**Universidad de Buenos Aires**

**FACULTAD DE INGENIERÍA**

**Departamento de Computación**

**75.10 – Técnicas de diseño**

**TRABAJO PRÁCTICO 2.1**

*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:** ***7-11-2013*** |
| **Fecha de Aprobación:** |
| **Firma Aprobación:** |

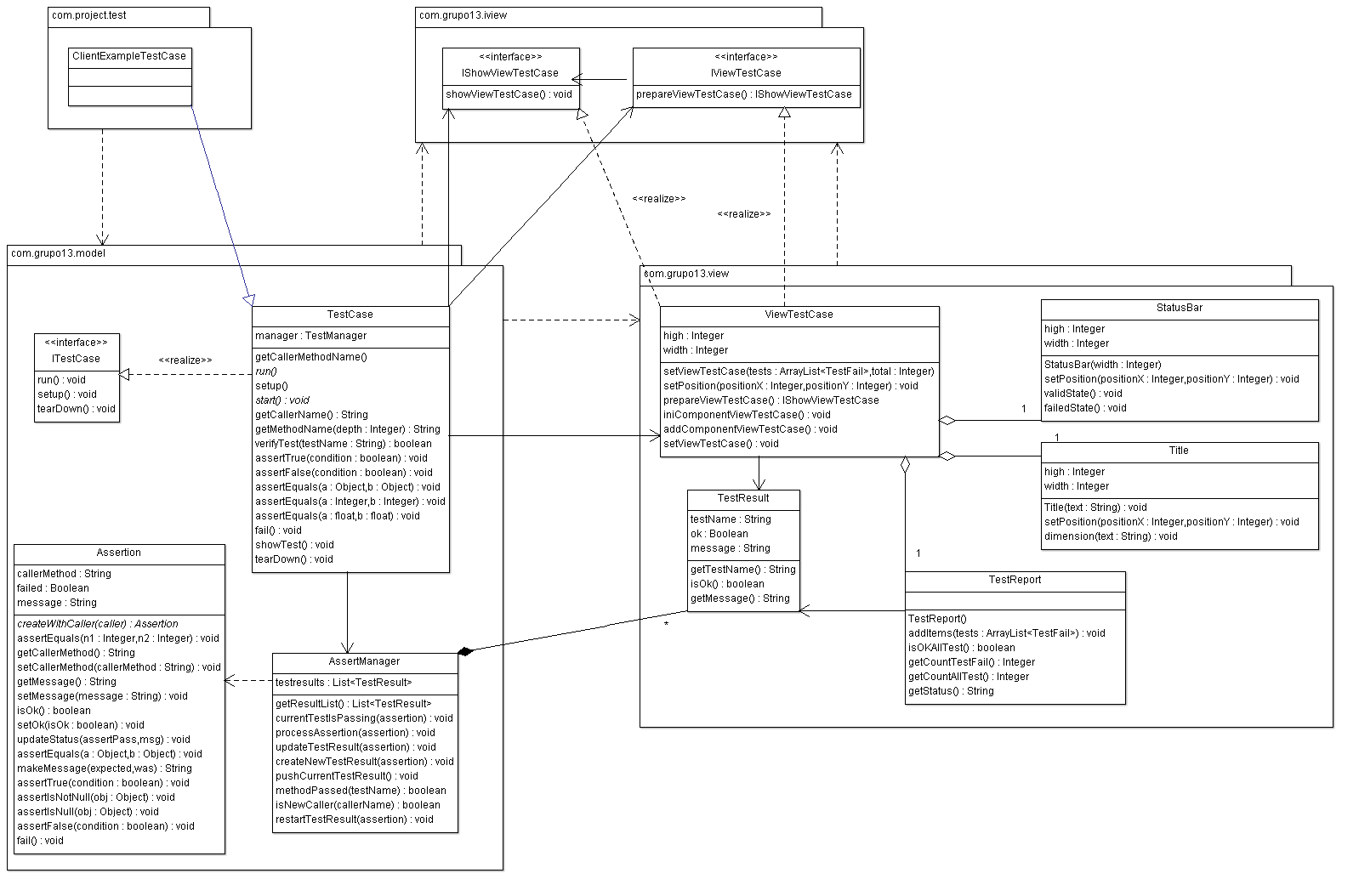
|  |
| --- |
| **Observaciones:** |

Objetivo

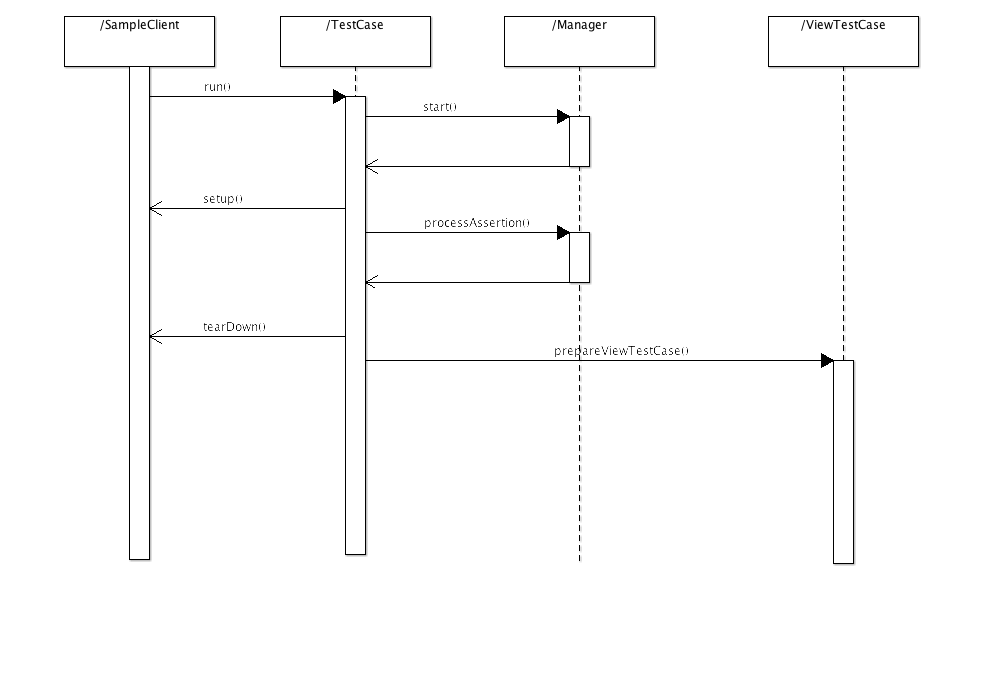
Producir código basados en los principios de la programación orientada a objetos, que éste sea descriptivo sin necesidad de comentar cada linea, haciendo un desarrollo basándonos en las buenas prácticas con la ayuda de herramientas como GIT, GITHUB, MAVEN, entre otras.

Diagrama de Clases

Diagrama de secuencias



Casos 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 este 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 aprovechar los siguientes métodos que le proporciona este FrameWork para validar 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. Opcional |
| tearDown | N/A | void | Se ejecuta al finalizar el TestCase para finalizar y limpiar las instancias necesarias. Opcional |

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));

}

}

Metodos Setup y TearDown pueden ser utilizados para inicializar instancias antes de ejecutar los tests, y limpiar posteriormente en caso de ser necesario.

Por Ejemplo

@Override

**public** **void** setup(){

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

}

@Override

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

testAdd();

}

**public** **void** testAddOnePlusTwo() {

assertEquals(3.0, calculator.addAwithB(1, 2));

}

**public** **void** testAddOnePlusOne() {

assertEquals(2.0, calculator.addAwithB(1, 1));

}

@Override

**public** **void** tearDown(){

calculator = **null**;

}

