Squeak Smalltalk: Classes Reference

Version 0.0, 20 November 1999, by Andrew C. Greenberg, werdna@mucow.com
Version 1.2, 26 April 2001, by Andrew P. Black, black@cse.ogi.edu

Based on:
Smalltalk-80: The Language and Its Implementation, Author: Adele Goldberg and David Robson
Squeak Source Code, and the readers of the Squeak mailing list.

Squeak site: http://www.squeak.org

Contents

See also the Squeak Language Reference page

Fundamental Classes and Methods

Class Object (Operations on all objects)

Instance Creation (Class Side)

Message	Description	Notes
new	Answer a new instance of the receiver (which is a class). This is the usual way of creating a new object. **new** is often overridden in subclasses to provide class-specific behavior.	1, 2
basicNew	This is the primitive that is ultimately called to implement **new**.	3
new: anInteger	Answer an instance of the receiver (which is a class) with size given by anInteger. Only allowed if it makes sense to specify a size.	4

Notes:

1. The usual body for a **new** method is `^ super new initialize`. Remember to put it on the class side, remember to type the ^, and remember to say `super`, not `self`!
2. Do *not* implement new if it makes no sense, For example, look at Boolean class>>new and MappedCollection class>>new.
3. **basicNew** is there so you can still make instances even if a superclass has overridden **new**. Consequently, never override **basicNew**, until you become a wizard.
4. If you need an initialization parameter other than a size, choose a more meaningful name than **new:** For example, look at the instance creation protocol for Pen class and Rectangle class.

Comparing Objects

Message	Description	Notes
== anObject	Are the receiver and the argument the same object? (Answer is **true** or **false**)	1
~~ anObject	Are the receiver and the argument different objects?	1
= anObject	Are the receiver and the argument equal? The exact meaning of equality depends on the class of the receiver.	2
~= anObject	Are the receiver and the argument unequal?	2
hash	Answer a SmallInteger whose value is related to the receiver's value.	2

Notes:

1. == and ~~ should not normally be redefined in any other class.
2. Since various classes (particularly Sets and Dictionaries) rely on the property that equal objects have equal hashes, you should override `hash` whenever you override =. It must be true that `(a = b)` implies `(a hash = b hash)`. The contrapositive and the converse will not generally hold.

Testing Objects

Message	Description	Notes
isNil	Is the receiver nil? (Answer is **true** or **false**)	1
notNil	Is the receiver not nil?	1
ifNil: aBlock	Evaluate aBlock if the receiver is nil, and answer the value of aBlock. Otherwise answers the receiver.	2
ifNotNil: aBlock	Evaluate aBlock if the receiver is not nil, and answer the value of aBlock. Otherwise answers the receiver, *i.e.*, nil	2
ifNotNilDo: aBlock	If the receiver is not nil, evaluate aBlock with the receiver as argument. Answers the receiver.	3

Notes:

1. `anObject isNil` is preferred to `anObject == nil`, and similarly for `anObject ~~ nil`.
2. `anobject ifNil: [ ... ]` is preferred to `anObject isNil ifTrue: [ ... ]`.
3. `ifNotNilDo: aBlock` is useful if the receiver is a complex expression, for example
   `self leftChild rightChild ifNotNilDo: [ :node | node balance ]`

Copying Objects

Message	Description	Notes
copy	Answer another instance just like the receiver. Subclasses typically override this method; they typically do not override shallowCopy.	
shallowCopy	Answer a copy of the receiver which shares the receiver's instance variables.	
deepCopy	Answer a copy of the receiver with its own copy of each instance variable.	
veryDeepCopy	Do a complete tree copy using a dictionary. An object in the tree twice is copied once and shared by both referents.	

Sending Messages to Objects

Message	Description	Notes
perform: aSymbol	Send the unary selector, aSymbol, to the receiver. Signal an error if the number of arguments expected by the selector is not zero.	
perform: aSymbol **with:** anObject	Send the selector aSymbol to the receiver with anObject as its argument. Fail if the number of arguments expected by the selector is not one.	1
perform: selector **withArguments:** argArray	Send the selector, aSymbol, to the receiver with arguments in argArray. Fail if the number of arguments expected by the selector does not match the size of argArray.	

Note:

1. Squeak objects also recognize `#perform:with:with:` and `#perform:with:with:with`

Indexing Objects

Message	Description	Notes
at: index	Answer the value of the index'th element of the receiver. Signal an Error if index is not an Integer or is out of bounds.	
at: index **put:** anObject	Store anObject in the receiver at the element index. Signal an Error if index is not an Integer or is out of bounds. Answer anObject.	
at: index **modify:** aBlock	Replace the element at index of the receiver with that element transformed by the block.	
size	Answer the number of indexable elements in the receiver. This value is the same as the largest legal subscript.	

Displaying and Storing Objects

Message	Description	Notes
printString	Answer a String whose characters describe the receiver.	
printOn: aStream	Append to aStream a String whose characters describe the receiver.	
storeString	Answer a String from which the receiver can be reconstructed.	
storeOn: aStream	Append to aStream a String from which the receiver can be reconstructed	

Interrogating Objects

Message	Description	Notes
class	Answers the receiver's class (an object).	
isKindOf: aClass	Is the receiver an instance of aClass or one of its subclasses?	
isMemberOf: aClass	Is the receiver an instance of aClass? (Same as `rcvr class == aClass`)	
respondsTo: aSelector	Can the receiver find a method for aSelector, either in the receiver's class or in one of its superclasses?	
canUnderstand: aSelector	Does the receiver, which must be a class, have a method for aSelector? The method can belong to the receiver or to any of its superclasses.	

