Preliminaries

Typesetting

- *Italic UpperCamelCase* is used for denoting class names. Example: *Object*
- *Italic lowerCamelCase* is used for denoting method names: Example: *method*
- **Bold lowercase** is used for denoting reserved constants. Examples: **nil**, **true**, **false**

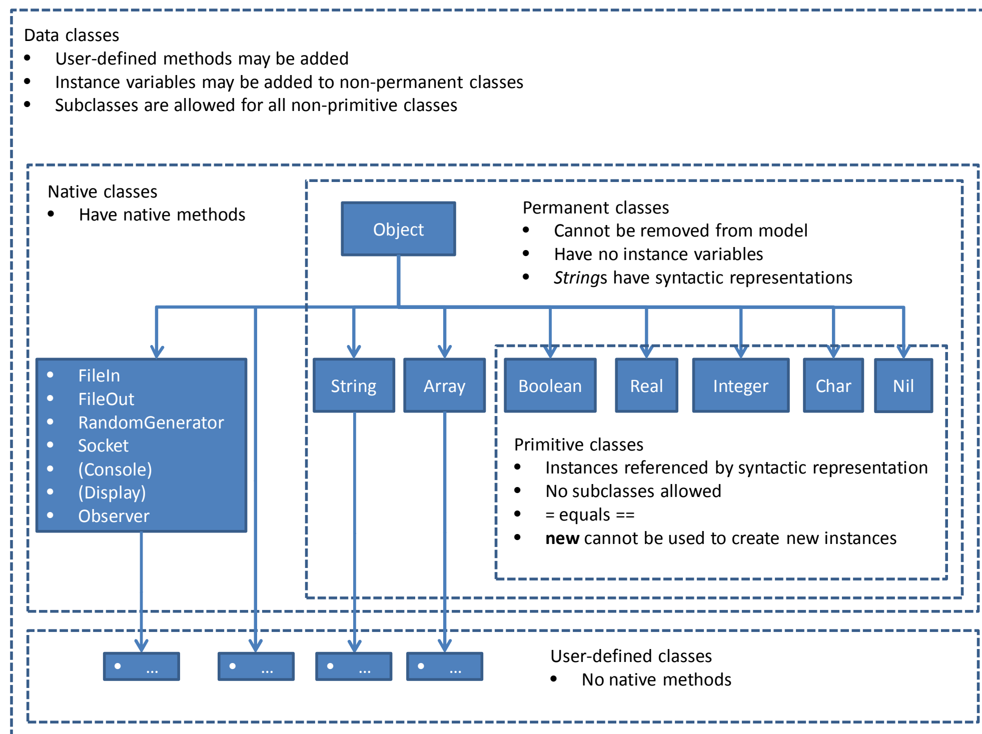

Figure 1. Overview of Basic Classes.

Overview of Basic Classes

Four types of data classes are distinguished (see Figure 1):

1. *Native classes* refer to data classes for instances and/or methods are directly (natively) implemented in the implementation language of an execution engine.
2. *Permanent classes* are native classes that must be part of any POOSL model, because they are used in native methods of primitive classes (see below). Permanent classes do not have instance variables. For example, class *Object* cannot have instance variables, because they would be inherited by primitive classes, which is not possible. Subclasses of permanent classes can be defined by the user to create extensions with instance variables. Class *String* is a permanent class that has syntactic representations of its instances (see below).
3. *Primitive classes* are permanent classes with a fixed (possibly infinite) collection of instances, that have syntactic representations of their instances and whose behavior is defined in the formal semantics of the language[1]. Primitive classes cannot have subclasses. Instances of primitive classes are called *primitive objects*. Primitive objects do not have state, in particular they have no references to other objects.

[1] Note that this documents updates methods of primitive classes that not explicitly defined in the latest formal semantics, but their interpretation is straightforward.

4. User-defined class is a term used for all non-native classes.

User-defined methods can be added for any data class, while user-defined instance variables are not allowed for permanent classes. Subclasses are not allowed for primitive classes. Instances of data classes can be created using **new**, except in case of primitive classes. For primitive classes and *Strings*, the syntactic representation allows identifying, respectively creating new instances.

