**Universidad de Buenos Aires**

**FACULTAD DE INGENIERÍA**

**Departamento de Computación**

**75.10 – Técnicas de diseño**

**TRABAJO PRÁCTICO 2**

*ANÁLISIS DEL REQUERIMIENTO – DISEÑO DE UNA FRAMEWORK*

**Curso:** *2013 – 2do Cuatrimestre*

**Turno:** *Jueves*

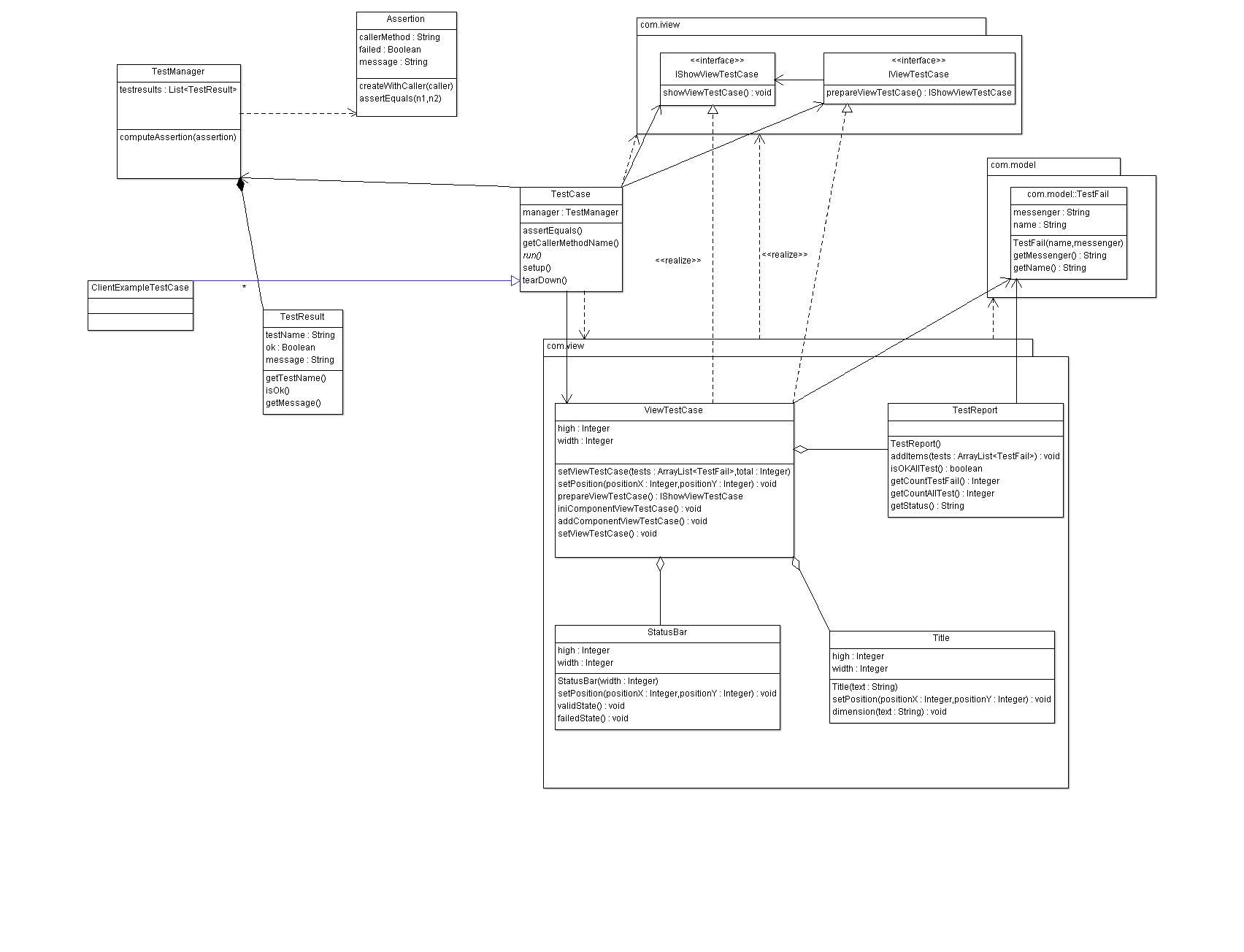
|  |  |  |
| --- | --- | --- |
| **GRUPO Nº 13** | | |
| **APELLIDO, Nombre** | **Número de Padrón** | **e-mail** |
| *Barrea, Ignacio* | *86225* | *ignacio@tictaps.com* |
| *Chavar, Hugo* | *90541* | *hechavar@gmail.com* |
| *Schmoll, Edward Erik* | *90135* | *erikschmoll@gmail.com* |
|  |  |  |
|  |  |  |
|  |  |  |
| **Fecha de inicio:** ***17-10-2013*** |
| **Fecha de Aprobación:** |
| **Firma Aprobación:** |

|  |
| --- |
| **Observaciones:** |

Objetivo

Producir código basados en los principios de la programación orientada a objetos, que éste sea descriptivo sin necesidad de comentar cada linea, haciendo un desarrollo basándonos en las buenas prácticas con la ayuda de herramientas como GIT, GITHUB, MAVEN, entre otras.Diagrama de Clases

Diagrama de secuenciasCasos de Prueba



Contamos con tres casos:

1. Prueba Unitaria
   1. Usando FrameWork JUNIT
2. Prueba Productiva
   1. Un proyecto “Calculator” esta dispuesto a utilizar nuestra FrameWork TestCase

2.a.

