7
Линейные величины

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

Система Smalltalk-80 содержит несколько классов, представляющих объекты, которые измеряют линейно упорядоченные количества. В окружающем нас мире примерами таких измеримых количеств могут быть: (1) величины, измеряющие промежутки времени, такие как дата и время; (2) пространственные величины, такие как расстояние; (3) численные величины, такие как вещественные и рациональные числа.

Класс Magnitude

Является ли одно число больше другого? Следует ли эта дата за той? Предшествует ли один момент времени другому? Стоит ли некоторая буква за другой буквой в алфавите? Равно ли одно расстояние другому или оно меньше его?

Общий протокол для ответа на все эти вопросы предоставляет класс Magnitude (Величина). Он обеспечивает сравнение объектов, допускающих линейное упорядочение. Класс Magnitude включает подклассы Time (Время), Date (Дата) и Number (Число). Класс Character (Символ), экземпляры которого используются как элементы строки, и класс LookupKey (КлючПоиска), экземпляры которого используются как ключи в ассоциативных словарях, тоже реализованы как подклассы класса Magnitude. Класс Character интересен как пример неизменяемых объектов системы и рассматривается в этой главе. Класс LookupKey менее интересен и мы его отложим до глав, посвященных наборам. Класс Distance (Расстояние) в системе Smalltalk-80 не определяется.

Magnitude instance protocol

comparing

< aMagnitude	Определяет, меньше ли получатель аргумента aMagnitude.
<= aMagnitude	Определяет, меньше ли или равен получатель аргументу aMagnitude.
> aMagnitude	Определяет, больше ли получатель аргумента aMagnitude.
>= aMagnitude	Определяет, больше ли или равен получатель аргументу aMagnitude.
between: min and: max	Определяет, больше ли или равен получатель аргументу min и при этом он меньше или равен аргументу max.

Хотя для сравнения на равенство двух величин класс Magnitude наследует из суперкласса Object метод =, каждый подкласс класса Magnitude должен переопределять этот метод. Метод = в классе Magnitude имеет вид:

```
self subclassResponsibility
```

Если подкласс класса Magnitude не реализует метод =, то при попытке послать экземпляру этого подкласса сообщение = будет выдано специальное сообщение об ошибке, которое укажет, что подкласс должен реализовать сообщение =, как это предписано ему суперклассом.

Экземпляры подклассов класса Magnitude могут также отвечать на сообщения, которые определяют, которая из двух линейно упорядоченных величин больше или меньше.

Magnitude instance protocol

comparing

min: aMagnitude	Из двух величин, получателя и аргумента, возвращает ту, которая меньше.
max: aMagnitude	Из двух величин, получателя и аргумента, возвращает ту, которая больше.

Заметим, что протокол сравнения на совпадение ==, ~= и ~~ наследуется из класса Object. Используя целые числа, как разновидность объектов класса Magnitude, получим:

выражение	*результат*
3 <= 4	true
3 > 4	false
5 between: 2 and: 6	true
5 between: 1 and: 4	false
34 min: 45	34
34 max: 45	45

Программист не может создать экземпляры класса Magnitude, такое возможно только для его подклассов. Объясняется это тем, что класс Magnitude не в состоянии реализовать все свои сообщения: часть из них он реализует с помощью выражения self subclassResponsibility.

Класс Date

После того, как определен общий протокол класса Magnitude, есть возможность дополнить его протоколом, который бы поддерживал арифметические операции и выполнял специальные линейные измерения. Первым из таких возможных расширений мы рассмотрим класс Date.

Экземпляр класса Date представляет собой конкретный день от начала юлианского календаря. Этот день существует в конкретном месяце и году. Классу Date известна некоторая очевидная информация:

1. в неделе 7 дней, каждый день имеет имя и индекс 1, 2,..., 7;
2. в году 12 месяцев, каждый месяц имеет имя и индекс 1, 2,..., 12;
3. в месяце может быть 28, 29, 30 или 31 день;
4. год может быть високосным.

Протокол, доступный для объекта Date, поддерживает запросы о датах вообще и о конкретных датах в частности. Оба класса, Date и Time, дают примеры таких классов системы, о которых существуют специальные знания, в большей степени характеризующие класс в целом и доступные всему классу, а не его экземплярам. Этот “протокол класса” определен в метаклассе. Для первого знакомства с протоколом класса Date рассмотрим сообщения, поддерживающие общие запросы.