Constants:

- Constant **nil** is the single instance of class *Nil*
- Constants **true** and **false** are the two only instances of class *Boolean*
- *Integer* constants can be expressed in the following syntax. There is no a priori upper bound or lower bound on the value of an integer constant (in particular, there is no fixed number of bits to represent an integer)

```
<Integer>             ::=  ["-"] <DecimalDigit>+ [("e" | "E") ["+"] <DecimalDigit>+]
                           | "0b" <BinaryDigit>+
                           | "0o" <OctalDigit>+
                           | "0x" <HexadecimalDigit>+
<BinaryDigit>         ::=  "0" | "1"
<OctalDigit>          ::=  <BinaryDigit> | "2" | "3" | "4" | "5" | "6" | "7"
<DecimalDigit>        ::=  <OctalDigit> | "8" | "9"
<HexadecimalDigit>    ::=  <DecimalDigit> | "a" | "b" | "c" | "d" | "e" | "f"
                           | "A" | "B" | "C" | "D" | "E" | "F"
```

- *Real* valued constants can be expressed in the following syntax:

```
<Real>  ::=  ["-"] <DecimalDigit>+ "." <DecimalDigit>+ [("e" | "E") ["+" | "-"]
             <DecimalDigit>+]
```

- The syntactic representation of a character, an instance of class *Char*, begins and ends with a single quote character ‘. The precise syntax is detailed in Appendix B. In this document, we refer to the following classes of special characters and character sequences:
  - White space characters are: Space(32), TAB(9), CR (13), LF(10), VT(11), FF(12)
  - New line character sequences are: CR (13), LF(10), and CR(13) followed by LF(10)
- The syntactic representation of a *String* begins and ends with a double quote character “. The precise syntax is detailed in Appendix B.

Types and classes:

- A type C associated with class C consists of **nil** and all instances of class C and all its subclasses.

Description of methods:

- In all method descriptions in the remainder of this document, the receiver refers to the object on which the discussed method is called.

Permanent Classes without Constants

Object

This class represents the root superclass of any other data class. It therefore provides generic methods that are applicable to all data objects.

!=(o: Object): Boolean
Returns **true** in case o is not equal to the receiver and **false** otherwise. The result is equivalent to (receiver = o) not. It has a special syntax of the form o1 != o2, where o1 acts as the receiver and o2 as the argument

!==(o: Object): Boolean
Returns **true** in case o is not identical to the receiver and **false** otherwise. The result is equivalent to: (**receiver** == o) not. It has a special syntax of the form o1 !== o2, where o1 acts as the receiver and o2 as the argument.

=(o: Object): Boolean
Implements the equality relation. For primitive objects, = is equivalent to ==. For user-defined classes, it returns **true** if o is an object of the same class as the receiver and all instance variables of o and the receiver are (recursively) equal as well. Otherwise, it returns **false**. For non-primitive native classes, the behavior is as defined for the user-defined classes, unless specified differently at the description of the class. It has a special syntax of the form o1 = o2, where o1 acts as the receiver and o2 as the argument.

==(o: Object): Boolean
Implements the identity relation. Returns **true** in case o refers to the same object as the receiver. Otherwise, it returns **false**. It has a special syntax of the form o1 == o2, where o1 acts as the receiver and o2 as the argument.

deepCopy: Object
For user-defined classes not extending non-permanent native classes, it returns a clone of the receiver as a new object. This means that a new instance of the receiver's class is returned, where each instance variable is assigned to clones in a recursive manner, such that all (indirect) references to the same original object result in references to a single cloned object. The original and cloned data object structures are isomorphic. For primitive classes, the receiver itself is returned. The non-permanent native classes (and any user-defined subclasses) do not support copying and therefore, an error is generated. For non-primitive permanent classes, the behavior is as defined for the user-defined classes, unless specified differently at the description of the class.

error(s: String): Object
This method allows to signal erroneous behavior. Semantically it does nothing, but tools tend to halt execution of a model after this method and show message s to the user.

assert(b: Boolean, s: String): Object
This method allows to signal erroneous behavior for a condition *b*. Semantically it does nothing, but tools tend to halt execution of a model after this method when expression b evaluates to **false** and show message s to the user.

printString: String
Returns a *String* representation of the receiver. It is the typical means used by tools to retrieve a representation of an object to users of those tools. The standard behavior in class *Object* returns just the class name. This is overridden in subclasses to display more specific information on the object. If the receiver is an instance of a primitive class or class *String*, the syntactic constant representation of the receiver is returned.

shallowCopy: Object
For user-defined classes not extending non-permanent native classes, it returns a shallow copy of the receiver. This means that a new object of the receiver's class is created, for which the instance variables refer to the same objects as the corresponding instance variables of the receiver. For primitive classes, the receiver itself is returned. For non-permanent native classes (and any user-defined subclasses), an error is generated. For non-primitive permanent classes, the behavior is as defined for the user-defined classes, unless specified differently at the description of the class.

isOfType(s: String): Boolean
If *s* does not refer to the name of an existing class, an error is generated. In case *s* does refer to the name of an existing class, **true** is returned in case the receiver is of the type associated with the class with name *s* and **false** otherwise.

