**LABORATORIO # 1**

|  |  |
| --- | --- |
| Carrera | : Ciencias de la computación |
| **Curso** | : Ingeniería de software |
| **Fecha** | : |

1. COMPETENCIAS

* Comprende el ciclo de vida ágil
* Realiza pruebas unitarias del software
* Comprende el TDD

1. REFERENCIAS CONCEPTUALES

**• Unit Testing**

**• Unit Testing and Mocking**

**• http://www.nunit.org/**

**•** [**https://stackoverflow.com/questions/2665812/what-is-mocking**](https://stackoverflow.com/questions/2665812/what-is-mocking)

1. DESARROLLO DEL LABORATORIO

## Parte I – Las pruebas Unitarias

* Historia y conceptos
* Tipos y lenguajes
* Mocking

## Parte II – Primeras implementaciones

* Cree un proyecto **De Pruebas** en el IDE de su preferencia
* Programe la Unidad de pruebas para evaluar el método calcularRaiz(x) de la siguiente clase:



* El cálculo debe calcular la raíz cuadrada para número enteros, cero y racionales. En el caso de números negativos, deberá devolver -999999 como resultado.

|  |  |
| --- | --- |
| X | Raíz(X) |
| 0 | 0 |
| 64 | 8 |
| 0.01 | 0.1 |
| -6 | Produce una excepción ó -999999 |

Assert.areequal(esperado, calculado)

Assert.ThrowsException<Exception>(obj.raíz(-6))

* Utilice el método assert.areequal (o similar) para las pruebas. Depende de su plataforma y lenguaje de programación.
* Programe una prueba para cada tipo de raíz
* Implemente la clase Raices con su método calcularRaiz, para que pase las pruebas definidas en la parte 1

## Parte III

Hacer un programa con lo siguiente:

1. Crear una unidad de pruebas que haga las pruebas[[1]](#footnote-1) a la clase **Convertidor**



1. Programar la clase Convertidor (ver diseño) para que pase las pruebas de a)

## Parte IV

* Cree un proyecto **De Pruebas** en el IDE de su preferencia
* Programe la Unidad de pruebas para evaluar el método ***distancia*** de la siguiente clase:

..

* El cálculo debe calcular la distancia entre los puntos x1 y x2. Cuando la distancia sea nula debe producir una excepción.
* Utilice el método assert.areequal para las pruebas
* Programe una prueba para cada cuadrante y para distancia cero.
* Implemente la clase Geométrica con su método distancia, para que pase las pruebas definidas en la parte I

## Parte V

Programe las pruebas para la clase **StringHelper**, que extrae los nombres y apellidos de los alumnos desde un String que viene en el formato: “<Nombre-1>/<Nombre-2>/<Apellido Paterno>/<Apellido Materno>”.

La clase tiene los métodos:

apaterno(input:String):String

amaterno(input:String):String

nombre1(input:String):String

nombre2(input:String):String

Si el alumno tiene un solo nombre aparece así: Raúl Pérez Del Águila => “Raúl//Pérez/Del Águila”

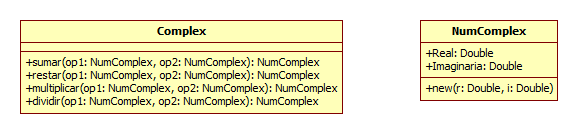
## Parte VI

Existen métodos en la unidad de pruebas que se pueden ejecutar antes de cada prueba y después de finalizada la prueba. Estos métodos preparan todo el escenario para que la prueba se ejecute (como inicializar variables, abrir bases de datos, generar datos etc)

En la XUnit eran <Setup> y <TearDown> que se ejecutaban previo y al finalizar cada prueba respectivamente. Usarlos en esta parte.

Hacer un programa con lo siguiente:

* Crear una unidad de pruebas que haga las pruebas[[2]](#footnote-2) a la clase **Complex**, que deberá hacer las 4 operaciones aritméticas básicas con números complejos usando la clase **NumComplex** como parámetros y como resultado (ver diseño)
* Programar la clase Complex y hacer que pase las pruebas unitarias



## Suma

Para sumar números complejos, se siguen las normas básicas de la aritmética, sumando los reales con los reales y los imaginarios con los imaginarios:


   (a + bi) + (c + di) =
   (a + c) + (b + d)i \,


## Resta

Al igual que en la suma, se opera como con los números reales ordinarios:


   (4 - 2i) - (3 + 5i) =
   
   (4 - 3) + (-2i - 5i) =
   (4 - 3) + (-2 - 5)i =
   1 - 7i  \,
: 


   (6-4i) - (6+5i) =

   (6-6) + (-4i-5i)=
   0 - 9i =
   -9i \,



## Multiplicación

Para multiplicar dos números complejos, se multiplica cada término del primero por los dos del segundo, con lo que obtenemos 4 términos (propiedad distributiva de la multiplicación):


   (a + bi)(c + di) =
   ac + adi + bci + bdi^2 =
  (ac - bd) + (ad + bc)i \,


## División

Así, la división de dos números complejos, la multiplicamos y dividimos por el conjugado del denominador, que es lo mismo decir que multiplicamos dividendo (numerador) y divisor (denominador), por el conjugado del divisor (denominador).


   \frac{a + bi}{c + di} =
   \frac{(a + bi)(c - di)}{(c + di)(c - di)} =
   \frac{ac -adi +bci + bd}{c^2 -cdi +cdi +d^2} =



    = \frac{ac + bd +(bc - ad) \; i}{c^2 + d^2} =
   \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad} {c^2 + d^2} \; i


**IV. ENUNCIADOS A RESOLVER**

* ¿Se pueden simular transacciones?
* ¿Se pueden simular interacciones con una base de datos?

**V. TAREA ASÍNCRONA**

* Sobrecargue el método calcularRaiz, para que acepte y devuelva una lista de números



* Sobrecargue el método calcularRaiz, para que acepte un arreglo de números:   
  v[ ] : Double y devuelva un arreglo con las raíces de cada elemento. Sugerencia: use el método collectionAssert (o similar) de la unidad de pruebas.
* Investigue y haga una implementación de Mock Objects para extender las pruebas y simular interacciones con otros sistemas o componentes. Por ejemplo, simular que se graba un dato en una tabla DB y después se verifica que se grabó haciendo la consulta correspondiente. O Simular que se consume un servicio web.

1. El método assert.areequal(esperado, calculado, *precisión*). El tercer parámetro *precisión* representa la precisión de la comparación (Nro de decimales). [↑](#footnote-ref-1)
2. El método assert.areequal(esperado, calculado, *precisión*). El tercer parámetro *precisión* representa la precisión de la comparación (Nro de decimales). [↑](#footnote-ref-2)