Date class protocol

general inquiries

dayOfWeek: dayName	Возвращает индекс (1, 2,... или 7) дня недели по его имени dayName.
nameOfDay: dayIndex	Возвращает имя дня недели по его индексу, где 1 — Monday (понедельник), 2 — Tuesday (вторник), и т.д.
indexOfMonth: monthName	Возвращает индекс месяца (1, 2,..., 12) по его имени monthName.
nameOfMonth: monthIndex	Возвращает имя месяца по его индексу, где 1 — January (январь), 2 — February (февраль), и т.д.

dayInMonth: monthName forYear: yearInteger
: Возвращает число дней в месяце monthName в году yearInteger (год должен быть известен, чтобы определить, является ли он високосным).

dayInYear: yearInteger
: Возвращает число дней в году yearInteger.

leapYear: yearInteger
: Возвращает 1, если год yearInteger високосный, иначе возвращает 0.

dateAndTimeNow
: Возвращает массив, первый элемент которого — текущая дата (экземпляр класса Date, представляющий сегодняшнюю дату), а второй элемент — текущее время (экземпляр класса Time, представляющий данный момент времени).

Итак, мы можем послать классу Date такие сообщения.

выражение	*результат*
Date daysInYear: 1982	365
Date dayOfWeek: #Wednesday	3
Date nameOfMonth: 10	October
Date leapYear: 1972	1 (означает, что год — високосный)
Date daysInMonth: #February forYear: 1972	29
Date daysInMonth: #Feb forYear: 1971	28

В классе Date для названий дней недели и месяцев можно использовать общепринятые сокращения[1].

Для создания экземпляров класса Date могут использоваться следующие четыре сообщения. Одно из них, а именно Date today, особенно часто используется в системе Smalltalk-80 при определении даты создания файла.

Date class protocol

instance creation

today
: Возвращает экземпляр класса Date, представляющий дату посылки этого сообщения.

fromDays: dayCount
: Возвращает дату, которая соответствует дню dayCount до или после 1 января 1901 г. (в зависимости от знака аргумента).

newDay: day month: monthName year: yearInteger
: Возвращает дату, которая соответствует дню day в месяце monthName года yearInteger.

newDay: dayCount year: yearInteger
: Возвращает дату, которая соответствует дню dayCount с начала года yearInteger.

Вот четыре примера сообщений, создающих даты:

выражение	*результат*
Date today	3 February 1982
Date fromDays: 200	20 July 1901
Date newDays: 6 month: #Feb year: 82	6 February 1982
Date newDays: 3 year: 82	3 January 1982

[1] Эти сокращения следующие: дни недели — Mon, Tue, Wen, Thu, Fri, Sat, Sun; месяцы — Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec. — Прим. ред. перев.

Сообщения, которые могут посылаться экземплярам класса Date, делятся на следующие категории: accessing (доступ), inquiries (запросы), arithmetic (арифметические) и printing (печать). Сообщения доступа и запросов о конкретных датах содержат:

- индекс дня, месяца или года;
- число секунд, дней или месяцев, прошедших с какой-либо другой даты;
- общее число дней в рассматриваемом месяце или годе;
- остаток дней в рассматриваемом месяце или годе;
- первый день в заданном месяце;
- имя дня недели или имя месяца для заданной даты;
- дату заданного дня недели, предшествующего некоторой дате.

Следующие простые арифметические операции с датами поддерживаются протоколом класса Date.

Date instance protocol

arithmetic	
addDays: dayCount	Возвращает дату спустя заданное число дней dayCount после получателя.
subtractDays: dayCount	Возвращает дату, предшествующую получателю на заданное число дней dayCount.
subtract: aDate	Возвращает целое число, представляющее число дней между получателем и аргументом.

