения on: aCollection from: firstIndex to: lastIndex. В первом сообщении аргумент aCollection является набором, доступным созданному потоку;

Конфиденциально

во втором сообщении, потоку доступна копия части набора aCollection от индекса firstIndex до индекса lastIndex.

PositionableStream class protocol

instance creation

| | |
|---|---|
| on: aCollection | Возвращает экземпляр вида PositionableStream с доступным потоку набором aCollection. |
| on: aCollection from: firstIndex to: lastIndex | Возвращает экземпляр вида PositionabieStream с доступным потоку набором, представляющим копию части аргумента aCollection от индекса firstIndex до индекса lastIndex. |

Класс PositionableStream поддерживает дополнительный протокол для доступа к элементам набора и тестирования содержимого набора. Есть возможность устанавливать позиционный указатель на заданный элемент потока, пропуская элемент или группу элементов. Ниже — некоторые примеры из протокола этого класса.

PositionableStream instance protocol

testing

| | |
|---|---|
| isEmpty | Возвращает true, если набор, доступный получателю, не имеет элементов, иначе возвращает false. |

accessing

| | |
|---|---|
| peek | Возвращает следующий элемент набора (как и сообщение next), но не изменяет позиционный указатель. Возвращает nil, если позиционный указатель в конце потока. |
| peekFor: anObject | Определяет ответ на сообщение peek. Если следующий элемент равен anObject, тогда позиционный указатель увеличивается и возвращается true. Иначе возвращается false и позиционный указатель не меняется. |
| upTo: anObject | Возвращает поднабор доступных получателю элементов, начиная со следующего доступного элемента и до, но не включая, элемента равного anObject. Если anObject отсутствует в наборе, то до конца набора. |

Поскольку известно, как экземпляр класса PositionableStream хранит определенный им позиционный указатель, протокол класса обеспечивает доступ к нему. В частности, возможна установка позиционного указателя в начало, в конец или в любую другую позицию доступного набора.

PositionableStream instance protocol

positioning

| | |
|---|---|
| position | Возвращает текущий позиционный указатель получателя. |
| position: anInteger | Устанавливает текущий позиционный указатель получателя на позицию anInteger. Если аргумент выходит за границы набора получателя, сообщает об ошибке. |
| reset | Устанавливает позиционный указатель получателя в начало набора. |
| setToEnd | Устанавливает позиционный указатель получателя в конец набора. |
| skip: anInteger | Устанавливает позиционный указатель получателя в позицию равную сумме текущей позиции и аргумента anInteger, подгоняя результат так, чтобы оставаться в границах набора. |
| skipThrough: anObject | Устанавливает позиционный указатель получателя после следующего вхождения аргумента anObject в набор. Возвращает true, если такое вхождение существует. |

Конфиденциально

### *Класс* ReadStream

Класс ReadStream (ПотокЧтения) — конкретный подкласс класса PositionableStream, который предоставляет доступ к элементам набора только для чтения. Мы приведем пример, подобный предыдущему, чтобы продемонстрировать использование дополнительного протокола класса PositionableStream, наследуемого всеми экземплярами класса ReadStream. Ни одно из сообщений с именами nextPut:, next:put:, nextPutAll: нет смысла посылать экземплярам класса ReadStream, поскольку это вызовет только сообщение об ошибке. Итак, пусть

```
accessor ←
    ReadStream on: #(Bob Dave Earl Frank Harold Jim Kim Mike
                Peter Rick Sam Tom).
```

| *выражение* | *результат* |
|---|---|
| accessor next | Bob |
| accessor nextMatchFor: #Dave | true |
| accessor peek | Earl |
| accessor next | Earl |
| accessor peekFor: #Frank | true |
| accessor next | Harold |
| accessor upTo: #Rick | (Jim Kim Mike Peter) |
| accessor position | 10 |
| accessor skip: 1 | сам набор accessor |
| accessor next | Tom |
| accessor atEnd | true |
| accessor reset | сам набор accessor |
| accessor skipThrough: #Frank | сам набор accessor |
| accessor next | Harold |

### *Класс* WriteStream

Класс WriteStream (ПотокЗаписи) — подкласс PositionableStream, предоставляющий доступ только для записи элементов в набор. Ни одно из сообщений с именами next, next:, do: нет смысла посылать экземплярам класса WriteStream, поскольку это вызовет только сообщение об ошибке.

