Mathematics and Computer Science
2017; 2(6): 130-138
http://www.sciencepublishinggroup.com/j/mcs
doi: 10.11648/j.mcs.20170206.16
ISSN: 2575-6036 (Print); ISSN: 2575-6028 (Online)

Calculator on Java Using the Method of Raoelina Andriambololona

Solofonjatovo Evariste Andriamasivelo, Raoelina Andriambololona

Theoretical Physics Department, Institute National des Sciences et Techniques Nucléaires (INSTN-Madagascar), Antananarivo, Madagascar

Email address:
masivelosolofo@outlook.com (S. E. Andriamasivelo), masivelosolofo@gmail.com (S. E. Andriamasivelo), raoelina.andriambololona@gmail.com (R. Andriambololona), raoelinasp@yahoo.fr (R. Andriambololona), jacquelineraoelina@hotmail.com (R. Andriambololona)

To cite this article:
Solofonjatovo Evariste Andriamasivelo, Raoelina Andriambololona. Calculator on Java Using the Method of Raoelina Andriambololona. *Mathematics and Computer Science.* Vol. 2, No. 6, 2017, pp. 130-138. doi: 10.11648/j.mcs.20170206.16

Received: November 20, 2017; **Accepted**: December 4, 2017; **Published**: January 2, 2018

Abstract: The method of Raoelina Andriambololona consists to apply tools from matricial calculation and linear algebra in Arithmetics. This method permits to simplify arithmetic operations and to suppress many existing inconsistencies in writing, in enunciating, and in the four basic operations of arithmetic: addition, subtraction, multiplication and division. The main objective of this article is to use this method to design a computer software using java language. We give here the kernel of the java program corresponding to the calculator, and an overview of the obtained results. This calculator displays the numbers from the left – hand side (L) to the right – hand side (R) in increasing order (i) and does all the operations in this order i.e. it performs the display and calculation in the same direction unlike current calculator in which displaying and calculation are performed in opposite directions. The result is quite consistent with the reading, the writing and all the operations in Malagasy language.

Keywords: Java, Arithmetic, Linear Algebra, Matrix Calculation

1. Introduction

RAOELINA ANDRIAMBOLOLONA has done research to improve the Malagasy scientific language instead of doing a translation from French language. He forged a scientific approach, starting from the concept itself.

The column - matrix layout of the basic vectors, which results from a convention, reveals the only four possible logical and coherent writings of integer and decimal numbers. The choice adopted, and therefore the convention chosen, must be consistent with the way of writing: line writing (LR) from the left-hand side (L) to the right-hand side (R) or line writing (RL) from the right-hand side (R) to the left-hand side (L). Thus, the arrangement of numbers is by construction consistent with writing; it must be the same for the reading, which must obviously follow the writing [1] [2] [4]

In Malagasy language, we write numbers from the left-hand side (L) to the right-hand side (R), and we read it from the right-hand side (R) to the left-hand side (L). This inconsistency between writing and reading in malagasy language has stimulated RAOELINA ANDRIAMBOLOLONA to write several articles on this subject. He designed a rational logical matrix operation technique consistent with the Malagasy spoken numeration of numbers (malagasy people write the number of LRi (increasing order) and read LRd (decreasing order).

In this work, we design a calculator, to understand better the use of this calculation technique, and to become familiar with the notation LRi i.e left hand side (L) to right hande side (R) and increasing (i) order. This calculator does not only display the LRi numbers, but is designed with the calculation basis (illustrated in examples of this article). Raoelina Andriambololona and Hanitriarivo Rakotoson have programed it in javaScript [9]. Nirina Gilbert Rasolofoson and Raoelina Andriambololona have designed an ALU and code converter using matrix calculation [7]. In this paper, we apply this method in framework of computer science using java programming language.

2. Recall of the Method of Raoelina Andriambololona in Arithmetics

[1 - 7]

A number is meaningless unless we give the basis. For

example 1101 makes no sense if we do not specify the number basis, for instance binary or decimal basis.

2.1. Basis, Systems, Writing of Numbers

Let b a fixed natural integer numbers called "basis", and the sequence formed by the powers b^i ($i > 0, i = 0, i < 0$). b^i is called the "basis vector" using the language of linear algebra.

The intrinsic (i.e. independent on the basis) number A may be written on the basis vector b^i as

$$A = \sum_{i=i_1}^{i_2} a_i b^i$$

where the a_i, not all of which are zero, are natural integers strictly less than b. a_i is called the "component" of a on the basis vector b^i.

The uniqueness of a_i is deduced from the linear independence of the basis vectors.

