This is a draft document. it is very incomplete...
Hadra Language Version 0.0.1.1
buit on 2020-06-14
Document Version 0.0.1
buit on 2020-06-14

THE PUBLIC IS MORE FAMILIAR WITH BAD DESIGN THAN GOOD DESIGN. IT IS, IN EFFECT, CONDITIONED TO PREFER BAD DESIGN, BECAUSE THAT IS WHAT IT LIVES WITH. THE NEW BECOMES THREATENING, THE OLD REASSURING.

PAUL RAND

A DESIGNER KNOWS THAT HE HAS ACHIEVED PERFECTION NOT WHEN THERE IS NOTHING LEFT TO ADD, BUT WHEN THERE IS NOTHING LEFT TO TAKE AWAY.

ANTOINE DE SAINT-EXUPÉRY

. . . THE DESIGNER OF A NEW SYSTEM MUST NOT ONLY BE THE IMPLEMENTOR AND THE FIRST LARGE-SCALE USER; THE DESIGNER SHOULD ALSO WRITE THE FIRST USER MANUAL. . . IF I HAD NOT PARTICIPATED FULLY IN ALL THESE ACTIVITIES, LITERALLY HUNDREDS OF IMPROVEMENTS WOULD NEVER HAVE BEEN MADE, BECAUSE I WOULD NEVER HAVE THOUGHT OF THEM OR PERCEIVED WHY THEY WERE IMPORTANT.

DONALD E. KNUTH

TAHA BEN SALAH

HADRA-LANG V0.1 BOOK

SOUSSE UNIVERSITY, ENISO

CONTRIBUTORS :

THEVPC

DESIGNER, DEVELOPER, MAINTAINER

PUBLISHED BY SOUSSE UNIVERSITY, ENISO

GITHUB.IO/THEVPC/HADRALANG

First printing, August 2021

Contents

List of Figures

List of Tables

Dedicated to my fabulous family.

Introduction

HadraLang is a new GPL for DSL, multi paradigm, statically typed, transpiled, programming language that is very inspired from Java, C#, C++, Scala, Kotlin and Haskell among other programming languages around. HadraLang (or **HL**) compiles mainly to java source code or directly to java bytecode. By GPL for DSL we mean a general purpose programming language that makes it easy to be used as a Domain Specific Language in a range of domains.

HadraLang is entended to be

- easy to learn for DSL users
- more precise semantics
- with better checking : compiler should do more work
- focus conciseness : write less, do more
- promote readability : meaningful statements
- enable extensibility : always make room for extensions

and brings features like

- Single file project
- Embeddable scripting
- Transpillable to a range of Languages (for now only Java is supported)
- Preprocessing
- Operator overloading with unicode support
- Extension of existing classes
- Extension of control statements
- null safety operators
- Checked and unchecked calls
- Tuples and Range support

Why do we need another Programming Language?

HadraLang was first designed as a backing scripting language for "Hadrumaths" java/scala library which is mainly an algebraic calculus library as a replacement of the scripting facilities implemented in scala. Java and Scala are very powerful programming languages however code can get very verbose or very complicated sooner or later. Initially, *HadraLang* was implemented as a simple imperative programming language with operator overloading, implicit conversion, and extension functions with support for unicode operators (used in algebra calculus). There we have seen the potential of such programming language as a GPL be re-sesigning the language to becore multi-paradigm.

Hello World

Writing a Hello world in *HadraLang* is straightforward. There is no need to encapsulate your code in a function/procedure/class or whatsoever. The following code is a fully compilable/runnable valid code :

```
1 println("Hello World");
```

Assuming this line of code is saved to "Main.hl", we need to compile it and run it using this command:

```
1 $ hl Main.hl
```

This command will actually compile the code, generate the corresponding byte code and execute it a new virtual machine.

If we need only to compile the file to a jar we issue this command:

```
1 $ hl --jar Main.hl
```

Getting Started

To use *HadraLang* we need to download the *HadraLang* Compiler and Runtime. The easiest way to do so is via *nuts* package manager. Both *nuts* and *HadraLang* are written in *java* (c) and require *java* 8 runtime (or later). We start checking if java is installed :

```
1 $ java -version
2 java version %221.8.0_211%22
3 Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
4 Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
```

If we have not nuts installed yet, we start downloading nuts and installing it as show here. We download nuts binaries, a bare jar file.

```
$ wget https://github.com/thevpc/vpc-public-maven\
/raw/master/net/vpc/app/nuts/nuts/0.7.0/nuts-0.7.0.jar
```

Then we run the installer :

```
$ java -jar nuts-0.7.0.jar
```

make sure you close and re-open you console after installing nuts for the environment to be configured correctly.