Responsabilidades de Clases:

* AssertManager: Esta es responsible de recibir los resultados de todos los asserts ejecutados individualmente relacionarlos con el método que lo ejecutó. Luego, genera una lista con los “resultados” de cada método para ser enviados a la Vista.
* TestResult: Es el objeto resultado de cada test corrido, generado por el Manager y enviado a la Vista.
* Assertion: Determina si una evaluación es verdadera, guarda el resultado, y, en caso de que haya fallado la evaluación almacena un mensaje explicativo.
* BaseTestCase: Es el punto de entrada al framework de tests, el usuario debe heredar de una clase derivada de esta e implementar los métodos abstractos junto con sus propios tests y el método público estático main.
* TestCase: Hereda de BaseTestCase redefiniendo un método para mostrar los resultados en forma gráfica o por consola, posee una referencia a la vista en la cual publica los resultados.
* ViewTestCase: Es la vista encargada de recibir el array de resultados “TestResult” y los muestra en una interfaz grafica.

Ejemplo de uso de Regex:

Al utilizar la clase heredada de TestSuite, se puede settear un String conteniendo una Expresion Regular, que se usara para validar el nombre del test a ser ejecutado.

Si el nombre concuerda con la Expresion Regular el test será agregado al conjunto de pruebas, de lo contrario será ignorado.

Se puede settear la RegEx a usar mediante el metodo “setRegex(String)” , y debe ser configurado previamente a llamar al start().

Poejemplo se podria utilizar para correr solo los tests que incluyan la palabra “Null” de la siguiente manera

public static void main(String[] args) {

TestSuiteCliente someTest = new TestSuiteCliente ();

someTest.setRegex("(.\*)Null(.\*)");

someTest.start();

someTest.showTest();

….