JDromadaire Documentation

Thimote75

Apr 28, 2021

CONTENTS

CHAPTER

ONE

JDROMADAIRE

Welcome to JDromadaire's Wiki, it is the implementation of Drom in Java.

CHAPTER

TWO

BASIC

The goal of this project is to create a language which takes java's syntax and python advantages to create a better language on the point of syntax but also on some flaws of python like sockets.

Here are the actual utils for JDrom :

2.1 Install JDromadaire

You will need java and eclipse to be installed

Download java here Download eclipse here

Then, clone the repository :

`git clone https://github.com/MrThimote/JDromadaire.git`

Open the Directory JDromadaire as a project with Eclipse

Then go to `main/EntryPoint.java`

Right click and press Run/Run as Java Application

Done !

2.2 Loops

2.2.1 For syntax

Base syntax

Example code:

```
for(var = 0; var < n; var = var + 1) {
    ## Your code
}
```

This syntax is the base syntax in informatic, it uses first a set expression, here `var = 0`, a comparate expression, here `var < n`, and an advance expression at the end of each iteration, here `var = var + 1`

It will run your code while the comparate expression returns true.

Iterator syntax

Example code:

```
for(var:iter) {
    ## Your code
}
```

This syntax can be used when iter is for an example an array. If it is an array, it will result in looping on every object of the array.

While

Example code:

```
while(true) {
   ## Your code
}
```

So this syntax is not implemented. Your code will run while the expression gives you a true, here `true`.

2.2.2 Break / Continue

Break

- Base Break

You can launch a base break by doing `break`. This will break the loop in which you are

For an example, this code:

```
for(i = 0; i < 5; i = i + 1) {
   if (i == 2) {
       break;
   }
   print(i);
}
```

returns `0 1`

- Multiple Breaks

As a dev, you may have already tryed to do double or triple break, but to do this, there is a syntax in Drom to do it : you can do `break(x)` to break x times. But you can not break with a variable, it is always a known number like 1,2,3...

This means that if you have two loops :

```
for (i = 0; i < x; i = i + 1) {
   for (j = 0; j < y; j = j + 1) {
       break(2)
   }
}
```

The `break(2)` will break both loops

Continue

- Base Continue

You can launch a base continue by doing `continue`. This will stop the actual iteration of the loop in which you are

For an example, this code:

```
for(i = 0; i < 5; i = i + 1) {
   if (i == 2) {
        continue;
   }
   print(i);
}
```

returns `0 1 3 4`

- Multiple Continues

Like the break, you can give parameters to the continue. The first parameter is the id of the loop. For an example if you set the first parameter x to 5, this will break 4 loops and will continue the 5th. The second parameter is the number of jump, this means the count of times you will execute the advance expression in a for loop before continuing normally.

For an example,

```
for(i = 0; i < 5; i = i + 1) {
   if (i == 2) {
        continue(1,2);
   }
   print(i);
}
```

returns `0 1 4`. Here, it has jumped two times the loop while launching `i = i + 1`.

2.3 Functions

2.3.1 Define functions

In Drom, you can define function with the keyword `function` It is used as the followed example :

```
function f(x){
        ## Your code
}
```

A function can take multiple arguments :

```
function f(x,y,z){
        ## Your code
}
```

2.3.2 Specificities function

Like others programming languages, a function can be called and return arguments

```
JDrom Console - Alpha [0.0.1]
> function f(x){return x^2;}
parser.nodes.FunctionNode@5cad8086
> f(6)
36
```

The main specificity of Drom's function, is that you don't have to specify the type of function (`void` / `int` / `long` ...)

All functions are defined with `function` whether they have return arguments or not.

2.4 Boolean, if else and logical operators

2.4.1 Boolean

Like others language, boolean are defined with `true` and `false`

For example, defined a `x` variable which is a boolean and will be true can be done as the follow :

```
x = true
```

2.4.2 If and else

For the moment, juste the if is implemented, but `else` and `elseif` will come soon.

You can use it as he follow :

```
a=true
if(a){
        print("a is true")
}
```

2.4.3 Logical operators

You can use the Java/C/C++ synthax for boolean logic :

For example :

```
> true || false
true
> true && false
false
> !true
false
```

2.5 Import module

2.5.1 Use

The `import` module is used like in Python to import some librairies.

For the moment, you can use it as the follow :

```
> import library
> library.a
```

But, a new feature will appeare soon :

```
> from library import a,b,c
```

Here, we just import some components of the library, and when we use them, we don't have to specify the name of the library.

