JOE native reference manual

This document describes the basic classes in JOE native.

These classes are written in C language following a particular convention and are embedded in the interpreter. New classes can be written in this way, put in shared libraries and made available to the interpreter

automatically generated on 2025-02-22T11:22:44.121 Sat

Class joe_Bang (the ! object)

extends joe_Object

This object implements a set of methods useful for creating objects, interacting with them and with the external environment. There is only 1 instance of this class and it is automatically instantiated by the interpreter.

addPath *aPath*

Adds *aPath* to the list of paths where the interpreter looks for scripts when the `joe` and `new` methods are called. The first path is always the path where the first script is loaded. Returns the Bang object itself.

array [*arg1* [, *arg2 ... ,argN*]]

Creates an array whose elements are reference id of the objects specified as arguments.

arraySort _anArray, *aBlock*

arraySort *aArrayList* , *aBlock*

Sorts the specified array according to the rules specified in *aBlock*. *aBlock* is invoked by the sort method passing 2 items of the array; it must return an Integer: 0 means that the 2 items are equal, a value greater than 0 means that the 1st argument is greater then the 2nd argument an a value less then 0 viceversa. Returns the array passed as argument sorted.

For example the following invocation sorts the array of integers `a` in ascending order.

```
!arraySort a,{:o1,o2. o1 - o2}.
```

asc *aString*

Returns a Integer with the ASCII code of the first character in the specified String.

deprecated the same result can be obtained using the expression `aString charCodeAt`

binarySearch *anArray*, *aObject*,*aBlock*

binarySearch *aArrayList*, *aObject*,*aBlock*

Searches the specified array for the *aObject* using the binary search algorithm. *anArray* (or *aArrayList*) must be sorted according to the rules specified in *aBlock*. Returns the index of the item corresponding to *aObject* or -1 if no object in the array is = *aObject*. *aBlock* is invoked by the method passing 2 items of the array; it must return an Integer: 0 means that the 2 items are equal, a value greater than 0 means that the 1st argument is greater then the 2nd argument an a value less then 0 viceversa. For example:

```
a:=!array 0,2,3,7,8,9.
!binarySearch a,3,{:o1,o2.o1 - o2}.
```

returns `2`

break [*aString*]

Exits from the Block whose name is *aString* (default is the current block). Leaves in the stack the result of the last execution.

breakLoop

Exits from the current loop. Leaves in the stack the result of the last execution.

chr *aInteger*

Returns a String with an ASCII character whose code has been specified.

deprecated the same result can be obtained using the expression `aInteger toChar`

debug

Activates the debugger on the next instruction

doWhile *aBlock1*,*aBlock2*

Executes *aBlock1* then executes *aBlock2*: if *aBlock2* returns Boolean <1> then executes *aBlock1* again. Repeats until *aBlock2* returns Boolean <0>. Returns the result of the last execution of *aBlock1*.

eprint [, *arg1 ... ,argN*]

Displays the arguments on the standard error. Returns the Bang object itself.

eprintln [, *arg1 ... ,argN*]

Displays the arguments on the standard error with a linefeed at the end. Returns the Bang object itself.

exec *program* [, *arg1 ... ,argN*]

Executes *program* with the arguments specified in a new process and waits for execution to terminate. Returns the return code of the execution.

execFromDir *directory*, *program* [, *arg1 ... ,argN*]

Executes *program* from the specified *directory* and waits for execution to terminate. Returns the return code of the execution..

execGetOut *program* [, *arg1 ... ,argN*]

Executes *program* with the arguments specified in a new process and returns the standard output of the runned process as a String.

for *start*, *end* [,*increment*], *aBlock*.

Executes aBlock passing *start* as argument then increments *start* by *increment* (default is 1) and repeats until the argument passed to *aBlock* is > *end*.

Returns the result of the last execution of *aBlock*.

foreach *aCollection*,*aBlock*.

