10
Иерархия классов наборов

Объекты и сообщения

Классы и экземпляры

Пример приложения

Системные классы

Арифметика

Структуры данных

Управляющие структуры

Среда программирования

Просмотр и взаимодействие

Коммуникации

Словарь терминов

Литералы

Числа

Символы

Строки

Имена

Массивы

Переменные

Присваивания

Имена псевдопеременных

Сообщения

Имена сообщений и аргументы

Возвращение значений

Синтаксический анализ

Соглашения о форматировании

Каскадирование

Блоки

Управляющие структуры

Условные выражения

Аргументы блока

Словарь терминов

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.

Описание подкласса

Пример подкласса

Выбор метода

Сообщения псевдопеременной self

Сообщения псевдопеременной super

Абстрактные суперклассы

Системные сообщения для работы с подклассами

Словарь терминов

Инициализация экземпляров

Пример метакласса

Иерархия наследования метаклассов

Инициализация переменных класса

Резюме о поиске метода

Словарь терминов

Проверка функциональности объекта

Сравнение объектов

Копирование объектов

Доступ к частям объекта

Печать и сохранение объектов

Обработка ошибок

Класс Magnitude

Класс Date

Класс Time

Класс Character

Протокол числовых классов

Классы Float **и** Fraction

Случайный выбор и карточные игры

Проблема пьяного таракана

Обход бинарных деревьев

Бинарное дерево слов

Класс Stream

Позиционируемые потоки

Класс ReadStream

Класс WriteStream

Класс ReadWriteStream

Потоки генерируемых элементов

Потоки для наборов без внешних ключей

Внешние потоки и файловые потоки

Класс Collection

Подклассы класса Collection

Класс Bag

Класс Set

Класс Dictionary

Класс SequenceableCollection

Подклассы класса SequenceableCollection

Класс MappedCollection

Класс UndefinedObject

Классы Boolean, True **и** False

Дополнительный протокол класса Object

Отношения зависимости между объектами

Обработка сообщений

Сообщения системных примитивов

Процессы

Планирование

Приоритеты

Семафоры

Взаимное исключение

Разделение ресурсов

Аппаратные прерывания

Класс SharedQueue

Класс Delay

Класс Behavior

Класс ClassDescription

Класс Metaclass

Класс Class

Окна

Выбор текста

Выбор из меню

Окна просмотра

Выбор из списка

Прокрутка

Определение класса

Тестирование

Инспекторы

Сообщения об ошибках

Окно извещений

Класс DisplayObject

Класс DisplayMedium

Формы

Другие формы

Курсоры

Класс DisplayScreen

Класс DisplayText

Пути

Преобразование форм

Увеличение

Вращение

Заполнение области

Игра “Жизнь”

Компилятор

Скомпилированные методы

Байткоды

Интерпретатор

Контексты

Контексты блоков

Сообщения

Примитивные методы

Память объектов

Аппаратные средства и дополнительные системные классы

Основные понятия распределения вероятностей

Определения

Простые примеры

Класс ProbabilityDistribution

Класс DiscreteProbability

Класс ContinuousProbability

Дискретные распределения вероятностей

Распределение Бернулли

Биномиальное распределение

Геометрическое распределение

Распределение Пуассона

Непрерывные распределения вероятностей

Равномерное распределение

Экспоненциальное распределение

Гамма-распределение

Нормальное распределение

Основные понятия компьютерного моделирования

Объекты модели

Модели

Пример "по умолчанию": класс NothingAtAll

Реализация классов моделирования

Класс SimulationObject

Класс DelayedEvent

Класс Simulation

Трассировка примера NothingAtAll

Временная статистика

Гистограммы пропускной способности

Подсчет событий

Наблюдение за событиями

Реализация классов ResourcesProvider **и** WaitingSimulationObject

Расходуемые ресурсы

Нерасходуемые ресурсы

Пример: файловая система

Возобновляемые ресурсы

Пример: обслуживание паромной переправы

Реализация класса ResourceCoordinator

Пример: модель мойки автомобилей

Пример: модель паромной переправы для спецгрузовиков

Пример: банк

Пример: информационная система

На рис. 10.1 приведена схема, на которой объяснены отличия между различными классами наборов. Выбор соответствующих ветвей на этой схеме позволяет определить вид набора, который необходимо использовать при программировании.

Одно из отличий состоит в том, определен или нет порядок среди элементов набора. Другое отличие состоит в том, что доступ к элементам некоторых наборов происходит по известным вне набора именам или, иначе говоря, *ключам*. Вид ключей определяет еще одно отличие между классами наборов. Элементы одних наборов доступны по целочисленным индексам, определяющим порядок элементов в наборе, в других в качестве ключей поиска используются явно связанные с элементами набора внешние объекты.

Один вид неотсортированного набора с внешними ключами представляет класс Dictionary (Словарь). Его ключи — обычно экземпляры класса String (Строка) или LookupKey (КлючПоиска); сравнение ключей на совпадение проверяется с помощью равенства (=). Класс Dictionary имеет подкласс IdentityDictionary (СловарьИдентичности), чьи внешние ключи обычно — экземпляры класса Symbol (Имя); их сравнение на совпадение проверяется с помощью идентичности (==). Элементы экземпляров классов Bag и Set не сортируются и не связываются ни с какими ключами. В экземплярах класс Bag (ПростойНабор) возможны дубликаты элементов, и в экземплярах класса Set (Множество) — дубликаты невозможны.

Все упорядоченные наборы имеют вид SequenceableCollection (Последовательность). Их элементы доступны через ключи — целые положительные индексы. Четыре подкласса класса SequenceableCollection поддерживают разные способы упорядочивания своих элементов. Еще одно различие между этими подклассами состоит в том, могут ли элементы быть любыми объектами или они должны быть экземплярами определенного вида.

