Grado en Ingeniería Información

PROGRAMACIÓN II - Sesión 4

Tema 2.

Tratamiento de excepciones

Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T2. Tratamiento de excepciones

- Bloques try/catch y sentencia throw

ÍNDICE

Excepciones: **eventos** que aparecen en **tiempo de ejecución** cuando se produce un ***error*** inesperado.

El **tratamiento de excepciones** permite preparar el código de la aplicación para **evitar la interrupción de la ejecución del programa** cuando se produce una **excepción**.

Hay que delimitar ***los bloques de código*** que se quieren controlar e indicar qué ***bloque de código deberá ejecutarse*** para cada ***excepción***.

T2. Tratamiento de excepciones

- **Bloques try/catch y sentencia throw**

ÍNDICE

try:

Define el bloque de código donde **detectar** y **gestionar** las excepciones.

```
try
{
   // bloque de código
}
```

Contiene el **código del programa normal**.

Si aparece un ***error*** en el bloque de código marcado con ***try***, entonces se lanza una ***excepción*** pasando el **control de la ejecución a otra zona de la aplicación** que el programador ha designado a tal efecto y el programa no tiene que terminar, se puede continuar si así se decide.

***throw*:**

Lanza una **excepción** cuando **aparece alguna situación de error**, siempre que el programador lo considere necesario.

throw ([Excepcion])

catch:

Especifica el **código que hay que ejecutar** ante una determinada **excepción**.

```
catch ([tipoExcepcion])
{
  // bloque de código
}
```

Esta zona de la aplicación es la que gestiona la aparición de una ***excepción*** en el bloque ***try***. Después de cada bloque ***try*** deberá haber, al menos, un bloque ***catch***. El bloque ***catch*** puede recibir como argumento el **tipo de excepción** que captura y eso permite que un bloque ***try*** pueda estar seguido de varios bloques ***catch*** para **gestionar diferentes excepciones con diferentes bloques de código**.

Estrategia general:

1. Se programa con ***try*** una operación para anticipar el ***error***.
2. Cuando se detecta el ***error***, se lanza la **excepción** con ***throw.***
3. Se captura la **excepción** con ***catch*** a través de la condición que se anticipa al ***error***.

Funcionamiento:

1. Se ejecuta el bloque de instrucciones ***try***:
   - Hay **error**:
     a. Se lanza la **excepción** con ***throw.***
     b. Se interrumpe el bloque ***try*** para ir al bloque ***catch*** correspondiente.
   - **No** hay **error:** se continua la ejecución después de los bloques ***catch***.

Funcionamiento:

- ✓ La ***excepción*** es capturada por el bloque ***catch*** cuyo parámetro coincida con el **tipo de objeto** lanzado por la sentencia ***throw***.
- ✓ El orden de los bloques ***catch*** es determinante porque la **búsqueda** del bloque ***catch*** se realiza siguiendo el **orden en que aparecen en el código** hasta que coincide el **tipo** o se llega a un ***catch*** **genérico**:

catch (...) { /* bloque genérico*/ }

Sintaxis:

Bloques try/catch y sentencia throw

```
try {
    // Codigo de ejecución normal
    throw (tipo_dato);
}
catch (Tipo1 &excep) {
    // Gestión de excep tipo 1
}
catch (Tipo2 &excep) {
    // Gestión de excep tipo 2
}
catch (...) {
    // Bloque catch genérico
    // Gestión de cualquier excep no capturada mediante los catch anteriores
}
// Continuación del código
```

Esquema:

Bloques try/catch y sentencia throw

```
void funcion()
int main ()
{     try {
         // llamada a la función que está preparada para cualquier error
         funcion();
         // código normal
      }
      catch (int error) {
         // captura las excepciones que se lanzan desde funcion()
         // se hace algo para dichas excepciones
         }
       // resto del código del main()
}
void funcion()
{   // código que produce las excepciones
         throw (i);
}
```

```
// Función que realiza la división entera
int divisionSegura (int numerador, int denominador)
{
    if (denominador==0)
    {
        // ERROR: Se está tratando de dividir por cero
        // Se lanza una excepción de tipo 1
        throw 1;
    }
    else  return numerador/denominador;

}
```

```cpp
#include <iostream>

using namespace std;

int divisionSegura (int numerador, int denominador);

int main ()
{   int dividendo, divisor, cociente;

    cout << "\n\n\tIntroduzca el dividendo: ";
    cin >> dividendo;

    cout << "\n\n\tIntroduzca el divisor: ";
    cin >> divisor;

    try
    {
        cociente = divisionSegura (dividendo, divisor);
        cout << "\n\n\tLa operacion es " << dividendo << "/" << divisor << " = " << cociente << endl;
    }
    catch(int error)
    {
        cout << "\n\tLo sentidmos, el divisor no puede ser cero.\n";
        cout <<  "\n\tEl programa se termina de ejecutar.\n";
        cout << "\n\n\t";
    }



    cout << "\n\n\t";
    return 0;
}
```

Realizar un programa en C++ que contenga una clase llamada ***CalcularNumPositivo***.

La clase tiene dos atributos, ***numero*** y ***resultado***, ambos de tipo decimal y los métodos:

- **Constructores** por **defecto** y **por copia**.
- **Set** y **Get** necesarios para manejar los atributos.
- **sumaNum,** método que recibe el valor de un numero que suma a ***numero***. Guarda la suma realizada en el atributo **resultado** y devuelve dicho valor para mostrarlo en pantalla desde el **main**.
- **restaNum**, método que recibe el valor de un numero que resta a ***numero***. Guarda la resta realizada en el atributo **resultado** y devuelve dicho valor para mostrarlo en pantalla desde el **main**.
- **multiplicaPor**, método que recibe el valor de un número que multiplica por ***numero***. Guarda el producto realizado en el atributo **resultado** y devuelve dicho valor para mostrarlo en pantalla desde el **main**.
- **dividePor**, método que recibe el valor del denominador para dividir por ***numero***. Guarda la división realizada en el atributo **resultado** y devuelve dicho valor para mostrarlo en pantalla desde el **main**. Además, debe lanzar dos excepciones (tipo ***string***), una si la división es por 0 y otra si se divide por un número negativo.