8
Числовые классы

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

Конфиденциально

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

Конфиденциально

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

Конфиденциально

Конфиденциально

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

Конфиденциально

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

Конфиденциально

Конфиденциально

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

Конфиденциально

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

Конфиденциально

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

Конфиденциально

Одна из важнейших целей системы программирования Smalltalk-80 — применение единой парадигмы для обработки всей информации. Парадигма Smalltalk-80, как уже отмечалось ранее, это взаимодействие объектов посредством передачи сообщений. Такая точка зрения близка к той, что использовалась в языке Simula для создания имитационных моделей. Одним из самых серьезных вызовов применению парадигмы Smalltalk-80 во всех аспектах программирования была арифметика. Язык Симула использует парадигму объект/сообщение только для высшего уровня взаимодействия в создаваемых имитационных моделях. Для арифметики, как и для большинства алгоритмических структур управления, Simula опирается на язык программирования Algol с его встроенными представлениями чисел, операциями и синтаксисом. Утверждение, что даже сложение двух целых чисел следует рассматривать как результат посылки сообщения, на первых порах становления языка Smalltalk встречало некоторое сопротивление. Практика показала, что преимущество максимального единообразия в построении языка программирования перевешивает некоторое неудобство его реализации. На опыте нескольких версий языка Smalltalk были разработаны методы реализации, которые позволили избежать лишних посылок сообщений для наиболее важных арифметических операций, поэтому сейчас почти нет никакой платы за концептуальные преимущества языка.

Объекты, представляющие числовые значения, применяются в большинстве систем, написанных на языке Smalltalk-80 (как и в большинстве других языков программирования). Числа, естественно, используются при проведении математических вычислений, а также в алгоритмах как индексы, счетчики, коды состояний (часто называемые флагами). Целые числа используются и как наборы двоичных цифр (битов), на которых выполняются логические операции маскирования.

Каждый вид числовых величин представляется в системе отдельным классом. Числовые классы реализованы таким образом, что все числа ведут себя так, как если бы они принадлежали наиболее общему числовому типу. Фактические классы конкретных числовых объектов определяются тем, какая необходима общность, чтобы представить их значения. Поэтому общий протокол всех числовых объектов наследуется из класса Number (Число). Number имеет три подкласса: Float (Вещественное), Fraction (Дробь) и Integer (Целое). В свою очередь класс Integer имеет три подкласса: SmallInteger (МалоеЦелое), LargeNegativeInteger (БольшоеОтрицательноеЦелое), LargePositiveInteger (БольшоеПоложительноеЦелое). Целые числа содержат дополнительный протокол для работы с числами как с последовательностями битов. Этот протокол определен в классе Integer. Числа в системе являются экземплярами классов Float, Fraction, SmallInteger, LargePositiveInteger, LargeNegativeInteger. Классы Number и Integer определяют общий для других классов протокол, но они не определяют конкретных представлений для числовых величин. Поэтому эти два класса не должны создавать экземпляры.

В отличие от других объектов, которые могут изменять свое внутреннее состояние, единственное состояние числа — его значение, которое не может меняться. Объект 3, например, никогда не может изменить свое состояние на 4, иначе это приведет к катастрофе в системе.

Протокол числовых классов

Класс Number определяет протокол всех числовых классов. Его сообщения поддерживают стандартные арифметические операции и операции сравнения. Большинство из них реализуются подклассами, поскольку существенно зависят от фактического представления числовых значений.

Протокол арифметических сообщений содержит сообщения для обычных бинарных арифметических операций (таких, как +, -, *, /) и несколько унарных и ключевых сообщений для вычисления абсолютной и противоположной величин числа, для вычисления целого частного и остатка.