Для экземпляров классов OrderedCollection (УпорядоченныйНабор), LinkedList (СвязанныйСписок) и ArrayedCollection (ИндексированныйНабор) порядок элементов определяется внешним образом. Для OrderedCollection и LinkedList порядок элементов определяет выполненная пользователем последовательность операций добавления и удаления элементов. Элементами экземпляров OrderedCollection могут быть любые объекты, в то время как элементами экземпляров LinkedList должны быть только экземпляры вида Link. Подклассами ArrayedCollection являются Array, String, ByteArray (МассивБайтов). Элементами экземпляров классов Array и RunArray (МассивГрупп) могут быть любые объекты, элементами экземпляров классов String или Text могут быть только экземпляры класса Character, а элементами экземпляров класса ByteArray могут быть только экземпляры класса SmallInteger от 0 до 255.

Порядок среди элементов экземпляров классов Interval (Интервал) и SortedCollection (ОтсортированныйНабор) определяется внутренними свойствами самих элементов. Элементы экземпляра класса Interval составляют арифметическую прогрессию, которая однозначно определяется во время инициализации этого экземпляра. Для класса SortedCollection порядок элементов определяется задаваемым для данного набора блоком сортировки. Элементы экземпляра класса Interval должны принадлежать классу Number, а элементы SortedCollection могут быть объектами любого вида.

В дополнение к уже упомянутым подклассам Collection, класс MappedCollection (ОтображенныйНабор) представляет косвенный путь доступа к наборам, чьи элементы доступны через внешние ключи. Отображение из множества внешних ключей в набор определяется в момент создания экземпляра класса MappedCollection.

Далее в этой главе описываются все подклассы класса Collection и особенности их протоколов сообщений, а также приводятся простые поясняющие примеры.

Класс Bag

Класс Bag (ПростойНабор) — подкласс класса Collection — описывает простейший вид наборов. Его экземпляр представляет набор, чьи элементы не упорядочены и не имеют клю-

чей, поэтому он не понимает сообщений at: и at:put:. В ответ на сообщение size простой набор возвращает число элементов набора.

Экземпляр класса Bag — это просто группа элементов, которая ведет себя в соответствии с протоколом всех наборов. Общее описание набора не ограничивает числа элементов в экземпляре. Класс Bag подчеркивает это правило следующим сообщением для добавления элементов.

Bag instance protocol

adding

add: newObject withOccurrences: anInteger

Добавляет аргумент newObject в качестве элемента получателя столько раз, сколько задано в аргументе anInteger. Возвращает значение аргумента newObject.

Предположим, мы определили класс Product (Продукт), экземпляры которого представляют наименования бакалейных товаров и их цену. Новый экземпляр класса Product создается сообщением of: name at: price, посылаемым классу, а цену бакалейного товара можно определить, посылая сообщение price соответствующему экземпляру. Наполним необходимыми товарами нашу продовольственную сумку:

```
sack ← Bag new.
sack add: (Product of: #steak at: 5.80).
sack add: (Product of: #potatoes at: 0.50) withOccurrences: 6.
sack add: (Product of: #carrot at: 0.10) withOccurrences: 4.
sack add: (Product of: #milk at: 2.20).
```

Тогда счет от бакалейщика может быть определен выполнением выражений

```
amount ← 0.
sack do: [:eachProduct | amount ← amount + eachProduct price]
```

или выражения

```
sack inject: 0
      into: [:amount :eachProduct | amount + eachProduct price]
```

и будет равен $11.40. Заметим, что сообщения add:, do: и inject:into:, посылаемые простому набору, наследуются из класса Collection.

Простые наборы не упорядочены и хотя они понимают сообщения перечисления, программист не может полагаться на порядок, в котором перечисляются элементы.

Класс Set

Экземпляр класса Set (Множество) — это набор, элементы которого не упорядочены и не имеют ключей. Множество не отвечает на сообщения at: и at:put:. Класс Set во всем подобен классу Bag, за исключением того, что его экземпляры не могут содержать дубликатов, поэтому добавление нового элемента происходит только тогда, когда его не было в экземпляре.

Классы Dictionary и IdentityDictionary

Класс Dictionary (Словарь) представляет множество ассоциативных пар "ключ-значение". Элементы словаря, экземпляры класса Association (Пара), представляют собой простые структуры данных для хранения и поиска ключей и значений таких пар.

С другой стороны, словари, экземпляры класса Dictionary, можно рассматривать как наборы, чьи элементы не упорядочены, но имеют точно определенные имена (ключи). С этой точки зрения элементы словаря — это произвольные объекты (значения) с внешними ключами. Такое представление о словарях находит отражение в протоколе сообщений этого класса. Сообщения, наследуемые из класса Collection, такие как includes:, do: и другие сообщения перечисления, относятся к *значениям* словарей. То есть, эти сообщения имеют дело с объектом "значение" (value) из каждой ассоциативной пары словаря, а не с ключом или самой парой.

Сообщения at: и at:put:, наследуемые из класса Object, обращаются к ключам словарей. По аналогии с этими сообщениями в протокол добавлены сообщения associationAt: и keyAtValue: для обращения к самим парам и их значениями. Для того чтобы обеспечить дополнительный контроль во время поиска элементов словарей, определяется сообщение at:ifAbsent:, с помощью которого программист может определить свои действия в том случае, когда нужный ключ не найден. Унаследованное сообщение at:, в случае если ключ не найден, выдает ошибку.

Dictionary instance protocol

accessing

at: key ifAbsent: aBlock	Возвращает значение по заданному ключу key. Если ключ не найден, возвращается результат выполнения блока aBlock.
associationAt: key	Возвращает ассоциативную пару с заданным ключом key. Если ключ не найден, сообщается об ошибке.
associationAt: key ifAbsent: aBlock	Возвращает ассоциативную пару с заданным ключом key. Если ключ не найден, возвращается результат выполнения блока aBlock.
keyAtValue: value	Возвращает ключ для первого найденного значения value. Если такого значения нет, возвращает nil.
keyAtValue: value ifAbsent: exceptionBlock	Возвращает ключ для первого найденного значения value. Если такого значения нет, возвращает результат выполнения блока exceptionBlock.
keys	Возвращает множество, содержащее все ключи получателя.
values	Возвращает простой набор, содержащий все значения получателя, включая все дубликаты.