Miscellaneous Messages on Objects

Message	Description	Notes
yourself	Answers self.	1
asString	Answers the receiver's printString.	
doesNotUnderstand: aSymbol	Report that the receiver does not understand aSymbol as a message.	
error: aString	Signal an Error.	
halt	Stops execution.	2

Notes:

1. the message `yourself` is mostly used as the last message in a cascade, when the previous message answered some object other than the receiver. For example,
   `#(1 2 3 5) at: 4 put: 4` answers `4`, the object that was put, whereas
   `#(1 2 3 5) at: 4 put: 4; yourself` answers `#(1 2 3 4)`, the receiver.
2. `self halt` is the usual way of forcing entry to the debugger. The halt can be resumed.

Class Boolean

This abstract class represents logical values, providing Boolean operations and conditional control structures. It has two subclasses, True and False, each of which have singleton instances represented by the Squeak keywords **true** and **false**, respectively.

Evaluating and Non-Evaluating Logical Operations for Boolean

Message	Description	Notes
& aBoolean	Evaluating conjunction (AND). Evaluate the argument. Then answer true if both the receiver and the argument are true.	
| aBoolean	Evaluating disjunction (OR). Evaluate the argument. Then answer true if either the receiver or the argument is true.	
eqv: aBoolean	Answer true if the receiver is equivalent to aBoolean.	
not	Negation. Answer true if the receiver is false, answer false if the receiver is true.	
xor: aBoolean	Exclusive OR. Answer true if the receiver is not equivalent to aBoolean.	
and: alternativeBlock	Nonevaluating conjunction. If the receiver is true, answer the *value* of alternativeBlock; otherwise answer false without evaluating alternativeBlock.	
or: alternativeBlock	Nonevaluating disjunction. If the receiver is false, answer the *value* of alternativeBlock; otherwise answer true without evaluating alternativeBlock.	

Class Magnitude

This abstract class embraces, among other classes, Numbers, Characters, Date and Time. It addresses classes whose instances can be linearly ordered.

Message	Description	Notes
< aMagnitude	Answer whether the receiver is strictly less than the argument.	
> aMagnitude	Answer whether the receiver is strictly greater than the argument.	
<= aMagnitude	Answer whether the receiver is less than or equal to the argument.	
>= aMagnitude	Answer whether the receiver is greater than or equal to the argument.	
between: min **and:** max	Answer whether the receiver is greater than or equal to the argument, min, and less than or equal to the argument, max.	
min: aMagnitude	Answer the receiver or the argument, whichever is the lesser magnitude.	
max: aMagnitude	Answer the receiver or the argument, whichever is the greater magnitude.	
min: firstMagnitude **max:** secondMagnitude	Take the receiver or the argument, firstMagnitude, whichever is the lesser magnitude, and answer that or the argument, secondMagnitude, whichever is the greater magnitude.	

Class Character

Squeak has its own set of 256 characters, which may differ from that of the host platform. Instances of class Character store an 8-bit character code.

- The characters 0-127 are the same as the corresponding ASCII characters, with a few exceptions: the assignment arrow replaces underscore, and characters for the enter, insert, page up, page down, home, and the 4 arrow keys replace some of the ACSII control characters. These characters can be accessed from Squeak using methods in class Character.
- The characters 128-255 are sparsely populated. Various symbols, such as bullets, trademark, copyright, cent, Euro and Yen, diphthongs and a fair number of accented characters as well as non-breaking space (Character nbsp) are available at the same codes as in the Macintosh character set, but fewer characters are assigned than on the Macintosh.
- The full character set can be viewed by doing a printIt on `Character allCharacters`.

Methods for instance creation (Class side)

Most of the time, characters literals `$a`, `$b`, *etc.* are used in preference to class methods. The principal exceptions are the non-printing characters listed here. Programs should never need to depend on the details of the character encoding.

Message	Description	Notes
value: n	n must be an integer in the range 0 to 255. Answer the Character with code n	1
digitValue: x	Answer the Character whose digit value is x. For example, answer $9 for x=9, $0 for x=0, $A for x=10, $Z for x=35.	
arrowDown arrowLeft arrowRight arrowUp backspace cr delete end enter escape euro home insert lf linefeed nbsp newPage pageDown pageUp space tab	Answer the appropriate character	

Note:

1. The invariant `(Character value: n) asciiValue = n` holds for all `n` in the range [0..255].

Methods for accessing Characters

Message	Description	Notes
asciiValue	Answer the value used in the receiver's encoding. This is not really ASCII, despite the name!	1
digitValue	Answer 0-9 if the receiver is $0-$9, 10-35 if it is $A-$Z, and < 0 otherwise. This is used to parse literal numbers of radix 2-36.	

Note:

1. Character has the unique instance property, so that all equal ("=") instances of a character are identical ("=="). That is, `a asciiValue == b asciiValue` if and only if `a == b`.

Methods for testing Characters

Message	Description	Notes
isAlphaNumeric	Answer whether the receiver is a letter or a digit.	
isDigit	Answer whether the receiver is a digit.	
isLetter	Answer whether the receiver is a letter.	
isLowercase	Answer whether the receiver is a lowercase letter.	
isSeparator	Answer whether the receiver is one of the separator characters: space, cr, tab, line feed, or form feed.	
isSpecial	Answer whether the receiver is one of the special characters	
isUppercase	Answer whether the receiver is an uppercase letter.	
isVowel	Answer whether the receiver is one of the vowels, AEIOU, in upper or lower case.	
tokenish	Answer whether the receiver is a valid token-character—letter, digit, or colon.	

Methods for converting Characters

Message	Description	Notes
asLowercase	If the receiver is uppercase, answer its matching lowercase Character.	
asUppercase	If the receiver is lowercase, answer its matching uppercase Character.	

Numeric Classes and Methods

Class Number

This abstract class embraces Integers, Floats and Fractions. Number is a subclass of Magnitude.