Конфиденциально

arithmetic

+ aNumber	Возвращает сумму получателя и аргумента aNumber.
– aNumber	Возвращает разность между получателем и аргументом.
* aNumber	Возвращает произведение получателя на аргумент.
/ aNumber	Возвращает результат деления получателя на аргумент. Заметим, что деление всегда вычисляется с максимально возможной точностью; если точное деление невозможно, результат представляется экземпляром класса Fraction.
// aNumber	Возвращает целое частное от деления получателя на аргумент с усечением к минус бесконечности.
\\ aNumber	Возвращает целочисленный остаток от деления получателя на аргумент с усечением к минус бесконечности.
abs	Возвращает абсолютную величину получателя.
negated	Возвращает число, которое является противоположным получателю сообщения.
quo: aNumber	Возвращает целое частное от деления получателя на аргумент с усечением к нулю.
rem: aNumber	Возвращает целочисленный остаток от деления получателя на аргумент с усечением к нулю.
reciprocal	Возвращает результат деления 1 на получатель сообщения; сообщает об ошибке, если получатель равен 0.

Несколько примеров.

выражение	*результат*
1 + 10	11
5.6 – 3	2.6
5 – 2.6	2.4
(–4) abs	4
6 / 2	3
7 / 2	(7/2) — экземпляр класса Fraction с числителем 7 и знаменателем 2.
7 reciprocal	(1/7) — экземпляр класса Fraction с числителем 1 и знаменателем 7.

Арифметические сообщения, возвращающие целочисленные частное и остаток, определены двумя способами. В одном округление происходит в сторону минус бесконечности, в другом — в сторону нуля. Для положительных ответов результат всегда один и тот же, так как нуль и минус бесконечность располагаются в одном направлении. Для отрицательных ответов результат будет разный.

Следующая таблица показывает взаимосвязь имен сообщений.

результат	*усечение в сторону минус бесконечности*	*усечение в сторону нуля*
частное	//	quo:
остаток	\\	rem:

Несколько примеров.

выражение	*результат*

Конфиденциально

6 quo: 2	3
7 quo: 2	3
(7 quo: 2) + 1	4
7 quo: 2 + 1	2
7 rem: 2	1
7 // 2	3
7 \\ 2	1
7 \\ 2 + 1	2
–7 quo: 2	–3
–7 rem: 2	–1
–7 // 2	–4
–7 \\ 2	1

Результаты посылки сообщений quo:, rem: и // всегда положительны, когда аргумент и получатель одного знака, и отрицательны, когда они разных знаков. Бинарное сообщение \\ всегда возвращает положительное число.

Дополнительно, протокол класса Number определяет следующие математические функции:

Number instance protocol

mathematical functions

exp	Возвращает вещественное число, равное экспоненте получателя.
ln	Возвращает вещественное число, равное натуральному логарифму получателя.
log: aNumber	Возвращает вещественное число, равное логарифму по основанию aNumber от получателя.
floorLog: radix	Возвращает целую часть от числа, равного логарифму по основанию radix от получателя.
raisedTo: aNumber	Возвращает вещественное число, равное получателю, возведенному в степень aNumber.
raisedToInteger: anInteger	Возвращает вещественное число, равное получателю, возведенному в целую степень anInteger (аргумент должен быть экземпляром класса Integer или его подклассов).
sqrt	Возвращает вещественное число, равное положительному квадратному корню получателя.
squared	Возвращает число, равное произведению получателя на самого себя.

Некоторые примеры.

выражение	*результат*
2.718282 ln	1.0
6 exp	403.429
2 exp	7.38906
7.38906 ln	1.99999 (то есть 2)
2 log: 2	1.0
2 floorLog: 2	1
6 log: 2	2.58496
6 floorLog: 2	2

Конфиденциально

6 raisedTo: 1.2	8.58579
6 raisedToInteger: 2	36
64 sqrt	8
8 squared	64

Определить, четные числа или нечетные, положительные или отрицательные, позволяют следующие сообщения:

Number instance protocol

testing	
even	Возвращает true, если получатель четное число.
odd	Возвращает true, если получатель нечетное число.
negative	Возвращает true, если получатель меньше 0.
positive	Возвращает true, если получатель больше или равен 0.
strictlyPositive	Возвращает true, если получатель больше 0.
sign	Возвращает 1, если получатель больше 0; возвращает –1, если получатель меньше 0; иначе возвращает 0.