Array

This class extends *Object*. It represents an indexed list of (arbitrary typed) objects. Creating a new *Array* yields an indexed list of size 0 (empty *Array*).

=(o: Object): Boolean
Returns **true** in case o is an *Array* of the same size as the receiver and for each index, the objects both *Array*s refer to are equal (in terms of =). Otherwise, it returns **false**.

deepCopy: Object
Returns a new *Array* object with the same size as the receiver and at each index, a recursive *deepCopy* of the object referred to by the receiver at that index.

shallowCopy: Object
Returns a new *Array* object with the same size as the receiver and at each index, it refers to the same object referred to by the receiver at that index.

printString: String
Returns a *String* equal to “Empty Array” in case the receiver has size 0. Otherwise, it consists of *String* “Array(“ followed by a comma separated list of *String*s, the result s of calling printString on the objects from index 1 to the size of the receiver, followed by “)”.

at(i: Integer): Object
Returns the object located at index i in case i ranges between 1 and the size the receiver. Otherwise, an index out-of-bounds error is generated.

size: Integer
Returns the size of the receiver.

putAt(i: Integer, o: Object): Array

Replaces the object at index i with o in case i ranges between 1 and the size of the receiver. Otherwise, an index out-of-bounds error is generated. It returns the receiver.

putAll(o: Object): Array

Makes all indices in the receiver refer to object o (without making copies). It returns the receiver.

resize(i: Integer): Array

Modifies the size of the receiver to i (in case i >= 0). In case i < 0, an error is generated. When i is larger than the original size of the receiver, all new locations are filled with **nil.** When i is smaller than the original size of the receiver, the objects at indices between i+1 and the original size will no longer be contained. It returns the receiver.

+(a: Array): Array

Returns a new *Array* consisting of a copy of the receiver that has the size of the receiver plus the size of a, where the indices between 1 and the size of the receiver are filled with the objects in the receiver (in the same order) and the indices between the size of the receiver + 1 and the size of the returned *Array* contain the objects in a (in the same order). It has a special syntax of the form a1 + a2, where a1 acts as the receiver and a2 as the argument.

concat(a: Array): Array

Modifies the receiver by increasing its size with the size of a, where the indices between the size of the receiver + 1 and the size of the returned *Array* contain the objects in a (in the same order).

find(i: Integer, o: Object): Integer

This method searches the receiver for object o, starting from index i. If an object equal to o is found, the index (between 1 and the size of the receiver) at which o is located is returned. In case i is smaller than 1 or larger than the size of the receiver, an index out-of-bounds error is generated. In all other cases, it returns 0.

subArray(i, l: Integer): Array

Returns a new *Array* of size l containing a copy of the objects in the receiver starting at index i in case i is between 1 and the size of the receiver, l is non-negative and i + l-1 is at most equal to the size of the receiver. Otherwise, an index out-of-bounds error is generated.

Permanent Classes with Constants

String

This class extends *Object*. It represents the class of strings (of arbitrary size).

=(o: Object): Boolean
Returns **true** in case o refers to a *String* identical to the receiver. Otherwise, it returns **false**. It has a special syntax of the form o1 = o2, where o1 acts as the receiver and o2 as the argument.

deepCopy: Object
Returns a new *String*, identical to the receiver.

shallowCopy: Object
Returns a new *String*, identical to the receiver.

+(s: String): String
Returns the concatenation of the receiver and s (as a new *String*). It has a special syntax of the form s1 + s2, where s1 acts as the receiver and s2 as the argument.

concat(s: String): String
Modifies the receiver by concatenation with s. It returns the receiver.

cr: String
Modifies the receiver by concatenation with a carriage return character CR(13). It returns the receiver.

lf: String
Modifies the receiver by concatenation with a line feed character LF(10). It returns the receiver.

tab: String
Modifies the receiver by concatenation with a tab character HT(9). It returns the receiver.

find(i: Integer, s: String): Integer
This method searches the receiver for a substring s, starting from index i. If pattern s is found, the index (between 1 and the size of the receiver) at which s starts is returned. In case i is smaller than 1 or larger than the size of the receiver, an index out-of-bounds error is generated. In all other cases, it returns 0.

at(i: Integer): Char
Returns the character at index i in case i ranges between 1 and the size of the receiver. Otherwise, an index out-of-bounds error is generated.

size: Integer
Returns the number of characters constituting the receiver.