We have put the basic vectors b in column - matrix or row - matrix form by following a well - defined order (either in decreasing (d) order or in increasing (i) order). The row - matrix or column - matrix formed by the components ranked in a well-defined order represents the intrinsic number A in the basis b

We have four choices corresponding to the conventions taken (column - matrix or row - matrix, increasing or decreasing order). We fix our choice, so that the writing of the number is compatible with the writing used (writing on row from left-hand side to right-hand side or from right-hand side to left-hand side).

We mark by a sign (by a comma in French, German, Malagasy,… or by a dot in English), put after the component corresponding to B^0, thus separating the components of a with the positive and negative powers b^i.

The values of a_i, strictly less than the number b, are the system digits.

2.2. Using Matrix Calculation in Arithmetics

2.2.1. Basis $[b^i]$ of $\mathbb{N}$

a. Theorem.

The set $\{b^i\}$ for $i \in I$ is a linear independent generator system of $\mathbb{N}$, that is to say a basis of $\mathbb{N}$.

b. Result

Theorem.

A natural integer A decomposes on the basis $\{b^i\}$ in

$A = \sum_{i=0}^{n} a_i b^i$ and the component a_i is unique (1)

2.2.2. Basis Column - Matrix

We introduce the columns - matrix basis vector

$$\bar{B} = \begin{bmatrix} b^0 \\ b^1 \\ \vdots \\ b^{n-1} \\ b^n \end{bmatrix}$$

(1) maybe written matricially as

$$A = [a_0 \quad a_1 \quad \dots \quad a_{n-1} \quad a_n] \begin{bmatrix} b^0 \\ b^1 \\ \vdots \\ b^{n-1} \\ b^n \end{bmatrix}$$

$$\text{or } A = \underline{E_b(A)}.\bar{B} \qquad (2)$$

where the row - matrix $\underline{E_b(A)}$ is the writing in increasing order of the number A in the numerical basis $[b^i]$.

2.2.3. Remarks

a. The choice of the decomposition has been fixed so as to have the writing of numbers from left-hand side to the right-hand side in increasing order (LRi).

b. Let us note that this arrangement (LRi) may be made to be consistent with the rules of the four operations (addition, subtraction, multiplication and division).

c. *Theorem*

The writing $\underline{E_b(A)}$ is unique.

d. Theorem

All the a_i of (1) and (2) are strictly less than b, ($a_i \in [0, 1, \dots, b-1]$).

the numbers $a_i = (0, 1, 2, \dots., b-1)$ are called the digits of the number in the numerical basis b.

e. We do not need to call upon the Euclidean division of A by b and its properties (in particular, the property of uniqueness and condition $a_i < b$) which is already contained in the decomposition.

f. Theorem

The column - matrix basis defined in the paragraph 2.2.2 is a bijection of the row - matrix on (2).

2.3. Rules of Operation

2.3.1. The Rules of Addition

Let $A = \sum_{i=1}^{n} a_i b^i$ and $A' = \sum_{i=1}^{n'} a'_i b^i$, (2.3.1). we can assume $n' > n$ without losing in generality with $a'_n \neq 0$

$$A + A' = \sum_{i=0}^{n"} (a_i + a'_i) b^i$$

with $a_i = 0$ for $(n+1) \leq i \leq n'$

n " is chosen in such a way that $a_i + a'_i < b$ for all i and in particular for $i = n'$.

All the elements of all the columns of the row - matrix $\underline{E_b(A + A')}$ are zero or strictly less than b.

Let $a''_i = a_i + a'_i$ If $a''_i = a_i + a'_i < b$, then a''_i is the element at the $(i+1)$th column of $\underline{E_b(A + A')}$ from the left.

If $a''_i > b$, we decompose it into $a''_i = a_{i0}b^0 + a_{i1}b^1 + \dots + a'_{is}b^s$ with $a_{i0} < b$

$a''_i b^i = a_{i0}b^i + a_{i1}b^{1+i} + \dots + a_{is}b^{s+1}$ (without summation over i)

The element at the $(i+1)$ - Column from the left of $\underline{E_b(A + A')}$ is a_{i0} while $a_{i1}, a_{i2}, \dots$ are to be "carried over the $(i+2), (i+3), \dots, (i+s+1)$ -th column from the left of $\underline{E_b(A + A')}$ respectively.

The addition is thus carried out from left-hand side to right-hand side in increasing order (LRi).

