まとめ

- **送出式**で例外オブジェクトを**送出**すると、コピーが作られた上で送出される。
- 部品の利用側では、**try ブロック**と**例外ハンドラ**によって例外を**捕捉**する。例外ハンドラは、例外宣言中に宣言されている特定の型の例外を捕捉する。
  ただし、受け取るのが例外クラスへのポインタ／参照である場合、そのクラスの下位クラスの例外も捕捉する。また、... と宣言された例外ハンドラは、未捕捉の全例外を捕捉する。
- 例外を捕捉したものの、それに対する処理が完了できなければ、その例外をそのまま**再送出**するか、あるいは別の例外として送出するとよい。
- 例外クラスを階層化すると、例外の分類や構造化が可能になるとともに、捕捉が容易かつ柔軟に行える。
- 例外クラスの階層構造を利用した例外ハンドラでは、上位のクラスをより先頭側に配置して、下位のクラスをより末尾側に配置する。
- 例外クラスを多相的クラスにした上で階層化すると、例外に対する対処を多相的に行える。
- 標準 C++ ライブラリが送出する例外型の最上位の基底クラスが、**<exception> ヘッダ**で定義されている ***exception* クラス**である。
- 例外指定に対する違反が行われたときに送出される例外 ***bad_exception*** の他にも、以下の ***bad_*** 系例外が提供される。

<new>	記憶域の確保に失敗したときに送出される例外 ***bad_alloc***
<typeinfo>	動的キャストに失敗したときに送出される例外 ***bad_cast***
	typeid 式中に空ポインタが含まれるときに送出される例外 ***bad_typeid***

- **<stdexcept>** では、**標準例外**（論理エラーと実行時エラー）が定義されている。
- **論理エラー *logic_error*** は、論理的な条件に対する違反などの、プログラム実行の前に検出可能であって理論的には回避可能なエラーである。以下の例外が提供される。

domain_error	ドメインエラーを報告
invalid_argument	不正な実引数を報告
length_error	最大長を超えた長さのオブジェクトが生成されようとしたことを報告
out_of_range	実引数の値が範囲外であることを報告

- **実行時エラー *runtime_error*** は、事前に予測することが困難であって、プログラムを実行しないと検出できないようなエラーである。以下の例外が提供される。

range_error	内部計算で発生する範囲エラーを報告
overflow_error	算術的なオーバフローエラーを報告
underflow_error	算術的なアンダフローエラーを報告

chap09/MyException.cpp

```
//--- 例外の送出・再送出と捕捉 ---//

#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;

//--- 自作の例外 ---//
class MyException : public logic_error {
public:
    // コンストラクタ
    MyException() : logic_error("マイ例外") { }

    virtual const char* what() const { return "マイ例外"; }
};

//--- swの値に応じて例外を送出 ---//
void work(int sw)
{
    switch (sw) {
     case 1: throw exception("exception例外");
     case 2: throw logic_error("logic_error例外");
     case 3: throw MyException();
    }
}

//--- workを呼び出す ---//
void test(int sw)
{
    try {
        work(sw);
    }
    catch (const MyException& e) {
        cout << e.what() << "を補足。対処完了!!\n";
    }
    catch (const logic_error& e) {
        cout << e.what() << "を補足。対処断念!!\n";
        throw;                                  // 再送出
    }
    catch (const exception& e) {
        cout << e.what() << "を補足。対処断念!!\n";
        throw "ABC";                            // 文字列として送出
    }
}

int main()
{
    int sw;

    cout << "sw: ";
    cin >> sw;

    try {
        test(sw);
    }
    catch (const logic_error& e) {
        cout << e.what() << "を補足。\n";
    }
    catch (const char* e) {
        cout << e << "を補足。\n";
    }
}
```

実行例 1

```
sw: 1
exception例外を補足。対処断念!!
ABCを補足。
```

実行例 2

```
sw: 2
logic_error例外を補足。対処断念!!
logic_error例外を補足。
```

実行例 3

```
sw: 3
マイ例外を補足。対処完了!!
```