Once you have nuts installed on one's machine, we issue this command to install *HadraLang* compiler and runtime.

```
$ nuts -y install hl
```

running Hello world

```
$ hl Main.hl
```

bla bla bla

Part I

Reference Guide

1
Lexical Structure

bla bla bla

1.1 Lexical Program Structure

bla bla bla

1.2 Comments

1.3 Keywords

abstract	assert	boolean	break	byte	case
catch	char	class	continue	default	do
double	else	enum	extends	final	finally
float	for	if	~~implements~~	import	~~instanceof~~
int	interface	long	native	new	null
package	private	protected	public	return	short
static	strictfp	super	switch	synchronized	this
throw	~~throws~~	transient	try	void	volatile
while	**is**	**def**	**constructor**	**operator**	

bla bla bla

1.4 Literals

bla bla bla

1.4.1 Numerical Literals

HadraLang supports byte, short, int, long and bigint (BigInteger) numerical ordinal types out of the box.

```
1 int twelveDecimal=12;
```

```
int oneThousand=1_000;
int sixteenHexa=0x10;
int sixteenOctal=020;
int sixteenBinary=0b10000;
```

long type

```
long twelveDecimal=12L;
long oneThousand=1_000L;
long sixteenHexa=0x10L;
int sixteenOctal=020L;
long sixteenBinary=0b10000L;
```

byte type

```
byte twelveDecimal=12b;
byte oneHandred=1_00b;
byte sixteenHexa=0x10b;
byte sixteenOctal=020b;
byte sixteenBinary=0b10000b;
```

short type

```
short twelveDecimal=12S;
short oneHandred=1_00S;
short sixteenHexa=0x10S;
short sixteenOctal=020S;
short sixteenBinary=0b10000S;
```

big integer type

```
BigInteger twelveDecimal=12B;
BigInteger oneHandred=1_00B;
BigInteger sixteenHexa=0x10B;
BigInteger sixteenOctal=020B;
BigInteger sixteenBinary=0b10000B;
```

HadraLang supports short, long and bigdecimal (BigDecimal) numerical floating types out of the box.

float type

```
float twelveDecimal=12.0f;
float oneHandred=1_00f;
```

double type

```
double twelveDecimal=12.0d;
double oneHandred=1_00d;
```

big decimal type

```
BigDecimal twelveDecimal=12.0D;
BigDecimal oneHandred=1_00D;
```

1.4.2 Character and String Literals

bla bla bla

1.4.3 Temporal Literals

```
LocalDate d=t"2020-02-01";
LocalDateTime dt=t"2020-02-01T12:00";
LocalTime t=t"12:00";
```

1.4.4 Pattern Literals

```
if (p"[0-9]+".matches("12")){
    println("12 is a number");
}
if (p"[0-9]+" ~ "12"){
    println("12 is a number");
}
if ("12" ~ p"[0-9]+"){
    println("12 is a number");
}
```

1.4.5 Interpolated String Literals

```
int x=12;
String s1="Hello";
String s2="World";
println($"$s1 ${s2} for number $x");
```

1.5 Identifiers

1.6 Operators

1.6.1 == operator

```
String s1="hello";
String s2(s1);
if(s1==s1){
    println("this is true.");
}
if(null==s1){
    println("this is false.");
}
```

1.6.2 === operator

```
String s1="hello";
String s2(s1);
if(s1==s1){
    println("this is always true. null safe 'equals' will be called");
}
if(s1===s1){
    println("this is false. ref equality will be used");
}
```

1.6.3 '.' member operator

```
Person p;
println(p.address.street.name);
```

1.6.4 '?' null expression member operator

```
Person p;
println(p?address?street?name);
```

1.6.5 '??' null member operator

```
            Person p;
            println(p??"null");
```