For each item in *aCollection* (it can be an Array, an ArrayList or a List) executes *aBlock* passing it as argument. Returns the result of the last execution of *aBlock*.

getErrno

Returns a Integer with the errno code of the last operation on the standard C library.

getGlob *wildcards* [, *aBoolean*]

Returns a Glob Object; this is a convenience method equivalent to `!newInstance "joe_Glob",wildcards [,aBoolean]`

getOSType

Returns a String with information about the underlying operating system.

getPath

Returns an array with the list of paths where the interpreter looks for scripts when the `joe` and `new` methods are called. The first path is always the path where the first script is loaded.

getcwd

Returns the current working directory.

if *aBoolean*,*aBlockTrue*[,*aBlockElse*]

If *aBoolean* = <1> then executes *aBlockTrue* else it executes *aBlockElse* when specified. Returns the result of the last execution.

isNull *aObject*

Returns Boolean <1> if the specified argument is Null, Boolean <0> otherwise.

deprecated the same result can be obtained using the expression `() = aObject`

joe *aJoeScript* [,*arg1 ... ,argN*]

joe *anArray*

Loads and executes *aJoeScript* with the specified arguments.

When *anArray* is passed the name of the script is the item at index 0 of *anArray* while the following items contain the arguments if any. Returns whatever the script returns.

loadSO *aString*

Loads a shared object whose name is _aString. Returns a BangSO object that can be used to call C functions in the loaded library via its `call` method which returns a Pointer object. Example:

```
lib:=!loadSO"".
lib call "printf",("%s"+(!nl)),"Hello, World!".
```

If the library contains a well-formed class/classes then those classes become available to the interpreter thru `newInstance` method.

new *aJoeObjectScript* [,*arg1 ... ,argn*]

Loads the specified *aJoeObjectScript* as an Object.

newArray *aInteger*

Creates an array whose size is *aInteger*. Each item contains Null.

newInstance *aString* [,*arg1 ... ,argn*]

Instantiates a new object whose name is in *aString*.

nl

Returns a String with the current end-of-line sequence.

print [, *arg1 ... ,argN*]

Displays the arguments on the standard output. Returns the Bang object itself.

println [, *arg1 ... ,argN*]

Displays the arguments on the standard output with a linefeed at the end. Returns the Bang object itself.

random

Returns a random Float >= 0 and < 1

readLine

Reads a line from standard input and returns a String containing it.

runAsBlock *aBlock*, *aJoeScript* [,*arg1 ... ,argN*]

Loads and executes *aJoeScript* as if were a block executed from *aBlock*. This gives to *aJoeScript* the visibility of the variables in *aBlock*. Return whatever the block returns.

runJoe *aJoeScript* [,*arg1 ... ,argN*]

runJoe *anArray*

deprecated same as `joe`

sleep *aInteger*

Stops the execution for *aInteger* milliseconds. Returns *aInteger*.

switch *aObject*

Returns a Switch object. This new object has 3 methods

1. **case *anotherObject* [,*aBlock*]** compares *anotherObject* with *aObject* and if the result is Boolean <1> then executes *aBlock*; if *aBlock* is not specified then any following execution of a **case** method with a *aBlock* specified on this same object causes the execution of *aBlock* regardless; only 1 Block is executed for any Switch object; this method returns the Switch object itself
2. **default *aBlock*** executes the specified *aBlock* if no Block has been executed on this Switch object;
3. **endSwitch** returns the return code of the only Block executed by this Switch Object. " This method allows to do multichoice statement like the following:

```
!switch aNum
case 1
case 2,{
   !println aNum, " is 1 or 2".
}
case 3,{
   !println aNum, " is 3".
}
default {
   !println aNum, " is not handled".
}
endSwitch.
```

system *program* [, *arg1 ... ,argN*]

Executes *program* with the arguments specified in a new shell and waits for execution to terminate. Returns the return code of the execution.

systemExit [*aInteger*]

Causes the execution stop with the return code specified as argument (default 0).

systemGetenv *variableName*

Returns a String with the content of the specified environment variable. If the variable doesn't exist it returns Null.

throw *aString*

Throws an exception whose description is in *aString*.

toString

Returns "!"

try *aBlock1*[,*aBlock2*]

Executes *aBlock1* and if some instruction raises an exception executes *aBlock2*, if specified, passing the Exception object as argument.

Returns the result of the last execution.

typename *aObject*

Returns the name of the class of *aObject*.

unixTime

Returns the number of second from the epoch in Unix mode

version

Returns a String with informations about the current interpreter.

while *aBlock1*,*aBlock2*

Executes *aBlock1* and if it returns Boolean <1> executes *aBlock2*. Repeats until *aBlock1* returns Boolean <0>. Returns the result of the last execution of *aBlock2*.

Class joe_Array

extends joe_Object

This class represents an array.