1. Creación de los casos de prueba

CP 1

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la suma de 2 con 2 |
| **RESULTADO ESPERADO** |
| La prueba debe finalizar correctamente |

CP 2

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la multiplicación de 2 con 10 |
| **RESULTADO ESPERADO** |
| La prueba debe finalizar correctamente |

CP 3

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la resta entre 5 y el resultado de hacer la multiplicación de 4 con 2 |
| **RESULTADO ESPERADO** |
| La prueba debe finalizar correctamente |

CP 4

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la división de 10 con 5 esperando como resultado un 0 |
| **RESULTADO ESPERADO** |
| La prueba debe finalizar con una falla |

1. Implementación de los casos de prueba

CP 1

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la suma de 2 con 2 |
| **RESULTADO ESPERADO** |
| …  @Override  **public** **void** run() {  testAdd();  }  **public** **void** testAdd() {  Calculator calculator = **new** Calculator();  assertEquals(4.0, calculator.addAwithB(2, 2));  }  …  Según lo especificado: La prueba debe finalizar correctamente  Prueba superada |

CP 2

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la multiplicación de 2 con 10 |
| **RESULTADO ESPERADO** |
| @Override  **public** **void** run() {  testMult();  }  **public** **void** testMult(){  Calculator calculator = **new** Calculator();  assertEquals(20.0, calculator.mulAwithB(2, 10));  }  Según lo especificado: La prueba debe finalizar correctamente  Prueba sueprada |

CP 3

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la resta entre 5 y el resultado de hacer la multiplicación de 4 con 2 |
| **RESULTADO ESPERADO** |
| @Override  **public** **void** run() {  testIntegral();  }  **public** **void** testIntegral() {  Calculator calculator = **new** Calculator();  calculator.rememberResul(calculator.mulAwithB(4, 2));  assertEquals(-3.0, calculator.minusAwithAccumulator(5));  }  Según lo especificado: La prueba debe finalizar correctamente  Prueba sueprada |

CP 4

|  |
| --- |
| **PRERREQUSITOS** |
| Crear un Calculator y hacer la división de 10 con 5 esperando como resultado un 0 |
| **RESULTADO ESPERADO** |
| @Override  **public** **void** run() {  testDivide();  }  **public** **void** testDivide(){  Calculator calculator = **new** Calculator();  assertEquals(0.0, calculator.divideAwithBparts(10, 5));  }  Según lo especificado: La prueba debe finalizar con una falla  Prueba sueprada |

Manual de usuario

Para utilizar el servicio que brinda esta Framekork usted debe crear una clase dode escribirá todos los métodos que desea probar y heredar de TestCase, la misma le obligará a usted a crear el método run, en este tiene que hacer los llamados de todos sus métodos que desea probar en la misma ejecución.

Usted va a poder gozar de los siguientes métodos que le proporciona esta FrameWork para validad los métodos de su clases a probar.

|  |  |  |  |
| --- | --- | --- | --- |
| Metodos | Parámetros | Valor de retorno | Descripción |
| start | N/A | void | Inicia el testeo |
| getCallerName | N/A | String | Obtiene el nombre del método que lo llamo |
| verifyTest | testName | boolean | Verifica si el método existe |
| assertTrue | condition | Void | Valida si una expresión booleana es verdadera |
| assertFalse | condition | Void | Valida si una expresión booleana es falsa |
| assertEquals | ObjA, ObjB | Void | Compara dos objetos del tipo OBJECT |
| assertEquals | EnteroA, EnteroB | Void | Compara dos enteros |
| assertEquals | DecimalA, DecimalB | Void | Compara dos decimales |
| fail | N/A | Void | Genera una excepción |
| setup | N/A | Void | Setea una única configuración para un set de test a probar |
| tearDown | N/A | void |  |

Método RUN, en este método se van a llamar todos los metodosTest que querramos probar en el mismo conjunto de ejecución:

Por ejemplo

**public** **class** TestCalculator **extends** TestCase{

@Override

**public** **void** run() {

testAdd();

}

**public** **void** testAdd() {

Calculator calculator = **new** Calculator();

assertEquals(4.0, calculator.addAwithB(2, 2));

}

}

Código Fuente