Methods for arithmetic on all Numeric Classes

Message	Description	Notes
+ aNumber	Answer the sum of the receiver and aNumber.	
- aNumber	Answer the difference of the receiver and aNumber.	
* aNumber	Answer the product of the receiver and aNumber.	
/ aNumber	Answer the result of dividing the receiver by aNumber, retaining as much precision as possible. If the result is not exact, the answer will be a Fraction or Float, as appropriate. Signal ZeroDivide if aNumber is Zero.	
// aNumber	Answer the result of dividing the receiver by aNumber, truncating toward negative infinity. Signal ZeroDivide if aNumber is Zero.	
\\ aNumber	Answer the remainder left when dividing the receiver by aNumber, truncating toward negative infinity. This is the modulus operator. Signal ZeroDivide if aNumber is Zero.	
quo: aNumber	Answer the result of dividing the receiver by aNumber, truncating toward zero. Signal ZeroDivide if aNumber is Zero.	
rem: aNumber	Answer the remainder left when dividing the receiver by aNumber, truncating toward zero. Signal ZeroDivide if aNumber is Zero.	
abs	Answer the absolute value of the receiver.	
negated	Answer the negation of the receiver.	
reciprocal	Answer 1 divided by the receiver. Signal ZeroDivide if the receiver is zero.	

Methods implementing mathematical functions for Numbers

Message	Description	Notes
exp	Answer a floating point number that is the exponential of the receiver	
ln	Answer the natural log of the receiver.	
log: aNumber	Answer the logarithm base aNumber of the receiver.	
floorLog: aNumber	Take the logarithm base aNumber of the receiver, and answer the integer nearest that value towards negative infinity.	
raisedTo: aNumber	Answer the receiver raised to the power of aNumber.	
raisedToInteger: anInteger	Answer the receiver raised to the power of anInteger. Signal an Error if anInteger is not an integer!	
sqrt	Answer a floating point number that is the positive square root of the receiver.	
squared	Answer the receiver multiplied by itself.	

Methods for testing Numbers

Message	Description	Notes
even	Answer whether the receiver is even.	
odd	Answer whether the receiver is odd.	
negative	Answer whether the receiver is less than zero.	
positive	Answer whether the receiver is greater than or equal to zero.	
strictlyPositive	Answer whether the receiver is greater than zero.	
sign	Answer 1 if the receiver is strictly positive, zero if the receiver is zero, and -1 if the receiver is strictly negative.	
isZero	Answer whether the receiver is zero.	

Methods for truncating and rounding Numbers

Message	Description	Notes
ceiling	Answer the Integer nearest the receiver toward positive infinity.	
floor	Answer the Integer nearest the receiver toward negative infinity.	
truncated	Answer an integer nearest the receiver toward zero.	
truncateTo: aNumber	Answer the multiple of aNumber that is nearest the receiver toward zero.	
rounded	Answer the integer nearest the receiver.	
roundTo: quantum	Answer the nearest number to the receiver that is a multiple of quantum.	
roundUpTo: quantum	Answer the multiple of quantum that is nearest the receiver toward infinity	

Methods for trigonometry on Numbers

Message	Description	Notes
sin	Answer the sine of the receiver taken as an angle in radians.	
cos	Answer the cosine of the receiver taken as an angle in radians.	
tan	Answer the tangent of the receiver taken as an angle in radians.	
degreeSin	Answer the sin of the receiver taken as an angle in degrees.	
degreeCos	Answer the cosine of the receiver taken as an angle in degrees.	
arcSin	Answer an angle in radians whose sine is the receiver.	
arcCos	Answer an angle in radians whose cosine is the receiver.	
arcTan	Answer an angle in radians whose tangent is the receiver.	
arcTan: denominator	Answer the angle in radians whose tan is the receiver divided by denominator.	
degreesToRadians	Answer the receiver in radians. Assumes the receiver is in degrees.	
radiansToDegrees	Answer the receiver in degrees. Assumes the receiver is in radians.	

Class Integer

Methods for arithmetic on Integers

Message	Description	Notes
isPowerOfTwo	Answer whether the receiver is a power of two.	
factorial	Answer the factorial of the receiver.	
gcd: anInteger	Answer the greatest common denominator of the receiver and the argument.	
lcm: anInteger	Answer the least common multiple of the receiver and the argument.	
take: r	Answer the number of combinations of the receiver, taken r at a time.	

Methods for bit manipulation on Integers

A range of bit manipulation operations are available on Integers. They are rarely needed, however, so they are not described here. Of course, they can be viewed using the browser.

Collection Classes and Methods

The Collection Hierarchy

Class	Description
Collection	Abstract Class for Collections
Bag	Unordered, unindexed collection of objects
Set	Unordered, unindexed collection of unique objects
Dictionary	Set of associations (values are indexable by keys)
IdentityDictionary	Dictionary, but comparisons are done using ==
IdentitySet	Set, but comparisons are done using ==
SequenceableCollection	Ordered collection, indexed by integers
OrderedCollection	Ordered according to manner elements are added and removed
SortedCollection	Ordered according to value of a "sortBlock"
LinkedList	Homogeneous SequenceableCollection of Links
Interval	Homogeneous sequence of arithmetic progression of Integers
ArrayedCollection	Ordered collection, indexed by fixed range of Integers
Array	ArrayedCollection of arbitrary Objects
Array2D	Homogeneous ArrayedCollection of Arrays
ByteArray	Homogeneous ArrayedCollection of Bytes (Integers -128..255)
FloatArray	Homogeneous ArrayedCollection of Floating point numbers
IntegerArray	Homogeneous ArrayedCollection of Signed 32-bit Integers
PointArray	Homogeneous ArrayedCollection of Points (with 32-bit values)
RunArray	Homogeneous ArrayedCollection of Integers (sparse RLE representation)

ShortIntegerArray	Homogeneous ArrayedCollection of Signed 16-bit Integers
ShortPointArray	Homogeneous ArrayedCollection of Points (with 16-bit values)
ShortRunArray	Homogeneous ArrayedCollection of Signed 16-bit Ints (sparse RLE rep)
String	Homogeneous ArrayedCollection of Characters
Symbol	Homogeneous ArrayedCollection of Characters (with unique instance property)
Text	Homogeneous ArrayedCollection of Characters with associated text attributes
WordArray	Homogeneous ArrayedCollection of Unsigned 32-bit Integers
Heap	Like SortedCollection, but stores information as a heap. (see Heapsort)
MappedCollection	Means for accessing an indexable Collection, using a mapping from a collection of "external" keys to the accessed collection's "indexing" keys. The MappedCollection can then be used directly, indexing and changing the accessed collection via the external keys.

