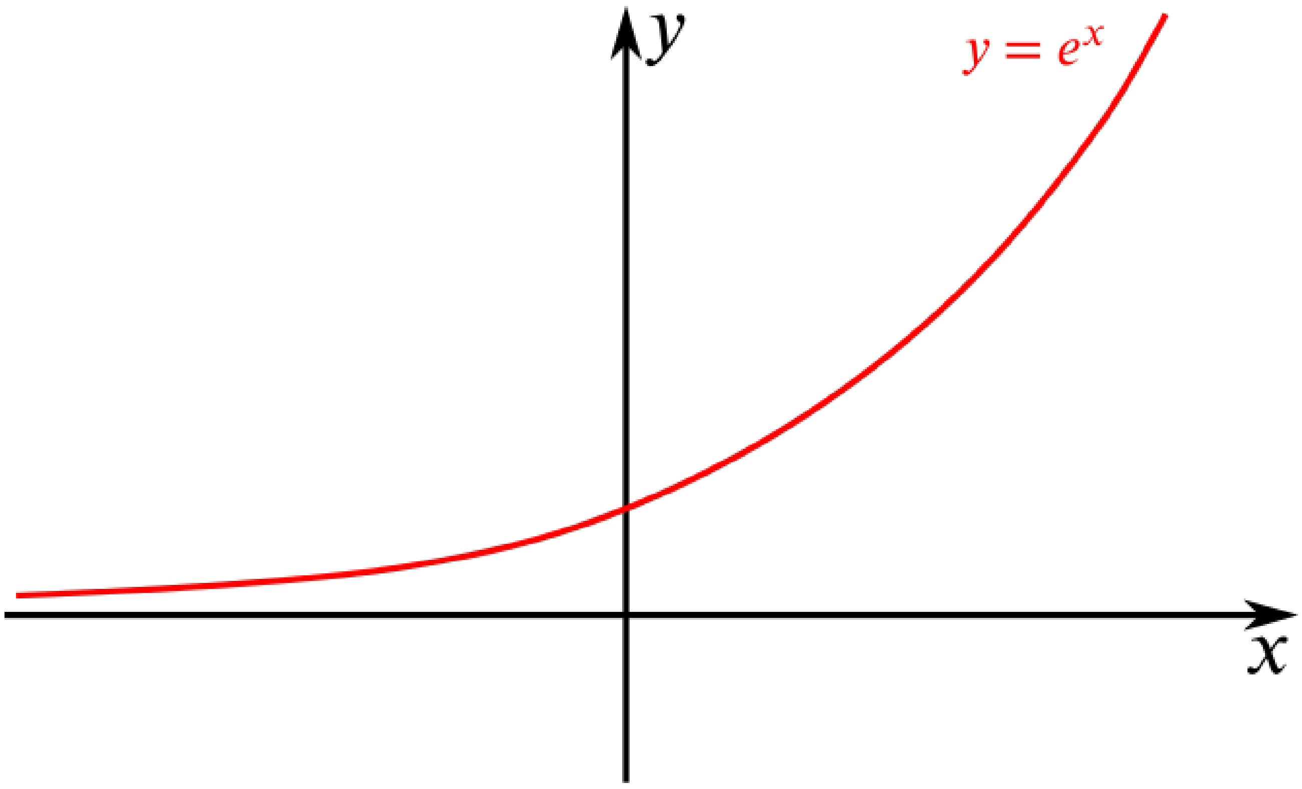
CAL 2 Report

Testing the exponential function



Ingeniería Informática – Software Engineering CAL2

Óscar García Azagra – DNI: 09123306B

Table of Contents

1 – Introduction

This document is used to explain some of the aspects of the assignment. As the delivered files will also contain an explanatory video, there will be some concepts that won’t be explained in detail here (mainly explanation of classes and methods).

2 – The exponential function

The natural exponential function is the function which is equal to its own derivative. This function is continuous in all the domain of x, and the value of it ascends with the x.