1.6.6 '?.' Operator (unchecked member)

```
Person p;
Object o=p;
println(o.?address.?street.?name);
println(o.?address.?street.?"name");
String attrName="name";
println(o.?address.?street.?attrName);
```

1.6.7 '()' Operator (apply)

```
TODO
```

1.6.8 '[]' Operator (indexed)

```
TODO
```

1.6.9 '??=' Operator (set if not default)

```
String a="hello";
String b=null;
String c??=b;
```

1.7 Tuples

```
Tuple2<int,int> x=(1,2);
Tuple<int,int> x=(1,2);
<int,int> x=(1,2);
```

Tuple deconstructors

```
int x,y;
(x,y)=m();
(x,y)=(y,x);
def m()->(1,2);
```

1.8 Arrays

Left Hand Side matching

```
int[10] x(i->i);
x[0..2]=[15,20,30];
x[0..2]=x[5..7];
```

Right Hand Side matching

```
int[10] x(i->i);
int[] y=x[5..7];
```

Array Length

```
int[10] a;
int[10] a(3);
int[10] a(x->Math.random());
int[10] a(Math::random);
```

1.9 Type conversion and cast

Hadra Language uses intensively conversions whenever possible. So type conversion is at the heart of the language. Type conversion can either create a new instance or retain the same one (when downcasting/upcasting). Conversion is done via one of three mechanisms : down/up casting constructor call converter call. The compiler will always check one of this three alternative in a Specific order to be

defined. BEsides, a type conversion can be lenient or not. when linient, failing to convert will return null, otherwise it will generate an exception at compile time.

```
B b=(B)a;
can be achieved with
if(a is B b){
  //
}else{
    throw ClassCastException();
}
```

```
Fruit f;
Apple a=(Apple)f;
```

Thr compiler will

1.9.1 *Cast First conversion*

Tests in the order if the cast matches, than if a factory method exists and finally if a constructor exists

```
Fruit f;
Apple a=(Apple)f;
```

1.9.2 *Factory First conversion*

Tests in the order if a factory method exists than if a constructor exists otherwise only is a cast applies

```
Fruit f;
Apple a=Apple(f);
```

1.9.3 *New First conversion*

Tests in the order if a constructor exists then if a factory method exists otherwise only is a cast applies

```
Fruit f;
Apple a=new Apple(f);
```

1.9.4 *Lenient conversion*

```
Fruit f;
Apple a=(Apple?)f;
Apple b=Apple?(f);
Apple c=new Apple?(f);
```

1.10 Lambda Expressions

1.11 Annotations

TODO

2
Control-flow

2.1 if/else syntax

2.1.1 if statement

2.1.2 if expression

2.2 switch expression

2.2.1 switch/case statement

2.2.2 switch case statement

2.2.3 switch is statement

2.2.4 switch if statement

2.2.5 switch default statement

2.2.6 switch on Tuples

2.2.7 switch Expression

2.3 while loop syntax

2.3.1 while/do statement

2.3.2 do/while statement

2.3.3 while/do expression

2.3.4 do/while expression

2.4 for loop

2.4.1 for loop statement

2.4.2 for iterator statement

loops on Iterable,Enumeration,Iterator and Streams. multi iterator does the cross loop filter the loop

2.4.3 for loop expression

2.4.4 for iterator expression

⟨*statement*⟩ ::= ⟨*ident*⟩ '`=`' ⟨*expr*⟩

|

'`for`' ⟨*ident*⟩ '`=`' ⟨*expr*⟩ '`to`' ⟨*expr*⟩ '`do`' ⟨*statement*⟩

|

'`{`' ⟨*stat-list*⟩ '`}`'

|

⟨*empty*⟩

⟨*stat-list*⟩ ::= ⟨*statement*⟩ ';' ⟨*stat-list*⟩ | ⟨*statement*⟩

```
int[10] a;
for(x=0;x<a.length;x++){
    println(x);
}
```

```
int[10] a;
int[10] b;
for(x:a){
    println(x);
}
for(x:a,y:b){
    println(x,y);
}
```

Statement Filter

```
int[10] a;
int[10] b;
for(x:a){
    println(x);
}
for(x:a,y:b;x<y){
    println(x,y);
}
```

For Expression: for expression is an enumeration comprehension that creates enumerated type that will be processed at will. The result is mainly a stream that is mapped implicitly to array/List or Iterable according to the given context.

```
Iterable<int> x=for(i=0;i<10;i++)->i;
IntStream<int> x=for(i=0;i<10;i++)->i;
List<int> x=for(i=0;i<10;i++)->i;
int[] x=for(i=0;i<10;i++)->i;
```

2.5 *try/catch*

All exceptions in Hadra-Lang are uncheked. This means there is no need at compile time to add try/catch blocks. Besides Exceptions are handled both as control blocs and as expressions.

