Github: LoreDN
Author: Lorenzo Di Napoli
Repository: https://github.com/LoreDN/Cpp

myExceptions — Documentation

Contents

Library Description

This library is a special one, because it is the foundation that allows all the other **C++** libraries to work the intended way.
Here are collected all the custom **Exceptions** that can be triggered when working with the libraries, togheter with some test functions, used in order to catch the wanted **Exception**.

CONTENTS OF THE LIBRARY

The library consists of two files:

- **"myExceptions.hpp":** the template library, it is the header file where you can find a brief description on how to use the library in practice, together with all the classes definitions and test functions prototypes.
- **"myExceptions.cpp":** the core of the library, it contains all the methods definitions for each class, togheter with the test functions definitions; it is the file that has to be given to the compiler in order to be able to use the library.

INHERITARY STRUCTURE

The library starts with the definition of a simple abstract class: *"myException"*.
Starting from this class, all the derivated ones create a Tree Structure, which allows to manage in a simple way all the **Exceptions**, and makes easy including new ones or derivating more specific **Exceptions** from already existing ones.

Classes Documentation

In the *"myExceptions"* library have been implemented a total of four classes:

R. **myException:** the Root abstract class, from which all the others derive from.

1. **myIndex_Exception:** handles out-of-memory limit acceses for a Data Structure (as accessing to `array[i > size]`).
2. **myDataSize_Exception:** handles Data-Structure size limits (as an Heap with `heap.usage == heap.size`).
3. **myFile_Exception:** handles filestreams and checks if they are valid (for all of ifstream/ofstream/fstream).

ROOT. myException

```
class myException {

     protected:

          // attributes
          std::string message;

     public:

          // constructor
          myException(std::string msg);

          // methods prototypes
          virtual void print() = 0;

};
```

`std::string message;`

The Exception message, it is a *protected* member, since the user is not able to modify it.
This allows each **Exception** to manage its error message indipendently from the others.

`myException(std::string msg);`

This is the *constructor*, which assignes a string `msg` (usually set when catching the **Exceptions**, in order to proper describe it) to the *protected* member `myException.message`.

`virtual void print() = 0;`

Core of the *"myException"* abstract class, this *virtual method* imposes that each **Exceptions** prints something in order to let the user know it was catched, usually the message.
Since it has been left as a *virtual method*, each **Exception** is able to make it useless when overriding it, but this is not the case in the already implemented ones, given in the library.

1. myIndex_Exception

```
class myIndex_Exception :  public myException {

    private:

        // attributes
        size_t index;

    public:

        // constructor
        myIndex_Exception(size_t idx, std::string msg);

        // methods prototypes
        void print() override;

};

void test_index(size_t index, size_t limit);
```

`size_t index;`

The index which caused the **Exception**, it is a *private* member, since the user is not able to modify it. It is stored in order to be printed by the `myIndex_Exception.print()` method.

`myIndex_Exception(size_t idx, std::string msg);`

This is the *constructor*, which assignes a string `msg` (set when catching the **Exception**, in order to proper describe it) to the *protected* member `myException.message` and an index variable `idx` to the *private* member `myIndex_Exception.index`.

`void print() override;`

Overrided method which first of all prints the message `myException.message`, followed by the the index `myIndex_Exception.index`, which caused the **Exception**.

`void test_index(size_t index, size_t limit);`

Test function indipendet from the **myIndex_Exception** class, which allows to catch the **Exception**. The test throws the **Exception** only if there is a try to access out-of-bounds memory; it is checked by a simple `if (index > limit)`.

2. myDataSize_Exception

```
class myDataSize_Exception :  public myException {

     private:

          // attributes
          size_t size;

     public:

          // constructor
          myDataSize_Exception(size_t dim, std::string msg);

          // methods prototypes
          void print() override;

};

void test_data_size(size_t size, size_t limit);
```

`size_t size;`

The Data-Structure size which caused the **Exception**, it is a *private* member, since the user is not able to modify it.
It is stored in order to be printed by the `myDataSize_Exception.print()` method.

`myDataSize_Exception(size_t sz, std::string msg);`

This is the *constructor*, which assignes a string `msg` (set when catching the **Exception**, in order to proper describe it) to the *protected* member `myException.message` and a size variable `dim` to the *private* member `myDataSize_Exception.size`.

`void print() override;`

Overrided method which first of all prints the message `myException.message`, followed by the size of the Data-Structure `myDataSize_Exception.size`, which caused the **Exception**.

3. myFile_Exception

```
class myFile_Exception :  public myException {

    private:

        // attributes
        std::string path;

    public:

        // constructor
        myFile_Exception(std::string name, std::string msg);

        // methods prototypes
        void print() override;

};

void test_infile(std::string path, std::ifstream *file);
void test_outfile(std::string path, std::ofstream *file);
void test_iofile(std::string path, std::fstream *file);
```

`std::string path;`

The file path, it is a *private* member, since the user is not able to modify it.
It is stored in order to be printed by the `myFile_Exception.print()` method.

`myFile_Exception(std::string name, std::string msg);`

This is the *constructor*, which assignes a string `msg` (set when catching the **Exception**, in order to proper describe it) to the *protected* member `myException.message` and a string `name` to the *private* member `myFile_Exception.path`.

`void print() override;`

Overrided method which first of all prints the message `myException.message`, followed by the the file path `myFile_Exception.path`, which caused the **Exception**.

`void test_infile(std::string path, std::ifstream *file);`

Test function indipendet from the **myFile_Exception** class, which allows to catch the **Exception**.
The test throws the **Exception** only if the std::ifstream file cannot be opened correctly; it is checked by a simple `if (!(*file).is_open())`.

```
void test_outfile(std::string path, std::ofstream *file);
```

Test function indipendet from the **myFile_Exception** class, which allows to catch the **Exception**. The test throws the **Exception** only if the std::ofstream file cannot be opened correctly; it is checked by a simple `if (!(*file).is_open())` .

```
void test_iofile(std::string path, std::fstream *file);
```

Test function indipendet from the **myFile_Exception** class, which allows to catch the **Exception**. The test throws the **Exception** only if the std::fstream file cannot be opened correctly; it is checked by a simple `if (!(*file).is_open())` .