Class Collection

Methods for creating Collections (Class Side)

Message	Description	Notes
with: anObject	Answer an instance of the receiver containing anObject	
with: firstObject **with:** secondObject	Answer an instance of the receiver containing all the arguments as elements. (Squeak recognizes instantiators of this type up to six "with:" clauses).	
withAll: aCollection	Answer an instance of the receiver containing all the elements from aCollection.	

Methods for testing Collections

Message	Description	Notes
isEmpty	Answer whether the receiver contains any elements.	
occurrencesOf: anObject	Answer how many of the receiver's elements are equal to anObject.	
anySatisfy: aBlock	Evaluate aBlock with the elements of the receiver. If aBlock returns true for any element return true. Otherwise return false	
allSatisfy: aBlock	Evaluate aBlock with the elements of the receiver. If aBlock returns true for all elements return true. Otherwise return false	
includes: anObject	Answer whether anObject is one of the receiver's elements.	
includesAllOf: aCollection	Answer whether all the elements of aCollection are in the receiver.	
includesAnyOf: aCollection	Answer whether any element of aCollection is one of the receiver's elements.	

Methods for adding and removing Collection elements

Message	Description	Notes
anyOne	Answer a specimen element of the receiver (any one at all). Signal an error if the receiver is empty.	
add: newObject	Include newObject as one of the receiver's elements. Answer newObject. ArrayedCollections cannot respond to this message.	
addAll: newObject	Include all the elements of aCollection as the receiver's elements. Answer aCollection.	
remove: oldObject	Remove oldObject as one of the receiver's elements. Answer oldObject unless no element is equal to oldObject, in which case, signal an Error.	
remove: oldObject **ifAbsent:** anExceptionBlock	Remove oldObject as one of the receiver's elements. If several of the elements are equal to oldObject, only one is removed. If no element is equal to oldObject, answer the result of evaluating anExceptionBlock. Otherwise, answer oldObject.	
removeAll: aCollection	Remove each element of aCollection from the receiver. If successful for each, answer aCollection. Otherwise signal an Error.	
removeAllFoundIn: aCollection	Remove from the receiver each element of aCollection that is present in the receiver.	
removeAllSuchThat: aBlock	Evaluate aBlock with each element as argument, and remove that element if the answer is true.	
difference: secondCollection	Answer a new collection that is computed by copying the receiver and removing all the elements found in secondCollection.	

Methods for enumerating Collections

Message	Description	Notes
do: aBlock	Evaluate aBlock with each of the receiver's elements as the argument.	
do: aBlock **separatedBy:** separatorBlock	Evaluate aBlock for all elements in the receiver, and if there is more than one element, evaluate the separatorBlock between each pair of elements in the receiver.	
select: aPredicateBlock	Evaluate aPredicateBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver, only those elements for which aPredicateBlock evaluates to true. Answer the new collection.	
reject: aPredicateBlock	Evaluate aPredicateBlock with each of the receiver's elements as the argument. Collect into a new collection like the receiver only those elements for which aPredicateBlock evaluates to false. Answer the new collection.	
collect: aMappingBlock	Evaluate aMappingBlock with each of the receiver's elements as the argument. Collect the resulting values into a collection like the receiver. Answer the new collection.	
detect: aPredicateBlock	Evaluate aPredicateBlock with each of the receiver's elements as the argument. Answer the first element for which aPredicateBlock answers true. Signal an Error if none are found.	
detect: aPredicateBlock **ifNone:** exceptionBlock	Evaluate aPredicateBlock with each of the receiver's elements as the argument. Answer the first element for which aPredicateBlock evaluates to true. If there is none, answer the result of evaluating exceptionBlock.	
inject: initialValue **into:** binaryBlock	Accumulate a running value associated with evaluating binaryBlock. The running value is initialized to initialValue. The current running value and the next element of the receiver are provided as the arguments to binaryBlock. For example, to compute the sum of the elements of a numeric collection, aCollection **inject:** 0 **into:** [:subTotal :next | subTotal + next].	
collect: aMappingBlock **thenSelect:** aPredicateBlock	Evaluate aMappingBlock with each of the receiver's elements as the argument. Collect the resulting values that satisfy aPredicateBlock into a collection like the receiver. Answer the new collection.	
select: aPredicateBlock **thenCollect:** aMappingBlock	Evaluate aMappingBlock with each of the receiver's elements for which aPredicateBlock answers true as the argument. Collect the resulting values into a collection like the receiver. Answer the new collection.	
count: aPredicateBlock	Evaluate aPredicateBlock with each of the receiver's elements as the argument. Return the number that answered true.	

Bag

Methods for accessing Bags

Message	Description	Notes
add: newObject **withOccurrences:** anInteger	Add the element newObject to the receiver. Do so as though the element were added anInteger number of times. Answer newObject.	

Dictionary and IdentityDictionary

Methods for Accessing Dictionaries

Dictionaries are homogenous Sets of key and value pairs. These pairs are called Associations: key and value can be any object. Instances of Association are created by sending the binary message "`key -> value`" (`->` is defined in Object). Dictionaries have the property that each key occurs at most once. IdentityDictionaries have the same property, but determine uniqueness of keys using == instead of =. In ordinary use, both kinds of Dictionary are indexed using the unique key to obtain the corresponding value.