An array can be created using the following methods:

```
aArray := !newInstance "joe_Object", aSize
aArray := !newArray aSize
aArray := !array [ e1  ... ,eN]
```

add *aObject*

Returns a new Array like this array with the addition of *aObject* as last element

clean

Replaces all the elements with ()

foreach [*aFirstIndex*,] *aBlock*

For each element of this array, *aBlock* is executed passing the index as argument. If *aFirstIndex* is specified then only the items with an index >= of *aFirstIndex* will be scanned.

get *aIndex*

Returns the *aIndex th* element of this array.

iterator

Returns a ArrayIterator for this array

length

size

Returns the length of this array.

set *aIndex*, *aObject*

Replaces the element at *aIndex* position in this array with *aObject*.

shift

Returns a new Array like this array without the first element

slice *fromIndex*, *toIndex*

Returns the specified range of this array as a new array. The value at *fromIndex* is placed into the initial element of the new array. The length of the returned array will be *toIndex* - *fromIndex*.

unshift *aObject*

Returns a new Array like this array with the addition of *aObject* as first element

---

Class joe_ArrayIterator

extends joe_Object

This class allows the scan of an Array.

An array can be created using the following methods:

```
aArrayIterator := !newInstance "joe_ArrayIterator", aArray
aArrayIterator := aArray iterator
```

hasNext

Returns Boolean <1> if this iterator has more elements to scan, <0> otherwise.

next

Returns the next element.

Class joe_ArrayIterator

extends joe_Object

This class allows the scan of an Array.

An array can be created using the following methods:

```
aArrayIterator := !newInstance "joe_ArrayIterator", aArray
aArrayIterator := aArray iterator
```

hasNext

Returns Boolean <1> if this iterator has more elements to scan, <0> otherwise.

next

Returns the next element.

Class joe_ArrayList ù

extends joe_Object

This implements a dynamic array.

An ArrayList can be created using the following method:

```
aArrayList := !newInstance "joe_ArrayList" [, aSize ]
```

aSize is the initial size of the new array list

add *aObject*

Adds *aObject* at the end of this list.

foreach [*aFirstIndex*,] *aBlock*

For each element of this list, *aBlock* is executed passing the index as argument. If *aFirstIndex* is specified then only the items with an index >= of *aFirstIndex* will be scanned.

get *aIndex*

Returns the *aIndex th* element of this list.

isEmpty

Returns Boolean <1> if this list contains no elements, <0> othewise

iterator

Returns a ArrayIterator for this array.

It is equivalent to:

```
thisList toArray; iterator
```

length

size

Returns the length of this list.

peek

Returns the last element of this list without removing it

pop

Returns the last element of this list and removes it from this list

set *aIndex*, *aObject*

Replaces the element at *aIndex* position in this array with *aObject*.

toArray

Returns a new Array containing all the elements of this list

Class joe_BangSO

extends joe_Object

This class allows to call C function in a shared library loaded using the `!loadSO` method.

call *aString* [,arg1 ... ,argN]

This method calls a C function in this shared object. It returns a Pointer object that can be interpreted either a C string or an integer or an address. The arguments can be Strings, Integers or an instance of ByteBuffer in order to pass complex data structures.

Example

```
lib := !loadSO"".
lib call "printf",("%s length =%d" + (!nl)),"string",("string" length).
```

Be careful in using this methods since an incorrect use can cause a memory corruption.

Class joe_BigDecimal

extends joe_Object

This class implements a immutable, arbitrary-precision signed decimal number.

Every literal number followed by the letter `m` is an instance of this class, e.g.: `-123.456m`

All the methods accept any kind of number as argument, i.e. BigDecimal, Integers or Float.

add *aNumber*

+ *aNumber*

Returns a new BigDecimal whose value is the sum of this number + *aNumber*.

bigDecimalValue

Returns this number.

divide *aNumber*

/ *aNumber*

Returns a new BigDecimal whose value is the quotient of this number / *aNumber*.

equals *aNumber*

= *aNumber*

Compares this number with *aNumber* and returns <1> if they are equal, <0> otherwise.

floatValue

Returns a Float containing this number. If the number exceeds the Float precision the result is undefined.

ge *aNumber*

>= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater or equal to *aNumber*, <0> otherwise.

gt *aNumber*

> *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater than *aNumber*, <0> otherwise.

intValue

Returns an Integer containing this number. If this number has a decimal part it is removed. If the number exceeds the Integer precision the result is undefined.

le *aNumber*

<= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less or equal to *aNumber*, <0> otherwise.