Экземпляры класса WriteStream используются системой Smalltalk-80 как составная часть методов для печати или хранения строк описания любого объекта. Каждый объект в системе может отвечать на сообщения printOn: aStream и storeOn: aStream. Методы, реализующие эти сообщения, состоят из последовательности сообщений к аргументу, который является экземпляром одного из подклассов класса Stream и позволяет записывать элементы в доступный ему поток. Среди них сообщение с именем nextPut: и аргументом-символом, и сообщение с именем nextPutAll: и аргументом — строкой или именем. Ниже мы проиллюстрируем эту идею на примере.

Протокол печати класса Object, как описано в главе 6, включает сообщение printString. Реализация этого сообщения такова

```
printString
    | aStream |
    aStream ← WriteStream on: (String new: 16).
    self printOn: aStream.
    ↑aStream contents
```

Конфиденциально

Если набору послать сообщение printString, то ответом на него будет строка, которая и будет описанием экземпляра. Метод вначале создаст экземпляр класса WriteStream, который может сохранять набор в себе, затем пошлет сообщение printOn: набору и, наконец, возвратит содержимое окончательно получившегося экземпляра класса WriteStream. Сообщение storeString, также понимаемое любым объектом системы, реализовано в классе Object аналогичным образом. Отличие от printString состоит только в том, что второе выражение соответствующего метода содержит сообщения storeOn: aStream вместо сообщения printOn: aStream.

Общее правило, по которому наборы печатают свою строку-описание, состоит в том, чтобы сначала напечатать имя класса, которому принадлежит набор, затем левую круглую скобку, затем строку-описание каждого элемента набора, отделяя описания друг от друга пробелами, и завершить все это правой круглой скобкой. Так, если набор является множеством и содержит четыре элемента-имени #one, #two, #three, #four, то этот набор будет напечатан на поток в виде

```
Set(one two three four)
```

Упорядоченный набор с теми же самыми элементами будет напечатан на поток в виде

```
OrderedCollection(one two three four)
```

и так далее.

Напомним, что определение сообщений с именами printOn: и storeOn: дано в главе 6, при этом любая подходящая строка-описание может быть результатом выполнения сообщения printOn:, но строка-описание, создаваемая сообщением storeOn:, должна представлять правильно построенное выражение, выполнение которого позволяло бы восстановить описываемый объект. Далее приводится реализация в классе Collection сообщения печати.

```
printOn: aStream
    aStream nextPutAll: self class name.
    aStream space.
    aStream nextPut: $(.
    self do: [:element |
            element printOn: aStream.
            aStream space].
    aStream nextPut: $)
```

Обратите внимание, что в теле этого метода экземпляру aStream класса WriteStream посылается сообщение space. Это и ряд других сообщений определяются в классе WriteStream, чтобы предоставить краткие выражения для сохранения разделителей в таких потоках. Вот они.

WriteStream instance protocol

character writing

| | |
|---|---|
| cr | Сохраняет символ возврата каретки как следующий элемент получателя. |
| crtab | Сохраняет символ возврата каретки и один символ табуляции как следующие два элемента получателя. |
| crtab: anInteger | Сохраняет символ возврата каретки и anInteger символов табуляции как следующие элементы получателя. |
| space | Сохраняет пробел как следующий элемент получателя. |
| tab | Сохраняет символ табуляции как следующий элемент получателя. |

Таким образом, чтобы создавать строку вида

```
'name    city

bob      New York
joe      Chicago
```

Конфиденциально

```
bill        Rochester
'
```

для двух соответствующих массивов

```
names ← #(bob joe bill).
cities ← #('New York'  'Chicago'  'Rochester').
```

выполним выражения

```
aStream ← WriteStream on: (String new: 16).
aStream nextPutAll: 'name'.
aStream tab.
aStream nextPutAll: 'city'.
aStream cr; cr.
names with: cities do:
      [:name :city |
       aStream nextPutAll: name.
       aStream tab.
       aStream nextPutAll: city.
       aStream cr].
```

Теперь желаемый результат можно получить, выполнив выражение aStream contents.

Предположим, что существует некоторый набор и мы хотим добавить в него новую информацию, используя для этого потоки. Класс WriteStream подходит для этих целей, поскольку он поддерживает протокол создания экземпляра, который принимает набор и устанавливает позиционный указатель записи в конец набора.