Как пример использования класса Dictionary, создадим словарь с именем opposites, состоящий из слов-ключей и их антонимов-значений.

```
opposites ← Dictionary new.
opposites at: #hot put: #cold.
opposites at: #push put: #pull.
opposites at: #stop put: #go.
opposites at: #come put: #go
```

Еще один способ, позволяющий добавить в словарь новую ассоциативную пару, состоит в применении сообщения с именем add: и аргументом — экземпляром класса Association.

```
opposites ← add: (Association key: #front value: #back).
opposites ← add: (Association key: #top value: #bottom)
```

Теперь словарь opposites будет состоять из следующих пар

ключ	*значение*
hot	cold
push	pull
stop	go
come	go
front	back
top	bottom

Воспользуемся протоколом проверок, наследуемым из класса Collection, для проверки значений словаря. Заметим, что сообщение includes: проверяет наличие значения, а не ключа.

выражение	*результат*
opposites size	6
opposites includes: #cold	true
opposites includes: #hot	false
opposites occurrencesOf: #go	2
opposites at: #stop put: #start	start

Четвертый пример показывает, что одно и то же значение в словаре может быть связано с любым числом ключей. А вот каждый ключ в словаре уникален и может появиться только один раз, поэтому в последнем примере для уже существовавшего ключа stop определяется новое значение start вместо "старого" значения go. В класс Dictionary добавлены также сообщения для проверки пар и ключей.

Dictionary instance protocol

dictionary testing

includesAssociation: anAssociation
Возвращает true, если получатель содержит пару anAssociation; иначе возвращает false.

includesKey: key
Возвращает true, если получатель содержит пару с ключом key; иначе возвращает false.

Тогда

выражение	*результат*
opposites includesAssociation: (Association key: #come value: #go)	true
opposites includesKey: #hot	true

Аналогично, протокол сообщений удаления для класса Collection расширяется в классе Dictionary для того, чтобы обеспечить доступ и к парам, и к их ключам, точно так же, как и к значениям. Сообщение remove: однако не поддерживается классом Dictionary, поскольку при удалении элементов обязательна ссылка на ключ.

Dictionary instance protocol

dictionary removing

removeAssociation: anAssociation
Удаляет из получателя пару anAssociation. Возвращает этот аргумент.

removeKey: key
Удаляет из получателя пару с ключом key. Возвращает значение, связанное с этим ключом. Если пары с указанным ключом в получателе нет, сообщает об ошибке.

removeKey: key ifAbsent: aBlock
Удаляет из получателя пару с ключом key. Возвращает значение, связанное с этим ключом. Если пары с указанным ключом в получателе нет, возвращает результат выполнения блока aBlock.

Примеры.

выражение	*результат*
opposites removeAssociation: (Association key: #top value: #bottom)	Пара с ключом top и значением bottom. Набор opposites имеет теперь на один элемент меньше.
opposites removeKey: #hot	Пара с ключом hot и значением cold. Набор opposites имеет теперь еще на один элемент меньше.
opposites removeKey: #cold ifAbsent: [opposites at: #cold put: #hot]	hot

Заметим, что пары с ключом cold в словаре opposites нет, поэтому в последнем примере будет выполнен блок и пара #cold и #hot станет элементом словаря.

Сообщение do: выполняет свой блок-аргумент для каждого значения из словаря. Протокол сообщений перечисления класса Dictionary, наследуемый из Collection, вновь расширяется для того, чтобы включить в него сообщения перечисления пар и ключей. Сообщения, аналогичные reject: и inject:into:, не поддерживаются.

Dictionary instance protocol

dictionary enumerating

associationsDo: aBlock
Выполняет блок aBlock для каждой пары получателя.

keysDo: aBlock
Выполняет блок aBlock для каждого ключа получателя.

Таким образом, мы имеем три различные возможности перечисления словарей. Предположим, что newWords — множество слов, которые еще не выучил ребенок. Пусть слова из словаря opposites, как ключи, так и значения, уже ребенку известны. Выполнение следующих двух выражений удалит уже известные слова из набора newWords (первое выражение удалит значения, а второе — ключи).

```
opposites do: [:word | newWords remove: word ifAbsent: []].
opposites keysDo: [:word | newWords remove: word ifAbsent: []].
```

Заметим, что когда слово из opposites отсутствует в наборе newWords, то ничего не происходит (не появится сообщение об ошибке). Аналогично можно использовать сообщение перечисления пар:

```
opposites associationsDo: [:each | newWords remove: each key ifAbsent: [].
                                   newWords remove: each value ifAbsent: []]
```

Сообщения доступа key и value можно использовать, чтобы получить соответствующие наборы слов из словаря opposites. Предположим, что все предыдущие примеры для словаря opposites выполнены, тогда

```
opposites key
```

вернет множество, содержащее элементы

```
push come front stop cold
```

а выражение

```
opposites value
```

вернет простой набор, содержащий элементы

```
pull go back start hot
```

Класс SequenceableCollection

Класс SequenceableCollection (Последовательность) представляет наборы, элементы которых упорядочены и могут именоваться с помощью целочисленных индексов. Класс SequenceableCollection — подкласс класса Collection и он поддерживает протокол доступа, копирования и перечисления элементов набора (последнее — когда известно, что существует порядок среди его элементов). В таких наборах есть первый и последний элементы, можно задать вопрос об индексе некоторого элемента (indexOf:), определить индекс первого элемента подпоследовательности в данном наборе (indexOfSubCollection:startingAt:). Все наборы наследуют из класса Object сообщения доступа к индексированным элементам. Как указано в главе 6, это сообщения at:, at:put: и size. Дополнительно, класс SequenceableCollection поддерживает размещение заданного объекта во всех задаваемых позициях (atAll:put:), размещение заданного объекта во всех позициях набора (atAllPut:). Можно заменить элементы одного набора на элементы другого (replaceFrom:to:with: и replaceFrom:to:with:startingAt:).