putAt(i: Integer, c: Char): String
Modifies the receiver by replacing the character at index i with c in case i ranges between 1 and the size of the receiver. Otherwise, an index out-of-bounds error is generated. It returns the receiver.

subString(i, l: Integer): String
Returns a new *String* containing a copy of the substring with size l, starting at index i in case i is

between 1 and the size of the receiver, l is non-negative and i + l-1 is at most equal to the size of the receiver. Otherwise, an index out-of-bounds error is generated.

unmarshal: Object
This method reconstructs an *Object* from a standardized *String* representation as created by the method *marshal* of class *Object*. If the receiver does not conform to the syntax and static semantics as described in Appendix A, an error occurs.

splitOn(c: Char): Array
Returns an *Array* of *String* objects, constructed by splitting the receiver into substrings at characters c. The new *String* objects in the returned *Array* do not contain character c. Notice that in case the receiver contains a sequence of characters c, the returned *Array* will contain empty *String*s. In case c is not included in the receiver, the returned *Array* solely contains a copy of the receiver.

splitOnWhiteSpace: Array
Returns an *Array* of *String* objects, constructed by splitting the receiver into substrings delimited by one or more white space characters. The new *String* objects in the returned *Array* do not contain any white space characters. White space characters at the beginning and end of the receiver are ignored and if the receiver consists of white space characters only, an empty *Array* is returned. In case the receiver does not contain any white-space characters, the returned *Array* solely contains a copy of the receiver.

isBoolean: Boolean
Returns **true** in case the receiver is the *String* representation of a *Boolean* object and **false** otherwise. No extra white space or other characters are allowed.

isChar: Boolean
Returns **true** in case the receiver is the *String* representation of a *Char* object and **false** otherwise. The character must include surrounding single quotes and may use escape characters. No extra white space or other characters are allowed. See preliminaries for the syntax.

isReal: Boolean
Returns **true** in case the receiver is the *String* representation of a *Real* object and **false** otherwise. No extra white space or other characters are allowed. See preliminaries for the syntax.

isInteger: Boolean
Returns **true** in case the receiver is the *String* representation of an *Integer* object and **false** otherwise. No extra white space or other characters are allowed. See preliminaries for the syntax.

toBoolean: Boolean
If the receiver is the *String* representation of a *Boolean* object (in line with the *isBoolean* method), this object is returned. Otherwise, **nil** is returned.

toChar: Char
If the receiver is the *String* representation of a *Char* object (in line with the *isChar* method), this object is returned. Otherwise, **nil** is returned.

toReal: Real
If the receiver is the *String* representation of a *Real* object (in line with the *isReal* method), this object is returned. Otherwise, **nil** is returned.

toInteger: Integer

If the receiver is the *String* representation of an *Integer* object (in line with the *isInteger* method), this object is returned. Otherwise, **nil** is returned.

<(s: String): Boolean

Returns **true** in case the receiver is lexicographically ordered before s and **false** otherwise. It has a special syntax of the form s1 < s2, where s1 acts as the receiver and s2 as the argument.

<=(s: String): Boolean

Returns **true** in case the receiver is lexicographically before or is equal to s and **false** otherwise. It has a special syntax of the form s1 <= s2, where s1 acts as the receiver and s2 as the argument.

>(s: String): Boolean

Returns **true** in case the receiver is lexicographically ordered after s and **false** otherwise. It has a special syntax of the form s1 > s2, where s1 acts as the receiver and s2 as the argument.

>=(s: String): Boolean

Returns **true** in case the receiver is lexicographically after or is equal to s and **false** otherwise. It has a special syntax of the form s1 >= s2, where s1 acts as the receiver and s2 as the argument.

Primitive Classes

Nil

This class extends *Object*. It represents the class of instance **nil**.

Boolean

This class extends *Object*. It represents the class of Booleans. This class has two instances represented with the constants ***true*** and ***false***.

&(b: Boolean): Boolean
Returns the logical and of the receiver and b. It has a special syntax of the form b1 & b2, where b1 acts as the receiver and b2 as the argument.

|(b: Boolean): Boolean
Returns the logical or of the receiver and b. It has a special syntax of the form b1 | b2, where b1 acts as the receiver and b2 as the argument.

not: Boolean
Returns the logical inverse of the receiver. In addition to the normal syntax, it also has a special syntax of the form ! b, where b acts as the receiver.

xor(b: Boolean): Boolean
Returns the logical xor of the receiver and b. It is equivalent to (receiver!= B).

Char

This class extends *Object*. It represents the class of individual Extended ASCII characters and therefore has 256 instances representing each of the extended ASCII characters.

asciiIndex: Integer
Returns the ASCII index number of the receiver.

asString: String
Returns a new *String* consisting of the single receiver character.