lt *aNumber*

< *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less than *aNumber*, <0> otherwise.

multiply *aNumber*

* *aNumber*

Returns a new BigDecimal whose value is the product of this number * *aNumber*.

ne *aNumber*

<> *aNumber*

Compares this number with *aNumber* and returns <0> if they are equal, <1> otherwise.

remainder *aNumber*

% *aNumber*

Returns a new BigDecimal whose value is the remainder of the division of this number / *aNumber*.

signum

Returns an Integer = 1 if this number is positive, -1 if it is negative an 0 if it is 0.

subtract *aNumber*

- *aNumber*

Returns a new BigDecimal whose value is the difference of this number - *aNumber*.

toString

Returns a string representation of this number.

Class joe_Block

extends joe_Object

Class joe_JOEObject

extends joe_Block

joe_Block implements a Block. All the blocks in a script are instances of this class. The current block is referenced by `!!`, The method **new** on a Block object returns a new JOEbject object (see `new`) .

doWhileFalse *aBlock*

Executes this block and then executes *aBlock*: if the result of *aBlock* execution is Boolean <0> then executes itself again. Repeats until *aBlock* returns an object not equal to <0>

Returns the result of this block last execution.

doWhileTrue *aBlock*

Executes this block and then executes *aBlock*: if the result of *aBlock* execution is Boolean <1> then executes itself again. Repeats until *aBlock* returns an object not equal to <1>

Returns the result of this block last execution.

exec [*arg1* [... , *argN*]]

Executes the statements contained in this block.

Returns the result of the last execution.

extends *aJOEObject*

This method only applies to JOEOBjects. It causes that, when a method is called on this object and it is not found the method is searched in *aJOEObject* too.

getName

Returns the name of this block. For example

```
{foo:. !println (!! getName)} exec.
```

will print "foo"

getVariable *aString*

Returns the content of a variable whose name is *aString*.

getVariablesNames

Returns an Array containing all the variables names that can be accessed by this block.

new [*arg1* [... , *argN*]]

Returns a JOEObject obtained by cloning this block. The block is executed and every variable assigned to a block will be treated as a method of the JOEObject object.

whileFalse *aBlock*

Executes this block and if its result is Boolean <0> then executes *aBlock*. Repeats until this block returns an object not equal to <0>

Returns the result of *aBlock* last execution.

whileTrue *aBlock*

Executes this block and if its result is Boolean <1> then executes *aBlock*. Repeats until this block returns an object not equal to <1>

Returns the result of *aBlock* last execution.

Class joe_Boolean

extends joe_Object

This class implements a boolean type.

and *aBoolean*

AND boolean operation

ifFalse *aBlock1* [, *aBlock2*]

If this boolean is <0> then this method executes *aBlock1* else *aBlock2* when supplied.

ifTrue *aBlock1* [, *aBlock2*]

If this boolean is <1> then this method executes *aBlock1* else *aBlock2* when supplied.

iif *aObject1*, *aObject2*

If this boolean is <1> then this method return *aObject1* else *aObject2*.

not

NOT operation

or *aBoolean*

OR boolean operation

toString

Returns the string representation of this boolean.

xor *aBoolean*

XOR boolean operation

---

joe_ByteArray

extends joe_Object

This class implements a byte array to be used in interfacing C functions.

You must specify the size when you instance an object, e.g.:

```
ba := !newInstance"joe_ByteArray",128.
```

byteValue

Returns an Integer whose value is obtained interpreting the memory of this array as if were a machine byte

child *startIndex*, *length*

Returns a new ByteArray whose memory is the same of this array starting from *startIndex* for *length* bytes. Modifying the memory of this array the content of the new object is modifyed as well.

init [*aString*]

Fills the content of this object with byte 0 or the first character of *aString* when supplied

intValue

Returns an Integer whose value is obtained interpreting the memory of this array as if were a machine int

length

size

Returns the number of bytes allocated by this object

longValue

Returns an Integer whose value is obtained interpreting the memory of this array as if were a machine long

pointerValue

Returns an Pointer whose value is obtained interpreting the memory of this array as if were a machine pointer

set [*aObject*]

Sets the content of *aObject* into this byte array: _aObject_can be a String, an Integer or another ByteArray.

setByte *aInteger*

Sets a byte number in machine format into this byte array.

setInt *aInteger*

Sets a int number in machine format into this byte array.