Свойства чисел подвергаться усечению или округлению описываются следующим протоколом.

Number instance protocol

truncation and round off	
ceiling	Возвращает ближайшее к получателю целое число, не меньшее получателя.
floor	Возвращает ближайшее к получателю целое число, не большее получателя.
truncated	Возвращает ближайшее к получателю целое число в сторону нуля.
truncateTo: aNumber	Возвращает ближайшее к получателю вещественное число в сторону нуля, кратное аргументу.
rounded	Возвращает ближайшее к получателю целое число.
roundTo: aNumber	Возвращает ближайшее к получателю вещественное число, кратное аргументу.

Сообщение truncated можно использовать еще и для того, чтобы вещественное число преобразовать в целое. Примеры.

выражение	*результат*
16.32 ceiling	17
16.32 floor	16
–16.32 ceiling	–16
–16.32 floor	–17
–16.32 truncated	–16
16.32 truncated	16
16.32 truncateTo: 5	15
16.32 truncateTo: 5.1	15.3
16.32 rounded	16
16.32 roundTo: 6	18
16.32 roundTo: 6.3	18.9

Конфиденциально

Протокол класса Number включает сообщения для преобразования чисел в другие объекты или в числа, выраженные в других единицах измерения. Прежде всего, это преобразование градусов в радианы. Следующие два сообщения выполняют такие преобразования.

Number instance protocol

converting

degreesToRadian	Предполагая, что получатель представлен в градусах, возвращает его представление в радианах.
radianToDegrees	Предполагая, что получатель представлен в радианах, возвращает его представление в градусах.

Тогда, например,

```
30 degreesToRadian = 0.523599
90 degreesToRadian = 1,5708.
```

В протокол математических функций включены тригонометрические и логарифмические функции. Получатель сообщений для тригонометрических функций cos, sin, tan задается в радианах. Результат, возвращаемый сообщениями arcSin, arcCos, arcTan, также представляет собой угол в радианах.

В следующих примерах 30 градусов — это 0.523599 радиан, 90 градусов — это 1.5708 радиан.

выражение	*результат*
0.523599 sin	0.5
0.523599 cos	0.866025
0.523599 tan	0.57735
1.5708 sin	1.0
0.57735 arcTan	0.523599
1.0 arcSin	1.5708

Когда складываются два целых числа, результат будет целым числом; когда складываются два вещественных числа, результат будет вещественным число. То есть, класс результата будет тот же самый, что и класс слагаемых. Когда два операнда принадлежат классу SmallInteger, а их сумма слишком велика для этого класса, результат будет представлен как экземпляр класса LargePositiveInteger или LargeNegativeInteger. Определить класс результата, когда операнды принадлежат разным классам, несколько сложнее. Здесь используются два критерия: во-первых, должно быть как можно меньше потерянной информации и, во-вторых, в коммутативных операциях, независимо от того, какой операнд — получатель сообщения, а какой — аргумент, результат должен быть один и тот же. Например, 3.1 * 4 вернет тот же самый результат, что и 4 * 3.1.

Соответствующее представление результата операции над числами разных классов определяется численной мерой общности, приписываемой в системе каждому числовому классу. Класс имеет тем большую общность, чем больше приписываемая ему численная мера. Каждый класс должен “уметь” превращать свои экземпляры в равные по значению экземпляры класса с большей общностью. Мера общности используется при определении того, какой из операндов следует преобразовывать. Таким образом, арифметические операции подчиняются закону коммутативности без потери числовой информации. Когда различие между двумя числовыми классами есть только вопрос точности (где “точность” — содержащаяся в числе мера информации), то более точным будет тот класс, у которого больше мера общности. Мы произвольно положили, что приближенные числа имеют более высокую общность в тех случаях, когда точность не важна (например, класс Float имеет большую общность, чем класс Fraction).

