**ANÁLISIS DE SOFTWARE**

**2019 – 1er Cuatrimestre**

Casos de uso

Triángulo

Integrantes

|  |  |  |
| --- | --- | --- |
| Sanchez | Fernando | 36822171 |
| Lorenz | Lautaro Enrique | 37661245 |
| Tejada | Sabrina | 37790024 |
| Riquelme | Mariano | 36287422 |
| Villca | Luis Alberto | 35277730 |
| Adrián | Morel | 34437202 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Caso | Lado 1 | Lado 2 | Lado 3 | Resultado obtenido | Resultado esperado |
| Escaleno Valido | 3 | 4 | 5 | Escaleno | Escaleno |
| Equilátero Valido | 2 | 2 | 2 | Equilátero | Equilátero |
| Isósceles Valido | 5 | 5 | 8 | Isósceles | Isósceles |
| Isósceles Valido Permutación 1 | 5 | 8 | 5 | Isósceles | Isósceles |
| Isósceles Valido Permutación 2 | 8 | 5 | 5 | Isósceles | Isósceles |
| Escaleno Valido Permutación 1 | 4 | 3 | 5 | Escaleno | Escaleno |
| Escaleno Valido Permutación 2 | 4 | 5 | 3 | Escaleno | Escaleno |
| Un lado con valor 0 | 0 | 1 | 2 | Fin del programa | Fin del programa |
| Un lado con valor negativo | -1 | 2 | 3 | Fin del programa | Fin del programa |
| Suma de 2 lados igual al restante | 2 | 2 | 4 | No es Triangulo | No es Triangulo |
| Suma de 2 lados igual al restante Permutación 1 | 2 | 4 | 2 | No es Triangulo | No es Triangulo |
| Suma de 2 lados igual al restante Permutación 2 | 4 | 2 | 2 | No es Triangulo | No es Triangulo |
| Suma de 2 lados mayor al restante | 3 | 3 | 4 | Isósceles | Isósceles |
| Suma de 2 lados mayor al restante permutación 1 | 3 | 4 | 3 | Isósceles | Isósceles |
| Suma de 2 lados mayor al restante permutación 2 | 4 | 3 | 3 | Isósceles | Isósceles |
| Todos los lados con valor 0 | 0 | 0 | 0 | Fin del programa | Fin del programa |
| Valores no enteros | 1,25 | 4,3 | 5,6 | Fin del programa | Fin del programa |
| Numero incorrecto de valores | 2 | 2 |  | La interfaz no lo permite | La interfaz no lo permite |

CASOS DE USO

CÓDIGO FUENTE

**App.java**

**package** triangulo;

**import** java.util.Scanner;

**import** triangulo.Triangulo;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// 9223372036854775807 es el numero mas grande soportado por el long

**long** ladoA,ladoB,ladoC;

**boolean** procesar = **true**;

System.***out***.println("El programa finaliza al ingresar un caracter incorrecto");

**try** {

System.***out***.println("----------------------------------------------");

Scanner sc = **new** Scanner(System.***in***);

**while**(procesar) {

System.***out***.print("ingresa un numero entero positivo: ");

ladoA = sc.nextLong();

**if**(ladoA <= 0)

**break**;

System.***out***.print("ingresa un numero entero positivo: ");

ladoB = sc.nextLong();

**if**(ladoB <= 0)

**break**;

System.***out***.print("ingresa un numero entero positivo: ");

ladoC = sc.nextLong();

**if**(ladoC <= 0)

**break**;

Triangulo t1 = **new** Triangulo(ladoA,ladoB,ladoC);

**if**(!t1.esTriangulo())

System.***out***.println("No es triangulo");

**else**

System.***out***.println("el triangulo ingresado es " + t1.getTipoTriangulo());

}

sc.close();

} **catch**(Exception e) {

} **finally**{

System.***out***.println("FIN DEL PROGRAMA");

}

}

}

**Triangulo.java**

**package** triangulo;

**public** **class** Triangulo {

**private** **long** ladoA;

**private** **long** ladoB;

**private** **long** ladoC;

**public** Triangulo(**long** a, **long** b, **long** c) {

**this**.ladoA = a;

**this**.ladoB = b;

**this**.ladoC = c;

}

**public** **boolean** esTriangulo() {

**if**( **this**.ladoA + **this**.ladoB <= **this**.ladoC ||

**this**.ladoC + **this**.ladoB <= **this**.ladoA ||

**this**.ladoA + **this**.ladoC <= **this**.ladoB) {

**return** **false**;

}

**return** **true**;

}

**public** String getTipoTriangulo() {

**if**(**this**.ladoA == **this**.ladoB && **this**.ladoA == **this**.ladoC)

**return** "Equilatero";

**if**((**this**.ladoA == **this**.ladoB && **this**.ladoA != **this**.ladoC)||

(**this**.ladoA == **this**.ladoC && **this**.ladoA != **this**.ladoB)||

(**this**.ladoB == **this**.ladoC && **this**.ladoB != **this**.ladoA))