WriteStream class protocol

instance creation

with: aCollection

Возвращает экземпляр класса WriteStream с доступным набором aCollection и устанавливает позиционный указатель для записи следующего элемента в конец набора.

with: aCollection from: firstIndex to: lastIndex

Возвращает экземпляр класса WriteStream с доступным набором, представляющим копию части набора aCollection от индекса firstIndex до индекса lastIndex и устанавливает позиционный указатель записи следующего элемента в конец получившегося поднабора.

Таким образом, если строка с именем header уже существует и имеет вид

```
'name      city
'
```

то строка, созданная в предыдущем примере, может быть создана при выполнении следующих выражений:

```
aStream <- WriteStream with: header.
names with: cities do:
      [:name :city |
       aStream nextPutAll: name.
       aStream tab.
       aStream nextPutAll: city.
       aStream cr].
aStream contents.
```

## *Класс* ReadWriteStream

Класс ReadWriteStream (ПотокЧтенияЗаписи) — подкласс WriteStream, который обеспечивает как чтение, так и запись в набор. Он поддерживает протоколы классов ReadStream и WriteStream в том виде, в каком они были приведены ранее. С

Конфиденциально

дополнительными возможностями протокола класса ReadWriteStream, как и с протоколами всех других классов системы, можно познакомиться с помощью встроенного окна просмотра иерархии классов.

## Потоки генерируемых элементов

Из трех способов создания позиционного указателя для потока над набором, второй способ, упомянутый во введении этой главы, состоит в том, чтобы определять источник, из которого генерируются следующие элементы набора. Этот вид потока разрешает только чтение элементов из набора, но не запись. Позиционный указатель, однако, может быть переустановлен заданием нового источника.

Класс Random, описанный в главе 8, — подкласс класса Stream, который определяет свои элементы на основе алгоритма, использующего в качестве источника некоторое начальное число. Класс Random обеспечивает свою реализацию сообщений next и atEnd. Поскольку число элементов набора теоретически бесконечно, процесс образования новых элементов никогда не кончается. Более того, экземпляры класса Random не отвечают на сообщение contents. Класс Random поддерживает сообщение do: aBlock, но связанный с ним метод никогда не закончится без целенаправленного вмешательства программиста.

Далее приведена реализация класса Random; читатель может обратиться к главам 11 и 21 за примерами использования экземпляров этого класса. Реализации сообщений do: aBlock и nextMatchFor: anObject наследуются из класса Stream.

```
class name                      Random
superclass                      Stream
instance variable names         seed

class methods

instance creation
    new
        ↑self basicNew setSeed

instance methods

testing
    atEnd
        ↑false

accessing
    next
        | temp |
        "Линейный конгруэнтный метод Лемера с модулем m = 2 в степени 16;
        a = 27181 - нечетное число, 5 = a \\ 8; c = 13849 - нечетное число,
        c/m равно приблизительно 0.21132"
        [seed ←13849 + (27181 * seed) bitAnd: 8r177777.
            temp ← seed / 65536.0.
            temp = 0] whileTrue.
        ↑temp

private
    setSeed
        "Для определения псевдослучайного начального значения источника, взять
        время из системных часов компьютера. Это большое положительное целое
```

Конфиденциально

```
        число, поэтому использовать только младшие 16 битов."
        seed ← Time millisecondClockValue bitAnd: 8rl77777
```

Другой возможный пример потока генерируемых элементов — вероятностные распределения, описываемые в главе 21. Суперкласс ProbabilityDistribution (ВероятностноеРаспределение) реализуется как подкласс класса Stream. Сообщение next: anInteger наследуется из класса Stream. Каждый подкласс класса ProbabilityDistribution определяет являются ли его экземпляры "только для чтения" и, если это так, реализуют сообщение с именем nextPut: как self shouldNotImplement. Класс SampleSpace, другой пример из главы 21, сохраняет набор данных и реализует сообщение nextPut: anObject как добавление новых данных к набору.

---

## Потоки для наборов без внешних ключей

Третий способ поддержки позиционного указателя для потока над набором, о котором упоминалось во введении к этой главе, состоит в том, чтобы поддерживать нечисловой указатель. Это может быть полезно в случаях, когда элементы набора недоступны по внешним целочисленным ключами или такой доступ не является самым эффективным способом поиска элемента.