Message	Description	Notes
at: aKey	Answer the value associated with aKey. Signal an Error if no value is associated with aKey.	
at: aKey **ifAbsent:** aBlock	Answer the value associated with aKey. If no value is associated with aKey, answer the value of aBlock.	
associationAt: aKey	Answer the association whose key is aKey. If there is none, signal an Error	
associationAt: aKey **ifAbsent:** aBlock	Answer the association whose key is aKey. If there is none, answer the value of aBlock.	
keyAtValue: aValue	Answer the key of the first association having aValue as its value. If there is none, signal an Error.	
keyAtValue: aValue **ifAbsent:** exceptionBlock	Answer the key of the first association having aValue as its value. If there is none, answer the result of evaluating exceptionBlock.	
keys	Answer a Set containing the receiver's keys.	
values	Answer an Array containing the receiver's values.	
includes: aValue	Does the receiver contain a value equal to aValue?	
includesKey: aKey>	Does the receiver have a key equal to aKey?	
do: aBlock	Evaluate aBlock with each of the receiver's values as argument.	
keysDo: aBlock	Evaluate aBlock with each of the receiver's keys as argument.	
valuesDo: aBlock	same as **do:**	
keysAndValuesDo: aBinaryBlock	Evaluate aBinaryBlock with each of the receiver's keys and the associated value as the *two* arguments.	
associationsDo: aBlock	Evaluate aBlock with each of the receiver's elements (key/value associations) as the argument.	

Sequenceable Collection

Methods for accessing SequenceableCollections

Message	Description	Notes
atAll: indexCollection	Answer a collection containing the elements of the receiver specified by the integer elements of the argument, indexCollection.	
atAll: aCollection **put:** anObject	Put anObject at every index specified by the integer elements of the argument, aCollection.	
atAllPut: anObject	Put anObject at every one of the receiver's indices.	
first	Answer the first element of the receiver. (Squeak also recognizes second, third, fourth, fifth and sixth). Signal an error if there aren't sufficient elements in the receiver.	
middle	Answer the median element of the receiver. Signal an error if the receiver is empty.	
last	Answer the last element of the receiver. Signal an error if the receiver is empty.	
allButFirst	Answer a collection equal to the receiver, but without the first element. Signal an error if the receiver is empty.	
allButLast	Answer a collection equal to the receiver, but without the last element. Signal an error if the receiver is empty.	
indexOf: anElement	Answer the index of anElement within the receiver. If the receiver does not contain anElement, answer 0.	
indexOf: anElement **ifAbsent:** exceptionBlock	Answer the index of anElement within the receiver. If the receiver does not contain anElement, answer the result of evaluating the argument, exceptionBlock.	
indexOfSubCollection: aSubCollection **startingAt:** anIndex	Answer the index of the receiver's first element, such that that element equals the first element of aSubCollection, and the next elements equal the rest of the elements of aSubCollection. Begin the search at element anIndex of the receiver. If no such match is found, answer 0.	
indexOfSubCollection: aSubCollection **startingAt:** anIndex **ifAbsent:** exceptionBlock	Answer the index of the receiver's first element, such that that element equals the first element of sub, and the next elements equal the rest of the elements of sub. Begin the search at element start of the receiver. If no such match is found, answer the result of evaluating argument, exceptionBlock.	
replaceFrom: start **to:** stop **with:** replacementCollection	This destructively replaces elements from start to stop in the receiver. Answer the receiver itself. Use **copyReplaceFrom:to:with:** for insertion/ deletion that may alter the size of the result.	
replaceFrom: start **to: stop: with:** replacementCollection **startingAt:** repStart	This destructively replaces elements from start to stop in the receiver starting at index, repStart, in the sequenceable collection, replacementCollection. Answer the receiver. No range checks are performed.	

Methods for copying SequenceableCollections

Message	Description	Notes
, otherCollection	Answer a new collection comprising the receiver concatenated with the argument, otherCollection.	
copyFrom: start **to:** stop	Answer a copy of a subset of the receiver that contains all the elements between index start and index stop, inclusive of both.	
copyReplaceAll: oldSubCollection **with:** newSubCollection	Answer a copy of the receiver in which all occurrences of oldSubstring have been replaced by newSubstring.	
copyReplaceFrom: start **to:** stop **with:** replacementCollection	Answer a copy of the receiver satisfying the following conditions: If stop is less than start, then this is an insertion; stop should be exactly start-1. start = 1 means insert before the first character, start = size+1 means append after last character. Otherwise, this is a replacement; start and stop have to be within the receiver's bounds.	
copyWith: newElement	Answer a copy of the receiver that is 1 bigger than the receiver and has newElement at the last element.	
copyWithout: oldElement	Answer a copy of the receiver from which all occurrences of oldElement have been left out.	
copyWithoutAll: aCollection	Answer a copy of the receiver from which all occurrences of all elements in aCollection have been removed.	
forceTo: length **paddingWith:** anElement	Answer a copy of the receiver with the specified length. If necessary, pad with anElement	
reversed	Answer a copy of the receiver in which the sequencing of all the elements has been reversed.	
shuffled	Answer a copy of the receiver in which the elements have been permuted randomly.	
sortBy: aBinaryBlock	Answer a copy that is sorted. Sort criteria is aBinaryBlock. When the block is true, the first arg goes first (so [:a :b | a > b] sorts in descending order).	