Integer

This class extends *Object*. It represents the class of (*unbounded*) integer numbers.

-: Integer
Returns the negation of the receiver. It has a special syntax of the form -i, where i acts as the receiver

-(i: Integer): Integer
Subtracts i from the receiver and returns the result. It has a special syntax of the form i1 - i2, where i1 acts as the receiver and i2 as the argument.

****(i: Integer): Integer***
Returns the product of the receiver and i. It has a special syntax of the form i1 * i2, where i1 acts as the receiver and i2 as the argument.

&(i: Integer): Integer
Returns an *Integer* representing the bit-wise and of the two's-complement of the receiver with i. It has a special syntax of the form i1 & i2, where i1 acts as the receiver and i2 as the argument.

+(i: Integer): Integer
Returns the sum of the receiver with i. It has a special syntax of the form i1 + i2, where i1 acts as the receiver and i2 as the argument.

<(i: Integer): Boolean
Returns **true** in case the receiver is smaller than i and **false** otherwise. It has a special syntax of the form i1 < i2, where i1 acts as the receiver and i2 as the argument.

<=(i: Integer): Boolean
Returns **true** in case the receiver is smaller than or equal to i and **false** otherwise. It has a special syntax of the form i1 <= i2, where i1 acts as the receiver and i2 as the argument.

>(i: Integer): Boolean
Returns **true** in case the receiver is greater than i and **false** otherwise. It has a special syntax of the form i1 > i2, where i1 acts as the receiver and i2 as the argument.

>=(i: Integer): Boolean
Returns **true** in case the receiver is greater than or equal to i and **false** otherwise. It has a special syntax of the form i1 >= i2, where i1 acts as the receiver and i2 as the argument.

|(i: Integer): Integer
Returns an *Integer* representing the bit-wise or of the two's-complement of the receiver with i. It has a special syntax of the form i1 | i2, where i1 acts as the receiver and i2 as the argument.

abs: Integer
Returns the absolute value of the receiver.

asAsciiChar: Char
Returns the character by using the receiver as its ASCII index number in case the receiver ranges between 0 and 255. Otherwise, an index out-of-bound error is generated.

asReal: Real
Returns a *Real* object with the same value as the receiver.

div(i: Integer): Integer
Returns the *Integer* A such that A*i + B equals the receiver for some B and 0 <= B < i if i > 0 and i < B <= 0 if i < 0.

/(i: Integer): Integer
This method is identical to the method *div*. Returns the *Integer* A such that A*i + B equals the receiver for some B and 0 <= B < i if i > 0 and i < B <= 0 if i < 0.

fac: Integer
Returns the factorial of the receiver in case the receiver is non-negative. Otherwise, an error is generated. (Notice that 0! = 1).

modulo(i: Integer): Integer
Returns the *Integer* B such that A * i + B equals the receiver for some *Integer* A and 0 <= B < i if i > 0 and i < B <= 0 if i < 0.

monus(i: Integer): Integer
Returns the difference of the receiver with i if the receiver > i or 0 otherwise.

not: Integer
Returns the bit-wise negation of the receiver. It is equivalent to: -receiver - 1.

power(i: Integer): Integer
Returns the receiver raised to the power of i in case i is non-negative. Otherwise, an error is generated.

sqr: Integer
Returns the square of the receiver.

max(i: Integer): Integer
Returns the maximum of the receiver and i.

min(i: Integer): Integer
Returns the minimum of the receiver and i.

xor(i: Integer): Integer
Returns the bit-wise xor of the receiver and i.

Real

This class extends *Object*. Its instances represents real numbers using IEEE 754-2008 floating point representations. Note that the arithmetic operators below operate as specified by this standard. Whenever the standard specifies exception occur, errors will be given.

-: Real
Returns the negation of the receiver. It has a special syntax of the form -r, where r acts as the receiver.

-(r: Real): Real
Subtracts r from the receiver and returns the result. It has a special syntax of the form r1 - r2, where r1 acts as the receiver and r2 as the argument.

****(r: Real): Real***
Returns the product of the receiver and r. It has a special syntax of the form r1 * r2, where r1 acts as the receiver and r2 as the argument.

/(r: Real): Real
Returns the quotient of the receiver with r. It has a special syntax of the form r1 / r2, where r1 acts as the receiver and r2 as the argument.

+(r: Real): Real
Returns the sum of the receiver with r. It has a special syntax of the form r1 + r2, where r1 acts as the receiver and r2 as the argument.

