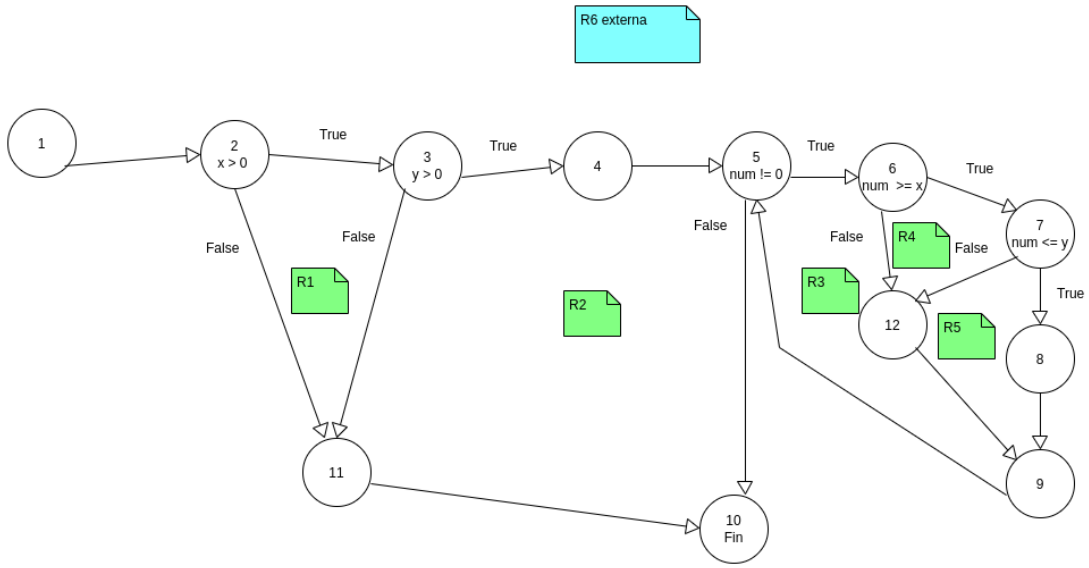
Soluciones ED Ordinario

TRIMESTRE 2

1a)



1b)

- **V (G) = a - n + 2** = 16 - 12 + 2 = 6
- **V (G) = r** +1 = 5 +1 = 6
- **V(G) = 5+ 1** = 6

Complejidad Ciclomática = 6

1c)

Camino1 = 1 2 3 4 5 6 7 8 9 5

Camino2 = 1 2 3 4 5 6 7 12 9 5

Camino3 = 1 2 3 4 5 6 12 9 5

Camino4 = 12 3 4 5 10

Camino5 = 1 2 3 11 10

Camino6 = 1 2 11 10

1d)

Camino | x | y | num | Salida

==========

1 2 4 2 Número en el rango c = 1

2 2 4 5 Número fuera de rango

3 2 4 1 Número fuera de rango

4	2	4	0
5	2	0	X
6	0	X	X

2)

Condición entrada	Clases de Equivalencia	Válidas	No Válidas
Empleado	rango	100>= Empleado<=999	Empleado<100 ; Empleado >999
Departamento	Lógica(sí o no) Valor	En blanco Cualquier num 2 dígitos	no es número num de > 2 dígitos ; num < 2 dígitos
Puesto	Miembro de un conjunto	"Programador"; "Analista"; "Diseñador"	"Tester"

3) Solución fácil, únicamente utilizamos los test parametrizados para pasar el valor del switch y comprobarlos todos. Se puede hacer algo más complejo con un sólo método de testeo pero controlando la entrada con un if.

```
import static org.junit.jupiter.api.Assertions.*;

import java.text.SimpleDateFormat;
import java.util.Date;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

import jdk.jfr.Description;

class FechaTest {

    Fecha f = new Fecha();
    Date d = new Date();


    @ParameterizedTest
    @Description("Test yyyy/MM")
    @CsvSource({"1"})
    void testDevuelveFecha1(int a) {
        SimpleDateFormat formato = new SimpleDateFormat("yyyy/MM");
        String s = formato.format(d);
        assertEquals(s, f.devuelveFecha(a));

    }

    @ParameterizedTest
    @Description("Test MM/yyyy")
```

```
    @CsvSource({"2"})
    void testDevuelveFecha2(int a) {
        SimpleDateFormat formato = new SimpleDateFormat("MM/yyyy");
        String s = formato.format(d);
        assertEquals(s, f.devuelveFecha(a));

    }

    @ParameterizedTest
    @Description("Test MM/yy")
    @CsvSource({"3"})
    void testDevuelveFecha3(int a) {
        SimpleDateFormat formato = new SimpleDateFormat("MM/yy");
        String s = formato.format(d);
        assertEquals(s, f.devuelveFecha(a));

    }

    @ParameterizedTest
    @Description("Test yyyy/MM")
    @CsvSource({"4,ERROR"})
    void testDevuelveFechaE(int a, String b) {

        assertEquals(b, f.devuelveFecha(a));

    }

}
```

TRIM1

Los ejercicios del 1 al 4 existen soluciones iguales en el aula virtual

5) Compilamos con `javac HelloWorld.java` y luego generamos el bytecode con `javap -c HellloWorld`. El bytecode es el código que se va a pasar a la MV para que interprete cada linea y la compile en la MV. Por eso se dice que es un lenguaje híbrido al tener un proceso de interpretación y otro de compilación.

6) Lo primero que hay que hacer es declarar los paquetes en cada clase y los import donde se utilicen los objetos.

```
package es.admon;

import es.man.Test1;
import es.dev.Test2;

public class Test3 {


        public static void main(String[] args) {
                Test1 t1 = new Test1();
                Test2 t2 = new Test2();
                System.out.println("Prueba General de Test3");
                t1.imprimir();
                t2.imprimir();
```

```
        }
}

package es.dev;

public class Test2 {
        public void imprimir(){
                System.out.println("Test2 dev");
        }
}

package es.man;

public class Test1 {
                public void imprimir() {
                     System.out.println("Test1 man");
                }

}
```

Luego se puede compilar sin problema y ejecutar de esta forma:

```
$ javac es/admon/Test3.java
$ java es.admon.Test3
Prueba General de Test3
Test1 man
Test2 dev
```

7. El jar sólo es hacer un META-INF/MANIFEST.MF con contenido:

```
Main-Class: es.admon.Test3
```

Para realizar el jar, mostrar su contenido y extraerlo

```
jar -cmf META-INF/MANIFEST.MF main2.jar es/admon/*.class es/man/*.class
es/dev/*.class
$ java -jar main2.jar
Prueba General de Test3
Test1 man
Test2 dev
$ jar -tf main2.jar
META-INF/
META-INF/MANIFEST.MF
es/admon/Test3.class
es/man/Test1.class
es/dev/Test2.class
$ jar xf main2.jar
```