IIP (E.T.S. de Ingeniería Informática)
Academic Year 2020-2020

# *Lab activity 6. Iteration: creation of an utility class*

*This lab practice lasts two sessions*

IIP teachers
Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

## Contents



## 1 Context and previous work

Sometimes, working processor has no predefined mathematical funcions that are needed. In that case, processor may lack of trigonometrical or logarithm functions, that are needed for usual calculations, such as position and movement in robotic elements, drones, etc. This fact is very usual when the processor has low capacities, which is frequent in microcontrollers (usual devices in keyboards, mice, low performance cell phones, etc.). When this happens, it is necessary to implement a version of the needed mathematical functions.

Taking that situation into account, this lab activity will be devoted to solve some numerical problems that employ iterative structures (loops). You will have to implement a utility class that will implement methods for calculating (in an approximated way) two well-known functions: square root and logarithm.

After implementing the functions, you must test their correction by comparing them with the predefined versions of Java `Math` library. Finally, you will have to graphically represent the two functions by using the Java library provided at lab activity 5.

Before the lab session, the student must read the lab report and try to solve (as far as possible) the proposed problem. The example 8.6 and the exercise 21 of the chapter 8 of the book "Empezar a

programar usando Java" ($3^{rd}$ edition)[1]. Both are good examples on how to solve this kind of problems. To take the best profit of the lab session, it is recommended that the student brings his/her own implemented code (a draft or the complete solution).

### Initial activity

You must create a new package `pract6`, corresponding to this activity, in the `iip` project that includes the lab activities of the subject. Once the above is done, you have to include, in that package, the Java classes given as help material. There is a class `IIPMath` where you will include in some Java methods to calculate the functions proposed below.

## 2 Problem 1: square root

The following recurrence allows to calculate the square root $t$ of a given non-negative number $x$:

$$t_1 = \frac{1+x}{2}, \qquad t_{i+1} = \frac{t_i + \frac{x}{t_i}}{2}$$

This recurrence is based on a sequential approximation, $t_1, \ldots t_n$, to the desired value (square root of $x$), where the last term ($t_n$) is the closest value to $x$ of all the terms ($t_i$) generated in the process.

### 2.1 Precision and termination

The succession $t_1, \ldots, t_n$ can be proved to converge strictly to the square root of $x$. Thus, the error strictly decreases in each new calculated term.

Given two consecutive terms, $t_{i-1}$ and $t_i$ (with $i > 1$), its difference can be taken as an error measure of the $i$th term.

Therefore, when the calculation of the square root allows a certain maximum error, the iterative process must finish when the difference between two consecutive terms is lower than the allowed error[2].

Following the previous statement, the termination criteria for the iteration is: when the difference between two consecutive terms is lower than the desired error.

### Activity #1

a) Given the previous recurrence, implement in the `IIPMath` class a public method that allows to calculate the square root of a non-negative real number $x$ with a precision $\epsilon$. The header must be:

```
/** Returns the square root of x >= 0 with a precision epsilon > 0 */
public static double sqrt(double x, double epsilon)
```

b) Implement another public method in the `IIPMath` class that overloads the previous one and calculates the square root of a non-negative real number $x$ with a precision of 1e-15, using a call to the previous method. The header must be:

```
/** Returns the square root of x >= 0 with a precision 1e-15 */
public static double sqrt(double x)
```

c) The code must be properly documented. Thus, apart from the comment that describes the activity of the method, the description of the parameters and the return value (by using the tags `@param` and `@return`, respectively), must be included. For example, the previous method may be documented as follows:

```
/** Returns the square root of x >= 0 with a precision 1e-15.
 *  @param x. Value, x >= 0.
 *  @return double. Square root of x with a maximum error 1e-15.
 */
```

d) To check the correction of the code you can employ the BlueJ Code Pad, comparing the results of the `IIPMath.sqrt(double)` method and the `Math.sqrt(double)` method provided by Java.

[1] Example 9.6 and exercise 24 in the $2^{nd}$ edition.

[2] Take into account that the maximum error is limited by the precision of the system or the number of digits used.