setLong *aInteger*

Sets a long number in machine format into this byte array.

setShort *aInteger*

Sets a short number in machine format into this byte array.

shortValue

Returns an Integer whose value is obtained interpreting the memory of this array as if were a machine short

toString

Returns a String whose value is obtained interpreting the memory of this array as if were a C string

---

Class joe_Class

extends joe_Object

This class implements a JOE basic class. An instance of this class can be obtained from any object thru the method `getClass`.

getMethods

Returns an array with the name of the methods of this class. In the current implementation it works only for basic class, i.e. JOEObjects will return only the basic methods, not the methods defined as blocks.

getName

Returns the name of this class.

toString

Returns a string representation of this class.

Class joe_Date

extends joe_Object

This class implements a timestamp for the Gregorian calendar. It can be obtained with the following invocation:

```
!newInstance "joe_Date" [ , aInteger ]
```

When *aInteger* is specified it is interpreted as the number of millisecond passed from 0001-01-01T00:00:00 plus 518400000: this correction number has been used because 0001-01-01 was a Saturday so you can get the day of the week with the following operation: t / 86400000 % 7 which will return 0 = Sunday, 1 = Monday etc.

With no argument, the Date will contain the time in which it has been instantiated.

addDays *aInteger*

Returns a new Date calculated adding *aInteger* to this date

diffDays *aDate*

Returns an Integer containing the number of days intercoursed between this date and *aDate*

equals *aDate*

= *aDate*

Returns Boolean <1> if this date is equal to *aDate*, <0> otherwise

getDate

Returns an Integer containing the day of month of this date

getDay

Returns an Integer containing the day of the week of this date (0 = Sunday, 1 = Monday etc.)

getEpochMillis

Returns an Integer containing the milliseconds of this date counted starting from 1970-01-01T00:00:00 (Unix fashion)

getHours

Returns an Integer containing the hours of this date

getMinutes

Returns an Integer containing the minutes of this date

getMonth

Returns an Integer containing the month of this date (1 = January, 2 = February etc)

getSeconds

Returns an Integer containing the seconds of this date

getTime

Returns an Integer containing the milliseconds of this date counted starting from 0001-01-01T00:00:00 plus 518400000

getYear

Returns an Integer containing the year of this date

gt *aDate*

> *aDate*

Returns Boolean <1> if this date is after *aDate*, <0> otherwise

lt *aDate*

< *aDate*

Returns Boolean <1> if this date is before *aDate*, <0> otherwise

setDate *aInteger*

Sets *aInteger* as day of the month of this date

setHours *aInteger*

Sets *aInteger* as hours of this date

setMinutes *aInteger*

Sets *aInteger* as minutes of this date

setMonth *aInteger*

Sets *aInteger* as month of this date (1 = Janaury, 2 = February etc)

setSeconds *aInteger*

Sets *aInteger* as seconds of this date

setTime *aInteger*

Sets *aInteger* as milliseconds of this date; the milliseconds are conted from 0001-01-01T00:00:00 plus 518400000

setYear *aInteger*

Sets *aInteger* as year of this date

toString

Returns a string representation of this date

Class joe_Exception

extends joe_Object

This class implements a JOE Exception. An instance of thi class can be obtained with the following call:

```
!newInstance "joe_Exception" [ ,aString ]
```

where *aString* is the exception message when specified.

getMessage

Returns a string with the message associated to this exception

throw

Raises this exception

toString

Returns a string representation of this exception

Class joe_Execute

extends joe_Object

An instance of this class allows the execution of arbitrary code contained in strings.. An instance of thi class can be obtained with the following call:

```
!newInstance "joe_Execute" [ , aBlock ]
```

aBlock is the block from which the calls are executed; if not specified a new block is used.

add *aString*

Adds *aString* to the list of calls to execute.

clear

Clears the list of calls to execute.

exec

Executes the list of calls previously load with `add`. Returns the result of the last call.