With this syntax, we can also import all components :

```
> from library import *
```

But warning !

When two libraries have a function with the same name, and you just want to use one, if you import both functions with :

```
> from library1 import *
> from library2 import *
```

There will be confusion to which library the function is related

You can see the documentation about libraries here

2.6 Libraries

2.6.1 Installed libraries

For the moment, three libraries are installed by default : `math`, `time` and `socket`

The `math` library is here to bring some mathematical operators such as `sqrt`, `cos`, `sin`, `tan`, `abs`

The `socket` library is here to bring some socket connection utils.

The `time` library proide some functions like `sleep` or `now`

You can find the documentations here :

- Math : here
- Socket : here
- Time : here

2.6.2 Make your own native libraries

You can also make your own native libraires in Java.

First, you need to open Eclipse, and create a new project

Then, add the jar of dromadaire to the build path

Create then a class in `src>config` named `Exporter.java`

The base of this class is :

```
package config;

import java.util.HashMap;

import libs.LibExporter;
import parser.Node;
import parser.nodes.NumberNode;

public class Exporter implements LibExporter{

        @Override
        public HashMap<String, Node> exportClasses() {
                // TODO Auto-generated method stub
                HashMap<String,Node> hash = new HashMap();
                hash.put("one", new NumberNode(1,-1,-1));
                ## here add your functions / variables
                return hash;
        }

}
```

You can then create other class where your functions are defined and import them.

2.6.3 File libraries

This feature isn't fully implemented yet, but it will come as soon as possible

Let's assume you wan't directly to create a library written in dromadaire.

You can create a file named : `library.dmd` where library is the name of your library and you just can import this library as the native libraries :

> import library

2.7 Math library

2.7.1 Introduction

As in Python, the math library is here to provide the basic utils and maths functions so you can make some mathematics using dromadaire.

You can import this library using :

```
> import math
##or
> from math import *
## (will come soon)
```

2.7.2 Variables

The math library doesn't have a lot of variables yet, but there will be a lot more in some days.

Pi

You can get an approximation of the pi number using

```
> math.pi
```

Version

You can get the version of the library using :

```
> math.version
```

2.7.3 Functions

Trigonometry

1. cosinus

You can get the cosinus of an angle (in radians) using :

```
> math.cos(angle)
```

2. sinus

You can get the sinus of an angle (in radians) using :

```
> math.sin(angle)
```

3. tangent

You can get the tangent of an angle (in radians) using :

```
> math.tan(angle)
```

4. radians

You can convert an angle from degrees to radians using :

```
> math.radians(angle)
```

5. Arc cosinus

You can get the arc cosinus (in radians) of a cosinus using :

```
> math.acos(angle)
```

6. Arc sinus

You can get the arc sinus (in radians) of a sinus using :

```
> math.asin(angle)
```

7. Arc tangent

You can get the arc tangent (in radians) of a tangent using :

```
> math.atan(angle)
```

Hyperbolic trigonometry

1. Hyperbolic cosinus

You can get the hyperbolic cosinus of an angle (in radians) using :

```
> math.cosh(angle)
```

2. Hyperbolic sinus

You can get the hyperbolic sinus of an angle (in radians) using :

```
> math.sinh(angle)
```

3. Hyperbolic tangent

You can get the hyperbolic tangent of an angle (in radians) using :

```
> math.tanh(angle)
```

4. Hyperbolic Arc cosinus

You can get the hyperbolic arc cosinus (in radians) of an hyperbolic cosinus using :

```
> math.acosh(angle)
```

5. Hyperbolic Arc sinus

You can get the hyperbolic arc sinus (in radians) of an hyperbolic sinus using :

```
> math.asinh(angle)
```

6. Hyperbolic Arc tangent

You can get the hyperbolic arc tangent (in radians) of an hyperbolic tangent using :

```
> math.atanh(angle)
```

Other arithmetical functions

1. Absolute value

You can get the absolute value of a number using :

```
> math.abs(number)
```

2. Decimal logarithm

You can get the decimal logarithm of a number using :

```
> math.log(number)
```

3. Natural logarithm

You can get the natural logarithm of a number using :

```
> math.abs(number)
```

4. Exponential

You can get the exponential of a number using :

```
> math.exp(number)
```

5. Square root

You can get the square root of a number using :

```
> math.sqrt(number)
```

6. Factorial

You can get the factorial of a number using :

```
> math.fac(number)
```

7. Root

You can get the xth root of a number using :

```
> math.root(number,x)
```

2.8 Socket Library

2.8.1 Introduction

The dromadaire's socket library is based on OpenNS, a socket java library.