# 3 Problem 2: logarithm of a value

For calculating the natural logarithm of any value $x \in \mathbb{R}^+$ the following series development can be used. This development allows to calculate the logarithm of a given $z$ with $\frac{1}{2} \leq z < 1$. Given:

$$y = \frac{1-z}{1+z} \tag{1}$$

it is known that:

$$\log(z) = -2\sum_{i=1}^{\infty} \frac{y^{2i-1}}{2i-1} \tag{2}$$

Representing the $i$-th term ($1 \leq i$) of the previous sum development by $u_i$, then:

$$\log(z) = -2(u_1 + u_2 + u_3 + \cdots + u_k + u_{k+1} + \cdots + u_n) - R_n$$

that is, all the series can be represented as a sum of terms $u_i$, along with a remainder $R_n$ that represents the sum of the rest of terms (those after the $n$-th term). In other words:

$$log(z) = -2\sum_{i=1}^{n}(u_i) - R_n$$

## 3.1 Calculating the terms

The method to be implemented must calculate each of the terms $u_i$ of the previous expression. By substituting in (2), the first term $u_1$ can be obtained, with $u_1 = y$. By substituting again in (2), it can be obtained that for any two consecutive terms $u_k$ and $u_{k+1}$ the following expressions:

$$u_k = \frac{y^{2k-1}}{2k-1} \qquad u_{k+1} = \frac{y^{2k+1}}{2k+1}$$

It can be seen from these expressions that the following relation appears between any two consecutive terms:

$$u_{k+1} = y^2 \frac{2k-1}{2k+1} u_k \tag{3}$$

As can be seen, it is possible to save calculations by obtaining each new term from the previous one, instead of using independent calculations.

## 3.2 Precision and termination

The features of the series allows to demonstrate that the total error $R_n$ is always lower than the value of the last calculated term $u_n$.

Therefore, when the desired error[3] must be lower than a certain $\epsilon$, it is enough with calculating a term after the other (accumulating them) until one of the terms has a value lower than that $\epsilon$.

## Activity #2

Having into account the recurrence in (3), implement in the `IIPMath` class a public method for calculating the logarithm of a value $z$, $\frac{1}{2} \leq z < 1$, with a maximum error value $\epsilon$, following this header:

```
/** Returns log(z), 1/2 <= z < 1, with an error epsilon > 0. */
public static double logBase(double z, double epsilon)
```

[3] Again, take into account that the error is limited by the precision of the numerical representation you are using.

### 3.3 General logarithm calculation

Knowing the previous calculation (that allows to calculate logarithm of a value in the interval $[\frac{1}{2}, 1[$, let's see how to apply it for any value $x \in \mathbb{R}^+$.

Given a non-negative value $x$, it is possible to transform it into a value $z$ in the interval $[\frac{1}{2}, 1[$ by dividing or multiplying $x$ as many times as needed by 2 (depending on $x \geq 1$ or $x < \frac{1}{2}$).

That is, $x$ can be expressed as:

$$x = 2^m z \qquad \left(\frac{1}{2} \leq z < 1\right) \tag{4}$$

where $m$ is an integer value (positive or negative). After applying logarithms to both parts of the equation, the result is:

$$\log(x) = m \cdot \log(2) + \log(z) \tag{5}$$

Thus, $\log(x)$ can be calculating by using the following processes:

1. If $x \in [\frac{1}{2}, 1[$ apply method `logBase(double, double)` directly.
2. If $x \notin [\frac{1}{2}, 1[$, then:
   (a) If $x \geq 1$, to make it accomplish Equation (4), $z = \frac{x}{2^m}$; thus, $z$ can be calculated from $m$ making divisions by 2 $m$ times, until a value in $[\frac{1}{2}, 1[$ is obtained (the $z$ value).
   (b) If $x < \frac{1}{2}$, $z$ and $m$ can be calculated by successive multiplications of $x$ by 2 (instead of divisions). It can be employed as well that $-\log(\frac{1}{x}) = \log(x)$ (e.g., for calculating $\log(0.2)$ $(0.2 < \frac{1}{2})$, since $-\log(5) = \log(0.2)$, the previous process can be used to calculate $\log(5)$ and the obtain $\log(0.2)$.
3. Apply Equation (5) for, given $\log(z)$, calculating the desired logarithm. Take into account that $\log(2)$ is a known value, whose approximation is given in the following activity.

