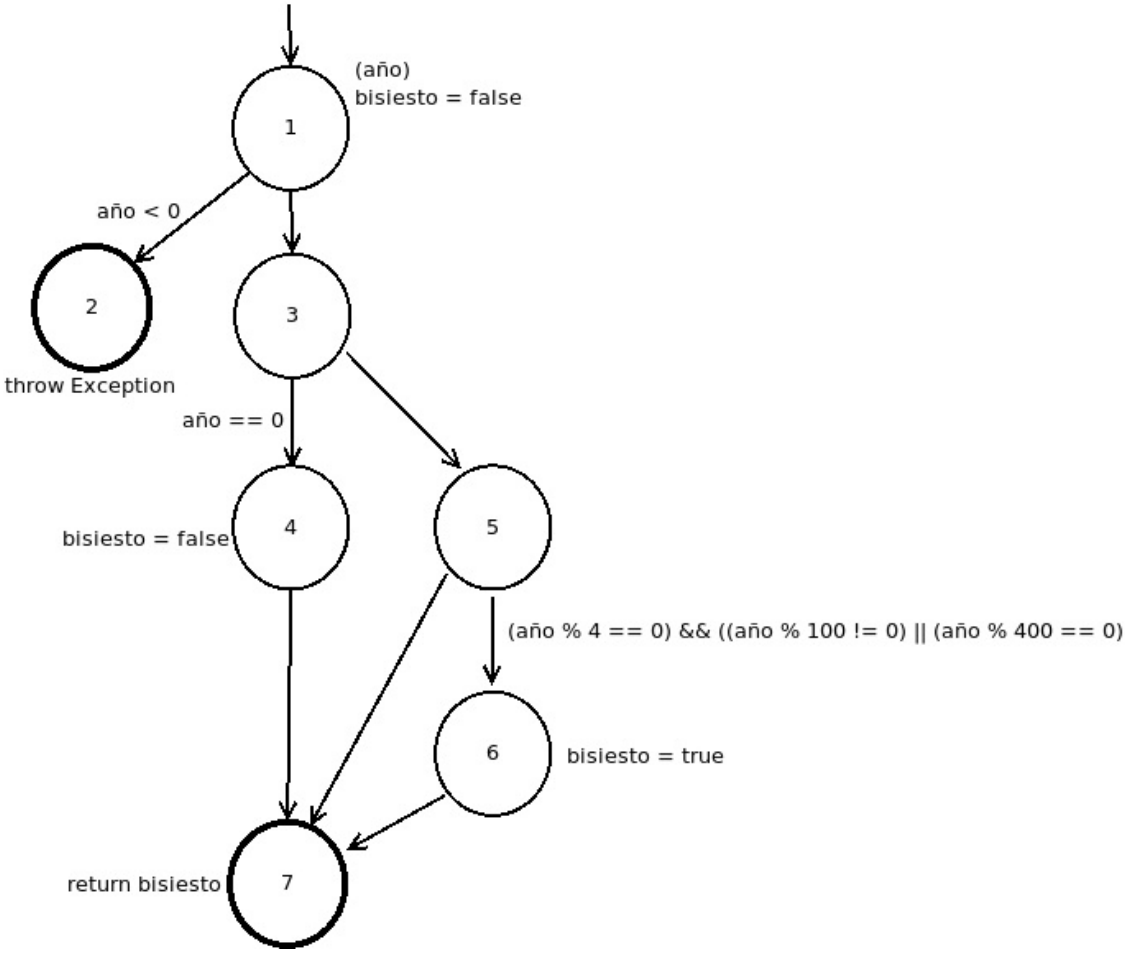
PRÁCTICA PRUEBAS IDM

Ejercicio 1 - Cálculo de años bisiestos

- **Apartado 1.** Se analizará la clase `Bisiestos`, donde uno de sus métodos llamado `esBisiesto`, devuelve true o false si el año que le proporcionamos es o no bisiesto.
- **Apartado 2.** Tenemos 1 parámetro:
  - Tenemos un único Parámetro `año`, de tipo *int*. Este será el año que queremos comprobar si es bisiesto o no.
- **Apartado 3.** Tendremos 3 caracterizaciones:
  - C1: año respecto del 0.
    - b1: año negativo
    - b2: año 0
    - b3: año positivo
  - C2: año múltiplo de 4.
    - b1: True
    - b2: False
  - C3: año múltiplo de 100.
    - b1: True
    - b2: False
  - C4: año múltiplo de 400.
    - b1: True
    - b2: False
- **Apartado 4.** Valores adecuados a cada bloque según criterios de cobertura:
  - C1: -200 | 0 | 2018
  - C2: 4 | 5
  - C3: 100 | 101
  - C4: 400 | 401
- **Práctica 2.a)** Enumerad los requisitos de prueba/test de acuerdo a PPC:
  - RT: [1, 2] [1, 3, 4, 7] [1, 3, 5, 7] [1, 3, 5, 6, 7]
- **Práctica 2.b)** Enumerad caminos de prueba/test de PPC:
  - Caminos de prueba: [1, 2] [1, 3, 4, 7] [1, 3, 5, 7] [1, 3, 5, 6, 7]
- **Práctica 2.c)** Diagrama + comentarios en tests:

- `testForNegativeYear` : Caminos recorridos: (1,2).
- `testForZeroYear` : Caminos recorridos: (1,3,4,7).
- `testMult4` , `testMult100` , `testMult400` : Caminos recorridos: (1,3,5,6,7).
- Añadimos `testForNoBisiesto` : Caminos recorridos: (1,3,5,7).



Ejercicio 2 - Conversión de números romanos a base 10

- **Apartado 1.** Se analizará la clase `RomanNumeral` , la cual contiene un método `convierte` que se encarga de transformar el *String* proporcionado en un número en base 10 (int).
- **Apartado 2.** Tenemos un único parámetro `s` de tipo *String*, que será el número romano que queremos convertir a base diez.
- **Apartado 3.** Tendremos 3 caracterizaciones:
  - C1: null (string vacío).
  - C2: string que sea romano.
  - C3: string que NO sea romano.
- **Apartado 4.** Valores adecuados a cada bloque según criterios de cobertura:

    - C1: string vacío | XVII | HJK / MMJ / IIIII

- **Práctica 2.a)** Enumerad los requisitos de prueba/test de acuerdo a PPC:

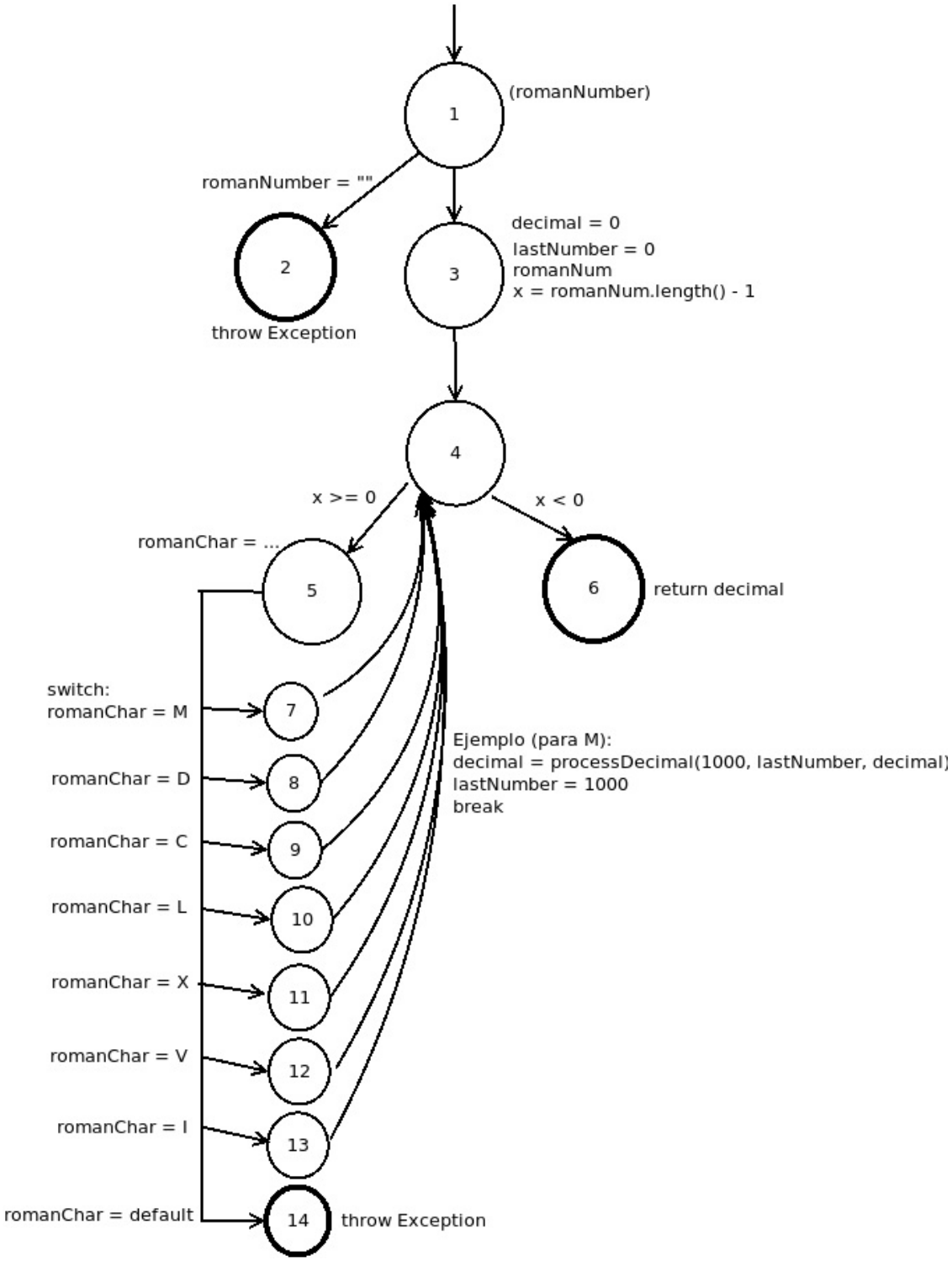
    - Por simplicidad, solo se hace el grafo del método `convierte`:
      NO ESTÁ BIEN RT: [1, 2] [1, 3, 4, 6] [1, 3, 4, 5, 7, 4, 6] [1, 3, 4, 5, 8, 4, 6] [1, 3, 4, 5, 9, 4, 6] [1, 3, 4, 5, 10, 4, 6] [1, 3, 4, 5, 11, 4, 6] [1, 3, 4, 5, 12, 4, 6] [1, 3, 4, 5, 13, 4, 6] [1, 3, 4, 5, 14, 4, 6]

- **Práctica 2.b)** Enumerad caminos de prueba/test de PPC:

    - Caminos de prueba: [1, 2] [1, 3, 4, 6] [1, 3, 4, 5, 7, 4, 6] [1, 3, 4, 5, 8, 4, 6] [1, 3, 4, 5, 9, 4, 6] [1, 3, 4, 5, 10, 4, 6] [1, 3, 4, 5, 11, 4, 6] [1, 3, 4, 5, 12, 4, 6] [1, 3, 4, 5, 13, 4, 6] [1, 3, 4, 5, 14, 4, 6]

- **Práctica 2.c)** Diagrama + comentarios en tests:

    - En nuestro caso, hay test que son redundantes y no se prueban, como por ejemplo (1,3,4,5,8,4,6) que sería solo testear que la entrada sea el número romano D. Para ello utilizamos *testForRoman*.
    - `testForNullString`: Caminos recorridos: (1,2).
    - `testForNoRoman`: Caminos recorridos: (1,3,4,5,14).
    - `testForRoman`: Caminos recorridos: (1,3,4,5,11,4,5,12,4,5,13,4,5,13,4,6).



Ejercicio 3 - Embotelladora

- **Apartado 1.** Se analizará la clase `Embotelladora`, la cual contiene un método llamado `calculaBotellasPequenas`, que debe devolver el número de botellas pequeñas, medianas y grandes necesarias para embotellar un número total de litros, minimizando el número de botellas grandes.

