Qué se necesita para una prueba unitaria

Necesitamos:

Netbeans 8.x y Java 8
Librería JUnit 4.x (Viene con Netbeans)
Librería Hamcrest 1.3 (Viene con Netbeans)
Plugin JaCoCoverage

Para instalar este plugin en Netbeans debemos:

– Abrir el menú -> Herramientas -> plugins. Seleccionar la pestaña Plugins disponibles.

– Buscar y seleccionar TikiOne JaCoCoverage, instalar.

– Nos pedirá reiniciar el IDE, hazlo.

Paso 2: El proyecto

Creamos un proyecto que llamaremos «Prueba de Cobertura», agregamos un paquete «example» y las librerías arriba mencionadas. Para finalizar, creamos una clase «Miclase.java«.

cobertura ide

Paso 3. El código

El código que probaremos, lo estudiamos en un post anterior [LINK] donde realizamos una prueba manual del algoritmo. El algoritmo corresponde al problema de detectar el numero mayor dados tres números.

algoritmia

Pasamos del algoritmo a código y tenemos:

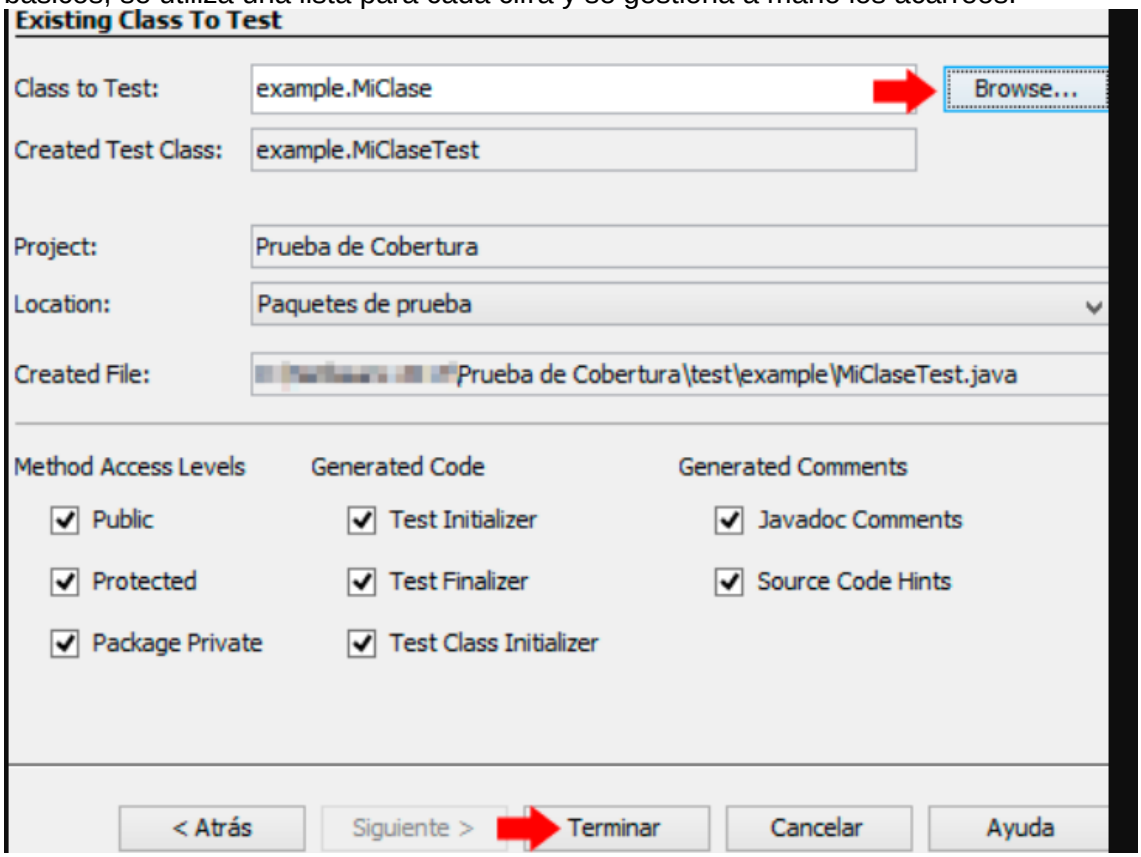
```
package example;
/**
 * @see https://www.jc-mouse.net/
 * @author mouse
 */
public class MiClase {

    public int numero_mayor(int a, int b, int c) {
        if (a > b && a > c) {
            return a;
        } else if (c > b) {
            return c;
        } else {
            return b;
        }
```

Paso 4: Paquete de Prueba

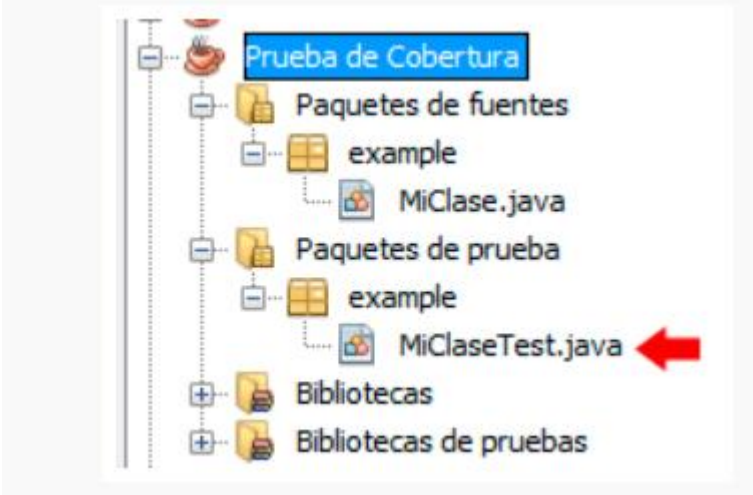
Ya que tenemos nuestra clase de prueba lista, procedemos a crear las clases de prueba. Clic derecho sobre nuestro proyecto -> New -> Other… En Categoría seleccionamos «Unit Tests» y en File Types, seleccionamos «Test for Existing Class» y presionamos el botón siguiente.