Поток над экземпляром класса LinkedList дает пример, в котором элементы набора доступны по внешним ключам-индексам, но каждый такой доступ требует поиска по цепочке связанных элементов. В этом случае более эффективно держать указатель на текущий элемент набора (экземпляр вида Link), а следующий элемент получать, посылая текущему элементу сообщение nextLink. Таким потоком обеспечиваются и чтение, и запись в экземпляр класса LinkedList.

Предположим, что мы создаем подкласс класса Stream с именем LinkedListStream (ПотокСвязанногоСписка). Каждый экземпляр содержит указатель на экземпляр класса LinkedList и позиционный указатель на элемент в наборе. Поскольку поддерживается и чтение и запись, в классе должны быть реализованы сообщения с именами next, nextPut:, atEnd, contents. (Обратите внимание, что эти четыре сообщения в классе Stream определяются как self subclassResponsibility). Новый экземпляр класса LinkedListStream создается посылкой этому классу сообщение on: aLinkedList.

```
class name                  LinkedListStream
superclass                  Stream
instance variable names     collection
                            currentNode

class methods
instance creation
    on: aLinkedList
        ↑self basicNew setOn: aLinkedList

instance methods
testing
    atEnd
        ↑currentNode isNil

accessing
    next
        | saveCurrentNode |
        saveCurrentNode ← currentNode.
        self atEnd
```

Конфиденциально

```
                ifFalse: [currentNode ← currentNode nextLink].
        ↑saveCurrentNode

    nextPut: aLink
        | index previousLink |
        self atEnd ifTrue: [↑collection addLast: aLink].
        index ← collection indexOf: currentNode.
        index = 1
            ifTrue: [collection addFirst: aLink]
            ifFalse: [previousLink ← collection at: index - 1.
                      previousLink nextLink: aLink].
        aLink nextLink: currentNode nextLink.
        currentNode ← aLink nextLink.
        ↑aLink

    contents
        ↑collection

private
    setOn: aLinkedList
        collection ← aLinkedList.
        currentNode ← aLinkedList first
```

Чтобы показать применение этого нового вида потока, создадим экземпляр класса LinkedList из узлов — экземпляров класса WordLink; класс WordLink — подкласс класса Link, экземпляр которого содержит строку или имя.

```
class name                  WordLink
superclass                  Link
instance variable names     word

class methods

instance creation
    for: aString
        ↑self new word: aString

instance methods

accessing
    word
        ↑word

    word: aString
        word ← aString

comparing
    = aWordLink
        ↑word = aWordLink word

printing
    printOn: aStream
        aStream nextPutAll: 'a WordLink for'.
        aStream nextPutAll: word
```

Из вышеизложенного видим, что экземпляр класса WordLink для слова #one создается выражением

```
WordLink for: #one
```

Конфиденциально

Его печатная строка имеет вид

```
'a WordLink for one'
```

Теперь можно создавать экземпляр класса LinkedList для экземпляров WordLink и затем определить экземпляр класса LinkedListStream, которому доступен созданный экземпляр класса LinkedList.

```
list <- LinkedList new.
list add: (WordLink for: #one).
list add: (WordLink for: #two).
list add: (WordLink for: #three).
list add: (WordLink for: #four).
list add: (WordLink for: #five).
accessor <- LinkedListStream on: list
```

Приведем пример последовательности сообщений к объекту accessor.

| *выражение* | *результат* |
|---|---|
| accessor next | a WordLink for one |
| accessor next | a WordLink for two |
| accessor nextMatchFor:<br>(WordLink for: #three) | true |
| accessor nextPut:<br>(WordLink for: #insert) | a WordLink for insert |
| accessor contents | LinkedList<br>(a WordLink for one<br>a WordLink for two<br>a WordLink for three<br>a WordLink for insert<br>a WordLink for five) |
| accessor next | a WordLink for five |
| accessor atEnd | true |

Аналогично, прохождение узлов древовидной структуры, такой как описанный в главе 11 класс Tree, может быть выполнено с помощью некоторого потока, который поддерживает указатель на текущий узел и получает доступ к следующему элементу посредством обращения к левому дереву текущего узла, корню и правому дереву. Этот вид потока несколько более сложен в реализации по сравнению с потоком над экземпляром класса LinkedList, потому что необходимо сохранять информацию относительно того, левое или правое поддерево было пройдено, а также помнить обратную ссылку на "родительский" узел текущего узла. Порядок обхода древовидной структуры может быть реализован в потоке независимо от метода, которым поддеревья добавлялись в структуру. Таким образом, хотя мы применяли прямой порядок обхода в реализациях классов Tree и Node, мы можем обеспечить поток над деревом с обходом и в обратном порядке, соответствующим образом реализуя сообщения next и nextPut:.

