EJERCICIO 2

===Fibonacci.h===

```
#pragma once
#include <iostream>
#include <string>
#include <stdexcept>

using namespace std;

template <typename T>
class Fibonacci {
  private:
  public:
    int sucesionFibonacci(int n);
};

#include "Fibonacci.tpp"
```

===Fibonacci.tpp===

```
#include <iostream>
#include "Fibonacci.h"
#include <stdexcept>

template <typename T>
int Fibonacci<T>::sucesionFibonacci(int n)
{
  if (n < 0) {
    throw invalid_argument("El numero debe ser no negativo.");
  }
  if(n == 0) { //CASO BASE
    return 0;
  }else if(n == 1) {
    return 1;
  }else {
    return sucesionFibonacci(n - 1) + sucesionFibonacci(n - 2); //CASO RECURSIVO
  }
}

/*CUÁNDO ESTO SERÍA INFINITO?
La recursión se volvería infinita si el valor de 'n' no se reduce correctamente en
cada llamada.
Esto puede ocurrir si no se define adecuadamente la resta (por ejemplo, usando n +
1 en lugar de n - 1),
o si se omiten los casos base (n == 0 o n == 1), lo que provocaría un
desbordamiento de pila.
También puede suceder si se manipula 'n' de forma incorrecta, generando valores
negativos o creciendo indefinidamente.
```

¿Por qué es una solución natural?
La definición matemática de Fibonacci es recursiva:
F(n) = F(n-1) + F(n-2)
Por eso, la solución recursiva es natural y refleja directamente la fórmula.*/

===main.cpp===

```
#include "Fibonacci.h"
#include <iostream>
#include <stdexcept>
#include <locale>

using namespace std;

int main(){

  Fibonacci<int> fib;
  typedef int Numero;

  Numero n, opciones;

  cout << "Ingrese un numero para calcular su Fibonacci: ";
  cin >> n;
  try {
    Numero resultado = fib.sucesionFibonacci(n);
    cout << "El Fibonacci de " << n << " es: " << resultado << endl;
  } catch (const invalid_argument& e) {
    cerr << "Error: " << e.what() << endl;
  }
}
```