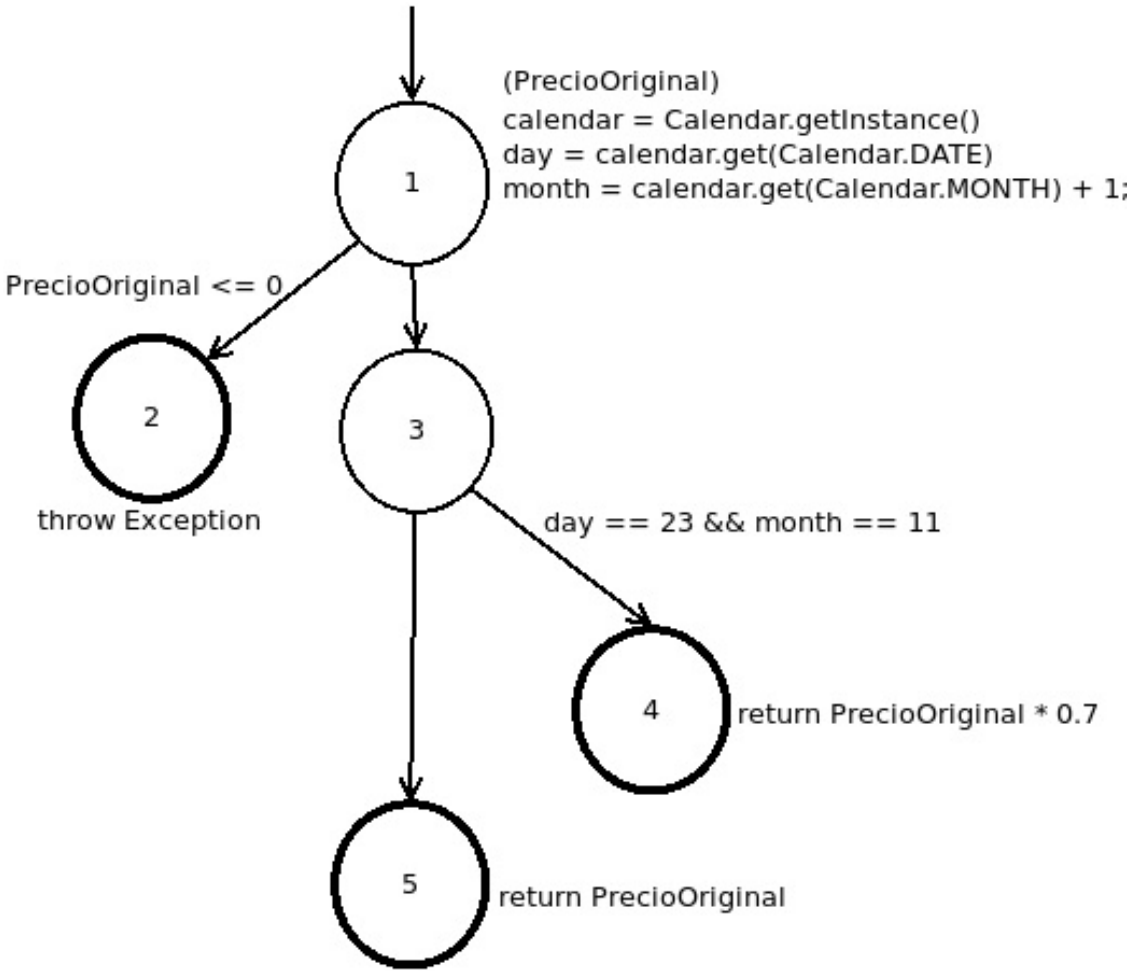
- **Apartado 2.** Hay 3 parámetros:
  - `pequeñas` : tipo int. Será el número de botellas de 1 litro disponibles en el almacén.
  - `grande` : tipo int. Será el número de botellas de 5 litros disponibles en el almacén.
  - `total` : tipo int. Número total de litros que queremos embotellar.
- **Apartado 3.** Tendremos en esta ocasión 3 caracterizaciones:
  - C1: Cantidad de botellas.
    - b1: ninguna (pequeñas y grandes son 0).
    - b2: solo pequeñas disponibles.
    - b3: solo grandes disponibles.
    - b4: ambas disponibles.
    - b5: pequeñas negativas.
    - b6: grandes negativas.
  - C2: Mayor cantidad de botellas.
    - b1: misma cantidad pequeñas que grandes
    - b2: mayor cantidad de pequeñas que de grandes
    - b3: mayor cantidad de grandes que de pequeñas
  - C3: Litros totales con respecto al 0.
    - b1: litros negativos
    - b2: litros 0
    - b3: litros positivos
  - C4: Abastecimiento dado litros totales y botellas