A continuación, presionando el botón «browse…» buscamos y seleccionamos la clase que testearemos, «Miclase«, dejamos todas las opciones tal cual están y finalizamos presionando el botón «Terminar».

El siguiente código muestra para calcular la potencia de dos de cualquier número con cualquier cantidad de cifras. Para evitar desbordamientos de tipos básicos, se utiliza una lista para cada cifra y se gestiona a mano los acarreos.



Se creara otra clase en la sección de «**Paquete de prueba**» como se ve a continuación



Paso 5: JUnit

Antes de realizar la prueba con JUnit, debemos estudiar un poco el código generado. En la parte inferior de la clase MiClaseTest encontramos el siguiente método:

```
@Test
 2      public void testNumero_mayor() {
 3          System.out.println("numero_mayor");
 4          int a = 0;
 5          int b = 0;
 6          int c = 0;
 7          MiClase instance = new MiClase();
 8          int expResult = 0;
 9          int result = instance.numero_mayor(a, b, c);
10          assertEquals(expResult, result);
11          // TODO review the generated test code and remove the
default call to fail.
12          fail("The test case is a prototype.");
13      }
```

donde:

1) @Test: Los métodos marcados con esta anotación, le indican a JUnit que es código que queremos que se ejecute. Son estos métodos donde se implementan el código de pruebas.

4,5,6) Las variables del método que probaremos

7) Instancia a nuestra de prueba

8) El resultado que esperamos obtener si la prueba tiene exito

9) la llamada al método de prueba, el resultado se almacena en otra variable

10) assertEquals: Es uno de varios métodos con los que cuenta JUnit, este método en particular compara si dos objetos son iguales, de no ser así, lanzará una excepción y la prueba se detiene.

12) fail: Este método hace que la prueba termine con fallo

Como vimos en un post anterior [Caja blanca: Prueba del camino básico], este algoritmo consta de 4 caminos posibles, eso quiere decir que debemos implementar 4 métodos de prueba.

grafos caminos

Partiendo como base el método de prueba arriba mencionado, creamos 4 métodos con sus respectivos valores de entrada y valores esperados, es decir:

```
    @Test
    public void testNumero_mayor_caso1() {
        int a = 5;
        int b = 3;
        int c = 7;
        MiClase instance = new MiClase();
        int expResult = 7;
        int result = instance.numero_mayor(a, b, c);
        assertEquals(expResult, result);
    }

    @Test
    public void testNumero_mayor_caso2() {
        int a = 5;
        int b = 3;
        int c = 4;
        MiClase instance = new MiClase();
        int expResult = 5;
        int result = instance.numero_mayor(a, b, c);
        assertEquals(expResult, result);
    }

    @Test
    public void testNumero_mayor_caso3() {
        int a = 5;
        int b = 7;
        int c = 6;
        MiClase instance = new MiClase();
        int expResult = 7;
        int result = instance.numero_mayor(a, b, c);
        assertEquals(expResult, result);
    }

    @Test
    public void testNumero_mayor_caso4() {
        int a = 5;
        int b = 7;
        int c = 9;
        MiClase instance = new MiClase();
        int expResult = 9;
        int result = instance.numero_mayor(a, b, c);
        assertEquals(expResult, result);
    }
```

Para ejecutar el test con JUnit, clic derecho sobre la clase MiClaseTest.java -> run file

Observamos que pasamos las 4 pruebas con éxito.

unity test

Si tuviésemos algún error, este aparecería en color rojo.

Paso 6. JaCoCoverage

Como mencionamos en un principio, este plugin nos permite ver más a detalle lo que pasa con el código y los caminos que este toma. Para ejecutarlo, clic derecho sobre el

proyecto -> Test with JaCoCoverage. Se abrirá una pagina en el navegador que tengamos configurado por defecto.

El siguiente código muestra para calcular la potencia de dos de cualquier número con cualquier cantidad de cifras. Para evitar desbordamientos de tipos básicos, se utiliza una lista para cada cifra y se gestiona a mano los acarreos.

```
public static String povOf2(int exp) {
    if (exp == 0)
        return "0";
    List<Integer> digits = new ArrayList<Integer>();
    digits.add(1);
    int size = 1;
    int tmp;
    int ac = 0;
    //Primer bucle
    for (int i = 0; i < exp; i++) {
        // Segundo bucle
        for (int c = 0; c < size; c++) {
            tmp = digits.get(c);
            tmp *= 2;
            tmp += ac;
            ac = 0;
            if (tmp > 9) {
                tmp -= 10;
                ac = 1;
            }
            digits.set(c, tmp);
        }
        // Segundo if
        if (ac == 1) {
            digits.add(ac);
            size++;
            ac = 0; // (1)
        }
    }
    String s = "";
    // Tercer bucle
    for (int i = digits.size()-1; i >= 0; i--) {
        s += digits.get(i);
    }
    return s;
}
```

¿Cuáles serían las pruebas que haría? El objetivo es buscar el número mínimo de pruebas que sea relevante, es decir, que prueben condiciones distintas.