<(r:Real): Boolean

Returns **true** in case the receiver is smaller than r and **false** otherwise. It has a special syntax of the form r1 < r2, where r1 acts as the receiver and r2 as the argument.

<=(r: Real): Boolean

Returns **true** in case the receiver is smaller than or equal to r and **false** otherwise. It has a special syntax of the form r1 <= r2, where r1 acts as the receiver and r2 as the argument.

>(r: Real): Boolean

Returns **true** in case the receiver is greater than r and **false** otherwise. It has a special syntax of the form r1 > r2, where r1 acts as the receiver and r2 as the argument.

>=(r: Real): Boolean

Returns **true** in case the receiver is greater than or equal to r and **false** otherwise. It has a special syntax of the form r1 >= r2, where r1 acts as the receiver and r2 as the argument.

abs: Real

Returns the absolute value of the receiver.

acos: Real

Returns the arccosine of the receiver if the receiver is in [-1.0, 1.0]. Otherwise, an error is generated.

asin: Real

Returns the arcsine of the receiver if the receiver is in [-1.0, 1.0]. Otherwise, an error is generated.

asInteger: Integer

Returns an *Integer* representation of the receiver denoting the integer number closest to the receiver. Rounding is as follows: for positive numbers: x rounds to $\left\lfloor x + \frac{1}{2} \right\rfloor$ for negative numbers x rounds to $\left\lceil x - \frac{1}{2} \right\rceil$.

atan: Real

Returns the arctangent of the receiver.

atan2(r: Real): Real

Returns the angle in radians between the vector (receiver ,r) and the vector (1,0).

ceiling: Real

Returns the smallest rounded *Real* that is not smaller than the receiver.

cos: Real

Returns the cosine of the receiver (as an angle in radians).

exp: Real

Returns e (the base of the natural logarithm) to the power of the receiver.

floor: Real

Returns the largest rounded Real that is not larger than the receiver.

ln: Real

Returns the natural logarithm of the receiver if the receiver is positive. Otherwise, an error is generated.

log: Real
Returns the 10-based logarithm of the receiver if the receiver is positive. Otherwise, an error is generated.

monus(r: Real): Real
Returns the difference of the receiver with r if the receiver > r or 0 otherwise.

power(r: Real): Real
Returns the receiver raised to the power of r.

round: Real
Returns the rounded *Real* closest to the receiver (as an *Integer*). Rounding is as follows: for positive numbers: x rounds to $\left\lfloor x + \frac{1}{2} \right\rfloor$ for negative numbers x rounds to $\left\lceil x - \frac{1}{2} \right\rceil$.

sin: Real
Returns the sine of the receiver (as an angle in radians).

sqr: Real
Returns the square of the receiver.

sqrt: Real
Returns the square root of the receiver in case the receiver is non-negative. Otherwise, an error is generated.

tan: Real
Returns the tangent of the receiver (as an angle in radians).

max(r: Real): Real
Returns the maximum of the receiver and r.

min(r: Real): Real
Returns the minimum of the receiver and r.

Native Non-Permanent Classes

RandomGenerator

This class extends *Object*. It represents a generator of pseudo-random values with a uniform distribution U[0,1).

random: Real
Returns a *Real* sample from distribution U[0,1).

randomInt(i: Integer): Integer
Returns an *Integer* sample from discrete uniform distribution [0, i-1] for i > 0. In case i <= 0, an error is generated.

randomiseSeed: RandomGenerator
This method arbitrarily modifies the seed for the sequence of pseudo-random numbers successively produced by calling methods *random* and *randomInt*. The exact behavior is implementation dependent, typically setting the seed to a time-dependent value. Note that when the *randomiseSeed* or seed methods are not used, every instance of this class will produce the same sequence of pseudo random numbers. Using *randomiseSeed* disables exact reproductions of executions. It returns the receiver.

seed(i: Integer): RandomGenerator
Sets the seed of the receiver to i. It returns the receiver.