Methods for enumerating SequenceableCollections

Message	Description	Notes
findFirst: aBlock	Return the index of the receiver's first element for which aBlock evaluates as true.	
findLast: aBlock	Return the index of the receiver's last element for which aBlock evaluates as true.	
keysAndValuesDo: aBinaryBlock	Evaluate aBinaryBlock once with each valid index for the receiver in order, along with the corresponding value in the receiver for that index.	
reverseDo: aBlock	Evaluate aBlock with each of the receiver's elements as the argument, starting with the last element and taking each in sequence up to the first. For SequenceableCollections, this is the reverse of the enumeration for #do:.	
with: otherCollection **do:** binaryBlock	Evaluate binaryBlock with corresponding elements from this collection and otherCollection.	
reverseWith: otherCollection **do:** aBinaryBlock	Evaluate aBinaryBlock with each of the receiver's elements, in reverse order, along with the corresponding element, also in reverse order, from otherCollection.	

OrderedCollections

Methods for accessing OrderedCollections

Message	Description	Notes
add: newObject **before:** oldObject	Add the argument, newObject, as an element of the receiver. Put it in the sequence just preceding oldObject. Answer newObject.	
add: newObject **after:** oldObject	Add the argument, newObject, as an element of the receiver. Put it in the sequence just succeeding oldObject. Answer newObject.	
add: newObject **afterIndex:** index	Add the argument, newObject, as an element of the receiver. Put it in the sequence just after index. Answer newObject.	
addFirst: anElement	Add newObject to the beginning of the receiver. Answer newObject.	
addAllFirst: anOrderedCollection	Add each element of anOrderedCollection at the beginning of the receiver. Answer anOrderedCollection.	
addLast: anElement	Add newObject to the end of the receiver. Answer newObject.	
addAllLast: anOrderedCollection	Add each element of anOrderedCollection at the end of the receiver. Answer anOrderedCollection.	
removeAt: anIndex	remove the element of the receiver at location anIndex. Answers the element removed.	
removeFirst	Remove the first element of the receiver and answer it. If the receiver is empty, signal an Error.	
removeLast	Remove the last element of the receiver and answer it. If the receiver is empty, signal an Error.	

Strings

String is an extensive class, built over the ages in something of an *ad hoc* manner. We describe here only a small fraction of the methods provided (there are about 300!)

Methods for accessing Strings

Message	Description	Notes
findAnySubStr: delimiters **startingAt:** start	Answer the index of the character within the receiver, starting at start, that begins a substring matching one of the delimiters; delimiters is an Array of Strings and/or Characters. If the receiver does not contain any of the delimiters, answer size + 1.	
findBetweenSubStrs: delimiters	Answer the collection of tokens that results from parsing the receiver. And of the Strings (or Characters) in the Array delimiters is recognized as separating tokens.	
findDelimiters: delimiters **startingAt:** start	Answer the index of the character within the receiver, starting at start, that matches one of the delimiters. If the receiver does not contain any of the delimiters, answer size + 1.	
findString: subString	Answer the first index of subString within the receiver. If the receiver does not contain subString, answer 0.	
findString: subString **startingAt:** start	Answer the index of subString within the receiver, starting at start. If the receiver does not contain subString, answer 0.	
findTokens: delimiters	Answer the collection of tokens that results from parsing the receiver. Any character in the argument, delimiters, marks a border. Several delimiters in a row are considered as just one separator	
indexOf: aCharacter	Answer the index of the first occurrence of aCharacter in the receiver. 0 Otherwise.	
indexOf: aCharacter **startingAt:** start	Answer the index of the first occurrence of aCharacter in the receiver, beginning at index start. 0 Otherwise.	
indexOf: aCharacter **startingAt:** start **ifAbsent:** aBlock	Answer the index of the first occurrence of aCharacter in the receiver, beginning at index start. If not present, answer the value of aBlock.	
indexOfAnyOf: aCharacterSet	Answers the index of the first occurrence in the receiver of any character in the given set. Returns 0 if none is found.	1

Notes

1. As with #indexOf:, there are corresponding messages #indexOfAnyOf:ifAbsent:, #indexOfAnyOf:startingAt: and #indexOfAnyOf:startingAt:ifAbsent:)

Methods for comparing Strings

Message	Description	Notes
= aString	Answer whether the receiver is equal to aString.The comparison is case-sensitive,	
< aString, <= aString > aString >= aString	Answer whether the receiver sorts as indicated with aString. The collation order is that of the Squeak character set, and therefore case-sensitive,	
sameAs: aString	Answer whether the receiver is equal to aString, ignoring differences of case.	
compare: aString	Answer a code defining how the receiver sorts relative to the argument, aString. 1 - receiver before aString; 2 - receiver equal to aString; and 3 - receiver after aString. The collation sequence is that of the Squeak character set and is case insensitive.	
match: text	Answer whether text matches the pattern in the receiver. Matching ignores upper/lower case differences. Where the receiver contains #, text may contain any character. Where the receiver contains *, text may contain any sequence of characters.	
beginsWith: prefix	Answer whether the receiver begins with the argument, prefix.	
endsWith: prefix	Answer whether the receiver ends with the argument, prefix.	
alike: aString	Answer a non-negative integer indicating how similar the receiver is to aString. 0 means "not at all alike". The best score is `aString size * 2.`	

Methods for converting Strings

Message	Description	Notes
asLowercase	Answer a new String that matches the receiver but without any upper case characters.	
asUppercase	Answer a new String that matches the receiver but without any lower case characters.	
capitalized	Answer a copy of the receiver with the first character capitalized if it is a letter.	
asDisplayText	Answer a copy of the receiver with default font and style information.	
asInteger	Attempts to parse the receiver as an Integer. Answers the Integer, or nil if the receiver does not start with a digit.	
asNumber	Attempts to parse the receiver as a Number. It is an error if the receiver does not start with a digit.	
asDate	Attempts to parse the receiver as a date, and answers an appropriate instance of class Date. Many formats are recognized.	