SequenceableCollection instance protocol

accessing

atAll: aCollection put: anObject
Заменяет каждый элемент получателя с ключом из аргумента aCollection (индексом или другим внешним ключом) на второй аргумент anObject.

atAllPut: anObject
Заменяет каждый элемент получателя на аргумент anObject.

first
Возвращает первый элемент получателя. Сообщает об ошибке, если получатель пуст.

last
Возвращает последний элемент получателя. Сообщает об ошибке, если получатель пуст.

indexOf: anElement
Возвращает первый индекс элемента получателя, равного anElement. Если такого элемента нет, возвращает 0.

indexOf: anElement ifAbsent: exceptionBlock
Возвращает индекс первого элемента получателя, равного anElement. Если такого элемента нет, возвращает результат выполнения блока exceptionBlock.

indexOfSubCollection: aSubCollection startingAt: anIndex
Если элементы в аргументе aSubCollection расположены в том же порядке, что и в получателе, то, начиная с индекса anIndex, ищет и возвращает индекс первого совпадения набора aSubCollection с поднабором получателя. Если такого нет, возвращает 0.

indexOfSubCollection: aSubCollection
  startingAt: anIndex
  ifAbsent: exceptionBlock

Если элементы в аргументе aSubCollection расположены в том же порядке, что и в получателе, то, начиная с индекса anIndex, ищет и возвращает индекс первого совпадения набора aSubCollection с поднабором получателя. Если такого нет, то возвращает результат выполнения блока exceptionBlock.

replaceFrom: start to: stop with: replacementCollection

Возвращает получатель, в котором, начиная с индекса start и заканчивая индексом stop, элементы заменены элементами аргумента replacementCollection; в replacementCollection должно быть точно stop – start + 1 элементов.

replaceFrom: start
  to: stop
  with: replacementCollection
  startingAt: repStart

Возвращает получатель, в котором, начиная с индекса start и заканчивая индексом stop, элементы заменены элементами аргумента replacementCollection, начиная с индекса repStart. Границы наборов не контролируются, исключая тот случай, когда получатель и replacementCollection совпадают, а repStart отличен от 1, — тогда сообщается об ошибке.

Рассмотрим примеры этих сообщений применительно к строкам.

выражение	*результат*
'aaaaaaaaaa' size	10
'aaaaaaaaaa' atAll: (2 to: 10 by: 2) put: $b	'abababababab'
'aaaaaaaaaa' atAllPut: $b	'bbbbbbbbbb'
'This string' first	$T
'This string' last	$g
'ABCDEFGHIJKLMNOP' indexOf: $F	6
'ABCDEFGHIJKLMNOP' indexOf: $M ifAbsent: [0]	13
'ABCDEFGHIJKLMNOP' indexOf: $Z ifAbsent: [0]	0
'The cow jumped' indexOfSubCollection: 'cow' startingAt: 1	5
'The cow jumped' replaceFrom: 5 to: 7 with: 'dog'	'The dog jumped'
'The cow jumped' replaceFrom: 5 to: 7 with: 'the spoon ran' startingAt: 5	'The spo jumped'

Все эти примеры аналогично переносятся и на экземпляры других подклассов класса SequenceableCollection, например, на Array. Так, провести замену в массиве #(The brown jug) слова brown на слово black можно, выполнив выражение

#(The brown jug) replaceFrom: 2 to: 2 with: #(black)

Заметим, что последним аргументом в этом выражении должен быть массив. Еще одно замечание: методы замены элементов не изменяют размеры набора-получателя, хотя и приводят к изменению его содержимого. Возможно, при таких преобразованиях следует сохранять набор-оригинал, делая предварительно его копию. Протокол копирования класса SequenceableCollection поддерживает копирование последовательности элементов в набор, копирование набора с заменой его части, копирование набора с удалением части элементов, копирование набора с присоединением новых элементов.

copying

, aSequenceableCollection — Конкатенация наборов. Возвращает копию получателя, к которой по порядку добавлены все элементы набора aSequenceableCollection.

copyFrom: start to: stop — Возвращает копию части получателя, состоящую из его элементов с индексами от start до stop.

copyReplaceAll: oldSubCollection with: newSubCollection — Возвращает копию получателя, в котором все вхождения подмножества oldSubCollection заменяются на набор newSubCollection.

copyReplaceFrom: start to: stop with: replacementCollection — Возвращает копию получателя, удовлетворяющую следующим условиям. Если stop меньше start, то это вставка и stop должен быть равен start–1, причем start=1 означает вставку перед первым символом, а start=size+1 — вставку за последним символом. В остальных случаях это замена и start и stop должны быть в границах получателя.

copyWith: newElement — Возвращает копию получателя, в конец которого добавлен элемент newElement.

copyWithout: oldElement — Возвращает копию получателя, из которого удалены все вхождения элементов oldElement.

Пользуясь сообщениями замены и копирования, можно написать простой текстовый редактор. Система Smalltalk-80 включает в себя классы String и Text. Протокол сообщений для класса Text тот же, что и для класса SequenceableCollection, но дополнительно класс Text связывает символы в строках с изменениями шрифтов и выделений, что позволяет использовать различные шрифты и выделения (жирный, курсив, подчеркивание) в одном тексте. Для примера воспользуемся строкой (можно было бы взять и экземпляр класса Text), предполагая, что первоначально она пуста.

```
line ← String new: 0
```

Тогда

выражение	*результат*
line ← line copyReplaceFrom: 1 to: 0 with: 'this is the first line tril'	'this is the first line tril'
line ← line copyReplaceAll: 'tril' with: 'trial'	'this is the first line trial'
line ← line copyReplaceFrom: (line size + 1) to: (line size) with: 'and so on'	'this is the first line trialand so on'
line indexOfSubCollection: 'trial' startingAt: 1	24
line ← line copyReplaceFrom: 29 to: 28 with: ' '	'this is the first line trial and so on'

Последние два сообщения из определенного выше протокола копирования полезны при копировании массивов с добавлением или удалением элементов. Например,

выражение	*результат*
#(one two three) copyWith: #four	(one two three four)
#(one two three) copyWithout: #two	(one three)

Так как элементы из экземпляров класса SequenceableCollection упорядочены, то возможно их перечисление — вызов по порядку, начиная с первого и до последнего. Возможно перечисление и в обратном порядке (от последнего к первому), для чего используется сообщение reverseDo: aBlock. Возможно также совместное перечисление двух экземпляров класса SequenceableCollection, при котором пара элементов, по одному из каждого набора, используется в качестве аргументов при выполнении блока.