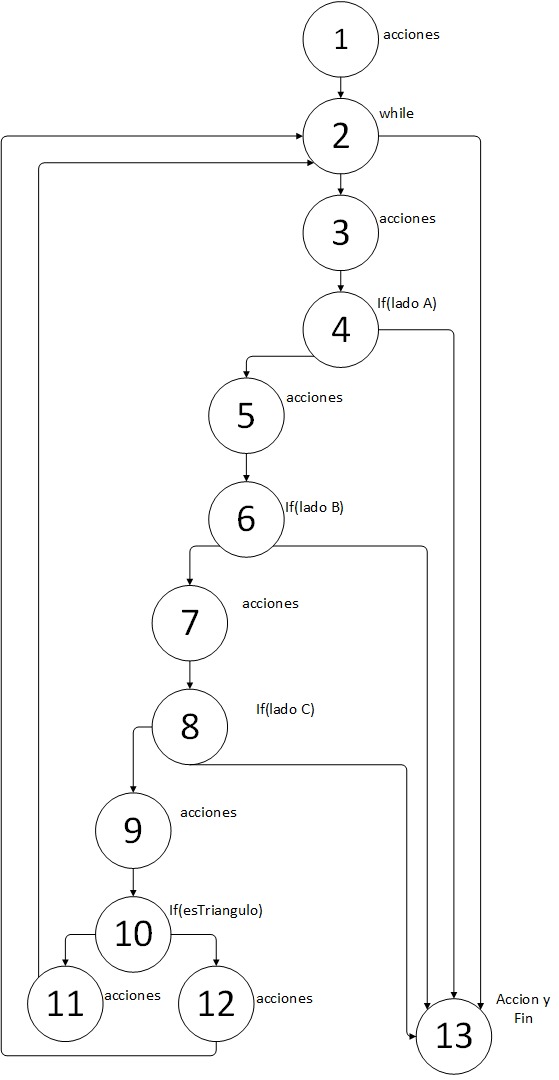
**return** "Isoceles";

**return** "Escaleno";

}

}

GRAFO DEL MÉTODO MAIN



RESULTADO DEL ANÁLISIS MEDIANTE HERRAMIENTA



VERIFICACIÓN DE RESULTADO VS GRAFO OBTENIDO

MÉTODO DE MC CABE

Recorrer todas las aristas con caminos linealmente independientes, donde un camino linealmente independiente es aquel que agrega una arista que no fue recorrida por ningún otro camino.

1. 1-2-13 *(camino más corto)*
2. 1-2-3-4-13
3. 1-2-3-4-5-6-13
4. 1-2-3-4-5-6-7-8-13
5. 1-2-3-4-5-6-7-8-9-10-11-2-13
6. 1-2-3-4-5-6-7-8-9-10-12-2-13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Lado A | Lado B | Lado C | Valor obtenido |
| 1 | - | - | - | - |
| 2 | 0 | - | - | FIN DEL PROGRAMA |
| 3 | 2 | 0 | - | FIN DEL PROGRAMA |
| 4 | 2 | 2 | 0 | FIN DEL PROGRAMA |
| 5 | 2 | 2 | 2 | EQUILATERO |
| 6 | 2 | 2 | 33 | NO ES TRIANGULO |

Caso 1 no factible porque el nodo 1 siempre es verdadero.

*NOTA: los valores esperados “isósceles” y “escaleno” no aparecen en la tabla porque están ligados al análisis del método “esTriangulo” en el archivo “Triangulo.java” el cual no fue analizado para este TP, ya que se limitó como alcance al método “main” del archivo “App.java”*

MÉTODO DE SENTENCIAS

Recorrer todos los nodos con la menor cantidad de caminos posibles

1. 1-2-3-4-5-6-7-8-9-10-11-2-13
2. 1-2-3-4-5-6-7-8-9-10-12-2-13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Caso | Lado A | Lado B | Lado C | Valor obtenido |
| 1 | 2 | 2 | 2 | EQUILATERO |
| 2 | 2 | 2 | 33 | NO ES TRIANGULO |