Class joe_Files

extends joe_Object

An instance of this class makes available some useful methods for handling files. You can get an instance with the following call:

```
!newInstance "joe_Files"
```

deleteIfExists *aString*

Deletes the file *aString* if exists: Returns Boolean <1> if the file has been deleted, <0> otherwise

exists *aString*

Returns Boolean <1> if the file *aString* exists, <0> otherwise

getAttribute *aString,aAttribute*

Gets information about the file *aString*. This method returns different object depending on the attribute specified in the string *aAttribute*, i.e.;

aAttribute	Object returned
"isRegularFile"	Boolean <1> if *aString* is a regular file <0>, otherwise
"isDirectory"	Boolean <1> if *aString* is a directory file <0>, otherwise
"isOther"	Boolean <1> if *aString* is neither a directory nor a file, <0> otherwise
"fileKey"	String that is unique for each file in the system (eg (dev=2049,ino=3282325)
"lastModifiedTime"	aDate object with the time of last modification
"lastAccessTime"	aDate object with the time of last access
"creationTime"	aDate object with the time of creation

These attributes depend from the underlying filesystem so the results can be inaccurate.

isAbsolute *aString*

Returns Boolean <1> if the path specified by *aString* is absolute, <0> otherwise.

isDirectory *aString*

Returns Boolean <1> if the file specified by *aString* is a directory, <0> otherwise.

listDirectory *aString*

Returns a String ArrayList containing the list of the file in the directory specified by *aString*

readAllLines *aString*

Returns a string array containing all the lines in the file whose path is *aString*

write *aString,aArray* [, *aOpenMode*]

Writes the representation string of all the objects in *aArray* in the file *aString*. If *aOpenMode* is specified it must be a String containing the open mode as in the fopen C call ("w", "w+","a","a+").

Class joe_Float

extends joe_Object

This class implements a immutable, double-precision floating-point number.

Every literal number that contains a dot (.) and is **not** followed by the letter `m` is an instance of this class. The E notation is supported as well.

e.g.: `-123.456,-1.23456e02`

All the methods accept any kind of number as argument, i.e. Float, Integers, or BigDecimal, and returns a different type of object depending on the argument: an operation with a Float or Integer argument returns a Float, an operation with a BigDecimal argument returns a BigDecimal.

add *aNumber*

+ *aNumber*

Returns a new number whose value is the sum of this number + *aNumber*.

bigDecimalValue

Returns a BigDecimal containing this number.

divide *aNumber*

/ *aNumber*

Returns a new number whose value is the quotient of this number / *aNumber*.

equals *aNumber*

= *aNumber*

Compares this number with *aNumber* and returns <1> if they are equal, <0> otherwise.

floatValue

Returns this number.

ge *aNumber*

>= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater or equal to *aNumber*, <0> otherwise.

gt *aNumber*

> *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater than *aNumber*, <0> otherwise.

intValue

Returns an Integer containing this number. If this number has a decimal part it is removed. If the number exceeds the Integer precision the result is undefined.

le *aNumber*

<= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less or equal to *aNumber*, <0> otherwise.

lt *aNumber*

< *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less than *aNumber*, <0> otherwise.

multiply *aNumber*

* *aNumber*

Returns a new number whose value is the product of this number * *aNumber*.

ne *aNumber*

<> *aNumber*

Compares this number with *aNumber* and returns <0> if they are equal, <1> otherwise.

remainder *aNumber*

% *aNumber*

In the current version just raises an exception.

signum

Returns an Integer = 1 if this number is positive, -1 if it is negative an 0 if it is 0.

subtract *aNumber*

- *aNumber*

Returns a new number whose value is the difference of this number - *aNumber*.

toString

Returns a string representation of this number.

Class joe_Glob

extends joe_Object

This implements a glob pattern.

An ArrayList can be created using the following methods:

```
!newInstance "joe_Glob" wildCards [, caseInsensitive ]
!getGlob" wildCards [, caseInsensitive ]
```

where *wildCards* is a String containing wildcards and *caseInsensitive* is a Boolean.

Wildcards supported are

- ? matches any character exactly once.
- * matches a string of zero or more characters.
- [...], where the first character within the brackets is not ', matches any single character among the characters specified in the brackets. If the first character within brackets is ', then the [^...] matches any single character that is not among the characters specified in the brackets.

A backslash (\) before a wildcard removes its special meaning.

matches *aString*

Returns Boolean <1> if *aString* is a match for this glob, <0> otherwise.

Class joe_HashMap

extends joe_Object

This implements a an hash table.

An HashMap can be created using the following method:

```
aHashMap := !newInstance "joe_HashMap" [, aSize ]
```

aSize is the initial size of the new hash table.

In this hash table you can use any kind of object as key, however its String representation is stored, hence for example the key 1 is equivalent to the key "1"

containsKey *aKey*

Look in this hash table for *aKey*.