SequenceableCollection instance protocol

enumerating

findFirst: aBlock	Выполняет блок aBlock с каждым элементом получателя в качестве аргумента, начиная с первого; возвращает индекс первого элемента, для которого выполнение блока завершается значением true.
findLast: aBlock	Выполняет блок aBlock с каждым элементом получателя в качестве аргумента, начиная с первого; возвращает индекс последнего элемента, для которого выполнение блока завершается значением true.
reverseDo: aBlock	Выполняет блок aBlock с каждым элементом получателя в качестве аргумента, начиная с последнего и кончая первым (все происходит в обратном порядке по сравнению с выполнением сообщения do: aBlock).
with: aSequenceableCollection do: aBlock	Выполняет двухаргументный блок aBlock для каждого элемента получателя и соответствующего элемента аргумента aSequenceableCollection в качестве аргументов блока. Размеры получателя и аргумента должны быть равны.

Следующее выражение создает словарь opposites, который ранее уже использовался как пример.

```
opposites ← Dictionary new.
#(come cold front hot push stop)
    with: #(go got back cold pull start)
    do: [:key :value | opposites at: key put: value]
```

В словаре opposites сейчас шесть элементов, каждый из которых — ассоциативная пара.

Любой экземпляр класса SequenceableCollection можно преобразовать в массив или в экземпляр класса MappedCollection. Соответствующие сообщения: asArray и mappedBy: aSequenceableCollection.

Подклассы класса SequenceableCollection

Подклассы класса SequenceableCollection суть классы OrderedCollection, ArrayedCollection, LinkedList, Interval. Класс ArrayedCollection представляет наборы элементов с фиксированным диапазоном целых чисел в качестве внешних ключей. Подклассами ArrayedCollection, например, являются классы String и Array.

Класс OrderedCollection

Экземпляры класса OrderedCollection упорядочены последовательностью, в которой их элементы добавляются и удаляются. Элементы доступны по внешним ключам — индексам. Протоколы доступа, добавления и удаления дополняются сообщениями для ссылки на первый и последний, на предыдущий и последующий элементы данного объекта.

Экземпляр класса OrderedCollection может выступать как стек или очередь. Стек — это последовательный список, в котором удаление и добавление элемента делаются в одном и том же конце списка (называемом “голова” или “вершина”). Часто о стеке говорят, что это

очередь с порядком “последним пришел — первым ушел” (“last-in first-out” или, сокращенно, LIFO-очередь).

сообщение класса OrderedCollection	*термин операции над стеком*
addLast: newObject	втолкнуть элемент newObject
removeLast	вытолкнуть элемент
last	верхний элемент
isEmpty	проверить на пустоту

Очередь — это последовательный список, в котором добавление элемента делается в одном конце списка (называемом “конец”), а удаление элемента — в другом конце списка (называемом “начало”). Часто об очереди говорят, что это очередь с порядком “первым пришел - первым ушел” (“first-in first-out” или сокращенно FIFO-очередь).

сообщение класса OrderedCollection	*термин операции над очередью*
addLast: newObject	добавить элемент newObject
removeFirst	удалить элемент
first	первый элемент
isEmpty	проверить на пустоту

Сообщение add: в классе OrderedCollection понимается так: “добавить элемент, как последний элемент в набор”; сообщение remove: понимается так: “удалить аргумент сообщения, как элемент из набора”. В дополнение к тому, что класс OrderedCollection наследует из классов SequenceableCollection и Collection, он включает следующие сообщения.

OrderedCollection instance protocol

accessing

after: oldObject
Возвращает элемент из получателя, стоящий после элемента oldObject. Если нет элемента oldObject или после него нет элемента, сообщает об ошибке.

before: oldObject
Возвращает элемент из получателя, стоящий до элемента oldObject. Если нет элемента oldObject или до него нет элемента, сообщает об ошибке.

adding

add: newObject after: oldObject
Добавляет в получатель элемент newObject после элемента oldObject. Возвращает newObject. Если элемента oldObject в получателе нет, сообщает об ошибке.

add: newObject before: oldObject
Добавляет в получатель элемент newObject непосредственно перед элементом oldObject. Возвращает newObject. Если элемента oldObject в получателе нет, сообщает об ошибке.

addAllFirst: anOrderedCollection
Добавляет все элементы из аргумента anOrderedCollection в начало получателя. Возвращает anOrderedCollection.

addAllLast: anOrderedCollection
Добавляет все элементы из аргумента anOrderedCollection в конец получателя. Возвращает anOrderedCollection.

addFirst: newObject
Добавляет элемент newObject в начало получателя. Возвращает newObject.

addLast: newObject	Добавляет элемент newObject в конец получателя. Возвращает newObject.

removing

removeFirst	Удаляет первый элемент из получателя и возвращает его. Если получатель пуст, сообщает об ошибке.
removeLast	Удаляет последний элемент из получателя и возвращает его. Если получатель пуст, сообщает об ошибке.

Класс SortedCollection

Класс SortedCollection (ОтсортированныйНабор) — подкласс класса OrderedCollection. Элементы в отсортированном наборе упорядочиваются с помощью функции двух переменных. Эта функция представляется двухаргументным блоком, который называется блоком сортировки. В отсортированные наборы новый элемент можно добавить только с помощью сообщения add:; такие сообщения, как addLast:, посредством которых программист может задать место нового элемента в наборе, не могут посылаться экземплярам класса SortedCollection.

Отсортированный набор может быть создан посылкой классу SortedCollection сообщения sortBlock: aBlock, в котором аргумент сообщения aBlock — это двухаргументный блок, устанавливающий порядок элементов в создаваемом наборе. Например,

```
SortedCollection sortBlock: [:a :b | a <= b]
```

Приведенный в этом примере блок сортировки по умолчанию считается функцией сортировки экземпляра класса SortedCollection, когда экземпляр создается посредством посылки классу простого сообщения new. Следующие примеры демонстрируют четыре способа образования нового отсортированного набора.

```
SortedCollection new
SortedCollection sortBlock: [:a :b | a > b]
anyCollection asSortedCollection
anyCollection asSortedCollection: [:a :b | a > b]
```

Есть возможность устанавливать и изменять определенный ранее блок сортировки, используя два сообщения к отсортированным наборам. Когда меняется блок сортировки, элементы набора, разумеется, переупорядочиваются. Обратим внимание на то, что одно и то же сообщение sortBlock: посылается и классу SortedCollection, когда необходимо создать новый экземпляр с указанным блоком сортировки, и экземпляру класса SortedCollection, когда необходимо изменить критерий сортировки элементов этого экземпляра.

