Actividad Complejidad Ciclomática

Decimos que un número es perfecto si la suma de los divisores de ese número (exceptuándose él mismo) es igual al propio número. Por ejemplo:

- El 6 es un número perfecto ya que la suma de sus divisores (1+2+3) es 6.
- El 28 es un número perfecto ya que la suma de sus divisores (1+2+4+7+14) es 28.

```
13   public boolean esPerfecto(int j)
14   {
15       int suma;
16       boolean resultado;
17       int i;
18
19       suma=0;
20       resultado=false;
21       // buscamos todos los divisores de número
22       for (i=1; i<j ;i++)
23          {
24          if (j%i == 0)
25          { // i es divisor de numero
26              suma = suma + i;
27           } /* if */
28       } /* for i */
29       if (suma==j)
30       {
31           resultado=true;
32        }/* if */
33        return resultado;
34   } // esPerfecto
```

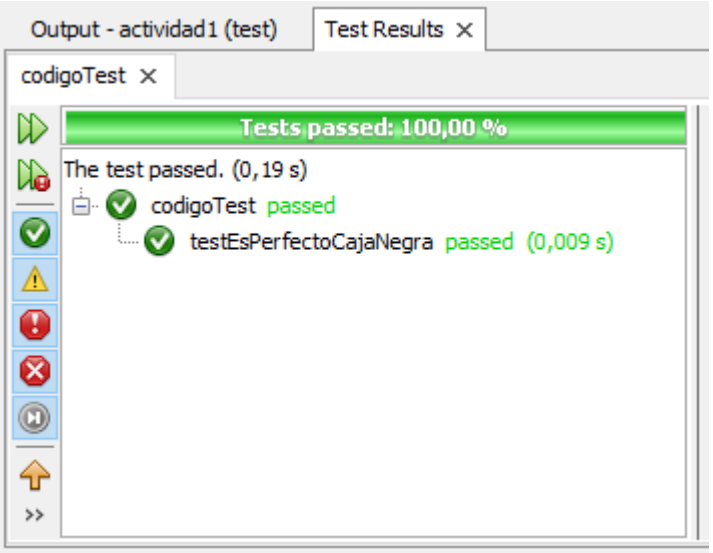
El anterior fragmento de código recibe como entrada un número j y retorna true si j es perfecto y false en caso contrario:

Para este fragmento de código se pide:

1. **Diseñar pruebas de caja negra mediante una de las técnicas estudiadas, e implementarlas mediante JUnit.**

En este caso, como nos devuelve un booleano, el resultado es binario: true / false. Probamos un par de casos de cada posible resultado.

```
41      @Test
42      public void testEsPerfectoCajaNegra() {
43          System.out.println("esPerfecto");
44          codigo instance = new codigo();
45
46          assertEquals(false, instance.esPerfecto(1));
47          assertFalse(instance.esPerfecto(3));
48          assertEquals(true, instance.esPerfecto(6));
49          assertTrue(instance.esPerfecto(28));
50
51      }
```

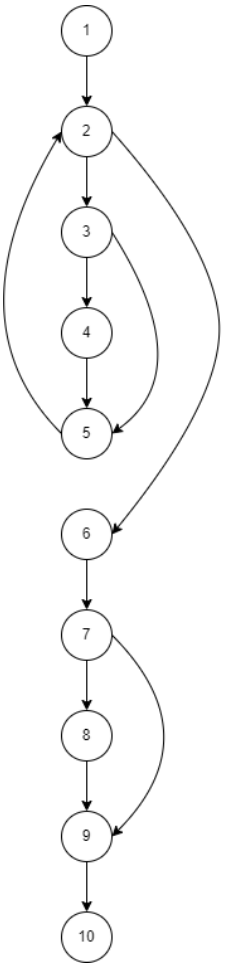


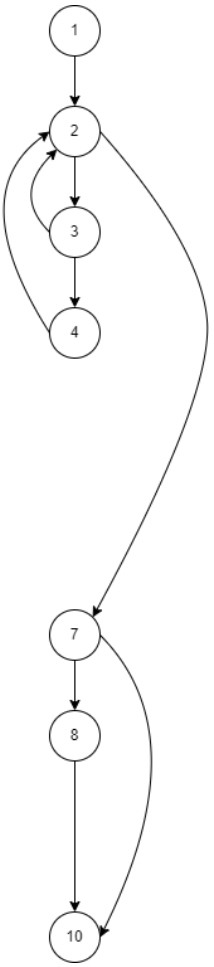
2. Diseñar pruebas de caja blanca mediante la técnica de cobertura decaminos para lo que se hará:

2.1. Representar el grafo.

```
9   public boolean esPerfecto(int j) {
10      int suma;
11      boolean resultado;
12      int i;
13
14      suma = 0;                                   // instrucción 1    - NODO 1
15      resultado = false;                          // instrucción 2
16
17      // buscamos todos los divisores de un número
18      for (i = 1; i < j; i++) {                   // instrucción 3    - NODO 2
19          if (j % i == 0) { // i es divisor del numero  // instrucción 4  - NODO 3
20              suma = suma + i;                    // instrucción 5    - NODO 4
21          } // fin if                            // instrucción 6    - NODO 5
22      } // fin for                               // instrucción 7    - NODO 6
23
24      if (suma == j) {                            // instrucción 8    - NODO 7
25          resultado = true;                       // instrucción 9    - NODO 8
26      } // fin if                                // instrucción 10   - NODO 9
27
28      return resultado;                           // instrucción 11   - NODO 10
29  } //fin metodo esPerfecto
```

El grafo completo (y simplificado) que nos saldría sería el siguiente:





2.2. Determinar la complejidad ciclomática del programa usando las distintas técnicas descritas.

CC = ARISTAS – NODOS + 2

Grafo completo: CC = 12 – 10 + 2 = 4

Grafo simplificado: CC = 9 – 7 + 2 = 4

CC = REGIONES CERRADAS + 1

Grafo completo: 3 regiones cerradas; CC = 3+1 = 4

Grafo simplificado: 3 regiones cerradas; CC = 3+1 = 4

CC = PREDICADOS SIMPLES + 1

Grafo completo: nodos con dos salidas = 3; CC = 3+1 = 4

Grafo simplificado: nodos con dos salidas = 3; CC = 3+1 = 4

2.3. Determinar los caminos básicos.

4 caminos básicos (cogemos como referencia el grafo simplificado):

1-2-7-10

1-2-7-8-10

1-2-3-2-7-10

1-2-3-4-2-7-10

¿Cuáles son los casos de prueba que siguen estos caminos?

1-2-7-10 → 1

1-2-7-8-10 → 0

1-2-3-2-7-10 → (Siempre que pasemos al nodo 3, vamos a pasar al menos una vez al nodo 4) Podemos probar con 2, por ejemplo

1-2-3-4-2-7-8-10 → 6

```
53          @Test
54      public void testEsPerfectoCajaBlanca() {
55          System.out.println("esPerfecto");
56          codigo instance = new codigo();
57
58          assertEquals(false, instance.esPerfecto(1));
59          assertTrue(instance.esPerfecto(0));
60          assertEquals(false, instance.esPerfecto(2));
61          assertTrue(instance.esPerfecto(6));
62
63      }
```