If the key is found then Boolean <1> is returned otherwise <0> is returned.

containsValue *aValue*

Look in this hash table for *aValue*.

If the value is found then Boolean <1> is returned otherwise <0> is returned.

get *aKey*

Look in this hash table for *aKey*.

If the key is found then the associated value is returned otherwise () is returned

keys

getKeys

Returns a String array containing all the keys in this hash table.

length

size

Returns the length of this hash table.

put *aKey*, *aValue*

Insert *aValue* in this hash table with key *aKey*.

Both *aKey* and *aValue* can be any kind of object, however for *aKey* its String representation is used.

If no item with the specified key is in the table then the couple is added to the table and the method returns () otherwise *aValue* is substituted to the previous value and the method returns the previous value.

values

Returns an array containing all the values in this hash table.

Class joe_Integer

extends joe_Object

This class implements a immutable, 64 bits long integer number.

Every literal integer, possibly followed by the letter `l`, is an instance of this class.

Nearly all the methods accept any kind of number as argument, i.e. Integers, Float or BigDecimal, and returns a different type of object depending on the argument: an operation with a Integer argument returns a Integer, an operation with a Float oargument returns a Float, an operation with a BigDecimal argument returns a BigDecimal.

add *aNumber*

+ *aNumber*

Returns a new number whose value is the sum of this number + *aNumber*.

and *aInteger*

Returns the results of a bitwise AND between this number and *aInteger*

and *aInteger*

Returns the results of a bitwise OR between this number and *aInteger*

and *aInteger*

Returns the results of a bitwise XOR between this number and *aInteger*

bigDecimalValue

Returns a BigDecimal containing this number.

divide *aNumber*

/ *aNumber*

Returns a new number whose value is the quotient of this number / *aNumber*.

equals *aNumber*

= *aNumber*

Compares this number with *aNumber* and returns <1> if they are equal, <0> otherwise.

floatValue

Returns a Float containing this number.

ge *aNumber*

>= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater or equal to *aNumber*, <0> otherwise.

gt *aNumber*

> *aNumber*

Compares this number with *aNumber* and returns <1> if this number is greater than *aNumber*, <0> otherwise.

intValue

Returns this number.

le *aNumber*

<= *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less or equal to *aNumber*, <0> otherwise.

lt *aNumber*

< *aNumber*

Compares this number with *aNumber* and returns <1> if this number is less than *aNumber*, <0> otherwise.

multiply *aNumber*

* *aNumber*

Returns a new number whose value is the product of this number * *aNumber*.

ne *aNumber*

<> *aNumber*

Compares this number with *aNumber* and returns <0> if they are equal, <1> otherwise.

not

Returns the results of a bitwise NOT on this number

remainder *aNumber*

% *aNumber*

Returns a new Integer whose value is the remainder of the division of this number / *aNumber*. If *aNumber* is a Float an exception is thrown

signum

Returns an Integer = 1 if this number is positive, -1 if it is negative an 0 if it is 0.

subtract *aNumber*

- *aNumber*

Returns a new number whose value is the difference of this number - *aNumber*.

toChar

Returns a String one character long containing the character whose code is this number.

toHexString

Returns a String containing the hexadecimal representation of this number.

toString

Returns a string representation of this number.

Class joe_Object

This is the base class of all the JOE classes, hence its methods can be used with every object.

clone

Returns a clone of this object.

equals *aObject*

= *aObject*

Returns Boolean <1> if *aObject* is the same as this object, <0> otherwise.

Usually this method is overridden in subclasses

getClass

Returns the Class of this object.

toString

Returns a String representation of this object.

Usually this method is overridden in subclasses

Class joe_Pointer

extends joe_Object

This class implements the functionality of an immutable C pointer

It can be created using the following method:

```
!newInstance "joe_Pointer"
```

however, the usual way of getting one of them is thru the `call` method of `joe_BangSO`

displace *aInteger*

Returns a new pointer obtained adding *aInteger* to this pointer

intValue

Interprets this pointer as an native int and returns a Integer containing that number.

isNull

Returns Boolean <1> if this is a null pointer, <0> otherwise

longValue

Interprets this pointer as an native long and returns a Integer containing that number.

shortValue

Interprets this pointer as an native short and returns a Integer containing that number.

stringValue

Interprets this pointer as an address to a C string and returns a String containing that string.

Beware if used on a bad pointer this method can cause a memory corruption.

toString

Returns the String representation of this pointer, usually the hexadecimal value.