SortedCollection class protocol

instance creation

sortBlock: aBlock	Возвращает отсортированный набор, элементы которого будут упорядочены согласно задаваемому аргументом aBlock критерию сортировки.

SortedCollection instance protocol

accessing

sortBlock	Возвращает блок сортировки получателя.
sortBlock: aBlock	Меняет блок сортировки в получателе на аргумент aBlock.

Предположим, что мы желаем создать упорядоченный по алфавиту список имен детей некоторой школьной группы.

```
children ← SortedCollection new
```

По умолчанию у набора children образовался блок сортировки [:a :b | a <= b]. Элементами этого набора могут быть строки и имена, которые, как будет показано далее, отвечают на сообщения <, >, <=, >=.

выражение	*результат*
children add: #Joe	Joe
children add: #Bill	Bill
children add: #Alice	Alice
children	SortedCollection (Alice Bill Joe)
children add: #Sam	Sam
children sortBlock: [:a :b | a > b]	SortedCollection (Sam Joe Bill Alice)
children add: #Henrietta	Henrietta
children	SortedCollection(Sam Joe Henrietta Bill Alice)

Шестое сообщение в этом примере поменяло порядок элементов в наборе children на обратный.

Класс LinkedList

Класс LinkedList (СвязанныйСписок) — другой подкласс класса SequenceableCollection, порядок элементов которого явно определяется той последовательностью, в которой они добавляются в набор и удаляются из набора. Подобно упорядоченным наборам, на элементы связанных списков можно ссылаться с помощью внешних ключей-индексов. Но в отличие от упорядоченных наборов, в которых элементами могут быть любые объекты, элементы связанных списков однородны и каждый из них должен быть экземпляром или класса Link (Связь) или его подклассов.

Экземпляр класса Link — это запись ссылки (связи) на другой экземпляр класса Link. Протокол сообщений этого класса состоит всего из трех сообщений. Одно и то же сообщение nextLink: aLink используется и для создания экземпляра класса Link, когда оно посылается классу, и для изменения связи, когда оно посылается экземпляру класса Link.

Link class protocol

instance creation

nextLink: aLink	Создает экземпляр класса Link, который ссылается на аргумент aLink.

Link instance protocol

accessing

nextLink	Возвращает ссылку получателя.
nextLink: aLink	Устанавливает ссылку получателя, равной аргументу aLink.

Так как класс Link не позволяет записать ссылку на конкретный элемент набора, то он трактуется в системе как абстрактный класс и, следовательно, его экземпляры создаваться не могут. Необходимо определять подклассы класса Link с механизмами сохранения одного или более элементов и создания экземпляров.

Так как LinkedList — подкласс класса SequenceableCollection, то его экземпляры могут отвечать на сообщения доступа, добавления, удаления и перечисления, определенные для всех наборов. Дополнительно, протокол класса LinkedList состоит из следующих сообщений:

LinkedList instance protocol

adding

addFirst: aLink	Добавляет аргумент aLink в начало списка получателя. Возвращает aLink.
addLast: aLink	Добавляет аргумент aLink в конец списка получателя. Возвращает aLink.

removing

removeFirst	Удаляет из получателя первый элемент и возвращает его. Если получатель пуст, сообщает об ошибке.
removeLast	Удаляет из получателя последний элемент и возвращает его. Если получатель пуст, сообщает об ошибке.

Подклассом класса Link в системе Smalltalk-80 является класс Process (Процесс), а подклассом класса LinkedList — класс Semaphore (Семафор), которые описываются в главе 15, посвященной многозадачности.

Разберем пример использования класса LinkedList. Как отмечалось, экземпляры класса Link не содержат никакой другой информации, кроме ссылки на другой экземпляр класса Link, поэтому создадим подкласс класса Link с именем Entry, экземпляры которого обеспечат возможность хранения одного объекта. Пусть создание экземпляра класса Entry обеспечивается сообщением for: anObject, а доступ к его элементу — сообщением element.

class name	Entry
superclass	Link
instance variable name	element

class methods

instance creation

```
for: anObject
    ↑self new setElement: anObject
```

instance methods

accessing

```
element
    ↑element
```

printing

```
printOn: aStream
    aStream nextPutAll: 'Entry for: ', element printString
```

private

```
setElement: anObject
    element ← anObject
```

Рассмотрим теперь примеры использования классов LinkedList и Entry.

выражение	*результат*
list ← LinkedList new	LinkedList()
list add: (Entry for: 2)	Entry for: 2
list add: (Entry for: 4)	Entry for: 4
list addLast: (Entry for: 5)	Entry for: 5
list addFirst: (Entry for: 1)	Entry for: 1
list	LinkedList(Entry for: 1 Entry for: 2 Entry for: 4 Entry for: 5)

```
list isEmpty                                    false
list size                                       4
list inject: 0 into: [:value :each |            12
        (each element) + value]
list last                                       Entry for: 5
list first                                      Entry for: 1
list removeFirst                                Entry for: 1
list removeLast                                 Entry for: 5
list first == list last                         false
```

Класс Interval

Другим видом наборов, подходящим под класс SequenceableCollection, является набор чисел, представляющих математическую прогрессию[1]. Например, набор может состоять из всех целых чисел в интервале от 1 до 100, или из всех четных чисел в интервале от 1 до 100. Набор может состоять из последовательности чисел, где каждое следующее число вычисляется умножением предшествующего на 2. Например, последовательность может начинаться с 1 и заканчиваться числом, меньшим или равным 100. Это будет последовательность 1, 2, 4, 8, 16, 32, 64.