Example 1

Add in the basis 10 the following intrinsic numbers A and A'

Table 1. *Addition of 126 (621) by 93868 (86.839).*

Our proposal: LRi writing	**International LRd writing**
$A = [1\ \ 2\ \ 6\ \ \cdot\ \ \cdot]\begin{bmatrix}10^0\\10^1\\10^2\\\cdot\\\cdot\end{bmatrix}$	$A = [\cdot\ \ \cdot\ \ 6\ \ 2\ \ 1]\begin{bmatrix}\cdot\\\cdot\\10^2\\10^1\\10^0\end{bmatrix}$
$A' = [9\ \ 3\ \ 8\ \ 6\ \ 8]\begin{bmatrix}10^0\\10^1\\10^2\\10^3\\10^4\\\cdot\end{bmatrix}$	$A' = [8\ \ 6\ \ 8\ \ 3\ \ 9]\begin{bmatrix}\cdot\\10^4\\10^3\\10^2\\10^1\\10^0\end{bmatrix}$
$E_{10}(A) =$ 126..	$E_{10}(A) =$ .. 621
$E_{10}(A') = 93868$	$E_{10}(A') = 86839$
$E_{10}(A + A')$ is obtained as follows (the arrow here below indicates the sense of the operation)	$E_{10}(A + A')$ is obtained as follows (the arrow here below indicates the sense of the operation)
carry over 1010	carry over 0101
126..	.. 621
938.68	86.839
$E_{10}(A + A')$ 064.78	$E_{10}(A + A')$ 87.460
⟹ (direction of operation)	(direction of operation)⟸

Example 2

We suppose that we are in basis 10. The reader will notice the positions of the carry over that is not indicated.

Table 1. *Subtraction of 064 78 (87 460) by126(621).*

Our proposal: LRi writing		**International LRd writing**	
$A"$	064.78	$A"$	87.460
A	126..	A	.. 621
$A' = A" - A$	938.68	$A' = A" - A$	86.839
⟹ (direction of operation)		(direction of operation)⟸	

2.3.2. Rules of Multiplication

Let A and A' be two integers, defined by the decompositions (1)

We are getting

$$A \times A' = \sum_{m=0}^{n"} \left(\sum_{p+q}^{m} a_p a'_q \right) b^m$$

taking into account the multiplication table

$$b^p \times b^q = b^{p+q}$$

we have, as for the addition, the condition

$$0 \leq \sum_{p+q}^{m} a_p a'_q \leq b$$

In calculating the components of product $A \times A'$ that must be less than b, we must carry over the deductions in the subtotals

We will illustrate the calculation rules by explaining the products, subtotals and deductions in

Example 3.

In the basis 10, perform the LRi disposition on multiplication

Table 2. *Multiplication of 518 (815) by 521 6 (6 125).*

5 1 8 × 5 2 1 6	Multiplication of 518 by 5216
"2""0""4"	carry over 518×5
5 7 0 4	Result of 518×5
"1""0""1"	Carry over 518×2
0 3 6 1	Result of 518×2
"0""0"	Carry over 518×1
5 1 8	Result of 518×1
"3""0""4"	Carry over 518×6
0 9 8 4	Result of 518×6
5 7 8. 1 9 9. 4	final Result
"0""0" "0" "1" "1""0"	Carry over final result
⟹ (direction of operation)	

It is easy to check

$$(5 \times 10^0 + 1 \times 10^1 + 8 \times 10^2) \times (5 \times 10^0 + 2 \times 10^1 + 1 \times 10^2 + 6 \times 10^3) = 5 \times 10^0 + 7 \times 10^1 + 8 \times 10^2 + 1 \times 10^3 + 9 \times 10^4 + 9 \times 10^5 + 4 \times 10^6$$

We see that all the operations (writing, adding, multiplying, and dividing) are done from left-hand side (L) to the right-hand side (R) in increasing order (i) or LRi. *Example 4: division of 9932 by 4*

Table 4. *Division of993 2 (2 399) y 4 (4).*

Our proposal: LRi writing								**International LRd writing**							
9	9	3	2		4			2	3	9	9		4		
6	3				9			2	0				5		
3	6	3	2						3	9	9				
↓	6	3		+		9			3	6		+		9	
↓	0	0	2							3	9				
↓		0	2				5			3	6				9
3	.	.	.		9	9	5		.	.	3		5	9	9
⇒	direction of operation							⇒	direction of operation						
⇒	direction of writing							⇒	direction of writing						

3. Calculator on Java Using the Method of Raoelina Andriambololona

In this third part, we will use Malagasy words like marika, isa, fanampiana, fangalana, fampitomboana, fizarana.... to name the classes of digits, numbers, addition, subtraction, multiplication and division.