### Activity #3

a) Define the approximated value for $\log(2)$ (0.6931471805599453) as a constant for your class.

b) Using the expressions in Equations (4) and (5), implement in the `IIPMath` class a public method for calculating the logarithm of a given positive value $x$, with a maximum error $\epsilon$, with header:

```
/** Returns log(x), x > 0, with an error epsilon > 0. */
public static double log(double x, double epsilon)
```

c) Implement in the class `IIPMath`, using the previous method, another public method that overloads the previous one, for calculating the logarithm of a a given positive value $x$, with a maximum error 1e-15, with header:

```
/** Returns log(x), x > 0, with error 1e-15. */
public static double log(double x)
```

d) As previously, properly document each method, with emphasis in what they do. Employ the `@param` and `@return` tags to describe their parameters and result.

## 4 Test class

You can compare the functions you implemented with those provided by the Java language in its library class `Math`. In that way, you can check the precision of your results.

To make easier this comparison, you are given the `IIPMathTest` class that you downloaded into the `pract6` package, and that allows you to compare, for a value `x` given by the user, the result of calculating its square root and its logarithm by using your implementation (`IIPMath.sqrt(double)` and `IIPMath.log(double)`) with the Java predefined methods (`Math.sqrt(double)` and `Math.log(double)`).

### Activity #4

Execute the class `IIPMathTest` to check that the results of your implemented functions are correct (comparable with those of Java standard library) with different values, both small (e.g., about $10^{-12}, 10^{-6}, 10^{-3}$, large (e.g., about $10^3$, $10^6$, $10^{12}$), and values in the range of units and tens. You can test any other value that you want.

## 5 Graphical representation of functions

To graphically show functions like those you implemented, you will use the graphic library (`Graph2D` class in `graph2D` package) that you installed in lab activity 5. Remember that `Graph2D.html` file provides the documentation for this library, which allows to make graphical representations of points and lines in a bidimensional space.

Figure 1 shows a sample of this representation done by using dot-by-dot drawing of two functions ($\sin(x)$ and $\sin(x)/x$) in the interval $[-1, 4\pi]$.

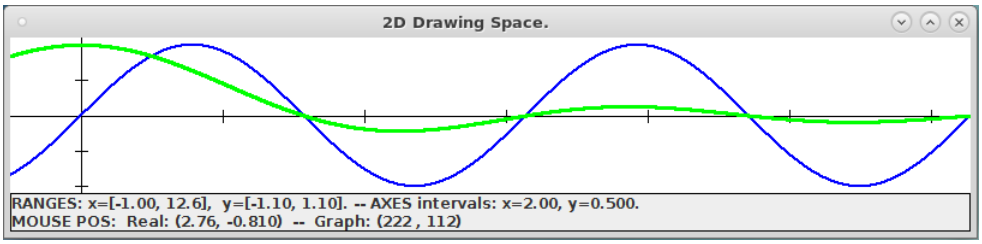


Figure 1: Dot-by-dot represenation of $\sin(x)$ and $\sin(x)/x$ in $[-1, 4\pi]$.

A dot-by-dot drawing for a function $f$ is obtained by calculating for the different possible values of $x$ (in the desired interval) the corresponding $f(x)$ values, and representing graphically the pair $(x, f(x))$ by using the corresponding primitive graphical operation (method `drawPoint(double, double)` of class `Graph2D`).

An example of use of `graph2D.Graph2D` is provided in the file `Graph2DTest.java`, that must be included into your `pract6` package and that, when executed, it will show a window similar to that of Figure 1. You can employ it as basis for solving the following activities.

Additionally, this report has a final appendix that describes how to graphically represent one of the functions ($\sin(x)/x$) in a similar way to that which appears in the file `Graph2DTest.java` and which is represented in Figure 1.

### Activity #5

Represent graphically in the interval $x \in [-1, 15]$ and $y \in [-3, 4]$, using dots, the values for the functions that you developed (square root and logarithm). In Figure 2 you have a sample on the expected result.

### Activity #6

Represent graphically in the interval $x \in [-1, 15]$ and $y \in [-3, 4]$, using lines. That is, you must join each two consecutive dots in the graph of the values of the two functions you developed, by using a line.

**TIP:** In both activities, you must take into account that square root is not defined in $[-\infty, 0[$ and logarithm is not defined in $[-\infty, 0]$.