Описанная прогрессия характеризуется первым числом, границей (максимумом или минимумом) для последнего вычисленного числа и правилом (методом) вычисления каждого последующего числа. Граница может быть положительной или отрицательной бесконечностью. Арифметическая прогрессия — одна из таких прогрессий, методом вычисления которой является добавление постоянной величины (инкремента). Например, это может быть последовательность чисел, в которой каждое следующее число получается добавлением -20 к предыдущему. Последовательность может начинаться со 100 и состоять из чисел, больших или равных 1. Это будет последовательность 100, 80, 60, 40, 20.

В системе Smalltalk-80 есть класс наборов, называемый Interval (Интервал), который состоит из конечных арифметических прогрессий. Он наследует сообщения из классов SequenceableCollection и Collection, а также поддерживает сообщения для инициализации экземпляров и для доступа к величинам, характеризующим данный экземпляр. Отметим, что новые элементы не могут быть добавлены в экземпляр класса Interval или удалены из него.

Протокол класса для Interval содержит следующие сообщения для создания экземпляров.

Interval class protocol

instance creation

from: startInteger to: stopInteger

Возвращает экземпляр класса Interval, начинающийся со startInteger, заканчивающийся stopInteger, и использующий добавление 1 для вычисления следующего элемента.

from: startInteger to: stopInteger by: stepInteger

Возвращает экземпляр класса Interval, начинающийся со startInteger, заканчивающийся stopInteger, и использующий добавление stepInteger для вычисления следующего элемента.

Все сообщения, подходящие для экземпляров класса SequenceableCollection, можно посылать и интервалам. В дополнение, протокол экземпляра класса Interval поддерживает сообщение increment для определения инкремента.

Класс Number содержит два сообщения, позволяющие применять более короткую запись для создания интервалов. Это сообщения to: stop и to: stop by: step. Таким образом, выражение

[1] В математике она называется возвратной или рекуррентной последовательностью. — Примеч. перев.

```
Interval from: 1 to: 10
```

эквивалентно выражению

```
1 to: 10
```

Аналогично, создать интервал, состоящий из положительных чисел с первым элементом 100 и инкрементом -20, можно как с помощью выражения

```
Interval from: 100 to: 1 by: -20
```

так и с помощью выражения

```
100 to: 1 by: -20
```

Результатом будет последовательность 100, 80, 60, 40, 20. Интервал не обязательно должен состоять из целых чисел. Например, арифметическую прогрессию 10, 10.2, 10.4, 10.6, 10.8, 11, 11.2, ..., 40 можно создать с помощью выражения

```
Interval from: 10 to: 40 by: 0.2
```

или с помощью выражения

```
10 to: 40 by: 0.2
```

Заметим, что можно было бы рассматривать более общий случай прогрессий, заменяя численное значение инкремента некоторым блоком. Когда вычисляется новый элемент, можно было бы посылать текущий элемент в качестве аргумента сообщения value: этому блоку. Реализация методов size и do: в этом случае должна учитывать возможность такого метода вычислений.

Сообщение do: применительно к интервалам обеспечивает функцию, аналогичную циклам в других языках программирования. Например, выражение языка Алгол

```
for i := 10 step 6 until 100 do
    begin
      <операторы>
    end
```

представляется выражением:

```
(10 to: 100 by: 6) do: [:i | <операторы> ]
```

Любое число отвечают на сообщение to:by:do:, поэтому приведенное выше выражение можно записать и без круглых скобок:

```
10 to: 100 by: 6 do: [:i | <операторы> ]
```

Например, увеличить на единицу каждый шестой элемент упорядоченного набора с именем numbers можно так:

```
6 to: numbers size
    by: 6
    do: [:index | numbers at: index put: (numbers at: index) + 1]
```

Это выражение создает интервал 6, 12, 18, ... до индекса последнего элемента набора numbers. Если размер набора меньше 6, то ничего не происходит. В противном случае элементы набора в позициях 6, 12, 18 и т. д. до последней возможной позиции увеличиваются на 1.

Класс ArrayedCollection

Класс ArrayedCollection (ИндексированныйНабор) — еще один подкласс класса Collection, экземпляр которого — набор элементов с фиксированным диапазоном внешних ключей-индексов. Класс ArrayedCollection в системе Smalltalk-80 имеет пять подклассов: Array, String, Text, RunArray и ByteArray.

Экземпляры класса Array (Массив) в качестве элементов могут содержать любые объекты и представляют собой простейшую структуру для хранения объектов, имеющих целые внешние ключи-индексы. Несколько примеров использования массивов уже приводились в этой главе.

Экземпляр класса String (Строка) представляет собой набор элементов — экземпляров класса Character (Символ). Класс String поддерживает дополнительный протокол для инициализации и сравнения своих экземпляров. Множество примеров использования строк уже приводились и в этой главе и в предыдущих главах книги.

Экземпляр класса Text (Текст) — это строка, в которой может изменяться шрифт и выделения символов. Текст используется в системе Smalltalk-80 для хранения информации, необходимой при создании документов, и имеет две переменные: экземпляр класса String и экземпляр класса RunArray, в котором сохраняется закодированная информация об используемых шрифтах и выделениях.

Класс RunArray (МассивГрупп) поддерживает эффективное хранение данных, которые остаются постоянными на длинных интервалах изменения индекса. Он сохраняет повторяющийся элемент только один раз и связывает с ним число последовательных вхождений этого элемента. Например, предположим, что экземпляр класса Text представляет собой строку 'He is a good boy.' из 17 символов, в которой слово boy записано жирным шрифтом, а остальной текст — обычным. Пусть код обычного шрифта в системе равен 1, а жирного шрифта — 2. Тогда экземпляр класса RunArray, связанный с приведенным экземпляром класса Text, будет состоять из 1 связанной с 13, 2 связанной с 3, 1 связанной с 1. Это означает, что первые 13 символов в тексте будут записаны в обычном шрифте, следующие 3 символа — в жирном шрифте, а затем 1 символ снова будет записан в обычном шрифте.

Экземпляр класса ByteArray (МассивБайтов) представляет собой индексированный набор целых чисел от 0 до 255. Реализация класса ByteArray хранит два байта в 16-битовом слове. Дополнительно, протокол класса ByteArray поддерживает доступ к словам памяти и словам памяти двойной длины. В системе Smalltalk-80 экземпляры класса ByteArray применяются для хранения времени в миллисекундах.