Streaming Classes and Methods

The Stream Hierarchy

Class	Description
Stream	Abstract Class for Accessors
PositionableStream	Accessors for Collections Indexable by an Integer
ReadStream	Read-Only
WriteStream	Write-Only
ReadWriteStream	Read and/or Write
FileStream	Accessors for collections whose elements are "paged in"
StandardFileStream	Accessors for files accessed from a file system
CrLfFileStream	Automatically handles system-specific line endings
DummyStream	Like /dev/null

Class Stream

Stream is an abstract class for an accessor to a sequence of objects, called the contents. The stream is said to be "advanced" when the stream is configured to access a later element of the contents.

Methods for accessing Streams

Message	Description	Notes
contents	Answer the entire contents of the receiver.	
next	Answer the next object accessible by the receiver.	
next: anInteger	Answer the next anInteger number of objects accessible by the receiver.	
next: n **put:** anObject	Make the next n objects accessible by the receiver anObject. Answer anObject.	
nextMatchAll: aColl	Answer true if next N objects are the ones in aColl, else false. Advance stream if true, leave as was if false.	
nextMatchFor: anObject	Answer whether the next object is equal to the argument, anObject, advancing the stream.	
nextPut: anObject	Insert the argument, anObject, as the next object accessible by the receiver. Answer anObject.	
nextPutAll: aCollection	Append the elements of aCollection to the sequence of objects accessible by the receiver. Answer aCollection.	
upToEnd	Answer the remaining elements in the string	
flush	Ensure that any objects buffered in the receiver are sent to their final destination.	

Methods for testing Streams

Message	Description	Notes
atEnd	Answer whether the receiver can access any more objects.	

Methods for enumerating Streams

Message	Description	Notes
do: aBlock	Evaluate aBlock for each of the remaining objects accessible by receiver.	

Class PositionableStream

PositionableStream is an abstract class for accessors to sequences of objects that can be externally named by indices so that the point of access can be repositioned. Concrete classes ReadStream, WriteStream and ReadWriteStream are typically used to instantiate a PositionableStream on Collections, depending upon the access mode. StandardFileStream and CRLFFileStream are typically used for instantiating PositionableStreams for Files.

Methods for accessing PositionableStreams

Message	Description	Notes
contentsOfEntireFile	Answer a collection containing the remainder of the receiver.	
last	Return the final element of the receiver.	
nextDelimited: terminator	Answer the contents of the receiver, from the current position up to the next terminator character; provided, however, that doubled terminators will be included as a single element.	
nextInto: buffer	Given buffer, an indexable object of size n, fill buffer with the next n objects of the receiver.	
nextLine	Answer next line (may be empty), or nil if at end	
originalContents	Answer the receiver's actual contents collection. (**contents** returns a copy)	
peek	Answer what would be returned if the message **next** were sent to the receiver, but don't advance the receiver. If the receiver is at the end, answer nil.	
peekFor: anObject	Answer false and do not move over the next element if it is not equal to anObject, or if the receiver is at the end. Answer true and advance the stream if the next element is equal to anObject.	
upTo: anObject	Answer a subcollection from the current access position to the occurrence (if any, but not inclusive) of anObject in the receiver. If anObject is not in the collection, answer the entire rest of the receiver.	
upToAll: aCollection	Answer a subcollection from the current access position to the occurrence (if any, but not inclusive) of aCollection. If aCollection is not in the stream, answer the entire rest of the stream.	

Methods for testing PositionableStreams

Message	Description	Notes
isEmpty	Answer whether the receiver's contents has no elements.	

Methods for positioning PositionableStreams

Message	Description	Notes
match: subCollection	Set the access position of the receiver to be past the next occurrence of the subCollection. Answer whether subCollection is found. No wildcards, case sensitive.	
padTo: nBytes **put:** aCharacter	Pad, using aCharacter, to the next boundary of nBytes.	
padToNextLongPut: char	Make position be on long word boundary, writing the padding character, char, if necessary.	
position	Answer the current position of accessing the sequence of objects.	
position: anInteger	Set the current position for accessing the objects to be anInteger, as long as anInteger is within the bounds of the receiver's contents. If it is not, create an error notification.	
reset	Set the receiver's position to the beginning of the sequence of objects.	
resetContents	Set the position and limits to 0.	
setToEnd	Set the position of the receiver to the end of the sequence of objects.	
skip: anInteger	Set the receiver's position to be the current position+anInteger. A subclass might choose to be more helpful and select the minimum of the receiver's size and position+anInteger, or the maximum of 1 and position+anInteger for the repositioning.	
skipTo: anObject	Set the access position of the receiver to be past the next occurrence of anObject. Answer whether anObject is found.	

Class WriteStream

Methods for writing characters on WriteStreams

Message	Description	Notes
cr	Append a return character to the receiver.	
crtab	Append a return character, followed by a single tab character, to the receiver.	
crtab: anInteger	Append a return character, followed by anInteger tab characters, to the receiver.	
space	Append a space character to the receiver.	
tab	Append a tab character to the receiver.	

ANSI-Compatible Exceptions

Evaluating Blocks with Exceptions

Methods for handling Exceptions raised in a BlockContext

Message	Description	Notes
ensure: aTerminationBlock	Evaluate aTerminationBlock after evaluating the receiver, regardless of whether the receiver's evaluation completes.	
ifCurtailed: aTerminationBlock	Evaluate the receiver. If it terminates abnormally, evaluate aTerminationBlock.	
on: exception **do:** handlerActionBlock	Evaluate the receiver in the scope of an exception handler, handlerActionBlock.	

Examples

```
["target code, which may abort"]
    ensure:
        ["code that will always be executed
            after the target code,
            whatever whatever may happen"]

["target code, which may abort"]
    ifCurtailed:
        ["code that will be executed
            whenever the target code terminates
            without a normal return"]

["target code, which may abort"]
    on: Exception
    do: [:exception |
            "code that will be executed whenever
            the identified Exception is signaled."]
```

Exceptions

Exception is an abstract class; instances should neither be created nor trapped. There are two common subclasses of Exception, Error and Notification, from which subclasses normally inherit. Errors are not resumable; a Notification is an indication that something interesting has occurred; if it is not handled, it will pass by without effect.