Same overview of the program excerpt is given

3.1. Calculator in Java

3.1.1. Class Marika

This is the class for the object Marika. Marika is an object of the form $a.b^n$ with a: coefficinet, b base and n: power

Code: Marika. java

```
public class Marika {
  private int base;
  private int coefficient;
  private int degre;
  public int getBase() {
    return base;
  }
  public void setBase(int base) {
    this.base = base;
  }
  public int getCoefficient() {
    return coefficient;
  }
  public void setCoefficient(int
            coefficient) {
    this.coefficient = coefficient;
  }
  public int getDegre() {
    return degre;
  }
  public void setDegre(int degre) {
    this.degre = degre;
  }
  public int getValue() {
    return
     (int)(getCoefficient()*Math.pow(getBase(),
     getDegre()));
  }

  public void printAll() {
    System.out.print(getCoefficient()+".(" +
      getBase() + "^"+getDegre()+")");
  }
  public void print() {
    System.out.print(getCoefficient());
  }


public String toString() {
    return getCoefficient()+"";
  }
  public void print(String s) {
    System.out.print(s+getCoefficient());
  }
  public int get() {
    return getCoefficient();
  }

  public Isa toIsa() {
   Isa isa = new Isa();
   try {
    if(getDegre()<0) {
     isa.addMarika(new Marika(base,0,0));
     int i=1;
     for (;i<Math.abs(getDegre());i++) {
       isa.addMarika(new Marika(base,0,-i));
     }
     isa.addMarika(this);
    } else {
     for (int i = 0; i < getDegre(); i++) {
                isa.addMarika(new Marika(base,0, i));
     }
     isa.addMarika(this);
          }
        } catch (Exception e) {}
        return isa;
    }
    public Marika(int base,int coeff,int degre) {
        super();
        this.base = base;
        this.coefficient = coeff;
        this.degre = degre;
    }
}
```