Класс String

Как уже упоминалось ранее, в протокол класса String добавлены сообщения fromString: aString — для создания копии строки и readFrom: aStream — для создания строки из символов потока, экземпляра класса Stream (Поток). Главная особенность второго сообщения в том, что пара вложенных апострофов читается и хранится как один символ апострофа. Дополнительно, класс String поддерживает протокол сравнения, подобный тому, что определен для класса Magnitude. Некоторые из этих сообщений были введены ранее при описании класса SortedCollection.

String instance protocol

comparing

< aString	Возвращает true, если символы получателя в коде ASCII предшествуют символам аргумента. Регистр букв игнорируется.
<= aString	Возвращает true, если символы получателя в коде ASCII предшествуют символам аргумента или совпадают с ними. Регистр букв игнорируется.
> aString	Возвращает true, если символы получателя в коде ASCII следуют за символами аргумента. Регистр букв игнорируется.

>= aString	Возвращает true, если символы получателя в коде ASCII следуют за символами аргумента или совпадают с ними. Регистр букв игнорируется.
match: aString	Рассматривает получатель как образец, который может содержать символы # и *. Возвращает true, если строка aString совпадает с образцом, причем при сравнении строчные и прописные буквы считаются совпадающими. Там, где получатель содержит #, аргумент может содержать любой символ; там, где получатель содержит *, аргумент может содержать любую последовательность символов (в том числе пустую).
sameAs: aString	Возвращает true, если получатель и аргумент представляют одну и ту же строку в коде ASCII. Регистр букв игнорируется.

Мы еще не давали примеров использования последних двух сообщений.

выражение	*результат*
'first string' sameAs: 'first string'	true
'First String' sameAs: 'first string'	true
'First String' = 'first string'	false
'#first String' match: 'first string'	true
'* string' match: 'any string'	true
'*.st' match: 'filename.st'	true
'first string' match: 'first *'	false

Строки можно преобразовывать в строки из только строчных или только прописных символов. Строку можно преобразовать в экземпляр класса Symbol.

String instance protocol

converting

asLowercase	Возвращает строку, составленную из получателя, символы которого приведены к нижнему регистру.
asUppercase	Возвращает строку, составленную из получателя, символы которого приведены к верхнему регистру.
asSymbol	Возвращает уникальный экземпляр класса Symbol, символы которого совпадают с символами получателя.

Приведем примеры применения этих сообщений.

выражение	*результат*
'first string' asUppercase	'FIRST STRING'
'First String' asUppercase	'first string'
'First' asSymbol	First

Класс Symbol

Экземпляр класса Symbol (Имя) — это последовательность символов, для которой гарантируется ее уникальность. Поэтому выражение

```
'a string' asSymbol == 'a string' asSymbol
```

имеет значение true. Класс Symbol поддерживает два сообщения для создания нового экземпляра.

instance creation

intern: aString	Возвращает экземпляр класса Symbol, символы которого те же, что и в строке aString.
internCharacter: aCharacter	Возвращает экземпляр класса Symbol, состоящий из одного символа aCharacter.

Кроме того, имя можно выразить литерально, если поставить символ "#" как префикс к идентификатору, имени бинарного сообщения, ключевому слову или конкатенации ключевых слов. Например, #dave — это имя, состоящее из четырех символов. Имя выводится на печать без префикса.

Класс MappedCollection[2]

Класс MappedCollection (ОтображенныйНабор) — подкласс класса Collection. Он предоставляет механизм доступа к некоторой области набора, элементы которого именованы внешними ключами. Этот механизм (отображение) может определять переупорядочение или фильтрацию элементов набора. Его основная идея состоит в том, что каждый экземпляр класса MappedCollection связан с двумя наборами: областью значений и отображением. Область значений — это набор, элементы которого доступны посредством внешних ключей, хранящихся в отображении. Отображение — набор, который связывает некоторое множество имен с другим множеством имен, причем это второе множество имен должно состоять из внешних ключей, которые могут быть использованы для доступа к элементам из области значений. Поэтому область значений и отображение должны быть экземплярами или класса Dictionary или подкласса класса SequenceableCollection.

Для примера рассмотрим определенный ранее словарь слов-антонимов opposites.

ключ	*значение*
come	go
cold	hot
front	back
hot	cold
push	pull
stop	start

Предположим, что мы создаем другой словарь — синонимов для некоторых ключей, входящих в словарь opposites. Назовем новый словарь именем alternates. Пусть он содержит следующие ассоциативные пары:

ключ	*значение*
cease	stop
enter	come
scalding	hot
shove	push

[2] Класс MappedCollection не получил распространения в дальнейших версиях языка Smalltalk-80. — Примеч. ред. перев.

Определим экземпляр класса MappedCollection с именем words, с помощью выполнения следующего выражения.

```
words ← MappedCollection collection: opposites map: alternates
```

Теперь с помощью words мы можем добраться до некоторых элементов словаря opposites. Например, выражение words at: #cease будет иметь значение start, так как значение по ключу cease в словаре alternates есть stop, а значение по ключу stop в словаре opposites есть start. Мы можем узнать на какую часть словаря opposites ссылается words посылкой ему сообщения contents:

```
words contents
```

В результате получиться экземпляр класса Bag, содержащий имена start, go, cold, pull.

Сообщение at:put: дает косвенный способ изменения области значений. Например,

выражение	*результат*
words at: #scalding	cold
words at: #cease	start
words at: #cease put: #continue	continue
opposites at: #stop	continue

Заключительное замечание о преобразовании наборов

В этой главе, описывающей основные типы наборов системы Smalltalk-80, мы указали также на возможность преобразования одних видов наборов в некоторые другие. Любой набор можно преобразовать в экземпляр классов Bag, Set, OrderedCollection или SortedCollection. Любой набор, исключая экземпляры классов Bag и Set, можно преобразовать в экземпляр класса Array или класса MappedCollection. Экземпляры классов String и Symbol могут преобразовываться друг в друга. Никакие наборы не могут преобразовываться в экземпляры классов Interval или LinkedList.