    - b1: abastecemos con botellas justas
    - b2: no abastecemos
    - b3: abastecemos y nos sobran botellas
- **Apartado 4.**
  - C1: 0,0,10 | 10,0,10 | 0,10,10 | 1,2,10 | -1,2,10 | 1,-2,10
  - C2: 3,3,8 | 3,1,8 | 1,3,8
  - C3: 10,0,-10 | 10,0,0 | 10,10,10
  - C4: 0,2,10 | 2,0,10 | 5,2,13
- **Apartado 5 (Práctica 2).** Enumerad los requisitos de prueba/test de acuerdo a PPC:
  - Por simplicidad, solo se hace el grafo del método `convierte` :
    NO ESTÁ BIEN RT: [1, 2] [1, 3, 4, 6] [1, 3, 4, 5, 7, 4, 6] [1, 3, 4, 5, 8, 4, 6] [1, 3, 4, 5, 9, 4, 6] [1, 3, 4, 5, 10, 4, 6] [1, 3, 4, 5, 11, 4, 6] [1, 3, 4, 5, 12, 4, 6] [1, 3, 4, 5, 13, 4, 6] [1, 3, 4, 5, 14, 4, 6]
- **Apartado 6 (Práctica 2).** Enumerad caminos de prueba/test de PPC:
  - Caminos de prueba: [1, 2] [1, 3, 4, 6] [1, 3, 4, 5, 7, 4, 6] [1, 3, 4, 5, 8, 4, 6] [1, 3, 4, 5, 9, 4, 6] [1, 3, 4, 5, 10, 4, 6] [1, 3, 4, 5, 11, 4, 6] [1, 3, 4, 5, 12, 4, 6] [1, 3, 4, 5, 13, 4, 6] [1, 3, 4,