Иерархия общности числовых классов в системе Smalltalk-80 по мере ее убывания следующая:

Конфиденциально

Float
Fraction
LargeNegativeInteger, LargePositiveInteger
SmallInteger

Сообщения в протоколе класса Number, которые обеспечивают преобразования одних типов чисел в другие, объединены в категорию coercing (приведение).

Number instance protocol

coercing

coerce: aNumber	Возвращает число, представленное аргументом в виде экземпляра того же самого числового класса, что и получатель. Этот метод должен быть определен во всех подклассах класса Number.
generality	Возвращает число, представляющее место получателя в иерархии общности.
retry: aSymbol coercing: aNumber	Арифметическая операция над получателем и аргументом aNumber, обозначенная аргументом aSymbol, не может быть выполнена из-за различий в представлениях операндов. Преобразуется либо получатель, либо аргумент aNumber, в зависимости от того, который из них имеет меньшую общность, и затем вновь делается попытка выполнить операцию. Если aSymbol равен #sign, то возвращается false, когда аргумент — не число. Если общности операндов одинаковы, то сообщение retry:coercing: не следует посылать, так как это вызовет сообщение об ошибке.

Так, если мы попытаемся вычислить 32.45 * 4, умножение вещественного числа на малое целое, то в результате выполнится выражение

```
32.45 retry: #* coercing: 4
```

и аргумент 4 будет преобразован в вещественное число 4.0 (Float имеет большую общность, чем SmallInteger). После чего будет успешно выполнено умножение.

Определение иерархии чисел в терминах числовой меры общности работает для типов чисел, входящих в стандартную систему Smalltalk-80, поскольку общность обладает свойством транзитивности для этих типов чисел. Однако понятие меры общности не может использоваться вообще со всеми типами чисел.

Экземпляры класса Interval (Интервал), детально описываемого в главе 10, могут создаваться посылкой одного из двух сообщений числу. Для каждого элемента такого интервала можно выполнить блок с этим элементом в качестве аргумента блока.

Number instance protocol

intervals

to: stop	Возвращает экземпляр класса Interval от получателя до аргумента stop, каждый следующий элемент которого вычисляется прибавлением 1 к предыдущему.
to: stop by: step	Возвращает экземпляр класса Interval от получателя до аргумента stop, каждый следующий элемент которого вычисляется прибавлением величины step к предыдущему.
to: stop do: aBlock	Создает экземпляр класса Interval от получателя до аргумента stop, каждый следующий элемент которого вычисляется прибавлением 1 к предыдущему. Выполняет аргумент aBlock для каждого элемента этого интервала.
to: stop by: step do: aBlock	Создает экземпляр класса Interval от получателя до аргумента stop, каждый следующий элемент которого вычисляется прибавлением величины step к предыдущему. Выполняет аргумент aBlock для каждого элемента этого интервала.

Итак, если выполним

Конфиденциально

```
a ← 0.
10 to: 100 by: 10 do: [:each | a ← a + each]
```

то окончательное значение переменной a будет равно 550.

Если a — массив #('one' 'two' 'three' 'four' 'five'), то каждый элемент этого массива доступен по индексу, которые вместе составляют интервал от 1 до размера массива. Следующее выражение изменяет каждый элемент массива так, что сохраняется только его первый символ.

```
1 to: a size do: [:index | a at: index put: ((a at: index) at: 1)]
```

Результат вычисления — массив #($o $t $t $f $f). Заметим, что подобно массиву, символы строки доступны с помощью сообщений at: и at:put:. Сообщения к объектам, подобным массивам и строкам, детально обсуждаются в главах 9 и 10.

Классы Float и Fraction

Классы Float и Fraction дают два способа представления нецелых величин. Экземпляры класса Float представляют (иногда приблизительно) вещественные числа. Они обеспечивают точность около 6 десятичных знаков в диапазоне между ±10 в степени ±32. Некоторые примеры:

```
8.0
13.34
0.3
2.5e6
1.27e-30
–12.987654e12
```

Экземпляры класса Fraction — дроби, точно представляющие рациональные числа. Все арифметические операции над рациональными дробями дают в результате приведенную дробь.