Catch Blocs

```
            int x;
            try {
                x=int(myString);
            }catch(Exception){
```

```
5                   //do some thing
6               }
```

Multi catch Blocs

```
1               int x;
2               try {
3                   x=int(myString);
4               }catch(NumberFormatException
5                   |NullPointerException ex){
6                   //do some thing
7               }catch(Exception e){
8                   //do some thing
9               }
```

Exception variable Name

```
1               int x;
2               try {
3                   x=int(myString);
4               }catch(NumberFormatException
5                   |NullPointerException){
6                   throw error;
7               }catch(Exception){
8                   throw error;
9               }
```

Catch Expressions

```
1               int x = int(myString) catch 0;
```

3
Declarations

3.0.1 Classes

HadraLang supports Object Oriented programming by providing syntax for creating and manipulating classes and methods. All features from Java such as class initializers, instance initializers, class constructors, default constructor inheritence and visibility apply. Howerver some differences will be discussed here after. a class in HadraLang must start with the class keyword. constructor and declared fields in on declaration statement.

```
1 class Complex{
2 }
```

As you can see, no modifiers are required. The default modifier is public (by opposition to package protected in java). As a matter of fact, there is NO package protected visibility in HadraLang. For top level classes, the modifiers that apply are public and final

3.0.2 Annotation Classes

```
1 @MathType
2 class Complex{
3 }
```

3.0.3 Exception Classes

3.0.4 Enumerations

3.0.5 Fields and Properties

Main Constructor Arguments

```
1 class my.Complex{
2     constructor(double real,double imag){
3         this.real=real;
```

```
        this.imag=imag;
    }
}
```

Field without Getters and Setters

```
class my.Complex{
    double real;
    double imag;
}
```

Default Getters and setters

```
class my.Complex{
    double real{ get;set;}
    double image{ get;set;}
}
```

Custom Getters and setters

```
class my.Complex{
    //create custom getter/setter
    double real{
        get {
            println("before get");
            return this.real;
        }
        set {
            //field access because we are in the setter
            this.real=value;
        }
    }
}
```

Dynamic Properties

```
class my.Complex{
    double real;
    double imag;
    double abs{
        get->sqrt(this.real^2+this.imag^2);
    }
    double absSquare->abs*abs;
}
```

Multiple Setters

```
class my.Complex{
    double real;
    double imag;
    double radius {
        set{
            real=value*sin(angle);
            imag=value*cos(angle);
```

```
8           }
9           set(int){
10              real=value*sin(angle);
11              imag=value*cos(angle);
12          }
13          set(String s){
14              set(double(s));
15          }
16      }
17  }
```

Init values

```
1  class my.Complex{
2      double real{
3          init->1;
4          get;set;
5      }
6  }
```

3.0.6 *Builders*

builder

```
1  class my.Complex(double real,double imag);
2  Complex c(0,0);
3  var r2=c.{real=0;image=5}.real;
4
5  Complex c2(0,0).{real=0;image=5};
```

3.0.7 *Constructors*

```
1  class Complex(double real,double imag){
2      constructor(double real)->this(real,0);
3  }
```

```
1  class Complex{
2      double real;
3      double imag;
4      constructor(double real,double imag){
5          this.real=real;
6          this.imag=imag;
7      }
8  }
```

```
1  class Complex{
2      double real;
3      double imag;
4  }
```

3.0.8 *Methods*

Main constructors define a simple way to create a class in a very consise way. It defines the class, a default constructor and declared fields in on declaration statement.

```
class Complex(double real,double imag){

}
```

This example creates the class Complex with a couple of public fields. Actually it is equivalent to this java code

```
//Equivalent Java
public class Complex(double real,double imag){
public double real;
public double imag;
public Complex(double real,double imag){
    this.real=real;
    this.imag=imag;
}

}
```