## Внешние потоки и файловые потоки

Потоки, которые мы исследовали до сих пор, предполагали, что элементами доступного потоку набора могут быть любые объекты, независимо от их представления. Однако, для связи с устройствами ввода-вывода, такими, как диск, это предположение недопустимо. В этих случаях элементы хранятся в двоичном виде и их размер вычисляется в байтах.

Конфиденциально

Элементы здесь могут рассматриваться как числа, строки, слова (из двух байтов) или байты. Таким образом, возникает необходимость поддерживать смесь неоднородных сообщений для чтения и записи этих разно размерных “кусков” информации.

Класс ExternalStream (ВнешнийПоток) — подкласс класса ReadWriteStream. Его назначение в том, чтобы добавить протокол доступа к неоднородным объектам. Кроме протокола доступа он включает также и протокол для позиционирования.

ExternalStream instance protocol

| | |
|---|---|
| **nonhomogeneous accessing** | |
| nextNumber: n | Возвращает следующие n байтов из набора, доступного получателю, рассматриваемые как положительный экземпляр класса SmallInteger или класса LargePositiveInteger. |
| nextNumber: n put: v | Сохраняет аргумент v, положительный экземпляр класса SmallInteger или класса LargePositiveInteger, как следующие n байтов из набора, доступного получателю, которые в случае необходимости дополняются нулями. |
| nextString | Возвращает строку, составленную из всех следующих элементов набора, доступного получателю. |
| nextStringPut: aString | Сохраняет аргумент aString в наборе, доступном получателю. |
| nextWord | Возвращает следующие два байта из набора, доступного получателю, рассматриваемые как экземпляр класса Integer. |
| nextWordPut: anInteger | Сохраняет аргумент anInteger как следующие два байта в наборе, доступном получателю. |
| **nonhomogeneous positioning** | |
| padTo: bsize | Пропустить до следующей границы bsize символов и вернуть число пропущенных символов. |
| padTo: bsize put: aCharacter | Пропускать, записывая аргумент aCharacter в набор, доступный получателю, для того, чтобы заполнить набор до следующей границы bsize символов. Вернуть число записанных символов. |
| padToNextWord | Сделать позиционный указатель четным (на границе слова). Вернуть пропускаемый при этом символ, если он есть. |
| padToNextWordPut: aCharacter | Сделать позиционный указатель четным (на границе слова), записывая, в случае необходимости, на место пропускаемого символа аргумент aCharacter. |
| skipWords: nWords | Установить позиционный указатель, отсчитав nWords слов от текущей позиции. |
| wordPosition | Возвратить текущую позицию, измеренную в словах. |
| wordPosition: wp | Установить текущую позицию в словах равной аргументу wp. |

Класс FileStream (ФайловыйПоток) — подкласс класса ExternalStream. Все обращения к внешним файлам выполняются через экземпляр класса FileStream. Он действует так, как будто ему доступна большая последовательность байтов или символов; элементы этой последовательности предполагаются целыми числами или символами. Протокол для класса FileStream это, по существу, протокол класса ExternalStream и его суперклассов. Кроме того, протокол обеспечивает установку и проверку состояния последовательности, доступной файловому потоку.

Классы ExternalStream и FileStream используются в системе Smalltalk-80 как основа, на которой создается и функционирует файловая система. Дополнительный протокол класса FileStream предполагает, что файловая система опирается на структуру, состоящую из каталога (или словаря файлов), в котором каждый файл — это последовательность файловых страниц. Файл однозначно идентифицируется или алфавитно-цифровым именем или серийным номером. Обычно программы пользователя не обращаются к файлу или страницам файла непосредственно; они обращаются к нему, как к последовательности

Конфиденциально

символов или байтов через файловый поток. Таким образом, программист может создавать файловый поток как средство доступа к файлу, используя выражение вида

```
FileStream fileNamed: 'name.smalltalk'
```

Экземпляру класса FileStream можно затем посылать последовательности сообщений чтения и записи, как это определено в протоколе этой главы.

Конфиденциально