Экземпляры класса Float могут быть созданы с помощью литеральной записи в тексте метода (например, 3.13159) или получены как результаты арифметических операций, в которых один из операндов экземпляр класса Float.

Экземпляры класса Fraction могут быть получены как результат арифметических операций, когда один из операндов рациональная дробь, а второй не является экземпляром класса Float (если бы он был таким, то результат также был бы вещественным числом, поскольку мера общности класса Float выше, чем у класса Fraction). Рациональная дробь может быть еще получена при выполнении операции деления (/) целых чисел, когда они не делятся нацело. Дополнительно к этому, протокол класса Fraction содержит сообщение вида numerator: numInteger denominator: denInteger (числитель: числЦелое знаменатель: знамЦелое), которое применяется для создания рациональных дробей.

Класс Float отвечает на сообщение pi, возвращая соответствующую константу. В протокол экземпляра класса Float в категорию truncation and round off (усечение и округление) добавлены два метода с именами fractionPart (дробнаяЧасть) и integerPart (целаяЧасть), которые возвращают дробную и целую части получателя; в категорию converting (преобразование) добавлен метод с именем asFraction (какДробь), который преобразует получатель в рациональную дробь. Аналогично, в протокол класса Fraction добавлен метод с именем asFloat (какВещественное), который преобразует рациональную дробь в вещественное число.

Конфиденциально

Классы целых чисел

Класс Integer добавляет протокол, специфичный для целых чисел. В Integer три подкласса. Класс SmallInteger дает компактное представление для широкого диапазона целых чисел, которые чаще всего применяются в индексах и счетчиках. Диапазон представления для таких чисел немного меньше, чем для величин, представляемых одним машинным словом. Большие целые числа, представляемые в зависимости от своего знака экземплярами классов LargePositiveInteger или LargeNegativeInteger, не имеют границ для своих значений. Цена за общность больших целых чисел — большее время вычислений. Заметим, что если результат арифметических операций над большими целыми может быть представлен малым целым числом, то он в действительности будет малым целым.

В дополнение к сообщениям, наследуемым из класса Number, протокол класса Integer расширяет категорию преобразований сообщениями asCharacter (какСимвол), asFloat, asFraction, а категорию печати сообщениями printOn: aStream base: b (печататьНа: поток основание: b) и printStringRadix: baseInteger (печататьСтрокуСОснованием: основаниеЦелое). Также вводится новая категория enumerating (перечисления).

Так, 8 printStringRadix: 2 даст 2r1000. В качестве примера перечислений можно, используя сообщение timesRepeat: aBlock (разПовторить: блок), повторить выполнение блока целое число раз. В примере, где блок не имеет аргументов, выполним:

```
a ← 1.
10 timesRepeat: [a ← a + a]
```

Окончательное значение переменной a будет число 1024, то есть 2 в степени 10.

Класс Integer вводит в протокол категорию factorization and divisibility (факторизация и делимость), не определенную для других чисел.

Integer instance protocol

factorization and divisibility

factorial	Возвращает факториал получателя. Получатель должен быть больше или равен нулю.
gcd: anInteger	Возвращает наибольший общий делитель получателя и аргумента.
lcm: anInteger	Возвращает наименьшее общее кратное получателя и аргумента.

Примеры.

выражение	*результат*
3 factorial	6
55 gcd: 30	5
6 lcm: 10	30

В дополнение к операциям с целыми как с числами, некоторые алгоритмы учитывают тот факт, что целые числа можно интерпретировать как последовательности битов. Следующий протокол сообщений определен в классе Integer для манипуляции с битами.