Class joe_StringBuilder

extends joe_Object

This class is useful for building strings

It can be created using the following method:

```
!newInstance "joe_StringBuilder" [ _aString ]
```

where *aString* is the initial value of this object when specified

add *aObject*

+ *aObject*

Appends the string representation of *aObject* at the end of this object

delete *aFirst,aLast*

Delete the substring delimited by *aFirst aLast* from this object The first character has index 0 and *aLast* is the index of the first character that isn't included in the deletion.

insert *aInteger,aObject*

Inserts the string representation of *aObject* in this object after the *aInteger th* character.

length

Return the length of this object.

toString

Return a String contaning the content of this object.

Class joe_String

extends joe_Object

This class represents character strings. All string literals, such as "Abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created.

add *aString*

+ *aString*

Concatenates the *aString* to the end of this string.

at *aIndex*

Convenience method for *substring aIndex,(aIndex + 1)*

bigDecimalValue

Returns a BigDecimal if the content of this string is a valid number, () otherwise.

charCodeAt [*aIndex*]

Returns a Integer with the ASCII code of the *aIndex th* character (default is 0) of this string.

compareTo *aString*

Returns 0 if this string is = *aString*, an integer greater than 0 if this string is > *aString*, an integer less than 0 if this string is < *aString*

endsWith *aString*

Returns Boolean <1> if this string last part is = *aString*, Boolean <0> otherwise.

equals *aString*

= *aString*

Returns Boolean <1> if this string is = *aString*, Boolean <0> otherwise.

equalsIgnoreCase *aString*

Returns Boolean <1> if this string is = *aString* ignoring case considerations, Boolean <0> otherwise.

floatValue

doubleValue

Returns a Float if the content of this string is a valid floating-point number, () otherwise.

ge *aString*

>= *aString*

Returns Boolean <1> if this string is > or = *aString*, Boolean <0> otherwise.

gt *aString*

> *aString*

Returns Boolean <1> if this string is > *aString*, Boolean <0> otherwise.

indexOf *aString* [, *aIndex*]

Returns the index within this string of the first occurrence of the specified substring, starting at *aIndex* (default is 0). If there isn't any occurence of *aString* in this string then the method returns -1.

intValue

longValue

Returns an Integer if the content of this string is a valid decimal number, `()` otherwise.

lastIndexOf *aString* [, *aIndex*]

Returns the index within this string of the last occurrence of the specified substring, starting at *aIndex* (default is 0). If there isn't any occurence of *aString* in this string then the method returns -1.

le *aString*

<= *aString*

Returns Boolean <1> if this string is < or = *aString*, Boolean <0> otherwise.

length

Returns the length of this string.

lt *aString*

< *aString*

Returns Boolean <1> if this string is < *aString*, Boolean <0> otherwise.

matches *aRegex*

Returns Boolean <1> if this string matches the regular expressio *aRegex*, Boolean <0> otherwise. Special characters in the regular expression are

```
* ? + [ ^ ] .
```

regex starting with `(?i)` makes the matching case-insensitive

split, *replaceFirst* and *replaceAll* support also anchors `^` and `$`

ne *aString*

<> *aString*

Returns Boolean <1> if this string is not = *aString*, Boolean <0> otherwise.

replace *aString1*, *aString2*

Returns a new String like this string in which any occurrence of *aString1* is replaced by *aString2*.

replaceAll *aRegex*, *aString*

Returns a new String like this string in which any substring matching *aRegex* is replaced by *aString*. (see `matches` for regex).

replaceFirst *aRegex*, *aString*

Returns a new String like this string in which the first substring matching *aRegex* is replaced by *aString*. (see `matches` for regex).

split *aRegex*

Splits this string around matches of the given regular expression (see `matches` for regex)

startsWith *aString*

Returns Boolean <1> if this string first part is = *aString*, Boolean <0> otherwise.

substring *aIndex1* [, *aIndex2*]

Returns a new String that is a substring of this string starting from the *aIndex1 th* character (first character has 0 index) till the end of this string. When *aIndex2* is specified, it represents the index of the first character not included in the substring. If an index is < 0 then it is equivalent to **string length; + index**

toLowerCase

Returns a new String like this string converted in lower case.

toString

Returns this string.

toUpperCase

Returns a new String like this string converted in upper case.

trim

Returns a new String like this string without spaces at the beginning and/or at the end.