3.1.2. Class Isa

This is the class for the object Isa. Isa $= a_i.b^i + \ldots . + a_n.b^n$

Code: Isa. class

```
public class Isa {
    private String signe = "";
    private int base;
    private List<Marika>list=new      ArrayList<Marika>();
    private List<Marika> listN = new      ArrayList<Marika>();
    public List<Marika> getList() {
        return list;
    }
    public void setList(List<Marika> list) {
        this.list = list;
    }
    public int getBase() {
        return base;
    }
    public void setBase(int base) {
        this.base = base;
    }
    public int maxDegreN() {
     int max = 0;
     for (int i = 0; i < listN.size(); i++) {
      max= Math.max(
        Math.abs(listN.get(i).getDegre()),max);
     }
     return max;
    }
    public String toString() {
     StringBuilder sb = new StringBuilder();
     checkList();
     sb.append(getSigne());
     int init = maxDegreN()>4?4:maxDegreN();
     for(int i = init; i>0; i--){
       sb.append(getMarika(-i).getCoefficient());
     }
     if(listN.size()>0)sb.append(",");
     for(int i=0; i<list.size(); i++) {
       sb.append(getMarika(i).getCoefficient());
     }
     if(list.size() == 0)sb.append("0");
        return sb.toString();
    }
    public void print() {
        System.out.println(this);
    }
    public void print(String s) {
        System.out.println(s + " " + this);
    }

    public void checkList() {
     while(list.size()>1&&getMarika(list.size()-1)
        .getCoefficient()==0) {
      list.remove(getMarika(list.size()-1));
     }
    }
    public Isa(int marika, int base) {
     super();
     this.base = base;
     int reste;;
```

```
int i = 0;
if(marika<0)signe = "-";
if(marika == 0) {
    list.add(new Marika(base, 0, 0));
    return;
}
marika=Math.abs(marika);
List<Marika>listTmp=new ArrayList<Marika>();
while( marika > base) {
    reste = marika%base;
    listTmp.add(new Marika(base,reste, i++));
    marika/=base;
}
listTmp.add(new Marika(base, marika, i));
for(i=0; i<listTmp.size(); i++) {
    Marika n = listTmp.get(i);
    n.setDegre((listTmp.size()-1) –
            n.getDegre());
    list.add(n);
}
}
public void addMarika(Marika n)
        throws Exception {
    if(findMarika(n.getDegre()) != null)
     throw new Exception("Twice exception");
     if(n.getDegre()>=0) {
        for (int i = 0; i < n.getDegre(); i++) {
            if(findMarika(i)==null)
                addMarika(new Marika(this.base, 0, i));
        }
        list.add(n);
    } else {
        listN.add(n);
    }
}
public Marika findMarika(int degre) {
    if(degre >= 0) {
        for (int i = 0; i < getList().size(); i++) {
            Marika marikaTemp = getList().get(i);
            if(degre == marikaTemp.getDegre()) {
                return marikaTemp;
            }
        }
    } else {
        for (int i = 0; i < getListN().size(); i++) {
            Marika marikaTemp = getListN().get(i);
            if(degre == marikaTemp.getDegre()) {
                return marikaTemp;
            }
        }
    }
    return null;
}
public Isa(String marika, int base) {
    super();
    this.base = base;
    String[] marikas = marika.split(",");
    int entier = marikas.length == 2? 1:0;
    char cs[] = marikas[entier].toCharArray();
    int i = 0;
    if(entier == 0 && cs[0]=='-') {
        setSigne("-");
        i=1;
    }
    for(int j = i; j < cs.length; j++) {
      try {
        addMarika(new Marika(base, Integer.valueOf(cs[j]+""), j-i));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    try {
        cs = marikas[entier-1].toCharArray();
        i  = cs.length;
        int fin = 0;
        if(cs[0]=='-') {
            setSigne("-");
            fin = 1;
        }
        for(int j=(i-1); j>=fin; j--) {
            addMarika(new Marika(base, Integer.valueOf(cs[j]+""), -
(i-j)));
        }
    } catch (Exception e) {
    }
}
public Marika getMarika(int index){
    Marika marika = findMarika(index);
    if(marika != null)
        return marika;
    else
        return new Marika(10, 0, index);
}
public List<Marika> getListN() {
    return listN;
}
public void setListN(List<Marika> listN) {
    this.listN = listN;
}
public void removeMarika(int index) {
    Marika marika = findMarika(index);
    if(index<0){
        listN.remove(marika);
    } else {
        findMarika(index).setCoefficient(0);
    }
    checkList();
}
public void removeLastMarika() {
    if(listN.size()>0) {
        listN.remove(findMarika(-listN.size()));
    } else {
        int n = list.size();
        if(n==1) {
            list.get(0).setCoefficient(0);
            list.get(0).setDegre(0);
            return;
        }
        list.remove(findMarika(0));
        for (int i = 1; i < n; i++) {
            findMarika(i).setDegre(i-1);
        }
    }
}
public Marika getLastMarika() {
    Marika n = null;
    if(listN.size()>0) {
        n = findMarika(-listN.size());
    } else {
        n = getMarika(0);
    }
    return new Marika(n.getBase(), n.getCoefficient(), n.getDegre());
}

public int size() {
    return listN.size() + list.size();
}
public void setMarika(Marika n) {
    Marika marikaTemp = findMarika(n.getDegre());
    if(marikaTemp == null) {
        try {
            addMarika(n);
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            marikaTemp.setCoefficient(n.getCoefficient());
        }
    }
    public String getSigne() {
        return signe;
    }
    public void setSigne(String signe) {
        this.signe = signe;
    }
    public Isa farany(int isa) {
        Isa chiffre = new Isa();
        int n = this.getList().size();
        int m = this.getListN().size();
        for(int j = -m; j < n; j++) {
            try {
                if(isa-->0) {
    chiffre.addMarika(this.getMarika(j));//break;
                } else {
                    if(j>0) break;
                    chiffre.addMarika(new Marika(10, 0, j));
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return chiffre;
    }
    public Isa getDivise(int isa) {
        Isa chiffre = new Isa();
        int j = -getListN().size();
        try {
            for (int i = 0; i < isa && i < size(); i++) {
                chiffre.addMarika(new Marika(10,
getMarika(j++).getCoefficient(), i));
                if(getListN().size() == 0 && getList().size() == 1)return
chiffre;
                if(i == isa-1) {
                    boolean notNull = false;
                    for (int k = j; k < size(); k++) {
    if(getMarika(k).getCoefficient()!=0) {
                            notNull = true;
                            break;
                        }
                    }
                    if(getMarika(j-1).getCoefficient() == 0 && notNull)
chiffre.addMarika(new Marika(10, 1, i+1));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        chiffre.checkList();
        return chiffre;
    }
    public Isa shift(int n) {
        Isa rep = new Isa();
        try {
            int i = -getListN().size();
            for (; i < getList().size(); i++) {
                rep.addMarika(new Marika(10,
getMarika(i).getCoefficient(), i + n));
            }
            for (int j = i + n; j <= 0; j++) {
                rep.addMarika(new Marika(10, 0, j));
            }
        } catch (Exception e) {}
        return rep;
    }
    public Isa() {
        super();
    }

}
```

3.2. The Addition and Subtraction Method

Here is a piece of code on addition and subtraction explained in Part 2.