Такая арифметика с датами полезна, например, для того, чтобы вычислить дату возвращения книги в библиотеку или число дней просрочки возврата. Если dueDate — дата возвращения книги, то число дней просрочки мы получим, выполняя выражение

```
Date today subtractDate: dueDate
```

Если выданная сегодня книга должна быть возвращена через две недели, то дата возврата книги в библиотеку вычисляется выражением

```
Date today addDays: 14
```

Если читатель желает закончить работу с книгой за 16 дней до католического Рождества 1997 года, то последним днем работы будет

```
(Date newDays: 25 month: #December year: 1997) subtractDays: 16.
```

Алгоритм определения суммы штрафа fine, которую должен заплатить читатель библиотеки за задержку возврата книги, может быть следующим: сначала сравнить сегодняшнюю дату с датой возврата книги и, если книга возвращена с опозданием, определить общую сумму штрафа из расчета оплаты $0.1 за каждый день задержки.

```
Date today < dueDate
    ifTrue: [fine ← 0]
    ifFalse: [fine ← 0.10 * (Date today subtractDate: dueDate)]
```

Класс Time

Экземпляр класса Time представляет некоторую секунду в течение дня. Считается, что день начинается в полночь. Time — подкласс Magnitude. Подобно классу Date, класс Time отвечает на сообщения из категории общих запросов, которые определяются в протоколе класса.

Time class protocol

general inquiries

millisecondClockValue	Возвращает число миллисекунд после последней установки системных часов на 0.
millisecondsToRun: timeBlock	Возвращает число миллисекунд, необходимых для выполнения блока выражений timeBlock.
timeWords	Возвращает число секунд по Гринвичу, прошедших с 1 января 1901 года. Ответ представляется четырехэлементным экземпляром класса ByteArray, который описывается в главе 10.
totalSeconds	Возвращает общее число секунд, прошедшее с 1 января 1901 года, скорректированное на часовой пояс и сезонное изменение времени.
dateAndTimeNow	Возвращает массив, чей первый элемент — текущая дата, а второй элемент — текущее время. Результат посылки этого сообщения классу Time тот же, что и результат посылки этого сообщения классу Date.

Сообщение millisecondsToRun: timeBlock — единственный неочевидный запрос в этом протоколе. Рассмотрим пример

```
Time millisecondsToRun: [Date today]
```

В результате его выполнения мы получим число миллисекунд, которые необходимы системе, чтобы вычислить текущую дату. Ввиду того, что существуют некоторые накладные расходы при выполнении этого сообщения, и того, что точность часов в компьютере воздействует на результат, осторожному программисту следует определить границы машинно-зависимых погрешностей, связанные с выбором приемлемых аргументов для этого сообщения.

Новый экземпляр класса Time создается посылкой сообщения now классу Time; при этом соответствующий метод читает текущее время на системных часах компьютера. Кроме того, экземпляр класса Time можно создать, если послать классу Time сообщение fromSeconds: secondCount, где secondCount — число секунд после полуночи.

Time class protocol

instance creation

now	Возвращает экземпляр класса Time, представляющий время посылки сообщения.
fromSeconds: secondCount	Возвращает экземпляр класса Time, представляющий время спустя число секунд secondCount после полуночи.

Протокол доступа к экземплярам класса Time обеспечивает получение информации о числе часов (hours), минут (minutes) и секунд (seconds) в рассматриваемом экземпляре.

Поддерживаются также арифметические операции.

Time instance protocol

arithmetic

addTime: timeAmount	Возвращает экземпляр класса Time, который показывает время на timeAmount позже получателя.

subtractTime: timeAmount	Возвращает экземпляр класса Time, который показывает время на timeAmount раньше получателя.

В приведенных выше сообщениях аргумент timeAmount должен быть экземпляром либо класса Time, либо класса Date. Чтобы это было возможно, система должна приводить экземпляры обоих классов к общей единице измерения. Такой единицей является секунда. Для класса Time перевод экземпляра класса в секунды дает число секунд после полуночи. Для класса Date перевод экземпляра класса в секунды дает число секунд между временем 1 января 1901 года и тем же временем даты получателя сообщения. Для поддержки этих методов каждый из классов умеет отвечать на преобразующее сообщение asSeconds.

Time instance protocol

converting

asSeconds	Возвращает число секунд между полуночью и временем, которое определяет получатель.

Date instance protocol

converting

asSeconds	Возвращает число секунд между временем 1 января 1901 года и тем же временем дня получателя.