FileIn

This class extends *Object*. It provides a means to read information from files. Creating a new *FileIn* yields an object without referring to any concrete file.

source(s: String): FileIn
Specifies the file to read information from. s is a file name (possibly with an absolute or relative path reference – where the syntactic symbols / and \ for path references can be used interchangeably independent of OS). It returns the receiver.

open: FileIn
Locks the referred file for read access. It assumes that method *source* has previously been called to identify the concrete file to refer to. Otherwise, an error is generated. It returns the receiver.

atEndOfFile: Boolean
Returns **true** in case the read pointer in the file points at the end of the file and **false** otherwise.

close: FileIn
This method and releases the referred file for further access. It returns the receiver.

read(i: Integer): String
This method reads the next first i characters from the referred file and returns that as a *String* if i is non-negative. If fewer than i characters are available, a *String* is returned consisting of the number of

available characters (until the end of the file). The read pointer in the file has been advanced till after the last read character. In case i is negative an error is produced.

readUntil(c: Char): String
This method returns **nil** if the read pointer is at the end of the file. Otherwise, it returns the sequence of characters (as a *String*) until the next occurrence of character c in the file, or until the end of the file, whichever comes first. The read pointer has been advanced till after the character c if c was found or is at the end of the file if the end of the file has been encountered. The character c is not part of the returned *String*.

readWord: String
This method returns **nil** if no non-white space characters exist until the end of the file. Otherwise, it returns the next consecutive sequence of non-white space characters (as a *String*) until the first white space character after this sequence, or until the end of the file, whichever comes first. The read pointer has been advanced till the first white space character after the sequence in the former case or is at the end of the file in the latter. None of the white space characters are part of the returned *String*.

readLine: String
This method returns **nil** if the read pointer is at the end of the file. Otherwise, it returns the next (possibly empty) sequence of non-newline characters (as a *String*) until the first new line character sequence , or until the end of the file, whichever comes first. The read pointer has been advanced till after the longest newline character sequence after the sequence of non-newline characters (if the line ends in CR(13) followed by LF(10) it advances till after the LF(10)). None of the newline characters are part of the returned *String*.

readString: String
This method advances until the next occurrence of a double quote character ". In case the character sequence starting from the double quote character is a valid syntactical representation of a *String* (see Appendix B), then this *String* is returned and the read pointer in the file has advanced till after the end of the *String* representation. If it is not a valid syntactical representation of a *String*, an error is generated. If no double quote character is encountered, **nil** is returned.

FileOut

This class extends *Object*. It provides a means to write information to files. Creating a new *FileOut* yields an object without referring to any concrete file.

destination(s: String): FileOut
Specifies the file to be referred to for writing information to. s is a file name (possibly with an absolute or relative path reference – where the syntactic symbols / and \ for path references can be used interchangeably independent of OS). In case a file with destination indicated by N already exists, that particular file is emptied. It returns the receiver.

open: FileOut
(Re-)opens the referred file for write access. Performing such write accesses will result in first clearing all existing information in the file (if any). It assumes that method *destination* has previously

been called to identify the concrete file to refer to. Otherwise, an error is generated. It returns the receiver.

append: FileOut

(Re-)opens the referred file for write access. Performing such write accesses will result in appending the written information without overwriting the existing information in the file (if any). It assumes that method *destination* has previously been called to identify the concrete file to refer to. Otherwise, an error is generated. It returns the receiver.

flush: FileOut
This method flushes all write buffers to file. It returns the receiver.

close: FileOut
This method flushes all write buffers and releases the referred file for further access. It returns the receiver.

write(s: String): FileOut
Writes the *String s* to the referred file (not its syntactic representation). It returns the receiver.

writeLine(s: String): FileOut
Writes the *String* s to the file (its characters, not its syntactic representation) followed by new line character LF(10). It returns the receiver.

writeString(s: String): FileOut
Writes a syntactic representation of *String s* to the file. In particular, s is encoded as detailed in Appendix B. It returns the receiver.

Socket

This class extends *Object*. It provides a means to communicate via TCP/IP through sockets. Creating a new *Socket* yields an unconnected TCP/IP socket. For simplicity reasons, a *Socket* supports at most one connection between a server and client. After a connection has been established, new requests to setup a connection with a different client are refused.

acceptFrom(i: Integer): Socket
Passively accepts a TCP/IP connection from local port i. It returns the receiver.

connectTo(s: String, i: Integer): Socket
Actively establishes a TCP/IP connection to a remote socket with remote server named s or with IP address s and remote port number i. It returns the receiver.

isConnected: Boolean
Returns **true** in case is the receiver is connected and **false** otherwise.

isDisconnected: Boolean[2]
Returns **true** in case is the receiver is disconnected and **false** otherwise.

close: Socket
Releases the concrete socket (if it was created) for further communication. It returns the receiver.