5, 14, 4, 6]

- **Apartado 7 (Práctica 2).** Diagrama + comentarios en tests:
  - En nuestro caso, hay test que son redundantes y no se prueban, como por ejemplo (1,3,4,5,8,4,6) que sería solo testear que la entrada sea el número romano D. Para ello utilizamos *testForRoman*.
  - `testForNullString` : Caminos recorridos: (1,2).
  - `testForNoRoman` : Caminos recorridos: (1,3,4,5,14).
  - `testForRoman` : Caminos recorridos: (1,3,4,5,11,4,5,12,4,5,13,4,5,13,4,6).
  - INSERTAR IMAGEN

Ejercicio 4 - Descuento Black Friday

- **Apartado 1.** Se analizará la clase `DescuentoBlackFriday` , la cual contiene un método llamado `PrecioFinal` , que devuelve un precio final un 30% menor que el precio original proporcionado, en el caso de que el día actual sea 23 de noviembre.
- **Apartado 2.** Un único parámetro `precioOriginal` de tipo *double*. Será el precio original de un producto marcado en su etiqueta, sin descuento.
- **Apartado 3.** Hay 1 caracterización:
  - C1: precio original respecto del 0:
    - b1: precio negativo
    - b2: precio 0
    - b3: precio positivo
- **Apartado 4.** Da valores adecuados a cada bloque según criterios de cobertura:
  - C1: -2.0 | 0.0 | 25.50
- **Práctica 2.a)** Enumerad los requisitos de prueba/test de acuerdo a PPC:
  RT: [1, 2] [1, 3, 4] [1, 3, 5]
- **Práctica 2.b)** Enumerad caminos de prueba/test de PPC:
  - Caminos de prueba: [1, 2] [1, 3, 4] [1, 3, 5]
- **Práctica 2.c)** Diagrama + comentarios en tests:
  - `testForNegativePrice` : Caminos recorridos: (1,2).
  - `testForZeroPrice` : Caminos recorridos: (1,2).
  - `testForValidPrice` : Caminos recorridos: (1,3,4).
  - Añadimos `testForValidPriceDesc` : Caminos recorridos: (1,3,5).


(PrecioOriginal)
calendar = Calendar.getInstance()
day = calendar.get(Calendar.DATE)
month = calendar.get(Calendar.MONTH) + 1;
1
PrecioOriginal <= 0
2
throw Exception
3
day == 23 && month == 11
4
return PrecioOriginal * 0.7
5
return PrecioOriginal