TAREFA
PROBAS UNITARIAS

1 Métodos

Método **calcularDivision**
Recibe un dividendo e un divisor de tipo *float* e devolve o resultado da división tamén de tipo *float* sempre que o divisor non sexa 0, en cuxo caso xera unha excepción

`public class Division {`
`  public float calcularDivision(float dividendo, float divisor) throws Exception {`
`  if (divisor == 0) {`
`  throw (new Exception("Error. O divisor non pode ser 0."));`
`  }`
`  float resultado = dividendo / divisor;`
`  return resultado;`
`  }`
`}`

A efectos do grafo, o "`throws Exception`" é como facer un print por pantalla e ir directamente ao fin de programa.
Método **factorial**.
Recibe un número n de tipo byte e devolve o seu factorial de tipo *float* agás no caso de que sexa negativo, en cuxo caso xera unha excepción. O factorial dun número *n* é o produto de tódolos números menores que el ata o número 2. Casos especiais do factorial son factorial(0)=1 e factorial(1)=1.

`public class Factorial {`
`  public float factorial(byte n) throws Exception {`
`  if (n < 0) {`
`  throw new Exception("Error. O número ten que ser >=0");`
`  }`
`  float resultado = 1;`
`  for (int i = 2; i <= n; i++) {`
`  resultado *= i;`
`  }`
`  return resultado;`
`  }`
`}`

Método **busca**
Recibe un carácter **c** e un *array* de caracteres **v** de **10** elementos como máximo ordenados de forma **ascendente.** Consultar no seguinte enlace para ter información sobre a búsqueda binaria ou diconómica

```
public class OperacionsArrays {
 public boolean busca(char c, char[] v) {
   int a, z, i;
   a = 0;
   z = v.length - 1;
   boolean resultado = false;
   while (a <= z && resultado == false) {
     i = (a + z) / 2;
     if (v[i] == c) {
      resultado = true;
     }
     else{
      if (v[i] < c) {
            a = i + 1;
      }
      else{
            z = i - 1;
      }
     }
   }
   return resultado;
  }
 }
```

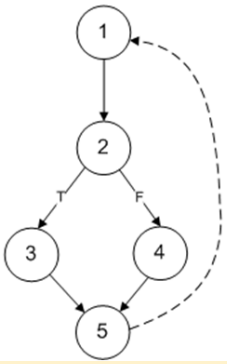
2 Solucións de grafos e complexidade de McCabe

Método **calcularDivision**
Recibe un dividendo e un divisor de tipo *float* e devolve o resultado da división tamén de tipo *float* sempre que o divisor non sexa 0, en cuxo caso xera unha excepción

```
public class Division {
   public float calcularDivision(float dividendo, float divisor) throws Ex-
ception {
     if (divisor == 0) {
      throw (new Exception("Error. O divisor non pode ser 0."));
     }
     float resultado = dividendo / divisor;
     return resultado;
 }
}
```

A efectos do grafo, o "`throws Exception`" é como facer un print por pantalla e ir directamente ao fin de programa.

```
public float calcularDivision(float dividendo, float divisor)      1
        throws Exception {
    if (divisor == 0) {2
        throw (new Exception("Error. El divisor no puede ser 0."));3
    }
    float resultado = dividendo / divisor;                          4
    return resultado;
}                                                                   5
```



Complexidade:

5 – 5 + 2 = 2
2 rexións
1+1 = 2

Camiños:

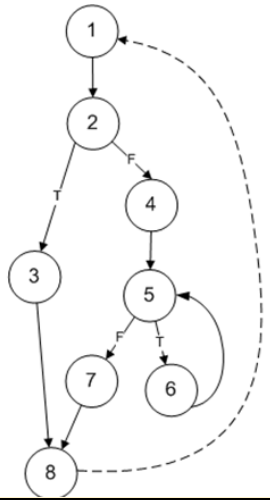
1-2-3-5
1-2-4-5

Método **factorial**.

Recibe un número n de tipo byte e devolve o seu factorial de tipo *float* agás no caso de que sexa negativo, en cuxo caso xera unha excepción. O factorial dun número *n* é o produto de tódolos números menores que el ata o número 2. Casos especiais do factorial son factorial(0)=1 e factorial(1)=1.

```
public class Factorial {
   public float factorial(byte n) throws Exception {
      if (n < 0) {
        throw new Exception("Error. O número ten que ser >=0");
      }
      float resultado = 1;
      for (int i = 2; i <= n; i++) {
              resultado *= i;
      }
      return resultado;
   }
}
```

```
public float factorial(byte n) throws Exception {                   1
    if (n < 0)2{
        throw new Exception("Error. El número tiene que ser >=0");3
    }
    float resultado = 1;                                            4
    for (int i = 2; i <= n;5 i++) {
        resultado *= i;                                             6
    }
    return resultado;                                               7
}                                                                   8
```



Complexidade:
9 – 8 + 2 = 3
3 rexións
2+1 = 3

Camiños:
1-2-3-8
1-2-4-5-6-5-..
1-2-4-5-7-8

Método **busca**

Recibe un carácter **c** e un *array* de caracteres **v** de **10** elementos como máximo ordenados de forma **ascendente.**

Consultar no seguinte enlace para ter información sobre a búsqueda binaria ou diconómica

```
public class OperacionsArrays {
 public boolean busca(char c, char[] v) {
   int a, z, i;
   a = 0;
   z = v.length - 1;
   boolean resultado = false;
   while (a <= z && resultado == false) {
     i = (a + z) / 2;
     if (v[i] == c) {
      resultado = true;
     }
     else{
      if (v[i] < c) {
            a = i + 1;
      }
      else{
            z = i - 1;
      }
     }
   }
   return resultado;
  }
 }
```

```
public class OperacionsArrays {
 public boolean busca(char c, char[] v) {
   int a, z, i;
   a = 0;
   z = v.length - 1;
   boolean resultado = false;                                  1
   while (a <= z && resultado == false) {                      2 3
     i = (a + z) / 2;                                          4
     if (v[i] == c) {                                          5
      resultado = true;                                        6
     }
     else{
      if (v[i] < c) {                                          7
            a = i + 1;                                         8
      }
      else{
            z = i - 1;                                         9
      }
     }
   }
   return resultado;                                           10
  }
 }                                                             11
```