---

[2] Note that methods *isConnected* and *isDisconnected* may both return **false** at some moment in time. Only one of them can return **true** at any moment in time.

hasCharacters(i: Integer): Boolean

Returns **true** in case there are at least i characters available for reading and **false** otherwise.

read(i: Integer): String

This method reads the next sequence of i available characters and returns them as a *String* if i is non-negative. If fewer than i characters are available, a *String* is returned consisting of the number of available characters. The read pointer has been advanced till after the last read character. In case i is negative, an error is produced.

hasCharacter(c: Char): Boolean

Returns **true** in case there is at least one occurrence of character c available for reading and **false** otherwise.

readUntil(c: Char): String

This method returns **nil** if there are no characters available for reading (without advancing the read pointer). Otherwise, it returns the sequence of characters (as a *String*) until the next first occurrence of character c, or the sequence of all available characters in case character c does not occur. The read pointer has been advanced till after the last read character. The character c is not returned as part of the *String.*

hasWord: Boolean

Returns **true** in case a non-empty sequence of non-white space characters is available for reading, preceded by a possibly empty sequence of white-space characters and succeeded by at least one white space character. Otherwise, it returns **false**.

readWord: String

This method returns **nil** in case no sequence of non-white space characters is available that is succeeded by a white space character (without advancing the read pointer). Otherwise, it returns the sequence of non-white space characters (as a *String*) after a possibly empty sequence of white-space characters, until the first white space character after this sequence. The read pointer has been advanced till immediately after the first white space character after the sequence of non-white space characters. None of the white space characters before or after the word are part of the returned *String*.

hasLine: Boolean

Returns **true** in case a newline character sequence is available (possibly after other characters) and **false** otherwise.

readLine: String

This method returns **nil** in case no new line character sequence is available (without advancing the read pointer). Otherwise it returns the sequence of non-new line characters (as a *String*) until the first new-line character sequence after this sequence. The read pointer has been advanced till after the longest new-line character sequence immediately following the non-newline characters (if the line ends in CR(13) followed by LF(10) it advances till after the LF(10)). The new line characters are not part of the returned *String*.

hasString: Boolean

Returns **true** in case in the sequence available for reading the first occurrence of a double quote forms with a sequence of following characters either a (complete) syntactic representation of a

String, or is an invalid beginning of a syntactic representation of a *String*, i.e., cannot be completed to a valid *String* constant. It returns **false** otherwise.

readString: String
This method advances until the next occurrence of a double quote character “. In case the character sequence starting from the double quote character is a valid syntactical representation of a *String* (see Appendix B), then this *String* is returned. The read pointer has advanced till after the end of the *String* representation. In case the character sequence is not a valid syntactical representation of a String and cannot be completed to a valid String, an error is generated. In all other cases, **nil** is returned (without advancing the read pointer).

write(s: String): Socket
Writes the *String s* to the referred socket (not its syntactic representation). It returns the receiver.

writeLine(s: String): Socket
Writes the *String* s to the referred file (not its syntactic representation) followed by new line character LF(10). It returns the receiver.

writeString(s: String): Socket
Writes the syntactic representation of *s* to the socket. In particular, s is encoded as detailed in Appendix B. It returns the receiver.

Appendix B: String and Character Constant Syntax

A string constant is defined by a sequence of characters from the extended ASCII set of 256 characters enclosed by double quotation marks. Special syntax is added to represent non-printable characters by means of *escape sequences* that all begin with the backslash (\) symbol. Characters with ASCII value below 32 (except ASCII value 9) and the two characters ” (double quotation mark, ASCII value 34) and \ (backslash, ASCII value 92) are not allowed in a string constant except as part of an escape sequence. The escape sequences that are supported are conforming to the C / Java specifications and provided in the table below.

Similarly, character constants are represented by a character between single quotation marks for characters with ASCII values 9 and 32 and above, except the single quotation mark character (ASCII value 39) and the backslash character (ASCII value 92). Moreover, any escape sequence can be used (between single quotation marks).

Character	**ASCII Representation**	**ASCII Value**	**Escape Sequence**
Newline	NL (LF)	10	\n
Horizontal tab	HT	9	\t
Vertical tab	VT	11	\v
Backspace	BS	8	\b
Carriage return	CR	13	\r
Formfeed	FF	12	\f
Alert	BEL	7	\a
Backslash	\	92	\\
Question mark	?	63	\?
Single quotation mark	'	39	\'
Double quotation mark	"	34	\"
Hexadecimal number	*hh*	any	\x*hh*
Null character	NUL	0	\0

The escape sequence using hexadecimal number representation can have 1 or 2 hexadecimal digits. It is allowed to have a 0 as a first of two digits.

Syntactic representations of String constants are not unique. Whenever a syntactic representation of a String constant is generated, the following encoding shall be used. For any character which has an individual encoding shown in the above table, this encoding shall be used. For any other character with an ASCII value below 32 (except ASCII value 9) the hexadecimal number escape sequence shall be used. All other characters are not encoded.