Code:

```
public static Isa fanampianaP(final Isa A, final Isa B) {
        Isa rep = new Isa();
        int n = A.getList().size();
        int m = B.getList().size();
        int base = B.getBase();
        int k = Math.max(m, n);
        int ret = 0;
        int i = 0;
        try {
            for(; i < k; i++) {
                int a = A.getMarika(i).getCoefficient();
                int b = B.getMarika(i).getCoefficient();
                int c = a + b + ret;
                int d = (int)(c%base);
                ret = (int)(c/base);
                rep.addMarika(new Marika(10, d, i));
            }
            if(ret!=0){
                rep.addMarika(new Marika(base, ret, i));
            }
        } catch (Exception e) {}

        return rep;
    }

    public static Isa fanampianaN(final Isa A, final Isa B) {
        Isa rep = new Isa();
        int n = A.getListN().size();
        int m = B.getListN().size();
        int base = A.getBase();
        if(n == 0 && m == 0) return new Isa(0, base);
        int k = Math.max(m, n);
        int ret = 0;
        try {
            for(int i=k; i>0; i--) {
                int a = A.getMarika(-i).getCoefficient();
                int b = B.getMarika(-i).getCoefficient();
                int c = a + b + ret;
                int d = (int)(c%base);
                ret = (int)(c/base);
                rep.addMarika(new Marika(10, d, -i));
            }
            if(ret!=0){
                rep.addMarika(new Marika(base, ret, 0));
            } else rep.addMarika(new Marika(base, 0, 0));
        } catch (Exception e) {}
        return rep;
    }
public static Isa fanampiana(final Isa A, final Isa B) {
        Isa sommeP = fanampianaP(A, B);
        Isa sommeN = fanampianaN(A, B);
        Isa rep = fanampianaP(sommeN, sommeP);
        rep.setListN(sommeN.getListN());
        return rep;
    }
```

```
public static Isa fangalanaP(Isa A, Isa B) {
    int n = A.getList().size();
    int m = B.getList().size();
    int k = Math.max(n, m);
    int ret = 0;
    if(inferieur(A, B)) {
        Isa r = fangalanaP(B, A);
        r.setSigne("-");
        return r;
    }

    Isa rep = new Isa();
    for(int iTmp=0; iTmp < k; iTmp++) {
        int a = A.getMarika(iTmp).getCoefficient();
        int b = B.getMarika(iTmp).getCoefficient();
        int c = a - b - ret;
        if(c<0) {
            ret = 1;
            c+=10;
        } else ret = 0;
        try {

            rep.addMarika(new Marika(10, c, iTmp));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return rep;
}

public static Isa fangalanaN(Isa A, Isa B) {
    int base = A.getBase();
    int k = Math.max(A.getListN().size(), B.getListN().size());
    int ret = 0;
    if(inferieur(A, B)) {
        return fangalanaN(B, A);
    }

    Isa rep = new Isa();
    try {
        for(int iTmp=k; iTmp>0; iTmp--) {
            int a = A.getMarika(-iTmp).getCoefficient();
            int b = B.getMarika(-iTmp).getCoefficient();
            int c = a - b - ret;
            if(c<0) {
                ret = 1;
                c+=10;
            } else ret = 0;
            rep.addMarika(new Marika(10, c, -iTmp));
        }
        rep.addMarika(new Marika(10, ret, 0));
    }catch (Exception e) {};
    return rep;
}
```

3.3. The Multiplication Method

Here is a piece of code on multiplication explained before.
Code:

```
public static Isa fampitomboanaP(Isa A, Isa B) {
    int n = A.getList().size();
    int m = B.getList().size();
    int base = A.getList().get(0).getBase();
    Isa rep = new Isa("0", 10);
    Marika ret = null;
    for(int jTmp=0; jTmp < m; jTmp++) {
      for(int iTmp= 0; iTmp < n; iTmp++) {
        int degre = A.getMarika(iTmp).getDegre() +
          B.getMarika(jTmp).getDegre();
            Isa c = fampitomboana(A.getMarika(iTmp),
          B.getMarika(jTmp));
        ret = c.getMarika(1);
        if(rep.findMarika(degre) != null) {
          c.getMarika(0).setDegre(degre);
          rep = fanampiana(rep, c.getMarika(degre).toIsa());
                } else {
                    c.getMarika(0).setDegre(degre);
                    try {
rep.addMarika(c.getMarika(degre));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                if(ret.getCoefficient() != 0) {
                    ret.setDegre(degre+1);
                    rep = fanampiana(rep, ret.toIsa());
                }
            }
        }
        return rep;
    }
    public static Isa fampitomboana(Isa A, Isa B) {
        int n = A.getList().size();
        int m = B.getList().size();
        int base = A.getList().get(0).getBase();
        Isa rep = new Isa("0", 10);
        Marika ret = null;
        for(int jTmp= -B.getListN().size(); jTmp < m; jTmp++) {
            for(int iTmp= -A.getListN().size(); iTmp < n; iTmp++) {
                int degre = A.getMarika(iTmp).getDegre() +
B.getMarika(jTmp).getDegre();
                Isa c = fampitomboana(A.getMarika(iTmp),
B.getMarika(jTmp));
                ret = c.getMarika(1);
                if(rep.findMarika(degre) != null) {
                    if(c.findMarika(0).getCoefficient() !=0)
                        rep = fanampiana(rep, new Marika(base,
c.findMarika(0).getCoefficient(), degre).toIsa());
                } else {
                    try {
                        rep.addMarika(new Marika(base,
c.findMarika(0).getCoefficient(), degre));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                if(ret.getCoefficient() != 0) {
                    ret.setDegre(degre+1);
                    rep = fanampiana(rep, ret.toIsa());
                }
            }
        }
        return rep;
    }

    public static Isa fampitomboana(final Marika n1, final Marika n2) {
        Marika n = new Marika(n2.getBase(), n2.getCoefficient(), 0);
        if(n1.getCoefficient() == 0 || n2.getCoefficient() == 0)
            return new Isa(0, n2.getBase());
        Isa rep = new Isa();
        try {
            rep.addMarika(n);
            for (int i = 1; i < n1.getCoefficient(); i++) {
                rep = fanampiana(rep, n.toIsa());
            }
        } catch (Exception e) {}
        return rep;
```

```
    }

    public Isa fampitomboana(final Isa n1, Marika n2) {
        Isa rep = n1;
        for (int i = 1; i < n2.getCoefficient(); i++) {
            rep = fanampiana(rep, n1);
        }
        return rep;
    }
```