While the constructor itself is always public (regardless of Class visibility), fields can be secured with lower visibility modifiers :

```
class Complex(double real,private double imag){

}
```

and hence the equivalent java code will be :

```
//Equivalent Java
public class Complex(double real,double imag){
    public double real;
    private double imag;
    public Complex(double real,double imag){
        this.real=real;
        this.imag=imag;
    }

}
```

3.0.9 *Functions and global variables*

Global functions //global function (outside any class)

```
def void ageOf(Person p){
    return p.age;
}
class Person{
```

```
5       int age;
6   }
```

main function

```
1   //global function (outside any class)
2   def void main(String[] args){
3   }
4   //or
5   def main(String[] args){
6   }
```

local functions

```
1   def void main(String[] args){
2       def int max(int a,int b)->if a<b a else b
3       int x=max(3,5);
4   }
5               //or
```

3.0.10 *main function*

```
1   def main(String[] args){
2   }
```

3.0.11 *Extension functions*

```
1   import OtherClass.**
2   class my.Complex(double real,double imag){
3       constructor(double real)->this(real,0);
4   }
5   //Other Compilation Unit
6   class OtherClass{
7       def String Complex.toXml() {
8           return "<xml>"+this+"</xml>";
9       }
10  }
```

3.0.12 *Operators*

```
1   class Complex{
2       //...
3       static def Complex
4           operator Complex left + Complex right
5           -> left.add(other);
6       def Complex operator this + Complex other -> add(other);
7       def Complex operator this - Complex -> add(value);
8       def Complex operator + this -> this;
```

```
    def Complex operator ++ this -> add(1);
    def Complex operator this ++ -> add(1);
}
class Matrix{
    //...
    def Complex operator this(int column,int row)
        ->get(column,row);
    def Complex operator this[int column,int row]
        ->get(column,row);
    def void operator this[int column,int row]
        =Object
        -> set(column,row,value);
}
```

3.0.13 *Extension constructors*

```
import OtherClass.**
class my.Complex(double real,double imag){
    constructor(double real)->this(real,0);
}
//Other Compilation Unit
class OtherClass{
        //newComplex
        constructor(String value) for my.Complex{
            return Complex(double(value));
        }
        //newIntArray2
        constructor (String value) for int[][]{
            return int[0][0];
        }
}
```

3.0.14 *Package Class*

package declaration must appear once in the project (unlike java).

```
package com.company;
class example.MyClass{
    //effective java package is com.company.example.MyClass
}
```

package group and version

```
package com.company:myapp#1.0;
class example.MyClass{
    //effective java package is com.company.example.MyClass
}
```

modules and dependencies //package declaration must appear once in the project (unlike java).

```
1 package com.company:myapp#1.0{
2     import com.lib:other#1.0 for compile;
3     import com.lib:test#1.0  for test;
4 }
```

4
Compilation and Execution

4.1 PreProcessor

//package declaration must appear once in the project (unlike java).

```
1 package com.company:myapp#1.0{
2     //effective java package is com.company.example.MyClass
3 }
```

4.2 Maven Support

4.3 Interpreter

```
1 hl myfile.hl
```

4.4 Java Transpiler/Compiler

```
1 hl -c src/
```

```
1 hl -j src/
```

Part II

Programming with Hadra-Lang

5
IDE Support

5.1 Netbeans Support

blabla

5.2 Intellij Support

blabla

5.3 Eclipse Support

blabla

5.4 Visualcode Support

blabla

5.5 Other Text Tools support

blabla

6
Standard Library

7
Java Integration

Part III

Future Features

7.1 Javascript/Typescript Transpiler

using [jsweet](http://www.jsweet.org)

```
hl --js src/
hl --ts src/
```

7.2 Javascript/Typescript Transpiler

using [j2c](https://bitbucket.org/arnetheduck/j2c/src/default/)
[j2c(alt)](https://github.com/arnetheduck/j2c)

```
hl --js src/
hl --ts src/
```

7.3 Native Compiler

using https://www.graalvm.org

```
hl --native src/
```

7.4 Language Server Protocol

using https://langserver.org

```
hl --slp
```

Bibliography

Index