Арифметические операции с экземплярами класса Time аналогичны таким же операциям с экземплярами класса Date. Предположим, что сотрудник работает над проектом на условиях почасовой оплаты. Он начинает работать во время startTime и работает без перерыва до настоящего момента; работа закончена и надо определить сегодняшний заработок. Если ставка составляет $5.0 в час и если учитываются только полностью отработанные часы, то на данный момент заработанная сумма равна

```
(Time now subtractTime: startTime) hours * 5.0
```

Если предположить, что любая часть часа, большая 30 минут, считается за полный час, то сотруднику следует добавить $5.0, если

```
(Time now subtractTime: startTime) minutes > 30
```

Кто продуктивнее работал — рабочий, который закончил работу во время timeA, или тот, который закончил работу во время timeB? Ответ очевиден: первый, если timeA < timeB. Протокол сравнения наследуется классом Time из суперклассов Magnitude и Object.

Предположим, что вычисляется время в течении нескольких дней, например, время пробега автомобиля в четырехдневном ралли. Если первый день ралли — startDate, а автомобили стартуют во время startTime, то время для автомобиля, только что пересекшего линию финиша вычисляется по следующему алгоритму.

Пусть время старта 6 часов утра:

```
startTime ← Time fromSeconds: (60 * 60 * 6)
```

2 февраля 1982 года:

```
startDate ← Date newDay: 2 month: #Feb year: 82
```

Тогда время, прошедшее от времени старта до начала текущего дня, равно

```
todayStart ← (((Time fromSeconds: 0) addTime: Date today)
                    subtractTime: startDate)
                        subtractTime: startTime)
```

То есть, надо взять число секунд с 1 января 1901 года до начала сегодняшней даты и из него вычесть число секунд с 1 января 1901 года до времени старта. Можно, конечно, вычислить и число секунд в прошедших днях, но тогда программист должен сам провести все необходимые преобразования.

```
(Date today subtractDate: startDate) * 24 * 60 * 60)
```

Добавляя еще текущее время дня, мы получим время, затраченное автомобилем на пробег:

```
todayStart addTime: Time now
```

Класс Character

Класс Character — третий подкласс класса Magnitude, который мы будем рассматривать. Он включен как подкласс в класс Magnitude потому, что все экземпляры класса Character образуют линейно упорядоченное множество, то есть для двух символов из этого множества всегда можно сказать, какой из символов предшествует (<) или следует (>) за другим. Всего в системе 256 экземпляров класса Character, каждый из которых связывается с кодом из расширенной ASCII-таблицы символов.

Экземпляр класса Character может быть представлен литерально в виде символа алфавита, которому предшествует знак доллара ($). Таким образом, $A — символ, представляющий прописную букву A. Протокол для создания экземпляров класса Character следующий.

Character class protocol

instance creating	
value: anInteger	Возвращает экземпляр класса Character, чье значение равно аргументу anInteger. Значение связывается с элементом ASCII-таблицы символов. Например, выражение Character value: 65 возвращает прописную латинскую букву 'A'.
digitValue: anInteger	Возвращает экземпляр класса Character, чье числовое значение есть аргумент anInteger. Например, возвращает $9, если аргумент 9; возвращает $0, если аргумент 0; возвращает $A, если аргумент 10; и $Z, если аргумент 35. Этот метод полезен при (грамматическом) разборе чисел в строке. Обычно используются только символы до $F (при шестнадцатеричной системе счисления).

Протокол класса, а именно, множество сообщений к объекту Character, обеспечивает словарь для доступа и к таким символам, которые нелегко распознать при печати: backspace (возврат на шаг), cr (возврат каретки), newPage (перевод формата), space (пробел) и tab (табуляция).

Сообщения к экземплярам класса Character предоставляют доступ к ASCII-значениям и к цифровым значениям экземпляров, проверяют тип символа. Единственное состояние экземпляра класса Character в системе — его значение, которое не может быть изменено. Объекты системы, которые не могут изменить свое внутреннее состояние, называются неизменяемыми объектами. Это означает, что однажды созданные, они не могут уничтожаться и воссоздаваться заново, когда в этом возникает нужда. 256 экземпляров класса Character создаются во время инициализации системы и остаются в ней без изменений. Как только запрашивается новый символ с ASCII-кодом от 0 до 255, устанавливается ссылка на уже существующий символ. В этом смысле все 256 экземпляров класса Character уникальны. Кроме класса Character, система Smalltalk-80 включает другие неизменяемые объекты — экземпляры классов SmallInteger и Symbol.