3.4. The Division Method

```
public static Isa fizarana(Isa a, final Isa b) {
    List<Isa> restes = new ArrayList<Isa>();
    List<Isa> resultats = new ArrayList<Isa>();
    int stop = 20;
    int iDiv = b.size()+1;
    int apresVirgule = -28;
    try {
        while(!egal(a,new Isa(0, 10)) &&
 superieur(a, new Marika(10, 1, apresVirgule -1)
.toIsa())) {
Isa reponse = new Isa(0, 10);
  int reponseDegre = 0;
  int diviseurDegre = apresVirgule;
  a = fanampiana(a, new Marika(10, 0,
apresVirgule).toIsa());
  Isa a_i = new Isa();
  while(superieur(a.getDivise(iDiv), b) ||
    egal(a.getDivise(iDiv), b)) {
   if(superieur(a.getDivise(iDiv), b) ||
    egal(a.getDivise(iDiv), b)) {
        Isa p = null;
        int iReponse = 9;
        a.getDivise(iDiv).print();
        for (; iReponse >= 0; ) {
        p = fampitomboana(new Isa(iReponse,10), b);
        if(superieur(a.getDivise(iDiv), p) ||
     egal(a.getDivise(iDiv), p))
            break;
            iReponse--;
        }
        reponse = fanampiana(reponse, new
Marika(10, iReponse, reponseDegre++).toIsa());
    p.shift(-a.getListN().size())
        a = fangalana(a, p.shift(-
    a.getListN().size()));
        Marika a_iMarika = a.getLastMarika();
     a_iMarika.setDegre(diviseurDegre++);
                    a_i.addMarika(a_iMarika);
                    a.removeLastMarika();
            }
        }
            //a_i.print("ai_0");
            Marika a_iMarika = a.getLastMarika();
            if(a_iMarika.getCoefficient() != 0) {
                a_iMarika.setDegre(diviseurDegre);
                a_i.addMarika(a_iMarika);
            }
            //a_i.print("ai_i");
            restes.add(a_i);
            resultats.add(reponse.shift(apresVirgule));
            a = a_i;
            try{
                a.addMarika(new Marika(10, 0, 0));
            } catch (Exception e) {}
            reponse.shift(apresVirgule);
            a.print();
                b.print();
            }
        } catch (Exception e) {e.printStackTrace();}
        for (int i = 0; i < restes.size(); i++) {
            restes.get(i).print();
        }
    }
```

3.5. Calculator Interface

The calculator using the LRi disposition and developed under java is illustrated in this part. In the figures 1, 2, 3 and 4

We put in brackets the numbers in international LRd writing in the figures 1, 2, 3 and 4