Exceptions play two distinct roles: that of the exception, and that of the exception handler.

Methods for describing Exceptions

Message	Description	Notes
defaultAction	The default action taken if the exception is signaled.	
description	Return a textual description of the exception.	
isResumable	Determine whether an exception is resumable.	
messageText	Return an exception's message text.	
tag	Return an exception's tag value.	

Methods for signalling Exceptions

Message	Description	Notes
signal	Signal the occurrence of an exceptional condition.	
signal: signalerText	Signal the occurrence of an exceptional condition with a specified textual description.	

Methods for dealing with a signaled Exception

Message	Description	Notes
isNested	Determine whether the current exception handler is within the scope of another handler for the same exception.	
outer	Evaluate the enclosing exception action for the receiver and return.	
pass	Yield control to the enclosing exception action for the receiver.	
resignalAs: replacementException	Signal an alternative exception in place of the receiver.	
resume	Return from the message that signaled the receiver.	
resume: resumptionValue	Return the argument as the value of the message that signaled the receiver.	
retry	Abort an exception handler and re-evaluate its protected block.	
retryUsing: alternativeBlock	Abort an exception handler and evaluate a new block in place of the handler's protected block.	
return	Return nil as the value of the block protected by the active exception handler.	
return: returnValue	Return the argument as the value of the block protected by the active exception handler.	

Class ExceptionSet

An ExceptionSet is used to specify a set of exceptions for an exception handler.

Creating ExceptionSet

Message	Description	Notes
, anException	Receiver may be an Exception class or an ExceptionSet. Answers an exception set that contains the receiver and anException.	

Example

```
["target code, which may abort"]
    on: Exception, Error, ZeroDivide
    do:
        [:exception |
            "code that will be executed whenever
            one of the identified Exceptions is
            signaled."]
```

The Squeak Class Hierarchy

In Smalltalk, "everything is an object." That is, everything is an instance of class Object or an instance of some subclass of class Object. Everything. Numbers, Classes, Metaclasses, everything. I refer to this as the "Object rule."

Actually, Squeak bends this rule a little bit; the Object rule does not apply for certain system objects, which derive from class ProtoObject. Nevertheless, except for these few system objects, the vast majority of Squeak objects, which I call, "proper objects," satisfy the Object Rule. Proper Objects and their classes and metaclasses, satisfy the following properties.

The Laws of Proper (Smalltalk) Classes

- Every proper class is a subclass of class Object, except for Object itself, which has no proper superclass. In particular, Class is a subclass of ClassDescription, which is a subclass of Behavior which is a subclass of Object.
- Every object is an instance of a class.
- Every class is an instance of a metaclass.
- All metaclasses are (ultimately) subclasses of Class.
- Every metaclass is an instance of MetaClass.
- The methods of Class and its superclasses support the behavior common to those objects that are classes.
- The methods of instances of MetaClass add the behavior specific to particular classes.

Class ProtoObject

Squeak additionally supports an improper class ProtoObject, from which object hierarchies other than proper instances and proper classes can inherit. ProtoObject is the superclass of class Object and has no instances. Presently, there are two subclasses of ProtoObject besides Object: ObjectOut and ImageSegmentRootStub, both of which are used to do magic involving objects that have been moved out of memory onto an external medium. You might need to subclass ProtoObject if you are doing something like implementing a remote message send system where you have proxies for remote objects (those on another computer).

However, as with proper classes, ProtoObject, is an instance of a metaclass, ProtoObject class, which in turn is an instance of class MetaClass.

Categories of Squeak Classes

This quick reference only scratches the surface of the functionality available through Squeak. To assist the beginner in surveying the system, the following outline of the major Squeak *packages* is provided.

Category	Description
Kernel	Primary Smalltalk classes for creating and manipulating Smalltalk objects, the Object hierarchy, coroutines and parallel processes. Subcategories include: Objects, Classes, Methods and Processes.
Numeric	Classes for numeric operations, including date and time operations. Subcategories include Magnitudes and Numbers
Collections	Classes for aggregations of Smalltalk objects.
Graphics	Core classes for Smalltalk graphic objects as well as facilities and applications for operating on graphic objects. Key classes include Form and BitBlt.
Interface	The "traditional" MVC User Interface Framework. Also found here are a number of useful Smalltalk applications, including: Squeak browsers, a mail client, a web browser, IRC chat client and facilities for operating on "projects."
Morphic	Squeak's Morphic User Interface Framework
Music	Classes supporting Squeak's Sound Synthesis capabilities. Also found here are several useful facilities and applications for manipulating MIDI data and other representations of musical scores.
System	Key System Facilities. Subclasses include: Compiler (Smalltalk compiler); Object Storage (virtual memory for Smalltalk objects); File facilities; Compression; Serial Data Transmission; Basic network facilities.
Exceptions	Class supporting Squeak's ANSI-compliant exceptions facilities.
Network	Classes implementing various Internet and Squeak related network protocols.
PluggableWebServer	A complete web-server application, including an implementation of Swiki, a collaborative world-wide-web environment. Key classes include: PWS
HTML	Classes for manipulating HTML data.
Squeak	Here lives the mouse. Key classes include: the Squeak VM and an interpreter; the Squeak Smalltalk Subset (Slang) to C translator; and facilities for developing native plugins (pluggable primitives).
Balloon	Classes for complex 2-D graphic objects and fast 2-D graphics rendering.
Balloon-3D	Classes for complex 3-D graphics objects and fast 3-D graphics rendering.
TrueType	Classes for manipulating and displaying TrueType data.
MM-Flash	Classes for manipulating and displaying Flash file data.
Alice & Wonderland	A remarkable interactive 3-D graphics environment.