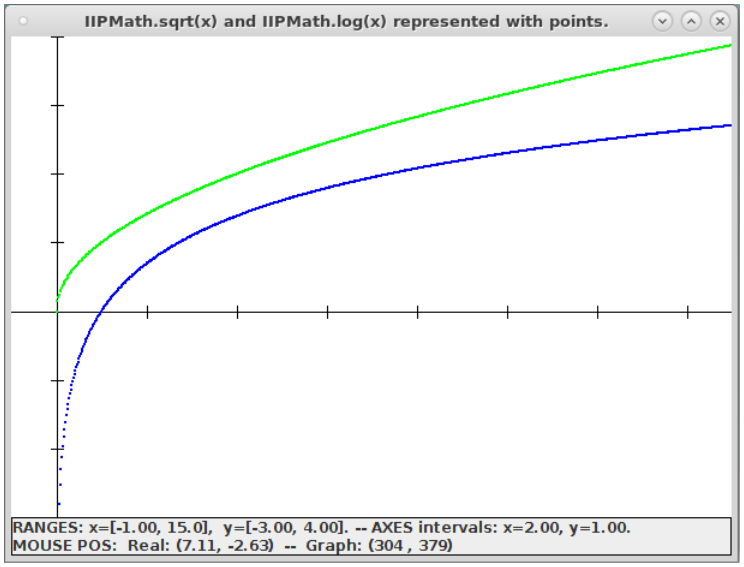


Figure 2: Representation of `IIPMath.sqrt(x)` and `IIPMath.log(x)` in $[-1, 15]$.

# 6 Appendix: Example for using the class `Graph2D`

As an example, the use of `Graph2D` to represent the function $\sin(x)/x$ is shown, as presented in Figure 1.

In first term, at the beginning of the Java file in where the program is implemented, you must include the import for that graphical representation class:

```
// Import class Graph2D (graph2D package).
import graph2D.Graph2D;
```

After this, you can define objects of this class and use them in the class to be developed. For the sake of simplicity, we will employ the constructor with less parameters: minimal and maximal values of $x$ and $y$. To make easier its posterior use, we define variables with these extreme values, that is:

```
// Define value ranges for x and y
double xMin = -1;
double xMax = Math.PI * 4;
double yMin = -1.1;
double yMax = +1.1;
// Create drawing space with desired dimensions
Graph2D gd1 = new Graph2D(xMin, xMax, yMin, yMax);
// Change thickness of elements to 3 (default is 1):
gd1.setThickness(3);
// Change color to GREEN for drawing from now on
gd1.setForegroundColor(Color.GREEN);
```

Afterwards, all possible `x` values (in their interval) must be taken and the point `(x, Math.sin(x) / x)` must be represented. For that, value of `x` must be incremented in small increments from its initial value to its final one. A `delta` variable is used for this:

```
// Calculate increment for each x step (delta)
double delta = (xMax - xMin) / Graph2D.INI_WIDTH;
```

where constant `Graph2D.INI_WIDTH` represents the number of points of the abscissa axis in the graphical window (800).

Finally, each possible `x` value is taken, the corresponding function value for that point is calculated, and that pair of coordinates is graphically represented:

```
// Take each point in x, calculate f(x), draw (x, f(x))
for (double x = xMin; x <= xMax; x = x + delta) {
    double y = Math.sin(x) / x;     // y is the value of the function
    gd1.drawPoint(x, y);            // represent (x, y)
}
```

The result of executing the previous code is show in Figure 3.

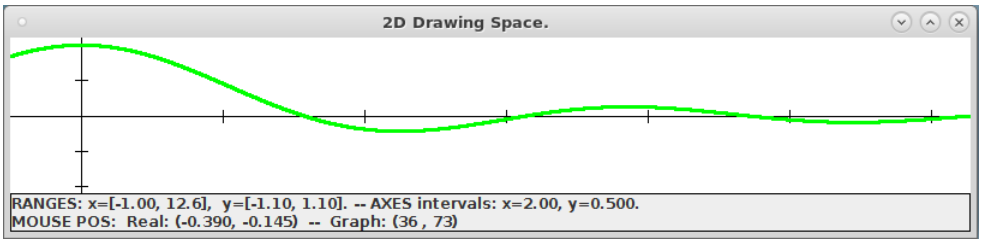


Figure 3: Representation of $\sin(x)/x$ en $[-1, 4\pi]$.