Figure 1. *Addition of 756 (657) by 89 (98).*

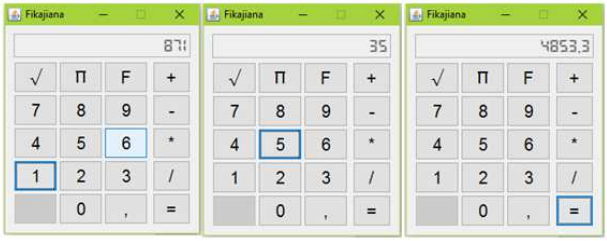

Figure 2. *Division of 871 (178) by 35 (53).*

Figure 3. *Subtraction of 24 (42) by 09 (90).*

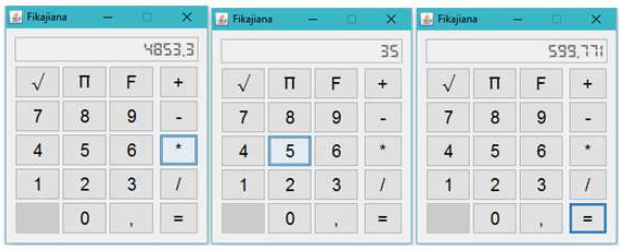

Figure 4. *Multiplication of 4853.3 (3.3584) by 35 (53).*

4. Conclusion

Let us point out that the malagasy people read the numbers in increasing order LRi even if they write them in decreasing order LRd. The investigation to look for to improve this

inconsistency has led Raoelina Andriambololona to use matrix calculation in arithmetics. The latter makes clear the writing and arrangement of operations on numbers. The LRi disposition is consistent with the rules of operations (addition, multiplication, subtraction, division), which cannot be the case of the international LRd writing of numbers with the usual rules of operations. Enunciating (reading of numbers) must be consistent with the writing of numbers too. This fact has encouraged us to write down a program on java, which is a free software, object-oriented, easy to learn, and platform-independent. It can be moved easily from one computer to another and it has the ability to run the same program on many different systems.

References

[1] Raoelina Andriambololona, “Théorie générale des numérations écrite et parlée". Bull. Acad. Malg. LXIV./1-2, Antananarivo, Madagascar, 1986.

[2] Raoelina Andriambololona, "Théorie générale des numérations écrite et parlée. II Utilisation du calcul matriciel en arithmétique. Nouvelle proposition d’écriture, d’énoncé des règles d’addition et de multiplication des nombres.". Bull. Acad. Malg LXV/1-2, Antananarivo, Madagascar, 1987.

[3] Raoelina Andriambololona, “Théorie générale des numérations écrite et parlée. II- Utilisation du calcul matriciel en arithmétique. Application au changement de bases de numération. Bull. Acad. Malg. LXV./1-2, Antananarivo, Madagascar ”, 1987 (1989).

[4] Raoelina Andriambololona, Ravo Tokiniaina Ranaivoson, Wilfrid Chrysante Solofoarisina. Arithmetic and Matricial Calculation. Pure and Applied Mathematics Journal. Vol. 5, No. 3, 2016, pp. 82-86. doi: 10.11648/j.pamj.20160503.14.

[5] Raoelina Andriambololona, Hanitriarivo Rakotoson “Mpikajy elekronika sy siantifika mampiasa ny fomba fanisana Malagasy (Electronic and scientific calculator based on malagasy counting method)”, communication at the Academie Malgache, Antananarivo Madagascar, 05 June 2008.

[6] Raoelina Andriambololona, “Algèbre linéaire et multilinéaire”, Collection LIRA, INSTN-Madagascar, Antananarivo, Madagascar, 1986.

[7] Nirina Gilbert Rasolofoson, Raoelina Andriambololona. Design of ALU and Code Converter Using Matrix Calculation. Pure and Applied Mathematics Journal. Vol. 6, No. 3, 2017, pp. 89-100. doi: 10.11648/j.pamj.20170603.11

[8] Claude Delannoya Programmer en Java. Édition Eyrolles - 940 pages, 9eédition, 12 juin 2014ISBN10.

[9] Herbert Java: The Complete Schildt Reference, Ninth Edition ISBN-13: 978-0071808552 ISBN-10: 0071808558| 2014.

[10] Herbert Schildt Java: The Complete Reference, Tenth Edition (Complete Reference Series) 10th Edition| New York: McGraw-Hill Education, 2017.| LCCN 2017036734 | ISBN-13: 978-1259589331 (paperback) ISBN-10: 1259589331.

[11] Use of Matrix Calculation in Arithmetics and Numeration Theory for Scientific Calculator Design Raoelina Andriambololona, R. Hanitriarivo. HEP MAD’07 International Conference, Antananarivo, Madagascar, 10 -15 Septembre 2007.

[12] H. Anton and C. Rorres. Elementary linear Algebra. Applications Version. John Wiley & Sons, Inc. New York, 2000.

[13] D. Danielson. Vectors and Tensors in Engineer and Physics. Addison-Wesley Publishing Company, 1992.

[14] Daniel GUIN. Algèbre: Tome 2, Anneaux, modules et algèbre multilinéaire. Enseignement sup. EDP Sciences, 2013.