Object Oriented Programming - part 2

Emmanuelle Darles

The exceptions in C++

Table of contents

Objectives

The objectives of this lecture are :

- to learn the mecanisms to manage exceptions in a C++ code ;
- to learn to implement exception classes in order to improve the robustness of your code.

I Why the exceptions ?

1. A brief example

Consider the code below :

```
int division(int a, int b)
{
   return a/b;
}
```

When b is equal to zero, this function return `nan` (*not a number)*.

How to treat this case ?

Solution 1

A common approach is to print a message :

```
#include <iostream>
int division(int a, int b)
{
   using std::cerr;
   int result;
   if (b!=0)
      result = a / b;
   else
      cerr << "ERROR";
   return result;
}
```

Problem : The program will not know that there is a problem !

Solution 2

Another solution is to modify the function signature :

```
const int SUCCESS = 0,
          ERROR = -1;

int division(int a, int b)
{
   using std::cerr;
   int result;
   if (b!=0){
      result = a / b;
      return SUCCESS;
   }
   else
      return ERROR;
}
```

Problem : This strategy complicates the code.

Error handling becomes a concern that alters the calling of the function.

Objectives of exceptions

The objectives of exceptions are to catch the errors and to express treatments in case of errors.

II The mechanism of exceptions

1. An example

Different stages :

1. The method detect an error and throw an exception to the calling function
2. The calling method catch the exception and specify treatments.

1. The exception thrown

The exception thrown is specify using the throw method.

Example

```
int division(int a, int b) throw()
{
   if(b!=0)
   {
      return a / b;
   }
   else
     throw std::string("Error");
}
```

2. The exception catch

The exception catch is made in a try / catch block.

Example

```
int main(void)
{
    try{
        int a,b;
        std::cout << "Input a number :";
        std::cin >> a;
        std::cout << "Input another number :";
        std::cin >> b;
        int result = division(a,b);
        std::cout << "a/b=" << result << std::endl;
    }
    catch(std::string e){
        std::cout << e << std::endl;
    }
}

```

Example:Execution example

```
Input a number :10
Input another number :0
terminate called after throwing an instance of 'std::string'
Abort trap: 6
```

III Write your own exceptions

1. User exception

It's possible to define your own exception by deriving of the class "exception".

```
class exception
{
   public :
      exception() throw();
      exception(const exception &e) throw();
      exception& operator=(const exception &e) throw();
      virtual ~exception() throw();
      virtual const char *what() const throw();
};
```

To define user exception, you must implement a derived class of the class "exception".

Example:DivisionException

```
#ifndef DIVISION_EXCEPTION_
#define DIVISION_EXCEPTION_

#include <exception>

class DivisionException : public std::exception
{
   public :
        DivisionException(const char* msg);
        const char* what() const throw() ;

    private :
        const char* msg;
};
#endif
```

```
#include "divisionException.hpp"

DivisionException::DivisionException(const char* msg):std::exception()
{
    this->msg = msg;
```

```cpp
}
const char* DivisionException::what() const throw()
{
  return msg;
}

```

```cpp
#include "divisionException.hpp"
#include <iostream>

int division(int a, int b) throw()
{
   if(b!=0)
   {
      return a/b;
   }
   else
     throw DivisionException("Division by zero !");
}

int main(void)
{
   int a, b;
   std::cout << "Input a number :";
   std::cin >> a;
   std::cout << "Input an another number :";
   std::cin >> b;
   try {
     division(a,b);
   }
   catch (DivisionException &e) {
     std::cout << e.what() << std::endl;
   }
}
```

Execution example :

```
Input a number :10
Input an another number :0
terminate called after throwing an instance of 'DivisionException'
  what():  Division by zero !
Abort trap: 6
```

2. Exception in a constructor

It's possible to throw an exception in a code constructor.

- If an exception is thrown during execution, the object is not built.
- In inheritance case, super-class constructor is called after the key word try.

Example:Class Circle

```cpp
#ifndef CIRCLE_
#define CIRCLE_

#include "form.hpp"
```

```
#include "point2D.hpp"

class Circle : public Form
{
   public :
      Circle(Point2D center, float radius) throw();
      ~Circle();

      //Getters
      Point2D getCenter();
      float   getRadius();

      //Setters
      void setCenter(Point2D center);
      void setRadius(float radius);

      //Overriding method
      void display();

      //Overriding method
      float area();

    private :
        Point2D center;
        float radius;
};
#endif
```

```
#include "circle.hpp"
#include <iostream>
#include <math.h>
#include <exception>

Circle::Circle(Point2D center, float radius) throw ()
    try : Form()
    {
        this->center = center;
        this->radius = radius;
        if(this->radius==0)
            throw std::exception();
    }
    catch(std::exception &e)
    {
        std::cout << e.what() << std::endl;
    }

Circle::~Circle()
{
}
Point2D Circle::getCenter()
{
    return this->center;
}
float Circle::getRadius()
{
    return this->radius;
}
void Circle::setCenter(Point2D center)
```

```
{
    this->center = center;
}
void Circle::setRadius(float radius)
{
    this->radius = radius;
}
void Circle::display()
{
   std::cout << "I'm a circle !" << std::endl;
   std::cout << "My center is ";
   this->center.display();
   std::cout << "My radius is " << this->radius << std::endl;
}
float Circle::area()
{
    return M_PI*this->radius*this->radius;
}

```

```
#include "circle.hpp"

int main(void)
{
  Circle C(Point2D(1,1),0);
}

```

Output :

```
terminate called after throwing an instance of 'std::exception'
  what():  std::exception
Abort trap: 6
```

3. Exception in a destructor

A destructor can throw an exception but it's not advisable.

- What do in thrown exception case ?

It's possible to ensure that a destructor not throw exception :

```
#ifndef CIRCLE_
#define CIRCLE_

#include "form.hpp"
#include "point2D.hpp"

class Circle : public Form
{
   public :
      Circle(Point2D center, float radius) throw();
      ~Circle() throw();

      //Getters
      Point2D getCenter();
      float   getRadius();
```

```

    //Setters
    void setCenter(Point2D center);
    void setRadius(float radius);

    //Overriding method
    void display();

    //Overriding method
    float area();

  private :
      Point2D center;
      float radius;
};
#endif
```

```
#include "circle.hpp"
#include <iostream>
#include <math.h>
#include <exception>

Circle::Circle(Point2D center, float radius) throw ()
    try : Form()
    {
        this->center = center;
        this->radius = radius;
        if(this->radius==0)
            throw std::exception();
    }
    catch(std::exception &e)
    {
        std::cout << e.what() << std::endl;
    }

Circle::~Circle() throw()
{
}
Point2D Circle::getCenter()
{
    return this->center;
}
float Circle::getRadius()
{
    return this->radius;
}
void Circle::setCenter(Point2D center)
{
    this->center = center;
}
void Circle::setRadius(float radius)
{
    this->radius = radius;
}
void Circle::display()
{
   std::cout << "I'm a circle !" << std::endl;
   std::cout << "My center is ";
   this->center.display();
   std::cout << "My radius is " << this->radius << std::endl;
}
float Circle::area()
{
    return M_PI*this->radius*this->radius;
}

```

IV Quiz: Exception Form

Question 1

Write a class ExceptionForm in order to manage exceptions in the form library (in the same way of the DivisionException example)

Question 2

Modify the classes Circle, Rectangle and Square to treat the case of degenerated forms by throwing an ExceptionForm object in this case.