For this practice, even though the function is continuous, it was decided the x could only take natural numbers.

Considering that, this function can be implemented in a simple way:

- We give the result an initial value of 1 (Since $e^0 = 1$)
- If x >= 0: Then we multiply the result x times by e (easily implemented with a loop)
- If x < 0: We get the result we would get doing e^{-x}, and we would get the inverse number ($1/e^{-x}$)

This function has some characteristics that will be used in order to test if the function is implemented correctly:

- $e^0 = 1$
- $e^1 = e$
- e^n with n > 1 grows quickly approaching infinity
- e^m with m < 1 decreases approaching 0

3 – JUnit tests

In order to answer the first question we need to create tests to check whether the function works correctly or not. In order to do that, we have created the following methods

- ExponentialOf0(): It should return 1
- ExponentialOf1(): It should return e
- ExponentialOf3(): It should return $e^3 \approx 20.0855$
- ExponentialOfDouble(): It should throw an exception, since Real Numbers aren't allowed in this implementation
- ExponentialOfMinus2(): It should return $e^{-2} \approx 0.1353$

We use the method ***assertEquals****(expectedValue, actualValue, delta)*

- expectedValue: The value the function should return, according to the mathematical calculus (done with a calculator, for example)
- actualValue: The value the function returns
- delta (optional): The error allowed between those values

In case the values are the same (considering the delta), it will compile at work. Otherwise, the program will crash, and a message will be displayed showing both values

There's an exception created, which is thrown if the number is not an integer

4 – Mock tests

For this section of the Assignment, a new class must be created. This class will be an interface, which will represent an unfinished class that it's needed for the program to work

A function is created in the already existing Class, which uses that interface. This function checks if the number is greater than 1 or not.

The most important methods in this section are:

- expect().andReturn()
- replay()
- verify()

Those two are used to (1) record what the program will return and (2) check if it's what it's actually happening.

5 - Using the Chidamber Kemerer Java Metrics tool

Using this tool, we can get some useful information about our code. “d”

To use it we have to use the following commands

```
cd “Path to the CKJM tool”\ckjm-1.9\build

java -jar ckjm-1.9.jar “Path to the project folder”\Exponential\target\classes\group17\*.class
```

The second command displays some numbers on the screen next to each of the class name.

Class Name	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM
ExponentialFunction	3	1	0	1	7	3	0	3
NumberChecker	2	1	0	0	2	1	1	2

In order to know what this number mean, we have to look at this tool’s documentation.

- WCM: Weighted Methods per Class: It represents the sum of the complexities of the methods
  - On ExponentialFunction it is 3, because
    - compute has a complexity of 1
    - checkNumber has a complexity of 2, since they call to another method
  - On NumberChecker it is 2 because there are 2 methods of complexity 1
- DIT: Depth of Inheritance Tree: Since there is no inheritance in any classes, and all classes inherit from Object by default, the result is 1
- NOC: Number of Children: Since there are no children of any class, the result is 0
- CBO: Coupling between object classes: It represents the number of classes coupled to a given class.
    - On ExponentialFunction it is 1, because the method checkNumber calls a method from another class
  - On NumberChecker it is 0 there aren’t method calls to external classes
- RFC: Response For a Class: It measures the number of different methods that can be executed when a method is invoked
  - On ExponentialFunction it is 7, because the method is called 7 times:
    - 5 times by the JUnits
    - 2 times by the Mock testing method
  - On NumberChecker it is 2 because the method is called 2 times, both of them in the Mock testing method (inside the checkNumber method)
- LCOM: Lack of COhesion in Methods: It counts the sets of methods that are not related through the sharing of some of the class fields
- Ca: Afferent Couplings: Measures how many other classes use the specific class
  - On ExponentialFunction it is 0, because this class isn’t used by any other
  - On NumberChecker it is 1 because this class is used in a method in ExponentialFunction
- NPM: Number of Public Methods: The number of public methods in each class