Character instance protocol

accessing	
asciiValue	Возвращает число, соответствующее ASCII-коду получателя.
digitValue	Возвращает число, соответствующее цифровому представлению получателя (см. сообщение создания экземпляра digitValue:).

testing

isAlphaNumeric	Возвращает true, если получатель — буква[2] или цифра.
isDigit	Определяет, является ли получатель цифрой.
isLetter	Определяет, является ли получатель буквой.
isLowercase	Определяет, является ли получатель строчной буквой.
isUppercase	Определяет, является ли получатель прописной буквой.
isSeparator	Определяет, является ли получатель одним из символов-разделителей: пробелом, табуляцией, возвратом каретки, переводом строки, переводом формата.
isVowel	Определяет, является ли получатель одной из гласных букв, то есть a, e, i, o или u (строчной или прописной).

Кроме этого, протокол экземпляра предоставляет возможность преобразовывать символ в верхний регистр (asUppercase), в нижний регистр (asLowcase) и в имя (asSymbol).

Простое сравнение символов в алфавитном порядке демонстрирует применение протокола экземпляров класса Character. Например, мы желаем узнать, предшествует ли одна строка другой в телефонной книге. Строка отвечает на сообщение at: возвращением элемента строки с индексом, равным аргументу сообщения; элементами строки являются символы. Таким образом, 'abc' at: 2 возвращает $b. Определим специальный метод в классе String с именем min:. Метод возвращает ту из двух строк, получателя сообщения или аргумент, которая стоит раньше по алфавиту.

```
min: aString
    1 to self size do:
      [:index |
       (index > aString size) ifTrue: [↑aString].
       (self at: index) > (aString at: index) ifTrue: [↑aString].
       (self at: index) < (aString at: index) ifTrue: [↑self]].
    ↑self
```

Алгоритм этого метода состоит из двух выражений. Первое — итерация по каждому элементу получателя сообщения. Итерация прекращается тогда, когда выполняется одно из трех условий: (1) в аргументе aString больше нет символов для сравнения с очередным символом получателя (index > aString size); (2) следующий символ в получателе расположен после соответствующего символа в аргументе aString ((self at: index) > (aString at: index)); (3) следующий символ в получателе расположен до соответствующего символа в аргументе aString. Как пример, реализующий ситуацию (1), проведем сравнение строк 'abcd' и 'abc'. Тогда при значении index = 4, итерация прервется и будет возвращена строка 'abc', как первая по алфавиту. Случай (2) реализуется, если строка 'abde', получатель сообщения, сравнивается со строкой 'abce'. Когда index = 3, выполняется условие $d > $c, и метод возвращает строку 'abce'. В том случае, когда сравниваются строки 'az' и 'by', реализуется ситуация (3). Метод прервется при index = 1 и возвратит строку 'az'. В том случае, когда получатель имеет меньше символов, чем аргумент, и даже тогда, когда получатель составляет начальную подстроку аргумента, выполнение первого выражения завершится и будет выполнено второе выражение, которое вернет получателя сообщения как результат. Последнее произойдет, например, при сравнении строк 'abc' и 'abcd'.

Отметим, что арифметические операции с символами не поддерживаются. Например, следующее выражение не допустимо:

```
a ← $A + 1
```

Будет выдано сообщение об ошибке, поскольку символы не понимают сообщения +.

---

[2] Здесь и далее под буквой в языке Smalltalk-80 понимается буква латинского алфавита. Некоторые локализованные версии систем Smalltalk позволяют настраиваться на национальные алфавиты. Следует, однако, отметить, что применение стандартных системных сообщений к национальным символам может привести к различному поведению программ в зависимости от кодировки этих символов. Для более безопасного применения лучше вводить новые методы для обработки национальных символов, например isRussianLetter (этоРусскаяБуква) и т.п. — Прим. ред. перев.