Integer instance protocol

bit manipulation

allMask: anInteger	Рассматривает аргумент как битовую маску. Возвращает true, если все биты, равные 1 в получателе, равны 1 и в маске.
anyMask: anInteger	Рассматривает аргумент как битовую маску. Возвращает true, если какой-либо из битов, равный 1 в получателе, равен 1 и в маске.
noMask: anInteger	Рассматривает аргумент как битовую маску. Возвращает true, если ни один из битов, равных 1 в получателе, не равен 1 в маске.

Конфиденциально

bitAnd: anInteger	Возвращает экземпляр класса Integer, все биты которого получены как результат выполнения операции логического “И” над соответствующими битами получателя и аргумента.
bitOr: anInteger	Возвращает экземпляр класса Integer, все биты которого получены как результат выполнения операции логического “ИЛИ” над соответствующими битами получателя и аргумента.
bitXor: anInteger	Возвращает экземпляр класса Integer, все биты которого получены как результат выполнения операции логического “ИСКЛЮЧАЮЩЕГО ИЛИ” над соответствующими битами получателя и аргумента.
bitAt: index	Возвращает бит (0 или 1) в позиции index получателя.
bitInvert	Возвращает экземпляр класса Integer, все биты которого обратны соответствующим битам получателя.
highBit	Возвращает индекс старшего бита в двоичном представлении получателя.
bitShift: anInteger	Возвращает экземпляр класса Integer, чье значение в двоичном представлении берется из получателя (в двоичном представлении) сдвигом влево на число битов, указанных аргументом. Отрицательный аргумент сдвигает вправо. При левом сдвиге справа ставятся нули. При правом сдвиге распространяется знаковый бит.

Несколько примеров. Заметим, что по умолчанию основание системы счисления при выводе чисел, принимается равным 10.

выражение	*результат*
2r1110001110000111	29127
2r101010101010101	21845
2r101000101000101	20805
2r0001110001110000	3640
29127 allMask: 20805	true
29127 allMask: 21845	false
29127 anyMask: 21845	true
29127 noMask: 3640	true
29127 bitAnd: 3640	0
29127 bitOr: 3640	32767
32767 radix: 2	2r111111111111111
29127 bitOr: 21845	30167
30167 printStringRadix: 2	2r111010111010111
3640 bitShift: 1	7280

Класс Random: генератор случайных чисел

Многие приложения требуют выбора случайных чисел. Случайные числа нужны, например, в статистических приложениях и в алгоритмах шифровки данных. Класс Random (СлучайноеЧисло) — это генератор случайных чисел, включенный в стандартную систему Smalltalk-80. Он обеспечивает простой способ получения последовательности случайных чисел, которые равномерно распределены на интервале между (но не включая) 0.0 и 1.0.

Конфиденциально

Экземпляр класса Random содержит источник, значение которого используется для генерации следующего случайного числа. Этот источник инициализируется псевдослучайным образом. Когда требуется следующая случайная величина, экземпляру класса Random посылается сообщение next (следующий).

Генератор случайных чисел создается с помощью выражения

```
rand ← Random new
```

Чтобы получить случайное число, необходимо выполнить выражение rand next, которое вернет вещественное число между 0.0 и 1.0.

Реализация сообщения next опирается на линейный конгруэнтный метод Лемера, как он изложен в книге Д. Кнута, *Искусство программирования для ЭВМ*, т.2, — М.: "Мир", 1977.

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

Можно послать экземпляру класса Random сообщение next: anInteger, чтобы получить упорядоченный набор, содержащий anInteger целых случайных чисел, и сообщение nextMatchFor: aNumber, чтобы определить, равно ли следующее случайное число числу aNumber.

Предположим, нам необходимо при помощи генератора случайных чисел rand случайным образом выбрать одно из 10 первых натуральных чисел (1, 2, ..., 10). Этого можно добиться, выполнив выражение

```
(rand next * 10) truncated + 1
```

По шагам:

выражение	*результат*
rand next	случайное число между 0.0 и 1.0
rand next * 10	случайное число между 0.0 и 10.0
(rand next * 10) truncated	целое число >= 0 и <= 9
(rand next * 10) truncated + 1	целое число >= 1 и <= 10

Конфиденциально