The main goal of OpenNS is to create three tunnels considered as one socket. The first tunnel is used to send data that can wait a little while, the second is for encrypted data and the third for data that cannot wait.

It's the same with socket, but it is more easy to use, and can directly be used with the dromadaire language (which of course is better than Java)

To import the library, just use

```
> import socket
```

2.8.2 Server

To create the server, it's pretty easy, you just have to run :

```
> serv = socket.ServerSocket(port)
```

where `serv` is the name of your server variable and `port` the port where the sockets will be send

Then, you need to accept client connections to send sockets with it.

```
> serv_socket=serv.accept()
```

It will return `false` when there isn't any connections, and a socket with the latest client connection.

2.8.3 Client

To create a client, simply run

```
> client=socket.Socket(ip,port)
```

where `client` is the name of the client variable, `ip` the ip and `port` the port

2.8.4 Send datas

With socket, a client can send datas of three type :

1. Normal datas

Normal datas can just simply be send with

```
> client.write(datas)
```

2. Encrypted datas

Encrypted datas can be send with

```
> client.writeSecure(datas)
```

3. Important datas

Important datas can be send with

```
> client.writeFallback(datas)
```

2.8.5 Receive datas

1. Normal datas

Normal datas can just simply be received with

```
> serv_socket.read()
```

2. Encrypted datas

Encrypted datas can be received with

```
> serv_socket.readSecure()
```

3. Important datas

Important datas can be received with

```
> serv_socket.readFallback()
```

2.9 Time library

2.9.1 Introduction

The time library provide some functions to mesure the time, or to wait for a moment.

It's like the python's one.

For the moment, just a fiew functions are implementes, but some will come soon

You can import it with

```
> import time
```

2.9.2 Functions

Version

You can get the version of the library using :

```
> time.version
```

Now

You can get the number of seconds since the 01/01/1970 00:00:00 with :

```
> time.now()
```

Sleep

You can sleep a number of seconds given with :

```
> time.sleep(seconds)
```

2.10 Natives Functions

2.10.1 Mathematic

Range

The range function is used as in python for the loop

It's used as the follow :

```
for(i:range(nb)){
        ##Your code
}
```

It can be used with : - one argument (from 0 to this number, increasing by 1) - two arguments (from first to second incresing by 1) - three arguments (from first to second incresing by the third)

Round

The round function is used to approxime the numbers with (or not) a given precision .

It can be used as the follow :

```
> round(1.4235346575685)
1
> round(math.pi,2)
3.14
```

Sum

The sum function take an array in argument, and return his sum :

```
> sum([1,2,3]
6.0
```

2.10.2 Utils

Print

The print function is used to print all given arguments :

```
> print(1,"hello",[1,2,3])
1
'hello'
[1, 2, 3]
```

Help

The help function gives you some help (laughing)

Input

The input function will give you the text written :

```
>a=input()
hello
>a
'hello'
```

Map

The map function is used as in python :

```
> a=[1,2,3]
> function f(x){return x^2;}
> map(f,a)
[1,4,9]
```

It's the equivalent to do :

```
> b=[]
> for(i:a){b.add(f(i));}
> b
[1,4,9]
```

2.10.3 Conversion

Int

The int function convert a string (or number) to an integer :

```
> int("12")
12
> int(1.4634575685658)
1
```

Str

The str function convert an integer (or array or string) to a string :

```
> str(12)
'12'
> str([1,2,3])
'[1,2,3]'
```

Number

The number function convert a string (or number) to a number :

```
> number("12.53")
12.53
> number(3.14)
3.14
```

2.10.4 Characters

Chr

The chr function convert a int to a character :

```
> chr(65)
'A'
> chr(66)
'B'
```

Ord

The ord function convert a character to an integer :

```
> ord("A")
65
> ord("B")
66
```

2.10.5 File

The file system uses java's file system and is already implemented, it can be used like this

```
file = File(path)
```

This creates a file, if it doesn't exists using

```
file.exists()
```

you can create the folder with

```
fold = File(pathWithoutEnd)
fold.mkdir()
```

And then create the file with

```
file.create()
```

You can read inside the file with `file.read()`, write with `file.write(text, append)` and delete it with `file.delete()`

2.11 Classes

The classes can't be created inside the code but the implementation can be done and can be seen in the functions/file folder which creates a File class to implement the file system.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search